HITACHI
Inspire the Next

JP1 Version 11

JP1/Advanced Shell

3021-3-B32-20

前書き

■ 著作権

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，JP1，uCosminexus は，株式会社 日立製作所の商標または登録商標です。

Active Directory は，米国 Microsoft Corporation の，米国およびその他の国における登録商標または商標です。

IBM，AIX は，世界の多くの国で登録された International Business Machines Corporation の商標です。

Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

Microsoft および Excel は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Microsoft および Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Microsoft および Windows Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Microsoft Office および Excel は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Oracle と Java は，Oracle Corporation 及びその子会社，関連会社の米国及びその他の国における登録商標です。

Red Hat は，米国およびその他の国で Red Hat, Inc. の登録商標もしくは商標です。

すべての SPARC 商標は，米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標がついた製品は，米国 Sun Microsystems, Inc. が開発したアーキテクチャに基づくものです。

SUSE は，米国およびその他の国における SUSE LLC の登録商標または商標です。

UNIX は，The Open Group の米国ならびに他の国における登録商標です。

Win32 は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

■ 発行

2017 年 1 月 3021-3-B32-20

変更内容

変更内容（3021-3-B32-20）JP1/Advanced Shell 11-10，JP1/Advanced Shell - Developer 11-10

追加・変更内容	変更個所
ジョブ定義スクリプトの作成手段について追加した。	1.3
次に示す UNIX 互換コマンドを使用できるようにした。 • tr • xargs また，which コマンドの注意事項を変更した。	2.1.1(4)，2.1.2(3)，2.2.7(3)(c)，8.1.2(3)，8.2.2(1)，8.4.36，8.4.40，8.4.41
初期設定スクリプトファイルについて説明を追加した。	2.2.7(2)(a)，2.6.23，5.5.3，7.2.1(2)，7.3.19，8.3.7，9.4.5
ユーザー応答機能管理デーモンの自動起動と自動停止について説明を変更した。	2.8.3(1)(c)
ジョブ定義スクリプト実行開始時のファイルモード作成マスクを設定する機能について説明を追加した。	2.12，7.2.1(2)，7.3.48
ウイルス対策ソフト実行時の注意事項を追加した。	2.14
拡張出力モードの出力先について説明を追加した。	3.4.1
スプールジョブ名指定機能について説明を追加，変更した。	3.4.2(3)，3.4.4(2)，3.7.8，3.9，3.13，5.5.2，8.3.6，8.3.9
最小出力モードで出力抑止するメッセージについて説明を変更した。	3.4.4(1)，3.5.1(2)(c)
【Windows 限定】外部コマンドの終了コードが格納されるシェル変数 ADSH_RC_EXTERNAL について説明を追加した。	5.5.1，5.5.4，5.8.8(3)，9.5.9
ulimit コマンドの引数について注意事項を追加した。	9.3.30
メッセージを追加した。 KNAX0298-E，KNAX6501-I，KNAX6502-I，KNAX6503-E，KNAX6504-E	12.2，12.3，KNAX0298-E，KNAX6501-I，KNAX6502-I，KNAX6503-E，KNAX6504-E
メッセージの説明を変更した。 KNAX4427-W，KNAX6097-E，KNAX6153-E，KNAX6380-I，KNAX6381-E	12.3，KNAX4427-W，KNAX6097-E，KNAX6153-E，KNAX6380-I，KNAX6381-E
次の用語を用語解説に追加した。 • 初期設定スクリプトファイル • スプールジョブ名	付録 D

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルでは，JP1/Advanced Shell を使用して，バッチジョブのためのジョブ定義スクリプトを作成・実行する方法について説明しています。

- JP1/Advanced Shell（バッチジョブ向けスクリプト実行基盤）
- JP1/Advanced Shell - Developer（バッチジョブ向けスクリプト開発基盤）

JP1/Advanced Shell と JP1/Advanced Shell - Developer をインストールした環境を区別する場合は，「実行環境」，「開発環境」と表記します。

■ 対象製品

- P-1M12-B1BL JP1/Advanced Shell 11-10（適用 OS：AIX V6.1，AIX V7.1，AIX V7.2）
- P-8112-B1BL JP1/Advanced Shell 11-10（適用 OS：Red Hat Enterprise Linux Server 6（64-bit x86_64），Red Hat Enterprise Linux Server 7（64-bit x86_64），Oracle Linux 6 (x64)，Oracle Linux 7，CentOS 6，CentOS 7，SUSE Linux 12）
- P-1J12-B1BL JP1/Advanced Shell 11-10（適用 OS：HP-UX 11i V3（IPF））
- P-9D12-B1BL JP1/Advanced Shell 11-10（適用 OS：Solaris 10（SPARC），Solaris 11（SPARC））
- P-2A12-B1BL JP1/Advanced Shell 11-10（適用 OS：Windows Server 2016，Windows 10，Windows 8，Windows Server 2012，Windows 7，Windows Server 2008）
- P-2A12-B2BL JP1/Advanced Shell - Developer 11-10（適用 OS：Windows Server 2016，Windows 10，Windows 8，Windows Server 2012，Windows 7，Windows Server 2008）

■ 対象読者

JP1/Advanced Shell を使用して，バッチジョブを開発・実行・管理する方を対象としています。また，このマニュアルは次の知識をある程度持つ方にお読みいただくことを前提に説明しています。

- Windows および UNIX
- JP1/AJS
- JP1/Base
- JP1/IM

■ 読書手順

このマニュアルは，次に示す章と付録から構成されています。

バッチジョブを実行する運用者とジョブ定義スクリプトを作成する開発者ごとに，必要な章または付録を選択して読むことができます。利用者に応じてお読みいただくことをお勧めします。

なお，このマニュアルは，Windows および UNIX の OS に共通のマニュアルです。OS ごとに差異がある場合は，本文中でそのつど内容を書き分けています。

編タイトル	章または付録のタイトル	記載内容	運用者	開発者
第 1 編 概要編	1. JP1/Advanced Shell の概要	JP1/Advanced Shell は，バッチ処理のためのジョブ定義スクリプトを作成・実行するための製品です。JP1/Advanced Shell の目的，業務への応用例，システムの全体構成，処理の流れ，クラスタシステムでの運用の概要および機能概要について説明しています。	○	○
第 2 編 構築編	2. JP1/Advanced Shell を利用するための準備	プログラムのインストール先と種類，前提条件，インストール，環境情報の設定，カスタムジョブの登録，ユーザー応答機能の設定，クラスタ運用の環境情報設定などの JP1/Advanced Shell を利用するために必要な項目について説明しています。	◎	○
第 3 編 運用編	3. バッチジョブの実行	JP1/Advanced Shell（実行環境）を使用したバッチジョブの運用として，バッチジョブの実行方法や動作について説明しています。	◎	△
	4. JP1/Advanced Shell - Developer を使用する【Windows 限定】	JP1/Advanced Shell - Developer を使用して，Windows 環境でジョブ定義スクリプトを開発するための JP1/Advanced Shell エディタの操作方法について説明しています。エディタを使用したジョブ定義スクリプトファイルのデバッグ方法についても説明します。	－	◎
	5. ジョブ定義スクリプトの作成	ジョブ定義スクリプトの文法について説明しています。	－	◎
	6. ジョブ定義スクリプトのデバッグ	JP1/Advanced Shell のデバッガ機能について説明しています。	○	○
第 4 編 リファレンス編	7. 環境ファイルで設定するパラメーター	環境ファイルで設定できる，次のパラメーターの記述形式と詳細について説明しています。 • 環境設定パラメーター • export パラメーター • 条件パラメーター	○	△
	8. 運用時に使用するコマンド	運用時に使用するコマンドの記述形式と詳細について説明しています。	◎	△
	9. ジョブ定義スクリプトのコマンドおよび制御文	ジョブ定義スクリプトファイルに使用する次のコマンドや制御文に関して，記述形式と詳細を説明しています。	△	◎

編タイトル	章または付録のタイトル	記載内容	運用者	開発者
第 4 編 リファレンス編	9.　ジョブ定義スクリプトのコマンドおよび制御文	• シェル標準コマンド • シェル拡張コマンド • スクリプト拡張コマンド • スクリプト制御文 • スクリプト予約語コマンド	△	◎
	10.　スクリプト開発部品	スクリプト開発部品の記述形式と詳細について説明しています。	－	◎
第 5 編 トラブルシューティング編	11.　トラブルシューティング	トラブルシューティングとして，対処の手順，ログ情報の種類，必要な資料，資料の採取方法について説明しています。	○	△
	12.　メッセージ	JP1/Advanced Shell が出力するメッセージとエラーの詳細について説明しています。	△	△
付録	付録 A　カバレージ情報を取得する対象	カバレージ情報を取得する対象について説明しています。	△	△
	付録 B　JP1/AJS 以外のジョブスケジューラから起動する場合【UNIX 限定】	実行環境で JP1/AJS 以外のジョブスケジューラを使用して JP1/Advanced Shell のバッチジョブを起動する方法について説明しています。	△	△
	付録 C　各バージョンの変更内容	各バージョンの変更内容を説明しています。	△	△
	付録 D　用語解説	このマニュアルで使用する用語について解説しています。	○	○

（凡例）

◎：中心の知識となるため，熟読してください。

○：一とおり読んでいただくことをお勧めします。

△：必要に応じて参照してください。

－：該当しません。

■ このマニュアルで使用する記号

このマニュアルの文法説明で使用する記号を次に示します。

記号	意味
[ ]	次の 2 つの意味があります。 • GUI 説明の場合，メニュー項目，ダイアログボックス，ボタンなどを示します。 **例** ［ファイル］－［新規作成］を選択する。 上記の例では，メニューバーの［ファイル］を選択して，ドロップダウンリストの［新規作成］を選択することを示します。

記号	意味
[ ]	• 文法説明の場合，[ ] で囲まれている項目は，省略できます。 複数の項目が記述されている場合，すべてを省略するか，どれか 1 つを選択します。 例 [A] 「何も指定しない」か「A を指定する」ことを示します。 [B|C] 「何も指定しない」か「B または C を指定する」ことを示します。
< >	各項目を記述するときに従わなくてはならない構文要素記号※を示します。
{ }	この記号で囲まれている複数の項目の中から，一組の項目を必ず選択します。項目と項目の区切りは「|」で示します。 例 {A|B|C} 「A，B または C のどれかを必ず指定する」ことを示します。
| (ストローク)	複数の項目に対し，項目間の区切りを示し，「または」の意味を示します。 例 「A|B|C」 「A，B または C」を示します。
+	この記号の直前に示された項目を繰り返して複数指定ができます。または，この記号の前後の項目が同時に設定されていることを示します。 例 {A|B} + 「A または B を任意の順序で 1 つ以上指定する」ことを示します。 [CR] + [LF] [CR] と [LF] が同時に設定されていることを示します。
*	この記号の直前に示された項目を指定しないか，または繰り返して複数指定ができます。 例 {A|B} * 「A または B を指定しないか，任意の順序で 1 つ以上指定する」ことを示します。
~	この記号の直前に示されている項目を，この記号に続く< >，《 》，(( ))などの文法規則に従って記述する必要があることを示します。
《 》	項目を省略したときのデフォルト値を示します。
(( ))	指定できる値の範囲を示します。
__（下線）	選択記号 [ ] で囲まれている項目を省略したときのデフォルト値を示します。
…および…（点線）	この記号の直前に示された項目を繰り返して複数個指定できることを示します。 例 A，B，…

記号	意味
…および… （点線）	「A のあとに B を必要個数指定する」ことを示します。
太字	可変および強調を示します。
斜体	可変を示します。
△	スペースを示します。次のように使い分けていることもあります。 $△_0$：0 個以上のスペース（スペースを省略できる） $△_1$：1 個以上のスペース（スペースを省略できない）

注※

このマニュアルの構文要素記号を次に示します。

構文要素記号	指定できる文字の内容
＜数字＞	0|1|2|3|4|5|6|7|8|9
＜英大文字＞	A|B|C|…|Z
＜英小文字＞	a|b|c|…|z
＜英字＞	＜英大文字＞|＜英小文字＞
＜特殊文字＞	,|.|/|'|(|)|*|&|+|-|=|△（半角スペース）|¥
＜ 8 進数＞	＜ 0|1|2|3|4|5|6|7 ＞ ＋
＜ 10 進数＞	＜数字＞ ＋
＜ 16 進数字＞	0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
＜整数＞	符号のある数字または符号のない数字の集まり。
＜符号なし整数＞	＜数字＞＋
＜記号名称＞	{＜英字＞|＜数字＞|@|#|_（アンダースコア）} ＋ **対象**：ジョブ名など。
＜環境変数名＞	{＜英字＞|_（アンダースコア）}{＜英字＞|_（アンダースコア）|＜数字＞} ＊ **対象**：ファイル環境変数定義名，環境変数名，スクリプト拡張コマンドなど。
＜パス名＞	UNIX または Windows のファイルパス名規則に従った文字列。
＜コマンド名＞	パス名で指定可能な文字からパス区切り文字を除いたもの。
＜論理ホスト名＞	{＜英字＞|＜数字＞|-（ハイフン）} ＋
＜任意文字列＞	任意の文字による文字列。 • JP1/Advanced Shell では文字種別をチェックしません。 • 指定個所に応じた適切な意味のある文字列とする必要があります。 • 記号名称の範囲での利用を推奨します。

構文要素記号	指定できる文字の内容
＜ASCII 文字列＞	ASCII 文字コード範囲中の，制御文字を除いた範囲（0x20～0x7e）の文字によって構成される文字列。

■ このマニュアルで使用する製品名の表記

製品名の表記

このマニュアルでは，製品名を次のように表記しています。

このマニュアルでの表記			正式名称
JP1/Advanced Shell			JP1/Advanced Shell
			JP1/Advanced Shell - Developer
JP1/AJS	JP1/AJS3		JP1/Automatic Job Management System 3 - Agent
			JP1/Automatic Job Management System 3 - Manager
			JP1/Automatic Job Management System 3 - View
JP1/AJS - Agent	JP1/AJS3 - Agent		JP1/Automatic Job Management System 3 - Agent
JP1/AJS - Definition Assistant	JP1/AJS3 - Definition Assistant		JP1/Automatic Job Management System 3 - Definition Assistant
JP1/AJS - Manager	JP1/AJS3 - Manager		JP1/Automatic Job Management System 3 - Manager
JP1/AJS - View	JP1/AJS3 - View		JP1/Automatic Job Management System 3 - View
JP1/IM	JP1/IM - Manager		JP1/Integrated Management - Manager
	JP1/IM - View		JP1/Integrated Management - View
UNIX	Linux	CentOS 6	CentOS 6
		CentOS 7	CentOS 7
		Oracle Linux 6	Oracle Linux® Operating System 6 (x64)
		Oracle Linux 7	Oracle Linux® Operating System 7
		RHEL 6	Red Hat Enterprise Linux® Server 6（64-bit x86_64）
		RHEL 7	Red Hat Enterprise Linux® Server 7（64-bit x86_64）
		SUSE Linux 12	SUSE Linux® Enterprise Server 12
	AIX		AIX V6.1
			AIX V7.1

このマニュアルでの表記		正式名称
UNIX	AIX	AIX V7.2
	HP-UX	HP-UX 11i V3（IPF）
	Solaris	Solaris 10（SPARC）
		Solaris 11（SPARC）

マイクロソフト製品の表記

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

このマニュアルでの表記		正式名称
Windows Server※	Windows Server 2016	Microsoft® Windows Server® 2016 Standard
		Microsoft® Windows Server® 2016 Datacenter
	Windows Server 2012	Microsoft® Windows Server® 2012 Standard
		Microsoft® Windows Server® 2012 Datacenter
		Microsoft® Windows Server® 2012 R2 Standard
		Microsoft® Windows Server® 2012 R2 Datacenter
	Windows Server 2008	Microsoft® Windows Server® 2008 R2 Datacenter
		Microsoft® Windows Server® 2008 R2 Enterprise
		Microsoft® Windows Server® 2008 R2 Standard
Windows※	Windows 10	Windows® 10 Home(32 ビット版)
		Windows® 10 Pro(32 ビット版)
		Windows® 10 Enterprise(32 ビット版)
		Windows® 10 Home(64 ビット版)
		Windows® 10 Pro(64 ビット版)
		Windows® 10 Enterprise(64 ビット版)
	Windows 8	Windows® 8.1(32 ビット版)
		Windows® 8.1 Pro(32 ビット版)
		Windows® 8.1 Enterprise(32 ビット版)
		Windows® 8.1(64 ビット版)
		Windows® 8.1 Pro(64 ビット版)
		Windows® 8.1 Enterprise(64 ビット版)

このマニュアルでの表記		正式名称
Windows※	Windows 8	Windows® 8(32 ビット版)
		Windows® 8 Pro(32 ビット版)
		Windows® 8 Enterprise(32 ビット版)
		Windows® 8(64 ビット版)
		Windows® 8 Pro(64 ビット版)
		Windows® 8 Enterprise(64 ビット版)
	Windows 7	Microsoft® Windows® 7 Enterprise
		Microsoft® Windows® 7 Professional
		Microsoft® Windows® 7 Ultimate
Excel		Microsoft® Excel
		Microsoft® Office Excel

注※ Windows Server および Windows を総称して Windows と表記することがあります。

■ このマニュアルで使用するフォルダおよびディレクトリの表記

Windows と UNIX で共用する部分では，ディレクトリという用語を使用しています。Windows 限定の記載がある場合には，フォルダを使用します。

上記に伴って，ディレクトリの区切りには，「/」を使用します。Windows 特有で使用する場合は，フォルダの区切りには，「¥」を使用します。

■ このマニュアルで使用する「インストール先フォルダ」について（Windows の場合）

このマニュアルで使用している「**インストール先フォルダ**」は，特に断りがないかぎり JP1/Advanced Shell のインストール先フォルダを示しています。製品を初期設定のままインストールした場合のインストール先フォルダは次のとおりです。

x86 環境の場合

システムドライブ:¥Program Files¥Hitachi¥JP1AS

x64 環境の場合

システムドライブ:¥Program Files(x86)¥Hitachi¥JP1AS

■ このマニュアルで使用する「Administrators 権限」について

このマニュアルで使用している「Administrators 権限」とは，ローカル PC に対する Administrators 権限です。ローカル PC に対して Administrators 権限を持つユーザーであれば，ローカルユーザー，ドメインユーザーおよび，Active Directory 環境で動作に違いはありません。

■ このマニュアルで使用する「共通アプリケーションフォルダ」について

このマニュアルで使用している「**共通アプリケーションフォルダ**」について次に示します。

システムドライブ:¥ProgramData

■ このマニュアルで使用する「共有ドキュメントフォルダ」について

このマニュアルで使用している「**共有ドキュメントフォルダ**」について次に示します。

システムドライブ:¥Users¥Public¥Documents

■ このマニュアルで使用する Windows のメニュー名について

このマニュアルで使用している Windows のメニュー名の表記は，次の OS を前提としています。

Windows 7，Windows Server 2008

Windows Server 2016，Windows 10，Windows 8 または Windows Server 2012 の場合は［スタート］メニューが表示されないため，画面左下から表示できる［スタート］画面からメニューを選択してください。

■ このマニュアルで使用する KB（キロバイト）などの単位表記

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

■ 関連マニュアル

関連マニュアルを次に示します。必要に応じてお読みください。

JP1/Advanced Shell 関連

- JP1 Version 11 ジョブ管理 基本ガイド（スクリプト言語編）（3021-3-B31）

JP1/AJS 関連

- JP1 Version 11 ジョブ管理 基本ガイド（ジョブスケジューラー編）（3021-3-B11）
- JP1 Version 11 JP1/Automatic Job Management System 3 設計ガイド（システム構築編）（3021-3-B13）

- JP1 Version 11 JP1/Automatic Job Management System 3 構築ガイド（3021-3-B15）
- JP1 Version 11 JP1/Automatic Job Management System 3 トラブルシューティング（3021-3-B17）
- JP1 Version 11 JP1/Automatic Job Management System 3 操作ガイド（3021-3-B18）
- JP1 Version 11 JP1/Automatic Job Management System 3 コマンドリファレンス（3021-3-B19）
- JP1 Version 11 JP1/Automatic Job Management System 3 - Definition Assistant（3021-3-B25）

JP1/NETM/DM 関連

- JP1 Version 10 JP1/NETM/DM 導入・設計ガイド(Windows(R)用)（3021-3-175）
- JP1 Version 10 JP1/NETM/DM 運用ガイド 1(Windows(R)用)（3021-3-177）
- JP1 Version 8 JP1/NETM/DM SubManager(UNIX(R)用)（3020-3-L42）
- JP1 Version 6 JP1/NETM/DM Manager（3000-3-841）

JP1/Base 関連

- JP1 Version 11 JP1/Base 運用ガイド（3021-3-A01）

JP1/IM 関連

- JP1 Version 11 JP1/Integrated Management - Manager 構築ガイド（3021-3-A08）
- JP1 Version 11 JP1/Integrated Management - Manager 運用ガイド（3021-3-A09）

uCosminexus Application Server 関連

- Cosminexus V9 アプリケーションサーバ 機能解説 拡張編（3020-3-Y08）
- Cosminexus V9 アプリケーションサーバ リファレンス コマンド編（3020-3-Y15）

目次

第 3 編　運用編

第 4 編　リファレンス編

第 5 編　トラブルシューティング編

1 JP1/Advanced Shell の概要

JP1/Advanced Shell は，バッチジョブのためのジョブ定義スクリプトを作成・実行するための製品です。この章では，JP1/Advanced Shell の目的，業務への応用例，システムの全体構成，処理の流れおよび機能概要について説明します。

1.1　JP1/Advanced Shell の目的

JP1/Advanced Shell は，バッチ業務の開発生産性や運用効率を向上するための製品です。バッチジョブのためのジョブ定義スクリプト（シェルスクリプト）を効率的に作成・実行できます。

JP1/Advanced Shell には，次に示す特長があります。

1.1.1　バッチ業務の OS 間での資産の継承

- **既存資産の活用**

  UNIX 環境で作成したシェルスクリプトを利用して Windows 環境でジョブ定義スクリプトを開発できます。

  JP1/Advanced Shell で使用するジョブ定義スクリプトでは，シェル標準互換の言語仕様を採用しています。したがって，習得しやすく，既存のシェルスクリプトからの移行も容易です。

- **クロスプラットフォームの対応**

  クロスプラットフォームとは，複数の OS 基盤のことです。クロスプラットフォームに対応した機能が利用できます。

  - Windows 環境で開発したジョブ定義スクリプトを Windows 環境でも UNIX 環境でも実行できます。
  - UNIX 互換コマンドを，Windows 環境でも UNIX 環境でも使用できます。

1.1.2　バッチ業務の構築のスピードアップ

- **ジョブの実行の制御**

  JP1/Advanced Shell では，バッチ業務で繰り返し使用される処理を自動化したり，簡潔に記述したりできるようにジョブ定義スクリプトを拡張しています。

  次の機能を使用してジョブ定義スクリプトの記述量を削減し，ジョブ定義スクリプトの可読性や保守性を向上できます。

  - ジョブステップの実行条件を指定できます。
  - ジョブステップ内で有効な変数を使用できます。
  - バッチジョブがエラー終了した場合，エラーメッセージを出力したり，終了コードを設定したりできます。
  - バッチジョブがエラー終了した場合，自動的に子プロセスを強制終了して，バッチジョブで使用した一時ファイルを自動的に削除できます。

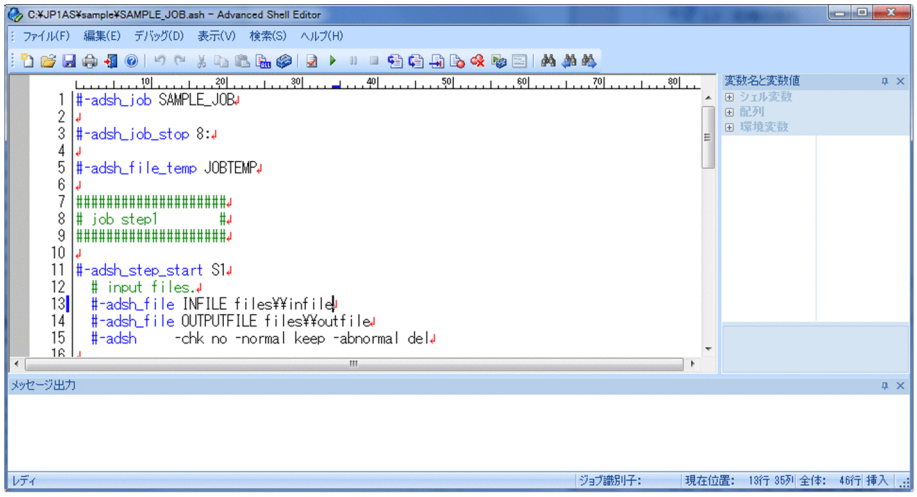
- **エディタでのジョブ定義スクリプトの開発（開発環境）**

  GUI（Graphical User Interface）の JP1/Advanced Shell エディタ（デバッグ機能付き専用エディタ）を使用した開発環境でジョブ定義スクリプトを開発し，デバッグできます。

- ジョブステップ単位で実行したり，ブレークポイントを設定したりできます。
- ジョブ定義スクリプトのカバレージ情報を蓄積できます。

JP1/Advanced Shell エディタウィンドウを次の図に示します。

図 1-1　JP1/Advanced Shell エディタウィンドウ



- ファイルの割り当ておよび後処理の効率化

  通常ファイルの存在チェックや一時ファイルの割り当てと削除などの処理を自動化でき，簡潔に記述できます。

  - バッチジョブの実行中に自動的に一時ファイルを割り当て，バッチジョブ終了時に削除できます。
  - バッチジョブの実行中に通常ファイルの存在チェック，およびジョブステップまたはジョブの結果によるファイルの後処理ができます。

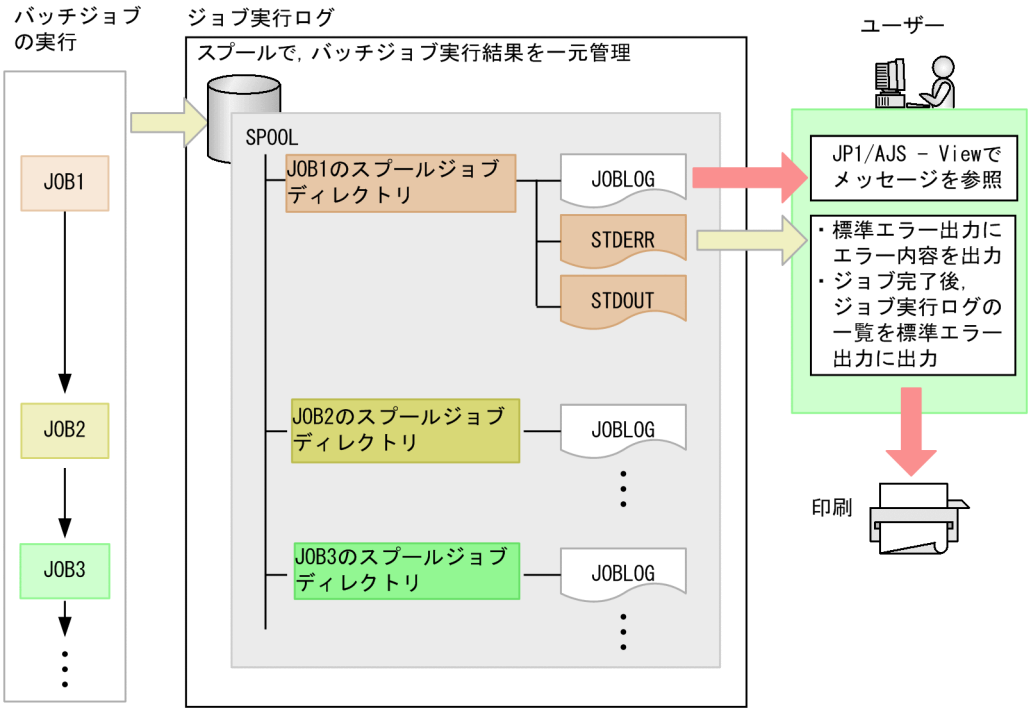
1.1.3　バッチジョブの実行結果の一元管理による運用性・保守性の向上

ジョブ実行ログを自動的に出力し，一元管理することで，障害時などの保守性を向上できます。

従来，オープンシステムでのバッチジョブの実行結果は，格納先が一元化されていなかったため管理が煩雑でした。JP1/Advanced Shell で運用した場合，ジョブ実行ログを採取することで，バッチジョブの実行結果をスプールに集めて一元管理できます。また，JP1/AJS - View を使用することで，ジョブ定義スクリプトの実行を自動化して定期的にバッチジョブを実行したり，バッチジョブの実行結果も参照したりできます。

各ジョブの実行結果は，スプールディレクトリの下にスプールジョブのディレクトリとして出力されます。バッチジョブの実行結果の一元管理を次の図に示します。

図 1-2　バッチジョブの実行結果の一元管理



ジョブ実行ログの出力内容については，「3.5　ジョブ実行ログ」を参照してください。

トラブルシューティングでは，ジョブ実行ログやシステム実行ログ，トレースログなどの資料を採取して，トラブルが発生した場合に対処できます。詳細については，「11.　トラブルシューティング」を参照してください。

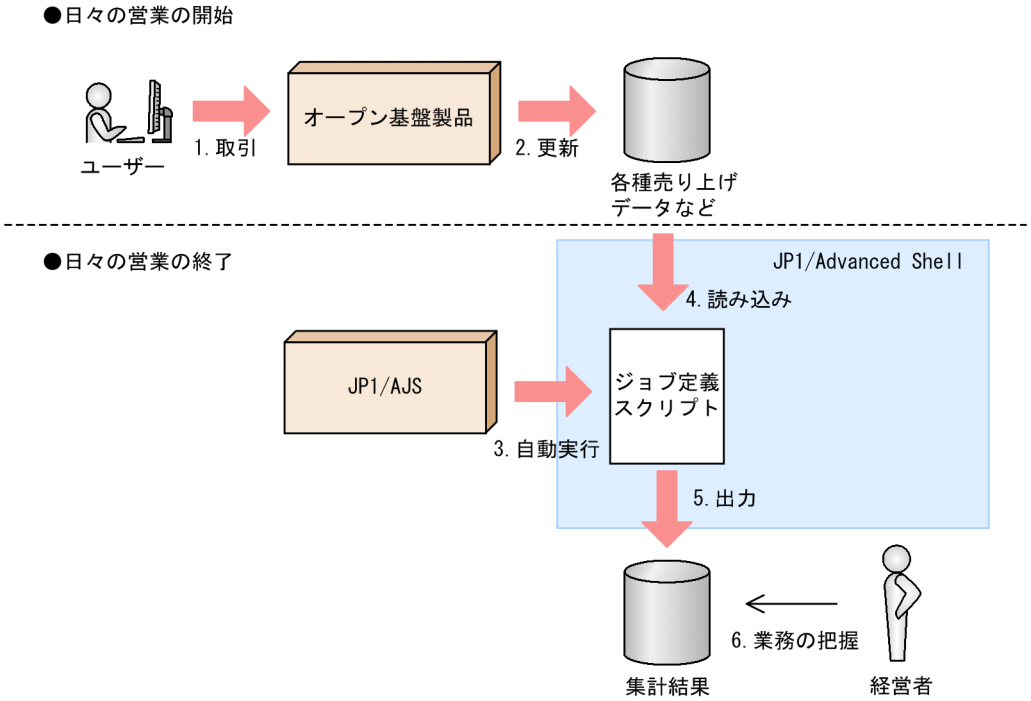
1.2　業務への応用例

JP1/Advanced Shell は，次のような業務に応用できます。

日中にオンラインシステムで多数の取引があり，夜間にその集計をする業態であれば，例えば，売り上げや商品の販売数，在庫数などを集計するバッチジョブを開発し，実行できます。また，日次処理，月次処理，期末処理といった定型集計用だけでなく，特定の用途や，臨時のタイミングで使用するバッチジョブも開発・実行できます。

JP1/Advanced Shell の運用例（日々の営業集計の場合）を次の図に示します。

図 1-3　JP1/Advanced Shell の運用例（日々の営業集計の場合）



1. 日々の営業を開始し，ユーザーは商品などの取引をします。
2. オープン基盤製品は，各種売り上げデータなどを更新します。
3. 日々の営業が終了して，JP1/AJS は指定された時間にジョブ定義スクリプトの自動実行を指示します。
4. JP1/Advanced Shell は各種売り上げデータを処理するためのジョブ定義スクリプトを実行します。
5. JP1/Advanced Shell はジョブ定義スクリプトの実行結果を出力します。
6. 経営者は，実行結果を基に集計情報や商品の売り上げの推移などを把握できます。

1.3　処理の流れ

JP1/Advanced Shell は，実行環境（JP1/Advanced Shell）と開発環境（JP1/Advanced Shell - Developer）とに分かれています。開発環境で作成したジョブ定義スクリプトを実行環境で実行します。ジョブ定義スクリプトは，任意のテキストエディタを使用して作成してもかまいません。

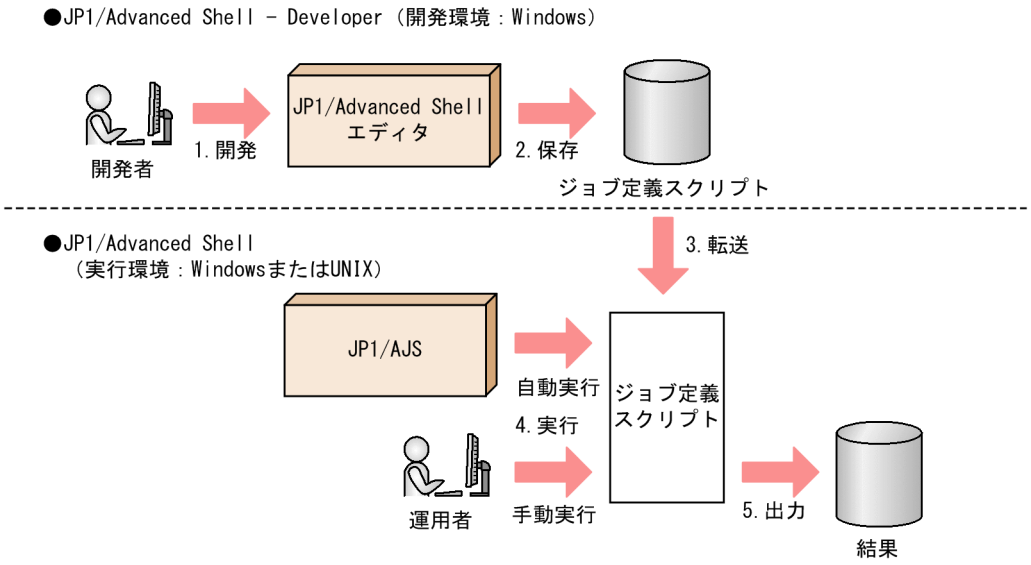
システムに対する権限で分類した場合，JP1/Advanced Shell の利用者は，システム管理者と一般ユーザーに分けられます。それぞれの役割について説明します。

表 1-1　JP1/Advanced Shell の利用者の分類

JP1/Advanced Shell の利用者		役割
システム管理者		システム運営上の責任者です。スーパーユーザー権限をあらかじめ与えられています。 システム管理者は，JP1/Advanced Shell を実行できる環境を管理して，JP1/Advanced Shell を使用する一般ユーザーをシステムに登録します。
一般ユーザー	開発者	ジョブ定義スクリプトを作成したり，デバッグなどをしたりします。
	運用者	JP1/Advanced Shell の定義，実行および結果の確認を実施して，JP1/Advanced Shell の実行に不具合があれば対処します。 JP1/AJS を使用する場合の運用者の作業については，「3.1.1　JP1/AJS のジョブに関する運用者の作業」を参照してください。

JP1/Advanced Shell のシステムの全体構成を次の図に示します。

図 1-4　JP1/Advanced Shell のシステムの全体構成



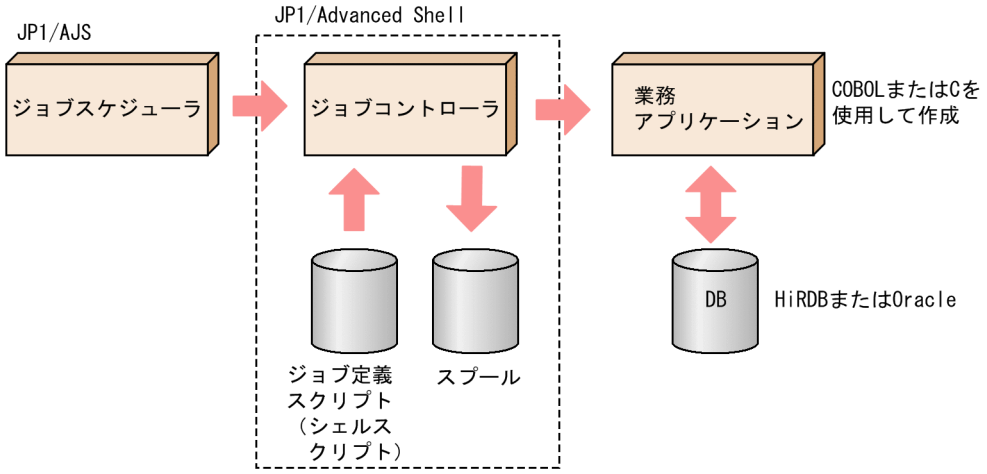
1. Windows 上の開発環境で開発者が JP1/Advanced Shell エディタや任意のテキストエディタを使ってジョブ定義スクリプトを入力します。
2. JP1/Advanced Shell エディタや任意のテキストエディタからジョブ定義スクリプトを保存します。
3. ジョブ定義スクリプトを実行環境に転送します。

4. 実行環境で運用者が次のような方法を使ってジョブ定義スクリプトの実行を指示します。
   - JP1/AJS を使った自動実行
   - コマンドプロンプトや UNIX のシェルからの手動実行など
5. JP1/Advanced Shell がジョブ定義スクリプトを実行した結果を出力するので，その内容を確認します。

1.3.1 バッチジョブを自動実行するときの処理の流れ（JP1/AJS と連携する場合）

JP1/Advanced Shell を使ったバッチジョブの運用では，ジョブスケジューラの JP1/AJS から実行環境を呼び出して，バッチジョブを自動実行できます。JP1/Advanced Shell は，ユーザーの業務アプリケーションの実行を制御するジョブコントローラとしての機能を提供します。JP1/Advanced Shell の業務アプリケーションに対する位置づけを次の図に示します。

図 1-5　JP1/Advanced Shell の業務アプリケーションに対する位置づけ

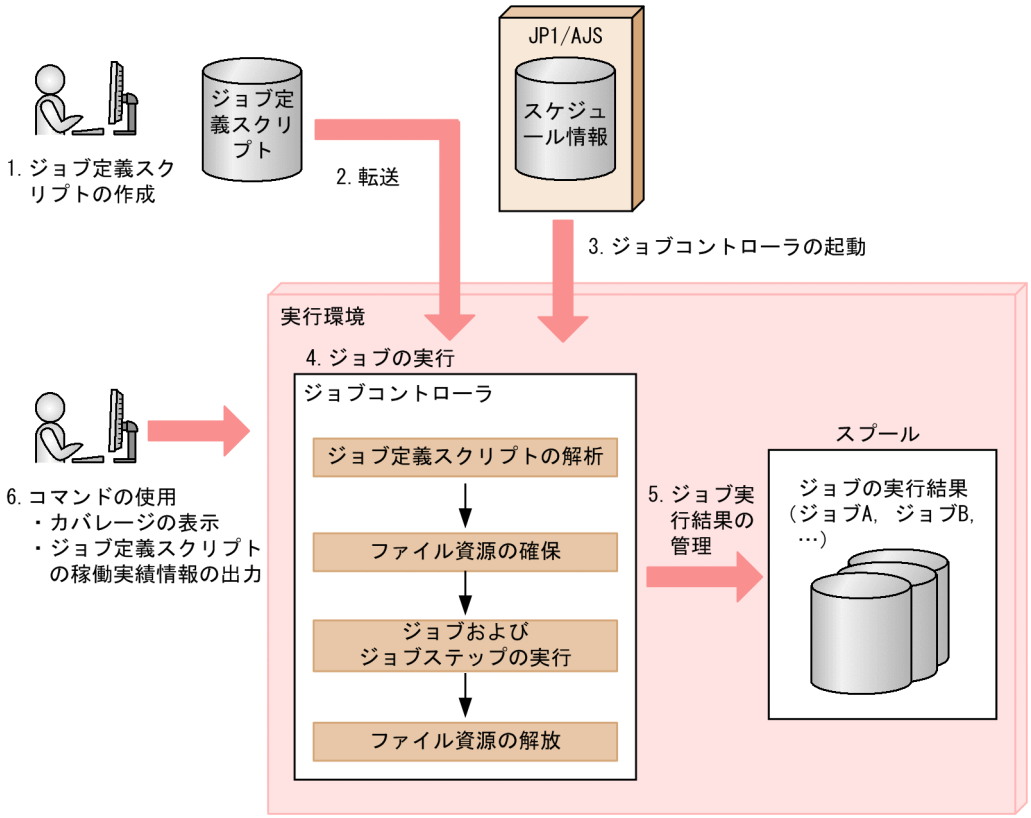


JP1/AJS と連携した場合，バッチジョブを実行するスケジュールを登録してバッチジョブを実行できます。

ジョブが定義されたジョブ定義スクリプトは，ジョブコントローラで解析されます。ジョブコントローラは，入出力装置や各種システム資源の割り当て，および解放処理をして，バッチジョブの実行・終了を制御します。また，JP1/Advanced Shell は，このジョブ定義スクリプトを実行し，実行結果をスプールに集めて，一元管理できます。

JP1/Advanced Shell の運用の流れを次の図に示します。図中の番号 3 が JP1/AJS の処理内容で，番号 4～6 が JP1/Advanced Shell が処理する内容です。

図 1-6　JP1/Advanced Shell の運用の流れ（JP1/AJS と連携する場合）



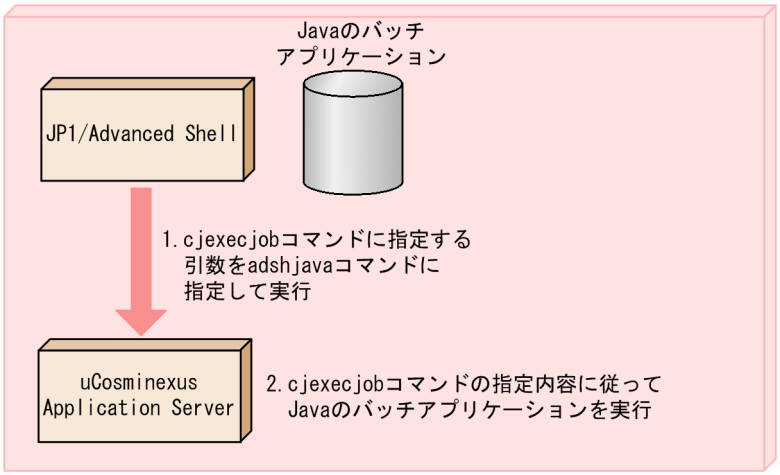
1. ジョブ定義スクリプトを作成しておきます。
2. JP1/Advanced Shell の実行環境にジョブ定義スクリプトを転送します。
3. JP1/AJS で登録されているスケジュールに従い，ジョブコントローラが起動されます。
4. 1 で作成したジョブ定義スクリプトの内容に従い，次に示す順序でジョブコントローラがバッチジョブを実行します。

   ジョブ定義スクリプトの解析→ファイル資源の確保→ジョブおよびジョブステップの実行→ファイル資源の解放
5. バッチジョブの実行結果をスプールに集めて，一元管理します。
6. 必要に応じて，カバレージ情報の表示やジョブ定義スクリプトの稼働実績情報の出力をコマンドで実行します。

1.3.2　JP1/Advanced Shell の機能を使って Java のバッチアプリケーションを実行するときの処理の流れ【Windows，Linux(R)，AIX，HP-UX 限定】

JP1/Advanced Shell が提供する adshjava コマンドを使用して，Java のバッチアプリケーションを実行できます。この場合の処理の流れを次の図に示します。

図 1-7　JP1/Advanced Shell が提供する adshjava コマンドを使用して Java のバッチアプリケーションを実行する場合の処理の流れ



1. adshjava コマンドは，uCosminexus Application Server に対して Java のバッチアプリケーションの実行を指示します。
2. adshjava コマンドから指示された内容に従って，uCosminexus Application Server は Java のバッチアプリケーションを実行します。

また，adshjava コマンドを使用して Java のバッチアプリケーションを実行した場合，ジョブを強制終了させると，adshjava コマンドは uCosminexus Application Server によって実行中の Java のバッチアプリケーションに対して cjkilljob コマンドを実行し，Java のバッチアプリケーションを自動的に停止させます。

詳細については，「3.3　adshjava コマンドを使用して Java のバッチアプリケーションを実行する【Windows，Linux，AIX，HP-UX 限定】」を参照してください。

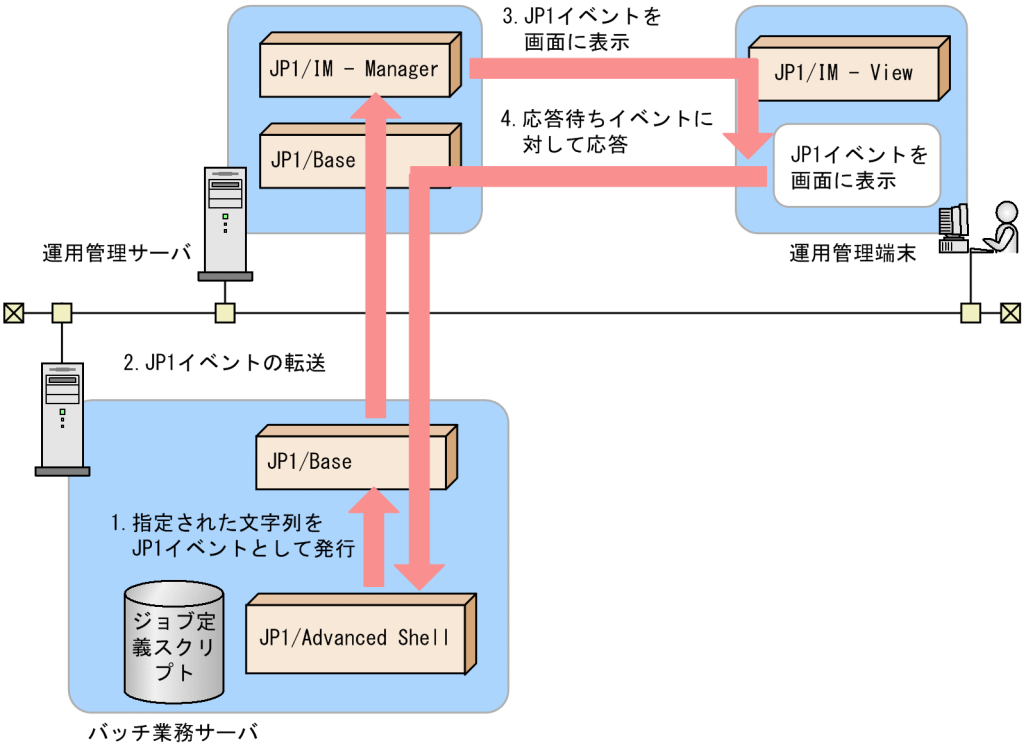
1.3.3　ユーザー応答機能を使用するときの処理の流れ

ユーザー応答機能を使用すると，ジョブ定義スクリプトから次のことができます。

- バッチジョブの情報を運用者に通知する。
- 運用者がジョブ定義スクリプトに対して応答する。

ユーザー応答機能は JP1/IM と連携して，指定した文字列を JP1 イベントとして発行します。ユーザー応答機能を使用する場合の処理の流れを次の図に示します。

図 1-8　ユーザー応答機能を使用する場合の処理の流れ



1. ジョブ定義スクリプトで，文字列をコマンドに指定して実行すると，指定された文字列を JP1 イベントとして発行します。
2. 発行された JP1 イベントは，JP1/Base によって，指定された運用管理サーバに転送されます。
3. JP1/IM - View の画面に，指定された文字列が表示されます。
4. 応答待ちイベントの場合，運用者は応答を入力できます。
   運用者から入力された応答は，ジョブ定義スクリプトで指定したシェル変数に格納されます。

詳細については，「3.8　ユーザー応答機能を使用する」を参照してください。

1.3.4　アプリケーション実行エージェント機能を使用するときの処理の流れ

アプリケーション実行エージェント機能は，ジョブコントローラから独立して動作することが可能であり，次のことができます。

- JP1/AJS のジョブの実行で Excel やユーザーが VC++などで独自に作った対話操作のプログラムなどを実行できます。
  <実行例>
  1. Excel を新規で起動して，Excel 上で日付，担当，作業内容確認のチェックボックスなどの特定情報を入力してから Excel 全体を閉じて，後続ジョブで特定のフォルダに移動させます。
  2. ユーザーが VC++などで独自に作った対話プログラムを実行し，OK ボタンを押すなどして対話 GUI が終了したらジョブが継続します。

- JP1/AJS から GUI 画面を表示し，自動応答するツールを使用する機能を持つほかの製品のプログラムを実行できます。

  <実行例>

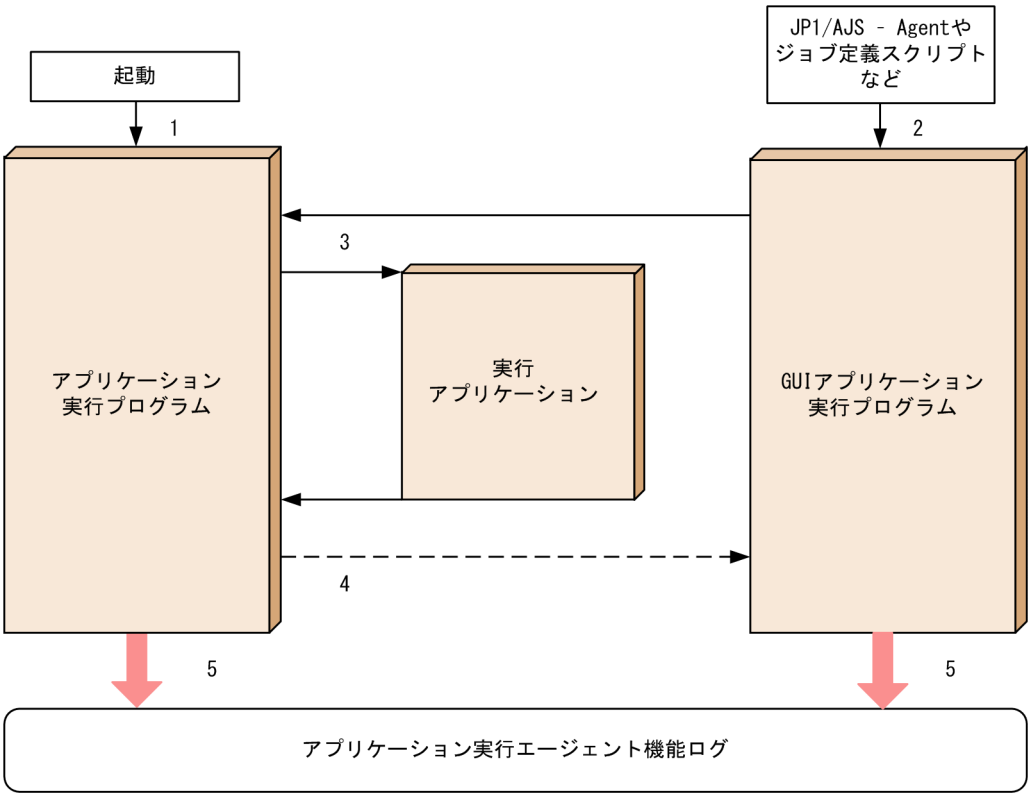
  1. JP1/AJS の PC ジョブの定義に指定する JP1/AS の GUI アプリケーション実行プログラムの引数に，ほかの製品のプログラムを指定します（アプリケーション実行エージェント機能のカスタムジョブを使用することもできます）。
  2. 1 の PC ジョブを実行すると，JP1/AJS - Agent で GUI 画面を表示して自動応答し，ジョブが終了します。

アプリケーション実行エージェント機能は，アプリケーション実行エージェントプログラムと GUI アプリケーション実行プログラムが連携して，ユーザーが指定した実行アプリケーションを実行します。アプリケーション実行エージェント機能を使用する場合の処理の流れを次に示します。

詳細については，「3.12　アプリケーション実行エージェント機能を使用する【Windows 実行環境限定】」を参照してください。

(1)　実行アプリケーションの終了を待つ場合【Windows 実行環境限定】

実行アプリケーションの終了を待つ場合の処理の流れを次に示します。

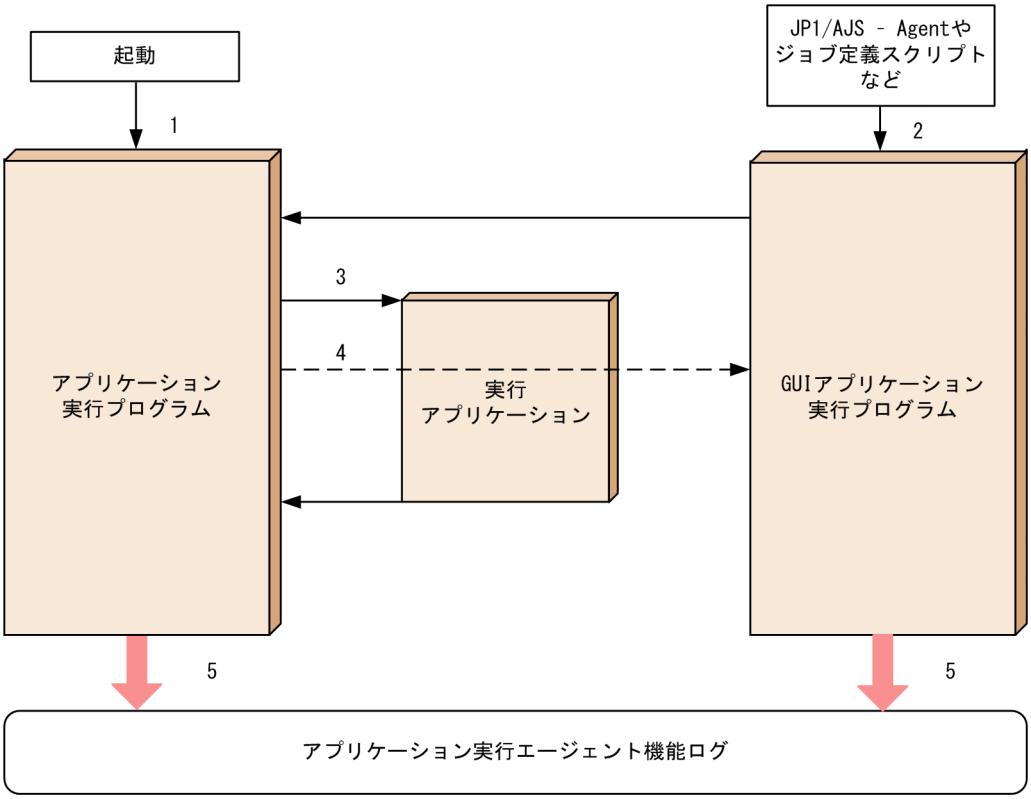


1. Windows のスタートアップなど，あらかじめアプリケーション実行エージェントプログラムを実行ユーザーごとに起動しておきます（1）。

2. 実行アプリケーションを動作させることが必要となった場合，引数-w を指定し，GUI アプリケーション実行プログラムを起動します（2）。
3. アプリケーション実行エージェントプログラムは，GUI アプリケーション実行プログラムが起動すると実行アプリケーションを起動します（3）。
4. 実行アプリケーションが終了すると GUI アプリケーション実行プログラムは，終了します（点線の矢印の流れ）（4）。
5. アプリケーション実行ログは，アプリケーション実行エージェントプログラム，および GUI アプリケーション実行プログラム実行時のログを格納します（5）。

(2)　実行アプリケーションの終了を待たない場合【Windows 実行環境限定】

実行アプリケーションの終了を待たない場合の処理の流れを次に示します。



1. Windows のスタートアップなど，あらかじめアプリケーション実行エージェントプログラムを実行ユーザーごとに起動しておきます（1）。
2. 実行アプリケーションを動作させることが必要となった場合，引数-n を指定し，GUI アプリケーション実行プログラムを起動します（2）。
3. アプリケーション実行エージェントプログラムは，GUI アプリケーション実行プログラムが起動すると実行アプリケーションを起動します（3）。
4. 実行アプリケーションを起動すると GUI アプリケーション実行プログラムは，終了します（点線の矢印の流れ）（4）。

5. アプリケーション実行ログは，アプリケーション実行エージェントプログラム，および GUI アプリケーション実行プログラム実行時のログを格納します（5）。

1.4　クラスタシステムでの運用の概要

クラスタシステムとは，複数のサーバシステムを用いて 1 つのシステムとして運用するシステムで，1 つのサーバで障害が発生しても，別のサーバで業務を継続できるように構成するシステムです。クラスタシステムでは，ホストを次のように分類します。

- 業務の実行中のサーバ（実行系サーバ）
- 実行系サーバの障害に備えて，業務を引き継げるよう待機しているサーバ（待機系サーバ）

業務を実行している実行系サーバに障害が発生した場合は，待機系サーバに業務を引き継ぐことができます。この障害時に業務を引き継ぐ機能のことを系切り替えと呼びます。また，系切り替え時にフェールオーバーの単位となる論理的なサーバのことを論理ホストと呼びます。

クラスタシステムで実行されるアプリケーションは，系切り替え後も業務を継続するために，論理ホスト環境で動作させる必要があります。論理ホストで動作するアプリケーションは，物理的なサーバに依存しないで，任意のサーバで動作できます。

論理ホストは次の要素で構成されています。デーモン・サービスとして動作するアプリケーションは，共有ディスクにデータを格納し，論理 IP アドレスで通信します。

表 1-2　論理ホストの構成要素

論理ホストの構成要素	構成要素の説明
デーモン・サービス	クラスタシステムで実行する JP1/AJS や JP1/Advanced Shell などのアプリケーションです。 実行系サーバの論理ホストで障害が発生すると，待機系サーバの論理ホストで同じ名称のデーモン・サービスを起動します。
共有ディスク	実行系サーバと待機系サーバの両方に接続されたディスク装置です。 系切り替え時に引き継ぐ情報（定義情報，実行状況など）を保存すると，実行系サーバの論理ホストで障害が発生した場合，待機系サーバが共有ディスクへの接続を引き継ぎます。
論理 IP アドレス	論理ホストの動作中に割り当てられる IP アドレスです。 実行系サーバで障害が発生したときは，同じ論理 IP アドレスの割り当てを待機系サーバが引き継ぎます。そのため，クライアントからは同じ IP アドレスでアクセスでき，1 つのサーバが常に動作しているように見えます。

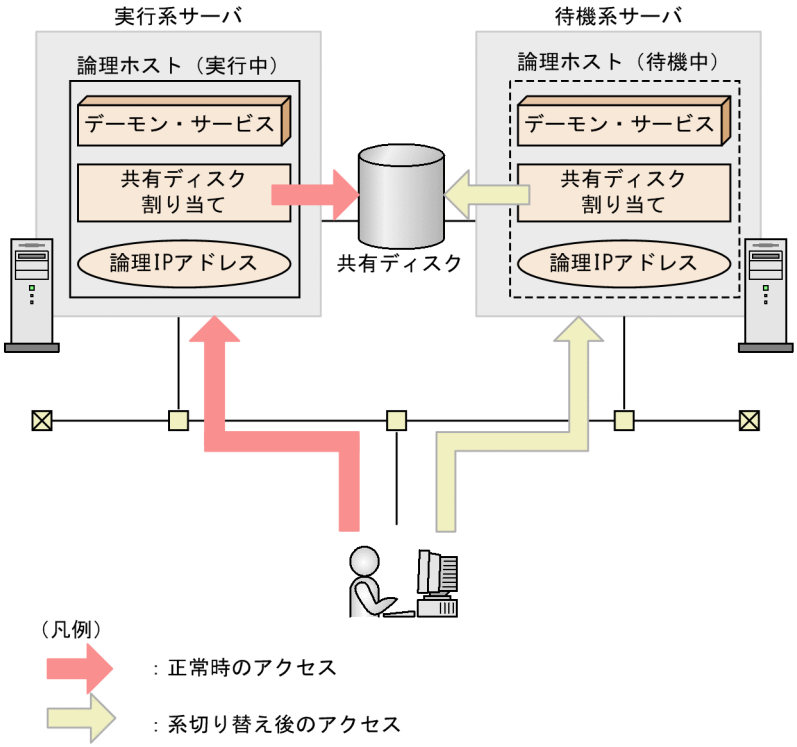
重要

このマニュアルでは，系切り替え時にフェールオーバーの単位となる論理的なサーバのことを「論理ホスト」という用語を使いますが，クラスタソフトやアプリケーションによっては「グループ」や「パッケージ」などの用語が使われています。クラスタソフトのマニュアルなどを参照し，対応する用語を確認してください。

また，系切り替え時にフェールオーバーの単位となる論理的なサーバを論理ホストと呼ぶのに対して，物理的なサーバを「物理ホスト」と呼びます。

正常時および系切り替え後のアクセスを次の図に示します。

図 1-9　正常時および系切り替え後のアクセス



図の内容について次に示します。

- 正常時のアクセス

  実行系サーバが稼働している間は，実行系サーバで共有ディスクや論理 IP アドレスが割り当てられてデーモン・サービスが動作します。

- 系切り替え後のアクセス

  実行系サーバで障害が発生した場合，待機系サーバが共有ディスクと論理 IP アドレスを引き継ぎ，実行系サーバと同じデーモン・サービスを起動します。系切り替えによって物理的なサーバが変わった場合でも，待機系サーバが共有ディスクと論理 IP アドレスを引き継ぐため，クライアントには同じ IP アドレスのサーバが動作しているように見えます。

JP1/Advanced Shell を論理ホスト環境で動作させるためには，系切り替え時に引き継ぎが必要なデータを格納するための共有ディスク，および論理 IP アドレスが必要となります。また，ユーザー応答機能を使用する場合は，クラスタソフトがユーザー応答機能管理デーモン・サービスの，起動・動作監視・停止を制御できるように，クラスタソフトで設定する必要があります。

論理ホスト環境で実行される JP1/Advanced Shell は，共有ディスクに格納したデータを使用し，系切り替え時に実行系サーバから待機系サーバにジョブを実行する環境を引き継ぐことができます。そのために JP1/Advanced Shell はスプールを共有ディスクに格納する必要があります。ただし，系切り替え時に実行中のジョブを継続して実行することはできません。

JP1/Advanced Shell でクラスタ運用に対応するための設定については，「2.9　クラスタ構成で運用する」を参照してください。

1.5　機能概要

JP1/Advanced Shell で利用できる機能を次に示します。

表 1-3　JP1/Advanced Shell の機能

機能	関連項目	参照先
ジョブの実行環境を定義する	ジョブ実行に必要な環境変数を設定する	2.5　環境変数を設定する
	環境ファイルを設定する	2.6　JP1/Advanced Shell の環境情報を設定する，7.　環境ファイルで設定するパラメーター
シェルスクリプトの文法に従ってジョブ定義スクリプトを作成する	ジョブ定義スクリプトを構成する基本要素	5.1　ジョブ定義スクリプトを構成する基本要素
	条件判定	5.2　条件判定，5.4　条件判定と算術演算の優先順位，9.6　スクリプト制御文
	算術演算	5.3　算術演算，5.4　条件判定と算術演算の優先順位
	シェル変数	5.1.2　変数，5.5　シェル変数
	シェルオプション	5.6　シェルオプション
ジョブ定義スクリプトからファイルを使用する	通常ファイル	5.9.1　通常ファイルの割り当ておよび後処理をする，9.5　スクリプト拡張コマンド
	一時ファイル	5.9.2　一時ファイルの割り当ておよび後処理をする，9.5　スクリプト拡張コマンド
	プログラム出力データファイル	5.9.3　プログラム出力データファイルの割り当てをする，9.5　スクリプト拡張コマンド
ジョブの実行を制御する	ジョブ名を宣言する	5.8.1　ジョブ名を宣言する，9.5　スクリプト拡張コマンド

機能	関連項目	参照先
ジョブの実行を制御する	ジョブの打ち切り条件を定義する	5.8.2　ジョブの打ち切り条件を定義する，9.5　スクリプト拡張コマンド
	ジョブステップを開始または終了する	5.8.3　ジョブステップを定義する，9.5　スクリプト拡張コマンド
	任意のコマンドを常に正常終了したと見なすように定義する	(2)　常に正常終了するコマンドを定義する，9.5　スクリプト拡張コマンド
	スクリプト拡張コマンドの終了コードを定義する	2.6.9　スクリプト拡張コマンドの終了コードを定義する，7.3　環境設定パラメーター
	外部スクリプトを呼び出す	5.8.6　実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルを呼び出す，9.5　スクリプト拡張コマンド
	子孫ジョブを起動する	2.6.5　子孫ジョブとして起動するファイルを定義する，(1)　ルートジョブと子孫ジョブ，3.2.3　ジョブ定義スクリプトを子孫ジョブとして実行する，5.1.11　外部コマンドの指定，7.3　環境設定パラメーター
	ジョブを強制終了する	2.6.21　ジョブ強制終了時にユーザー固有の後処理を実行する，3.11　ジョブを強制終了する，7.3　環境設定パラメーター
シェルスクリプト中でジョブの情報を取得する	ジョブステップの終了コードが設定されたシェル変数を使用する	5.5.1　JP1/Advanced Shell が設定するシェル変数
	ジョブの情報が設定された環境変数を使用する	2.5　環境変数を設定する，5.7　ジョブ情報の環境変数

機能	関連項目	参照先
エディタを使ってジョブ定義スクリプトを作成する※1		4.　JP1/Advanced Shell - Developer を使用する【Windows 限定】，5.　ジョブ定義スクリプトの作成
コマンドを実行する	シェル運用コマンド	8.3　シェル運用コマンド
	UNIX 互換コマンド	2.6.6　UNIX 互換コマンドを使用するための定義をする，8.4　UNIX 互換コマンド，8.5　UNIX 互換コマンド（スクリプト形式）【Windows 限定】
	シェル標準コマンド	9.3　シェル標準コマンド
	シェル拡張コマンド	9.4　シェル拡張コマンド
	スクリプト拡張コマンド	9.5　スクリプト拡張コマンド
	スクリプト制御文	9.6　スクリプト制御文
	スクリプト予約語コマンド	9.7　スクリプト予約語コマンド
カスタムジョブを登録する※2		2.7.1　JP1/AJS - View でカスタムジョブを登録する
ユーザー応答機能を使用する	任意の文字列を JP1 イベントとして発行する	9.4　シェル拡張コマンド
	任意の文字列を応答待ちイベントとして発行する	9.4　シェル拡張コマンド
	ユーザー応答機能管理デーモン・サービスを起動する	8.3　シェル運用コマンド
ジョブ定義スクリプトの稼働実績情報を採取する※2	ジョブ定義スクリプトの稼働実績情報を蓄積する	7.3　環境設定パラメーター
	ジョブ定義スクリプトの稼働実績情報を出力する	8.3　シェル運用コマンド
同じジョブ定義スクリプトを異なるプラットフォームで使用する	Windows と UNIX の両方で使用できるようにジョブ定義スクリプトを変換する	2.6.2　パス名を変換する，2.6.3　ファイルの入出力時にファイルパスを変換する，2.6.4　コマンド実行時に引数

機能	関連項目	参照先
同じジョブ定義スクリプトを異なるプラットフォームで使用する	Windows と UNIX の両方で使用できるようにジョブ定義スクリプトを変換する	を変換する，2.6.12 ジョブ定義スクリプトを UNIX から Windows へ移行する，7.3　環境設定パラメーター
	UNIX 互換コマンドを使用する	2.6.6　UNIX 互換コマンドを使用するための定義をする，8.4 UNIX 互換コマンド，8.5　UNIX 互換コマンド（スクリプト形式）【Windows 限定】
スプールジョブを削除する		3.9　スプールジョブを削除する，8.3　シェル運用コマンド
カバレージ情報を採取する	カバレージ情報を取得する	3.10　カバレージ情報を取得する，8.3　シェル運用コマンド，付録 A　カバレージ情報を取得する対象
	カバレージ情報を表示する	3.10　カバレージ情報を取得する，8.3　シェル運用コマンド
	カバレージ情報をマージする	3.10　カバレージ情報を取得する，8.3　シェル運用コマンド
	カバレージ情報の取得を常に有効にする	3.10　カバレージ情報を取得する， 8.3　シェル運用コマンド
ジョブ定義スクリプトをデバッグする	CUI でデバッグする※3	6.1.2　CUI でのデバッグ【UNIX 限定】，6.1.4　デバッガのコマンド一覧【UNIX 限定】，6.1.5　ジョブ定義スクリプトの構成要素に対する停止可否，6.2　CUI のデバッガ【UNIX 限定】
	GUI でデバッグする※1	4.2.2　デバッグモード，4.4.6　デバッグをする，6.1.1　GUI でのデバッグ

機能	関連項目	参照先
ジョブ定義スクリプトをデバッグする	GUI でデバッグする※1	【Windows 限定】，6.1.3　GUI デバッガの機能一覧 【Windows 限定】，6.1.5　ジョブ定義スクリプトの構成要素に対する停止可否
ジョブ実行ログを出力する		1.1.3　バッチジョブの実行結果の一元管理による運用性・保守性の向上，3.5　ジョブ実行ログ
アプリケーション実行エージェント機能を使用する。※1　※2		2.11　アプリケーション実行エージェント機能を設定する 【Windows 実行環境限定】 8.3　シェル運用コマンド 9.4　シェル拡張コマンド
トラブルシューティング	ジョブ実行ログ，システム実行ログ，トレースログなどの資料を採取する	11.　トラブルシューティング
	JP1/IM が使用できない場合に，応答要求メッセージに応答する	8.3　シェル運用コマンド，11.　トラブルシューティング

注※1

開発環境だけで使用できます。

注※2

実行環境だけで使用できます。

注※3

UNIX 版だけで使用できます。

2 JP1/Advanced Shell を利用するための準備

プログラムのインストール先と種類，前提条件，インストール，環境情報の設定，カスタムジョブの登録などの JP1/Advanced Shell を利用するために必要な項目について説明します。

2.1　プログラムのインストール先ディレクトリ

JP1/Advanced Shell のプログラムのインストール先は，使用する OS によって異なります。Windows 環境の場合は，インストール先をデフォルトから変更できます。UNIX 環境の場合は，固定のディレクトリになります。

この節では，JP1/Advanced Shell のプログラムのインストール先ディレクトリと，JP1/Advanced Shell が出力・参照する各種ファイルを格納するディレクトリの構成について説明します。

2.1.1　インストール先フォルダ【Windows 限定】

(1)　インストール先フォルダ

Windows ではインストール先として任意のフォルダを指定できます。指定したインストール先フォルダの下には環境ごとに次のフォルダが生成されます。

インストールする環境	インストール先	備考
実行環境	インストール先フォルダ ¥JP1ASE	サーバにインストールすることを推奨します。
開発環境	インストール先フォルダ ¥JP1ASD	クライアント PC にインストールすることを推奨します。
実行環境に含まれるカスタムジョブ定義プログラム（JP1/Advanced Shell - Custom Job）	インストール先フォルダ ¥JP1ASV	JP1/AJS - View をインストールしている運用管理端末にインストールします。

インストール先フォルダの構成を次に示します。実際には選択した環境のフォルダだけが生成されます。

```
インストール先フォルダ※1
  ├──JP1ASD                          ：開発環境フォルダ
  │      ├──bin                      ：プログラムのフォルダ
  │      ├──cmd                      ：UNIX互換コマンドのフォルダ
  │      ├──doc                      ：ヘルプのフォルダ
  │      │  ├──en                    ：ヘルプ（英語版）のフォルダ
  │      │  └──ja                    ：ヘルプ（日本語版）のフォルダ
  │      ├──maintenance              ：障害発生時に使用するフォルダ
  │      ├──parts                    ：スクリプト開発部品のフォルダ
  │      │  ├──en                    ：スクリプト開発部品（英語版）のフォルダ
  │      │  └──ja                    ：スクリプト開発部品（日本語版）のフォルダ
  │      └──sample                   ：サンプルデータのフォルダ
  ├──JP1ASE                          ：実行環境フォルダ
  │      ├──bin                      ：プログラムのフォルダ
  │      ├──cmd                      ：UNIX互換コマンドのフォルダ
  │      ├──doc                      ：ヘルプのフォルダ
  │      │  ├──en                    ：ヘルプ（英語版）のフォルダ
  │      │  └──ja                    ：ヘルプ（日本語版）のフォルダ
  │      ├──maintenance              ：障害発生時に使用するフォルダ
  │      ├──parts                    ：スクリプト開発部品のフォルダ
  │      │  ├──en                    ：スクリプト開発部品（英語版）のフォルダ
```

```
│      │  └──ja                          ：スクリプト開発部品（日本語版）のフォルダ
│      ├──sample                         ：サンプルデータのフォルダ
│      └──util─setup.exe                 ：カスタムジョブ定義プログラムのインストーラ
└──JP1ASV                                ：カスタムジョブ定義プログラムのフォルダ
       ├──bin                            ：プログラムのフォルダ
       ├──doc─ja─help─INDEX.HTM          ：ヘルプ（マニュアル）
       ├──image─custom                   ：カスタムジョブアイコンのフォルダ
       │  ├──CUSTOM_PC_ADSHPC.gif        ：PCジョブのカスタムジョブアイコン
       │  ├──CUSTOM_PC_ADSHUX.gif        ：UNIXジョブのカスタムジョブアイコン
       │  └──CUSTOM_PC_ADSHAPPEXEC.gif   ：GUIアプリケーション実行ジョブのカスタムジョブ
アイコン※2
       └──maintenance                    ：障害発生時に使用するフォルダ
```

注※1

インストール先フォルダに&[]{}^=;!'+,`~#%の記号を含まないでください。記号を含むインストール先フォルダにインストールした場合，正常に動作しません。

注※2

JP1/AJS3 - View 11-00 より古い製品がインストールされている場合にカスタムジョブ定義プログラムをインストールするときは，カスタムジョブアイコンを次のフォルダにコピーする必要があります。

```
JP1/AJS - Viewのインストール先フォルダ¥image¥custom
```

(2)　トレース出力先フォルダおよびシステム環境ファイルの作成フォルダ

トレース出力先フォルダおよびシステム環境ファイルを作成するフォルダは，共通アプリケーションフォルダに作成されます。

```
共通アプリケーションフォルダ
  └──Hitachi─JP1AS─JP1ASD       ：開発環境フォルダ
                 │     ├──conf   ：システム環境ファイル格納フォルダ
                 │     ├──trace  ：トレース出力先フォルダ
                 │     └──uxpl   ：ログフォルダ
                 ├──JP1ASE       ：実行環境フォルダ
                 │     ├──conf   ：システム環境ファイル格納フォルダ
                 │     ├──trace  ：トレース出力先フォルダ
                 │     └──uxpl   ：ログフォルダ
                 ├──JP1ASV       ：カスタムジョブ定義プログラムのフォルダ
                 │     └──trace  ：トレース出力先フォルダ
                 └──misc         ：製品共通ライブラリのフォルダ
                       ├──trace  ：トレース出力先フォルダ
                       └──uxpl   ：ログフォルダ
```

(3)　システム実行ログ，スプールおよび一時ファイル

システム実行ログ，スプールおよび一時ファイルのフォルダは，共有ドキュメントフォルダに作成されます。

```
共有ドキュメントフォルダ
  └──Hitachi─JP1AS─JP1ASD       ：開発環境フォルダ
                 │     ├──log    ：システム実行ログフォルダ
                 │     ├──spool  ：スプールフォルダ
```

```
│       └──temp  ：一時ファイル用フォルダ
├──JP1ASE       ：実行環境フォルダ
│       ├──log   ：システム実行ログフォルダ
│       ├──appexec：アプリケーション実行エージェント機能ログフォルダ
│       ├──spool：スプールフォルダ
│       └──temp  ：一時ファイル用フォルダ
└──misc         ：製品共通ライブラリのフォルダ
        └──log   ：ログフォルダ
```

(4)　プログラムの種類

JP1/Advanced Shell で使用する主なプログラムの格納場所とファイル名を次の表に示します。

表 2-1　JP1/Advanced Shell で使用する主なプログラム【Windows 限定】

格納フォルダ	ファイル名	プログラムの概要 （括弧内：アイコン）	説明
インストール先フォルダ¥各環境のフォルダ※1¥bin	adshappagent.exe	アプリケーション実行エージェントプログラム	実行環境で実行アプリケーションを実行するプログラム。このプログラムのスタートアップへの登録，解除も行います。
	adshappexec.exe	GUI アプリケーション実行プログラム（ ）	実行環境で実行アプリケーションを実行する場合に，アプリケーション実行エージェントプログラムに連絡するプログラムです。
	adshchmsg.exe	障害発生時の応答要求メッセージに対する手動応答	障害発生時に，応答要求メッセージに対して手動で応答する場合に使用するコマンドです。Administrators 権限を持つユーザーが使用します。
	adshctmj.exe	JP1/Advanced Shell 実行定義プログラム（ ）	カスタムジョブ定義プログラムで JP1/Advanced Shell の実行環境を定義するプログラムです。
	adshctmjapp.exe	GUI アプリケーション実行定義プログラム（ ）	カスタムジョブ定義プログラムで GUI アプリケーションの実行環境を定義するプログラムです。
	adshctmjapp.bat	GUI アプリケーション実行定義プログラム	カスタムジョブ定義プログラムで GUI アプリケーションの実行環境を定義するプログラムです。
	adshctmjpc.bat	PC ジョブの JP1/Advanced Shell 実行定義プログラム	カスタムジョブ定義プログラムで PC ジョブの JP1/Advanced Shell の実行環境を定義するプログラムです。
	adshctmjunix.bat	UNIX ジョブの JP1/Advanced Shell 実行定義プログラム	カスタムジョブ定義プログラムで UNIX ジョブの JP1/Advanced Shell の実行環境を定義するプログラムです。
	adshcvmerg.exe	カバレージ情報のマージ	カバレージ情報をマージするコマンドです。実行環境と開発環境で使用できます。

格納フォルダ	ファイル名	プログラムの概要 （括弧内：アイコン）	説明
インストール先フォルダ¥各環境のフォルダ※1¥bin	adshcvshow.exe	コマンドからのカバレージ情報の表示	カバレージ情報を表示するコマンドです。実行環境と開発環境で使用できます。
	adshcvview.exe	エディタからのカバレージ情報の表示	カバレージ情報を表示するプログラムです。開発環境のエディタからカバレージ情報を表示できます。
	adshedit.exe	JP1/Advanced Shell エディタ （ ）	開発環境でジョブ定義スクリプトを編集するエディタです。アイコンをダブルクリックすると，JP1/Advanced Shell エディタが開きます。
	adshesub.exe	エディタでのデバッグ	開発環境でジョブ定義スクリプトをデバッグするプログラムです。このプログラムは adshedit.exe から自動的に起動されます。
	adshevtout.exe	ジョブ定義スクリプトの稼働実績情報の出力	実行環境で，ジョブ定義スクリプトの稼働実績情報を CSV 形式で出力するコマンドです。
	adshexec.exe	バッチジョブの実行	ジョブ定義スクリプトを解析して実行を制御するジョブコントローラを起動するコマンドです。
	adshexecsub.exe		実行環境でバッチジョブを実行するコマンドです。このコマンドはadshexec.exe から自動的に起動されます。
	adshfile.exe	ファイルの後処理登録	指定したファイルをジョブステップまたはジョブの終了時にどのように処置するかを定義するコマンドです。実行環境と開発環境で使用できます。
	adshhk.exe	スプールジョブの削除	スプールジョブを削除するコマンドです。実行環境と開発環境で使用できます。
	adshlsmsg.exe	障害発生時の応答要求メッセージの一覧表示	障害発生時に，応答要求メッセージの一覧を表示するコマンドです。Administrators 権限を持つユーザーが使用します。
	adshmsvcd.exe	ユーザー応答機能管理サービス	ユーザー応答機能のための共有メモリを管理するサービスを登録するコマンドです。開発環境で使用できます。Administrators 権限を持つユーザーが使用します。
	adshmsvce.exe		ユーザー応答機能のための共有メモリを管理するサービスを登録するコマンドです。実行環境で使用できます。Administrators 権限を持つユーザーが使用します。
インストール先フォルダ¥各環境のフォルダ※1¥cmd	awk.exe, basename.exe, cat.exe, cmp.exe, cp.exe, cut.exe, date.exe, diff.exe, dirname.exe, egrep.exe, expand.exe, expr.exe, find.exe,	UNIX 互換コマンド※2	UNIX のバッチ業務で主に使われるコマンドを Windows 環境で使用できるようにした，UNIX 互換コマンドです。実行環境と開発環境で使用できます。

格納フォルダ	ファイル名	プログラムの概要 （括弧内：アイコン）	説明
インストール先フォルダ¥各環境のフォルダ※1¥cmd	getopt.exe, grep.exe, gunzip.exe, gzip.exe, head.exe, hostname.exe, ln.exe, ls.exe, mkdir.exe, mv.exe, paste.exe, printf.exe, rm.exe, rmdir.exe, sed.exe, sleep.exe, sort.exe, split.exe, stat.exe, tail.exe, tar.exe, touch.exe, tr.exe, uname.exe, uniq.exe, wc.exe, which.exe, xargs.exe	UNIX 互換コマンド※2	UNIX のバッチ業務で主に使われるコマンドを Windows 環境で使用できるようにした，UNIX 互換コマンドです。実行環境と開発環境で使用できます。
インストール先フォルダ¥各環境のフォルダ※1¥maintenance	adshcollect.bat	資料の採取	トラブルシューティングで資料を採取するコマンドです。実行環境と開発環境で使用できます。

注※1

*各環境のフォルダ*は，開発環境フォルダの場合は「JP1ASD」，実行環境フォルダの場合は「JP1ASE」，カスタムジョブ定義プログラムのフォルダの場合は「JP1ASV」です。

注※2

UNIX 互換コマンドにはこのほかにchmod コマンド，su コマンドおよびwho コマンドがあります。chmod コマンド，su コマンドおよびwho コマンドを使用する場合は，JP1/Advanced Shell が提供するサンプルスクリプトファイルを「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」に示す手順で事前に編集する必要があります。

2.1.2　インストール先ディレクトリ【UNIX 限定】

(1)　インストール先ディレクトリ

UNIX の実行環境は，固定のディレクトリ（/opt/jp1as）にインストールされます。UNIX 環境には，開発環境はありません。

インストール先ディレクトリの構成を次に示します。

```
/opt/jp1as
  ├──bin                          ：プログラムのディレクトリ
```

```
    ├──cmd                          ：UNIX互換コマンドのディレクトリ
    ├──conf                         ：システム環境ファイル格納ディレクトリ
    ├──instlog                      ：インストールのログ情報のディレクトリ
    ├──lib                          ：ライブラリのディレクトリ
    │     └──nls                    ：メッセージカタログ格納ディレクトリ
    ├──log                          ：システム実行ログのディレクトリ
    ├──maintenance                  ：障害発生時に使用するディレクトリ
    ├──parts                        ：スクリプト開発部品のディレクトリ
    │     ├──en                     ：スクリプト開発部品（英語版）のディレクトリ
    │     └──ja                     ：スクリプト開発部品（日本語版）のディレクトリ
    ├──sample                       ：サンプルデータのディレクトリ
    ├──sbin                         ：システム管理者用プログラムのディレクトリ
    ├──system                       ：ユーザー応答機能管理デーモンが使用するディレクトリ
    ├──trace                        ：トレースのディレクトリ
    └──util──setup.exe              ：カスタムジョブ定義プログラムのインストーラ
```

(2)　スプールディレクトリと一時ファイル用ディレクトリ

スプールディレクトリと一時ファイル用ディレクトリは，次のディレクトリに作成されます。

```
/var/opt/jp1as──spool  ：スプールディレクトリ
             └──temp   ：一時ファイル用ディレクトリ
```

(3)　プログラムの種類

JP1/Advanced Shell で使用する主なプログラムの格納場所とファイル名を次の表に示します。

表 2-2　JP1/Advanced Shell で使用する主なプログラム【UNIX 限定】

格納ディレクトリ	ファイル名	プログラムの概要	説明
/opt/jp1as/bin	adshcvmerg	カバレージ情報のマージ	カバレージ情報をマージするコマンドです。
	adshcvshow	カバレージ情報の表示	カバレージ情報を表示するコマンドです。
	adshevtout	ジョブ定義スクリプトの稼働実績情報の出力	ジョブ定義スクリプトの稼働実績情報を CSV 形式で出力するコマンドです。
	adshexec	バッチジョブの実行	バッチジョブを実行するコマンドです。
	adshfile	ファイルの後処理登録	指定したファイルをジョブステップまたはジョブの終了時にどのように処置するかを定義するコマンドです。
	adshhk	スプールジョブの削除	スプールジョブを削除するコマンドです。
/opt/jp1as/cmd	awk，basename，cat，cmp，cp，cut，date，diff，dirname，	UNIX 互換コマンド	ジョブ定義スクリプトから UNIX コマンドを使用できるようにしたものです。

格納ディレクトリ	ファイル名	プログラムの概要	説明
/opt/jp1as/cmd	egrep, expand, expr, find, getopt, grep, gunzip, gzip, head, hostname, ln, ls, mkdir, mv, paste, printf, rm, rmdir, sed, sleep, sort, split, stat, tail, tar, touch, tr, uname, uniq, wc, which, xargs	UNIX 互換コマンド	ジョブ定義スクリプトから UNIX コマンドを使用できるようにしたものです。
/opt/jp1as/maintenance	adshcollect	資料の採取	トラブルシューティングで資料を採取するコマンドです。
/opt/jp1as/sbin	adshchmsg	障害発生時の応答要求メッセージに対する手動応答	障害発生時に，応答要求メッセージに対して手動で応答する場合に使用するコマンドです。スーパーユーザー権限を持つユーザーが使用します。
	adshlsmsg	障害発生時の応答要求メッセージの一覧表示	障害発生時に，応答要求メッセージの一覧を表示するコマンドです。スーパーユーザー権限を持つユーザーが使用します。
	adshmdctl	ユーザー応答機能管理デーモン	ユーザー応答機能のための共有メモリを管理するデーモンを起動および停止するコマンドです。スーパーユーザー権限を持つユーザーが使用します。

2.2　インストール前の検討事項

ここでは，システム構成，前提プログラム，関連プログラムおよび使用するファイルなど，インストール前の検討事項について説明します。

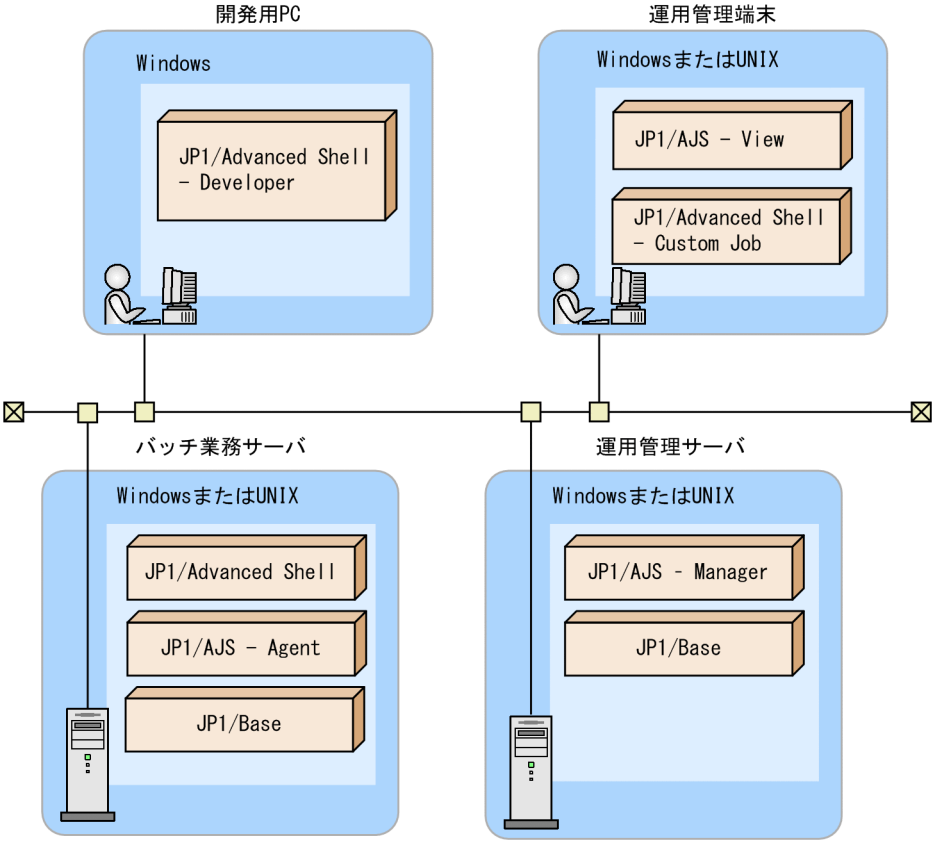
2.2.1　システム構成

JP1/Advanced Shell のシステム構成を実行形態ごとに説明します。

(1)　JP1/AJS からバッチジョブを実行する場合

JP1/AJS からバッチジョブを実行する場合のシステム構成を次に示します。

図 2-1　JP1/AJS からバッチジョブを実行する場合のシステム構成



システムの各構成要素の役割を次に説明します。

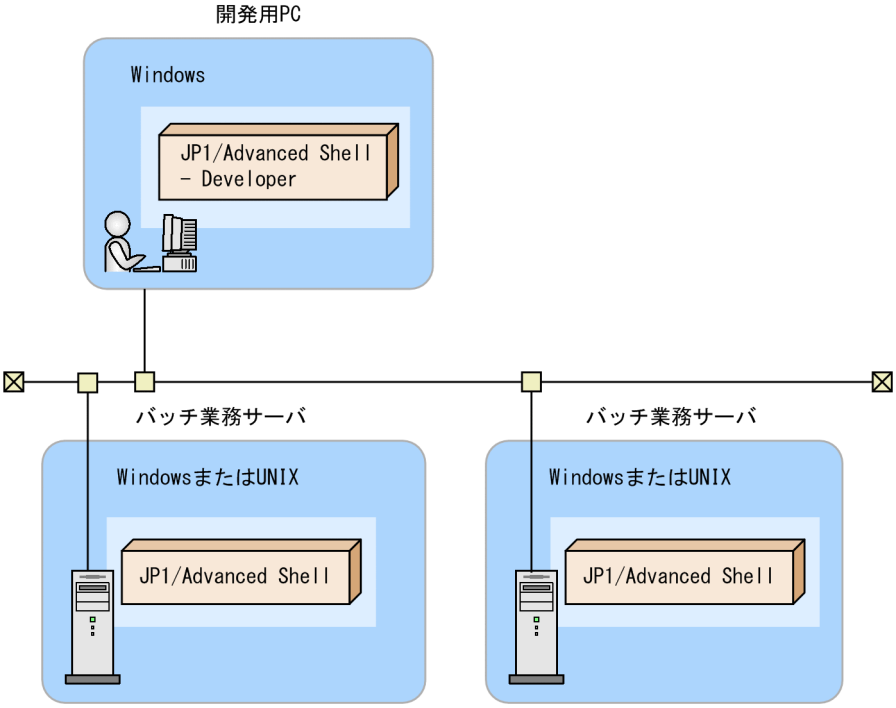
- 開発用 PC：JP1/Advanced Shell - Developer を使用してジョブ定義スクリプトを作成します。
- バッチ業務サーバ：ジョブ定義スクリプトを手動または自動実行します。
- 運用管理サーバ：実行されたジョブを管理します。

- 運用管理端末：JP1/AJS - View を使用してジョブの実行結果を表示したり，自動実行するジョブ定義スクリプトを定義したりします。運用管理端末で，JP1/Advanced Shell のジョブ定義を行うには JP1/Advanced Shell - Custom Job（カスタムジョブ定義プログラム）が必要になります。

(2)　手動でバッチジョブを実行する場合

手動でバッチジョブを実行する場合のシステム構成を次に示します。

図 2-2　手動でバッチジョブを実行する場合のシステム構成



システムの各構成要素の役割を次に説明します。

- 開発用 PC：JP1/Advanced Shell - Developer を使用してジョブ定義スクリプトを作成します。
- バッチ業務サーバ：ジョブ定義スクリプトを手動で実行します。

(3)　uCosminexus Application Server と連携する場合

uCosminexus Application Server と連携する場合のシステム構成は，スケジューリング機能を使用しない場合と使用する場合とで，次の図のように異なります。

図 2-3　uCosminexus Application Server と連携する場合のシステム構成（スケジューリング機能を使用しない場合）

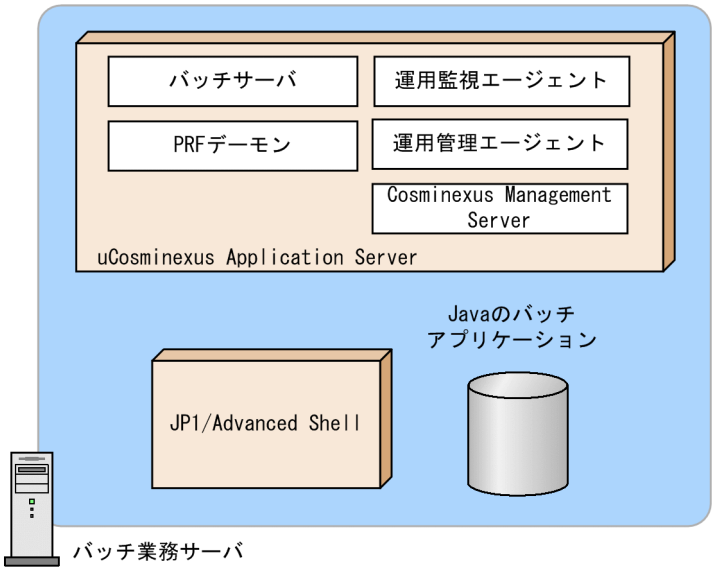
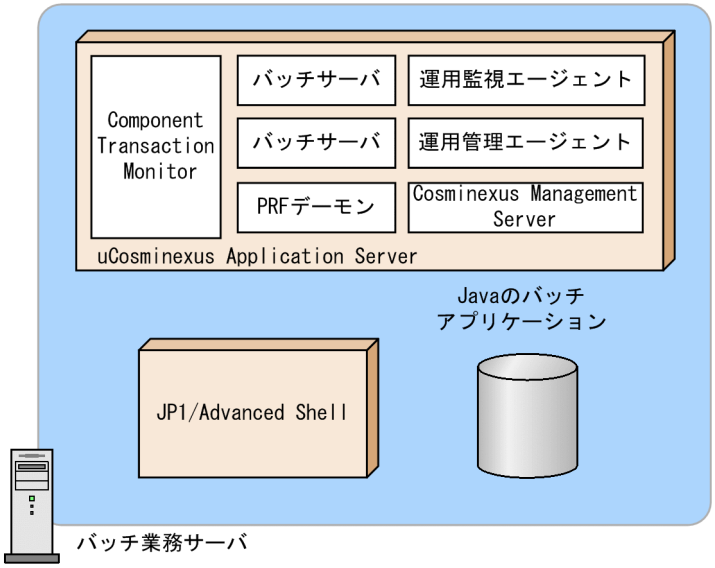


図 2-4　uCosminexus Application Server と連携する場合のシステム構成（スケジューリング機能を使用する場合）

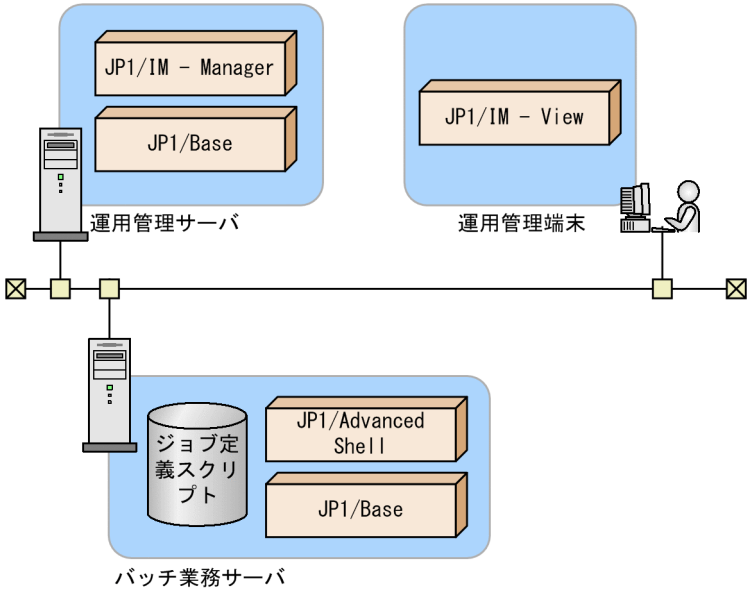


詳細については，「3.3　adshjava コマンドを使用して Java のバッチアプリケーションを実行する【Windows，Linux，AIX，HP-UX 限定】」を参照してください。

(4)　ユーザー応答機能を使用する場合

ユーザー応答機能を使用する場合のシステム構成を次の図に示します。

図 2-5　ユーザー応答機能を使用する場合のシステム構成



詳細については，「3.8　ユーザー応答機能を使用する」を参照してください。

2.2.2　環境ごとに必要なプログラム

JP1/Advanced Shell の前提プログラムと関連プログラムを説明します。

(1)　実行環境の前提プログラムと関連プログラム

(a)　実行環境の前提プログラム

実行環境の前提プログラムを次の表に示します。

表 2-3　実行環境の前提プログラム

サーバの種類	OS
JP1/Advanced Shell と同じバッチ業務サーバ【Windows 限定】	Windows
JP1/Advanced Shell と同じバッチ業務サーバ【UNIX 限定】	AIX, HP-UX, Linux，または Solaris

(b)　実行環境の関連プログラム

実行環境のサーバごとに関連するプログラムを表に示します。

表 2-4　実行環境の関連プログラム（JP1/AJS からバッチジョブを実行する場合）

サーバの種類	実施する処理	プログラム
JP1/Advanced Shell と同じバッチ業務サーバ	JP1/AJS からジョブ定義スクリプトを実行する	JP1/Base JP1/AJS - Agent※
運用管理サーバ	ジョブを管理する	JP1/Base JP1/AJS - Manager※
運用管理端末【Windows 限定】	ジョブの実行結果を表示する	JP1/AJS - View

注※

JP1/AJS - Manager には JP1/AJS - Agent の機能が含まれているため，JP1/Advanced Shell と同一サーバ上に JP1/AJS - Manager があれば，JP1/AJS - Agent は不要です。

表 2-5　実行環境の関連プログラム（uCosminexus Application Server と連携して Java のバッチアプリケーションを実行する場合）【Windows，Linux，AIX，HP-UX 限定】

サーバの種類	プログラム
JP1/Advanced Shell と同じバッチ業務サーバ	uCosminexus Application Server

表 2-6　実行環境の関連プログラム（ユーザー応答機能を使用する場合）

サーバの種類	プログラム
JP1/Advanced Shell と同じバッチ業務サーバ	JP1/Base
運用管理サーバ	JP1/IM - Manager JP1/Base
運用管理端末	JP1/IM - View

注

ユーザー応答機能の出力先に標準出力を指定してデバッグ実行する場合は，これらのプログラムは必須ではありません。

(2)　開発環境の前提プログラムと関連プログラム【Windows 限定】

(a)　開発環境の前提プログラム

開発環境の前提プログラムを次の表に示します。

表 2-7　開発環境の前提プログラム【Windows 限定】

サーバの種類	OS
JP1/Advanced Shell - Developer と同じ開発用 PC	Windows

(b) 開発環境の関連プログラム

開発環境の関連プログラムをサーバごとに示します。

表 2-8 開発環境の関連プログラム（ユーザー応答機能を使用する場合）

サーバの種類	プログラム
JP1/Advanced Shell - Developer と同じ開発用 PC	JP1/Base
運用管理サーバ	JP1/IM - Manager JP1/Base
運用管理端末	JP1/IM - View

注

ユーザー応答機能の出力先に標準出力を指定してデバッグ実行する場合は，これらのプログラムは必須ではありません。

(3) カスタムジョブ定義プログラムの前提プログラムと関連プログラム【Windows 限定】

(a) カスタムジョブ定義プログラムの前提プログラム

カスタムジョブ定義プログラムの前提プログラムを次の表に示します。カスタムジョブ定義プログラムは Windows 限定となりますが，Windows と UNIX のジョブの定義を作成できます。

表 2-9 カスタムジョブ定義プログラムの前提プログラム【Windows 限定】

サーバの種類	OS およびプログラム
JP1/Advanced Shell - Custom Job と同じ運用管理端末	• OS Windows • プログラム JP1/AJS - View

(b) カスタムジョブ定義プログラムの関連プログラム

カスタムジョブ定義開発環境の関連プログラムをサーバごとに示します。

表 2-10 開発環境の関連プログラム（uCosminexus Application Server と連携して Java のバッチアプリケーションを実行する場合）

サーバの種類	プログラム
JP1/Advanced Shell - Developer と同じ開発用 PC	uCosminexus Application Server

表 2-11　開発環境の関連プログラム（ユーザー応答機能を使用する場合）

サーバの種類	プログラム
JP1/Advanced Shell - Developer と同じ開発用 PC	JP1/Base
運用管理サーバ	JP1/IM - Manager JP1/Base
運用管理端末	JP1/IM - View

2.2.3　JP1/Advanced Shell で使用するファイル

(1)　JP1/Advanced Shell で使用するファイルの一覧

JP1/Advanced Shell で使用するファイルを次の表に示します。ファイルサイズが 2GB を超えられるかどうかは，「(2)　2GB を超過するファイル（ラージファイル）の扱い」を参照してください。

表 2-12　JP1/Advanced Shell で使用するファイル

ファイル名（アイコン）	拡張子	ファイルの位置	ファイルの内容
ジョブ定義スクリプトファイル（ ）	.ash	ユーザー任意	ジョブ定義スクリプトを保存するファイルです。ファイル名は，ユーザー任意に指定できます。
環境ファイル※1	.ase	ユーザー任意	JP1/Advanced Shell の環境情報を設定するファイルです。
システム環境ファイル	.ase	「システム環境ファイルの設定」を参照してください。	JP1/Advanced Shell の環境情報を設定するファイルです。
カバレージ情報ファイル	.asc	ユーザー任意	JP1/Advanced Shell のカバレージ環境情報です。
デバッグ情報ファイル	.asd	ジョブ定義スクリプトファイルと同じディレクトリ※2	エディタ（開発環境）で使用するデバッグ情報です。
システム実行ログ※1	.log	環境ファイルの LOG_DIR パラメーターで指定したディレクトリ	バッチジョブの実行履歴を統括的に参照するためのシステム管理者向けのログ情報です。
トレースログ※1	.log	• adshexec コマンドの場合，環境ファイルの TRACE_DIR パラメーターで指定したディレクトリ • 上記以外の場合，プログラムで規定したディレクトリ	JP1/Advanced Shell の内部トレースログです。
一時ファイル	.tmp	• #-adsh_file_temp コマンドで指定した一時ファイルの場合，環境ファイルの TEMP_FILE_DIR	JP1/Advanced Shell が使用する一時ファイルです。

ファイル名（アイコン）	拡張子	ファイルの位置	ファイルの内容
一時ファイル	.tmp	パラメーターで指定したディレクトリ • 上記以外の場合，プログラムで規定したディレクトリ	JP1/Advanced Shell が使用する一時ファイルです。
カバレージ表示一時ファイル	.txt	システムで規定する一時ファイルディレクトリ	カバレージ情報の表示で使用する一時ファイルです。ファイル名の形式を次に示します。 adshexec_view_**ジョブ定義スクリプトファイル名**_**年月日**_**時分秒**.txt
起動ログ【UNIX 限定】	.log	/opt/jp1as/system	ユーザー応答機能管理デーモンの起動・停止時に採取されるログ情報です。
pid ファイル【UNIX 限定】	.pid	/opt/jp1as/system	ユーザー応答機能管理デーモンと adshmdctl コマンドで使用するファイルです。
アプリケーション実行エージェント機能ログ※1【Windows 実行環境限定】	.log	共有ドキュメントフォルダ¥ Hitachi ¥JP1AS¥ JP1ASE¥ appexec	アプリケーション実行エージェント機能の内部ログです。

注※1

これらのファイルは adshcollect コマンドで採取できます。採取方法については，「11.3.1　adshcollect コマンド（資料を採取する）」を参照してください。

注※2

次のような場合には，デバッグ情報ファイルを保存できないためエラーが表示されます。

・書き込み権限がないディレクトリのジョブ定義スクリプトファイルを編集している場合

・圧縮フォルダ内のジョブ定義スクリプトファイルを編集している場合

ファイルとパスの指定に関する注意事項

- ディレクトリ区切り文字は，Windows であれば「¥」※，UNIX であれば「/」を使用できます。それ以外の文字を使用した場合の動作は保証できません。

  UNIX のディレクトリ区切り文字に「¥」を使用した場合，ディレクトリ区切り文字とはみなされず，正しく動作しません。

  Windows のディレクトリ区切り文字に「/」を使用した場合，ディレクトリ区切り文字とみなされることがあります。ただし，使用方法によってはディレクトリ区切り文字とみなされず，正しく動作しないこともあります。

  注※

  ジョブ定義スクリプトに記述した「¥」はエスケープ文字と見なされるため，「¥¥」と記述するか，シングルクォーテーションで囲んでください。

- ファイル名に.（ドット）で始まる名称を使用しないでください。

- 指定できるパス名の最大長は，使用している OS の仕様に従います。
- ファイル名は最大 246 バイトです。【Windows 限定】
- ファイル名に予約デバイス名（CON，AUX，NUL など）は使用しないでください。【Windows 限定】
- ファイル名に NTFS のストリームは使用しないでください。【Windows 限定】
- ジャンクションは使用しないでください。【Windows 限定】
- ファイル名やパス名には UNC 形式の名称（例：**¥¥コンピュータ名¥共有名¥ファイル名**）を使用できますが，パス名の最後が「**共有名**」（または「**共有名¥**」）にならないように指定してください。また，シェル標準コマンドの cd コマンドには UNC 形式を指定できません。【Windows 限定】

  【使用できる UNC 形式】

  ¥¥server¥share¥dir

  ¥¥10.111.222.33¥share¥dir

  【使用できない UNC 形式】

  ¥¥server¥share

  ¥¥10.111.222.33¥share
- トレース，システム実行ログ，スプールおよび一時ファイルのフォルダパス名に UNC 形式の名称は使用しないでください。【Windows 限定】

(2)　2GB を超過するファイル（ラージファイル）の扱い

JP1/Advanced Shell では 2GB を超過するファイル（ラージファイル）を一部使用できます。JP1/Advanced Shell で使用できるファイルのうち，ラージファイルに対応するファイルおよびコマンドを次に示します。

- スプールジョブディレクトリ内に作成されるファイルのうち，ユーザーデータが出力されるファイル「STDOUT」，「STDERR」，「**ステップ番号_ステップ名**_STDOUT」，および「**ステップ番号_ステップ名**_STDERR」

  ただし，JP1/AJS3 から実行するジョブは，ファイル「STDERR」と「**ステップ番号_ステップ名**_STDERR」の内容を JP1/AJS3 のマネージャーホストへ転送するため，ファイルサイズが大きくなるとシステム全体の処理に影響を与えるおそれがあります。詳細については，マニュアル「JP1/Automatic Job Management System 3 設計ガイド（システム構築編）」を参照してください。

  なお，スプールジョブ作成抑止機能を使用している場合は，ジョブ定義スクリプトの実行時にスプールジョブは作成されません。スプールジョブ作成抑止機能については，「(a)　スプールジョブ作成抑止機能の使用有無を決定する」を参照してください。
- リダイレクト指定のうち，「>file」，「< file」，「>>file」，「>|file」，「<>file」，「n>file」および「n<file」で指定するファイル
- test コマンドまたは let コマンドで評価する条件式のうち，「-t fd」以外の条件式で指定するファイル
- スクリプト拡張コマンドのうち，#-adsh_file コマンド，#-adsh_file_temp コマンド，および#-adsh_spoolfile コマンドで割り当てるファイル

- シェル拡張コマンドのうち，adshmktemp コマンドで作成するファイル
- UNIX 互換コマンドで扱うファイル
  ただし，ファイルに対して次に示す操作をする場合は 2GB を超過できません。
  - 2GB を超過するバイト数での編集やバイト数表示をする場合
  - 2GB を超過する行数での編集や行数表示をする場合
  - 大きいサイズのファイルを指定するとメモリを大量に消費する UNIX コマンド（diff コマンド，sort コマンド）を実行する場合

ラージファイルの使用可否は，ファイルシステムの種類や OS の設定（例：ulimit の設定）によって異なるため，ラージファイルが使用できる環境であることを運用設計前に確認してください。

(3) ファイルシステムに関する注意事項

JP1/Advanced Shell で使用する場合に注意が必要なファイルシステムを次に示します。

- NFS
  サポートしません。
- HSFS
  HSFS を使用する場合，次の注意事項があります。
  - HSFS 上に JP1/Advanced Shell をインストールできません。
  - HSFS 上にシステム実行ログおよびトレースを作成できません。
  - ユーザー応答機能を使用する場合，スプールジョブディレクトリを HSFS 上に設定できません。
  - HSFS 07-00 より前のバージョンを使用する場合，UNIX 互換コマンドを使って HSFS 上のファイルおよびディレクトリを参照・更新するには，事前に HSFS のシステムオプションである CPFS_CACHE_POLICY に NOCACHE を指定する必要があります。
  - HSFS 07-00 以降のバージョンを使用する場合，UNIX 互換コマンドを使って HSFS 上のファイルおよびディレクトリを参照・更新するには，事前に HSFS のシステムオプションである CPFS_COMPAT_LINKCNT=0 を指定する必要があります。デフォルトは指定されている状態です。

2.2.4 JP1/Advanced Shell を使用するときのエンコーディング

JP1/Advanced Shell が使用するジョブ定義スクリプトファイルおよび環境ファイルのエンコーディングは，JP1/Advanced Shell が動作する環境の LANG 環境変数の値と一致させてください。一致していない場合の動作は保証できません。また，異なる OS で共通のジョブ定義スクリプトファイルを実行する場合は，各 OS で使用できるエンコーディングに統一してください。

UNIX の場合，JP1/Advanced Shell が出力するメッセージの言語およびエンコードは，LANG 環境変数の値によって決まります。

JP1/Advanced Shell を使用するときに設定する LANG 環境変数値，ジョブ定義スクリプトファイルおよび環境ファイルのエンコーディングを次の表に示します。

表 2-13　LANG 環境変数に対応するエンコーディング

OS	LANG 環境変数の値	ジョブ定義スクリプトファイルと環境ファイルのエンコーディング
Windows	－	Shift-JIS
Linux	ja_JP.UTF-8 C	UTF-8 アスキー文字列
AIX	Ja_JP ja_JP JA_JP JA_JP.UTF-8 C	Shift-JIS EUC UTF-8 UTF-8 アスキー文字列
HP-UX	ja_JP.SJIS ja_JP.eucJP ja_JP.utf8 C	Shift-JIS EUC UTF-8 アスキー文字列
Solaris	ja_JP.PCK ja ja_JP.UTF-8 C	Shift-JIS EUC UTF-8 アスキー文字列

（凡例）

－：該当しません。

2.2.5　ローカルタイムの設定

JP1/Advanced Shell では，ローカルタイム情報を環境変数から参照して情報を入出力するため，それらを環境変数で事前に設定しておく必要があります。

JP1/Advanced Shell が提供するコマンドは，OS のタイムゾーンの設定（Windows の場合），または環境変数 TZ（UNIX の場合）に従って情報を出力します。環境変数 TZ は，次に示すどれかの方法で指定してください。なお，カスタムジョブ定義プログラムの場合は，環境変数を定義できません。

- JP1/AJS のジョブ定義，環境変数の定義で指定
- システムプロファイル（/etc/profile）に指定
- ユーザープロファイル（$HOME/.profile）に指定

2.2.6　標準入力についての注意事項

JP1/Advanced Shell が提供するコマンドを使用して端末装置から標準入力へ入力する場合，入力できる最大長は，OS，端末装置，シェル，およびプログラム言語の言語仕様などによって異なります。

2.2.7　ハードリンク，シンボリックリンクを使用する

JP1/Advanced Shell ではハードリンク，シンボリックリンクを作成して使用できます。JP1/Advanced Shell で使用できるリンクファイルは次のとおりです。

- ファイルへのハードリンク
- ファイルへのシンボリックリンク
- ディレクトリへのシンボリックリンク

以降，UNIX 版と Windows 版に分けて記載します。

(1)　UNIX 版

UNIX 版の JP1/Advanced Shell では UNIX 互換コマンドの ln コマンド，および OS が提供するコマンドを使用することでハードリンク，シンボリックリンクを作成して，ジョブ定義スクリプト内で使用できます。

(2)　Windows 版

(a)　ハードリンク，シンボリックリンクに対応するファイル，フォルダ

Windows 版の JP1/Advanced Shell では UNIX 互換コマンドの ln コマンド，および OS が提供するコマンドを使用することでハードリンク，シンボリックリンクを作成できます。ハードリンク，シンボリックリンクに対応するファイル，フォルダは次のとおりです。

- UNIX 互換コマンド，シェル標準コマンドで扱うファイル，およびフォルダ
- リダイレクト記号に指定するファイル
- 条件式のファイル属性で指定するファイル，およびフォルダ

また，次に示すファイル，フォルダに対してはハードリンク，シンボリックリンクを作成しないでください。

- JP1/AS が提供する各種コマンド
- JP1/AS インストール時に作成するファイル，およびフォルダ
- ジョブ定義スクリプトファイル
- 外部スクリプトファイル
- 初期設定スクリプトファイル

- 子孫ジョブで実行するスクリプトファイル
- スプールディレクトリ配下のファイル，およびフォルダ
- 一時ファイルディレクトリ配下のファイル，およびフォルダ
- トレースディレクトリ配下のファイル，およびフォルダ
- ログディレクトリ配下のファイル，およびフォルダ
- 環境ファイル

(b) シンボリックリンクを使用する前の確認事項

シンボリックリンクを使用する際は事前に次の点を確認してください。

- シンボリックリンクを作成，削除，移動，または複製する場合は，シンボリックリンク作成権限を持つユーザーで実行してください。シンボリックリンク作成権限を持たないユーザーはシンボリックリンクに対して操作することはできません。また，ユーザーアカウント制御（UAC）が有効な環境では，シンボリックリンク作成権限を持つユーザーでもシンボリックリンクを作成できません。UAC が有効な環境でシンボリックリンクを作成する場合は，シンボリックリンク作成権限を持つユーザーに管理者特権を付与してください。
- シンボリックリンクを使用する前に，実行するマシンがシンボリックリンクの使用を許可されているか確認してください。シンボリックリンクの使用が許可されているかを確認する方法は Windows のマニュアルを参照してください。

(3) 注意事項

ハードリンク，シンボリックリンクを使用する際の注意事項を次に示します。

(a) 全プラットフォーム共通の注意事項

- リンクファイルの名称は JP1/Advanced Shell が提供するコマンドと同じ名称にしないでください。
- シンボリックリンクファイルのネストの回数が OS の上限を超えた場合の動作は，各 OS の仕様に従います。
- 1 つのファイルに対して複数のハードリンクを作成できます。しかし，OS，またはファイルシステムによって，1 つのファイルに対して作成できるハードリンク数の上限が設けられています。上限を超えた場合はハードリンクの作成に失敗するため，注意してください。

(b) UNIX 版の注意事項

カバレージ情報ファイル（asc ファイル）の出力先にハードリンク，シンボリックリンクは指定できません。指定した場合，次のように動作します。

- asc ファイルの出力先にシンボリックリンクを指定すると，シンボリックリンクは削除され，同名の通常ファイルが作成されます。

- 1 つの asc ファイルに複数のハードリンクを作成した状態で asc ファイルを更新しても，asc ファイルのカバレージ情報は更新されません。

(c) Windows 版の注意事項

- NTFS 以外のファイルシステムにリンクファイルを作成できません。
- ファイルシステムを保護することを目的とした製品をインストールしている環境では，シンボリックリンクを使用できない場合があります。シンボリックリンクを使用する際はすでにインストールされている製品がシンボリックリンクに対応していることを確認してください。
- シンボリックリンクを使用し実行ファイルを起動する場合は，シンボリックリンクと実行ファイルの両方の拡張子が「.bat」，「.com」，「.cmd」，「.exe」のどれかである必要があります。ただし，次の書式は実行ファイルを起動する実行プログラムの仕様に従うため，実行権限の判定も実行プログラムの仕様に従います。
  - awk コマンドのsystem 関数
  - awk コマンドの書式 *コマンド名* | getline [*変数名*]
  - awk コマンドの書式 print [*式*[, ...]] | *コマンド名*
  - find コマンドの-exec プライマリおよび-ok プライマリ
  - xargs コマンド
- シンボリックリンクを使用しファイル，フォルダにアクセスする場合は，リンク先のアクセス権限に従います。シンボリックリンク自身にアクセスする場合は，シンボリックリンク自身のアクセス権限に従います。
- ディレクトリへのシンボリックリンクのリンク先が通常ファイル，またはファイルへのシンボリックリンクのリンク先がディレクトリとなっている場合，リンク先にアクセスできずエラー終了します。
- リンクファイル対応に伴い，リンクファイルに関する情報も出力します。そのため，ls コマンドなどUNIX 互換コマンドの出力形式が一部変更になります。リンクファイルを使用しないで，かつ出力形式を11-00 より前に戻したい場合は環境変数ADSH_LINK_SUPPORT にL0 を指定してください。

(4) 環境変数 ADSH_LINK_SUPPORT（JP1/Advanced Shell のリンク対応レベルを定義する）

JP1/Advanced Shell では 11-00 からハードリンク，シンボリックリンクを使用できます。しかし，ハードリンク，シンボリックリンクに対応したことに伴い，実行結果以外にも出力形式，出力内容も一部変更となります。そこで，11-00 より前から JP1/Advanced Shell を使用しており，リンクファイルを使用しないユーザーは環境変数 ADSH_LINK_SUPPORT の指定をご検討ください。

(a) 環境変数に設定できる値

環境変数 ADSH_LINK_SUPPORT に指定できる値を次に示します。

表 2-14　環境変数 ADSH_LINK_SUPPORT に設定できる値

値	意味
L0	ハードリンク，シンボリックリンクは使用できません。 L0 を指定した場合の主な変更点は次のとおりです。 • find コマンド，ls コマンド，stat コマンドでハードリンク数が出力されません。 • UNIX 互換コマンド，シェル標準コマンドのシンボリックリンクに関するオプションを指定しても無視されます。 • adshscripttool コマンドの-L オプションを使用できません。 • 環境設定パラメーター UNSUPPORT_TEST を指定しないでファイル属性を判定する演算子-h，-L, -ef を使用するとジョブはエラー終了します。
L1	次の機能でハードリンク，シンボリックリンクを使用することができます。 • UNIX 互換コマンド • シェル標準コマンド • ファイル属性の条件式 • ファイルへのリダイレクト
上記以外	ジョブ，およびコマンドを実行しないでエラー終了します。

環境変数 ADSH_LINK_SUPPORT に L0 と L1 を指定した場合の出力形式の違いを，ls コマンドを例に示します。なお，環境変数 OSCMD_DIR には ls コマンドがインストールされているディレクトリへのパスが格納されていると仮定しています。

- L0 の場合

```
C:¥TEMP>%OSCMD_DIR%¥ls -l
total 439744
-rw------- Administrators  102000 Jul 06 16:26 HARDLINK.txt
-rw------- Administrators  102000 Jul 06 16:20 SYMLINK.txt
drwx------ Administrators         Jul 06 16:58 TestLog
-rw------- Administrators  102000 Jul 06 16:20 test_data.txt
-rw------- Administrators  102000 Jul 06 16:26 test_result.txt
-rwx------ Administrators   31744 Jun 12 16:23 uap.exe
```

- L1 の場合

```
C:¥TEMP>%OSCMD_DIR%¥ls -l
total 337744
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58 TestLog
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

(b)　注意事項

- 環境変数 ADSH_LINK_SUPPORT が未定義の場合は，L1 が指定されたと解釈し動作します。

- 環境変数 ADSH_LINK_SUPPORT には L0，または L1 を指定してください。L0，L1 以外の値を指定した場合，ジョブ，コマンドは終了コード 255 でエラー終了します。
- 環境変数 ADSH_LINK_SUPPORT は Windows 版で有効です。UNIX 版では無効です。
- ジョブ定義スクリプトファイルや環境ファイルで設定した場合，その設定は，ジョブ定義スクリプトから起動する子孫ジョブ，ルートジョブ，別プロセス実行，一部の UNIX 互換コマンドにだけ有効となります。

2.3　インストール／アンインストール【Windows 限定】

ここでは，Windows 環境の JP1/Advanced Shell のインストール方法およびアンインストール方法について説明します。必要になる前提プログラムおよび関連プログラムについては，事前に該当するマニュアルを参照してインストールしておいてください。

製品および関連プログラムのインストールの流れを次に示します。

1. 運用管理サーバで必要な製品のインストールとセットアップを実施する。
   - JP1/AJS でジョブを運用する場合
     JP1/AJS - Manager
   - ユーザー応答機能を使用する場合
     JP1/IM - Manager
2. 運用管理端末で必要な製品のインストールとセットアップを実施する。
   - JP1/AJS でジョブを運用する場合
     JP1/AJS - View
   - ユーザー応答機能を使用する場合
     JP1/IM - View
3. 運用管理端末に JP1/Advanced Shell - Custom Job をインストールする。
   カスタムジョブ定義プログラムのインストールについては，「2.3.3　JP1/Advanced Shell - Custom Job をインストールする」を参照してください。
4. バッチ業務サーバで必要な製品のインストールとセットアップを実施する。
   - JP1/AJS でジョブを運用する場合
     JP1/AJS - Agent
   - ユーザー応答機能を使用する場合
     JP1/Base
5. バッチ業務サーバで JP1/Advanced Shell のインストールと環境情報の設定などをする。
   Windows 環境のインストールについては，「2.3.1　JP1/Advanced Shell をインストールする【Windows 限定】」を参照してください。
   ユーザー応答機能を使用する場合のセットアップ手順については，「2.8.2　JP1/Advanced Shell をインストールしたあとのユーザー応答機能の設定【Windows 限定】」を参照してください。

2.3.1　JP1/Advanced Shell をインストールする【Windows 限定】

JP1/Advanced Shell をインストールする場合，管理者権限を持つユーザーで実行してください。インストール方法として，JP1/NETM/DM を使ったリモートインストールと CD-ROM 媒体を使ったインストールがあります。JP1/Advanced Shell - Developer のインストールも同様に実行できます。

(1) JP1/NETM/DM を使ったリモートインストール

JP1/Advanced Shell は，JP1/NETM/DM を使ったリモートインストール（ソフトウェア配布）に対応しています。

JP1/NETM/DM を使った実際のリモートインストール方法については，マニュアル「JP1/NETM/DM 導入・設計ガイド(Windows(R)用)」およびマニュアル「JP1/NETM/DM 運用ガイド 1(Windows(R)用)」を参照してください。

(2) CD-ROM 媒体を使ったインストール

JP1/Advanced Shell をインストールする方法には次の 3 種類があります。

- 新規で導入する場合，新規インストールをします。
- バージョンアップする場合，上書きインストールをします。
- 同一バージョンで再インストールする場合，修復インストールをします。

各手順について次に説明します。

(a) 新規インストールの場合

JP1/Advanced Shell を新規インストールする手順を次に示します。実行環境をインストールする場合は，通常，サーバにインストールします。開発環境をインストールする場合は，通常，クライアント PC にインストールします。ただし，1 つの PC に実行環境と開発環境の両方をインストールすることもできます。

1. JP1/Advanced Shell をインストールする Windows マシンに管理者権限を持つユーザーでログオンする。
2. すべてのプログラムを終了する。
3. JP1/Advanced Shell の CD-ROM 媒体を CD-ROM ドライブに入れる。
4. 起動したインストーラの指示に従って必要な情報を入力し，インストールする。
   インストール時に定義する情報を次に示します。
   - インストールする製品（JP1/Advanced Shell または JP1/Advanced Shell - Developer）の選択
   - ユーザー情報
   - インストール先フォルダ
5. ［完了］ダイアログが表示されたら，［完了］をクリックする。
   インストールが完了します。

(b) バージョンアップによる上書きインストールの場合

新規インストールと同じ手順で上書きインストールしてください。

JP1/Advanced Shell のバージョンアップによる上書きインストールを行う場合，アンインストールしなくてもアップグレードできます。

アプリケーション実行エージェントプログラムが起動している場合，アンインストール前に起動しているユーザーでログオンし停止します。【実行環境限定】

(c)　同一バージョンによる修復インストールの場合

JP1/Advanced Shell がインストール済み状態で，製品の不具合が発生した場合に製品を修復するときは，次の手順を実施します。

1. アプリケーション実行エージェントプログラムが起動している場合，アンインストール前に起動しているユーザーでログオンし停止します。【実行環境限定】
2. JP1/Advanced Shell をインストールする Windows マシンに管理者権限を持つユーザーでログオンする。
3. すべてのプログラムを終了する。
4. JP1/Advanced Shell の CD-ROM 媒体を CD-ROM ドライブに入れる。
5. 起動したインストーラの指示に従って必要な情報を入力する。
6. ［完了］ダイアログが表示されたら，［完了］をクリックする。
   修復インストールが完了します。

2.3.2　JP1/Advanced Shell をアンインストールする【Windows 限定】

(1)　手動によるアンインストール

JP1/Advanced Shell をアンインストールする手順を次に示します。また，JP1/Advanced Shell - Developer のアンインストールも同様に実行できます。

1. アプリケーション実行エージェントプログラムが起動している場合，起動しているユーザーでログオンし停止します。さらにスタートアップに登録している場合，登録を解除します。【実行環境限定】
2. JP1/Advanced Shell をインストールしてある Windows マシンに管理者権限を持つユーザーでログオンする。
3. すべてのプログラムを終了する。また，ユーザー応答機能を使用している場合は，JP1/Advanced Shell のサービスを停止してから登録を解除する。
4. JP1/Advanced Shell の CD-ROM 媒体を CD-ROM ドライブに入れる。
5. 起動したインストーラの指示に従って必要な情報を入力し，［プログラムの保守］を選択する。
6. ［プログラムの保守］で［削除］を選択する。
7. ［完了］ダイアログが表示されたら，［完了］をクリックする。
   アンインストールが完了します。

8. スプール，トレース，デバッグ情報ファイルなどで不要なファイルが残っている場合は，削除する。

ユーザー応答機能を使用していた場合は，アンインストールが完了したあと，JP1/Base に対して設定したユーザー応答機能用アダプタコマンド設定ファイルを削除します。ユーザー応答機能用アダプタコマンド設定ファイルの格納先フォルダについては，「(2)　アダプタコマンドの設定（実行環境の場合）」または「(3)　アダプタコマンドの設定（開発環境の場合）」を参照してください。

(2)　JP1/NETM/DM を使ったアンインストール

JP1/NETM/DM を使ったアンインストール方法については，マニュアル「JP1/NETM/DM 導入・設計ガイド(Windows(R)用)」およびマニュアル「JP1/NETM/DM 運用ガイド 1(Windows(R)用)」を参照してください。

2.3.3　JP1/Advanced Shell - Custom Job をインストールする

JP1/AJS - View をインストール済みの運用管理端末で，カスタムジョブ定義プログラムをインストールする手順について説明します。カスタムジョブ定義プログラムは，JP1/Advanced Shell のインストール先から運用管理端末へ転送してインストールします。

カスタムジョブ定義プログラムは，Windows 環境にインストールできますが，Windows と UNIX のジョブの定義を作成できます。新規インストール，上書きインストール，および修復インストールについて説明します。

(1)　新規インストール

JP1/Advanced Shell - Custom Job を新規インストールする手順を次に示します。

1. JP1/Advanced Shell - Custom Job をインストールする運用管理端末に，管理者権限を持つユーザーでログインする。
2. JP1/Advanced Shell - Custom Job のインストーラを取得する。
   インストーラの格納先を次に示します。
   - Windows 版 JP1/Advanced Shell を使用する場合
     インストール先フォルダ¥JP1ASE¥util¥setup.exe
   - UNIX 版 JP1/Advanced Shell を使用する場合
     /opt/jp1as/util/setup.exe
3. JP1/Advanced Shell - Custom Job のインストーラ（setup.exe）を運用管理端末へ転送する。
4. コマンドプロンプトを起動してインストーラを転送したフォルダに位置づけ，次のコマンドを実行する。

```
setup.exe
```

5. インストーラの指示に従って必要な情報を指定し，インストールする。
   インストールするカスタムジョブ定義プログラムの言語を選択してから，次の情報を指定します。

- ユーザー情報：ユーザー名などを指定します。
- インストール先フォルダ：JP1/Advanced Shell - Custom Job をインストールするフォルダを指定します。

6. ［完了］ダイアログが表示されたら，［完了］をクリックする。
   インストールが完了します。
7. アプリケーション実行エージェント機能を使用する場合で，JP1/AJS3 - View 11-00 より古い製品がインストールされている場合にカスタムジョブ定義プログラムをインストールしたときは，カスタムジョブアイコンを次のフォルダにコピーする。
   JP1/AJS - Viewのインストール先フォルダ¥image¥custom
   コピーするカスタムジョブアイコンのフォルダとファイル名については，「2.1.1　インストール先フォルダ【Windows 限定】」を参照してください。

(2)　バージョンアップによる上書きインストール

JP1/Advanced Shell - Custom Job のバージョンアップによる上書きインストールを行う場合，アンインストールしなくてもアップグレードできます。

新規インストールと同じ手順で上書きインストールしてください。

以前のバージョンでカスタムジョブアイコンを JP1/AJS - View のインストール先フォルダにコピーしている場合は，バージョンアップ時に再度コピーする必要はありません。

(3)　同一バージョンによる修復インストール

JP1/Advanced Shell - Custom Job がインストール済み状態で製品の不具合が発生した場合に製品を修復するときは，次の手順を実施します。

1. JP1/Advanced Shell - Custom Job をインストールする Windows マシンに，管理者権限を持つユーザーでログインする。
2. JP1/Advanced Shell - Custom Job のインストーラを取得する。
   インストーラの格納先を次に示します。
   - Windows の場合
     インストール先フォルダ¥JP1ASE¥util¥setup.exe
   - UNIX の場合
     /opt/jp1as/util/setup.exe
3. JP1/Advanced Shell - Custom Job のインストーラ（setup.exe）を運用管理端末へ転送する。
4. コマンドプロンプトを起動してインストーラを転送したフォルダに位置づけ，次のコマンドを実行する。

```
setup.exe
```

5. ［プログラムの保守］から［修復］を選択する。

6. [完了] ダイアログが表示されたら，[完了] をクリックする。
   修復インストールが完了します。

2.3.4 JP1/Advanced Shell - Custom Job をアンインストールする

JP1/Advanced Shell - Custom Job をアンインストールする手順を次に示します。

1. JP1/Advanced Shell - Custom Job をインストールしてある Windows マシンに管理者権限を持つユーザーでログオンする。
2. すべてのプログラムを終了する。
3. JP1/AJS - View 11-00 より古い製品がインストールされている場合で，アプリケーション実行エージェント機能のアイコンが存在した場合，次のフォルダにコピーしたカスタムジョブアイコンを削除する。
   JP1/AJS - Viewのインストール先フォルダ¥image¥custom
4. コントロールパネルの [プログラムのアンインストール] から製品を選択する。
   ユーザーアカウント制御（UAC）が有効な環境でアンインストールを実行した場合，[ユーザーアカウント制御] ウィンドウが表示されます。このウィンドウに対して，[はい] を選択してください。
5. [完了] ダイアログが表示されたら，[完了] をクリックする。
   アンインストールが完了します。
6. 不要なトレースファイルが残っている場合は，削除する。

2.4　インストール／アンインストール【UNIX 限定】

ここでは，UNIX 環境の JP1/Advanced Shell のインストール方法およびアンインストール方法について説明します。UNIX 環境では，実行環境だけをバッチ業務サーバにインストールできます。前提プログラムおよび関連プログラムについては，事前に該当するマニュアルを参照してインストールしておいてください。

製品および関連プログラムのインストールの流れを次に示します。

1. 運用管理サーバで必要な製品のインストールとセットアップを実施する。
   - JP1/AJS でジョブを運用する場合
     JP1/AJS - Manager
   - ユーザー応答機能を使用する場合
     JP1/IM - Manager
2. Windows 環境の運用管理端末で必要な製品のインストールとセットアップを実施する。
   - JP1/AJS でジョブを運用する場合
     JP1/AJS - View
   - ユーザー応答機能を使用する場合
     JP1/IM - View
3. 運用管理端末に JP1/Advanced Shell - Custom Job をインストールする。
   カスタムジョブ定義プログラムのインストールについては，「2.3.3　JP1/Advanced Shell - Custom Job をインストールする」を参照してください。
4. バッチ業務サーバで必要な製品のインストールとセットアップを実施する。
   - JP1/AJS でジョブを運用する場合
     JP1/AJS - Agent
   - ユーザー応答機能を使用する場合
     JP1/Base
5. バッチ業務サーバに JP1/Advanced Shell のインストールと環境情報の設定などをする。
   UNIX 環境へのインストールについては，「2.4.1　JP1/Advanced Shell をインストールする【UNIX 限定】」を参照してください。
   ユーザー応答機能を使用する場合のセットアップ手順については，「2.8.3　JP1/Advanced Shell をインストールしたあとのユーザー応答機能の設定【UNIX 限定】」を参照してください。

2.4.1　JP1/Advanced Shell をインストールする【UNIX 限定】

JP1/Advanced Shell をインストールする場合，管理者権限を持つユーザーで実行してください。インストール方法として，JP1/NETM/DM を使ったリモートインストールと CD-ROM 媒体を使ったインストールがあります。

(1)　JP1/NETM/DM を使ったリモートインストール

JP1/Advanced Shell は，JP1/NETM/DM を使ったリモートインストール（ソフトウェア配布）に対応しています。

JP1/NETM/DM を使った実際のリモートインストール方法については，マニュアル「JP1/NETM/DM Manager」およびマニュアル「JP1/NETM/DM SubManager(UNIX(R)用)」を参照してください。

(2)　CD-ROM 媒体を使ったインストール

CD-ROM 媒体を使った UNIX へのインストール手順を次に示します。

なお，CD-ROM のディレクトリ名やファイル名は，ハードウェア環境などによって記述した内容と見え方が異なります。ls コマンドで確認し，表示されたファイル名をそのまま入力してください。

1. ユーザー権限を設定する。

   JP1/Advanced Shell をインストールするサーバに，スーパーユーザーでログインします。または，su コマンドでユーザー権限をスーパーユーザーに変更します。

2. すべてのプログラムを終了する。

   既存の JP1 シリーズのプログラム，および JP1/Advanced Shell のプログラムが動作している場合，必ず停止させます。

3. JP1/Advanced Shell の媒体をセットする。

4. 次のコマンドを実行して，CD-ROM 装置をマウントする。

   ```
   /bin/mount -r -o mode=0544 /dev/cdrom /cdrom
   ```

   /cdrom は CD-ROM デバイススペシャルファイルのマウントポイントです。マウントポイントディレクトリがない場合は，作成してください。なお，デバイススペシャルファイル名およびマウントポイントは，使用する環境によって異なる場合があります。

5. 次のコマンドを実行して，Hitachi PP Installer を起動する。

   Linux の場合

   `/cdrom/LINUX/setup /cdrom`※

   AIX の場合

   `/cdrom/AIX/setup /cdrom`※

   HP-UX の場合

   `/cdrom/IPFHPUX/setup /cdrom`※

Solaris の場合

```
/cdrom/SOLARIS/setup /cdrom※
```

注※　ここでは，マウントポイントに/cdrom を仮定します。

Hitachi PP Installer が起動し，初期画面が表示されます。

Hitachi PP Installer の初期画面の例を次に示します。

```
  Hitachi PP Installer  05-24


    L) List Installed Software.
    I) Install Software.
    D) Delete  Software.
    Q) Quit.


    Select Procedure ===>


    +-----------------------------------------------------------------------+
      CAUTION!
      YOU SHALL INSTALL AND USE THE SOTFWARE PRODUCT LISTED IN THE
      ”List Installed Software.” UNDER THE TERMS AND CONDITION OF
      THE SOFTWARE LICENSE AGREEMENT ATTACHED TO SUCH SOFTWARE PRODUCT.
    +-----------------------------------------------------------------------+



  All Rights Reserved. Copyright (C) 1994, 2015, Hitachi, Ltd.
```

6. Hitachi PP Installer の初期画面で「I」を入力する。

   インストールできるプログラムの一覧が表示されます。

7. JP1/Advanced Shell を選択して「I」を入力する。

   JP1/Advanced Shell がインストールされます。なお，プログラムを選択するには，カーソルを移動させてスペースバーで選択します。

   PP インストール画面の例を次に示します。

```
       PP-No.              VR        PP-NAME
<@>001 P-8112-B1BL         1100      Advanced Shell
:
:
F) Forward B) Backward J) Down K) Up Space) Select/Unselect I) Install Q) Quit
```

選択した PP の左側に，<@>が表示されます。続いて［I］を入力すると，最下行に次に示すメッセージが表示されます。

```
Install PP? (y: install, n: cancel) ==>
```

ここで，［y］または［Y］を選択するとインストールが開始されます。［n］または［N］を選択すると，インストールが中止され，PP インストール画面に戻ります。

8. インストールが正常終了したら，「Q」を入力する。
   Hitachi PP Installer の初期画面に戻ります。

なお，インストール時にインストーラのログとして次のファイルが作成されます。

```
/opt/jp1as/instlog/ADSH_INST_LOG
/opt/jp1as/instlog/ADSH_INST_USERLOG
```

インストーラのログファイルが作成されていない場合，次に示す問題が考えられます。

- インストーラのログファイルが通常のファイルでない
- インストーラのログファイルを作成するディレクトリに書き込み権限がない
- インストーラのログファイルのパス名を構成する各ファイルパスに，同一のファイルが存在する
  同一のファイルが存在する状態を次に示します。
  - "/opt"がディレクトリでない
  - "/opt/jp1as"がディレクトリでない
  - "/opt/jp1as/instlog"がディレクトリでない

インストールが完了すると，デフォルトの環境が設定されています。デフォルトから設定を変更する場合は，「2.6　JP1/Advanced Shell の環境情報を設定する」以降の該当する部分を参照してください。

2.4.2　JP1/Advanced Shell をアンインストールする【UNIX 限定】

(1)　手動によるアンインストール

JP1/Advanced Shell をアンインストールする手順を次に示します。Hitachi PP Installer の指示に従って JP1/Advanced Shell をアンインストールします。

JP1/Advanced Shell をアンインストールする場合は，JP1/Advanced Shell が提供するプログラムをすべて終了してください。また，ユーザー応答機能を使用している場合は，ユーザー応答機能管理デーモンを停止してください。アンインストールでは，インストーラのログファイルおよび新規に作成したファイルは削除されません。したがって，完全に環境を削除するには，ユーザーが自分で削除する必要があります。

1. 次のコマンドを実行して，Hitachi PP Installer を起動する。

```
/etc/hitachi_setup
```

Hitachi PP Installer が起動し，初期画面が表示されます。

Hitachi PP Installer の初期画面の例を次に示します。

```
  Hitachi PP Installer  05-24


    L) List Installed Software.
    I) Install Software.
    D) Delete  Software.
    Q) Quit.


    Select Procedure ===>


    +-------------------------------------------------------------------+
      CAUTION!
      YOU SHALL INSTALL AND USE THE SOFTWARE PRODUCT LISTED IN THE
      ”List Installed Software.” UNDER THE TERMS AND CONDITION OF
      THE SOFTWARE LICENSE AGREEMENT ATTACHED TO SUCH SOFTWARE PRODUCT.
    +-------------------------------------------------------------------+



  All Rights Reserved. Copyright (C) 1994, 2015, Hitachi, Ltd.
```

2. Hitachi PP Installer の初期画面で「D」を入力する。

   アンインストールできるソフトウェアの一覧が表示されます。

3. JP1/Advanced Shell を選択して「D」を入力する。

   JP1/Advanced Shell がアンインストールされます。なお，プログラムを選択するには，カーソルを移動させてスペースバーで選択します。

   アンインストール画面の例を次に示します。

```
       PP-No.              VR        PP-NAME
<@>001 P-8112-B1BL         1100      Advanced Shell
:
:
F) Forward B) Backward J) Down K) Up Space) Select/Unselect I) Install Q) Quit
```

選択したPPの左側に，<@>が表示されます。続いて［D］を入力すると，最下行に次に示すメッセージが表示されます。

```
Install PP? (y: install, n: cancel) ==>
```

ここで，［y］または［Y］を選択するとアンインストールが開始されます。［n］または［N］を選択すると，アンインストールが中止され，アンインストール画面に戻ります。

4. アンインストールが正常終了したら，「Q」を入力する。
   Hitachi PP Installerの初期画面に戻ります。
5. 実行ログ，トレースなどで不要なファイルが残っている場合は，削除する。

なお，アンインストール時にインストーラのログとして次のファイルが作成されます。

```
       PP-No.              VR        PP-NAME
<@>001 P-8112-B1BL         1100      Advanced Shell
   002 P-812C-6LBL         1100      JP1/Base
   003 P-9S12-A111         0806/A    uCosminexus Batch Job Execution Server
   004 P-CC8112-4KBL       1100      JP1/AJS3 - Manager
```

インストーラのログファイルが作成されていない場合，次に示す問題が考えられます。

- インストーラのログファイルが通常のファイルでない
- インストーラのログファイルを作成するディレクトリに書き込み権限がない
- インストーラのログファイルのパス名を構成する各ファイルパスに，同一のファイルが存在する
  同一のファイルが存在する状態を次に示します。
  - "/opt"がディレクトリでない
  - "/opt/jp1as"がディレクトリでない
  - "/opt/jp1as/instlog"がディレクトリでない

注意

ユーザー応答機能管理デーモンが起動されている場合は，アンインストールが中断されます。この時，次のメッセージが/opt/jp1as/instlog/ADSH_INST_LOGに出力されます。

```
F) Forward B) Backward J) Down K) Up Space) Select/Unselect D) Delete Q) Quit
Delete PP? (y: delete, n: cancel) ==>
```

上記メッセージが出力されている場合は，adshmdctlコマンドを実行してユーザー応答機能管理デーモンを停止したあと，アンインストールを再実行してください。

また，ユーザー応答機能管理デーモンが正常に停止されなかった場合も，アンインストールが中断されます。この場合，一度ユーザー応答機能管理デーモンを起動して停止してから，再度アンインストールを実行してください。

(2) JP1/NETM/DM を使ったアンインストール

JP1/NETM/DM を使ったアンインストール方法については，マニュアル「JP1/NETM/DM Manager」およびマニュアル「JP1/NETM/DM SubManager(UNIX(R)用)」を参照してください。

(3) ユーザー応答機能を使用していた場合

ユーザー応答機能を使用していた場合は，アンインストールが完了したあと，次の作業を実施してください。

- JP1/Base に対して設定したユーザー応答機能用アダプタコマンド設定ファイルを削除します。ユーザー応答機能用アダプタコマンド設定ファイルの格納先ディレクトリについては，「(2) JP1/Base の設定」を参照してください。
- ユーザー応答機能管理デーモンの自動起動および自動停止を設定している場合は，自動起動および自動停止の設定を解除します。

  【AIX の場合】

  1. 次のコマンドを実行して，ユーザー応答機能管理デーモンの自動起動の設定を解除する。

     ```
     rmitab adshmd
     ```

  2. 論理ホスト用のユーザー応答機能管理デーモンの自動起動を設定している場合は，論理ホスト用ユーザー応答機能管理デーモンのレコードを指定して rmitab コマンドを実行する。
  3. システム終了時の自動停止機能を解除するには，/etc/rc.shutdown の次に示す記述を削除する。

     ```
     test -x /opt/jp1as/sbin/adshmdctl && /opt/jp1as/sbin/adshmdctl stop
     ```

  4. 論理ホスト用のユーザー応答機能管理デーモンの自動起動を設定している場合は，論理ホスト用ユーザー応答機能管理デーモンの自動停止の記述を/etc/rc.shutdown から削除する。

  【RHEL 6，Oracle Linux 6，CentOS 6，HP-UX，Solaris の場合】

  1. /opt/jp1as/sample ディレクトリからコピーしたスクリプトファイル jp1_as_md をコピー先のディレクトリから削除する。
  2. 論理ホスト用の自動起動および自動停止スクリプトファイルを作成している場合は，スクリプトファイル作成先のディレクトリから削除する。
  3. jp1_as_md スクリプトファイルへのリンクとして作成したシンボリックリンクを削除する。
  4. 論理ホスト用の自動起動および自動停止スクリプトファイルのシンボリックリンクを作成している場合は削除する。

  【RHEL 7，SUSE Linux 12，Oracle Linux 7，CentOS 7 の場合】

  1. 次のコマンドを実行して，ユーザー応答機能管理デーモンの自動起動・自動停止を無効にする。

     ```
     systemctl disable jp1_as_md.service
     ```

  2. /opt/jp1as/sample ディレクトリからコピーした Unit ファイル jp1_as_md.service をコピー先の/usr/lib/systemd/system ディレクトリから削除する。
  3. 論理ホスト用のユーザー応答機能管理デーモンの自動起動・自動停止を設定している場合は，次の作業を実施する。

・1 の手順で指定する jp1_as_md.service を，作成した論理ホスト用の Unit ファイル名に置き換えて systemctl コマンドを実行する。

・/etc/systemd/system ディレクトリに作成した論理ホスト用の Unit ファイルを削除する。

自動起動・停止するためのスクリプトファイルのコピー先ディレクトリについては，「(1)　ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。論理ホスト用の自動起動および自動停止スクリプトファイルの作成先ディレクトリおよびシンボリックリンクの作成先ディレクトリについては，「(2)　非クラスタ環境の論理ホスト用ユーザー応答機能管理デーモンの自動起動と自動停止【UNIX 限定】」を参照してください。

2.4.3　Hitachi PP Installer でバージョン情報を確認する【UNIX 限定】

UNIX 版の JP1/Advanced Shell は，Hitachi PP Installer を使ってインストールするため，Hitachi PP Installer から JP1/Advanced Shell のバージョン情報を表示できます。

表示する手順を次に示します。

1. 次のコマンドを実行して，Hitachi PP Installer を起動する。

```
/etc/hitachi_setup
```

2. 初期画面で「L」を入力する。

   インストール済みの日立製品の一覧が表示されます。バージョン情報を確認してください。

2.5　環境変数を設定する

JP1/Advanced Shell で使用できる環境変数を次の表に示します。

重要

JP1/Advanced Shell では，名称がADSH から始まるシェル変数・環境変数を設定・参照しています。このため，このマニュアルに記載されている使用目的以外では，名称がADSH から始まるシェル変数・環境変数は使用しないでください。

表 2-15　JP1/Advanced Shell で使用できる環境変数

環境変数名	設定する内容	値が自動的に設定される場合の設定時期	値の設定可否
ADSH_AJS_APPEXEC【Windows 実行環境限定】	カスタムジョブ用 GUI アプリケーション実行プログラム識別用。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_APPNAME【Windows 実行環境限定】	カスタムジョブ用実行アプリケーションのパス名。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_APPARG【Windows 実行環境限定】	カスタムジョブ用実行アプリケーションの引数。	ジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_WORKF【Windows 実行環境限定】	カスタムジョブ用ワークフォルダ。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_SHOWN【Windows 実行環境限定】	カスタムジョブ用表示名。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_AFEXECMV【Windows 実行環境限定】	カスタムジョブ用実行アプリケーション実行後の動作。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_MESOUT【Windows 実行環境限定】	カスタムジョブ用メッセージの出力。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_ENVF	カスタムジョブ用ジョブ環境ファイル名。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_GCHE	カスタムジョブ用事前チェックオプション。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_AJS_LHOST	カスタムジョブ用論理ホスト名。	カスタムジョブで起動した場合のジョブ開始時	可※1

環境変数名	設定する内容	値が自動的に設定される場合の設定時期	値の設定可否
ADSH_AJS_SCRF	カスタムジョブ用ジョブ定義スクリプトファイル名。	カスタムジョブで起動した場合のジョブ開始時	可※1
ADSH_ENV	ジョブ環境ファイルのファイル名。	カスタムジョブで起動した場合のジョブ開始時	可※2
ADSH_CMD_ARGORDER※3	コマンドラインに指定するコマンド引数の指定順序規則。設定できる値はseq だけである。 cp, cut, date, diff, expand, gunzip, gzip, ln, ls, mv, stat コマンドで有効となる。また，getopt コマンドのユーザー定義のオプションの解析でも有効となる。	（自動的には設定されない）	可
ADSH_CMDDATE_FORMAT	JP1/Advanced shell 固有の共通書式処理を行うdate コマンドの書式指定コード。	（自動的には設定されない）	可
ADSH_CMDEXPR_LENGTH	文字列長。expr コマンドの length 演算子を使用する場合に設定する。 文字列のバイト数を取得するときはb，文字列の文字数を取得するときにはc を設定する。 この環境変数の設定がないか，上記以外の値を設定した場合は，length は演算子として扱われない。	（自動的には設定されない）	可
ADSH_CMDLN_FOLLOW	ln コマンドの引数ターゲットにディレクトリに対するシンボリックリンクを指定した場合に，リンクをたどるかたどらないかを設定する。 リンクをたどらない場合はNO を設定する。リンクをたどる場合はNO 以外を設定する。	（自動的には設定されない）	可
ADSH_CMDLN_OPT_I_F	ln コマンドで-i オプションと-f オプションを同時に指定した場合にどちらを有効にするか設定する。 最後に指定した方を有効にしたいときはLAST を設定する。	（自動的には設定されない）	可
ADSH_CMDTAR_ROOTPATH	tar コマンドのルートディレクトリの動作の変更。値にabsolute を指定することでルートディレクトリは取り除かないで格納，抽出，および表示するようにする。	（自動的には設定されない）	可
ADSH_JOB_NAME	ジョブ名。	ジョブ開始時	不可
ADSH_JOBID	ジョブ識別子（先頭に 0 を付加した 6 桁固定の 10 進数）。	ジョブ開始時	不可
ADSH_JOBRC_FATAL	構文エラーなど，ジョブを続行できない致命的なエラーが発生した場合のジョブの終了コード。 設定方法については「(2)　環境変数 ADSH_JOBRC_FATAL（ジョブ続行不可エラー発生時の終了コードを設定する）」を参照してください。	（自動的には設定されない）	可※2
ADSH_LANG 【UNIX 限定】※4※6	JP1/Advanced Shell が出力するメッセージの言語とエンコード。 特定のジョブのadshexec コマンドが出力するメッセージを，一時的に切替えたい場合に設定する。	（自動的には設定されない）	可

環境変数名	設定する内容	値が自動的に設定される場合の設定時期	値の設定可否
ADSH_LANG 【UNIX 限定】※4※6	設定できる値については「2.2.4　JP1/Advanced Shell を使用するときのエンコーディング」を参照してください。 ジョブ定義スクリプトファイルや環境ファイルで設定した場合，その指定は，ジョブ定義スクリプトから起動する子孫ジョブ，ルートジョブ，およびadshexec コマンド以外のシェル運用コマンドにだけ有効となる。	（自動的には設定されない）	可
ADSH_LANG_JP1EVENT 【UNIX 限定】※5	ユーザー応答機能で使用する JP1 イベントで出力するメッセージの言語。出力先の JP1/IM の設定に合わせて，JP1/Advanced Shell が出力するメッセージの言語とは異なる言語の JP1 イベントのメッセージを出力したい場合に設定する。 設定できる値については「2.2.4　JP1/Advanced Shell を使用するときのエンコーディング」を参照してください。 ジョブ定義スクリプトファイルで設定した場合，その指定は，ジョブ定義スクリプトから起動する子孫ジョブ，ルートジョブ，およびadshexec コマンド以外のシェル運用コマンドにだけ有効となる。環境ファイルで設定した場合，ジョブ定義スクリプトから出力する JP1 イベントのメッセージの言語がその値に従った言語となる。	（自動的には設定されない）	可
ADSH_LINK_SUPPORT 【Windows 限定】	JP1/Advanced Shell のリンク対応レベルを指定します。 設定方法については「(4)　環境変数 ADSH_LINK_SUPPORT（JP1/Advanced Shell のリンク対応レベルを定義する）」を参照してください。	（自動的には設定されない）	可※7
ADSH_STEP_NAME	ジョブステップ名。 ジョブステップ外のコマンド実行時や，ジョブステップ名省略時は，環境変数を定義しない。	ジョブステップ開始時	不可
AJS_BJEX_STOP	JP1/AJS からの強制終了で使用するインターフェース。PC ジョブや UNIX ジョブで JP1/Advanced Shell のバッチジョブを定義する場合は，この環境変数を定義する必要がある。OS の設定で定義するのではなく，PC ジョブや UNIX ジョブの定義で定義すること。	カスタムジョブで起動した場合のジョブ開始時	可（TERM だけ設定可能）
BLOCKSIZE	1 ブロック当たりのバイト数。ls コマンドとstat コマンドで使用する。 デフォルトは512。	（自動的には設定されない）	可
COLUMNS	コマンド実行結果出力時の 1 行あたりの出力幅。ls コマンドの-C オプションとsed コマンドのl コマンド（編集コマンド）で使用する。	（自動的には設定されない）	可
GETOPT_COMPATIBLE	パラメーターの解析方法の指定。getopt コマンドで使用する。	（自動的には設定されない）	可

環境変数名	設定する内容	値が自動的に設定される場合の設定時期	値の設定可否
GETOPT_COMPATIBLE	設定する値に決まりはない。設定されている場合，解析される引数はすべて，getopt コマンドが形式 1 で指定されていると解釈して処理される。	（自動的には設定されない）	可
GZIP※8	gzip コマンド，およびgunzip コマンドのオプションを設定する。次のコマンドで使用する。 • gzip コマンド • gunzip コマンド • tar コマンド（-z オプション指定時） gzip コマンド，およびgunzip コマンドの場合，引数に指定したオプションを優先する。オプションを複数指定する場合は，スペースまたはタブ文字で区切って設定する。	（自動的には設定されない）	可
POSIXLY_CORRECT※3	コマンドラインに指定するコマンド引数の指定順序規則。 設定する値に決まりはない。値が設定されている場合，環境変数ADSH_CMD_ARGORDER にseq が設定されているのと同じ意味になる。	（自動的には設定されない）	可
TMPDIR 【UNIX 限定】	一時ファイル出力先ディレクトリ。diff コマンドとsort コマンドで使用する。	（自動的には設定されない）	可

注※1

これらの環境変数は，JP1/AJS のajsdefine コマンドのユニット定義，または JP1/AJS - Definition Assistant のジョブ定義だけで使用できます。JP1/Advanced Shell のジョブ定義スクリプトや，ユーザープロファイルやシステムプロファイルなどのユーザー環境でこれらの環境変数は設定しないでください。

注※2

ジョブ定義スクリプトファイルや環境ファイルで設定した場合，その指定は，ジョブ定義スクリプトから起動する子孫ジョブやルートジョブにだけ有効となります。

注※3

環境変数POSIXLY_CORRECT は，環境変数ADSH_CMD_ARGORDER で有効なコマンドのほかに Linux の OS 標準のコマンドにも適用されますが，コマンドによってはコマンド引数の指定順序規則以外の機能にも作用する場合があります。そのため，UNIX 互換コマンドだけにコマンド引数の指定順序規則を設定したい場合は，環境変数ADSH_CMD_ARGORDER を使用してください。

注※4

環境変数ADSH_LANG の指定は，環境変数LANG より優先して使用します。環境変数ADSH_LANG を指定しない場合は，環境変数LANG と同じ言語とエンコードでメッセージを出力します。環境変数ADSH_LANG および環境変数LANG を指定しない場合は，C が指定されたように動作します。

注※5

環境変数ADSH_LANG_JP1EVENT の指定値は，環境変数ADSH_LANG および環境変数LANG より優先されます。

環境変数ADSH_LANG_JP1EVENTを指定しない場合は，環境変数ADSH_LANGと同じ言語でメッセージを出力します。環境変数ADSH_LANG_JP1EVENTも環境変数ADSH_LANGも指定しない場合は，環境変数LANGと同じ言語でメッセージを出力します。

環境変数ADSH_LANGがC以外，または環境変数ADSH_LANGが未指定で環境変数LANGがC以外の場合，JP1イベントで出力するメッセージの言語は日本語になります。このときに出力するJP1イベントのメッセージを英語で出力したい場合は，環境変数ADSH_LANG_JP1EVENTにCを指定します。

注※6

adshmdctlコマンドに使用した場合，syslogに出力するメッセージも環境変数ADSH_LANGに従った言語とエンコードで出力されます。システムによってはsyslogにその言語とエンコードの文字コードが出力できない場合があります。その場合にはadshmdctlコマンドに使用しないでください。

注※7

ジョブ定義スクリプトファイルや環境ファイルで設定した場合，その指定は，ジョブ定義スクリプトから起動する子孫ジョブやルートジョブ，一部のUNIX互換コマンドで有効となります。

注※8

環境変数GZIPにオプション値のあるオプションを設定する場合，次のことに注意してください。

- オプション値をクォーテーション（'または"）で囲んだ場合，クォーテーション（'または"）をオプション値として扱うことがあります。
- オプション値にスペースやタブ文字を含めた場合，スペースやタブ文字をオプションの区切り文字と見なし，複数の引数の指定として扱うことがあります。

この表の環境変数に加えて，OSが標準的に設定する環境変数や，環境ファイルにexportパラメーターで設定した環境変数も，ジョブ定義スクリプトから参照できます。exportパラメーターについては，「7.3.18 exportパラメーター（環境変数を定義する）」を参照してください。

2.6　JP1/Advanced Shell の環境情報を設定する

インストールが終了したあと，環境情報を設定するために次の作業を実施してください。環境情報を設定することで，該当する環境情報に基づいたバッチジョブを実行できるようになります。

- 必要に応じて，JP1/Advanced Shell の環境ファイルの情報（環境情報）および環境変数などを設定します。

  環境ファイルや環境変数の設定については，「2.6.1　環境ファイルを設定する」～「2.6.15　ジョブを続行できないエラーが発生したときの終了コードを定義する」を参照してください。

  また，必要に応じて次の個所も参照してください。

  - 「2.6.16　ユーザー応答機能を設定する」
  - 「2.6.17　JP1 環境を確認する【UNIX 限定】」
  - 「2.6.18　シェルを設定する【UNIX 限定】」
- JP1/Advanced Shell で必要なディレクトリとファイルをデフォルト内容から変更したい場合は，変更後のディレクトリやファイルを作成します。

  詳細については，「2.6.19　JP1/Advanced Shell で必要なディレクトリを作成する」を参照してください。
- 保守情報を採取するための定義ファイルを設定します。

  保守情報の採取の詳細については，「11.3　資料の採取方法」を参照してください。

重要

JP1/Advanced Shell では，名称が「ADSH」から始まるシェル変数・環境変数を設定・参照しています。このため，このマニュアルに記載されている使用目的以外では，名称が「ADSH（Windows の場合は小文字での表記も含む）」から始まるシェル変数・環境変数は使用しないでください。

2.6.1　環境ファイルを設定する

環境ファイルには，システム環境ファイルとジョブ環境ファイルがあります。設定できるパラメーターは同じです。各ファイルの説明を次の表に示します。

表 2-16　環境ファイルの種類

環境ファイルの種類	説明
システム環境ファイル	システム管理者が設定する，システム共通の環境ファイルです。サービスまたはデーモンはシステム環境ファイルの設定内容を使用します。固定のディレクトリに格納することで自動的に使用される環境ファイルです。
ジョブ環境ファイル	開発者がジョブごとに設定する環境ファイルです。次の方法で指定します。

環境ファイルの種類	説明
ジョブ環境ファイル	• 環境変数 ADSH_ENV で指定する環境ファイル • JP1/Advanced Shell エディタの［実行環境の設定］ダイアログボックスで指定するジョブ環境ファイル • JP1/Advanced Shell カスタムジョブ定義時に指定するジョブ環境ファイル

JP1/Advanced Shell のサービスまたはデーモンはシステム環境ファイルの設定内容を使用します。システム環境ファイルに設定した内容は，JP1/Advanced Shell のサービスまたはデーモンの起動時から有効になります。

ジョブコントローラは，システム環境ファイルおよびジョブ環境ファイルの設定内容を使用します。

環境ファイルに指定できるパラメーターの詳細については，「7.　環境ファイルで設定するパラメーター」を参照してください。

システム環境ファイルのパラメーターに指定する各種ディレクトリは，存在するディレクトリを指定する必要があります。そのため，デフォルトのディレクトリを変更する場合は，変更後のディレクトリをあらかじめ作成してください。

また，UNIX 環境でシステム環境ファイルを変更したあとは，必ず adshmdctl コマンドに conftest オプションを指定して実行し，エラーのないことを確認してください。

それぞれの環境ファイルの設定方法を次に示します。

(1)　システム環境ファイルの設定

システム環境ファイルはシステム管理者が作成および設定します。作成したシステム環境ファイルは，次の表に示すファイルパスに格納することで有効になります。

表 2-17　システム環境ファイルのファイル名

環境	システム環境ファイルのファイル名
Windows（開発環境）	共通アプリケーションフォルダ¥HITACHI¥JP1AS¥JP1ASD¥conf¥adshrc.ase
Windows（実行環境）	共通アプリケーションフォルダ¥HITACHI¥JP1AS¥JP1ASE¥conf¥adshrc.ase
UNIX	/opt/jp1as/conf/adshrc.ase

(2)　ジョブ環境ファイルの設定

ジョブ環境ファイルを使用してバッチジョブを実行する場合は，環境変数 ADSH_ENV にファイルパスを設定します。次に示す手順でジョブ環境ファイルを作成して設定してください。

なお，JP1/Advanced Shell エディタでは，［実行環境の設定］ダイアログボックスでジョブ環境ファイルのパスを指定できます。また，JP1/Advanced Shell カスタムジョブ定義時には，使用するジョブ環境ファイルのパスを指定できます。

1. 次のディレクトリにある環境ファイルのサンプルデータ sample.ase を，任意のディレクトリ・ファイル※にコピーする。
   - Windows の実行環境の場合

     インストール先フォルダ¥JP1ASE¥sample¥sample.ase
   - Windows の開発環境の場合

     インストール先フォルダ¥JP1ASD¥sample¥sample.ase
   - UNIX の実行環境の場合

     /opt/jp1as/sample/sample.ase
2. コピーしたジョブ環境ファイルに必要なパラメーターを定義する。

   ジョブ環境ファイルに必要なパラメーターを「7.　環境ファイルで設定するパラメーター」に記載していますので参照して定義してください。また，ジョブ環境ファイルのエンコーディングとジョブ定義スクリプトを実行する環境の LANG 環境変数の値は一致させてください。ジョブ環境ファイルのエンコーディングおよび LANG 環境変数については，「2.2.4　JP1/Advanced Shell を使用するときのエンコーディング」を参照してください。
3. 作成したジョブ環境ファイルをバッチジョブ実行時に使用するよう，作成したジョブ環境ファイルのパスを環境変数 ADSH_ENV に設定する。

   環境変数 ADSH_ENV は，次に示すどれかの方法で指定します。
   - 【Windows 限定】OS の設定で環境変数を指定する
   - 【UNIX 限定】システムプロファイル/etc/profile を指定する
   - 【UNIX 限定】ユーザープロファイル（$HOME/.profile）を指定する

注※

ジョブ環境ファイルの任意のディレクトリ名およびファイル名に&，(,)，[,]，{,}，^，=，;，!，'，+，,，`，～，#，%の記号を含めないでください。これらの記号を含む場合，正常に動作しません。

2.6.2　パス名を変換する

ジョブ定義スクリプトに記載したパス名を Windows と UNIX の両方で使用できるよう，パスの変換内容をパラメーターで定義します。

JP1/Advanced Shell では，プラットフォームに対応してジョブ定義スクリプトに次のようにパスを記載できます。

表 2-18　Windows 環境と UNIX 環境で使用できるパスの規則

項目	Windows 環境	UNIX 環境
ディレクトリ区切り文字	¥¥ ※1	/
パス区切り文字	;	:

項目	Windows 環境	UNIX 環境
パス名の大文字と小文字	区別する ※2	区別する
絶対パス	パス名の先頭文字は，「ドライブレター:¥¥」 ※1，※3	パス名の先頭文字は，「/」

注※1

Windows 環境では，¥はエスケープ文字と見なされるため，¥¥と記載します。またはパス名全体をシングルクォーテーションで囲みます。

注※2

パス名の変換では，Windows 環境でも大文字と小文字を区別します。

注※3

UNC 形式の名称も指定できます。ただし，ジョブ定義スクリプトでパス名の変換を定義する場合，変換結果のパスの末尾が共有名（後ろに「¥」を指定した場合も含む）にならないよう定義してください。パス名の末尾が共有名の場合，動作は保証されません。

上記の規則によって，パスを変換するにはパラメーターに次の定義が必要です。

- Windows 環境でジョブ定義スクリプトを実行させたい場合
  UNIX 環境の区切り文字を読めるようにするため，「/」と「:」を定義します。
- UNIX 環境でジョブ定義スクリプトを実行させたい場合
  Windows 環境の区切り文字を読めるようにするため，「¥¥」と「;」を定義します。

パス名を変換するためのパラメーターを次に示します。

- PATH_CONV_ENABLE パラメーター
  パスを変換する機能を有効にします。変換前のパス区切り文字およびディレクトリ区切り文字を指定します。
  Windows 環境では「/」と「:」を定義します。
  UNIX 環境では「¥¥」と「;」を定義します。
- PATH_CONV_RULE パラメーター【Windows 限定】
  パス名の変換対象として次のどちらかを定義します。
  - ダブルクォーテーションで囲まれた範囲を変換対象とする（パス変換ルール 1）
  - シングルクォーテーションで囲まれた範囲を除く全体を変換対象とする（パス変換ルール 2）

  パラメーターの指定を省略した場合や UNIX の場合は，パス変換ルール 1 が適用され，ダブルクォーテーションで囲まれた範囲だけが変換されます。
- PATH_CONV パラメーター
  パス名の変換前・変換後の文字列を定義します。ジョブ定義スクリプトを実行するときにパラメーターで定義した規則に従い置換します。パス名が変換されるのは，PATH_CONV_RULE パラメーターで定義された範囲だけです。

また，PATH_CONV パラメーターで定義した変換文字列に合致していれば，パス区切り文字およびディレクトリ区切り文字も変換されます。

(1) ファイルパスの変換例（パス変換ルール 1 の場合）

環境ファイルの情報に従って，実行前のジョブ定義スクリプトが実行後にどのように変換されるかを次に示します。

- 環境ファイルの情報

  Windows の場合の環境ファイルの例を次に示します。

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_RULE  1
#-adsh_conf PATH_CONV /home/hitachi/bin ”C:¥¥Program Files”  ←1.
#-adsh_conf PATH_CONV /tmp ”C:¥¥temp”                        ←2.
```

- 実行前のジョブ定義スクリプト

```
#-adsh_path_var DIR,DIR2
”/home/hitachi/bin/myprog1” ”/tmp/file”              ←1., 2.

DIR=”/home/hitachi/bin”                              ←1.
”$DIR/myprog1” ”/tmp/file”                           ←2.
DIR2=$DIR
”$DIR2/myprog2” ”/tmp/file”                          ←2.
```

- 実行後のジョブ定義スクリプト

  パスを変換すると次のようになります。

```
”C:¥¥Program Files¥¥myprog1” ”C:¥¥temp¥¥file”        ←1., 2.

DIR=”C:¥¥Program Files”                              ←1.
”$DIR¥¥myprog1” ”C:¥¥temp¥¥file”                     ←2.
DIR2=$DIR
”$DIR2¥¥myprog2” ”C:¥¥temp¥¥file”                    ←2.
```

1. PATH_CONV パラメーターの定義に従い，パスが「/home/hitachi/bin」から「C:¥¥Program Files」へ変換されています。

   また，PATH_CONV_ENABLE パラメーターの定義に従い，ディレクトリ区切り文字が「/」から「¥¥」へ変換されています。

2. PATH_CONV パラメーターの定義に従い，パスが「/tmp」から「C:¥¥temp」へ変換されています。

   また，PATH_CONV_ENABLE パラメーターの定義に従い，ディレクトリ区切り文字が「/」から「¥¥」へ変換されています。

(2) ファイルパスの変換例（パス変換ルール 2 の場合）

環境ファイルの情報に従って，実行前のジョブ定義スクリプトが実行後にどのように変換されるかを次に示します。

- 環境ファイルの情報

  Windows の場合の環境ファイルの例を次に示します。

```
#-adsh_conf  PATH_CONV_ENABLE  / :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV /home/user01  d:¥¥home¥¥user01
#-adsh_conf  PATH_CONV BB/AA BB¥¥AA
```

- 実行前のジョブ定義スクリプト

```
#-adsh_job  JOB001
#-adsh_path_var DIR01
echo -E ”/home/user01/file”
cat /home/user01/file
DIR01=/home/user01
cat $DIR01/file02                          ←1.
PATH=/home/user01/prog:$PATH               ←2.
uap01
DIR02=/home/user01
PATH=”$DIR02:/home/user01/prog:$PATH”
AA=10
BB=200
let ANS=BB/AA                              ←3.
echo $ANS
cat BB/AA
```

- 実行後のジョブ定義スクリプト

  パスを変換すると次のようになります。

```
#-adsh_job  JOB001
#-adsh_path_var DIR01
echo -E ”d:¥¥home¥¥user01¥¥file”
cat ”d:¥¥home¥¥user01”¥¥file
DIR01=”d:¥¥home¥¥user01”
cat ”$DIR01”¥¥file02                       ←1.
PATH=”d:¥¥home¥¥user01”¥¥prog”;””$PATH”     ←2.
uap01
DIR02=”d:¥¥home¥¥user01”
PATH=”$DIR02;d:¥¥home¥¥user01¥¥prog;$PATH”
AA=10
BB=200
let ANS=”BB¥¥AA”                           ←3.
echo $ANS
cat ”BB¥¥AA”
```

1. 変換前のジョブ定義スクリプトにパス名を扱うシェル変数 DIR01 を定義しているため，変換結果ではシェル変数 DIR01 が「"（ダブルクォーテーション）」で囲まれ，その後ろにディレクトリ区切り文字が付加されます。
2. 文字列の先頭部分が変換規則と一致するため「"（ダブルクォーテーション）」で囲まれます。また，パス区切り文字が「";"」に変換されます。さらに，文字列がパスとして変換されたので，変数名 PATH もダブルクォーテーションで囲まれます。

3. 演算式がパス変換規則に一致したため変換されています。変換されないようにするには，ジョブ定義スクリプトを次のどちらかの方法で修正する必要があります。
   - 「let ANS='BB/AA'」のようにシングルクォーテーションで囲む。
   - 「let ANS=$BB/$AA」のように参照する変数名の先頭に$を付加する。

(3) 注意事項

- この定義を使用してパス名を Windows 用に変換すると，パス名中のディレクトリ区切り文字が「¥」となります。そのため，パス名を無条件に echo コマンドで表示しているジョブ定義スクリプトでは，「¥」およびそれに続く文字がエスケープ文字に置き換わります。

  エスケープ文字に置き換えない場合は，echo コマンドに-E オプションを指定して実行してください。詳細については，「9.3　シェル標準コマンド」の「echo コマンド（引数で指定した内容を標準出力に出力する）」を参照してください。
- メタキャラクタの「~」，「~+」および「~-」は，クォーテーションで囲んだり，クォーテーションで囲まれた文字列やエスケープ文字（¥）の直前に記述されたりすると，置換されません。メタキャラクタについては，「5.1.6　メタキャラクタ」を参照し，対応するシェル変数を使用してください。

2.6.3　ファイルの入出力時にファイルパスを変換する

ファイルを入出力する場合に定義した規則に従って，ジョブ定義スクリプトに指定したファイルパスを入出力の対象となるファイルパスに変換します。指定したファイルパスと入出力するファイルパスは，完全に一致させる必要があります。

(1) ファイル入出力時のファイルパス変換の実行条件

ファイル入出力時のファイルパス変換は，リダイレクト文字（<，>，<>，>>）によって，ファイルへの入出力が発生するタイミングで変換します。

なお，.（ドット）コマンドまたは#-adsh_script コマンドでジョブ定義スクリプトを実行する場合も，ジョブ定義スクリプトの読み込み（入力）が発生します。しかし，これらについてはファイル入出力時のファイルパス変換では変換されません。変換する場合は，コマンド実行時に引数を変換する COMMAND_CONV_ARG パラメーターで変換の規則を定義してください。

ファイル入出力時のファイルパス変換は，「2.6.2　パス名を変換する」で説明されているパス変換によって変換されたジョブ定義スクリプトの内容に対して実行されます。

ファイル入出力時のファイルパス変換は，異なるプラットフォーム間の変換（UNIX→Windows，Windows→UNIX）だけでなく，同じプラットフォーム間の変換（UNIX→UNIX，Windows→Windows）も実行できます。

(2) ファイル入出力時のファイルパスの変換例

ファイルを入出力する場合に定義した環境ファイルの情報（PATH_CONV_ACCESS パラメーター）に従って，ジョブ定義スクリプトがファイルの入出力時にどのように変換されるかを以降に説明します。

(a) 環境ファイルの情報

環境ファイルの例を次に示します。

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /tmp "D:¥¥tmp"
#-adsh_conf PATH_CONV_ACCESS /dev/null nul
```

(b) 実行前のジョブ定義スクリプト

実行前のジョブ定義スクリプトの例を次に示します。

```
while read LOG
do
  echo $LOG > /dev/null
done < "/tmp/input.txt"
```

(c) 実行時のジョブ定義スクリプト

実行時には次のように解釈されて実行します。

```
while read LOG
do
  echo $LOG > nul
done < "D:¥tmp¥input.txt"
```

(3) PATH_CONV パラメーターと PATH_CONV_ACCESS パラメーターとの組み合わせ例

Windows 版で，PATH_CONV パラメーターと PATH_CONV_ACCESS パラメーターを組み合わせて使用する例を示します。この 2 種類のパラメーターは，PATH_CONV パラメーターの方が優先的に処理されます。同じパラメーターの中では，先頭から順に処理されます。

(a) 環境ファイルの内容

環境ファイルの内容を，各行に番号を付けて示します。

```
1.   #-adsh_conf PATH_CONV_ENABLE / :
2.   #-adsh_conf PATH_CONV /tmp "C:¥¥temp"
3.   #-adsh_conf PATH_CONV_ACCESS /tmp/result.log "C:¥¥jp1as_tmp¥¥result3.log"
4.   #-adsh_conf PATH_CONV_ACCESS "C:¥temp¥result.log" "C:¥¥jp1as_tmp¥¥result4.log"
5.   #-adsh_conf PATH_CONV_RULE 1
```

(b) ジョブ定義スクリプトファイルの内容と変換方法

(a)の環境ファイルに対して次のジョブ定義スクリプトを実行した場合，それぞれが異なる規則で変換されます。

```
cat data.txt > ”/tmp/result.log”
```

この場合，PATH_CONV_RULE パラメーターは 1 が指定されていることから，"（ダブルクォーテーション）で囲まれた範囲が PATH_CONV パラメーターによって変換されます。

cat コマンドに指定した"/tmp/result.log"は"（ダブルクォーテーション）で囲まれているため，環境ファイルの内容の行番号 2 の規則に従って"C:¥¥temp¥¥result.log"に変換されます。その結果，行番号 3 の定義には合致しないで，行番号 4 が定義に合致し，最終的に"C:¥¥jp1as_tmp¥¥result4.log"に変換されます。

```
cat data2.txt > /tmp/result.log
```

この場合，cat コマンドに指定した/tmp/result.log が，"（ダブルクォーテーション）で囲まれていないため，行番号 2 の PATH_CONV パラメーターの変換対象になりません。その結果，行番号 3 の定義に合致し，"C:¥¥jp1as_tmp¥¥result3.log"に変換されます。

2.6.4 コマンド実行時に引数を変換する

コマンド（シェル標準コマンド，シェル拡張コマンド，スクリプト拡張コマンド，スクリプト予約語コマンド，関数，および外部コマンド，ユーザープログラム）の実行時に，変換の定義に従ってコマンドの引数を変換します。この変換は，次のプラットフォーム間で実行できます。

- 同じプラットフォーム間：UNIX→UNIX，Windows→Windows
- 異なるプラットフォーム間：UNIX→Windows，Windows→UNIX

ジョブ定義スクリプトの 1 行を定義された規則に従って解析し，指定された引数の文字列と実行するコマンドの引数の文字列が完全に一致した場合，置換後のコマンド引数の文字列に変換します。変換の定義規則は，COMMAND_CONV_ARG パラメーターで設定します。

(1) PATH_CONV パラメーターと COMMAND_CONV_ARG パラメーターとの組み合わせ例

Windows 版で，PATH_CONV パラメーターと COMMAND_CONV_ARG パラメーターを組み合わせて使用する例を示します。この 2 種類のパラメーターは，PATH_CONV パラメーターの方が優先的に処理されます。同じパラメーターの中では，先頭から順に処理されます。

(a) 環境ファイルの内容

環境ファイルの内容を，各行に番号を付けて示します。

```
1.  #-adsh_conf PATH_CONV_ENABLE / :
2.  #-adsh_conf PATH_CONV /tmp "C:¥¥temp"
3.  #-adsh_conf COMMAND_CONV_ARG /tmp/data.txt "C:¥¥jp1as_tmp¥¥data3.txt"
4.  #-adsh_conf COMMAND_CONV_ARG "C:¥temp¥data.txt" "C:¥¥jp1as_tmp¥¥data4.txt"
5.  #-adsh_conf PATH_CONV_RULE 1
```

(b)　ジョブ定義スクリプトファイルの内容と変換方法

(a)の環境ファイルに対して次のジョブ定義スクリプトを実行した場合，それぞれが異なる規則で変換されます。

```
cat "/tmp/data.txt" > ./result.log
```

この場合，PATH_CONV_RULE パラメーターは 1 が指定されていることから，"（ダブルクォーテーション）で囲まれた範囲が PATH_CONV パラメーターによって変換されます。

cat コマンドに指定した"/tmp/data.txt"は"（ダブルクォーテーション）で囲まれているため，環境ファイルの内容の行番号 2 で"/tmp/data.txt"が"C:¥¥temp¥¥data.txt"に変換されます。その結果，行番号 3 の定義に合致しないで，行番号 4 の定義に合致し，最終的に"C:¥¥jp1as_tmp¥¥data4.txt"に変換されます。

```
cat /tmp/data.txt > ./result.log
```

この場合，cat コマンドに指定した/tmp/result.log が，"（ダブルクォーテーション）で囲まれていないため，行番号 2 の PATH_CONV パラメーターの変換対象になりません。その結果，行番号 3 の定義に合致し， "C:¥¥jp1as_tmp¥¥data3.txt"に変換されます。

2.6.5　子孫ジョブとして起動するファイルを定義する

ジョブ定義スクリプト中にコマンド名としてほかのジョブ定義スクリプトを指定できます。これによって，adshexec コマンドに指定したジョブ定義スクリプトを JP1/Advanced Shell のジョブとして実行できます。次のような場合に有効です。

- ユーザーの既存資産のシェルスクリプトを UNIX 環境から Windows 環境へ移行する場合
- UNIX 環境で OS のシェルで動作していた既存のシェルスクリプトを，内容を書き換えないで JP1/Advanced Shell のジョブとして実行する場合

子孫プロセスとして実行されるジョブ定義スクリプトのうち，特定の環境設定パラメーターによって実行されたジョブを子孫ジョブと呼びます。ルートジョブと子孫ジョブの詳細については，「(1)　ルートジョブと子孫ジョブ」を参照してください。ジョブ定義スクリプトを子孫ジョブとして実行する方法については，「3.2.3　ジョブ定義スクリプトを子孫ジョブとして実行する」を参照してください。

ジョブ定義スクリプトファイルを子孫ジョブとして起動させる場合，動作させるファイルの条件を環境ファイルに設定しておきます。環境設定パラメーターの概要を次に示します。

- CHILDJOB_EXT パラメーター
  子孫ジョブとして実行するジョブ定義スクリプトファイルの拡張子を定義する
- CHILDJOB_PGM パラメーター
  子孫ジョブとして実行するよう読み替えるパスを定義する
- CHILDJOB_SHEBANG パラメーター
  子孫ジョブとして実行するジョブ定義スクリプトファイルの実行プログラムパスを定義する

また，CHILDJOB_SHEBANG パラメーターのデフォルト定義に合致するジョブ定義スクリプトファイルを作成すれば，子孫ジョブとして実行されます。

各パラメーターの詳細については，「7.　環境ファイルで設定するパラメーター」を参照してください。

重要

ルートジョブと子孫ジョブを同じ環境ファイルパラメーターで動作させたい場合は，ジョブの実行中に環境変数 ADSH_ENV の値や環境ファイルの内容を変更しないでください。

2.6.6　UNIX 互換コマンドを使用するための定義をする

(1)　既存のジョブ定義スクリプトで実行ファイル形式の UNIX 互換コマンドを使用するための定義

既存のジョブ定義スクリプトで実行ファイル形式の UNIX 互換コマンドを使用する場合は，PATH 環境変数に UNIX 互換コマンドがインストールされているディレクトリへのパスを設定することで，ジョブ定義スクリプトの修正が不要になります。このとき，環境ファイルの export パラメーターを使用して PATH 環境変数の値の先頭にパスを設定することで，すでに UNIX 互換コマンドと同名のコマンドが存在していても，JP1/Advanced Shell のジョブ定義スクリプトでは必ず UNIX 互換コマンドを動作させることができます。

export パラメーターについては，「7.　環境ファイルで設定するパラメーター」の「7.3.18　export パラメーター（環境変数を定義する）」を参照してください。ジョブ定義スクリプトを実行する各環境で，正しいパスが設定されていることを確認してからジョブ定義スクリプトを実行してください。

(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】

スクリプト形式の UNIX 互換コマンドは，JP1/Advanced Shell が提供するサンプルスクリプトファイルを使用します。

スクリプト形式の UNIX 互換コマンド（chmod コマンドと su コマンドなど）は，JP1/Advanced Shell が提供するサンプルスクリプトファイルを基に，次の手順で実行します。

1. 次の場所にあるサンプルスクリプトファイルのうち，使用するファイルを任意のフォルダにコピーする。
   - Windows の実行環境の場合

     インストール先フォルダ¥JP1ASE¥sample
   - Windows の開発環境の場合

     インストール先フォルダ¥JP1ASD¥sample

   サンプルスクリプトファイルの種類については，「8.5　UNIX 互換コマンド（スクリプト形式）【Windows 限定】」を参照してください。
2. コピーしたファイルの名称を各コマンド名にリネームする。

   例えば，サンプルスクリプトファイル「script_chmod1」の場合はファイル名を「chmod」へ，サンプルスクリプトファイル「script_su1」の場合はファイル名を「su」へリネームします。何もしないコマンドとしたい場合は，サンプルスクリプトファイル「script_0」をコピーしてリネームします。
3. サンプルスクリプトを絶対パスや相対パス指定でなく，ファイル名だけ指定して実行したい場合は，次のどちらかを実行する。
   - 環境変数 PATH で定義されたフォルダに，実行するサンプルスクリプトを格納する
   - 環境変数 PATH に，実行するサンプルスクリプトを格納したフォルダのパスを追加する
4. 必要に応じて，KNAX6831-I メッセージの出力抑止を定義する。

   サンプルスクリプトの実行後に KNAX6831-I メッセージが出力させたくない場合は，ジョブ環境ファイルに次の内容を記述してください。

   ```
   #-adsh_conf JOBLOG_SUPPRESS_MSG     KNAX6831-I
   ```

   システム内のすべてのジョブ定義スクリプトに対して KNAX6831-I メッセージの出力を抑止したい場合は，システム環境ファイルに記述してください。
5. ジョブ定義スクリプトを実行する。

   手順 4.で作成したジョブ環境ファイルを使用して，ジョブ定義スクリプトを実行してください。システム環境ファイルに指定した場合は，手順 4.で指定した内容は自動的に読み込まれます。

2.6.7　サポートしていない条件式を実行した場合の動作を定義する【Windows 限定】

test コマンドでサポートしていない条件式を実行した場合の動作を次のパラメーターで定義します。

- UNSUPPORT_TEST パラメーター

Windows 環境では，ファイル属性を評価する場合の次の条件式がサポートされていないため，エラーとなります。このため，上記のパラメーターを指定することで，メッセージを出力してエラーとしたり，正常にしたりできます。サポートされていない条件式を次に示します。

- -G file：ファイルの属するグループが呼び出し元のプロセスの実行グループと一致していることを確認します。
- -O file：ファイルの所有者がプロセスの有効ユーザー ID であるか確認します。

また，次の条件式は JP1/Advanced Shell ではサポートしていますが，11-00 より前のバージョンではサポートしていなかったため，指定することができます。

- -h file：ファイルがシンボリックリンクであるか判定します。
- -L file：ファイルがシンボリックリンクであるか判定します（-h の場合と同じ）。
- file1 -ef file2：file1 と file2 が存在し，file1 と file2 の実体が同じ（シンボリックリンク先が同じまたはハードリンク先が同じ）であるか判定します。

2.6.8 ジョブ実行結果とログの出力情報を定義する

ジョブの実行結果はスプールディレクトリに出力され，出力内容の一部はジョブ実行ログとして参照できます。また，トラブル発生時にログを採取し，トラブルの原因を調査できます。これらのログの出力先や出力内容を環境ファイルで定義します。

JP1/Advanced Shell を運用しているときに出力されるログ情報と格納先について，次の表に示します。

表 2-19 JP1/Advanced Shell 運用時に出力されるログ情報と格納先

ログ情報	出力される情報	格納先
ジョブ実行ログ	バッチジョブのログ	スプールルートディレクトリの配下
システム実行ログ	JP1/Advanced Shell の統括的な実行ログ	環境ファイルの LOG_DIR パラメーター※1 で指定したディレクトリ
トレースログ	JP1/Advanced Shell の内部トレースログ	環境ファイルの TRACE_DIR パラメーター※1 で指定したディレクトリ
【UNIX 限定】 起動ログ※2	ユーザー応答機能管理デーモンの起動・終了ログ	/opt/jp1as/system の配下

注※1

パラメーターの指定がない場合，デフォルト値が適用されます。

注※2

ユーザー応答機能管理デーモンの起動・停止時に採取されるログ情報です。

このログ情報は，/opt/jp1as/system ディレクトリ下に次のファイル名で作成されます。

・物理ホストのユーザー応答機能管理デーモンの場合：adshmd.log

・論理ホストのユーザー応答機能管理デーモンの場合：adshmd_**論理ホスト名**.log

スプールの出力情報と，それぞれのログの出力情報の定義方法を次に説明します。

(1) スプールの出力情報を定義する

スプールに関するパラメーターについて，定義する出力情報ごとに説明します。

(a) スプールジョブ作成抑止機能の使用有無を決定する

スプールジョブ作成抑止機能を使用すると，スプールジョブが作成されなくなるため，スプールディレクトリのディスク容量の単調増加を防げます。また，スプールディレクトリ下の不要なディレクトリやファイルを削除する必要がなくなります。

スプールジョブ作成抑止機能の使用は SPOOLJOB_CREATE パラメーターで設定します。詳細については，「7.3.41 SPOOLJOB_CREATE パラメーター（スプールジョブの作成要否を選択する）」を参照してください。

スプールジョブ作成抑止機能を使用している間は，常に次の設定で動作します。

- #-adsh_conf EVENT_COLLECT NO
  稼働実績情報取得機能を無効とする
- #-adsh_conf OUTPUT_MODE_CHILD MINIMUM
  子孫ジョブを最小出力モードで実行する
- #-adsh_conf OUTPUT_MODE_ROOT MINIMUM
  ルートジョブを最小出力モードで実行する
- #-adsh_conf SPOOLJOB_CHILDJOB DELETE
  子孫ジョブの終了時に子孫ジョブのスプールジョブを削除する
- adshexec -m MINIMUM
  -m オプションの指定に関係なく，指定したジョブを最小出力モードで実行する
- adshscripttool -exec -m MINIMUM
  -m オプションの指定に関係なく，指定した子孫ジョブを最小出力モードで実行する

なお，スプールジョブ作成抑止機能を使用すると，スプールジョブディレクトリを使う次の機能は使用できません。

- #-adsh_spoolfile コマンド
  使用すると KNAX6385-E メッセージを出力してスクリプトは終了します。
- adshfile コマンド
  使用すると KNAX1880-E メッセージを出力してコマンドは終了します。
- ジョブ実行ログの採取
  ジョブ実行ログは採取しません。
- 稼働実績採取機能
  稼働実績は採取しません。

スプールジョブ作成抑止機能を使用した場合でも，スプールディレクトリは必要です。

また，スプールジョブ作成抑止機能を使用した場合には，次の環境設定パラメーターの指定は無視します。

- EVENT_COLLECT
- JOBEXECLOG_PRINT
- JOBLOG_SUPPRESS_MSG
- OUTPUT_MODE_CHILD
- OUTPUT_MODE_ROOT
- OUTPUT_STDOUT
- PERMISSION_SPOOLJOB_DIR
- PERMISSION_SPOOLJOB_FILE
- SPOOLJOB_CHILDJOB

CUI デバッグ時は次の名称の DBG ファイルを生成します。このファイルはデバッグの終了時に自動的に削除されます。このファイルの削除に失敗すると，標準エラー出力とシステム実行ログにエラーメッセージが出力されます。

```
一時ファイルディレクトリ/ADSH_DBG_プロセスID_ジョブ識別子
```

- **一時ファイルディレクトリ**
  TEMP_FILE_DIR パラメーターで定義する一時ファイルディレクトリ
- **プロセスID**
  5 桁以上のプロセス ID
- **ジョブ識別子**
  6 桁のジョブ識別子

(b) スプールルートディレクトリのパス名を定義する

スプールルートディレクトリのパス名を定義するパラメーターを次に示します。

- SPOOL_DIR パラメーター：スプールルートディレクトリのパス名を定義します。

なお，ユーザー応答機能を使用する場合は，SPOOL_DIR パラメーターはシステム環境ファイルだけに定義してください。

(c) スプールジョブディレクトリまたはファイルのパーミッションを変更する【UNIX 限定】

ジョブが終了すると，実行結果はジョブごとに作成されるスプールジョブディレクトリへ出力されます。このときディレクトリまたはその下のファイルに設定されるパーミッションは，次のパラメーターで変更できます。

- PERMISSION_SPOOLJOB_DIR パラメーター
  スプールジョブディレクトリのパーミッションを変更したい場合に指定します。
  デフォルトは 700 です。
- PERMISSION_SPOOLJOB_FILE パラメーター
  スプールジョブディレクトリ下のファイルのパーミッションを変更したい場合に指定します。
  デフォルトは 600（.DBG ファイルは 666）です。

(d) スプールジョブの標準出力と標準エラー出力の出力内容を定義する

ジョブを実行すると，ジョブの出力結果のほかに，ジョブコントローラの情報メッセージ，警告メッセージ，およびジョブ実行ログが出力されます。また，標準出力と標準エラー出力はスプールジョブディレクトリ下のファイルへ出力されます。

ジョブの実行結果だけをほかのプログラムで利用するなどの理由で，標準出力と標準エラー出力のスプールジョブディレクトリ下のファイルへの出力を抑止したい場合には，次のパラメーターまたはコマンドのオプションで簡潔出力モードまたは最小出力モードを指定します。

- OUTPUT_MODE_ROOT パラメーター
  ルートジョブに対して，拡張出力モード，簡潔出力モードまたは最小出力モードを指定します。
- OUTPUT_MODE_CHILD パラメーター
  子孫ジョブに対して，拡張出力モード，簡潔出力モードまたは最小出力モードを指定します。
- adshexec コマンドの-m オプション
  ジョブに対して，拡張出力モード，簡潔出力モードまたは最小出力モードを指定します。
- adshscripttool コマンドの-m オプション
  子孫ジョブに対して，簡潔出力モードまたは最小出力モードを指定します。

これらのパラメーターとオプションを指定しなかった場合は拡張出力モードとなり，標準出力と標準エラー出力はスプールジョブディレクトリ下のファイルへ出力されます。

なお，簡潔出力モードと最小出力モードでは，ジョブコントローラの情報メッセージと警告メッセージは標準出力，標準エラー出力へ出力しません。また，ジョブ終了時にジョブ実行ログを標準エラー出力へ出力しません。さらに，最小出力モードでは，出力抑止したメッセージはスプールジョブディレクトリ下のジョブ実行ログにも出力しません。

簡潔出力モード，拡張出力モードおよび最小出力モードの出力内容の差異については，「3.4.4　ジョブ実行ログへの情報メッセージと警告メッセージの出力を抑止する」を参照してください。

(2)　ジョブ実行ログの出力情報を定義する

ジョブ実行ログに関して環境設定時に設定する内容について説明します。ジョブ実行ログの出力内容については「3.5　ジョブ実行ログ」を参照してください。

(a)　標準エラー出力に出力するジョブ実行ログの種類を定義する

ジョブが終了すると，ジョブ実行ログとして次に示す内容が標準エラー出力へ出力されます。出力されたジョブ実行ログは，adshexec コマンドの実行時の端末画面や，JP1/AJS - View の［実行結果詳細］ダイアログボックスなどに表示されます。

- JOBLOG ファイル（コマンドの実行結果やファイルの割り当て結果など，ジョブの動作状況を示すメッセージ）
- ジョブ定義スクリプト
- ジョブ実行中の標準エラー出力の内容

このうち，ジョブ実行中の標準エラー出力の内容だけを標準エラー出力へ出力させるには，次のパラメーターを指定します。これによって，ジョブ実行ログの出力内容を限定できます。

- JOBEXECLOG_PRINT パラメーター

なお，JP1/Advanced Shell - Developer で実行，またはジョブコントローラをデバッガモードで起動した場合ではこのパラメーターの指定に関係なく，実行に合わせて JOBLOG，標準出力，および標準エラー出力の情報をコンソールに出力します。ジョブの終了時にはジョブ実行ログの内容を標準エラー出力へ出力しません。

ジョブを簡潔出力モードまたは最小出力モードで実行した場合は，JOBEXECLOG_PRINT パラメーターの指定に関係なく，ジョブ終了時にジョブ実行ログを標準エラー出力に出力しません。

(b)　子孫ジョブとルートジョブのジョブ実行ログをマージする

次のパラメーターでは，子孫ジョブの終了時に子孫ジョブのスプールジョブを削除するか，ルートジョブのスプールジョブにマージするかを選択できます。

- SPOOLJOB_CHILDJOB パラメーター

ルートジョブのスプールジョブへマージした場合，子孫ジョブは終了順に出力されます。また，ルートジョブと子孫ジョブが判別できる形式で出力されます。

(3)　システム実行ログの出力情報を定義する

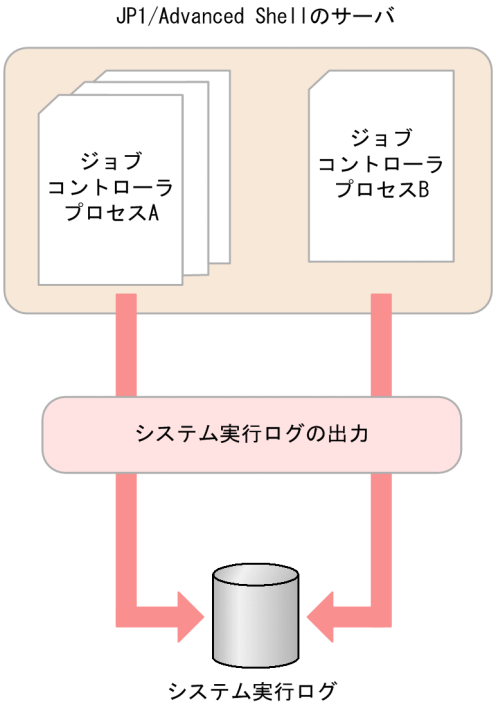
システム実行ログとは，バッチジョブの実行履歴を統括的に参照するためのシステム管理者向けのログ情報です。

このログ情報は，環境ファイルの LOG_DIR パラメーターで指定したディレクトリ下の AdshLog.log に出力されます。パラメーターに設定した条件（サイズなど）に従って，ファイル名のローテーション（AdshLog_1.log，AdshLog_2.log，…AdshLog_N.log）が実施されます。システム実行ログのファイルはローテーション時に新しく作成されるため，ファイルの所有者はローテーション時のユーザーになります。

(a)　システム実行ログの出力の流れ

システム実行ログには，各ジョブコントローラプロセスで実行しているバッチジョブの情報が出力されます。環境ファイルにシステム実行ログの出力先，サイズおよび面数を指定できます。システム実行ログの出力の流れを次に示します。

図 2-6　システム実行ログの出力の流れ



システム実行ログは次のように作成されます。

- システム実行ログに出力するメッセージを集め，CSV 形式で出力します。
  出力するメッセージについては，「12.2　メッセージの出力先」を参照してください。
- ローテーションを行い，バックアップが作成されます。
  - 環境ファイルの LOG_FILE_SIZE パラメーターに指定されたファイルサイズを超える直前に，システム実行ログのファイル名を変更してバックアップを作成し，新たにシステム実行ログを作成して出力を継続します。
  - バックアップのファイル名は，AdshLog_N.log（N は整数）となります。N には新しいバックアップから昇順に 1 から番号を割り当てます。
  - 環境ファイルの LOG_FILE_CNT パラメーターに指定された面数のバックアップを作成し，面数を超えた場合は古いバックアップを削除します。

(b) システム実行ログを出力するために必要なパラメーター

システム実行ログを出力するために必要なパラメーターを次に示します。

- LOG_DIR パラメーター
  システム実行ログを出力するディレクトリのパス名を定義します。
- LOG_FILE_CNT パラメーター
  システム実行ログをバックアップする面数を定義します。
- LOG_FILE_SIZE パラメーター
  システム実行ログを出力するファイルサイズを定義します。

複数のユーザーが同じファイルにシステム実行ログを出力している場合には，LOG_FILE_CNT と LOG_FILE_SIZE は，最後にシステム実行ログの出力を開始したユーザーの設定値が有効になります。このため，LOG_FILE_CNT と LOG_FILE_SIZE は同じ値で運用することを推奨します。

(c) システム実行ログの出力内容

システム実行ログに出力されるメッセージの例を次に示します。

```
seqnum=1, date=2013-12-06T18:27:12.045+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX0004-I, msg="ジョブ識別子=000070, JP1NBQSQueueName=, ジョブ番号="
seqnum=2, date=2013-12-06T18:27:12.060+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX0091-I, msg="JOB1 ジョブが開始しました。"
seqnum=3, date=2013-12-06T18:27:12.060+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX7901-I, msg="ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了
を待ちます。"
seqnum=4, date=2013-12-06T18:27:12.060+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX7902-I, msg="ジョブコントローラは，""端末入力モード""で動作します。"
seqnum=5, date=2013-12-06T18:27:12.062+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX0092-I, msg="JOB1.STEP1 ステップが開始しました。"
seqnum=6, date=2013-12-06T18:27:12.076+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX6116-I, msg="コマンド（D:¥bin¥uap01.exe, 行番号=5）が正常終了しました。rc=0 E-
Time=0.010s C-Time=0.000s"
seqnum=7, date=2013-12-06T18:27:12.076+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX6597-I, msg="JOB1.STEP1 ジョブステップが正常終了しました。rc=0 E-Time=0.015s C-
Time=0.016s"
seqnum=8, date=2013-12-06T18:27:12.076+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX0092-I, msg="JOB1.STEP2 ステップが開始しました。"
seqnum=9, date=2013-12-06T18:27:12.087+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX6116-I, msg="コマンド（D:¥bin¥uap02.exe, 行番号=10）が正常終了しました。rc=0 E-
Time=0.009s C-Time=0.015s"
seqnum=10, date=2013-12-06T18:27:12.087+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX6597-I, msg="JOB1.STEP2 ジョブステップが正常終了しました。rc=0 E-Time=0.011s C-
Time=0.015s"
seqnum=11, date=2013-12-06T18:27:12.087+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX0092-I, msg="JOB1.STEP2 ステップが開始しました。"
seqnum=12, date=2013-12-06T18:27:12.097+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX6116-I, msg="コマンド（D:¥bin¥uap03.exe, 行番号=15）が正常終了しました。rc=0 E-
Time=0.007s C-Time=0.000s"
seqnum=13, date=2013-12-06T18:27:12.097+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX6597-I, msg="JOB1.STEP2 ジョブステップが正常終了しました。rc=0 E-Time=0.010s C-
Time=0.000s"
```

```
seqnum=14, date=2013-12-06T18:27:12.097+09:00, pgmid=adshexec, jobid=70, pid=6616,
msgid=KNAX0098-I, msg="JOB1 ジョブが終了しました。rc=0 E-Time=0.041s C-Time=0.031s"
```

システム実行ログで，メッセージテキストの前に付加されるデータの意味を次に示します。

システム実行ログに出力されるデータ	データの意味
seqnum	メッセージの通し番号
date	出力した日時（yyyy-mm-ddThh:mm:ss.sssTZD の形式）
pgmid	プログラム ID ジョブコントローラの場合，adshexec が出力されます。
jobid	ジョブ識別子
pid	プロセス ID
msgid	出力メッセージのメッセージ ID
msg	出力メッセージのメッセージテキスト

(4)　トレースログの出力情報を定義する

トレースログは，JP1/Advanced Shell の内部トレースログです。JP1/Advanced Shell でトラブルが発生した場合に，問題点を解明するために採取する情報です。

トレースログの種類を次の表に示します。

表 2-20　トレースログの種類

項番	トレースログの種類	トレースログの出力先（デフォルト値）	ファイル面数	ファイルサイズ
1	JP1/Advanced Shell の実行環境（Windows および UNIX）のトレースログ※ および JP1/Advanced Shell のジョブ定義スクリプト稼働実績情報出力コマンド（adshevtout コマンド）のトレースログ※	• Windows の場合 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥trace • UNIX の場合 /opt/jp1as/trace	《4》 ((1～64))	《2》 ((1～16))
2	JP1/Advanced Shell - Developer のエディタ以外のトレースログ※	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥trace	《4》 ((1～64))	《2》 ((1～16))
3	JP1/Advanced Shell のカスタムジョブのトレースログ	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASV¥trace	1	1
4	JP1/Advanced Shell - Developer のエディタのトレースログ	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥adshedit¥trace	1	1

項番	トレースログの種類	トレースログの出力先（デフォルト値）	ファイル面数	ファイルサイズ
5	JP1/Advanced Shell，JP1/Advanced Shell - Developer の共通コマンドのトレースログ	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥misc¥trace	4	2

注※

トレースログの出力先，面数，およびファイルサイズは，環境設定パラメーターで変更できます。

トレースを定義するために必要なパラメーターを次に示します。

- TRACE_DIR パラメーター
  トレースを出力するディレクトリのパス名を定義します。トレースログのファイル名は AdshTrace_n.log（n は面数）となります。
- TRACE_FILE_CNT パラメーター
  トレースを出力する面数を定義します。トレースログファイルは，指定した面数のファイルを順番にラップアラウンドして使用します。
- TRACE_FILE_SIZE パラメーター
  トレースを出力するファイルサイズを定義します。
- TRACE_LEVEL パラメーター
  トレースを出力するレベルを定義します。

複数のユーザーが同じファイルにトレースを出力している場合，次のように動作します。

- TRACE_FILE_CNT と TRACE_FILE_SIZE は，それぞれ，より大きい値を指定したユーザーの指定が有効になります。
- TRACE_FILE_CNT と TRACE_FILE_SIZE を変更した場合は，既存のトレースファイルの面数およびファイルサイズの設定値と比較して，それぞれ，より大きい値を指定したユーザーの指定が有効になります。

トレースファイルの面数およびファイルサイズを小さくする場合は，トレースフォルダにあるファイルをすべて削除してください。

トレースフォルダにあるファイルをすべて削除する場合は，同じトレースファイルにトレースを出力しているジョブがないことを事前に確認してください。

2.6.9 スクリプト拡張コマンドの終了コードを定義する

スクリプト拡張コマンドが失敗および成功した場合の終了コードを標準の値から変更したいときに指定します。

- ADSHCMD_RC_ERROR パラメーター：スクリプト拡張コマンドが失敗した場合の終了コードを定義します。
- ADSHCMD_RC_SUCCESS パラメーター：スクリプト拡張コマンドが成功した場合の終了コードを定義します。

詳細については，「5.8.7　スクリプト拡張コマンドの終了コードとエラー発生時の動作」時のを参照してください。

2.6.10　複数の環境で共用する

次の環境設定パラメーターで指定するディレクトリを別々に分けることで，同一ホスト内で複数環境を使い分けられます。

- LOG_DIR パラメーター
- SPOOL_DIR パラメーター
- TEMP_FILE_DIR パラメーター
- TRACE_DIR パラメーター

クラスタ運用で待機系サーバに情報を引き継ぎたい場合は，引き継ぐディレクトリをホスト間で共用します。その場合は，少なくとも次のパラメーターのディレクトリをホスト間で共用します。

- SPOOL_DIR パラメーター

2.6.11　バッチジョブの実行時にカバレージ情報を採取するオプションを指定しなくても有効にする

次の環境設定パラメーターでカバレージ情報を採取するように設定しておけば，adshexec コマンドによるバッチジョブの実行時にカバレージ情報を採取するオプション（-t）を指定しなくても有効にする機能です。

- BATCH_CVR パラメーター：カバレージ採取の一括有効化機能を使用します。
- ASC_FILE パラメーター：カバレージ採取の一括有効化機能で使用する蓄積ファイル名の生成規則を定義します。

環境ファイルの設定例と実行するコマンドを次に示します。

(1)　環境ファイルの設定例

環境ファイルに次の例のようなパラメーターを設定しておきます。

```
#-adsh_conf BATCH_CVR YES
#-adsh_conf ASC_FILE ./cvrg/ver001-*
```

設定例の意味を行ごとに次に示します。

1. BATCH_CVR：カバレージ採取の一括有効化機能を使用する。
2. ASC_FILE：カバレージ採取の一括有効化機能で使用する蓄積ファイル名の生成規則を定義する。

(2) 実行するコマンド

上記の環境ファイルの設定で次のコマンドを実行します。

```
adshexec sample.ash
```

この場合，「adshexec -t -o ./cvrg/ver001-sample sample.ash」と指定して adshexec コマンドを実行したときと同じ動作になります。ただし，-t オプションを指定して「adshexec -t sample.ash」と指定して adshexec コマンドを実行したときは，終了コード 1 となり，エラー終了します。

2.6.12 ジョブ定義スクリプトを UNIX から Windows へ移行する

UNIX のジョブ定義スクリプトを Windows のジョブ定義スクリプトに移行する場合，次の手順を実施します。なお，この手順を実施する前に，ジョブ定義スクリプトと環境ファイルのエンコーディングが，移行するプラットフォームで使用する LANG 環境変数と一致していることを確認してください。

1. パスを変換する機能を有効にする。

   UNIX のジョブ定義スクリプトの区切り文字を Windows のプラットフォームに変換するために，環境ファイルに次のパラメーターを指定します。

   ```
   #-adsh_conf PATH_CONV_ENABLE / :
   ```

2. 記述されたパスを変換するための設定を実施する。

   ジョブ定義スクリプト中で，プログラムのパスを明示的に指定している場合は，Windows の環境に合わせて変換するために，環境ファイルに次のパラメーターを指定します。UNIX 互換コマンドのパスを変換する例を記載します。

   ```
   #-adsh_conf PATH_CONV /opt/jp1as "C:¥¥Program Files¥¥HITACHI¥¥JP1AS¥¥JP1ASE"
   ```

3. パスを扱うシェル変数の周りの区切り文字を変換するための設定を実施する。

   ジョブ定義スクリプト中でプログラムのパスをシェル変数を使って記述している場合は，そのシェル変数を使用しているパスの区切り文字を変換するためにパスを扱うシェル変数を指定します。パスを扱うシェル変数は環境ファイル，またはジョブ定義スクリプトで指定します。

   - 環境ファイルでは次のように指定します。

     ```
     #-adsh_conf PATH_CONV_VAR VAR
     ```

- ジョブ定義スクリプトでは次のコマンドを追加します。

```
#-adsh_path_var VAR
```

4. パス変換ルールを選択する。【Windows 限定】

   パスの変換ルールを PATH_CONV_RULE パラメーターで指定します。

   ダブルクォーテーション（"）で囲んだ個所だけを変換したい場合，次のようにパス変換ルール 1 を指定します。

```
#-adsh_conf PATH_CONV_RULE  1
```

   ダブルクォーテーション（"）で囲んだ個所以外も変換したい場合，次のようにパス変換ルール 2 を指定します。

```
#-adsh_conf PATH_CONV_RULE  2
```

5. 変換が有効であることを確認する。

   パス変換ルール 1 の場合，パスを変換したい個所が次の例のようにダブルクォーテーション（"）で囲まれていることを確認します。

```
”$VAR/cmd/ls” -l ”/opt/jp1as/sample”
```

   パス変換ルール 2 の場合，変換したくない個所がシングルクォーテーション（'）で囲まれていることを確認します。

6. パスの変換結果を事前に確認する。

   ジョブ定義スクリプトの文法チェック（adshexec -c コマンド）を実施し，生成されたスクリプトイメージでパスの変換結果を確認します。変換結果が適切でない場合は，パス変換ルールを変更するか，ジョブ定義スクリプトを変更して再実行してください。

パス変換ルール 1 とパス変換ルール 2 の変換例を次に示します。

- パス変換ルール 1 での変換例

  ＜環境設定パラメーター＞

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_RULE 1
#-adsh_conf PATH_CONV /opt/jp1as ”c:¥¥Program Files¥¥HITACHI¥¥JP1AS¥¥JP1ASE”
```

  ＜変換例＞

```
D:¥home¥user001>”C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin¥adshexec” -c sample1.ash

***** D:¥home¥user001¥sample1.ash *****
0001 : #-adsh_job SAMPLE
0002 : #-adsh_path_var VAR
0003 :
0004 : VAR=/opt/jp1as
0005 : ”$VAR/cmd/ls” -l ”/opt/jp1as/sample”
0006 :

***** パス変換行（D:¥home¥user001¥sample1.ash） *****
```

```
0005 : ”$VAR¥¥cmd¥¥ls” -l ”c:¥¥Program Files¥¥HITACHI¥¥JP1AS¥¥JP1ASE¥¥sample”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0

D:¥home¥user001>
```

- パス変換ルール 2 での変換例

  ＜環境設定パラメーター＞

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_RULE 2
#-adsh_conf PATH_CONV /opt/jp1as ”c:¥¥Program Files¥¥HITACHI¥¥JP1AS¥¥JP1ASE”
```

  ＜変換例＞

```
D:¥home¥user001>”C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin¥adshexec” -c sample1.ash

***** D:¥home¥user001¥sample1.ash *****
0001 : #-adsh_job SAMPLE
0002 : #-adsh_path_var VAR
0003 :
0004 : VAR=/opt/jp1as
0005 : ”$VAR/cmd/ls” -l ”/opt/jp1as/sample”
0006 :

***** パス変換行（D:¥home¥user001¥sample1.ash） *****
0004 : VAR=”c:¥¥Program Files¥¥HITACHI¥¥JP1AS¥¥JP1ASE”
0005 : ”$VAR¥¥cmd¥¥ls” -l ”c:¥¥Program Files¥¥HITACHI¥¥JP1AS¥¥JP1ASE¥¥sample”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0

D:¥home¥user001>
```

2.6.13　シェル変数 ENV に指定されたファイルを読み込む

ジョブコントローラ起動時に環境変数 ENV に指定された.env ファイルを読み込むかどうかを，環境設定パラメーター（KSH_ENV_READ パラメーター）に指定できます。このパラメーターを省略した場合，OS ごとにデフォルト値が次のように異なります。

- YES（ENV ファイルを読み込む）：Linux および Windows の場合
- NO（ENV ファイルを読み込まない）：AIX，HP-UX および Solaris の場合

2.6.14　パイプの最終コマンドの実行プロセスを定義する

パイプの最終コマンドをカレントプロセスと別プロセスのどちらで実行するかは，環境設定パラメーター PIPE_CMD_LAST で次のように指定します。

- CURRENT：カレントプロセスで実行する
- OTHER：別プロセスで実行する

- SEQUENTIAL【Windows 限定】：すべてのコマンドをカレントプロセスで逐次実行する

PIPE_CMD_LAST パラメーターを省略した場合，UNIX 版では CURRENT，Windows 版では SEQUENTIAL が適用されます。

2.6.15　ジョブを続行できないエラーが発生したときの終了コードを定義する

メモリ不足やジョブ定義スクリプトの解析エラーなど，ジョブの実行を継続できないエラーによってジョブが終了した場合，ジョブコントローラの終了コードは 1 となります。しかし，環境変数 ADSH_JOBRC_FATAL に値を設定しておくことで，この場合の終了コードを 1～255 に変更できます。

環境変数 ADSH_JOBRC_FATAL の設定方法については，「(2)　環境変数 ADSH_JOBRC_FATAL（ジョブ続行不可エラー発生時の終了コードを設定する）」の説明を参照してください。

(1)　環境変数 ADSH_JOBRC_FATAL の適用可否

ジョブコントローラの動作中に発生するエラーの種類と，それに対する環境変数 ADSH_JOBRC_FATAL での終了コード定義の適用可否を次の表に示します。

表 2-21　エラーの種類ごとの環境変数 ADSH_JOBRC_FATAL の適用可否

項番	エラーの発生時期	エラーの種類	適用可否※1
1	adshexec コマンドでのジョブ起動時， またはエディタからのジョブ定義スクリプトのデバッグ実行時	OS が adshexec コマンドの実行を開始できないエラーです。 例えば，adshexec コマンドが使用するロードモジュールが存在しないエラーが該当します。	×
2	ジョブ定義スクリプトの実行前	環境変数 ADSH_JOBRC_FATAL の解析エラーです。	×※2
3		イベントファイルの初期化エラーです。	×
4		【UNIX 限定】シグナル受信によるエラーです。	×※3
5		【Windows 限定】TerminateProcess などによって，次に示すどれかのプロセスを即時終了したことによるエラーです。 • adshexec.exe • adshexecsub.exe • adshesub.exe • adshedit.exe	×
6		項番 2～項番 5 以外で発生するすべてのエラーです。代表的なエラーは次のとおりです。 • adshexec コマンドのコマンドライン解析エラー	○

項番	エラーの発生時期	エラーの種類	適用可否※1
6	ジョブ定義スクリプトの実行前	• adshexec コマンドの引数に指定したジョブ定義スクリプトファイルの状態不正エラー • 環境ファイル解析エラー • ジョブ定義スクリプトの解析エラー • ジョブ実行ログ，システム実行ログ，トレースログの初期化エラー • asc ファイルの初期化エラー • 【Windows 限定】ライセンスチェックエラー • 【Windows 限定】制御信号（CTRL + C，CTRL + BREAK，CTRL_CLOSE_EVENT）の受信によるエラー	○
7	ジョブ定義スクリプトの実行中	ジョブ定義スクリプトの処理を続行するエラーです。※4	×
8		デバッグ実行中に，次に示すデバッガのコマンドやメニュー，ボタンを使用してジョブを途中で終了させたことによるエラーです。 • 【UNIX 限定】kill コマンドの実行，quit コマンドの実行および run コマンドの再実行 • 【Windows 限定】［デバッグの中止］メニューの選択，［デバッグの中止］ボタンのクリック，およびエディタウィンドウを閉じることによるデバッグ実行の中断	×
9		【UNIX 限定】シグナル受信によるエラーです。	×※3
10		【Windows 限定】TerminateProcess などによって，次に示すどれかのプロセスを即時終了したことによるエラーです。 • adshexec.exe • adshexecsub.exe • adshesub.exe • adshedit.exe	×
11		項番 7～項番 10 以外で発生するすべてのエラーです。代表的なエラーを次に示します。 • 特殊組み込みコマンドのエラー（ただし，typeset コマンドのエラー，関数内または外部スクリプト内で実行した return コマンドのエラーを除く） • 代入演算のエラー※5 • ジョブを終了する書式の変数置換の実行で発生するエラー※6 • 配列の要素の範囲外（0～65535 以外）を指定したことによるエラー • メモリやディスク容量不足など資源が確保できないエラー • 入出力エラー • 内部矛盾エラー • 【Windows 限定】制御信号（CTRL + C，CTRL + BREAK，CTRL_CLOSE_EVENT）の受信によるエラー • 【Windows 限定】演算子-h，-G，-L，-O，または-ef を使った条件式を実行したことによるエラー（UNSUPPORT_TEST パラメーターで ERR 以外を指定した場合は除く）	○
12	ジョブ定義スクリプトの実行後	【UNIX 限定】シグナル受信によるエラーです。	×※3
13		ファイルやディレクトリに関する，次に示すエラーです。 • スクリプト拡張コマンドで割り当てたファイルの解放エラー	×

項番	エラーの発生時期	エラーの種類	適用可否※1
13	ジョブ定義スクリプトの実行後	• スプールジョブ管理ファイルの後処理エラー • ルートジョブのスプールジョブディレクトリの後処理エラー • イベントファイルの後処理エラー	×
14		【Windows 限定】TerminateProcess などによって，次に示すどれかのプロセスを即時終了したことによるエラーです。 • adshexec.exe • adshexecsub.exe • adshesub.exe • adshedit.exe	×
15		項番 12～項番 14 以外で発生するすべてのエラーです。代表的なエラーを次に示します。 • asc ファイルの後処理エラー • 子孫ジョブのスプールジョブディレクトリの削除エラー • 【UNIX 限定】.DBG ファイルの解析エラー • 内部矛盾エラー • 【Windows 限定】制御信号（CTRL + C，CTRL + BREAK，CTRL_CLOSE_EVENT）の受信によるエラー	○

（凡例）

○：環境変数 ADSH_JOBRC_FATAL の設定が適用されます。

×：環境変数 ADSH_JOBRC_FATAL の設定が適用されません。

注※1

環境変数 ADSH_JOBRC_FATAL の設定の適用可否は，adshexec コマンドで指定したオプションには左右されません。例えば，adshexec コマンドに-c オプションを指定して実行し，文法エラーが発生した場合にも適用されます。

適用可否は OS によって次のように異なります。

- UNIX 版で adshexec コマンドに-d オプションを指定した場合

  デバッガと，run コマンドで実行したデバッグ対象のジョブとで，この表の次に示す項番が適用可否の判定対象となります。

  デバッガ：項番 1～項番 6，項番 12～項番 15

  デバッグ対象のジョブ：項番 7～項番 15

- Windows 版でエディタからデバッグを実行した場合

  デバッグ対象のジョブに対して，この表の項番 1～項番 15 が適用可否の判定対象となります。

注※2

終了コードは 255 になります。

注※3

ジョブコントローラがシグナルを受信してエラー終了した場合のジョブの終了コードは「128+シグナル番号」になります。

注※4

ジョブ定義スクリプトの実行中にスクリプト拡張コマンドがエラーとなった場合，次に示す後続のジョブステップとコマンドは実行されませんが，続行できないエラーとは見なされません。そのため，環境変数 ADSH_JOBRC_FATAL の設定は適用されません。

- run 属性が省略または normal のジョブステップ
- ジョブステップ外の命令

注※5

正規組み込みコマンドの引数に代入演算を指定した場合は除きます。

代入演算の指定分類と環境変数 ADSH_JOBRC_FATAL の設定の適用可否を次の表に示します。この表ではエラーとなる例として，「readonly NUM」で読み込み専用として定義されている変数 NUM に対し，値を代入しようとしてエラーとなるケースを使用しています。

代入演算の指定分類	エラーとなる例	適用可否
代入演算をそのまま指定した場合	NUM=100	○
代入演算をスクリプト予約語コマンドの引数に指定した場合	time NUM=100	○
代入演算を特殊組み込みコマンドの引数に指定した場合	export NUM=100	○
代入演算を正規組み込みコマンドの引数に指定した場合	let NUM=100	×

（凡例）

○：環境変数 ADSH_JOBRC_FATAL の設定が適用されます。

×：環境変数 ADSH_JOBRC_FATAL の設定が適用されません。

注※6

変数置換の結果，変数の状態によってはジョブが続行できないためエラー終了する場合があります。エラーとなる変数の状態を書式ごとに次に示します。

${variable:?[word]}

variable が定義済みで値が NULL（空文字列）の場合，または variable が未定義の場合，エラーとなります。

${variable?[word]}

variable が未定義の場合，エラーとなります。

(2) 環境変数 ADSH_JOBRC_FATAL（ジョブ続行不可エラー発生時の終了コードを設定する）

ジョブが続行できなくなってエラー終了した場合のジョブコントローラの終了コードを設定します。設定した終了コードは，adshexec コマンドまたは JP1/Advanced Shell - Developer のエディタから実行されたジョブに対して適用されます。

システム全体でこの環境変数の設定値を有効にする方法を次に示します。

- Windows の場合
  システム環境変数として環境変数 ADSH_JOBRC_FATAL を定義する。
- UNIX 版の場合
  /etc/profile に環境変数 ADSH_JOBRC_FATAL の設定処理を記述する。

この環境変数を設定しなかった場合，ジョブが続行できないエラーで終了した際のジョブコントローラの終了コードは 1 となります。

(a) 環境変数に設定できる値

終了コード　～<符号なし整数>((1～255))

ジョブ続行不可時の終了コードを指定します。前の桁を 0 で埋めて「001」のように指定した場合は，上位のゼロを削除して「1」と扱います。

(b) 注意事項

- 環境ファイルの export パラメーターを使用して環境変数 ADSH_JOBRC_FATAL を定義した場合や，シェル変数 ENV に指定したファイル内およびジョブ定義スクリプト内で環境変数 ADSH_JOBRC_FATAL の定義・変更をした場合，そのジョブ内ではこの機能は有効になりません。ただし，そのジョブから起動した別のジョブで有効になります。
- 環境変数 ADSH_JOBRC_FATAL はジョブの最終的な終了コードを定義します。各コマンドやジョブステップの終了コードには影響しません。
- 次の値が設定された場合，ジョブは実行されないで終了コード 255 でエラー終了します。
  - 4 文字以上の値（例：1234）
  - 指定範囲外の値（例：500）
  - 数値以外の文字（例：1A4，+8，8.0）
  - 0 文字の値（空文字列）
- エラー発生時に環境変数 ADSH_JOBRC_FATAL を適用するかどうかの判定は，ジョブごとに独立しています。そのため，ルートジョブや子孫ジョブの中だけでジョブの実行を継続できないエラーが発生した場合に，ほかのルートジョブや子孫ジョブに対して環境変数 ADSH_JOBRC_FATAL が適用されて終了コードが変更されることはありません。

(c) 使用例

UNIX 版で環境変数 ADSH_JOBRC_FATAL に 8 を設定して起動したジョブが続行不可エラーで終了した場合の例を次に示します。

環境ファイルの SPOOL_DIR パラメーターに指定したディレクトリが存在しなかったため，ジョブを続行できなかった場合

/etc/profile の記載内容

```
ADSH_JOBRC_FATAL=8
export ADSH_JOBRC_FATAL
```

ジョブ起動時のコマンド指定内容

```
$ /opt/jp1as/bin/adshexec test.sh
```

この場合の実行結果を次に示します。

```
KNAX0441-E 環境ファイルのパラメーター（”SPOOL_DIR”）で指定したディレクトリは存在しませ
ん。line=1
KNAX0410-E 環境ファイル（”sample.ase”）の解析でエラーが発生しました。エラーの内容はこの
メッセージの前に出力されているメッセージを参照してください。
KNAX0240-I 環境変数（ADSH_JOBRC_FATAL）の設定（8）が適用されました。 …1.
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。
rc=8                                                                    …2.
```

実行結果の 1.と 2.について次に説明します。

1. 環境変数 ADSH_JOBRC_FATAL が適用されたことを示すメッセージです。
2. ジョブコントローラの終了コードとして，環境変数 ADSH_JOBRC_FATAL の設定値が適用されています。

特殊組み込みコマンド（unset コマンド）のエラーで子孫ジョブが終了した場合

/etc/profile の記載内容

```
ADSH_JOBRC_FATAL=8
export ADSH_JOBRC_FATAL
```

環境ファイルの内容

```
#-adsh_conf CHILDJOB_SHEBANG /bin/sh
```

ルートジョブのジョブ定義スクリプト（prt.sh）の内容

```
./cld.sh
./cmdA
```

子孫ジョブのジョブ定義スクリプト（cld.sh）の内容

```
#!/bin/sh
val=10
unset
./cmdX $val
```

ジョブ起動時のコマンド指定内容

```
$ /opt/jp1as/bin/adshexec prt.sh
```

この場合の実行結果を次に示します。

```
********  ジョブコントローラのメッセージ出力  ********
14:59:57 000040 KNAX0091-I ADSH000040 ジョブが開始しました。
14:59:57 000040 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
14:59:57 000040 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
14:59:57 000040 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則
に合致しました。script=”./cld.sh” shebang=”/bin/sh”

>>>>>> [JOBLOG] /home/usr/cld.sh
14:59:57 000041 KNAX6571-I ADSH000041 子孫ジョブを開始しました。parent
jobname=ADSH000040, parent jobid=000040
14:59:57 000041 KNAX6572-I 子孫ジョブ（ADSH000041）はジョブ環境ファイル（”/opt/jp1as/
conf/adsh.conf”）を使用します。
14:59:57 000041 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
14:59:57 000041 KNAX6110-I コマンド（val=10, 行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:59:57 000041 KNAX6015-E 引数が必要な組み込みコマンドに対して，引数を指定しないで実行し
ました。filename=”/home/usr/cld.sh” line=3
14:59:57 000041 KNAX6521-E コマンド（unset, 行番号=3）がエラー終了しました。rc=1 E-
Time=0.000s C-Time=0.000s
14:59:57 000041 KNAX6584-I ジョブ定義スクリプトの実行を停止するコマンドが実行されたため，
バッチジョブの実行を中断しました。
14:59:57 000041 KNAX0101-E ADSH000041 ジョブを実行中にエラーが発生しました。
14:59:57 000041 KNAX6578-I ADSH000041 子孫ジョブが終了しました。rc=8 E-Time=0.001s C-
Time=0.000s
<<<<<< [JOBLOG] /home/usr/cld.sh

14:59:57 000040 KNAX6521-E コマンド（./cld.sh, 行番号=1）がエラー終了しました。rc=8 E-
Time=0.012s C-Time=0.000s
14:59:57 000040 KNAX6116-I コマンド（./cmdA, 行番号=2）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
14:59:57 000040 KNAX0101-E ADSH000040 ジョブを実行中にエラーが発生しました。
14:59:57 000040 KNAX0098-I ADSH000040 ジョブが終了しました。rc=0 E-Time=0.016s C-
Time=0.000s

********   ジョブ定義スクリプトの内容     ********

***** /home/usr/prt.sh *****
0001 : ./cld.sh
0002 : ./cmdA
0003 :

***** パス変換情報 *****

***** /home/usr/cld.sh *****
0001 : #!/bin/sh
0002 : val=10
0003 : unset
0004 : ./cmdX $val
0005 :

***** パス変換情報 *****
```

```
********    実行ジョブのSTDERRファイルの内容      ********

>>>>>> [STDERR] /home/usr/cld.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000041
KNAX0101-E ADSH000041 ジョブを実行中にエラーが発生しました。
KNAX0240-I 環境変数（ADSH_JOBRC_FATAL）の設定（8）が適用されました。
<<<<<< [STDERR] /home/usr/cld.sh

KNAX0101-E ADSH000040 ジョブを実行中にエラーが発生しました。
KNAX0098-I ADSH000040 ジョブが終了しました。rc=0 E-Time=0.016s C-Time=0.000s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/000040-ADSH000040/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

実行結果の 1.と 2.について次に説明します。

1. 子孫ジョブの終了コードとして，環境変数 ADSH_JOBRC_FATAL の設定値が適用されています。
2. 子孫ジョブで環境変数 ADSH_JOBRC_FATAL が適用されたことを示すメッセージです。

Windows 版でシステム環境変数 ADSH_JOBRC_FATAL に 16 を設定して起動したジョブが続行不可エラーで終了した場合の例を次に示します。

環境ファイルの LOG_DIR パラメーターに指定したディレクトリが存在しなかったため，ジョブの実行を継続できなかった場合

システム環境変数に次の変数と値を設定しておきます。

- 変数：ADSH_JOBRC_FATAL
- 値：16

ジョブ起動時のコマンド指定内容

```
adshexec test.ash
```

この場合の実行結果を次に示します。

```
KNAX0441-E 環境ファイルのパラメーター（"LOG_DIR"）で指定したディレクトリは存在しません。
line=1
KNAX0410-E 環境ファイル（"sample.ase"）の解析でエラーが発生しました。エラーの内容はこの
メッセージの前に出力されているメッセージを
参照してください。
KNAX0240-I 環境変数（ADSH_JOBRC_FATAL）の設定（16）が適用されました。…1.
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。
rc=16                                                                    …2.
```

実行結果の 1.と 2.について次に説明します。

1. システム環境変数 ADSH_JOBRC_FATAL が適用されたことを示すメッセージです。
2. ジョブの終了コードとして，システム環境変数 ADSH_JOBRC_FATAL の設定値が適用されています。

特殊組み込みコマンド（unset コマンド）のエラーで子孫ジョブが終了した場合

システム環境変数に次の変数と値を設定しておきます。

- 変数：ADSH_JOBRC_FATAL
- 値：16

環境ファイルの内容

```
#-adsh_conf CHILDJOB_SHEBANG /bin/sh
```

ルートジョブのジョブ定義スクリプト（prt.sh）の内容

```
./cld.sh
./cmdA
```

子孫ジョブのジョブ定義スクリプト（cld.sh）の内容

```
#!/bin/sh
val=10
unset
./cmdX $val
```

ジョブ起動時のコマンド指定内容

```
adshexec prt.sh
```

この場合の実行結果を次に示します。

```
********  ジョブコントローラのメッセージ出力  ********
17:36:17 000044 KNAX0091-I ADSH000044 ジョブが開始しました。
17:36:17 000044 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
17:36:17 000044 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
17:36:17 000044 KNAX6832-I CHILDJOB_EXTパラメーターによる子孫ジョブ実行の読み替え規則に合
致しました。script=”.¥cld.sh”

>>>>>> [JOBLOG] d:¥home¥usr¥cld.sh
17:36:17 000045 KNAX6571-I ADSH000045 子孫ジョブを開始しました。parent
jobname=ADSH000044, parent jobid=000044
17:36:17 000045 KNAX6572-I 子孫ジョブ（ADSH000045）はジョブ環境ファイル（”c:¥jp1as¥conf
¥adsh.conf”）を使用します。
17:36:17 000045 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
17:36:17 000045 KNAX6110-I コマンド（val=10，行番号=2）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
17:36:17 000045 KNAX6015-E 引数が必要な組み込みコマンドに対して，引数を指定しないで実行し
ました。filename=”d:¥home¥usr¥cld.sh” line=3
17:36:17 000045 KNAX6521-E コマンド（unset，行番号=3）がエラー終了しました。rc=1 E-
Time=0.001s C-Time=0.000s
17:36:17 000045 KNAX6584-I ジョブ定義スクリプトの実行を停止するコマンドが実行されたため，
バッチジョブの実行を中断しました。
17:36:17 000045 KNAX0101-E ADSH000045 ジョブを実行中にエラーが発生しました。
17:36:17 000045 KNAX6578-I ADSH000045 子孫ジョブが終了しました。rc=16 E-Time=0.006s C-
Time=0.015s
<<<<<< [JOBLOG] d:¥home¥usr¥cld.sh

17:36:17 000044 KNAX6521-E コマンド（.¥cld.sh，行番号=1）がエラー終了しました。rc=16 E-
Time=0.184s C-Time=0.171s
```

```
17:36:17 000044 KNAX6116-I コマンド（.¥cmdA.exe，行番号=2）が正常終了しました。rc=0 E-
Time=0.049s C-Time=0.046s
17:36:17 000044 KNAX0101-E ADSH000044 ジョブを実行中にエラーが発生しました。
17:36:17 000044 KNAX0098-I ADSH000044 ジョブが終了しました。rc=0 E-Time=0.254s C-
Time=0.233s

********   ジョブ定義スクリプトの内容    ********

***** d:¥home¥usr¥prt.sh *****
0001 : .¥¥cld.sh
0002 : .¥¥cmdA
0003 :

***** パス変換情報 *****

***** d:¥home¥usr¥cld.sh *****
0001 : #!/bin/sh
0002 : val=10
0003 : unset
0004 : .¥¥cmdX $val
0005 :

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容    ********

>>>>>> [STDERR] d:¥home¥usr¥cld.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000045
KNAX0101-E ADSH000045 ジョブを実行中にエラーが発生しました。
KNAX0240-I 環境変数（ADSH_JOBRC_FATAL）の設定（16）が適用されました。
<<<<<< [STDERR] d:¥home¥usr¥cld.sh

KNAX0101-E ADSH000044 ジョブを実行中にエラーが発生しました。
KNAX0098-I ADSH000044 ジョブが終了しました。rc=0 E-Time=0.254s C-Time=0.233s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="C:¥Documents and Settings¥All Users¥Documents¥Hitachi¥JP1AS¥JP1ASE¥spool
¥000044-ADSH000044¥"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

実行結果の 1.と 2.について次に説明します。

1. 子孫ジョブの終了コードとして，システム環境変数 ADSH_JOBRC_FATAL の設定値が適用されています。
2. 子孫ジョブでシステム環境変数 ADSH_JOBRC_FATAL が適用されたことを示すメッセージです。

2.6.16　ユーザー応答機能を設定する

ユーザー応答機能を使用する場合，JP1/Advanced Shell および JP1/IM での環境設定が必要です。これらの環境設定については，「2.8　ユーザー応答機能を設定する」を参照してください。

2.6.17　JP1 環境を確認する【UNIX 限定】

JP1 の環境は，/etc/opt/jp1base/conf/jp1bs_param.conf で決定されます。使用する環境に合わせて文字コードを設定してください。詳細については，マニュアル「JP1/Base 運用ガイド」を参照してください。

2.6.18　シェルを設定する【UNIX 限定】

JP1/AJS からジョブを起動する場合に使用するログインシェルを次に示します。該当するログインシェルが使用できるよう設定してください。

OS の種類	ログインシェル
Linux	bash
AIX	ksh
HP-UX	
Solaris	

注意

JP1/AJS からジョブを起動する場合，adshexec コマンドがログインシェルの子プロセスとして動作すると，強制終了などが発生した際に，adshexec コマンドのジョブ実行結果が JP1/AJS に渡る前にログインシェルのプロセスが終了し，JP1/AJS - View 上にジョブ実行結果が反映されないときがあります。

この現象を防ぐには，ログインシェルのプロセスを上書きしてから起動する処理になっているか，ログインスクリプトファイルの定義を確認してください（trap コマンドの指定を削除するなど）。

定義の詳細については，マニュアル「JP1/Automatic Job Management System 3 構築ガイド」またはマニュアル「JP1/Automatic Job Management System 3 トラブルシューティング」を参照してください。

2.6.19　JP1/Advanced Shell で必要なディレクトリを作成する

JP1/Advanced Shell のインストールが完了したあと，実行に必要なディレクトリをデフォルト設定から変更する場合，変更後のディレクトリを作成し，環境ファイルに指定します。

JP1/Advanced Shell で必要なディレクトリの種類と指定内容を次に示します。JP1/Advanced Shell を実行するユーザーは，これらのディレクトリに対して必要な権限を割り当ててください。

- 一時ファイル用のディレクトリ
  バッチジョブ内だけで利用するファイルを一時的に作成するディレクトリを指定します。
- スプール用のディレクトリ
  ジョブ実行ログやプログラム出力データファイルを格納するディレクトリを指定します。
- システム実行ログ用のディレクトリ

システム管理者がバッチジョブの実行を監視するために，バッチジョブの履歴をシステム実行ログとして保存するディレクトリを指定します。

- トレース用のディレクトリ

  システム障害時に，トラブルシューティングとして状況を保存するためのディレクトリを指定します。

JP1/Advanced Shell で必要なディレクトリを次に示します。環境設定パラメーターの設定方法については，「7. 環境ファイルで設定するパラメーター」を参照してください。

表 2-22 JP1/Advanced Shell で必要なディレクトリ

ディレクトリの種類	環境設定パラメーター	デフォルトディレクトリまたはパス	デフォルトの権限
一時ファイル用のディレクトリ	TEMP_FILE_DIR	• 実行環境【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥temp • 開発環境【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥temp • 実行環境【UNIX 限定】 /var/opt/jp1as/temp	CRWD【Windows 限定】 1777【UNIX 限定】
スプール用のディレクトリ	SPOOL_DIR	• 実行環境【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥spool • 開発環境【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥spool • 実行環境【UNIX 限定】 /var/opt/jp1as/spool	CRWD【Windows 限定】 1777【UNIX 限定】
システム実行ログ用のディレクトリ	LOG_DIR LOG_FILE_CNT LOG_FILE_SIZE	• 実行環境【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥log • 開発環境【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥log • 実行環境【UNIX 限定】 /opt/jp1as/log	CRWD【Windows 限定】 0777【UNIX 限定】
トレース用のディレクトリ	TRACE_DIR TRACE_FILE_CNT TRACE_FILE_SIZE TRACE_LEVEL	• 実行環境【Windows 限定】 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥trace • 開発環境【Windows 限定】 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥trace • カスタムジョブ定義プログラム【Windows 限定】 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASV¥trace • 実行環境【UNIX 限定】 /opt/jp1as/trace	CRWD【Windows 限定】 1777【UNIX 限定】

（凡例）

次に示すデフォルトの権限の英字は，権限の種類を示します。

Ｃ：作成，Ｒ：読み込み，Ｗ：書き込み，Ｄ：削除

(1)　必要な権限

バッチジョブを実行するユーザーに対して，必要な権限を次に示します。

(a)　Windows の場合

バッチジョブを実行するユーザーに対して，フルコントロールを設定してください。

(b)　UNIX の場合

バッチジョブを実行するユーザーに対して，各ディレクトリには次に示すファイルパーミッションが必要です。

表 2-23　ディレクトリのファイルパーミッション

ディレクトリの種類	読み込み許可（r）	書き込み許可（w）	実行許可（x）	スティッキービット（t）
一時ファイル用のディレクトリ	○	○	○	△
スプール用のディレクトリ	○	○	○	△
システム実行ログ用のディレクトリ	○	○	○	×
トレース用のディレクトリ	○	○	○	△

（凡例）

○：設定が必須です。

△：システムの運用方針に従って設定します。

×：設定しません。

ディレクトリのスティッキービットは，システムの運用方針に従い，設定します。

ディレクトリにスティッキービットの設定がない場合，ディレクトリに書き込み許可があれば，ディレクトリ直下の任意のファイルを削除できます。

ディレクトリにスティッキービットを設定した場合，ディレクトリに書き込み許可があっても，ディレクトリの所有者，またはファイルの所有者の場合だけ，ディレクトリ直下の任意のファイルを削除できます。

(2)　ファイルシステム

スプールなどは，業務によって大容量となる場合があるため，専用のファイルシステムを作成して使用することをお勧めします。

2.6.20　JP1/AJS 環境を設定する

JP1/AJS から JP1/Advanced Shell を実行するために，事前に実行環境を設定する必要があります。ここでは，JP1/AJS の環境設定の留意点について説明します。

(1)　JP1/AJS のログ容量の見積もり

JP1/AJS から JP1/Advanced Shell を実行する場合，次に示すジョブ実行内部ログに出力するログ出力量が 1 ジョブ当たり約 350 バイト増加します。

Windows Server 2008 および Windows Server 2012 の場合

```
%ALLUSERSPROFILE%¥Hitachi¥JP1¥JP1_DEFAULT¥JP1AJS2¥log¥jpqagent¥jpqagt_{00|01|02|03|04|05|
06|07}.log
```

%ALLUSERSPROFILE%のデフォルトは「**システムドライブ**¥Program Data」です。

UNIX の場合

```
/var/opt/jp1ajs2/log/jpqagent/jpqagt_{00|01|02|03|04|05|06|07}.log
```

JP1/AJS から JP1/Advanced Shell を実行する場合のログサイズの総量を見積もる場合は，JP1/AJS のマニュアルに記載されている計算式を参考に見積もりをしてください。

2.6.21　ジョブ強制終了時にユーザー固有の後処理を実行する

ジョブコントローラでは，JP1/AJS からの強制終了，UNIX 版での SIGTERM シグナルによる強制終了，および Windows での taskkill コマンドによる強制終了（TerminateProcess などによるプロセス即時終了）を受けたときにユーザー固有の後処理を実行できます。これによって，強制終了要求を受けたことを契機として独自の終了処理を実行するなど，柔軟な運用ができます。強制終了要求を受けたときにユーザー固有の後処理を実行する場合は，環境設定パラメーター TRAP_ACTION_SIGTERM を定義してください。

なお，環境設定パラメーター TRAP_ACTION_SIGTERM は UNIX 版と Windows 版で指定できるオペランドが異なります。環境設定パラメーター TRAP_ACTION_SIGTERM については，「7.3.47 TRAP_ACTION_SIGTERM パラメーター（ジョブコントローラが強制終了要求を受けたときの動作を定義する）」を参照してください。

使用例を次に示します。

環境ファイルの内容

```
#-adsh_conf TRAP_ACTION_SIGTERM TERM
```

ジョブ定義スクリプトの内容

```
#-adsh_job JOB01
trap "UAP_TERM" TERM
UAP01
```

→UAP01 実行中に強制終了要求を受けた場合，ジョブコントローラは「UAP_TERM」というユーザープログラムを実行したあと，子孫プロセスの強制終了といった後処理を実行してからジョブを終了します。

2.6.22　スクリプト開発部品を使うための準備

JP1/Advanced Shell では，スクリプト開発部品を使用することによって，ジョブ定義スクリプトの生産効率を向上させることができます。スクリプト開発部品の詳細については，「10.　スクリプト開発部品」を参照してください。

スクリプト開発部品は，次の手順で使用します。

1. スクリプト開発部品は次の場所に格納されています※。コピーして別の場所に格納してもかまいません。

   Windows の実行環境の場合

   インストール先フォルダ¥JP1ASE¥parts¥en

   インストール先フォルダ¥JP1ASE¥parts¥ja

   Windows の開発環境の場合

   インストール先フォルダ¥JP1ASD¥parts¥en

   インストール先フォルダ¥JP1ASD¥parts¥ja

   UNIX の実行環境の場合

   /opt/jp1as/parts/en

   /opt/jp1as/parts/ja

   注※ JP1/Advanced Shell のアンインストール時には削除されます。また，上書きインストール時には更新されます

2. 環境設定パラメーター CMDRC_CMDGRP_CHECK に FUNCTION を指定します。
3. 次のどれかの方法で，使用したいスクリプト開発部品を読み込みます。

   - .(ドット)コマンドでスクリプト開発部品のファイルを読み込む（#-adsh_script コマンドでも読み込み可能）。

     日本語のコメントが記述されたスクリプト開発部品 cmpDate を読み込む場合のジョブ定義スクリプトの例を次に示します。

     ```
     . "${ADSH_DIR_PARTS_JA}cmpDate"
     ```

     英語のコメントが記述されたスクリプト開発部品 getFileSize を読み込む場合のジョブ定義スクリプトの例を次に示します。

     ```
     . "${ADSH_DIR_PARTS_EN}getFileSize"
     ```

   - 関数のオートロード機能でスクリプト開発部品のファイルを読み込む

日本語のコメントが記述されたスクリプト開発部品を読み込む場合の環境ファイルの例を次に示します。

```
export FPATH=/opt/jp1as/parts/ja
```

各機能の使用方法については，次の項目を参照してください。

- 「9.3　シェル標準コマンド」の「9.3.1　.コマンド（シェルスクリプトを実行する）」
- 「9.5　スクリプト拡張コマンド」の「9.5.7　#-adsh_script コマンド（実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルを呼び出す）」
- 「(3)　関数のオートロード機能」

4. ジョブ定義スクリプトの中でスクリプト開発部品を呼び出します。

2.6.23　初期設定スクリプトファイルを実行する

JP1/Advanced Shell のジョブコントローラがジョブ定義スクリプトファイルを実行する前に，特定の共通スクリプトファイルを実行できます。ジョブ定義スクリプトファイルを実行する前に特定の共通スクリプトファイルを実行すると，システム共通の初期設定（変数への値代入，エイリアスの定義，作業ディレクトリへの移動など）ができます。

(1)　初期設定スクリプトファイルの位置づけ

ジョブコントローラでは，次のファイルをスクリプトファイルとして扱います。

```
ジョブコントローラが扱うスクリプトファイル
  |---ジョブ定義スクリプトファイル
  |---外部スクリプト
  |           |---#-adsh_scriptの外部スクリプトファイル
  |           |---.(ドット)コマンドの外部スクリプトファイル
  |           |---初期設定スクリプトファイル
  |
  |---.envファイル
  |---【Windows限定】UNIX互換コマンド（スクリプト形式）
```

初期設定スクリプトファイルは，外部スクリプトファイルの一種として位置づけられます。そのため，ジョブコントローラは，初期設定スクリプトファイルを，ジョブの一部である外部スクリプトファイルとして実行します。

(2)　初期設定スクリプトファイル

初期設定スクリプトファイルは，ジョブコントローラがジョブ定義スクリプトを実行する時に実行する初期化用のスクリプトファイルです。環境設定パラメーターINIT_SCRIPT_READ を定義すると，ルートジョブを実行する直前に初期化用のスクリプトファイルを読み込み，実行します。ただし，次の場合，ジョブコントローラは初期設定スクリプトを実行しないで，KNAX6504-E メッセージを出力し，エラー終了します。

- 初期設定スクリプトファイルが存在しない場合
- 実行に必要な権限が割り当てられていない場合

環境設定パラメーターINIT_SCRIPT_READ の指定と，初期設定スクリプトファイルの状態の組み合わせを次の表に示します。

表 2-24　INIT_SCRIPT_READ の指定と初期設定スクリプトファイルの状態の組み合わせ

項番	INIT_SCRIPT_READ の指定	初期設定スクリプトファイルの状態		ジョブコントローラの動作
		ファイルの存在	読み込み権限	
1	NO，または定義されていない	－	－	初期設定スクリプトファイルを実行しません。
2	YES	存在しない	－	
3		存在する	読み込み権限がない	
4			読み込み権限がある	初期設定スクリプトファイルを実行します。

（凡例）

－：該当なし

初期設定スクリプトファイルは，システム管理者が作成および設定します。作成した初期設定スクリプトファイルは，次の表に示すファイルパスに格納すると有効になります。

表 2-25　初期設定スクリプトファイルのファイル名

項番	環境	初期設定スクリプトファイルのファイル名
1	Windows（開発環境）	*共通アプリケーションフォルダ*¥HITACHI¥JP1AS¥JP1ASD¥conf¥adshinit_root.ash
2	Windows（実行環境）	*共通アプリケーションフォルダ*¥HITACHI¥JP1AS¥JP1ASE¥conf¥adshinit_root.ash
3	UNIX	/opt/jp1as/conf/adshinit_root.ash

ジョブコントローラは次の表に示す順序でファイルを読み込みます。そのため，環境ファイルと初期設定スクリプトファイルで同じ環境変数に値を設定すると，初期設定スクリプトファイルで設定した値が有効になります。

表 2-26　ジョブコントローラのファイル読み込み順序

順序	ファイル種別	備考
1	システム環境ファイル	－
2	ジョブ環境ファイル	－
3	.env ファイル	環境設定パラメーターKSH_ENV_READ にYES を指定した場合
4	初期設定スクリプトファイル	環境設定パラメーターINIT_SCRIPT_READ にYES を指定した場合

順序	ファイル種別	備考
5	ジョブ定義スクリプトファイル	－

（凡例）

－：該当なし

初期設定スクリプトファイルにはジョブコントローラ起動時の初期化処理を記述してください。ただし，次のコマンドは使用しないでください。

- adshappexec を除くシェル運用コマンド
- シェル拡張コマンドのadshecho，adshread，adshjoberr
- スクリプト拡張コマンド

初期設定スクリプトファイルに記述したコマンドの実行結果は，ジョブ定義スクリプトファイルに記述したコマンドの実行結果と同様に，ジョブ実行ログに出力します。また，初期設定スクリプトファイルの内容はスクリプトイメージファイルに出力しません。

(3) 注意事項

- ジョブ稼働中に初期設定スクリプトファイルの内容を変更しないでください。また，初期設定スクリプトファイルからジョブ定義スクリプトファイルの内容を変更する処理を実行しないでください。
- 初期設定スクリプトファイルのエンコーディング，ジョブ定義スクリプトファイルのエンコーディング，およびジョブ定義スクリプトを実行する環境のLANG 環境変数の値は，すべて一致させてください。一致していない場合の動作は保証できません。
- 初期設定スクリプトファイル内で継続できないエラーが発生した場合，ジョブコントローラはジョブ定義スクリプトを実行しないで，終了コード1，または環境変数ADSH_JOBRC_FATAL に設定した値でエラー終了します。
- 初期設定スクリプトファイルからルートジョブを起動した場合，次のように動作します。
  - 初期設定スクリプトファイルから起動したルートジョブは，ジョブ定義スクリプトファイルを実行しないで，終了コード1，または環境変数ADSH_JOBRC_FATAL に設定した値でエラー終了します。
  - 初期設定スクリプトファイルを実行するジョブコントローラは，処理を継続します。
- 初期設定スクリプトファイルはリンクファイルに対応していません。初期設定スクリプトファイル本体をリンクファイルに置き換えないでください。
- 初期設定スクリプトファイルはデバッグ実行の対象外です。初期設定スクリプトファイルを CUI および GUI デバッガでデバッグしたい場合は，環境設定パラメーターINIT_SCRIPT_READ にNO を指定した環境で，ジョブ定義スクリプトファイルとして実行してください。
- カバレージ情報の採取の対象は，ジョブ定義スクリプトファイルだけです。そのため，初期設定スクリプトファイルのカバレージ情報を採取したい場合は，ジョブ定義スクリプトファイルとして実行してください。

- 初期設定スクリプトファイルは，文法チェックモードの対象外です。初期設定スクリプトファイルの文法をチェックする場合は，ジョブ定義スクリプトファイルとして実行してください。
- 初期設定スクリプトファイル内で，関数情報配列は使用できません。

(4) 初期設定スクリプトファイルを実行するための準備

初期設定スクリプトファイルは次の手順で実行できます。

1. 環境設定パラメーターINIT_SCRIPT_READ にYES を指定します。

```
#-adsh_conf INIT_SCRIPT_READ YES
```

2. 実行する初期設定スクリプトファイルを「初期設定スクリプトファイルのファイル名」に示す場所に格納します。

(5) 使用例

初期設定スクリプトファイルが正常終了するケース

- 初期設定スクリプトファイルの内容

```
ADSH_SPOOL_JOBNAME=$( "${ADSH_DIR_CMD}basename" ${AJSJOBNAME} )
alias ls='"${ADSH_DIR_CMD}ls" -la'
cd 'C:¥ExecUser¥init'
```

- ジョブ定義スクリプトファイルの内容

```
#-adsh_step_start S01
  echo "Job step start"
#-adsh_step_end
```

- 実行結果

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちま
す。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000003
----------------------------------------------------------------
 Advanced Shell 11-10

 [ジョブ情報]
   ジョブ識別子                  : 000003
   スプールジョブディレクトリパス : C:¥ExecUser¥result¥spool¥000003¥
   実行日付                      : 2016/08/19
   システム環境ファイルパス       :
   ジョブ環境ファイルパス         : C:¥ExecUser¥conf.ase
   ホスト名                      : HOST0001
 [Automatic Job Management Systemから渡された環境変数]
   JP1JobName      : adshexec.exe"
   JP1JobID        : 0000002362
   JP1_USERNAME    : jobadmin
   JP1UNCName      : HOST0001
   JP1NBQSQueueName: ¥¥HOST0001¥@SYSTEM
   JP1Priority     : 64
```

```
   AJSEXECID       : @A665
   AJSJOBNAME      : /ExecUser/test/ASinit
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
14:51:19 000003 KNAX0091-I ADSH000003 ジョブが開始しました。
14:51:19 000003 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
14:51:19 000003 KNAX7902-I ジョブコントローラは，”非端末入力モード”で動作します。
14:51:19 000003 KNAX6501-I ジョブは初期設定スクリプトファイル（”C:¥ProgramData¥HITACHI
¥JP1AS¥JP1ASE¥conf¥adshinit_root.ash”）を実行します。
14:51:19 000003 KNAX6126-I コマンド（C:¥PROGRA~2¥Hitachi¥JP1AS¥JP1ASE¥cmd¥basename.exe,
コマンド置換）が正常終了しました。rc=0 E-Time=0.015s C-Time=0.015s
14:51:19 000003 KNAX6110-I コマンド（ADSH_SPOOL_JOBNAME=ASinit, 行番号=1）が正常終了しま
した。rc=0 E-Time=0.000s C-Time=0.000s
14:51:19 000003 KNAX6112-I コマンド（alias, 行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:51:19 000003 KNAX6112-I コマンド（cd, 行番号=3）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:51:19 000003 KNAX6502-I 初期設定スクリプトは終了しました。
14:51:19 000003 KNAX0092-I ADSH000003.S01 ステップが開始しました。
14:51:19 000003 KNAX6112-I コマンド（echo, 行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:51:19 000003 KNAX6597-I ADSH000003.S01 ジョブステップが正常終了しました。rc=0 E-
Time=0.032s C-Time=0.015s
14:51:19 000003 KNAX0098-I ADSH000003 ジョブが終了しました。rc=0 E-Time=0.063s C-
Time=0.046s

********   ジョブ定義スクリプトの内容     ********

***** C:¥ExecUser¥test.ash *****
0001 : #-adsh_step_start S01
0002 :   echo ”Job step start”
0003 : #-adsh_step_end

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容     ********
KNAX6597-I ADSH000003.S01 ジョブステップが正常終了しました。rc=0 E-Time=0.032s C-
Time=0.015s
KNAX0098-I ADSH000003 ジョブが終了しました。rc=0 E-Time=0.063s C-Time=0.046s

******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=S01 出力先=STDERR

KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”C:¥ExecUser¥result¥spool¥000003-ASinit¥”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

2.7　JP1/AJS の環境情報を設定する（JP1/AJS を使用する場合）

JP1/AJS を使用する場合の環境情報の設定について説明します。

JP1/AJS でジョブの実行を自動化する場合，JP1/AJS - View でジョブを登録します。JP1/AJS - View では，運用に使用するコマンド，バッチファイルなどをジョブとして定義して，それぞれのジョブの実行順序を関連づけることで，システム運用を自動化します。

JP1/AJS - View でジョブを定義する方法には，次の種類があります。

- カスタムジョブ
- PC ジョブ（Windows の場合）
- UNIX ジョブ（UNIX の場合）

JP1/AJS - View の詳細については，マニュアル「JP1/Automatic Job Management System 3 操作ガイド」を参照してください。

2.7.1　JP1/AJS - View でカスタムジョブを登録する

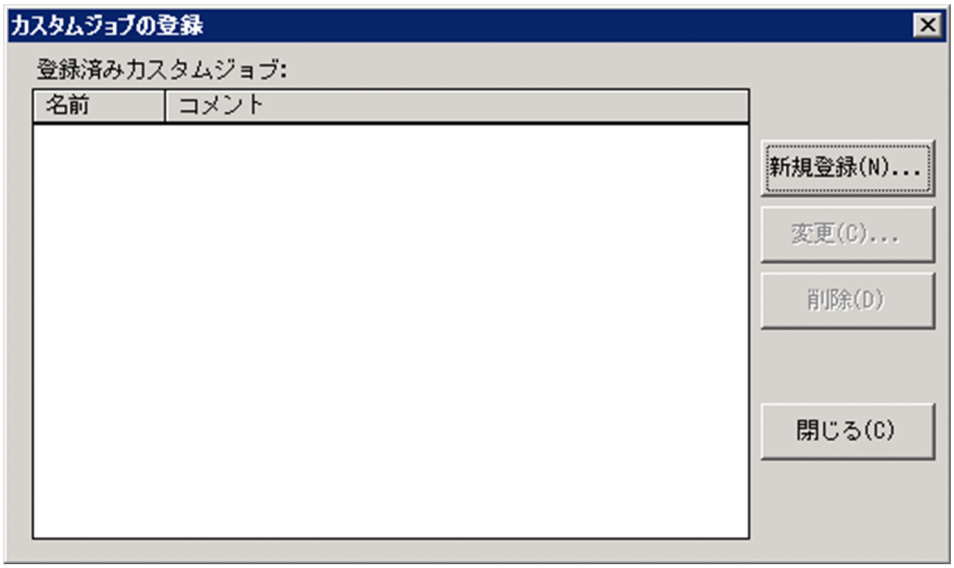
ジョブを定義する場合にカスタムジョブを使用すると，直接コマンドやバッチファイルをジョブに指定するのに比べて，容易で間違いなく定義できます。

カスタムジョブとは，JP1/AJS - View 以外のプログラムと JP1/AJS - View が連携するジョブを定義する場合に，目的のジョブを容易に作成するために使用できるジョブのテンプレートです。

このカスタムジョブを使用すると，GUI 操作で JP1/Advanced Shell 用のジョブが定義できます。

JP1/AJS3 - View でのカスタムジョブの登録手順を次に示します。

1. Windows の［スタート］メニューから，［すべてのプログラム］－［JP1_Automatic Job Management System 3 - View］－［カスタムジョブ登録］を選択する。
   ［カスタムジョブの登録］ダイアログボックスが表示されます。

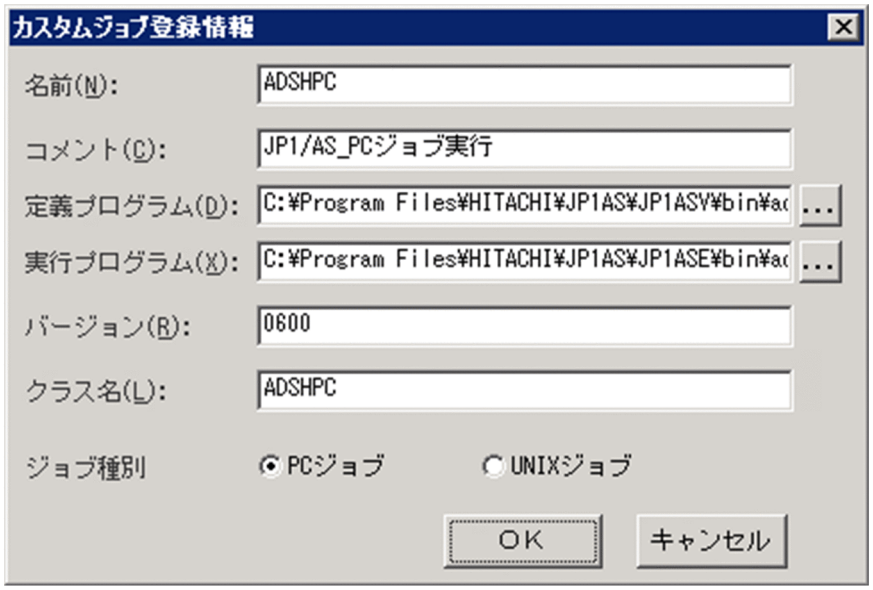


2. ［新規登録］ボタンをクリックする。

   ［カスタムジョブ登録情報］ダイアログボックスが表示されます。

3. JP1/Advanced Shell 用のカスタムジョブを登録する。実行環境が Windows の場合，UNIX の場合，GUI アプリケーション実行ジョブの場合でそれぞれ次のように登録する。

   - Windows の場合



名前：ADSHPC を指定します。

コメント：「JP1/AS_PC ジョブ実行」という固定の文字列を推奨します。ただし，1～40 バイトで任意の文字列を指定できます。また，省略もできます。

定義プログラム：**インストール先フォルダ**¥JP1ASV¥bin¥adshctmjpc.bat となります。カスタムジョブをインストールした PC のフォルダになります。

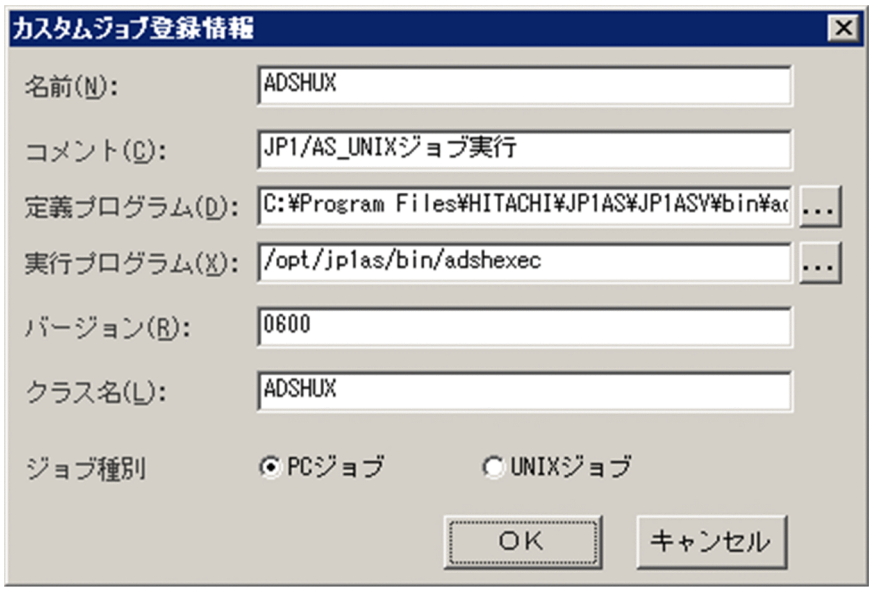
実行プログラム：**インストール先フォルダ**¥JP1ASE¥bin¥adshexec.exe となります。実行環境をインストールした PC のフォルダになります。

バージョン：0600。JP1/AJS - View のインターフェースのバージョンです。

クラス名：ADSHPC を指定します。

ジョブ種別：JP1/Advanced Shell 用のジョブの場合，必ず PC ジョブを選択します。

- UNIX の場合



名前：ADSHUX を指定します。

コメント：「JP1/AS_UNIX ジョブ実行」という固定の文字列を推奨します。ただし，1～40 バイトで任意の文字列を指定できます。また，省略もできます。

定義プログラム：**インストール先ディレクトリ**¥JP1ASV¥bin¥adshctmjunix.bat となります。カスタムジョブをインストールした PC のフォルダになります。

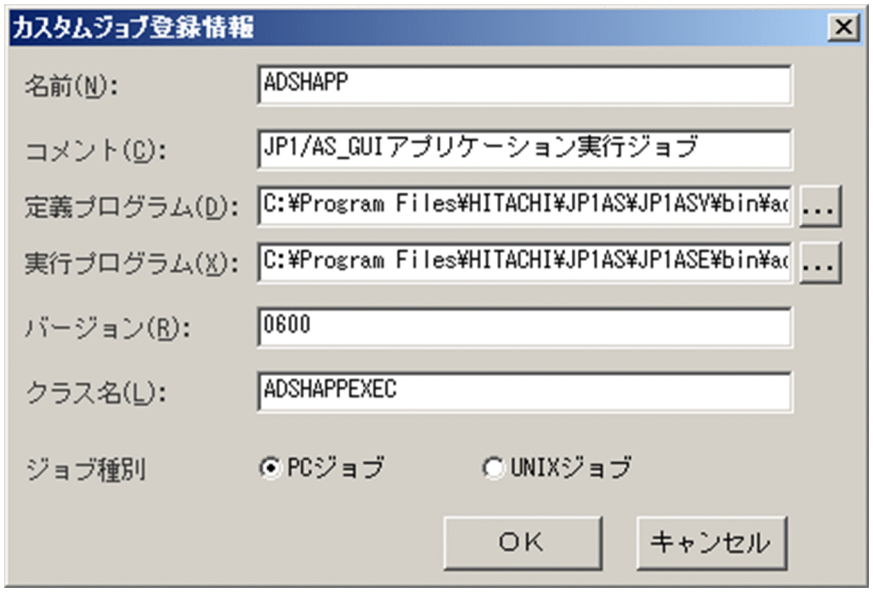
実行プログラム：/opt/jp1as/bin/adshexec となります。

バージョン：0600。JP1/AJS - View のインターフェースのバージョンです。

クラス名：ADSHUX を指定します。

ジョブ種別：JP1/Advanced Shell 用のジョブの場合，必ず PC ジョブを選択します。

- GUI アプリケーション実行ジョブの場合



名前：ADSHAPP を指定します。

コメント：「JP1/AS_GUI アプリケーション実行ジョブ」という固定の文字列を推奨します。ただし，1～40 バイトで任意の文字列を指定できます。また，省略もできます。

定義プログラム：**インストール先フォルダ**¥JP1ASV¥bin¥adshctmjapp.bat となります。カスタムジョブをインストールした PC のフォルダになります。

実行プログラム：**インストール先フォルダ**¥JP1ASE¥bin¥adshappexec.exe となります。

バージョン：0600。JP1/AJS - View のインターフェースのバージョンです。

クラス名：ADSHAPPEXEC を指定します。

ジョブ種別：JP1/Advanced Shell 用のジョブの場合，必ず PC ジョブを選択します。

4. [OK] ボタンをクリックする。

   カスタムジョブが JP1/AJS - View に登録されます。

JP1/AJS - View 11-00 より古い製品がインストールされている場合で，アプリケーション実行エージェント機能を使用する場合も，カスタムジョブアイコンを次のフォルダにコピーしてください。

```
JP1/AJS - Viewインストール先フォルダ¥image¥custom
```

JP1/AJS で使用するアイコンの説明を次に示します。

表 2-27　JP1/AJS で使用するアイコンの説明

アイコン	名称	ファイル名	説明
	Windows で実行されるジョブアイコン	CUSTOM_PC_ADSHPC.gif	JP1/AJS - View のジョブネットエディタ上で使用する Windows のカスタムジョブのアイコンです。[ジョブネットエディタ] ウィンドウでの [カスタムジョブ] タブで表示されます。

アイコン	名称	ファイル名	説明
	UNIX で実行されるジョブアイコン	CUSTOM_PC_ADSHUX.gif	JP1/AJS - View のジョブネットエディタ上で使用する UNIX のカスタムジョブのアイコンです。［ジョブネットエディタ］ウィンドウでの［カスタムジョブ］タブで表示されます。
	GUI アプリケーション実行ジョブアイコン	CUSTOM_PC_ADSHAPPEXEC.gif	JP1/AJS - View のジョブネットエディタ上で使用する GUI アプリケーション実行ジョブのカスタムジョブのアイコンです。［ジョブネットエディタ］ウィンドウでの［カスタムジョブ］タブで表示されます。
	JP1/Advanced Shell（実行定義）のアイコン	adshctmj.exe	JP1/AJS とカスタムジョブ連携を行う定義プログラムです。
	GUI アプリケーション実行（実行定義）のアイコン	adshctmjapp.exe	GUI アプリケーション実行ジョブとカスタムジョブ連携を行う定義プログラムです。

注意事項

JP1/Advanced Shell がインストールされているマシンが複数あり，各マシンで共通のパスになっていない場合は，JP1/Advanced Shell のインストールパスを変数に設定して，フルパスの代わりに変数名をカスタムジョブ登録の実行プログラムの項目に指定してください。設定方法については，マニュアル「JP1/Automatic Job Management System 3 構築ガイド」の「ジョブ実行時のワークパスを変数として定義する」を参照してください。

指定方法の例を次に示します。

```
jajs_config -k "[JP1_DEFAULT¥JP1NBQAGENT¥Variable]" "jp1asebin"="C:¥Program Files¥Hitachi
¥JP1AS¥JP1ASE¥bin"
```

上記のように JP1/Advanced Shell がインストールされている各マシンの JP1/AJS でパスを変数として設定し，JP1/AJS - View のカスタムジョブ登録の実行プログラムの項目で「$jp1asebin$¥adshexec.exe」のように指定することで，複数のパスを JP1/AJS - View で共通して扱うことができます。

2.7.2　ジョブネットを定義して実行する

JP1/AJS でジョブの実行を自動化する場合，登録したカスタムジョブ，PC ジョブ（Windows の場合），UNIX ジョブ（UNIX の場合），または GUI アプリケーション実行ジョブを JP1/AJS - View でジョブネットに定義し，ジョブネットを実行します。JP1/AJS - View の詳細については，マニュアル「JP1/Automatic Job Management System 3 操作ガイド」のジョブ定義の説明を参照してください。

JP1/AJS3 - View でのジョブネットを定義して実行する手順を次に示します。

1. Windows の［スタート］メニューから，［すべてのプログラム］－［JP1_Automatic Job Management System 3 - View］－［ジョブシステム運用］を選択する。
   ［ログイン］画面が表示されます。
2. ユーザー名，パスワード，および接続ホスト名を指定してログインする。

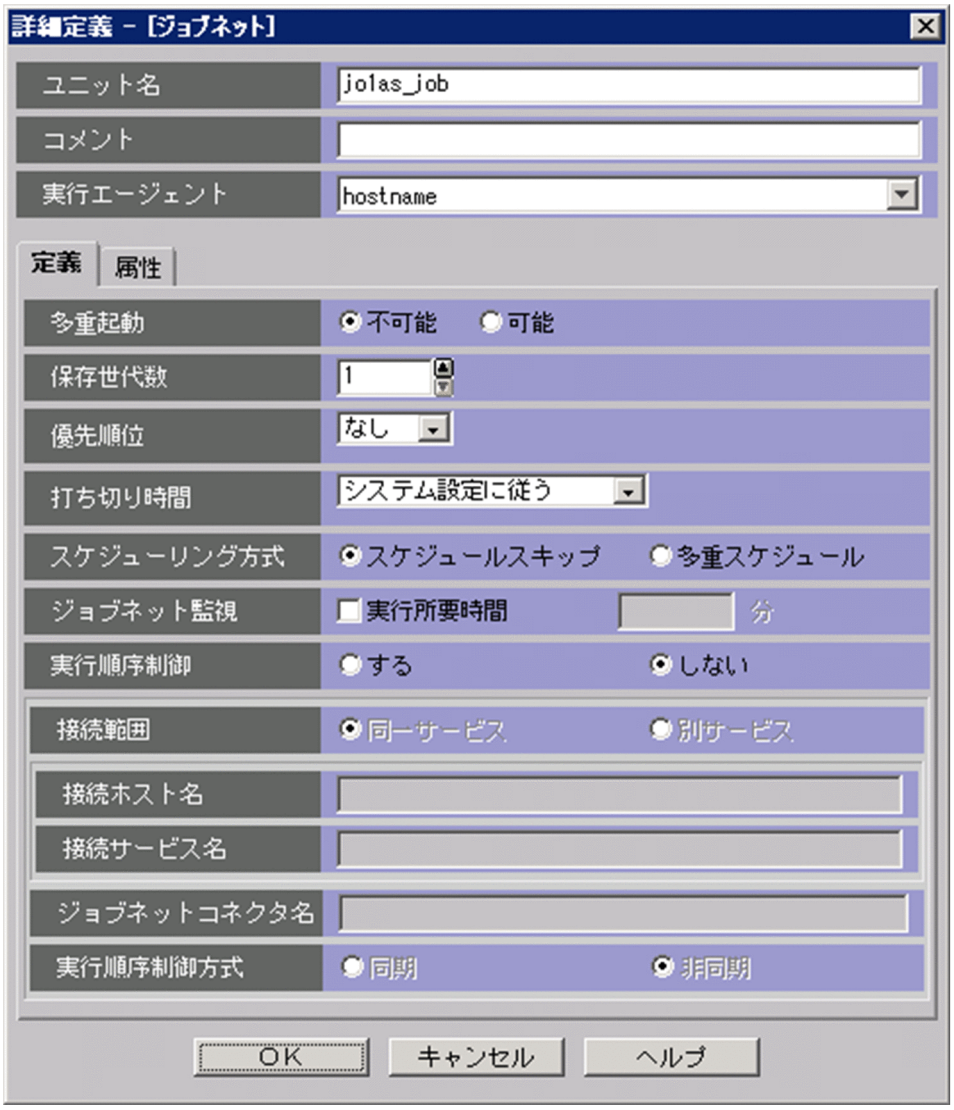
［JP1/AJS3 - View］ウィンドウが表示されます。

3. ［編集］－［新規作成］－［ジョブネット］を選択する。

   ［詳細定義］－［ジョブネット］ダイアログボックスが表示されます。

4. ジョブネットに属性などを定義して，［OK］ボタンをクリックする。

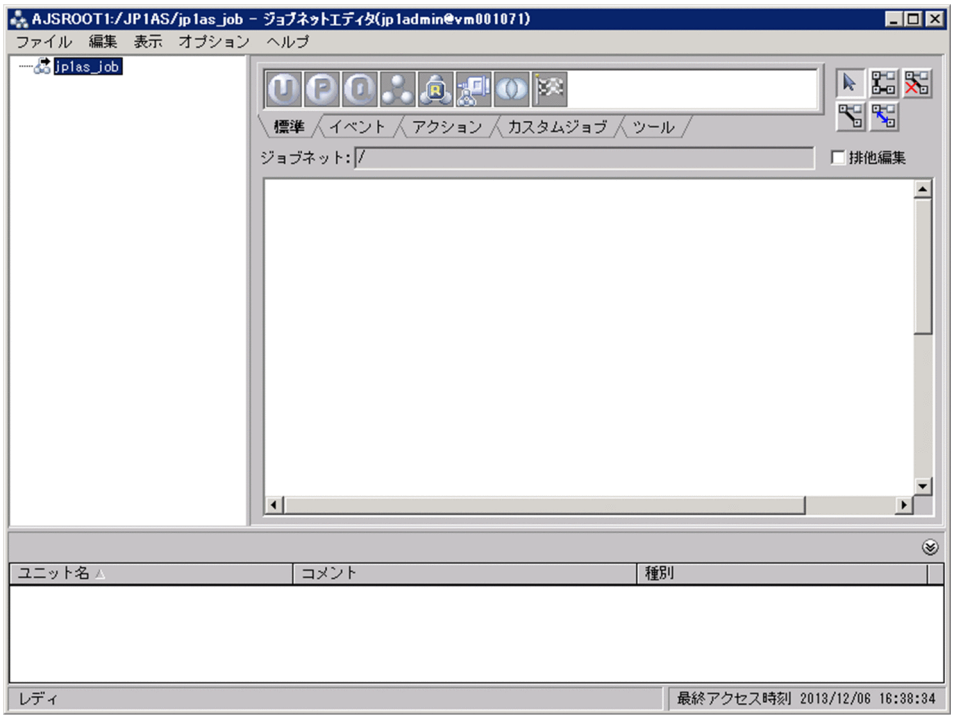
   運用環境に応じて，［実行エージェント］に適切な内容を指定してください。省略することもできます。JP1/AJS の項目については，JP1/AJS のマニュアルを参照してください。



   これによってジョブネットが作成され，リストエリアに表示されます。

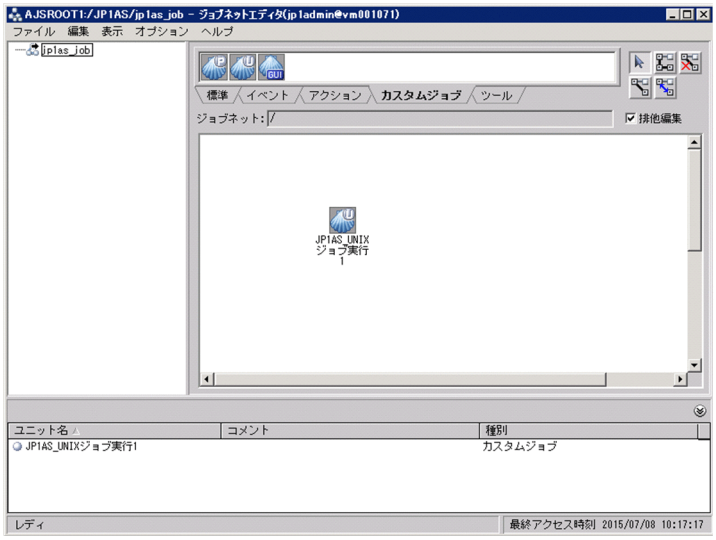
5. 作成したジョブネットをダブルクリックする。

   ［ジョブネットエディタ］ウィンドウが表示されます。



6. ジョブを定義したり関連づけたりするときにほかのユーザーがアクセスできないように，［排他編集］をチェックする。

7. アイコンリストから必要なカスタムジョブ，PC ジョブ，UNIX ジョブ，または GUI アプリケーション実行ジョブのアイコンをマップエリアへドラッグする。



［詳細定義-［Custom Job］］ダイアログボックス，［詳細定義－［PC Job］］ダイアログボックス，または［詳細定義－［UNIX Job］］ダイアログボックスが表示されます。

以降の手順では，カスタムジョブでの定義方法を示します。PC ジョブまたは UNIX ジョブを使用する場合に JP1/Advanced Shell で必要な設定については，「2.7.3　PC ジョブ／UNIX ジョブによるジョブの定義」を参照してください。

8. ［詳細定義-［Custom Job］］ダイアログボックスでジョブに属性などを定義する。

JP1/AJS のマニュアルに従って，［定義］タブを指定した場合と［属性］タブを指定した場合の両方に適切な内容を指定します。

また，運用環境に応じて，［実行エージェント］に適切な内容を指定してください。省略することもできます。

なお，PC ジョブ，UNIX ジョブの場合で次のどちらかが指定されているとき，［定義］タブで［標準出力ファイル名］を指定すると，空のファイルが出力されます。

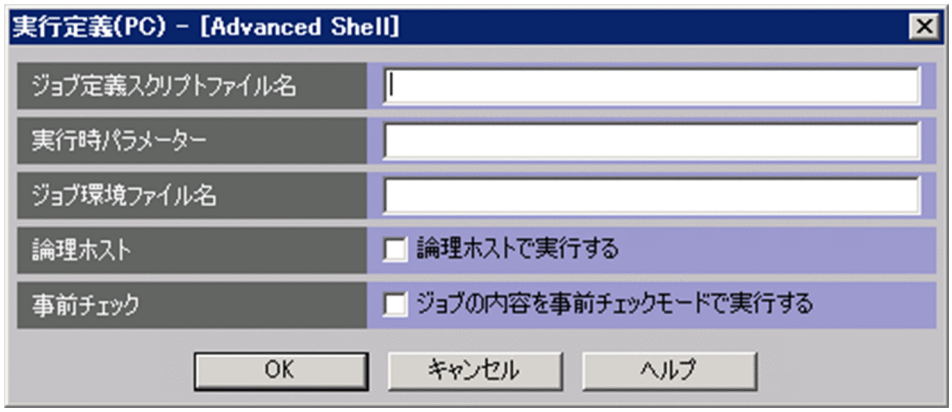
- adshexec コマンドの-s オプション，または環境ファイルの OUTPUT_STDOUT パラメーターに SPOOL を指定（標準出力をスプール内のファイルに出力）
- OUTPUT_MODE_ROOT パラメーターで EXTENDED を指定（拡張出力モードを選択）

標準出力の内容は JP1/Advanced Shell のジョブコントローラで別ファイルに出力していて，JP1/AJS に返す標準出力には何も出力していないためです。

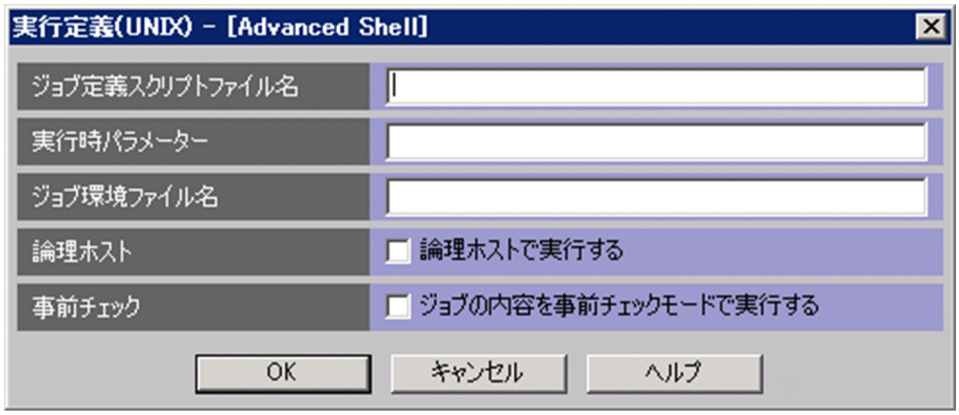
9. ［定義］タブを選択して［詳細］ボタンをクリックする。

カスタムジョブの種類に対応した，［実行定義］ダイアログボックスが表示されます。Windows，UNIX，および GUI アプリケーション実行ジョブの場合の表示を次に示します。

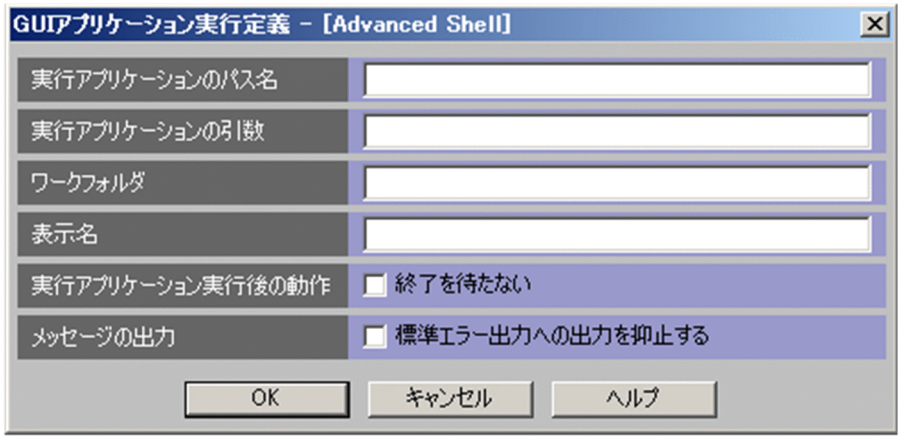
- Windows の場合



- UNIX の場合



- GUI アプリケーション実行ジョブの場合



手順 9.と手順 10.については，「(2)　カスタムジョブによるジョブネットの定義手順に関する補足」に示す補足も参照してください。

10. ［実行定義］ダイアログボックスに JP1/Advanced Shell のジョブコントローラの実行に必要な情報を設定して［OK］ボタンをクリックする。

PC ジョブ，UNIX ジョブの場合

- ジョブ定義スクリプトファイル名 ～<パス名> Windows の場合：((1～247 バイト))，UNIX の場合：((1～1,023 バイト))

  ジョブ定義スクリプトファイル名を指定します。省略できません。

- 実行時パラメーター～< ASCII 文字列>((0～1,022 バイト))

  ジョブ定義スクリプトファイルの実行時に渡すパラメーターを定義します。複数のパラメーターを定義する場合は，スペースで区切って指定します。

- ジョブ環境ファイル名～<パス名> Windows の場合：((0～247 バイト))，UNIX の場合：((0～1,023 バイト))

  ジョブ環境ファイル名を指定します。

  ジョブ環境ファイル名に指定がある場合は JP1/Advanced Shell のジョブコントローラの環境で環境変数 ADSH_ENV が定義されていても，ジョブ環境ファイル名に指定した値を優先して使用して実行します。

  ジョブ環境ファイル名に指定がなく，JP1/Advanced Shell のジョブコントローラの環境で環境変数 ADSH_ENV が定義されている場合は，環境変数 ADSH_ENV の値を使用して実行します。

  ジョブ環境ファイル名に指定がなく，環境変数 ADSH_ENV の指定もない場合，ジョブ環境ファイルの指定はないものとして実行します。ジョブ環境ファイルの指定は省略できます。

  JP1/AJS が起動した JP1/Advanced Shell のジョブコントローラは，使用したジョブ環境ファイルのパスを環境変数 ADSH_ENV に設定してから，ジョブの実行を開始します。別のジョブコントローラやジョブ環境ファイルを読み込むほかのコマンドを子孫プロセスとして起動する場合，それらのジョブコントローラやコマンドは環境変数 ADSH_ENV の値を使用します。したがって，ジョブ実行中に環境変数 ADSH_ENV の値を再設定していない場合，JP1/AJS が起動した JP1/Advanced Shell のジョブコントローラと同一のジョブ環境ファイルを使用します。ジョブ実行中に環境変数 ADSH_ENV の値を再設定している場合，再設定した値のジョブ環境ファイルを使用します。

- 論理ホスト

  論理ホストで実行するかどうかを指定します。チェックボックスをチェックして実行した場合，JP1/AJS で設定された論理ホスト（環境変数 JP1_HOSTNAME の値）で実行します。

- 事前チェック

  ジョブの内容を事前にチェックするかどうかを指定します。事前チェックのチェックボックスにチェックして実行した場合，ジョブ定義スクリプトファイルを実行しないでジョブ定義スクリプトファイルの内容をチェックします。

GUI アプリケーション実行ジョブの場合

- コマンドライン　～< ASCII 文字列> ((1～1,022 バイト))

  実行アプリケーションのファイル名および実行アプリケーションの実行時に指定するパラメーターを指定します。省略できません。

- 実行アプリケーションのパス名 ～< ASCII 文字列> ((1～247 バイト))

  実行アプリケーション名のパス名を指定します。

- 実行アプリケーションの引数 ～< ASCII 文字列> ((0～1,023 バイト))

  実行アプリケーションの引数を指定します。

- ワークフォルダ　～<パス名>((0～247 バイト))

  実行アプリケーションの実行時のワークフォルダを指定します。

ワークフォルダは省略できます。省略した場合は，JP1/AJS-Agent（または JP1/AJS-Manager）が GUI アプリケーション実行プログラムを起動するときの実行時ディレクトリ（ワークディレクトリ）になります。JP1/AJS-Agent（または JP1/AJS-Manager）が JP1/AS のジョブコントローラを起動するときの実行時ディレクトリ（ワークディレクトリ）については，マニュアル「JP1/Automatic Job Management System 3 構築ガイド」を参照してください。

表示名　～＜ ASCII 文字列＞((0～247 バイト)) アプリケーション実行エージェントアイコンを左クリックしたときに表示される表示名を指定します。表示名は省略できます。省略した場合はコマンドラインの内容となります。

- 実行アプリケーション実行後の動作

  実行アプリケーション実行後に終了を待たないかどうかを指定します。実行アプリケーション実行後の動作のチェックボックスで，チェックをしないで実行した場合，実行アプリケーション実行後に終了を待ちます。実行アプリケーション実行後の動作のチェックボックスで，チェックをして実行した場合，実行アプリケーション実行後に終了を待ちません。

- メッセージの出力

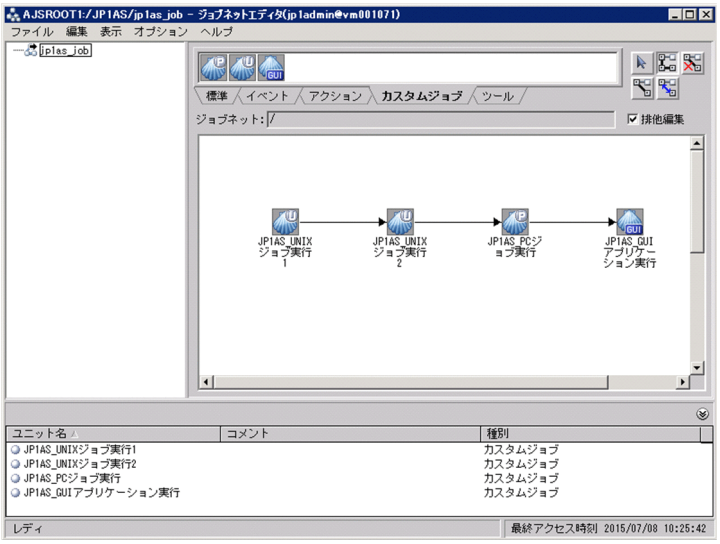
  メッセージの出力を抑止するかどうかを指定します。「標準エラー出力への出力を抑止する」のチェックボックスで，チェックをして実行した場合，標準エラー出力へメッセージの出力を抑止します。

11. ［詳細定義-［Custom Job］］ダイアログボックスに戻ってから［OK］ボタンをクリックする。

    ジョブネットにジョブが定義されます。必要に応じて，同様の手順で，ジョブを定義してください。

12. ジョブ同士を実行順序に従って関連づける。

    ジョブネットが定義されます。JP1/AJS - View でのジョブの定義例を次に示します。



13. JP1/AJS の操作によってジョブネットを実行する。

    JP1/Advanced Shell のジョブコントローラの終了コードをジョブの終了コードとして JP1/AJS に返します。

(1)　ジョブネットの定義内容に関する注意事項

- カバレージの指定

カスタムジョブからカバレージを有効にしたい場合は，環境ファイルにカバレージ採取の一括有効化機能を指定してください。

- **JP1/AJS から JP1/Advanced Shell のジョブコントローラを起動したときの実行時ディレクトリ**

  JP1/AJS から JP1/Advanced Shell のジョブコントローラを起動した場合の実行時ディレクトリは，JP1/AJS - Agent（または JP1/AJS - Manager）が JP1/Advanced Shell のジョブコントローラを起動するときの実行時ディレクトリになります。JP1/AJS - Agent（または JP1/AJS - Manager）が JP1/Advanced Shell のジョブコントローラを起動するときの実行時ディレクトリについては，マニュアル「JP1/Automatic Job Management System 3 構築ガイド」を参照してください。実行時ディレクトリは，JP1/AJS のマニュアルでは，ワークパス（ワークディレクトリ）と表記されています。

- **JP1/AJS から JP1/Advanced Shell のジョブコントローラを起動したときの環境変数**

  JP1/AJS から Windows の JP1/Advanced Shell のジョブコントローラを起動した場合，通常 JP1/AJS サービス起動時にはシステム環境変数の設定を有効にし，ユーザー環境変数は読み込まれません。詳細は JP1/AJS のマニュアルを参照してください。

- **言語種別を英語に設定した海外版の JP1/AJS - Manager と接続して使用する**

  言語種別を英語に設定した海外版の JP1/AJS - Manager と接続してジョブを定義する場合，［実行定義］ダイアログボックスのジョブの定義情報には英数字（ASCII）だけを使用してください。

- **Shift-JIS 以外の文字コード種別を設定した JP1/AJS からジョブコントローラを起動したときの定義情報の入力範囲【UNIX 限定】**

  カスタムジョブの［実行定義］ダイアログボックスのジョブ定義スクリプトファイル名とジョブ環境ファイル名の設定項目では，入力バイト数のチェックを Shift-JIS のバイト数でしています。ただし，実行環境の文字コードが Shift-JIS 以外の場合は，［実行定義］ダイアログボックスに入力できるファイル名であっても，実行環境の文字コードに変換したバイト数が OS が定めるファイル名長の制限を超えていると，ジョブコントローラがエラーとなります。この場合は，ジョブコントローラのエラーメッセージを参照して正しい定義情報を入力するようにしてください。

- **JP1/AJS から GUI アプリケーション実行プログラムを起動したときの環境変数**

  JP1/AJS から GUI アプリケーション実行プログラムを起動した場合，実行ユーザーの環境変数を使用します。

(2)　カスタムジョブによるジョブネットの定義手順に関する補足

「2.7.2　ジョブネットを定義して実行する」の手順 9 と手順 10 の実行定義の設定について，JP1/AJS の ajsdefine コマンドのユニット定義，および JP1/AJS - Definition Assistant のジョブの定義で定義することもできます。

ユニット定義の詳細については，マニュアル「JP1/Automatic Job Management System 3 コマンドリファレンス」を参照してください。

JP1/AJS - Definition Assistant については，マニュアル「JP1/Automatic Job Management System 3 - Definition Assistant」を参照してください。

カスタムジョブを使用する場合の，ユニット定義および JP1/AJS - Definition Assistant で記述する環境変数とパラメーターの詳細を次に記載します。

(a) PC ジョブ，UNIX ジョブの場合

- JP1/Advanced Shell の［実行定義］ダイアログボックスで設定する環境変数
  ADSH_AJS_SCRF：ジョブ定義スクリプトファイル名
  ADSH_AJS_ENVF：ジョブ環境ファイル名
  ADSH_AJS_LHOST：論理ホスト
  ADSH_AJS_GCHE：事前チェック
- JP1/Advanced Shell のジョブコントローラに渡す環境変数
  ユニット定義ファイルの env パラメーターとして環境変数を定義します。
  JP1/Advanced Shell のジョブコントローラに渡す環境変数の設定項目を次の表に示します。

表 2-28　JP1/Advanced Shell のジョブコントローラに渡す環境変数の設定項目

項目名	説明
入力範囲	環境変数の値として指定できる文字列（"="の後ろの文字列）のバイト数（Shift-JIS）
設定値	定義できる文字の種類 • 文字列：制御文字（0x00～0x1f，0x7f）以外の文字 • 記号名称：半角英数字および「@」「#」「_」 • 数字
初期値	新規作成ジョブとしてカスタムジョブを起動したときに読み込む値
省略	値を省略できないものは，JP1/Advanced Shell のジョブコントローラでエラーとなります。

JP1/Advanced Shell のジョブコントローラに渡す環境変数の設定項目の入力範囲と設定値を次の表に示します。

表 2-29　JP1/Advanced Shell のジョブコントローラに渡す環境変数の設定項目の入力範囲と設定値

環境変数	入力範囲	設定値	初期値	省略
ADSH_AJS_SCRF	PC ジョブ：1～247 バイト UNIX ジョブ：1～1,023 バイト	文字列	空文字列	できない
ADSH_AJS_ENVF	PC ジョブ：0～247 バイト UNIX ジョブ：0～1,023 バイト	文字列	空文字列	できる
ADSH_AJS_LHOST	0～2 バイト	• チェックありの場合 -h • チェックなしの場合 空文字列	空文字列	できる
ADSH_AJS_GCHE	0～2 バイト	• チェックありの場合 -c	空文字列	できる

環境変数	入力範囲	設定値	初期値	省略
ADSH_AJS_GCHE	0～2 バイト	• チェックなしの場合 空文字列	空文字列	できる
AJS_BJEX_STOP	4 バイト	"TERM"	"TERM"	できない

- JP1/Advanced Shell のジョブコントローラに渡すパラメーターの詳細

  フィールド「実行時パラメーター」はユニット定義ファイルの prm パラメーターとして定義し，JP1/Advanced Shell のジョブコントローラに対するパラメーターとして渡します。指定できる値は次のとおりです。

  入力範囲：1～1,023 バイト※

  設定値：文字列

  初期値：空文字列

  省略：できる

  実行時パラメーターでは，［実行定義］ダイアログボックスの指定の空文字列と同等の定義をする場合は，ユニット定義ファイルの prm パラメーターにスペース 1 バイト分を定義してください。

  注※

  prm パラメーターに 1,023 バイトまで入力できるのは，スペースだけの文字列だけです。スペース以外の文字列を入力する場合は 1,022 バイトまでで指定してください。prm パラメーターにスペースだけの文字列を指定した場合，［実行定義］ダイアログボックスの［実行時パラメーター］のテキストボックスにスペースを 1 バイト分削除して表示します。

重要

ユニット定義ファイルで定義情報を削除する場合は，次の点に注意してください。

・env パラメーターは環境変数の値（「=」の後ろの文字列）を削除するか，env パラメーターの指定そのものを削除してください。

・prm パラメーターは prm の値に 1 バイト分のスペースを指定するか，prm パラメーターの指定を削除してください。

ユニット定義の例を次に示します。

```
unit=ユニット名,,実行ユーザー,;
{
        ty=cpj;
        cty="ADSHUX";
        sc="/opt/jp1as/bin/adshexec";
        env="AJS_BJEX_STOP=TERM";                          →1.
        env="ADSH_AJS_SCRF=/tmp/JP1AS/scr/samplescrfile.ash";  →2.
        prm="param1 param2";                               →3.
        env="ADSH_AJS_ENVF=/tmp/JP1AS/env/sampleenvfile";  →4.
        env="ADSH_AJS_LHOST=-h";                           →5.
        env="ADSH_AJS_GCHE=-c";                            →6.
```

```
        tho=0;
}
```

例の右側に付記した番号に対応する説明を次に示します。

1. AJS_BJEX_STOP：AJS_BJEX_STOP 環境変数はシステムで使用するため，設定値に TERM を指定して必ず定義してください。
2. ADSH_AJS_SCRF：ジョブ定義スクリプトファイル名を指定します。
   ADSH_AJS_SCRF 環境変数は必ず定義してください。
3. ユニット定義ファイルの prm パラメーターを指定します。
4. ADSH_AJS_ENVF：ジョブ環境ファイル名を指定します。
5. ADSH_AJS_LHOST：論理ホストで実行する場合は-h を指定し，実施しない場合は何も指定しません。
6. ADSH_AJS_GCHE：事前チェックを実施する場合は-c を指定し，実施しない場合は何も指定しません。

ユニット定義ファイルを作成したあと，JP1/AJS の ajsdefine コマンドおよび JP1/AJS3 - Definition Assistant でジョブを定義できます。

(b) GUI アプリケーション実行ジョブの場合

- JP1/Advanced Shell の［実行定義］ダイアログボックスで設定する環境変数
  ADSH_AJS_APPNAME：実行アプリケーションのパス名
  ADSH_AJS_APPARG：実行アプリケーションの引数
  ADSH_AJS_WORKF：ワークフォルダ
  ADSH_AJS_SHOWN：表示名
  ADSH_AJS_AFEXECMV：実行アプリケーション実行後の動作
  ADSH_AJS_MESOUT：メッセージの出力
- GUI アプリケーション実行プログラムに渡す環境変数
  ユニット定義ファイルの env パラメーターとして環境変数を定義します。
  GUI アプリケーション実行プログラムに渡す環境変数の設定項目を次の表に示します。

表 2-30　JP1/Advanced Shell のジョブコントローラに渡す環境変数の設定項目

項目名	説明
入力範囲	環境変数の値として指定できる文字列（"="の後ろの文字列）のバイト数（Shift-JIS）
設定値	定義できる文字の種類 • 文字列：制御文字（0x00～0x1f，0x7f）以外の文字 • 記号名称：半角英数字および「@」「#」「_」 • 数字
初期値	新規作成ジョブとしてカスタムジョブを起動したときに読み込む値
省略	値を省略できないものは，GUI アプリケーション実行プログラムでエラーとなります。

GUI アプリケーション実行プログラムに渡す環境変数の設定項目の入力範囲と設定値を次の表に示します。

表 2-31　GUI アプリケーション実行プログラムに渡す環境変数の設定項目の入力範囲と設定値

環境変数	入力範囲	設定値	初期値	省略
ADSH_AJS_APPNAME	1～247 バイト	文字列	空文字列	できない
ADSH_AJS_APPARG	0～1,023 バイト	文字列	空文字列	できる
ADSH_AJS_WORKF	0～247 バイト	文字列	空文字列	できる
ADSH_AJS_SHOWN	0～247 バイト	文字列	空文字列	できる
ADSH_AJS_AFEXECMV	0～2 バイト	• チェックありの場合 -n • チェックなしの場合 空文字列	空文字列	できる
ADSH_AJS_MESOUT	0～2 バイト	• チェックありの場合 -m • チェックなしの場合 空文字列	空文字列	できる
ADSH_AJS_APPEXEC	7 バイト	" APPEXEC "	" APPEXEC "	できない

重要

ユニット定義ファイルで定義情報を削除する場合は，env パラメーターは環境変数の値（「=」の後ろの文字列）を削除するか，env パラメーターの指定そのものを削除してください。

ユニット定義の例を次に示します。

```
unit=ユニット名,,実行ユーザー,;
{
        ty=cpj;
        cty="ADSHAPPEXEC";
        sc=" C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin¥adshappexec.exe";
        env="ADSH_AJS_APPEXEC=APPEXEC";                          →1.
        env="ADSH_AJS_APPNAME=C:¥ExecApp.exe";                   →2.
        env="ADSH_AJS_APPARG=param1 param2";                     →3.
        env="ADSH_AJS_WORKF=C:¥work-folder";                     →4.
        env="ADSH_AJS_SHOWN=ExecApplication";                    →5.
        env="ADSH_AJS_AFEXECMV=-n";                              →6.
        env="ADSH_AJS_MESOUT=-m";                                →7.
        tho=0;
}
```

例の右側に付記した番号に対応する説明を次に示します。

1. ADSH_AJS_APPEXEC 環境変数はシステムで使用するため，設定値に APPEXEC を指定して必ず定義してください。
2. ADSH_AJS_APPNAME：実行アプリケーションのパス名を指定します。
   ADSH_AJS_APPNAME は必ず定義してください。
3. ADSH_AJS_APPARG：実行アプリケーションの引数を指定します。
4. ADSH_AJS_WORKF：ワークフォルダを指定します。
5. ADSH_AJS_SHOWN：表示名を指定します。
6. ADSH_AJS_AFEXECMV：実行アプリケーション実行後の動作で終了を待たない場合は-n を指定し，実施しない場合は何も指定しません。
7. ADSH_AJS_MESOUT：標準エラー出力への出力を抑止する場合は-m を指定し，実施しない場合は何も指定しません。

ユニット定義ファイルを作成したあと，JP1/AJS の ajsdefine コマンドおよび JP1/AJS3 - Definition Assistant でジョブを定義できます。

2.7.3　PC ジョブ／UNIX ジョブによるジョブの定義

(1)　PC ジョブによるジョブの定義

PC ジョブで JP1/Advanced Shell のバッチジョブを定義する場合に必要な項目について示します。

(a)　バッチジョブを定義する場合

- 実行ファイル名
  ［詳細定義－［PC Job］］ダイアログボックスの［定義］タブの「実行ファイル名」，またはユニット定義ファイルの「sc="スクリプトファイル名"」に，adshexec コマンドのパスを指定します。

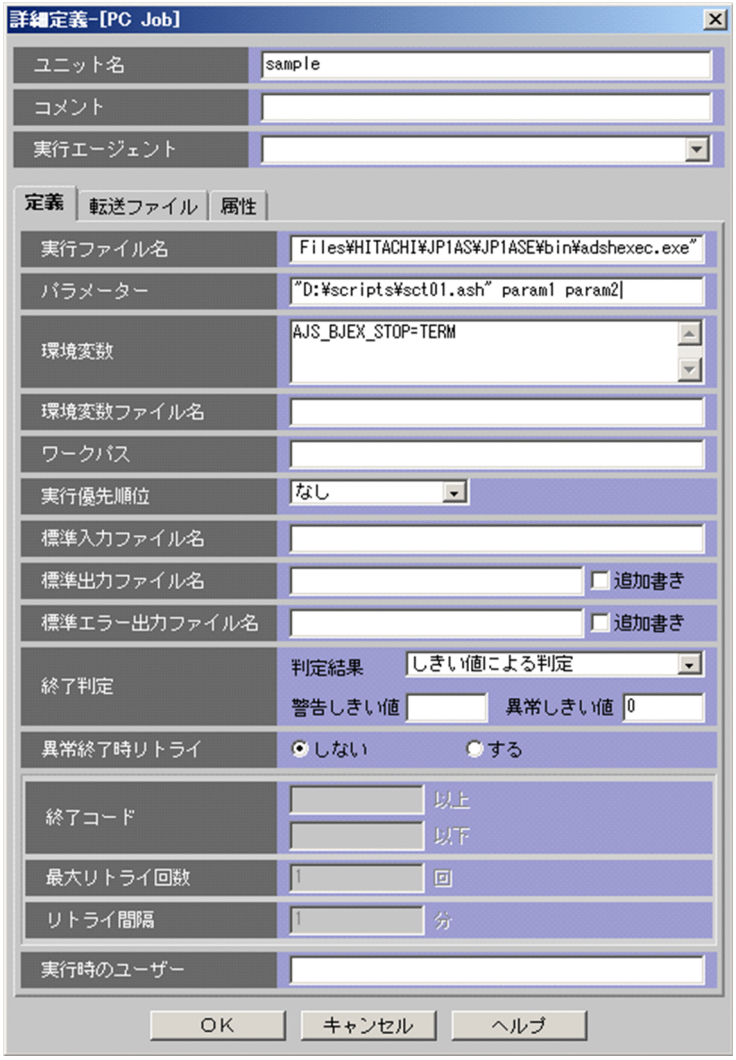
```
”インストール先フォルダ¥JP1ASE¥bin¥adshexec.exe”
```

- パラメーター
  ［詳細定義－［PC Job］］ダイアログボックスの［定義］タブの「パラメーター」，またはユニット定義ファイルの「prm="パラメーター"」に，adshexec コマンドのオプション，ジョブ定義スクリプトファイル名，および実行時パラメーターを指定します。
- 環境変数
  ［詳細定義－［PC Job］］ダイアログボックスの［定義］タブの「環境変数」，またはユニット定義ファイルの「env="環境変数"」に，次の内容を指定します。

```
AJS_BJEX_STOP=TERM
```

JP1/Advanced Shell のバッチジョブとしての指定例を次の図に示します。

図 2-7　［詳細定義－［PC Job]］ダイアログボックスの［定義］タブの指定例



(b)　GUI アプリケーション実行ジョブを定義する場合

PC ジョブで JP1/Advanced Shell の GUI アプリケーション実行ジョブを定義する場合に必要な項目について示します。

- 実行ファイル名

  ［詳細定義－［PC Job]］ダイアログボックスの［定義］タブの「実行ファイル名」，またはユニット定義ファイルの「sc="スクリプトファイル名"」に，adshappexec コマンドのパスを指定します。

  ```
  ”インストール先フォルダ¥JP1ASE¥bin¥adshappexec.exe”
  ```

- パラメーター

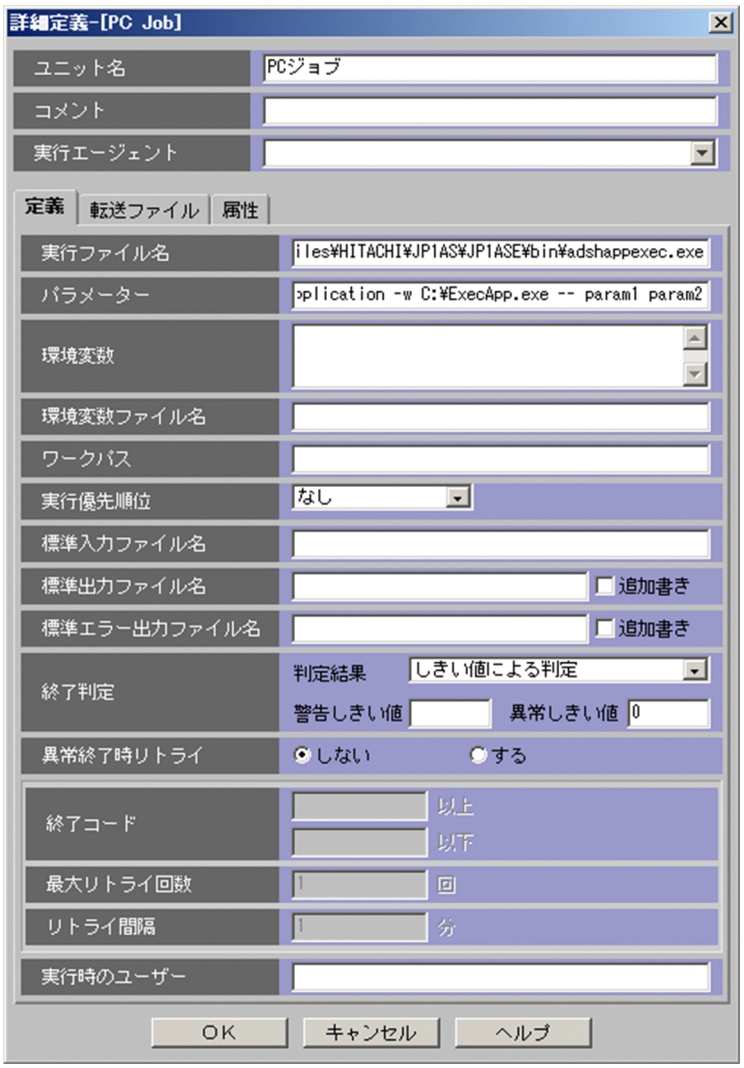
  ［詳細定義－［PC Job]］ダイアログボックスの［定義］タブの「パラメーター」，またはユニット定義ファイルの「env="パラメーター"」に，adshappexec コマンドのパラメーターを指定します。

- 環境変数，環境変数ファイル名

  指定しても環境変数，および環境変数ファイル名は無視します。

JP1/Advanced Shell のバッチジョブとしての指定例を次の図に示します。

図 2-8 ［詳細定義－［PC Job]］ダイアログボックスの［定義］タブの指定例



(2) UNIX ジョブによるジョブの定義

UNIX ジョブで JP1/Advanced Shell のバッチジョブを定義する場合に必要な項目について示します。

- スクリプトファイル名

  ［詳細定義-［UNIX Job]］ダイアログボックスの［定義］タブの「スクリプトファイル名」，またはユニット定義ファイルの「sc="スクリプトファイル名"」に，adshexec コマンドのパスを指定します。

  ```
  /opt/jp1as/bin/adshexec
  ```

  また，1 行目に#!で開始する adshexec コマンドのパス（例：#!/opt/jp1as/bin/adshexec）を記述し，実行権限を付与したジョブ定義スクリプトファイルのパスを指定することもできます。

  ```
  ジョブ定義スクリプトファイルのパス
  ```

- コマンド文

［詳細定義－［UNIX Job］］ダイアログボックスの［定義］タブの「コマンド文」，またはユニット定義ファイルの「te="コマンドテキスト"」に，コマンドテキストの一部として「スクリプトファイル名」と同様の方法で adshexec コマンドのパス，またはジョブ定義スクリプトファイルのパスを指定できます。コマンド文で定義したジョブを JP1/AJS で強制終了した場合，次の制限があります。

- 強制終了の操作をするタイミングによっては，JP1/AJS-View からジョブ実行ログやジョブの標準エラー出力の内容を参照できないことがあります。この場合，ジョブ実行ログについては，スプールジョブディレクトリを参照することで内容を確認できます。
- ジョブ実行ログに出力されるジョブの終了コードは 143 ですが，JP1/AJS - View から参照できるジョブの終了コードは-1 になります。

- パラメーター

  ［詳細定義-［UNIX Job］］ダイアログボックスの［定義］タブの「パラメーター」，またはユニット定義ファイルの「prm="パラメーター"」に，adshexec コマンドのオプション，ジョブ定義スクリプトファイル名，および実行時パラメーターを指定します。

  スクリプトファイル名にジョブ定義スクリプトファイル名を指定した場合は，実行時パラメーターだけを指定します。

- 環境変数

  ［詳細定義-［UNIX Job］］ダイアログボックスの［定義］タブの「環境変数」，またはユニット定義ファイルの「env="環境変数"」に，次の内容を指定します。

```
AJS_BJEX_STOP=TERM
```

JP1/Advanced Shell のバッチジョブとしての指定例を次の図に示します。この例では，「スクリプトファイル名」に adshexec コマンド，およびジョブ定義スクリプトファイルパスを指定しています。

図 2-9　［詳細定義-［UNIX Job]］ダイアログボックスの［定義］タブの指定例（adshexec コマンドを指定する場合）

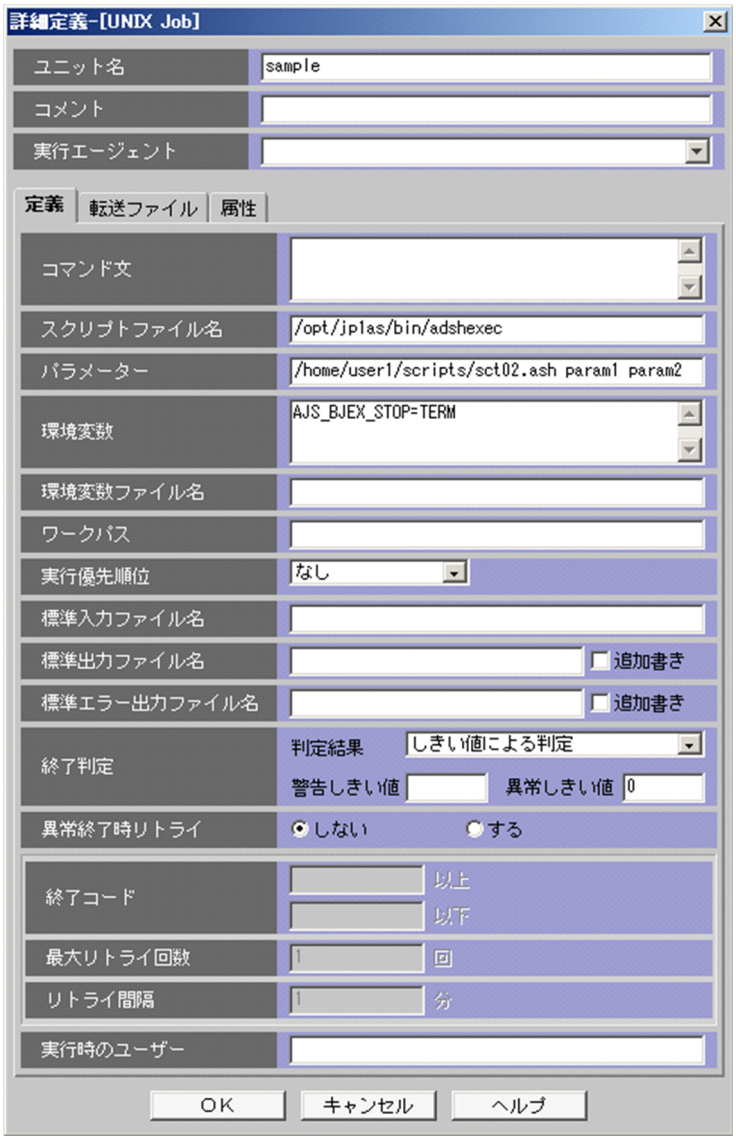
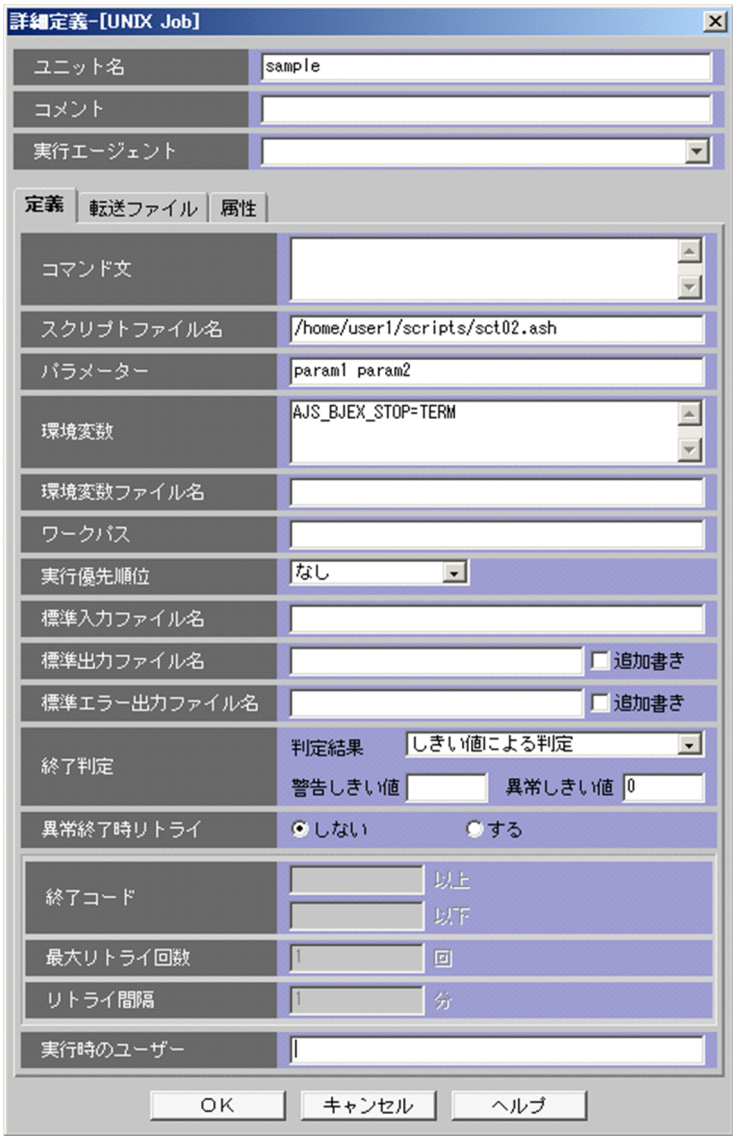


図 2-10　［詳細定義-［UNIX Job]］ダイアログボックスの［定義］タブの指定例（ジョブ定義スクリプトファイルパスを指定する場合）



2.8　ユーザー応答機能を設定する

2.8.1　ユーザー応答機能を使用するための環境ファイルの設定

JP1/Advanced Shell は，ジョブ間で動作環境を統一したい場合に設定するシステム環境ファイルと，ジョブごとに指定することができるジョブ環境ファイルの 2 種類で動作環境を指定できます。ユーザー応答機能を使用する場合は，システム環境ファイルをシステムの環境に合わせて変更してください。

システム環境ファイルの変更後は，デーモンまたはサービスを再起動する必要があります。

(1)　スプールルートディレクトリの指定

ジョブの実行結果の出力先となるスプールルートディレクトリは，SPOOL_DIR パラメーターで定義します。ユーザー応答機能を使用する場合は，SPOOL_DIR パラメーターはシステム環境ファイルだけに定義してください。パス名にマルチバイト文字は使用できません。

SPOOL_DIR パラメーターの詳細については，「7.　環境ファイルで設定するパラメーター」の「SPOOL_DIR パラメーター（スプールルートディレクトリのパス名を定義する）」を参照してください。

(2)　JP1 イベントの送信先ホストの指定

JP1 イベントの送信先である JP1/IM - Manager が稼働している運用管理サーバを，システム環境ファイルの HOSTNAME_JP1IM_MANAGER パラメーターで設定する必要があります。このパラメーターを設定しない場合，JP1/Advanced Shell が稼働しているサーバ上で hostname コマンドを実行したときに表示されるホスト名が，JP1 イベントの送信先として仮定されます。HOSTNAME_JP1IM_MANAGER パラメーターはシステム環境ファイルだけに定義してください。また，このパラメーターで指定したホスト上で，JP1/Advanced Shell が動作するホストのホスト名が名前解決できることを確認してください。

HOSTNAME_JP1IM_MANAGER パラメーターの詳細については，「7.　環境ファイルで設定するパラメーター」の「HOSTNAME_JP1IM_MANAGER パラメーター（JP1 イベントの送信先である JP1/IM - Manager が稼働している運用管理サーバを指定する）」を参照してください。

(3)　JP1 イベントの流量制御の指定

ユーザー応答機能を使用する場合，次に示す JP1 イベントの出力数の条件を満たす必要があります。この条件は，同一の JP1/IM が受信する，すべての JP1 イベントが対象となります。

項番	説明	出力数の条件
1	JP1 イベント	1 秒当たりの出力数が 2 件未満であること
2	応答待ちイベント	1 分当たりの出力数が 1 件未満であること

JP1/Advanced Shell のジョブコントローラでは，USERREPLY_JP1EVENT_INTERVAL パラメーターを使用して，adshecho コマンドまたは adshread コマンドによる JP1 イベントの最小発行間隔を指定できます。複数のジョブからイベントが発行される場合は，すべてのジョブからの，adshecho コマンドや adshread コマンドによる JP1 イベントの発行頻度が上記の条件を満たすように，パラメーターの値を設定してください。

USERREPLY_JP1EVENT_INTERVAL パラメーターの詳細については，「7. 環境ファイルで設定するパラメーター」の「USERREPLY_JP1EVENT_INTERVAL パラメーター（JP1 イベントの最小発行間隔を指定する）」を参照してください。

(4) 応答要求メッセージの最大同時出力数の指定

USERREPLY_WAIT_MAXCOUNT パラメーターを使用することで，ユーザー応答機能で物理ホストまたは論理ホストごとに，応答要求メッセージの最大同時出力数を指定できます。このパラメーターは，同時に adshread コマンドを実行するジョブ数以上になるように指定してください。

USERREPLY_WAIT_MAXCOUNT パラメーターの詳細については，「7. 環境ファイルで設定するパラメーター」の「USERREPLY_WAIT_MAXCOUNT パラメーター（物理ホストまたは論理ホストごとに応答要求メッセージの最大同時出力数を指定する）」を参照してください。

2.8.2 JP1/Advanced Shell をインストールしたあとのユーザー応答機能の設定【Windows 限定】

ユーザー応答機能を使用する場合の，JP1/Advanced Shell のインストール後の設定について説明します。管理者権限を持つユーザーが実行してください。

(1) サービスの設定

(a) JP1/Advanced Shell のサービスの起動方法を設定する

JP1/Advanced Shell のサービスを自動起動する手順を次に示します。

1. Windows の［コントロールパネル］－［管理ツール］－［サービス］を開く。
2. 表示されたサービス名の一覧から次の名前のサービスのプロパティを開く。
   - 実行環境からユーザー応答機能を使用する場合は，［AdshmSvcE］から始まるサービス名
   - 開発環境からユーザー応答機能を使用する場合は，［AdshmSvcD］から始まるサービス名
3. ［全般］タグの「スタートアップの種類」を次のように操作する。

   JP1/Advanced Shell のインストール後の初期状態は，「手動」が設定されています。Windows の起動時に自動でサービスを開始したい場合は，「自動」に変更します。

(b)　JP1/Advanced Shell のサービスを起動する

JP1/Advanced Shell のサービスを手動で起動する手順を次に示します。「(a)　JP1/Advanced Shell のサービスの起動方法を設定する」で「スタートアップの種類」を「自動」に設定して Windows を起動した場合は，この操作は必要ありません。

1. Windows の［コントロールパネル］－［管理ツール］－［サービス］を開く。
2. 表示されたサービス名の一覧から次の名前のサービスのプロパティを開く。
   - 実行環境からユーザー応答機能を使用する場合は，［AdshmSvcE］から始まるサービス名
   - 開発環境からユーザー応答機能を使用する場合は，［AdshmSvcD］から始まるサービス名
3. ［全般］タグの「開始」ボタンをクリックする。

サービスが開始されなかった場合は，イベントログに出力されているエラー情報を確認してください。

エラー情報に KNAX7552-E メッセージが出力された場合は，システム環境ファイルを見直し，サービスを起動し直してください。

(c)　サービスを登録する

インストール時に自動登録されるサービス（AdshmSvcD および AdshmSvcE）が削除された場合に，ユーザー応答機能を使用する場合は，サービスを再登録する必要があります。サービスの登録は adshmsvcd コマンドおよび adshmsvce コマンドで実行できます。

サービスの登録方法を次に示します。

- サービスの再登録方法

  ［サービス］管理ツールにサービス名［AdshmSvcD］および［AdshmSvcE］が表示されない場合は，サービスの登録が削除されているおそれがあります。次のコマンドを実行することで，サービスを再登録できます。

  - サービス AdshmSvcD を登録する場合

    adshmsvcd -install
  - サービス AdshmSvcE を登録する場合

    adshmsvce -install

  コマンドが正常に終了すると，登録したサービスが［サービス］管理ツールに表示されます。

登録されたサービスを起動する手順については，「(b)　JP1/Advanced Shell のサービスを起動する」を参照してください。

(2)　アダプタコマンドの設定（実行環境の場合）

実行環境でユーザー応答機能を使用するためには，JP1/Base に対して，次に示すアダプタコマンドの設定が必要です。この設定は，JP1/Advanced Shell のインストール後に一度だけ実施してください。ただし，JP1/Base を再インストールした場合は，この操作を再度実行してください。

1. ユーザー応答機能用アダプタコマンド設定ファイルの「cmdpath」に指定されているパスを確認し，JP1/Advanced Shell のインストールフォルダと異なる場合は，インストールフォルダへ修正する。

   ユーザー応答機能用アダプタコマンド設定ファイルは次の場所にあります。

   ```
   インストール先フォルダ¥JP1ASE¥sample¥Adapter_HITACHI_JP1_AS_ASE_USERREPLY.conf
   ```

   JP1/Advanced Shell インストール時のユーザー応答機能用アダプタコマンド設定ファイルの内容を次に示します。

   64 ビット版の Windows の場合

   ```
   fileversion 07000000
   upperpp /HITACHI/JP1/IM/CC
   componenttype JP1_AS_ASE_USERREPLY
   cmdpath C:¥Program Files (x86)¥Hitachi¥JP1AS¥JP1ASE¥bin¥adshuserreply.exe
   ```

   64 ビット版の Windows 以外の場合

   ```
   fileversion 07000000
   upperpp /HITACHI/JP1/IM/CC
   componenttype JP1_AS_ASE_USERREPLY
   cmdpath C:¥Program Files¥Hitachi¥JP1AS¥JP1ASE¥bin¥adshuserreply.exe
   ```

2. JP1/Base のインストール先に，手順 1 のユーザー応答機能用アダプタコマンド設定ファイルをコピーする。

   コピー先のフォルダは次のとおりです。

   ```
   JP1/Baseのインストール先フォルダ¥plugin¥conf
   ```

これによって，JP1/IM - View に表示される応答待ちイベントから，応答が入力できるようになります。

(3) アダプタコマンドの設定（開発環境の場合）

JP1/Advanced Shell の開発環境でユーザー応答機能を使用するためには，次に示す設定が必要です。ただし，ユーザー応答機能の出力先に標準出力を指定する場合は，この設定は不要です。

この設定は，JP1/Advanced Shell のインストール後に一度だけ実施してください。ただし，JP1/Base を再インストールした場合は，この操作を再度実行してください。

1. ユーザー応答機能用アダプタコマンド設定ファイルの「cmdpath」に指定されているパスを確認し，JP1/Advanced Shell のインストールフォルダと異なる場合は，インストールフォルダへ修正する。

   ユーザー応答機能用アダプタコマンド設定ファイルは次の場所にあります。

   ```
   インストール先フォルダ¥JP1ASD¥sample¥Adapter_HITACHI_JP1_AS_ASD_USERREPLY.conf
   ```

   JP1/Advanced Shell インストール時のユーザー応答機能用アダプタコマンド設定ファイルの内容を次に示します。

   64 ビット版の Windows の場合

   ```
   fileversion 07000000
   upperpp /HITACHI/JP1/IM/CC
   ```

```
componenttype JP1_AS_ASD_USERREPLY
cmdpath C:¥Program Files (x86)¥Hitachi¥JP1AS¥JP1ASD¥bin¥adshuserreply.exe
```

64 ビット版の Windows 以外の場合

```
fileversion 07000000
upperpp /HITACHI/JP1/IM/CC
componenttype JP1_AS_ASD_USERREPLY
cmdpath C:¥Program Files¥Hitachi¥JP1AS¥JP1ASD¥bin¥adshuserreply.exe
```

2. JP1/Base のインストール先に，手順 1 のファイルをコピーする。

   JP1/Base のコピー先のフォルダは次のとおりです。

```
JP1/Baseのインストール先フォルダ¥plugin¥conf
```

これによって，JP1/IM - View に表示される応答待ちイベントから，応答が入力できるようになります。

2.8.3 JP1/Advanced Shell をインストールしたあとのユーザー応答機能の設定【UNIX 限定】

ユーザー応答機能を使用する場合の，JP1/Advanced Shell のインストール後の設定について説明します。

なお，JP1/Advanced Shell の設定後に，JP1/IM - Manager で応答待ちイベントに関する機能を有効にする必要があります。詳細については，「2.8.4 JP1/IM - Manager で環境情報を設定する」を参照してください。

(1) ユーザー応答機能管理デーモンの自動起動と自動停止

システムの起動時および終了時に，ユーザー応答機能管理デーモンを自動起動・自動停止させるための設定方法を次に示します。

(a) AIX の場合

- システム起動時の自動起動機能の設定

  システム起動時にユーザー応答機能管理デーモンを自動起動させるには，mkitab コマンドで設定します。設定内容は次のシステム起動時から有効になります。

  mkitab コマンドの設定例を次に示します。

```
mkitab "adshmd:2:wait:/opt/jp1as/sbin/adshmdctl start"
```

  ユーザー応答機能管理デーモンの起動は，連携する JP1 シリーズ製品のサービスの起動よりあとになるよう設定してください。例えば，JP1/Base，JP1/IM - Manager，JP1/Advanced Shell の順に自動起動させるには，次のように指定して mkitab コマンドを実行します。

```
mkitab -i hntr2mon "jp1base:2:wait:/etc/opt/jp1base/jbs_start"
mkitab -i jp1base "jp1cons:2:wait:/etc/opt/jp1cons/jco_start"
mkitab -i jp1cons "adshmd:2:wait:/opt/jp1as/sbin/adshmdctl start"
```

設定後は lsitab コマンドを次のように実行して，設定内容を確認してください。

```
lsitab -a
```

コマンド実行後の出力例を次に示します。

```
init:2:initdefault:
brc::sysinit:/sbin/rc.boot 3 >/dev/console 2>&1 # Phase 3 of
system boot
    :
hntr2mon:2:once:/opt/hitachi/HNTRLib2/etc/D002start
jp1base:2:wait:/etc/opt/jp1base/jbs_start
jp1cons:2:wait:/etc/opt/jp1cons/jco_start
adshmd:2:wait:/opt/jp1as/sbin/adshmdctl start
```

- システム終了時の自動停止機能の設定

  システム終了時にユーザー応答機能管理デーモンを自動停止させるには，/etc/rc.shutdown を編集して，連携する JP1 シリーズ製品のサービスより先に停止させるように次に示す記述を追加してください。

```
test -x /opt/jp1as/sbin/adshmdctl && /opt/jp1as/sbin/adshmdctl stop
    :
連携するJP1シリーズ製品のサービスの停止処理
    :
```

(b) RHEL 6，Oracle Linux 6，CentOS 6 の場合

JP1/Advanced Shell では，ユーザー応答機能管理デーモンを自動起動・停止するためのスクリプトファイル jp1_as_md を/opt/jp1as/sample ディレクトリに格納しています。このスクリプトファイルを利用して次の手順で設定します。

- /etc/rc.d/init.d ディレクトリへの追加

  /opt/jp1as/sample ディレクトリに格納されている jp1_as_md を/etc/rc.d/init.d に追加します。jp1_as_md を追加するには次のように指定します。

```
cp /opt/jp1as/sample/jp1_as_md /etc/rc.d/init.d
chmod u=rwx,go=rx /etc/rc.d/init.d/jp1_as_md
chown root:root /etc/rc.d/init.d/jp1_as_md
```

- 自動起動のためのシンボリックリンク作成

  /etc/rc.d/rc**N**.d ディレクトリ（**N** は起動時の実行レベル）に/etc/rc.d/init.d/jp1_as_md へのシンボリックリンクを作成します。このとき，連携する JP1 シリーズ製品のサービスが起動したあとに起動されるように，シンボリックリンクの名称を付ける必要があります。

  例えば，/etc/rc.d/rc3.d ディレクトリ，/etc/rc.d/rc5.d ディレクトリに提供されている JP1/Base の自動起動スクリプトのシンボリックリンクの名称が「S99_JP1_10_BASE」，JP1/IM - Manager の自動起動スクリプトのシンボリックリンクの名称が「S99_JP1_20_CONS」の場合，文字列の大小関係で「S99_JP1_20_CONS」よりも大きくなるようにシンボリックリンクの名称を「S99_JP1_70_AS」とし，次のように指定することで JP1/Base，JP1/IM - Manager よりあとに起動されます。

```
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc3.d/S99_JP1_70_AS
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc5.d/S99_JP1_70_AS
```

```
chown -h root:root /etc/rc.d/rc3.d/S99_JP1_70_AS
chown -h root:root /etc/rc.d/rc5.d/S99_JP1_70_AS
```

- 自動停止のためのシンボリックリンク作成

  /etc/rc.d/rc**N**.d ディレクトリ（**N** は停止時の実行レベル）に/etc/rc.d/init.d/jp1_as_md へのシンボリックリンクを作成します。このとき，連携する JP1 シリーズ製品のサービスの停止よりも前に停止されるようにシンボリックリンクの名称を付ける必要があります。

  例えば，/etc/rc.d/rc0.d ディレクトリ，/etc/rc.d/rc6.d ディレクトリに提供されている JP1/Base の自動停止スクリプトのシンボリックリンクの名称が「K01_JP1_90_BASE」，JP1/IM - Manager の自動起動スクリプトのシンボリックリンクの名称が「K01_JP1_80_CONS」の場合，文字列の大小関係で「K01_JP1_80_CONS」よりも小さくなるようにシンボリックリンクの名称を「K01_JP1_30_AS」とし，次のように指定することで JP1/Base，JP1/IM - Manager より先に停止されます。

```
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc0.d/K01_JP1_30_AS
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc6.d/K01_JP1_30_AS
chown -h root:root /etc/rc.d/rc0.d/K01_JP1_30_AS
chown -h root:root /etc/rc.d/rc6.d/K01_JP1_30_AS
```

  なお，シンボリックリンクの名称の「K01_JP1_30_AS」の「_JP1_30_AS」部分をほかの名称にする場合は，スクリプトファイル jp1_as_md の次の記述の「_JP1_30_AS」を変更する必要があります。

```
touch /var/lock/subsys/_JP1_30_AS
rm -f /var/lock/subsys/_JP1_30_AS
```

(c) RHEL 7，SUSE Linux 12，Oracle Linux 7，CentOS 7 の場合

JP1/Advanced Shell では，ユーザー応答機能管理デーモンを自動起動・停止するための Unit ファイル jp1_as_md.service を/opt/jp1as/sample ディレクトリに格納しています。この Unit ファイルを利用して次の手順で設定します。

- /usr/lib/systemd/system ディレクトリへの追加

  /opt/jp1as/sample ディレクトリに格納されている jp1_as_md.service を/usr/lib/systemd/system に追加します。jp1_as_md.service を追加するには次のように指定します。

```
cp /opt/jp1as/sample/jp1_as_md.service /usr/lib/systemd/system
chmod u=rw,go=r /usr/lib/systemd/system/jp1_as_md.service
chown root:root /usr/lib/systemd/system/jp1_as_md.service
```

- 自動起動・自動停止の設定

  ユーザー応答機能管理デーモンの自動起動・自動停止の設定は，systemctl コマンドを次のように指定して実行します。

  systemctl --system enable jp1_as_md.service

  なお，上記の設定によりユーザー応答機能管理デーモンの自動起動と自動終了が行われた場合，タイミングによっては次の動作が発生します。

  - 自動起動時

OS が出力するユーザー応答機能管理デーモンの起動結果に，OS がユーザー応答機能管理デーモンの pid ファイルへアクセスしたときの状態についての情報が出力される場合がありますが，対処は不要です。

- 自動終了時

  システム停止時，OS が応答要求メッセージの応答待ち状態のジョブに終了シグナルを通知する場合があります。このとき，未応答の応答要求メッセージはジョブコントローラによってキャンセルされます。

(d) HP-UX の場合

JP1/Advanced Shell では，ユーザー応答機能管理デーモンを自動起動・停止するためのスクリプトファイル jp1_as_md を/opt/jp1as/sample ディレクトリに格納しています。このスクリプトファイルを利用して次の手順で設定します。

- /sbin/init.d ディレクトリへの追加

  /opt/jp1as/sample ディレクトリに格納されている jp1_as_md を/sbin/init.d に追加します。jp1_as_md は次のように指定して追加します。

```
cp /opt/jp1as/sample/jp1_as_md /sbin/init.d
chmod u=rx,go=r /sbin/init.d/jp1_as_md
chown root:sys /sbin/init.d/jp1_as_md
```

- 自動起動のためのシンボリックリンク作成

  /sbin/rc2.d ディレクトリに/sbin/init.d/jp1_as_md へのシンボリックリンクを作成します。このとき，連携する JP1 シリーズ製品のサービスが起動したあとに起動されるように，シンボリックリンクの名称を付ける必要があります。

  例えば，/sbin/rc2.d ディレクトリに提供されている JP1/Base の自動起動スクリプトのシンボリックリンクの名称が「S900jp1_base」，JP1/IM - Manager の自動起動スクリプトのシンボリックリンクの名称が「S901jp1_cons」の場合，S901 の 901（数字）部分よりも大きい数字になるようにシンボリックリンクの名称を「S905jp1_as_md」とし，次のように指定することで JP1/Base，JP1/IM - Manager よりあとに起動されます。

```
ln -s /sbin/init.d/jp1_as_md /sbin/rc2.d/S905jp1_as_md
chown -h root:sys /sbin/rc2.d/S905jp1_as_md
```

- 自動停止のためのシンボリックリンク作成

  /sbin/rc1.d ディレクトリに/sbin/init.d/jp1_as_md へのシンボリックリンクを作成します。このとき，連携する JP1 シリーズ製品のサービスの停止よりも前に停止されるようにシンボリックリンクの名称を付ける必要があります。

  例えば，/sbin/rc1.d ディレクトリに提供されている JP1/Base の自動停止スクリプトのシンボリックリンクの名称が「K100jp1_base」，JP1/IM - Manager の自動起動スクリプトのシンボリックリンクの名称が「K099jp1_cons」の場合，K099 の 099（数字）部分よりも小さい数字になるようにシンボリックリンクの名称を「K095jp1_as_md」とし，次のように指定することで JP1/Base，JP1/IM - Manager より先に停止されます。

```
ln -s /sbin/init.d/jp1_as_md /sbin/rc1.d/K095jp1_as_md
chown -h root:sys /sbin/rc1.d/K095jp1_as_md
```

(e) Solaris の場合

JP1/Advanced Shell では，ユーザー応答機能管理デーモンを自動起動・停止するためのスクリプトファイル jp1_as_md を/opt/jp1as/sample ディレクトリに格納しています。このスクリプトファイルを利用して次の手順で設定します。

- /etc/init.d ディレクトリへの追加

  /opt/jp1as/sample ディレクトリに格納されている jp1_as_md を/etc/init.d に追加します。jp1_as_md は次のように指定して追加します。

  ```
  cp /opt/jp1as/sample/jp1_as_md /etc/init.d
  chmod u=rwx,go=r /etc/init.d/jp1_as_md
  chown root:sys /etc/init.d/jp1_as_md
  ```

- 自動起動のためのシンボリックリンク作成

  /etc/rc2.d ディレクトリに/etc/init.d/jp1_as_md へのシンボリックリンクを作成します。このとき，連携する JP1 シリーズ製品のサービスが起動したあとに起動されるように，シンボリックリンクの名称を付ける必要があります。

  例えば，/etc/rc2.d ディレクトリに提供されている JP1/Base の自動起動スクリプトのシンボリックリンクの名称が「S99_JP1_10_BASE」，JP1/IM - Manager の自動起動スクリプトのシンボリックリンクの名称が「S99_JP1_20_CONS」の場合，文字列の大小関係で「S99_JP1_20_CONS」よりも大きくなるようにシンボリックリンクの名称を「S99_JP1_70_AS」とし，次のように指定することで JP1/Base，JP1/IM - Manager よりあとに起動されます。

  ```
  ln -s /etc/init.d/jp1_as_md /etc/rc2.d/S99_JP1_70_AS
  chown -h root:sys /etc/rc2.d/S99_JP1_70_AS
  ```

- 自動停止の設定

  /etc/rc0.d ディレクトリに/etc/init.d/jp1_as_md へのシンボリックリンクを作成します。このとき，連携する JP1 シリーズ製品のサービスの停止よりも前に停止されるようにシンボリックリンクの名称を付ける必要があります。

  例えば，/etc/rc0.d ディレクトリに提供されている JP1/Base の自動停止スクリプトのシンボリックリンクの名称が「K01_JP1_90_BASE」，JP1/IM - Manager の自動起動スクリプトのシンボリックリンクの名称が「K01_JP1_80_CONS」の場合，文字列の大小関係で「K01_JP1_80_CONS」よりも小さくなるようにシンボリックリンクの名称を「K01_JP1_30_AS」とし，次のように指定することで JP1/Base，JP1/IM - Manager より先に停止されます。

  ```
  ln -s /etc/init.d/jp1_as_md /etc/rc0.d/K01_JP1_30_AS
  chown -h root:sys /etc/rc0.d/K01_JP1_30_AS
  ```

(2) JP1/Base の設定

ユーザー応答機能を使用するためには，JP1/Advanced Shell が提供するユーザー応答機能用アダプタコマンド設定ファイルを JP1/Base の対応するディレクトリにコピーしておく必要があります。次に示すユーザー応答機能用アダプタコマンド設定ファイルを JP1/Base の対応するディレクトリにコピーしてください。

- コピー元のディレクトリ（JP1/Advanced Shell が提供するユーザー応答機能用アダプタコマンド設定ファイルの格納ディレクトリ）

```
/opt/jp1as/sample
```

  JP1/Advanced Shell が提供するユーザー応答機能用アダプタコマンド設定ファイル名：

  `Adapter_HITACHI_JP1_AS_USERREPLY.conf`

- コピー先のディレクトリ（JP1/Base の対応するディレクトリ）

```
/opt/jp1base/plugin/conf
```

この設定は，JP1/Advanced Shell のインストール後に一度だけ実施してください。ただし，JP1/Base を再インストールした場合は，この操作を再度実行してください。

2.8.4 JP1/IM - Manager で環境情報を設定する

ユーザー応答機能を使用するための，JP1/IM - Manager の設定手順を次に示します。この設定は，「(2) JP1 イベントの送信先ホストの指定」で HOSTNAME_JP1IM_MANAGER パラメーターに設定したホストの JP1/IM - Manager で実施してください。

(1) JP1/IM - Manager への拡張属性定義ファイルのコピー

JP1/Advanced Shell のインストール先ディレクトリの sample ディレクトリに格納されている拡張属性定義ファイルを JP1/IM - Manager へコピーします。拡張属性定義ファイルをコピーして有効にするまでの手順は次のとおりです。なお，この手順はスーパーユーザー権限で実行してください。

1. JP1/IM - Manager がインストールされたマシンに，拡張属性定義ファイルをコピーする。
   - コピー元の拡張属性定義ファイル

     コピーする拡張属性定義ファイルは，運用する言語に応じて次のように異なります。

運用する言語	コピー対象の拡張属性定義ファイル
日本語	hitachi_jp1_as_base_attr_ja.conf
英語	hitachi_jp1_as_base_attr_en.conf
中国語	hitachi_jp1_as_base_attr_cn.conf （ただし英語で表示されます）

   - コピー先のディレクトリ

JP1/IM - Manager がインストールされたマシンの，拡張属性定義ファイルのコピー先となるディレクトリは次のとおりです。

コピー先の OS の種類	コピー先のディレクトリ
Windows の場合	JP1/IM - ManagerのConsoleパス¥conf¥console¥attribute¥ JP1/IM - Manager がクラスタ運用の場合は次のとおりです。 共有フォルダ¥jp1cons¥conf¥console¥attribute¥
UNIX の場合	/etc/opt/jp1cons/conf/console/attribute/ JP1/IM - Manager がクラスタ運用の場合は次のとおりです。 共有ディレクトリ/jp1cons/conf/console/attribute/

JP1/IM - Manager の Console パスについては，マニュアル「JP1/Integrated Management - Manager 構築ガイド」を参照してください。

共有フォルダおよび共有ディレクトリについては，マニュアル「JP1/Integrated Management - Manager 構築ガイド」のクラスタシステムの記述を参照してください。

2. JP1/IM - Manager を再起動する。

(2) JP1/IM - Manager および JP1/IM - View の設定

JP1/IM - Manager および JP1/IM - View で，応答待ちイベントに関する機能を有効にします。有効になっていないと，JP1/IM - View による応答の入力ができません。

JP1/IM - Manager および JP1/IM - View の設定方法，ならびに JP1/Advanced Shell と JP1/IM - Manager との通信の設定については，マニュアル「JP1/Integrated Management - Manager 運用ガイド」の JP1/Advanced Shell との連携に関する記述を参照してください。

2.8.5 JP1/Base の環境情報を設定する

ユーザー応答機能を使用する場合，JP1/Advanced Shell と同一のホストで稼働する JP1/Base の文字コードは，adshecho コマンドおよび adshread コマンドで指定する事象通知メッセージおよび応答要求メッセージの文字コードとあわせる必要があります。

JP1/Base の文字コードの設定の詳細については，マニュアル「JP1/Base 運用ガイド」のインストールとセットアップの記述個所を参照してください。

2.9　クラスタ構成で運用する

2.9.1　クラスタ運用の前提条件とサポート範囲

JP1/Advanced Shell は，クラスタシステムでは論理ホスト環境で動作し，系切り替え時にジョブを実行する環境を引き継ぐことができます。ただし，系切り替え時に実行中のジョブを継続して実行することはできません。系切り替え後にジョブを再実行する場合は，手動でジョブを再実行してください。

JP1/Advanced Shell を論理ホスト環境で実行する場合，共有ディスクや論理 IP アドレスの割り当て・削除・動作監視がクラスタソフトによって制御されている必要があります。また，次に示す前提条件を満たすよう，システム構成や環境設定を行ってください。

(1)　論理ホスト環境の前提条件

JP1/Advanced Shell を論理ホスト環境で実行する場合，論理 IP アドレスと共有ディスクについて，次に示す前提条件があります。

表 2-32　論理ホスト環境の前提条件

論理ホストの構成要素	前提条件
共有ディスク	• 実行系サーバから待機系サーバへ引き継ぎ可能な共有ディスクが使用できること。 • JP1/Advanced Shell のプログラムが起動する前に，共有ディスクが割り当てられること。 • ユーザー応答機能管理デーモン・サービスを実行中に，共有ディスクの割り当てが解除されないこと。 • ユーザー応答機能管理デーモン・サービスを停止したあとに，共有ディスクの割り当てが解除されること。 • 共有ディスクが，不当に複数のノードから使用されないよう排他制御されていること。 • システムダウンなどでファイルが消えないよう，ジャーナル機能を持つファイルシステムなどでファイルを保護すること。 • 系切り替え時に稼働中のプログラムを計画停止した場合，ファイルに書き込んだ内容が保証されて引き継がれること。 • 系切り替え時に共有ディスクを使用中のプロセスがあっても，強制的に系切り替えができること。 • 共有ディスクの障害を検知した場合の回復処置はクラスタソフトで制御すること。回復処置の延長でユーザー応答機能管理デーモン・サービスの起動や停止が必要な場合は，クラスタソフトからユーザー応答機能管理デーモン・サービスに起動や停止の実行要求をすること。
論理 IP アドレス	• 引き継ぎ可能な論理 IP アドレスを使って通信できること。 • 論理ホスト名から論理 IP アドレスが一意に求められること。 • ユーザー応答機能管理デーモン・サービスを起動する前に，論理 IP アドレスが割り当てられること。 • ユーザー応答機能管理デーモン・サービスを実行中に，論理 IP アドレスが削除されないこと。 • ユーザー応答機能管理デーモン・サービスを実行中に，論理ホスト名と論理 IP アドレスの対応が変更されないこと。 • ユーザー応答機能管理デーモン・サービスを停止したあとに，論理 IP アドレスが削除されること。 • ネットワーク障害を検知した場合の回復処置はクラスタソフトなどが制御し，ユーザー応答機能管理デーモン・サービスが回復処理を意識する必要がないこと。また，回復処置の延長でユーザー応答機能管理デーモン・サー

論理ホストの構成要素	前提条件
論理 IP アドレス	ビスの起動や停止が必要な場合は，クラスタソフトからユーザー応答機能管理デーモン・サービスに起動や停止の実行要求をすること。 • 同一の物理ホストで複数の論理ホストを起動する場合，論理ホストごとに 1 つずつの IP アドレスを割り当てられること。

(2) 物理ホスト環境の前提条件

JP1/Advanced Shell を論理ホストで運用するクラスタシステムでは，各サーバの物理ホスト環境が次に示す前提条件を満たしている必要があります。

表 2-33 物理ホスト環境の前提条件

物理ホストの構成要素	前提条件
サーバ本体	• 2 台以上のサーバ機によるクラスタ構成になっていること。 • 実行する処理に応じた CPU 性能があること（例えば，論理ホストを多重起動する場合などに，対応できる CPU 性能があること）。 • 実行する処理に応じた実メモリ容量があること（例えば，論理ホストを多重起動する場合などに，対応できる実メモリ容量があること）。
ディスク	• システムダウンなどでファイルが消えないよう，ジャーナル機能を持つファイルシステムなどでファイルを保護すること。
ネットワーク	• 物理ホスト環境のユーザー応答機能管理デーモン・サービスを使用する場合，物理ホスト名（hostname コマンドの結果）に対応する IP アドレスで通信が可能なこと（クラスタソフトなどによって通信ができない状態に変更されないこと）※。 • ユーザー応答機能管理デーモン・サービスの動作中に，ホスト名と IP アドレスの対応が変更されないこと（クラスタソフトやネームサーバなどによって変更がされないこと）。 • Windows の場合，ホスト名に対応した LAN ボードがネットワークのバインド設定で最優先になっていること（ハートビート用などほかの LAN ボードが優先になっていないこと）。
OS，クラスタソフト	• 系切り替え後も同じ処理ができるよう，各サーバの環境が同一の設定であること。 • JP1/Advanced Shell がサポートするクラスタソフトおよびバージョンであること。 • JP1/Advanced Shell およびクラスタソフトが前提とするパッチやサービスパックが適用済みであること。

注※

クラスタソフトによっては，物理ホスト名（hostname コマンドで表示されるホスト名）に対応する IP アドレスで通信ができなくなる構成の場合があります。この場合，物理ホスト環境のユーザー応答機能管理デーモン・サービスは動作できません。論理ホスト環境のユーザー応答機能管理デーモン・サービスだけを使用してください。

(3) JP1/Advanced Shell がサポートする範囲

クラスタシステムで JP1/Advanced Shell を運用する場合，JP1/Advanced Shell がサポートする範囲は JP1/Advanced Shell 自体の動作だけです。論理ホスト環境（共有ディスクおよび論理 IP アドレス）の制御はクラスタソフトの制御に依存します。

また，論理ホスト環境および物理ホスト環境の前提条件が満たされていない，または論理ホスト環境の制御に問題がある場合は，JP1/Advanced Shell が正常に動作しないことがあります。この場合は，物理ホスト環境および論理ホスト環境の前提条件を見直す，またはクラスタソフトの設定を見直してください。

(4) 論理ホスト名の条件

論理ホスト名の条件を次に示します。

- 使用できる文字数
  Windows の場合：1～196 バイト（推奨：63 バイト以内）
  UNIX の場合：1～255 バイト（推奨：63 バイト以内）
  ただし，使用するクラスタソフトやほかの JP1 製品に上限がある場合，上限を超えないように論理ホスト名を設定してください。
- 使用できる文字
  英数字，および-（ハイフン）

2.9.2 クラスタ運用の環境情報の設定

クラスタ運用に対応するための，JP1/Advanced Shell の環境情報の設定について説明します。

(1) 連携する JP1 シリーズ製品をインストール・セットアップする

実行系サーバおよび待機系サーバで連携する JP1 シリーズ製品のインストール・セットアップを実施してください。連携する JP1 シリーズ製品のインストール・セットアップについては，各製品のマニュアルを参照してください。

(2) JP1/Advanced Shell をインストールする

実行系サーバ，待機系サーバそれぞれのローカルディスク上に JP1/Advanced Shell をインストールしてください。

また，共有ディスク上には JP1/Advanced Shell をインストールしないでください。

(3) JP1/Advanced Shell の環境情報を設定する

JP1/Advanced Shell をクラスタシステムで運用するには，次の作業を実施してください。

(a)　ディレクトリとファイル構成の検討

システムの運用方針に従い，以下の表の項目のディレクトリとファイル構成を検討します。

表 2-34　ディレクトリとファイル構成の検討項目

ディレクトリまたはファイル種別	作成基準
一時ファイル用のディレクトリ	△
スプール用のディレクトリ	○
システム実行ログ用のディレクトリ	△
トレース用のディレクトリ	×
システム環境ファイル	×
ジョブ環境ファイル	△
ジョブ定義スクリプト	△
ジョブ定義スクリプトから参照するファイル	△
上記以外	△

(凡例)

○：共有ディスク上に作成する。

△：システムの運用方針に従い，共有ディスクまたはローカルディスク上に作成する。

×：ローカルディスク上に作成する。

(b)　物理ホストの環境情報を設定する

実行系サーバおよび待機系サーバの物理ホストで，JP1/Advanced Shell の環境情報を設定します。環境情報の設定については，「2.6　JP1/Advanced Shell の環境情報を設定する」を参照してください。

(c)　論理ホストの環境情報を設定する

実行系サーバの論理ホストで，JP1/Advanced Shell の環境情報を次の手順で設定します。

1. 実行に必要なディレクトリを作成する。

   「(a)　ディレクトリとファイル構成の検討」で検討したディレクトリ構成に従って，JP1/Advanced Shell の実行に必要な次に示すディレクトリを共有ディスク上またはローカルディスク上に作成します。JP1/Advanced Shell の実行に必要なディレクトリの詳細については，「2.6.19　JP1/Advanced Shell で必要なディレクトリを作成する」を参照してください。

   - 一時ファイル用のディレクトリ
   - スプール用のディレクトリ
   - システム実行ログ用のディレクトリ
   - トレース用のディレクトリ

ディレクトリの作成方法を次に示します。

- 共有ディスク上に作成する場合

  実行系サーバから共有ディスクにアクセスできるようにし，共有ディスク上にディレクトリを作成する。

- ローカルディスク上に作成する場合

  実行系サーバおよび待機系サーバのそれぞれのローカルディスク上にディレクトリを作成する。

2. 環境ファイルを設定する。

JP1/Advanced Shell のシステム環境ファイルで，論理ホストごとの設定をする必要があります。そのためには，条件パラメーター lhost_start と lhost_end の間に，各論理ホストの環境設定パラメーターを設定します。条件パラメーター lhost_start と lhost_end の詳細については，「7.4　条件パラメーター」の「lhost_start パラメーター，lhost_end パラメーター（論理ホストだけで有効なパラメーターを定義する）」を参照してください。

論理ホスト環境で実行するためには，少なくとも次のパラメーターをシステム環境ファイルに設定します。

- 実行に必要なディレクトリのパラメーターの設定

  「(a)　ディレクトリとファイル構成の検討」で検討したディレクトリ構成に従って，手順 1.で作成したディレクトリの構成をシステム環境ファイルに設定します。これらのディレクトリはシステムの切り替え時にあわせて切り替わるよう，システム環境ファイルだけに指定し，ジョブ環境ファイルには指定しないでください。

  JP1/Advanced Shell の実行に必要なディレクトリのパラメーターの詳細については，「2.6.19 JP1/Advanced Shell で必要なディレクトリを作成する」を参照してください。

- ユーザー応答機能のパラメーターの設定

  論理ホスト環境でユーザー応答機能を使用する場合は，システム環境ファイルに，論理ホスト用のユーザー応答機能のパラメーターを設定してください。

  ユーザー応答機能のパラメーターの詳細は，「2.8.1　ユーザー応答機能を使用するための環境ファイルの設定」を参照してください。

物理ホストおよび論理ホストを指定したシステム環境ファイルの設定例を次に示します。ここで，「/shdsk1/lhost001」「/shdsk2/lhost002」は共有ディスク上のディレクトリ，「/lhost001」「/lhost002」「/phost」はローカルディスク上のディレクトリです。

```
###
### 物理ホストおよび論理ホストで共通の設定
###

#-adsh_conf USERREPLY_JP1EVENT_INTERVAL   500

###
### 物理ホストおよび論理ホストごとの設定
###

#### specify parameter for only logical host (lhost001).
#-adsh_conf lhost_start                    lhost001
#-adsh_conf LOG_DIR                        ”/shdsk1/lhost001/log”
```

```
#-adsh_conf SPOOL_DIR                    "/shdsk1/lhost001/spool"
#-adsh_conf TEMP_FILE_DIR                "/shdsk1/lhost001/temp"
#-adsh_conf TRACE_DIR                    "/lhost001/trace"
#-adsh_conf HOSTNAME_JP1IM_MANAGER       IMlhost001
#-adsh_conf USERREPLY_WAIT_MAXCOUNT      5
#-adsh_conf lhost_end

#### specify parameter for only logical host (lhost002).
#-adsh_conf lhost_start                  lhost002
#-adsh_conf LOG_DIR                      "/shdsk2/lhost002/log"
#-adsh_conf SPOOL_DIR                    "/shdsk2/lhost002/spool"
#-adsh_conf TEMP_FILE_DIR                "/shdsk2/lhost002/temp"
#-adsh_conf TRACE_DIR                    "/lhost002/trace"
#-adsh_conf HOSTNAME_JP1IM_MANAGER       IMlhost002
#-adsh_conf USERREPLY_WAIT_MAXCOUNT      5
#-adsh_conf lhost_end

#### specify parameter for physical host.
#-adsh_conf phost_start
#-adsh_conf LOG_DIR                      "/phost/log"
#-adsh_conf SPOOL_DIR                    "/phost/spool"
#-adsh_conf TEMP_FILE_DIR                "/phost/temp"
#-adsh_conf TRACE_DIR                    "/phost/trace"
#-adsh_conf HOSTNAME_JP1IM_MANAGER       IMphost001
#-adsh_conf USERREPLY_WAIT_MAXCOUNT      10
#-adsh_conf phost_end
```

3. 論理ホスト用のユーザー応答機能管理サービスを登録する。【Windows 限定】

   論理ホスト環境でユーザー応答機能を使用する場合は，実行系サーバおよび待機系サーバで論理ホスト用のユーザー応答機能管理サービスを登録する必要があります。登録は adshmsvcd コマンドおよび adshmsvce コマンドで実行できます。論理ホストに対応したサービスを登録する場合，-install オプションと-lhostname オプションを指定してコマンドを実行します。

   コマンドが正常に終了すると，登録したサービスが［サービス］管理ツールに表示されます。

   例えば，実行系サーバの実行環境に論理ホストとして lhost001 を使用する場合，次のコマンドを実行します。

```
adshmsvce -install -lhostname lhost001
```

   この例の場合，コマンドが正常に終了すると，［AdshmSvcE_ lhost001］が［サービス］管理ツールに表示されます。

4. 実行系サーバと待機系サーバのファイル構成を確認する。

   「(a)　ディレクトリとファイル構成の検討」で検討したファイル構成に応じて，次の作業を実施してください。

   - ローカルディスク上に作成する場合

     実行系サーバと待機系サーバで，参照するファイルの構成とファイルの中身を同じにする必要があります。手順 2.で作成したシステム環境ファイルなど，実行系サーバのローカルディスク上に作成したファイルを，待機系サーバの同一のパスにコピーしてください。

   - 共有ディスク上に作成する場合

     実行系サーバから共有ディスクにアクセスできるようにし，作成してください。

重要

作成したファイルは，実行系サーバおよび待機系サーバのどちらからもアクセスできるように権限を設定してください。特に，特定のユーザーやグループにアクセス権限を設定している場合は，実行系サーバと待機系サーバでユーザー名とユーザー ID（UID），グループ名とグループ ID（GID）を同じにする必要があります。

(4) クラスタソフトへの登録【Windows の場合】

論理ホストのユーザー応答機能管理サービスをクラスタソフトに登録して，クラスタソフトからの制御で起動・停止するように設定します。ただし，論理ホスト環境でユーザー応答機能を使用しない場合は，登録は不要です。

(a) クラスタソフトへの登録

Windows の場合，クラスタソフトに登録するのは，論理ホスト用のサービスとして登録された次の名称のサービスです。

名前	サービス名
AdshmSvcE_**論理ホスト名**	ユーザー応答機能管理サービス

登録方法の詳細については，各クラスタソフトのマニュアルを参照してください。クラスタソフトに登録したユーザー応答機能管理サービスは，クラスタソフトの操作で起動または停止してください。

(b) 起動停止順序の設定

論理ホストのユーザー応答機能管理サービスを実行するには，共有ディスクおよび論理 IP アドレスが使用可能になっている必要があります。また，起動停止の際の順序は，連携する JP1 シリーズ製品と依存関係があります。

- 論理ホストの起動時
  1. 共有ディスクおよび論理 IP アドレスを割り当てて使用可能にする。
  2. 連携する JP1 シリーズ製品（JP1/AJS を除く）のサービスを起動する。※
  3. ユーザー応答機能管理サービスを起動する。
  4. JP1/AJS のサービスを起動する。
- 論理ホストの停止時
  1. JP1/AJS のサービスを停止する。
  2. ユーザー応答機能管理サービスを停止する。
  3. 連携する JP1 シリーズ製品（JP1/AJS を除く）のサービスを停止する。※
  4. 共有ディスクおよび論理 IP アドレスの割り当てを解除する。

注※

連携する JP1 シリーズ製品間のサービスの起動停止順序は，各製品のマニュアルを参照してください。

(5) クラスタソフトへの登録【UNIX の場合】

論理ホストのユーザー応答機能管理デーモンをクラスタソフトに登録して，クラスタソフトからの制御で起動・停止するように設定します。ただし，論理ホスト環境でユーザー応答機能を使用しない場合は，登録は不要です。

(a) クラスタソフトへの登録

ユーザー応答機能管理デーモンをクラスタソフトへ登録する場合に必要な情報を次の表に示します。

表 2-35 クラスタソフトに登録する機能と各機能で使用するコマンド

登録する機能	説明

起動	ユーザー応答機能管理デーモンを起動します。
	使用するコマンド
	`adshmdctl`
	使用例
	`adshmdctl -h` **論理ホスト名** `start`
	起動結果の判定
	ユーザー応答機能管理デーモンを起動した結果は終了コードで判定しないで，後述する動作監視によって判定してください。

停止	ユーザー応答機能管理デーモンを停止します。
	使用するコマンド
	`adshmdctl`
	使用例
	`adshmdctl -h` **論理ホスト名** `stop`
	停止結果の判定
	ユーザー応答機能管理デーモンを停止した結果は終了コードで判定しないで，後述する動作監視によって判定してください。

動作監視	ユーザー応答機能管理デーモンが正常に動作していることを監視します。
	使用するコマンド
	`adshmdctl`
	使用例
	`adshmdctl -h` **論理ホスト名** `status`
	動作監視結果の判定
	各終了コードの判定方法を次に示します。
	終了コード=0（稼働中）
	ユーザー応答機能管理デーモンは正常に動作しています。
	終了コード=1（停止）

登録する機能	説明
動作監視	ユーザー応答機能管理デーモンは何らかの問題によって停止しています。異常と判定してください。

adshmdctl コマンドについては，「8.　運用時に使用するコマンド」の「adshmdctl コマンド（ユーザー応答機能管理デーモンを起動および停止する）【UNIX 限定】」を参照してください。

重要

ユーザー応答機能管理デーモンが何らかの障害によって共有メモリを解放しないまま終了した場合，次回の起動に失敗します。この場合，「8.　運用時に使用するコマンド」の「adshmdctl コマンド（ユーザー応答機能管理デーモンを起動および停止する）【UNIX 限定】」の「機能」の説明に従って対処してください。

(b)　起動停止順序の設定

論理ホストのユーザー応答機能管理デーモンを実行するには，共有ディスクおよび論理 IP アドレスが使用可能になっている必要があります。また，起動停止の際の順序は，連携する JP1 シリーズ製品と依存関係があります。

- 論理ホストの起動時
  1. 共有ディスクおよび論理 IP アドレスを割り当てて使用可能にする。
  2. 連携する JP1 シリーズ製品（JP1/AJS を除く）のデーモン・サービスを起動する。※
  3. ユーザー応答機能管理デーモンを起動する。
  4. JP1/AJS のサービスを起動する。
- 論理ホストの停止時
  1. JP1/AJS のサービスを停止する。
  2. ユーザー応答機能管理デーモンを停止する。
  3. 連携する JP1 シリーズ製品（JP1/AJS を除く）のデーモン・サービスを停止する。※
  4. 共有ディスクおよび論理 IP アドレスの割り当てを解除する。

注※

連携する JP1 シリーズ製品間のサービスの起動停止順序は，各製品のマニュアルを参照してください。

2.9.3　クラスタ運用の場合のコマンドの指定方法

コマンドを論理ホストで実行させるためには，コマンドに論理ホスト名を指定する必要があります。論理ホスト名を指定しないと，物理ホストでコマンドが実行されます。論理ホストは実行するコマンドによって次のように指定してください。

(1) adshexec コマンド（バッチジョブを実行する）の場合

(a) JP1/AJS のカスタムジョブから実行する場合

adshexec をカスタムジョブから実行する場合は，カスタムジョブの「詳細定義-［Custom Job］」ダイアログボックス」の「詳細情報の設定」の「論理ホスト」のチェックボックスにチェックして実行することで，論理ホストとして実行できます。

ただし，カスタムジョブを実行する JP1/AJS の実行エージェントが論理ホストで動作している必要があります。JP1/AJS の実行エージェントが物理ホストで動作している場合，正常に動作しません。

カスタムジョブの詳細については，「2.7 JP1/AJS の環境情報を設定する（JP1/AJS を使用する場合）」を参照してください。

(b) JP1/AJS のカスタムジョブ以外から実行する場合

カスタムジョブ以外から実行する場合は，次のように実行する。

```
adshexec -h ""（ダブルクォーテーション）ジョブ定義スクリプトファイルのパス名
```

論理ホストで稼働している JP1/AJS - Agent から実行した場合，環境変数 JP1_HOSTNAME に論理ホスト名が設定されます。adshexec コマンドは-h オプションに空文字列（「""（ダブルクォーテーション）」を記述）を指定した場合，環境変数 JP1_HOSTNAME から論理ホスト名を取得します。JP1_HOSTNAME 環境変数については，マニュアル「JP1/Base 運用ガイド」を参照してください。

ただし，実行する JP1/AJS の実行エージェントが論理ホストで動作している必要があります。JP1/AJS の実行エージェントが物理ホストで動作している場合，「(c) JP1/AJS 以外から実行する場合」に示すコマンドを実行してください。

(c) JP1/AJS 以外から実行する場合

次のようにコマンドに-h オプションで論理ホスト名を指定して実行してください。

```
adshexec -h 論理ホスト名 ジョブ定義スクリプトファイルのパス名
```

(2) adshexec コマンド以外の場合

次のようにコマンドに-h オプションで論理ホスト名を指定して実行してください。

```
コマンド -h 論理ホスト名
```

adshlsmsg コマンドを論理ホストで実行する場合の例を次に示します。

```
adshlsmsg -h 論理ホスト名
```

ユーザー応答機能を使用する場合，ユーザー応答機能管理デーモン・サービスに指定する論理ホスト名とadshexec コマンドに指定する論理ホスト名は同じにする必要があります。論理ホストとして実行できる各コマンドの詳細については，「8.3　シェル運用コマンド」を参照してください。

2.9.4　クラスタ運用に関する注意事項

JP1/Advanced Shell のクラスタ運用に関する注意事項を次に示します。

- 【Windows 限定】ファイル名やパス名に UNC 形式の名称を指定できます。ただし，パス名の末尾が共有名（後ろに「¥」を指定した場合を含む）になる指定はサポートしていません。
- JP1/Advanced Shell では一部のファイルシステムをサポートしていません。詳細は，「(2)　ファイルシステム」を参照してください。
- 論理ホストを複数構築して JP1/Advanced Shell を複数稼働している場合でも，ユーザー応答機能は論理ホストごとに実行されます。ある論理ホストのユーザー応答機能から，別の論理ホストのユーザー応答機能の情報を参照することはできません。
- ユーザー応答機能の応答要求メッセージを使用したジョブを実行中に，系切り替えをした場合，JP1/IM - View 上に応答待ちイベントが滞留したままになる場合があります。その場合，JP1/IM - View を操作して手動で応答待ちイベントの滞留を解除してください。
- 【UNIX 限定】JP1/Advanced Shell のジョブの実行中に，クラスタソフトによって共有ディスクが切り離された場合，実行中のジョブはシグナルによってエラー終了します。
- 【Windows 限定】JP1/Advanced Shell のジョブの実行中に，クラスタソフトによって共有ディスクが切り離された場合，実行中のジョブが共有ディスク上のファイルにアクセスするタイミングでエラー終了します。
- クラスタ環境で運用する場合，カバレージ情報は採取しないでください。

2.9.5　非クラスタ環境で論理ホストを運用する場合の設定

クラスタ環境で運用しない論理ホストの構築および運用についての概要を説明します。クラスタ環境で運用しない論理ホストも，通常のクラスタシステムで運用する場合の論理ホストと同じ環境情報を設定して運用します。

(1)　非クラスタシステムで論理ホストを運用する場合の環境情報の設定

クラスタソフトと連携しないで，クラスタ環境で運用しない論理ホスト環境で JP1/Advanced Shell を運用する手順を次に示します。

(a)　論理ホスト環境の準備

論理ホスト環境を作成するために，論理ホスト用のディスク領域および IP アドレスを用意してください。

- 論理ホスト用のディスク領域

  物理ホストやほかの論理ホストのJP1シリーズ製品が使用しているものとは別に，論理ホストのJP1/Advanced Shellが使用するファイルの格納先ディレクトリを，ローカルディスク上に作成してください。

- 論理ホスト用のIPアドレス

  論理ホストのJP1/Advanced Shellが使用するIPアドレスを，OSに割り当ててください。IPアドレスの割り当ては，実IPアドレスでもエイリアスIPアドレスでもかまいません。ただし，論理ホストから一意に特定できるIPアドレスにしてください。

  これらに対する前提条件は，クラスタシステムでの運用の場合と同じです。ただし，クラスタ環境での運用方法ではないため，「クラスタソフト」に関連する条件などは除きます。

なお，「2.9　クラスタ構成で運用する」で，共有ディスク・論理IPアドレスと説明している部分は，上記で割り当てた論理ホスト用のディスク領域・IPアドレスに読み替えてください。

- 性能の見積もり

  性能を見積もる際は，以下のような観点でシステムとして動作可能か見積もってください。

  - システム内で複数のJP1シリーズ製品が起動できるリソースを割り当てられるかどうかを見積もってください。リソースが十分に割り当てられないと，正しく動作しなかったり，十分な性能が確保できなかったりします。

(b)　論理ホストの環境情報の設定

クラスタシステムの実行系サーバと同じ手順で，論理ホストの環境情報の設定を行ってください。クラスタシステムの環境情報の設定については，「2.9.2　クラスタ運用の環境情報の設定」を参照してください。なお，クラスタシステムでは系切り替えをする両側のサーバに対して設定する必要がありますが，クラスタ環境で運用しない論理ホストでは，動作するサーバだけ設定してください。

(2)　非クラスタ環境の論理ホスト用ユーザー応答機能管理デーモンの自動起動と自動停止【UNIX限定】

システムの起動時および終了時に，ユーザー応答機能管理デーモンを自動起動・自動停止させるための設定方法を次に示します。

(a)　AIXの場合

- システム起動時の自動起動機能

  システム起動時の自動起動機能を設定するには，mkitabコマンドを使用して次のコマンドを実行します。

```
mkitab "論理ホスト用ユーザー応答機能管理デーモンのレコード:2:wait:/opt/jp1as/sbin/
adshmdctl -h 論理ホスト名 start"
```

  論理ホスト用のユーザー応答機能管理デーモンの起動は，連携するJP1シリーズ製品の論理ホスト用サービスの起動よりあとになるよう設定してください。例えば，論理ホストのJP1/Base，論理ホストのJP1/IM - Managerの順に自動起動させるには，次のように指定してmkitabコマンドを実行します。

```
mkitab -i 論理ホスト用JP1/Baseのレコード "論理ホスト用JP1/IM - Managerのレコード:
2:wait:/etc/opt/jp1cons/jco_start.cluster 論理ホスト名"
mkitab -i 論理ホスト用JP1/IM - Managerのレコード "論理ホスト用ユーザー応答機能管理デーモ
ンのレコード:2:wait:/opt/jp1as/sbin/adshmdctl -h 論理ホスト名 start"
```

- システム終了時の自動停止機能

  システム終了時の自動停止機能を設定するには，/etc/rc.shutdown を編集して，連携する JP1 シリーズ製品の論理ホスト用サービスの停止より先に停止させるように次に示す記述を追加します。

```
test -x /opt/jp1as/sbin/adshmdctl && /opt/jp1as/sbin/adshmdctl -h 論理ホスト名 stop
    :
連携するJP1シリーズ製品の論理ホスト用サービスの停止処理
    :
```

(b) RHEL 6，Oracle Linux 6，CentOS 6 の場合

- 自動起動および自動停止スクリプトの作成

  論理ホスト用の自動起動および自動停止スクリプトを/etc/rc.d/init.d ディレクトリに作成します。作成例を次に示します。

```
#!/bin/sh

JP1_HOSTNAME=論理ホスト名

case $1 in
'start')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME start
              touch /var/lock/subsys/ロックファイル名
      fi
      ;;
'stop')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME stop
              rm -f /var/lock/subsys/ロックファイル名
      fi
      ;;
esac

exit 0
```

  なお，ロックファイル名には自動停止のために作成するシンボリックリンクの名称の先頭から数字部分（KXX 部分）を除いた名称を指定します。例えば，自動停止のためのシンボリックリンクの名称が「K01_JP1_AS_CLUSTER」の場合，「_JP1_AS_CLUSTER」を指定します。

- 自動起動のためのシンボリックリンク作成

  作成した自動起動および自動停止スクリプトへのシンボリックリンクを/etc/rc.d/rc<N>.d ディレクトリ（<N>は起動時の実行レベル）に作成します。このとき，連携する JP1 シリーズ製品の論理ホスト用サービスが起動したあとに起動されるようにシンボリックリンクの名称を付ける必要があります。

シンボリックリンクの作成手順については，「(1)　ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。

- 自動停止のためのシンボリックリンク作成

  作成した自動起動および自動停止スクリプトへのシンボリックリンクを/etc/rc.d/rc<N>.d ディレクトリ（<N>は停止時の実行レベル）に作成します。このとき，連携する JP1 シリーズ製品の論理ホスト用サービスよりも先に停止されるようにシンボリックリンクの名称を付ける必要があります。シンボリックリンクの作成手順については，「(1)　ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。

なお，連携する JP1 シリーズ製品の論理ホスト用サービスの自動起動および自動停止スクリプトの作成と，作成した自動起動および自動停止スクリプトへのシンボリックリンクの作成は，ユーザー自身が実行します。連携する JP1 シリーズ製品の論理ホスト用サービスの自動起動および自動停止スクリプト名とシンボリックリンクの名称については，連携する JP1 シリーズ製品のマニュアルを参照してください。

(c)　RHEL 7，SUSE Linux 12，Oracle Linux 7，CentOS 7 の場合

- 論理ホスト用ユーザー応答機能管理デーモンのUnit ファイルの作成

  論理ホスト用ユーザー応答機能管理デーモンのUnit ファイルを/etc/systemd/system ディレクトリに作成します。作成するUnit ファイルの名称は，「jp1_as_md_*論理ホスト名*.service」のように，拡張子を「.service」にします。

  作成するUnit ファイルの記述例を次に示します。なお，「論理ホスト用 JP1/Base の Unit ファイル名」については，JP1/Base のドキュメントを参照してください。

```
[Unit]
# Service name
Description=Advanced Shell - adshmd 論理ホスト名

# Depecdencies
Requires=論理ホスト用JP1/BaseのUnitファイル名
After=論理ホスト用JP1/BaseのUnitファイル名
ConditionFileIsExecutable=/opt/jp1as/sbin/adshmdctl

[Service]
# Service type
Type=forking
PIDFile=/opt/jp1as/system/adshmd_論理ホスト名.pid

# Service operations
ExecStart=/opt/jp1as/sbin/adshmdctl -h 論理ホスト名 start
ExecStop=/opt/jp1as/sbin/adshmdctl -h 論理ホスト名 stop

KillMode=none

[Install]
WantedBy=multi-user.target graphical.target
```

  /etc/systemd/system ディレクトリに作成したUnit ファイルの所有者，属するグループ，パーミッションを次のように設定します。

chmod u=rw,go=r /etc/systemd/system/*論理ホスト用ユーザー応答機能管理デーモンの Unit ファイル名*

chown root:root /etc/systemd/system/*論理ホスト用ユーザー応答機能管理デーモンの Unit ファイル名*

- 自動起動・自動停止の設定

  論理ホスト用ユーザー応答機能管理デーモンの自動起動・自動停止の設定は，systemctl コマンドを次のように指定して実行します。

  systemctl --system enable *論理ホスト用ユーザー応答機能管理デーモンの Unit ファイル名*

なお，上記の設定によってユーザー応答機能管理デーモンを自動起動，自動終了した場合に，タイミングによって発生する動作については，「(1)　ユーザー応答機能管理デーモンの自動起動と自動停止」の「(c) RHEL 7，SUSE Linux 12，Oracle Linux 7，CentOS 7 の場合」を参照してください。

(d)　HP-UX の場合

- 自動起動および自動停止スクリプトの作成

  論理ホスト用の自動起動および自動停止スクリプトを/sbin/init.d ディレクトリに作成します。作成例を次に示します。

```
#!/bin/sh

## Set Environment-variables
PATH=/sbin:/bin:/usr/bin:/opt/jp1as/sbin
export PATH
JP1_HOSTNAME=論理ホスト名

case $1 in
start_msg)
      echo "Start Advanced Shell - adshmd $JP1_HOSTNAME"
      ;;
stop_msg)
      echo "Stop Advanced Shell - adshmd $JP1_HOSTNAME"
      ;;
'start')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME start
      fi
      ;;
'stop')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME stop
      fi
      ;;
esac

exit 0
```

- 自動起動のためのシンボリックリンク作成

作成した自動起動および自動停止スクリプトへのシンボリックリンクを/sbin/rc2.d ディレクトリに作成します。このとき，連携する JP1 シリーズ製品の論理ホスト用サービスが起動したあとに起動されるようにシンボリックリンクの名称を付ける必要があります。シンボリックリンクの作成手順については，「(1)　ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。

- 自動停止のためのシンボリックリンク作成

  作成した自動起動および自動停止スクリプトへのシンボリックリンクを/sbin/rc1.d ディレクトリに作成します。このとき，連携する JP1 シリーズ製品の論理ホスト用サービスよりも先に停止されるようにシンボリックリンクの名称を付ける必要があります。シンボリックリンクの作成手順については，「(1) ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。

なお，連携する JP1 シリーズ製品の論理ホスト用サービスの自動起動および自動停止スクリプトの作成と，作成した自動起動および自動停止スクリプトへのシンボリックリンクの作成は，ユーザー自身が実行します。連携する JP1 シリーズ製品の論理ホスト用サービスの自動起動および自動停止スクリプト名とシンボリックリンクの名称については，連携する JP1 シリーズ製品のマニュアルを参照してください。

(e)　Solaris の場合

- 自動起動および自動停止スクリプトの作成

  論理ホスト用の自動起動および自動停止スクリプトを/etc/init.d ディレクトリに作成します。作成例を次に示します。

```
#!/bin/sh

JP1_HOSTNAME=論理ホスト名

case $1 in
'start')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME start
      fi
      ;;
'stop')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME stop
      fi
      ;;
esac

exit 0
```

- 自動起動のためのシンボリックリンク作成

  作成した自動起動および自動停止スクリプトへのシンボリックリンクを/etc/rc2.d/ディレクトリに作成します。このとき，連携する JP1 シリーズ製品の論理ホスト用サービスが起動したあとに起動されるようにシンボリックリンクの名称を付ける必要があります。シンボリックリンクの作成手順については，「(1)　ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。

- 自動停止のためのシンボリックリンク作成

作成した自動起動および自動停止スクリプトへのシンボリックリンクを/etc/rc0.d/ディレクトリに作成します。このとき，連携するJP1シリーズ製品の論理ホスト用サービスよりも先に停止されるようにシンボリックリンクの名称を付ける必要があります。シンボリックリンクの作成手順については，「(1) ユーザー応答機能管理デーモンの自動起動と自動停止」を参照してください。

なお，連携するJP1シリーズ製品の論理ホスト用サービスの自動起動および自動停止スクリプトの作成と，作成した自動起動および自動停止スクリプトへのシンボリックリンクの作成は，ユーザー自身が実行します。連携するJP1シリーズ製品の論理ホスト用サービスの自動起動および自動停止スクリプト名とシンボリックリンクの名称については，連携するJP1シリーズ製品のマニュアルを参照してください。

(3) 論理ホストの指定方法

コマンドを論理ホストで実行させるためには，クラスタシステムで動作する論理ホストと同様に指定してください。クラスタシステムの論理ホストの指定方法については，「2.9.3 クラスタ運用の場合のコマンドの指定方法」を参照してください。

2.10 HTML マニュアルを組み込む

所定のフォルダに HTML マニュアルをコピーすることで，JP1/Advanced Shell のカスタムジョブプログラム，および JP1/Advanced Shell エディタから HTML マニュアルを参照できます。

HTML マニュアルの組み込み手順を次に示します。

1. プログラムプロダクトに標準添付されているマニュアル CD-ROM を用意する。
2. マニュアル CD-ROM からマニュアル「JP1/Advanced Shell」のすべての HTM ファイルおよび CSS ファイルと，GRAPHICS フォルダを次のフォルダの下にコピーする。
   - JP1/Advanced Shell からヘルプを参照する場合

     インストール先ディレクトリ¥JP1ASE¥doc¥ja¥help
   - JP1/Advanced Shell エディタからヘルプを参照する場合

     インストール先ディレクトリ¥JP1ASD¥doc¥ja¥help
   - JP1/Advanced Shell のカスタムジョブ定義プログラムからヘルプを参照する場合

     インストール先ディレクトリ¥JP1ASV¥doc¥ja¥help

2.11 アプリケーション実行エージェント機能を設定する【Windows 実行環境限定】

アプリケーション実行エージェント機能を使用する場合，次のように設定することを推奨します。

1. アプリケーション実行エージェント機能を使用するユーザーでログオンする。

   アプリケーション実行エージェント機能を使用するユーザーは，Windows の管理ツールの［ローカルセキュリティポリシー］-［ローカルポリシー］-［ユーザー権利の割り当て］-［グローバルオブジェクトの作成］で権限を付与する必要があります。

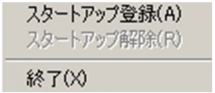
   セキュリティ上の問題などから「グローバルオブジェクトの作成」権限を有効にできない場合は，「グローバルオブジェクトの作成」権限を有効にできるユーザーでアプリケーション実行エージェント機能を使用してください。

2. Windows の［スタート］メニューから，［すべてのプログラム］－［Advanced Shell］－［アプリケーション実行エージェント］を選択する。

   タスクバーの通知領域に［アプリケーション実行エージェント］アイコンが表示されます。

3. ［アプリケーション実行エージェント］アイコンを右クリックし，［スタートアップ登録］を選択します。

   次回，ログオン時から，自動的にアプリケーション実行エージェントが起動します。

スタートアップ登録をした場合の注意事項

JP1/Advanced Shell をアンインストールする場合，アプリケーション実行エージェントをスタートアップに登録をしているときは，必ず［アプリケーション実行エージェント］アイコンを右クリックし，［スタートアップ削除］を選択して，スタートアップからアプリケーション実行エージェントを削除してください。

2.12 ジョブ定義スクリプト実行開始時のファイルモード作成マスクを設定する【UNIX 限定】

ジョブ定義スクリプト実行開始時のファイルモード作成マスクは，環境設定パラメーターUMASK_INHERITの指定に従って，次のようになります。

環境設定パラメーター	ジョブ定義スクリプト実行開始時のファイルモード作成マスク※	備考
UMASK_INHERIT NO	0	デフォルト
UMASK_INHERIT YES	親プロセスのファイルモード作成マスク	－

（凡例）

－：該当なし

注※

子孫ジョブを含む

注意事項

子孫ジョブ起動時の環境設定パラメーターでUMASK_INHERIT NOを指定した場合，ルートジョブでumaskコマンドを使用してファイルモード作成マスクを変更しても子孫ジョブには継承されません。そのため，子孫ジョブは，ファイルモード作成マスクが0で実行されます。

2.13　メモリ所要量およびディスク占有量

2.13.1　仮想メモリ所要量

仮想メモリ使用量の見積もり式を次に示します。

(1)　ジョブコントローラ

#	環境	OS	プロセス	見積もり式
1	実行環境	Linux	adshexec	12000KB ＋ (S × 1KB) ＋ (J × 2KB) ＋ (F × 3KB) ＋ (FS × 5KB)
2		AIX	adshexec	2000KB ＋ (S × 1KB) ＋ (J × 2KB) ＋ (F × 2KB) ＋ (FS × 5KB)
3		HP-UX	adshexec	12000KB ＋ (S × 1KB) ＋ (J × 2KB) ＋ (F × 3KB) ＋ (FS × 5KB)
4		Solaris	adshexec	10000KB ＋ (S × 1KB) ＋ (J × 3KB) ＋ (F × 3KB) ＋ (FS × 6KB)
5		Windows	adshexec ＋ adshexecsub	17000KB ＋ (S × 2KB) ＋ (J × 4KB) ＋ (F × 5KB) ＋ (FS × 7KB)
6	開発環境	Windows	adshedit ＋ adshesub	40000KB ＋ (S × 2KB) ＋ (J × 4KB) ＋ (F × 5KB) ＋ (FS × 14KB)

（凡例）

S：ジョブ定義スクリプト内で呼び出しているコマンドの個数

J：ジョブステップの定義数

F：ファイルの割り当て数

FS：ジョブ定義スクリプトのファイルサイズ(KB)

(2)　ユーザー応答機能管理デーモン・サービス

#	環境	OS	プロセス	見積もり式
1	実行環境	Linux	adshmd	10MB × 起動するユーザー応答機能管理デーモンの数
2		AIX	adshmd	2MB × 起動するユーザー応答機能管理デーモンの数
3		HP-UX	adshmd	10MB × 起動するユーザー応答機能管理デーモンの数
4		Solaris	adshmd	8MB × 起動するユーザー応答機能管理デーモンの数
5		Windows	adshmsvce	9MB × 起動するユーザー応答機能管理サービスの数
6	開発環境	Windows	adshmsvcd	9MB × 起動するユーザー応答機能管理サービスの数

(3) ユーザー応答機能管理デーモン・サービスが使用する共有メモリ

#	環境	OS	見積もり式
1	実行環境	Linux	1MB × 起動するユーザー応答機能管理デーモンの数
2		AIX	1MB × 起動するユーザー応答機能管理デーモンの数
3		HP-UX	1MB × 起動するユーザー応答機能管理デーモンの数
4		Solaris	1MB × 起動するユーザー応答機能管理デーモンの数
5		Windows	1MB × 起動するユーザー応答機能管理サービスの数
6	開発環境	Windows	1MB × 起動するユーザー応答機能管理サービスの数

(4) アプリケーション実行エージェントプログラムが使用する共有メモリ【Windows 限定】

12KB × GUI アプリケーション実行プログラムを実行するユーザー数

2.13.2 ディスク占有量

(1) 実行モジュールとライブラリのディスク占有量

#	環境	OS	見積もり式
1	実行環境	Linux	34MB
2		AIX	32MB
3		HP-UX	50MB
4		Solaris	31MB
5		Windows	34MB※1
6	開発環境	Windows	17MB※2

注※1

Windows の OS の種類によっては，この値より小さくなる場合があります。また，上記ディスク占有量に加え，インストール時は，一時的に最大 112MB のディスク容量が必要となります。

注※2

Windows の OS の種類によっては，この値より小さくなる場合があります。また，上記ディスク占有量に加え，インストール時は，一時的に最大 62MB のディスク容量が必要となります。

(2) ディレクトリごとのディスク占有量

(a) システム実行ログ

デフォルトは次のとおりですが，環境ファイルにより変更できます。

- 位置

#	環境	OS	位置
1	実行環境	Linux	/opt/jp1as/log (LOG_DIR パラメーター)
2		AIX	/opt/jp1as/log (LOG_DIR パラメーター)
3		HP-UX	/opt/jp1as/log (LOG_DIR パラメーター)
4		Solaris	/opt/jp1as/log (LOG_DIR パラメーター)
5		Windows	全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASE¥log (LOG_DIR パラメーター)
6	開発環境	Windows	全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASD¥log (LOG_DIR パラメーター)

- サイズ

#	環境	OS	サイズ
1	実行環境	Linux	2MB × 5 (LOG_FILE_CNT パラメーター)
2		AIX	2MB × 5 (LOG_FILE_CNT パラメーター)
3		HP-UX	2MB × 5 (LOG_FILE_CNT パラメーター)
4		Solaris	2MB × 5 (LOG_FILE_CNT パラメーター)
5		Windows	2MB × 5 (LOG_FILE_CNT パラメーター)
6	開発環境	Windows	2MB × 5 (LOG_FILE_CNT パラメーター)

(b) トレースログ

デフォルトは次のとおりですが，環境ファイルにより変更できます。

- 位置

#	環境	OS	位置
1	実行環境	Linux	/opt/jp1as/trace (TRACE_DIR パラメーター)
2		AIX	/opt/jp1as/trace (TRACE_DIR パラメーター)
3		HP-UX	/opt/jp1as/trace (TRACE_DIR パラメーター)
4		Solaris	/opt/jp1as/trace (TRACE_DIR パラメーター)

#	環境	OS	位置
5	実行環境	Windows	共通 AP データフォルダ¥Hitachi¥JP1AS¥JP1ASE¥trace (TRACE_DIR パラメーター)
6	開発環境	Windows	共通 AP データフォルダ¥Hitachi¥JP1AS¥JP1ASD¥trace (TRACE_DIR パラメーター)

- サイズ

#	環境	OS	サイズ
1	実行環境	Linux	2MB × 4 (TRACE_FILE_CNT パラメーター)
2		AIX	2MB × 4 (TRACE_FILE_CNT パラメーター)
3		HP-UX	2MB × 4 (TRACE_FILE_CNT パラメーター)
4		Solaris	2MB × 4 (TRACE_FILE_CNT パラメーター)
5		Windows	2MB × 4 (TRACE_FILE_CNT パラメーター)
6	開発環境	Windows	2MB × 4 (TRACE_FILE_CNT パラメーター)

(c) スプール

デフォルトは次のとおりですが，位置は環境ファイルにより変更できます。

- 位置

#	環境	OS	位置
1	実行環境	Linux	/var/opt/jp1as/spool (SPOOL_DIR パラメーター)
2		AIX	/var/opt/jp1as/spool (SPOOL_DIR パラメーター)
3		HP-UX	/var/opt/jp1as/spool (SPOOL_DIR パラメーター)
4		Solaris	/var/opt/jp1as/spool (SPOOL_DIR パラメーター)
5		Windows	全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASE¥spool(SPOOL_DIR パラメーター)
6	開発環境	Windows	全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASD¥spool(SPOOL_DIR パラメーター)

- サイズ

  次の表の「ファイルサイズ」列に示した計算式で算出した値の合計によって，ジョブごとのディスク所要量を算出してください。単位は KB です。

  なお，ジョブ定義スクリプトファイルのパス名の長さや環境変数の個数など，さまざまな要因によってスプールに必要となるディスク所要量は変化するため，次の計算式では余裕を持たせるために典型的なジョブの場合よりも大きめの所要量を算出します。

#	出力情報	ファイル情報	ファイルサイズ
1	ヘッダ情報など固定情報	－	500.0KB
2	実行されるステップごとのログ情報(※)	ジョブ実行ログ(JOBLOG)	実行されるスクリプトのコマンドまたは制御文の実行回数×0.2KB ＋ #-adsh_step_start コマンドの数×0.4KB
3	実行されるスクリプト情報	ファイル名に SCRIPT が含まれるファイル	ジョブ定義スクリプトファイルのファイルサイズ
4	プログラム出力データファイルに格納した情報	#-adsh_spoolfile コマンドによって割り当てたファイル	#-adsh_spoolfile コマンドで割り当てたファイルに出力されるデータ量

注※：このデータは環境設定パラメーター JOBLOG_SUPPRESS_MSG を使用することで，ジョブ実行ログへの特定の情報メッセージの出力量を削減できます。

また，実行環境については，上記に加えて，次の表の「ファイルサイズ」列に示した計算式で算出した値を所要量に加算する必要があります。

#	出力情報	ファイル情報	ファイルサイズ
1	標準出力と標準エラー出力情報	ファイル名に STDOUT および STDERR が含まれるファイル	コマンドやユーザーアプリケーションが出力するデータ量
2	稼働実績情報※	ファイル名に EVENTFILE が含まれるファイル	実行されるスクリプトのコマンドまたは制御文の実行回数×2.5KB

※：このデータは環境設定パラメーター EVENT_COLLECT を「NO」とすることで稼働実績情報の作成を抑止できます。

以上の計算式で算出した値(KB 単位)に，実行するジョブの個数を掛けて，一回の運用で必要となるスプールのディスク所要量を見積もってください。

- コマンドまたは制御文の実行回数の考え方

  「コマンドまたは制御文の実行回数」とは，実際にジョブ定義スクリプトが動作したときに実行されるコマンド・制御文の数を意味します。例を次に示します。

  例 1:

```
commandA
commandB
varA=”paramA”
commandC $varA
```

  上記のスクリプトの場合，コマンドまたは制御文の実行回数は次のようになります。

  - commandA
  - commandB
  - 代入式(varA="paramA")

- commandC

の 4 つのコマンド・制御文が動作するため，コマンドまたは制御文の実行回数は「4」となる。

例 2:

```
lop=1
while [ $lop -le 5 ] ; do
  echo ”Loop count: $lop”
  ((lop+=1))
done
```

上記のスクリプトの場合，コマンドまたは制御文の実行回数は次のようになります。

- 代入式(lop=1)
- 条件評価([$lop -le 5]) <--+
- echo コマンド　　<--+-- ループで 5 回繰り返すので， 15 ステップ相当
- 算術演算(((lop+=1))) <--+
- 条件評価([$lop -le 5]) <----- ループから抜け出すための最後の判定処理

コマンドまたは制御文の実行回数は合計「17」となります。

例 2 のようなループ処理は，ループ回数がその時々の運用の状況によって動的に定まる場合もあり得るため，考えられる最大回数を想定して，余裕を持たせた値とすることを推奨します。

(d)　ユーザー応答機能管理デーモンの起動ログ【UNIX 限定】

ユーザー応答管理デーモンの起動ログの位置およびサイズを次に示します。

- 位置

```
/opt/jp1as/system
```

- サイズ

  最大 1MB。通常は 1KB 以内。

  このファイルは，論理ホスト用のユーザー応答機能管理デーモンを動作させた場合は，起動したユーザー応答機能管理デーモンごとに作成されます。また，ユーザー応答機能管理デーモンの再起動時に再作成します。

  ユーザー応答機能管理デーモンの停止時に応答待ち状態の応答要求メッセージをキャンセルした場合や，エラーが発生した場合に出力内容が 1KB 以上になる場合があります。

(e)　アプリケーション実行エージェント機能ログ【Windows 限定】

アプリケーション実行エージェント機能ログの位置およびサイズを次に示します。

- 位置

```
全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASE¥appexec
```

- サイズ

最大 6MB+1KB(CONF ファイル)。

2.14　ウイルス対策ソフト実行時の注意事項

JP1/Advanced Shell の実行中にウイルスチェックを実行した場合，期待する動作にならないおそれがあります。また，ジョブ定義スクリプトの実行が遅延するおそれがあります。このため，JP1/Advanced Shell の実行中にウイルスチェックを実施する場合は，次に示すファイルおよびディレクトリをウイルスチェックの対象から外してください。

JP1/Advanced Shell のファイルおよびフォルダ（Windows の場合）

- *インストール先フォルダ*以下すべて
- *共通アプリケーションフォルダ*¥Hitachi¥JP1AS 以下のすべて
- *共有ドキュメントフォルダ*¥Hitachi¥JP1AS 以下のすべて
- 環境設定パラメーターLOG_DIR に指定したフォルダ以下のすべて
- 環境設定パラメーターSPOOL_DIR に指定したフォルダ以下のすべて
- 環境設定パラメーターTEMP_FILE_DIR に指定したフォルダ以下のすべて
- 環境設定パラメーターTRACE_DIR に指定したフォルダ以下のすべて
- すべてのジョブ環境ファイル
- すべてのスクリプトファイル

JP1/Advanced Shell のファイルおよびディレクトリ（UNIX の場合）

- /opt/jp1as ディレクトリ以下すべて
- /var/opt/jp1as ディレクトリ以下のすべて
- 環境設定パラメーターLOG_DIR に指定したディレクトリ以下のすべて
- 環境設定パラメーターSPOOL_DIR に指定したディレクトリ以下のすべて
- 環境設定パラメーターTEMP_FILE_DIR に指定したディレクトリ以下のすべて
- 環境設定パラメーターTRACE_DIR に指定したディレクトリ以下のすべて
- すべてのジョブ環境ファイル
- すべてのスクリプトファイル

3 バッチジョブの実行

この章では，JP1/Advanced Shell（実行環境）を使用したバッチジョブの運用として，バッチジョブの実行方法や動作について説明します。

3.1　ジョブの構成

この節では，ジョブの構成について説明します。

3.1.1　JP1/AJS のジョブに関する運用者の作業

JP1/AJS を使用してジョブを実行する場合の，運用者の作業の流れを次に示します。

(1)　ジョブの定義

JP1/AJS を使用してジョブを実行する場合には，「2.7.2　ジョブネットを定義して実行する」に示す手順に従ってジョブを定義しておきます。

(2)　ジョブの実行

JP1/AJS を使用してジョブを実行する方法には，計画実行，確定実行および即時実行があり，おのおのの方法に従って実行します。実行方法については，マニュアル「JP1/Automatic Job Management System 3 操作ガイド」を参照してください。

JP1/AJS を使用しない場合，コマンドプロンプトまたはシェルからコマンドを入力してジョブ（ジョブ定義スクリプト）を実行することもできます。

(3)　ジョブネットの監視

JP1/AJS でジョブネットモニタを起動し，ジョブの実行状態を確認します。

(4)　ジョブの再実行

ジョブの再実行が必要な場合，JP1/AJS - View の画面から再実行します。

3.1.2　ジョブ

JP1/AJS，Windows のコマンドプロンプト，または UNIX のシェルなどからジョブコントローラを起動する要求は，ジョブとして受け付けられます。ジョブを利用する一般ユーザーは，指示をまとめて記述したジョブ定義スクリプトをジョブコントローラに渡します。

ジョブを利用する一般ユーザーが指定した指示は，ジョブコントローラによって何が要求されているかが分析され，システム資源が効率良く利用されるようにジョブを実行します。

(1) ルートジョブと子孫ジョブ

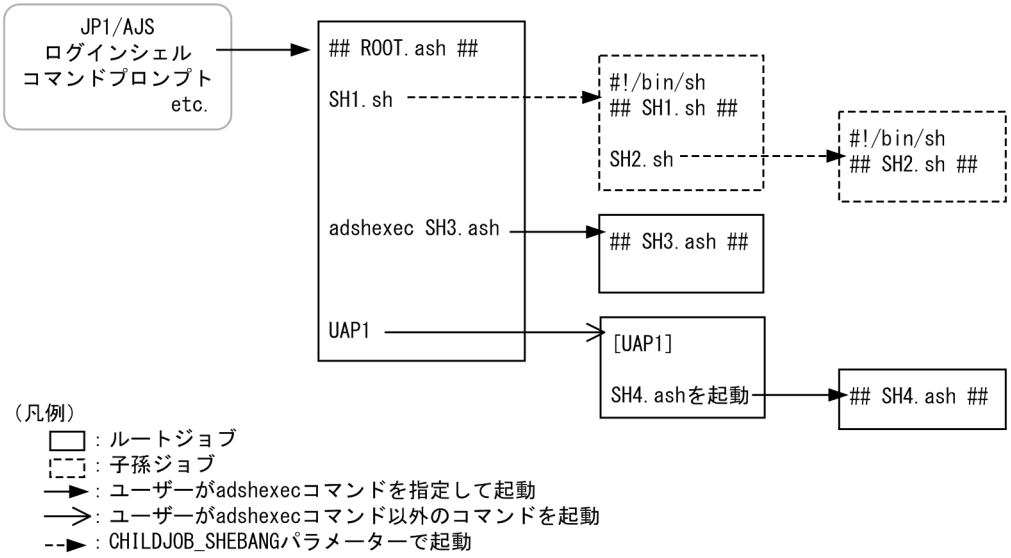
ジョブとは一般に，一般ユーザーが用意する 1 つのまとまった仕事をシステムに要求する単位です。要求する仕事は互いに独立したものと考えられます。

ジョブは一連の処理プログラムから構成されています。これらの処理プログラムを実行するには，その実行の順序，実行の条件，および処理プログラムの実行に必要なファイルを定義する必要があります。

ジョブには，ルートジョブと子孫ジョブがあります。

- ルートジョブ
  JP1/AJS やログインシェルなどから実行するジョブのうち，子孫ジョブ以外のジョブのことです。
- 子孫ジョブ
  ルートジョブの子孫プロセスとして実行されるジョブ定義スクリプトのうち，次のどれかのパラメーターの指定，またはパラメーターのデフォルト定義によって実行されるジョブのことです。
  - CHILDJOB_EXT パラメーター
  - CHILDJOB_PGM パラメーター
  - CHILDJOB_SHEBANG パラメーター

ルートジョブと子孫ジョブの起動の流れの例を次の図に示します。



（解説）
・CHILDJOB_SHEBANGパラメーターで起動したジョブSH1.shとSH2.shは子孫ジョブとなる。
・JP1/AJSなどから起動したROOT.ashはルートジョブとなる。
・adshexecコマンドを明記して起動したSH3.ashはルートジョブとなる。
・ジョブ定義スクリプトの子孫プロセスだが，adshexecでないプロセスを仲介して起動したSH4.ashはルートジョブとなる。

(2) ジョブの入力モード

ジョブは，標準入力の状態によって次に示すどちらかのモードで実行します。

- 端末入力モード

  ジョブ起動時に標準入力が端末に関連づけられている場合，このモードで動作します。このモードで起動する方法の例を次に示します。

  - ログインシェルで，標準入力を端末に関連づけた状態で adshexec コマンドを実行する。

- 非端末入力モード

  ジョブ起動時に標準入力が端末に関連づけられていない場合，このモードで動作します。

  このモードで起動する方法の例を次に示します。

  - JP1/AJS からジョブを起動する。
  - ログインシェル上で，標準入力をファイルからリダイレクトして adshexec コマンドを実行する。
  - ログインシェル上で，パイプの途中または末尾で adshexec コマンドを実行する。

ジョブ起動時に，ジョブをどちらのモードで実行するかを示す KNAX7902-I メッセージを出力します。

UNIX の場合，ジョブの入力モードによってジョブの強制終了時の動作に差異があります。詳細は，「3.11 ジョブを強制終了する」を参照してください。

Windows の場合，ジョブの入力モードによるジョブの動作の差異はありません。

(3) ジョブとジョブの関係

ルートジョブ同士は互いに独立しています。したがって，同時に処理するルートジョブ同士，または先に実行したルートジョブがあとに実行するルートジョブに影響を及ぼすことはありません。さらに，ファイル上の情報を除けば，ルートジョブからルートジョブに情報を引き継がれることもありません。

ただし，次の場合にジョブ同士が関連を持つことがあります。

- JP1/AJS によるスケジュールによっては，ルートジョブ相互に実行順序関係ができます。
- 複数のジョブで同時に同一の通常ファイルを使用する場合は，それらのジョブ同士が関連を持ちます。ジョブが適切な順序で実行されるよう JP1/AJS のスケジュールを設計する必要があります。
- ルートジョブと子孫ジョブ，および子孫ジョブと子孫ジョブは，互いに関連を持つ場合があります。

(4) ジョブと環境ファイルの関係

ルートジョブと子孫ジョブは，どちらもジョブ起動時にシステム環境ファイルおよびジョブ環境ファイルを読み込みます。したがって，次のようなケースではルートジョブと子孫ジョブは異なる環境ファイルのパラメーターで動作します。

- ルートジョブが起動時に環境ファイルを読み込んでから，子孫ジョブが起動時に環境ファイルを読み込むまでの間に，環境変数 ADSH_ENV の値が別のジョブ環境ファイルパスに書き変えられた場合。
- ルートジョブが起動時に環境ファイルを読み込んでから，子孫ジョブが起動時に環境ファイルを読み込むまでの間に，システム環境ファイルおよびジョブ環境ファイルの内容が書き変えられた場合。

ルートジョブと子孫ジョブを同じ環境ファイルのパラメーターで動作させたい場合は，環境変数 ADSH_ENV の値，および環境ファイルの内容をジョブ実行中に変更しないでください。

また，環境ファイルに export パラメーターを定義している場合，ジョブが親プロセスから引き継いだ環境変数の値よりも，環境ファイルの export パラメーターで設定した値が優先されます。例を次に示します。

- ルートジョブ root.ash

```
export ENV1=SCRIPTFILE
childjob.ash  #子孫ジョブを起動
```

- 子孫ジョブ childjob.ash

```
echo $ENV1
```

- ルートジョブ root.ash と子孫ジョブ childjob.ash が読み込む環境ファイル adshrc.ase の内容

```
export ENV1=ENVFILE
```

- 子孫ジョブ childjob.ash の echo の出力結果

```
export ENV1=ENVFILE
```

この例では次の順に動作します。

1. ルートジョブ root.ash が起動し，環境ファイル adshrc.ase の export パラメーターによって，ENV1 には ENVFILE が設定されます。
2. ルートジョブ root.ash は，子孫ジョブ childjob.ash を起動する前に，環境変数 ENV1 に SCRIPTFILE を設定します。
3. 子孫ジョブ childjob.ash が起動すると，ルートジョブ root.ash から環境変数 ENV1 を引き継ぎます。子孫ジョブ childjob.ash のプロセス起動直後の ENV1 の値は SCRIPTFILE です。
4. 子孫ジョブ childjob.ash は起動時に環境ファイル adshrc.ase を読み込みます。export パラメーターによって ENV1 には ENVFILE が設定されます。
5. 子孫ジョブ childjob.ash の echo の結果は，4.で設定された ENVFILE になります。

(5) 一時ファイルと通常ファイル

バッチジョブでは，オープン基盤製品からの情報などを一時ファイルまたは通常ファイルとして参照し，処理を実行します。

(a) 一時ファイル

一時ファイルとは，ジョブ実行時に一時的に使用するファイルです。ジョブまたはジョブステップによって自動的に作成され，ジョブ終了時には自動的に削除されます。一時ファイルは環境ファイルで定義したディレクトリに作成されます。

バッチジョブの一時ファイルとほかのアプリケーションの一時ファイルは，分けて管理することを推奨します。JP1/Advanced Shell のジョブコントローラで一時ファイルを格納するディレクトリパスは，TEMP_FILE_DIR パラメーターに指定します。通常，一時ファイルは削除されますが，障害が発生した場合は残ることもあるため，定期的に一時ファイルを削除する必要があります。

(b) 通常ファイル

通常ファイルとはジョブ定義スクリプトの入力および出力に使用するファイルで，任意のディレクトリに配置できます。ジョブ終了後にジョブ結果として残すファイルですが，ジョブの実行中に削除することもできます。

(6) 非同期実行プロセスの動作

JP1/Advanced Shell のジョブコントローラでは，関連するすべてのルートジョブ，子孫ジョブ，およびコマンドが終了するまでは，ジョブは終了しません。

(a) &や|&による非同期実行

&や|&を指定して実行したプロセスがすべて完了するまで，ジョブは終了しません。

ただし，非同期実行したプロセスが SIGSTOP などを受信し，ジョブの終了と同時に停止状態になった場合，そのプロセスの終了を待たないでジョブを終了することがあります。非同期実行したプロセスを待たないでジョブを終了したい場合は，非同期実行したい部分だけ OS のシェルを使用するように，ジョブ定義スクリプトを作成してください。

例を次に示します。

- UNIX 版での例

  非同期実行したプロセスの終了を待つジョブ定義スクリプトの例

  ```
  #!/opt/jp1as/bin/adshexec
  mycommand A &
  ```

  このジョブ定義スクリプトをジョブコントローラで実行すると，mycommand の完了を待ってからジョブが終了します。mycommand の完了を待たないでジョブを終了したい場合は，次のようにジョブ定義スクリプトを作成してください。なお，この例では，OS のシェルとして/bin/ksh を使用しています。

  非同期実行したプロセスの終了を待たないで終了するジョブ定義スクリプトの例

  ```
  #!/opt/jp1as/bin/adshexec
  /bin/ksh exec_cmdA.sh
  ```

  exec_cmdA.sh の内容

  ```
  mycommand A &
  ```

- Windows 版での例

非同期実行したプロセスの終了を待つジョブ定義スクリプトの例

```
mycommand A &
```

このジョブ定義スクリプトをジョブコントローラで実行すると，mycommand の完了を待ってからジョブが終了します。mycommand の完了を待たないでジョブを終了したい場合は，次のようにジョブ定義スクリプトを作成してください。

非同期実行したプロセスの終了を待たないで終了するジョブ定義スクリプトの例

```
cmd.exe “/c start exec_cmdA.bat”
```

exec_cmdA.bat の内容

```
mycommand A
```

(b) exec コマンドによる外部コマンドの実行

exec コマンドの引数に外部コマンドを指定した場合，adshexec コマンドは外部コマンドを子プロセスとして実行し，完了を待ちます。外部コマンドが完了したら，exec コマンドより後のコマンドは実行しないで，完了した外部コマンドの終了コードがジョブ定義スクリプトの終了コードになります。

(c) 非同期実行プロセスを wait することを通知するメッセージ

adshexec コマンド起動時，ジョブ終了時に非同期実行プロセスを wait することを通知するメッセージ KNAX7901-I が出力されます。このメッセージは，通常実行時はジョブ実行ログ，システム実行ログおよび標準エラー出力へ出力されます。デバッグ実行時は，標準エラー出力へ出力されます。

(d) 非同期実行プロセスが停止状態になっている場合のジョブの動作【UNIX 限定】

非同期実行プロセスが SIGSTOP などを受信して停止状態になっている場合，ジョブ終了時に子プロセスまたは子孫プロセスに対して SIGHUP と SIGCONT を送信します。送信が完了したあと，1 秒後にジョブの後処理を実施します。

ジョブの入力モードによって，SIGHUP と SIGCONT の送信方法が次のとおり異なります。

- 端末入力モード

  adshexec コマンドの子プロセスだけに SIGHUP と SIGCONT を送信します。adshexec コマンドの孫プロセス以下には，SIGHUP と SIGCONT を送信しません。孫プロセス以下が残った場合は，残ってしまったプロセスのプロセス ID を ps コマンドで確認し，kill コマンドを使用して手動で終了させてください。

- 非端末入力モード

  adshexec コマンドの子孫プロセスに SIGHUP と SIGCONT を送信します。

3.1.3　ジョブステップ

ジョブステップとは，ジョブを構成する実行の単位で，ジョブ定義スクリプトの一部分を，一まとまりのコマンド群としてグループ化したものです。通常ファイルや一時ファイルの割り当てをする場合，ジョブステップ内でだけ有効なファイルを定義できます。このようなファイルは，割り当てが行われたジョブステップの終了時に後処理を行います。

幾つかのジョブステップは互いに関連を持っていて，前のジョブステップが正しく処理されないと次のジョブステップの実行が意味を持たない場合があります。この場合，ジョブステップの実行条件を指定して，処理をスキップすることもできます。

ジョブステップは，次のような目的で使用します。

- コマンドやプログラムのエラー処理を自動化する
- ジョブステップ単位でシェルスクリプトの実行を制御する

(1)　コマンドエラー時の終了処理およびログ出力自動化

従来のスクリプトでは，コマンドの実行ごとに終了コードを判定し，エラーメッセージ出力，一時ファイルの削除，およびその他のエラー処理を作り込む必要がありました。

ジョブステップを使用すると，ジョブステップ内で実行するコマンドの終了コードを監視し，エラーメッセージの出力，一時ファイルの削除，あらかじめ定義したエラー処理の実行などを行うことができます。

ジョブステップを使用して，エラー時の終了処理やログ出力を自動化したスクリプトの例を次に示します。

ジョブステップを使用する場合と使用しない場合との比較

ジョブステップを使用しない場合

```
01  progA
02  ret=$?
03  if [[ $ret != 0 ]]; then    …(1)
04    echo ”progA error”        …(1)
05    exit $ret                 …(1)
06  fi                          …(1)
07
08  TEMP=”/tmp/tempfile”
09
10  progB ${INFILE_B} ${TEMP}
11  ret=$?
12  if [[ $ret != 0 ]]; then
13    echo ”progB error”
14    rm ${TEMP}                …(2)
15    exit $ret
16  fi
17
18  progC ${TEMP} ${OUTFILE_C}
19  ret=$?
20  if [[ $ret != 0 ]]; then
21    echo ”progC error”
```

```
22  fi
23
24  rm ${TEMP}                    …(2)
25  exit $ret
```

ジョブステップを使用する場合

```
01  #-adsh_job J01
02
03  #-adsh_step_start S01         …(1)
04    progA
05  #-adsh_step_end
06
07  #-adsh_step_start S02
08    #-adsh_file_temp TEMP       …(2)
09    progB ${INFILE_B} ${TEMP}
10    progC ${TEMP} ${OUTFILE_C}
11  #-adsh_step_end
```

スクリプト例の(1)および(2)について次に解説します。

(1)について

ジョブステップを使用しない場合，コマンドを実行するたびにエラー判定をし，エラーメッセージ出力処理やスクリプトの中断処理を記述する必要があります。

ジョブステップを使用する場合，コマンド群をジョブステップとして定義すれば，エラーが発生した時点でエラーメッセージを自動出力し，後続のコマンドを実行しないでジョブステップを終了できます。

(2)について

ジョブステップを使用しない場合，作成した一時ファイルの削除処理をユーザーが漏れなく作り込む必要があります。

ジョブステップを使用する場合，JP1/Advanced Shell の一時ファイル機能でそのジョブステップ用に割り当てた一時ファイルを，ジョブステップ終了時に自動的に削除します。

ジョブステップを使用すると，上記(1)(2)などの処理を自動化できます。progB がエラーになった場合のログ出力結果を次に示します。

ジョブ実行ログ（抜粋）

```
********  ジョブコントローラのメッセージ出力  ********
17:16:12 000521 KNAX0091-I J01 ジョブが開始しました。
17:16:12 000521 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
17:16:12 000521 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
17:16:12 000521 KNAX0092-I J01.S01 ステップが開始しました。
17:16:12 000521 KNAX6116-I コマンド（./progA，行番号=4）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
17:16:12 000521 KNAX6597-I J01.S01 ジョブステップが正常終了しました。rc=0 E-Time=0.001s C-Time=0.000s
17:16:12 000521 KNAX0092-I J01.S02 ステップが開始しました。
17:16:12 000521 KNAX1601-I J01.S02 ジョブステップのファイル割り当てを開始しました。
17:16:12 000521 KNAX6409-I ファイル環境変数定義名（TEMP）へ処理指定（”create”）に従ってファイルを割り当てました。path=/var/opt/jp1as/temp/TEMP_000521_J01_Z0XxJR
```

```
17:16:12 000521 KNAX6521-E コマンド（./progB，行番号=9）がエラー終了しました。rc=1 E-
Time=0.000s C-Time=0.000s        …(1)
17:16:12 000521 KNAX6410-I ファイル環境変数定義名（TEMP）で割り当てたファイルを処理指定
（”del”）に従って解放しました。path=/var/opt/jp1as/temp/TEMP_000521_J01_Z0XxJR
17:16:12 000521 KNAX1604-I 後処理指定値によってファイル（/var/opt/jp1as/temp/
TEMP_000521_J01_Z0XxJR）を削除しました。 …(2)
17:16:12 000521 KNAX6596-E J01.S02 ジョブステップがエラー終了しました。rc=1 E-
Time=0.002s C-Time=0.000s
17:16:12 000521 KNAX0101-E J01 ジョブを実行中にエラーが発生しました。
17:16:12 000521 KNAX0098-I J01 ジョブが終了しました。rc=1 E-Time=0.007s C-
Time=0.000s                                …(3)
```

ジョブ実行ログの(1)～(3)について次に解説します。

(1)について

コマンドがエラーになった時点で，後続のコマンドは実行しません。

(2)について

JP1/Advanced Shell の一時ファイル機能で割り当てた一時ファイルを自動的に削除します。

(3)について

エラー処理で指定した終了コードでジョブを終了します。

(2)　ジョブステップ単位での実行制御

関連するコマンド群をまとめてジョブステップとして定義し，ジョブステップの結果に応じてジョブの実行を制御できます。ジョブステップ単位で実行を制御するために，さまざまな機能を提供しています。

ジョブステップを使用して，ジョブステップ単位で実行を制御するスクリプトの例を次に示します。

ジョブステップを使用する場合と使用しない場合との比較

ジョブステップを使用しない場合

```
01  retmax=0
02  VAR=`progA`
03  export VAR
04  progB
05
06  tempVAR=$VAR                        …(1)
07  VAR=`progC`
08  progD
09  retD=$?                             …(2)
10  if [[ $retmax -lt $retD ]]; then
11    retmax=$retD                      …(3)
12  fi
13  VAR=$tempVAR                        …(1)
14
15  if [[ $retD -ge 16 ]]; then         …(4)
16    exit $retD
17  fi
18
19  if [[ $retD -ne 0 ]]; then          …(5)
20    if [[ $retD -eq 4 ]]; then        …(2)
```

```
21      result="progD: warning"
22    else
23      result="progD: error"
24    fi
25    progE $result
26    retE=$?
27    if [[ $retmax -lt $retE ]]; then
28      retmax=$retE                    …(3)
29    fi
30    if [[ $retE -ge 16 ]]; then       …(4)
31      exit $retE
32    fi
33  fi
34
35  progF                               …(6)
36  exit $retmax                        …(3)
```

ジョブステップを使用する場合

```
01  #-adsh_job_stop 16:                       …(4)
02  VAR=`progA`
03  export VAR
04  progB
05
06  #-adsh_step_start S01 -stepVar VAR        …(1)
07    VAR=`progC`
08    export VAR
09    progD
10  #-adsh_step_end
11
12  #-adsh_step_start S02 -run abnormal       …(5)
13    if [[ $ADSH_STEPRC_S01 -eq 4 ]]; then …(2)
14      result="STEP01: warning"
15    else
16      result="STEP01: error"
17    fi
18    progE $result
19  #-adsh_step_end
20
21  #-adsh_step_start S03 -run always         …(6)
22    progF
23    exit $ADSH_RC_STEPMAX                   …(3)
24  #-adsh_step_end
```

スクリプト例の(1)～(6)について次に解説します。

(1)について

ジョブステップを使用しない場合，同一名のシェル変数を別の用途で使用するには，一時的に別の変数に値を退避するなどの処理が必要です。

ジョブステップを使用する場合，スクリプト拡張コマンド#-adsh_step_start の stepVar 属性を使用すると，変数名を宣言するだけでジョブステップ内でだけ有効なシェル変数を使用できます。

(2)について

ジョブステップを使用しない場合，特定のコマンドの終了コードで処理を分岐するために，コマンドの終了コードを個別のシェル変数に格納しておく必要があります。

ジョブステップを使用する場合，ジョブコントローラが自動的に設定する各ジョブステップの終了コードを参照できます。

(3)について

ジョブステップを使用しない場合，コマンドの終了コードの最大値を参照するには，コマンドの実行ごとに最大値を更新する必要があります。

ジョブステップを使用する場合，ジョブコントローラが自動的に設定する各ジョブステップの終了コード最大値を参照できます。

(4)について

ジョブステップを使用しない場合，コマンドの終了コードがしきい値を超えたことによってスクリプトの実行を終了するには，コマンドの実行ごとにしきい値を判断する必要があります。

ジョブステップを使用する場合，スクリプト拡張コマンドの#-adsh_job_stop コマンドを使用すると，しきい値を宣言するだけでジョブステップの終了コードを自動的に監視できます。

(5)について

ジョブステップを使用しない場合，コマンドの終了コードによって，後続処理を実行するかどうかの判定を行って実行を制御します。

ジョブステップを使用する場合，スクリプト拡張コマンド#-adsh_step_start の run 属性に abnormal を指定することで，先行処理でエラーが発生した場合だけ実行するジョブステップを定義できます。

(6)について

ジョブステップを使用する場合，スクリプト拡張コマンド#-adsh_step_start の run 属性に always を指定することで，先行処理の成功・エラーに関係なく常に実行するジョブステップを定義できます。

3.2　バッチジョブの起動

バッチジョブの起動方法を実行方法ごとに分けて説明します。また，バッチジョブ起動後のジョブコントローラの処理について説明します。

3.2.1　実行環境から JP1/AJS を使用してジョブを起動する

実行環境から JP1/AJS を使用して JP1/Advanced Shell のバッチジョブ業務を起動する方法について説明します。

JP1/AJS でのバッチジョブ業務の自動化の詳細については，JP1/AJS のマニュアルを参照してください。JP1/Advanced Shell のジョブをジョブネットに定義して実行する方法については，「2.7.2　ジョブネットを定義して実行する」を参照してください。

バッチジョブ業務を自動化することで，コストを削減できるだけでなく，少ない人員で確実にシステムを運用できます。JP1/AJS は，このような定型的なバッチジョブ業務を自動化するための製品です。JP1/AJS は，複雑なバッチジョブ業務の組み合わせの自動化にも対応できます。また，JP1/AJS の運用時に JP1/Advanced Shell を使用することで，次のメリットが得られます。

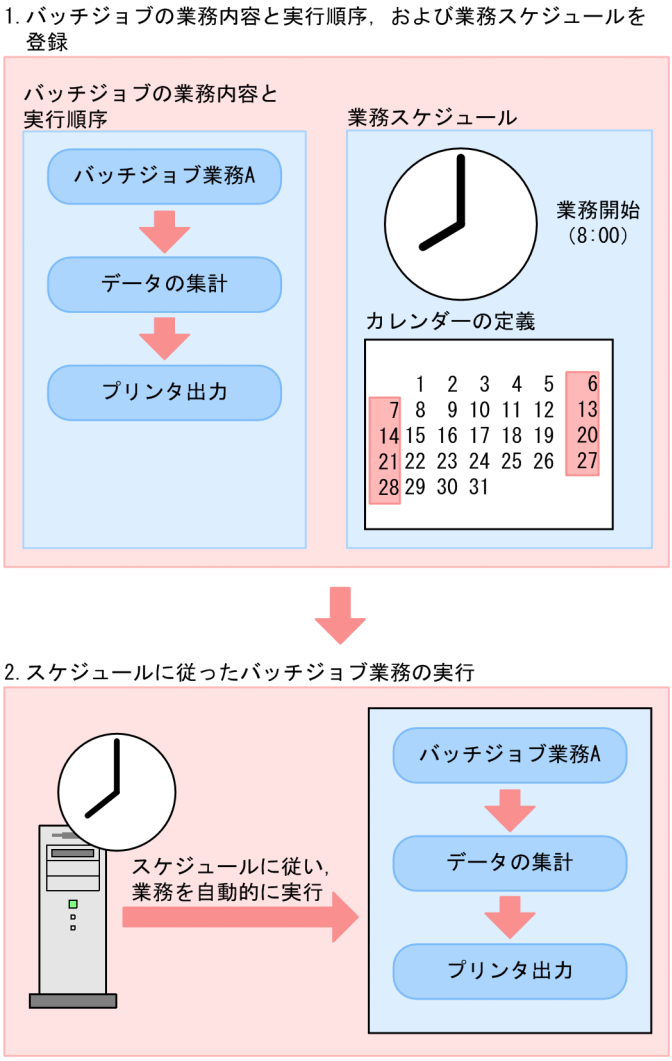
- 一時ファイル機能によって，一時的に使用するファイルを割り当てて，ジョブ終了時またはジョブステップ終了時に削除できます。
- 外部スクリプトの呼び出しによって，ジョブの定義を業務間で共用できます。
- ジョブ定義スクリプトに対する変更，追加，または削除によって，柔軟なジョブ定義ができます。

JP1/AJS を使用して，バッチジョブ業務を自動的に実行する場合，次に示す内容を定義する必要があります。

- バッチジョブ業務内容と順序
- バッチジョブ業務を実行するスケジュールまたはバッチジョブ業務の契機となる事象の登録

JP1/AJS を使用したバッチジョブ業務の自動化の概要を次の図に示します。図中の番号は，そのあとに示す説明の項番と対応しています。

図 3-1　JP1/AJS を使用したバッチジョブ業務の自動化の概要



1. バッチジョブの業務内容と実行順序，および業務スケジュールを登録します。
2. 登録されたスケジュールに従って，バッチジョブ業務が自動的に実行されます。

(1)　バッチジョブ業務と実行順序の定義

多くの業務は，決まった時間に定められた順序に従って実行されます。

例えば，売上伝票の集計は，次の順序で実行されます。

1. データベースからのデータの抽出
2. データのソート
3. プリンタ出力

ジョブコントローラのジョブステップとして，1.～3.の流れをジョブ定義スクリプトファイルに定義することで自動化を実現できますが，12:00 にデータベースからデータを抽出するという作業は自動化できません。JP1/Advanced Shell と JP1/AJS でバッチジョブ業務と実行順序の定義を実行するには，業務を構

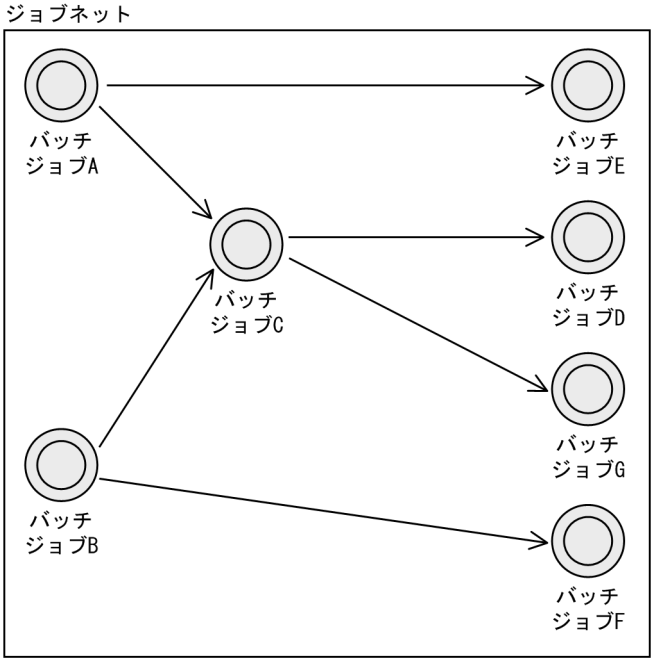
成する一連の流れをジョブコントローラで定義し，それぞれのバッチジョブ業務と実行順序の定義の関係を JP1/AJS の実行順序，または実行時間として定義します。

JP1/AJS だけでもコマンド，アプリケーションプログラム，またはジョブ定義スクリプトなどのそれぞれの作業単位に分解すれば，JP1/Advanced Shell 相当のジョブを実現できます。これらを JP1/AJS でもジョブと呼びます。

JP1/Advanced Shell と JP1/AJS でバッチジョブ業務と実行順序を定義する場合には，JP1/AJS ではバッチジョブの実行順序をジョブネットで定義します。

JP1/Advanced Shell と JP1/AJS でバッチジョブ業務と実行順序の定義を行う場合のジョブネットを次の図に示します。

図 3-2　JP1/Advanced Shell と JP1/AJS でバッチジョブ業務と実行順序の定義を行う場合のジョブネット



（説明）

JP1/AJS のジョブネットで定義するバッチジョブの実行順序の流れを次に説明します。

- バッチジョブ A が終了した場合，バッチジョブ E を実行します。
- バッチジョブ A と B が終了した場合，バッチジョブ C を実行します。
- バッチジョブ C が終了した場合，バッチジョブ D と G を実行します。
- バッチジョブ B が終了した場合，バッチジョブ F を実行します。

(2)　バッチジョブ業務と実行順序の定義スケジュールの定義

複数のバッチジョブ業務と実行順序の定義スケジュールの定義を自動化するには，この定義をいつ実行するかを決めるスケジュールの定義が必要です。

JP1/AJS のスケジュールの定義では，会社の営業日・休業日を設定したカレンダー，実行を開始する日時や実行間隔などを定義します。この定義に基づいて，JP1/AJS が実行予定を決め，その日時になると自動的に JP1/Advanced Shell のジョブ実行を始めます。

(3) バッチジョブ業務を開始する契機を登録する

ファイルの作成またはイベントの発生などをバッチジョブ業務開始の契機として登録できます。登録の結果，決まった時刻にバッチジョブ業務を開始するだけでなく，ファイルの作成またはイベントの発生など何らかの事象が起こった場合にも，バッチジョブ業務が開始できます。

3.2.2 実行環境からコマンドでバッチジョブを起動する

(1) adshexec コマンドの引数にジョブ定義スクリプトを指定する方法

実行環境からコマンドを使用してバッチジョブを起動するためには，次のように adshexec コマンドを使用します。Windows ではコマンドプロンプトからコマンドを入力し，UNIX ではシェルからコマンドを入力します。

```
adshexec batchjob1.ash
```

また，-r オプションを使用してジョブ定義スクリプトの内容を adshexec コマンドに直接指定できます。複数のコマンドを指定するには，次のように adshexec コマンドを使用します。

```
adshexec -r ”export DATA=file01 ; pgm001”
```

UNIX では，adshexec コマンドに-d オプションを指定すれば，バッチジョブをデバッグすることもできます。adshexec コマンドの詳細については，「8.3 シェル運用コマンド」の「adshexec コマンド（バッチジョブを実行する）」を参照してください。

(2) ジョブ定義スクリプトをコマンドとして指定する方法

UNIX では，1 行目に#!で始まる adshexec コマンドのパス（例：#! /opt/jp1as/bin/adshexec）を記述し，実行権限を付与したジョブ定義スクリプトファイルであれば，ジョブ定義スクリプトのファイル名だけを入力してバッチジョブを起動できます。

ジョブ定義スクリプトファイル（ファイル名：/home/user1/scripts/batchjob2.ash）

```
#! /opt/jp1as/bin/adshexec
#-adsh_job SAMPLE
（以降，ジョブ定義スクリプトの本文）
```

バッチジョブ起動の実行例

```
/home/user1/scripts/batchjob2.ash
```

注意事項

Windows では，1 行目に#!で始まる adshexec コマンドのパスを記述してコマンドプロンプトなどからジョブ定義スクリプトのファイル名だけを入力しても，バッチジョブを起動することはできません。

ただし，1 行目に#!に続けて「/opt/jp1as/bin/adshexec」または「/opt/jp1as/bin/adshexec -mMINIMUM」を記述したジョブ定義スクリプトを用意し，別のジョブ定義スクリプトからそのファイル名だけを入力することで，Windows，UNIX どちらでも子孫ジョブを起動できます。そのため，Windows でも新規に作成するジョブ定義スクリプトでは，1 行目に#!に続けて「/opt/jp1as/bin/adshexec」または「/opt/jp1as/bin/adshexec -mMINIMUM」を記述することを推奨します。

ただし，既存のシェルスクリプトを移行した場合など，1 行目に「#!/bin/sh」などが記述されているときは，シェルスクリプトを修正しないで子孫ジョブとして実行することもできます。

子孫ジョブについての詳細は「3.2.3　ジョブ定義スクリプトを子孫ジョブとして実行する」を参照してください。

3.2.3　ジョブ定義スクリプトを子孫ジョブとして実行する

ジョブ定義スクリプトを子孫ジョブとして実行する方法，および子孫ジョブの動作について説明します。優先順序については，「(3)　コマンドの実行方法の優先順序」および「(4)　関数と同名の子孫ジョブまたは外部コマンドの優先順序」を参照してください。

(1)　子孫ジョブの実行方法

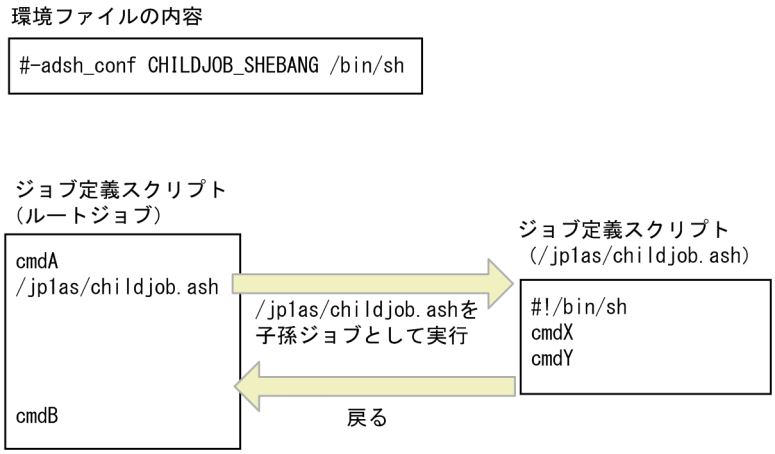
(a)　環境ファイルにパラメーターを指定して子孫ジョブを実行する方法

ルートジョブの子孫プロセスとして実行されるジョブ定義スクリプトのうち，次のどれかのパラメーターの指定，またはパラメーターのデフォルト定義によって実行されたジョブを子孫ジョブと呼びます。

- CHILDJOB_EXT パラメーター
- CHILDJOB_PGM パラメーター
- CHILDJOB_SHEBANG パラメーター

CHILDJOB_SHEBANG パラメーターの指定によって子孫ジョブを起動する動作の例を次の図に示します。

図 3-3　子孫ジョブを起動する動作の例



この例では，ルートジョブのジョブ定義スクリプトに，ほかのジョブ定義スクリプト childjob.ash を指定しています。このとき，childjob.ash は，CHILDJOB_SHEBANG パラメーターの定義に合致するため，ジョブコントローラは子プロセスとして JP1/Advanced Shell のジョブを起動し，childjob.ash を子孫ジョブとして実行します。

なお，子孫ジョブを起動するルートジョブが論理ホストで実行された場合は，子孫ジョブも論理ホストで実行されます。

(b)　パラメーターのデフォルト定義で子孫ジョブを実行する方法

CHILDJOB_SHEBANG パラメーターのデフォルト定義を使用することで，環境ファイルにパラメーターを指定することなく，子孫ジョブを起動できます。

CHILDJOB_SHEBANG パラメーターには，次の 2 つがデフォルトで定義されています。

デフォルト定義	子孫ジョブ起動時の出力モード
/opt/jp1as/bin/adshexec	OUTPUT_MODE_CHILD パラメーターの指定に従って動作します。
/opt/jp1as/bin/adshexec -mMINIMUM	最小出力モードで動作します。

これによって，1 行目に「#! /opt/jp1as/bin/adshexec」を記述したジョブ定義スクリプトをほかのジョブ定義スクリプト中に指定することで，子孫ジョブとして実行できます。また，特定の子孫ジョブだけを最小出力モードで実行したい場合は，ジョブ定義スクリプトの 1 行目に「#! /opt/jp1as/bin/adshexec -mMINIMUM」を記述してください。

デフォルト定義で実行された子孫ジョブの動作は，「(a)　環境ファイルにパラメーターを指定して子孫ジョブを実行する方法」で起動された子孫ジョブと同じです。

ただし，CHILDJOB_SHEBANG パラメーターのデフォルト定義よりも，環境ファイルに指定した CHILDJOB_SHEBANG パラメーターが優先されます。そのため，デフォルト定義と同じ内容を環境ファイルの CHILDJOB_SHEBANG パラメーターに指定した場合，次のように動作します。

- 環境ファイルの内容

```
#-adsh_conf CHILDJOB_SHEBANG ”/opt/jp1as/bin/adshexec -mMINIMUM”
```

- 子孫ジョブで起動するジョブ定義スクリプト

```
#! /opt/jp1as/bin/adshexec -mMINIMUM
                  :
```

この場合，ジョブ定義スクリプトに指定した「/opt/jp1as/bin/adshexec -mMINIMUM」は，環境ファイルの CHILDJOB_SHEBANG パラメーターの定義に合致します。そのため，子孫ジョブの出力モードは OUTPUT_MODE_CHILD パラメーターの指定に従います。

(2) ルートジョブや外部スクリプトとの機能比較

ルートジョブ，子孫ジョブおよび外部スクリプトの機能比較を次に示します。

機能	ジョブ		外部スクリプト	
	ルートジョブ	子孫ジョブ	.（ドット）コマンドの外部スクリプト	#-adsh_script の外部スクリプト
呼び出し元ジョブとのプロセスの関係	呼び出し元ジョブの子プロセスで動作する	呼び出し元ジョブの子プロセスで動作する	呼び出し元ジョブと同一プロセスで動作する	呼び出し元ジョブと同一プロセスで動作する
起動するジョブコントローラ	• UNIX の場合 adshexec コマンド • Windows の場合 adshexec.exe コマンド＋ adshexecsub.exe コマンド	• UNIX の場合 adshexec コマンド • Windows の場合 adshexecsub.exe コマンド	なし	なし
スプールジョブディレクトリ	作成する	ルートジョブのスプールジョブディレクトリ内に作成し，ジョブ終了時に削除する。 ジョブ実行ログの出力内容は次のどちらかをユーザーが選択する。 • JOBLOG だけを stderr に出力する • ルートジョブのジョブ実行ログにマージする	作成しない (コマンド実行結果を呼び出し元ジョブの JOBLOG に出力する。また，スクリプトイメージは出力しない)	作成しない (コマンド実行結果を呼び出し元ジョブの JOBLOG に出力する。また，スクリプトイメージを呼び出し元ジョブの SCRIPT に出力する)
ジョブ開始・終了メッセージ	あり (KNAX0091-I および KNAX0098-I)	あり (KNAX6571-I および KNAX6578-I)	なし	なし
環境ファイルの読み込み	する	する	しない	しない

機能	ジョブ		外部スクリプト	
	ルートジョブ	子孫ジョブ	.（ドット）コマンドの外部スクリプト	#-adsh_script の外部スクリプト
環境ファイルの読み込み	する	する	（呼び出し元ジョブの動作に従う）	（呼び出し元ジョブの動作に従う）
標準入力の使用可否	使用可	使用可	使用可	使用可
標準出力の出力先	-s オプション，-m オプション，OUTPUT_STDOUT パラメーターおよび OUTPUT_MODE_ROOT パラメーターの指定に従う	親プロセスから継承した出力先になる	呼び出し元ジョブの動作に従う	呼び出し元ジョブの動作に従う

(3) シグナル受信時の子孫ジョブの動作

ここでは，シグナル受信時の子孫ジョブの動作を説明します。

次に示すジョブを例に，ルートジョブ，子孫ジョブ，および外部コマンドへ終了要求シグナルを送信した場合の動作を示します。

```
adshexec(1)-adshexec(2)-adshexec(3)-sleep
```

JP1/AJS から強制終了（JP1/AJS から adshexec(1)へ SIGTERM 送信）した際，およびログインシェルから adshexec(1)～(3)および sleep へ SIGTERM 送信した際の動作を次に示します。

時期	adshexec(1)	adshexec(2)	adshexec(3)	sleep
JP1/AJS からの強制終了時	rc=143 でエラー終了	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了
ログインシェルから adshexec(1)への SIGTERM 送信時	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了
ログインシェルから adshexec(2)への SIGTERM 送信時	• ジョブステップの指定がある場合 rc=143 でジョブステップエラー終了。ステップエラーブロックや run abnormal/always の後続ステップを実行 • ジョブステップの指定がない場合 後続処理を続行	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了

時期	adshexec(1)	adshexec(2)	adshexec(3)	sleep
ログインシェルから adshexec(3)への SIGTERM 送信時	adshexec(2)の結果に従う	• ジョブステップの指定がある場合 rc=143 でジョブステップエラー終了。ステップエラーブロックや run abnormal/always の後続ステップを実行 • ジョブステップの指定がない場合 後続処理を続行	rc=143 でシグナルによるエラー終了	rc=143 でシグナルによるエラー終了
ログインシェルから sleep への SIGTERM 送信時	adshexec(2)の結果に従う	adshexec(3)の結果に従う	• ジョブステップの指定がある場合 rc=143 でジョブステップエラー終了。ステップエラーブロックや run abnormal/always の後続ステップを実行 • ジョブステップの指定がない場合 後続処理を続行	rc=143 でシグナルによるエラー終了

(4) 子孫ジョブからさらに実行する子孫ジョブがある場合の注意事項

子孫ジョブからさらに実行する子孫ジョブがある場合，中間のジョブが UNIX の SIGKILL や Windows の TerminateProcess で即時終了すると，ルートジョブがすべての子孫ジョブの完了を待たないで終了することがあります。そのため，このような即時終了操作は実行しないでください。

もし，この現象が発生した場合は，関連するルートジョブや子孫ジョブの実行結果を調査してください。なお，即時終了したジョブ以外のすべての子孫ジョブは，スプールジョブディレクトリが削除に失敗して残っているか，削除されていても JOBLOG の内容が標準エラー出力へ出力されているため，ログは失われません。

（例）

次のように，子孫ジョブからさらに子孫ジョブを実行するケースについて説明します。「→」は，ジョブが呼び出しによって実行されることを示します。

［ルートジョブ］→［子孫ジョブ（子）］→［子孫ジョブ（孫）］

このとき［子孫ジョブ（子)］が即時終了すると，［子孫ジョブ（孫)］より［ルートジョブ］が先に終了する場合があります。この場合の各ジョブの動作と，スプールジョブディレクトリの状態を次に示します。

項目	ジョブの種類		
	ルートジョブ	子孫ジョブ（子）	子孫ジョブ（孫）
ジョブの動作	子孫ジョブ（子）がエラー終了したものとして動作します。 ユーザープログラムがエラーで即時終了した場合と同じ動作となります。	即時終了します。	ジョブは通常どおり終了します。 ただし，Windows では，ほかの関連するジョブの状態によっては強制終了として動作することがあります。
スプールジョブディレクトリの状態	Windows で子孫ジョブ（孫）がジョブ実行ログをまだオープンしている場合，スプールジョブディレクトリのリネームに失敗します。 上記以外の場合および UNIX の場合，スプールジョブディレクトリは通常どおりリネームされます。	削除されないで残ります。	ルートジョブがスプールジョブディレクトリのリネームに成功している場合，スプールジョブディレクトリはリネームに失敗します。 上記以外の場合，通常どおり JOBLOG の内容を stderr に出力して削除されます。

3.2.4　ジョブで実行する内容をコマンドラインに指定する

adshexec コマンドで-r オプションのコマンドラインに，シェル標準コマンド，UNIX 互換コマンドなどのジョブ定義スクリプトファイルに記述できるコマンドを指定することで，ジョブ定義スクリプトファイルを作成することなく実行できます。コマンドラインにシェル標準コマンドの pwd コマンドを指定するには，次のように adshexec コマンドを実行します。

```
adshexec -r pwd
```

コマンドラインにはコマンドセパレータによる複数コマンド記述など，ジョブ定義スクリプトファイルに記述する内容を指定できます。コマンドラインに複数のコマンドを指定するには，次のように adshexec コマンドを実行します。

```
adshexec -r ”export DATA=file01 ; pgm001”
```

コマンドラインにスペースを指定する場合，クォーテーション（'および"）で囲む必要があります。また，adshexec コマンドを実行するシェルによっては，コマンドラインに指定した$，*，；（セミコロン）などのメタキャラクタが展開されるため，クォーテーション（'および"）で囲むか，エスケープ文字（¥）を使用する必要があります。メタキャラクタを指定するには，次のように adshexec コマンドを実行します。

【UNIX の場合】

・エスケープ文字を指定した場合

入力内容

```
adshexec -m MINIMUM -r ”A=(1 2 3); echo ¥${A[@]}”
```

出力結果

```
1 2 3
```

・エスケープ文字を指定しなかった場合

入力内容

```
adshexec -m MINIMUM -r "A=(1 2 3); echo ${A[@]}"
```

出力結果

この場合，何も出力されません。

・位置パラメーター$0 の出力（エスケープ文字を指定した場合）

入力内容

```
adshexec -m SIMPLE -r "echo ¥$0"
```

出力結果

```
adshexec
```

・位置パラメーター$0 の出力（エスケープ文字を指定しなかった場合）

入力内容

```
adshexec -m SIMPLE -r "echo $0"
```

出力結果

```
-bash
```

ログインシェルで位置パラメーター$0 を変換した内容を adshexec が受け取ります。ログインシェルが bash の場合は「-bash」が出力されます。

【Windows の場合】

・エスケープ文字を指定した場合

入力内容

```
adshexec -m MINIMUM -r "A=(1 2 3); echo ¥${A[@]}"
```

出力結果

```
${A[@]}
```

・エスケープ文字を指定しなかった場合

入力内容

```
adshexec -m MINIMUM -r "A=(1 2 3); echo ${A[@]}"
```

出力結果

```
1 2 3
```

・位置パラメーター$0 の出力（エスケープ文字を指定した場合）

入力内容

```
adshexec -m SIMPLE -r ”echo ¥$0”
```

出力結果

```
$0
```

・位置パラメーター$0 の出力（エスケープ文字を指定しなかった場合）

入力内容

```
adshexec -m SIMPLE -r ”echo $0”
```

出力結果

```
adshexec
```

-r オプションを指定して実行する場合は次の点に注意してください。

- コマンドラインの実行結果をほかのプログラムで利用したい場合や，コマンドラインの実行結果をコンソールやファイルに出力したい場合は，「-m SIMPLE」または「-m MINIMUM」を同時に指定してください。
- -t オプションまたは BATCH_CVR パラメーターによるカバレージ情報の採取はできません。
- ジョブ定義スクリプトファイルのパス名が出力される次の部分には，"-r CMDLINE"が出力されます。
  - スクリプトイメージファイルに出力されるジョブ定義スクリプトファイルのパス名
  - ジョブ定義スクリプトの稼働実績情報に出力されるジョブ定義スクリプトファイルのパス名
  - ジョブコントローラが出力するメッセージテキスト中のジョブ定義スクリプトファイルのパス名
- 位置パラメーター$0 には実行プログラム名"adshexec"を格納します。
- -r オプションを指定して実行した場合も，スプールジョブディレクトリは作成されます。なお，スプールジョブディレクトリが作成されるのは，ルートジョブとして-r オプションを指定して実行する場合だけです。ただし，頻繁に-r オプションを指定して実行すると，スプール内のスプールジョブが増加するので注意してください。

3.2.5　バッチジョブ起動後のジョブコントローラの処理

バッチジョブは，ジョブコントローラのプロセスとして実行されます。ジョブコントローラの起動方法は次のとおりです。

- 実行環境で JP1/AJS のスケジューリングに従って，JP1/AJS - Agent からコントローラが起動される
- 実行環境でユーザーがコマンドを入力してジョブコントローラと呼ばれるプロセスを起動する
- 開発 PC でユーザーが開発環境の編集中にテスト実行をする

ジョブを起動したあと，ジョブコントローラは，次のようにジョブを処理します。

1. ジョブコントローラが，バッチジョブを起動するオプションおよびJP1/Advanced Shellの環境ファイルを解析する。
2. ジョブコントローラは，入力されたジョブ定義スクリプトファイルを初期段階で解析する。この解析処理では，コマンドを実行しないで，構文の解析とジョブの情報を格納するテーブルの作成を行う。
3. ジョブコントローラのジョブ実行制御が，ジョブ定義スクリプトファイルを解析して実行する。
4. スクリプト拡張コマンドで使用するファイル管理機能では，通常ファイル，一時ファイルおよびプログラム出力データファイルの割り当て・解放を行う。
5. スクリプト拡張コマンドに指定されたシェル変数・環境変数では，ジョブステップの終了コードをシェル変数に格納したり，ジョブの情報を環境変数に設定したりして，ユーザープログラムから参照できるようにする。

WindowsまたはUNIXの実行環境では，ジョブ定義スクリプトを解析して実行します。ジョブ定義スクリプトの作成の詳細については，「4.　JP1/Advanced Shell - Developerを使用する【Windows限定】」および「5.　ジョブ定義スクリプトの作成」を参照してください。

3.3 adshjava コマンドを使用して Java のバッチアプリケーションを実行する【Windows，Linux，AIX，HP-UX 限定】

JP1/Advanced Shell が提供する adshjava コマンドを使用して，Java のバッチアプリケーションを実行するための前提条件と実行方法を次に示します。

- 前提条件

  JP1/Advanced Shell と同じホストに uCosminexus Application Server をインストールしてください。また，バッチサーバの設定後，バッチサーバを起動しておいてください。必要なプログラムについては，「2.2.2　環境ごとに必要なプログラム」を参照してください。

- 実行方法

  Java のバッチアプリケーションを実行する手順を次に示します。

  1. Java のバッチアプリケーションを作成する。

     Java のバッチアプリケーションの作成時の注意事項については，マニュアル「Cosminexus V9 アプリケーションサーバ 機能解説 拡張編」を参照してください。

  2. JP1/Advanced Shell の adshjava コマンドを使用して Java のバッチアプリケーションを実行する。adshjava コマンドには uCosminexus Application Server の cjexecjob コマンド（Java のバッチアプリケーションの実行）に渡す引数を指定する。

     adshjava コマンドの実行権限と実行方法については，「8.3.10　adshjava コマンド（Java のバッチアプリケーションを実行する）【Windows，Linux，AIX，HP-UX 限定】」を参照してください。

  3. adshjava コマンドは指定された引数を cjexecjob コマンドに指定して実行する。

     cjexecjob コマンドについては，マニュアル「Cosminexus V9 アプリケーションサーバ リファレンス コマンド編」を参照してください。

  4. adshjava コマンドは uCosminexus Application Server の cjexecjob コマンドの実行結果を返す。

     cjexecjob コマンドの実行結果は，adshjava コマンドの終了コードで確認できます。

     なお，ジョブの実行中にジョブコントローラが強制終了を検知した場合は，uCosminexus Application Server に対して cjkilljob コマンド（バッチアプリケーションの停止）を自動的に実行し，実行中の Java のバッチアプリケーションを強制終了してからジョブを終了します。

3.4 ジョブの実行結果を出力する

ジョブの実行結果は，スプールルートディレクトリの下にスプールジョブディレクトリとして出力されます。スプールジョブディレクトリの一部のディレクトリ内容はジョブ実行ログとして出力され，ジョブ実行時のメッセージなどを確認できます。

また，ジョブ実行ログへ出力されるメッセージは，メッセージの種類によっては出力されないよう設定できます。

3.4.1 標準出力，標準エラー出力の出力先に関する指定

JP1/Advanced Shell で実行するジョブの標準出力・標準エラー出力は，オプションの指定やジョブの実行形態によって，出力先は次のようになります。

項目		標準エラー出力		標準出力		
		ルートジョブ	子孫ジョブ	ルートジョブ		子孫ジョブ
				OUTPUT_STDOUT パラメーター※1 に SPOOL を指定	OUTPUT_STDOUT パラメーター※1 に PARENT を指定	
通常実行	拡張出力モード※2	スプールのファイル（ジョブ実行ログの標準エラー出力）※3	プロセス起動時の出力先	スプールのファイル（ジョブ実行ログの標準出力）	プロセス起動時の出力先	プロセス起動時の出力先
	簡潔出力モードまたは最小出力モード※2	プロセス起動時の出力先	プロセス起動時の出力先	プロセス起動時の出力先	プロセス起動時の出力先	プロセス起動時の出力先
デバッグ実行		プロセス起動時の出力先	プロセス起動時の出力先	プロセス起動時の出力先	プロセス起動時の出力先	プロセス起動時の出力先

注※1

OUTPUT_STDOUT パラメーターのほかに，adshexec コマンドの-s オプションでも指定できます。

注※2

拡張出力モード，簡潔出力モードおよび最小出力モードは，次のコマンドまたはパラメーターで指定します。

- adshexec コマンドの-m オプション
- adshscripttool コマンドの-m オプション
- OUTPUT_MODE_ROOT パラメーター（ルートジョブの場合）
- OUTPUT_MODE_CHILD パラメーター（子孫ジョブの場合）

注※3

ジョブ実行終了時に，ジョブコントローラ起動時の標準エラー出力に出力します。

3.4.2　ジョブの実行結果をスプールに出力する

環境ファイルに設定されたスプールルートディレクトリに，ジョブごとのディレクトリを作成し，ジョブの実行結果を出力します。ジョブごとのディレクトリには，ジョブ実行ログやジョブステップのプログラムが出力したファイルが出力されます。

スプールディレクトリの構造を次に示します。

```
スプールルートディレクトリ
 ├ロックファイル
 └スプールジョブディレクトリ
     ├.adshallocfileまたはadshallocfile※1
     ├.joborderまたはadsh.joborder※1
     ├.sysoutまたはsysout.ini※1
     ├EVENTFILE_ROOT_INF_000000_000000_000001
     ├EVENTFILE_実行開始日時_ジョブ識別子
       :
     ├EVENTFILE_実行開始日時_ジョブ識別子
     ├JOBLOG※2
     ├JOBLOG_ジョブ識別子_通番
     ├JOBLOG_子孫ジョブ起動順序通し番号※1
     ├SCRIPT※2
     ├SCRIPT_子孫ジョブ起動順序通し番号※1
     ├STDERR※2
     ├STDOUT※2
     ├ステップ番号_ステップ名_STDOUT※2
     ├ステップ番号_ステップ名_STDERR※2
     ├0000_ジョブ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名※3
     ├C子孫ジョブ起動順序通し番号_0000_ジョブ名_ファイル環境変数定義名通し番号_ファイル環境
変数定義名※3
     ├ステップ番号_ステップ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名※3
       :
     ├ステップ番号_ステップ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名
     └C子孫ジョブ起動順序通し番号_ステップ番号_ステップ名_ファイル環境変数定義名通し番号_
ファイル環境変数定義名※3
```

注※1

ジョブの実行中に一時的に作成されるファイルです。内容を次に示します。

ファイル名	説明
.adshallocfile または adshallocfile	割り当て管理ファイル
.joborder または adsh.joborder	子孫ジョブ起動順序管理ファイル
.sysout または sysout.ini	スプールジョブ管理ファイル

ファイル名	説明
JOBLOG_子孫ジョブ起動順序通し番号	環境ファイルの SPOOLJOB_CHILDJOB パラメーターに MERGE を指定（子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージする）した場合に出力される，マージ用の子孫ジョブのジョブ実行ログ
SCRIPT_子孫ジョブ起動順序通し番号	

UNIX の SIGKILL や Windows の TerminateProcess などによってジョブが即時終了した場合，これらのファイルがスプールジョブディレクトリ内に残ることがあります。スプールジョブディレクトリが不要になって削除する際は，これらのファイルも一緒に削除してください。

注※2

このファイルの内容はジョブ実行ログにも出力されます。ジョブ実行ログの出力内容については，「3.5 ジョブ実行ログ」を参照してください。

注※3

#-adsh_spoolfile コマンドによって割り当てられたプログラム出力データファイルです。プログラム出力データファイルについては，「5.9.3　プログラム出力データファイルの割り当てをする」を参照してください。

重要

- Windows の場合，EVENTFILE 以外のファイルは，ファイル名の後ろに拡張子「.sysout」が付きます。
- スプールのディレクトリの下には，JP1/Advanced Shell が作成するファイルやディレクトリではないユーザー独自のファイルは置かないでください。

一時ファイル以外のファイルおよびディレクトリについて次に説明します。

(1)　スプールルートディレクトリ

ディレクトリ名は環境ファイルの SPOOL_DIR パラメーターで設定します。

(2)　ロックファイル

同じイベントファイルが複数のコマンドから同時に使用されないよう，スプールディレクトリ単位で排他制御をするために使用されます。adshevtout コマンドおよび adshhk コマンドの実行時に生成されます。

ロックファイルのファイル名は次のとおりです。生成されたロックファイルは削除しないでください。

- UNIX の場合：.spool.lck
- Windows の場合：spool.lck

(3)　スプールジョブディレクトリ

スプールジョブディレクトリは，ジョブ識別子と同じ名称のディレクトリで，ジョブ単位に作成されます。ジョブ終了時に「*ジョブ識別子-スプールジョブ名*」に変更されます。このスプールジョブ名には JP1/

Advanced Shell のジョブ名が使用されます。シェル変数ADSH_SPOOL_JOBNAME を使用して，任意の文字列を*スプールジョブ名*に指定することもできます。

蓄積されたスプールジョブはadshhk コマンドで削除できます。adshhk コマンドについては「3.9　スプールジョブを削除する」を参照してください。

ジョブ終了後のスプールジョブディレクトリのリネーム時，同一名のディレクトリがすでに存在すると，リネームに失敗します。リネームに失敗すると「*ジョブ識別子*」のディレクトリ名のまま残ります。ジョブの動作は完了しており後続ジョブが続行できるためrc=0 で終了しますが，ディレクトリが残ったままである間，そのジョブ*識別子*は使用できなくなると同時にadshhk コマンドによる削除がされなくなります。

(4)　EVENTFILE_ROOT_INF_000000_000000_000001

ルートジョブ検索イベントファイルです。adshevtout コマンド（ジョブ定義スクリプトの稼働実績情報の出力）で指定された条件に該当するかを判定するための情報が格納されます。このファイルはルートジョブ単位に作成します。

次の場合，作成しません。

- JP1/Advanced Shell - Developer の場合
- デバッガモードで実行する場合
- KNAX0091-I メッセージを出力する前に，adshexec コマンドが実行を終了した場合

(5)　EVENTFILE_実行開始日時_ジョブ識別子

イベントファイルです。イベントファイルの作成中はファイル名の後ろに「_making」が付きます。このファイルは，ルートジョブ，子孫ジョブ単位に作成します。

イベントファイル名の例

EVENTFILE_20120422_193502_123456

次の場合，作成しません。

- JP1/Advanced Shell - Developer の場合
- デバッガモードで実行する場合
- KNAX0091-I メッセージを出力する前に，adshexec コマンドが実行を終了した場合

実行開始日時

ルートジョブまたは子孫ジョブの実行開始日時（UTC）が，次の形式で出力されます。

```
YYYYMMDD_hhmmss_dddddd
```

YYYY：西暦年を 4 桁の 10 進数（1970～2038）で出力します。

MM：月を 2 桁の 10 進数（01～12）で出力します。

DD：日を 2 桁の 10 進数（01～31）で出力します。

hh：時を 2 桁の 10 進数（00～23）で出力します。

mm：分を 2 桁の 10 進数（00～59）で出力します。

ss：秒を 2 桁の 10 進数（00～59）で出力します。

dddddd：マイクロ秒を 6 桁の 10 進数（000000～999999）で出力します。

ジョブ識別子

ルートジョブまたは子孫ジョブに付与される 6 桁の 10 進数が出力されます。

(6) JOBLOG

コマンドの実行結果やファイルの割り当て結果など，ジョブの動作状況を示すメッセージが出力されます。

(7) JOBLOG_ジョブ識別子_通番

子孫ジョブのジョブ実行ログです。

子孫ジョブ起動時に次のどちらかの方法で，子孫ジョブを簡潔出力モードまたは最小出力モードを指定した場合だけ作成されます。

- adshexec コマンドの-m オプションで SIMPLE または MINIMUM を指定する
- adshscripttool コマンドの-m オプションで SIMPLE または MINIMUM を指定する
- OUTPUT_MODE_CHILD パラメーターで SIMPLE または MINIMUM を指定する

ただし，SPOOLJOB_CHILDJOB パラメーターに MERGE（子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージする）を指定した場合は作成されません。

(8) SCRIPT

スクリプトイメージファイルです。最初に起動したジョブ定義スクリプトファイルと，#-adsh_script コマンドで指定した外部のジョブ定義スクリプトファイルの内容が出力されます。その他の.（ドット）コマンドなどで指定する外部のジョブ定義スクリプトファイルは出力されません。ログとしてジョブ定義スクリプトの内容を出力したい場合は，#-adsh_script コマンドを使用します。

なお，SPOOLJOB_CHILDJOB パラメーターで MERGE を指定し，ルートジョブを拡張出力モード，子孫ジョブを最小出力モードで動作する場合は，子孫ジョブの SCRIPT はルートジョブの SCRIPT にマージされません。詳細については，「(c) ルートジョブと子孫ジョブで出力モードが異なる場合」を参照してください。

(9) STDERR

ジョブの標準エラー出力です。ルートジョブ起動時に次のどちらかの方法で，ルートジョブを簡潔出力モードまたは最小出力モードを指定した場合は作成されません。

- adshexec コマンドの-m オプションで SIMPLE または MINIMUM を指定する

- OUTPUT_MODE_ROOT パラメーターに SIMPLE または MINIMUM を指定する

ファイルの先頭行には次に示すヘッダが出力されます。

```
********    実行ジョブのSTDERRファイルの内容      ********
```

(10) STDOUT

ジョブの標準出力です。adshexec コマンドの-s オプションおよび環境ファイルの OUTPUT_STDOUT パラメーターに SPOOL を指定した場合に作成されます。ルートジョブ起動時に次のどちらかの方法で，ルートジョブを簡潔出力モードまたは最小出力モードを指定した場合は作成されません。

- adshexec コマンドの-m オプションで SIMPLE または MINIMUM を指定する
- OUTPUT_MODE_ROOT パラメーターに SIMPLE または MINIMUM を指定する

ファイルの先頭行には次に示すヘッダが出力されます。

```
********    実行ジョブのSTDOUTファイルの内容      ********
```

(11) ステップ番号_ステップ名_STDOUT

ジョブステップを定義した場合の，該当するジョブステップ中の標準出力です。ジョブステップ名が 8 バイトを超える場合，ファイル名に含むステップ名は，ジョブステップ名の最初の 8 バイトになります。

adshexec コマンドの-s オプションおよび環境ファイルの OUTPUT_STDOUT パラメーターに SPOOL を指定した場合に作成されます。ルートジョブ起動時に次のどちらかの方法で，ルートジョブを簡潔出力モードまたは最小出力モードを指定した場合は作成されません。

- adshexec コマンドの-m オプションで SIMPLE または MINIMUM を指定する
- OUTPUT_MODE_ROOT パラメーターに SIMPLE または MINIMUM を指定する

(12) ステップ番号_ステップ名_STDERR

ジョブステップを定義した場合の，該当するジョブステップ中の標準エラー出力です。ジョブステップ名が 8 バイトを超える場合，ファイル名に含むステップ名は，ジョブステップ名の最初の 8 バイトになります。

ルートジョブ起動時に次のどちらかの方法で，ルートジョブを簡潔出力モードまたは最小出力モードを指定した場合は作成されません。

- adshexec コマンドの-m オプションで SIMPLE または MINIMUM を指定する
- OUTPUT_MODE_ROOT パラメーターに SIMPLE または MINIMUM を指定する

(13) 0000_ジョブ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

ジョブステップ外で#-adsh_spoolfile コマンドによって割り当てられたプログラム出力データファイルです。

(14) C 子孫ジョブ起動順序通し番号_0000_ジョブ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

子孫ジョブのジョブステップ外で#-adsh_spoolfile コマンドによって割り当てられたプログラム出力データファイルです。

環境ファイルの SPOOLJOB_CHILDJOB パラメーターに MERGE（子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージ）を指定した場合だけ作成されます。

(15) ステップ番号_ステップ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

ジョブステップ内で#-adsh_spoolfile コマンドによって割り当てられたプログラム出力データファイルです。

(16) C 子孫ジョブ起動順序通し番号_ステップ番号_ステップ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

子孫ジョブのジョブステップ内で#-adsh_spoolfile コマンドによって割り当てられたプログラム出力データファイルです。

環境ファイルの SPOOLJOB_CHILDJOB パラメーターに MERGE（子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージ）を指定した場合だけ作成されます。

3.4.3 ジョブ実行ログへの特定の情報メッセージの出力を抑止する

特定の情報メッセージをジョブ実行ログファイルへ出力させないことで，ジョブ実行ログファイルの出力量を抑えられます。この機能を使用するには，環境ファイルに JOBLOG_SUPPRESS_MSG パラメーターを指定します。

出力を抑止できるメッセージの種類と，JOBLOG_SUPPRESS_MSG パラメーターの詳細については，「7. 環境ファイルで設定するパラメーター」の「JOBLOG_SUPPRESS_MSG パラメーター（ジョブ実行ログへ出力させないメッセージを定義する）」を参照してください。

3.4.4 ジョブ実行ログへの情報メッセージと警告メッセージの出力を抑止する

ジョブの実行結果をほかのプログラムで利用する場合などに，次の出力が実行されないように設定できます。

- スプールジョブディレクトリ下のファイルへの，標準出力と標準エラー出力の出力
- 標準出力と標準エラー出力への，情報メッセージと警告メッセージの出力（一部の例外メッセージを除く）
- 標準エラー出力への，ジョブ終了時のジョブ実行ログの出力

そのためには，次のどちらかの方法で簡潔出力モードまたは最小出力モードを指定します。

- 環境設定時に OUTPUT_MODE_ROOT パラメーター（ルートジョブの場合）または OUTPUT_MODE_CHILD パラメーター（子孫ジョブの場合）で設定

  OUTPUT_MODE_ROOT パラメーターについては「OUTPUT_MODE_ROOT パラメーター（ルートジョブの実行結果の出力情報に関する出力方式を定義する）」，OUTPUT_MODE_CHILD パラメーターについては「OUTPUT_MODE_CHILD パラメーター（子孫ジョブの実行結果の出力情報に関する出力方式を定義する）」を参照してください。
- ジョブ実行時に adshexec コマンドの-m オプションで指定

  adshexec コマンドについては「adshexec コマンド（バッチジョブを実行する）」を参照してください。
- adshscripttool コマンドの-m オプションで指定

  adshscripttool コマンドについては「adshscripttool コマンド（ジョブ定義スクリプトの作成を支援する）」を参照してください。

環境設定パラメーターとコマンドの両方を指定した場合はコマンドの指定が優先されます。どちらも指定しなかった場合は拡張出力モードで実行します。

(1) 拡張出力モード，簡潔出力モードおよび最小出力モードの出力内容の差異

拡張出力モード，簡潔出力モードおよび最小出力モードの出力内容の差異を次に示します。

表 3-1 拡張出力モード，簡潔出力モードおよび最小出力モードの出力内容の差異

出力時期	拡張出力モードの場合	簡潔出力モードの場合	最小出力モードの場合
ジョブ実行時	標準出力，標準エラー出力は，ジョブの種類によって次のように異なります。 • ルートジョブ 標準出力と標準エラー出力をスプール	標準出力，標準エラー出力は，プロセス起動時の出力先へ出力します。 標準出力，標準エラー出力へ出力するジョブコントローラのメッセージは，エラーメッセージだけを出力します。※1	標準出力，標準エラー出力は，プロセス起動時の出力先へ出力します。 標準出力，標準エラー出力へ出力するジョブコントローラのメッセージは，一部を除くエラーメッセー

出力時期	拡張出力モードの場合	簡潔出力モードの場合	最小出力モードの場合
ジョブ実行時	ジョブディレクトリへ出力します。 • 子孫ジョブ プロセス起動時の出力先へ出力します。	標準出力，標準エラー出力は，プロセス起動時の出力先へ出力します。 標準出力，標準エラー出力へ出力するジョブコントローラのメッセージは，エラーメッセージだけを出力します。※1	ジだけを出力します。※1 ※3
ジョブ終了時	ジョブ実行ログを標準エラー出力（子孫ジョブの場合はルートジョブの標準エラー出力※2）に出力します。	ジョブ実行ログを標準エラー出力に出力しません。ただし，JOBLOG だけに出力するエラーメッセージは，エラーを通知するため，ジョブ実行中に標準エラー出力にも出力します。 子孫ジョブのJOBLOG は，ルートジョブのスプールジョブディレクトリ下に作成され，ジョブ終了後も残ります。※2 JOBEXECLOG_PRINT パラメーターの指定に関係なく，この動作となります。	同左
デバッグ実行時	JOBLOG を標準エラー出力にタイムリーに出力します。 標準出力，標準エラー出力へ出力するジョブコントローラのメッセージは起動時の標準出力，標準エラー出力へ出力します。	JOBLOG は標準エラー出力に出力しません。 標準出力，標準エラー出力へ出力するジョブコントローラのメッセージは起動時の標準出力，標準エラー出力へ出力します。 デバッグ終了時はエラーメッセージ以外のメッセージは出力しません。ただし，デバッグ対象ではない子孫ジョブは，通常実行と同様に動作します。	JOBLOG は標準エラー出力に出力しません。 標準出力，標準エラー出力へ出力するジョブコントローラのメッセージは起動時の標準出力，標準エラー出力へ出力します。 デバッグ終了時は出力抑止対象外のメッセージだけを出力します。ただし，デバッグ対象ではない子孫ジョブは，通常実行と同様に動作します。

注※1

エラーメッセージ以外にも，例外として出力されるメッセージがあります。例外として出力されるメッセージと，メッセージの種類ごとの出力先については，「12.2　メッセージの出力先」を参照してください。

注※2

子孫ジョブのジョブ実行ログをルートジョブのジョブ実行ログへマージする場合（SPOOLJOB_CHILDJOB パラメーターにMERGE を指定）は出力されません。

注※3

最小出力モードで出力抑止されるエラーメッセージについては「3.5.1　ジョブの種類ごとのジョブ実行ログの出力内容」を参照してください。

ジョブ定義スクリプトから別のジョブ定義スクリプトを簡潔出力モードまたは最小出力モードで起動する場合は，子孫ジョブを使用してください。ルートジョブを簡潔出力モードまたは最小出力モードで起動した場合，標準エラー出力にエラーメッセージが表示されます。

(2)　簡潔出力モードまたは最小出力モードで実行するジョブのスプールジョブディレクトリを探す方法

簡潔出力モードまたは最小出力モードを選択すると，割り当てられたジョブ識別子や，スプールジョブディレクトリ名を出力するメッセージが出力されなくなります。簡潔出力モードまたは最小出力モードで実行したジョブのスプールジョブディレクトリを探す方法を次に示します。

- 事前に#-adsh_job コマンド（ジョブ名の宣言）で一意なジョブ名を指定する。または，シェル変数ADSH_SPOOL_JOBNAME で一意なスプールジョブ名を指定する。
- ジョブ開始時に次の環境変数またはシェル変数の値を標準エラー出力に出力させるか，特定のファイルに出力させて必要なときに参照できるようにしておく。
  - 環境変数ADSH_JOBID（ジョブ識別子が格納される）
  - 環境変数ADSH_JOB_NAME（ジョブ名が格納される）
  - シェル変数ADSH_SPOOL_JOBNAME（スプールジョブ名を格納する）
- ジョブの実行日時を基に探す。

3.5　ジョブ実行ログ

ジョブ実行ログとは，バッチジョブの実行結果を通知する利用者向けのログ情報のことです。このログ情報はスプールジョブディレクトリ下のファイルに出力され，ユーザープログラムの標準出力以外はジョブ終了時に標準エラー出力に出力されます。標準エラー出力の結果は JP1/AJS - View などによって確認できます。

ジョブ実行ログには，次の情報が出力されます。

- バッチジョブの開始・終了メッセージ
- ジョブステップの開始・終了メッセージ
- ジョブ定義スクリプトの内容
- 実行したコマンドの結果
- 準備したファイルの状況，後処理の結果
- ユーザープログラムの標準出力（stdout）※1
- ユーザープログラムの標準エラー出力（stderr）※2
- カバレージ取得に関するメッセージ

注※1

次のどちらかが指定されている場合に，ジョブ実行中に起動時の標準出力へ出力されます。

- adshexec コマンドの-s オプションまたは環境ファイルの OUTPUT_STDOUT パラメーターに PARENT を指定した場合
- ルートジョブが簡潔出力モードまたは最小出力モードの場合

注※2

ルートジョブが簡潔出力モードまたは最小出力モードの場合は，スプールジョブディレクトリ下のファイルには出力されませんが，ジョブ実行中に起動時の標準エラー出力へ出力されます。

JP1/AJS を使用しない場合は，環境ファイルの SPOOL_DIR パラメーターに指定したスプールルートディレクトリ中の，バッチジョブごとのディレクトリに格納されている JOBLOG ファイルなどを参照してください。

なお，情報メッセージの一部を JOBLOG ファイルへ出力させないようにするには，JOBLOG_SUPPRESS_MSG パラメーターで設定します。指定できるメッセージなどについては，「7.　環境ファイルで設定するパラメーター」の「JOBLOG_SUPPRESS_MSG パラメーター（ジョブ実行ログへ出力させないメッセージを定義する）」を参照してください。

3.5.1　ジョブの種類ごとのジョブ実行ログの出力内容

ジョブ実行ログの出力内容は，実行したジョブの種類によって次のように異なります。

(1) ルートジョブ実行時のジョブ実行ログの出力先と出力内容

ルートジョブ実行時のジョブ実行ログの出力先と出力内容について，拡張出力モード，簡潔出力モードおよび最小出力モード（OUTPUT_MODE_ROOT パラメーターで指定）に分けて説明します。

(a) 拡張出力モードを選択した場合

拡張出力モードを選択した場合のジョブ実行ログの出力先を次に示します。

メッセージの出力先	内容
JOBLOG	スプール内のファイルに出力されます。 デバッグ実行時は，標準エラー出力にもジョブ実行中に出力されます。
スクリプトイメージ	スプール内のファイルに出力されます。
標準出力の出力先	次のどちらかで指定した出力先へ出力されます。 • adshexec コマンドの-s オプション • 環境ファイルの OUTPUT_STDOUT パラメーター デバッグ実行時は起動時の標準出力に出力されます。
標準エラー出力の出力先	スプール内のファイルに出力されます。 デバッグ実行時は起動時の標準エラー出力に出力されます。

ジョブごとにスプールジョブディレクトリが作成されます。

ジョブ実行後，標準出力を除いたジョブ実行ログの内容が標準エラー出力に出力されます。

デバッグ実行時は，ジョブ終了後のジョブ実行ログの内容を標準エラー出力に出力しません。

(b) 簡潔出力モードを選択した場合

簡潔出力モードを選択した場合のジョブ実行ログの出力先を次に示します。

メッセージの出力先	内容
JOBLOG	スプール内のファイルに出力されます。
スクリプトイメージ	
標準出力の出力先	スプール内のファイルには出力されません。プロセス起動時の出力先に出力されます。 標準エラー出力のエラーメッセージと，標準出力のエラーメッセージが出力されます（デバッグ実行時にはエラーメッセージ以外のメッセージも出力されます）。 また，JOBLOG のエラーメッセージが標準エラー出力に出力されます。 通常実行時には，メッセージ種別が W および I のメッセージ（シグナル受信，イベント受信メッセージは除く）は出力抑止されます。
標準エラー出力の出力先	

なお，ジョブ終了時にジョブ実行ログは標準エラー出力に出力されません。

(c)　最小出力モードを選択した場合

最小出力モードを選択した場合のジョブ実行ログの出力先を次に示します。

メッセージの出力先	内容
JOBLOG	スプール内のファイルに，出力抑止対象外のメッセージが出力されます。
スクリプトイメージ	
標準出力の出力先	スプール内のファイルには出力されません。プロセス起動時の出力先に出力されます。 標準エラー出力と標準出力の出力抑止対象外のメッセージが出力されます（デバッグ実行時には出力抑止対象外のメッセージも出力されます）。 また，JOBLOG の出力抑止対象外のメッセージが標準エラー出力に出力されます。
標準エラー出力の出力先	

なお，ジョブ終了時にジョブ実行ログは標準エラー出力に出力されません。

出力が抑止されるメッセージは次のとおりです。ただし，出力モードにかかわらず，出力する例外メッセージもあります。この例外メッセージは「12.2　メッセージの出力先」を参照してください。

時期	出力抑止されるメッセージ
通常実行時	• メッセージ種別が W および I のメッセージ（KNAX7893-I および KNAX7896-I 以外のシグナル受信，イベント受信メッセージは除く） • メッセージ種別が E の次に示すメッセージ KNAX0101-E，KNAX2201-E，KNAX6521-E，KNAX6522-E，KNAX6541-E，KNAX6542-E，KNAX6551-E，KNAX6552-E，KNAX6561-E，KNAX6562-E，KNAX6586-E，KNAX6591-E，KNAX6592-E，KNAX6593-E，KNAX6594-E，KNAX6596-E • メッセージ種別が I の次に示すメッセージ KNAX7893-I，KNAX7896-I
デバッグ実行時	• メッセージ種別が W および I のメッセージ（シグナル受信，イベント受信メッセージは除く） • メッセージ種別が E の次に示すメッセージ KNAX0101-E，KNAX2201-E，KNAX6521-E，KNAX6522-E，KNAX6541-E，KNAX6542-E，KNAX6551-E，KNAX6552-E，KNAX6561-E，KNAX6562-E，KNAX6586-E，KNAX6591-E，KNAX6592-E，KNAX6593-E，KNAX6594-E，KNAX6596-E

(2)　子孫ジョブ実行時のジョブ実行ログの出力先と出力内容

子孫ジョブ実行時のジョブ実行ログの出力先と出力内容について，拡張出力モード，簡潔出力モードおよび最小出力モード（OUTPUT_MODE_CHILD パラメーターで指定）に分けて説明します。子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージする場合の出力内容は，「(3)　子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージした場合」を参照してください。

(a)　拡張出力モードを選択した場合

拡張出力モードを選択した場合のジョブ実行ログの出力先を次に示します。

メッセージの出力先	内容
JOBLOG	一時的にスプール内のファイルに出力されます。 子孫ジョブ終了時に，プロセス起動時の標準エラー出力に出力したあと，スプール内のファイルは削除されます。
スクリプトイメージ	一時的にスプール内のファイルに出力されますが，子孫ジョブ終了時に削除されます。
標準出力の出力先	プロセス起動時の出力先に出力されます。
標準エラー出力の出力先	

ジョブ実行中にスプールジョブディレクトリが作成されますが，ジョブ実行後に削除されます。その際，スプールジョブディレクトリ内に割り当てたプログラム出力データファイルも削除されます。そのため，スプールジョブディレクトリのリネーム成功を示す KNAX6380-I メッセージは出力されません。

ジョブ実行後，JOBLOG の内容が標準エラー出力に出力されます。ただし，次に示すヘッダ行は出力されません。

```
-----------------------------------------------------------------
 Advanced Shell バージョン番号

 [ジョブ情報]
   ジョブ識別子                    : ジョブ識別子
   スプールジョブディレクトリパス  : スプールジョブディレクトリパス
   実行日付                        : 実行日付
   システム環境ファイルパス        : 環境ファイルパス（システム環境ファイル）
   ジョブ環境ファイルパス          : 環境ファイルパス（ジョブ環境ファイル）
   ホスト名                        : ホスト名
 [Automatic Job Management Systemから渡された環境変数]
   JP1/AJSから渡された環境変数
-----------------------------------------------------------------
********   ジョブコントローラのメッセージ出力   ********
```

JOBLOG 以外のジョブ実行ログは出力されないため，JP1/AJS やログインシェルなどからジョブ実行ログは参照できません。

なお，ジョブの終了コードは親プロセスのジョブの JOBLOG へ出力されるため，adshexec コマンドの実行終了を示す KNAX7999-I メッセージは，標準エラー出力には出力されません。

(b) 簡潔出力モードを選択した場合

簡潔出力モードを選択した場合のジョブ実行ログの出力先を次に示します。

メッセージの出力先	内容
JOBLOG	ルートジョブのスプール内の子孫ジョブごとのファイルに出力されます。
スクリプトイメージ	一時的にスプール内のファイルに出力されますが，子孫ジョブ終了時に削除されます。
標準出力の出力先	プロセス起動時の出力先に出力されます。
標準エラー出力の出力先	

メッセージの出力先	内容
標準エラー出力の出力先	標準エラー出力のエラーメッセージと，標準出力のエラーメッセージが出力されます。また，JOBLOG のエラーメッセージが標準エラー出力に出力されます。 エラーメッセージ以外のメッセージは出力されません。 メッセージ種別が W および I のメッセージ（シグナル受信，イベント受信メッセージは除く）は出力抑止されます。

ジョブ実行中，JOBLOG にメッセージを出力します。このうち，エラーメッセージは標準エラー出力にも出力されます。

ジョブ実行ログは，ルートジョブのスプールジョブディレクトリ下に，子孫ジョブごとにファイル出力されます。ジョブの実行後もジョブ実行ログは標準エラー出力へ出力されません。

(c)　最小出力モードを選択した場合

最小出力モードを選択した場合のジョブ実行ログの出力先を次に示します。

メッセージの出力先	内容
JOBLOG	ルートジョブのスプール内の子孫ジョブごとのファイルに，出力抑止対象外のメッセージが出力されます。
スクリプトイメージ	一時的にスプール内のファイルに出力されますが，子孫ジョブ終了時に削除されます。
標準出力の出力先	プロセス起動時の出力先に出力されます。 標準エラー出力と標準出力の出力抑止対象外のメッセージが出力されます。また，JOBLOG の出力抑止対象外のメッセージが標準エラー出力に出力されます。 出力抑止したメッセージは出力されません。
標準エラー出力の出力先	

出力が抑止されるメッセージは次のとおりです。ただし，出力モードにかかわらず，出力する例外メッセージもあります。この例外メッセージは「12.2　メッセージの出力先」を参照してください。

- メッセージ種別がW および I のメッセージ（KNAX7893-I およびKNAX7896-I 以外のシグナル受信，イベント受信メッセージは除く）
- メッセージ種別が E の次に示すメッセージ

  KNAX0101-E，KNAX2201-E，KNAX6521-E，KNAX6522-E，KNAX6541-E，KNAX6542-E，KNAX6551-E，KNAX6552-E，KNAX6561-E，KNAX6562-E，KNAX6586-E，KNAX6591-E，KNAX6592-E，KNAX6593-E，KNAX6594-E，KNAX6596-E
- メッセージ種別がI の次に示すメッセージ

  KNAX7893-I，KNAX7896-I

(3) 子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージした場合

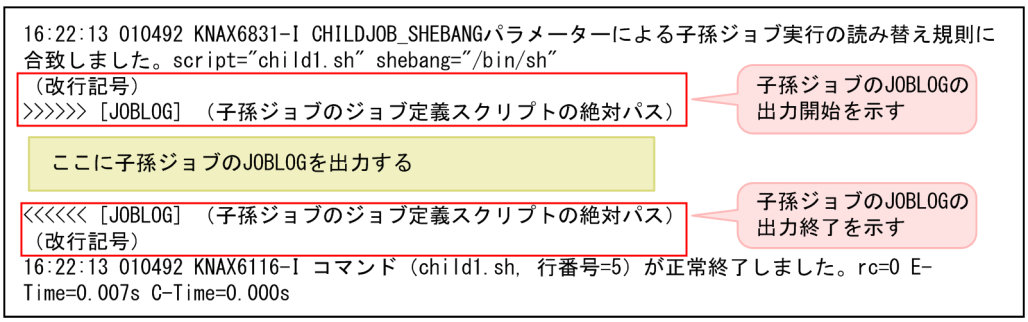
SPOOLJOB_CHILDJOB パラメーターに MERGE を指定した場合，子孫ジョブのスプールジョブは，ルートジョブのスプールジョブへマージされます。その場合のジョブ実行ログの出力内容の概要を次に示します。ジョブ実行ログの全体の出力例は，「3.5.3 ジョブ実行ログの出力例（子孫ジョブのスプールジョブを削除した場合）」を参照してください。

(a) 通常実行時

- JOBLOG

  子孫ジョブの JOBLOG は，子孫ジョブ実行の読み替え規則に合致した旨のメッセージと，子孫ジョブ実行コマンドの終了メッセージとの間に出力されます。

  出力される JOBLOG の前後に，次の図のように子孫ジョブの JOBLOG 出力開始と出力終了を示す記号を出力します。



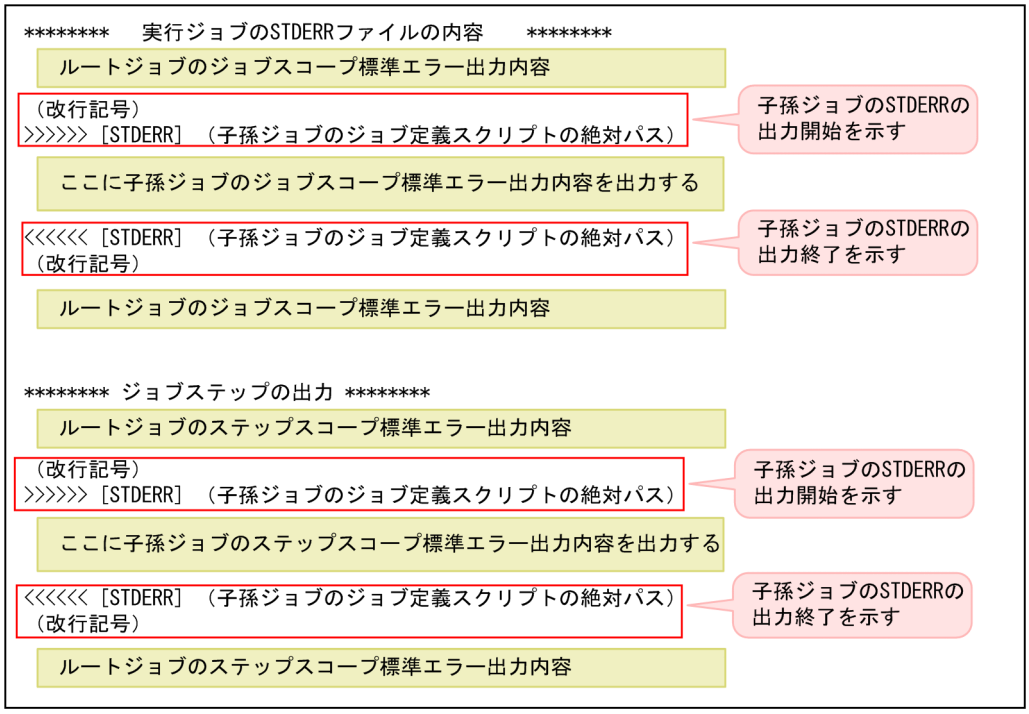
- SCRIPT

  ルートジョブの SCRIPT の下に，子孫ジョブの SCRIPT を出力します。子孫ジョブの出力部分には，ヘッダ「******** ジョブ定義スクリプトの内容 ********」は出力しません。

- STDERR

  子孫ジョブの標準エラー出力の前後に，次の図のように子孫ジョブの STDERR 出力開始と出力終了を示す記号を出力します。

  この記号は，子孫ジョブが簡潔出力モードまたは最小出力モードの場合は出力しません。



- STDOUT

  STDOUT はマージされません。

(b) デバッグ実行時

- JOBLOG

  通常実行時と同じ形式でマージされます。なお，子孫ジョブの終了直後に，標準エラー出力にも同じ内容が出力されます。

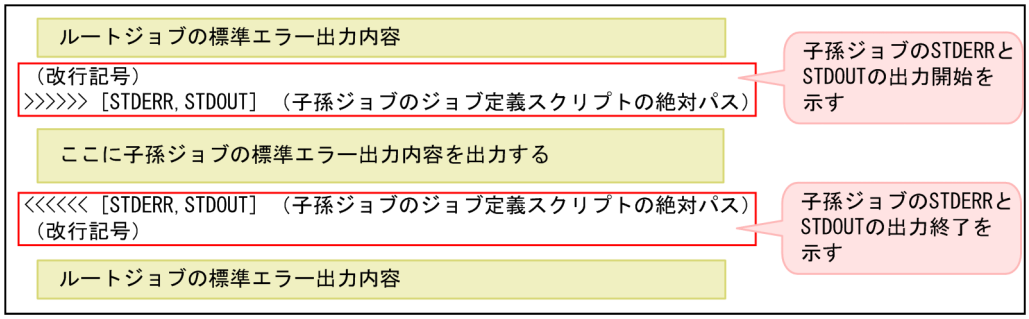
- SCRIPT

  通常実行時と同じ形式でマージされます。なお，SCRIPT の内容は標準エラー出力には出力されません。

- STDERR，STDOUT

  子孫ジョブの標準エラー出力の前後に，次の図のように子孫ジョブの STDERR と STDOUT を合わせたものの出力開始と出力終了を示す記号を出力します。子孫ジョブの標準エラー出力や標準出力をリダイレクトしている場合でも，[STDERR,STDOUT]という記号は標準エラー出力に出力されます。

  この記号は，子孫ジョブが簡潔出力モードまたは最小出力モードの場合は出力しません。



(c) ルートジョブと子孫ジョブで出力モードが異なる場合

SPOOLJOB_CHILDJOB 環境設定パラメーターに MERGE を指定し，かつルートジョブが拡張出力モード，子孫ジョブが最小出力モードで動作する場合，JOBLOG と SCRIPT のマージ結果は次のようになります。

- JOBLOG
  - 最小出力モードで動作する子孫ジョブの JOBLOG に，メッセージが出力されていない場合

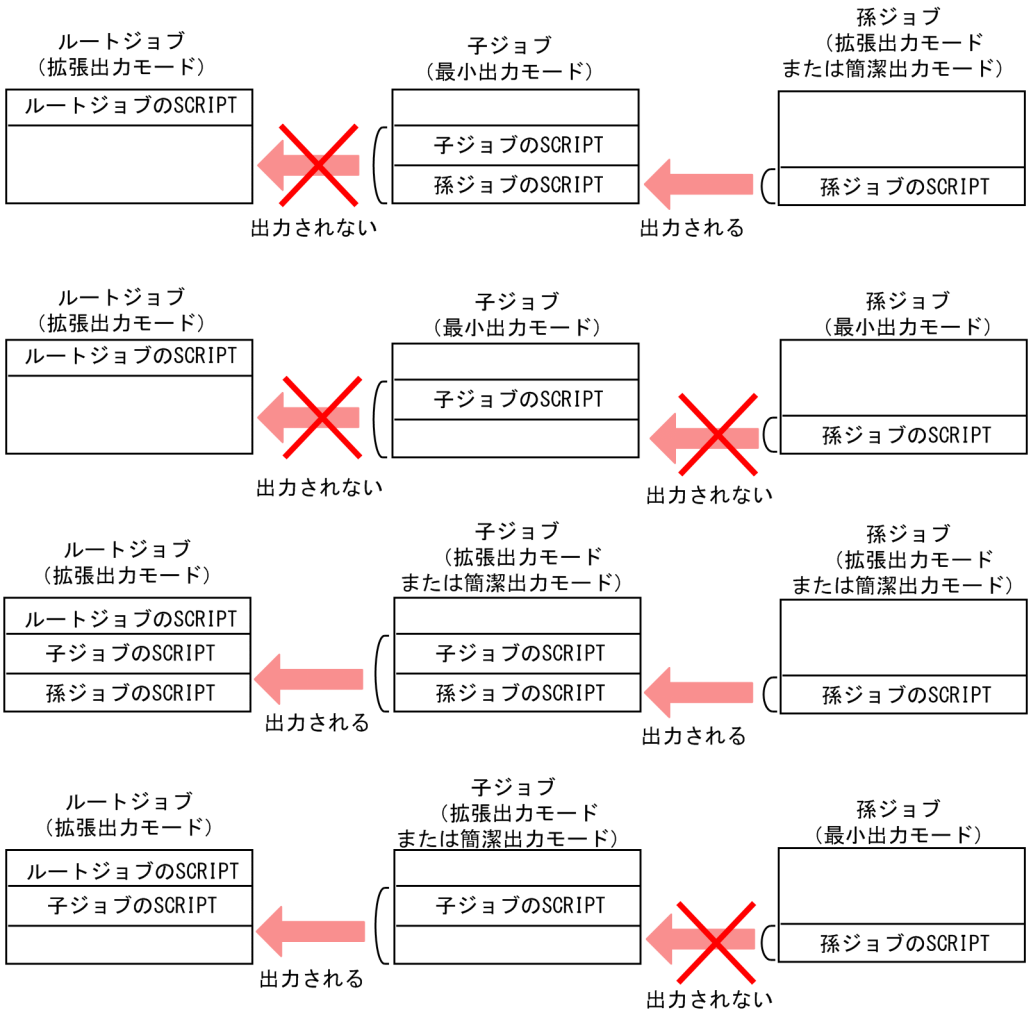
    この子孫ジョブの JOBLOG はルートジョブの JOBLOG にマージされません。そのため，子孫ジョブの JOBLOG の出力開始と出力終了を示す記号も出力されません。
  - 最小出力モードで動作する子孫ジョブの JOBLOG に，メッセージが出力されている場合

    この子孫ジョブの JOBLOG はルートジョブの JOBLOG にマージされます。また，子孫ジョブの JOBLOG の出力開始と出力終了を示す記号も出力されます。

  子孫ジョブの JOBLOG の出力開始と出力終了を示す記号は，通常実行時・デバッグ実行時とも ">>>>>> [JOBLOG] パス名"と"<<<<<< [JOBLOG] パス名"です。
- SCRIPT

  子孫ジョブの SCRIPT は，ルートジョブの SCRIPT にマージされません。

  また，子ジョブが最小出力モードで，そこから起動する孫ジョブが拡張出力モードまたは簡潔出力モードの場合も，子ジョブの SCRIPT がマージされないため，子ジョブの SCRIPT にマージされた孫ジョブの SCRIPT も出力されません。



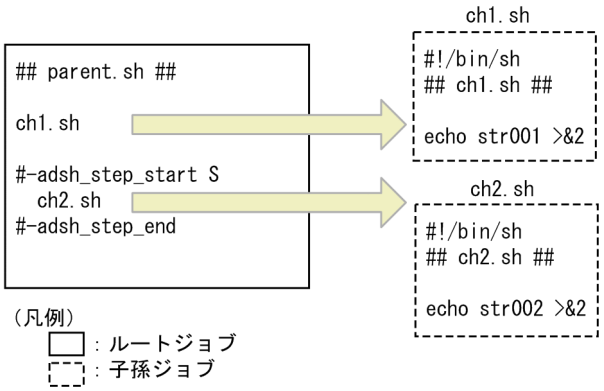
ルートジョブが簡潔出力モードまたは最小出力モードの場合も同様です。ただし，その場合は標準エラー出力に SCRIPT を出力しません。

3.5.2　ジョブ実行ログの出力例（子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージした場合）

環境設定パラメーター SPOOLJOB_CHILDJOB で MERGE を選択（子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージ）した場合の，ジョブ実行ログの出力例を次に示します。

(1)　例 1（ch1.sh と ch2.sh を定義）

次のように，ルートジョブから起動する子孫ジョブ ch1.sh と，ルートジョブのジョブステップ内から起動する子孫ジョブ ch2.sh を定義した場合について説明します。



この場合に標準エラー出力へ出力されるルートジョブのジョブ実行ログについて，構成と出力例を次に示します。

(a)　ジョブ実行ログの構成

ジョブ実行ログの構成と，子孫ジョブ ch1.sh と ch2.sh の実行結果の出力個所を次に示します。子孫ジョブの実行結果が JOBLOG と SCRIPT にも出力されます。

```
ジョブID表示
（環境ファイル解析エラーメッセージなど）
----------------------------------------
ジョブ実行ログヘッダ
----------------------------------------

  JOBLOG
  コマンド実行結果など

    JOBLOG（ch1.shのもの）
    ch1.shのコマンド実行結果など

    JOBLOG（ch2.shのもの）
    ch2.shのコマンド実行結果など

  SCRIPT
  スクリプトイメージ

    SCRIPT（ch1.shのもの）
    ch1.shのスクリプトイメージ

    SCRIPT（ch2.shのもの）
    ch2.shのスクリプトイメージ

  JOB STDERR
  ジョブスコープの標準エラー出力

    ch1.shの標準エラー出力

  STEP STDERR
  ステップスコープの標準エラー出力

    ch2.shの標準エラー出力

スプールジョブディレクトリ名称変更メッセージ
ルートジョブのジョブコントローラがバッチジョ
ブを終了しました。rc=XXX
```

(b)　ジョブ実行ログの出力例

ジョブ実行ログの出力例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000074
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                    : 000074
   スプールジョブディレクトリパス  : /var/opt/jp1as/spool/000074/
   実行日付                        : 2015/10/30
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          : /home/usr/adsh_job.ase
   ホスト名                        : HOST01
 [Automatic Job Management Systemから渡された環境変数]
   JP1JobName       : adshexec
   JP1JobID         : 110
   JP1_USERNAME     : jp1admin
   JP1UNCName       : HOST01
   JP1NBQSQueueName: ¥¥HOST01¥@SYSTEM
   JP1Priority      : 1
   AJSEXECID        : @A116
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
18:48:50 000074 KNAX0091-I ADSH000074 ジョブが開始しました。
18:48:50 000074 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
18:48:50 000074 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:48:50 000074 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合
致しました。script="/home/usr/ch1.sh" shebang="/bin/sh"

>>>>>> [JOBLOG] /home/usr/ch1.sh
18:48:50 000075 KNAX6571-I ADSH000075 子孫ジョブを開始しました。parent jobname=ADSH000074,
parent jobid=000074
18:48:50 000075 KNAX6572-I 子孫ジョブ（ADSH000075）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
18:48:50 000075 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:48:50 000075 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:48:50 000075 KNAX6578-I ADSH000075 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
<<<<<< [JOBLOG] /home/usr/ch1.sh

18:48:50 000074 KNAX6116-I コマンド（/home/usr/ch1.sh，行番号=2）が正常終了しました。rc=0 E-
Time=0.012s C-Time=0.000s
18:48:50 000074 KNAX0092-I ADSH000074.S ステップが開始しました。
18:48:50 000074 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合
致しました。script="/home/usr/ch2.sh" shebang="/bin/sh"
```

ch1.sh出力

（次の図に続く）

（前の図の続き）

ch2.sh出力

```
>>>>>> [JOBLOG] /home/usr/ch2.sh
18:48:50 000076 KNAX6571-I ADSH000076 子孫ジョブを開始しました。parent jobname=ADSH000074,
parent jobid=000074
18:48:50 000076 KNAX6572-I 子孫ジョブ（ADSH000076）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
18:48:50 000076 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:48:50 000076 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:48:50 000076 KNAX6578-I ADSH000076 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
<<<<<< [JOBLOG] /home/usr/ch2.sh

18:48:50 000074 KNAX6116-I コマンド（/home/usr/ch2.sh，行番号=5）が正常終了しました。rc=0 E-
Time=0.011s C-Time=0.020s
18:48:50 000074 KNAX6597-I ADSH000074.S ジョブステップが正常終了しました。rc=0 E-Time=0.011s
C-Time=0.020s
18:48:50 000074 KNAX0098-I ADSH000074 ジョブが終了しました。rc=0 E-Time=0.026s C-Time=0.020s

********   ジョブ定義スクリプトの内容    ********

***** /home/usr/parent.sh *****
0001 : ## parent.sh ##
0002 : ch1.sh
0003 :
0004 : #-adsh_step_start S
0005 :   ch2.sh
0006 : #-adsh_step_end

***** パス変換情報 *****
```

ch1.sh出力

```
***** /home/usr/ch1.sh *****
0001 : #!/bin/sh
0002 : ## ch1.sh ##
0003 :
0004 : echo str001 >&2

***** パス変換情報 *****
```

ch2.sh出力

```
***** /home/usr/ch2.sh *****
0001 : #!/bin/sh
0002 : ## ch2.sh ##
0003 :
0004 : echo str002 >&2

***** パス変換情報 *****
```

（次の図に続く）

（前の図の続き）

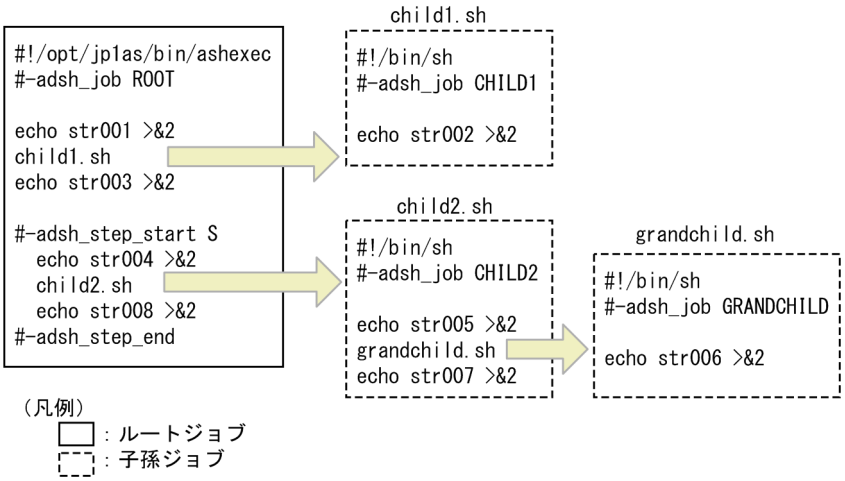
```
********    実行ジョブのSTDERRファイルの内容      ********
>>>>>> [STDERR] /home/usr/ch1.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000075
str001
<<<<<< [STDERR] /home/usr/ch1.sh

KNAX6597-I ADSH000074.S ジョブステップが正常終了しました。rc=0 E-Time=0.011s C-Time=0.020s
KNAX0098-I ADSH000074 ジョブが終了しました。rc=0 E-Time=0.026s C-Time=0.020s

******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=S 出力先=STDERR
>>>>>> [STDERR] /home/usr/ch2.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000076
str002
<<<<<< [STDERR] /home/usr/ch2.sh


KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/000074-ADSH000074/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

ch1.sh出力

ch2.sh出力

(2) 例 2（child1.sh，child2.sh，grandchild.sh を定義）

次のように子孫ジョブ child1.sh，child2.sh，さらに子孫ジョブ child2.sh から起動する grandchild.sh を定義した場合について説明します。

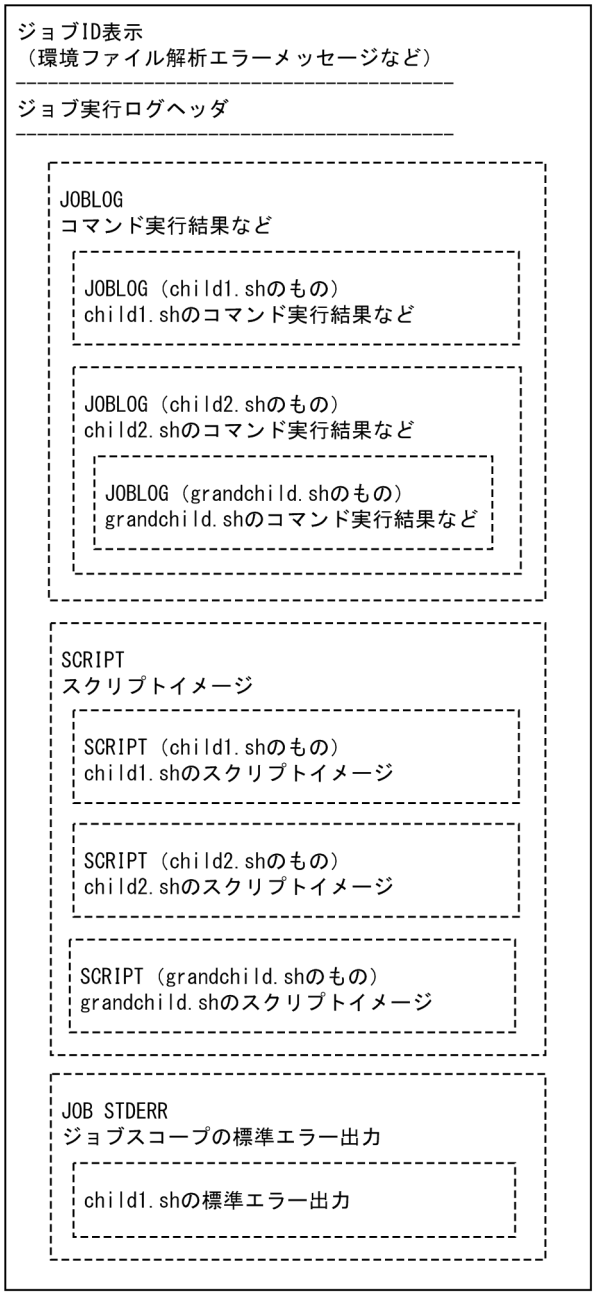


この場合に標準エラー出力へ出力されるルートジョブのジョブ実行ログについて，構成と出力例を次に示します。

(a) ジョブ実行ログの構成

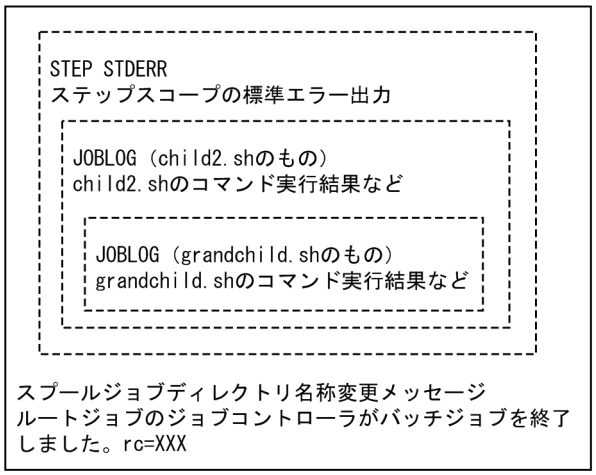
ジョブ実行ログの構成と，子孫ジョブ child1.sh，child2.sh，grandchild.sh の実行結果の出力個所を次に示します。子孫ジョブの実行結果が JOBLOG と SCRIPT にも出力されます。

- JP1/AJS などから参照できるジョブ実行ログ

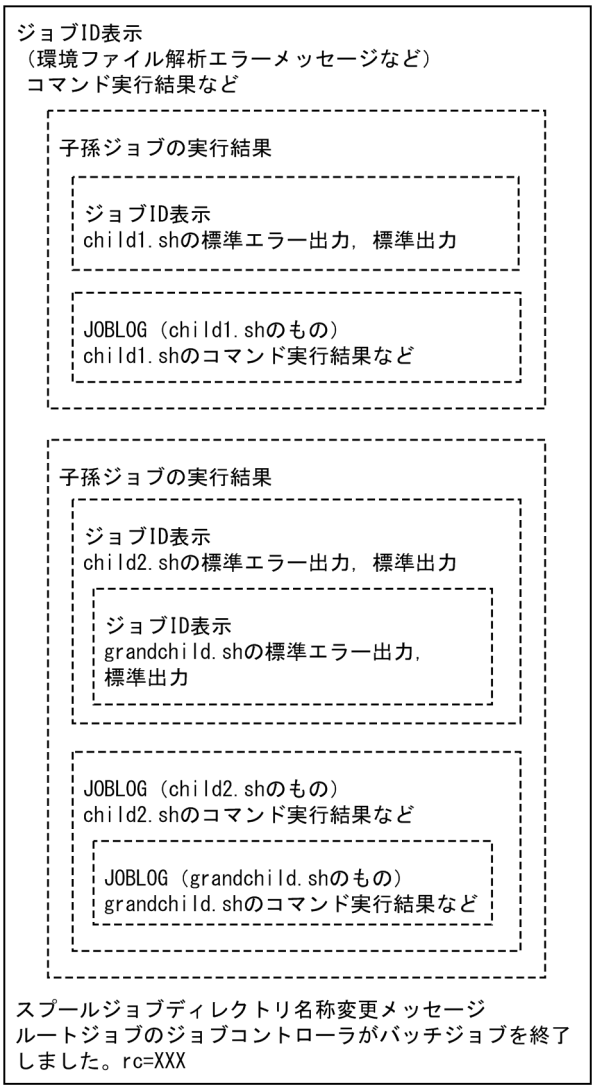


（次の図に続く）

（前の図の続き）



- デバッグ実行時のジョブ実行ログ



(b) ジョブ実行ログの出力例

ジョブ実行ログの出力例を次に示します。

- JP1/AJS などから参照できるジョブ実行ログ

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000078
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                    : 000078
   スプールジョブディレクトリパス  : /var/opt/jp1as/spool/000078/
   実行日付                        : 2015/10/30
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          : /home/usr/adsh_job.ase
   ホスト名                        : HOST01
 [Automatic Job Management Systemから渡された環境変数]
   JP1JobName       : adshexec
   JP1JobID         : 112
   JP1_USERNAME     : jp1admin
   JP1UNCName       : HOST01
   JP1NBQSQueueName: ¥¥HOST01¥@SYSTEM
   JP1Priority      : 1
   AJSEXECID        : @A118
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
18:55:40 000078 KNAX0091-I ROOT ジョブが開始しました。
18:55:40 000078 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの
完了を待ちます。
18:55:40 000078 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:55:40 000078 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000078 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に
合致しました。script="/home/usr/child1.sh" shebang="/bin/sh"
```

child1.sh出力

```
>>>>>> [JOBLOG] /home/usr/child1.sh
18:55:40 000079 KNAX6571-I CHILD1 子孫ジョブを開始しました。parent jobname=ROOT，parent
jobid=000078
18:55:40 000079 KNAX6572-I 子孫ジョブ（CHILD1）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
18:55:40 000079 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:55:40 000079 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000079 KNAX6578-I CHILD1 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-Time=0.000s
<<<<<< [JOBLOG] /home/usr/child1.sh

18:55:40 000078 KNAX6116-I コマンド（/home/usr/child1.sh，行番号=5）が正常終了しました。rc=0
E-Time=0.017s C-Time=0.010s
18:55:40 000078 KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000078 KNAX0092-I ROOT.S ステップが開始しました。
18:55:40 000078 KNAX6112-I コマンド（echo，行番号=9）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000078 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に
合致しました。script="/home/usr/child2.sh" shebang="/bin/sh"
```

（次の図に続く）

（前の図の続き）

child2.sh出力

```
>>>>>> [JOBLOG] /home/usr/child2.sh
18:55:40 000080 KNAX6571-I CHILD2 子孫ジョブを開始しました。parent jobname=ROOT, parent
jobid=000078
18:55:40 000080 KNAX6572-I 子孫ジョブ（CHILD2）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
18:55:40 000080 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:55:40 000080 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000080 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に
合致しました。script="/home/usr/grandchild.sh" shebang="/bin/sh"
```

grandchild.sh出力

```
>>>>>> [JOBLOG] /home/usr/grandchild.sh
18:55:40 000081 KNAX6571-I GRANDCHILD 子孫ジョブを開始しました。parent jobname=CHILD2,
parent jobid=000080
18:55:40 000081 KNAX6572-I 子孫ジョブ（GRANDCHILD）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
18:55:40 000081 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
18:55:40 000081 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000081 KNAX6578-I GRANDCHILD 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
<<<<<< [JOBLOG] /home/usr/grandchild.sh
```

```
18:55:40 000080 KNAX6116-I コマンド（/home/usr/grandchild.sh，行番号=5）が正常終了しました。
rc=0 E-Time=0.012s C-Time=0.000s
18:55:40 000080 KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000080 KNAX6578-I CHILD2 子孫ジョブが終了しました。rc=0 E-Time=0.014s C-Time=0.000s
<<<<<< [JOBLOG] /home/usr/child2.sh
```

child2.sh出力

```
18:55:40 000078 KNAX6116-I コマンド（/home/usr/child2.sh，行番号=10）が正常終了しました。
rc=0 E-Time=0.025s C-Time=0.020s
18:55:40 000078 KNAX6112-I コマンド（echo，行番号=11）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
18:55:40 000078 KNAX6597-I ROOT.S ジョブステップが正常終了しました。rc=0 E-Time=0.026s C-
Time=0.020s
18:55:40 000078 KNAX0098-I ROOT ジョブが終了しました。rc=0 E-Time=0.046s C-Time=0.030s

********    ジョブ定義スクリプトの内容    ********

***** /home/usr/parent.sh *****
0001 : #!/opt/jp1as/bin/adshexec
0002 : #-adsh_job ROOT
0003 :
0004 : echo str001 >&2
0005 : child1.sh
0006 : echo str003 >&2
0007 :
0008 : #-adsh_step_start S
0009 :   echo str004 >&2
0010 :   child2.sh
0011 :   echo str008 >&2
0012 : #-adsh_step_end
```

（次の図に続く）

（前の図の続き）

```
***** パス変換情報 *****

***** /home/usr/child1.sh *****                                  child1.sh出力
0001 : #!/bin/sh
0002 : #-adsh_job CHILD1
0003 :
0004 : echo str002 >&2

***** パス変換情報 *****

***** /home/usr/child2.sh *****                                  child2.sh出力
0001 : #!/bin/sh
0002 : #-adsh_job CHILD2
0003 :
0004 : echo str005 >&2
0005 : grandchild.sh
0006 : echo str007 >&2

***** パス変換情報 *****

***** /home/usr/grandchild.sh *****                              grandchild.sh出力
0001 : #!/bin/sh
0002 : #-adsh_job GRANDCHILD
0003 :
0004 : echo str006 >&2

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容    ********
str001
                                                                child1.sh出力
>>>>>> [STDERR] /home/usr/child1.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000079
str002
<<<<<< [STDERR] /home/usr/child1.sh

str003
KNAX6597-I ROOT.S ジョブステップが正常終了しました。rc=0 E-Time=0.026s C-Time=0.020s
KNAX0098-I ROOT ジョブが終了しました。rc=0 E-Time=0.046s C-Time=0.030s
```

（次の図に続く）

（前の図の続き）

```
******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=S 出力先=STDERR
str004
                                                                child2.sh出力
>>>>>> [STDERR] /home/usr/child2.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000080
str005
                                                                grandchild.sh出力
>>>>>> [STDERR] /home/usr/grandchild.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=000081
str006
<<<<<< [STDERR] /home/usr/grandchild.sh

str007
<<<<<< [STDERR] /home/usr/child2.sh

str008

KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/000078-ROOT/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

- デバッグ実行時のジョブ実行ログ

```
$ adshexec -d parent.sh
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=010505
(adshdb) run
KNAX7007-I ジョブ定義スクリプトの実行を開始します。: /home/usr/parent.sh

KNAX0724-I ジョブ識別子を割り当てました。Jobid=010506
KNAX0091-I ROOT ジョブが開始しました。
KNAX7902-I ジョブコントローラは，"端末入力モード"で動作します。
str001
KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合致しました。
script="child1.sh" shebang="/bin/sh"

>>>>>> [STDERR,STDOUT] /home/usr/child1.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=010507
str002
<<<<<< [STDERR,STDOUT] /home/usr/child1.sh

>>>>>> [JOBLOG] /home/usr/child1.sh
16:49:27 010507 KNAX6571-I CHILD1 子孫ジョブを開始しました。parent jobname=ROOT，parent
jobid=010506
16:49:27 010507 KNAX6572-I 子孫ジョブ（CHILD1）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
16:49:27 010507 KNAX7902-I ジョブコントローラは，"端末入力モード"で動作します。
16:49:27 010507 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
16:49:27 010507 KNAX6578-I CHILD1 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-Time=0.000s
<<<<<< [JOBLOG] /home/usr/child1.sh

KNAX6116-I コマンド（child1.sh，行番号=5）が正常終了しました。rc=0 E-Time=0.008s C-
Time=0.000s
str003
KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
KNAX0092-I ROOT.S ステップが開始しました。
str004
KNAX6112-I コマンド（echo，行番号=9）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合致しました。
script="child2.sh" shebang="/bin/sh"

>>>>>> [STDERR,STDOUT] /home/usr/child2.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=010508
str005

>>>>>> [STDERR,STDOUT] /home/usr/grandchild.sh
KNAX0726-I 子孫ジョブのジョブ識別子を割り当てました。Jobid=010509
str006
<<<<<< [STDERR,STDOUT] /home/usr/grandchild.sh

str007
<<<<<< [STDERR,STDOUT] /home/usr/child2.sh
```

child1.sh出力

child1.sh出力

child2.sh出力

grandchild.sh出力

（次の図に続く）

（前の図の続き）

child2.sh出力

```
>>>>>> [JOBLOG] /home/usr/child2.sh
16:49:28 010508 KNAX6571-I CHILD2 子孫ジョブを開始しました。parent jobname=ROOT, parent
jobid=010506
16:49:28 010508 KNAX6572-I 子孫ジョブ（CHILD2）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
16:49:28 010508 KNAX7902-I ジョブコントローラは，"端末入力モード"で動作します。
16:49:28 010508 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
16:49:28 010508 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に
合致しました。script="grandchild.sh" shebang="/bin/sh"
```

grandchild.sh出力

```
>>>>>> [JOBLOG] /home/usr/grandchild.sh
16:49:28 010509 KNAX6571-I GRANDCHILD 子孫ジョブを開始しました。parent jobname=CHILD2,
parent jobid=010508
16:49:28 010509 KNAX6572-I 子孫ジョブ（GRANDCHILD）はジョブ環境ファイル（"/home/usr/
adsh_job.ase"）を使用します。
16:49:28 010509 KNAX7902-I ジョブコントローラは，"端末入力モード"で動作します。
16:49:28 010509 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
16:49:28 010509 KNAX6578-I GRANDCHILD 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
<<<<<< [JOBLOG] /home/usr/grandchild.sh
```

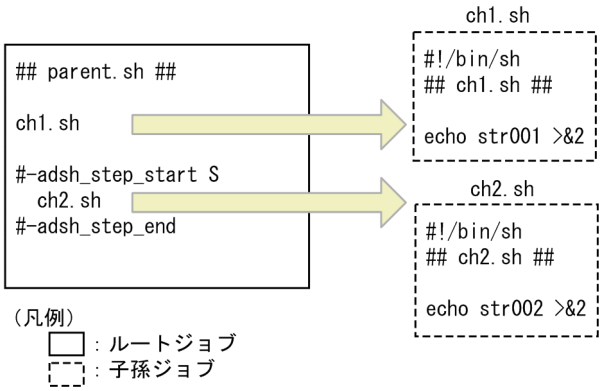
```
16:49:28 010508 KNAX6116-I コマンド（grandchild.sh，行番号=5）が正常終了しました。rc=0 E-
Time=0.006s C-Time=0.000s
16:49:28 010508 KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
16:49:28 010508 KNAX6578-I CHILD2 子孫ジョブが終了しました。rc=0 E-Time=0.008s C-Time=0.010s
<<<<<< [JOBLOG] /home/usr/child2.sh

KNAX6116-I コマンド（child2.sh，行番号=10）が正常終了しました。rc=0 E-Time=0.016s C-
Time=0.020s
str008
KNAX6112-I コマンド（echo，行番号=11）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
KNAX6597-I ROOT.S ジョブステップが正常終了しました。rc=0 E-Time=0.018s C-Time=0.020s
KNAX0098-I ROOT ジョブが終了しました。rc=0 E-Time=0.030s C-Time=0.020s
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/010506-ROOT/"

(adshdb) quit
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/010505-ROOT/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

3.5.3　ジョブ実行ログの出力例（子孫ジョブのスプールジョブを削除した場合）

ルートジョブと子孫ジョブを実行した場合の，ジョブ実行ログの出力例を次に示します。

(1)　例 1（ch1.sh と ch2.sh を定義）

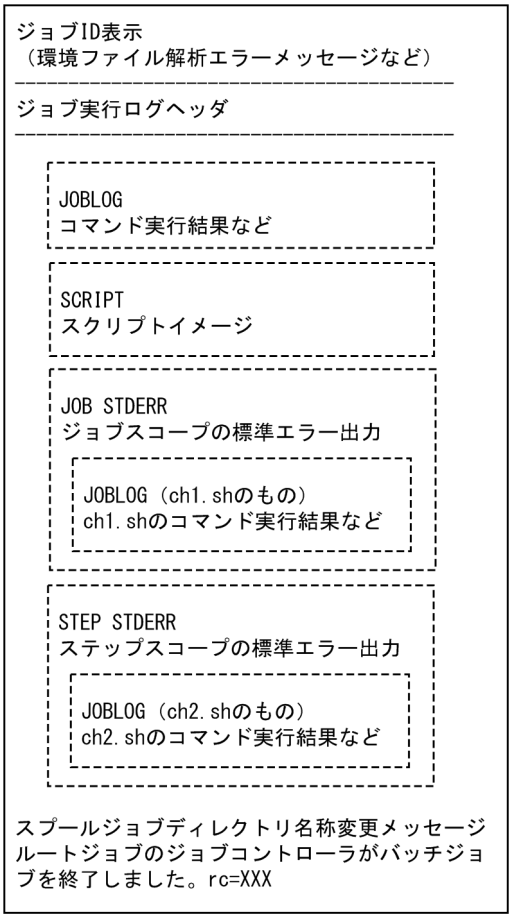
次のように，ルートジョブから起動する子孫ジョブ ch1.sh と，ルートジョブのジョブステップ内から起動する子孫ジョブ ch2.sh を定義した場合について説明します。



この場合に標準エラー出力へ出力されるルートジョブのジョブ実行ログについて，構成と出力例を次に示します。

(a)　ジョブ実行ログの構成

ジョブ実行ログの構成と，子孫ジョブ ch1.sh と ch2.sh の実行結果の出力個所を次に示します。ch1.sh の実行結果はジョブスコープ内に，ch2.sh の実行結果はステップスコープ内に出力されます。



(b)　ジョブ実行ログの出力例

ジョブ実行ログの出力例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000100
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                    : 000100
   スプールジョブディレクトリパス  : /var/opt/jp1as/spool/000100/
   実行日付                        : 2015/10/30
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          : /home/usr/adsh_job.ase
   ホスト名                        : HOST01
 [Automatic Job Management Systemから渡された環境変数]
   JP1JobName      : adshexec
   JP1JobID        : 121
   JP1_USERNAME    : jp1admin
   JP1UNCName      : HOST01
   JP1NBQSQueueName: ¥¥HOST01¥@SYSTEM
   JP1Priority     : 1
   AJSEXECID       : @A127
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
19:17:27 000100 KNAX0091-I ADSH000100 ジョブが開始しました。
19:17:27 000100 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの
完了を待ちます。
19:17:27 000100 KNAX7902-I ジョブコントローラは，″非端末入力モード″で動作します。
19:17:27 000100 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に
合致しました。script=″/home/usr/ch1.sh″ shebang=″/bin/sh″
19:17:27 000100 KNAX6116-I コマンド（/home/usr/ch1.sh，行番号=2）が正常終了しました。rc=0 E-
Time=0.011s C-Time=0.010s
19:17:27 000100 KNAX0092-I ADSH000100.S ステップが開始しました。
19:17:27 000100 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に
合致しました。script=″/home/usr/ch2.sh″ shebang=″/bin/sh″
19:17:27 000100 KNAX6116-I コマンド（/home/usr/ch2.sh，行番号=5）が正常終了しました。rc=0 E-
Time=0.028s C-Time=0.010s
19:17:27 000100 KNAX6597-I ADSH000100.S ジョブステップが正常終了しました。rc=0 E-Time=0.029s
C-Time=0.010s
19:17:27 000100 KNAX0098-I ADSH000100 ジョブが終了しました。rc=0 E-Time=0.043s C-Time=0.030s
```

（次の図に続く）

（前の図の続き）

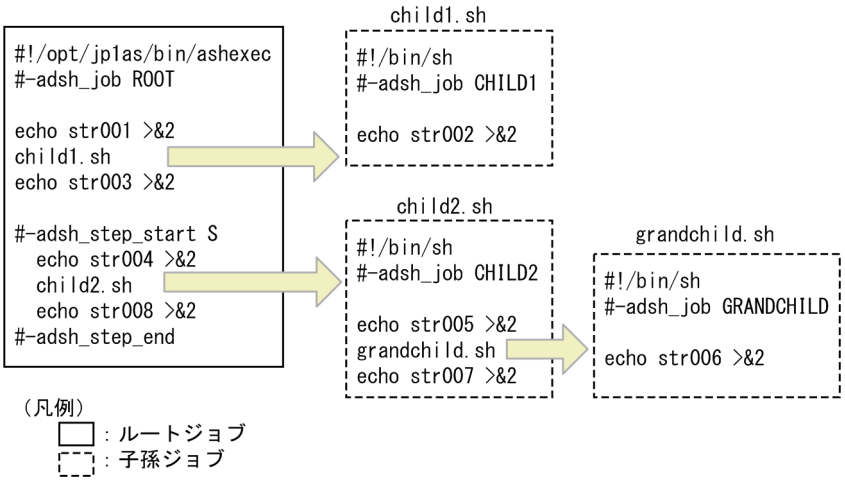
```
********    ジョブ定義スクリプトの内容      ********

***** /home/usr/parent.sh *****
0001 : ## parent.sh ##
0002 : ch1.sh
0003 :
0004 : #-adsh_step_start S
0005 :   ch2.sh
0006 : #-adsh_step_end

***** パス変換情報 *****

********    実行ジョブのSTDERRファイルの内容      ********
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000101
str001
19:17:27 000101 KNAX6571-I ADSH000101 子孫ジョブを開始しました。parent jobname=ADSH000100,
parent jobid=000100
19:17:27 000101 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
19:17:27 000101 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
19:17:27 000101 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:17:27 000101 KNAX6578-I ADSH000101 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
KNAX6597-I ADSH000100.S ジョブステップが正常終了しました。rc=0 E-Time=0.029s C-Time=0.010s
KNAX0098-I ADSH000100 ジョブが終了しました。rc=0 E-Time=0.043s C-Time=0.030s

******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=S 出力先=STDERR
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000102
str002
19:17:27 000102 KNAX6571-I ADSH000102 子孫ジョブを開始しました。parent jobname=ADSH000100,
parent jobid=000100
19:17:27 000102 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
19:17:27 000102 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
19:17:27 000102 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:17:27 000102 KNAX6578-I ADSH000102 子孫ジョブが終了しました。rc=0 E-Time=0.004s C-
Time=0.000s

KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/000100-ADSH000100/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

ch1.sh出力

ch2.sh出力

(2) 例 2（child1.sh，child2.sh，grandchild.sh を定義）

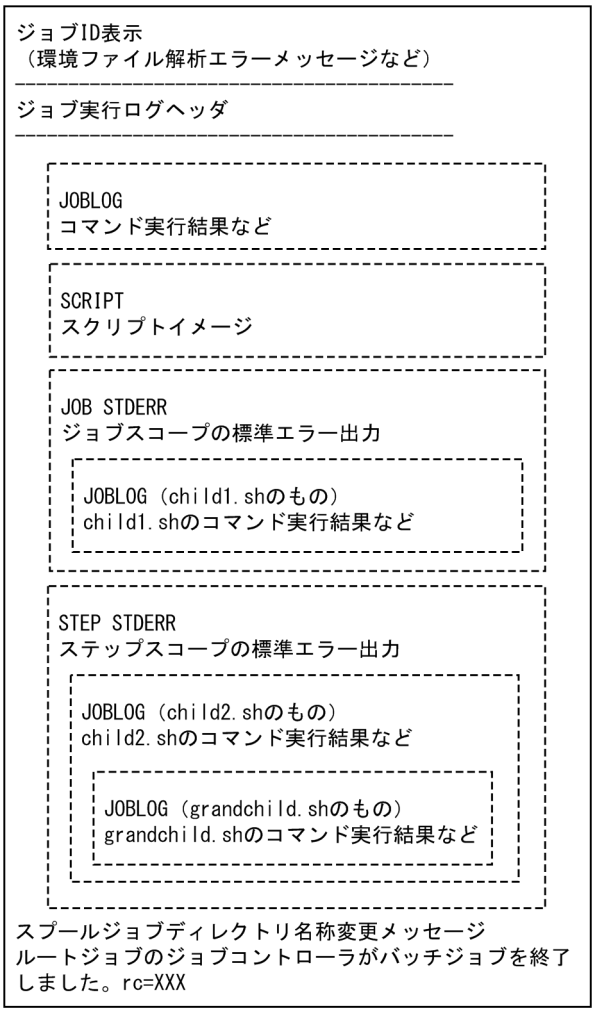
次のように子孫ジョブ child1.sh，child2.sh，さらに子孫ジョブ child2.sh から起動する grandchild.sh を定義した場合について説明します。



この場合に標準エラー出力へ出力されるルートジョブのジョブ実行ログについて，構成と出力例を次に示します。

(a)　ジョブ実行ログの構成

ジョブ実行ログの構成と，子孫ジョブ child1.sh，child2.sh，grandchild.sh の実行結果の出力個所を次に示します。grandchild.sh の実行結果は，子孫ジョブ child2.sh の実行結果の中に出力されます。



(b) ジョブ実行ログの出力例

ジョブ実行ログの出力例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000103
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                  : 000103
   スプールジョブディレクトリパス : /var/opt/jp1as/spool/000103/
   実行日付                      : 2015/10/30
   システム環境ファイルパス       :
   ジョブ環境ファイルパス         : /home/usr/adsh_job.ase
   ホスト名                      : HOST01
 [Automatic Job Management Systemから渡された環境変数]
   JP1JobName      : adshexec
   JP1JobID        : 122
   JP1_USERNAME    : jp1admin
   JP1UNCName      : HOST01
   JP1NBQSQueueName: ¥¥HOST01¥@SYSTEM
   JP1Priority     : 1
   AJSEXECID       : @A128
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
19:20:54 000103 KNAX0091-I ROOT ジョブが開始しました。
19:20:54 000103 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
19:20:54 000103 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
19:20:54 000103 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000103 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合
致しました。script="/home/usr/child1.sh" shebang="/bin/sh"
19:20:54 000103 KNAX6116-I コマンド（/home/usr/child1.sh，行番号=5）が正常終了しました。rc=0
E-Time=0.016s C-Time=0.000s
19:20:54 000103 KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000103 KNAX0092-I ROOT.S ステップが開始しました。
19:20:54 000103 KNAX6112-I コマンド（echo，行番号=9）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000103 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合
致しました。script="/home/usr/child2.sh" shebang="/bin/sh"
19:20:54 000103 KNAX6116-I コマンド（/home/usr/child2.sh，行番号=10）が正常終了しました。rc=0
E-Time=0.024s C-Time=0.030s
19:20:54 000103 KNAX6112-I コマンド（echo，行番号=11）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000103 KNAX6597-I ROOT.S ジョブステップが正常終了しました。rc=0 E-Time=0.025s C-
Time=0.040s
19:20:54 000103 KNAX0098-I ROOT ジョブが終了しました。rc=0 E-Time=0.045s C-Time=0.050s
```

（次の図に続く）

（前の図の続き）

```
********    ジョブ定義スクリプトの内容      ********

***** /home/usr/parent.sh *****
0001 : #!/opt/jp1as/bin/adshexec
0002 : #-adsh_job ROOT
0003 :
0004 : echo str001 >&2
0005 : child1.sh
0006 : echo str003 >&2
0007 :
0008 : #-adsh_step_start S
0009 :   echo str004 >&2
0010 :   child2.sh
0011 :   echo str008 >&2
0012 : #-adsh_step_end

***** パス変換情報 *****

********    実行ジョブのSTDERRファイルの内容      ********
str001
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000104
str002
19:20:54 000104 KNAX6571-I CHILD1 子孫ジョブを開始しました。parent jobname=ROOT, parent
jobid=000103
19:20:54 000104 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
19:20:54 000104 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
19:20:54 000104 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000104 KNAX6578-I CHILD1 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-Time=0.000s
str003
KNAX6597-I ROOT.S ジョブステップが正常終了しました。rc=0 E-Time=0.025s C-Time=0.040s
KNAX0098-I ROOT ジョブが終了しました。rc=0 E-Time=0.045s C-Time=0.050s
```

child1.sh出力

（次の図に続く）

（前の図の続き）

```
******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=S 出力先=STDERR
str004
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000105
str005
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000106
str006
19:20:54 000106 KNAX6571-I GRANDCHILD 子孫ジョブを開始しました。parent jobname=CHILD2, parent
jobid=000105
19:20:54 000106 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
19:20:54 000106 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
19:20:54 000106 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000106 KNAX6578-I GRANDCHILD 子孫ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
str007
19:20:54 000105 KNAX6571-I CHILD2 子孫ジョブを開始しました。parent jobname=ROOT, parent
jobid=000103
19:20:54 000105 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完
了を待ちます。
19:20:54 000105 KNAX7902-I ジョブコントローラは，"非端末入力モード"で動作します。
19:20:54 000105 KNAX6112-I コマンド（echo，行番号=4）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000105 KNAX6831-I CHILDJOB_SHEBANGパラメーターによる子孫ジョブ実行の読み替え規則に合
致しました。script="/home/usr/grandchild.sh" shebang="/bin/sh"
19:20:54 000105 KNAX6116-I コマンド（/home/usr/grandchild.sh，行番号=5）が正常終了しました。
rc=0 E-Time=0.011s C-Time=0.000s
19:20:54 000105 KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
19:20:54 000105 KNAX6578-I CHILD2 子孫ジョブが終了しました。rc=0 E-Time=0.014s C-Time=0.000s
str008

KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/000103-ROOT/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

child2.sh出力

grandchild.sh出力

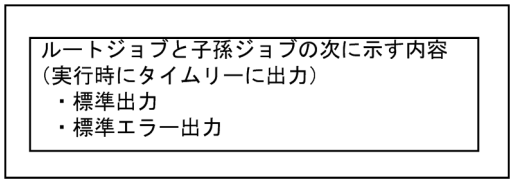
3.5.4 ジョブ実行ログの出力例（簡潔出力モードまたは最小出力モードを選択した場合）

環境設定パラメーター OUTPUT_MODE_ROOT または OUTPUT_MODE_CHILD で，簡潔出力モードまたは最小出力モードを選択した場合の，ジョブ実行ログの出力例を次に示します。なお，簡潔出力モードと最小出力モードでは，出力されるエラーメッセージが異なります。

(1) ジョブ実行ログの構成

ジョブ実行ログの構成を次に示します。

- JP1/AJS などから参照できるジョブ実行ログ



- デバッグ実行時のジョブ実行ログ

```
ジョブID表示
（環境ファイル解析エラーメッセージなど）

  ルートジョブと子孫ジョブの次に示す内容
  （実行時にタイムリーに出力）
    ・標準出力
    ・標準エラー出力
    ・ジョブステップ終了メッセージ
    ・ジョブ終了メッセージ
```

(2) ジョブ実行ログの出力例

ジョブ実行ログの出力例を次に示します。

- JP1/AJS などから参照できるジョブ実行ログ

■環境設定パラメーター

```
#-adsh_conf OUTPUT_MODE_ROOT SIMPLE
#-adsh_conf OUTPUT_MODE_CHILD SIMPLE
#-adsh_conf CHILDJOB_EXT ash
```

■ジョブ定義スクリプト：logroot.ash

```
#-adsh_job SampleJobRoot
#-adsh_file_temp WORK01
#-adsh_file_temp WORK02
./logsub.ash data tokyo 2>$WORK01
./logsub.ash data fukuoka 2>$WORK02
echo -E ”***WORK01*****” >&2
cat $WORK01 >&2
echo -E ”***WORK02*****” >&2
cat $WORK02 >&2
```

■ジョブ定義スクリプト：logsub.ash

```
#-adsh_job SampleSub
cat $1 | grep $2 >&2
```

■入力データ：data

```
aichi     nagoya      052
fukuoka   kurume      0942
fukushima iwaki       0246
tokyo     machida     042
tokyo     tachikawa   042
```

■実行例

```
[user001@hosta user001]$ /opt/jp1as/bin/adshexec logroot.ash
***WORK01*****
tokyo     machida     042
tokyo     tachikawa   042
***WORK02*****
fukuoka   kurume      0942
```

- デバッグ実行時のジョブ実行ログ

```
[user001@hosta user001]$ /opt/jp1as/bin/adshexec -d logroot.ash
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000837
(adshdb) list
1: #-adsh_job SampleJobRoot
2: #-adsh_file_temp WORK01
3: #-adsh_file_temp WORK02
4: ./logsub.ash data tokyo 2>$WORK01
5: ./logsub.ash data fukuoka 2>$WORK02
6: echo -E "***WORK01*****" >&2
7: cat $WORK01 >&2
8: echo -E "***WORK02*****" >&2
9: cat $WORK02 >&2
(adshdb) b
KNAX7018-I ブレークポイント "1": ファイル名="logroot.ash" 行番号=1
(adshdb) 4
KNAX7001-E 指定したコマンド（"4"）は存在しません。
(adshdb) run
KNAX7007-I ジョブ定義スクリプトの実行を開始します。: /home/user001/logroot.ash

KNAX0724-I ジョブ識別子を割り当てました。Jobid=000838

KNAX7018-I ブレークポイント "1": ファイル名="logroot.ash" 行番号=1
KNAX7032-I ジョブ定義スクリプトファイル（"logroot.ash"）の中で実行を停止しました。
1: #-adsh_job SampleJobRoot
現在位置: #-adsh_job SampleJobRoot
(adshdb) info b
Num  Type        What
1    breakpoint  logroot.ash:1
(adshdb) s
KNAX7032-I ジョブ定義スクリプトファイル（"logroot.ash"）の中で実行を停止しました。
2: #-adsh_file_temp WORK01
現在位置: #-adsh_file_temp WORK01
(adshdb) c
KNAX7034-I ジョブ定義スクリプトの実行を継続します。
***WORK01*****
tokyo     machida     042
tokyo     tachikawa   042
***WORK02*****
fukuoka   kurume      0942
KNAX0098-I SampleJobRoot ジョブが終了しました。rc=0 E-Time=15.428s C-Time=0.010s

(adshdb) quit
[user001@hosta user001]$
```

3.5.5　ジョブ実行ログの出力例（標準エラー出力だけを出力する場合）

環境設定パラメーター JOBEXECLOG_PRINT で STDERR を選択（ジョブ実行ログに標準エラー出力だけを出力する）した場合の，ジョブ実行ログの出力例を次に示します。

この例では，次に示すジョブ定義スクリプト「sample.ash」「samplesub1.ash」「samplesub2.ash」が定義されていることを前提としています。

- ジョブ定義スクリプト「sample.ash」の内容

```
#-adsh_job SAMPLEJOB
echo JOB_STDERR_001 >&2
cd /home/user001/dir
cd xxx
cd /home/user001
```

```
#-adsh_step_start S1 -run always -onError cont
echo STEP_STDERR_001 >&2
cd /home/user001/dir
cd xxx
cd /home/user001
#-adsh_step_end
echo JOB_STDERR_002 >&2
#-adsh_step_start S2 -run always -onError cont
echo STEP_STDERR_002 >&2
./samplesub1.ash
#-adsh_step_end
cd /home/user001/dir
cd xxx
cd /home/user001
echo JOB_STDERR_003 >&2
./samplesub2.ash
```

- ジョブ定義スクリプト「samplesub1.ash」の内容

```
#-adsh_job SAMPLE_SUB1
echo SUB1_JOB_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB1
cd /home/user001
#-adsh_step_start SUB1_S1 -run always -onError cont
echo SUB1_STEP_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB1
cd /home/user001
#-adsh_step_end
echo SUB1_JOB_STDERR_002 >&2
#-adsh_step_start SUB1_S2 -run always -onError cont
echo SUB1_STEP_STDERR_002 >&2
#-adsh_step_end
cd /home/user001/dir
cd xxxSUB1
cd /home/user001
echo SUB1_JOB_STDERR_003 >&2
```

- ジョブ定義スクリプト「samplesub2.ash」の内容

```
#-adsh_job SAMPLE_SUB2
echo SUB2_JOB_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB2
cd /home/user001
#-adsh_step_start SUB2_S1 -run always -onError cont
echo SUB2_STEP_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB2
cd /home/user001
#-adsh_step_end
echo SUB2_JOB_STDERR_002 >&2
#-adsh_step_start SUB2_S2 -run always -onError cont
echo SUB2_STEP_STDERR_002 >&2
#-adsh_step_end
cd /home/user001/dir
cd xxxSUB2
```

```
cd /home/user001
echo SUB2_JOB_STDERR_003 >&2
```

このときのジョブ実行ログの出力例を次に示します。

```
[user001@hosta ~]$ /opt/jp1as/bin/adshexec sample.ash
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=046287
********   実行ジョブのSTDERRファイルの内容    ********
JOB_STDERR_001
KNAX6030-E ディレクトリを移動できません。/home/user001/dir/xxx - No such file or directory.
filename="/home/user001/sample.ash" line=4
KNAX6521-E コマンド（cd，行番号=4）がエラー終了しました。rc=1 E-Time=0.000s C-Time=0.000s
KNAX6597-I SAMPLEJOB.S1 ジョブステップが正常終了しました。rc=0 E-Time=0.001s C-Time=0.000s
JOB_STDERR_002
KNAX6597-I SAMPLEJOB.S2 ジョブステップが正常終了しました。rc=0 E-Time=0.004s C-Time=0.000s
KNAX6030-E ディレクトリを移動できません。/home/user001/dir/xxx - No such file or directory.
filename="/home/user001/sample.ash" line=18
KNAX6521-E コマンド（cd，行番号=18）がエラー終了しました。rc=1 E-Time=0.000s C-Time=0.000s
JOB_STDERR_003
SUB2_JOB_STDERR_001
SUB2_STEP_STDERR_001
SUB2_JOB_STDERR_002
SUB2_STEP_STDERR_002
SUB2_JOB_STDERR_003
KNAX0101-E SAMPLEJOB ジョブを実行中にエラーが発生しました。
KNAX0098-I SAMPLEJOB ジョブが終了しました。rc=0 E-Time=0.014s C-Time=0.010s

******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=S1 出力先=STDERR
STEP_STDERR_001
KNAX6030-E ディレクトリを移動できません。/home/user001/dir/xxx - No such file or directory.
filename="/home/user001/sample.ash" line=9
KNAX6521-E コマンド（cd，行番号=9）がエラー終了しました。rc=1 E-Time=0.000s C-Time=0.000s

KNAX0719-I STEP ステップ番号=0002 ステップ名=S2 出力先=STDERR
STEP_STDERR_002
SUB1_JOB_STDERR_001
SUB1_STEP_STDERR_001
SUB1_JOB_STDERR_002
SUB1_STEP_STDERR_002
SUB1_JOB_STDERR_003

KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/046287-SAMPLEJOB/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
[user001@hosta ~]$
```

STDERRの出力

3.6 実行したコマンドとその引数を出力する

シェルオプション xtrace を有効にすると，実行したコマンドとその引数がトレース情報として標準エラー出力へ出力されます。

シェルオプション xtrace を有効にする方法は次の 3 種類です。

- ジョブ定義スクリプトの set コマンドに，-x オプションまたは-o xtrace オプションを指定して実行する
- ジョブ実行時に adshexec コマンドに-x オプションを指定して実行する
- JP1/Advanced Shell エディタの［実行環境の設定］ダイアログボックスで，「xtrace を指定する」を選択する

トレース情報は次の形式で出力されます。

- トレース情報の先頭にはシェル変数 PS4 の値が付与されます。
- 変数の値を参照している場合は，変数の置換結果が出力されます。
- コマンドの引数にワイルドカードが含まれている場合，ワイルドカードによる置換結果が出力されます。

トレース情報の出力例

実行したジョブ定義スクリプトと，それによって出力されるトレース情報を次の例に示します。

ジョブ定義スクリプトの内容

```
0001 : set -o xtrace
0002 : typeset -i cnt=1
0003 : if [ $cnt -eq 1 ]
0004 : then
0005 :   echo ”--- JOB START ---”
0006 : fi
0007 : date
```

標準エラー出力への出力結果

```
+ typeset -i cnt=1
+ [ 1 -eq 1 ]
+ echo --- JOB START ---
+ date
```

トレース情報に関する注意事項

シェルオプション xtrace を有効にしても，次に示すコマンドとその引数は出力されません。

- test コマンドの省略形である[[ ]]コマンド
- スクリプト拡張コマンド

let コマンドの省略形である(( ))コマンドは，トレース情報では let コマンドに置き換えて出力されます。(( ))コマンドの指定例と出力情報を次に示します。

ジョブ定義スクリプトの内容

```
0001 : set -o xtrace
0002 : typeset -i a=0
0003 : (( a=(2+3)*9 ))
0004 : echo $a
```

標準エラー出力への出力結果

```
+ typeset -i a=0
+ let  a=(2+3)*9
+ echo 45
```

トレース情報には関数自体のトレース情報は出力されますが，関数内のコマンドのトレース情報は出力されません。関数内のコマンドのトレース情報を出力するには，typeset コマンドを実行して関数のトレースモードを有効にしてください。typeset コマンドの指定例と出力情報を次に示します。

ジョブ定義スクリプトの内容

```
0001 : set -o xtrace
0002 : fn1(){
0003 :   echo ”call $1 $2”
0004 :   echo $LINENO
0005 : }
0006 : echo ”in main”
0007 : fn1 ”function” ”1”
0008 : typeset -ft fn1
0009 : fn1 ”function” ”2”
```

標準エラー出力への出力結果

```
+ echo in main
+ fn1 function 1
+ typeset -ft fn1
+ fn1 function 2
+ echo call function 2
+ echo 4
```

シェルオプション xtrace を有効にすると，子孫ジョブで実行したジョブ定義スクリプト自体のトレース情報は出力されますが，子孫ジョブ内のコマンドのトレース情報は出力されません。子孫ジョブで実行するジョブ定義スクリプトのトレース情報を出力する場合は，子孫ジョブ用のジョブ定義スクリプト内で別途，xtrace シェルオプションを有効にする必要があります。

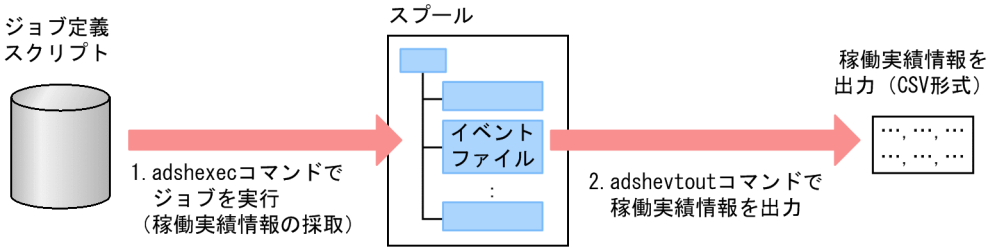
シェルオプション xtrace を有効にしても，ヒアドキュメントの入力内容はトレース情報として出力されません。

3.7　ジョブ定義スクリプトの稼働実績情報を出力する

ジョブ定義スクリプトの稼働実績情報は，ジョブで実行した各コマンドの実行時間，CPU 時間，出力されたメッセージ，ジョブステップの実行結果などの情報です。ジョブの実行状況の把握や，ジョブの実行遅延の原因の調査などに利用できます。

ジョブ定義スクリプトの稼働実績情報の採取と出力の流れを次に示します。

図 3-4　ジョブ定義スクリプトの稼働実績情報の採取と出力



1. ユーザーが adshexec コマンドでジョブを実行すると，adshexec コマンドがジョブのジョブ定義スクリプトの稼働実績情報を採取し，スプールのイベントファイルに出力します。
2. adshevtout コマンドは，イベントファイル内の，ジョブ定義スクリプトの稼働実績情報を CSV 形式で出力します。
   CSV 形式のジョブ定義スクリプトの稼働実績情報は，表計算ソフトウェアなどを使用して解析します。

3.7.1　ジョブ定義スクリプトの稼働実績情報の採取

ジョブ定義スクリプトの稼働実績情報の採取の可否を次の表に示します。

環境	実行状態	ジョブ定義スクリプトの稼働実績情報の採取
実行環境	通常	○
	デバッガモード	×
開発環境	（非該当）	×

（凡例）

○：採取できる

×：採取できない

標準では，adshexec コマンドでジョブを実行すると，adshexec コマンドがジョブのジョブ定義スクリプトの稼働実績情報を採取し，スプールのイベントファイルに出力します。

ただし，環境ファイルで次のように指定すると，adshexec コマンドはジョブ定義スクリプトの稼働実績情報をイベントファイルに出力しません。

```
#-adsh_conf EVENT_COLLECT NO
```

3.7.2　ジョブ定義スクリプトの稼働実績情報の出力

adshevtout コマンドを使用して，スプールにあるイベントファイル内のジョブ定義スクリプトの稼働実績情報を CSV 形式で出力します。

adshevtout コマンドの指定方法や，ジョブ定義スクリプトの稼働実績情報の出力例については，「8.　運用時に使用するコマンド」の「adshevtout コマンド（ジョブ定義スクリプトの稼働実績情報を出力する）」を参照してください。

(1)　ジョブ定義スクリプトの稼働実績情報を出力するジョブの指定

adshevtout コマンドで，ジョブ定義スクリプトの稼働実績情報を出力するジョブを指定できます。

次の内容で，出力するジョブを指定します。

- ジョブの実行開始日時の範囲
- JP1/AJS のジョブ名，ジョブ実行 ID，ジョブ番号など
- JP1/Advanced Shell のジョブ名，ジョブ識別子，ジョブ定義スクリプトファイルのパス名など

複数の条件を指定した場合，すべての条件を満たす情報が出力されます。

条件を指定しなければ，スプールにあるすべてのジョブの，ジョブ定義スクリプトの稼働実績情報を出力します。ただし，アクセスできないイベントファイル内にある，ジョブ定義スクリプトの稼働実績情報は出力しません。

(2)　ジョブ定義スクリプトの稼働実績情報の出力情報の制御

adshevtout コマンドの指定で，次のように出力情報を制御できます。

- ヘッダ情報を出力しない。
- ヘッダ情報だけを出力する（ジョブ定義スクリプトの稼働実績情報を出力しない）。
- ジョブ定義スクリプトの稼働実績情報内のメッセージだけを出力する。
- ジョブ定義スクリプトの稼働実績情報の環境変数の情報を出力しない。

(3)　参照するスプール

adshevtout コマンドは，スプールにあるイベントファイルを参照し，ジョブ定義スクリプトの稼働実績情報を出力します。

adshevtout コマンドが参照するスプールは，adshexec コマンドと同じく，指定された環境ファイルから決定します。

adshevtout コマンドで論理ホストを指定された場合，adshexec コマンドと同じく，論理ホストに対応づけられたスプールを参照します。

(4)　ジョブ定義スクリプトの稼働実績情報の出力先

ジョブ定義スクリプトの稼働実績情報は，adshevtout コマンドの標準出力（stdout）へ出力します。

リダイレクト機能を使用すると，ファイルに出力することもできます。

3.7.3　稼働実績情報の日時とタイムゾーンの関係

adshevtout コマンドは実行されたときのタイムゾーンに従い，日時を年月日，時分秒の形式で解釈し，出力します。タイムゾーンは環境変数 TZ で指定します。

次の例に示すように，同一の日時指定例 1 であっても，タイムゾーンが異なれば，年月日，時分秒の形式での日時の表現は変わります。日時指定例 2，日時指定例 3 についても同様です。

タイムゾーン	日時指定例 1	日時指定例 2	日時指定例 3
UTC-2	2012-06-10 08:00:00	2012-06-10 23:00:00	2012-06-11 15:00:00
UTC	2012-06-10 10:00:00	2012-06-11 01:00:00	2012-06-11 17:00:00
UTC+3	2012-06-10 13:00:00	2012-06-11 04:00:00	2012-06-11 20:00:00
UTC+9	2012-06-10 19:00:00	2012-06-11 10:00:00	2012-06-12 02:00:00

備考

UTC+9 は，協定世界時（UTC）から 9 時間進んでいるタイムゾーンであることを示します。環境変数 TZ に設定する場合と符号が異なります。

3.7.4　複数の OR 条件でジョブ定義スクリプト稼働実績情報を出力する

adshevtout コマンドは，引数に指定されたすべての条件を満たすジョブのジョブ定義スクリプト稼働実績情報を出力します。

複数の条件のどれかの条件に合致する情報を出力したい場合，OR となる各条件の数だけ adshevtout コマンドを実行して，ジョブ定義スクリプト稼働実績情報を 1 つの CSV ファイルに連結して出力します。

次の例ではジョブ定義スクリプト稼働実績情報を「outfile」ファイルへ連結して出力します。

```
adshevtout  -d                                  >  outfile
adshevtout  -t  条件1を指定するオプション        >> outfile
```

```
adshevtout  -t  条件2を指定するオプション        >> outfile
    ........
adshevtout  -t  条件nを指定するオプション        >> outfile
```

- 1 回目の adshevtout コマンドで，ヘッダ行だけを出力する（-d の指定でジョブ定義スクリプト稼働実績情報を出力しない）。
- 2 回目の adshevtout コマンドで，ヘッダ行の出力なし（-t の指定）で，条件 1 に該当するジョブの稼働実績情報を追加出力する。
- 3 回目の adshevtout コマンドで，ヘッダ行の出力なし（-t の指定）で，条件 2 に該当するジョブの稼働実績情報を追加出力する。
- n ＋ 1 回目の adshevtout コマンドで，ヘッダ行の出力なし（-t の指定）で，条件 n に該当するジョブの稼働実績情報を追加出力する。

このように adshevtout コマンドを実行することで，複数の or 条件に一致する稼働実績情報を出力できます。

3.7.5　異なるスプールのジョブ定義スクリプト稼働実績情報を出力する

adshevtout コマンドを 1 回実行して出力できるジョブ定義スクリプト稼働実績情報は，adshevtout コマンドの実行時に環境ファイルで指定されているスプールにあるジョブのものだけです。ほかのスプールにあるジョブのジョブ定義スクリプト稼働実績情報は，環境ファイルを切り替えて出力してください。

例を次に示します。次に示すように，各環境ファイルにスプールルートディレクトリが指定されているとします。

（例）

環境ファイル「envfile1」：スプールルートディレクトリ「spooldir1」を指定
環境ファイル「envfile2」：スプールルートディレクトリ「spooldir2」を指定
環境ファイル「envfile3」：スプールルートディレクトリ「spooldir3」を指定

この各スプールルートディレクトリにあるジョブのジョブ定義スクリプト稼働実績情報は，次のように環境ファイルを切り替えて，adshevtout コマンドを実行します。

```
export ADSH_ENV=envfile1
adshevtout

export ADSH_ENV=envfile2
adshevtout

export ADSH_ENV=envfile3
adshevtout
```

3.7.6　稼働実績情報の形式

adshevtout コマンドは，稼働実績情報を CSV 形式で出力します。

(1)　稼働実績情報の種類

adshevtout コマンドで出力される稼働実績情報は，大きく分けて次に示す情報があります。各情報が 1 個のレコードとなります。

- adshexec コマンドの実行開始
- 環境変数
- コマンド
- メッセージ
- ジョブステップの実行開始
- ジョブステップの実行終了
- ジョブステップの実行スキップ
- ジョブの実行終了

(2)　稼働実績情報の構成

稼働実績情報は，ジョブ単位に次に示す順序で出力します。

順序	出力項目	備考
1	ヘッダ情報	－
2	adshexec コマンドの実行開始	－
3	環境変数	環境変数がある場合に出力
4	コマンド メッセージ ジョブステップの実行開始 ジョブステップの実行終了 ジョブステップの実行スキップ ジョブの実行終了	ジョブ定義スクリプトの実行に応じて出力

（凡例）

－：該当なし

メッセージと次に示すレコードの出力順序は前後することがあります。

- コマンド
- ジョブステップの実行開始

- ジョブステップの実行終了
- ジョブステップの実行スキップ
- ジョブの実行終了

また，次に示す項目の間で，稼働実績情報の出力順序は規定されていません。

- スプールジョブ間の出力順序
- スプールジョブ内の，ルートジョブ，子孫ジョブの出力順序

(3)　稼働実績情報のレコードの構成

稼働実績情報のレコードは，複数の項目から構成されます。

項目の値は，ダブルクォーテーション（"）で囲まれます。項目の値がダブルクォーテーション（"）を含む場合，1 個のダブルクォーテーション（"）を，連続する 2 個のダブルクォーテーション（"）で表します。

項目は，コンマ（,）で区切ります。

3.7.7　CSV 形式の稼働実績情報のレコードと出力項目

次の表に，CSV 形式の稼働実績情報のレコードと出力項目の関係を示します。

表 3-2　稼働実績情報のレコードと出力項目

項番	項目名	adshexec コマンドの実行開始	環境変数	コマンド	関数	コマンド（スクリプト拡張コマンド）	メッセージ	ジョブステップの実行開始	ジョブステップの実行終了	ジョブステップの実行スキップ	ジョブの実行終了
1	EvtName	○	○	○	○	○	○	○	○	○	○
2	RecTZ	○	○	○	○	○	○	○	○	○	○
3	RecTime	○	○	○	○	○	○	○	○	○	○
4	PhysicalHostName	○	○	○	○	○	○	○	○	○	○
5	LogicalHostName	△	△	△	△	△	△	△	△	△	△
6	OsName	○	○	○	○	○	○	○	○	○	○
7	Jp1ajsService	△	△	△	△	△	△	△	△	△	△
8	Jp1ajsRootJobnet	△	△	△	△	△	△	△	△	△	△
9	Jp1ajsJobName	△	△	△	△	△	△	△	△	△	△
10	Jp1ajsExecId	△	△	△	△	△	△	△	△	△	△
11	Jp1ajsJobId	△	△	△	△	△	△	△	△	△	△

項番	項目名	adshexec コマンドの実行開始	環境変数	コマンド	関数	コマンド(スクリプト拡張コマンド)	メッセージ	ジョブステップの実行開始	ジョブステップの実行終了	ジョブステップの実行スキップ	ジョブの実行終了
12	Jp1asJobName	○	○	○	○	○	○	○	○	○	○
13	Jp1asJobId	○	○	○	○	○	○	○	○	○	○
14	Jp1asJobTime	○	○	○	○	○	○	○	○	○	○
15	Jp1asJobPid	○	○	○	○	○	○	○	○	○	○
16	Jp1asUid	○	○	○	○	○	○	○	○	○	○
17	Jp1asGid	○	○	○	○	○	○	○	○	○	○
18	Jp1asUserName	○	○	○	○	○	○	○	○	○	○
19	Jp1asGroupName	○	○	○	○	○	○	○	○	○	○
20	Jp1asScriptPath	○	○	○	○	○	○	○	○	○	○
21	Jp1asEnvPath	○	○	○	○	○	○	○	○	○	○
22	Jp1asSpoolPath	○	○	○	○	○	○	○	○	○	○
23	Jp1asJobLang	○	○	○	○	○	○	○	○	○	○
24	Jp1asJobEncode	○	○	○	○	○	○	○	○	○	○
25	RecTZExec	○	○	○	○	○	○	○	○	○	○
26	Jp1asJobParentJobName	△	△	△	△	△	△	△	△	△	△
27	Jp1asJobParentJobId	△	△	△	△	△	△	△	△	△	△
28	Jp1asJobParentJobTime	△	△	△	△	△	△	△	△	△	△
29	Jp1asJobParentJobPid	△	△	△	△	△	△	△	△	△	△
30	Jp1asJobRc	×	×	×	×	×	×	×	×	×	○
31	Jp1asJobSig	×	×	×	×	×	×	×	×	×	○
32	EnvVar	×	○	×	×	×	×	×	×	×	×
33	Jp1asScriptLineNo	×	×	○	○	○	×	○	○	○	×
34	Jp1asStepSeq	×	×	○	○	×	×	○	○	○	×
35	Jp1asStepName	×	×	△	△	×	×	△	△	△	×
36	Jp1asStepSkip	×	×	×	×	×	×	×	×	○	×
37	Jp1asStepRc	×	×	×	×	×	×	×	○	×	○
38	Jp1asStepSig	×	×	×	×	×	×	×	○	×	○

項番	項目名	adshexec コマンドの実行開始	環境変数	コマンド	関数	コマンド（スクリプト拡張コマンド）	メッセージ	ジョブステップの実行開始	ジョブステップの実行終了	ジョブステップの実行スキップ	ジョブの実行終了
39	JplasCmdExec	×	×	○	○	○	×	×	×	×	×
40	JplasCmdType	×	×	○	○	○	×	×	×	×	×
41	JplasCmdPath	×	×	△	△	○	×	×	×	×	×
42	JplasCmdArg	×	×	△	△	△	×	×	×	×	×
43	JplasLang	×	×	○	○	○	×	×	×	×	×
44	JplasCharEncode	×	×	○	○	○	×	×	×	×	×
45	JplasCmdStart	×	×	○	○	×	×	×	×	×	×
46	JplasCmdEnd	×	×	○	○	×	×	×	×	×	×
47	JplasCmdElaps	×	×	○	○	×	×	×	×	×	×
48	JplasCmdRc	×	×	○	○	×	×	×	×	×	×
49	JplasCmdSig	×	×	○	○	×	×	×	×	×	×
50	JplasCmdPid	×	×	△	△	×	×	×	×	×	×
51	JplasCmdCpuUser	×	×	△	△	×	×	×	×	×	×
52	JplasCmdCpuSys	×	×	△	△	×	×	×	×	×	×
53	Reserved1	×	×	×	×	×	×	×	×	×	×
54	Reserved2	×	×	×	×	×	×	×	×	×	×
55	JplasMsgId	×	×	×	×	×	○	×	×	×	×
56	JplasMsgText	×	×	×	×	×	○	×	×	×	×
57	JplasMsgLang	×	×	×	×	×	○	×	×	×	×
58	JplasMsgEncode	×	×	×	×	×	○	×	×	×	×

（凡例）

○：項目の情報が出力されます。

△：項目の情報が出力されます。出力する情報がない場合は空文字列となります。

×：出力する情報がないため，空文字列となります。

3.7.8　CSV 形式の稼働実績情報の出力項目

次の表に，CSV 形式の稼働実績情報の出力項目（列）を示します。

表 3-3　稼働実績情報の出力項目（列）

項番	項目名	内容
1	EvtName	稼働実績情報（レコード）の種類を示す名前。 • adshStart：adshexec コマンドの実行開始 • envVar：環境変数 • command：コマンド • function：関数 • message：メッセージ • stepStart：ジョブステップの実行開始 • stepEnd：ジョブステップの実行終了 • stepSkip：ジョブステップの実行スキップ • jobEnd：ジョブの実行終了
2	RecTZ	adshevtout コマンドの実行時のタイムゾーン※。 稼働実績情報の日時を表現するためのタイムゾーンとして使用します。
3	RecTime	稼働実績情報の記録日時※。
4	PhysicalHostName	物理ホスト名。
5	LogicalHostName	論理ホスト名。 論理ホストで実行していない場合，空文字列。
6	OsName	OS 名。 次に示す文字列。 • Windows • Linux • AIX • HP-UX • Solaris
7	Jp1ajsService	JP1/AJS のスケジューラーサービス名。 JP1/AJS が環境変数AJS_AJSCONF に設定する値。 JP1/AJS からジョブを起動した場合に出力します。
8	Jp1ajsRootJobnet	JP1/AJS のルートジョブネット名。 JP1/AJS が環境変数AJSNETNAME に設定する値。 JP1/AJS からジョブを起動した場合に出力します。
9	Jp1ajsJobName	JP1/AJS のジョブ名。 JP1/AJS が環境変数AJSJOBNAME に設定する値。 JP1/AJS からジョブを起動した場合に出力します。
10	Jp1ajsExecId	JP1/AJS のジョブの実行 ID。 JP1/AJS が環境変数AJSEXECID に設定する値。 JP1/AJS からジョブを起動した場合に出力します。
11	Jp1ajsJobId	JP1/AJS のジョブ番号。 JP1/AJS が環境変数JP1JobID に設定する値。

項番	項目名	内容
11	Jp1ajsJobId	JP1/AJS からジョブを起動した場合に出力します。
12	Jp1asJobName	JP1/Advanced Shell のジョブ名。
13	Jp1asJobId	JP1/Advanced Shell のジョブ識別子。
14	Jp1asJobTime	adshexec コマンドの実行開始日時※。
15	Jp1asJobPid	adshexec コマンドのプロセス id。
16	Jp1asUid	adshexec コマンドのプロセスのユーザー id。 Windows の場合，空文字列。
17	Jp1asGid	adshexec コマンドのプロセスのグループ id。 Windows の場合，空文字列。
18	Jp1asUserName	adshexec コマンドのプロセスのユーザー名。
19	Jp1asGroupName	adshexec コマンドのプロセスのグループ名。 Windows の場合，空文字列。
20	Jp1asScriptPath	ジョブ定義スクリプトのパス名※。
21	Jp1asEnvPath	環境ファイルのパス名※。
22	Jp1asSpoolPath	スプールディレクトリのパス名※。 スプールジョブのディレクトリ名は，ジョブが終了した状態では「*ジョブ識別子-スプールジョブ名*」ですが，この項目は，「*ジョブ識別子*」と出力します。
23	Jp1asJobLang	adshexec コマンドの実行開始時の環境変数LANG の値。
24	Jp1asJobEncode	adshexec コマンドの実行開始時の文字コード※。
25	RecTZExec	adshexec コマンドの実行時のタイムゾーン※。 このタイムゾーンは，稼働実績情報の日時を表現するためには使用しません。
26	Jp1asJobParentJobName	親ジョブのジョブ名。 ルートジョブの場合，空文字列。
27	Jp1asJobParentJobId	親ジョブのジョブ識別子。 ルートジョブの場合，空文字列。
28	Jp1asJobParentJobTime	親ジョブの実行開始日時※。 ルートジョブの場合，空文字列。
29	Jp1asJobParentJobPid	親ジョブのプロセス id。 ルートジョブの場合，空文字列。
30	Jp1asJobRc	adshexec コマンドの終了コード。
31	Jp1asJobSig	adshexec コマンドがシグナルで終了した場合のシグナル番号。 シグナルで終了しない場合，0。 Windows の場合，空文字列。
32	EnvVar	adshexec コマンドを起動した時の環境変数。

項番	項目名	内容
32	EnvVar	「*環境変数名＝値*」の形式。 詳細は「項目 EnvVar の環境変数」を参照してください。
33	Jp1asScriptLineNo	ジョブ定義スクリプトの行番号。 詳細は「項目 Jp1asScriptLineNo の行番号」を参照してください。
34	Jp1asStepSeq	ジョブステップ番号。 ステップ外の場合，空文字列。
35	Jp1asStepName	ジョブステップ名。 ジョブステップ名の省略時，ステップ外の場合，空文字列。
36	Jp1asStepSkip	ジョブステップの実行をスキップした時の先行コマンド，ジョブステップの状態。 • normal：正常であるためスキップ。 • abnormal：エラー終了が発生しているためスキップ。
37	Jp1asStepRc	ジョブステップの終了コード。
38	Jp1asStepSig	ジョブステップがシグナルで終了した場合のシグナル番号。 シグナルで終了しない場合，0。 Windows の場合，空文字列。
39	Jp1asCmdExec	コマンドの実行形態。 • *空文字列*：フォアグラウンド実行。 • back：バックグラウンド実行。 詳細は「項目 Jp1asCmdExec のコマンド実行形態」を参照してください。
40	Jp1asCmdType	コマンドの種類。 • assign：シェル変数更新のシェル標準コマンド。 • embStd：シェル標準コマンド，シェル拡張コマンド。 • embAdsh：スクリプト拡張コマンド。 • extCmd：外部コマンド。 • function：関数
41	Jp1asCmdPath	コマンド名。 外部コマンドの場合，コマンドのパス名※。 詳細は「項目 Jp1asCmdPath のコマンド名，コマンドのパス名」を参照してください。
42	Jp1asCmdArg	コマンドの引数。各引数はスペースで区切ります。 コマンドライン，ジョブ定義スクリプトに記述された引数そのものではなく，その引数を評価（展開）した結果の文字列で，コマンドに引き渡す文字列です。
43	Jp1asLang	コマンド実行時の環境変数LANG の値。
44	Jp1asCharEncode	コマンド実行時の文字コード※。
45	Jp1asCmdStart	コマンドの実行開始日時※。
46	Jp1asCmdEnd	コマンドの実行終了日時※。
47	Jp1asCmdElaps	コマンドの実行経過時間。

項番	項目名	内容
47	Jp1asCmdElaps	コマンドの実行開始日時と実行終了日時の差（単位：マイクロ秒）。
48	Jp1asCmdRc	コマンドの終了コード（リターンコード）。
49	Jp1asCmdSig	コマンドがシグナルで終了した場合のシグナル番号。 シグナルで終了しない場合，0。 Windows の場合，空文字列。
50	Jp1asCmdPid	外部コマンドのプロセス id。
51	Jp1asCmdCpuUser	コマンドのユーザー CPU 時間※（単位：マイクロ秒）。
52	Jp1asCmdCpuSys	コマンドのシステム CPU 時間※（単位：マイクロ秒）。
53	Reserved1	予約項目。空文字列。
54	Reserved2	予約項目。空文字列。
55	Jp1asMsgId	メッセージ id。
56	Jp1asMsgText	メッセージテキスト。
57	Jp1asMsgLang	メッセージ出力時の環境変数LANG の値。
58	Jp1asMsgEncode	メッセージ出力時の文字コード※。

注※

出力項目の内容について次の表で説明します。

項目	出力内容
改行文字	各項目の値に含まれる改行文字はスペースに置換します。 具体的には，LF(0x0A)，CR(0x0D)の各文字コードをそれぞれスペース(0x20)に置き換えます。 Windows の改行，CR(0x0D)，LF(0x0A)の 2 バイトは，スペース(0x20)，スペース(0x20)の 2 バイトに置き換えます。
日時	日時は次の形式で表現します。 *YYYY-MM-DD hh:mm:ss.nnn* *YYYY*：西暦年 *MM*：月 *DD*：日 *hh*：時 *mm*：分 *ss*：秒 *nnn*：ミリ秒 日時は，adshevtout コマンドを実行した時の環境変数TZ に指定されたタイムゾーンで表現します。 adshexec コマンドで稼働実績情報を採取する際に時刻取得関数でエラーが発生し，日時情報を取得できていない場合は，日時を次のように表現します。 EEEE-EE-EE ee:ee:ee.eee
タイムゾーン	UTC との時差を次のどちらかの形式で示します。符号は環境変数TZ の設定とは異なります。 +*hh*:*mm*:*ss*

項目	出力内容
タイムゾーン	*-hh*:*mm*:*ss* *hh*：時 *mm*：分 *ss*：秒 時差が 0 である場合，"+00:00:00" です。 （例） 協定世界時（UTC）から 9 時間進んでいるタイムゾーンの場合，" +09:00:00" です。 タイムゾーンを上記の形式に変換できない場合，項目の値を "+EE:EE:EE" とします。
文字コード	• Windows の場合 実際に使用している文字コードに関係なく SJIS 固定です。 • Linux，AIX，HP-UX，Solaris の場合 日本語の場合，環境変数LANG の値から判定した文字コードを次のように示します。 SJIS：シフト JIS コード EUC：Extended UNIX Code UTF8：UTF-8 上記以外の場合，次の値とします。 OTHER（環境変数LANG の値）
パス名	パス名の連続したディレクトリ区切り文字は，そのまま出力します。 連続したディレクトリ区切り文字を 1 個のディレクトリ区切り文字に置換しません。 たとえば，指定されたパス名が"C:¥¥dir"である場合，"C:¥dir"とは出力しません。"C:¥¥dir"と出力します。
CPU 時間	外部コマンドの場合 外部コマンドを実行したプロセスの CPU 時間。 外部コマンドでない場合 コマンドを実行したadshexec のプロセスの CPU 時間。 各コマンドを実行した時の CPU 時間ではありません。

(1) 項目 EnvVar の環境変数

出力する環境変数は次のとおりです。

- ジョブの起動時に設定されていた環境変数を出力します。
- 環境ファイルで環境変数を設定している場合，環境ファイルでの設定を反映したあとの環境変数を出力します。
- adshexec コマンドが設定する環境変数を出力します。
- adshexec コマンドが削除した環境変数は出力しません。
- 環境変数 ADSH_JOB_NAME の値はジョブ名の標準値，ADSHxxxxxx （xxxxxx：ジョブ識別子）です。#-adsh_job で指定するジョブ名ではありません。
- 子孫ジョブについても上記と同様です。

(2) 項目 Jp1asScriptLineNo の行番号

次の場合，項目の値は空文字列です。

- trap コマンドで指定した，アクションのコマンドを実行する場合。
- #-adsh_script コマンドを実行する場合。
- コマンド置換で指定されたコマンドを実行する場合。

関数を実行した場合，関数の本体に定義された実行コマンドの行番号を出力します。

(3) 項目 Jp1asCmdExec のコマンド実行形態

「cmd1 | cmd2 | cmd3」を実行した場合，cmd1，cmd2 は別プロセスで実行するため，項目の値は"back"となります。

(4) 項目 Jp1asCmdPath のコマンド名，コマンドのパス名

(a) コマンドを別プロセスで実行した場合

次の例のように，コマンドを別プロセスで実行した場合，項目の値は"Another process script"となります。

（例）

```
(echo1 a ; echo2 b)
(echo1 a) &
(echo1 a ; echo2 b) &
{ echo1 a ; echo2 b ; } &
{ echo1 a ; echo2 b ; } |&
```

次の場合，「{ echo1 a ; echo2 b ; }」の部分の値が"Another process script"となります。

```
{ echo1 a ; echo2 b ; } | echo3 c
```

(b) パイプで実行した場合

「cmd1 | cmd2 | cmd3」を実行した場合，cmd1，cmd2，cmd3 がすべて外部コマンドのときは項目の値は次のようになります。

- cmd1：コマンド名（ファイル名）
- cmd2：コマンド名（ファイル名）
- cmd3：パス名

3.7.9　ジョブ定義スクリプトの稼働実績情報の出力内容

ジョブ定義スクリプトの稼働実績情報の出力例を次に示します。この例では 1 行目にヘッダ行を出力しています。

```
”EvtName”,”RecTZ”,”RecTime”,”PhysicalHostName”,”LogicalHostName”,…
”adshStart”,”+09:00:00”,”2012-07-12 12:23:15.381”,”HOST01”,””,…
”envVar”,”+09:00:00”,”2012-07-12 12:23:15.382”,”HOST01”,””,…
    :（ジョブ定義スクリプトの稼働実績情報の出力内容）
```

3.8　ユーザー応答機能を使用する

ユーザー応答機能は，JP1 イベントを発行することで JP1/IM にバッチジョブの情報を通知し，また，それに対して応答を返すことができる機能です。この機能を利用すると，JP1/AJS から起動した場合などのように標準入出力となる画面を持たない環境でも，指定した文字列を運用者に対して通知できます。

運用者は，コマンド実行によって発行された JP1 イベントを，JP1/IM - Manager に接続した JP1/IM - View から監視できます。JP1/IM - View の統合コンソールでは複数のサーバの JP1 イベントを表示できるため，運用者は複数のサーバを一元的に監視できます。

3.8.1　前提条件

JP1/IM など必要なプログラムについては，「2.2.2　環境ごとに必要なプログラム」を参照してください。また，「2.8　ユーザー応答機能を設定する」に示す環境設定を実施してください。

3.8.2　実行方法

ユーザー応答機能で JP1 イベントとして発行する文字列は，次のコマンドで指定します。これらのコマンドはジョブ定義スクリプトに指定します。

コマンド名	用途	コマンド指定方法の記載個所
adshecho	事象通知メッセージを JP1 イベントとして発行する	「シェル拡張コマンド」の「adshecho コマンド（指定した事象通知メッセージを JP1 イベントとして発行する）」
adshread	応答要求メッセージを応答待ちイベントとして発行する	「シェル拡張コマンド」の「adshread コマンド（指定した応答要求メッセージを応答待ちイベントとして発行する）」

発行された JP1 イベントは JP1/IM - View の画面に表示され，応答待ちイベントの場合，運用者は JP1/IM - View から応答を入力できます。

3.8.3　JP1/IM - View との関係

ユーザー応答機能を使用すると，ジョブ定義スクリプトで指定した事象通知メッセージおよび応答要求メッセージが JP1 イベントとして出力されます。JP1 イベントとして出力した事象通知メッセージおよび応答要求メッセージは，JP1/IM - View で参照できます。

adshread コマンドで指定した応答要求メッセージの場合は，JP1/IM - Manager で応答待ちイベントとして扱われます。応答待ちイベントは JP1/IM - View で滞留して表示され，JP1/IM - View から応答が入力できます。

応答手順についてはJP1/IMのマニュアルを参照してください。JP1/IM - Viewから応答入力する場合に［応答入力］画面に表示される状態と意味を次の表に示します。

表3-4　JP1/IM - Viewの［応答入力］画面に表示される状態と意味

状態	意味	JP1/IM - Viewからの応答入力可否
READY TO RESPOND	応答を入力できる状態である。	○
NO LONGER MANAGED BY JP1/AS	応答要求メッセージが，JP1/Advanced Shellによって管理されていないため，応答できない状態である。	×
RESPONDED SUCCESSFULLY	応答要求メッセージに対する応答に成功した。	×
ALREADY RESPONDED	応答要求メッセージに対する応答がすでに入力されている。	×
INTERNAL ERROR	JP1/Advanced Shellで内部エラーが発生したため，応答できない。	×

（凡例）

○：応答できる。

×：応答できない。

3.8.4　ユーザー応答機能の入出力先に標準入出力を指定する方法

adshechoコマンドやadshreadコマンドはJP1イベントを発行するため，ジョブ定義スクリプトをデバッグする場合など，JP1/BaseやJP1/IMが存在しない環境で使用すると，これらのコマンドはエラー終了します。このような場合でもジョブ定義スクリプトのデバッグ作業を行えるように，ユーザー応答機能では次の機能を提供します。

- JP1/IM - Viewではなく標準出力に文字列を出力する機能
- JP1/IM - Viewからではなく標準入力から応答を入力する機能

adshechoコマンドやadshreadコマンドを使用したときにJP1イベントを発行するか標準入出力を使用するかは，USERREPLY_DEBUG_DESTINATIONパラメーターで選択できます。USERREPLY_DEBUG_DESTINATIONパラメーターは，システム環境ファイルまたはジョブ環境ファイルで指定します。また，adshechoコマンドやadshreadコマンドに-dオプションを指定することで，文字列の入出力先を標準入出力にすることができます。

ユーザー応答機能の入出力先に標準入出力を指定する機能は，デバッグ実行時（UNIXの場合はadshexec -dで実行，Windowsの場合は開発環境から実行）だけ有効です。この機能を使用する場合，ユーザー応答機能の関連プログラムであるJP1/Base，JP1/Integrated Management - Manager，JP1/Integrated Management - Viewは不要です。また，ユーザー応答機能管理デーモン・サービスを起動する必要はありません。

また，次のパラメーターの指定は無効となります。

- HOSTNAME_JP1IM_MANAGER パラメーター
- USERREPLY_JP1EVENT_INTERVAL パラメーター
- USERREPLY_WAIT_MAXCOUNT パラメーター

3.8.5 adshecho コマンドまたは adshread コマンドがエラー終了した場合の対処

adshecho コマンドまたは adshread コマンドがエラー終了した場合でも，再実行によって成功する場合があります。再実行する場合は次の例を参考に，adshecho コマンドまたは adshread コマンドを再実行するジョブ定義スクリプトを作成してください。

```
#! /opt/jp1as/bin/adshexec

###
#adshreadコマンドを実行する関数
#引数   ：応答要求メッセージ
#終了コード：0（正常終了）
#        1（リトライ可能なエラーで終了）
#        2（リトライ不可能なエラーで終了）
###

func_adshread()
{
    adshread ans "$1"

  case "$?" in
    # 正常終了
    0 )             return 0 ;;
    # リトライ可能なエラーで終了した場合
    3 | 4 | 6 | 8 ) return 1 ;;
    # 上記以外のエラーで終了
    * )             return 2 ;;
  esac
}

###
### スクリプトの本体
###

#
# ジョブとして実行する処理を記述
#

###
### オペレータの応答を待って処理を続行する
###

  while :
  do
```

```
    #adshreadコマンドを実行する関数を呼び出す。
    func_adshread "処理を続行しますか？(Y/N)【ホスト名：$HOSTNAME, スクリプト名：$0】"

    if [ $? = 0 ]; then       #adshreadコマンドが正常に終了した
      break                   #ループを抜ける
    elif [ $? = 1 ] ; then    #adshreadコマンドがリトライ可能なエラーで終了した
      continue                #adshreadコマンドを再実行する
    else                      #adshreadコマンドがリトライ可能なエラー以外で終了した
      echo "adshreadコマンドがエラー終了しました。"
      exit 1                  #スクリプトを終了する
    fi
  done

###
### adshreadが受け取った応答に応じて処理を行う
###

  if [ "$ans" = "Y" ] ; then
    adshecho "「Y」が入力されました。処理を続行します。"
  elif [ "$ans" = "N" ] ; then
    adshecho "「N」が入力されました。処理を終了します。"
    exit 1                        #スクリプトを終了する
  else
    adshecho "指定以外の応答が入力されました。処理を終了します。"
    exit 1                        #スクリプトを終了する
  fi
```

3.8.6 注意事項

- HOSTNAME_JP1IM_MANAGER パラメーターで指定した JP1/IM - Manager に接続した JP1/IM - View 以外からは応答はできません。
- JP1/IM - Manager が滞留できる応答待ちイベント数には上限があるため，これを超えない運用を設計してください。
- JP1/IM - View から入力できる文字コードは，ASCII 文字コード（制御文字を除く）の範囲内です。範囲以外の文字コードを入力するとエラーメッセージが表示されるため，範囲内の文字コードで再入力してください。
- adshecho コマンドおよび adshread コマンドを実行すると，HOSTNAME_JP1IM_MANAGER パラメーターで指定したホストに対して，JP1/Base が TCP/IP のコネクションの切断と新たなポートで接続します。

  コネクションが使用していたポートは，OS の MSL（Maximum Segment Lifetime）× 2（秒）の間は使用できなくなるため，MSL の値が大きいまたはポート数が少ない場合には，ポートが枯渇するおそれがあります。

  そのため，MSL × 2（秒）の間に出力する JP1 イベント数，MSL，およびポート数は，次に示す条件を満たすようにしてください。

  n × MSL × 2/3 ＜ **ポート数**

n：MSL × 2（秒）の間にユーザー応答機能によって出力する JP1 イベントの数

- UNIX の場合，応答要求メッセージを共有メモリ上で管理するために，ユーザー応答機能管理デーモンの起動時に次の名称のファイルをスプールディレクトリに作成します。
  - .adsh_mqueue
  - .adsh_mqueue_**論理ホスト名**（論理ホストのユーザー応答機能管理デーモンを起動した場合）

  上記のファイルは，ユーザー応答機能管理デーモンの動作中は削除しないでください。なお，上記のファイルはユーザー応答機能管理デーモンを終了しても削除されないで，次のユーザー応答機能管理デーモン起動時に再利用されます。

  HP-UX の場合は，上記のファイルのほかに，次の名称のファイルをスプールディレクトリに作成します。

  - .adsh_mqueueS
  - .adsh_mqueue_**論理ホスト名**S（論理ホストのユーザー応答機能管理デーモンを起動した場合）

  上記のファイルは，ユーザー応答機能管理デーモンの動作中は削除しないでください。なお，上記のファイルはユーザー応答機能管理デーモンの終了時に削除されます。

- OS をシャットダウンする場合は，ユーザー応答機能管理デーモンおよびサービスを停止してからシャットダウンしてください。ユーザー応答機能管理デーモンおよびサービスの停止時に応答待ち状態の応答要求メッセージが存在する場合，これらのデーモンやサービスは，応答要求メッセージをキャンセルしてから停止します。しかし，OS のシャットダウン処理が実行されると，応答要求メッセージのキャンセルが実行される前にシャットダウンする場合があります。この場合，JP1/IM - View に応答待ちイベントが滞留したままになります。

3.9　スプールジョブを削除する

スプールに格納されたスプールジョブは，スプールディレクトリに保存されたまま，増加します。このため，定期的に古いスプールジョブを削除してディスクの空き容量を増やす必要があります。

- スプールジョブの削除方法

  スプールジョブを削除するためには，次のadshhk コマンドを入力します。adshhk コマンドの指定方法については「8.3.9　adshhk コマンド（スプールジョブを削除する）」を参照してください。

  ```
  adshhk 対象リストファイル名 レポートファイル名 ログファイル名 ［日数］
  ```

  *対象リストファイル名*のファイルには，削除したいスプールジョブのあるスプールディレクトリ名などをadshhk コマンドの実行前に記載する必要があります。

  *レポートファイル名*のファイルにadshhk コマンドの実行結果が出力されます。実行結果は，トレースログにも出力されます。

  *ログファイル名*のファイルには，エラーメッセージを出力します。

  adshhk コマンド実行日の前日を基点にして，指定した日数以前のスプールジョブを削除します。例えば，2 を指定した場合，昨日（基点）とおとといの 2 日間の指定となるため，おとといい以前のスプールジョブを削除します。なお，当日のスプールジョブは削除できません。

- adshhk コマンドで作成されるレポートファイル

  adshhk コマンドを実行した場合，実行結果がレポートファイルに出力されます。次のレポートファイルの例では，先頭行にヘッダ情報が出力されています。

  ```
  ”jobid”,”jobname”,”rc”,”start date”,”end date”,”act”,”info”,”spool”,”target
  days”,”execute date”
  ”000056”,”JOB001”,”1”,”2011/06/13 09:03:31”,”2011/06/13 09:03:31”,”delete”,””,”C:
  ¥Documents and Settings¥All Users¥Documents¥Hitachi¥jp1as¥jp1ase¥spool”,”15”,”2011/06/30
  18:19:58”
                          :
  ```

  （凡例）

  実行結果の 1 行目の見出しの意味は次のとおりです。2 行目以降には対応した項目の値が表示されます。

見出し	意味
jobid	ジョブ識別子
jobname	JP1/Advanced Shell のジョブ名 ただし，JP1/Advanced Shell のジョブ名が正しく求められない場合，スプールジョブ名を出力します。
rc	ジョブの終了コード
start date	ジョブの実行開始日時（*yyyy/mm/dd hh:mm:ss* の形式）。 デバッグモードで起動したときにデバッガ自身に割り当てるスプールは，ジョブの実行結果ではなくデバッガのログ出力用のため，デバッグの開始日時が出力されます。
end date	ジョブの終了日時（*yyyy/mm/dd hh:mm:ss* の形式）。

見出し	意味
end date	デバッグモードで起動したときにデバッガ自身に割り当てるスプールは，ジョブの実行結果ではなくデバッガのログ出力用のため，ジョブ終了日時は出力されません。
act	適用した動作（keep：保存，delete：削除，error：削除処理中にエラー発生）
info	エラーの詳細情報
spool	スプールディレクトリ
target days	対象日数
execute date	コマンドの実行開始日時（*yyyy/mm/dd hh:mm:ss* の形式）

重要

スプールのディレクトリの下に，JP1/Advanced Shell が作成するファイルやディレクトリではないユーザー独自のファイルやディレクトリが存在する場合，adshhk コマンドはKNAX4419-E メッセージなどを出力して終了します。

3.10　カバレージ情報を取得する

ユーザーがジョブ実行時にカバレージ情報を取得する指定をすると，JP1/Advanced Shell は，ジョブ定義スクリプトのコマンドが実行されたかどうかをカバレージ情報ファイル（asc ファイル）に記録します。

ユーザーがカバレージ情報を操作するコマンドを実行して，カバレージ情報をマージしたり，表示したりできます。

3.10.1　カバレージ情報の概要

カバレージ情報とは，プログラムのテストがどれだけ網羅されているかを示す指標です。カバレージ情報の指標には C0 情報と C1 情報があります。

(1)　C0 情報と C1 情報

C0 情報と C1 情報について説明します。

- C0（ステートメントカバレージ情報）：コマンド網羅性

  テスト対象となるジョブ定義スクリプト中のコマンドをどれだけ実行したかの指標で，次の式で計算できます。

  C0 ＝（実行が済んだコマンドの数）／（全コマンドの数）×100（%）

- C1（ブランチカバレージ情報）：分岐網羅性

  テスト対象となるジョブ定義スクリプト中の分岐をどれだけ実行したかの指標で，次の式で計算できます。

  C1 ＝（実行が済んだ分岐の数）／（実行できる分岐の数）×100（%）

(2)　カバレージ情報の機能

カバレージ情報は，ジョブ定義スクリプトをテストするときの参考資料として使用できます。また，カバレージ情報を蓄積，表示またはマージできます。カバレージ情報の取得対象については，「付録 A　カバレージ情報を取得する対象」を参照してください。

カバレージ情報の採取，表示およびマージ方法には，Windows と UNIX とで次の表で示す相違点があります。

表 3-5　Windows と UNIX でのカバレージ情報の採取，表示およびマージの相違点

環境	カバレージ情報の採取	カバレージ情報の表示	カバレージ情報のマージ
Windows の開発環境	JP1/Advanced Shell エディタのデバッグ機能を使用してカバレージ情報を採取します。採取した情報はカバレージ情報ファイルに蓄積されます。	次のどちらかの方法で表示できます。 • adshcvshow コマンド	adshcvmerg コマンド※でマージします。 エディタではマージできません。

環境	カバレージ情報の採取	カバレージ情報の表示	カバレージ情報のマージ
Windows の開発環境	実行環境の設定で「カバレージを蓄積する」を選択しないとカバレージ情報は採取しません。 カバレージ採取の一括有効化機能は利用できません。	• JP1/Advanced Shell エディタ	adshcvmerg コマンド※でマージします。 エディタではマージできません。
Windows の実行環境	ジョブ定義スクリプト実行時に adshexec コマンドでカバレージの蓄積オプション（-t）を指定して，カバレージ情報を採取します。採取した情報はカバレージ情報ファイルに蓄積されます。 デバッグ機能は使用できません。	adshcvshow コマンドで表示します。	adshcvmerg コマンド※でマージします。
UNIX の実行環境	adshexec コマンドで次のオプションを指定します。両方のオプションを指定することもできます。 • カバレージの蓄積オプション（-t）を指定してカバレージ情報ファイルに採取します。 • デバッグオプション（-d）を指定してメモリに採取します。	• カバレージ情報ファイルに採取した場合，adshcvshow コマンドで表示します。 • メモリに採取した場合，info coverage コマンドで表示します。	adshcvmerg コマンド※でマージします。

注

異なるプラットフォーム間でのカバレージ情報の相互運用で，次の注意が必要です。

- 異なるプラットフォーム間でカバレージ情報を転送しないでください。
- 採取したカバレージ情報ファイルは，他 OS のコマンドでは処理できません。

注※

adshcvmerg コマンドによって，1 回の実行につき 2 つのカバレージ情報ファイルのカバレージ情報をマージできます。3 つ以上のカバレージ情報ファイルのカバレージ情報をマージする場合は，このコマンドを複数回実行して，カバレージ情報のマージを繰り返してください。

3.10.2　カバレージ情報の管理

カバレージ情報はカバレージ情報ファイル（asc ファイル）で管理します。

(1)　カバレージ情報ファイルのファイル名と格納ディレクトリ

(a)　ファイル名

カバレージ情報はジョブ定義スクリプトとユーザー単位にまとめられます。このため，デフォルトの asc ファイル名には，ジョブ定義スクリプト名とユーザー名が含まれます。

実行環境の場合，コマンドオプションで任意の asc ファイル名を指定できます。

エディタの場合，デフォルトの asc ファイル名となります。任意の asc ファイル名は指定できません。

デフォルトの asc ファイル名を次に示します。

ジョブ定義スクリプト名（拡張子を除きます）_**ユーザー名**.asc

asc ファイルの名称のバイト数が OS の上限値を超えた場合，カバレージの取得に失敗します。実行するジョブ定義スクリプトのファイル名の文字数に注意してください。

(b) 格納ディレクトリ

実行環境の場合，デフォルトの asc ファイルは，コマンド実行時のカレントディレクトリに作成します。

開発環境の場合，Windows のエディタからカバレージ情報を蓄積しているときは，ジョブ定義スクリプトファイルがあるディレクトリに asc ファイルを作成します。

(2) asc ファイルの更新

asc ファイルは，カバレージ情報の蓄積，またはマージ時に更新されます。

asc ファイルは複数のユーザーで同時に共用できません。ほかのユーザーで使用中の asc ファイルを使おうとすると，KNAX6211-E メッセージを出力してコマンドがエラーとなります。

(3) asc ファイルの出力処理

ディスク容量不足などの理由で，asc ファイルへのカバレージ情報の書き込みが失敗すると，カバレージ情報が失われます。

カバレージ情報を採取するとき，asc ファイルがすでに存在する場合は前に採取したカバレージ情報を失わないように，次のように asc ファイルが更新されます。存在する asc ファイルは直接更新しません。

1. 一時 asc ファイルに新しいカバレージ情報を出力する。
2. 既存の asc ファイルをバックアップ asc ファイルに変更する。
3. 一時 asc ファイルを指定された asc ファイル（デフォルトの asc ファイル名の場合を含む）に変更する。
4. バックアップ asc ファイルを削除する。

このため，asc ファイルへのカバレージ情報の出力処理が途中で終了しても，コマンドの再実行時にカバレージ情報が適切に回復して，コマンドの処理を続行します。

一時 asc ファイルとバックアップ asc ファイルは，コマンドの処理が正常に終了した場合は残りませんが，asc ファイルへのカバレージ情報の出力処理が途中で終了すると残る場合があります。これらのファイルは，コマンドを再実行した場合，またはコマンドを処理して正常に終了した場合，削除されます。

一時 asc ファイルは手動で削除してもかまいませんが，バックアップ asc ファイルは削除しないでください。削除すると，コマンドがエラー終了する直前までに蓄積してきたカバレージ情報を失うことになります。

カバレージ機能を使用する場合，通常，上記のことを意識する必要はありません。

指定された asc ファイル（デフォルトの asc ファイル名の場合を含む）が，新しいカバレージ情報で更新されたかどうかは，直近のコマンドの実行で，KNAX6242-I メッセージの出力の有無で判断できます。KNAX6242-I メッセージを出力した場合，指定された asc ファイル（デフォルトの asc ファイル名の場合を含む）の内容は，新しいカバレージ情報で更新されています。

(4) asc ファイルに関連するファイル名

カバレージ情報を asc ファイルに採取するとき，一時的に格納するファイルを作成します。このファイルを一時カバレージ情報ファイル（一時 asc ファイル）と呼びます。

一時 asc ファイルのファイル名

次のように，ファイル名の先頭部分が固定の文字列となります。

- adshexec コマンドの場合：adshexec_temp_**任意**

バックアップ asc ファイルのファイル名

asc ファイルを一時的にリネームするファイル名（バックアップ asc ファイル）は次のようになります。

- adshexec コマンドの場合：adshexec_backup_**任意**

コマンドの引数に指定する asc ファイル（デフォルトの asc ファイルを含む）は，前述したファイル名が OS が許容するパス名の最大バイト数以下となるように指定する必要があります。

コマンドの実行中に処理がキャンセルされると，一時 asc ファイルとバックアップ asc ファイルが残ることがあります。残ったファイルの扱いを次に示します。

- 一時 asc ファイルが残っていても，コマンドを再実行できます。なお，一時 asc ファイルは手動で削除して再実行することもできます。
- バックアップ asc ファイルが残っている場合は，このバックアップファイルを削除しないでください。残っているバックアップ asc ファイルは，adshexec コマンドが自動的に元の asc ファイル名に戻してカバレージ情報を採取します。また，手動で元の asc ファイル名に戻して再実行することもできます。

一時 asc ファイル，バックアップ asc ファイルと同じ名称を持つユーザーファイルを作成しないでください。asc ファイルと同一のディレクトリに，asc ファイルに対応する一時 asc ファイルと同名のファイルがあると削除されます。asc ファイルに対応するバックアップ asc ファイルと同名のファイルがあると，asc ファイルと扱ったり，削除されたりします。

(5) 一時 asc ファイルとバックアップ asc ファイルの使用

実行環境では，adshexec コマンドの-t および-d オプションの指定によって，一時 asc ファイルとバックアップ asc ファイルを使用するかどうかが変わります。実行環境での一時 asc ファイルとバックアップ asc ファイルの使用を次の表に示します。

表 3-6　実行環境での一時 asc ファイルとバックアップ asc ファイルの使用

adshexec コマンドのオプション		Windows		UNIX	
-t	-d	一時 asc ファイル	バックアップ asc ファイル	一時 asc ファイル	バックアップ asc ファイル
なし	なし	×	×	×	×
なし	あり	－	－	○	×
あり	なし	○	○	○	○
あり	あり	－	－	○	○

（凡例）

○：ファイルを使用します。

×：ファイルを使用しません。

－：デバッグオプションは，Windows でサポートしていません。

Windows の開発環境では，［実行環境の設定］ダイアログボックスで指定する「カバレージの蓄積」の指定によって，一時 asc ファイルとバックアップ asc ファイルを使用するかどうかが変わります。開発環境での一時 asc ファイルとバックアップ asc ファイルの使用を次の表に示します。

表 3-7　開発環境での一時 asc ファイルとバックアップ asc ファイルの使用

「カバレージの蓄積」の指定	Windows の開発環境（エディタ）	
	一時 asc ファイル	バックアップ asc ファイル
蓄積しない	×	×
蓄積する	○	○
蓄積する（ジョブ定義スクリプトファイル更新時はカバレージ情報ファイルを上書き）	○	○

（凡例）

○：ファイルを使用します。

×：ファイルを使用しません。

(6)　コマンドと一時 asc ファイル

コマンドの実行時，一時 asc ファイルを作業ファイルとして使用します。コマンドの実行前に一時 asc ファイルと同名のファイルがすでに存在する場合は，削除します。

(7)　adshexec コマンド実行時の asc ファイルの処理方法

adshexec コマンド実行時の asc ファイルの処理方法は，次のようになります。

表 3-8　adshexec コマンド実行時の asc ファイルの処理方法

コマンド起動時の状態		コマンドの処理		
job.asc	adshexec_backup_job.asc	job.asc	adshexec_backup_job.asc	asc ファイルの状態と処理方法
×	×	新規作成	なし	新旧の asc ファイルがない状態です。 asc ファイルを新規に作成します。
×	○	なし	job.asc へ名称を変更	旧 asc ファイル job.asc をバックアップファイル adshexec_backup_job.asc に名称を変更した状態です。 バックアップ asc ファイル adshexec_backup_job.asc から，asc ファイル job.asc を回復します。
○	×	このファイルを使用	なし	job.asc は，旧 asc ファイル，新 asc ファイルのどちらかです。 新 asc ファイルである場合，直前の実行で KNAX6242-I メッセージを出力しています。
○	○	このファイルを使用	このファイルを削除	job.asc は新 asc ファイル，adshexec_backup_job.asc は旧 asc ファイルです。 旧 asc ファイルである adshexec_backup_job.asc を削除します。 新 asc ファイルである job.asc を使用します。

（凡例）

○：ファイルが存在します。

×：ファイルが存在しません。

job.asc：asc ファイル名

adshexec_backup_job.asc：バックアップ asc ファイル名

3.10.3　カバレージ情報の蓄積

(1)　カバレージ情報の蓄積方法と形式

ジョブ定義スクリプトを実行する場合に実行コマンドのオプションでカバレージ情報を蓄積することを指定します。カバレージ情報の蓄積を指定すると，asc ファイルにカバレージ情報が蓄積されます。

カバレージ情報を蓄積するためのオプションは，-t オプションです。-o オプションで asc ファイルのファイル名を任意に変更できます。カバレージ情報を蓄積するオプションを指定した実行コマンドの形式を次に示します。通常の場合，実行するジョブ定義スクリプトに差分があったときはエラーとします。ただし，-f オプションを指定した場合，エラーにはならないでカバレージ情報を上書きします。

```
adshexec ［ほかのオプション…］ ［-t ［-f ］ ［-o ascファイルのパス名］］
ジョブ定義スクリプトファイルのパス名 ［実行時パラメーター］
```

UNIX の場合に，-t オプションではなく-d オプションを指定して adshexec コマンドを実行したときは，メモリ上だけでカバレージ情報を採取します。このとき，デバッガの info coverage コマンドでカバレージ情報を表示できます。デバッガの quit コマンドを入力してデバッガを終了すると，採取したカバレージ情報を破棄してメモリを解放します。

(2) 蓄積方法の種類

カバレージ情報の蓄積方法には，1 回目の蓄積である初回蓄積と，2 回目以降の蓄積である継続蓄積とがあります。1 回目か 2 回目以降かの判定は，asc ファイルがあるかどうかで決まります。

ジョブ定義スクリプトに変更があった場合，変更部分の行番号が不一致になります。このため，asc ファイルにジョブ定義スクリプトの内容のバックアップ情報を保持します。asc ファイルのバックアップ情報と差分が発生した場合，ジョブ定義スクリプトを実行しないでエラー終了となります。

初回蓄積の場合，asc ファイルを新たに作成し，実行時のカバレージ情報を書き込みます。継続蓄積の場合，asc ファイルの内容を読み込み，asc ファイルの情報に現在の実行のカバレージ情報を追加して，asc ファイルを更新します。

(a) 初回蓄積の例

初回蓄積の例を示します。

例 1

カバレージ情報ファイル（asc ファイル）が存在しない状態で，カバレージ情報を採取します。

例 2

次に示す条件がすべて成立する場合は，初回蓄積になります。

- カバレージ情報ファイル（asc ファイル）が存在する
- ジョブ定義スクリプトファイルが，すでに存在するカバレージ情報ファイルのカバレージ情報を採取したときと異なる
- ジョブ定義スクリプトファイルが異なっていてカバレージ情報ファイルを初期化するオプション（adshexec コマンドで-f オプション）の指定がある

(b) 継続蓄積の例

次に示す条件がすべて成立する場合は，継続蓄積になります。

- すでにカバレージ情報ファイル（asc ファイル）が存在する
- ジョブ定義スクリプトファイルが，すでに存在するカバレージ情報ファイルのカバレージ情報を採取したときと同一である

継続蓄積では，ジョブ定義スクリプトが前回カバレージ情報を採取したときと同一であることが継続蓄積の条件の 1 つになっています。次に示す場合，ジョブ定義スクリプトは同一であると判断します。

- ジョブ定義スクリプトをバイナリ比較してサイズと内容が同一である

上記の条件が成立すれば，ジョブ定義スクリプトはファイル名やパスが異なっていても同一として取り扱います。

(3) カバレージ情報ファイルに登録するジョブ定義スクリプトのファイル名

継続蓄積の場合，最後にカバレージ情報を採取したときのジョブ定義スクリプトのファイル名が有効となります。

例

次のファイルは，ファイル名は異なりますが，同一のジョブ定義スクリプトとします。

- /dir1/file1
- /dir2/file2

出力 asc ファイルを out.asc として，上記のジョブ定義スクリプトを使用して，継続蓄積を実行すると，out.asc 内のスクリプトファイル名は次に示すようになります。

1. adshexec -t -o out.asc /dir1/file1 を実行すると，out.asc 内のスクリプトファイル名は，/dir1/file1 となります。
2. adshexec -t -o out.asc /dir2/file2 を実行すると，out.asc 内のスクリプトファイル名は，/dir2/file2 となります。
3. 再び adshexec -t -o out.asc /dir1/file1 を実行すると，out.asc 内のスクリプトファイル名は，/dir1/file1 となります。

(4) カバレージ情報ファイルの拡張子

カバレージ情報ファイル（asc ファイル）の拡張子のデフォルトは，「.asc」です。ただし，カバレージ情報ファイルの拡張子「.asc」でなくてもかまいません。カバレージ情報を取得する場合，カバレージ情報ファイルとして指定した任意の拡張子のファイルを asc ファイルとして扱います。

(5) ジョブ定義スクリプトのサイズ

ジョブ定義スクリプトファイルのサイズは，2GB 未満とします。

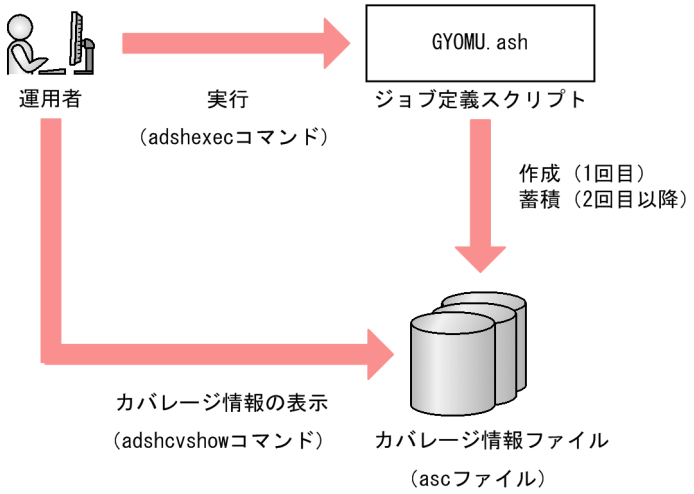
(6) 蓄積したカバレージ情報の初期化

蓄積したカバレージ情報を初期化する場合は，該当する asc ファイルを rm コマンドなどで削除し，初回蓄積として再度カバレージ情報を採取してください。

3.10.4　カバレージ情報の表示

ジョブ定義スクリプトを実行し，カバレージ情報を表示するまでの流れを次の図に示します。

図 3-5　カバレージ情報の表示の流れ



なお，開発環境のエディタでもカバレージ情報を表示できます。エディタを使用してカバレージ情報を表示する方法の詳細については，「4.4.7　カバレージ情報を表示する」を参照してください。

(1)　表示方法とコマンドの形式

カバレージ情報は，カバレージ情報を表示するコマンド（adshcvshow コマンド）で表示します。コマンドに指定した asc ファイルの内容を表示します。-l オプションのオプション値としてジョブ定義スクリプトの範囲を指定することで，指定した範囲内のカバレージ情報を表示できます。

-s オプションを指定した場合，バックアップされているジョブ定義スクリプトの内容だけを表示します。-s オプションは asc ファイルのバックアップされているジョブ定義スクリプトの内容の確認やジョブ定義スクリプトとの差分を確認したいときに使用します。-s オプションを指定した場合，-l オプションによる範囲指定はできません。

カバレージ情報表示コマンドの形式を次に示します。

```
adshcvshow  {[-l n1[-[n2]][,n3[-[n4]]]...]|-s}  ascファイルのパス名
```

行指定の形式は，","で複数指定し"-"で範囲を指定します。例えば，1 行目～10 行目と 15 行目と 21 行目～30 行目を指定したい場合，次のように指定します。

```
adshcvshow -l 1-10,15,21-30 ascファイルのパス名
```

"-"の後ろに数字を指定しなければ，前の値から最終行までの範囲を意味します。例えば，21 行目から最終行まで指定したい場合，次のように指定します。

```
adshcvshow -l 21- ascファイルのパス名
```

(2) カバレージ情報の表示形式

カバレージ情報の表示形式の説明を，次の表に示します。

表 3-9 カバレージ情報の表示形式の説明

項目	説明
JP1/Advanced Shell カバレージ情報（Advanced Shell Coverage Information）	JP1/Advanced Shell で取得したカバレージ情報を示します。
日時（右上の部分）	adshcvshow コマンドを実行した日時を yyyy-mm-dd hh:mm:ss の形式で表示します。月，日，時，分，秒が 1 桁の場合は，先頭に 0 を付加します。
見出し情報（Header Information）	見出しを示します。
ジョブ定義スクリプト名（Shellscript Name）	ジョブ定義スクリプトの絶対パス名を表示します。
asc ファイルのバージョン（Asc version）	asc ファイルのバージョン番号を表示します。
カバレージ情報の採取開始時刻（Coverage Start Time）	カバレージ情報の採取開始時刻を表示します。形式は，日時と同じです。
カバレージ情報の採取終了時刻（Coverage End Time）	カバレージ情報の採取終了時刻を表示します。形式は，日時と同じです。
カバレージ情報の採取回数（Test Count）	カバレージ情報を採取した回数を表示します。 カバレージ情報の採取回数が 9999 を超えた場合，「9999+」と表示します。 バッチジョブを実行するコマンド（adshexec コマンド）のオプションの指定によって採取回数の数え方が変わります。 adshexec コマンド（-t と-d を指定） • メモリ上のカバレージ情報の場合 ・初期値 asc ファイルがあるときは，asc ファイルのカバレージ情報の採取回数となります。 asc ファイルがないときは，0 となります。 ・更新 デバッガの run コマンドの実行ごとにカバレージ情報の採取回数を 1 増やします。 • asc ファイルの場合 adshexec コマンドの終了時にデバッガの run コマンドの実行回数だけカバレージ情報の採取回数を増やします。 adshexec コマンド（-t を指定） adshexec コマンドの終了時にカバレージ情報の採取回数を 1 増やします。 adshexec コマンド（-d を指定） • メモリ上のカバレージ情報の場合 ・初期値 カバレージ情報の採取回数の初期値は 0 になります。 ・更新 デバッガの run コマンドの実行ごとにカバレージ情報の採取回数を 1 増やします。

項目	説明
カバレージ情報の採取回数（Test Count）	• asc ファイルの場合 asc ファイルは更新されません。
メイン情報（Main Information）	カバレージ情報（C0，C1 情報）を表示します。
行番号（Line）	1 から始まります。 行番号が 9999 を超えた場合，「9999+」と表示します。
付加情報（Info）	C0，C1 情報の見出しです。カバレージ情報は，行単位に表示します。コマンドラインが複数行にまたがっている場合，コマンドが存在する行に C0，C1 情報を表示します。 C0，C1 情報のどちらも 32 個以内ならカバレージ情報を記録でき，この項目はスペースとなります。記録できない場合に表示される文字と意味を次に示します。 overC0：C0 情報が 32 を超えています。 overC1：C1 情報が 32 を超えています。 over：C0 情報と C1 情報の両方が 32 を超えています。
C0 情報（C0）	C0 情報を表示します。 @：コマンドが実行されました。 -：コマンドは実行されていません。 行に複数のコマンドが存在する場合，行の先頭から最大 4 個のコマンドまでの C0 情報を 4 文字で表示します。
C1 情報（C1）	C1 情報を表示します。 @：実行パスが実行されました。 -：実行パスは実行されていません。 行に複数の実行パスが存在する場合，行の先頭から最大 4 個の実行パスまでの C1 情報を 4 文字で表示します。
ジョブ定義スクリプト（<Shellscript Image>）	ジョブ定義スクリプトの内容を，行単位に表示します。範囲指定した場合，指定された範囲の行だけを表示します。
合計の情報（Total Information）	C0，C1 情報の合計を表示します。範囲指定した場合，合計の情報以降は，表示されません。個数が 99999999 を超えた場合，「99999999+」と表示します。
C0，C1 対象の総数（Target Total）	<C0>に対象コマンドの総数，<C1>に実行パスの総数を表示します。
<C0>	ジョブ定義スクリプト内のすべての C0 対象コマンドの個数を含みます。C0 対象のコマンド数が 32 個を超える行が存在する場合でもすべての個数がカウントされます。
<C1>	ジョブ定義スクリプト内のすべての C1 対象実行パスの個数を含みます。C1 対象の実行パス数が 32 個を超える行が存在する場合でもすべての個数がカウントされます。
C0，C1 対象の実行した総数（Executed Total）	<C0>に実行したコマンドの総数，<C1>に実行した実行パスの総数を表示します。

項目	説明
<C0>	カバレージ情報として，実行を記録するのは，各行内で先頭から C0 対象である 32 個のコマンドまでです。この 32 個のコマンド内の実行したコマンドが，カウントされます。
<C1>	カバレージ情報として，実行を記録するのは，各行内で先頭から C1 対象である 32 個の実行パスまでです。この 32 個の実行パス内の実行した実行パスが，カウントされます。
C0，C1 対象の未実行の総数（Unexecuted Total）	<C0>に未実行のコマンドの総数，<C1>に未実行の実行パスの総数を表示します。
<C0>	「C0 対象の総数（Target Total）－C0 対象の実行した総数（Executed Total）」になります。
<C1>	「C1 対象の総数（Target Total）－C1 対象の実行した総数（Executed Total）」になります。
実行比率（Coverage Rate）	C0 と C1 の実行比率（%）を表示します。小数点第 2 位以下を切り捨てて小数点第 1 位までを表示します。
<C0>	「C0 対象の実行した総数（Executed Total）／C0 対象の総数（Target Total）」になります。 C0 対象のコマンド数が 32 個を超える行が存在した場合，すべてのコマンドを実行しても 100%より小さくなります。
<C1>	「C1 対象の実行した総数（Executed Total）／C1 対象の総数（Target Total）」になります。 C1 対象の実行パス数が 32 個を超える行が存在した場合，すべての実行パスを実行しても 100%より小さくなります。

カバレージ情報を表示するコマンドの表示例を，C0，C1 に情報を行ごとに 1 個まで表示した場合と 4 個まで表示した場合とに分けて示します。

(a) カバレージ情報を表示するコマンドの表示例（C0，C1 に情報を行ごとに 1 個まで表示した場合）

この例では，C0 と C1 が取得されたことが，Main Information のところに 1 個の「@」と「-」で表示されています。

```
                    * Advanced Shell Coverage Information *

                                                        2013-12-06 12:29:15
****    Header Information    ***************************************************
Shellscript Name     : /home/testuser/sample
Asc version          : 1.0
Coverage Start Time  : 2013-12-06 12:21:38
Coverage End Time    : 2013-12-06 12:21:39
Test Count           :    1

****    Main Information      ***************************************************
Line   Info   C0   C1   <Shellscript Image>
    1
```

```
    2         @         echo 1
    3
    4         @    @    if true
    5                   then
    6         @           echo 2
    7              -    fi
    8
    9         @         echo 3
   10
   11         @    -    if false
   12                   then
   13         -           echo 4
   14              @    fi
   15
   16         @         echo 5
   17
   18         @    @    if true
   19                   then
   20         @           echo 6
   21              -    else
   22         -           echo 7
   23                   fi
   24
   25         @         echo 8
   26
   27         @    -    if false
   28                   then
   29         -           echo 9
   30              @    else
   31         @           echo 10
   32                   fi
   33
   34         @         echo 11
   35
   36

****    Total Information     ************************************************
                          <C0>      <C1>
  Target     Total          15         8
  Executed   Total          12         4
  Unexecuted Total           3         4
-------------------------------------------------------------------------------
                          <C0>      <C1>
  Coverage   Rate         80.0 %    50.0 %
```

(b) カバレージ情報を表示するコマンドの表示例（C0，C1 に情報を行ごとに 4 個まで表示した場合）

この例では，C0 と C1 が複数回取得されたことが，Main Information の 13 行目と 37 行目に表示されています。

- Line の 13 行は，Line の 3 行から 7 行までを 1 行に記述した場合です。
  C0 の列の"@@@@"の各文字は，先頭から，"echo 1"，"echo 2"，"echo 3"，"echo 4"の各コマンドを実行したことを示します。

C0 の列の"@@@@"には，"echo 5"のコマンドを実行したことを示す情報はありません。

- Line の 37 行は，Line の 20 行から 31 行までを 1 行に記述した場合です。
  - C0 の列の"@@--"の各文字は，先頭から順に対応します。

    1 文字目の"@"は，"true"のコマンドを実行したことを示します。

    2 文字目の"@"は，"echo 1"のコマンドを実行したことを示します。

    3 文字目の"-"は，先頭から 1 個目の"elif"の次にある"true"のコマンドを実行しなかったことを示します。

    4 文字目の"-"は，"echo 2"のコマンドを実行しなかったことを示します。

    上記以外のコマンド，つまり，先頭から 2 番目の"elif"の次にある"true"以降のコマンドについての実行の有無は，C0 の列の"@@--"の各文字には表示しません。
  - C1 の列の"@---"の各文字は，先頭から順に対応します。

    1 文字目の"@"は，"if"の最初の"then"の実行パスを実行したことを示します。

    2 文字目の"-"は，先頭から最初の"elif"の"then"の実行パスを実行しなかったことを示します。

    3 文字目の"-"は，先頭から 2 番目の"elif"の"then"の実行パスを実行しなかったことを示します。

    4 文字目の"-"は，"else"の実行パスを実行しなかったことを示します。
- Line の 73 行は，Line の 43 行から 67 行までを 1 行に記述した場合です。

  C0 の列の"@@--"の各文字の意味，C1 の列の"@---"の各文字の意味は，Line の 37 行と同じです。

  2 番目の"if"の"then"の実行パス（先頭から 5 番目の実行パス）以降の実行パスの実行の有無は，C1 の列の"@---"の各文字には表示しません。

```
                    * Advanced Shell Coverage Information *

                                                        2013-12-06 12:33:01
****    Header Information    ***************************************************
Shellscript Name     : /home/testuser/sample1.ash
Asc version          : 1.0
Coverage Start Time  : 2013-12-06 12:21:49
Coverage End Time    : 2013-12-06 12:21:50
Test Count           :    1

****    Main Information      ***************************************************
Line   Info   C0   C1   <Shellscript Image>
    1
    2
    3         @         echo 1
    4         @         echo 2
    5         @         echo 3
    6         @         echo 4
    7         @         echo 5
    8
    9
   10
   11
   12
   13         @@@@      echo 1;echo 2;echo 3;echo 4;echo 5
   14
   15
```

```
   16
   17
   18
   19
   20         @    @    if true
   21                   then
   22         @          echo 1
   23         -    -    elif true
   24                   then
   25         -          echo 2
   26         -    -    elif true
   27                   then
   28         -          echo 3
   29              -    else
   30         -          echo 4
   31                   fi
   32
   33
   34
   35
   36
   37         @@-- @--- if true ;then echo 1 ;elif true ;then echo 2 ;elif true ;then
echo 3 ;else echo 4 ;fi
   38
   39
   40
   41
   42
   43         @    @    if true
   44                   then
   45         @          echo 1
   46         -    -    elif true
   47                   then
   48         -          echo 2
   49         -    -    elif true
   50                   then
   51         -          echo 3
   52              -    else
   53         -          echo 4
   54                   fi
   55
   56         @    @    if true
   57                   then
   58         @          echo 5
   59         -    -    elif true
   60                   then
   61         -          echo 6
   62         -    -    elif true
   63                   then
   64         -          echo 7
   65              -    else
   66         -          echo 8
   67                   fi
   68
   69
   70
   71
   72
```

```
   73         @@-- @--- if true ;then echo 1 ;elif true ;then echo 2 ;elif true ;then
echo 3 ;else echo 4 ;fi; if true ;then echo 5 ;elif true ;then echo 6 ;elif true ;then
echo 7 ;else echo 8 ;fi
   74

****    Total Information     ************************************************
                          <C0>      <C1>
  Target     Total          52        24
  Executed   Total          22         6
  Unexecuted Total          30        18
--------------------------------------------------------------------------------
                          <C0>      <C1>
  Coverage   Rate         42.3 %    25.0 %
```

(3) C0 情報と C1 情報の表示方法

ジョブ定義スクリプトのスクリプト制御文の実行のしかたでカバレージ情報を採取する個所が変わります。カバレージ情報を表示した場合に実行された個所に「@」が表示されます。実行されなかった個所には，「-」が表示されます。

(a) if 文の場合

- else がないときの表示方法

  then のパスを実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    if           ←  C1を取得
@           true       ←  C0を取得
          then
@           cmd2       ←  C0を取得
@           cmd3       ←  C0を取得
     -    fi           ←  C1を取得しない
```

  then のパスを実行しなかった場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     -    if           ←  C1を取得しない
@           false      ←  C0を取得
          then
-           cmd2       ←  C0を取得しない
-           cmd3       ←  C0を取得しない
     @    fi           ←  C1を取得
```

  then のパスと then でないパスの両方を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    if           ←  C1を取得
@           false      ←  C0を取得
          then
@           cmd2       ←  C0を取得
@           cmd3       ←  C0を取得
     @    fi           ←  C1を取得
```

- else があるときの表示方法

then を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    if          ←  C1を取得
@           true       ←  C0を取得
          then
@           cmd2       ←  C0を取得
     -    else        ←  C1を取得しない
-           cmd3       ←  C0を取得しない
          fi          ←  なし
```

else を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     -    if          ←  C1を取得しない
@           false      ←  C0を取得
          then
-           cmd2       ←  C0を取得しない
     @    else        ←  C1を取得
@           cmd3       ←  C0を取得
          fi          ←  なし
```

then および else の両方を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    if          ←  C1を取得
@           false      ←  C0を取得
          then
@           cmd2       ←  C0を取得
     @    else        ←  C1を取得
@           cmd3       ←  C0を取得
          fi          ←  なし
```

(b)　for 文の場合

- ループを実行する場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    for         ←  C1を取得
          do
@           cmd1       ←  C0を取得
     -    done        ←  C1を取得しない
```

- ループを実行しなかった場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     -    for         ←  C1を取得しない
          do
-           cmd1       ←  C0を取得しない
     @    done        ←  C1を取得
```

- ループを実行する場合およびループを実行しなかった場合の両方を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    for         ←  C1を取得
```

```
          do
@           cmd1        ←  C0を取得
     @    done        ←  C1を取得
```

(c)　while 文および until 文の場合

while 文の表示方法を示します。until 文も同じ表示になります。

- ループを実行する場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    while       ←  C1を取得
          do
@           cmd1        ←  C0を取得
     -    done        ←  C1を取得しない
```

- ループを実行しなかった場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     -    while       ←  C1を取得しない
          do
-           cmd1        ←  C0を取得しない
     @    done        ←  C1を取得
```

- ループを実行する場合およびループを実行しなかった場合の両方を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
     @    while       ←  C1を取得
          do
@           cmd1        ←  C0を取得
     @    done        ←  C1を取得
```

(d)　case 文の場合

*パターンの有無で，C1 情報の表示方法が異なります。*パターンとは，case 文でどのパターンにも一致しなかった場合のパターンです。

- *パターンがある

  esac に C1 情報を表示しません。

- *パターンがない

  esac に C1 情報を表示します。

- *パターンがあるときの表示方法

  ケース 1 を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
          case $A in
     @      1)            ←  C1を取得
@               echo ”abc”  ←  C0を取得
                ;;
     -      *)            ←  C1を取得しない
```

```
-               echo ”efg”  ←  C0を取得しない
                ;;
          esac              ←  なし
```

*パターンを実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
          case $A in
     -      1)             ←  C1を取得しない
                echo ”abc”
                ;;
     @      *)             ←  C1を取得
@               echo ”efg”  ←  C0を取得
                ;;
          esac              ←  なし
```

ケース 1 および*パターンの両方を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
          case $A in
     @      1)             ←  C1を取得
@               echo ”abc”  ←  C0を取得
                ;;
     -      *)             ←  C1を取得しない
-               echo ”efg”  ←  C0を取得しない
                ;;
          esac              ←  なし
```

- *パターンがないときの表示方法

ケース 1 を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
          case $A in
     @      1)             ←  C1を取得
@               echo ”abc”  ←  C0を取得
                ;;
     -      2)             ←  C1を取得しない
-               echo ”efg”  ←  C0を取得しない
                ;;
     -    esac             ←  C1を取得しない
```

ケース 2 を実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
          case $A in
     -      1)             ←  C1を取得しない
-               echo ”abc”  ←  C0を取得しない
                ;;
     @      2)             ←  C1を取得
@               echo ”efg”  ←  C0を取得
                ;;
     -    esac             ←  C1を取得しない
```

*パターンを実行した場合は，次のように表示されます。

```
C0   C1   ジョブ定義スクリプト
@         case $A in       ←  C0を取得
```

```
     -      1)              ←  C1を取得しない
-               echo ”abc”  ←  C0を取得しない
                ;;
     -      2)              ←  C1を取得しない
-               echo ”efg”  ←  C0を取得しない
                ;;
     @    esac              ←  C1を取得
```

(e)　#-adsh_step_start コマンドの場合

#-adsh_step_start コマンドの次の引数を指定した場合，先行のジョブステップやジョブ定義スクリプト中のスクリプト拡張コマンドの状態によって，そのジョブステップを実行するかどうかが変わります。

[-run　{normal|abnormal|always}]

ジョブステップを実行したかどうかを判断できるように，C1 情報に次の情報を表示します。

- --：この#-adsh_step_start まで実行が到達していません。
- N-：先行のジョブステップ，またはジョブ定義スクリプトが正常です。
- -A：先行のジョブステップ，またはジョブ定義スクリプトが異常です。
- NA：「N-」と「-A」の両方のケースを実行しました。

(f)　#-adsh_step_error コマンドの場合

ジョブステップ内でエラーが発生した場合，#-adsh_step_error コマンドに続くジョブ定義スクリプトを実行します。このエラー処理を実行したかどうかを判断できるように，C1 情報に次の情報を表示します。

- --：この#-adsh_step_error を含むジョブステップまで実行が到達していません。
- N-：ジョブステップ内でエラーは発生していないため，エラー処理を実行していません。
- -E：ジョブステップ内でエラーは発生していたため，エラー処理を実行しました。
- NE：「N-」と「-E」の両方のケースを実行しました。

(g)　関数の場合

関数の実行例を次に示します。

```
C0   C1    ジョブ定義スクリプト
           funcAAA(){                  →1.
@              echo ”start funcAAA”    →2.
     @         if true                 →2.
               then                    →2.
@                echo true             →2.
     -         else                    →2.
-                echo false            →2.
               fi                      →2.
           }
              :
@          funcAAA                     →3.
```

1. 関数を実行した場合には，関数が定義されているところには，C0 情報，C1 情報を表示しません。
2. 関数の本体には，実行したコマンド，実行パスの C0，C1 情報を表示します。
3. 関数を実行した場合には，関数を呼び出したところに C0 情報を表示します。

(h)　(cmd1; cmd 2)の場合

括弧で囲ったコマンドを別プロセスで実行します。この場合，コマンドグループ全体，コマンドグループ内の各コマンドに対して，カバレージ情報は採取しません。

(i)　{ cmd1; cmd2 }の場合

中括弧で囲んだコマンドを，adshexec コマンドと同一プロセスで実行します。この場合，コマンドグループ内の各コマンドに対して，カバレージ情報を採取します。

(j)　cmd1 &の場合

adshexec コマンドによるジョブ定義スクリプトの実行と並行して，バックグラウンドで別プロセスを生成してコマンドを実行します。バックグラウンドで実行されるジョブ定義スクリプトでは，カバレージ情報は採取しません。

(k)　trap のアクションの場合

trap のアクションでは，カバレージ情報を採取しません。

- 例

```
trap ”date; echo xxx” INT
```

(l)　コマンド置換の場合

コマンド置換で実行するコマンド，スクリプト制御文のカバレージ情報は採取しません。

- 例

```
ls `which adshexec`
```

(m)　time コマンドの引数の場合

time コマンドの引数として実行するコマンドは，カバレージ情報を採取しません。

- 例

```
time adshexec script1
```

(n)　eval コマンドの引数の場合

eval コマンドの引数として実行するコマンドは，カバレージ情報を採取しません。

- 例

```
eval ls dir1
```

(o) パイプ機能の場合

パイプ機能を使用して実行しているコマンドのカバレージ情報は採取しません。

- 例

```
ls | cat
```

(p) 外部スクリプトの場合

外部スクリプトの呼び出し先では，カバレージ情報を採取しません。外部スクリプトの呼び出し元では，カバレージ情報を採取します。なお，外部スクリプトの呼び出しは，C0 の対象ですが，C1 については，対象外です。

(4) メモリ上に採取しているカバレージ情報の表示【UNIX 限定】

デバッグ用の info coverage コマンドを使用した場合，メモリ上に採取している，カバレージ情報を表示できます。

表示するカバレージ情報は，初回蓄積か継続蓄積かどうかで変化します。初回蓄積の場合は，中断点までのカバレージ情報を表示します。継続蓄積の場合は，蓄積されたカバレージ情報に中断点までのカバレージ情報を合わせた結果を表示します。蓄積を指定しなかった場合は，初回蓄積の場合と同様に中断点までのカバレージ情報を表示します。

表示する情報の種類と形式は，カバレージ情報を表示するコマンド（adshcvshow コマンド）の場合と同じです。

(5) C1 実行比率 100%とならないケース

#-adsh_step_start コマンドを使用していて，そのジョブステップに先行するジョブステップ，またはコマンドがない場合，すべての実行パスを実行しても C1 実行比率が 100%となりません。#-adsh_step_start コマンドは次の両方のケースを C1 情報として取得しますが，#-adsh_step_start コマンドで定義するジョブステップに先行するジョブステップまたはコマンドがない場合，2.のケースを実行できないためです。

1. 先行するすべてのジョブステップ，コマンドが正常終了している。
2. 先行するジョブステップ，コマンドで正常終了していないものがある。

この場合，デバッグ時にエラー注入モードを有効にして該当個所でエラーをシミュレートする方法があります。これによって C1 情報が取得され，C1 実行比率を 100%にすることができます。エラーのシミュレート方法を次に示します。

- GUI でのデバッグの場合【Windows 限定】

JP1/Advanced Shell エディタのメニュー［エラー注入モード］でエラーをシミュレートできます。手順については，「(4)　エラーをシミュレートする」を参照してください。

- CUI でのデバッグ（adshexec コマンドの-d オプションで開始）の場合【UNIX 限定】

  joberrmode コマンドを使用することでエラーをシミュレートできます。joberrmode コマンドについては，「6.2.21　エラー注入モードの有効/無効を設定する（joberrmode コマンド）」を参照してください。

  なお，エラー注入モードが有効かどうかは info status コマンドで確認できます。info status コマンドについては「6.2.19　ステータスを表示する（info status コマンド）」を参照してください。

3.10.5　カバレージ情報のマージ

カバレージ情報をマージする目的は，複数のユーザーが同一のジョブ定義スクリプトのテストを行った結果を 1 つにすることです。あるジョブ定義スクリプトのテストケースを複数の人で分担してテストした場合，カバレージ情報を 1 つにまとめて作業のむだを省くことができます。

(1)　マージの方法

カバレージ情報は，カバレージ情報をマージするコマンド（adshcvmerg コマンド）でマージします。コマンドに指定した 2 つの asc ファイルをマージします。コマンドの形式を次に示します。

```
adshcvmerg -o 出力先ファイル ascファイル1 ascファイル2
```

asc ファイル 1 と asc ファイル 2 の情報をマージし，その結果を出力先ファイルに asc ファイル形式で出力します。

(2)　マージする情報の種類

マージする情報はテスト回数やカバレージ情報です。例えば，adshcvmerg -o out c1 c2 コマンドを入力して，マージ処理を実行した場合，情報は次のように変更されます。

- ジョブ定義スクリプトのフルパス名：c1 のフルパス名
- テスト回数：c1 のテスト回数＋ c2 のテスト回数
- カバレージ情報の採取開始日時：c1 と c2 で早い開始日時
- カバレージ情報の採取終了日時：c1 と c2 で遅い終了日時

3.10.6　カバレージ採取の一括有効化機能

カバレージ採取の一括有効化機能を使用すると，adshexec コマンドのパラメーターを変更することなく，カバレージを採取できるようにします。

次の環境設定パラメーターでカバレージ情報を採取するように設定しておけば，adshexec コマンドによるバッチジョブの実行時にカバレージ情報を採取するオプション（-t）を指定しなくても有効にします。

- BATCH_CVR パラメーター：カバレージ採取の一括有効化機能を使用する
- ASC_FILE パラメーター：カバレージ採取の一括有効化機能で使用する蓄積ファイル名の生成規則を定義する

コマンドの指定例を次に示します。

```
adshexec ジョブ定義スクリプト名.ash
```

カバレージ情報を採取するオプション（-t および-o）を指定しないでジョブ定義スクリプトを実行します。

カバレージ採取の一括有効化機能を使用する場合，adshexec コマンドに-t オプションを指定できません。したがって，-t オプションを指定して「adshexec -t sample.ash」と指定して adshexec コマンドを実行したときは，終了コード 1 となり，エラー終了します。

環境ファイルに#-adsh_conf BATCH_CVR YES を設定することで，カバレージ情報を採取できます。

カバレージ採取の一括有効化機能でカバレージ機能を有効化すると，asc ファイル（カバレージ情報ファイル）は，各コマンドを実行したカレントディレクトリに出力します。

環境ファイルに#-adsh_conf ASC_FILE cvr/ver001-*を指定している場合は，上記のコマンドは，「adshexec -t -o cvr/ver001-ジョブ定義スクリプト名 ジョブ定義スクリプト名.ash」と指定して adshexec コマンドを実行したときと同じ動作となります。

コマンドごとにカレントディレクトリが異なる場合，asc ファイルを作成するディレクトリがさまざまなディレクトリに分散します。#-adsh_conf ASC_FILE を指定することで，asc ファイルを出力するディレクトリを特定のディレクトリに設定できます。また，asc ファイルのファイル名を統一できます。

環境ファイルの設定の詳細については，「2.6.11　バッチジョブの実行時にカバレージ情報を採取するオプションを指定しなくても有効にする」を参照してください。

3.11　ジョブを強制終了する

ジョブの強制終了について説明します。

3.11.1　ジョブの強制終了の方法

(1)　強制終了の方法

ジョブの強制終了には次の 2 とおりの方法があります。

- JP1/AJS からジョブを起動している場合，JP1/AJS の強制終了操作を実行します。
  JP1/AJS から Windows または UNIX で実行されたジョブアイコンのジョブを強制終了する場合，環境変数 AJS_BJEX_STOP=TERM を設定しておく必要があります。Windows または UNIX で実行されたジョブアイコンのジョブの詳細については，「2.7.2　ジョブネットを定義して実行する」を参照してください。
- adshexec コマンドのプロセスに対して，終了要求シグナルを送付します。Windows の場合は，taskkill コマンドなどを用いて adshexec のプロセスを終了させます。

ジョブを強制終了すると，ジョブコントローラは実行中の子プロセスまたは子孫プロセスを強制終了します。詳細は，「(2)　子プロセスまたは子孫プロセスの強制終了」を参照してください。

子プロセスまたは子孫プロセスを強制終了したあと，割り当てたファイルの後処理をして後続のジョブステップ・コマンドを一切実行しないで終了します。後続ジョブステップの run 属性に abnormal や always が指定されていても実行しません。ジョブを強制終了すると，UNIX では adshexec コマンドがシグナルによってエラー終了します。UNIX での SIGTERM 受信時のジョブの動作については「3.11.2　シグナル受信時の動作【UNIX 限定】」，Windows での強制終了時のジョブの動作については「3.11.3　強制終了時のジョブの動作【Windows 限定】」を参照してください。

重要

Windows の場合，adshexec コマンドの起動時に adshexecsub コマンドをあわせて起動し，adshexec コマンドを強制終了すれば adshexecsub コマンドも終了します。そのため，adshexecsub コマンドは強制終了しないでください。adshexecsub コマンドを強制終了すると，次の現象が発生する場合があります。

- 実行中の子孫プロセスが終了されません。
- 一時ファイルが残ったままとなる場合があります。

これらの現象が発生した場合は，taskkill コマンドやタスクマネージャーを使用して子孫プロセスを強制終了させ，一時ファイルを手動で削除してください。

重要

Windows 環境の JP1/Advanced Shell のジョブコントローラでは，孫プロセスの強制終了のためにジョブオブジェクトを使用しているため，次の 2 点に注意してください。

- ジョブコントローラから生成した子プロセスをジョブオブジェクトに関連づけることはできません。
- ジョブコントローラプロセスが，すでにジョブオブジェクトに関連づけられていた場合，ジョブの強制終了でジョブコントローラの子プロセスが生成したプロセスは終了しません。

重要

Windows では，子孫プロセスを生成する外部コマンドを実行するジョブを強制終了した場合，孫以下のプロセスが同時に 256 個以上存在すると，メッセージ KNAX6381-E を出力してスプールジョブディレクトリの名称変更が失敗することがあるため，次の 3 点に注意してください。

- 失敗したスプールジョブディレクトリを参照する場合は，直後のメッセージ KNAX6382-I に出力されるディレクトリ名を参照してください。
- 名称変更に失敗したスプールジョブディレクトリは adshhk コマンドでは削除されません。削除する場合は，手動で削除してください。
- 実行環境でスプールジョブディレクトリの名称変更に失敗したジョブは，adshevtout コマンドでジョブ定義スクリプト稼働実績情報を出力しません。

(2) 子プロセスまたは子孫プロセスの強制終了

ジョブを強制終了すると，ジョブコントローラは子プロセスまたは子孫プロセスを強制終了してからジョブを終了します。

(a) UNIX の場合

ジョブの入力モードによって，子プロセスまたは子孫プロセスの強制終了の方法が次のとおり異なります。

- 端末入力モード

  adshexec コマンドの子プロセスだけに SIGTERM を送信します。adshexec コマンドの孫プロセス以下には，SIGTERM を送信しません。これらのプロセスの後処理を実行したい場合は，次のどちらかの方法でジョブを作成・実行してください。

  - 外部コマンドをユーザーが作成する場合，SIGTERM を受信したら子孫プロセスにも自動的に SIGTERM を送信するなど，子孫プロセスの後処理を外部コマンド側で実行するようあらかじめ設計してください。
  - 端末入力モードのジョブを強制終了する場合，「Ctrl+C」や「Ctrl+¥」などの操作は，adshexec コマンドだけでなく，孫プロセス以下を含むすべての子孫プロセスが対象になります。

強制終了後に孫プロセス以下が残ってしまった場合は，残ってしまったプロセスのプロセス ID を ps コマンドで確認し，kill コマンドによって手動で終了させてください。

- 非端末入力モード

  adshexec コマンドの子孫プロセスに SIGTERM を送信します。

(b) Windows の場合

adshexec コマンドの子孫プロセスを TerminateProcess 関数および TerminateJobObject 関数で強制終了します。ジョブの入力モードによる強制終了方法の差異はありません。

(3) Ctrl+C などの操作に関する注意事項【UNIX 限定】

非端末入力モードでジョブを実行した場合，ルートジョブ，子孫ジョブ，およびほかに起動した外部コマンドを「Ctrl+C」や「Ctrl+¥」などの操作で一度に強制終了させることができないときがあります※。これらのジョブおよびコマンドを一度にすべて強制終了させたいときは，ログインシェル直下のルートジョブに対して，kill コマンドで SIGTERM などの終了要求シグナルを送信してください。

注※

非端末入力モードでジョブを実行した場合，adshexec コマンドのプロセスとその子プロセスとは，別々のプロセスグループになります。そのため，ログインシェルからジョブを実行中に「Ctrl+C」や「Ctrl+¥」などの操作を実行した場合，現在フォアグラウンドにいるプロセスグループに対してだけ，SIGINT や SIGQUIT が送られます。

シグナルを受信したジョブの子孫プロセスとして動作するジョブおよび外部コマンドは強制終了されますが，親プロセスやそれより上位のプロセスとして動作するジョブおよび外部コマンドは強制終了されません。

3.11.2 シグナル受信時の動作【UNIX 限定】

通常実行時とデバッグ実行時の，ジョブコントローラがシグナルを受信した場合の動作を説明します。

(1) 通常実行時

通常実行時にジョブコントローラがシグナルを受信した場合の動作について，SIGTERM シグナルと，それ以外に分けて示します。

(a) SIGTERM の場合

SIGTERM を受信した場合の動作は環境設定パラメーター TRAP_ACTION_SIGTERM の指定に従います。

表 3-10　環境設定パラメーター TRAP_ACTION_SIGTERM に DISABLE を指定した場合※1

trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
• ルートジョブが SIGTERM を受信した場合 1 回目：メッセージを出力し，後処理を実行してから，後続処理を実行しないで終了します。このとき JP1/AJS からの起動でない場合は，自分に SIGTERM を送信してシグナル終了します。 2 回目：即時終了します。 • 子孫ジョブが SIGTERM を受信した場合 SIGTERM を受信した子孫ジョブは，メッセージを出力し，後処理を実行してから，後続処理を実行しないで終了します。このとき自分に SIGTERM を送信してシグナル終了します。 また，子孫ジョブの親ジョブは，子プロセスが終了コード「128 ＋ SIGTERM のシグナル番号」で終了した場合の動作に従って，後続処理を実行します。※2	trap コマンドで動作を定義できません。※3

注※1

環境設定パラメーター TRAP_ACTION_SIGTERM を指定しなかった場合も含みます。

注※2

子孫ジョブのシグナル受信時の動作の詳細については，「(3)　シグナル受信時の子孫ジョブの動作」を参照してください。

注※3

trap コマンドで SIGTERM に対する動作を定義しようとすると，trap コマンドがエラーとなりジョブが終了します。

表 3-11　環境設定パラメーター TRAP_ACTION_SIGTERM に TERM を指定した場合

trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
• ルートジョブが SIGTERM を受信した場合 1 回目：メッセージを出力し，後処理を実行してから，後続処理を実行しないで終了します。このとき JP1/AJS からの起動でない場合は，自分に SIGTERM を送信してシグナル終了します。 2 回目：即時終了します。 • 子孫ジョブが SIGTERM を受信した場合 SIGTERM を受信した子孫ジョブは，メッセージを出力し，後処理を実行してから，後続処理を実行しないで終了します。このとき自分に SIGTERM を送信してシグナル終了します。 また，子孫ジョブの親ジョブは，子プロセスが終了コード「128 ＋ SIGTERM のシグナル番号」で終了した場合の動作に従って，後続処理を実行します※。	• ルートジョブが SIGTERM を受信した場合 メッセージを出力し，trap コマンドで SIGTERM に対して定義したアクションを実行します。アクションを実行した後は後続処理を実行しないで終了します。JP1/AJS からの起動でない場合は，自分に SIGTERM を送信してシグナル終了します。 • 子孫ジョブが SIGTERM を受信した場合 メッセージを出力し，trap コマンドで SIGTERM に対して定義したアクションを実行します。アクションを実行した後は後続処理を実行しないで終了します。 また，子孫ジョブの親ジョブは，子プロセスが SIGTERM を受信した動作結果に従って，後続処理を実行します※。

注※

子孫ジョブのシグナル受信時の動作の詳細については，「(3)　シグナル受信時の子孫ジョブの動作」を参照してください。

表 3-12　環境設定パラメーター TRAP_ACTION_SIGTERM に CONT を指定した場合

ジョブの起動方法	trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
JP1/AJS から起動した場合 （カスタムジョブからの起動，または環境変数 AJS_BJEX_STOP に TERM を設定した状態での起動）	ジョブ定義スクリプトを実行しないでジョブがエラー終了します（環境ファイル解析時のエラー）。	
JP1/AJS 以外から起動した場合 （上記以外の方法による起動）	• ルートジョブが SIGTERM を受信した場合 1 回目：メッセージを出力し，後処理を実行してから，後続処理を実行しないで終了します。このとき自分に SIGTERM を送信してシグナル終了します。 2 回目：即時終了します。 • 子孫ジョブが SIGTERM を受信した場合 SIGTERM を受信した子孫ジョブの動作は，ルートジョブと同じです。 また，子孫ジョブの親ジョブは，子プロセスが終了コード「128 + SIGTERM のシグナル番号」で終了した場合の動作に従って，後続処理を実行します※。	• ルートジョブが SIGTERM を受信した場合 メッセージを出力し，trap コマンドで SIGTERM に対して定義したアクションを実行します。アクションを実行した後はジョブ定義スクリプトの後続処理を続行します。 • 子孫ジョブが SIGTERM を受信した場合 SIGTERM を受信した子孫ジョブの動作は，ルートジョブと同じです。 また，子孫ジョブの親ジョブは，子プロセスが SIGTERM を受信した動作結果に従って，後続処理を実行します※。

注※

子孫ジョブのシグナル受信時の動作の詳細については，「(3)　シグナル受信時の子孫ジョブの動作」を参照してください。

表 3-13　環境設定パラメーター TRAP_ACTION_SIGTERM に AUTO を指定した場合

ジョブの起動方法	trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
JP1/AJS から起動した場合 （カスタムジョブからの起動，または環境変数 AJS_BJEX_STOP に TERM を設定した状態での起動）	TERM を指定した場合と同じ動作です。	
JP1/AJS 以外から起動した場合 （上記以外の方法による起動）	CONT を指定した場合と同じ動作です。	

(b) SIGTERM 以外の場合

表 3-14 シグナル受信時の動作

シグナルの種類		trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
終了要求シグナル	SIGHUP，SIGINT，SIGXCPU，SIGXFSZ，SIGQUIT，SIGUSR1，SIGUSR2，SIGPIPE，SIGALRM，SIGVTALRM，SIGPROF	• ルートジョブがシグナルを受信した場合 子孫プロセスの終了や一時ファイルの削除などの後処理を実行してから，後続命令を実行しないでシグナルによってエラー終了します。 • 子孫ジョブがシグナルを受信した場合 受信した子孫ジョブの動作はルートジョブ受信時と同じです。 ここで受信した子孫ジョブの親のジョブは，終了した子孫ジョブの結果に従って，後続処理を実行します。※1	• ルートジョブがシグナルを受信した場合 trap コマンドで定義した動作に従います。 • 子孫ジョブがシグナルを受信した場合 受信した子孫ジョブの動作はルートジョブ受信時と同じです。 ここで受信した子孫ジョブの親のジョブは，子孫ジョブの結果に従って，後続処理を実行します。
	SIGMSG，SIGDANGER，SIGMIGRATE，SIGPRE，SIGVIRT，SIGALRM1，SIGRECONFIG，SIGCPUFAIL，SIGGRANT，SIGRETRACT，SIGSOUND	上記と同様です。 【AIX 限定】	上記と同様です。 【AIX 限定】
	SIGLOST	上記と同様です。 【HP-UX，Solaris 限定】	上記と同様です。 【HP-UX，Solaris 限定】
異常通知シグナル	SIGILL，SIGTRAP，SIGABRT，SIGFPE，SIGBUS，SIGSEGV，SIGSYS	• ルートジョブがシグナルを受信した場合 対象シグナルに対する OS のデフォルトの動作に従ってプログラムを終了します。 • 子孫ジョブがシグナルを受信した場合 受信した子孫ジョブの動作はルートジョブ受信時と同じです。 ここで受信した子孫ジョブの親のジョブは，終了した子孫ジョブの結果に従って，後続処理を実行します。※1	• ルートジョブがシグナルを受信した場合 trap コマンドで定義した動作に従います。 • 子孫ジョブがシグナルを受信した場合 受信した子孫ジョブの動作はルートジョブ受信時と同じです。 ここで受信した子孫ジョブの親のジョブは，子孫ジョブの結果に従って，後続処理を実行します。
	SIGIOT，SIGEMT	上記と同様です。 【AIX，HP-UX，Solaris 限定】	上記と同様です。 【AIX，HP-UX，Solaris 限定】
	SIGLOST	上記と同様です。 【AIX 限定】	上記と同様です。 【AIX 限定】

シグナルの種類	trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
上記以外	• ルートジョブがシグナルを受信した場合 対象シグナルに対する OS のデフォルトの動作に従います。 • 子孫ジョブがシグナルを受信した場合 受信した子孫ジョブの動作はルートジョブ受信時と同じです。 ここで受信した子孫ジョブの親のジョブは，子孫ジョブの結果に従って，後続処理を実行します。※1	• ルートジョブがシグナルを受信した場合 trap コマンドで定義した動作に従います。※2 • 子孫ジョブがシグナルを受信した場合 受信した子孫ジョブの動作はルートジョブ受信時と同じです。 ここで受信した子孫ジョブの親のジョブは，子孫ジョブの結果に従って，後続処理を実行します。

注※1

子孫ジョブのシグナル受信時の動作の詳細については，「(3)　シグナル受信時の子孫ジョブの動作」を参照してください。

注※2

SIGKILL と SIGSTOP については，trap コマンドで動作を定義できません。

SIGWAITING については，trap コマンドで動作を定義できません。【AIX 限定】

重要

- trap コマンド使用時，動作に「-」を設定するとシグナル受信時の動作がデフォルトに戻ります。
- デバッグ実行時の動作がこれらの表と異なるシグナルがあります。trap コマンドによる動作定義の有無とシグナルの動作の差異については，「(2)　デバッグ実行時」を参照してください。

(2)　デバッグ実行時

表 3-15　デバッグ実行時のシグナル受信時の動作

シグナルの種類	trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
SIGINT	デバッガがジョブ定義スクリプトの実行を停止させて，デバッガのコマンド入力待ちとなります。※	デバッガがジョブ定義スクリプトの実行を停止させて，デバッガのコマンド入力待ちとなります。※ また，trap コマンドで定義した動作に従います。
SIGCHLD, SIGTSTP,	後続処理を続行します。	trap コマンドで定義した動作に従います。

シグナルの種類	trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
SIGTTOU, SIGURG, SIGWINCH, SIGIO, SIGPWR	後続処理を続行します。	trap コマンドで定義した動作に従います。
SIGSTKFLT 【Linux 限定】		
SIGWAITING，SIGLWP，SIGFREEZE，SIGTHAW，SIGCANCEL，SIGXRES，SIGJVM1，SIGJVM2 【Solaris 限定】		
リアルタイムシグナル 【HP-UX，Linux，Solaris 限定】		

注※

ジョブ定義スクリプトの停止については，「6.2　CUI のデバッガ【UNIX 限定】」を参照してください。

3.11.3　強制終了時のジョブの動作【Windows 限定】

Windows での強制終了時のジョブの動作を次に示します。

なお，trap コマンドによって，TerminateProcess などによるプロセス即時終了に対する動作を定義する場合，TRAP_ACTION_SIGTERM パラメーターに TERM を指定してください。

表 3-16　強制終了時のジョブの動作

強制終了方法		trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
制御信号	CTRL + C CTRL + BREAK CTRL_CLOSE_EVENT	制御信号は，ルートジョブ，子孫ジョブ，およびコマンドとして動作するすべてのプロセスグループに対して送信されます。 • 制御信号を受信したルートジョブ（adshexec.exe）の場合 子プロセス adshexecsub.exe が後続のスクリプトを実行しないで後処理を実行して終了します。制御信号を受信した adshexec.exe	trap コマンドで動作を定義できません

強制終了方法		trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
制御信号	CTRL + C CTRL + BREAK CTRL_CLOSE_EVENT	は子プロセスの終了を待ってから終了します。 • 制御信号を受信したルートジョブ（adshexecsub.exe）および子孫ジョブの動作 制御信号を受信した adshexecsub.exe は，メッセージ KNAX7896-I を出力し，後続のスクリプトを実行しないで後処理を実行して終了します。	trap コマンドで動作を定義できません
	CTRL_LOGOFF_EVENT	OS のログオフやシャットダウン処理を優先するため，後処理を実行しないで即時終了します。	trap コマンドで動作を定義できません
	CTRL_SHUTDOWN_EVENT		
TerminateProcess などによるプロセス即時終了		• 即時終了の対象がルートジョブ（adshexec.exe）の場合 対象となる adshexec.exe は即時終了しますが，その子プロセスの adshexecsub.exe および子孫ジョブは後続のスクリプトを実行しないで後処理を実行して終了します。 • 即時終了の対象がルートジョブ（adshexecsub.exe）の場合 対象となる adshexecsub.exe は後処理を実行しないで即時終了します（このプロセスは即時終了しないでください）。 • 即時終了の対象が子孫ジョブの場合 対象となる子孫ジョブは後処理を実行しないで即時終了します（このプロセスは即時終了しないでください）。このとき，終了させられた子孫ジョブ	• 即時終了の対象がルートジョブ（adshexec.exe）の場合 対象となる adshexec.exe は即時終了しますが，その子プロセスの adshexecsub.exe および子孫ジョブは trap コマンドで定義した動作を実行し，後処理を実行して終了します。 • 即時終了の対象がルートジョブ（adshexecsub.exe）の場合 対象となる adshexecsub.exe は後処理を実行しないで即時終了します（このプロセスは即時終了しないでください）。 • 即時終了の対象が子孫ジョブの場合 対象となる子孫ジョブは後処理を実行しないで即時終了します（このプロセスは即時終了しないでください）。このとき，終了させられた子孫ジョブ

強制終了方法	trap コマンドによる動作定義がない場合	trap コマンドによる動作定義がある場合
TerminateProcess などによるプロセス即時終了	の親のジョブは，子プロセスが終了コード 1 でエラー終了した場合の動作に従って，後処理を実行します。	の親のジョブは，子プロセスが終了コード 1 でエラー終了した場合の動作に従って，後処理を実行します。

注意事項

trap コマンド使用時，動作に「-」を設定すると，該当強制終了要求を受けたときの動作が無効になり，何も設定されていない状態に戻ります。

3.12 アプリケーション実行エージェント機能を使用する【Windows 実行環境限定】

アプリケーション実行エージェント機能は，ログオン空間で動作させたいアプリケーション（コンソールアプリケーション，バッチプログラム，choice コマンドなど）を実行アプリケーションとして起動できるようにします。

この機能を使用することによって，JP1/AJS から GUI を表示し，入力待ちになる実行アプリケーションをジョブとして実行できるようになります。

3.12.1 前提条件

ジョブを実行するユーザーでログインします。

3.12.2 実行方法

実行アプリケーションに Windows 標準のアプリケーションである notepad.exe を使用し，JP1/AJS の PC ジョブから実行した場合を例にして，実行方法を次に示します。

1. JP1/AJS の PC ジョブ定義を行います。
   - 実行アプリケーションの実行終了を待つ場合

     実行ファイル名に JP1/Advanced Shell の GUI アプリケーション実行プログラム（インストール先フォルダ¥JP1ASE¥bin¥adshappexec.exe）を指定します。

     パラメーターに-w と実行アプリケーション名を指定します。ここでの例は「-w notepad.exe」とします。
   - 実行アプリケーションの実行終了を待たない場合

     実行ファイル名に JP1/Advanced Shell の GUI アプリケーション実行プログラム（インストール先フォルダ¥JP1ASE¥bin¥adshappexec.exe）を指定します。

     パラメーターは-n と実行アプリケーション名を指定します。ここでの例は「-n notepad.exe」とします。
2. アプリケーション実行エージェントを起動します。

   JP1/AJS のジョブを実行するユーザーでログインして，次の操作を行います。

   Windows の［スタート］メニューから，［すべてのプログラム］－［Advanced Shell］－［アプリケーション実行エージェント］を選択すると，タスクバーの通知領域に［アプリケーション実行エージェント］アイコンが表示されます。

すでにアプリケーション実行エージェントがスタートアップに登録されている場合は，この操作は不要です。

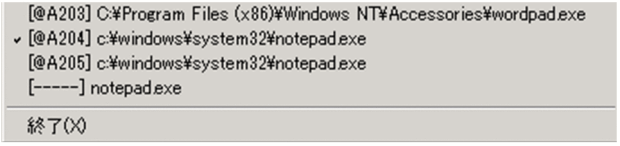
3. JP1/AJS のジョブの実行を行います。
   - 実行アプリケーションの実行終了を待つ場合
     JP1/AJS のジョブを実行すると notepad.exe が起動します。
     notepad.exe を閉じると，ジョブが終了します。
   - 実行アプリケーションの実行終了を待たない場合
     notepad.exe を閉じるのを待たずにジョブが終了します。

3.12.3　アプリケーション実行エージェントの操作

アプリケーション実行エージェントの操作を次に示します。

(1)　アイコンの左クリックメニュー

［アプリケーション実行エージェント］アイコンを左クリックすると，ポップアップメニューに，現在起動中の実行アプリケーションが表示されます。



ポップアップメニューには，次の形式で表示されます。

実行状態　［JP1/AJSのジョブの実行ID］　表示名

実行アプリケーション情報を出力しているエントリを選択した場合，選択した実行アプリケーションのタスクバーを 5 秒間点滅させフォアグランドさせます。

ウィンドウが最小化されていた場合は，最小化のままとします。

GUI を持たないものなど実行アプリケーションによってはフォアグランドしないものもあります。

ポップアップに表示される内容を次に示します。

実行状態	何も表示されていない場合は，実行アプリケーションが正常に起動していることを示します。 ✔ が表示されているエントリは，実行アプリケーションが起動している状態で，JP1/AJS からの強制終了などによってジョブが終了した場合を意味します。 ✔ が表示されている実行アプリケーションは，終了することを推奨します。

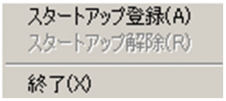
実行状態	adshappexec コマンドの同時実行数の対象になるため，同時実行数が増えると adshappexec コマンドの起動で待ち状態が発生します。
JP1/AJS のジョブの実行 ID	実行アプリケーションを起動した JP1/AJS ジョブの実行 ID が表示されます。 adshappexec コマンドを JP1/AJS から起動していない場合，または JP1/AJS のジョブの実行 ID が求められなかった場合は「-----」と表示されます。 JP1/AJS から実行した場合，ジョブ定義スクリプトからの起動でも JP1/AJS ジョブの実行 ID は表示されます。
表示名	起動中の実行アプリケーションの表示名（adshappexec コマンドの引数-v で指定した内容）が表示されます。 adshappexec コマンドの引数-v を省略した場合は，実行アプリケーション名が表示されます。
終了	アプリケーション実行エージェント機能を使用しない場合に選択します。アプリケーション実行エージェントを終了し，タスクバーの通知領域から［アプリケーション実行エージェント］アイコンを削除します。

表示順序は，共有メモリの格納順となりますが，エントリは空き領域を探して登録するため順序は変動します。

実行アプリケーションを表示しているエントリを選択した場合，選択した実行アプリケーションをフォアグランドできます。GUI を持たないものなど実行アプリケーションによってはフォアグランドできないものもあります。

(2) アイコンの右クリックメニュー

［アプリケーション実行エージェント］アイコンを右クリックすると，次のポップアップメニューが表示されます。



ポップアップに表示される内容を次に示します。

スタートアップ登録	アプリケーション実行エージェントをスタートアップに登録します。スタートアップが登録されていない場合，「スタートアップ登録」は活性化され，スタートアップに登録できます。 すでにスタートアップに登録されている場合，「スタートアップ登録」が非活性になり，スタートアップに登録できません。
スタートアップ解除	アプリケーション実行エージェントをスタートアップから削除します。スタートアップが登録されている場合，「スタートアップ解除」は活性化され，スタートアップから削除できます。 スタートアップ登録されていない場合，「スタートアップ解除」が非活性になり，スタートアップ解除はできません。
終了	アプリケーション実行エージェント機能を使用しない場合に選択します。アプリケーション実行エージェントを終了し，タスクバーの通知領域からアプリケーション実行エージェントアイコンを削除します。

3.12.4 注意事項

- アンインストール時にはアプリケーション実行エージェントを終了し，アプリケーション実行エージェントをスタートアップに登録したユーザーでログインし，スタートアップに登録したアプリケーション実行エージェントを削除する必要があります。

  スタートアップに登録したアプリケーション実行エージェントを削除しないでアンインストールした場合，再度 JP1/Advanced Shell をインストールし，スタートアップが残っているユーザーでログインし，スタートアップに残っているアプリケーション実行エージェントを削除する必要があります。

- アプリケーション実行エージェントをユーザーが手動でスタートアップに追加した場合，アプリケーション実行エージェントをスタートアップに新たに追加する必要はありません。追加した場合は，ログイン時に二重にアプリケーション実行エージェントが起動します。

- アプリケーション実行エージェントアイコンを左クリックしても，次の場合，実行アプリケーションが点滅やフォアグランドしないことがあります。

  1. GUI を持たない実行アプリケーション
  2. 実行アプリケーションの初期化に時間が掛かったもの（5 秒以上）

3.13 スプールジョブ名を指定する

ジョブ終了時，スプールジョブディレクトリ名は「*ジョブ識別子-スプールジョブ名*」に変更されます。このスプールジョブ名には，JP1/Advanced Shell のジョブ名が使用されます。シェル変数 ADSH_SPOOL_JOBNAME を使用して，任意の文字列をスプールジョブ名に指定することもできます。

シェル変数名	指定する値
ADSH_SPOOL_JOBNAME	スプールジョブディレクトリのリネームに使用するスプールジョブ名を指定します。 シェル変数が関数内ローカル変数の場合，指定した値はスプールジョブディレクトリのリネームに使用されません。

ジョブ終了時にスプールジョブディレクトリ名をリネームする際，このシェル変数に指定された値をスプールジョブ名として使用します。このシェル変数を指定していない場合は，JP1/Advanced Shell のジョブ名でリネームします。

スプールジョブ名は，ディレクトリ名として使用できる文字で指定してください。また，スプールジョブ名は，スプールジョブディレクトリ下の各ファイルのパス名の長さが，最大長を超えない範囲で指定してください。これらに違反した場合，スプールジョブディレクトリのリネームに失敗します。シェル変数 ADSH_SPOOL_JOBNAME に指定したスプールジョブ名に次の文字が含まれている場合，_（アンダースコア）に置き換えてリネームします。

アンダースコアに置き換える文字	/，¥，.

また，不当なマルチバイト文字が含まれている場合，不当なマルチバイト文字はバイト単位に_（アンダースコア）に置き換えてリネームします。

3.13.1 使用例

スプールジョブ名を初期設定スクリプトで指定する例を示します。

初期設定スクリプトでは，環境変数AJSJOBNAME の JP1/AJS のジョブ名から文字列を切り出して，シェル変数ADSH_SPOOL_JOBNAME にスプールジョブ名を指定します。JP1/AJS からジョブを実行しない場合や，JP1/AJS のジョブ名から切り出した文字列が 101 バイト以上の場合，ジョブの実行日時をスプールジョブ名とします。なお，この動作は環境設定パラメーターVAR_SHELL_GETLENGTH パラメーターにBYTE を指定していることを前提としています。CHARACTER を指定している場合は，「101 バイト以上の場合」ではなく「101 文字以上の場合」となります。

- /で区切られた最後の要素を使う初期設定スクリプトの例

  環境変数AJSJOBNAME が「/user01/*AJSユニット名*/PCジョブ」の場合，スプールジョブディレクトリ名は「*ジョブ識別子*-PCジョブ」となります。

```
WK="$( "${ADSH_DIR_CMD}basename" "$AJSJOBNAME" )"
if [[ ${#WK} -le 100 ]] && [[ ${#WK} -ge 1 ]]
then
```

```
    ADSH_SPOOL_JOBNAME=”${WK}”
else
    ADSH_SPOOL_JOBNAME=”$( ”${ADSH_DIR_CMD}date” ”+%Y%m%d_%H%M%S” )”
fi
```

- /で区切られた最後の２つの要素を_で連結して使う初期設定スクリプトの例

  環境変数AJSJOBNAME が「/user01/*AJSユニット名*/PCジョブ」の場合，スプールジョブディレクトリ名は「*ジョブ識別子-AJSユニット名*_PCジョブ」となります。

```
SV_IFS=”$IFS”
IFS=”/”
WK=($AJSJOBNAME)
IFS=”$SV_IFS”
i=${#WK[*]}
if [[ i -gt 1 ]] && [[ ”(( ${#WK[i-1]} + ${#WK[i-2]} ))” -le 99 ]] && [[ ${#WK[i-1]} -ge
1 ]] && [[ ${#WK[i-2]} -ge 1 ]]
then
    ADSH_SPOOL_JOBNAME=”${WK[i-2]}_${WK[i-1]}”
else
    ADSH_SPOOL_JOBNAME=”$( ”${ADSH_DIR_CMD}date” ”+%Y%m%d_%H%M%S” )”
fi
```

3.13.2　注意事項

- シェル変数ADSH_SPOOL_JOBNAME で，スプールジョブディレクトリのスプールジョブ名を変更できますが，JP1/Advanced Shell のジョブ名は変更しません。
- CUI デバッガのスプールジョブ名は，次のどちらかの方法でだけ変更できます。
  - CUI デバッガの起動前に環境変数ADSH_SPOOL_JOBNAME で指定します。
  - 環境設定パラメーターexport で環境変数ADSH_SPOOL_JOBNAME に指定します。
- シェル変数ADSH_SPOOL_JOBNAME は#-adsh_step_start スクリプト拡張コマンドの-stepVar には指定しないでください。

4 JP1/Advanced Shell - Developer を使用する【Windows 限定】

JP1/Advanced Shell - Developer を使用して，Windows 環境でジョブ定義スクリプトを開発するための JP1/Advanced Shell エディタの操作方法について説明します。エディタを使用したジョブ定義スクリプトファイルのデバッグ方法についても説明します。

4.1　JP1/Advanced Shell - Developer の起動と終了【Windows 限定】

JP1/Advanced Shell の開発環境では，ジョブ定義スクリプトファイルの作成やデバッグができます。JP1/Advanced Shell の開発環境の起動と終了方法について説明します。

4.1.1　JP1/Advanced Shell - Developer の起動

JP1/Advanced Shell - Developer の起動方法を説明します。ジョブ定義スクリプトファイルの作成や編集をする場合は，エディタを起動します。エディタには，2 つの起動方法があります。

(1)　スタートメニューからの起動方法

1. ［スタート］から［すべてのプログラム］－［Advanced Shell - Developer］を選択する。
2. Advanced Shell - Developer のグループから「エディタ」アイコンを選択する。

(2)　右クリックメニューからの起動方法

1. エクスプローラからジョブ定義スクリプトファイルを右クリックする。
2. ［編集］を選択する。

4.1.2　JP1/Advanced Shell - Developer の終了

JP1/Advanced Shell - Developer を終了するには，JP1/Advanced Shell エディタウィンドウで次のどちらかの操作を実行します。

- ［ファイル］－［終了］を選択します。
- ツールバーの［終了］ボタンをクリックします。

これによってエディタ機能が終了します。

4.2　JP1/Advanced Shell エディタの状態【Windows 限定】

エディタには編集モードとデバッグモードがあります。

4.2.1　編集モード

ジョブ定義スクリプトファイルを作成・編集している状態です。エディタの起動時に設定されるモードです。

4.2.2　デバッグモード

作成したジョブ定義スクリプトファイルをデバッグしている状態です。エディタの編集画面はグレーアウトされ，ジョブ定義スクリプトの編集はできません。デバッグモードには，次の 2 つの機能があります。

- 文法チェック

  ［デバッグ］－［文法チェック］メニューを選択する，またはツールバーの［文法チェック］ボタンをクリックすると文法チェックが行われます。

- デバッグ実行

  次のメニューを選択またはボタンをクリックすると，デバッグが実行されます。

  - ［デバッグ］－［ブレークポイントまで実行］メニューを選択する，またはツールバーの［ブレークポイントまで実行］ボタンをクリックする
  - ［デバッグ］－［ステップイン］，［ステップオーバー］もしくは［ステップアウト］メニューを選択する，またはツールバーの［ステップイン］，［ステップオーバー］もしくは［ステップアウト］ボタンをクリックする

文法チェックについては「4.4.4　文法をチェックする」を，デバッグについては「4.4.6　デバッグをする」を参照してください。

4.3　JP1/Advanced Shell エディタの操作【Windows 限定】

エディタはジョブ定義スクリプトを新規に作成したり，既存のジョブ定義スクリプトを編集したりするためのプログラムです。ここでは，エディタを起動すると表示される JP1/Advanced Shell エディタウィンドウについて説明します。また，エディタの機能をメニューごとに説明します。

JP1/Advanced Shell エディタウィンドウでできる操作の一覧を次に示します。( ) 内は参照先です。

- ジョブ定義スクリプトを新規に作成する（4.4.1　ジョブ定義スクリプトを新規に作成する）
- エディタの動作環境を設定する（4.4.2　エディタの動作環境を設定する）
- ジョブ定義スクリプトの実行環境を設定する（4.4.3　ジョブ定義スクリプトの実行環境を設定する）
- 文法をチェックする（4.4.4　文法をチェックする）
- 文字列を検索および置換する（4.4.5　文字列を検索および置換する）
- デバッグ実行時のブレークポイントを設定・解除する（4.4.6　デバッグをするの(1)　デバッグ実行時のブレークポイントを設定・解除する）
- デバッグを実行・中止する（4.4.6　デバッグをするの(2)　デバッグを実行・中止する）
- デバッグ中に変数値を参照・更新する（4.4.6　デバッグをするの(3)　デバッグ中に変数値を参照・更新する）
- エラーをシミュレートする（4.4.6　デバッグをするの(4)　エラーをシミュレートする）
- trap コマンドのアクションを実行する（4.4.6　デバッグをするの(5)　trap コマンドのアクションを実行する）
- メッセージ出力モードを変更する※
- デバッグ実行時のカバレージ情報を表示する（4.4.7　カバレージ情報を表示する）
- 既存のジョブ定義スクリプトを編集する（4.5　既存のジョブ定義スクリプトを編集する【Windows 限定】）
- ジョブ定義スクリプトを保存する（4.6　ジョブ定義スクリプトを保存する【Windows 限定】）
- ジョブ定義スクリプトファイルの内容を印刷する※
- 直前の操作を元に戻す※
- 直前の操作をやり直す※
- 選択した文字列をクリップボードに切り抜く※
- 選択した文字列をクリップボードにコピーする※
- クリップボードの文字列を指定の位置に貼り付ける※
- すべての文字列を選択する※
- 実行ポイント行へジャンプする※
- ツールバーの表示・非表示を切り替える※

- アプリケーションの外観を切り替える※
- ステータスバーの表示・非表示を切り替える※
- ルーラーの表示・非表示を切り替える※
- 縦スクロールバーの表示・非表示を切り替える※
- 横スクロールバーの表示・非表示を切り替える※
- 行番号の表示・非表示を切り替える※
- ファイルの先頭を表示する※
- ファイルの末尾を表示する※
- ヘルプを表示する※
- 入力支援機能を使用する

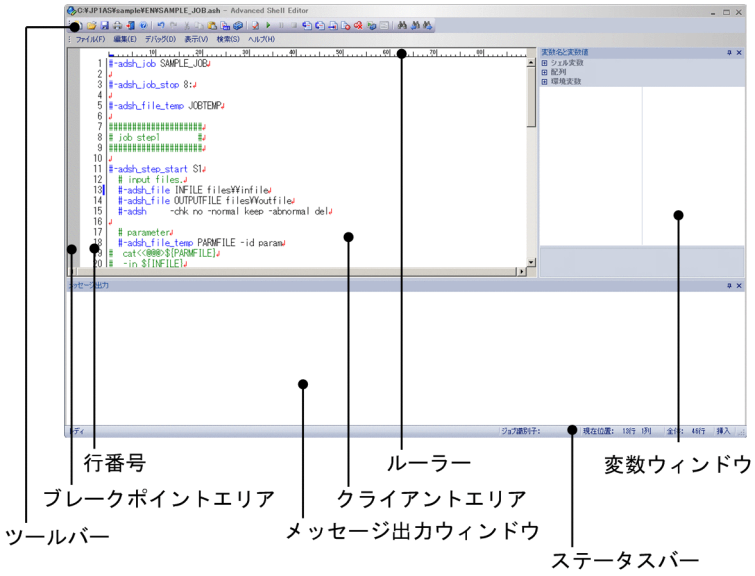
注※

これらの操作はこのマニュアルでは特に説明していません（これらは，Windows の標準の操作に準じているか，メニュー一覧で該当するメニューを選択するだけで実行できる操作です）。

4.3.1　JP1/Advanced Shell エディタウィンドウ

JP1/Advanced Shell エディタウィンドウ，およびウィンドウの各部の名称を，次の図に示します。

図 4-1　JP1/Advanced Shell エディタウィンドウ



(1)　ツールバー

ツールバーには，メニューバーから選択できるコマンドの中から，頻繁に使用するコマンドだけをボタンの形で表示しています。ボタンをマウスでクリックするだけで，該当するコマンドを実行できます。表示メニューによって非表示にすることもできます。なお，ボタンにマウスを移動させると説明が表示されます。

ツールバーのボタンと機能を次の表に示します。

ボタン	機能
［新規作成］ボタン	新規にジョブ定義スクリプトファイルを作成します。
［開く］ボタン	既存のジョブ定義スクリプトファイルを開きます。
［保存］ボタン	作業中のジョブ定義スクリプトファイルを保存します。
［印刷］ボタン	作業中のジョブ定義スクリプトファイルを印刷します。
［終了］ボタン	JP1/Advanced Shell エディタを終了し，ファイルを保存するかどうかを選択します。
［ヘルプ］ボタン	JP1/Advanced Shell のオンラインヘルプを表示します。
［元に戻す］ボタン	直前に行った動作を元に戻します。
［やり直し］ボタン	直前に行った動作をやり直します。
［切り抜き］ボタン	選択範囲を切り取ってクリップボードに保存します。
［コピー］ボタン	選択範囲をコピーしてクリップボードに保存します。
［貼り付け］ボタン	クリップボードの内容を指定の位置に貼り付けます。
［すべて選択］ボタン	ファイル全体を選択します。
［オプション］ボタン	エディタの動作環境を設定します。
［文法チェック］ボタン	記述したジョブ定義スクリプトの文法をチェックします。
［ブレークポイントまで実行］ボタン	ブレークポイントまでの実行スタート，およびリスタートをします。
［スクリプトの停止］ボタン	ジョブ定義スクリプトを停止します。［スクリプトの停止］ボタンをクリックしたときに実行していたコマンドはそのまま実行を続け，次のコマンドの実行に移る前に停止します。
［デバッグの中止］ボタン	［デバッグの中止］ボタンをクリックしたときに実行していたコマンドはそのまま実行を続け，次のコマンドの実行に移る前に停止します。そのあとジョブ定義スクリプトを停止し，デバッグを中止します。
［ステップイン］ボタン	次のコマンドまたはステートメントを 1 つずつ実行します。関数を呼び出す場合は，その関数の中も 1 行ずつ実行して停止します。
［ステップオーバー］ボタン	次のコマンドまたはステートメントを 1 つずつ実行します。関数を呼び出す場合，関数の中は 1 行ずつ停止しませんが，ブレークポイントがあるときは停止します。
［ステップアウト］ボタン	関数の呼び出し元まで実行します。関数を呼び出した次の行またはブレークポイントで停止します。
［ブレークポイントの設定／解除］ボタン	ブレークポイントを設定，または解除します。
［ブレークポイントをすべて解除］ボタン	設定されているブレークポイントをすべて解除します。

ボタン	機能
［実行環境の設定］ボタン	スクリプトファイルの実行環境を設定します。
［カバレージ情報の表示］ボタン	デバッグ実行時のカバレージ情報を表示します。
［検索］ボタン	検索または置換する文字列を入力します。
［前検索］ボタン	文字列を上に向かって検索または置換します。
［次検索］ボタン	文字列を下に向かって検索または置換します。

(2)　ルーラー

行のカラム数を表示する目盛です。

(3)　行番号

ジョブ定義スクリプトの行番号を表示するための領域です。

(4)　ブレークポイントエリア

ブレークポイントを示す記号（●）と，次に実行する位置を示す記号（➢）およびデバッガのプロセスの終了を示す記号（➢）を表示するための領域です。

(5)　ステータスバー

ステータスバーは，JP1/Advanced Shell エディタが現在実行している処理に関するメッセージや，処理終了後の状態に関するメッセージを表示するための領域です。JP1/Advanced Shell エディタウィンドウのステータスバーの機能を次の表に示します。

表 4-1　JP1/Advanced Shell エディタウィンドウのステータスバーの機能

ステータスバー	機能
ジョブ識別子	デバッグ実行したジョブのジョブ識別子を表示します。
現在位置	カーソルの位置を表示します。
全体	編集中のジョブ定義スクリプトファイルの行数を表示します。
上書き状態	Insert キーで切り替える上書き状態を表示します。次の 2 つのモードがあります。デフォルトは挿入モードです。 • 上書：上書きモード • 挿入：挿入モード

(6)　クライアントエリア

クライアントエリアには，ジョブ定義スクリプトファイルの内容が表示されます。

(7) メッセージ出力ウィンドウ

メッセージ出力ウィンドウには，デバッグ実行中に発生したエラーメッセージが表示されます。

(8) 変数ウィンドウ

変数ウィンドウには，デバッグ実行中に変数名と変数の値が表示されます。

4.3.2 JP1/Advanced Shell エディタウィンドウのメニュー

JP1/Advanced Shell エディタウィンドウのメニューバーに表示されるメニュー，および JP1/Advanced Shell エディタウィンドウで表示されるポップアップメニューについて説明します。

(1) メニューバーのメニュー

エディタウィンドウのメニューについて説明します。エディタウィンドウのメニューを次の表に示します。

表 4-2 JP1/Advanced Shell エディタウィンドウのメニュー

メニュー		機能
[ファイル]	[新規作成]	新規にジョブ定義スクリプトファイルを作成します。
	[開く]	既存のジョブ定義スクリプトファイルを開きます。
	[保存]	作業中のジョブ定義スクリプトファイルを保存します。
	[名前を付けて保存]	作業中のジョブ定義スクリプトファイルに名前を付けて保存します。
	[印刷]	作業中のジョブ定義スクリプトファイルを印刷します。
	[終了]	エディタを終了し，ファイルを保存するかどうかを選択します。
	(ファイル名)	指定のファイルを開きます。 直前に保存したジョブ定義スクリプトファイルが最大 9 個表示されます。
[編集]	[元に戻す]	直前に行った動作を元に戻します。
	[やり直し]	直前に行った動作をやり直します。
	[切り抜き]	選択範囲を切り取ってクリップボードに保存します。
	[コピー]	選択範囲をコピーしてクリップボードに保存します。
	[貼り付け]	クリップボードの内容を指定の位置に貼り付けます。
	[すべて選択]	ファイル全体を選択します。
	[オプション]	エディタの動作環境を設定します。
[デバッグ]	[文法チェック]	ジョブ定義スクリプトの文法をチェックします。
	[ブレークポイントまで実行]	デバッグモードで，ブレークポイントまでの実行スタート，およびリスタートします。

メニュー		機能
［デバッグ］	［スクリプトの停止］	ジョブ定義スクリプトの実行を次の行で停止します。［スクリプトの停止］を選択したときに実行していたコマンドはそのまま実行を続け，次のコマンドの実行に移る前に停止します。
	［デバッグの中止］	［デバッグの中止］を選択したときに実行していたコマンドはそのまま実行を続け，次のコマンドの実行に移る前に停止します。そのあとジョブ定義スクリプトを停止し，デバッグを中止します。
	［ステップイン］	デバッグモードで，次のコマンドまたはステートメントを 1 つずつ実行します。関数を呼び出すときは，その関数の中も 1 行ずつ実行して停止します。
	［ステップオーバー］	デバッグモードで，次のコマンドまたはステートメントを 1 つずつ実行します。関数を呼び出すときは，関数の中は 1 行ずつ停止しませんが，ブレークポイントがあるときは停止します。
	［ステップアウト］	関数の呼び出し元まで実行します。関数を呼び出した次の行またはブレークポイントで停止します。
	［ブレークポイントの設定／解除］	ブレークポイントを設定，または解除します。
	［ブレークポイントをすべて解除］	設定されているブレークポイントをすべて解除します。
	［実行環境の設定］	スクリプトファイルの実行環境を設定します。
	［ウォッチへ変数の追加］	指定した変数をウォッチウィンドウへ追加します。
	［エラー注入モード］	ジョブ定義スクリプトの実行停止中にエラー注入モードを有効，または無効にします。
	［トラップアクションの実行］	trap コマンドのアクションを実行し，ブレークポイントまで実行します。
	［メッセージ出力モード］	実行中のスクリプトのメッセージ出力モードを切り替えます。
	［実行ポイント行へジャンプ］	現在の実行ポイント行へジャンプします。
［表示］	［ツールバーとドッキングウィンドウ］－［ツールバー］	ツールバーの表示・非表示を切り替えます。
	［ツールバーとドッキングウィンドウ］－［メッセージ出力］	メッセージ出力ウィンドウの表示・非表示を切り替えます。
	［ツールバーとドッキングウィンドウ］－［変数名と変数値］	変数ウィンドウの表示・非表示を切り替えます。
	［ステータスバー］	ステータスバーの表示・非表示を切り替えます。
	［アプリケーションの外観］	ウィンドウの外観を変更します。

メニュー		機能
[表示]	[ルーラー]	ルーラーの表示・非表示を切り替えます。
	[縦スクロールバー]	縦スクロールバーの表示・非表示を切り替えます。
	[横スクロールバー]	横スクロールバーの表示・非表示を切り替えます。
	[行番号を表示]	行番号の表示・非表示を切り替えます。
	[ファイルの先頭を表示]	ジョブ定義スクリプトファイルの先頭を表示します。
	[ファイルの末尾を表示]	ジョブ定義スクリプトファイルの末尾を表示します。
	[カバレージ情報の表示]	デバッグ実行時のカバレージ情報を表示します。
[検索]	[検索]	検索または置換する文字列を入力します。
	[置換]	文字列を指定した文字列に置換します。
	[前検索]	文字列を上に向かって検索または置換します。
	[次検索]	文字列を下に向かって検索または置換します。
[ヘルプ]	[ヘルプを開く]	JP1/Advanced Shell オンラインヘルプを表示します。
	[使用例題を表示]	入力支援機能として，使用例題を表示します。
	[バージョン情報]	プログラムの情報，バージョンおよび著作権を表示します。

(2) ポップアップメニュー

JP1/Advanced Shell エディタウィンドウのクライアントエリアでマウスの右ボタンをクリックすると，ポップアップメニューが表示されます。ポップアップメニューの内容は編集モードの場合とデバッグモードの場合とで異なります。

- 編集モードのポップアップメニュー

  編集モードの場合に表示されるポップアップメニューを次の表に示します。

ポップアップメニュー	機能
[新規作成]	新規にジョブ定義スクリプトファイルを作成します。
[開く]	既存のジョブ定義スクリプトファイルを開きます。
[保存]	作業中のジョブ定義スクリプトファイルを保存します。
[元に戻す]	直前に行った動作を元に戻します。
[やり直し]	直前に行った動作をやり直します。
[切り抜き]	選択範囲を切り取ってクリップボードに保存します。
[コピー]	選択範囲をコピーしてクリップボードに保存します。
[貼り付け]	クリップボードの内容を指定の位置に貼り付けます。
[すべて選択]	ファイル全体を選択します。

- デバッグモードのポップアップメニュー
  デバッグモードの場合に表示されるポップアップメニューを次の表に示します。

ポップアップメニュー	機能
［コピー］	選択範囲をコピーしてクリップボードに保存します。
［ブレークポイントの設定／解除］	ブレークポイントを設定，または解除します。

4.3.3　JP1/Advanced Shell エディタウィンドウでのマウスとキーの操作

JP1/Advanced Shell エディタウィンドウでのマウスとキーの操作について説明します。

(1)　マウス操作

JP1/Advanced Shell エディタウィンドウのクライアントエリアでのマウス操作を次の表に示します。

表 4-3　JP1/Advanced Shell エディタウィンドウでのマウス操作

操作	機能
クリック	それまでの選択を解除して，新たに対象を選択します。
ダブルクリック	文字列を選択します。
右クリック	ポップアップメニューを表示します。

(2)　キー操作

JP1/Advanced Shell エディタウィンドウのクライアントエリアでのキー操作と，モードごとの操作の可否を次の表に示します。

表 4-4　JP1/Advanced Shell エディタウィンドウでのキー操作

操作	編集モード	デバッグモード	機能
Ctrl+A	○	○	ファイル全体を選択します。
Ctrl+C	○	○	選択範囲をコピーします。
Ctrl+E	○	×	スクリプトファイルの実行環境を設定します。
Ctrl+F	○	△	検索または置換する文字列を入力します。
Ctrl+H	○	×	文字列を指定した文字列に置換します。
Ctrl+K Ctrl+F1	○	○	入力支援機能として，使用例題を表示します。
Ctrl+N	○	×	新規にジョブ定義スクリプトファイルを作成します。
Ctrl+O	○	×	既存のジョブ定義スクリプトファイルを開きます。

操作	編集モード	デバッグモード	機能
Ctrl+P	○	×	作業中のジョブ定義スクリプトファイルを印刷します。
Ctrl+S	○	×	作業中のジョブ定義スクリプトファイルを保存します。
Ctrl+V	○	×	クリップボードの内容を指定の位置に貼り付けます。
Ctrl+X	○	×	選択範囲を切り取ります。
Ctrl+Z	○	×	直前に行った動作を元に戻します。
Ctrl+Home	○	○	ジョブ定義スクリプトファイルの先頭を表示します。
Ctrl+End	○	○	ジョブ定義スクリプトファイルの末尾を表示します。
F1	○	○	JP1/Advanced Shell ヘルプを表示します。
F3	○	△	文字列を下に向かって検索または置換します。
F5	○	○	ブレークポイントまでの実行スタート，およびリスタートします。
F7	○	×	ジョブ定義スクリプトの文法をチェックします。
F9	○	○	ブレークポイントを設定，または解除します。
F11	○	○	次のコマンドまたはステートメントを 1 つずつ実行します。関数を呼び出すときは，その関数の中も 1 行ずつ実行して停止します。
Alt+F4	○	○	JP1/Advanced Shell エディタを終了し，ファイルを保存するかどうかを選択します。
Shift+F3	○	△	文字列を上に向かって検索または置換します。
Shift+F5	×	○	ジョブ定義スクリプトを停止し，デバッグを中止します。
Shift+F9	○	○	指定ブレークポイントをすべて解除します。
Shift+F11	○	○	関数の呼び出し元まで実行します。関数を呼び出した次の行またはブレークポイントで停止します。
Shift+Ctrl+F11	○	○	次のコマンドまたはステートメントを 1 つずつ実行します。関数を呼び出すときは，関数の中は 1 行ずつ停止しませんが，ブレークポイントがあるときは停止します。
Shift+Ctrl+Z	○	×	直前に行った動作をやり直します。
Enter	○	×	行頭からのスペースとタブをコピーして新しい行を作成します。「{」の次に改行を入力したときは，次の行にはタブを追加し，さらにその次の行に「}」を追加します。

（凡例）

○：操作できる機能

△：一部操作できる機能

×：操作できない機能

4.4　新規にジョブ定義スクリプトを作成する【Windows 限定】

JP1/Advanced Shell エディタで新規にジョブ定義スクリプトを作成する方法について説明します。

4.4.1　ジョブ定義スクリプトを新規に作成する

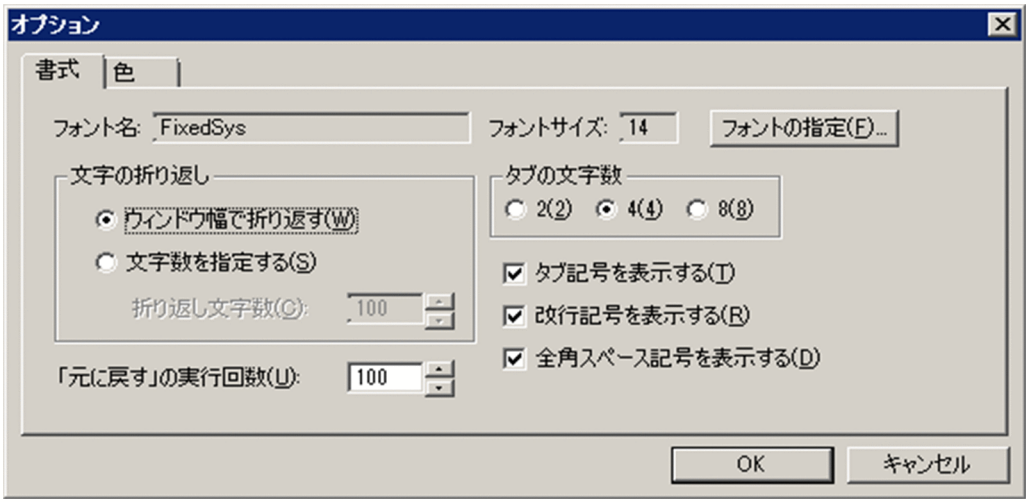
ジョブ定義スクリプトファイルを新規に作成します。

1. ［ファイル］－［新規作成］メニューを選択する。
   新規に JP1/Advanced Shell エディタウィンドウが表示されます。

4.4.2　エディタの動作環境を設定する

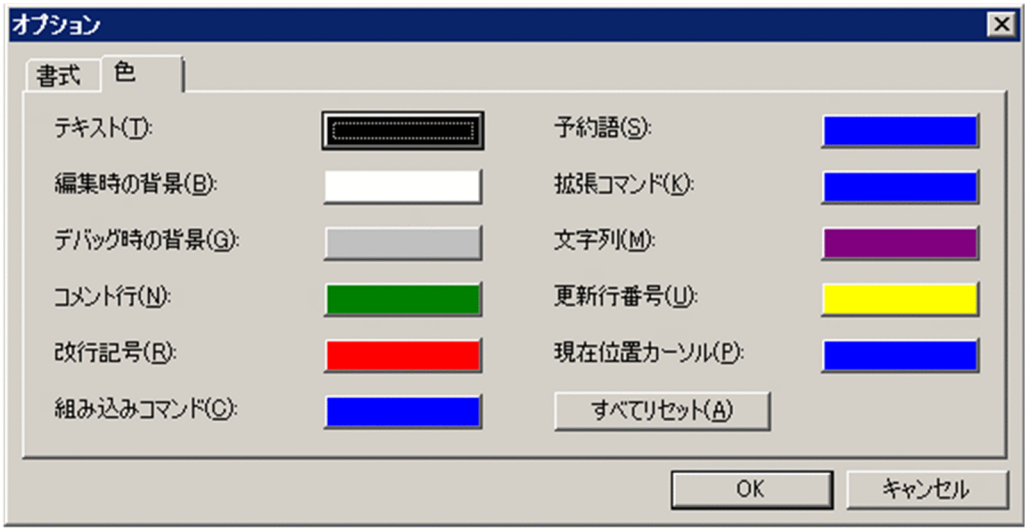
エディタの動作環境を設定します。

1. ［編集］－［オプション］メニューを選択する。
   ［オプション（書式）］ダイアログボックスが表示されます。



   ダイアログボックスの設定方法については「4.7.1　オプション（書式）ダイアログボックス」を参照してください。

2. 書式に関する情報を設定する。
3. ［色］タブをクリックする。
   ［オプション（色）］ダイアログボックスが表示されます。



4. 表示色に関する情報を設定する。

   すべての項目の表示色をデフォルトに戻すには［すべてリセット］ボタンをクリックしてください。

   ダイアログボックスの設定方法については，「4.7.2　オプション（色）ダイアログボックス」を参照してください。

5. ［OK］ボタンをクリックする。

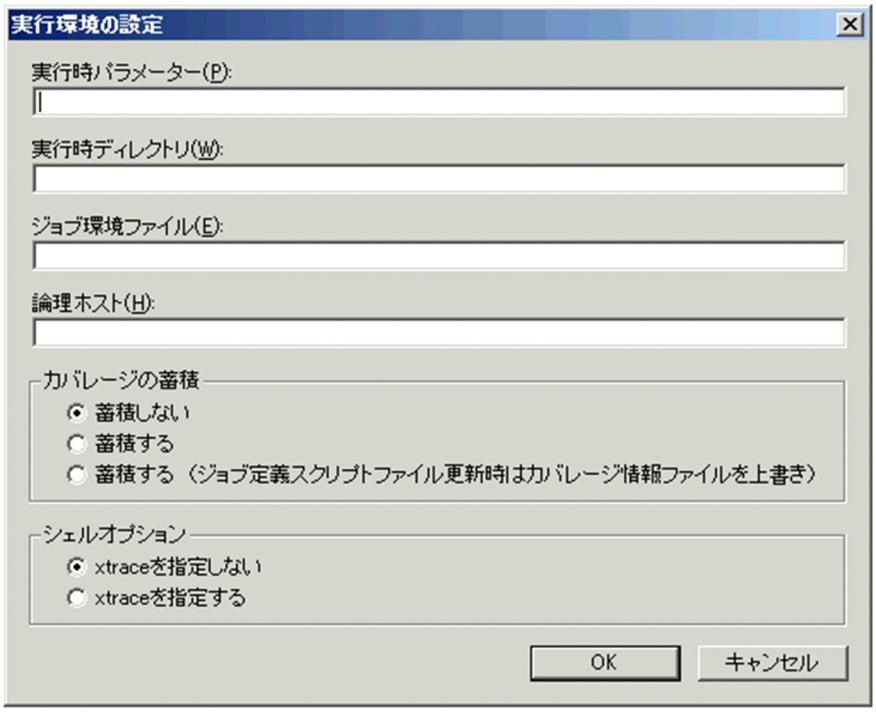
   エディタの動作環境が設定され，ダイアログボックスが閉じます。

4.4.3　ジョブ定義スクリプトの実行環境を設定する

ジョブ定義スクリプトファイルごとに，実行時パラメーター，実行時ディレクトリ，ジョブ環境ファイル，および論理ホストを設定できます。設定した情報は，デバッグ情報ファイルに保存されます。

1. ［デバッグ］－［実行環境の設定］メニューを選択する。

   ［実行環境の設定］ダイアログボックスが表示されます。



ダイアログボックスの設定方法については，「4.7.3　実行環境の設定ダイアログボックス」を参照してください。

2. ［OK］ボタンをクリックする。

   実行環境が設定され，ダイアログボックスが閉じます。

   カバレージの蓄積で「蓄積しない」を選択している場合，カバレージ情報は採取されません。

4.4.4　文法をチェックする

ジョブ定義スクリプトファイルの文法をチェックします。ジョブ定義スクリプトの文法が正しいかどうかのチェックだけ行い，実行しません。カバレージを蓄積するオプションを指定しても実行しないため，蓄積しません。adshexec コマンドの-c オプションの指定に相当します。

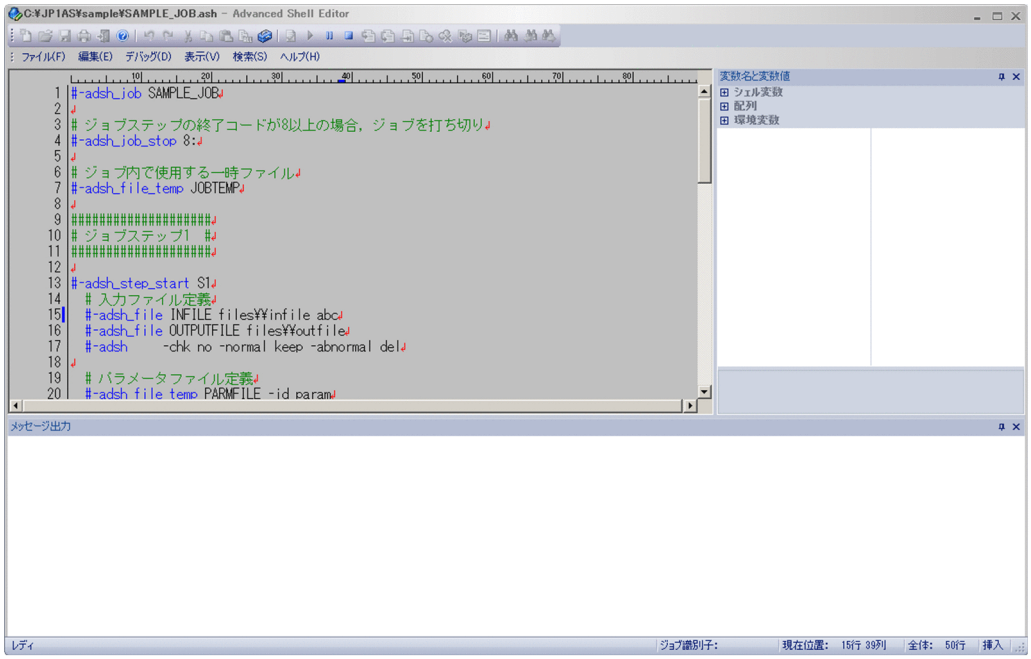
コンソールは表示されません。エラーはエラーウィンドウに表示されます。

1. ［デバッグ］－［文法チェック］メニューを選択する。

   エディタがデバッグモードになり，文法チェックが開始されます。

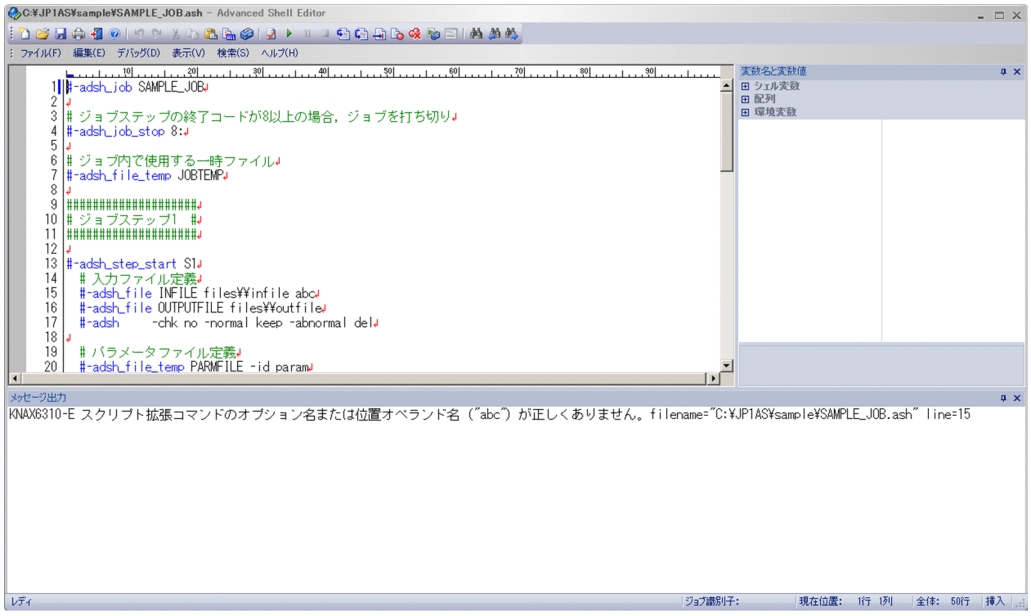
   文法チェック時は一瞬，画面が暗くなります。

   - **文法チェック中の表示**



- **文法チェック終了の表示**

  文法エラーがある場合はメッセージ出力ウィンドウにエラー内容が表示されます。



2. メッセージ出力ウィンドウの内容を確認する。

   メッセージ出力ウィンドウについては，「4.7.5　メッセージ出力ウィンドウ」を参照してください。

注意事項

- デバッグモードのときは，メニューがグレーアウトされて［デバッグ］－［文法チェック］は選択できません。
- ファイル名のないジョブ定義スクリプトファイルに対して文法チェックを実行しようとすると，ファイル名を付けて保存するためのダイアログボックスが表示されます。ジョブ定義スクリプトファイル名（.ash）を付けて保存しないと，文法チェックは実行できません。

- ジョブ定義スクリプトファイルの内容が更新されている場合，ファイルを更新するかどうかを問い合わせるメッセージが表示されます。更新する場合はファイルを保存して文法チェックを実行してください。

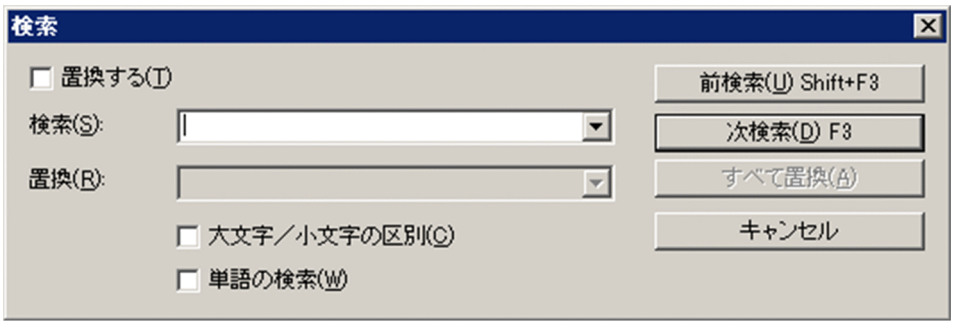
4.4.5　文字列を検索および置換する

ジョブ定義スクリプトファイル中の文字列の検索および置換について説明します。

(1)　文字列を検索する

ジョブ定義スクリプトファイル中の文字列を検索します。

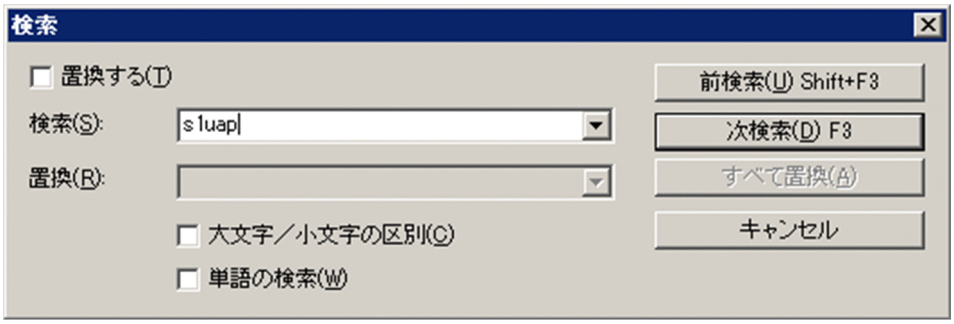
1. ［検索］－［検索］メニューを選択する。
   ［検索］ダイアログボックスが表示されます。



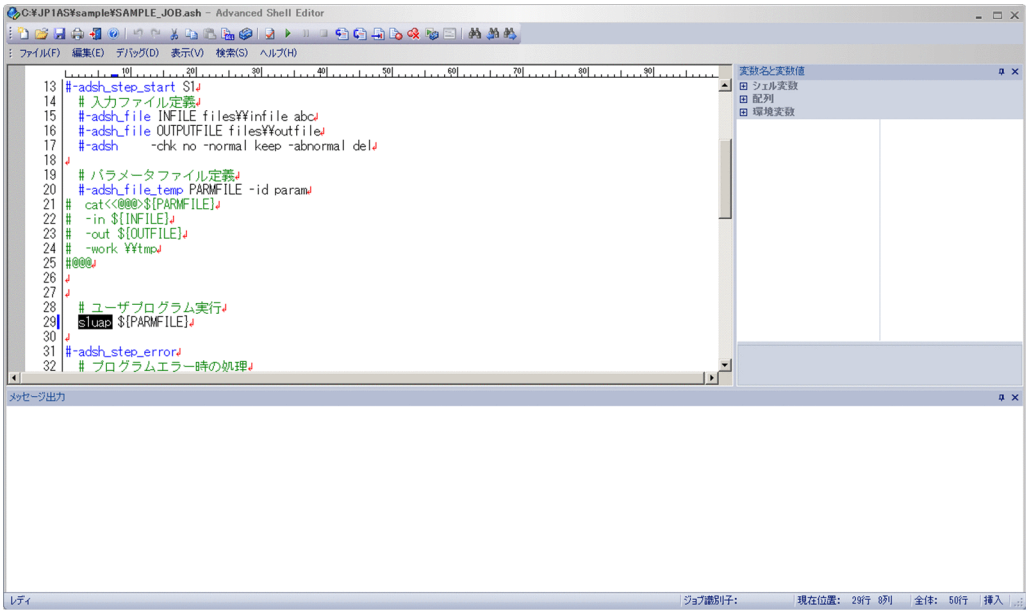
   ダイアログボックスの設定方法については，「4.7.4　検索ダイアログボックス」を参照してください。

2. ［置換する］がチェックされていないことを確認する。
   ［置換する］がチェックされている場合は，チェックを外してください。

3. ［検索］に検索文字列を指定する。また，必要に応じて［大文字／小文字の区別］，および［単語の検索］をチェックする。



4. ［前検索］ボタン，または［次検索］ボタンをクリックする。
   検索文字列が検索されます。検索文字列がない場合は，ビープ音が鳴ります。

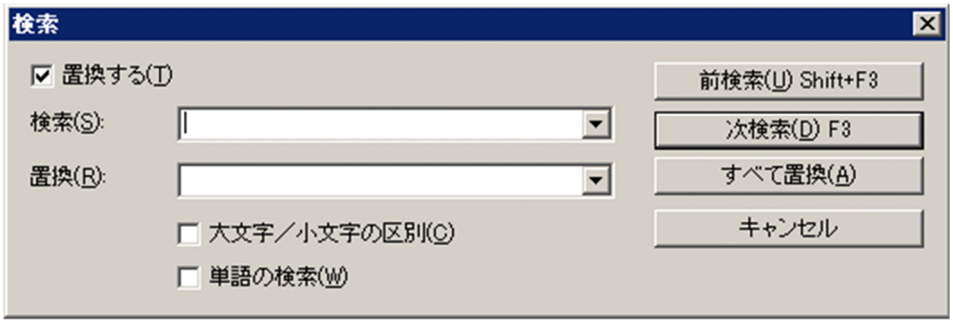


5. 検索を終了する場合は，［キャンセル］ボタンをクリックする。
   ［検索］ダイアログボックスが閉じます。

(2) 文字列を置換する

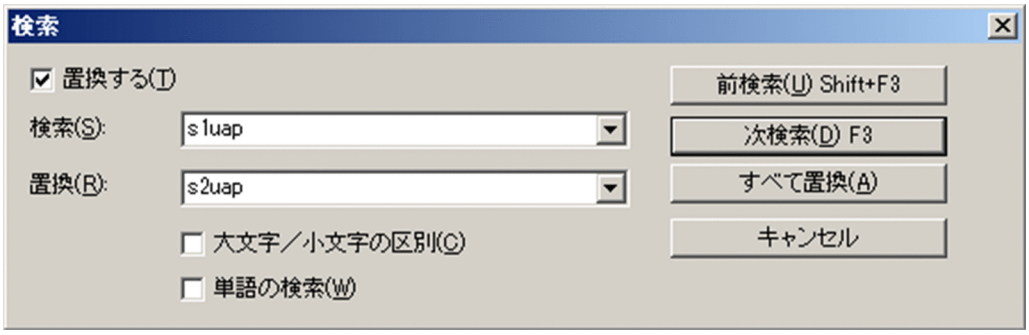
ジョブ定義スクリプトファイル中の文字列を置換します。

1. ［検索］－［置換］メニューを選択する。
   ［検索］ダイアログボックスが表示されます。



   ダイアログボックスの設定方法については，「4.7.4 検索ダイアログボックス」を参照してください。

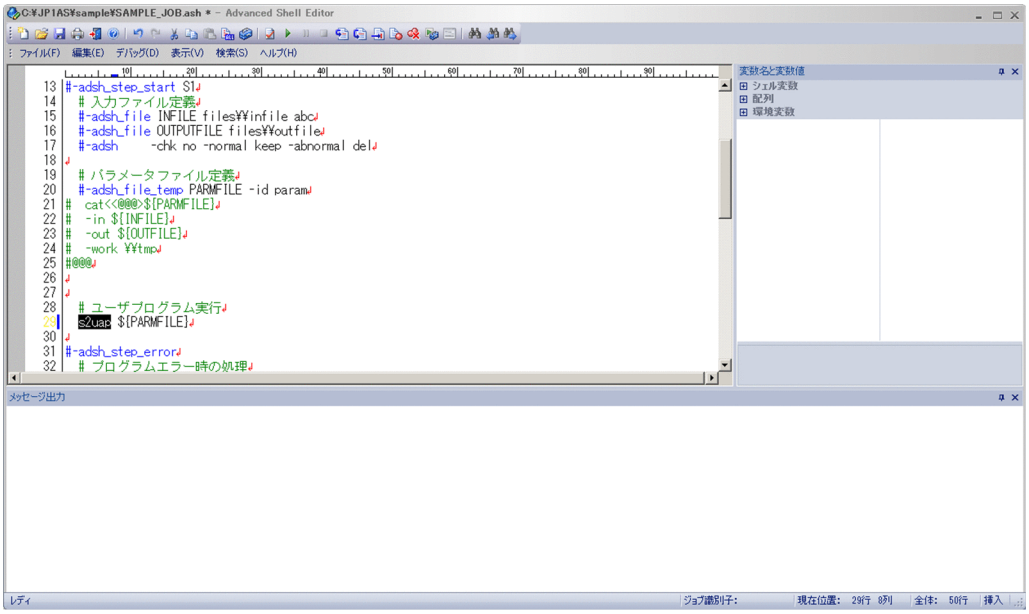
2. ［置換する］がチェックされていることを確認する。
   ［置換する］がチェックされていない場合は，チェックしてください。
3. ［検索］に置換前の文字列を指定する。また，必要に応じて［大文字／小文字の区別］，および［単語の検索］をチェックする。



4. ［置換］に置換後の文字列を指定する。

5. ［前検索］ボタン，または［次検索］ボタンをクリックする。

   指定した内容で置換が始まります。置換するための文字列がない場合は，ビープ音が鳴ります。



6. 置換を終了する場合は，［キャンセル］ボタンをクリックする。

   ［検索］ダイアログボックスが閉じます。

4.4.6 デバッグをする

ジョブ定義スクリプトファイルの動作を確認しながらデバッグ実行することをデバッグといいます。

adshexec コマンドの-d オプションの指定に相当します。デバッグをすると，コンソールが表示されます。エラーメッセージはエラーウィンドウに表示されます。

コンソールの内容を確認するためにプロセス終了の直前で停止します。初期化処理や構文解析でエラーがあった場合，プロセス終了前で停止しないでエラーウィンドウにエラーメッセージが表示されます。

デバッグの方法には，次の 2 種類があります。

方法	操作	概要
実行	［デバッグ］－［ブレークポイントまで実行］メニュー	ブレークポイントまでの実行スタート，およびリスタートをします。
ステップ実行	［デバッグ］－［ステップイン］メニュー	ジョブ定義スクリプトを 1 行ずつ実行して停止します。関数を呼び出す場合は，その関数の中も 1 行ずつ実行して停止します。
	［デバッグ］－［ステップオーバー］メニュー	ジョブ定義スクリプトを 1 行ずつ実行して停止します。関数を呼び出す場合，関数の中は 1 行ずつ停止しませんが，ブレークポイントがあるときは停止します。
	［デバッグ］－［ステップアウト］メニュー	関数を呼び出した次の行またはブレークポイントで停止します。

注意事項

- ファイル名のないジョブ定義スクリプトファイルに対してデバッグを実行しようとすると，ファイル名を付けて保存するためのダイアログボックスが表示されます。ジョブ定義スクリプトファイル名（.ash）を付けて保存しないと，デバッグは実行できません。
- ジョブ定義スクリプトファイルの内容が更新されている場合，ファイルを更新するかどうかを問い合わせるメッセージが表示されます。ファイルを更新するとデバッグが実行できます。
- 「プログラムの終了」のダイアログで「すぐに終了」を選択したときなど，デバッグ実行中にエディタが強制終了された場合，デバッガのプロセスである adshesub.exe だけ終了せずに，コンソールを表示し続ける場合があります。その場合，taskkill コマンドまたはタスクマネージャーから adshesub.exe プロセスを終了させてください。

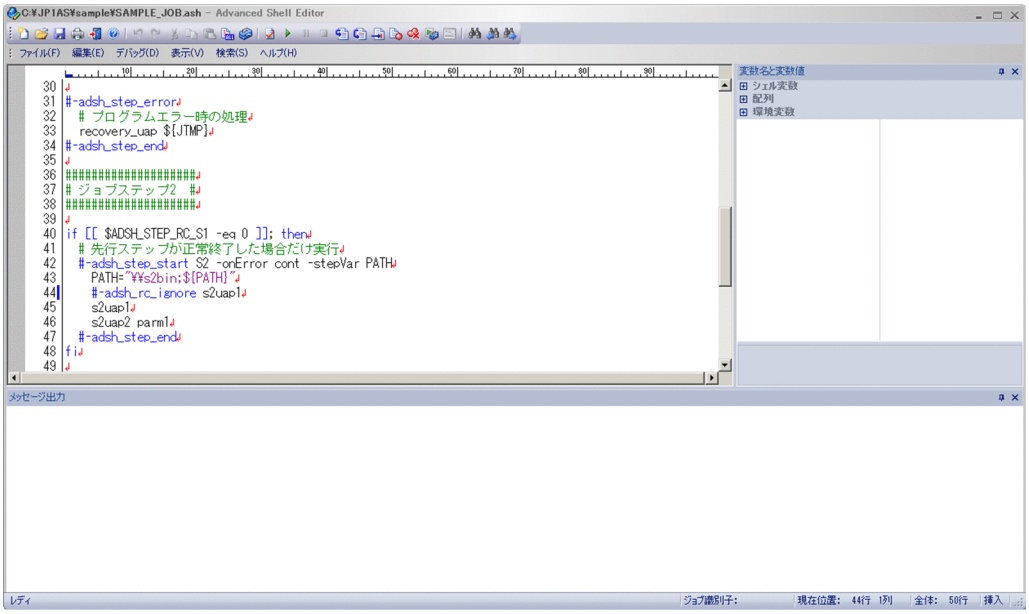
(1)　デバッグ実行時のブレークポイントを設定・解除する

デバッグの実行時に，一時的に実行を停止させる位置を設定，または解除します。

JP1/Advanced Shell エディタの場合，カーソルがある行にブレークポイントを設定するため，外部スクリプトには設定できません。事前に外部スクリプトにブレークポイントを設定していても，外部スクリプトのブレークポイントでは停止しません。設定できるブレークポイントの数の上限は 999 です。

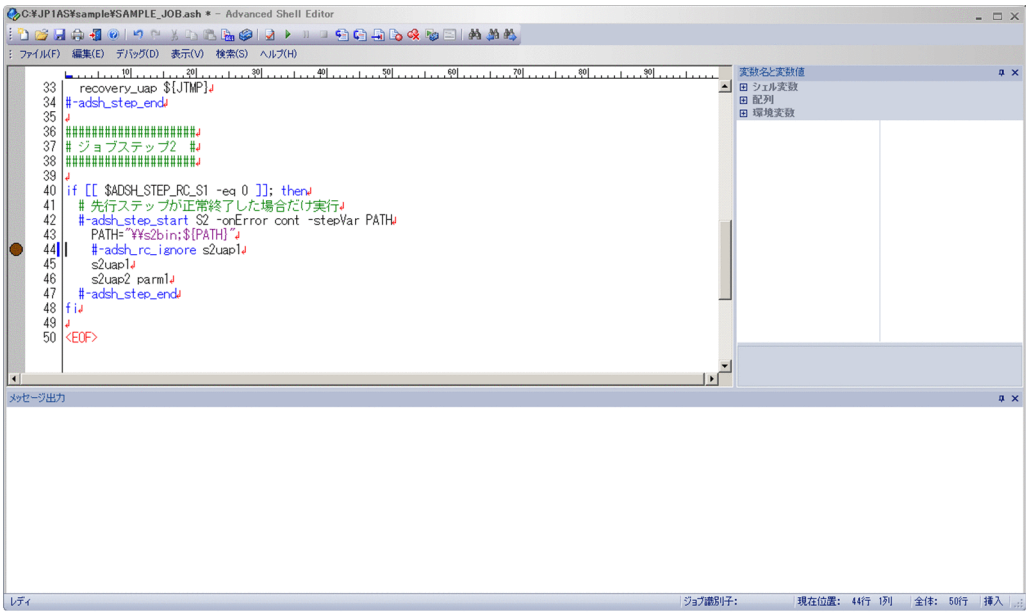
(a)　ブレークポイントを設定する

1. ブレークポイントを設定したい行にカーソルを移動させる。



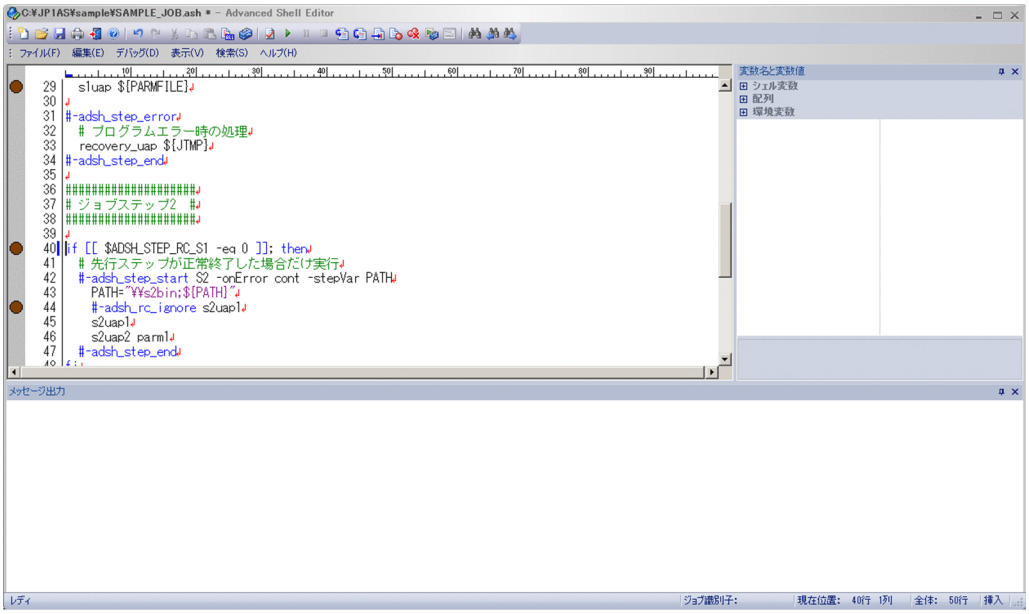
2. [デバッグ] － [ブレークポイントの設定] メニューを選択する。

   カーソルのある行にブレークポイントが設定されます。ブレークポイントは，行の左側に ● で表示されます。ジョブ定義スクリプトは，ブレークポイントを設定した行の先頭（実行前）まで実行して停止します。



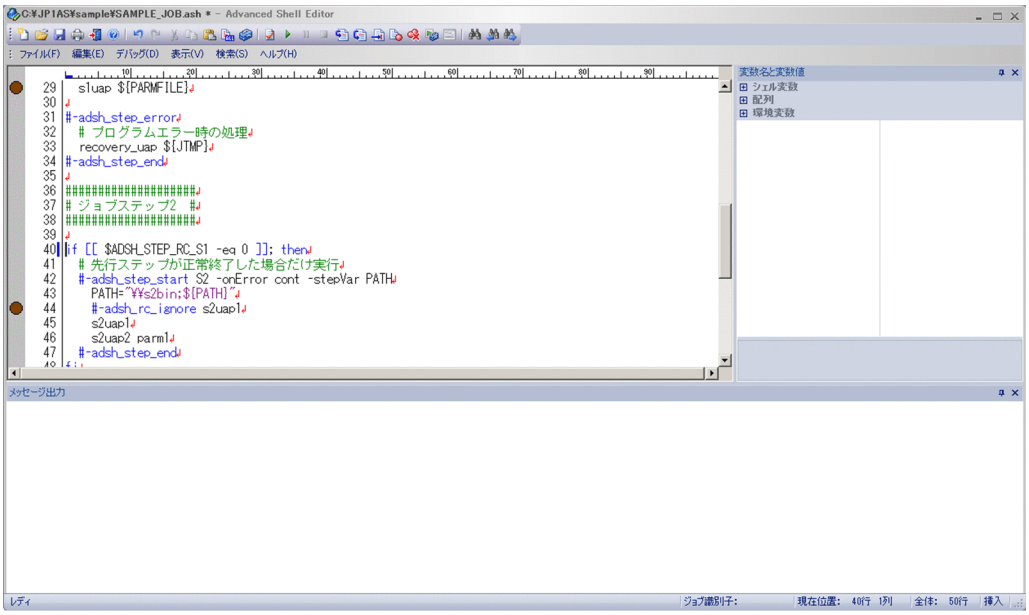
(b) 一部のブレークポイントを解除する

1. ブレークポイントを解除したい行にカーソルを移動させる。



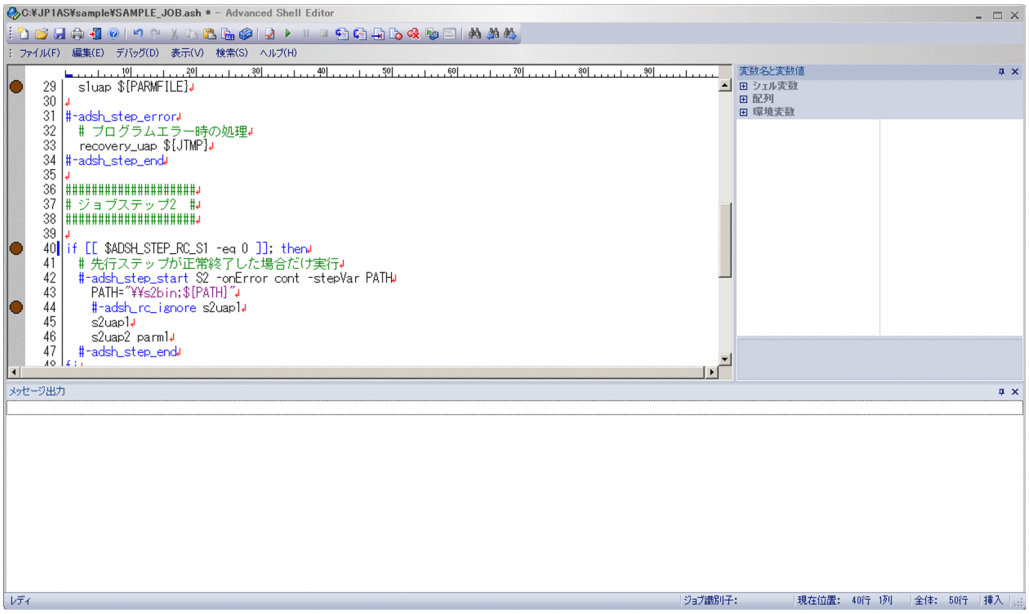
2. [デバッグ] － [ブレークポイントの解除] メニューを選択する。

   カーソルのある行のブレークポイントが解除されます。



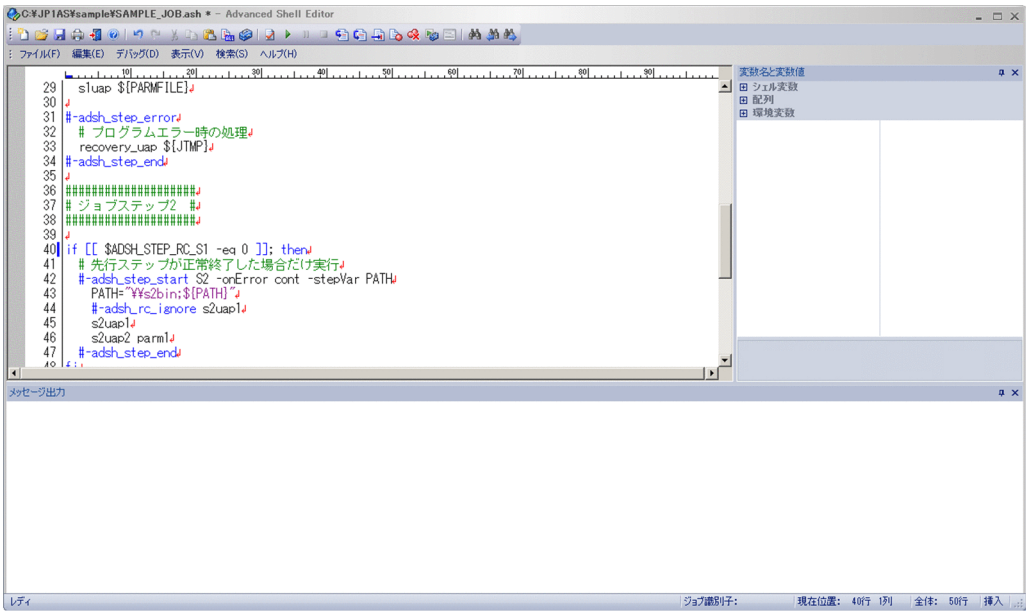
(c) すべてのブレークポイントを解除する

1. ブレークポイントが設定されているジョブ定義スクリプトを表示する。



2. ［デバッグ］－［ブレークポイントをすべて解除］メニューを選択する。

   表示されているジョブ定義スクリプトファイルのブレークポイントがすべて解除されます。



注意事項

- 編集モードの場合は，任意の行にブレークポイントを設定できます。デバッグモードの場合，ブレークポイントを設定できるのは，実行対象となっているコマンド，またはステートメント単位に限られます。
- 実行できない行にブレークポイントが設定されている場合は，デバッグモード開始時に，エディタが自動的に下方向に適切な位置を検索して，そこにブレークポイントを設定します。
- ブレークポイントは，999 個まで指定できます。

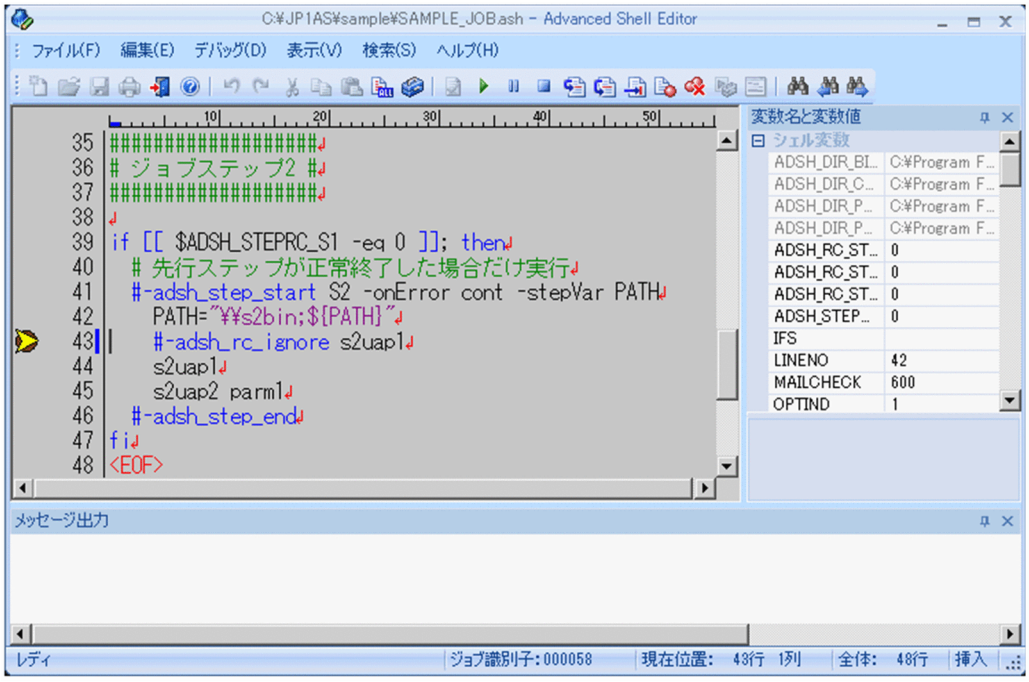
(2)　デバッグを実行・中止する

(a)　ブレークポイントまで実行する場合

1. ［デバッグ］－［ブレークポイントまで実行］メニューを選択する，またはツールバーの［ブレークポイントまで実行］ボタンをクリックする。

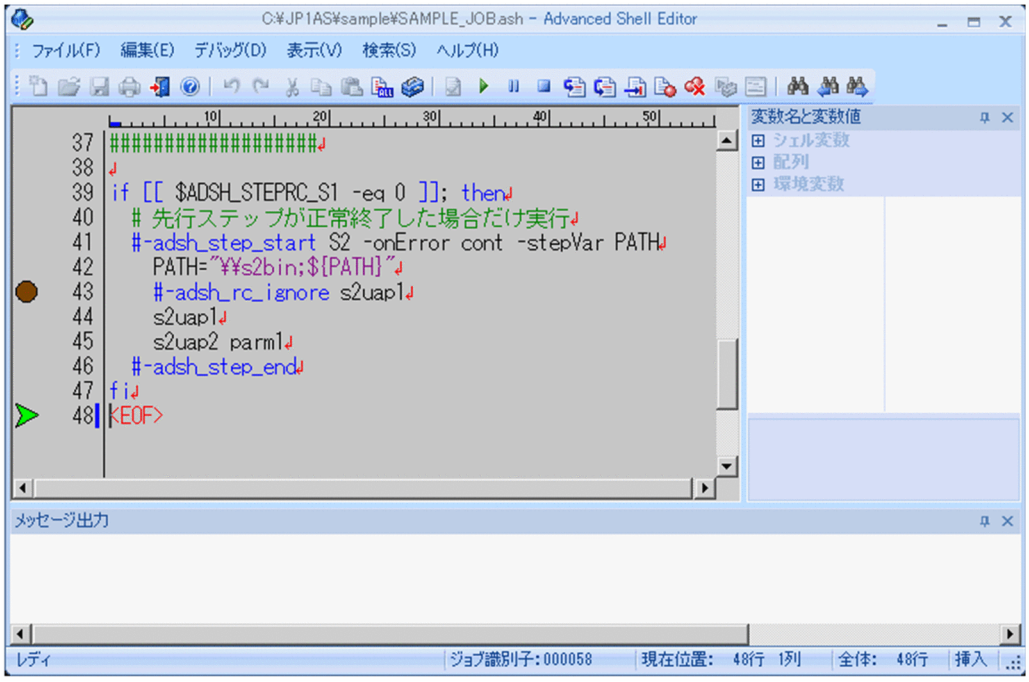
   JP1/Advanced Shell エディタがデバッグモードになり，デバッグが始まります。次に実行される行の左側には，実行する位置を示す記号（ ）が表示されます。コメント行やスペース行は無視されます。

   ブレークポイント（ ）を指定している行まで実行されると，実行が一時的に停止されます。



   ブレークポイントの設定方法については，「(1)　デバッグ実行時のブレークポイントを設定・解除する」を参照してください。

   ジョブ定義スクリプトの最後の行まで実行すると，デバッガのプロセスの終了を示す記号（ ）が表示されます。

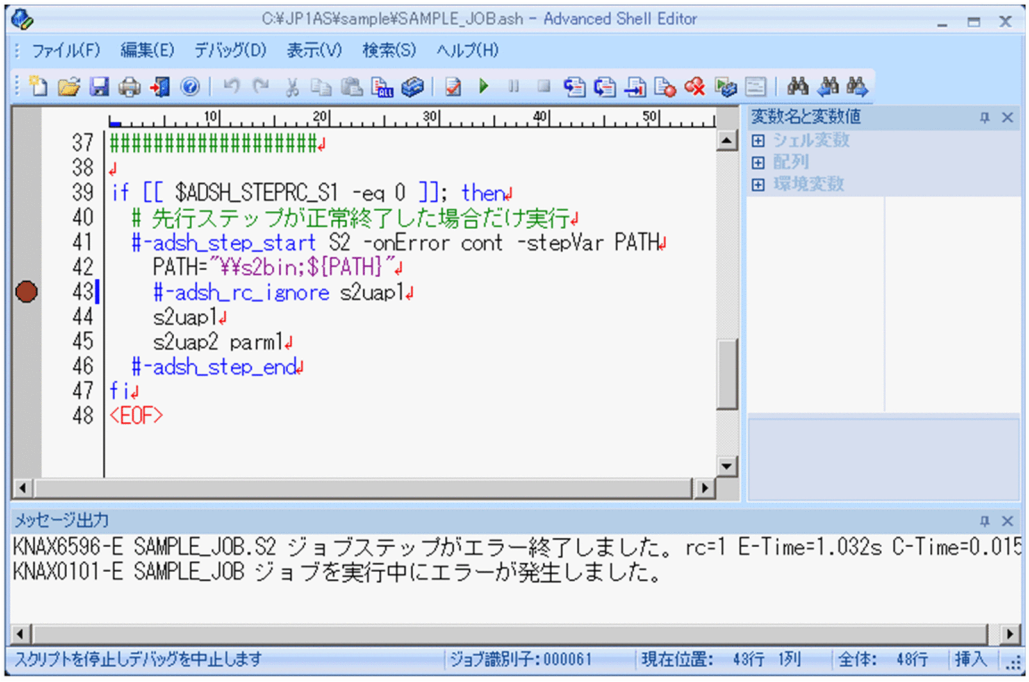


2. 実行中のジョブ定義スクリプトを停止させたい場合は，［デバッグ］－［スクリプトの停止］メニューを選択する，またはツールバーの［スクリプトの停止］ボタンをクリックする。

   ［スクリプトの停止］ボタンをクリックしたときに実行していたコマンドはそのまま実行を続け，次のコマンドの実行に移る前に停止します。

3. デバッグを中止したい場合は，［デバッグ］－［デバッグの中止］メニューを選択する，またはツールバーの［デバッグの中止］ボタンをクリックする。

   メニューを選択した時点で，デバッグが中止され，メッセージを出力して後処理をして終了します。プロセス終了前で停止しないでデバッグを終了します。エディタは編集モードに戻ります。

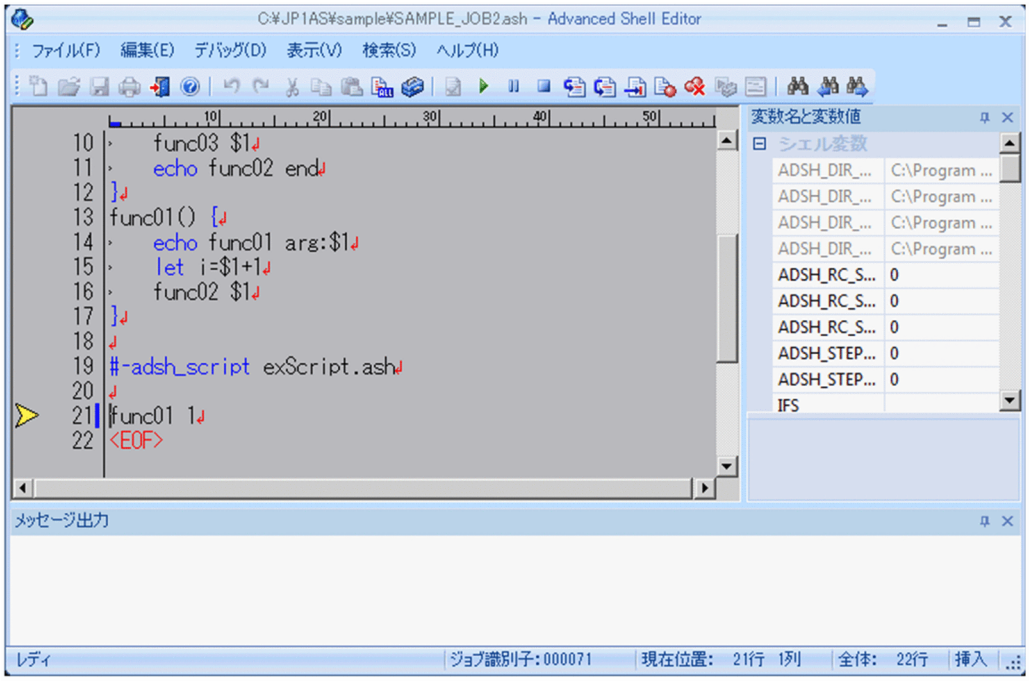


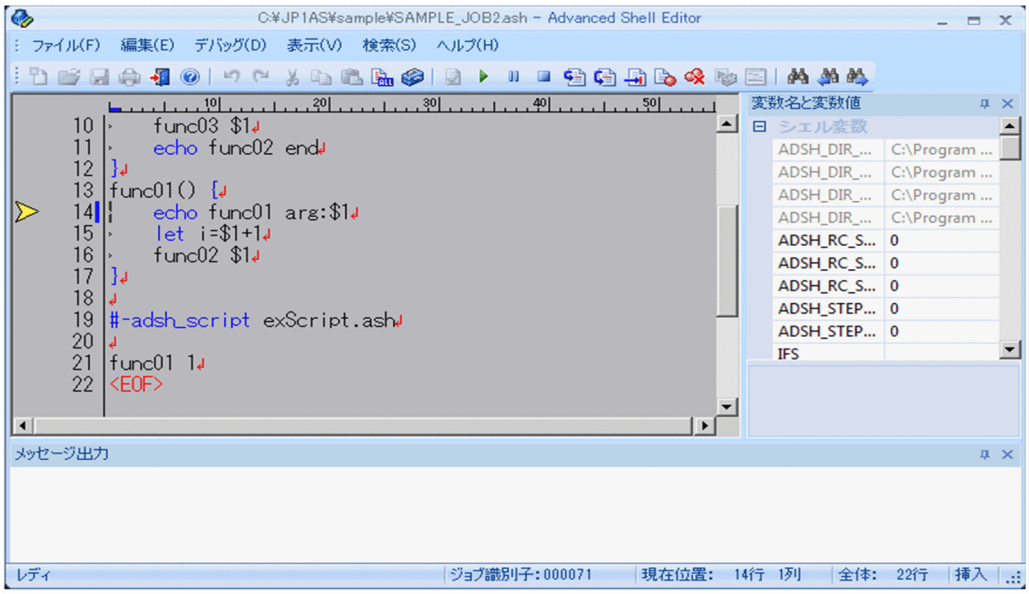
(b) 1 行ずつ実行（関数の中もステップ実行）する場合

1. ［デバッグ］－［ステップイン］メニューを選択する，またはツールバーの［ステップイン］ボタンをクリックする。

   エディタがデバッグモードになり，デバッグが始まります。

   CUI とは異なって，外部スクリプトを実行する場合，外部スクリプトでは停止しないで，エディタで表示しているジョブ定義スクリプトの次のコマンドで停止します。





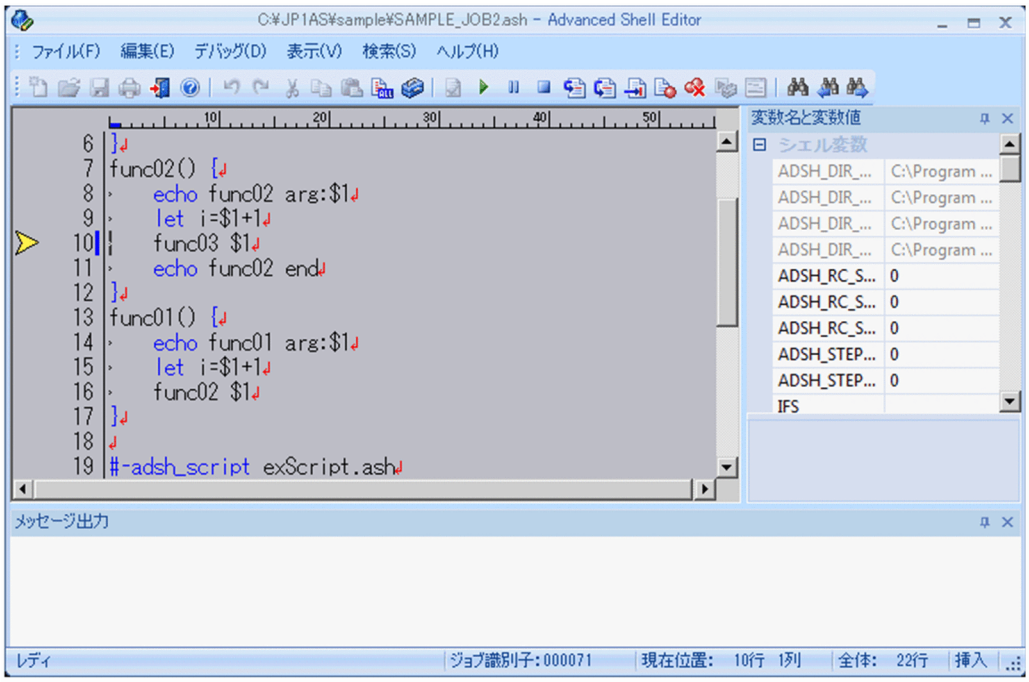
2. デバッグを中止したい場合は，［デバッグ］－［デバッグの中止］メニューを選択する，またはツールバーの［デバッグの中止］ボタンをクリックする。

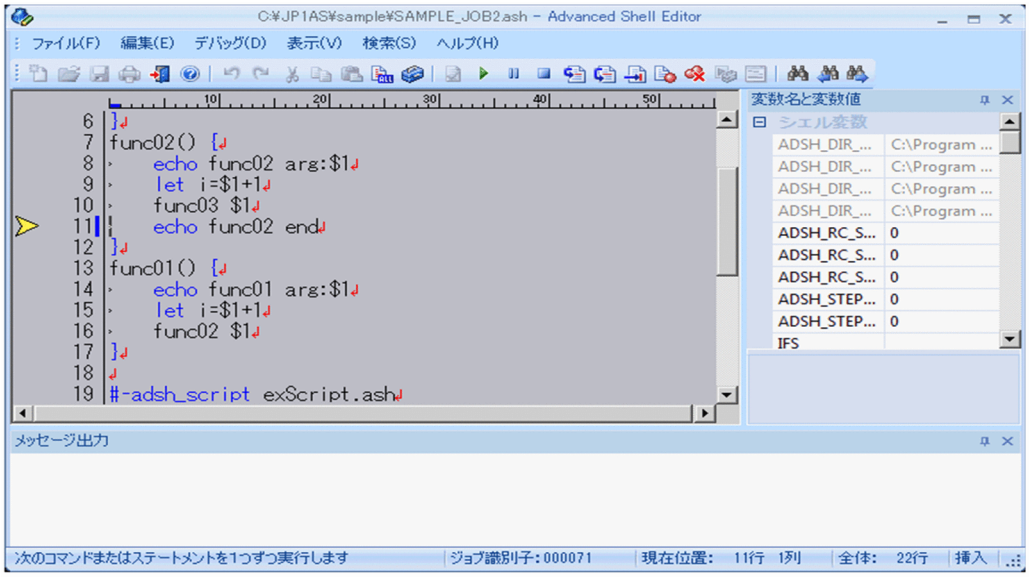
   メニューを選択した時点で，デバッグが中止され，メッセージを出力して後処理をして終了します。なお，プロセス終了前で停止しないでデバッグを終了します。エディタは編集モードに戻ります。

(c)　1 行ずつ実行（関数の中はステップ実行しない）する場合

1. ［デバッグ］－［ステップオーバー］メニューを選択する，またはツールバーの［ステップオーバー］ボタンをクリックする。

   エディタがデバッグモードになり，デバッグが始まります。





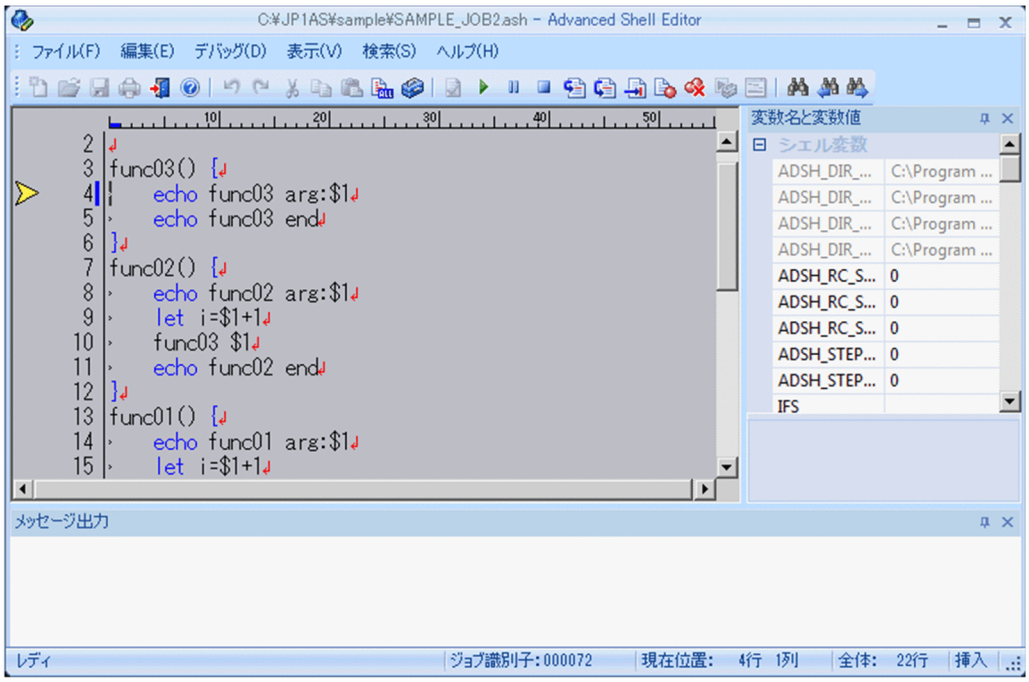
2. デバッグを中止したい場合は，［デバッグ］－［デバッグの中止］メニューを選択する，またはツールバーの［デバッグの中止］ボタンをクリックする。

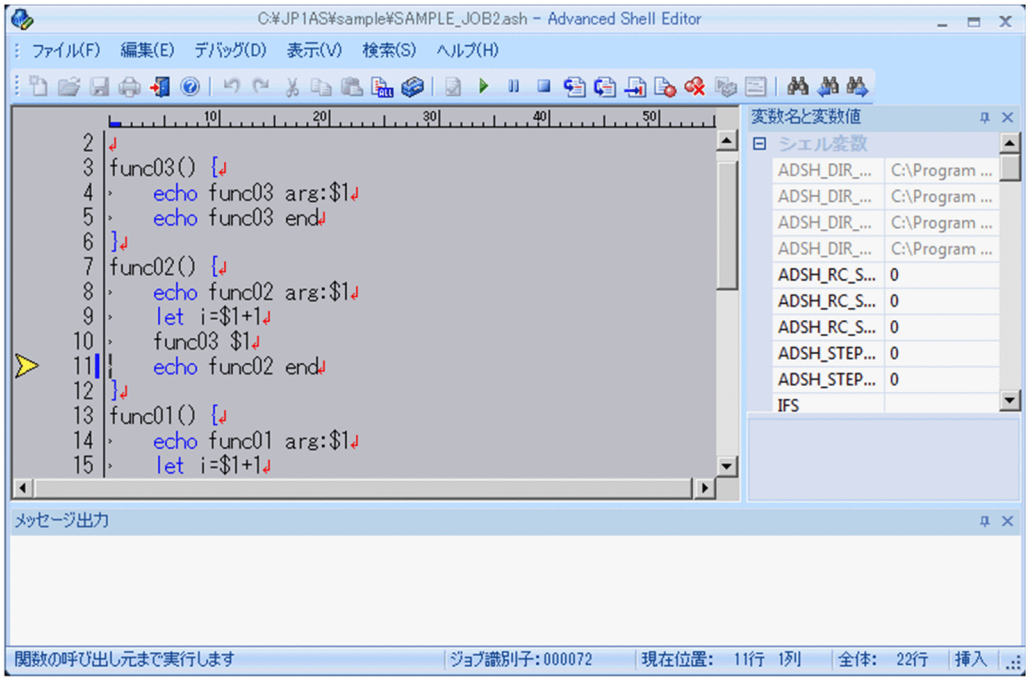
   メニューを選択した時点で，デバッグが中止され，メッセージを出力して後処理をして終了します。なお，プロセス終了前で停止しないでデバッグを終了します。エディタは編集モードに戻ります。

(d)　関数の終わりまで実行する場合

1. ［デバッグ］－［ステップアウト］メニューを選択する，またはツールバーの［ステップアウト］ボタンをクリックする。

   エディタがデバッグモードになり，デバッグが始まります。





(3)　デバッグ中に変数値を参照・更新する

変数ウィンドウには，ジョブ定義スクリプト停止時に変数名とその変数の値が更新して表示されます。

変数値領域を選択するとフィールドの値を変更できます。編集中に Esc キーを押すことで編集前の値に戻せます。

読み込み専用属性の変数は，変数名および変数値を淡色表示しています。この状態のときには，変数値を変更できません。

整数型として宣言された変数に，整数値以外を設定しようとしたときには，元の値に戻ります。また，空文字列を指定した場合は，0 が設定されます。

(4)　エラーをシミュレートする

ジョブ定義スクリプト内のすべての実行パスを実行しても，C1 実行比率が 100%とならない場合があります。この現象は，#-adsh_step_start コマンドのジョブステップに先行するジョブステップまたはエラーとなるコマンドがない場合に発生します。

この場合，#-adsh_step_start コマンド以前に実行する個所でエラーをシミュレートすることで，先行のジョブステップ，またはジョブ定義スクリプトが異常な C1 情報を取得することが可能になり，C1 実行比率を 100%にすることができます。なお，エラー注入モードを有効にした場合のジョブ定義スクリプトの動作は，「6.2.21　エラー注入モードの有効/無効を設定する（joberrmode コマンド）」を参照してください。

手順を次に示します。

1. エラーをシミュレートしたい行にカーソルを移動させ，ブレークポイントを設定する。

   ブレークポイントの設定方法については，「(1)　デバッグ実行時のブレークポイントを設定・解除する」を参照してください。

2. エラーをシミュレートしたい行までデバッグを実行する。

   1.で設定したブレークポイントまでデバッグを実行します。ブレークポイントまで実行するとデバッグの実行が一時的に停止します。デバッグをブレークポイントまで実行する方法については，「(2)　デバッグを実行・中止する」を参照してください。

3. ［デバッグ］－［エラー注入モード］メニューを選択する。

   ［エラー注入モード］メニューは，ブレークポイントなどでジョブ定義スクリプトが停止している場合だけ選択できます。

   これによってエラー注入モードが有効になります。

   この状態でデバッグを再開することでエラーがシミュレートされ，C1 情報が取得されます。

   エラー注入モードを解除するには，デバッグを再開する前に［エラー注入モード］メニューを再選択してください。

4. デバッグを再開する。

   ステップイン，ステップオーバー，ステップアウトまたはブレークポイントまで実行することで，エディタがデバッグモードになり，デバッグが再開されます。

   ジョブ定義スクリプトの最後の行まで実行が終了すると，エラー注入モードは解除されます。

注意事項

- CMDRC_CMDGRP_CHECK パラメーターに FUNCTION を指定した場合，関数内で停止中に［デバッグ］-［エラー注入モード］を選択しても，エラー注入モードを有効にすることはできません。

(5)　trap コマンドのアクションを実行する

ジョブコントローラが強制終了要求を受けた場合の動作は，trap コマンドのアクションによって定義できます。デバッグ実行時に trap コマンドのアクションを実行する方法を次に示します。

1. デバッグ実行によって，トラップアクションが定義された後の任意の行でジョブ定義スクリプトを停止する。

   デバッグ実行の方法については，「(2)　デバッグを実行・中止する」を参照してください。

2. ［デバッグ］-［トラップアクションの実行］メニューを選択する。

   ［トラップアクションの実行］メニューは，ブレークポイントなどでジョブ定義スクリプトが停止している場合にだけ選択できます。

   メニュー選択後，ブレークポイントまで実行されます。このとき，次の順序でコマンドが実行されます。

   1) 現在停止している位置のコマンドを実行

   2) アクション部分のコマンドを実行

   3) 1) の後続のコマンドを実行

   なお，環境設定パラメーター TRAP_ACTION_SIGTERM に DISABLE を指定している場合，または trap コマンドによるアクションの定義がない場合，このメニューを選択しても trap コマンドのアクションを実行しないでブレークポイントまで実行されます。

注意事項

- trap コマンドのアクションの実行中に，ジョブ定義スクリプトの実行は停止できません。
- この機能によるアクションの実行中にジョブが終了した場合，ジョブを強制終了した場合とは異なり，最後に実行したコマンドの終了コードがジョブの終了コードに反映されます。例えば，この機能を使用して，アクションで「exit 2」を実行した場合，終了コード 2 でジョブが終了しますが，ジョブを強制終了してアクションで「exit 2」が実行された場合，終了コード 1 でジョブがエラー終了します。
- このメニューを選択した後，直後のコマンドの実行がスキップされる場合※，アクションは実行されません。

  注※　例えば，次に示す場合にコマンドの実行がスキップされます。

  ・エラー注入モードを有効にする。

  ・onError 属性に stop を指定したジョブステップ内のコマンドがエラー終了する。

4.4.7　カバレージ情報を表示する

カバレージを採取している場合，エディタで開いているジョブ定義スクリプトまたはデバッグ実行中のジョブ定義スクリプトに対してカバレージ情報を表示します。デバッグ中の場合は採取中のカバレージ情報が表示され，デバッグ終了後の場合は最後に採取したカバレージ情報が表示されます。カバレージ情報は一時ファイルに出力され，メモ帳（notepad.exe）で表示されます。

カバレージ情報を表示するための［カバレージ情報の表示］メニューは，［実行環境の設定］でカバレージを［蓄積する］に設定した場合に有効となります。［蓄積する］に設定していない場合，メニューがグレーアウトされ，選択できません。

エディタの終了やデバッグの終了時でも，メモ帳は表示されたままです。デバッグ実行中にカバレージ情報が変更されても，表示内容は更新されません。

カバレージ情報を表示したあと，カバレージ情報の表示を中止する手順を次に説明します。

1. ［表示］－［カバレージ情報の表示］メニューを選択する，またはツールバーの［カバレージ情報の表示］ボタンをクリックする。

   メモ帳を開いて，カバレージ情報が表示されます。表示されたカバレージ情報は任意のファイル名で保存することもできます。

2. カバレージ情報の表示を終了する場合は，メモ帳を閉じて終了する。

   カバレージ情報の表示を終了します。

4.5　既存のジョブ定義スクリプトを編集する【Windows 限定】

JP1/Advanced Shell エディタで既存のジョブ定義スクリプトを編集する方法について説明します。ジョブ定義スクリプトファイルを編集するには，3 つの開始方法があります。

右クリックメニューからの開始方法

1. エクスプローラからジョブ定義スクリプトファイルを右クリックする。
2. ［編集］を選択する。

ドラッグアンドドロップでの開始方法

1. エクスプローラからジョブ定義スクリプトファイルをドラッグする。
2. 「エディタ」アイコン，またはすでに起動しているエディタウィンドウにドロップする。

なお，エディタウィンドウについては，「4.3　JP1/Advanced Shell エディタの操作【Windows 限定】」を参照してください。

エディタのメニューからの開始方法

1. ［ファイル］－［開く］メニューを選択する，または［ファイル］－編集履歴から編集を開始する。
2. 編集する既存のジョブ定義スクリプトファイルを選択する。

4.6　ジョブ定義スクリプトを保存する【Windows 限定】

JP1/Advanced Shell エディタでジョブ定義スクリプトを保存する方法について説明します。

1. ジョブ定義スクリプトファイルを上書き保存したい場合は，［ファイル］－［保存］メニューを選択する。
   ジョブ定義スクリプトファイルを上書き保存します。
2. ジョブ定義スクリプトファイルを別名で保存したい場合は，［ファイル］－［名前を付けて保存］メニューを選択する。
   ジョブ定義スクリプトファイルを別名で保存します。

4.7 JP1/Advanced Shell エディタウィンドウの画面の詳細【Windows 限定】

JP1/Advanced Shell エディタウィンドウの操作中に表示されるダイアログボックスおよびウィンドウの一覧を，次に示します。

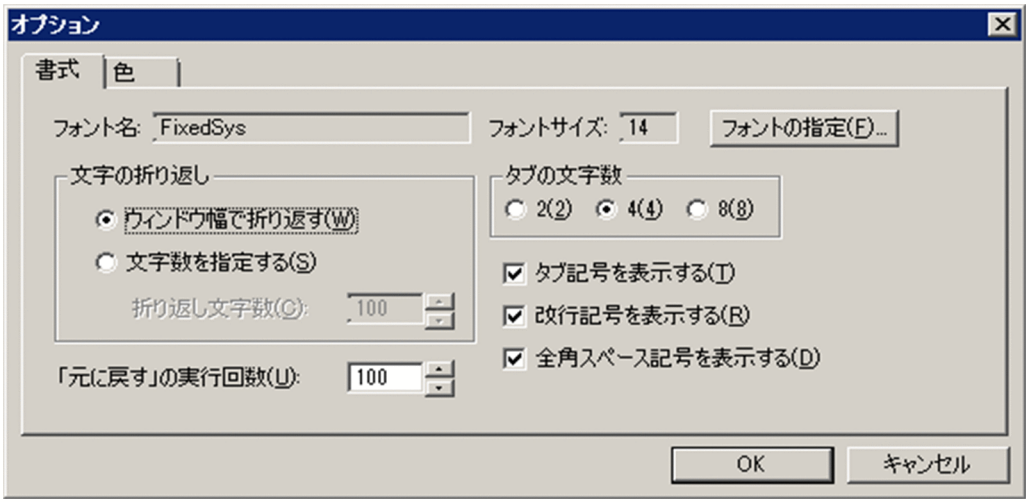
- オプション（書式）ダイアログボックス
- オプション（色）ダイアログボックス
- 実行環境の設定ダイアログボックス
- メッセージ出力ウィンドウ
- 検索ダイアログボックス
- 変数ウィンドウ
- コンソール

4.7.1 オプション（書式）ダイアログボックス

JP1/Advanced Shell エディタウィンドウで，［編集］－［オプション］メニューを選択すると，［オプション］ダイアログボックスが表示されます。

［オプション］ダイアログボックスには，「書式」と「色」のタブがあります。

「書式」タブを選択すると，［オプション（書式)］ダイアログボックスが表示されます。



(1) ダイアログボックスの項目

フォント名

現在使用している文字のフォント名が表示されます。フォントを変更するには，［フォントの指定］ボタンをクリックします。

デフォルトでは「FixedSys」が設定されています。

フォントサイズ

現在使用している文字のフォントサイズを指定します。フォントサイズを変更するには，［フォントの指定］ボタンをクリックします。

デフォルトでは「14」が設定されています。

文字の折り返し

文字を折り返す方法を選択します。

デフォルトでは「ウィンドウ幅で折り返す」が選択されています。

ウィンドウ幅で折り返す

文字をウィンドウ幅で折り返す場合に選択します。

文字数を指定する

文字を固定の文字数で折り返す場合に選択します。

折り返し文字数

折り返す文字数を指定します。

「文字数を指定する」が選択されている場合にだけ指定できます。

20～512（単位：バイト）の範囲で指定してください。

デフォルトでは「100」が指定されています。

タブの文字数

タブの文字数（単位：バイト）を選択します。

デフォルトでは「4」が選択されています。

タブ記号を表示する

タブを示す記号の表示／非表示を指定します。

デフォルトではチェックされています。

改行記号を表示する

改行を示す記号の表示／非表示を指定します。

デフォルトではチェックされています。

全角スペース記号を表示する

全角スペースを表す記号の表示／非表示を指定します。

デフォルトではチェックされています。

「元に戻す」の実行回数

［編集］－［元に戻す］を何回まで実行できるようにするかを指定します。

10～999 の範囲で指定してください。デフォルトでは「100」が指定されています。

(2)　ダイアログボックスでの操作

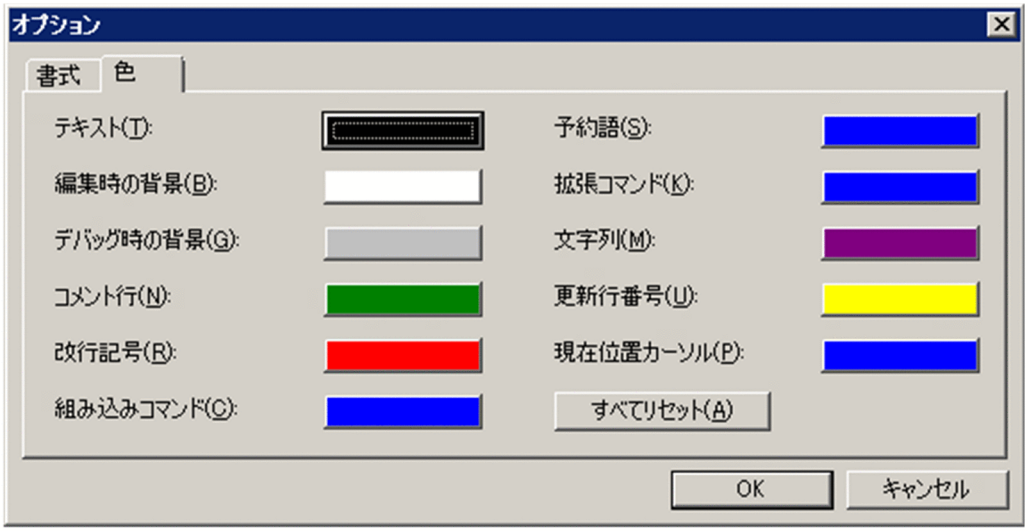
- ［OK］ボタンをクリックすると，指定した書式が設定されてダイアログボックスが閉じます。
- ［キャンセル］ボタンをクリックすると，書式を変更しないでダイアログボックスが閉じます。

4.7.2　オプション（色）ダイアログボックス

JP1/Advanced Shell エディタウィンドウで，［編集］－［オプション］メニューを選択すると，［オプション］ダイアログボックスが表示されます。

［オプション］ダイアログボックスには，「書式」と「色」のタブがあります。

「色」タブを選択すると，［オプション（色)］ダイアログボックスが表示されます。



(1)　ダイアログボックスの項目

テキスト

テキストの色を指定します。

デフォルトでは「システムカラー」が設定されています。

編集時の背景

編集モードのときの背景色を指定します。

デフォルトでは「システムカラー」が設定されています。

デバッグ時の背景

デバッグモードのときの背景色を指定します。

デフォルトでは「灰色」が設定されています。

コメント行

コメント行の色を指定します。

デフォルトでは「緑色」が設定されています。

改行記号

改行記号の色を指定します。

デフォルトでは「赤色」が設定されています。

組み込みコマンド

組み込みコマンドの色を指定します。

デフォルトでは「青色」が設定されています。

予約語

予約語および「]]」の色を指定します。

デフォルトでは「青色」が設定されています。

拡張コマンド

拡張コマンドの色を指定します。

デフォルトでは「青色」が設定されています。

文字列

文字列の色を指定します。

デフォルトでは「暗紫色」が設定されています。

更新行番号

更新行番号の色を指定します。

デフォルトでは「黄色」が設定されています。

現在位置カーソル

現在位置を示すカーソルの色を指定します。

デフォルトでは「青色」が設定されています。

すべてリセット

すべての項目に対する色の指定をデフォルト値に戻します。

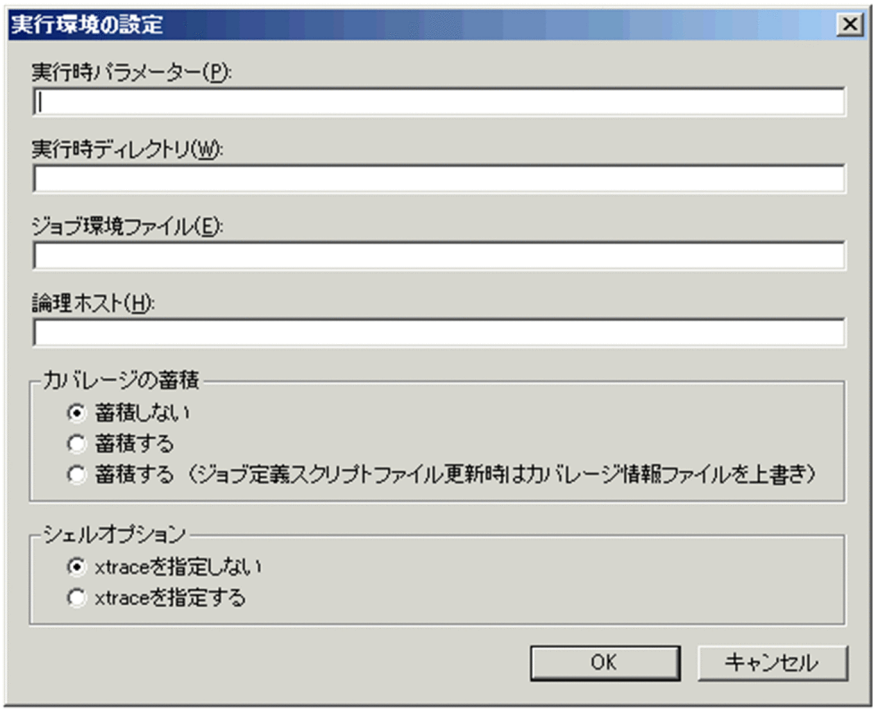
(2) ダイアログボックスでの操作

- ［すべてリセット］以外の各項目のボタンをクリックすると，［色の設定］ダイアログボックスが表示され，色が選択できます。さらに，［色の設定］ダイアログボックスの［色の作成］ボタンをクリックすると，［色の作成］ダイアログボックスが表示され，色を作成して指定できます。
- ［OK］ボタンをクリックすると，指定した色が設定されてダイアログボックスが閉じます。

- ［キャンセル］ボタンをクリックすると，色を変更しないでダイアログボックスが閉じます。

4.7.3　実行環境の設定ダイアログボックス

JP1/Advanced Shell エディタウィンドウで，［デバッグ］－［実行環境の設定］メニューを選択すると，［実行環境の設定］ダイアログボックスが表示されます。



(1)　ダイアログボックスの項目

実行時パラメーター

ジョブ定義スクリプトに渡す実行時パラメーターを指定します。

例

ファイル名が「a.ash」のジョブ定義スクリプトを使用する場合，実行時パラメーターに「ABC」を指定すると，「ABC」が「a.ash」に対する第 1 引数になります。

実行時ディレクトリ

ジョブ定義スクリプトを実行するカレントドライブ，またはフォルダを指定します。

指定がない場合は，エディタのカレントフォルダになります。

エディタのカレントフォルダは次のどれかになります。

- ジョブ定義スクリプトファイルがあるフォルダ
- プログラムフォルダ
- プログラムのカレントフォルダ

- ショートカットの作業フォルダ

例

ジョブ定義スクリプト名が a.ash のファイルにジョブ定義スクリプト「pwd」が記載されていたとします。実行時ディレクトリに「C:¥」を指定すると，次のように Windows 環境で指定した場合と同じになります。

```
C:¥> a.ash
C:¥
C:¥>
```

ジョブ環境ファイル

デバッグ時に使用するジョブ環境ファイルを指定します。指定したファイルは，デバッグ対象のジョブで使用されます。

指定を省略した場合は，環境変数 ADSH_ENV に指定されたファイルが使用されます。環境変数 ADSH_ENV にもファイルが指定されていない場合は，デフォルト値で実行されます。

論理ホスト

ユーザー応答機能で使用する論理ホストを指定します。デバッグ対象のジョブで実行するユーザー応答機能は，指定した論理ホストで実行されます。

指定を省略した場合は，物理ホストで実行されます。

カバレージの蓄積

カバレージ情報を蓄積するかどうかを指定します。

- 蓄積しない

  カバレージを蓄積しません。adshexec コマンドの-t オプションを指定しない場合に相当します。デバッグ実行中のカバレージ情報の表示機能は有効になりません。

- 蓄積する

  カバレージを蓄積します。ジョブ定義スクリプトファイルが更新されている場合は，ジョブ定義スクリプトを実行しないで終了します。adshexec コマンドの-t オプションの指定に相当します。エディタからカバレージ情報を表示できます。

- 蓄積する（ジョブ定義スクリプトファイル変更時はカバレージ情報ファイルを上書き）

  カバレージを蓄積します。ジョブ定義スクリプトファイルが更新されている場合は，蓄積されたカバレージ情報を破棄し，新規に蓄積を開始します。adshexec コマンドの-t オプションと-f オプションを同時に指定した場合に相当します。エディタからカバレージ情報を表示できます。

デフォルトでは「蓄積しない」が選択されています。

カバレージ情報は，カバレージ情報ファイル（asc ファイル）に保存されます。asc ファイルは，ジョブ定義スクリプトファイルがあるディレクトリに作成されます。asc ファイルがすでに存在する場合は，ジョブ定義スクリプトファイルがあるディレクトリの asc ファイルを使用します。

カバレージ情報ファイルに蓄積されたカバレージ情報を表示する場合は，エディタの［表示］－［カバレージ情報の表示］メニューまたは adshcvshow コマンドを使用します。また，2 つのカバレージ情報ファイルのカバレージ情報をマージする場合は，adshcvmerg コマンドを使用します。カバレージ情報については，「3.10　カバレージ情報を取得する」を参照してください。

シェルオプション

シェルオプション xtrace を指定するかどうかを指定します。

- xtrace を指定しない

  デバッグ実行開始時にシェルオプション xtrace を指定しません。adshexec コマンドに-x オプションを指定しない場合に相当します。

- xtrace を指定する

  デバッグ実行開始時にシェルオプション xtrace を指定します。adshexec コマンドに-x オプションを指定する場合に相当します。

  この項目を選択すると，実行したコマンドとその引数がトレース情報として標準エラー出力へ出力されます。詳細については，「3.6　実行したコマンドとその引数を出力する」を参照してください。

デフォルトでは「xtrace を指定しない」が選択されています。

このほかにシェルオプションには，使用できる機能の制限や，実行モードの切り替えをするための機能があります。シェルオプションについては，「5.6　シェルオプション」を参照してください。

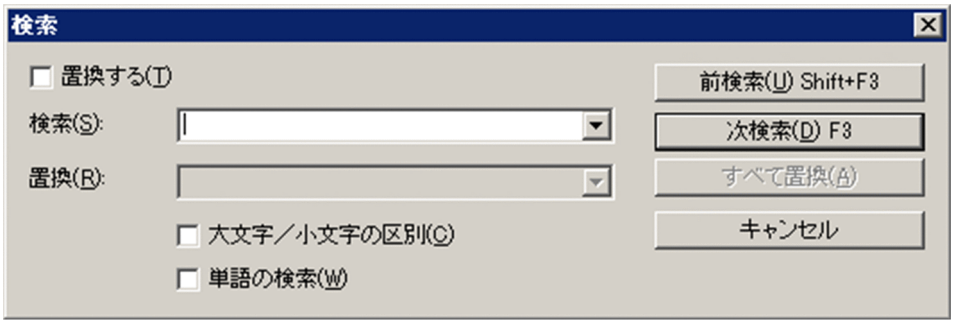
(2)　ダイアログボックスでの操作

- ［OK］ボタンをクリックすると，ダイアログボックスで設定した情報が各実行環境ファイルとして作成されます。
- ［キャンセル］ボタンをクリックすると，実行環境ファイルを作成しないでダイアログボックスが閉じます。

4.7.4　検索ダイアログボックス

JP1/Advanced Shell エディタウィンドウで，［検索］－［検索］メニューを選択すると，［検索］ダイアログボックスが表示されます。

［検索］－［検索］メニューを選択しないで［検索］－［前検索］，または［次検索］メニューを選択した場合も，［検索］ダイアログボックスが表示されます。



(1) ダイアログボックスの項目

置換する

文字列を置換する場合はチェックします。

デフォルトではチェックされていません。

検索

検索する文字列を指定します。

置換

検索文字列を置き換える文字列を指定します。「置換する」がチェックされている場合だけ，指定できます。

大文字／小文字の区別

大文字と小文字を区別して検索する場合はチェックします。

デフォルトではチェックされていません。

単語の検索

単語だけを検索する場合にチェックします。

デフォルトではチェックされていません。

(2) ダイアログボックスでの操作

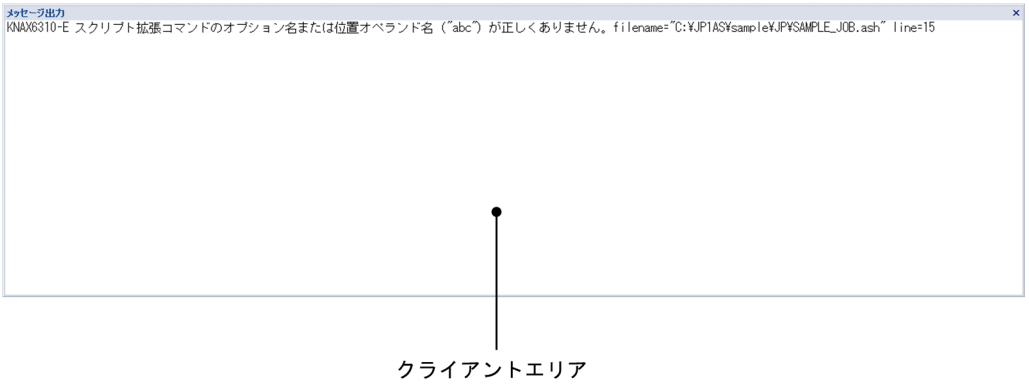
- ［前検索］ボタンをクリックすると，検索文字列を上方向に検索します。
- ［次検索］ボタンをクリックすると，検索文字列を下方向に検索します。
- ［すべて置換］ボタンをクリックすると，ジョブ定義スクリプトファイル中にあるすべての検索文字列を「置換」で指定した文字列に置換します。
- ［キャンセル］ボタンをクリックすると，ダイアログボックスが閉じます。

(3) 注意事項

- 「置換」，および「検索」に指定した文字列は，過去 10 件分だけドロップダウンリストに記憶されます。
- 検索文字列がない場合はビープ音が鳴ります。

4.7.5 メッセージ出力ウィンドウ

デバッグ実行で発生したエラーを表示します。ジョブ定義スクリプト停止中に表示します。



(1)　クライアントエリア

内容

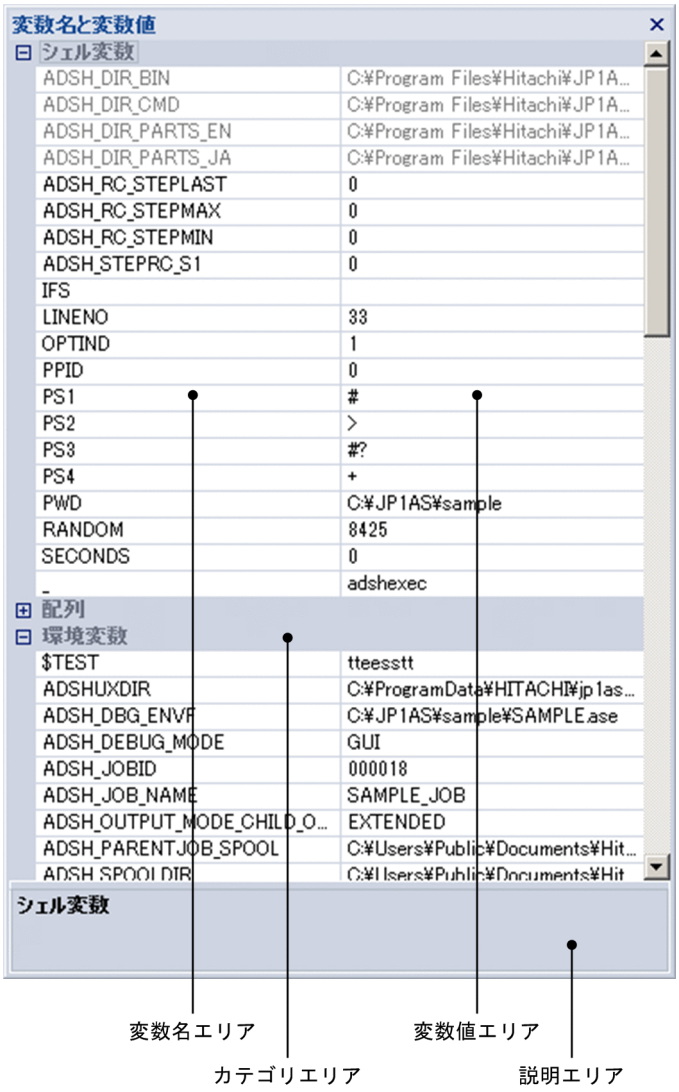
エラーの内容を表示します。

(2)　エラーウィンドウの操作

- 現在編集中のジョブ定義スクリプトファイルで発生したエラーの場合，エラーメッセージが表示されている行をダブルクリックすると，該当するエラー行の先頭にカーソルが移動します。
- 現在編集中のジョブ定義スクリプトファイルで発生したエラーの場合，エラーメッセージが表示されている行を右クリックすると，該当するエラーの個所にカーソルを移動するための，［ジャンプ］ポップアップメニューが表示されます。
- Ctrl キーと同時に左クリックをすると複数行を選択することができます。Shift キーと同時に左クリックをすると，連続した行を選択することができます。選択した行は右クリックすると表示される［コピー］ポップアップメニューまたは Ctrl+C でクリップボードにコピーすることができます。

4.7.6　変数ウィンドウ

デバッグ実行時に変数名と変数値を表示します。



(1)　変数名エリア

現在実行中のジョブ定義スクリプトで定義されている変数名が表示されます。

(2)　変数値エリア

変数の値が表示されます。読み込み専用属性の場合は淡色表示になります。

(3)　カテゴリエリア

変数のカテゴリを表示します。次に示す文字列を表示します。

- シェル変数
  「エクスポート属性が付いている変数（環境変数）と配列」以外の変数を表示します。
- 配列
  配列を表示します。配列カテゴリの下には，配列名をカテゴリ名として表示します。

- 環境変数
  エクスポート属性が付いている変数（環境変数）を表示します。

(4)　説明エリア

選択された変数の属性を表示します。次に示す文字列を表示します。

斜体は可変文字列です。

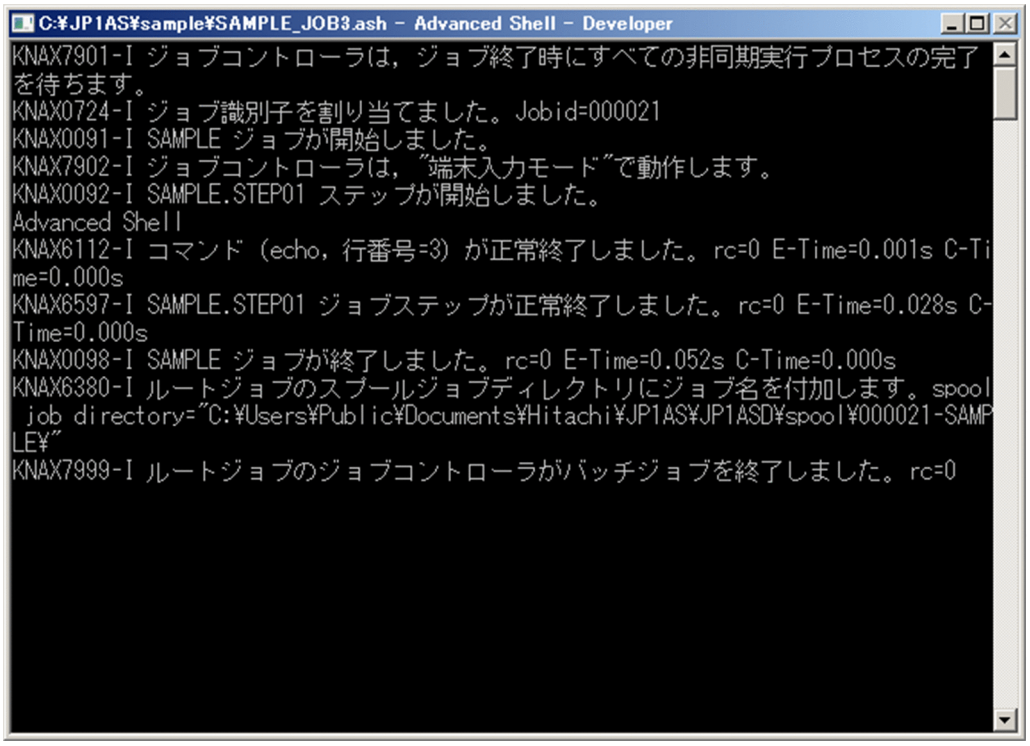
- 変数名：*変数名*
  配列の場合は，配列名称と添え字を合わせて表示します。配列以外は変数名をそのまま表示します。
- 環境変数
  エクスポートされている変数の場合に表示します。
- 整数型
  typeset -i で指定された変数の場合に表示します。変数値エリアで，整数値以外を指定することができません。
- 文字列型
  typeset -i で指定されていない変数の場合に表示します。
- ステップローカル属性
  スクリプト拡張コマンドの#-adsh_step_start コマンドの stepVar 属性に指定したシェル変数の場合に表示します。
- 読み込み専用属性
  読み込み専用の変数の場合に表示します。
- パス変換属性
  パス変換が有効な場合に，パス変換の対象となる変数の場合に表示します。

(5)　変数ウィンドウの操作

変数値をダブルクリックすると，変数の値を編集することができます。

4.7.7　コンソール

デバッグ実行時に，標準出力，標準エラー出力およびジョブ実行ログ相当の情報を表示します。



(1) コンソールの操作

- デバッグ対象のジョブが終了するとコンソールは閉じます。環境ファイルおよびジョブ定義スクリプトの文法エラーなどがある場合，エラーウィンドウおよびジョブ識別子を基にスプールに出力されている実行結果を参照して対応してください。
- プロパティの設定方法はコマンドプロンプトの設定方法と同じです。

5 ジョブ定義スクリプトの作成

ジョブ定義スクリプトの文法について説明します。

5.1　ジョブ定義スクリプトを構成する基本要素

この節では，ジョブ定義スクリプトを構成する基本要素について説明します。

重要

ジョブ定義スクリプトを記述する際は次の点に注意してください。

- ジョブ定義スクリプトの 1 行は 8,191 バイト以下にしてください。1 行が 8,192 バイト以上の場合，ジョブ定義スクリプトはエラー終了します。

  ただし，継続行については異なる行として扱います。継続行については，「(3)　行継続」を参照してください。

- スクリプト拡張コマンドを使用した行については，スクリプト拡張コマンドの制限事項に従って，継続する行を含めて，8,191 バイト以下にしてください。

  スクリプト拡張コマンドの制限事項については，「(1)　制限事項」を参照してください。

- ファイルの入出力時にファイルパスを変換する場合は，ファイル入出力が発生するタイミングでバイト数が変わります。そのため，変換による 1 行の上限を超えたときでもエラーとはしないで処理を継続します。

5.1.1　予約語

JP1/Advanced Shell では，ジョブ定義スクリプト内で使用する特殊な字句を予約語として登録しています。予約語はコマンドの最初の単語として使用された場合に意味を持ち，引用符で囲まれていないかぎり予約語として認識されます。予約語をコマンドの 2 番目以降に使用した場合，通常の変数として扱われます。そのため，予約語と同じ字句を使用する場合は注意が必要です。

予約語の確認はシェル標準コマンドの command -V，whence -v で行います。command コマンドおよび whence コマンドについては，「9.3　シェル標準コマンド」の「command コマンド（コマンドを実行する）」または「whence コマンド（文字列をコマンドとした場合の解釈を表示する）」を参照してください。

予約語を次に示します。

! [[{ } case do done elif else esac fi for function if in select then time until while

5.1.2　変数

変数とはジョブ定義スクリプト内で値を代入する領域のことです。変数の作成および変数の値を参照できます。また，変数は，エクスポートによって子プロセスに環境変数として引き継がせることができます。変数のことをシェル変数ともいいます。

(1)　変数の命名規則

作成する変数の名称は，命名規則の範囲で任意に指定できます。また，英字は大文字と小文字を区別するため，同じスペルであっても異なる変数名になります。

ただし，Windows 環境で小文字が含まれる変数をエクスポートし，環境変数として使用しようとすると，VAR_ENV_NAME_LOWERCASE パラメーターの指定によってはエラーとなります。

変数の命名規則を次に示します。環境変数の命名規則については「(a)　環境変数の命名規則【Windows 限定】」を参照してください。

- 使用できる文字は英数字と_（アンダースコア）だけです。
- 先頭文字は数字以外にしてください。
- 変数名の長さは無制限です。ただし，入力行の上限および CUI デバッガのコマンド入力文字数の上限が存在します。そのため，ジョブ定義スクリプト内で使用する変数の長さは上限以下であることを推奨します。

  入力行の上限については，「5.1　ジョブ定義スクリプトを構成する基本要素」を参照してください。
  CUI デバッガのコマンド入力文字数の上限については，「6.1.4　デバッガのコマンド一覧【UNIX 限定】」を参照してください。

(a)　環境変数の命名規則【Windows 限定】

VAR_ENV_NAME_LOWERCASE パラメーターの指定によって，使用できる環境変数名は次のように異なります。

- VAR_ENV_NAME_LOWERCASE パラメーターにDISABLE を指定した場合

  小文字を含む環境変数名は使用できません。

  小文字が含まれる変数名をエクスポートしようとするとエラーとなります。

- VAR_ENV_NAME_LOWERCASE パラメーターにENABLE を指定した場合

  小文字を含む環境変数名を使用できます。ただし，「ADSH」から始まる環境変数名は小文字では定義しないでください。

  指定時の注意事項を次に示します。

  - ジョブ定義スクリプトで小文字のシェル変数名をエクスポートした場合，そのあと呼ばれる外部コマンドには，環境変数名が小文字のまま渡ります。Windows では大文字と小文字を区別しないため，スペルが同じ環境変数名を同一と解釈し，最後にエクスポートした値が環境変数として渡されます。しかし，シェル変数は大文字・小文字が区別されるため，それぞれ別の値を保持するので，注意してください。

    例えば，次の指定では環境変数名の「ABC」と「abc」は区別されないため，最初は環境変数値として「lowercase」が設定されますが，最後に「uppercase」が設定されます。

    ```
    export abc=lowercase
    export ABC=uppercase
    ```

- シェル変数をエクスポートした場合，そのシェル変数のスコープ内で同じスペルのシェル変数のexport属性は無効になります。配列でもすべての要素のexport属性が無効になります。

  ただし，スコープ外でエクスポートされていた変数と同じスペルの関数内ローカル変数をエクスポートした場合は，export属性は無効になりません。

コマンドやパラメーターで小文字を指定できるかどうかは，VAR_ENV_NAME_LOWERCASEパラメーターの設定によって次のように異なります。

コマンドまたはパラメーター	VAR_ENV_NAME_LOWERCASEパラメーターの設定値	
	DISABLE	ENABLE
exportパラメーター（環境設定パラメーター）	大文字・小文字を指定できるが，区別はされない。	同左
exportコマンド， typeset -xコマンド	小文字のシェル変数名は指定できない。	小文字のシェル変数名も指定できるが，環境変数名の大文字・小文字は区別されない。 ただし，シェル変数としては大文字・小文字は区別され，別のシェル変数と見なされる。
setコマンド （-aオプション）	setコマンド以降のシェル変数をすべてエクスポートするが，小文字のシェル変数に値を設定するとエラーになる。	setコマンド以降のシェル変数をすべてエクスポートするが，小文字のシェル変数でも値を設定すると，そのシェル変数がエクスポートされる。 ただし，環境変数名の大文字・小文字は区別されない。
unsetコマンド， readonlyコマンド， readコマンド	シェル変数名なので小文字を指定できる。	同左
#-adsh_fileコマンド， #-adsh_file_tempコマンド， #-adsh_spoolfileコマンド	ファイル環境変数定義名には小文字を指定できない。	ファイル環境変数定義名に小文字を指定できる。 ただし，環境変数名の大文字・小文字は区別されない。
#-adsh_step_startコマンド	ジョブステップ名はステップの戻り値を格納するシェル変数名に使用されるが，小文字を指定できる。 -stepVarで指定するシェル変数名は小文字を指定できる。	同左
#-adsh_path_varコマンド	変数名に小文字を指定できる。	同左
スクリプト拡張コマンドの{*環境変数名*}による置換	環境変数名に小文字を指定できる。	同左
GUIデバッガでウォッチするシェル変数	シェル変数名に小文字を指定できる。	同左

コマンドまたはパラメーター	VAR_ENV_NAME_LOWERCASE パラメーターの設定値	
	DISABLE	ENABLE
adshread コマンド adshvarconv コマンド	シェル変数名に小文字を指定できる。	同左
for *シェル変数*	シェル変数名に小文字を指定できる。	同左
awk コマンドのENVIRON 組み込み変数	添え字に環境変数名を指定できるが，小文字も指定できる。	同左
adshjava コマンドのバッチアプリケーションに渡す引数	システムプロパティで環境変数名を指定できるが，adshjava コマンドは文字種別をチェックしない。	同左
稼働実績情報取得機能で採取する環境変数	環境変数名と値を採取するが，環境変数名の文字種別をチェックしない。	同左
PATH_CONV_VAR パラメーター PATH_CONV_NOVAR パラメーター	シェル変数名に小文字を指定できる。	同左

(2)　変数の作成と値の代入

変数を作成，および値を代入する場合の書式を次に示します。

変数名=値

変数は=の左項に変数名を記述することで作成されます。作成された変数には，値の書き込みおよび値の読み込みができます。変数に値を代入する場合は，=の右項に値を記述します。変数は次の点に注意して作成してください。

- 変数の属性が読み込み専用の場合，変数への値の代入はエラー終了し，ジョブは終了します。変数の属性を読み込み専用に変更する場合は，シェル標準コマンドの readonly コマンドを使用します。

  readonly コマンドについては，「9.3　シェル標準コマンド」の「readonly コマンド（変数の属性を読み込み専用に変更する，または読み込み専用の変数を表示する）」を参照してください。

- 変数名に未作成の変数を指定した場合は，変数の作成と同時に値が代入されます。変数に代入する値が文字列の場合，何文字でも代入できます。

  しかし，typeset コマンドで整数型に定義した変数に数値を代入する場合や，変数に代入した数値を使用して算術演算する場合は，変数の値および算術結果が-2147483648～2147483647 の範囲内でなければなりません。範囲外の値を指定した場合，正しい結果を求めることができません。

- =の両脇にはスペースを入力しないでください。スペースが入っていた場合，変数は作成されません。

- 変数にスペースやメタキャラクタを含む文字列を代入する場合は，クォーテーション（'または"）で囲んでメタキャラクタを無効化にするか，エスケープ文字を使用してください。メタキャラクタおよびメタキャラクタの無効化については，「5.1.6　メタキャラクタ」を参照してください。

(3) 変数の値の参照

(a) 参照方法

変数の値を参照する場合の書式を次に示します。

```
$変数名
または
${変数名}
```

変数に代入された値は，変数名に$を付けることで参照できます。参照する変数は変数名と完全に一致した変数が対象となります。ただし，変数名に使用できない文字を指定した場合，それまでの文字を変数名と認識し処理をします。

参照する変数名の文字列に，変数名には指定できない文字を指定した場合の実行例

```
abc=xxx
echo $abc@zzz
```

→xxx@zzz が標準出力に出力されます。

また，参照する変数名を明示的に指定したい場合は，変数名を{ }で囲むことで参照できます。ただし，{ }で囲った場合は，変数名に使用できない文字が含まれていても変数名の一部として扱い処理をします。

変数 abc を明示的に指定し，参照する場合の実行例

```
abc=xxx
abcdef=yyy
echo ${abc}def
```

→xxxdef が標準出力に出力されます。

(b) offset（参照起点）と length（参照長）を指定した参照方法

offset（参照起点）と length（参照長）を指定して変数の値を参照する場合の書式を次に示します。

```
${変数名:offset}
```

または

```
${変数名:offset:length}
```

または

```
${変数名::length}
```

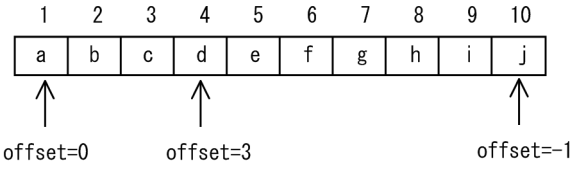
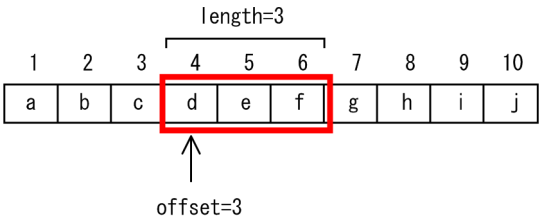
参照する変数に代入された値の特定の部分を参照したい場合，変数名に続けて「:offset」または「:offset:length」を指定します。

offset や length には変数名の命名規則に示されている文字および数値以外は指定できません。offset や length に指定する変数名および数値は，次の規則に従い記述してください。

指定値	指定規則
数値（符号なし）の場合	指定値の前後にはスペース，タブは指定できません。
数値（符号あり）の場合	符号の直前にはスペースを指定する必要があります。それ以外の指定値の前後にはスペース，タブは指定できません。
変数の場合	指定値の前後にはスペース，タブは指定できません。
offset の指定省略の場合	スペースの指定はできますが，タブの指定はできません。

複数行にわたって記述する場合，行の終端に¥を付けて改行できます。

offset と length に指定できる数値の範囲を次に示します。

種類	指定範囲（単位：文字）※	指定例
offset	-65535～65535	0 を指定した場合，文字列の先頭から出力されます。 正の値を指定した場合，文字列の先頭+1 番目を起点として出力されます。 負の値を指定した場合，文字列の終端から数えた位置が起点となります。 （例） 文字列数 10 のデータに対し，offset に 0，3，または-1 を指定した場合の参照先を次に示します。 
length	0～65536	（例） 文字列数 10 のデータに対し，offset=3，length=3 を指定した場合，出力できる文字の範囲は次の図の太線で囲まれた部分です。 

注※

数値は次のどちらかの方法で指定できます。

- 8 進数，10 進数，16 進数の数値

  指定した数値は次のように自動判別されます。

  ・0x から始まる文字列(0xa)は，16 進数と判別されます。

・0 から始まる文字列(012)は，8 進数と判別されます。

・8 進数でも 16 進数でもない数値は，10 進数と判別されます。

・-0 は，0 を指定した場合と同等に扱われます。

・数値に符号（「-」または「+」）を付けた場合，「:」と符号との間には 1 つ以上のスペースを入れる必要があります。

- 数値を代入した変数名

  指定した変数名の変数が存在しない場合，または変数に代入値が指定されていない場合は，変数値には 0 が仮定されます。

  複数の変数を再帰的に参照する変数名を指定した場合，**offset** の再帰回数が 1,024 回，**length** の再帰回数が 1,025 回を超えると，エラーになります。

offset および **length** の指定に関する注意事項を次に示します。

- 次の指定は構文エラーとなります。
  - **offset** または **length** に，数値でも変数名でもない文字列を指定した場合

    ```
    echo ${ABC:123D}
    ```

    →「123D」は数値でも変数名でもないためエラーとなる
  - **offset** または **length** に指定した変数名に，数値でも変数名でもない文字列が設定されている場合

    ```
    CNT=123D
    echo ${ABC:CNT}
    ```

    →指定した変数の値「123D」は数値でも変数名でもないためエラーとなる
  - **offset** を不当に省略した場合

    ```
    echo ${ABC:}
    ```

    →**offset** の指定がないため，エラーとなる
  - **offset** および **length** に算術式を指定した場合

    ```
    echo ${ABC:10-2}
    ```

    →算術式が指定されているため，エラーとなる
  - **offset** および **length** に指定した変数に$や${}を付けた場合

    ```
    ABC=abcdefghijklmn
    AA=1
    echo ${ABC:$AA}
    ```

    →**offset** および **length** に指定する変数に$や${}を付けているため，エラーとなる
  - **offset** の直前または **length** の直前にタブを指定した場合

    （次の例では **offset** の直前にタブを指定しています。「→」はタブを示します）

    ```
    ABC=abcdefghijklmn
    AA=1
    BB=1
    ```

echo ${ABC:→AA:BB}

→**offset** の直前にタブを指定しているため，エラーとなる

- **offset** または**変数**に設定されている数値が，**変数**に設定された文字列の長さを超えている場合，**変数**に設定されている文字列は取り出されません。

```
ABC=abcdefghijklmn
echo ${ABC:20}
```

→変数 ABC の文字列の長さが 14 文字しかないため，取り出せない

- **offset** または**length** に指定した変数名が未定義か，変数値が空の場合，**offset** または**length** は 0 が指定されたものとして処理します。
- **offset** や**length** に指定する変数の値を typeset コマンドで属性変更した場合，変更方法によっては基数が変更（-Z オプションでゼロ詰めを指定すると 8 進数扱いとなる）されて，参照範囲が変わってしまう場合があります。また，基数変更で指定値が不正となることがあります。

  次の例では，typeset コマンドでの属性変更でゼロ詰めを指定することで，変数 L1 に指定した 12 が 012 となり，8 進数と解釈されます。

```
ABC=abcdefghijklmnopqrstuvwxyz
typeset -Z3 D1=4
typeset -Z3 L1=12
echo ${ABC:D1:L1}
```

→012（8 進数）が 10（10 進数）と解釈され，efghijklmn（10 文字）が STDOUT に出力される

また，次の例では，typeset コマンドでの属性変更でゼロ詰めを指定することで，変数 D1 に指定した 8 は 008 となり 8 進数と解釈されます。しかし，8 進数では 0～7 の数値しか指定できないため，エラーとなります。

```
ABC=abcdefghijklmnopqrstuvwxyz
typeset -Z3 D1=8
typeset -Z3 L1=12
echo ${ABC:D1:L1}
```

→8 進数に 8 が指定されていると解釈され，エラーとなる

offset や**length** に指定する変数が，再帰的な参照指定または循環参照指定となっている場合はエラーとなります。例を次に示します。

- 再帰的な参照指定の例

  ABC=abcdefghijklmnopqrstuvwxyz

  D1=D1

  echo ${ABC:D1}

- 循環参照指定の例

  ABC=abcdefghijklmnopqrstuvwxyz

  D1=D2

  D2=D1

  echo ${ABC:D1}

offset および **length** の実行例を次に示します。

- **offset**(5)を指定

```
ABC=abcdefghijklmn
echo ${ABC:5}
```

→「fghijklmn」が標準出力に出力される。

- **offset**(5)を指定し**length**(4)を指定

```
ABC=abcdefghijklmn
echo ${ABC:5:4}
```

→「fghi」が標準出力に出力される。

- **offset**(-1)を指定

```
ABC=abcdefghijklmn
echo ${ABC: -1}
```

→「n」が標準出力に出力される。

- **offset** を定義した変数名を指定

```
DEF=abcdefghijklmn
CNT=5
echo ${DEF:CNT}
```

→「fghijklmn」が標準出力に出力される。

- **offset** と **length** を定義した変数名を指定

```
DEF=abcdefghijklmn
CNT=5
LEN=4
echo ${DEF:CNT:LEN}
```

→「fghi」が標準出力に出力される。

- **offset** を定義した変数名を指定

```
DEF=abcdefghijklmn
CNT=-1
echo ${DEF:CNT}
```

→「n」が標準出力に出力される。

- 変数 xyz を明示的に指定し，変数に指定した値の 5 文字目から 3 文字分までを参照する場合，変数 xyz にマルチバイト文字が含まれている状態で，**offset** と **length** の値を変数 CNT と LEN に代入して参照

```
xyz=あいうえおかきくけこさしすせabcdefghそたち
CNT=4
LEN=3
echo ${xyz: CNT: LEN}
```

→「おかき」が標準出力に出力される。

- 変数 xyz を明示的に指定し，変数に指定した値の-17 文字目から 14 文字分までを参照する場合，変数 xyz にマルチバイト文字が含まれている状態で，**offset** と**length** の値を変数 CNT と LEN に代入して参照

```
xyz=あいうえおかきくけこさしすせabcdefghそたち
CNT=-17
LEN=14
echo ${xyz: CNT: LEN}
```

→「けこさしすせ abcdefgh」が標準出力に出力される。

(4)　変数に設定できる書式，属性

JP1/Advanced Shell では変数に対して書式および属性を設定できます。設定できる書式と属性を次の表に示します。

表 5-1　変数に対して設定できる書式

書式	意味
左詰め	変数に代入されている値を左詰めに変換します。
右詰め	変数に代入されている値を右詰めに変換します。
ゼロ詰め	変数に代入されている値を右詰めに変換します。さらに，値が数値の場合は，値の先頭までのスペースに 0 を挿入します。
小文字変換	変数に代入されている値のうち，大文字を小文字に変換します。
大文字変換	変数に代入されている値のうち，小文字を大文字に変換します。

表 5-2　変数に対して設定できる属性

属性	意味
整数型属性	変数に代入されている値を整数として扱います。 また，出力時の基数を定義できます。
読み込み専用属性	変数を読み込み専用とします。
エクスポート属性	変数をエクスポートします。

書式および属性は typeset コマンドで設定します。typeset コマンドの詳細については，「typeset コマンド（変数や関数の属性と値を明示的に宣言する）」を参照してください。

変数に対して書式を設定した例を次に示します。この例では，次の変数が定義されている場合を仮定しています（△はスペース）。例の各行の右側には，定義内容に対する説明を記載しています。

```
STRn=”△AbCdeFgHiJk△”
NUMn=”12345”
```

- ジョブ定義スクリプトの内容

```
typeset -L STR1          # STR1を左詰め書式に変更
echo $STR1               # STR1を出力
typeset -L5 STR2         # STR2を領域長5バイト，左詰め書式に変更
echo $STR2               # STR2を出力
typeset -R STR3          # STR3を右詰め書式に変更
echo $STR3               # STR3を出力
typeset -R4 NUM1         # NUM1を領域長4バイト，右詰め書式に変更
echo $NUM1               # NUM1を出力
typeset -Z9 STR4         # STR4を領域長9バイト，ゼロ詰め書式に変更
echo $STR4               # STR4を出力
typeset -Z9 NUM2         # NUM2を領域長9バイト，ゼロ詰め書式に変更
echo $NUM2               # NUM2を出力
typeset -l STR5          # STR5を小文字変換書式に変更
echo $STR5               # STR5を出力
typeset -u STR6          # STR6を大文字変換書式に変更
echo $STR6               # STR6を出力
typeset -i16 NUM3        # NUM3を16進数表記の整数型属性に変更
echo $NUM3               # NUM3を出力
```

- 実行したジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
AbCdeFgHiJk              ← STR1の出力結果
AbCde                    ← STR2の出力結果
AbCdeFgHiJk              ← STR3の出力結果
2345                     ← NUM1の出力結果
CdeFgHiJk                ← STR4の出力結果
000012345                ← NUM2の出力結果
abcdefghijk              ← STR5の出力結果
ABCDEFGHIJK              ← STR6の出力結果
16#3039                  ← NUM3の出力結果
```

5.1.3　配列

JP1/Advanced Shell では変数の 1 つとして，配列を作成および参照できます。

要素番号 0 から 65,535 までの，最大 65,536 個の要素を保持する 1 次元配列を作成できます。また，2 つの要素番号からなる配列で最大 65,536×64 個の要素を保持できる 2 次元配列を作成できます。要素を 1 つも指定しなかった場合，配列は設定されません。

(1)　配列の作成

配列の作成方法を次に示します。

- 複数の要素を一度に作成する場合（set コマンドを使用する方法）

  1 次元配列

```
set -A 配列名 値 値 …
set +A 配列名 値 値 …
```

  2 次元配列

```
set -D 配列名 { 値 値 … } { 値 値 … } …
set +D 配列名 { 値 値 … } { 値 値 … } …
```

1 次元配列の場合，作成する配列名と配列に登録する要素は区切り文字として半角スペースを 1 つ以上指定します。

```
set △-A△配列名△値△値△…
```

△には半角スペースを 1 つ以上指定してください。

使用例（1 次元配列）

```
set -A abc 1 2 3
echo ${abc[1]}
```

→2 が標準出力に出力されます。

2 次元配列の場合，作成する配列名と配列要素を括る”{“や”}”，および配列に登録する要素は区切り文字として半角スペースを 1 つ以上指定します。

```
set△-D△配列名△{△値△値△…△}△{△値△値△…△}△…
```

△には半角スペースを 1 つ以上指定してください。

使用例（2 次元配列）

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[1][1]}
```

→5 が標準出力に出力されます。

この方法では複数の要素を一度に作成できます。set -A コマンドおよび set -D コマンドについては，「9.3　シェル標準コマンド」の「set コマンド（シェルオプションを設定する，配列を作成する，または変数の値を表示する）」を参照してください。

- 1 つの要素を作成する場合

1 次元配列

```
配列名[要素番号]=値
```

2 次元配列

```
配列名[要素番号1][要素番号2]=値
```

使用例（1 次元配列）

```
abc[0]=1
abc[1]=2
abc[2]=3
echo ${abc[1]}
```

→2 が標準出力に出力されます。

使用例（2 次元配列）

```
abc[0][0]=1
abc[0][1]=2
```

```
abc[0][2]=3
echo ${abc[0][1]}
```

→2 が標準出力に出力されます。

この方法では要素を 1 つずつ作成します。複数の要素を作成する場合は，作成する要素分実行してください。また，要素番号 0（abc[0]または abc[0][0]）の配列は，変数と同じになります。

- 複数の要素を一度に作成する場合（set コマンドを使用しない方法）

1 次元配列

```
配列名=(値 値 …)
```

2 次元配列

```
配列名[]=({ 値 値 … } { 値 値 … } …)
```

1 次元配列の場合，作成する配列名の要素は区切り文字として半角スペースを 1 つ以上指定します。

```
配列名=(値△値△…)
```

△には半角スペースを 1 つ以上指定してください。

使用例（1 次元配列）

```
abc=(1 2 3)
echo ${abc[1]}
```

→2 が標準出力に出力されます。

2 次元配列の場合，作成する配列名の要素を括る”{“や”}”，および配列に登録する要素は区切り文字として半角スペースを 1 つ以上指定します。

```
配列名[]=({△値△値△…△}△{△値△値△…△}△…
```

△には半角スペースを 1 つ以上指定してください。

使用例（2 次元配列）

```
abc[]=({ 1 2 3 } { 4 5 6 })
echo ${abc[1][2]}
```

→6 が標準出力に出力されます。

この方法では複数の要素を一度に作成できます。作成方法の詳細については，「(2)　配列名=(値 値 …)による配列の作成」を参照してください。

(2)　配列名=(値 値 …)による配列の作成

「配列名=(値 値 …)」の形式で定義した 1 次元配列は，「set -A 配列名 値 値 …」の形式で登録されます。

「配列名[]=({ 値 値 … } { 値 値 … } …)」の形式で定義した 2 次元配列は，「set -D 配列名 { 値 値 … } { 値 値 … }」の形式で登録されます。

JOBLOG には「配列名=(値 値 …)」および「配列名[]=({ 値 値 … } { 値 値 … } …)」の形式でなく，「set コマンド」が実行されたように出力されます。

配列を「**配列名**=(**値 値** …)」の形式で作成した場合も，配列要素の管理方法はほかの配列と同じです。例えば，次の定義で作成した 1 次元配列は，「set -A ARRAY x1 x2 x3 x4 x5」で作成した配列と同じです。

ARRAY=(x1 x2 x3 x4 x5)と定義した場合の配列要素

```
ARRAY[0]=x1
ARRAY[1]=x2
ARRAY[2]=x3
ARRAY[3]=x4
ARRAY[4]=x5
```

n※	要素
0	x1
1	x2
2	x3
3	x4
4	x5

注※

n は配列要素番号を示します。

2 次元配列についても，配列を「**配列名[]**=({ **値 値** … } { **値 値** … } …)」の形式で作成した場合，配列要素の管理方法は set -D コマンドで作成した配列と同じです。例えば，次の定義で作成した 2 次元配列は，「set -D ARRAY { x1 x2 x3 } { x4 x5 x6 }」で作成した配列と同じです。

ARRAY[]=({ x1 x2 x3 } { x4 x5 x6 })と定義した場合の配列要素

```
ARRAY[0][0]=x1
ARRAY[0][1]=x2
ARRAY[0][2]=x3
ARRAY[1][0]=x4
ARRAY[1][1]=x5
ARRAY[1][2]=x6
```

n※1	m※2		
	0	1	2
0	x1	x2	x3
1	x4	x5	x6

注※1

n は配列要素番号を示します。

注※2

m は 2 次元配列要素番号を示します。

したがって，JOBLOG への出力，カバレージの採取，およびシェルオプション xtrace の出力は，set コマンドで配列定義をした場合と同じ出力結果になります。

なお，「**配列名**=()」と定義した場合は，名称が"**配列名**"で値が空文字列のシェル変数が作成されます。これは「**配列名**=」と定義した場合と同じです。

(a) 配列の生成例

次に示す変数を使用して，シェル変数を含む配列要素を生成する例を表に示します。

```
A=a
B=b
C=c
MA=’ a b c’ ※
MB=d
```

注※

「'」はスペースの有無を明確にするために便宜的に記載しています。実際の変数値には含まれません。

表 5-3 配列要素の生成例

配列定義	生成される配列の内容	生成される配列の数
(a b c)	[0]=a [1]=b [2]=c	3
($A $B $C)	[0]=a [1]=b [2]=c	3
(${A}${B}${C})	[0]=a [1]=b [2]=c	3
($A $B `echo 1`)	[0]=a [1]=b [2]=1	3
(AB $C)	[0]=ab [1]=c	2
(${A}xyz ${B}stu)	[0]=axyx [1]=bstu	2
($MA $MB)	[0]=a [1]=b [2]=c [3]=d	4
(MAMB)	[0]=a [1]=b [1]=cd	3

(b) 配列を使用した場合の JOBLOG の出力例

配列の定義例と，それによる JOBLOG の出力例を示します。

- 1 次元配列 SEQ1 に配列数 3 の配列要素(x1 x2 x3)を設定する。

  ```
  SEQ1=(x1 x2 x3)
  echo ${SEQ1[@]}
  ```

  → "x1 x2 x3"が標準出力に出力されます。

  配列 SEQ1 を使用した JOBLOG の出力例を次に示します。

  ```
  KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
  KNAX0724-I ジョブ識別子を割り当てました。Jobid=046187
  ```

```
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                    : 046187
   スプールジョブディレクトリパス : /var/opt/jp1as/spool/046187/
   実行日付                        : 2015/10/29
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          :
   ホスト名                        : host01
 [Automatic Job Management Systemから渡された環境変数]
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
14:46:13 046187 KNAX0091-I ADSH046187 ジョブが開始しました。
14:46:13 046187 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
14:46:13 046187 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
14:46:13 046187 KNAX6112-I コマンド（set，行番号=1）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:46:13 046187 KNAX6112-I コマンド（echo，行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:46:13 046187 KNAX0098-I ADSH046187 ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.010s

********   ジョブ定義スクリプトの内容     ********

***** /home/user001/SAMPLE_JOB6.ash *****
0001 : SEQ1=(x1 x2 x3)
0002 : echo ${SEQ1[@]}
0003 :

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容     ********
KNAX0098-I ADSH046187 ジョブが終了しました。rc=0 E-Time=0.001s C-Time=0.010s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”/var/opt/jp1as/spool/046187-ADSH046187/”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0

********   実行ジョブのSTDOUTファイルの内容     ********
x1 x2 x3>
```

- 2 次元配列 SEQ2 に配列数 3x2 の配列要素({ x1 x2 x3 } { x4 x5 x6 })を設定する。

```
SEQ2[]=({ x1 x2 x3 }{ x4 x5 x6 })
echo ${SEQ2[@]}
```

→ "x1 x2 x3 x4 x5 x6"が標準出力に出力されます。

配列 SEQ2 を使用した JOBLOG の出力例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちま
す。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=046189
----------------------------------------------------------------
 Advanced Shell 11-00
```

```
 [ジョブ情報]
   ジョブ識別子                    : 046189
   スプールジョブディレクトリパス : /var/opt/jp1as/spool/046189/
   実行日付                        : 2015/10/29
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          :
   ホスト名                        : host01
 [Automatic Job Management Systemから渡された環境変数]
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
15:01:31 046189 KNAX0091-I ADSH046189 ジョブが開始しました。
15:01:31 046189 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
15:01:31 046189 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
15:01:31 046189 KNAX6112-I コマンド（set，行番号=1）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:01:31 046189 KNAX6112-I コマンド（echo，行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:01:31 046189 KNAX0098-I ADSH046189 ジョブが終了しました。rc=0 E-Time=0.001s C-
Time=0.000s

********   ジョブ定義スクリプトの内容    ********

***** /home/user001/SAMPLE_JOB7.ash *****
0001 : SEQ2[]=({ x1 x2 x3 } { x4 x5 x6 })
0002 : echo ${SEQ2[@]}

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容    ********
KNAX0098-I ADSH046189 ジョブが終了しました。rc=0 E-Time=0.001s C-Time=0.000s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”/var/opt/jp1as/spool/046189-ADSH046189/”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0

********   実行ジョブのSTDOUTファイルの内容    ********
x1 x2 x3 x4 x5 x6
```

- 1 次元配列 SEQ1 に，次のように配列要素を代入した変数名を定義し，配列数 3 の配列を設定する。

```
ARR1=x1
ARR2=x2
ARR3=x3
SEQ1=($ARR1 $ARR2 $ARR3)
echo ${SEQ1[@]}
```

→ "x1 x2 x3"が標準出力に出力されます。

配列 SEQ1 を使用した JOBLOG の出力例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちま
す。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=046191
----------------------------------------------------------------
 Advanced Shell 11-00
```

```
 [ジョブ情報]
   ジョブ識別子                    : 046191
   スプールジョブディレクトリパス : /var/opt/jp1as/spool/046191/
   実行日付                        : 2015/10/29
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          :
   ホスト名                        : host01
 [Automatic Job Management Systemから渡された環境変数]
-----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
15:10:08 046191 KNAX0091-I ADSH046191 ジョブが開始しました。
15:10:08 046191 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
15:10:08 046191 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
15:10:08 046191 KNAX6110-I コマンド（ARR1=x1，行番号=1）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:10:08 046191 KNAX6110-I コマンド（ARR2=x2，行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:10:08 046191 KNAX6110-I コマンド（ARR3=x3，行番号=3）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:10:08 046191 KNAX6112-I コマンド（set，行番号=4）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:10:08 046191 KNAX6112-I コマンド（echo，行番号=5）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:10:08 046191 KNAX0098-I ADSH046191 ジョブが終了しました。rc=0 E-Time=0.002s C-
Time=0.000s

********   ジョブ定義スクリプトの内容     ********

***** /home/user001/SAMPLE_JOB8.ash *****
0001 : ARR1=x1
0002 : ARR2=x2
0003 : ARR3=x3
0004 : SEQ1=($ARR1 $ARR2 $ARR3)
0005 : echo ${SEQ1[@]}

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容     ********
KNAX0098-I ADSH046191 ジョブが終了しました。rc=0 E-Time=0.002s C-Time=0.000s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”/var/opt/jp1as/spool/046191-ADSH046191/”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0

********   実行ジョブのSTDOUTファイルの内容     ********
x1 x2 x3
```

- 2 次元配列 SEQ2 に，次のように配列要素を代入した変数名を定義し，配列数 3x2 の配列を設定する。

```
ARR1=x1
ARR2=x2
ARR3=x3
ARR4=x4
ARR5=x5
ARR6=x6
```

```
SEQ2[]=({ $ARR1 $ARR2 $ARR3 } { $ARR4 $ARR5 $ARR6 })
echo ${SEQ2[@]}
```

→ "x1 x2 x3 x4 x5 x6"が標準出力に出力されます。

配列 SEQ2 を使用した JOBLOG の出力例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちま
す。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=046193
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                    : 046193
   スプールジョブディレクトリパス : /var/opt/jp1as/spool/046193/
   実行日付                        : 2015/10/29
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          :
   ホスト名                        : host01
 [Automatic Job Management Systemから渡された環境変数]
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
15:17:35 046193 KNAX0091-I ADSH046193 ジョブが開始しました。
15:17:35 046193 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
15:17:35 046193 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
15:17:35 046193 KNAX6110-I コマンド（ARR1=x1, 行番号=1）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6110-I コマンド（ARR2=x2, 行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6110-I コマンド（ARR3=x3, 行番号=3）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6110-I コマンド（ARR4=x4, 行番号=4）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6110-I コマンド（ARR5=x5, 行番号=5）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6110-I コマンド（ARR6=x6, 行番号=6）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6112-I コマンド（set, 行番号=7）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX6112-I コマンド（echo, 行番号=8）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
15:17:35 046193 KNAX0098-I ADSH046193 ジョブが終了しました。rc=0 E-Time=0.003s C-
Time=0.000s

********   ジョブ定義スクリプトの内容     ********

***** /home/user001/SAMPLE_JOB9.ash *****
0001 : ARR1=x1
0002 : ARR2=x2
0003 : ARR3=x3
0004 : ARR4=x4
0005 : ARR5=x5
0006 : ARR6=x6
0007 : SEQ2[]=({ $ARR1 $ARR2 $ARR3 } { $ARR4 $ARR5 $ARR6 })
0008 : echo ${SEQ2[@]}
```

```
***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容     ********
KNAX0098-I ADSH046193 ジョブが終了しました。rc=0 E-Time=0.003s C-Time=0.000s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/046193-ADSH046193/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0

********   実行ジョブのSTDOUTファイルの内容     ********
x1 x2 x3 x4 x5 x6
```

(c) 注意事項

1 行に記述できるバイト数は 8,192 バイトまでです。そのため，配列要素番号を拡張した配列を定義した場合，最大配列数を 1 行のコマンドラインにすべて記述するとエラーになります。8,192 バイトを超える場合は，継続行指定（'¥'）で 1 行に指定できるバイト数に区切ってください。

継続行指定（'¥'）を使った定義例を次に示します。

<set コマンドでの定義例>

```
set -A ARRAY x0¥
 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 … x1000 ¥
x1001 x1002 x1003 x1004 x1005 x1006 x1007 x1008 x1009 x1010 x1011 … x2000 ¥
      :
x65001 x65002 x65003 x65004 x65005 x65006 x65007 x65008 x65009 … x65535
```

<代入式での定義例>

```
ARRAY=( x0¥
 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 … x1000 ¥
x1001 x1002 x1003 x1004 x1005 x1006 x1007 x1008 x1009 x1010 x1011 … x2000 ¥
      :
x65001 x65002 x65003 x65004 x65005 x65006 x65007 x65008 x65009 … x65535)
```

(3) 配列の値の参照

配列の値の参照方法を次に示します。

- 1 次元配列の 1 要素の値を参照する場合

  ```
  ${配列名[要素番号]}
  ```

  使用例

  ```
  set -A abc 1 2 3
  echo ${abc[1]}
  ```

  →2 が標準出力に出力されます。

- 2 次元配列の 1 要素の値を参照する場合

```
${配列名[要素番号1][要素番号2]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[1][2]}
```

→6 が標準出力に出力されます。

2 次元配列も 1 次元配列と同様に，配列要素番号に@または*を指定した場合，配列に定義されている値の全要素を参照できます。1 次元配列と異なるのは，配列要素番号の指定が 2 つあるためそれぞれに別々の記号や配列番号を指定できることから，@や*が指定されても，配列要素すべての参照指定とはなりません。${array[n][m]}を指定した場合の参照範囲について次に示します。

表 5-4　${array[n][m]}を指定した場合の参照範囲

n	m				
	指定なし	[]	[m]	[@]	[*]
[]	array[0][0]の値	array[0][0]の値	array[0][m]の値	array すべて	array すべて
[n]	array[n][0]の値	array[n][0]の値	array[n][m]の値	n 行のすべて	n 行のすべて
[@]	array すべて	array すべて	m 列すべて	array すべて	array 全て
[*]	array すべて	array すべて	m 列すべて	array すべて	aray 全て

@と*が混在して指定されている場合，要素番号 1 に指定した記号を優先します。

array[@][*]と指定した場合 array[@][@]とみなし，array[*][@]と指定した場合は array[*][*]とみなします。

- 配列の要素の値を参照する場合
  - 1 次元配列の要素の値を参照する場合

    参照方法 1

    ```
    ${配列名[*]}
    ```

    使用例

    ```
    set -A abc 1 2 3
    echo ${abc[*]}
    ```

    →1 2 3 が標準出力に出力されます。

    参照方法 2

    ```
    ${配列名[@]}
    ```

    使用例

    ```
    set -A abc 1 2 3
    echo ${abc[@]}
    ```

    →1 2 3 が標準出力に出力されます。

参照方法 3

```
"${配列名[*]}"
```

注　参照方法 3 の場合，IFS シェル変数の値で区切られます。

使用例

```
set -A abc 1 2 3
IFS=:
echo "${abc[*]}"
```

→1:2:3 が標準出力に出力されます。

参照方法 4

```
"${配列名[@]}"
```

使用例

```
set -A abc 1 2 3
echo "${abc[@]}"
```

→1 2 3 が標準出力に出力されます。

- 2 次元配列の素の値を参照する場合

参照方法 1

```
${配列名[*][*]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[*][*]}
```

→1 2 3 4 5 6 が標準出力に出力されます。

参照方法 2

```
${配列名[@][@]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[@][@]}
```

→1 2 3 4 5 6 が標準出力に出力されます。

参照方法 3

```
${配列名[1][*]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[1][*]}
```

→4 5 6 が標準出力に出力されます。

参照方法 4

```
${配列名[1][@]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[1][@]}
```

→4 5 6 が標準出力に出力されます。

参照方法 5

```
${配列名[*][1]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[*][1]}
```

→2 5 が標準出力に出力されます。

参照方法 6

```
${配列名[@][1]}
```

使用例

```
set -D abc { 1 2 3 } { 4 5 6 }
echo ${abc[@][1]}
```

→2 5 が標準出力に出力されます。

これら以外の指定の組み合わせについては，「表 5-4　${array[n][m]}を指定した場合の参照範囲」を参照してください。

配列の値の参照例を次に示します。

ジョブ定義スクリプトの内容

```
set -A myArray a01 a02 a03     #myArrayを配列として定義する。
for myElement in ${myArray[*]} #myArrayの全要素をfor文のwordlistsに展開する。
do
  echo $myElement
done
```

標準出力への出力結果

```
a01
a02
a03
```

(4)　大量の配列要素を確保する場合のメモリ所要量について

2 次元配列では，最大 65,536×64 個の配列要素の生成ができます。

しかし，2 次元配列の生成を実行するプロセスに与えられている使用可能なメモリ量によっては，最大配列要素数を確保できない場合があります。

2 次元配列の生成に必要なメモリ量の見積もり式は次のとおりです。

```
（324＋x）　× y
x：平均配列要素データ長（8の倍数単位に切り上げて指定）
y：生成する最大配列要素数（1～65,536×64）
```

上記のメモリ所要量と，他の構文などで使用するメモリ所要量との合計が，プロセスで使用可能なメモリ量を超えていないかを確認してください。

5.1.4　関数

同じジョブ定義スクリプトファイル内や外部ファイルに関数を定義すると使用できます。関数を定義する場合の書式を次に示します。

書式 1

```
関数名() {
        command
        :
         （略）
}
```

書式 2

```
function 関数名 {
        command
        :
         （略）
}
```

関数名の命名規則は変数と同じです。また，シェル標準コマンド，シェル拡張コマンド，および次のコマンド名と同名の関数は定義できません。

- bg
- bind
- fc
- fg

変数の命名規則については，「(1)　変数の命名規則」を参照してください。

関数を実行するジョブ定義スクリプトと異なるファイルに定義した場合，関数を実行する前に.（ドット）コマンド，または#-adsh_script コマンドで関数を定義したジョブ定義スクリプトを呼び出す必要があります。

書式を次に示します。.（ドット）コマンドについては，「9.3　シェル標準コマンド」の「.コマンド（シェルスクリプトを実行する）」，#-adsh_script コマンドについては，「9.5　スクリプト拡張コマンド」の「#-adsh_script コマンド（実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルを呼び出す）」を参照してください。

```
. 関数を定義したファイル名
```

または

```
#-adsh_script 関数を定義したファイル名
```

ジョブ定義スクリプト内で定義した関数の名称が，同一ジョブ定義スクリプト内，.（ドット）コマンドで呼び出したジョブ定義スクリプト内，または#-adsh_script コマンドで呼び出したジョブ定義スクリプト内で定義したほかの関数と重複していた場合，関数を実行する位置の直前に定義された関数が実行されます。

関数を実行する場合の書式を次に示します。関数には引数を指定できます。指定した引数は関数内では位置パラメーター$1 以降に格納されます。

```
関数名 [args]
```

なお，関数内の位置パラメーター$0 には，書式に応じて次の内容が格納されます。

- 書式 1 で定義した関数の場合，シェルスクリプト名称が格納される。
- 書式 2 で定義した関数の場合，関数名が格納される。

関数の引数の指定有無と位置パラメーターの関係を次の表に示します。

表 5-5　関数の引数の指定有無と位置パラメーターの関係

関数呼び出し元の位置パラメーター	関数の引数	関数開始時の位置パラメーター	関数からの回復時の位置パラメーター
指定あり	指定あり	引数に指定された値	関数呼び出し元の位置パラメーターの値
指定あり	指定なし	値なし	関数呼び出し元の位置パラメーターの値
指定なし	指定あり	引数に指定された値	値なし（関数呼び出し元の位置パラメーターの値）
指定なし	指定なし	値なし	値なし（関数呼び出し元の位置パラメーターの値）

(1)　関数内ローカル変数

JP1/Advanced Shell では，関数内で typeset コマンドを使用して変数を定義すると，関数内で有効なローカル変数を定義できます。定義した変数は，関数が完了した時点で関数実行前の状態に回復されます。関数内ローカル変数の実行例を次に示します。

ジョブ定義スクリプトの内容

```
0001 : myfunc(){              # 関数myfuncの定義
0002 :   typeset -r var=abc   # 関数内ローカル変数にvarを読み込みで定義
0003 :   echo $var            # 変数varの値を出力
0004 :   readonly -p          # 読み込み専用属性の変数を出力
0005 :   return 0
0006 : }
0007 : typeset -i var=123     # 変数varを整数型で定義
0008 : typeset | grep var     # 変数varの属性を出力
0009 : myfunc                 # 関数myfuncを実行
0010 : echo $var              # 関数実行後の変数varの値を出力
0011 : readonly -p            # 読み込み専用属性の変数を出力
0012 : typeset | grep var     # 変数varの属性を出力
0013 : exit 0
0014 :
```

実行ジョブの STDOUT ファイルの内容

```
********   実行ジョブのSTDOUTファイルの内容     ********
typeset -i var              ←8行目の結果。変数varは整数型で定義済み
abc                         ←3行目の結果。変数varはabcに更新
readonly var=abc            ←4行目の結果。変数varは読み込み専用属性
readonly ADSH_DIR_BIN=/opt/jp1as/bin/
readonly ADSH_DIR_CMD=/opt/jp1as/cmd/
readonly ADSH_DIR_PARTS_EN=/opt/jp1as/parts/en/
readonly ADSH_DIR_PARTS_JA=/opt/jp1as/parts/ja/
123                         ←10行目の結果。関数完了後，変数varの値は回復
readonly ADSH_DIR_BIN=/opt/jp1as/bin/
readonly ADSH_DIR_CMD=/opt/jp1as/cmd/
readonly ADSH_DIR_PARTS_EN=/opt/jp1as/parts/en/
readonly ADSH_DIR_PARTS_JA=/opt/jp1as/parts/ja/
typeset -i var              ←12行目の結果。8行目の出力結果と同じ
```

(2)　トレースモード

関数のトレースモードを有効にすることで，関数に記述されたコマンドを実行すると同時にコマンドの内容を標準エラー出力に出力できます。

関数のトレースモードへの切り替えはシェル標準コマンドの typeset コマンドで実行します。typeset コマンドについては，「9.3　シェル標準コマンド」の「typeset コマンド（変数や関数の属性と値を明示的に宣言する）」を参照してください。

例として，typeset コマンドの指定内容と，標準エラー出力への出力結果を次に示します。

ジョブ定義スクリプトの内容

```
0001 : fn(){
0002 :   echo "--- `date`"
0003 : }
0004 : typeset -ft fn
0005 : fn
```

関数のトレースモードによる標準エラー出力への出力結果

```
+ date
+ echo --- 2015年 10月 29日 木曜日 15:41:00 JST
```

(3) 関数のオートロード機能

関数のオートロード機能を使用すると，実行するシェルスクリプト内で使用する関数だけが実行時に定義されます。オートロード機能を使用しない場合と比べて，共通化された関数が実行有無に関係なく一度に実行されることがなくなるため，バッチジョブの実行性能が向上します。

関数のオートロード機能を使用する手順は次のとおりです。

1. 関数を定義したファイル（関数定義ファイル）を作成し，関数名と同じファイル名で保存する。

   関数定義ファイル内に複数の関数を定義した場合，記述したすべての関数が定義されます。そのため，関数定義ファイルで定義した関数名がすでに定義されている関数と重複していた場合，関数の定義内容が上書きされるため，注意してください。

   なお，関数定義ファイル内にコマンドなど関数定義以外の内容を記述した場合は，外部スクリプトと同等に動作します。

2. ジョブ定義スクリプトに次のコマンドを指定して，関数のオートロード機能を有効にする。

   ```
   typeset -fu 関数名 [関数名2...]
   ```

   関数名には，関数定義ファイル名と同じ名前の関数を指定します。

   また，JP1/Advanced Shell では「typeset -fu」コマンドの別名として autoload を提供します。autoload の形式は次のとおりです。

   ```
   autoload 関数名 [関数名2...]
   ```

   typeset コマンドについては，「9.3　シェル標準コマンド」の「typeset コマンド（変数や関数の属性と値を明示的に宣言する）」を参照してください。

   autoload については，「5.1.5　コマンドの別名定義」を参照してください。

3. シェル変数 FPATH に，関数定義ファイルを格納したディレクトリを指定する。

   オートロード機能を有効にした関数が関数定義ファイルに定義されていない場合，シェル変数 FPATH に指定されたディレクトリから関数名と一致する関数定義ファイルの内容を読み込んで，記述されているすべての関数を定義します。シェル変数 FPATH は，ジョブ定義スクリプトおよび環境ファイルで指定できます。

(a) オートロード機能の使用例（typeset -fu コマンドを使用した場合）

typeset -fu コマンドを使用した関数のオートロード機能の使用例を次に示します。なお，JP1/Advanced Shell では typeset -fu コマンドのエイリアスとして autoload を提供しています。この例では関数 auto1，関数 auto2，関数 auto3 に対してオートロード機能を実行しています。

関数定義ファイル（/home/jp1as/autoload/auto1）の内容

```
0001 : function auto1 {                        # 関数auto1の定義
0002 :  echo ”start auto1 in FPATH file”
0003 :  return 11
0004 : }
```

関数定義ファイル（/home/jp1as/autoload/auto2）の内容

```
0001 : autoxx() {          # 関数autoxxの定義(auto2は定義しない)
0002 :   echo ”start autoxx in FPATH file”
0003 :   return 255
0004 : }
```

/home/jp1as/autoload ディレクトリの関数定義ファイル

auto1

auto2

（関数定義ファイル「auto3」は存在しない）

ジョブ定義スクリプト（/home/jp1as/test.ash）の内容

```
0001 : export FPATH=”/home/jp1as/autoload”     # FPATHの設定
0002 :
0003 : typeset -fu auto1 auto2  # 関数auto1,auto2にオートロード機能を適用
0004 : autoload auto3           # 関数auto3にオートロード機能を適用
0005 :
0006 : auto1                    # 関数auto1を実行
0007 : auto2                    # 関数auto2を実行
0008 : auto3                    # 関数auto3を実行
```

実行結果

```
********  ジョブコントローラのメッセージ出力  ********
14:40:16 152262 KNAX0091-I ADSH152262 ジョブが開始しました。
14:40:16 152262 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセス
の完了を待ちます。
14:40:16 152262 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
14:40:16 152262 KNAX6112-I コマンド（export，行番号=1）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:40:16 152262 KNAX6112-I コマンド（typeset，行番号=3）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:40:16 152262 KNAX6112-I コマンド（typeset，行番号=4）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:40:16 152262 KNAX6112-I コマンド（echo，行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
14:40:16 152262 KNAX6112-I コマンド（return，行番号=3）が正常終了しました。rc=11 E-
Time=0.000s C-Time=0.000s
14:40:16 152262 KNAX6046-E 関数（”auto2”）が関数定義ファイル（”/home/jp1as/autoload/
auto2”）内に定義されていません。filename=”/home/jp1as/test.ash” line=7
14:40:16 152262 KNAX6044-E FPATHシェル変数に指定されたディレクトリに関数（”auto3”）の定義
ファイルが見つかりません。filename=”/home/jp1as/test.ash” line=8
14:40:16 152262 KNAX0101-E ADSH152262 ジョブを実行中にエラーが発生しました。
14:40:16 152262 KNAX0098-I ADSH152262 ジョブが終了しました。rc=127 E-Time=0.003s C-
Time=0.000s
```

注※1

関数 auto1 の実行結果が正常終了であることを示します。

注※2

関数 auto2 は，FPATH に指定されたディレクトリに同名の関数定義ファイルが存在していますが，その関数定義ファイルに関数 auto2 の定義がないため，KNAX6046-E メッセージを出力しエラー終了します。

注※3

関数 auto3 は，FPATH に指定されたディレクトリに同名の関数定義ファイルが存在しないため，KNAX6044-E メッセージを出力しエラー終了します。

(b) 注意事項

- オートロード機能で関数定義ファイルを読み込んで定義された関数は，それ以降，同じファイル名の関数定義ファイルを読み込みません。ただし，unset コマンドで無効化したあとに関数を再実行した場合は，同名の関数定義ファイルを読み込みます。
- オートロード機能を有効にした関数であっても，すでに同一シェルスクリプト内，.（ドット）コマンドで呼び出したシェルスクリプト内，またはスクリプト拡張コマンド#-adsh_script で呼び出したシェルスクリプト内で定義済みであれば，関数定義ファイルを読み込みません。
- 同名の関数を異なる関数定義ファイルに定義した場合，あとで定義された関数が有効になります。関数定義ファイルに複数の関数を定義する場合は，関数名が重複していないか確認してください。

  関数「fn1」を異なる関数定義ファイルに定義した例と，その場合の出力結果を次に示します。

  関数定義ファイル（/home/jp1as/autoload/fn1）の内容

```
0001 : fn1(){                          # 関数fn1の定義1
0002 :   echo ”start fn1”
0003 :   return 1
0004 : }
```

  関数定義ファイル（/home/jp1as/autoload/fn2）の内容

```
0001 : fn2(){                          # 関数fn2の定義
0002 :   echo ”start fn2”
0003 :   return 2
0004 : }
0005 : fn1(){                          # 関数fn1の定義2
0006 :   echo ”start fn1 in fn2”
0007 :   return 21
0008 : }
```

  ジョブ定義スクリプト（/home/jp1as/test.ash）の内容

```
0001 : export FPATH=”/home/jp1as/autoload”  # FPATHの設定
0002 : typeset -fu fn1 fn2                  # 関数fn1,fn2のオートロード機能を有効にする
0003 : fn1                                  # 関数fn1を実行
0004 : fn2                                  # 関数fn2を実行
0005 : fn1                                  # 関数fn1を再度実行
```

  標準出力への出力結果

```
start fn1                          ←定義1のfn1が実行されます
start fn2
start fn1 in fn2                    ←定義2のfn1が実行されます
```

- 定義されていない関数に対して，オートロード機能を有効にした場合，関数定義ファイルが読み込まれるまで未定義状態となります。そのため，CUI デバッガの info functions コマンドで表示されません。

5.1.5　コマンドの別名定義

JP1/Advanced Shell ではコマンドを別名（エイリアス）定義できます。別名定義には alias コマンドを使用します。書式を次に示します。

書式

```
alias エイリアス名=値
```

値にスペースを含む文字列を指定する場合は，クォーテーションで囲む必要があります。

また，エイリアス名には組み込みコマンド名を指定して再定義できますが，予約語を別名として再定義することはできません。例を次に示します。

組み込みコマンドを再定義した場合

```
alias read="read STR"  # エイリアス名readに"read STR"を定義する
uname | read           # readは"read STR"として実行される
echo $STR              # 変数STRに設定されたunameの結果が出力される
```

予約語を別名に指定した場合

```
alias while="echo JP1/AS"  # 予約語whileに対してエイリアスを定義する
while                      # whileは予約語として解釈され，
                           # while文の書式不正でエラー終了する
```

別名の定義および別名の一覧出力は alias コマンドで実行します。また，定義した別名定義を無効にするには unalias コマンドで実行します。これらのコマンドの詳細については，「9.　ジョブ定義スクリプトのコマンドおよび制御文」の「alias コマンド（エイリアスを定義する）」または「unalias コマンド（エイリアス定義を無効にする）」を参照してください。

なお，JP1/Advanced Shell では次のエイリアスが定義されています。

エイリアス名	定義内容
autoload	typeset -fu
functions	typeset -f
integer	typeset -i
local	typeset

エイリアス名	定義内容
type	whence -v

5.1.6　メタキャラクタ

メタキャラクタとは，ジョブ定義スクリプト内で特別な意味を持つキャラクタのことです。次に示す文字をジョブ定義スクリプトに記述すると，ジョブコントローラはメタキャラクタと解釈します。

| & ; < > ( ) $ ` ' ¥ " ~ # * [ ] ?

メタキャラクタを一般的な文字として使用したい場合，メタキャラクタの無効化を行う必要があります。メタキャラクタの無効化の方法を次の表に示します。

表 5-6　メタキャラクタの無効化の方法

無効化の方法	意味
'*str*'	文字列*str* は' 以外のすべての文字をそのままの文字として処理します。
"*str*"	文字列*str* は$，¥，`以外の文字をそのままの文字として処理します。 ただし，¥の直後の文字が$，¥，`，"の場合，¥はメタキャラクタとして有効となります。文字列*str* にそのままの文字として¥を含める場合は，¥¥と記述してください。
¥*char*	文字*char* の特殊な意味を無効（エスケープ文字）にします。

ただし，メタキャラクタを無効化した内容を出力する場合は，echo コマンドおよびprint コマンドなど，出力する組み込みコマンドの仕様に従って処理されます。

ジョブ定義スクリプトの内容

```
echo 'JP1/AS¥n'         # 1.
echo "JP1/AS¥n"         # 2.
echo JP1/AS¥¥n          # 3.
echo 'JP1/AS¥¥n'        # 4.
echo "JP1/AS¥¥n"        # 5.
```

実行ジョブのSTDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
JP1/AS          ←1.の結果 echoコマンドが¥nを改行として出力

JP1/AS          ←2.の結果

JP1/AS          ←3.の結果

JP1/AS¥n        ←4.の結果
JP1/AS          ←5.の結果
```

4.の場合は，''（シングルクォーテーション）で囲まれているため，¥¥n がecho コマンドに渡ります。そのため，echo コマンドは¥¥を文字¥と解釈し，文字¥と文字n を出力します。

5.の場合は，""（ダブルクォーテーション）で囲まれているため，¥¥を文字¥と解釈し，¥n がecho コマンドに渡ります。そのため，echo コマンドは¥n を改行文字として出力します。

メタキャラクタを使用した機能について説明します。

(1)　位置パラメーター

ジョブ定義スクリプトを実行する場合，ジョブ定義スクリプトファイル名の後ろに実行時パラメーターを指定すると，ジョブ定義スクリプトに引数として渡すことができます。JP1/Advanced Shell はこの引数を「位置パラメーター」と呼ぶ特殊な変数に代入します。位置パラメーターは「$0」から「$9」の 10 個が用意されており，「$0」には実行したジョブ定義スクリプトファイル名が代入されます。$1 から$9 には指定した順に引数が代入されます。JP1/Advanced Shell の位置パラメーターおよび関連する特殊文字について次の表に示します。

表 5-7　JP1/Advanced Shell の位置パラメーターおよび関連する特殊文字

位置パラメーターおよび特殊文字		意味
位置パラメーター	$0	ジョブ定義スクリプトのファイル名です。
	$n	ジョブ定義スクリプトに指定した第 n 引数の値（n は 1～9 まで）です。
関連する特殊文字	$#	ジョブ定義スクリプトに指定した引数の数です。
	$*	ジョブ定義スクリプトに指定した引数すべてです。
	$@	ジョブ定義スクリプトに指定した引数すべてです。
	"$*"	ジョブ定義スクリプトに指定した引数すべてを一まとめにして扱います。 例："$1 $2 $3 … " IFS シェル変数を変更していた場合，IFS シェル変数の値で区切られます。
	"$@"	ジョブ定義スクリプトに指定した引数すべてを個別に扱います。 例："$1" "$2" … " IFS シェル変数を変更していた場合，スペースで区切られます。

ジョブ定義スクリプト内で位置パラメーターを変更するには，シェル標準コマンドの set コマンドを使用します。set コマンドについては，「9.3　シェル標準コマンド」の「set コマンド（シェルオプションを設定する，配列を作成する，または変数の値を表示する）」を参照してください。

(2)　文字列区切り

JP1/Advanced Shell は，IFS シェル変数に指定された文字を「文字列を区切る文字」として扱います。IFS シェル変数の初期値はスペース，タブ文字または改行文字であるため，スペースおよびタブ文字が文字列の区切りとして扱われます。複数のスペース，タブ文字が連続しても，1 つの区切り文字として扱われます。IFS シェル変数の詳細については，「5.5　シェル変数」を参照してください。

また，スペースやタブ文字を文字列として使用する場合は，スペース，タブ文字を含む文字列をクォーテーション（’および”）で囲む必要があります。

(3)　行継続

複数行にわたってコマンドを記述する場合の記述手順を次に示します。

1. 行の終端に¥を付けて改行します。
2. 行の終端をクォーテーションで囲まないで改行し，複数行の最初と最後だけクォーテーションで囲みます。

このように記述すると行が継続していると判断し，同一行として処理されます。

(4)　コメント

ジョブ定義スクリプト内にコメントを記述できます。行中に「#」を記述すると，「#」以降から行末までをコメントとして扱い，処理を行います。ただし，「#」の後ろに「-adsh」が記述されている場合は，次のように処理をします。

- 行の先頭に#-adsh を記述している場合
  スクリプト拡張コマンドとして扱います。
- 前の行から継続して#-adsh を記述している場合
  コメント行として扱います。例を次に示します。

  ```
  echo ABC ¥
  #-adsh
  ```

  ただし，前の行から継続していない，かつ行の先頭に記述していないときは，エラーになります。
- #-adsh 以外を記述している場合
  コメント行として扱います。

なお，スクリプト拡張コマンドの行にはコメントを記述できません。

(5)　ワイルドカード

条件に合致するファイル名やディレクトリ名の特定，任意の文字列との比較などに，ワイルドカードが使用できます。

JP1/Advanced Shell で使用できるワイルドカードを次の表に示します。

表 5-8　JP1/Advanced Shell で使用できるワイルドカード

ワイルドカード	意味
?	任意の 1 文字に合致します。ただし，ドットファイルのドット「.」は除きます。

ワイルドカード	意味
*	0 文字以上の文字列に合致します。ただし，ドットファイルのドット「.」は除きます。
[…]	[ ]に囲まれた文字列のどれか 1 文字に合致します。[ ]に囲まれた文字列の先頭に!を付加した場合，[ ]に囲まれていない文字に合致します。]を文字として指定する場合は，文字列の先頭に指定してください。 ハイフン（-）で区切ると，ハイフンで区切られたその間にある任意の文字（その 2 文字も含む）に合致します。ハイフンを文字として指定する場合は，文字列の先頭または末尾に指定してください。記述例を次の表に示します。
{str,…}	ブレース展開によって，コンマで区切られた文字列 str を展開します。ただし，次の場合は展開しません。 • braceexpand シェルオプションが無効の場合 • noglob シェルオプションが有効の場合

ワイルドカード[ ]の記述例を次の表に示します。

表 5-9　ワイルドカード[ ]の記述例

記述例	意味
[]a]	文字列]a と合致します。
[!abc]	文字列 abc 以外と合致します。
[0-9]	0 から 9 までのどれかに合致します。
[a-z]	英小文字に合致します。
[A-Z]	英大文字に合致します。
[0-9a-zA-Z]	英数字に含まれます。
[-abc]	文字列-abc と合致します。
[!-abc]	文字列-abc 以外と合致します。

(6)　置換

置換機能として次の 3 つの機能があります。

- 変数置換
  変数の状態，変数の値の文字列長（単位：バイト），変数が配列の場合の要素数，変数の値と指定されたパターンとのマッチングによって，展開する変数の値を置き換えます。
- コマンド置換
  コマンドの標準出力を変数の値として扱って処理します。
- ファイル名置換
  「*」や「?」などのワイルドカードを使用し，条件に合致するファイル名を展開します。ただし，シェルオプションの noglob が有効の場合は，ファイル名置換は行われません。シェルオプションの noglob については「5.6.1　set コマンドで設定できるシェルオプション」を参照してください。

(a) 変数置換

変数置換には，変数の状態によって実行される変数置換，変数の値の文字列長や配列の要素数への変数置換，パターンマッチングの結果によって実行される変数置換，部分文字列展開があります。

- 変数の状態によって実行される変数置換

  変数の状態によって実行される変数置換の書式を次の表に示します。variable には変数名，word には variable の状態によって展開される変数を指定します。また，使用例および結果の a は未定義の変数，b=NULL，c=1 とします。

表 5-10　変数の状態によって実行される変数置換の書式

書式	説明	使用例	結果
${variable:-word}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。 variable は定義されていて NULL または未定義の場合，word の展開結果を返します。variable の値は変更しません。	cnt=${a:-7}	cnt に 7 を代入
		cnt=${b:-8}	cnt に 8 を代入
		cnt=${c:-9}	cnt に c の値を代入
${variable-word}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。 variable は定義されていて NULL の場合，variable の値(NULL)を返します。variable が未定義のときは，word の展開結果を返します。variable の値は変更しません。	cnt=${a-7}	cnt に 7 を代入
		cnt=${b-8}	cnt に NULL を代入
		cnt=${c-9}	cnt に c の値を代入
${variable:=word}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。 variable は定義されていて NULL または未定義の場合，word の展開結果を variable に代入し，variable の値を返します。ただし，この書式は変数だけに使用でき，位置パラメーターには使用できません。	cnt=${a:=7}	a に 7 を代入し，cnt に a の値を代入
		cnt=${b:=8}	b に 8 を代入し，cnt に b の値を代入
		cnt=${c:=9}	cnt に c の値を代入
${variable=word}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。 variable は定義されていて NULL の場合，variable の値(NULL)を返します。 variable が未定義の場合，word の展開結果を variable に代入し，variable の値を返します。ただし，この書式は変数だけに使用でき，位置パラメーターには使用できません。	cnt=${a=7}	cnt に 7 を代入
		cnt=${b=8}	cnt に NULL を代入
		cnt=${c=9}	cnt に c の値を代入
${variable:?[word]}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。 word を指定し，かつ variable は定義されていて NULL または未定義の場合，word の展開結果を標準エラー出力に出力し，ジョブ定義スクリプトは終了します。	cnt=${a:?7}	展開結果を標準エラー出力に出力し，シェル終了

書式	説明	使用例	結果
${variable:?[word]}	word を省略し，かつ variable は定義されていて NULL または未定義の場合，未定義を表す KNAX6050-E メッセージを出力し，ジョブ定義スクリプトは終了します。 variable の値は変更しません。	cnt=${a:?}	メッセージを出力し，シェル終了
		cnt=${b:?8}	展開結果を標準エラー出力に出力し，シェル終了
		cnt=${c:?9}	cnt に c の値を代入
${variable?[word]}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。 variable は定義されていて NULL の場合，variable の値(NULL)を返します。 word を指定し，かつ variable が未定義の場合，word の展開結果を標準エラー出力に出力し，ジョブ定義スクリプトは終了します。 word を省略し，かつ variable が未定義の場合，未定義を表す KNAX6050-E メッセージを出力し，ジョブ定義スクリプトは終了します。 variable の値は変更しません。	cnt=${a?7}	展開結果を標準エラー出力に出力し，シェル終了
		cnt=${a?}	メッセージを出力し，シェル終了
		cnt=${b?8}	cnt に NULL を代入
		cnt=${c?9}	cnt に c の値を代入
${variable:+word}	variable を変数として定義しかつ値を代入している場合，word の展開結果を返します。それ以外の場合は，NULL を返します。 variable の値は変更しません。	cnt=${a:+7}	cnt に NULL を代入
		cnt=${b:+8}	cnt に NULL を代入
		cnt=${c:+9}	cnt に 9 を代入
${variable+word}	variable を変数として定義しかつ値を代入している場合または NULL の場合，word の展開結果を返します。それ以外の場合は，NULL を返します。 variable の値は変更しません。	cnt=${a+7}	cnt に NULL を代入
		cnt=${b+8}	cnt に 8 を代入
		cnt=${c+9}	cnt に 9 を代入

- 変数の値の文字列長や配列の要素数への変数置換

  変数の値の文字列長，および配列の要素数への変数置換の書式を次の表に示します。variable には変数名，array には配列名を指定します。

表 5-11　変数の値の文字列長および配列の要素数への変数置換の書式

書式	説明
${#variable}	variable が「*」または「@」の場合，位置パラメーターの番号に置換されます。それ以外の場合は，環境設定パラメーター VAR_SHELL_GETLENGTH の指定に応じて次のように置換されます。 • BYTE

書式	説明
${#variable}	variable に格納されている値の長さをバイト数に置換 • CHARACTER variable に格納されている値の長さを文字数に置換 環境設定パラメーター VAR_SHELL_GETLENGTH が指定されていない場合は，環境設定パラメーター VAR_SHELL_GETLENGTH に BYTE を指定した場合と同じ動作となります。
${#array[*]}	array で指定された配列の要素数に置換されます。
${#array[@]}	
${#array[n][*]}	array[n][*]で指定された配列の行の要素数に置換されます。
${#array[n][@]}	array[n][@]で指定された配列の行の要素数に置換されます。
${#array[*][m]}	array[*][m]で指定された配列の列の要素数に置換されます。
${#array[@][m]}	array[@][m]で指定された配列の列の要素数に置換されます。
${#array[*][*]}	array で指定された配列の要素数に置換されます。
${#array[@][@]}	
${?MAX:array}	array で指定された配列の最大行数(1 次元配列)と最大列数(2 次元配列)に置換される。

- ${#array[*]}書式の使用例

```
ARRAY=(a b c d e f g h i j k l m n o)
echo ${#ARRAY[*]}
```

→15

- ${#array[n][*]}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o })
echo ${#ARRAY[1][*]}
```

→5

- ${#array[*][m]}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o })
echo ${#ARRAY[*][1]}
```

→3

- ${#array[*][*]}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o })
echo ${#ARRAY[*][*]}
```

→15

- ${?MAX:array}書式の使用例

```
ARRAY[]=({ a1 a2 a3 a4 a5 } { b1 b2 b3 b4 b5 } { c1 c2 c3 c4 c5 } ¥
         { d1 d2 d3 d4 d5 } { e1 e2 e3 e4 e5 } { f1 f2 f3 f4 f5 } ¥
         { g1 g2 g3 g4 g5 } { h1 h2 h3 } { i1 i2 i3 i4 })
```

```
unset ARRAY[2][3]
ARRAY[2][5]=c6
echo ${?MAX:ARRAY}
```

→9 6

n※1	m※2					
	0	1	2	3	4	5
0	a1	a2	a3	a4	a5	-
1	b1	b2	b3	b4	b5	-
2	c1	c2	-	c4	c5	c6
3	d1	d2	d3	d4	d5	-
4	e1	e2	e3	e4	e5	-
5	f1	f2	f3	f4	f5	-
6	g1	g2	g3	g4	g5	-
7	h1	h2	h3	-	-	-
8	i1	i2	i3	-	-	-

注※1

1 次元配列要素番号を示します。

注※2

2 次元配列要素番号を示します。

- パターンマッチングの結果によって実行される変数置換

  パターンマッチングの結果によって実行される変数置換の書式を次の表に示します。variable には変数名，pattern には variable とパターンマッチングを行う文字列を指定します。pattern にはワイルドカードによる指定もできます。

表 5-12　パターンマッチングの結果によって実行される変数置換の書式

分類	書式	説明
前方一致	${variable#pattern}	pattern が変数の値の先頭と一致する場合，一致する部分で最も短い部分を削除した値に置換します。それ以外の場合は variable の値に置換します。
	${variable##pattern}	pattern が変数の値の先頭と一致する場合，一致する部分で最も長い部分を削除した値に置換します。それ以外の場合は variable の値に置換します。
後方一致	${variable%pattern}	pattern が変数の値の終端と一致する場合，一致する部分で最も短い部分を削除した値に置換します。それ以外の場合は variable の値に置換します。

分類	書式	説明
後方一致	${variable%%pattern}	pattern が変数の値の終端と一致する場合，一致する部分で最も長い部分を削除した値に置換します。それ以外の場合は variable の値に置換します。

変数の値を文字列指定（前方一致）で削除し出力する場合の実行例を次に示します。

ジョブ定義スクリプトの内容

```
abc=abcd1234xyz987abcd1234efg
echo ${abc#abcd}          # 1.
echo ${abc#a*2}           # 2.
echo ${abc##a*2}          # 3.
echo ${abc#*1234}         # 4.
echo ${abc##*1234}        # 5.
echo ${abc#1234}          # 6.
```

実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
1234xyz987abcd1234efg       ←1.の結果 先頭のabcdを削除
34xyz987abcd1234efg         ←2.の結果 a…2の文字列を削除(最短一致)
34efg                       ←3.の結果 a…2の文字列を削除(最長一致)
xyz987abcd1234efg           ←4.の結果 先頭…1234の文字列削除(最短一致)
efg                         ←5.の結果 先頭…1234の文字列削除(最長一致)
abcd1234xyz987abcd1234efg   ←6.の結果 先頭が一致しないためabcの値を出力
```

変数の値を文字列指定（後方一致）で削除し出力する場合の実行例を次に示します。

ジョブ定義スクリプトの内容

```
abc=abcd1234xyz987abcd1234
echo ${abc%1234}          # 1.
echo ${abc%d*4}           # 2.
echo ${abc%%d*4}          # 3.
echo ${abc%34*}           # 4.
echo ${abc%%34*}          # 5.
echo ${abc%abcd}          # 6.
```

実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
abcd1234xyz987abcd      ←1.の結果 最後尾の1234を削除
abcd1234xyz987abc       ←2.の結果 d…4の最後尾の文字列を削除(最短一致)
abc                     ←3.の結果 d…4の最後尾の文字列を削除(最長一致)
abcd1234xyz987abcd12    ←4.の結果 34以降の文字列を削除(最短一致)
abcd12                  ←5.の結果 34以降の文字列を削除(最長一致)
abcd1234xyz987abcd1234 ←6.の結果 最後尾が一致しない場合abcの値を出力
```

- 部分文字列展開

  部分文字列展開の結果によって実行される変数置換の書式を次の表に示します。

表 5-13　部分文字列展開の書式

書式	説明
${variable:offset}	variable の展開結果から文字を取り出します。取り出す文字の先頭位置は offset で指定します。
${variable:offset:length}	variable の展開結果から最大 length 文字を取り出します。取り出す文字の先頭位置は offset で指定します。
${array[*]:offset}	配列の${array[offset]}を先頭とする要素を取り出します。
${array[*]:offset:length}	配列の${array[offset]}を先頭とする要素を length 個取り出します。
${array[@]:offset}	配列の${array[offset]}を先頭とする要素を取り出します。
${array[@]:offset:length}	配列の${array[offset]}を先頭とする要素を length 個取り出します。
${array[*][m]:offset}	配列の${array[offset][m]}を先頭とする m 列の要素を取り出します。
${array[*][m]:offset:length}	配列の${array[offset][m]}を先頭とする m 列の要素を length 個取り出します。
${array[n][*]:offset}	配列の${array[n][offset]}を先頭とする行の要素を取り出します。
${array[n][*]:offset:length}	配列の${array[n][offset]}を先頭とする行の要素を length 個取り出します。
${array[@][m]:offset}	配列の${array[offset][m]}を先頭とする m 列の要素を取り出します。
${array[@][m]:offset:length}	配列の${array[offset][m]}を先頭とする m 列の要素を length 個取り出します。
${array[n][@]:offset}	配列の${array[n][offset]}を先頭とする行の要素を取り出します。
${array[n][@]:offset:length}	配列の${array[n][offset]}を先頭とする行の要素を length 個取り出します。
${array[*][*]:offset}	配列の${array[offset]}を先頭とする要素を取り出します。
${array[*][*]:offset:length}	配列の${array[offset]}を先頭とする要素を length 個取り出します。
${array[@][@]:offset}	配列の${array[offset]}を先頭とする要素を取り出します。
${array[@][@]:offset:length}	配列の${array[offset]}を先頭とする要素を length 個取り出します。

（凡例）

variable：変数名を指定します。

array：配列名を指定します。

offset：部分展開する文字列および配列要素の先頭位置を指定します。

length：展開する文字数および配列要素数を指定します。

n：2 次元配列の行を指定します。

m：2 次元配列の列を指定します。

- ${array[*]:offset}書式の使用例

```
ARRAY=(a b c d e f g h i j k l m n o)
echo ${ARRAY[*]:3}
```

→d e f g h i j k l m n o

- ${array[*]:offset:length}書式の使用例

```
ARRAY=(a b c d e f g h i j k l m n o)
echo ${ARRAY[*]:3:3}
```

→d e f

- ${array[*][m]:offset}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o } { p q r s t })
echo ${ARRAY[*][1]:1}
```

→g l q

- ${array[*][m]:offset:length}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o } { p q r s t })
echo ${ARRAY[*][1]:1:2}
```

→g l

- ${array[n][*]:offset}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o } { p q r s t })
echo ${ARRAY[1][*]:1}
```

→g h i j

- ${array[n][*]:offset:length}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o } { p q r s t })
echo ${ARRAY[1][*]:1:2}
```

→g h

- ${array[*][*]:offset}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o } { p q r s t })
echo ${ARRAY[*][*]:3}
```

→d e f g h i j k l m n o p q r s t

- ${array[*][*]:offset:length}書式の使用例

```
ARRAY[]=({ a b c d e } { f g h i j } { k l m n o } { p q r s t })
echo ${ARRAY[*][*]:3:3}
```

→d e f

(b) コマンド置換

コマンド置換の書式を次の表に示します。command には実行するコマンド名および引数を指定します。

Windows の場合，CMDSUB_PROCESS パラメーターに CURRENT を指定すると，コマンド置換は外部コマンドを除いてカレントプロセスで実行されます（デフォルト値は CURRENT です）。

表 5-14　コマンド置換の書式

書式名	書式	説明
$( )形式	$(command)	command の文字列に「¥」が含まれていても「¥」は意味を持ちません。
逆引用符形式	`command`	command の文字列に「¥」が含まれていると「¥」は意味を持ちます。 コマンド置換の中にコマンド置換を記述する場合，`command ¥`command¥``のように内側の逆引用符の直前に「¥」を記述してください。

逆引用符形式では command の文字列内の「¥」はメタキャラクタとして扱われるため，$( )形式と逆引用符形式では，command の文字列に「¥」が含まれている場合の実行結果が異なります。そのため，$( )形式の使用をお勧めします。

実行例および実行結果を次に示します。

- $( )形式の場合

  実行例

  ```
  VAL=$(echo '¥$x')
  ```

  実行結果（VAL に代入される値）

  ```
  ¥$x
  ```

- 逆引用符形式の場合

  実行例

  ```
  VAL=`echo '¥$x'`
  ```

  実行結果（VAL に代入される値）

  ```
  $x
  ```

次のときコマンド置換はデータの受け渡しのために一時ファイルを使用します。

- Windows の場合

  CMDSUB_PROCESS パラメーターに CURRENT を指定したとき（デフォルト値）。

- UNIX の場合

  COMPATIBLE_CMDSUB パラメーターと同時に，PIPE_CMD_LAST パラメーターに CURRENT を指定した環境で，パイプ（|）でつながれた 1 つのコマンド群だけをコマンド置換したとき。例を次に示します。

  例：

  - 次の処理の場合，cmd2 によって標準出力に出力されるデータは，一時ファイルに出力されてから，ジョブ定義スクリプト中に文字列として展開されます。

    ```
    `cmd1 | cmd2`
    ```

  - 次の処理の場合は，コマンド群が 2 つあるため，一時ファイルは作成されません。

```
`cmd1 | cmd2; cmd3 | cmd4`
```

一時ファイルは，環境設定パラメーター TEMP_FILE_DIR で指定された場所に作成します。

CMDSUB_PROCESS パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「7.3.11 CMDSUB_PROCESS パラメーター（コマンド置換の実行プロセスを定義する）【 Windows 限定】」を参照してください。

COMPATIBLE_CMDSUB パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「7.3.14 COMPATIBLE_CMDSUB パラメーター（コマンド置換の動作を定義する）【UNIX 限定】」を参照してください。

(c) ファイル名置換

ファイル名置換の書式を次の表に示します。pattern はパターンマッチングを行う文字列を指定します。pattern にはワイルドカードによる指定もできます。

表 5-15 複数パターンによるファイル名置換の書式

書式	意味	使用例	一致する文字列
?(pattern|pattern ...)	pattern に指定された文字列のうち，どれか 1 つと合致します。	?(h)	空文字列，h
*(pattern|pattern ...)	pattern に指定された文字列のうち，0 または 1 つ以上と合致します。	*(h)	空文字列，h，h，hh，hhh，…
+(pattern|pattern ...)	pattern に指定された文字列のうち，1 つ以上と合致します。	+(h)	h，hh，hhh，…
@(pattern|pattern ...)	pattern に指定された文字列のうち，1 つだけ合致します。	@(h)	h
!(pattern|pattern ...)	pattern に指定された文字列のうち，1 つを除くすべてと合致します。	!(h)	h を除く任意の文字列

1 つ以上のパターンを「|」で区切ることで，複数のパターンを同時に指定し，合致するファイル名を置換できます。ただし，パターンの区切り文字である「|」の前後にスペースを挿入しないでください。また，パターンの前後にスペースを挿入した場合，スペースもパターンの一部として扱われます。

使用例を次に示します。

ファイル名置換の使用例

ジョブ定義スクリプトの内容

```
ls -C                    # 1.カレントディレクトリに存在するファイルを表示
ls -C ?(*.sh|*.exe|*.dot) # 2.拡張子がsh,exec,dotのどれかのファイル名を表示
ls -C *(*.sh|*.exe)      # 3.拡張子がsh,exeのどちらかのファイル名を表示
ls -C +(*.jhs|*h)        # 4.拡張子がjhsまたは終端がhのファイル名を表示
ls -C @(*.c|*.jhs)       # 5.拡張子がcまたはjhsのファイル名を表示
ls -C !(*.c|*.jhs)       # 6.拡張子cまたはjhs以外のファイル名を表示
```

実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
a.jhs  a.sh  a.txt  func.c  ←1.の実行結果
a.sh                        ←2.の実行結果
a.sh                        ←3.の実行結果
a.jhs  a.sh                 ←4.の実行結果
a.jhs  func.c               ←5.の実行結果
a.sh  a.txt                 ←6.の実行結果
```

ファイル名の先頭文字がドット(.)の場合は，パターンとの合致対象から外されます。先頭文字がドットのファイル名をパターンとの合致対象に含める場合は，明示的にドットを指定する必要があります。

(7)　算術展開

算術展開とは，算術をした上でその結果に置き換えることです。算術展開の書式と実行例を次に示します。

```
$((算術式))
```

算術展開の実行例

ジョブ定義スクリプトの内容

```
x=100          # 1.xに代入されている値は100
x=$((x-1))     # 2.算術を実行し，その結果をxに代入する
echo $x        # 3.変数xの値を標準出力へ出力する
```

実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
99              ←xに代入されている値は99
```

(8)　入出力リダイレクト

ジョブ定義スクリプトでは，コマンド実行前に実行結果の出力先の変更やコマンド実行に必要な情報の入力先を変更できます。これを入出力リダイレクトといいます。JP1/Advanced Shell で使用できる入出力リダイレクトについて説明します。

(a)　リダイレクト

使用例（2 次元配列） JP1/Advanced Shell で使用できるリダイレクトを次の表に示します。リダイレクトは左から右に解釈されます。

表 5-16　JP1/Advanced Shell で使用できるリダイレクト

リダイレクト	意味
>file	file を標準出力として使用します。file が存在しない場合は，ファイルを作成します。すでに file が存在する場合は，上書きします。※1
< file	file を標準入力として使用します。※1

リダイレクト	意味
command_1 | command_2	パイプです。command_1 の標準出力を command_2 の標準入力とします。
>>file	file を標準出力として使用します。file が存在しない場合は，ファイルを作成します。すでに file が存在する場合は，追記します。※1
>|file	file を標準出力として使用します。file が存在しない場合は，ファイルを作成します。すでに file が存在する場合は，上書きします。※1
<>file	file を読み取りと書き込みのために標準入力としてオープンします。※1
<<label	ヒアドキュメントです。
n>file	n 番のファイルディスクリプタの出力先を file にします。※1
n<file	n 番のファイルディスクリプタは file から入力します。※1
>&n	標準出力を n 番ファイルディスクリプタにコピーします。
<&n	標準入力を n 番ファイルディスクリプタからコピーします。
>&-	標準出力をクローズします。
<&-	標準入力をクローズします。
|&※2	親プロセスからの入出力を伴うバックグラウンドプロセスを起動します。
>&p※3	コマンドの出力先をバックグラウンドプロセスにリダイレクトします。
<&p※3	コマンドへの入力をバックグラウンドプロセスにリダイレクトします。

注※1

file にシンボリックリンクを指定した場合はリンク先が入出力の対象となります。なお，出力のためのリダイレクト（>，>>，>|，<>）に対して，リンク先が存在しないシンボリックリンクを指定した場合，UNIX 版と Windows 版で動作が異なります。

UNIX 版	リンク先にファイルを作成します。
Windows 版	ファイルに対するシンボリックリンクの場合はリンク先にファイルを作成します。ディレクトリに対するシンボリックリンクの場合はファイルの作成に失敗します。

注※2

「|&」を使用して，親プロセスからの入出力を伴うバックグラウンドプロセスを複数起動しようとすると，KNAX6029-E メッセージが出力され，ジョブ定義スクリプトがエラー終了する場合があります。そのため，「|&」を使用する場合は，wait コマンドなどを使用し，バックグラウンドプロセスを複数同時に起動しないようにしてください。

注※3

UNIX 版では，バックグラウンドプロセスへの入力を異なるファイルディスクリプタに再割り当てすることで，複数のバックグラウンドプロセスを起動できます。Windows 版では，バックグラウンドプロセスへの入力を異なるファイルディスクリプタに再割り当てしても，2 つ以上バックグラウンドプロセスを起動できません。

(b) ファイルディスクリプタ

JP1/Advanced Shell のジョブコントローラの入出力種別ごとのファイルディスクリプタを次の表に示します。

表 5-17 JP1/Advanced Shell の入出力種別ごとのファイルディスクリプタ

入出力種別	ファイルディスクリプタ	備考
標準入力	0	−
標準出力	1	次の場合は，ジョブコントローラプロセス起動時の標準出力を継承します。 • 簡潔出力モードまたは最小出力モードで動作する場合 • 環境ファイルの OUTPUT_STDOUT パラメーターに PARENT を指定した場合（デフォルトの動作です） • adshexec コマンドの-s オプションに PARENT を指定した場合 それ以外の場合は，ジョブコントローラがスプールの STDOUT への出力（stdout）として使用しているため，このファイルディスクリプタで端末をオープンしていません。
標準エラー出力	2	次の場合は，ジョブコントローラプロセス起動時の標準エラー出力を継承します。 • 簡潔出力モードまたは最小出力モードで動作する場合 それ以外の場合は，ジョブコントローラがスプールの STDERR への出力（stderr）として使用しているため，このファイルディスクリプタで端末をオープンしていません。
上記以外	3～9	ユーザーがジョブ定義スクリプト内で使用できるファイルディスクリプタです。

（凡例）

−：特になし。

(c) ヒアドキュメント

ヒアドキュメントとは，標準入力をジョブ定義スクリプト内で生成することです。ヒアドキュメントに関する書式を次の表に示します。なお，ヒアドキュメントではデータの受け渡しに一時ファイルを使用します。一時ファイルは，環境設定パラメーター TEMP_FILE_DIR で指定されたディレクトリに作成します。

表 5-18 ヒアドキュメントに関する書式

書式	意味

`<< label`
`document`
`label`
	label に囲まれた document の部分が標準入力に渡されます。document の部分に記述された変数は置換され，スペース，タブ文字はそのまま標準入力に渡されます。

`<<- label`
`document`
`label`
	label に囲まれた document の部分が標準入力に渡されます。document の部分に記述された変数は置換され，スペースはそのまま標準入力に渡されます。しかし，各行の行頭のタブ文字は削除されます。

書式	意味
<< ¥label document label	label に囲まれた document の部分が標準入力に渡されます。document の部分に記述された変数は置換されません。
<< 'label' document label	

(d) パイプ

複数のコマンドを接続し，1 つのコマンドの標準出力を別のコマンドの標準入力として使用したい場合に，パイプでコマンドを接続します。パイプで接続されたコマンドのセットをパイプラインと呼びます。

パイプラインは左から右へ処理が実行されます。最後のコマンドをカレントプロセス※と別プロセスのどちらで実行するかは PIPE_CMD_LAST パラメーターで定義できます。デフォルトはカレントプロセスです。

注※

別プロセスで実行するコマンド（外部コマンド，子孫ジョブ，UNIX 互換コマンド，シェル運用コマンド，サブシェル，バックグラウンド指定のコマンド，バックグラウンドプロセス指定のコマンド）は除きます。

- Windows の場合

  パイプラインのコマンドはカレントプロセスで逐次的に実行して，コマンド間のデータの受け渡しに一時ファイルを使用します。また，一時ファイルは環境設定パラメーター TEMP_FILE_DIR で指定されたディレクトリに作成します。パイプラインのコマンドを別々のプロセスとして実行したい場合は，PIPE_CMD_LAST パラメーターで定義できます。

  入力側のコマンドが実行終了してプロセスが存在しない状態で出力側のコマンドがパイプへ出力すると，出力側のコマンドはエラー終了します。

- UNIX の場合

  パイプラインのコマンドは別々のプロセスとして実行されます。

  複数のコマンドをパイプで接続して実行する場合，入力側のコマンドが実行終了してプロセスが存在しない状態で，出力側のコマンドがパイプへ出力すると，出力側のコマンドが SIGPIPE を受信する場合があります。

  ジョブコントローラでは，コマンドが SIGPIPE をハンドリングしている場合を除き，KNAX6522-E メッセージを出力してエラー終了します。

PIPE_CMD_LAST パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「7.3.38 PIPE_CMD_LAST パラメーター（パイプの最終コマンドの実行プロセスを定義する）」を参照してください。

(9)　コマンドセパレータ

コマンドセパレータを使用すると，ジョブ定義スクリプトの 1 行に複数のコマンドを記述できます。コマンドセパレータに関する書式を次の表に示します。

表 5-19　コマンドセパレータに関する書式

書式	意味
command;	セミコロン（;）までをコマンドおよびコマンド引数として解釈します。
command_1△$_0$&&△$_0$command_2	AND 制御演算子です。左項のコマンドが終了コード 0 で終了すると，右項のコマンドを実行します。
command_1△$_0$||△$_0$command_2	OR 制御演算子です。左項のコマンドが終了コード 0 以外で終了すると，右項のコマンドを実行します。

また，コマンドセパレータは，グループ化したコマンドを 1 つのコマンド群として扱い，実行できます。

(10)　コマンドのグループ化

コマンドをグループ化すると，複数のコマンドをまとめて実行できます。また，グループ化したコマンド群をグループ化することもできます。コマンドのグループ化に関する書式を次の表に示します。

表 5-20　コマンドのグループ化に関する書式

書式	意味
(command_1;△$_0$command_2; ...)	別プロセスでグループ化されたコマンドを実行します。コマンド実行中に環境を変更しても変更内容はカレントシェルに引き継がれません。グループ化するコマンドはセミコロンまたは改行文字で区切ります。
(command_1（改行） command_2（改行） ...)	
{△$_1$command_1;△$_0$command_2; ...;}	カレントプロセスでグループ化されたコマンドを実行します。コマンド実行中に環境を変更すると変更内容がコマンド終了後も引き継がれます。グループ化するコマンドはセミコロンまたは改行文字で区切ります。 この書式の場合，「{」の後ろに 1 つ以上のスペースを挿入しかつ最後のコマンドにセミコロン（;）または改行文字を挿入してください。
{△$_1$command_1（改行） command_2（改行） ...（改行） }	

(11)　そのほかのメタキャラクタ

そのほかに，次の表に示すメタキャラクタが使用できます。

表 5-21　使用できるメタキャラクタ

メタキャラクタ	意味
～※1	シェル変数の HOME※2 に置き換えられます。
～任意の文字列	【UNIX 限定】

メタキャラクタ	意味
~任意の文字列	/または引数区切り文字までの文字列と一致するユーザー名が/etc/passwd ファイルに登録されているか確認します。 登録されている場合は，一致するユーザーのログインディレクトリに置き換えられます。 登録されていない場合は，そのままの文字列として解釈されます。
~+※1	シェル変数の PWD に置き換えられます。
~-※1	シェル変数の OLDPWD に置き換えられます。
&	ジョブ定義スクリプトや関数をバックグラウンドで実行します。

注※1

「~」，「~+」および「~-」は，クォーテーションで囲んだり，クォーテーションで囲まれた文字列やエスケープ文字（¥）の直前に記述されたりすると，対応するシェル変数に置き換わりません。このような場合はシェル変数を使用してください。

注※2

シェル変数の HOME は自動では設定されません。環境変数として定義してください。

5.1.7　別プロセスでの実行

ジョブ定義スクリプト内に次の書式が記述されていた場合，カレントプロセスとは異なるプロセスで実行します。別プロセスで変更した内容については，カレントプロセスには引き継がれません。

書式	詳細記載個所
command_1 | command_2	(d)　パイプ
$(command) , ` command `	(b)　コマンド置換
(command)	(10)　コマンドのグループ化
command &	(11)　そのほかのメタキャラクタ
command |&	(a)　リダイレクト

別プロセスで実行する使用例を次に示します。

パイプ（|）による別プロセスの実行

パイプ（|）による別プロセス実行は，PIPE_CMD_LAST パラメーターの指定によって動作が異なります。

（例）

```
hostname | read STR
```

・PIPE_CMD_LAST パラメーターに CURRENT を指定した場合

hostnameコマンドは別プロセスで実行されますが，readコマンドはカレントプロセスで実行されます。

・PIPE_CMD_LASTパラメーターにOTHERを指定した場合

hostnameコマンド，readコマンドともに別プロセスで実行されます。そのため，変数STRにはhostnameコマンドの結果が代入されません。

コマンド置換（$()や``）による別プロセスの実行

（例）

```
$(date '+%Y%m%d')  #dateコマンドの出力結果を名称とするコマンドを実行
`date '+%Y%m%d'`   #dateコマンドの出力結果を名称とするコマンドを実行
```

|&によるバックグラウンドプロセスの実行

（例）

```
echo abc |&        # 文字列abcをバックグラウンドプロセス実行で出力
sleep 1
read -p STR        # バックグラウンドプロセスで出力した内容を読み込む
echo $STR
```

コマンドのグループ化によるサブシェル実行

（例）

```
(TZ=GMT; export TZ; date) # 環境変数TZを一時的にGMTに変更して時刻を出力
```

&によるバックグラウンド実行

（例）

```
sleep 10 &
```

文字列の置換は別プロセス側で行います。そのため，例えば上記の書式で変数に代入した文字列をコマンドとして実行すると，ジョブ実行ログファイルのコマンド実行結果には置換前の変数名が出力されます。ただし，エイリアスはカレントプロセスで解決した上で実行するため，エイリアス解決後のコマンド名が出力されます。

ジョブ定義スクリプトの内容

```
ls="ls -lt"                        # 変数lsに"ls -lt"を代入
alias gt="grep test"               # エイリアス名gtに"grep test"を定義
$ls | gt                           # ls -lt | grep testを実行
```

実行ジョブのジョブ実行ログファイルの内容

```
<省略>
********  ジョブコントローラのメッセージ出力  ********
16:01:06 152285 KNAX0091-I ADSH152285 ジョブが開始しました。
16:01:06 152285 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
16:01:06 152285 KNAX7902-I ジョブコントローラは，"端末入力モード"で動作します。
16:01:06 152285 KNAX6110-I コマンド（ls=ls -lt，行番号=1）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
```

```
16:01:06 152285 KNAX6112-I コマンド（alias，行番号=2）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
16:01:06 152285 KNAX6116-I コマンド（$ls，行番号=3）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
16:01:06 152285 KNAX6116-I コマンド（/opt/jp1as/cmd/grep，行番号=3）が正常終了しました。
rc=0 E-Time=0.001s C-Time=0.000s
16:01:06 152285 KNAX0098-I ADSH152285 ジョブが終了しました。rc=0 E-Time=0.013s C-
Time=0.000s
```

上記の書式で指定したコマンドに対して#-adsh_rc_ignore コマンドの指定を有効にする場合，#-adsh_rc_ignore コマンドの引数に指定するコマンド名は，置換前の文字列のベース名を指定する必要があります。

次のコマンドは別プロセスで実行しないでください。

種類	コマンド
シェル運用コマンド	adshfile
	adshmsvcd【Windows 限定】
	adshmsvce【Windows 限定】
	adshmdctl【UNIX 限定】
シェル拡張コマンド	adshecho
	adshread
	adshjoberr
	adshcmdrc
スクリプト拡張コマンド	すべて

Windows 版と UNIX 版の差異

次のコマンドを別プロセス化する場合，別プロセス実行には Windows 版と UNIX 版で違いがあります。

- シェル標準コマンド
- スクリプト制御文
- シェル拡張コマンド
- スクリプト予約語コマンド
- 代入式
- 関数
- コマンド置換を含むコマンド(例：cmdA $(cmdB) (cmdA,cmdB は外部コマンドとする))
- 別プロセスで，次のケースでは，定義されたエイリアスが有効になります。
  定義されたエイリアスによる置換を実行したくない場合は，コマンド名をクォーテーション('または")で囲んでください。

ケース 1

次のすべてに該当するとき。

1. エイリアスの定義がされていないときに読み込んだスクリプトを別プロセスで実行する。
2. エイリアスを定義する。
3. 別プロセスで実行するコマンドが，定義されたエイリアスの置換対象である。

例:

```
fn(){
 (ls)            #3の条件
}
alias ls='ls -l' #2の条件
fn               #1の条件
```

ケース 2

次のすべてに該当するとき。

1. 別プロセスでエイリアスを定義する。
2. 1 と同一のプロセスで実行するコマンドが，定義されたエイリアスの置換対象である。

例:

```
(alias ls='ls -l' #1の条件
 ls)              #2の条件
```

UNIX 版と比較した Windows 版の別プロセス実行の違いは次のとおりです。

- ルートジョブが拡張出力モードで，別プロセスでエラーが発生した場合，エラーの情報は標準エラー出力に出力されます。また，エラー情報にファイル名と行番号は出力されません。
- 子孫ジョブを別プロセス実行した際に，子孫ジョブは SPOOLJOB_CHILDJOB パラメーターに DELETE を指定した場合と同様の動作をする場合があります。
- 別プロセスは環境ファイルの設定内容を使用します。そのため，環境変数 ADSH_ENV に相対パスを指定したとき，カレントディレクトリを移動した状態で別プロセスを起動すると，環境ファイルを読み込めずにエラー終了します。
- 関数を別プロセス実行した場合，それ以降の関数情報配列の関数呼び出し行番号配列は，関数を別プロセス実行した行番号になります。
- 別プロセスにファイルディスクリプタが引き継がれないで，クローズされます。例えば，別プロセスで実行するスクリプト内で，親プロセスがオープンしている最中のファイルディスクリプタに対して，別プロセスが再度オープンしないで入出力を行うとエラーになります。標準出力と標準エラー出力は再度オープンされた状態です。
- 別プロセスは最小出力モードで動作します。そのため，別プロセスは情報メッセージを表示せず，エラーメッセージの出力先も最小出力モードに従います。

- 別プロセスは SPOOLJOB_CREATE パラメーターに NO を指定した状態で動作します。SPOOLJOB_CREATE パラメーターに NO を指定した状態の動作については，「(a)　スプールジョブ作成抑止機能の使用有無を決定する」を参照してください。
- #-adsh_path_var コマンドで指定した変数は，別プロセスではパスを扱う変数として定義されません。別プロセスでもパスを扱う変数として定義したい場合は，PATH_CONV_VAR パラメーターと PATH_CONV_NOVAR パラメーターで指定してください。

重要

別プロセスで次のシェル標準コマンドを実行した場合，コマンドの実行結果が正常終了とエラー終了のどちらになるかは，カレントプロセスで実行した場合と異なります。異なる点は次のとおりです。

- let コマンド
  別プロセスで実行した場合，算術式を指定しないで実行すると，終了コード 1 で正常終了します。
- exit コマンド，return コマンド
  別プロセスで実行した場合，引数に数字以外を指定して実行すると，終了コード 1 で正常終了します。
- getopts コマンド
  別プロセスで実行した場合，オプションの終了を検出すると，終了コード 1 でエラー終了します。#-adsh_step_start コマンドの successRC 属性や，#-adsh_rc_ignore コマンドを使用して，エラー終了とならないようにしてください。
- read コマンド
  別プロセスで実行した場合，ファイルの終了（EOF）を検出すると，終了コード 1 でエラー終了します。#-adsh_step_start コマンドの successRC 属性や，#-adsh_rc_ignore コマンドを使用して，エラー終了とならないようにしてください。

UNIX 版では，別プロセスでの実行中に，そのプロセスが終了要求シグナルを受信すると，シグナルの種類によっては別プロセスでの実行を継続する場合があります。別プロセスで実行した内容を確実に強制終了したい場合は，そのプロセスに SIGTERM を送信してください。

5.1.8　パターンマッチング

JP1/Advanced Shell では，一部の標準コマンドやスクリプト制御文でパターンとの比較ができます。例えば，「${variable#pattern}」の書式の場合，変数 variable の値のうち，パターン pattern と一致する部分で最も短い部分を削除した値に置換します。使用例を次に示します。

使用例

```
$ var=abcd
$ echo ${var#a[b]}
cd
```

パターンと比較できるかどうかは機能ごとに決まっています。パターンの書式が不正な場合，パターンを通常の文字列として比較します。例えば，「[a*(b])」の場合，本来パターンとして特別な意味を持つ「[」，「*(」および「]」はすべて通常の文字として比較されます。使用例を次に示します。

使用例

```
$ var='[a*(b])cd'
$ echo ${var#[a*(b])}
cd
```

5.1.9　エスケープ文字

(1)　エスケープ文字の種類

echo コマンドと print コマンドでは，次の表に示す文字をエスケープ文字として解釈します。

エスケープ文字	意味	echo コマンド	print コマンド
¥a	アラート文字（ベル）	○	○
¥b	バックスペース文字	○	○
¥c	行末の改行を抑止する（¥c の後ろに指定した文字は出力されない）	○	○
¥f	フォームフィード文字（改ページ）	○	○
¥n	改行文字	○	○
¥r	復帰文字	○	○
¥t	タブ文字	○	○
¥v	垂直タブ文字	○	○
¥0nnn※1	1～3 桁の 8 進数で表された ASCII コードの文字（0～7）	○	○
¥xnn※2	1～2 桁の 16 進数で表された ASCII コードの文字（0～9，a～f，A～F）	○	×
¥¥	1 つのバックスラッシュ文字	○	○

（凡例）

○：指定できます。

×：指定できません。

注※1

指定した ASCII コード文字が 1 桁または 2 桁の場合，前に 0 を付けて 3 桁で指定しても，同じ意味として解釈されます。例えば，次の 3 つの指定は同じ意味として解釈され，アラート文字（ベル）が 3 回出力されます。

echo -e "¥07"

echo -e "¥007"

echo -e "¥0007"

注※2

環境設定パラメーター ESCAPE_SEQ_ECHO_HEX に YES を指定した場合だけ有効になります。ESCAPE_SEQ_ECHO_HEX パラメーターについては，「7.　環境ファイルで設定するパラメーター」の「ESCAPE_SEQ_ECHO_HEX パラメーター（16 進数表記の ASCII コード文字をエスケープ文字として解釈するかを定義する）」を参照してください。

指定した ASCII コード文字が 1 桁の場合，前に 0 を付けて 2 桁で指定しても，同じ意味として解釈されます。例えば，次の 2 つの指定は同じ意味として解釈され，改行文字が 2 回出力されます。

echo -e "¥xA"

echo -e "¥x0A"

(2)　echo コマンドに-e オプションと-E オプションを指定しなかった場合の解釈

echo コマンドに-e オプション（エスケープ文字を解釈する）も-E オプション（エスケープ文字を解釈しない）も指定しなかった場合，エスケープ文字の解釈は環境設定パラメーター ESCAPE_SEQ_ECHO_DEFAULT の定義に従います。ESCAPE_SEQ_ECHO_DEFAULT パラメーターについては，「7.　環境ファイルで設定するパラメーター」の「ESCAPE_SEQ_ECHO_DEFAULT パラメーター（エスケープ文字関連のオプション省略時の echo コマンドの動作を定義する）」を参照してください。

5.1.10　スクリプト拡張コマンドの指定

ジョブ定義スクリプトの行の先頭に「#-adsh」が記述されている場合，スクリプト拡張コマンドへの要求と見なされ，スクリプト拡張コマンドを実行します。

スクリプト拡張コマンドについては，「9.5　スクリプト拡張コマンド」を参照してください。

5.1.11　外部コマンドの指定

ジョブ定義スクリプトの組み込みコマンド以外のプログラムを外部コマンドと総称します。外部コマンドには，UNIX 互換コマンド，OS が提供するコマンドおよびユーザーが独自に作成したプログラムなどが

該当します。ジョブ定義スクリプト中にこれらをコマンド名として記述することで，外部コマンドを実行できます。

外部コマンドの指定方法を次に示します。

```
$ 外部コマンドのパス
```

(1) Windows での外部コマンドの実行

(a) 外部コマンドの拡張子の定義

Windows の場合，拡張子「.com」，「.exe」，「.cmd」，「.bat」の実行ファイルは，ジョブ定義スクリプトから実行できます。ただし，環境変数 PATHEXT にこれらの拡張子を登録しておくと，ジョブ定義スクリプトに拡張子の指定が不要になります。

ジョブ定義スクリプトから外部コマンドを実行する場合，環境変数 PATHEXT に登録されている拡張子の順に実行されます。例えば，環境変数 PATHEXT の値に「.COM;.EXE;」が指定され，シェル変数 PATH の示す場所に「ls.com」と「ls.exe」がある場合は，最初に「ls.com」が実行されます。

外部コマンドの検索例を次に示します。環境変数 PATHEXT の値が「.COM;.EXE;」である場合，次のように実行されます。

```
ls      ←拡張子がないため，「.com」「.exe」の順に拡張子を付与して検索し実行される。
ls.exe  ←拡張子があるため，「ls.exe」が実行される。
```

ただし，シンボリックリンクを使用して実行ファイルを実行する場合は，シンボリックリンクだけに拡張子を付与して検索をします。シンボリックリンクが指す実行ファイルには拡張子「.com」，「.exe」，「.cmd」，「.bat」のどれかを指定してください。

(b) 外部コマンド実行前の処理のスキップ

外部コマンドを実行する前に，外部コマンドのパス，およびその引数に対して次の処理が実行されます。これらの処理が不要な場合，command コマンドに-w オプションを指定して，外部コマンドを実行してください。

- "（ダブルクォーテーション）の前の「¥」を「¥¥」に変換する処理
- "（ダブルクォーテーション）の前に「¥」を付与する処理
- "（ダブルクォーテーション）で囲む処理（COMPATIBLE_CMD_EXEC パラメーターに V10 を指定した場合）

command コマンドの-w オプションについては，「9.3 シェル標準コマンド」の「command コマンド（コマンドを実行する）」を参照してください。

使用例

```
command -w .”¥¥prog.exe” ”ABC”
```

(c)　バッチファイルの実行時の制限

バッチファイルパス，および引数にスペース，&，(,)，[,]，{,}，^，＝，;，!，'，＋，,，`，～を含むバッチファイルを実行する場合，次の制限があります。

- バッチファイルは次のように実行してください。

```
<cmd.exeパス> /c <バッチファイルパス>［<引数1> <引数2> … <引数n>］
```

- バッチファイルパスに&，(,)，[,]，{,}，^，＝，;，!，'，＋，,，`，～を含む場合，バッチファイルパス中の記号を"'¥でエスケープしてください。
- スペースまたは&，(,)，[,]，{,}，^，＝，;，!，'，＋，,，`，～を含む引数は，スペースや記号を"'¥でエスケープしてください。
- 環境設定パラメーター COMPATIBLE_CMD_EXEC に V10 を指定した環境で実行する場合，引数は"で囲まれたものがバッチファイルに渡ります。また，最後の引数は先頭にだけ"が付きます。このため，バッチファイルから引数を参照するときは"を自動的に取り除く%~1, %~2 … %~n の形式で参照してください。

(2)　UNIX での外部コマンドの実行

実行権限が付与された実行形式のバイナリファイルのときは，ジョブ定義スクリプトから実行できます。

また，実行権限の付与されたテキストファイルは，ファイルの先頭に「#!**実行プログラムパス**」を記述することで，ジョブ定義スクリプトから実行できます。この場合，#!の指定に従って実行プログラムが実行されます。

(3)　コマンドの実行方法の優先順序

ジョブコントローラはジョブ定義スクリプトに指定されたファイルを実行する場合，次の順に条件判定してファイルを実行します。

【Windows 限定】

1. CHILDJOB_PGM パラメーターの条件を満たした場合，子孫ジョブとして実行します。
2. CHILDJOB_SHEBANG パラメーターの条件を満たした場合，子孫ジョブとして実行します。
3. CHILDJOB_SHEBANG パラメーターのデフォルト定義に合致した場合，子孫ジョブとして実行します。
4. CHILDJOB_EXT パラメーターの条件を満たした場合，子孫ジョブとして実行します。
5. 拡張子が exe，bat，cmd，または com のファイルの場合，コマンドとして実行します。

6. 上記条件のどれにも該当しない場合，コマンドの起動に失敗します。ただし，ジョブの実行は継続されます。

【UNIX 限定】

1. CHILDJOB_PGM パラメーターの条件を満たした場合，子孫ジョブとして実行します。
2. ファイルの実行権限がある場合，手順 3.以降の判定を実施します。実行権限がない場合，コマンドの起動に失敗しエラー終了します。ただし，ジョブの実行は継続します。
3. CHILDJOB_SHEBANG パラメーターの条件を満たした場合，子孫ジョブとして実行します。
4. CHILDJOB_SHEBANG パラメーターのデフォルト定義に合致した場合，子孫ジョブとして実行します。
5. CHILDJOB_EXT パラメーターの条件を満たした場合，子孫ジョブとして実行します。
6. ファイルがバイナリファイルの場合，コマンドとして実行します。
7. ファイルがテキストファイルの場合，スクリプトとして実行します。

   #!が指定されている場合は，#!に指定された実行プログラムで実行します。

   #!が指定されていない場合は，「/bin/sh」で実行します。

子孫ジョブ関連のパラメーターを同時に指定した場合の使用例を次に示します（test.sh は存在し，実行権限ありとします）。

例 1

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#!/bin/ksh
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
/bin/sh ./test.sh
```

→CHILDJOB_PGM パラメーターが適用され，test.sh は子孫ジョブとして実行します。

例 2

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#!/bin/ksh
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
./test.sh
```

→CHILDJOB_SHEBANG パラメーターが適用され，test.sh は子孫ジョブとして実行します。

例 3

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#!/bin/sh
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
./test.sh
```

→CHILDJOB_EXT パラメーターが適用され，test.sh は子孫ジョブとして実行します。

例 4

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#! /opt/jp1as/bin/adshexec
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
./test.sh
```

→CHILDJOB_SHEBANG パラメーターのデフォルト定義に合致するため，test.sh は子孫ジョブとして実行します。

例 5

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#!/bin/ksh
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
/bin/ksh ./test.sh
```

→CHILDJOB_EXT パラメーター，CHILDJOB_PGM パラメーター，CHILDJOB_SHEBANG パラメーター，CHILDJOB_SHEBANG パラメーターのデフォルト定義のどの定義にも該当しないため，test.sh は子孫ジョブとして実行されないで，/bin/ksh で実行します。

例 6

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /opt/jp1as/bin/adshexec
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#!/bin/ksh
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
/opt/jp1as/bin/adshexec ./test.sh
```

→CHILDJOB_PGM パラメーターが適用され，test.sh は子孫ジョブとして実行します。

例 7

■環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM ./test.sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

■test.sh の内容

```
#!/bin/ksh
echo JP1AS
```

■ジョブ定義スクリプトの内容

```
./test.sh
```

→条件判定の順序に従い，CHILDJOB_PGM パラメーターを判定します。CHILDJOB_PGM パラメーターの定義に一致するため，読み替えが実施されます。ただし，子孫ジョブとして実行するファイルが指定されていないため，実行エラーになります。

(4) 関数と同名の子孫ジョブまたは外部コマンドの優先順序

関数と同名の子孫ジョブ，または外部コマンドの優先順序を次に示します。

- メインスクリプトで関数を定義済み，または#-adsh_script，.（ドット）コマンドによって関数定義ファイルを読み込んでいる場合は，常に関数が優先されます。
- 関数にオートロード属性が付加されていて，かつシェル変数 FPATH に指定されたディレクトリ内に関数と同名の関数定義ファイルが存在する場合は，関数が優先されます。
- 関数にオートロード属性が付加されていないが，シェル変数 FPATH に指定されたディレクトリ内に関数と同名の関数定義ファイルが存在する場合は，次の順に判定します。
  1. 環境変数 PATH に指定されたディレクトリに関数と同名のジョブ定義スクリプト，または外部コマンドが存在する場合は，子孫ジョブまたは外部コマンドが実行されます。
  2. 環境変数 PATH に指定されたディレクトリに該当するジョブ定義スクリプト，または外部コマンドが存在しない場合は，関数が実行されます。
- 環境変数 PATH とシェル変数 FPATH に指定されたディレクトリのどちらにも該当するファイルが存在しない場合は，実行するコマンドが存在しないためエラー終了します。

5.1.12 UNIX 互換コマンドの指定

Windows のコマンドプロンプトおよび UNIX のシェルから UNIX 互換コマンドを入力できます。また，ジョブ定義スクリプト中で UNIX 互換コマンドを使用して Windows または UNIX で実行できます。UNIX 互換コマンドを使用すると，ファイルやディレクトリの表示，作成，検索などの機能が実行できます。

実行ファイル形式の UNIX 互換コマンドを使用するジョブ定義スクリプトを Windows や UNIX で共用するためには，シェル変数 ADSH_DIR_CMD を使用して UNIX 互換コマンドを定義してください。シェル変数 ADSH_DIR_CMD については「5.5.1 JP1/Advanced Shell が設定するシェル変数」を参照してください。

シェル変数 ADSH_DIR_CMD を使用して UNIX 互換コマンド（date コマンド）をジョブ定義スクリプトに定義する例を次に示します。

```
"${ADSH_DIR_CMD}date"
```

5.1.13 ジョブ定義スクリプト実行時のシェルと書式チェックの指定

(1) シェルの指定

UNIX の場合，ジョブ定義スクリプトファイルの 1 行目に，ジョブ定義スクリプトを実行するためのシェルを指定できます。指定方法を次に示します。

```
#!実行ファイルパス
```

この状態で次のコマンドを実行すると，ジョブ定義スクリプトの 1 行目に記述した実行ファイルのパスのシェルでジョブ定義スクリプトが実行されます。なお，実行ファイルのパスを省略した場合，/bin/sh で実行されます。

```
$ ジョブ定義スクリプトファイルのパス
```

また，次のように adshexec コマンドのパスを先頭に付記してコマンドを実行した場合，ジョブ定義スクリプトの 1 行目の記述に関係なく，adshexec コマンドのパスでジョブ定義スクリプトが実行されます。

```
$ adshexecコマンドのパス ジョブ定義スクリプトファイルのパス
```

(2) 字句の書式チェック

adshexec コマンドでジョブ定義スクリプトファイルを実行すると，ジョブ定義スクリプトファイルに記述された字句の書式チェックをしてから実行します。ただし，ジョブ定義スクリプトファイル内で.（ドット）コマンドで読み込んでいる外部ファイルについては，書式チェックをしません。

ジョブ定義スクリプトを実行しないで字句の書式チェックだけを実行する場合は，adshexec コマンドに-c オプションを指定して実行してください。

5.2　条件判定

ジョブ定義スクリプトは，制御文に記述された条件式の結果を基に，実行する処理を制御します。条件判定として，制御文と条件式について説明します。

5.2.1　制御文

JP1/Advanced Shell は，次に示す制御文を使用できます。詳細については，「9.6　スクリプト制御文」を参照してください。

- case
  文字列の内容に応じて，幾つかある処理のうち，1 つを実行します。
- for
  値を順次変化させながら，同じ処理を繰り返し実行します。
- if
  条件に合わせて分岐することで，各条件に沿った処理を実行します。
- until
  条件が成立するまで，同じ処理を繰り返し実行します。
- while
  条件が成立している間，同じ処理を繰り返し実行します。

制御文の次に示す個所に，コマンド置換，またはスクリプト拡張コマンドを除く任意のコマンドを指定できます。

コマンド置換を指定できる個所

- case 文の式およびパターン
- for 文の wordlist

コマンド置換，またはスクリプト拡張コマンドを除く任意のコマンドを指定できる個所

- if 文の条件 1（if の後ろの条件），および条件 2（elif の後ろの条件）
- until 文の条件
- while 文の条件

指定した任意のコマンドの終了コードは，ジョブおよびジョブステップの終了コードに反映されません。制御文の実行が完了した時点でのジョブおよびジョブステップの終了コードは制御文のブロック内で最後に実行されたコマンドの終了コードになります。

ただし，次の条件のどれかに該当する場合は，指定した任意のコマンドの終了コードは，ジョブおよびジョブステップの終了コードに反映されます。ジョブステップが完了した時点でのジョブおよびジョブステップの終了コードは，指定したコマンドの終了コードになります。

1. 指定したコマンドがコマンド置換ではなく，exit コマンドまたは return コマンドである場合
2. 指定したコマンドがコマンド置換ではなく，特殊組み込みコマンドであり，かつエラーとなった場合
3. 指定した 1.，2.以外のコマンドがエラーとなり，かつ制御文が onError 属性に stop を指定したジョブステップ内であった場合

JP1/AJS でジョブをエラーとしたい場合など，エラーとなったコマンドの終了コードをジョブの終了コードに反映したい場合は，onError 属性に stop を指定したジョブステップ内に制御文を記述してください。

if 文での例を次に示します。

例 1

```
if `cmdA true`    ←cmdAは存在しないコマンド名
then
  echo ”true”
else
  echo ”false”  ←else節が実行される
fi
# この時点でのジョブの終了コードは0となる
```

上記の例 1 で，if 文の条件 1 に指定したコマンド置換の cmdA が存在しないコマンド名の場合，条件が偽と見なされ else 節の「echo "false"」が実行されます。このとき，if 文が完了した時点でのジョブの終了コードは 0 になります。

例 2

```
#-adsh_step_start S1 -onError stop
if `cmdA true`       ←cmdAは存在しないコマンド名
then
  echo ”true”
else
  echo ”false”
fi
#-adsh_step_end     ←then節もelse節も実行しないでジョブステップ中断
# ジョブステップS1，およびこの時点でのジョブの終了コードは127となる
```

上記の例 2 で，if 文の条件 1 に指定したコマンド置換の cmdA が存在しないコマンド名の場合，if 文の then 節と else 節のどちらも実行しないでジョブステップの実行を中断します。このとき，ジョブステップが完了した時点でのジョブの終了コードは，指定したコマンドが存在しないことを示す 127 になります。ジョブステップの定義，および onError 属性については，「9.5　スクリプト拡張コマンド」の「#-adsh_step_start コマンド，#-adsh_step_error コマンド，#-adsh_step_end コマンド（ジョブステップを定義する）」を参照してください。

5.2.2　条件式

条件式では，数値比較，文字列比較，ファイル属性，論理演算子および三項演算子を使用します。

条件式の共通仕様を次に説明します。

- test コマンドまたは let コマンドを使用して条件評価を行います。

  test コマンドは代替書式である[]および[[]]を含みます。また，let コマンドは代替書式である(())を含みます。

- test コマンドを使用して条件評価する場合，変数と演算子の間にスペースを入れてください。[[]]を使用する場合は，「[[」の直後と「]]」の直前にスペースを入れてください。

  [[]]を使用して条件判定するときの使用例を次に示します。

```
if [[ $arg1 -eq $args ]]; then
    echo TRUE
fi
```

- let コマンドの引数に test コマンドの引数（-eq など）を指定した場合，let コマンドは変数と解釈して動作します。

(1)　数値比較

数値の比較に使用する演算子を次の表に示します。なお，数値 1，および数値 2 は-2147483648～2147483647 の範囲内で指定してください。範囲外の値を指定した場合，正しい結果を求めることができないため，注意してください。

表 5-22　test コマンドで数値を比較するときに使用する演算子

演算子を使った条件式	判定
数値 1 -eq 数値 2	数値 1 と数値 2 が等しい場合は真。
数値 1 -ne 数値 2	数値 1 と数値 2 が等しくない場合は真。
数値 1 -ge 数値 2	数値 1 が数値 2 以上の場合は真。
数値 1 -gt 数値 2	数値 1 が数値 2 より大きい場合は真。
数値 1 -le 数値 2	数値 1 が数値 2 以下の場合は真。
数値 1 -lt 数値 2	数値 1 が数値 2 より小さい場合は真。

表 5-23　let コマンドで数値を比較する時に使用する演算子

演算子を使った条件式	判定
数値 1 == 数値 2	数値 1 と数値 2 が等しい場合は真。
数値 1 != 数値 2	数値 1 と数値 2 が等しくない場合は真。
数値 1 >= 数値 2	数値 1 が数値 2 以上の場合は真。
数値 1 > 数値 2	数値 1 が数値 2 より大きい場合は真。
数値 1 < 数値 2	数値 1 が数値 2 より小さい場合は真。
数値 1 <= 数値 2	数値 1 が数値 2 以下の場合は真。

数値比較の使用例を次に示します。

```
a=1
b=2
if [[ $a -lt $b ]]; then
  echo TRUE
else
  echo FALSE
fi

while (( $a != $b )); do
  echo LOOP
  ((a+=1))
done
```

(2)　文字列比較

文字列の比較に使用する演算子を次の表に示します。文字列比較の演算子は test コマンド，[[ ]]，[ ]で使用できます。let コマンド，(( ))では使用できません。

表 5-24　文字列の比較に使用する演算子

演算子を使った条件式	判定
文字列	文字列の長さが 1 文字以上の場合は真。ただし，[[ ]]コマンドでは使用できません。
-n 文字列	文字列の長さが 1 文字以上の場合は真。 文字列の長さが 0 なら，KNAX6041-E メッセージを出力し，終了コード 2 でエラー終了します。ただし，[[ ]]コマンドで文字列を指定しなかった場合は書式不正となり，KNAX6041-E メッセージを出力し，終了コード 1 でエラー終了します。
-z 文字列	文字列の長さが 0 の場合は真。
-o 文字列	文字列が現在有効に設定されているシェルオプションの文字列と一致する場合は真。 シェルオプションの文字列については，「5.6.1　set コマンドで設定できるシェルオプション」を参照してください。
文字列 = pattern	文字列と pattern が一致する場合は真。
文字列 == pattern	文字列と pattern が一致する場合は真。
文字列 != pattern	文字列と pattern が不一致の場合は真。
文字列 1 < 文字列 2※	文字列 1 と文字列 2 を ASCII コード順に比較します。文字列 1 より文字列 2 が大きい場合は真。
文字列 1 > 文字列 2※	文字列 1 と文字列 2 を ASCII コード順に比較します。文字列 1 より文字列 2 が小さい場合は真。

注※

[[ ]]で使用できます。その他の書式では使用できません。

比較する文字列にはスペースなどが含まれる場合があるため，"（ダブルクォーテーション）で囲むことを推奨します。使用例を次に示します。

```
str1=”aaa”
str2=”bbb”
test ”$str1” == ”$str2”
[[ ”$str1” == ”$str2” ]]
```

比較する文字列には*，?，[…]のワイルドカードが指定できます。ただし，[[ ]]だけで使用できます。その他の書式ではワイルドカードを使用できません。また，ワイルドカードを使用した文字列を"（ダブルクォーテーション）で囲んだ場合，ワイルドカードが持つ意味が無効化されてしまうため，注意してください。使用例を次に示します。

```
str1=”adsh”
str2=”ads?”
str3=”ad*”
[[ ”$str1” == ”$str2” ]]    ←ワイルドカードが無効。文字列”ads?”の比較をする
[[ $str1 != $str3 ]]        ←ワイルドカードが有効
```

[[ ]]による文字列比較の使用例を次に示します。ワイルドカードは，*，?，[…]が指定できます。

```
if [[ abc == ab* ]]; then
    echo TRUE
fi
```

ワイルドカードについては，「(5)　ワイルドカード」を参照してください。

(3)　ファイル属性

ファイルの形式や権限などの属性を評価する場合に使用する演算子を次の表に示します。ファイル属性評価の演算子は test コマンド，[[ ]]，[ ]で使用できます。let コマンド，(( ))では使用できません。

表 5-25　ファイルの形式や権限などの属性を評価する演算子

演算子を使った条件式	判定
-a file	file が存在する場合は真。
-b file	file が存在し，ブロック型デバイスの場合は真。
-c file	file が存在し，キャラクタ型デバイスの場合は真。
-d file	file が存在し，ディレクトリの場合は真。
-e file	file が存在する場合は真。
-f file	file が存在し，通常ファイルの場合は真。
-g file	file が存在し，setgid ビットがセットされている場合は真。
-h file	file が存在し，シンボリックリンクの場合は真。
-k file	file が存在し，スティッキービットがセットされている場合は真。
-p file	file が存在し，パイプファイルの場合は真。

演算子を使った条件式	判定
-r file	Windows の場合，file が存在する場合は真。 UNIX の場合，file が存在し，カレントプロセスから読み込み可能なときは真。
-s file	次の条件をすべて満たす場合は真。 【Windows の場合】 • file が存在する • file がフォルダ以外である • ファイルサイズが 1 バイト以上である 【UNIX の場合】 • file が存在する • ファイルサイズが 1 バイト以上，またはディレクトリである -s は UNIX と Windows で真の条件が異なるため，ディレクトリまたはフォルダの存在チェックには-d を使用してください。
-t fd	端末をオープンしている fd の場合は真。
-u file	file が存在し，setuid ビットがセットされている場合は真。
-w file	Windows の場合，読み取り専用属性が設定されていないか，またはディレクトリのときは真。 UNIX の場合，file が存在し，カレントプロセスから書き込み可能なときは真。
-x file	【Windows の場合】 次のどれかに該当するときは真。 • file がシンボリックリンク以外の場合 ・拡張子が「.com」,「.exe」,「.cmd」または「.bat」である ・ディレクトリである ・環境ファイルの CHILDJOB_EXT パラメーターまたは CHILDJOB_SHEBANG パラメーター（デフォルト定義含む）で設定した条件に一致するファイルである※ • file がシンボリックリンクの場合 ・シンボリックリンク，およびリンク先ファイルの拡張子が「.com」,「.exe」,「.cmd」または「.bat」である ・ディレクトリへのシンボリックリンク，かつ参照先がディレクトリである。 【UNIX の場合】 file が存在し，カレントプロセスから実行可能なときは真。
-G file	file が存在し，file が属するグループが呼び出し元のプロセスの実効グループ ID と一致している場合は真。
-L file	file が存在し，シンボリックリンクの場合は真。
-O file	file が存在し，所有者がプロセスの有効ユーザー ID の場合は真。
-S file	file が存在し，ソケットの場合は真。
file1 -ef file2	file1 と file2 が存在し，file1 と file2 の実体が同じ（シンボリックリンク先が同じまたはハードリンク先が同じ）場合は真。
file1 -nt file2	file1 と file2 が存在し，file1 の最終修正日時が file2 よりも新しい場合は真。また，file1 が存在し，file2 が存在しない場合も真。

演算子を使った条件式	判定
file1 -ot file2	file1 と file2 が存在し，file1 の最終修正日時が file2 よりも古い場合は真。また，file2 が存在し，file1 が存在しない場合も真。
-H file	常に偽。

注※

CHILDJOB_SHEBANG パラメーターの詳細については，「7.3.7　CHILDJOB_SHEBANG パラメーター（子孫ジョブとして実行するジョブ定義スクリプトファイルの実行プログラムパスを定義する）」を参照してください。

CHILDJOB_EXT パラメーターの詳細については，「7.3.5　CHILDJOB_EXT パラメーター（子孫ジョブとして実行するジョブ定義スクリプトファイルの拡張子を定義する）」を参照してください。

ファイル属性の演算子を使用する際の注意事項を次に示します。

- file にシンボリックリンクを指定した場合，リンク先が評価の対象となります。ただし，次の演算子は評価の対象が異なります。
  - -h，-L はシンボリックリンクが評価の対象となります。
  - -x はシンボリックリンク，リンク先の両方が評価の対象となります【Windows 版】。
- 次の演算子は Windows 環境では，存在しないファイル種別やフラグについて判定されるため，常に偽となります【Windows 版】。

  -b，-c，-g，-k，-p，-u，-S
- 次の演算子は UNSUPPORT_TEST パラメーターを指定することで，メッセージを出力してエラーとしたり，正常にしたりすることもできます。パラメーターの詳細については，「7.3　環境設定パラメーター」の「7.3.49　UNSUPPORT_TEST パラメーター（サポートしていない条件式の実行時の動作を定義する）【Windows 限定】」を参照してください【Windows 版】。

  -G，-O
- 次の演算子は UNSUPPORT_TEST パラメーターを指定すると，パラメーターの指定に従い判定結果を返します。演算子として判定を行う場合は次の演算子に対して UNSUPPORT_TEST パラメーターを指定しないでください【Windows 版】。

  -h，-L，-ef
- 演算子-t の引数には 10 以上の値を指定しないでください。指定した場合，値は保証できません。

ファイル属性の使用例を次に示します。

```
FILE="$HOME/script/test.ash"
if [[ -a $FILE ]];
then
  echo "$FILE exists."
else
  echo "$FILE does not exist."
fi
```

(4) 論理演算

論理演算で評価する場合に使用する演算子を次の表に示します。

表 5-26 論理演算で評価する場合に使用する演算子

演算子を使った条件式	判定	test コマンドでの使用可否	let コマンドでの使用可否
expr1 -a **expr2**	expr1 と expr2 の結果が両方とも真の場合，真。	○※	×
expr1 -o **expr2**	expr1 と expr2 の結果がどちらか一方でも真の場合，真。	○※	×
expr1 && **expr2**	expr1 と expr2 の結果が両方とも真の場合，真。	○	○
expr1 || **expr2**	expr1 と expr2 の結果がどちらか一方でも真の場合，真。	○	○
! **expr**	expr の結果が偽の場合，真。	○	○

（凡例）

○：使用できます。

×：使用できません。

注※

[[ ]]では使用できません。

論理演算の使用例を次に示します。

```
DIR="/tmp"
FILE="/tmp/test.ash"
a=2
b=4
if test -d $DIR -a -a $FILE
then
  echo "$DIR is directory and $FILE exists."
else
  echo "$DIR is not directory or $FILE does not exist."
fi

while ((a*0 || b-3)); do
  echo LOOP
  let b-=1
done
```

ただし，test コマンドで論理演算子「&&」および「||」を使用する場合は，次のように記述してください。

```
a=1
b=2
c=3
if test "$a" == 1 && test "$b" == 2; then
    echo "True"
else
```

```
    echo ”False”
fi
if test ”$a” != ”$b” || test ”$a” != ”$c”; then
    echo ”True”
else
    echo ”False”
fi
```

(5)　三項演算子

if-else の省略記法である三項演算子を使用できます。JP1/Advanced Shell で使用できる三項演算子を次の表に示します。三項演算子は let コマンド，(( ))で使用できます。test コマンド，[[ ]]，[ ]では使用できません。

表 5-27　JP1/Advanced Shell で使用できる三項演算子

演算子を使った条件式	判定
expr1?expr2: expr3	expr1 の結果が真であれば expr2 の結果，偽であれば expr3 の結果を返します。

三項演算子の使用例を次に示します。

```
VAR1=3
VAR2=2
ANSWER=0

((ANSWER=VAR1>VAR2?8+VAR1:8*VAR2))
echo $ANSWER
```

5.3　算術演算

ジョブ定義スクリプト内では，typeset コマンドの-i オプションで明示的に整数型と宣言しないかぎり，変数の値は数字であっても文字として扱われます。しかし，let コマンドまたは(())の中に算術演算を行う演算子を指定すると，変数に代入されている値を数値として扱い，算術演算が行えます。

JP1/Advanced Shell は，算術演算子，増分・減分演算子，ビットごとの論理演算子，代入演算子が使用できます。演算子の共通仕様を説明します。

- let コマンドで算術演算する場合，変数と演算子の間にスペースを入れないでください。スペースを入れた場合は，書式不正でエラー終了します。変数と演算子の間にスペースを入れたい場合は，let コマンドの省略形である(( ))を使用するか，またはクォーテーションで算術式を囲む必要があります。

  使用例

  ```
  let   NUM = 100 - 99      # クォーテーションで囲まれていないため，エラー終了する
  let ” NUM = 100 - 99 ”    # クォーテーションで囲まれているため，算術式は実行される
  (( NUM    = 100 - 99 ))   # 省略形(( ))を使用しているため，算術式は実行される
  ```

- 算術式には数値および数字が代入された変数を指定できます。数値は基数表記（基数#値）で指定できます。

  基数表記を省略した場合は 10 進数と解釈され，演算が実行されます。

- 数字以外の文字が代入されている変数を指定した場合は，エラー終了します。

5.3.1　算術演算子

算術演算子は，ジョブ定義スクリプト内で変数の値に対して四則演算を行うための演算子です。JP1/Advanced Shell で使用できる算術演算子を次の表に示します。

表 5-28　JP1/Advanced Shell で使用できる算術演算子

算術演算子	意味
-num	単項マイナス演算子です。num の値を負の値にします。
num1*num2	num1 と num2 を掛けた結果を返します。
num1/num2	num1 を num2 で割った結果を返します。
num1%num2	num1 を num2 で割ったときの余りを返します。
num1+num2	num1 と num2 を足した結果を返します。
num1-num2	num1 から num2 を引いた結果を返します。
num1**num2	num1 を num2 で累乗した結果を返します。 num2 に 0 より小さい値を指定した場合は KNAX6068-E メッセージを出力し，終了コード 2 でエラー終了します。

5.3.2　増分・減分演算子

増分・減分演算子は，同一変数への増分，減分処理を簡潔に表現するための演算子です。JP1/Advanced Shell で使用できる増分・減分演算子を次の表に示します。

表 5-29　JP1/Advanced Shell で使用できる増分・減分演算子

増分・減分演算子	意味
num++	num を参照したあと，num を 1 加算します。
num--	num を参照したあと，num を 1 減算します。
++num	num を 1 加算したあと，num の値を参照します。
--num	num を 1 減算したあと，num の値を参照します。

5.3.3　ビットごとの論理演算子

ビットごとの演算子は，変数の値に対してビット単位で論理演算処理をするための演算子です。JP1/Advanced Shell で使用できるビットごとの論理演算子を次の表に示します。

表 5-30　JP1/Advanced Shell で使用できるビットごとの論理演算子

ビットごとの論理演算子	意味
num1&num2	num1 と num2 をビット単位で論理積演算した結果を返します。
num1|num2	num1 と num2 をビット単位で論理和演算した結果を返します。
num1^num2	num1 と num2 をビット単位で排他的論理和演算した結果を返します。
num1<<num2	num1 を num2 ビットだけ左シフトした結果を返します。
num1>>num2	num1 を num2 ビットだけ右シフトした結果を返します。
~num	num をビット否定した結果です。1 の補数を返します。

5.3.4　代入演算子

代入演算子は，変数への値の代入を行うための演算子です。また，変数の四則演算，ビットごとの論理演算を行った結果を代入できます。JP1/Advanced Shell で使用できる代入演算子を次の表に示します。

表 5-31　JP1/Advanced Shell で使用できる代入演算子

代入演算子	意味
num1=num2	num1 に num2 を代入します。
num1*=num2	num1 と num2 を掛けた結果を num1 に代入します。

代入演算子	意味
num1/=num2	num1 を num2 で割ったときの結果を num1 に代入します。
num1%=num2	num1 を num2 で割ったときの余りを num1 に代入します。
num1+=num2	num1 と num2 を足した結果を num1 に代入します。
num1-=num2	num1 から num2 を引いた結果を num1 に代入します。
num1<<=num2	num1 を num2 ビットだけ左シフトした結果を num1 に代入します。
num1>>=num2	num1 を num2 ビットだけ右シフトした結果を num1 に代入します。
num1&=num2	num1 と num2 をビット単位で論理積演算した結果を num1 に代入します。
num1|=num2	num1 と num2 をビット単位で論理和演算した結果を num1 に代入します。
num1^=num2	num1 と num2 をビット単位で排他的論理和演算した結果を num1 に代入します。

5.4　条件判定と算術演算の優先順位

優先順位は，let コマンドで使用できる次の演算子を対象とします。

- 数値比較
- 論理演算子
- 三項演算子
- 算術演算子

条件式および算術演算の優先順位を次の表に示します。優先順位は項番 1 が最も高く，以降項番の順に低くなります。演算処理は優先順位が高い方から順に行われます。

表 5-32　演算子の優先順位

優先順位	演算子
1	-（単項マイナス演算子），!，++，--，～
2	**
3	*，/，%
4	+，-
5	<<，>>
6	<，<=，>，>=
7	==，!=
8	&
9	^
10	|
11	&&
12	||
13	?:（三項演算子）
14	=，+=，-=，*=，/=，%=，&=，^=，|=，<<=，>>=

例えば次の計算式では，「**」の方が「*」よりも優先順位が高いため，「3**3」が先に計算されます。その結果，a に代入される値は 54 になるため，「54」が標準出力に出力されます。

```
let a=2*3**3    ← 3の3乗に2を掛ける
echo $a         ← aとして「54」が標準出力に出力される
```

5.5　シェル変数

シェル変数とは，値を定義してからジョブ定義スクリプトが終了するまで引き継がれる変数のことです。JP1/Advanced Shell はシェル変数を設定および使用できます。シェル変数の値を参照・変更すると，ジョブ定義スクリプトを実行する環境をカスタマイズできます。

Windows の場合，システムのプロパティで設定した環境変数など，ジョブコントローラ起動時に設定されている環境変数の名前は，VAR_ENV_NAME_LOWERCASE パラメーターを次のように指定することで，大文字に変換されてシェル変数として取り込まれます。

VAR_ENV_NAME_LOWERCASE パラメーターの指定	ジョブコントローラ起動時に設定されている環境変数名の変換
ENABLE	環境変数の名称は HOME，PATH，SHELL だけ大文字に変換し，それ以外の環境変数名はそのままの名称でシェル変数として取り込みます。
DISABLE	環境変数の名称はすべて大文字に変換して，シェル変数として取り込みます。

また，ジョブコントローラ起動時に設定されている環境変数の名前が JP1/Advanced Shell で使用できる変数の命名規則に一致しない場合，シェル変数に取り込まれますが，そのシェル変数を使用することはできません。変数の命名規則については，「(1)　変数の命名規則」を参照してください。

5.5.1　JP1/Advanced Shell が設定するシェル変数

JP1/Advanced Shell が設定するシェル変数を次の表に示します。これらのシェル変数に対して，値の設定，属性の変更，設定の解除はしないでください。

表 5-33　JP1/Advanced Shell が設定するシェル変数

シェル変数名	設定される値
#	現在のジョブ定義スクリプトまたは関数に渡された引数の数が設定されます。
-	シェルに設定されているシェルオプションの省略形の文字列が設定されます。 ただし，省略形の文字がないシェルオプションは，この変数には設定されません。
?	直前に実行したコマンドの終了コードが設定されます。
$	シェルのプロセス ID として，次に示すプログラムのプロセス ID が設定されます。 【Windows 限定】 adshexecsub.exe またはadshesub.exe のプロセス ID 【UNIX 限定】 adshexec のプロセス ID
_	adshexec コマンド起動時に設定されている値が設定されます。値が設定されていない場合は，adshexec コマンド起動時のargv[0]の内容が設定されます。

シェル変数名	設定される値
_	また，子プロセスとして起動した外部コマンド，子孫ジョブの起動時には，argv[0]の内容が設定されます。
!	最後にバックグラウンドで実行したコマンドのプロセス ID が設定されます。
ADSH_DIR_BIN※1	JP1/Advanced Shell のプログラムフォルダ（bin）のパス名が設定されます。※2
ADSH_DIR_CMD※1	JP1/Advanced Shell の UNIX 互換コマンドのフォルダ（cmd）のパス名が設定されます。※3
ADSH_DIR_PARTS_JA※1	JP1/Advanced Shell のスクリプト開発部品フォルダ（parts）配下の日本語フォルダ（ja）のパス名が設定されます。
ADSH_DIR_PARTS_EN※1	JP1/Advanced Shell のスクリプト開発部品フォルダ（parts）配下の英語フォルダ（en）のパス名が設定されます。
ADSH_RC_EXTERNAL【Windows 限定】	最後に実行した外部コマンドの終了コードが設定されます。詳細は「5.5.4　外部コマンドの終了コードが設定されるシェル変数【Windows 限定】」を参照してください。
ADSH_RC_STEPLAST※1	過去に実行した最終ジョブステップの終了コードが設定されます。 ジョブステップが 1 つも実行されていない場合は，シェル変数が未定義です。
ADSH_RC_STEPMAX※1	過去に実行した全ジョブステップの終了コードの最大値が設定されます。 ジョブステップが 1 つも実行されていない場合は，シェル変数が未定義です。
ADSH_RC_STEPMIN※1	過去に実行した全ジョブステップの終了コードの最小値が設定されます。 ジョブステップが 1 つも実行されていない場合は，シェル変数が未定義です。
ADSH_STEPRC_*ジョブステップ名*※1	「*ジョブステップ名*」で示すジョブステップの終了コードが設定されます。「*ジョブステップ名*」で示すジョブステップが実行されていない場合は，シェル変数が未定義です。 重複するジョブステップ名が存在する場合，最後に実行されたジョブステップの終了コードが格納されます。
LINENO	実行中のジョブ定義スクリプトの現在行の行番号が設定されます。
OLDPWD	cd コマンドで設定された直前の作業ディレクトリが設定されます。
OPTARG	getopts コマンドで処理された最後のオプション引数の値が設定されます。
OPTIND	getopts コマンドで処理された最後のオプション引数のインデックスが設定されます。
PPID	Windows の場合は0 が設定されます。 UNIX の場合はシェルの親のプロセス番号が設定されます。
PWD	現在の作業ディレクトリが設定されます。
RANDOM	0 から32767（=0x7FFF）までの整数の乱数が設定されます。
REPLY	引数を指定しないread コマンドによって読み込まれた内容が設定されます。
SECONDS	シェルが起動してからの経過秒数が設定されます。
関数情報配列※1	adshexec コマンドで実行中の関数の情報が一次元配列で設定されます。次に示す配列があります。 • 呼び出し関数名称配列 • 関数呼び出し行番号配列 • 関数定義スクリプトファイル名配列

シェル変数名	設定される値
関数情報配列※1	関数情報配列については，「関数情報配列」を参照してください。

注※1

特別な意味を持つこれらのシェル変数を総称してシェル拡張変数と呼びます。

注※2

ジョブ定義スクリプトでadshfile コマンドを使用する場合の定義例を次に示します。

```
”${ADSH_DIR_BIN}adshfile” -s job -n keep -a del ${VAL01}
```

注※3

ジョブ定義スクリプトでexpr コマンドを使用する場合の定義例を次に示します。

```
num=`”${ADSH_DIR_CMD}expr” $NUM - 1`
```

シェル変数の使用例を次に示します。

スクリプト制御文のif で条件判定し，先行ジョブステップの実行結果によって後続ジョブステップの実行を制御する場合

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP01
    uap01
#-adsh_step_end
if [[ $ADSH_STEPRC_STEP01 -eq 0 ]]; then   ←STEP01が終了コード0の場合
    #-adsh_step_start STEP02               だけ，STEP02を実行
        uap02
    #-adsh_step_end
fi
```

ジョブ定義スクリプトをexit コマンドで終了するとき，ジョブステップ終了コードの最大値で終了する場合

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

#-adsh_step_start STEP01
  uap01
#-adsh_step_end

#-adsh_step_start STEP02 -run always
  uap02
#-adsh_step_end

#-adsh_step_start STEP03 -run always
  exit $ADSH_RC_STEPMAX        ←ジョブステップ終了コードの最大値を，
#-adsh_step_end                  ジョブの終了コードとする。
```

5.5.2　ユーザーが値を設定するシェル変数

JP1/Advanced Shell でユーザーが値を設定して使用できるシェル変数を次の表に示します。

表 5-34　JP1/Advanced Shell で使用できるシェル変数

シェル変数名	ユーザーが設定する値
CDPATH	cd コマンドで移動するディレクトリが作業ディレクトリの下に存在しない場合，検索する候補のパスを指定します。
ENV	• 【Windows，Linux 限定】 KSH_ENV_READ パラメーターがYES，または省略されていた場合，シェル起動時に読み込む.env ファイル名を指定します。 • 【AIX，HP-UX，Solaris 限定】 KSH_ENV_READ パラメーターがYES の場合，シェル起動時に読み込む.env ファイル名を指定します。
FPATH	関数定義ファイルの格納ディレクトリを指定します。オートロード機能が有効な関数が参照された場合，または実行する関数が定義されていない場合に，指定したディレクトリを検索します。関数名と同じ名称のファイルの内容を読み込み，現在の環境で定義して実行します。
HOME	ホームディレクトリを指定します。
IFS	Internal Field Separator の略です。指定された文字によって文字列の区切りを示します。また，IFS の先頭文字は「$*」を置換用の引数を区切る文字として使用します。初期値はスペース，タブ文字，改行文字です。
PATH	コマンドの検索パスを指定します。
PS4	シェルオプションxtrace が有効の場合に，各行の先頭に配置されるプロンプト文字列です。初期値は+です。
SHELL	シェル実行時に保持されるシェルのパス名を指定します。
TMPDIR	一時ファイルはすべて環境設定パラメーターTEMP_FILE_DIR に指定されたディレクトリに作成されるため，このシェル変数を変更しても無効となります。
ADSH_PARSER_LANG	JP1/Advanced Shell が動作する環境の環境変数LANG の値と異なるエンコーディングのJSON データを入力する場合，このシェル変数に値を設定しておくことで，adshparsejson コマンドを実行する間はエンコーディングを統一して動作させることができます。
ADSH_SPOOL_JOBNAME	スプールジョブディレクトリのリネームに使用するスプールジョブ名を指定します。 シェル変数が関数内ローカル変数の場合，指定した値はスプールジョブディレクトリのリネームに使用されません。

このほかに，PATH_CONV_VAR パラメーター，または#-adsh_path_var コマンドを使用した場合，Windows と UNIX 間でディレクトリのパスを変換するためのシェル変数を定義して使用できます。シェル変数の定義については，「5.8.5　パス名を扱うシェル変数を定義する」を参照してください。

5.5.3　関数情報配列

adshexec コマンドが実行する関数の情報は，一次元配列である関数情報配列に格納されます。

関数情報配列の使用有無は環境設定パラメーターVAR_SHELL_FUNCINFOで定義します。配列名も環境設定パラメーターVAR_SHELL_FUNCINFOの指定によって決まります。環境設定パラメーターVAR_SHELL_FUNCINFOについては，「7.3.54　VAR_SHELL_FUNCINFOパラメーター（関数情報配列の使用有無を選択する）」を参照してください。

関数情報配列の特徴は次のとおりです。

- 関数情報配列はジョブ定義スクリプトの実行時から終了まで存在します。ただし，.envファイル内，初期設定スクリプトファイル内で関数を実行した場合，関数情報配列は存在しません。
- 要素数の範囲は0～65,535です。
- 属性は文字列書式属性へ変更できます。属性を関数内ローカル変数へ変更することはできません。
- 関数情報配列は読み込み専用属性です。そのため，配列の値の参照はできますが，値の設定や配列の無効化はできません。

(1)　関数情報配列の種類

関数情報配列には次の表に示す種類があります。

表5-35　関数情報配列の種類と配列名

配列の種類	説明	配列名	
		VAR_SHELL_FUNCINFOにTYPE_Aを指定した場合	VAR_SHELL_FUNCINFOにTYPE_Bを指定した場合
呼び出し関数名称配列	呼び出しスタックにあるすべての関数名を格納する配列です。 要素番号0には現在実行中の関数名を格納します。最も下の要素には"main"を格納します。 シェル標準コマンドの.（ドット）コマンド，およびスクリプト拡張コマンドの#-adsh_scriptで外部スクリプトを呼び出した場合は"source"を格納します。	ADSH_FUNCNAME	FUNCNAME
関数呼び出し行番号配列※1	呼び出しスタックにあるすべての関数が呼び出されたスクリプトファイルの行番号を格納する配列です。 要素番号0には現在実行中の関数を呼び出した行番号を格納します。最も下の要素には"0"を格納します。 外部スクリプトの場合はシェル標準コマンドの.（ドット）コマンド，およびスクリプト拡張コマンドの#-adsh_scriptを実行した行番号を格納します。 属性は整数型へ変更することもできます。	ADSH_LINENO	BASH_LINENO
関数定義スクリプトファイル名配列※2	呼び出しスタックにある関数が定義されたスクリプトファイル名を格納する配列です。	ADSH_SOURCE	BASH_SOURCE

配列の種類	説明	配列名	
		VAR_SHELL_FUNCINFO に TYPE_A を指定した場合	VAR_SHELL_FUNCINFO に TYPE_B を指定した場合
関数定義スクリプトファイル名配列※2	要素番号 0 には現在実行中の関数を定義したスクリプトファイル名を格納します。最も下の要素にはジョブ定義スクリプト名を絶対パスで格納します。 外部スクリプトの場合は外部スクリプトファイルを絶対パスで格納します。	ADSH_SOURCE	BASH_SOURCE

注※1

シグナルや強制終了要求を受けたときに trap アクション内で関数を呼び出した場合，関数呼び出し行番号配列には，trap アクション内ではなく，trap アクションを呼び出した処理の行番号が格納されます。例えば，次の定義では行番号 4 で関数 fn1 を呼び出していますが，行番号 4 は trap アクション内のため，行番号 6 が配列に格納されます。

```
1  fn1(){
2    echo ${ADSH_LINENO[*]}
3  }
4  trap fn1 INT
5
6  kill -INT $$
7  pwd
```

注※2

adshexec コマンドを-r オプションで実行した場合，関数定義スクリプトファイル名配列のスクリプトファイル名には「-r CMDLINE」が格納されます。例を次に示します。

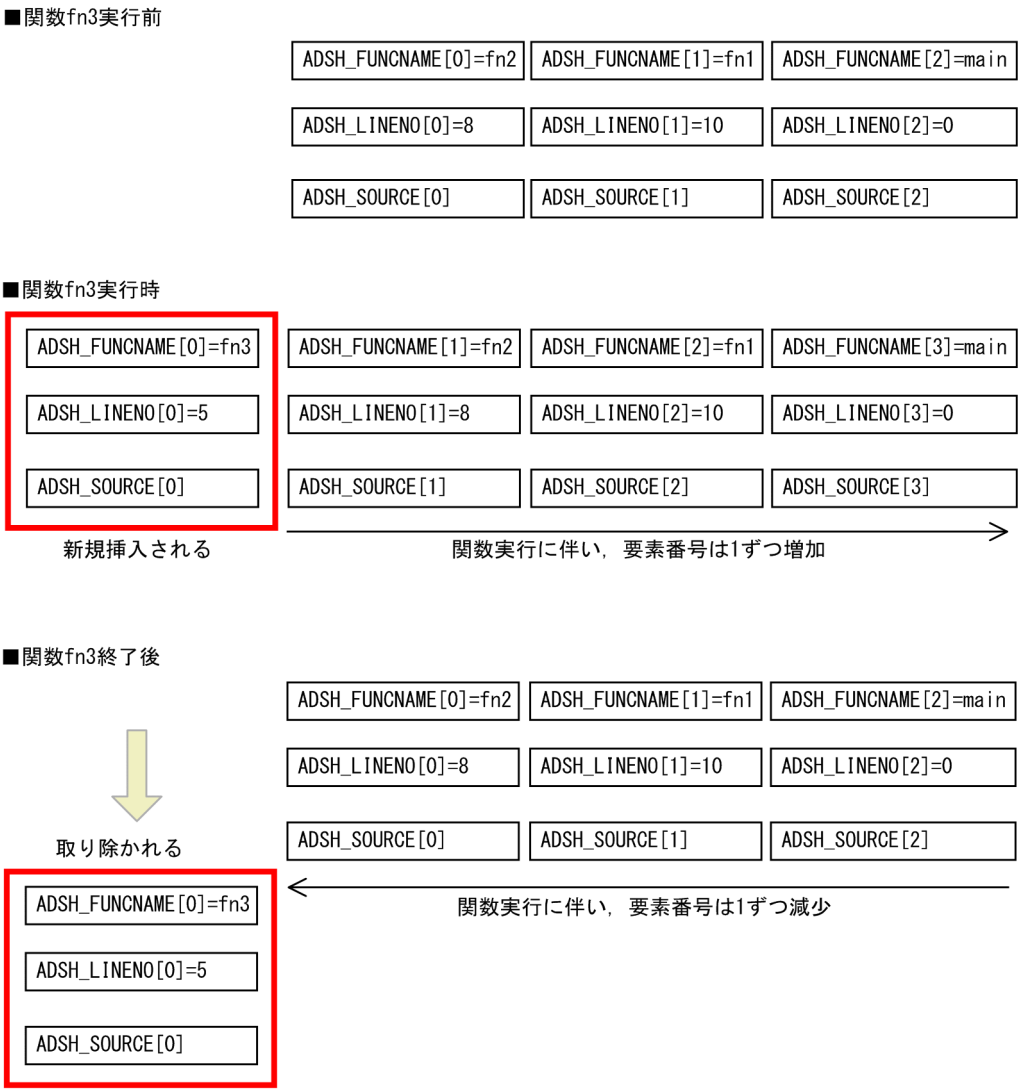
```
C:¥tmp>adshexec -m SIMPLE -r ”echo ${ADSH_SOURCE[*]}”
-r CMDLINE

C:¥tmp>
```

(2) 関数情報配列の構造

次のジョブ定義スクリプト（ファイル名：func.ash）を例に，配列の遷移について説明します。

```
1    fn3(){
2      echo ”JP1/AS”
3    }
4    fn2(){
5      fn3
6    }
7    fn1(){
8      fn2
9    }
10   fn1
```

このジョブ定義スクリプトを実行すると，配列の状態は関数 fn3 の実行によって次の図のように遷移します。



注：ADSH_SOURCE[0]～ADSH_SOURCE[3]には「func.ashの絶対パス」が格納されている。

(3)　関数情報配列に関する注意事項

- 関数情報配列はジョブ定義スクリプト内で変更できません。そのため，CUI や GUI でのデバッグ時は，値の参照（watch，print，および info variables コマンド）はできますが，更新（set コマンド）はできません。
- 関数情報配列をエクスポートすると，次のように動作します。
  - 配列ではなく，変数として要素 0 の内容が引き継がれます。ただし，関数情報配列はジョブ起動時に再設定されるため，子孫ジョブの場合は上書きされてしまいます。
  - 子孫ジョブの稼働実績情報には，上書き前のエクスポートされた値が出力されます。
- #-adsh_step_start コマンドの stepVar 属性には，関数情報配列の名称は指定できません。指定すると KNAX6312-E メッセージを出力してエラー終了します。

- 【Windows 版限定】関数定義スクリプトファイル名配列には「¥」を含むスクリプトファイルの絶対パスが格納されます。echo コマンド，print コマンドではシェル変数展開後の「¥」をエスケープ文字として扱います。そのため，echo コマンド，print コマンドで関数定義スクリプトファイル名配列の値を出力する場合は，次のように指定し実行してください。
  - echo コマンドで出力する場合は-E オプションを指定するか，または環境設定パラメーターESCAPE_SEQ_ECHO_DEFAULT に NO を指定してください。
  - print コマンドで出力する場合は-r オプションを指定してください。

5.5.4　外部コマンドの終了コードが設定されるシェル変数【Windows 限定】

JP1/Advanced Shell では，ジョブコントローラが実行する外部コマンドの終了コードを，シェル変数ADSH_RC_EXTERNAL に設定します。

(1)　シェル変数ADSH_RC_EXTERNAL

JP1/Advanced Shell のジョブコントローラでは，外部コマンドの終了コードを0～255 とすることを推奨しています。0～255 の範囲外の終了コードでリターンするバッチファイルやプログラムを実行し，その実行結果を終了コードで判定する場合は，シェル変数ADSH_RC_EXTERNAL で実行結果を取得してください。
シェル変数ADSH_RC_EXTERNAL の意味と有効範囲を次の表に示します。

表 5-36　シェル変数ADSH_RC_EXTERNAL の意味と有効範囲

シェル変数名	意味	有効範囲
ADSH_RC_EXTERNAL	最後に実行した外部コマンドの終了コードが設定されます。次のコマンドをジョブ定義スクリプト内で実行すると，ジョブコントローラはシェル変数ADSH_RC_EXTERNAL の値を更新します。 • 外部コマンド • UNIX 互換コマンド • シェル運用コマンド • 子孫ジョブ また，上記以外のコマンドでも，別プロセスでコマンドを実行（パイプ，コマンド置換，|&，&を使用）した場合は値が更新されます。初期値は0，属性は整数型，読み込み専用です。	-2147483648～2147483647

(2)　使用例

シェル変数ADSH_RC_EXTERNAL の使用例を次に示します。

外部コマンドがリターンする0～255 の範囲以外の値を取得する場合

```
"D:¥¥bin¥¥uap01.exe"     ←終了コード800を返すUAP
echo $?                  ←標準出力に32が出力される※
echo $ADSH_RC_EXTERNAL   ←標準出力に800が出力される
```

注※　800=0x320。シェル変数?は外部コマンドがリターンする値の下位８ビットを終了コードとして扱います。

外部コマンドがリターンする値が512の時，ジョブを終了コード2で終了する場合

```
”D:¥¥bin¥¥uap02.exe”
if [ $ADSH_RC_EXTERNAL -eq 512 ]
then
  exit 2
fi
```

(3)　注意事項

- シェル変数ADSH_RC_EXTERNALの値の変更，属性の変更，または無効化はできません。
- スクリプト拡張コマンド#-adsh_step_startのstepVar属性に，シェル変数ADSH_RC_EXTERNALを指定できません。指定するとKNAX6312-Eメッセージを出力してエラー終了します。
- ジョブコントローラは，ジョブ定義スクリプトのほか，.envファイル内，初期設定スクリプト内，または外部スクリプト内でコマンドを実行した場合も，シェル変数ADSH_RC_EXTERNALの値を更新します。
- ジョブコントローラは，ジョブ定義スクリプトを実行する直前に，シェル変数ADSH_RC_EXTERNALを0で初期化します。そのため，親プロセスで環境変数としてADSH_RC_EXTERNALを定義してもジョブコントローラに環境変数ADSH_RC_EXTERNALの値は引き継がれません。また，環境ファイル内で，exportパラメーターでADSH_RC_EXTERNALを定義しても値は引き継がれません。
- シェル変数ADSH_RC_EXTERNALは，外部コマンドが正常終了したか，エラー終了したかの判断に影響を及ぼしません。
- ジョブコントローラでは，パイプやコマンド置換で実行するコマンドの実行プロセスがカレントプロセスか別プロセスかが，次の環境設定パラメーターの指定によって異なります。そのため，同じジョブ定義スクリプトであっても，環境設定パラメーターの指定によっては，シェル変数ADSH_RC_EXTERNALを更新する契機が異なることがあります。
  - PIPE_CMD_LASTパラメーター
  - CMDSUB_PROCESSパラメーター
- コマンドを別プロセスで実行した場合，シェル変数ADSH_RC_EXTERNALに設定される値は，コマンドがリターンする値の下位８ビットになることがあります。別プロセスで実行したコマンドの結果取得に，シェル変数ADSH_RC_EXTERNALを使用しないでください。

5.6　シェルオプション

シェルオプションは，使用できる機能を制限したり，実行モードの切り替えをしたりできます。シェルオプションの設定方法は 2 つあります。

- ジョブ定義スクリプト内で set コマンドを実行し，設定します。
- adshexec コマンドのオプションとして設定します。

5.6.1　set コマンドで設定できるシェルオプション

set コマンドで設定できるシェルオプションを次の表に示します。set コマンドについては「9.3　シェル標準コマンド」の「set コマンド（シェルオプションを設定する，配列を作成する，または変数の値を表示する）」を参照してください。

表 5-37　set コマンドで設定できるシェルオプション

名称	設定方法	シェルオプションを設定した場合の意味	デフォルト値
allexport※1	-a -o allexport	変数をすべて自動的にエクスポートします。	無効
braceexpand	-o braceexpand	ブレース展開を有効にします。ブレース展開とは，ブレース({ })で囲んだ部分を複数の単語にする展開のことです。ブレースで囲まれた 1 つ以上のコンマで区切られた文字をブレースの前後の文字と結合し，1 つの変数として展開します。例えば，a{1,2,3}は a1，a2，a3 に展開されます。	有効
bgnice※2	-o bgnice	すべてのバックグラウンドジョブの優先順位を下げて実行します。	無効
noglob	-f -o noglob	ファイル名置換を禁止します。ファイル名置換については，「置換」を参照してください。 また，ブレース展開を無効にします。ブレース展開を有効にする場合は，noglob シェルオプションを無効にしてください。シェルオプションを無効にする方法については，「シェル標準コマンド」の「set コマンド（シェルオプションを設定する，配列を作成する，または変数の値を表示する）」を参照してください。	無効
nounset	-u -o nounset	置換対象の変数に値が設定されていない場合，ジョブはエラー終了し，シェルが終了します。	無効
verbose※3	-v -o verbose	ジョブ定義スクリプトから読み込んだ内容をコマンド実行前に標準エラー出力に出力します。	無効
xtrace	-x -o xtrace	シェル変数 PS4 の値を行の先頭に配置した上で，実行されたコマンドとその引数を標準エラー出力に出力します。ただし，[[ ]]コマンド，スクリプト拡張コマンドおよびその引数は出力しません。また，(( ))による算術演算は let コマンドに置き換えられて出力されます。	無効

注※1

Windows 版では，環境設定パラメーターで VAR_ENV_NAME_LOWERCASE DISABLE を指定した場合は，変数名に英小文字を含む変数をエクスポートできません。そのため，allexport シェルオプションを有効にしたあとに，英小文字を含むシェル変数を定義または値を更新すると，エラーメッセージを出力してバッチジョブを終了します。

注※2

Windows 版では，adshjava コマンドの優先度は変更されません。また，COMPATIBLE_CMD_EXEC パラメーターに V10 を指定した場合，UAP などの外部コマンドの優先度も変更されません。

Windows 版では，親プロセスの優先度が「通常」である場合，バックグラウンドジョブの優先度を「通常以下」にします。親プロセスの優先度が「通常」より高い場合，バックグラウンドジョブの優先度は「通常」となります。親プロセスの優先度が「通常」より低い場合，バックグラウンドジョブの優先度は親プロセスと同様になります。

注※3

verbose オプションを指定すると，ジョブステップを定義するコマンドの入力行出力先が変わります。

- #-adsh_step_start コマンドの場合

  実行した時点でジョブの STDERR からジョブステップの STDERR に切り替わるため，#-adsh_step_start 自身の入力行出力はジョブの STDERR に出力されます。

- #-adsh_step_end コマンドの場合

  実行した時点でジョブステップの STDERR からジョブの STDERR に切り替わるため，#-adsh_step_end 自身の入力行出力はジョブステップの STDERR に出力されます。

例を次に示します。

ジョブ定義スクリプト

```
set -o verbose        ←またはset -v

#-adsh_step_start S1
  cmdA
#-adsh_step_error
  cmdB
#-adsh_step_end
  cmdC
```

ジョブの STDERR

```
#-adsh_step_start S1
  cmdC
   :
```

ジョブステップ S1 の STDERR

```
  cmdA
#-adsh_step_error
  cmdB
#-adsh_step_end
```

注意事項

実行時パラメーターや JP1/Advanced Shell エディタでデバッグ実行をする場合，コマンドの実行結果は標準エラー出力へ出力されます。set コマンドで verbose オプションを指定すると，コマンドの実行結果のメッセージは次の行の内容を出力したあとで出力されます。例を次に示します。

ジョブ定義スクリプト

```
001:  set -o verbose
002:  echo "Line 002"
003:  echo "Line 003"
```

デバッグ実行時の出力例を次に示します。

```
KNAX7018-I ブレークポイント "1": ファイル名="test.ash" 行番号=1
KNAX7032-I ジョブ定義スクリプトファイル（"test.ash"）の中で実行を停止しました。
1: set -o verbose
現在位置: set
(adshdb) step       ←ジョブ定義スクリプトの001行目のsetコマンドを実行
echo "Line 002"     ←ジョブ定義スクリプトの002行目を読み込んだ内容を出力
KNAX6112-I コマンド（set, 行番号=1）が正常終了しました。rc=0 E-Time=0.000s C-
Time=0.000s       ←ジョブ定義スクリプトの001行目のsetコマンドの結果出力
KNAX7032-I ジョブ定義スクリプトファイル（"test.ash"）の中で実行を停止しました。
2: echo "Line 002"
現在位置: echo
(adshdb) step
Line 002
        :
```

5.6.2　adshexec コマンドで設定できるシェルオプション

adshexec コマンドで設定できるシェルオプションを次の表に示します。adshexec コマンドについては「8.3　シェル運用コマンド」の「adshexec コマンド（バッチジョブを実行する）」を参照してください。

表 5-38　adshexec コマンドで設定できるシェルオプション

名称	設定方法	シェルオプションを設定した場合の意味
noexec	-c	コマンドを読み取り，構文エラーがないかをチェックします。ただし，コマンドは実行しません。
xtrace	-x	シェルオプション xtrace を有効にした場合と同じ動作になります。詳細については，「実行したコマンドとその引数を出力する」を参照してください。

5.7　ジョブ情報の環境変数

ジョブ開始時やジョブステップ開始時に，ジョブ名，ジョブ識別子およびジョブステップ名を環境変数に設定し，ジョブ定義スクリプトファイルやユーザープログラムから参照できます。

- ADSH_JOB_NAME（ジョブ開始時にジョブ名が設定される）
- ADSH_JOBID（ジョブ開始時にジョブ識別子が設定される）
- ADSH_STEP_NAME（ジョブステップ開始時にジョブステップ名が設定される）

これらの環境変数については，「2.5　環境変数を設定する」を参照してください。

5.8　ジョブ，ジョブステップおよびコマンドを定義する

スクリプト拡張コマンドを使用してジョブ名を宣言したり，ジョブ，ジョブステップおよびコマンドの定義をしたりできます。スクリプト拡張コマンドについては，「9.5　スクリプト拡張コマンド」を参照してください。

5.8.1　ジョブ名を宣言する

#-adsh_job コマンドを使用して，ジョブ定義スクリプトのジョブ名を宣言します。

指定方法を次に示します。指定方法 1 または指定方法 2 のどちらかの方法で指定してください。

指定方法 1

```
1行目：#!任意文字列
2行目：△0#-adsh_job ジョブ名
```

指定方法 2

```
1行目：△0#-adsh_job ジョブ名
```

#-adsh_job コマンドを指定しない場合，デフォルトの属性値は次の表のようになります。

属性	省略または未宣言時の仮定値	例
ジョブ名	ADSH**ジョブ識別子**	ジョブ識別子が 000010 の場合：ADSH000010

5.8.2　ジョブの打ち切り条件を定義する

#-adsh_job_stop コマンドを使用して，ジョブステップ終了時に，ジョブを打ち切るかどうかを判断する条件を定義します。

(1)　判定するタイミング

ジョブステップ終了時に，終了コードがこの属性で定義されているかを毎回判定します。定義されていれば後続のジョブ定義スクリプトを実行しないでジョブが終了します。

(2)　有効範囲

指定した個所以降のジョブ定義スクリプトの実行で有効になります。また，先行ジョブ定義スクリプトでこのコマンドが指定されている場合，先行ジョブ定義スクリプトの指定はリセットされ，新たに指定した条件だけが有効になります。

指定例を次に示します。

```
01: #!/opt/jp1as/bin/adshexec
02: #-adsh_job JOB0001
03:
04: #-adsh_step_start STEP1
05: #-adsh_step_end
06:
07: #-adsh_job_stop 4:              ←この定義は行09～13の指定が有効範囲
08:
09: #-adsh_step_start STEP2
10: #-adsh_step_end
11:
12: #-adsh_step_start STEP3
13: #-adsh_step_end
14:
15: #-adsh_job_stop 8:16,24:32      ←この定義は行17～18の指定が有効範囲
16:
17: #-adsh_step_start STEP4
18: #-adsh_step_end
```

(3) ジョブの打ち切り条件定義の例

#-adsh_job_stop コマンドでジョブの打ち切り条件を定義した場合，次のようになります。

- #-adsh_job_stop コマンドで指定した終了コードで終了しても，ジョブステップ外のコマンドはジョブを中断しません。
- #-adsh_job_stop コマンドで指定した終了コードで終了した場合，ジョブステップはジョブを中断します。
- #-adsh_job_stop コマンドを実行してジョブを中断した場合，後続のジョブステップ外コマンドは実行しません。また，run 属性の指定に関係なく後続のジョブステップも実行しません。

実行例を次に示します。

```
#-adsh_job JOB_STOP
#-adsh_rc_ignore CBLRTN
#-adsh_job_stop 4                    ←rc=4でジョブを打ち切るよう指定

echo "Job start."
CBLRTN  004  #rc=4で成功するコマンド ←ジョブステップ外のコマンドがrc=4となるが，
                                       中断しない

#-adsh_step_start STEP01
echo "Step start."
CBLRTN  004  #rc=4で成功するコマンド
#-adsh_step_end                      ←ジョブステップがrc=4で終了し，中断する

#-adsh_step_start STEP03 -run always  ←run属性に関係なく後続ジョブステップを実行しない
  echo "command in step"
#-adsh_step_end

echo "Job end."                      ←後続コマンドを実行しない
```

5.8.3　ジョブステップを定義する

#-adsh_step で始まるジョブステップ定義コマンドを使用して，ジョブ定義スクリプトの一部分を，ジョブステップとしてグループ化します。ジョブステップとは，グループ化した一まとまりのコマンド群のことです。

(1)　グループ化の方法

ジョブステップとして通常実行するコマンド群は，#-adsh_step_start コマンドから#-adsh_step_error コマンドまたは#-adsh_step_end コマンドまでのブロック内に記述します。このブロックをジョブステップ正常ブロックと呼びます。

ジョブステップ正常ブロック内の最後のコマンドがエラー終了した場合にだけ実行するコマンド群を，#-adsh_step_error コマンドから#-adsh_step_end コマンドまでのブロック内に記述します。このブロックをジョブステップエラーブロックと呼びます。

(2)　ジョブステップ実行の流れ

ジョブステップ実行の流れを，次に示します。

1. エラー終了した先行ジョブステップやエラー終了したコマンドの有無と run 属性の指定によって，ジョブステップをスキップするかどうかを決定します。run 属性については，「9.5　スクリプト拡張コマンド」の「#-adsh_step_start コマンド，#-adsh_step_error コマンド，#-adsh_step_end コマンド（ジョブステップを定義する）」を参照してください。
2. ジョブステップ正常ブロック内のコマンドを順に実行します。コマンドがエラー終了した場合，onError 属性が stop のときは，後続コマンドを実行しないでジョブステップ正常ブロックを抜けます。onError 属性が cont のときは後続コマンドを実行してからジョブステップ正常ブロックを抜けます。
3. #-adsh_step_error が定義されている場合，ジョブステップ正常ブロック内で最後に実行したコマンドがエラー終了したときだけ，ジョブステップエラーブロック内のコマンドを順に実行します。

(3)　ジョブステップ内でだけ有効なシェル変数の宣言

`stepVar` 属性を指定すると，このジョブステップ内でだけ有効なシェル変数を宣言できます。宣言したシェル変数をエクスポートしても，ジョブステップ内でだけエクスポートされた状態になります。

ジョブステップ開始時，ジョブコントローラが自動的にシェル変数を未定義の状態とします。ただし，ジョブステップ内で有効なシェル変数として`PATH` を指定した場合は，ジョブステップ開始前の値を引き継ぎます。

ジョブステップ終了時，ジョブコントローラが自動的にシェル変数をジョブステップ開始時の状態に戻します。

宣言するシェル変数は，ジョブステップ外のシェル変数と同名のシェル変数を宣言できます。その場合の注意事項を次に示します。

- 宣言したシェル変数は，ジョブステップ外の同名シェル変数とは，まったく別の変数として扱います。
- ジョブステップ開始時は，宣言したシェル変数は未定義状態になります。ジョブステップ外の同名シェル変数の値は，別のシェル変数として扱うため引き継ぎません。
- ジョブステップ外の同名シェル変数を，ジョブステップ内で参照・更新できません。
- ジョブステップ終了後は，再びジョブステップ外の同名シェル変数が参照・更新できます。

使用例を次に示します。

ジョブ定義スクリプトファイルの使用例

```
01: VAL1=AAA
02: echo ”ステップ開始前（ステップ外）です”
03: echo ”beforeStepVar1=”$VAL1
04: echo ”beforeStepVar2=”$VAL2
05:
06: #-adsh_step_start S1 -stepVar VAL1,VAL2
07:   echo ”ステップを開始しました”
08:   echo ”startStepVar1=”$VAL1
09:   echo ”startStepVar2=”$VAL2
10:   VAL1=XXX
11:   VAL2=YYY
12:   echo ”endStepVar1=”$VAL1
13:   echo ”endStepVar2=”$VAL2
14: #-adsh_step_end
15:
16: echo ”ステップを終了しました”
17: echo ”afterStepVar1=”$VAL1
18: echo ”afterStepVar2=”$VAL2
```

ジョブステップ外に同名シェル変数が存在するVAL1 と，ジョブステップ外に同名シェル変数が存在しないVAL2 を宣言しています。

実行結果を次に示します。

```
ステップ開始前(ステップ外)です
beforeStepVar1=AAA  ←ジョブステップ外のVAL1を参照。ジョブステップ内のVAL1とは別物
beforeStepVar2=     ←ジョブステップ外のVAL2を参照するが，存在しない
ステップを開始しました
startStepVar1=      ←ジョブステップ内のVAL1を参照。ジョブステップ外のVAL1とは別物
startStepVar2=
endStepVar1=XXX
endStepVar2=YYY
ステップを終了しました
afterStepVar1=AAA   ←ジョブステップ外のVAL1を参照。ジョブステップ内のVAL1とは別物
afterStepVar2=      ←ジョブステップ外のVAL2を参照するが，存在しない
```

ジョブステップ内でだけ有効なシェル変数にPATH を指定すると，シェル変数PATH へのパス追加をジョブステップローカルに行えます。シェル変数PATH の初期値はジョブステップ開始前の値を引き継ぎます。

ジョブステップ内でだけ有効なシェル変数PATH へのパス追加の例を次に示します。ジョブ定義スクリプト実行開始時のシェル変数PATH の値を，a:b と仮定します。

```
#-adsh_job J1                          →1.
cmdA
PATH=x:$PATH                           →2.
cmdB
#-adsh_step_start S1 -stepVar PATH     →3.
  cmdC
  PATH=y:$PATH                         →4.
  cmdD
#-adsh_step_end                        →5.
```

ジョブステップ内でだけ有効なシェル変数PATHへのパス追加の例の番号は，次に示す説明の順番と対応しています。

1. PATHの初期値は「a:b」とする。
2. ジョブステップ内で有効。PATHの値は「x:a:b」になる。
3. stepVarにPATHを指定する。シェル変数は削除されないで，値は「x:a:b」のまま。
4. ジョブステップ内で有効。PATHの値は「y:x:a:b」になる。
5. PATHをジョブステップ開始時の値に戻す。PATHの値は「x:a:b」になる。

PATH_CONV_VARパラメーター，または#-adsh_path_varコマンドを使用すると，WindowsとUNIX間でディレクトリのパスを変換するためのシェル変数を定義して使用できます。機能の詳細については，「5.8.5 パス名を扱うシェル変数を定義する」を参照してください。

(4) ジョブステップエラー時のジョブステップ終了コードの指定

ジョブステップでエラーが発生した場合，ジョブステップの終了コードを任意に設定できます。終了コードを任意に設定するには，ジョブステップエラーブロック内でexitコマンドの引数に任意の終了コードを指定して実行します。このとき，exitコマンドによってジョブが終了するため，ジョブの終了コードもexitコマンドの引数に指定した値となります。

ジョブステップエラーブロック内で.（ドット）コマンドまたは#-adsh_scriptコマンドを用いて外部スクリプトを呼び出し，呼び出した外部スクリプト内でexitコマンドに引数を指定して実行した場合も，引数に指定した値がジョブステップの終了コードになります。

ジョブステップエラーブロック内でexitコマンドに引数を指定して実行する例を次に示します。

```
#-adsh_step_start STEP1
  cmdA
  cmdB                  ←終了コード1でエラー終了
  cmdC
#-adsh_step_error
  exit 4                ←ジョブステップはエラー終了し，exitの引数4が
#-adsh_step_end           ジョブステップの終了コードになる
```

ジョブステップエラーブロック内でexitコマンドに引数を指定しないで実行した場合は，ジョブステップ正常ブロック内で最後に実行したコマンドの終了コードがジョブステップの終了コードとなります。

ジョブステップエラーブロック内で exit コマンドに引数を指定しないで実行する例を次に示します。

```
#-adsh_step_start STEP1
  cmdA
  cmdB              ←終了コード1でエラー終了
  cmdC
#-adsh_step_error
  exit              ←ジョブステップはエラー終了し，exitの引数なしのため，
#-adsh_step_end       cmdBの終了コード1がジョブステップの終了コードになる
```

(5) ジョブステップの実行例

すべてのコマンドが正常終了する場合と，途中でコマンドがエラー終了する場合のジョブ定義スクリプトファイルの実行例を次に示します。

- すべてのコマンドが正常終了する場合の実行例

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP001
  command1                              ←実行する
  command2                              ←実行する
  command3                              ←実行する
#-adsh_step_error
  command4                              ←実行しない
  command5                              ←実行しない
#-adsh_step_end
#-adsh_step_start STEP002
  command6                              ←実行する
#-adsh_step_end
#-adsh_step_start STEP003 -run abnormal
  command7                              ←実行しない
#-adsh_step_end
#-adsh_step_start STEP004 -run always
  command8                              ←実行する
#-adsh_step_end
```

- command2 がエラー終了する場合の実行例（onError 属性が stop）

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP001 -onError stop
  command1                              ←実行する
  command2                              ←実行する（エラー終了）
  command3                              ←実行しない
#-adsh_step_error
  command4                              ←実行する
  command5                              ←実行する
#-adsh_step_end
#-adsh_step_start STEP002
  command6                              ←実行しない
#-adsh_step_end
#-adsh_step_start STEP003 -run abnormal
  command7                              ←実行する
#-adsh_step_end
```

```
#-adsh_step_start STEP004 -run always
  command8                              ←実行する
#-adsh_step_end
```

- command2 がエラー終了する場合の実行例（onError 属性が cont）

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP001 -onError cont
  command1                              ←実行する
  command2                              ←実行する（エラー終了）
  command3                              ←実行する
#-adsh_step_error
  command4                              ←実行しない
  command5                              ←実行しない
#-adsh_step_end
#-adsh_step_start STEP002
  command6                              ←実行する
#-adsh_step_end
#-adsh_step_start STEP003 -run abnormal
  command7                              ←実行しない
#-adsh_step_end
#-adsh_step_start STEP004 -run always
  command8                              ←実行する
#-adsh_step_end
```

ジョブステップ外のコマンドがエラーとなった場合，後続ジョブ定義スクリプトは次のようになります。

- 後続のジョブステップ外コマンドは実行します。
- 後続の run 属性が normal のジョブステップは実行しません。
- 後続の run 属性が abnormal または always のジョブステップは実行します。

実行例を次に示します。

```
#-adsh_job CMD_ERROR

echo "Job start."
cd -x  #エラーとなるコマンド            ←このコマンドがエラーとなる
echo "Job end."                       ←ジョブステップ外のコマンドは実行する

#-adsh_step_start STEP01 -run normal    ←run normal指定のジョブステップは実行しない
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP02 -run abnormal  ←run abnormal指定のジョブステップは実行する
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP03 -run always    ←run always指定のジョブステップは実行する
  echo "command in step"
#-adsh_step_end
```

ジョブステップがエラーとなった場合，後続ジョブステップを実行するかどうかは後続ジョブステップのrun属性によって決定します。しかし，後続のジョブステップ外コマンドは実行しません。実行例を次に示します。

```
#-adsh_job STEP_ERROR

#-adsh_step_start STEP01
  echo "Step start."
  cd -x  #エラーとなるコマンド
  echo "Step end."
#-adsh_step_end     ←ジョブステップはエラーで終了する

echo "Job end."     ←先行ジョブステップがエラーの場合，後続のジョブステップ外コマンドは
                      実行しない
```

ジョブステップエラーブロック内で実行したコマンドの終了コードは，ジョブステップの終了コードには影響しません。ジョブステップ正常ブロック内で最後に実行したコマンドの終了コードが，ジョブステップの終了コードになります。実行例を次に示します。

```
#-adsh_job STEP_ERRBLK_RC

#-adsh_step_start STEP01
  echo "Step start."
  cd -x  #rc=1でエラーとなるコマンド       ←この結果がジョブステップの終了コードになる
  echo "Step end."
#-adsh_step_error
  echo "step error block"  #rc=0となるコマンド   ←ジョブステップのrcに影響を与えない
#-adsh_step_end    ←ジョブステップはcdコマンドのエラー結果が反映され，rc=1で終了する
```

(6) 関数，trap コマンドとの関係

関数およびtrapコマンドのアクションは，それらを定義した場所には関係なく，実行された場所によってジョブステップ内外のどちらで実行されたかが決まります。

5.8.4 正常終了するコマンドを定義する

(1) 終了コードが 0 以外でも正常終了となるコマンドを定義する

コマンドの終了コードが0以外でも正常終了と認識されるよう，正常終了とする終了コードのしきい値を設定できます。終了コードがこのしきい値を超えなければ，正常終了と処理されます。

終了コードのしきい値は次に示す環境設定パラメーターで設定します。

- CMDRC_THRESHOLD_USE_PRESET パラメーター
  次のすべてのUNIX互換コマンドに対し，終了コードのしきい値を設定します。設定できるしきい値は0または1です。

- cmp コマンド
- diff コマンド
- egrep コマンド
- expr コマンド
- grep コマンド
- sort コマンド

詳細については，「CMDRC_THRESHOLD_USE_PRESET パラメーター（UNIX 互換コマンドの終了コードのしきい値を定義する）」を参照してください。

- CMDRC_THRESHOLD_DEFINE パラメーター

  対象となるコマンドと，終了コードのしきい値を設定します。指定できるコマンドの種類は次のとおりです。

  - 外部コマンド
  - UNIX 互換コマンド
  - シェルスクリプト
  - シェル運用コマンド
  - 子孫ジョブ

  詳細については，「CMDRC_THRESHOLD_DEFINE パラメーター（コマンドの終了コードのしきい値を定義する）」

(2) 常に正常終了するコマンドを定義する

#-adsh_rc_ignore コマンドを使用すると，定義した名称のコマンドは，終了コードに関係なく常に正常終了します。その場合，対象コマンドの終了コードはジョブステップの成功または失敗の判定に影響しません。

ただし，コマンドがシグナル終了した場合は，指定に関係なくコマンドエラー終了になります。

なお，次に示すコマンドは，終了コードが 0 以外でもエラーになりません。したがって，このコマンドの指定に関係なく終了コードを無視します。

- true コマンド，false コマンド

#-adsh_rc_ignore コマンドの定義は，指定した個所以降のジョブ定義スクリプト実行で有効となります。ジョブステップ外に指定した場合はジョブ定義スクリプト全体に有効で，ジョブステップ内に指定した場合はジョブステップ内だけで有効です。ジョブステップ内に指定した場合，指定した個所以降からジョブステップ終了まで有効となり，ジョブステップ外に指定した値は一時的に無効になります。また，ジョブステップ内に指定するまではジョブステップ外に指定した値が有効になります。

指定例を次に示します。

```
01: #!/opt/jp1as/bin/adshexec
02: #-adsh_job JOB0001
03:
04: uap01
05: uap02
06: #-adsh_rc_ignore uap03,uap04    ←1.  ジョブステップ外に指定
07: uap03                           ←行07～14の指定が1.の有効範囲
08:
09: #-adsh_step_start STEP1
10:   uap04
11: #-adsh_step_end
12:
13: #-adsh_step_start STEP2
14:   uap05
15:   #-adsh_rc_ignore uap06,uap07  ←2.  ジョブステップ内に指定
16:   uap06                         ←行16～17の指定が2.の有効範囲
17:   uap07
18: #-adsh_step_end
19:
20: #-adsh_step_start STEP4         ←行20～22の指定が1.の有効範囲
21:   uap08
22: #-adsh_step_end
```

(3)　終了コードが 0 以外でも正常終了となる機能の優先順位

CMDRC_THRESHOLD_DEFINE パラメーター，#-adsh_rc_ignore コマンド，#-adsh_step_start コマンドの successRC 属性，および adshcmdrc コマンドを同時に指定したときの優先順位は次のようになります。数字が小さい方が優先順位が高くなります。また，CMDRC_THRESHOLD_USE_PRESET パラメーターの対象となるコマンドに対して定義する場合は，CMDRC_THRESHOLD_USE_PRESET パラメーターは優先順位 7 となります。

```
1. ジョブステップ内に定義した#-adsh_rc_ignoreコマンド
2. ジョブステップ外に定義した#-adsh_rc_ignoreコマンド
3. ジョブステップ内に定義したadshcmdrcコマンド
4. #-adsh_step_startコマンドのsuccessRC属性
5. ジョブステップ外に定義したadshcmdrcコマンド
6. CMDRC_THRESHOLD_DEFINEパラメータ
7. CMDRC_THRESHOLD_USE_PRESETパラメータ
```

*コマンド名*のコマンドに対して，終了コードが 0 以外でも正常終了となる各機能を定義したときの例を次に示します。

ジョブ定義スクリプト

```
#-adsh_rc_ignore コマンド名                    …2
adshcmdrc コマンド名 1                         …5

#-adsh_step_start STEP1 -successRC 2           …4
  #-adsh_rc_ignore コマンド名                  …1
  adshcmdrc コマンド名 3                       …3
  コマンド名
#-adsh_step_end
```

環境ファイル

```
#-adsh_conf CMDRC_THRESHOLD_DEFINE コマンド名 10  …6
```

5.8.5　パス名を扱うシェル変数を定義する

PATH_CONV_VAR パラメーター，または#-adsh_path_var コマンドを使用すると，パス名を扱うシェル変数を定義できます。パス名を扱うシェル変数を使用すると，それらを含む文字列のパス区切り文字およびディレクトリ区切り文字を Windows や UNIX などの環境に合わせて変換できます。

PATH_CONV_NOVAR パラメーターを使用することで，パス名を扱わないシェル変数名を定義できます。PATH_CONV_VAR の指定と組み合わせて，複雑な指定ができます。

ジョブコントローラは，次の条件をすべて満たす文字列のパス区切り文字およびディレクトリ区切り文字を変換します。パス変換ルール 1 とパス変換ルール 2 については，「PATH_CONV_RULE パラメーター」を参照してください。

- 「"（ダブルクォーテーション）」で囲まれた文字列（パス変換ルール 1 の場合）
- 「'（シングルクォーテーション）」で囲まれていない文字列（パス変換ルール 2 の場合）
- 環境ファイルの PATH_CONV_ENABLE パラメーターで定義されたパス区切り文字で区切られた文字列の中で，文字列「$**パスを扱うシェル変数名**」または文字列「${**パスを扱うシェル変数名**}」と前方一致する文字列

#-adsh_path_var コマンドは，次のどちらかの場合だけ使用できます。

- 1 行目の「#!**任意文字列**」の次の行
- #-adsh_job コマンドの次の行

なお，次のようにすると継続行を指定できます。ただし，1 行目の#-adsh_path_var コマンドの後ろにはシェル変数名を指定できません。

```
1行目：#-adsh_path_var
2行目：#-adsh  var001,var002,var003,var004
```

(1)　使用例

パス名を扱うシェル変数「PATH」「DIR」「DIR3」を#-adsh_path_var コマンドおよび PATH_CONV_VAR パラメーターに定義する例を次に示します。

(a)　Windows の場合

- 環境ファイルの指定

```
#-adsh_conf PATH_CONV_ENABLE / :                    ←パス変換の有効化
#-adsh_conf PATH_CONV_RULE 1                        ←パス変換ルール1を選択
```

```
#-adsh_conf PATH_CONV /home/hitachi "C:¥¥hitachi"  ←パス文字列の置換1
#-adsh_conf PATH_CONV /tmp/jp1as "D:¥¥jp1as_tmp"   ←パス文字列の置換2
#-adsh_conf PATH_CONV /tmp "C:¥¥temp"              ←パス文字列の置換3
#-adsh_conf PATH_CONV_ACCESS /dev/null nul         ←ファイル入出力時のファイルパスの変換
#-adsh_conf PATH_CONV_VAR DIR3                     ←シェル変数の定義1
```

- ジョブ定義スクリプトの指定

```
#-adsh_path_var PATH,DIR          ←シェル変数の定義2

DIR="/home/hitachi/bin"           ←「DIR="C:¥¥hitachi¥¥bin"」にパス文字列の置換1で変換

"$DIR/myprog"                     ←シェル変数の定義2に従い「"$DIR¥¥myprog"」に変換

"${DIR}/myprog"                   ←シェル変数の定義2に従い「"${DIR}¥¥myprog"」に変換

DIR2=$DIR
"$DIR2/myprog"                    ←「$DIR2」はシェル変数の定義1，2にないため，変換されな
い

$DIR/myprog                       ←"(ダブルクォーテーション)で囲まれていないため，変換さ
れない

FILE1="/tmp/jp1as/file"           ←パス文字列の置換2に従い「"D:¥¥jp1as_tmp¥¥file"」に変換

DIR3=""
ls "$DIR3../bin"                  ←シェル変数の定義1に従い「"..¥¥bin"」に変換。相対パスも
変換

DIR4="/home/hitachi/sbin:$DIR2"   ←パス文字列の置換1に従い「C:¥¥hitachi¥¥sbin;$DIR2」に変
換。
                                    パス区切り文字も変換

PATH="../bin/:$DIR"               ←シェル変数の定義2に従い「..¥¥bin¥¥;$DIR」に変換。
                                    パス区切り文字も変換

"$DIR2/myprog" > /dev/null        ←「nul」にファイル入出力時のファイルパスの変換で変換
```

(b) UNIX の場合

- 環境ファイルの指定

```
#-adsh_conf PATH_CONV_ENABLE ¥¥ ;                  ←パス変換の有効化
#-adsh_conf PATH_CONV "C:¥¥hitachi" /home/hitachi  ←パス文字列の置換1
#-adsh_conf PATH_CONV "D:¥¥jp1as_tmp" /tmp/jp1as   ←パス文字列の置換2
#-adsh_conf PATH_CONV "C:¥¥temp" /tmp              ←パス文字列の置換3
#-adsh_conf PATH_CONV_ACCESS nul /dev/null         ←ファイル入出力時のファイルパスの変換
#-adsh_conf PATH_CONV_VAR DIR3                     ←シェル変数の定義1
```

- ジョブ定義スクリプトの指定

```
#-adsh_path_var PATH,DIR          ←シェル変数の定義2

DIR="C:¥¥hitachi¥¥bin"            ←パス文字列の置換1に従い「DIR="/home/hitachi/bin"」に変
換
"$DIR¥¥myprog"                    ←シェル変数の定義2に従い「"$DIR/myprog"」に変換
```

```
"${DIR}¥¥myprog"                ←シェル変数の定義2に従い「"${DIR}/myprog"」に変換

DIR2=$DIR
"$DIR2¥¥myprog"                 ←「$DIR2」はシェル変数の定義1，2にないため，変換されない

$DIR¥¥myprog                    ←"(ダブルクォーテーション)で囲まれていないため，変換され
ない

FILE1="D:¥¥jp1as_tmp¥¥file"     ←パス文字列の置換2に従い「"/tmp/jp1as/file"」に変換

DIR3=""
ls "$DIR3..¥¥bin"               ←シェル変数の定義1に従い「"../bin"」に変換。相対パスも変
換

DIR4="C:¥¥hitachi¥¥sbin;$DIR2"  ←パス文字列の置換1に従い「/home/hitachi/sbin:$DIR2」に変
換。
                                  パス区切り文字も変換

PATH="..¥¥bin¥¥;$DIR"           ←シェル変数の定義2に従い「../bin/:$DIR」に変換
                                  パス区切り文字も変換

"$DIR2¥¥myprog" > nul           ←ファイル入出力時のファイルパスの変換に従い「/dev/null」
に変換
```

(c) PATH_CONV_VAR パラメーターと PATH_CONV_NOVAR パラメーターの使用例

- 環境ファイルの指定

```
#-adsh_conf PATH_CONV_VAR DIR*                ←シェル変数の定義1
#-adsh_conf PATH_CONV_NOVAR DIRNO*            ←シェル変数の定義2
```

シェル変数の定義 1 は，名称の先頭が DIR のシェル変数はパス名を扱うシェル変数として定義しています。シェル変数の定義 2 は，名称の先頭が DIRNO のシェル変数はパス名を扱わないシェル変数として定義しています。後続の指定が優先するため，名称の先頭が DIRNO のシェル変数を除き，名称の先頭が DIR のシェル変数はパス名を扱うシェル変数として定義したことになります。

- ジョブ定義スクリプトの指定

```
#-adsh_path_var DIRNORTH                      ←シェル変数の定義3
```

DIRNORTH はパス名を扱うシェル変数として定義しています。#-adsh_path_var コマンドの定義は環境ファイルの定義よりも優先します。このため，名称の先頭が DIRNO であっても DIRNORTH はパス名を扱う変数となります。

(2) ジョブ定義スクリプトイメージへの出力例

ジョブ実行ログには，パスを変換する前のジョブ定義スクリプトに加え，変換したあとの行も出力します。また，ジョブ定義スクリプト中で合致した変換規則，ジョブ定義スクリプト名および行番号をメッセージとして出力します。

```
********  ジョブコントローラのメッセージ出力  ********
(省略)
19:41:24 000071 KNAX6803-I PATH_CONV_ACCESSパラメーターの変換規則に合致しました。
```

```
filename="D:¥home¥user001¥path_conv.ash" line=4 rule_str="./local.log":".¥¥mylog"
(省略)

********   ジョブ定義スクリプトの内容     ********

***** D:¥home¥user001¥path_conv.ash *****
0001 : #-adsh_job JOB001
0002 : #-adsh_path_var DIR
0003 : DIR="/home/hitachi/bin"; DIR2="/tmp/tmpfile"
0004 : "$DIR/myprog" > ./local.log
0005 : exit

***** パス変換行 (D:¥home¥user001¥path_conv.ash)  *****
0003 : DIR="c:¥¥Program Files"; DIR2="c:¥¥temp¥¥tmpfile"
0004 : "$DIR¥¥myprog" > ./local.log

***** パス変換情報 *****
KNAX6800-I パスの変換規則に合致しました。 filename="D:¥home¥user001¥path_conv.ash" line=3
rule_str="/home/hitachi/bin":"c:¥¥Program Files"
KNAX6800-I パスの変換規則に合致しました。 filename="D:¥home¥user001¥path_conv.ash" line=3
rule_str="/tmp":"c:¥¥temp"
KNAX6801-I パスを扱うシェル変数による変換規則に合致しました。 filename="D:¥home
¥user001¥path_conv.ash" line=4 rule_var="DIR"
```

5.8.6 実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルを呼び出す

#-adsh_script コマンドを使用して，外部のジョブ定義スクリプトファイルのジョブコントローラ起動時点での内容を，現在実行中のジョブ定義スクリプトファイルに挿入します。

このコマンドはシェル標準コマンドの.（ドット）コマンドとは異なって，ジョブコントローラ起動時点での外部スクリプトの内容を，呼び出し元のジョブ定義スクリプト内に展開します。呼び出し元のジョブ定義スクリプトと展開後のジョブ定義スクリプトは全体を 1 個のジョブ定義スクリプトとして構文解析し実行します。

外部スクリプトの定義と，呼び出し元ジョブ定義スクリプトの例を次に示します。

- /scripts/exScript.ash（ジョブ開始時点での内容）

```
#-adsh_step_start exS1
  exUap01
#-adsh_step_end
exUap02
```

- script.ash

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

uap01
#-adsh_script /scripts/exScript.ash
#-adsh_step_start S2 -run normal
```

```
  uap2
#-adsh_step_end
```

script.ash は，次のジョブ定義スクリプトと同等の内容です。

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

uap01
#-adsh_step_start exS1     ←ここから4行がexScript.ashが展開された部分
  exUap01
#-adsh_step_end
exUap02
#-adsh_step_start S2 -run normal
  uap2
#-adsh_step_end
```

また，外部スクリプト/scripts/exScript.ash 内のexUap02 が終了コード0 以外でエラー終了した場合，後続のジョブステップS2 はrun 属性にnormal が指定されているため，実行されないでジョブが終了します。

.（ドット）コマンドと#-adsh_script コマンドの違いを次の表に示します。

表 5-39　.（ドット）コマンドと#-adsh_script コマンドの違い

項番	比較項目		.（ドット）コマンド	#-adsh_script コマンド
1	外部のジョブ定義スクリプト内でのスクリプト拡張コマンドの扱い		コメントとして扱います	スクリプト拡張コマンドとして扱います
2	スクリプトイメージへの出力		出力されません	出力されます
3	相対パス指定時の動作		相対パスで指定できます ただし，環境変数PATH の値を参照してパスを解決します	相対パスで指定できます ただし，環境変数PATH の値を参照しないで，adshexec 起動時のカレントディレクトリからの相対パスとして解釈します
4	ジョブ内で使用できる上限数		制限がありません	4,095 個が上限です
5	外部のジョブ定義スクリプト内からのコマンド実行		実行できます	実行できます ただし，同じ外部のジョブ定義スクリプトを再帰的に呼び出し，実行することはできません
6	外部のジョブ定義スクリプトへの引数の指定		指定できます	指定できません
7	CUI デバッガ	break コマンドによる外部のジョブ定義スクリプト内へのブレークポイントの設定	設定できません	設定できます
8		info functions コマンドによる外部のジョブ	表示できません ただし，外部のジョブ定義スクリプト内での関数定義が完了し	表示できます

項番	比較項目		.（ドット）コマンド	#-adsh_script コマンド
8	CUI デバッガ	定義スクリプト内で定義した関数の情報表示	た時点で表示できるようになります	表示できます
9		list コマンドによる外部のジョブ定義スクリプト内容の表示	表示できません	表示できます
10		ジョブ定義スクリプトの実行を停止した際に表示される内容	行番号：表示できます ソースファイル行：表示できません コマンド文字列：表示できます	行番号：表示できます ソースファイル行：表示できます コマンド文字列：表示できます

(1)　相対パスで指定する場合

外部スクリプトを相対パスで指定すると，先行ジョブ定義スクリプトの動作に関係なく，adshexec 起動時のカレントディレクトリからのパスになります。

また，ほかのディレクトリからの相対パス指定はできません。その場合は，絶対パスで指定してください。

指定例を次に示します。

adshexec コマンド起動時のカレントディレクトリ：/scripts

```
#/opt/jp1as/bin/adshexec
cd /work
#-adsh_script ex_script.ash       ←/scripts/ex_script.ashが実行される
```

この例では，#-adsh_script コマンドで外部スクリプトファイル/scripts/ex_script.ash が実行されます。直前の cd コマンドでカレントディレクトリを移動していますが，呼び出す外部スクリプトファイルのパスには影響しません。

(2)　絶対パスで指定する場合

/work/ex_script.ash を実行したい場合は，次のように絶対パスで指定してください。

adshexec コマンド起動時のカレントディレクトリ：/scripts

```
#/opt/jp1as/bin/adshexec
cd /work
#-adsh_script /work/ex_script.ash       ←/work/ex_script.ashが実行される
```

5.8.7　スクリプト拡張コマンドの終了コードとエラー発生時の動作

スクリプト拡張コマンドの終了コードを次の表に示します。終了コードは環境設定パラメーターで定義できます。

表 5-40　スクリプト拡張コマンドの終了コード

スクリプト拡張コマンド	実行結果	終了コードのデフォルト値	終了コードを設定する環境設定パラメーター
#-adsh_file #-adsh_file_temp #-adsh_job #-adsh_job_stop #-adsh_path_var #-adsh_rc_ignore #-adsh_spoolfile #-adsh_step_start #-adsh_step_error	正常終了	0	ADSHCMD_RC_SUCCESS
	エラー終了	1	ADSHCMD_RC_ERROR
#-adsh_step_end	ジョブステップ正常終了	ジョブステップ正常ブロック内で最後に終了したコマンドの終了コード	－
	ジョブステップエラー終了		
	ジョブステップエラーブロック内で引数を指定した exit コマンドを実行して終了	exit コマンドの引数	－
	#-adsh_step_end 自身のエラー終了	1	ADSHCMD_RC_ERROR
#-adsh_script	正常終了	呼び出した外部スクリプト内で最後に終了したコマンドの終了コード	－
	エラー終了	1	ADSHCMD_RC_ERROR

（凡例）

－：該当しません。

スクリプト拡張コマンドを実行して，エラー終了またはジョブステップエラー終了となった場合，次のとおり動作します。

- run 属性に abnormal または always が指定されている場合，ジョブステップを実行します。
- run 属性が省略されているまたは normal が指定されている場合，ジョブステップを実行しません。
- ジョブステップ外のコマンドは実行しません。

実行例を次に示します。

```
#-adsh_job EXCMD_ERROR

echo "Job start."

#-adsh_file ERRFILE file01 -chk exist    ←このスクリプト拡張コマンドがエラーとなる

#-adsh_step_start STEP01 -run normal     ←run属性にnormalを指定したジョブステップは
```

```
                                                実行しない
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP02 -run abnormal  ←run属性にabnormalを指定した
                                          ジョブステップは実行する
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP03 -run always    ←run属性にalwaysを指定したジョブステップは
                                          実行する
  echo "command in step"
#-adsh_step_end

echo "Job end."                         ←ジョブステップ外のコマンドは実行しない
```

5.8.8 ジョブ，ジョブステップおよびコマンドの終了コード

終了コードおよび実行結果の正常・異常について説明します。

(1) ジョブの終了コード

ジョブの終了コードは最後に実行したジョブ定義スクリプトの終了コードになります。

JP1/Advanced Shell はジョブの実行結果について正常・異常を区別しません。終了コードを JP1/AJS などにそのまま返します。

JP1/AJS からジョブを実行した場合，JP1/AJS - View などで確認できる JP1/AJS のジョブの終了コードは，adshexec コマンドの終了コードではなく，JP1/AJS が定める終了コードとなることがあります。この場合，JP1/AJS のジョブの終了コードと，ジョブ実行ログなどに出力される JP1/Advanced Shell のジョブの終了コードとは，値が異なることがあります。

例

JP1/AJS から起動した UNIX 版の adshexec コマンドが SIGINT を受信した場合，JP1/Advanced Shell のジョブの終了コードは 130 になりますが，JP1/AJS のジョブの終了コードは-1 になります。

ただし，ジョブの中でエラーが発生していた場合は，エラーメッセージを出力します。

(2) ジョブステップの終了コード

ジョブステップの終了コードは，ジョブステップ正常ブロック内で最後に実行したコマンドの終了コードになります。ただし，ジョブステップエラーブロックで exit コマンドに引数を指定して実行すると，exit コマンドの引数をジョブステップの終了コードにできます。それ以外でジョブステップエラーブロック内で実行したコマンドの終了コードは，ジョブステップの終了コードに影響しません。ジョブステップの正常終了およびエラー終了について説明します。

- ジョブステップの正常終了

ジョブステップ正常ブロック内で最後に実行したコマンドが正常終了しています。

- ジョブステップのエラー終了

  ジョブステップ正常ブロック内で最後に実行したコマンドがエラー終了しています。

(3) 外部コマンドの終了コード

外部コマンドの終了コードは，各コマンドで定められた終了コードになります。

外部コマンドがリターンできる値の範囲は，プラットフォームや外部コマンドを記述したプログラム言語の言語仕様などによって異なりますが，0～255 とすることを推奨します。この範囲を超えた場合，ジョブコントローラでは，次に示す値を外部コマンドの終了コードとして扱います。

【UNIX 限定】

外部コマンドがリターンする値の下位 8 ビット

【Windows 限定】

- 外部コマンドがリターンする値が0 以上の場合，値の下位 8 ビット
- 外部コマンドがリターンする値が0 未満の場合，255
- 外部コマンドで例外※が発生して終了した場合，例外コードの下位 8 ビット

注※

例外として扱う例外コードと意味は次のとおりです。

表 5-41　例外として扱う例外コードと意味

項番	例外コード	例外が発生した場合の外部コマンドの終了コード	意味
1	0xC0000005	5	スレッドがアクセス権を持っていない仮想アドレスへアクセスしようとしました。
2	0x80000003	3	ブレークポイントに到達しました。
3	0x80000002	2	メモリアクセスに関してアライメント規約の存在するハードウェア上で，ミスアライメントされたデータにアクセスしました（例えば，16 ビット値が 2 バイト境界にまたがったり，32 ビット値が 4 バイト境界にまたがったりすることはできません）。
4	0x80000004	4	トレースまたはシングルステップ機構による 1 命令ごとの実行です。
5	0xC000008C	140	スレッドが配列の範囲外にアクセスしようとしたことが，ハードウェアによって検出されました。
6	0xC000008D	141	浮動小数点演算で，オペランドのうち少なくとも 1 つが非正規化数（普通の浮動小数点フォーマットでは表現できないほど小さな値）です。
7	0xC000008E	142	スレッドが，浮動小数点演算で0 による除算をしようとしました。
8	0xC000008F	143	浮動小数点演算の結果，正確な値が計算できませんでした。

項番	例外コード	例外が発生した場合の外部コマンドの終了コード	意味
9	0xC0000030	48	この表に列挙した以外の浮動小数点演算の例外です。
10	0xC0000091	145	浮動小数点演算の結果，指数部の値が制限範囲を超えました。
11	0xC0000032	50	浮動小数点演算の結果，スタックがオーバーフローまたはアンダーフローを起こしました。
12	0xC0000033	51	浮動小数点演算の結果，指数部の値が制限範囲を下回りました。
13	0xC0000094	148	スレッドが，整数演算で0 による除算をしようとしました。
14	0xC0000035	53	整数演算の結果，オーバーフローが発生しました。
15	0xC0000096	150	現在のマシンモードでは実行できない命令（特権命令）を実行しようとしました。
16	0xC0000025	37	継続できない例外を起こした命令に対して，再実行を試みました。

例えば外部コマンドがリターンする値が512 の場合，外部コマンドの終了コードは0 になります。また，ジョブコントローラは，外部コマンドが正常終了したものとして扱います。

```
'E:¥bin¥uap01.exe'          ←uap01.exeがリターンした値は512
if [ $? -ne 0 ]             ←下位8ビットを終了コードと扱うため，終了コードは0
then
  exit 2                    ←終了コードは0のため，exit2は実行されない
fi
```

外部コマンドがリターンする値が0～255 以外の場合は，シェル変数ADSH_RC_EXTERNAL を使用して，外部コマンドの実行結果を取得してください。

```
'E:¥bin¥uap01.exe'              ←uap01.exeがリターンした値は512
if [ $ADSH_RC_EXTERNAL -ne 0 ] ←シェル変数ADSH_RC_EXTERNALには外部コマンドがリターンした
                                  値が設定される
then
  exit 2                        ←ADSH_RC_EXTERNALの値は512のため，exit 2は実行される
fi
```

詳細は「5.5.4　外部コマンドの終了コードが設定されるシェル変数【Windows 限定】」を参照してください。

外部コマンドの実行結果が次の表のどれかに該当する場合，その外部コマンドがエラー終了したと見なします。

外部コマンド実行結果	終了コード
外部コマンドの終了コードが0 ではない場合（successRC 属性，CMDRC_THRESHOLD_USE_PRESET パラメーター，CMDRC_THRESHOLD_DEFINE パラメーター，adshcmdrc コマンドで値を変更できる）	外部コマンドの終了コード

外部コマンド実行結果	終了コード
外部コマンドがシグナル終了した場合	外部コマンドが定めるシグナル終了時の終了コード※
指定された外部コマンドに実行権限がなく，実行できなかった場合	126
指定された外部コマンドが存在しないで，実行できなかった場合	127

注※

シグナルを受信すると，受信したシグナルの種別を示す文字列を標準エラー出力に出力することがあります。

なお，#-adsh_rc_ignore コマンドで指定した外部コマンドは，終了コードに関係なくエラーにはなりません。

(4) 組み込みコマンドの終了コード

組み込みコマンドの終了コードは，各コマンドで定められた終了コードになります。組み込みコマンドの場合，実行結果の正常・エラーは，終了コードの値ではなく各コマンドの実行時にエラーとなる現象が発生したかどうかで判定されます。ただし，#-adsh_rc_ignore コマンドで指定した組み込みコマンドは，コマンドの実行結果に関係なく，常に正常終了します。

各コマンドが正常・エラーとなる条件については，「9. ジョブ定義スクリプトのコマンドおよび制御文」に記載している各コマンドの終了コードをご確認ください。

(5) 関数の終了コード

関数では，関数内で最後に実行したコマンドの終了コードが関数の終了コードになります。CMDRC_CMDGRP_CHECK パラメータに FUNCTION を指定して実行することで，関数の終了コードに従ってジョブおよびジョブステップのエラー判定ができるようになります。

CMDRC_CMDGRP_CHECK パラメータの詳細については，「7.3.8 CMDRC_CMDGRP_CHECK パラメーター（関数の終了コードに従ってジョブおよびジョブステップのエラー判定をする）」を，関数の詳細については「5.1.4 関数」を参照してください。

(6) 注意事項

UNIX 互換コマンドおよびユーザーが作成したコマンドは，正常終了してもコマンドの終了コードが 0 ではない場合があります。例えば，diff コマンドは比較したファイルが異なるとき，終了コードが 1 となります。

このようなコマンドについては，正常かエラーかが正しく判定されるように，次の方法で記述してください。UNIX 互換コマンドの終了コードの詳細については，「8.4 UNIX 互換コマンド」を参照してください。

(a) 環境設定パラメーターで指定する場合

- UNIX 互換コマンドが正常かエラーかを正しく判定するには，CMDRC_THRESHOLD_USE_PRESET パラメーターに ENABLE を指定します。
- OS 提供のコマンドまたはユーザーが作成したコマンドが正常かエラーかを正しく判定するには，CMDRC_THRESHOLD_DEFINE パラメーターで指定します。

(b) ジョブ定義スクリプトで指定する場合

- コマンドの実行結果を常に正常終了とする場合は，#-adsh_rc_ignore コマンドの引数に，正常終了とするコマンド名を指定します。
- 正常かエラーかをコマンドの仕様に従って判定する場合は，対象コマンドをジョブステップ内に定義し，successRC 属性で正常終了となる終了コードを指定します。successRC 属性による指定は，ジョブステップ内のすべてのコマンドに対して有効となります。
- ジョブステップ内またはジョブステップ外に指定した対象コマンドに対して，正常かエラーかをコマンドの仕様に従って判定する場合は，adshcmdrc コマンドの引数に対象コマンドと正常終了と見なす終了コードしきい値を指定します。

5.8.9 シェル標準コマンドによるジョブの中断

シェル標準コマンドを実行した場合，シェル標準コマンドの種類およびシェル標準コマンドの実行結果によって，ジョブの実行を中断することがあります。この場合，KNAX6584-I メッセージを出力して，後続ジョブステップおよび後続ジョブ定義スクリプトを実行しません。run 属性が abnormal または always のジョブステップについても実行しません。

また，この場合，実行したコマンドに対しては，#-adsh_rc_ignore コマンドの指定および#-adsh_step_start コマンドの successRC 属性の指定は有効になりません。

(1) ジョブ定義スクリプトを即時終了するコマンドの実行

ジョブ定義スクリプトを即時終了するシェル標準コマンドを実行した場合，ジョブの実行を中断します。ジョブ定義スクリプトを即時終了するコマンドを次に示します。

- exit コマンド
- return コマンド※
- 引数に実行可能なコマンドが指定された exec コマンド

注※

次の場合は終了しません。

- 関数内で実行した場合
- 外部スクリプト内で実行した場合

(2) 続行できないエラーの発生

シェル標準コマンドを実行した場合，文法エラーなどジョブ定義スクリプト自体が正常に動作しないエラーが発生する場合があります。この場合，ジョブの実行を中断します。文法エラーが発生する例を次に示します。

- 引数を指定しないで unset コマンドを実行する場合
- return コマンドの引数に文字列を指定する場合※
- Windows 版で UNSUPPORT_TEST パラメーターに「ERR」が設定されている状態で，サポートしていない条件式を実行する場合

注※

次の場合，ジョブの実行は中断されません。

- 関数内で実行した場合
- 外部スクリプト内で実行した場合

5.8.10 ジョブ実行中にエラーが発生した場合の動作

ジョブ実行中にエラーが発生した場合の，後続のコマンド・制御文の動作について説明します。発生するエラーの種類を次に示します。

- スクリプト拡張コマンドのエラー

  #-adsh_file コマンドでファイル割り当てに失敗したときなどで発生します。
- シェル標準コマンドのエラー
  - 続行できる場合

    指定したコマンド名が見つからないとき，正規組み込みコマンドがエラーになったときなどで発生します。
  - 続行できない場合

    特殊組み込みコマンドのエラーなどで発生します。

(1) ジョブステップ外でエラーが発生した場合

ジョブステップ外でエラーが発生した場合の動作を次の表に示します。

表 5-42 ジョブステップ外でエラーが発生した場合の動作

発生したエラーの種類	後続のコマンド・制御文の動作
スクリプト拡張コマンドのエラー	• run 属性が abnormal または always のジョブステップを実行します。 • 上記以外のすべてのコマンド・制御文は実行しません。

発生したエラーの種類	後続のコマンド・制御文の動作
シェル標準コマンドのエラー（続行できる場合）	• run 属性が normal のジョブステップは実行しません。 • 上記以外のすべてのコマンド・制御文を実行します。
シェル標準コマンドのエラー（続行できない場合）	ジョブ定義スクリプトの実行を終了します。

ジョブステップ外でエラーが発生した場合の例を次に示します。

表 5-43　ジョブステップ外でエラーが発生した場合の例

ジョブ定義スクリプトのコーディング例	スクリプト拡張コマンドのエラー	シェル標準コマンドのエラー	
		続行できる場合	続行できない場合
`#-adsh_file JOBFILE jobfile`	×	－	－
`cmdA`	△	×	－
`shift $n`	△	○	×
`cmdB`	△	○	△
	－	－	－
`#-adsh_step_start NO -run normal`	△	△	△
`  echo "run normal step"`	△	△	△
`  cmdNormal`	△	△	△
`#-adsh_step_end`	△	△	△
	－	－	－
`#-adsh_step_start AB -run abnormal`	○	○	△
`  echo "run abnormal step"`	○	○	△
`  cmdAbnormal`	○	○	△
`#-adsh_step_end`	○	○	△
	－	－	－
`#-adsh_step_start AL -run always`	○	○	△
`  echo "run always step"`	○	○	△
`  cmdAlways`	○	○	△
`#-adsh_step_end`	○	○	△

（凡例）

○：実行します。

△：実行しません。

×：エラーになります。

－：該当しません。

注

空行の個所は何も指定していないことを示します。

(2) ジョブステップ正常ブロック内でエラーが発生した場合

ジョブステップ正常ブロック内でエラーが発生した場合の動作を次の表に示します。

表 5-44 ジョブステップ正常ブロック内でエラーが発生した場合の動作

発生したエラーの種類	後続のコマンド・制御文の動作
スクリプト拡張コマンドのエラー シェル標準コマンドのエラー（続行できる場合）※	• ジョブステップ正常ブロック内のすべてのコマンド・制御文は実行しません。 • ジョブステップエラーブロックが定義されている場合，ジョブステップエラーブロックを実行します。 • ジョブステップがエラー終了します。後続のコマンド・制御文の動作はジョブステップ外でスクリプト拡張コマンドのエラーが発生したときの動作と同一です。
シェル標準コマンドのエラー（続行できない場合）	ジョブ定義スクリプトの実行を終了します。

注※

onError 属性が cont の場合，シェル標準コマンドのエラー（続行できる場合）がジョブステップ正常ブロック内の最後のコマンドであるときに限り，ジョブステップ正常ブロック内でエラーが発生したと見なします。

ジョブステップ正常ブロック内でエラーが発生した場合の例を，次に示します。

表 5-45 ジョブステップ正常ブロック内でエラーが発生した場合の例

ジョブ定義スクリプトのコーディング例	スクリプト拡張コマンドのエラー	シェル標準コマンドのエラー		
		続行できる場合	続行できる場合（onError 属性が cont の場合）	続行できない場合
`#-adsh_step_start S1 -onError stop`	－	－	－	－
`#-adsh_file JOBFILE jobfile`	×	－	－	－
`cmdA`	△	×	×	－
`shift $n`	△	△	○	×
`cmdB`	△	△	○	△
`#-adsh_step_error`	○	○	△	△
`echo "step error block"`	○	○	△	△
`#-adsh_step_end`	○	○	○	△

ジョブ定義スクリプトのコーディング例	スクリプト拡張コマンドのエラー	シェル標準コマンドのエラー		
		続行できる場合	続行できる場合（onError 属性が cont の場合）	続行できない場合
	－	－	－	－
#-adsh_step_start NO -run normal	△	△	○	△
echo ”run normal step”	△	△	○	△
cmdNormal	△	△	○	△
#-adsh_step_end	△	△	○	△
	－	－	－	－
#-adsh_step_start AB -run abnormal	○	○	△	△
echo ”run abnormal step”	○	○	△	△
cmdAbnormal	○	○	△	△
#-adsh_step_end	○	○	△	△
	－	－	－	－
#-adsh_step_start AL -run always	○	○	○	△
echo ”run always step”	○	○	○	△
cmdAlways	○	○	○	△
#-adsh_step_end	○	○	○	△

（凡例）

○：実行します。

△：実行しません。

×：エラーになります。

－：該当しません。

注

空行の個所は何も指定していないことを示します。

(3)　ジョブステップエラーブロック内でエラーが発生した場合

ジョブステップエラーブロック内でエラーが発生した場合の，後続のコマンド・制御文の動作を次の表に示します。

表 5-46　ジョブステップエラーブロック内でエラーが発生した場合の動作

発生したエラーの種類	後続のコマンド・制御文の動作
スクリプト拡張コマンドのエラー	• ジョブステップエラーブロック内のすべてのコマンド・制御文を実行しません。

発生したエラーの種類	後続のコマンド・制御文の動作
スクリプト拡張コマンドのエラー	• ジョブステップがエラー終了します。後続のコマンド・制御文の動作はジョブステップ外でスクリプト拡張コマンドのエラーが発生したときの動作と同一です。
シェル標準コマンドのエラー（続行できる場合）	• ジョブステップエラーブロック内のすべてのコマンド・制御文を実行します。 • ジョブステップがエラー終了します。後続のコマンド・制御文の動作はステップ外でスクリプト拡張コマンドのエラーが発生したときの動作と同一です。
シェル標準コマンドのエラー（続行できない場合）	ジョブ定義スクリプトの実行を終了します。

ジョブステップエラーブロック内でエラーが発生した場合の例を，次に示します。

表 5-47　ジョブステップエラーブロック内でエラーが発生した場合の例

ジョブ定義スクリプトのコーディング例	スクリプト拡張コマンドのエラー	シェル標準コマンドのエラー	
		続行できる場合	続行できない場合
#-adsh_step_start S1 -onError stop	－	－	－
echo ”step normal block”	－	－	－
#-adsh_step_error	－	－	－
#-adsh_file JOBFILE jobfile	×	－	－
cmdA	△	×	－
shift $n	△	○	×
cmdB	△	○	△
#-adsh_step_end	○	○	△
	－	－	－
#-adsh_step_start NO -run normal	△	△	△
echo ”run normal step”	△	△	△
cmdNormal	△	△	△
#-adsh_step_end	△	△	△
	－	－	－
#-adsh_step_start AB -run abnormal	○	○	△
echo ”run abnormal step”	○	○	△
cmdAbnormal	○	○	△
#-adsh_step_end	○	○	△
	－	－	－
#-adsh_step_start AL -run always	○	○	△

ジョブ定義スクリプトのコーディング例	スクリプト拡張コマンドのエラー	シェル標準コマンドのエラー	
		続行できる場合	続行できない場合
echo ”run always step”	○	○	△
cmdAlways	○	○	△
#-adsh_step_end	○	○	△

（凡例）

○：実行します。

△：実行しません。

×：エラーになります。

－：該当しません。

注

空行の個所は何も指定していないことを示します。

(4) 注意事項

ジョブステップ外で続行できるシェル標準コマンドのエラーが発生した場合，run 属性が normal のジョブステップを除き，後続のコマンド・制御文をすべて実行します。この場合，ジョブの終了コードも後続のコマンド・制御文の終了コードで上書きされます。エラーの要因となった終了コードをジョブの終了コードに反映したい場合（JP1/AJS でジョブをエラーとしたい場合など）は，onError 属性に stop を指定したジョブステップ内にコマンド・制御文を記述してください。

5.8.11 コマンド実行結果の出力に関する注意事項

ジョブ実行ログファイルに出力されたコマンドの実行結果を確認するときは，次の点に注意してください。

(1) 別プロセスでコマンドグループ化した場合のコマンド実行結果の出力

「(」，「)」で囲んだコマンドグループ化によって実行したコマンド群は，コマンド群を 1 つのジョブ定義スクリプトとして別プロセスで動作します。この場合のコマンド実行結果は，1 つのジョブ定義スクリプトの実行結果として，次のどれかのメッセージが出力されます。

KNAX6540-I，KNAX6541-E，KNAX6542-E，KNAX6560-I，KNAX6561-E，KNAX6562-E

(2) バックグラウンド実行時の注意事項

「&」および「|&」を付与してバックグラウンド実行したコマンドの終了メッセージ出力の場合，次の点に注意してください。

- バックグラウンド実行のコマンドがすべて完了するのを待ってからジョブ終了するため，終了メッセージを必ず出力します。
- ジョブステップ内で起動したコマンドでも，コマンドの終了はジョブステップ終了後の可能性があります。その場合は，コマンドを起動したジョブステップの終了メッセージ出力後に，コマンドの終了メッセージを出力します。なお，「&」および「|&」を付与してバックグラウンド実行したコマンドの終了コードはジョブステップやジョブの終了コードには影響しません。
- バックグラウンド実行したコマンドのジョブ実行ログへの出力順序は，実際のプロセス開始順序や終了順序とは関係なく，順不同となります。「|」を使って連結したコマンド群についても同様です。
- バックグラウンド実行したコマンドの実行時間は，ジョブコントローラがコマンドの終了を検知した際に計測します。そのため，入力待ちなどによってジョブコントローラが停止している間にバックグラウンド実行したコマンドが終了した場合，出力される実行時間は，実際にかかった時間よりも長く表示されることがあります。

(3) builtin コマンド，command コマンド，eval コマンド，time コマンド，.（ドット）コマンド，exec コマンドの注意事項

それぞれのコマンドを実行したときの，実行結果の出力についての注意事項を次に示します。

- 組み込みコマンドの builtin コマンド，command コマンド，eval コマンド，time コマンド，exec コマンドの場合

  引数として指定したコマンドの実行結果だけを出力します。builtin コマンド，command コマンド，eval コマンド，time コマンド，および exec コマンド自身の実行結果は出力しません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用しません。

  command コマンドに対して，実行中のプラットフォームではサポートしないオプションを指定した場合は，command コマンドの実行結果を出力してジョブやジョブステップの正常終了またはエラー終了を判定します。
- 組み込みコマンドの.（ドット）コマンドの場合

  指定した外部スクリプト内の各コマンドの実行結果だけを出力します。

  .（ドット）コマンド自身は正常終了しますが，その実行結果は出力しません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用しません。

  指定した外部スクリプトがなく，.（ドット）コマンドがエラー終了した場合は，.（ドット）コマンドの実行結果を出力してジョブやジョブステップの正常終了またはエラー終了を判定します。

(4) コマンドの書式によるコマンド実行結果出力の違い

コマンドを次の書式で指定して実行した場合，変換前のコマンド名がコマンドの実行結果に出力されます。

- パイプ（|）による別プロセスの実行
- コマンド置換（$(), ``）による別プロセスの実行
- |&によるバックグラウンドプロセスの実行

- 単一コマンドのグループ化によるサブシェル実行
- &によるバックグラウンド実行

例を次に示します。

- 変数をコマンドとして実行した場合は，変数展開前の文字列がコマンド名として出力されます。

  ジョブ定義スクリプトの定義

  ```
  01: $CMD &
  ```

  実行結果の出力

  ```
  KNAX6116-I コマンド（$CMD，行番号=1）が正常終了しました。rc=0 E-Time=0.001s C-Time=0.000s
  ```

- 環境ファイルの CHILDJOB_PGM パラメーターに定義した規則に一致する文字列をコマンドとして実行した場合は，CHILDJOB_PGM パラメーターによって読み替えられる前の文字列がコマンド名として出力されます。

  環境ファイルの定義

  ```
  #-adsh_conf CHILDJOB_PGM /bin/sh
  ```

  ジョブ定義スクリプトの定義

  ```
  01: /bin/sh ./test.ash &
  ```

  実行結果の出力

  ```
  KNAX6116-I コマンド（/bin/sh，行番号=1）が正常終了しました。rc=0 E-Time=0.010s C-
  Time=0.000s
  ```

5.9　ファイルの割り当ておよび後処理をする

スクリプト拡張コマンドまたは adshfile コマンドを使用して，通常ファイル，一時ファイルおよびプログラム出力データファイルに対して，割り当ておよび後処理ができます。

割り当てとは，各コマンド実行時の動作であり，ジョブステップやジョブ終了時の各ファイルの扱いを登録したり，ファイル名称やファイル実体を生成したりすることをいいます。

後処理とは，ジョブステップやジョブ終了時の動作であり，割り当て時の定義に従って各ファイルを削除したり残したりすることをいいます。

スクリプト拡張コマンドについては，「9.5　スクリプト拡張コマンド」を参照してください。adshfile コマンドについては，「adshfile コマンド（通常ファイルの割り当ておよび後処理を指定する）」を参照してください。

5.9.1　通常ファイルの割り当ておよび後処理をする

#-adsh_file コマンド（スクリプト拡張コマンド）または adshfile コマンド（シェル運用コマンド）を使用して，次の作業を実施します。

- 該当するジョブステップまたはジョブの結果によって，割り当てたファイルの後処理をします。
- #-adsh_file コマンドの場合は，ジョブ，ジョブステップ，およびこれらから起動するコマンドで使用する通常ファイルについて，ファイルパスをシェル変数および環境変数に割り当てます。

#-adsh_file コマンドと adshfile コマンドの機能差異を次の表に示します。#-adsh_file コマンドについては「#-adsh_file コマンド（通常ファイルの割り当ておよび後処理を指定する）」を参照してください。adshfile コマンドについては「adshfile コマンド（通常ファイルの割り当ておよび後処理を指定する）」を参照してください。

機能	#-adsh_file コマンド	adshfile コマンド
環境変数へのファイルパス設定	○	×
ファイルの後処理のタイミング	• ステップ外の場合 ジョブ終了時 • ステップ内の場合 ステップ終了時	ジョブ終了時またはステップ終了時を指定
.コマンドで記述する外部スクリプトでの記述可否	×	○
繰り返し処理内での記述可否	×	○
関数内での記述可否	×	○

機能	#-adsh_file コマンド	adshfile コマンド
ファイル割り当て処理中にエラーとなった場合の割り当て済み通常ファイルの後処理	keep を仮定し，ファイルを削除しません。 詳細については「#-adsh_file コマンドの場合」を参照してください。	引数-a の指定に従います。 詳細については「adshfile コマンドの場合」を参照してください。

（凡例）

○：実行できます。

×：実行できません。

重要

adshfile コマンドと#-adsh_file コマンドで割り当てた通常ファイルは別々に管理され，後処理は adshfile コマンド，#-adsh_file コマンドの順に実行されます。そのため，両方のコマンドで同じファイルを割り当てると，ファイルの後処理が二重に実行されることになり，エラーが発生する場合があるので注意してください。

(1) 通常ファイルの割り当て

#-adsh_file コマンドでは，指定した通常ファイルのファイルのパスを，ファイル環境変数定義名と同名のシェル変数および環境変数に割り当てます。

#-adsh_file コマンドおよび adshfile コマンドは，指定した通常ファイルのファイルの実体は作成しません。

割り当て時にファイルが存在するかどうかを確認する場合は，#-adsh_file コマンドでは-chk に，adshfile コマンドでは-c に exist を指定します。exist が指定された場合，割り当て時にファイルが存在しなければエラーとなります。

ファイルが存在するかどうかに関係なく割り当てる場合は，#-adsh_file コマンドでは-chk に，adshfile コマンドでは-c に no を指定します。no が指定されたとき，ファイルが存在しなくてもエラーとしないで割り当てます。

#-adsh_file コマンドで，通常ファイル test1 を FILE に割り当てる場合の使用例を次に示します。

- Windows の場合

```
#-adsh_file FILE 'C:¥home¥test¥test1' -chk exist -normal keep -abnormal keep
```

- UNIX の場合

```
#-adsh_file FILE /home/test/test1 -chk exist -normal keep -abnormal keep
```

UNIX の場合の chk 属性の使用例を次に示します。

```
#-adsh_job FILE_CHK
#-adsh_step_start STEP01
  #-adsh_file FILE01 /home/test/test1 -chk no -normal keep -abnormal keep
```

```
  #-adsh_file FILE02 /home/test/test2 -chk exist -normal keep -abnormal keep
  cmdA ${FILE01} ${FILE02}
#-adsh_step_end
```

FILE01 の割り当てでは，割り当て時にファイルが存在するかどうかを確認しません。したがって，/home/test/test1 が存在しなくてもファイルを割り当てます。

FILE02 の割り当てでは，割り当て時にファイルが存在するかどうかを確認します。したがって，/home/test/test2 が存在しない場合，ファイル割り当て処理はエラーとなります。

(2) 通常ファイルの後処理

通常ファイルは，割り当てたジョブステップまたはジョブ終了時に後処理をします。#-adsh_file コマンドの場合，後処理ではファイルパスを設定したシェル変数および環境変数を設定前の値に戻します。また，ジョブステップやジョブの終了状態と，コマンドの指定値に従って次に示す処理をします。

ジョブステップおよびジョブが正常終了した場合，#-adsh_file コマンドの-normal または adshfile コマンドの-n の指定に従って後処理をします。指定値ごとの通常ファイルの後処理について次に示します。

- del を指定した場合，ジョブステップまたはジョブの終了時に通常ファイルを削除します。
- keep を指定した場合，ジョブステップまたはジョブの終了時に通常ファイルを削除しません。

ジョブステップまたはジョブがエラー終了した場合，#-adsh_file コマンドの-abnormal または adshfile コマンドの-a の指定に従って後処理をします。指定値ごとの通常ファイルの後処理について次に示します。

- del を指定した場合，ジョブステップまたはジョブの終了時に通常ファイルを削除します。
- keep を指定した場合，ジョブステップまたはジョブの終了時に通常ファイルを削除しません。

#-adsh_file コマンドに-abnormal，または adshfile コマンドに-a オプションを指定した場合の，後処理の扱いを次の表に示します。

ジョブステップやジョブがエラーとなる要因	#-adsh_file コマンドの割り当て済み通常ファイルの後処理	adshfile コマンドの割り当て済み通常ファイルの後処理
後続の#-adsh_file コマンドの通常ファイル割り当て処理中にエラー	無条件に keep が仮定される	各 adshfile コマンドの-a オプションに従う※ (keep は仮定されない)
後続の adshfile コマンドの通常ファイル割り当て処理中にエラー	各#-adsh_file コマンドの-abnormal オプションに従う	各 adshfile コマンドの-a オプションに従う
後続の#-adsh_temp コマンドの一時ファイル割り当て処理中にエラー	無条件に keep が仮定される	各 adshfile コマンドの-a オプションに従う※ (keep は仮定されない)
後続の#-adsh_spoolfile コマンドのプログラム出力データファイル割り当て処理中にエラー	無条件に keep が仮定される	各 adshfile コマンドの-a オプションに従う※ (keep は仮定されない)

ジョブステップやジョブがエラーとなる要因	#-adsh_file コマンドの割り当て済み通常ファイルの後処理	adshfile コマンドの割り当て済み通常ファイルの後処理
後続の上記以外のコマンドやプログラムなどのエラー	各#-adsh_file コマンドの-abnormal オプションに従う	各 adshfile コマンドの-a オプションに従う

注※

割り当てエラー発生時に無条件にファイルを保持するよう運用したい場合は，#-adsh_file コマンドを使用してください。

通常ファイルの割り当てではファイルの実体を作成しないため，後処理をするときに存在しないファイルは存在しない状態のままになります。

(a) #-adsh_file コマンドの場合

Windows での後処理の実行例を次に示します。

```
#-adsh_job JOB
#-adsh_file FILE01 'C:¥user¥file01' -chk exist -normal keep -abnormal del

#-adsh_step_start STEP
    #-adsh_file FILE02 'C:¥user¥file02' -chk exist -normal del -abnormal del
    #-adsh_file FILE03 'C:¥user¥file03' -chk exist -normal keep -abnormal del
    uap
#-adsh_step_end

#-adsh_file FILE04 'C:¥user¥file04' -chk exist -normal del -abnormal del
```

例 1

FILE03 の割り当てでエラーになった場合

- FILE01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。abnormal 属性の指定に従って通常ファイルを削除します。
- FILE02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。normal 属性および abnormal 属性の指定に関係なく keep を仮定し，通常ファイルは削除しません。
- FILE03 に割り当てようとした通常ファイルは，割り当てが完了していないため後処理をしません。
- FILE04 の割り当て処理は動作しません。

例 2

FILE04 の割り当てでエラーになった場合

- FILE01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。normal 属性および abnormal 属性の指定に関係なく keep を仮定し，通常ファイルは削除しません。
- FILE02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。normal 属性の指定に従って通常ファイルを削除します。
- FILE03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。normal 属性の指定に従って通常ファイルは削除しません。

- FILE04 に割り当てようとした通常ファイルは，割り当てが完了していないため後処理をしません。

例 3

uap でエラーになった場合

- FILE01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。abnormal 属性の指定に従って通常ファイルを削除します。
- FILE02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。abnormal 属性の指定に従って通常ファイルを削除します。
- FILE03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。abnormal 属性の指定に従って通常ファイルを削除します。
- FILE04 の割り当て処理は動作しません。

例 4

正常終了した場合

- FILE01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。normal 属性の指定に従って通常ファイルは削除しません。
- FILE02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。normal 属性の指定に従って通常ファイルを削除します。
- FILE03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。normal 属性の指定に従って通常ファイルは削除しません。
- FILE04 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。normal 属性の指定に従って通常ファイルを削除します。

(b) adshfile コマンドの場合

後処理の実行例を次に示します。この例では通常ファイルのファイル名を「file01_**実行日付**」～「file05_**実行日付**」とし，「**実行日付**」の部分にジョブの実行時刻を入れます。

```
#-adsh_job JOB

VAL01=file01_`date +%y%m%d`
VAL02=file02_`date +%y%m%d`
VAL03=file03_`date +%y%m%d`
VAL04=file04_`date +%y%m%d`
VAL05=file05_`date +%y%m%d`


adshfile -s job -n keep -a del ${VAL01}

#-adsh_step_start STEP
    adshfile -s step -n del -a del ${VAL02}
    adshfile -s step -n keep -a keep ${VAL03}
    adshfile -s step -n keep -a del ${VAL04}
    uap
#-adsh_step_end
```

```
adshfile -s job -n del -a del ${VAL05}
```

例 1

VAL04 の割り当てでエラーになった場合

- VAL01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。-a の指定に従って通常ファイルを削除します。
- VAL02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-a の指定に従って通常ファイルを削除します（#-adsh_file コマンドと異なり，keep は仮定されないで，-a オプションの指定に従って後処理をします）。
- VAL03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-a の指定に従って通常ファイルを削除しません。
- VAL04 はコマンドエラーとなり，割り当てようとした通常ファイルは，割り当てが完了していないため後処理をしません。
- VAL05 の割り当て処理は動作しません。

例 2

VAL05 の割り当てでエラーになった場合

- VAL01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。-a の指定に従って通常ファイルを削除します（#-adsh_file コマンドと異なり，keep は仮定されないで，-a オプションの指定に従って後処理をします）。
- VAL02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-n の指定に従って通常ファイルを削除します。
- VAL03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-n の指定に従って通常ファイルは削除しません。
- VAL04 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-n の指定に従って通常ファイルは削除しません。
- VAL05 はコマンドエラーとなり，割り当てようとした通常ファイルは，割り当てが完了していないため後処理をしません。

例 3

uap でエラーになった場合

- VAL01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。-a の指定に従って通常ファイルを削除します。
- VAL02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-a の指定に従って通常ファイルを削除します。
- VAL03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-a の指定に従って通常ファイルを削除しません。

- VAL04 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-a の指定に従って通常ファイルを削除します。
- VAL05 の割り当て処理は動作しません。

例 4

正常終了した場合

- VAL01 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。-n の指定に従って通常ファイルは削除しません。
- VAL02 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-n の指定に従って通常ファイルを削除します。
- VAL03 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-n の指定に従って通常ファイルは削除しません。
- VAL04 に割り当てた通常ファイルは，ジョブステップ終了時に後処理をします。-n の指定に従って通常ファイルは削除しません。
- VAL05 に割り当てた通常ファイルは，ジョブ終了時に後処理をします。-n の指定に従って通常ファイルを削除します。

(3) adshfile コマンドの指定例（繰り返し処理内に指定した場合）

UNIX で adshfile コマンドを繰り返し処理内に指定する例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=046257
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                    : 046257
   スプールジョブディレクトリパス  : /var/opt/jp1as/spool/046257/
   実行日付                        : 2015/10/30
   システム環境ファイルパス        :
   ジョブ環境ファイルパス          : sample.ase
   ホスト名                        : host01
 [Automatic Job Management Systemから渡された環境変数]
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
10:17:20 046257 KNAX0091-I LOOP ジョブが開始しました。
10:17:20 046257 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの
完了を待ちます。
10:17:20 046257 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
10:17:20 046257 KNAX6112-I コマンド（echo，行番号=3）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/bin/ls，行番号=4）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile，行番号=8）が正常終了しました。
rc=0 E-Time=0.002s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile，行番号=9）が正常終了しました。
rc=0 E-Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6112-I コマンド（echo，行番号=10）が正常終了しました。rc=0 E-
```

```
Time=0.000s C-Time=0.000s
10:17:20 046257 KNAX6112-I コマンド（echo, 行番号=11）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/bin/cat, 行番号=12）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=8）が正常終了しました。
rc=0 E-Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=9）が正常終了しました。
rc=0 E-Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6112-I コマンド（echo, 行番号=10）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:17:20 046257 KNAX6112-I コマンド（echo, 行番号=11）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/bin/cat, 行番号=12）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=8）が正常終了しました。
rc=0 E-Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=9）が正常終了しました。
rc=0 E-Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX6112-I コマンド（echo, 行番号=10）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:17:20 046257 KNAX6112-I コマンド（echo, 行番号=11）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:17:20 046257 KNAX6116-I コマンド（/bin/cat, 行番号=12）が正常終了しました。rc=0 E-
Time=0.001s C-Time=0.000s
10:17:20 046257 KNAX1890-I ファイル割り当て処理指定（”keep”）に従って解放しました。path=/
home/user001/test.txt
10:17:20 046257 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/output_test.txt
10:17:20 046257 KNAX1604-I 後処理指定値によってファイル（/home/user001/output_test.txt）を削
除しました。
10:17:20 046257 KNAX1890-I ファイル割り当て処理指定（”keep”）に従って解放しました。path=/
home/user001/test01.txt
10:17:20 046257 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/output_test01.txt
10:17:20 046257 KNAX1604-I 後処理指定値によってファイル（/home/user001/output_test01.txt）を
削除しました。
10:17:20 046257 KNAX1890-I ファイル割り当て処理指定（”keep”）に従って解放しました。path=/
home/user001/test02.txt
10:17:20 046257 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/output_test02.txt
10:17:20 046257 KNAX1604-I 後処理指定値によってファイル（/home/user001/output_test02.txt）を
削除しました。
10:17:20 046257 KNAX0098-I LOOP ジョブが終了しました。rc=0 E-Time=0.025s C-Time=0.010s

********   ジョブ定義スクリプトの内容     ********

***** /home/user001/Tloop.ash *****
0001 : #-adsh_job LOOP
0002 :
0003 :   echo -E ”<<< list >>>” >&2
0004 :   ls test* >&2
0005 :
0006 :   for MEMBER in test*
0007 :   do
0008 :     ${ADSH_DIR_BIN}adshfile -s job -n keep -a keep $MEMBER
0009 :     ${ADSH_DIR_BIN}adshfile -s job -n del -a del output_$MEMBER
0010 :     echo -E ”MEMBER=$MEMBER” >output_$MEMBER
```

```
0011 :     echo -E "<<< output_$MEMBER >>>" >&2
0012 :     cat output_$MEMBER >&2
0013 :   done

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容     ********
<<< list >>>
test.txt
test01.txt
test02.txt
<<< output_test.txt >>>
MEMBER=test.txt
<<< output_test01.txt >>>
MEMBER=test01.txt
<<< output_test02.txt >>>
MEMBER=test02.txt
KNAX0098-I LOOP ジョブが終了しました。rc=0 E-Time=0.025s C-Time=0.010s

******** ジョブステップの出力 ********
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/046257-LOOP/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

(4)　adshfile コマンドの指定例（関数内に指定した場合）

UNIX で adshfile コマンドを関数内に指定する例を次に示します。

```
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=046260
----------------------------------------------------------------
 Advanced Shell 11-00

 [ジョブ情報]
   ジョブ識別子                   : 046260
   スプールジョブディレクトリパス : /var/opt/jp1as/spool/046260/
   実行日付                       : 2015/10/30
   システム環境ファイルパス       :
   ジョブ環境ファイルパス         : sample.ase
   ホスト名                       : host01
 [Automatic Job Management Systemから渡された環境変数]
----------------------------------------------------------------
********  ジョブコントローラのメッセージ出力  ********
10:31:28 046260 KNAX0091-I FUNCTION ジョブが開始しました。
10:31:28 046260 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの
完了を待ちます。
10:31:28 046260 KNAX7902-I ジョブコントローラは，"端末入力モード"で動作します。
10:31:28 046260 KNAX0092-I FUNCTION.STEP001 ステップが開始しました。
10:31:28 046260 KNAX6112-I コマンド（echo，行番号=15）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile，行番号=4）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile，行番号=5）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cp，行番号=6）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.000s
```

```
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=7）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cat, 行番号=8）が正常終了しました。rc=0
E-Time=0.001s C-Time=0.000s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=15）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=4）が正常終了しました。
rc=0 E-Time=0.002s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=5）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cp, 行番号=6）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.000s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=7）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cat, 行番号=8）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.010s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=15）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=4）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=5）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cp, 行番号=6）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.010s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=7）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cat, 行番号=8）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.000s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=15）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=4）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/bin/adshfile, 行番号=5）が正常終了しました。
rc=0 E-Time=0.003s C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cp, 行番号=6）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.000s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=7）が正常終了しました。rc=0 E-Time=0.000s
C-Time=0.000s
10:31:28 046260 KNAX6116-I コマンド（/opt/jp1as/cmd/cat, 行番号=8）が正常終了しました。rc=0
E-Time=0.002s C-Time=0.010s
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/infile1
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/infile1）を削除しま
した。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/outfile1
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/outfile1）を削除しま
した。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/infile2
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/infile2）を削除しま
した。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/outfile2
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/outfile2）を削除しま
した。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/
home/user001/infile3
```

```
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/infile3）を削除しました。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/home/user001/outfile3
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/outfile3）を削除しました。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/home/user001/infile4
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/infile4）を削除しました。
10:31:28 046260 KNAX1890-I ファイル割り当て処理指定（”del”）に従って解放しました。path=/home/user001/outfile4
10:31:28 046260 KNAX1604-I 後処理指定値によってファイル（/home/user001/outfile4）を削除しました。
10:31:28 046260 KNAX6597-I FUNCTION.STEP001 ジョブステップが正常終了しました。rc=0 E-Time=0.055s C-Time=0.050s
10:31:28 046260 KNAX6112-I コマンド（echo, 行番号=21）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
10:31:28 046260 KNAX0098-I FUNCTION ジョブが終了しました。rc=0 E-Time=0.057s C-Time=0.050s

********   ジョブ定義スクリプトの内容     ********

***** /home/user001/Tfunction.ash *****
0001 : #-adsh_job FUNCTION
0002 :
0003 : myfunc(){
0004 :    ${ADSH_DIR_BIN}adshfile -s step -n del -a del -c exist $1
0005 :    ${ADSH_DIR_BIN}adshfile -s step -n del -a del $2
0006 :    ${ADSH_DIR_CMD}cp $1 $2
0007 :    echo -E ”<<< $2 >>>” >&2
0008 :    ${ADSH_DIR_CMD}cat $2 >&2
0009 : }
0010 :
0011 : #-adsh_step_start STEP001
0012 :
0013 :    for CNT in 1 2 3 4
0014 :    do
0015 :      echo -E ”dddd$CNT” >infile$CNT
0016 :      myfunc infile$CNT outfile$CNT
0017 :    done
0018 :
0019 : #-adsh_step_end
0020 :
0021 : echo -E ”JOB_FUNCTION_END” >&2

***** パス変換情報 *****

********   実行ジョブのSTDERRファイルの内容     ********
KNAX6597-I FUNCTION.STEP001 ジョブステップが正常終了しました。rc=0 E-Time=0.055s C-Time=0.050s
JOB_FUNCTION_END
KNAX0098-I FUNCTION ジョブが終了しました。rc=0 E-Time=0.057s C-Time=0.050s

******** ジョブステップの出力 ********
KNAX0719-I STEP ステップ番号=0001 ステップ名=STEP001 出力先=STDERR
<<< outfile1 >>>
dddd1
<<< outfile2 >>>
```

```
dddd2
<<< outfile3 >>>
dddd3
<<< outfile4 >>>
dddd4

KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/var/opt/jp1as/spool/046260-FUNCTION/"
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

5.9.2 一時ファイルの割り当ておよび後処理をする

#-adsh_file_temp コマンドを使用して，ジョブ定義スクリプト内で一時的に使用するファイルを生成し，ファイルパスをシェル変数および環境変数に割り当てます。割り当てた一時ファイルはジョブ終了時には削除します。

(1) 一時ファイルの割り当て

ジョブ定義スクリプト内で一時的に使用するファイルを作成し，指定されたファイル環境変数定義名と同名のシェル変数および環境変数にファイルパスを割り当てます。

一時ファイルの割り当てには，次の 2 つの方法があります。

- 新規に一時ファイルを作成して割り当てる場合
- 既存の一時ファイルを割り当てる場合

(a) 新規に一時ファイルを作成して割り当てる場合

chk 属性に create を指定します。作成するファイルは 0 バイトです。UNIX の場合，作成される一時ファイルのパーミッションは，ファイルの所有者（作成者）の部分だけ umask 値の指定に従い，グループおよびその他のユーザーのアクセス権限部分は常に 0 になります。Windows の場合は権限の指定はありません。

ジョブステップ内で割り当てた一時ファイルを後続ジョブステップでも使用する場合は，一時ファイル識別名を指定し normal 属性に keep を指定します。

ジョブステップ内で割り当てた一時ファイルを後続ジョブステップで使用しない場合または，ジョブステップ外に指定する場合は，normal 属性に del を指定します。

ジョブステップ外で一時ファイルを割り当てる場合，一時ファイル識別名の指定および normal 属性に keep の指定はできません。

(b) 既存の一時ファイルを割り当てる場合

chk 属性に exist を指定し，先行ジョブステップで指定した一時ファイル識別名を指定します。

先行ジョブステップで作成していない一時ファイルおよび先行ジョブステップの後処理で削除した一時ファイルの識別名は指定できません。

割り当てた一時ファイルを後続のジョブステップで使用する場合は，normal 属性に keep を指定します。

割り当てた一時ファイルを後続のジョブステップで使用しない場合は，normal 属性に del を指定します。

(2) 一時ファイルの後処理

一時ファイルは，割り当てたジョブステップまたはジョブ終了時に後処理をします。後処理では，ファイルパスを設定したシェル変数および環境変数を設定前の値に戻します。また，ジョブステップまたはジョブの終了状態および normal 属性の指定値に従って次に示す処理をします。

ジョブステップおよびジョブが正常終了した場合

- normal 属性に keep を指定した場合は，ジョブステップ終了時に一時ファイルを削除しません。normal 属性に keep を指定したあと，後続ジョブステップで一時ファイルを使用しない場合は，ジョブ終了時に削除します。
- normal 属性に del を指定した場合は，ジョブステップ終了時またはジョブ終了時に一時ファイルを削除します。

ジョブステップおよびジョブがエラー終了した場合

- normal 属性に keep を指定した場合は，ジョブステップ終了時に一時ファイルを削除しないで，ジョブ終了時に一時ファイルを削除します。
- normal 属性に del を指定した場合は，ジョブステップ終了時またはジョブ終了時に一時ファイルを削除します。

(3) 一時ファイルの名称

一時ファイルの名称は，Windows と UNIX とで異なります。それぞれで作成するファイル名称を次に示します。

Windows の場合

システム固定の文字列「ASH」，ディレクトリ内で一意の名称から作成します。

```
ASH一意の名称.tmp
```

UNIX の場合

一時ファイルであることを示す文字列「TEMP」，ジョブ名，一時ファイル識別名とディレクトリ内で一意の名称から作成します。

- 一時ファイル識別名を指定した場合の一時ファイル名

  TEMP_**ジョブ通し番号**_**ジョブ名**_**一時ファイル識別名**_**一意の名称**

- 一時ファイル識別名を省略した場合の一時ファイル名

  TEMP_**ジョブ通し番号**_**ジョブ名**_**一意の名称**

(4) 格納ディレクトリ

一時ファイルを作成するディレクトリは，環境設定パラメーターの TEMP_FILE_DIR パラメーターで指定します。環境設定パラメーターに指定がない場合は，TEMP_FILE_DIR パラメーターの仮定値となります。TEMP_FILE_DIR パラメーターの詳細については，「7. 環境ファイルで設定するパラメーター」の「TEMP_FILE_DIR パラメーター（一時ファイルディレクトリのパス名を定義する）」を参照してください。

(5) 一時ファイルの使用例

一時ファイルを割り当てた場合の使用例を次に示します。

- ジョブステップ内で割り当てた一時ファイルを後続ジョブステップで使用しない場合，またはジョブステップ外で一時ファイルを割り当てる場合

```
#-adsh_file_temp TEMP1 -chk create -normal del
     echo "test" > ${TEMP1}
```

- 先行ジョブステップ内で作成した一時ファイルを後続ジョブステップで使用する場合

```
#-adsh_step_start STEP1
#-adsh_file_temp TEMP1 -id TEST1 -chk create -normal keep    →1.
echo "test1" > ${TEMP1}
#-adsh_step_end

#-adsh_step_start STEP2
#-adsh_file_temp TEMP2 -id TEST1 -chk exist -normal keep     →2.
echo "test2" >> ${TEMP2}
#-adsh_step_end

#-adsh_step_start STEP3
#-adsh_file_temp TEMP3 -id TEST2 -chk create -normal keep    →3.
echo "test3" >> ${TEMP3}
#-adsh_step_end

#-adsh_step_start STEP4
#-adsh_file_temp TEMP4 -id TEST1 -chk exist -normal del      →4.
echo "test4" >> ${TEMP4}
#-adsh_step_end

#-adsh_step_start STEP5
#-adsh_file_temp TEMP5 -id TEST2 -chk exist -normal del      →5.
echo "test5" >> ${TEMP5}
#-adsh_step_end
```

1. 後続ジョブステップで使用できる一時ファイルを作成し割り当てる。一時ファイル識別名には TEST1 を指定する。
2. 1.で作成した一時ファイル（識別名：TEST1）を割り当てる。割り当てた一時ファイルは，後続ジョブステップで使用可能とする。
3. 後続ジョブステップで使用できる一時ファイルを作成し割り当てる。一時ファイル識別名には TEST2 を指定する。

4. 2.で使用した一時ファイル（識別名：TEST1）を割り当てる。割り当てた一時ファイルは，ジョブステップ（ジョブステップ名：STEP4）終了時に削除する。
5. 3.で作成した一時ファイル（識別名：TEST2）を割り当てる。割り当てた一時ファイルは，ジョブステップ（ジョブステップ名：STEP5）終了時に削除する。

(6) プログラムの入力ファイルをジョブ定義スクリプトに記述するための一時ファイル利用方法

ユーザープログラムのパラメーターを一時ファイルに記述して標準入力とする場合，パラメーターをジョブ定義スクリプト内に直接記述して，一時ファイルを自動的に作成・削除できます。使用例を次に示します。

```
#-adsh_step_start
#-adsh_file_temp SYSIN -id parmfile -chk create -normal keep
cat << @@@ > ${SYSIN}
-in /files/indata
-out /files/outdata
-work /tmp
@@@
uap  ${SYSIN}
#-adsh_step_end
```

一時ファイルを作成し，そこにヒアドキュメントを用いてジョブ定義スクリプトに記述した複数行文字列を書き込みます。ユーザープログラムは一時ファイルを標準入力として実行し，実行完了後に一時ファイルを削除します。

5.9.3 プログラム出力データファイルの割り当てをする

ユーザープログラムの出力結果をジョブ実行ログと同様に一元管理するため，ジョブコントローラは実行結果の出力用ファイルを自動的に作成します。このファイルをプログラム出力データファイルといいます。

#-adsh_spoolfile コマンドを使用して，ユーザープログラムが実行結果の出力に使用するプログラム出力データファイルのファイルパスを自動生成し，シェル変数および環境変数に割り当てます。

(1) プログラム出力データファイルの割り当て

プログラム出力データファイルのファイルパスを自動生成し，指定されたファイル環境変数定義名と同名のシェル変数および環境変数に割り当てます。これらの変数は，ジョブステップまたはジョブ終了時に割り当て前の値に戻します。ファイルの実体は作成しません。

(2) プログラム出力データファイルの名称

プログラム出力データファイルの名称は，ジョブ名またはジョブステップ名や番号，#-adsh_spoolfile コマンドで指定されたファイル環境変数定義名などから，ジョブ定義スクリプト内のファイル定義ごとに一意の名称とします。

割り当てるプログラム出力データファイルの格納ディレクトリは，環境設定パラメーターの SPOOL_DIR パラメーターに指定されたスプールルートディレクトリ内の，ジョブごとのディレクトリとなります。環境設定パラメーターに指定がない場合は，スプールルートディレクトリは SPOOL_DIR パラメーターの仮定値となります。SPOOL_DIR パラメーターの詳細については，「7. 環境ファイルで設定するパラメーター」を参照してください。

プログラム出力データファイル名の形式を次に示します。Windows の場合，次のファイル名の後ろに「.sysout」の拡張子が付きます。

- ジョブステップ外で割り当てた場合

  0000_ジョブ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

- ジョブステップ内で割り当てた場合

  ステップ番号_ステップ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

なお，ルートジョブの環境ファイルの SPOOLJOB_CHILDJOB パラメーターに MERGE（子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージする）を指定して実行された子孫ジョブの場合は，次のファイル名になります。

- ジョブステップ外で割り当てた場合

  C子孫ジョブ起動順序通し番号_0000_ジョブ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

- ジョブステップ内で割り当てた場合

  C子孫ジョブ起動順序通し番号_ステップ番号_ステップ名_ファイル環境変数定義名通し番号_ファイル環境変数定義名

プログラム出力データファイル名の可変部分には次の内容が出力されます。

ステップ番号

すべてのジョブステップの通し番号です。ステップ番号は 4 桁の 10 進数で，先頭ジョブステップのステップ番号は 1 です。

例： 0001，0034，4095

ジョブ名

#-adsh_job コマンドで指定したジョブ名です。可変長で，最大 8 バイトです。指定した文字列が 8 バイトを超える場合は，最初の 8 バイトをジョブ名とします。

ステップ名

#-adsh_step_start コマンドで指定したジョブステップ名です。可変長で，最大 8 バイトです。指定した文字列が 8 バイトを超える場合は，最初の 8 バイトをジョブステップ名とします。

ファイル環境変数定義名通し番号

ジョブステップ外で割り当てたプログラム出力データファイルの連番，または各ジョブステップ内で割り当てたプログラム出力データファイルの連番です。ファイル環境変数定義名通し番号は 3 桁の 10 進数です。ジョブステップ外および各ジョブステップ内で 1 から 255 の値になります。

例：　001，034，255

ファイル環境変数定義名

#-adsh_spoolfile コマンドで指定したファイル環境変数定義名です。

子孫ジョブ起動順序通し番号

1 個のルートジョブから起動された子孫ジョブの起動順序を示す連番です。7 桁の 10 進数で，0000001 から 9999999 までの値です。

(3)　プログラム出力データファイルの使用例

次のジョブ定義スクリプトを実行した場合を例に，プログラム出力データファイルの作成結果を説明します。

```
#-adsh_job JOBSAMPLE001

#-adsh_spoolfile SYS001                        →1.
#-adsh_spoolfile SYS002                        →2.
echo ”----- job01 --------” 1>&2
echo ”SYS001” > $SYS001
echo ”SYS002” > $SYS002

#-adsh_step_start STEP01
    #-adsh_spoolfile SYS001                    →3.
    #-adsh_spoolfile SYS002                    →4.
    echo ”----- Step001 --------” 1>&2
    echo ”SYS001” > $SYS001
    echo ”SYS002” > $SYS002
#-adsh_step_end

#-adsh_spoolfile SYS001                         →5.
#-adsh_spoolfile SYS002                         →6.
echo ”----- job02 --------” 1>&2
echo ”SYS001” > $SYS001
echo ”SYS002” > $SYS002

#-adsh_step_start STEP02
    #-adsh_spoolfile SYS001                       →7.
    #-adsh_spoolfile SYS002                       →8.
    echo ”----- Step002 --------” 1>&2
    echo ”SYS001” > $SYS001
    echo ”SYS002” > $SYS002
#-adsh_step_end

#-adsh_spoolfile SYS001                           →9.
#-adsh_spoolfile SYS002                           →10.
echo ”----- job03 --------” 1>&2
echo ”SYS001” > $SYS001
echo ”SYS002” > $SYS002
```

このジョブ定義スクリプトを実行した場合，右端に付加した番号に応じて，次のファイル名のプログラム出力データファイルが作成されます。

Windows の場合

```
1.      0000_JOBSAMPL_001_SYS001.sysout
2.      0000_JOBSAMPL_002_SYS002.sysout
3.      0001_STEP01_001_SYS001.sysout
4.      0001_STEP01_002_SYS002.sysout
5.      0000_JOBSAMPL_003_SYS001.sysout
6.      0000_JOBSAMPL_004_SYS002.sysout
7.      0002_STEP02_001_SYS001.sysout
8.      0002_STEP02_002_SYS002.sysout
9.      0000_JOBSAMPL_005_SYS001.sysout
10.     0000_JOBSAMPL_006_SYS002.sysout
```

UNIX の場合

```
1.      0000_JOBSAMPL_001_SYS001
2.      0000_JOBSAMPL_002_SYS002
3.      0001_STEP01_001_SYS001
4.      0001_STEP01_002_SYS002
5.      0000_JOBSAMPL_003_SYS001
6.      0000_JOBSAMPL_004_SYS002
7.      0002_STEP02_001_SYS001
8.      0002_STEP02_002_SYS002
9.      0000_JOBSAMPL_005_SYS001
10.     0000_JOBSAMPL_006_SYS002
```

5.10　シェル変数の値を変換する

adshvarconv コマンドでシェル変数の値を変換できます。

5.10.1　パス変換ルールによる変換

シェル変数の値を PATH_CONV_ENABLE パラメーター，PATH_CONV パラメーターに従って変換します。変換ではあらかじめ環境変数に設定しているパス名やジョブ定義スクリプトの引数に格納されたパス名などを変換できます。

- 環境ファイル

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /home d:¥¥home
```

- ジョブ定義スクリプト（script.ash）

```
infile=$1
adshvarconv -p infile
 “${ADSH_DIR_CMD}cat”  ${infile}
```

- ファイル(d:¥home¥user001¥test.txt)

```
This is test data.
```

- 実行例

```
D:¥home¥user01> adshexec -m MINIMUM script.ash /home/user001/test.txt
This is test data.
```

5.10.2　文字列による変換

シェル変数の値の文字列置換ができます。これによって，変換ルールの事前定義が困難なパス名の置換ができます。

- ジョブ定義スクリプト（script.ash）

```
while read LINE
do
  adshvarconv -b ”/home/user1” -a ”$1” LINE
  adshvarconv -b ”/” -a ”¥¥” LINE
  echo -E ”$LINE” >&2
done  < input.txt
```

- ファイル（input.txt）

```
/home/user1/data001
/home/user1/data002
```

- 実行例

```
D:¥home¥user001>adshexec -m MINIMUM script.ash D:¥home¥winuser001
D:¥home¥winuser001¥data001
D:¥home¥winuser001¥data002
```

5.10.3　¥の増加

シェル変数の値に含まれる¥の数を増加することができます。これによって，awk コマンドに渡した値の¥が多重的に削除されることに，容易に対応できます。

- ジョブ定義スクリプト（script.ash）

```
aa=d:¥¥g1234z¥¥azzzz
echo -E 'hitachi' | "${ADSH_DIR_CMD}awk" -v VVV1=$aa '/hitachi/ { ¥
print "VVV1=" VVV1 ; ¥
}' >&2
adshvarconv -i 1 aa
echo -E 'hitachi' | "${ADSH_DIR_CMD}awk" -v VVV1=$aa '/hitachi/ { ¥
print "VVV1=" VVV1 ; ¥
}' >&2
```

- 実行例

```
D:¥home¥user001>adshexec -m MINIMUM script.ash
VVV1=d:g1234zzzzz
VVV1=d:¥g1234z¥azzzz
```

5.10.4　変数の値のコード変換

シェル変数の値をコード変換できます。これによって，UTF-8 で記述された値を変数に格納し，コード変換してスクリプト上で利用することができます。

- Shift-JIS で記述されたジョブ定義スクリプト（script.ash）

```
"${ADSH_DIR_CMD}rm" -f outfile.txt
while read LINE
do
  adshvarconv -e UTF8 SJIS  LINE
  LINE='駅名='$LINE
  adshvarconv -e SJIS UTF8  LINE
  echo -E "$LINE"  >>outfile.txt
done  < input.txt
```

- UTF-8 で記述したファイル（input.txt）

```
東京
京都
```

- 実行例

```
D:¥home¥user001>adshexec -m MINIMUM script.ash
```

- 作成された UTF-8 のファイル（outfile.txt）

```
駅名=東京
駅名=京都
```

5.11　ジョブ定義スクリプトファイルの記述例

ジョブ定義スクリプトファイルの記述例を次に示します。

```
#!/opt/jp1as/bin/adshexec        # ジョブ全体の制御 # →1.
#-adsh_job SAMPLE_JOB
####################
# ジョブステップの終了コードが8以上の場合，ジョブ打ち切り
#-adsh_job_stop 8:
# ジョブ内で使用する一時ファイル
#-adsh_file_temp JOBTEMP
####################
# ジョブステップ1 #                                    →2.
####################
#-adsh_step_start S1
# 入出力ファイル定義
#-adsh_file INFILE /files/infile -chk exist
#-adsh_file OUTFILE /files/outfile
#-adsh -chk no -normal keep -abnormal del
# パラメーターファイル定義
#-adsh_file_temp PARMFILE -id param
cat<<@@@>${PARMFILE}
-in ${INFILE}
-out ${OUTFILE}
-work /tmp
@@@
# ユーザープログラム実行
s1uap ${PARMFILE}
#-adsh_step_error
# プログラムエラー時の処理
recovery_uap ${JTMP}
#-adsh_step_end
####################
# ジョブステップ2 #                                    →3.
####################
if [[ $ADSH_STEPRC_S1= -eq 0 ]]; then
# 先行ジョブステップが正常の場合だけ実行
#-adsh_step_start S2 -onError cont -stepVar PATH
PATH=/s2bin:$PATH
#-adsh_rc_ignore s2uap
echo "s2uap1"
echo "s2uap2 parm1"
#-adsh_step_end
fi
####################
# ジョブステップ3 #                                    →4.
####################
#-adsh_step_start Java_STEP -run normal
adshjava -grp GROUPA -java javaAP prm1
#-adsh_step_end
```

例の右側に付加した番号は，次に示す説明の順番と対応しています。

1. ジョブ全体の制御

ジョブステップに制御を渡したり，ジョブを終了したりする。

2. ジョブステップ 1 の処理
   - 入出力ファイル定義
   - パラメーターファイル定義
   - ユーザープログラムの実行
   - エラー時の処理
3. ジョブステップ 2 の処理
   ジョブステップ 1 が正常時の場合の処理。

6 ジョブ定義スクリプトのデバッグ

この章では，JP1/Advanced Shell のデバッガ機能について説明します。

6.1　デバッガとは

デバッガとは，プログラムのデバッグを支援するツールです。JP1/Advanced Shell では，ジョブ定義スクリプトファイルに対するデバッガを利用できます。デバッガは，Windows の開発環境での GUI および UNIX の実行環境での CUI で使用できます。GUI では JP1/Advanced Shell エディタのボタン，メニューおよびショートカットキーを使用して，CUI ではデバッガのコマンドを使用して，デバッガから応答を得ることで，対話的にジョブ定義スクリプトファイルをデバッグできます。

デバッガを利用すると次のことができます。

- デバッガの起動
- デバッガの終了
- ジョブ定義スクリプトの実行
- ジョブ定義スクリプトの終了
- ジョブ定義スクリプトの実行停止
- ジョブ定義スクリプトの実行再開
- ジョブ定義スクリプトの情報表示※
- 変数の設定・表示
- バックトレースの表示※
- ソースファイルの表示※
- ディレクトリの移動※
- ログインシェルの起動※
- エラーの注入
- ヘルプの表示

注※

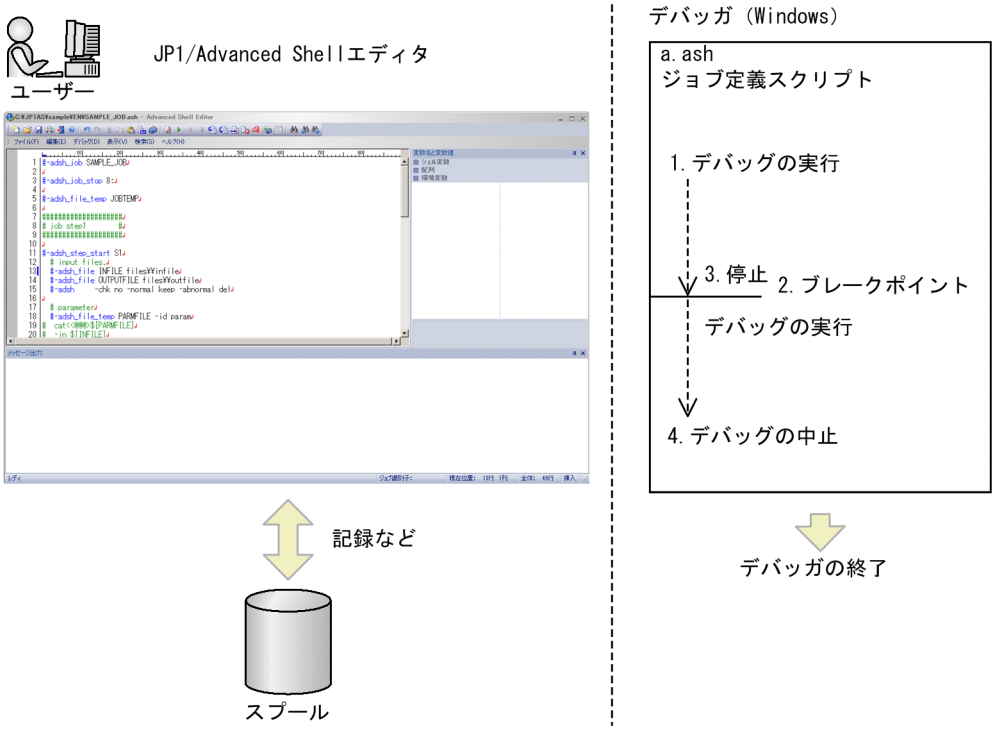
CUI でのデバッガの場合に利用できます。

6.1.1　GUI でのデバッグ【Windows 限定】

JP1/Advanced Shell エディタから GUI 操作でデバッグできます。

デバッグの実行の概要について次の図に示します。

図 6-1　デバッグの実行の概要（GUI の場合）



1. エディタからジョブ定義スクリプトをデバッグします。
2. ブレークポイントを設定します。
3. ジョブ定義スクリプトを実行するとブレークポイントで停止します。
4. デバッグを中止します。

(1)　出力

デバッグ実行時，ジョブ定義スクリプトを対話的に実行するため，標準出力および標準エラー出力は実行に合わせてコンソールにタイムリーに表示されます。通常実行時のように実行終了後に出力しません。ただし，エラーメッセージはエラーウィンドウにも表示されます。エラーウィンドウについては，「4.7.5 メッセージ出力ウィンドウ」を参照してください。また，スプールジョブディレクトリには標準出力および標準エラー出力のファイルを作成しません。

通常実行時はジョブ定義スクリプト完了後にジョブ実行ログを標準エラー出力に出力していますが，デバッグ実行時は実行に合わせてジョブ実行ログ相当の情報を標準エラー出力に出力しています。

(2)　情報の初期化

ジョブ定義スクリプトを一度実行したあとに，再びジョブ定義スクリプトを実行すると，前の実行で設定したシェル変数および環境変数の情報が初期化されます。

(3) スプール

ジョブ定義スクリプトをデバッグ実行するたびにスプールジョブフォルダを作成し，次のファイルを格納します。

- スクリプトイメージ：実行したスクリプトの内容
- ジョブ実行ログ：ジョブコントローラのメッセージ
- 出力ファイル：#-adsh_spoolfile コマンドの実行で作成したファイル
- sysout 管理ファイル（.sysout）

(4) 注意事項

「[[条件式]]」で条件判定を行った場合，実行結果のメッセージに含まれる E-Time に，デバッガの処理時間が含まれることがあります。

6.1.2 CUI でのデバッグ【UNIX 限定】

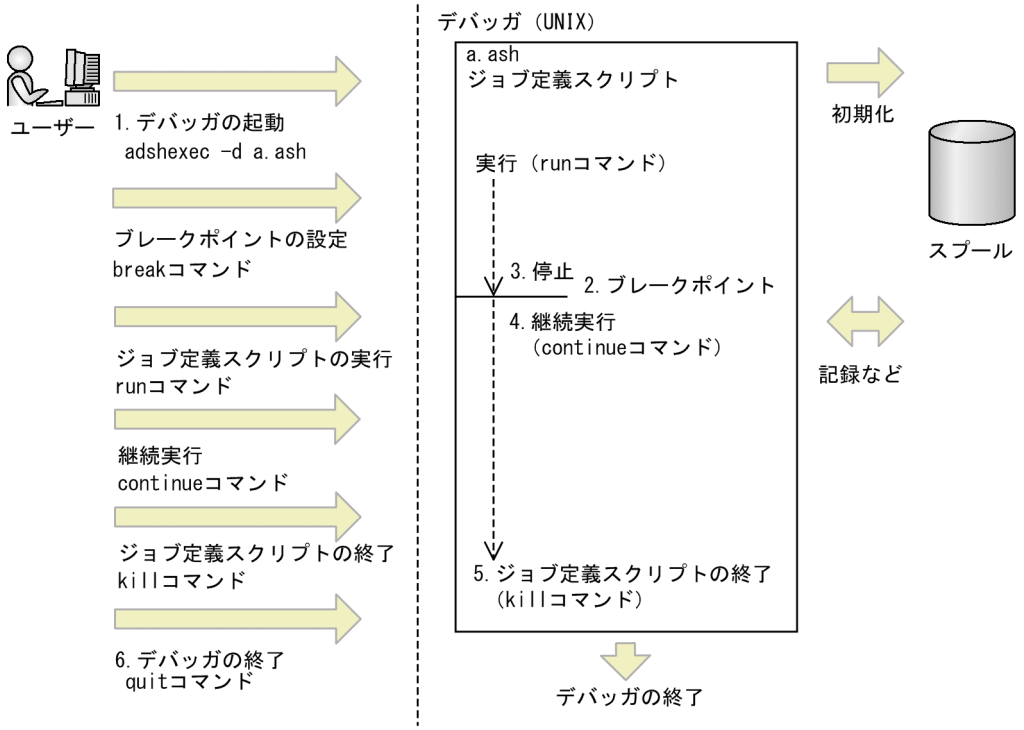
adshexec コマンドに-d オプションを指定して実行すると，ジョブコントローラをデバッガモードで起動し，CUI 操作でデバッグできます。実行環境からコマンドを使用してバッチジョブをデバッグするためには，次のように adshexec コマンドを使用します。UNIX のシェルからコマンドを入力します。

```
adshexec -d /script/batchjob2.ash
```

adshexec コマンドの詳細については，「8.3 シェル運用コマンド」の「adshexec コマンド（バッチジョブを実行する）」を参照してください。

デバッグの実行の概要について次の図に示します。

図 6-2　デバッグの実行の概要（CUI の場合）



1. adshexec コマンドの-d オプションを入力すると，デバッガを起動します。
2. break コマンドを入力すると，ブレークポイントが設定されます。
3. run コマンドを入力すると，ジョブ定義スクリプトを実行してブレークポイントで停止します。
4. continue コマンドを入力すると，ブレークポイントから継続実行されます。
5. kill コマンドを入力すると，ジョブ定義スクリプトを終了します。
6. quit コマンドを入力すると，デバッガを終了します。

(1)　出力

デバッグ実行時，ジョブ定義スクリプトを対話的に実行するため，標準出力および標準エラー出力は実行に合わせてタイムリーに表示されます。通常実行時のように実行終了後に出力しません。また，スプールジョブディレクトリには標準出力および標準エラー出力のファイルを作成しません。

通常実行時はジョブ定義スクリプト完了後にジョブ実行ログを標準エラー出力に出力していますが，デバッグ実行時は実行に合わせてジョブ実行ログ相当の情報を標準エラー出力に出力しています。

(2)　情報の初期化

run コマンドでジョブ定義スクリプトを一度実行したあとに，再び run コマンドでジョブ定義スクリプトを実行すると，前回実行時の設定情報のうち，次に示す情報が初期化されます。

- シェル変数
- 環境変数

- エラー注入モード

また，次に示す情報はデバッガを終了するまで引き継がれます。

- ブレークポイントおよびウォッチポイントの情報
- デバッガの作業ディレクトリパス
- ファイル※

注※

スクリプト拡張コマンドで作成したファイルについては，該当する後処理に従います。

(3) スプール

CUI デバッグ実行では，デバッガと run コマンドで実行したジョブ定義スクリプトの 2 種類のスプールジョブディレクトリを作成します。一度のデバッグ実行でデバッガのスプールジョブディレクトリは 1 つ，ジョブ定義スクリプトのスプールジョブディレクトリは run コマンドを実行した回数分作成します。デバッガとジョブ定義スクリプトのスプールジョブディレクトリについて説明します。

(a) デバッガ

デバッガではジョブ定義スクリプトを実行しません。一度のデバッグで実行したジョブ定義スクリプトの数を管理ファイルに格納したり，デバッガの内部データを記述したファイルを格納したりするために，スプールジョブディレクトリを作成します。

デバッガのスプールジョブディレクトリには，次のファイルを格納します。

- スクリプトイメージ：実行したスクリプトの内容
- ジョブ実行ログ：デバッガのメッセージ（生成したプロセスの pid など）
- ブレークポイント情報（.DBG）：デバッガの内部データを記述する
- sysout 管理ファイル（.sysout）

(b) ジョブ定義スクリプト

run コマンドを実行するたびにスプールジョブディレクトリを作成し，次のファイルを格納します。

- スクリプトイメージ：実行したスクリプトの内容スクリプトイメージ
- ジョブ実行ログ：ジョブコントローラのメッセージ
- 出力ファイル：#-adsh_spoolfile コマンドの実行で作成したファイル
- sysout 管理ファイル（.sysout）

(4) 注意事項

「[[条件式]]」で条件判定を行った場合，またはコマンド間をパイプで連結した場合は，実行結果のメッセージに含まれる E-Time に，デバッガの処理時間が含まれることがあります。

6.1.3 GUI デバッガの機能一覧【Windows 限定】

GUI デバッガの機能と，このマニュアルでの参照先を次の表に示します。

表 6-1 GUI デバッガの機能

機能	マニュアルの参照先
ジョブ定義スクリプトを実行する	4.4.6 デバッグをする
デバッグを中止する	(2) デバッグを実行・中止する
ジョブ定義スクリプトを停止する	(d) 関数の終わりまで実行する場合
ブレークポイントを設定する	(1) デバッグ実行時のブレークポイントを設定・解除する
ブレークポイントを解除する	(1) デバッグ実行時のブレークポイントを設定・解除する
逐次実行する	(b) 1 行ずつ実行（関数の中もステップ実行）する場合，(c) 1 行ずつ実行（関数の中はステップ実行しない）する場合
継続実行する	(a) ブレークポイントまで実行する場合
関数を実行する	(d) 関数の終わりまで実行する場合
変数の値を設定または表示する	4.7.6 変数ウィンドウ
ステータスを表示する	(1) ツールバー
エラー注入モードを変更する	(1) ツールバー
トラップアクションを実行する	(5) trap コマンドのアクションを実行する

6.1.4　デバッガのコマンド一覧【UNIX 限定】

デバッガのコマンドをアルファベット順で次の表に示します。また，各コマンドの短縮形と，このマニュアルでの参照先を示します。

表 6-2　デバッガコマンドの一覧

コマンド名	機能	コマンドの短縮形	マニュアルの参照先
break	ブレークポイントを設定します。	b	6.2.4　ブレークポイントを設定する（break コマンド）
cd	ディレクトリを移動します。	cd	6.2.26　ディレクトリを移動する（cd コマンド）
continue	継続実行をします。	c	6.2.9　継続実行をする（continue コマンド）
delete	ブレークポイント・ウォッチポイントを削除します。	d	6.2.6　ブレークポイント・ウォッチポイントを削除する（delete コマンド）
exec	ログインシェルを起動します。	ex	6.2.27　ログインシェルを起動する（exec コマンド）
finish	関数を実行します。	f	6.2.10　関数を実行する（finish コマンド）
help	ヘルプを表示します。	h	6.2.28　ヘルプを表示する（help コマンド）
info breakpoints	ブレークポイントとウォッチポイントの情報を表示します。	i b	6.2.13　ブレークポイント・ウォッチポイントの情報を表示する（info breakpoints コマンド）
info coverage	デバッグ途中のカバレージ情報を表示します。	i c	6.2.14　カバレージ情報を表示する（info coverage コマンド）
info functions	関数情報を表示します。	i f	6.2.15　関数情報を表示する（info functions コマンド）
info jobsteps	ジョブステップ情報を表示します。	i j	6.2.16　ジョブステップ情報を表示する（info jobsteps コマンド）

コマンド名	機能	コマンドの短縮形	マニュアルの参照先
info pathvars	変数名がパス名を扱う変数であるかどうかを表示します。	i p	6.2.17　パスを扱う変数名情報を表示する（info pathvars コマンド）
info signals	シグナル情報を表示します。	i si	6.2.18　シグナル情報を表示する（info signals コマンド）
info status	ステータスを表示します。	i st	6.2.19　ステータスを表示する（info status コマンド）
info variables	変数情報を表示します。	i v	6.2.20　シェル変数情報を表示する（info variables コマンド）
joberrmode	エラー注入モードの有効/無効を設定します。	jem	6.2.21　エラー注入モードの有効/無効を設定する（joberrmode コマンド）
kill	ジョブ定義スクリプトを終了します。	k	6.2.3　ジョブ定義スクリプトを終了する（kill コマンド）
list	ソースファイルを表示します。	l	6.2.25　ソースファイルを表示する（list コマンド）
next	関数内で停止しないで逐次実行をします。	n	6.2.8　逐次実行をする（step コマンド，next コマンド）
print	変数の値を表示します。	p	6.2.23　変数の値を表示する（print コマンド）
quit	デバッガを終了します。	q	6.2.1　デバッガを終了する（quit コマンド）
return	関数を終了します。	ret	6.2.11　関数を終了する（return コマンド）
run	ジョブ定義スクリプトを実行します。	r	6.2.2　ジョブ定義スクリプトを実行する（run コマンド）
set	変数の値を設定します。	set	6.2.22　変数の値を設定する（set コマンド）

コマンド名	機能	コマンドの短縮形	マニュアルの参照先
signal	シグナルを送信します。	si	6.2.12　シグナルを送信する（signal コマンド）
step	関数内も含んで逐次実行をします。	s	6.2.8　逐次実行をする（step コマンド，next コマンド）
watch	ウォッチポイントを設定します。	wa	6.2.5　ウォッチポイントを設定する（watch コマンド）
where	バックトレースを表示します。	whe	6.2.24　バックトレースを表示する（where コマンド）

デバッガコマンドを使用する場合，次の点に注意してください。

- 上の表にないコマンド名またはコマンドの短縮形をコマンドとして入力しないでください。
- 引数を指定できるコマンドで，引数の個数の上限を超えて指定しないでください。
- 引数を指定できないコマンドに引数を指定しないでください。
- デバッガの標準入力への入力文字数の上限は 4,094 バイトです。それ以上の文字を入力しないでください。

6.1.5　ジョブ定義スクリプトの構成要素に対する停止可否

ジョブ定義スクリプトの各要素に対して，停止できるかどうかを次の表に示します。次の表で示す要素では，ブレークポイントや逐次実行などによる停止ができます。

表 6-3　ジョブ定義スクリプトの各要素に対しての停止可否

ジョブ定義スクリプトで使用する要素	内容	停止の可否
for 文	for 文であることを示します。	○
case 文	case 文であることを示します。	○
if 文	if 文であることを示します。	○
while 文	while 文であることを示します。	○
until 文	until 文であることを示します。	○
elif 文	elif 文であることを示します。	○
else 文	else 文であることを示します。	×

ジョブ定義スクリプトで使用する要素	内容	停止の可否
case のパターン文	case 文に指定するパターン文であることを示します。パターン文の内部の処理は含みません。	×
条件文の終端	if 文，elif 文の then，または for 文，while 文，until 文の do など，条件文の終端を示します。	×
ブロック終端	for 文，while 文，until 文の done，case 文の esac，if 文の fi など，ブロック終端を示します。	×
関数定義開始	関数定義開始を示します。	×
関数定義終了	関数定義終了を示します。	×
関数実行開始	関数が開始することを示します。	○
関数実行終了	関数が終了することを示します。	×
スクリプト拡張コマンド	#-adsh で始まるスクリプト拡張コマンドを示します。	○
シェル標準コマンド	シェルに組み込まれたコマンドを示します。	○
シェル拡張コマンド	シェル拡張コマンドを示します。	○
外部コマンド	実行可能な外部コマンドを示します。	○
代入演算，算術演算	代入演算および算術演算を示します。	○
ジョブ定義スクリプト終端（EOF）	メインのジョブ定義スクリプトが終了することを示します。	○
コメントおよびスペース	コメントおよびスペースを示します。	×

（凡例）

○：停止できます。

×：停止できません。

注意事項

- 別プロセスで実行されるコマンドでは停止しません。

  例

```
1: funcA(){
2:  a=100
3:  echo $a
4: }
5: funcA &
6: pwd
```

  上記ジョブ定義スクリプトで，5 行目の「funcA &」の実行前で停止しているときに step コマンドを実行すると，6 行目の「pwd」の実行前で停止します。&の指定によって，関数 funcA の内部のコマンドは別プロセスで実行されるため，停止しません。

- シェルコマンドの引数に指定したコマンドでは停止しません。

- Windows の場合，シェルコマンドの引数にコマンド置換を指定すると，引数部分の実行前で 1 回停止でき，そのあと実際のコマンドの実行前で 1 回停止できます。また，コマンド置換を引数ではなくコマンドとして実行する場合でも，実行時に 2 回停止できます。

  UNIX の場合，シェルコマンドの引数にコマンド置換を指定すると，コマンド全体の実行前に 1 回だけ停止できます。例を次に示します。

  例

  Windows の場合

```
echo `pwd`  ←「`pwd`」の実行前と「echo」の実行前で1回ずつ停止できます。
`echo pwd`  ←「`echo pwd`」の実行前で2回停止できます。
```

  UNIX の場合

```
echo `pwd`  ←「echo `pwd`」の実行前で1回停止できます。
`echo pwd`  ←「`echo pwd`」の実行前で1回停止できます。
```

- スクリプト拡張コマンドを継続行で指定した場合，それぞれの行で停止できます。ただし，実際にスクリプト拡張コマンドが実行されるのは，最後の継続行を実行したときです。
- 代入演算のあとにスペース区切りでコマンドを指定した場合，デバッガの set コマンドによる変数の書き換えは，先頭の代入演算での停止時に行ってください。また，各コマンドで停止するとき，それらのコマンド名には変数展開後の文字列が表示されます。
- ジョブ定義スクリプト終端（EOF）にブレークポイントを設定できません。ただし，EOF ではウォッチポイント，逐次実行および停止シグナルの受信によって停止できます。
- シェルの trap コマンドの action を実行中の場合，ブレークポイントで停止しません。ただし，UNIX の場合，ウォッチポイント，逐次実行，および停止シグナルの受信によって停止できます。
- time コマンドの引数に関数呼び出しを指定して実行した場合，呼び出された関数内でウォッチポイントまたは停止シグナルの受信によって停止判定を満たすと，time コマンドを実行したあとに停止できるコマンドの実行前で停止します。
- コマンドをグループ化（子プロセスで実行）している場合，「)」が記述されている部分に停止できます。ただし，「)」の位置で停止しているとき，グループ化されたコマンド群は実行していない状態です。また，ブレークポイントも同様に，「)」が記述されている行に設定できます。「)」が記述されている行以外の行にブレークポイントを設定しようとすると，自動的に「)」が記述されている行に設定されます。例を次に示します。

  例

```
1: (      ←停止できない
2: pwd    ←停止できない
3: date   ←停止できない
4: )      ←停止できる
```

- コマンドを「{}」で括ってグループ化し，「&」などを付加して別プロセスでの実行を指定した場合，ブレークポイントを設定するとき，グループ化を 1 行で記述してください。複数行にわたって記述すると，設定したブレークポイントで停止できません。例を次に示します。

  例

1 行で指定する場合

```
1: { pwd; date;} &
```

設定したブレークポイントで停止できない場合

```
1: echo "test"
2: {            ←設定および停止できない
3: pwd          ←設定できるが，停止できない
4: date         ←設定できるが，停止できない
5: } &          ←設定できないが，停止できる
```

上記の例では，3 行目および 4 行目にブレークポイントを設定できますが，停止はできません。実際に停止できるのは 5 行目です。例えば，グループ化したコマンド群の 1 個前のコマンド（例では 1 行目の echo）から逐次実行した場合，5 行目で停止できます。また，「}」の位置で停止している場合，グループ化したコマンド群は実行していない状態です。

- コマンド置換だけをコマンドとして記述した場合，停止できる位置はコマンド置換の終端記号が記述されている行です。ただし，終端記号の位置で停止している場合，コマンド置換は実行していない状態です。ブレークポイントも同様に，終端記号の記述されている行に設定できます。終端記号の記述されている行以外の行にブレークポイントを設定すると，自動的に終端記号の記述されている行に設定されます。例を次に示します。

  例 1

```
1: $(          ←停止できない
2: echo pwd    ←停止できない
3: )           ←停止できる
```

  例 2

```
1: `echo pwd`  ←停止できる
```

- コマンド置換だけをコマンドとして記述した場合および builtin コマンド，command コマンド，time コマンドの引数にコマンド置換だけを記述した場合，コマンド置換の次のコマンドの実行前で停止したいときは，コマンド置換の結果が NULL，スペースおよびコメントのどれにもならないようにしてください。
- パイプで連結したコマンド群の実行中に停止させた場合や，コマンドのバックグラウンド実行後に停止させた場合，プロンプト文字列「(adshdb)」が連続して出力されることがあります。

6.2 CUI のデバッガ【UNIX 限定】

UNIX の実行環境の場合，デバッガはコマンドを利用して実行できます。CUI のデバッガのコマンドの記述形式を次に示します。

△$_0$**コマンド名**［△$_1$**オプション**］…［△$_1$**オプション**］［△$_1$**オペランド**］

- 最初にオプションを指定し，次にオペランドを指定します。オペランドとは，オプション名とオプション値のほかにコマンドに指定できる引数のことです。オプションの前にオペランドを指定した場合は，指定内容をすべてオペランドとして処理します。
- オプションは「-**オプション名**［△$_1$ **値**］」の形式で指定します。オプションを複数指定する場合，指定順序は任意です。
- 値のないオプションは連続して指定できます（例：「-a -b -c」と「-abc」は同じです）。その場合，最後のオプションには値を指定できます（例：「-abc xyz」の「xyz」は，オプション-c の値となります）。
- 不当なオプション，または指定できる範囲外の値を指定した場合，エラーとなります。

デバッガを起動する

デバッガは，バッチジョブを実行するコマンド（adshexec コマンド）に-d オプションとジョブ定義スクリプトファイルのパス名を指定することで起動できます。

起動したデバッガはプロンプト文字列”(adshdb)”を出力し，ユーザーからの入力待ち状態となります。ユーザーからのコマンド入力を受け付けたデバッガは，コマンドに対応する処理を実行します。処理が終了したら，プロンプト文字列を出力し，入力待ち状態となります。この動作をデバッガが終了するまで続けます。

入力待ち状態時にシグナルを受信し，再度入力待ちに戻る場合にも，プロンプト文字列「(adshdb)」を出力します。

デバッガを起動する形式を次に示します。バッチジョブを実行するコマンドについては，「8.3 シェル運用コマンド」の「adshexec コマンド（バッチジョブを実行する）」を参照してください。

```
adshexec -d ジョブ定義スクリプトファイルのパス名
```

ジョブ定義スクリプトを一時停止する

デバッグ実行中に Ctrl ＋ C を入力することで，ジョブ定義スクリプトを一時停止できます。ジョブ定義スクリプトが無限ループしている場合などに停止させる手段として有効です。

注意事項

・スクリプト拡張コマンド，シェル拡張コマンド，シェル標準コマンドまたはスクリプト予約語コマンドの実行時に Ctrl ＋ C を入力すると，コマンドの実行完了後に，次に停止可能な命令の前で停止します。

・外部コマンドをフォアグラウンドで実行しているときに Ctrl ＋ C を入力した場合は，外部コマンドの動作に従います。

6.2.1　デバッガを終了する（quit コマンド）

デバッガを終了するコマンドは，quit コマンドです。quit コマンドの短縮形は"q"です。quit コマンドの形式を次に示します。

```
quit
```

quit コマンドを実行した場合の動作を次に示します。

quit コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトが実行されている状態で quit コマンドを実行すると，デバッガを終了するかどうかの確認メッセージが出力されます。デバッガを終了する場合は，y または Y を入力してください。

ジョブ定義スクリプトが実行されていないときは，そのままデバッガを終了します。

quit コマンドの後ろに引数を指定した場合

コマンドを実行すると，エラーとなります。

注意事項

quit コマンドで終了させたジョブ定義スクリプトは，エラー終了とします。該当するジョブおよびジョブステップの終了コードは 128 になります。

6.2.2　ジョブ定義スクリプトを実行する（run コマンド）

デバッグ対象のジョブ定義スクリプトを実行するコマンドは，run コマンドです。run コマンドを実行した場合，adshexec コマンド起動時の環境変数を取り込み，ジョブ定義スクリプトの実行を開始します。run コマンドの短縮形は"r"です。run コマンドの形式を次に示します。

```
run [ 引数]
```

ジョブ定義スクリプトが実行されている状態で run コマンドを実行すると，再実行するかどうかの確認メッセージが出力されます。

run コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトを再実行する場合は，y または Y を入力してください。

ジョブ定義スクリプトが実行されていないときは，実行を開始するメッセージのあとにジョブ定義スクリプトを実行します。

run コマンドの後ろに引数を指定した場合

指定した引数を実行時パラメーターとして定義してジョブ定義スクリプトを再実行する場合は，y または Y を入力してください。

ジョブ定義スクリプトが実行されていないときは，実行を開始するメッセージのあとに，指定した引数を実行時パラメーターとして定義してジョブ定義スクリプトを実行します。

注意事項

- run コマンドで再実行した場合，それまでに実行していたジョブ定義スクリプトはエラー終了とします。該当するジョブおよびジョブステップの終了コードは 128 になります。

6.2.3　ジョブ定義スクリプトを終了する（kill コマンド）

デバッグ中のジョブ定義スクリプトを終了するコマンドは，kill コマンドです。ジョブ定義スクリプトを終了しても，デバッガ自体は終了しません。kill コマンドの短縮形は"k"です。kill コマンドの形式を次に示します。

```
kill
```

ジョブ定義スクリプトが実行されている状態で kill コマンドを実行すると，ジョブ定義スクリプトを終了するかどうかの確認メッセージが出力されます。ジョブ定義スクリプトを終了する場合は，y または Y を入力してください。

ジョブ定義スクリプトが実行されていない，または kill コマンドの後ろに引数を指定した場合は，エラーメッセージが出力されます。

注意事項

kill コマンドで終了したジョブ定義スクリプトは，エラー終了とします。該当するジョブおよびジョブステップの終了コードは 128 になります。

6.2.4　ブレークポイントを設定する（break コマンド）

ブレークポイントを設定するコマンドは，break コマンドです。break コマンドの短縮形は"b"です。

設定したブレークポイントには，1 から順に番号が割り当てられます。番号の割り当てはウォッチポイントと共通です。実行中のジョブ定義スクリプトがブレークポイントに到達すると実行を停止し，停止位置のブレークポイント情報を表示します。break コマンドの形式を次に示します。

行番号を指定する場合

```
break [ [ジョブ定義スクリプトファイル名:] 行番号]
```

break コマンドの引数に行番号を指定すると，指定した行にブレークポイントを設定します。

関数名を指定する場合

```
break 関数名
```

break コマンドの引数に関数名を指定すると，指定した関数にブレークポイントを設定します。

ジョブステップ名を指定する場合

```
break -s ［ジョブ定義スクリプトファイル名:］ジョブステップ名
```

-s オプションを指定し，ジョブステップ名を指定すると，ジョブステップ名が定義されている行にブレークポイントを設定できます。

引数に”:”を使用すれば，ジョブ定義スクリプトファイル名を指定してブレークポイントを設定できます。ただし，関数名を指定する場合はそのとき有効になっている 1 個の関数が対象となるため，ファイル名を指定する必要はありません。

break コマンドを実行した場合の動作を次に示します。

break コマンドの後ろに引数を指定しない場合

現在停止中の行にブレークポイントを設定します。ただし，ジョブ定義スクリプトが実行されていないときは，1 行目以降で最初に停止できる行にブレークポイントを設定します。設定したブレークポイントの情報を表示します。

break コマンドの後ろに引数を指定した場合

指定した引数によって動作が異なります。詳細を次に示します。

- 行番号

  引数に指定した行にブレークポイントを設定します。設定したブレークポイントの情報を表示します。指定した行がない場合，エラーとなります。

- 関数名

  引数に指定した関数にブレークポイントを設定します。設定したブレークポイントの情報を表示します。ブレークポイントの位置は，関数定義内で最初に停止できる行になります。指定した関数がない場合，エラーとなります。

- -s ジョブステップ名

  引数に指定したジョブステップにブレークポイントを設定します。設定したブレークポイントの情報を表示します。指定したジョブステップがない場合，エラーとなります。

- ジョブ定義スクリプトファイル名

  指定するファイル名には，バッチジョブを実行するコマンドに指定したファイルまたはスクリプト拡張コマンドの#-adsh_script に指定したファイルを指定してください。指定したファイル名をブレークポイント設定の対象とします。また，ファイル名の指定を省略すると，カレントファイルをブレークポイント設定の対象とします。

注意事項

- ジョブ定義スクリプトファイル名の指定は，最後に入力された”:”の前までの文字列をファイル名と見なします。
- 行番号は 0 以上の整数を入力してください。先頭に「+」は付けません。
- 行番号の指定時に int 型の範囲を超えた数値を入力した場合，int 型の上限値に丸めて扱います。
- 同じ名称のジョブステップが複数存在する場合，それらすべてにブレークポイントを設定します。

- 設定できるブレークポイントとウォッチポイントの番号を合わせて上限は 999 です。合計数が上限の 999 に達してから新たにブレークポイントまたはウォッチポイントを設定したい場合は，デバッガをいったん終了してください。上限値に達した場合，delete コマンドでブレークポイントやウォッチポイントを削除しても，設定できません。
- 停止できるコマンドが 1 つ以上含まれている行であれば，ブレークポイントを設定できます。1 行に複数の停止できるコマンドがある場合，各コマンドの実行前に停止します。
- 行番号の指定で，停止できないコマンドだけで構成される行を指定した場合，警告メッセージが出力され，次に停止できるコマンドがある行にブレークポイントが設定されます。

  次に停止できるコマンドがない場合は，エラーメッセージが出力されます。
- 関数の内部に停止できるコマンドが 1 つも定義されていない状態で関数名を指定すると，関数定義の終了以降の行で停止できるコマンドがある行にブレークポイントを設定します。また，関数定義と同じ行に停止できるコマンドがある場合，そのコマンドが関数内のコマンドかどうかに関係なく，関数定義のある行にブレークポイントを設定します。
- ジョブステップを定義するスクリプト拡張コマンドを複数行に分けて記述している場合，ジョブステップ指定で設定するブレークポイントの行番号は，先頭のスクリプト拡張コマンドが記述されている行番号になります。
- 外部スクリプトに対するブレークポイントの設定は，その外部スクリプトを呼び出す側のジョブ定義スクリプト中で，その外部スクリプトをスクリプト拡張コマンド#-adsh_script に指定して呼び出したときにだけ可能です。#-adsh_script に指定していない外部スクリプト内では停止できません。
- 同一の行に設定できるブレークポイントの数は 1 個です。ブレークポイントを設定済みの行には設定できません。
- デバッグ実行中のジョブ定義スクリプトが次に示すどちらかの状態の場合は，break コマンドに引数を指定しないで実行しないでください。

  ・ジョブ定義スクリプト終端(EOF)で停止している

  ・trap コマンドの action を実行中に停止している

使用例

ジョブ定義スクリプトの 8 行目にブレークポイントを設定してジョブ定義スクリプトを実行すると，8 行目の「funcA」の実行前で停止します。

```
1: funcA(){
2:   echo ”funcA”
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num
```

```
KNAX7018-I ブレークポイント ”1”: ファイル名=”test.ash” 行番号=8
(adshdb) run
KNAX7007-I ジョブ定義スクリプトの実行を開始します。: /home/test/test.ash
…
```

```
KNAX7018-I ブレークポイント ”1”: ファイル名=”test.ash” 行番号=8
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
8: funcA
現在位置: funcA
(adshdb)
```

6.2.5 ウォッチポイントを設定する（watch コマンド）

ウォッチポイントを設定するコマンドは，watch コマンドです。watch コマンドの短縮形は"wa"です。

設定したウォッチポイントには，1 から順に番号が割り当てられます。番号割り当てはブレークポイントと共通です。watch コマンドの形式を次に示します。

```
watch 変数名
```

watch コマンドの引数には変数名を指定します。指定した変数の値が更新された場合，ジョブ定義スクリプトは次に停止可能なコマンドの前で実行を停止し，ウォッチポイント情報を表示します。

watch コマンドを実行した場合の動作を次に示します。

watch コマンドの後ろに引数を指定した場合

指定した変数にウォッチポイントを設定します。また，設定したウォッチポイントの情報を表示します。

watch コマンドの後ろに引数を指定しない場合

エラーメッセージが出力されます。

ウォッチポイントによってジョブ定義スクリプトの実行が停止した場合，ウォッチポイント情報として更新前の値，更新後の値および変数を更新した行の行番号を表示します。表示形式を次に示します。

```
更新前の値 = 更新前の値
更新後の値 = 更新後の値
行番号 = 行番号
```

- **更新前の値**：更新される前のウォッチ対象の変数の値です。値がない場合，<No value>と表示します。
- **更新後の値**：更新されたあとのウォッチ対象の変数の値です。値がない場合，<No value>と表示します。
- **行番号**：変数を更新した行の行番号です。trap コマンドの action を実行中の場合は，<Trap action>と表示します。ジョブ定義スクリプト終端の場合，<EOF>と表示します。

注意事項

- 変数名に配列を指定する場合は，要素ごとに指定します。

  例

  通常の変数指定：aaa

  配列指定：aaa[1]または aaa[0][1]

- 変数とその配列の 0 番目（例：aaa，aaa[0]，および aaa[0][0]）は同じです。ただし，ウォッチポイントはそれぞれに設定できます。
- 変数名に$を付けると変数名として認識されません。
- 引数に変数名を指定する段階では，指定した変数が存在するかどうかは判断しません。
- 変数の命名規則から外れた変数名を指定した場合，エラーメッセージが出力されます。
- 更新前の変数の値と同じ値が代入された場合でも，ウォッチポイントとして停止します。
- シェル標準コマンドの typeset コマンドを使用して変数の値を文字列型から整数型に変更した場合，または整数型から文字列型に変更した場合，ウォッチポイントとして停止します。
- 関数名と変数名の命名規則が同じため，引数に関数名を指定するとウォッチポイントとして設定できます。ただし，設定した関数名と同名の変数の値が更新されないかぎり，ジョブ定義スクリプトの実行は停止しません。
- 1 行に複数のコマンドがあり，ウォッチ対象の変数の値が更新された場合，同じ行であっても次に停止できるコマンドの前で停止します。
- 設定できるブレークポイントとウォッチポイントの番号を合わせて上限は 999 です。合計数が上限の 999 に達してから新たにブレークポイントまたはウォッチポイントを設定したい場合は，デバッガをいったん終了してください。上限値に達した場合，delete コマンドでブレークポイントやウォッチポイントを削除しても，設定できません。
- ジョブ定義スクリプトが停止しているときに set コマンドでウォッチ対象の変数の値を変更し，ジョブ定義スクリプトの実行を再開すると，次に停止できるコマンドの前で実行を停止します。
- 同一の変数に設定できるウォッチポイントの数は 1 個です。ウォッチポイントとして設定済みの変数は，ウォッチポイントとして設定できません。
- 次のコマンドでウォッチ対象の変数の値が更新された場合，ジョブ定義スクリプトの実行は停止しません。

  ・バックグラウンドで実行したコマンド(&または|&)

  ・パイプで連結したコマンド

  ・「(」「)」で囲んでコマンドのグループ化をしたコマンド群

  ・外部コマンドのように別プロセスで実行されるコマンド

  例

  ```
  1： a=1 &
  2： b=2
  3： c=3
  ```

  上記の例で変数 a にウォッチポイントを設定し，ジョブ定義スクリプトを実行した場合，代入式「a=1」はバックグラウンドで実行されるため，ジョブ定義スクリプトの実行は停止しません。

使用例

次のジョブ定義スクリプトに対して「watch b」と指定すると，変数 b にウォッチポイントを設定できます。

```
1: echo ”start”
2: a=1
3: b=5
4: c=10
5: echo ”end”
```

その後，ジョブ定義スクリプトを実行して代入式「b=5」が実行されると，ウォッチポイント情報を表示し，4行目の「c=10」の実行前で停止します。

```
KNAX7023-I ウォッチポイント ”1”: 変数名 ”b”
更新前の値 = <No value>
更新後の値 = 5
行番号 = 3
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
4: c=10
現在位置: c=10
(adshdb)
```

6.2.6 ブレークポイント・ウォッチポイントを削除する（delete コマンド）

ブレークポイントまたはウォッチポイントを削除するコマンドは，delete コマンドです。delete コマンドの短縮形は"d"です。delete コマンドの形式を次に示します。

```
delete [ ブレークポイント・ウォッチポイント番号 [-ブレークポイント・ウォッチポイント番号] ]
```

delete コマンドの引数に番号を指定すると，対応するブレークポイントまたはウォッチポイントを削除できます。また，”-”を使用して番号の範囲を指定できます。例えば，番号1から番号5までのポイントを削除したい場合は，「1-5」と指定します。引数を省略すると，すべてのブレークポイントとウォッチポイントを削除します。

delete コマンドを実行した場合の動作を次に示します。

delete コマンドの後ろに引数を指定しない場合

ブレークポイントまたはウォッチポイントが1つ以上設定されているときは，すべてのブレークポイントとウォッチポイントを削除するかどうかの確認メッセージが出力されます。削除する場合は，y または Y を入力してください。

ブレークポイントまたはウォッチポイントが1つも設定されていないときは，エラーメッセージが出力されます。

delete コマンドの後ろに引数を指定した場合

- ブレークポイント・ウォッチポイント番号

  指定した番号のブレークポイントまたはウォッチポイントを削除します。指定した番号が存在しない場合，エラーメッセージが出力されます。

- 番号-番号

「-」の前の番号から後ろの番号までの範囲にあるブレークポイントとウォッチポイントを削除します。指定した範囲内に含まれる番号のウォッチポイントおよびブレークポイントがすべて削除されます。指定した範囲内にウォッチポイントおよびブレークポイントがひとつも含まれない場合，エラーメッセージが出力されます。

- 上記以外

  エラーメッセージが出力されます。

注意事項

- 引数の番号は 0 以上の整数を入力してください。先頭に「+」は付けません。
- 番号を範囲指定した場合，前の値と後ろの値が等しいときは，その番号だけを対象とします。
- 番号を範囲指定した場合，前の値より後ろの値が小さいときは，エラーメッセージが出力されます。
- 番号の指定時に int 型の範囲を超えた数値を入力した場合，int 型の上限値に丸めて扱います。

6.2.7　ジョブ定義スクリプトの実行を再開するコマンド

ジョブ定義スクリプトの実行を再開する場合，次の 3 つの方法があります。

- 逐次実行

  ジョブ定義スクリプトが停止した位置から 1 つのコマンドを実行します。逐次実行をするコマンドは，step コマンドおよび next コマンドです。

- 継続実行

  停止したジョブ定義スクリプトの実行を継続します。継続実行をするコマンドは，continue コマンドです。

- 関数を実行

  関数の中で実行を停止しているときに，関数からリターンするまでジョブ定義スクリプトを実行します。関数を実行するコマンドは，finish コマンドです。

また，関数を終了するコマンドは，return コマンドです。ジョブ定義スクリプトにシグナルを送信するコマンドは，signal コマンドです。

コマンドを実行したあとにジョブ定義スクリプトの実行が停止すると，メッセージ，次に実行予定の行番号，ソースファイル行を表示します。表示形式を次に示します。

バッチジョブを実行するコマンドに指定したジョブ定義スクリプトファイルおよびスクリプト拡張コマンドの#-adsh_script に指定したジョブ定義スクリプトファイルの場合

```
行番号：ソースファイル行
現在位置：コマンド文字列
```

- **行番号**：次に実行するコマンドの行番号です。
- **ソースファイル行**：行番号に対応するソースファイルの行の内容です。

- **コマンド文字列**：次に実行するコマンド文字列です。

スクリプト拡張コマンドの#-adsh_script に指定していない外部スクリプトの場合

```
行番号：行番号
現在位置 コマンド文字列
```

- **行番号**：次に実行するコマンドの行番号です。
- **コマンド文字列**：次に実行するコマンド文字列です。

注意事項

- 別プロセスで実行されるジョブ定義スクリプトの場合，コマンド文字列に<Another process script>と表示されます。
- デバッグ中のジョブ定義スクリプトが trap コマンドの action を実行中の場合は，次の表示形式になります。

```
行番号：<Trap action>
現在位置：コマンド文字列
```

- ジョブ定義スクリプト終端(EOF)の場合は，次の表示形式になります。

```
現在位置：<EOF>
```

出力例

次に実行予定の行番号とソースファイル行を表示します。

バッチジョブを実行するコマンドに指定したジョブ定義スクリプトファイルおよびスクリプト拡張コマンドの#-adsh_script に指定したジョブ定義スクリプトファイルの場合

```
100: echo ”aaa”    ←次に実行する処理は100行目の「echo ”aaa”」
現在位置: echo      ←その時に実行されるコマンドは「echo」
```

スクリプト拡張コマンドの#-adsh_script に指定していない外部スクリプトの場合

```
行番号: 50          ←次に実行されるのは外部スクリプトの50行目にある処理
現在位置: num=1     ←その時に実行される処理は「num=1」
```

6.2.8　逐次実行をする（step コマンド，next コマンド）

ジョブ定義スクリプトが停止した位置から 1 つのコマンドを実行するコマンドは，step コマンドおよび next コマンドです。step コマンドは停止した位置の 1 つのコマンドで関数が呼ばれている場合，関数内に入ります。next コマンドは停止した位置の 1 つのコマンドで関数が呼ばれていても，関数内で停止しないで実行します。step コマンドの短縮形は"s"，next コマンドの短縮形は"n"です。

(1)　step コマンド

step コマンドの形式を次に示します。

```
step
```

step コマンドを実行した場合の動作を次に示します。

step コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトが実行中で停止している状態で step コマンドを実行すると，ジョブ定義スクリプトが停止した位置から 1 つのコマンドを実行します。関数が呼ばれている場合は関数内に入ります。

ジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

step コマンドの後ろに引数を指定した場合

コマンドを実行すると，エラーとなります。

注意事項

シェル標準コマンドの eval コマンドの引数に関数呼び出しを指定している場合，eval コマンドを逐次実行すると，その関数呼び出しを実行したあとの停止位置は，step コマンドの動作に従います。

使用例

6 行目の「val=1」の実行前で停止しているときに step コマンドを実行すると，7 行目の「num=2」の実行前で停止します。

```
1: funcA(){
2:   echo ”funcA”
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num
```

```
6: val=1
現在位置: val=1
(adshdb) step
…
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
7: num=2
現在位置: num=2
(adshdb)
```

8 行目の関数呼び出し「funcA」の実行前で停止しているときに step コマンドを実行すると，2 行目の「echo」の実行前で停止します。

```
8: funcA
現在位置: funcA
(adshdb) step
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
2:   echo ”funcA”
現在位置: echo
(adshdb)
```

(2) next コマンド

next コマンドの形式を次に示します。

```
next
```

next コマンドを実行した場合の動作を次に示します。

next コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトが実行中で停止している状態で next コマンドを実行すると，ジョブ定義スクリプトが停止した位置から 1 つのコマンドを実行します。関数が呼ばれていても関数内で停止しません。

ジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

next コマンドの後ろに引数を指定した場合

コマンドを実行すると，エラーとなります。

注意事項

- next コマンドを実行した場合，次の 1 コマンドで関数が呼ばれて，その関数内でブレークポイント，ウォッチポイントおよびシグナルによる停止判定を満たしたとき，ジョブ定義スクリプトの実行を停止します。
- シェル標準コマンドの eval コマンドの引数に関数呼び出しを指定している場合，eval コマンドを逐次実行すると，その関数呼び出しを実行したあとの停止位置は，next コマンドの動作に従います。

使用例

6 行目の「val=1」の実行前で停止しているときに next コマンドを実行すると，7 行目の「num=2」の実行前で停止します。

```
1: funcA(){
2:   echo ”funcA”
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num
```

```
6: val=1
現在位置: val=1
(adshdb) next
…
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
7: num=2
現在位置: num=2
(adshdb)
```

8 行目の関数呼び出し「funcA」の実行前で停止しているときに next コマンドを実行すると，9 行目の「echo」の実行前で停止します。

```
8: funcA
現在位置: funcA
(adshdb) next
...
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
9: echo $num
現在位置: echo
(adshdb)
```

6.2.9　継続実行をする（continue コマンド）

停止したジョブ定義スクリプトの実行を継続するコマンドは，continue コマンドです。continue コマンドは，停止した位置からジョブ定義スクリプトの実行を継続します。continue コマンドの短縮形は"c"です。continue コマンドの形式を次に示します。

```
continue
```

continue コマンドを実行した場合の動作を次に示します。

continue コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトが実行中で停止している状態で continue コマンドを実行すると，継続実行を示すメッセージが出力され，ジョブ定義スクリプトの実行を再開します。

ジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

continue コマンドの後ろに引数を指定した場合

コマンドを実行すると，エラーとなります。

6.2.10　関数を実行する（finish コマンド）

関数からリターンするまでジョブ定義スクリプトを実行するコマンドは，finish コマンドです。finish コマンドの短縮形は"f"です。finish コマンドの形式を次に示します。

```
finish
```

finish コマンドを実行した場合の動作を次に示します。

finish コマンドの後ろに引数を指定しない場合

関数内で停止しているときは，現在の関数の終わりまで実行するというメッセージが出力され，関数の終わりまでジョブ定義スクリプトを実行します。ジョブ定義スクリプトの実行が停止すると，停止位置のフレーム情報を表示し，次に実行予定の行番号とそのソースファイル行を表示します。停止位置のフレーム情報の表示形式を次に示します。

停止位置のフレーム情報

```
Num  Function  File:Line
フレーム番号  関数名  ファイル名:行番号
```

・**フレーム番号**：フレームの番号です。フレーム番号には常に 0 を表示します。

・**関数名**：フレーム情報に対応する関数名に，関数を呼び出しているジョブ定義スクリプトファイル名を付けた名称です。関数が呼び出されていない状態の場合，関数名を<main>として表示します。63 バイトまで表示できます。

・**ファイル名**：現在停止中のファイル名を示します。

・**行番号**：現在停止中の行番号を示します。

ジョブ定義スクリプト終端で停止している場合は<EOF>と表示します。trap コマンドの action を実行中に停止している場合は<Trap action>と表示します。

どの関数にも入っていない状態で停止しているときおよびジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

finish コマンドの後ろに引数を指定した場合

コマンドを実行すると，エラーとなります。

注意事項

関数内の後続の行で，ブレークポイント，ウォッチポイントおよびシグナルによる停止判定を満たした場合，ジョブ定義スクリプトの実行を停止します。

使用例

2 行目の「echo」の実行前で停止しているときに finish コマンドを実行すると，9 行目の「echo」の実行前で停止し，フレーム情報を表示します。

```
1: funcA(){
2:   echo "funcA"
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num
```

```
2:   echo "funcA"
現在位置: echo
(adshdb) finish
KNAX7036-I 現在の関数の終わりまで実行します。
…
Num  Function  File:Line
0    <main>    test.ash:9
KNAX7032-I ジョブ定義スクリプトファイル（"test.ash"）の中で実行を停止しました。
9: echo $num
現在位置: echo
(adshdb)
```

6.2.11　関数を終了する（return コマンド）

関数を終了するコマンドは，return コマンドです。return コマンドを実行した場合，関数内の現在位置以降の行は実行されないで，関数の呼び出し元に戻ります。return コマンドの短縮形は"ret"です。return コマンドの形式を次に示します。

```
return
```

return コマンドを実行した場合の動作を次に示します。

return コマンドの後ろに引数を指定しない場合

関数内で停止しているときは，現在の関数を終了するかどうかの確認メッセージが出力されます。現在の関数を終了し，関数の呼び出し元に戻る場合は，y または Y を入力してください。呼び出し元のフレーム情報を表示し，次に実行予定の行番号とそのソースファイル行を表示します。呼び出し元のフレーム情報の表示形式を次に示します。

呼び出し元のフレーム情報

```
Num  Function  File:Line
フレーム番号　関数名　ファイル名:行番号
```

・**フレーム番号**：フレームの番号です。フレーム番号には常に 0 を表示します。

・**関数名**：フレーム情報に対応する関数名に，関数を呼び出しているジョブ定義スクリプトファイル名を付けた名称です。関数が呼び出されていない状態の場合，関数名を<main>として表示します。63 バイトまで表示できます。

・**ファイル名**：現在停止中のファイル名を示します。

・**行番号**：現在停止中の行番号を示します。

ジョブ定義スクリプト終端で停止している場合は<EOF>と表示します。trap コマンドの action を実行中に停止している場合は<Trap action>と表示します。

どの関数にも入っていない状態で停止しているときおよびジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

return コマンドの後ろに引数を指定した場合

コマンドを実行すると，エラーとなります。

注意事項

関数内の後続の行で，ブレークポイント，ウォッチポイントおよびシグナルによる停止判定を満たしても，ジョブ定義スクリプトの実行を停止しないで関数を終了します。

使用例

2 行目の「echo」の実行前で停止しているときに return コマンドを実行すると，9 行目の「echo」の実行前で停止し，フレーム情報を表示します。3 行目の「num=10」はスキップされます。

```
1: funcA(){
2:   echo ”funcA”
3:   num=10
4: }
```

```
5:
6: val=1
7: num=2
8: funcA
9: echo $num
```

```
2:   echo ”funcA”
現在位置: echo
(adshdb) return
KNAX7037-I 現在の関数を終了しますか？(y または n)
y
KNAX7068-I 関数の終わりまでコマンドをスキップします。
…
Num  Function  File:Line
0    <main>    test.ash:9
KNAX7032-I ジョブ定義スクリプトファイル（”test.ash”）の中で実行を停止しました。
9: echo $num
現在位置: echo
(adshdb)
```

6.2.12　シグナルを送信する（signal コマンド）

ジョブ定義スクリプトにシグナルを送信するコマンドは，signal コマンドです。signal コマンドの短縮形は"si"です。signal コマンドの形式を次に示します。

```
signal  {シグナル名|シグナル番号}
```

引数にシグナル名またはシグナル番号を指定すると，対応するシグナルを送信し，ジョブ定義スクリプトを継続実行します。引数に指定できるシグナルの情報を表示するには，info signals コマンドを使用してください。また，シグナル受信時の動作については，「3.11.2　シグナル受信時の動作【UNIX 限定】」を参照してください。

signal コマンドを実行した場合の動作を次に示します。

signal コマンドの後ろに引数を指定した場合

ジョブ定義スクリプトが実行されている状態で signal コマンドを実行すると，次のようになります。

シグナル番号およびシグナル名

指定したシグナルを送信するというメッセージが出力されます。そのあと，指定したシグナルをジョブ定義スクリプトに送信し，ジョブ定義スクリプトを継続実行します。

存在しないシグナルのときおよびジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

signal コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトが実行中で停止している状態で signal コマンドを実行すると，エラーメッセージが出力されます。

ジョブ定義スクリプトが実行されていないときも，エラーメッセージが出力されます。

注意事項

- 引数の番号は 0 以上の整数を入力してください。先頭に「+」は付けません。
- 番号の指定時に int 型の範囲を超えた数値を入力した場合，int 型の上限値に丸めて扱います。
- AIX の場合に次のシグナルを送信したいときは，シグナル番号，または同一シグナル番号であるほかのシグナル名を指定してください。

  SIGLOST または SIGIOT を送信したいとき：シグナル番号 6，または SIGABRT を指定します。

  SIGPOLL を送信したいとき：シグナル番号 23，または SIGIO を指定します。
- HP-UX の場合に次のシグナルを送信したいときは，シグナル番号，または同一シグナル番号であるほかのシグナル名を指定してください。

  SIGIOT を送信したいとき：シグナル番号 6，または SIGABRT を指定します。

  SIGPOLL を送信したいとき：シグナル番号 22，または SIGIO を指定します。
- Solaris の場合に次のシグナルを送信したいときは，シグナル番号，または同一シグナル番号であるほかのシグナル名を指定してください。

  SIGIOT を送信したいとき：シグナル番号 6，または SIGABRT を指定します。

  SIGPOLL を送信したいとき：シグナル番号 22，または SIGIO を指定します。

6.2.13 ブレークポイント・ウォッチポイントの情報を表示する（info breakpoints コマンド）

設定されているブレークポイントとウォッチポイントの情報を表示するコマンドは，info breakpoints コマンドです。info breakpoints コマンドの短縮形は"i b"です。info breakpoints コマンドの形式を次に示します。

```
info breakpoints [ ブレークポイント・ウォッチポイント番号]
```

引数にブレークポイントまたはウォッチポイントの番号を指定すると，対応するブレークポイントまたはウォッチポイントの情報を表示します。引数を省略すると，すべてのブレークポイントとウォッチポイントの情報を表示します。表示形式を次に示します。

```
Num  Type         What
番号  breakpoint/watchpoint  ファイル名:行番号/変数名
…
```

- **番号**：ブレークポイント・ウォッチポイント番号です。左詰めで 3 桁まで表示できます。
- **ファイル名**：ジョブ定義スクリプトファイル名です。
- **行番号**：ブレークポイントが設定されている行番号です。
- **変数名**：watch コマンドで指定した変数名です。

info breakpoints コマンドを実行した場合の動作を次に示します。

info breakpoints コマンドの後ろに引数を指定しない場合

ブレークポイントまたはウォッチポイントが 1 つ以上設定されているときは，すべてのブレークポイントとウォッチポイントを表示します。

ブレークポイントまたはウォッチポイントが 1 つも設定されていないときは，メッセージが出力されます。

info breakpoints コマンドの後ろに引数を指定した場合

- ブレークポイント・ウォッチポイント番号

  番号が存在するときは，指定した番号のブレークポイントまたはウォッチポイントの情報を表示します。

  番号が存在しないときは，エラーメッセージが出力されます。

- 上記以外

  エラーメッセージが出力されます。

注意事項

- 引数の番号は 0 以上の整数を入力してください。先頭に「+」は付けません。
- 番号の指定時に int 型の範囲を超えた数値を入力した場合，int 型の上限値に丸めて扱います。

出力例

ブレークポイントとウォッチポイントを表示します。

```
Num  Type        What
1    breakpoint  sample.ash:100
2    watchpoint  rc
```

6.2.14　カバレージ情報を表示する（info coverage コマンド）

デバッグ途中のカバレージ情報を表示するコマンドは，info coverage コマンドです。info coverage コマンドの短縮形は"i c"です。info coverage コマンドの形式を次に示します。

```
info coverage [ n1 [- [n2] ]
              [,n3 [- [n4] ] ] ...]
```

n1，n2，n3，n4 などの引数には行番号を指定します。引数に指定した範囲の行のカバレージ情報を表示します。引数を指定しない場合，すべてのカバレージ情報を表示します。

使用例

1 行目～10 行目と 15 行目と 21 行目以降のカバレージ情報を表示します。

```
info coverage 1-10,15,21-
```

カバレージ情報の表示の詳細については，「3.10　カバレージ情報を取得する」を参照してください。

6.2.15　関数情報を表示する（info functions コマンド）

関数情報を表示するコマンドは，info functions コマンドです。info functions コマンドの短縮形は"i f"です。info functions コマンドの形式を次に示します。

```
info functions [ 関数名]
```

引数に関数名を指定すると，その関数名とそれに対応するファイル名および行番号を表示します。引数を省略すると，すべての関数名とそれに対応するファイル名および行番号を表示します。表示形式を次に示します。

```
Function  File:Line
関数名   ファイル名:行番号
…
```

- **関数名**：定義されている関数名です。関数名ごとに ASCII コード順で表示します。31 バイトまで表示できます。32 バイト目以降は表示されません。関数名の長さに合わせて調節し，カラムを合わせます。
- **ファイル名**：関数が定義されているジョブ定義スクリプトファイル名です。
- **行番号**：関数が定義されている行番号です。

info functions コマンドを実行した場合の動作を次に示します。

info functions コマンドの後ろに引数を指定しない場合

すべての関数名とそれに対応するファイル名および行番号を表示します。

info functions コマンドの後ろに引数を指定した場合

- 存在する関数名
  指定した関数名とそれに対応するファイル名および行番号を表示します。
- 上記以外
  エラーメッセージが出力されます。

注意事項

ジョブ定義スクリプトと，スクリプト拡張コマンドの#-adsh_script に指定した外部スクリプトの関数情報は，adshexec コマンドの起動時に自動的に取り込まれるため，ジョブ定義スクリプトを実行する前でも参照できます。ただし，スクリプト拡張コマンドの#-adsh_script に指定していない外部スクリプト中に記述している関数は，run コマンドによるジョブ定義スクリプト実行中，かつその関数を定義している処理を完了するまでは，関数情報は表示されません。

出力例

関数名とそれに対応するファイル名および行番号を表示します。

```
Function  File:Line
funcA     script1.ash:100
funcB     script2.ash:10
funcXXX   script1.ash:50
```

6.2.16　ジョブステップ情報を表示する（info jobsteps コマンド）

ジョブステップ情報を表示するコマンドは，info jobsteps コマンドです。info jobsteps コマンドの短縮形は"i j"です。info jobsteps コマンドの形式を次に示します。

```
info jobsteps [ ジョブステップ名]
```

引数にジョブステップ名を指定すると，そのジョブステップ名とそれに対応するファイル名および行番号を表示します。引数を省略すると，すべてのジョブステップ名とそれに対応するファイル名および行番号を表示します。表示形式を次に示します。

```
Jobstep  File:Line
ジョブステップ名　ファイル名:行番号
…
```

- **ジョブステップ名**：定義されているジョブステップ名です。ジョブステップ名を ASCII コード順に並べて表示します。31 バイトまで表示できます。ジョブステップ名の長さに合わせて調節し，カラムを合わせます。ジョブステップ名が省略されている場合，ジョブステップ名を<No name>と表示します。
- **ファイル名**：ジョブステップが定義されているジョブ定義スクリプトファイル名です。
- **行番号**：ジョブステップが定義されている行番号です。

info jobsteps コマンドを実行した場合の動作を次に示します。

info jobsteps コマンドの後ろに引数を指定しない場合

すべてのジョブステップ名とそれに対応するファイル名および行番号を表示します。

info jobsteps コマンドの後ろに引数を指定した場合

ジョブステップ名が存在するときは，指定したジョブステップ名とそれに対応するファイル名および行番号を表示します。

ジョブステップ名が存在しないときは，エラーメッセージが出力されます。

注意事項

ジョブステップ情報は adshexec コマンド起動時に取り込んでいるため，run の実行前後に関係なく表示できます。

出力例

ジョブステップ名とそれに対応するファイル名および行番号を表示します。

```
Jobstep  File:Line
step1    script1.ash:10
step2    script1.ash:30
step3    script2.ash:10
```

6.2.17　パスを扱う変数名情報を表示する（info pathvars コマンド）

info pathvars コマンドは，次の情報を標準エラー出力に表示します。

- 指定したシェル変数名が「パス名を扱うシェル変数」であるか「パス名を扱わないシェル変数」であるか

info pathvars コマンドの形式を次に示します。

```
info pathvars シェル変数名
```

info pathvars コマンドの動作を次に示します。

引数	動作内容
存在する変数名	指定したシェル変数がパス名を扱う変数名であるかどうかを表示します（表示形式 1）。
上記以外	パス名を扱わない変数として，同等の表示をします。

表示形式 1

```
種別 ： シェル変数名
```

- 種別：「パス名を扱うシェル変数」または「パス名を扱わないシェル変数」を示します。
  var：「パス名を扱うシェル変数」
  novar：「パス名を扱わないシェル変数」
- シェル変数名：コマンドで指定したシェル変数名です。

6.2.18　シグナル情報を表示する（info signals コマンド）

シグナルの情報を表示するコマンドは，info signals コマンドです。info signals コマンドの短縮形は"i si"です。info signals コマンドの形式を次に示します。

```
info signals [ シグナル名| シグナル番号]
```

引数にシグナル名またはシグナル番号を指定すると，対応するシグナルの情報を表示します。引数を省略すると，すべてのシグナルの情報を表示します。表示形式を次に示します。

```
Num  Signal       Stop  Print
シグナル番号    シグナル名  Yes/No   Yes/No
…
```

- **シグナル番号**：シグナル番号です。シグナル番号を昇順に並べて表示します。左詰めで 2 桁まで表示できます。

- **シグナル名**：シグナル名です。左詰めで 11 バイトまで表示できます。

info signals コマンドを実行した場合の動作を次に示します。

info signals コマンドの後ろに引数を指定しない場合

すべてのシグナルの情報を表示します。

info signals コマンドの後ろに引数を指定した場合

- シグナル番号
  シグナル番号が存在するときは，指定した番号のシグナルの情報を表示します。
  シグナル番号が存在しないときは，エラーメッセージが出力されます。
- シグナル名
  シグナル名が存在するときは，指定したシグナルの情報を表示します。
  シグナル名が存在しないときは，エラーメッセージが出力されます。
- Stop
  Yes：Signal で示すシグナルを受信した場合に，実行中のジョブ定義スクリプトを停止します。
  No：Signal で示すシグナルを受信しても，実行中のジョブ定義スクリプトを停止しません。
  シグナル受信時の動作については，「3.11.2　シグナル受信時の動作【UNIX 限定】」を参照してください。
- Print
  Yes：Signal で示すシグナルを受信した場合に，シグナル受信のメッセージを表示します。
  No：Signal で示すシグナルを受信しても，シグナル受信のメッセージを表示しません。

注意事項

- 番号の指定は 0 以上の整数に限定します。先頭に「+」は付けません。それ以外を入力した場合，エラーメッセージが出力されます。
- 番号として int 型の範囲を超えた数値を指定した場合，int 型の上限値に丸めて扱います。

出力例

シグナルの情報を表示します。

```
Num  Signal       Stop  Print
1    SIGHUP       No    No
2    SIGINT       Yes   Yes
```

6.2.19　ステータスを表示する（info status コマンド）

デバッグ中のジョブ定義スクリプトのステータスを表示するコマンドは，info status コマンドです。info status コマンドの短縮形は"i st"です。info status コマンドの形式を次に示します。

```
info status [ joberrmode]
```

引数に joberrmode を指定するか，または引数を省略すると，ジョブ定義スクリプトのエラー注入モードの状態を表示します。joberrmode の短縮形は"jem"です。

出力例

ジョブ定義スクリプトのステータス（エラー注入モードが有効の場合）を表示します。

```
joberrmode:on
```

ジョブ定義スクリプトのステータス（エラー注入モードが無効の場合）を表示します。

```
joberrmode:off
```

6.2.20　シェル変数情報を表示する（info variables コマンド）

すべてのシェル変数情報を表示するコマンドは，info variables コマンドです。info variables コマンドの短縮形は"i v"です。info variables コマンドの形式を次に示します。

```
info variables [ 変数名]
```

引数に変数名を指定すると，指定したシェル変数情報を表示します。引数を省略すると，すべてのシェル変数情報を表示します。表示形式を次に示します。

```
変数名 = 変数値 [(ステップローカル変数)]
…
```

- **変数名**：シェル変数名です。変数名を ASCII コード順に並べて表示します。
- **変数値**：シェル変数の値です。ジョブステップ内でだけ有効なシェル変数の場合，補足情報（ステップローカル変数）を付けて表示します。ジョブステップ内でだけ有効なシェル変数とは，スクリプト拡張コマンドの#-adsh_step_start コマンドの stepVar 属性に指定したシェル変数です。

値を持っていない変数の場合，「=」と変数値を表示しません。表示形式を次に示します。

```
変数名 [(ステップローカル変数)]
```

info variables コマンドを実行した場合の動作を次に示します。

info variables コマンドの後ろに引数を指定しない場合

すべてのシェル変数情報を表示します。

info variables コマンドの後ろに引数を指定した場合

表示する変数名が存在するときは，指定したシェル変数情報を表示します。

表示する変数名が存在しないときは，エラーメッセージが出力されます。

注意事項

- run コマンドによるジョブ定義スクリプト実行中以外では変数が定義されていないため，変数を表示できません。
- 変数名に配列を指定する場合は，要素ごとに指定します。
- 変数とその配列の 0 番目（例：aaa，aaa[0]，および aaa[0][0]）は同一のものであるため，ジョブ定義スクリプト内で配列となっているときは添え字を付けて表示し，配列となっていないときは添え字を付けないで表示します。
- シェルは配列を作成すると配列の 0 番目が自動的に作成されるため，すべてのシェル変数を表示する場合，それらを含めて表示されます。

出力例

すべてのシェル変数情報を表示します。

```
SHELL = /bin/sh
TEMPFILE = /tmp/file01
num = 1（ステップローカル変数）
val = 100
```

6.2.21　エラー注入モードの有効/無効を設定する（joberrmode コマンド）

デバッグ中のジョブ定義スクリプトのエラー注入モードを有効または無効にするには，joberrmode コマンドを実行します。joberrmode コマンドの短縮形は"jem"です。joberrmode コマンドの形式を次に示します。

```
joberrmode  {on|off}
```

on を選択した場合

エラー注入モードを有効にします。これによって，ジョブ内でエラーが発生した場合のケースをテストできます。すべての実行パスを実行しても C1 実行比率が 100%とならない場合に，エラーをシミュレートするために使用します。C1 実行比率が 100%とならないケースについては，「(5)　C1 実行比率 100%とならないケース」を参照してください。

エラー注入モードが有効な場合は次のように動作します。このとき，終了コードは変更しません。

- run 属性に abnormal または always が指定されている場合，ジョブステップを実行する。
- run 属性が省略されているまたは normal が指定されている場合，ジョブステップを実行しない。
- ジョブステップ外のコマンドは実行しない。
- ステップ正常ブロック内でエラー注入した場合，そのステップのステップエラーブロック内のコマンドは実行する。onError 属性に cont が指定されていても，そのステップ内の後続のコマンドは実行しないで，ステップエラーブロックを実行する。

エラー注入モードが on の場合に run コマンドを実行すると，エラー注入モードは off に戻ります。

CMDRC_CMDGRP_CHECK パラメータに FUNCTION を指定した場合，関数内で停止中に「joberrmode on」を実行しても，エラー注入モードは on に変更できません。

off を選択した場合

エラー注入モードを無効にします。

実行例 1

次に示すジョブ定義スクリプトを実行した場合について説明します。左側の行番号は，実行結果に出力される行番号との対応づけのため記載しています。

```
001  #!/bin/sh
002
003  #-adsh_step_start STEP001                  ←STEP001開始
004
005  ./cmd1                                     ←ステップ正常ブロック部
006
007  #-adsh_step_error                          ←ステップエラーブロック部
008
009  ./cmd2
010
011  #-adsh_step_end
012
013  ./cmd3
014
015  #-adsh_step_start STEP002 -run abnormal    ←STEP002開始　異常時に実行
016
017  ./cmd4                                     ←ステップ正常ブロック部
018
019  #-adsh_step_error                          ←ステップエラーブロック部
020
021  ./cmd5
022
023  #-adsh_step_end
024
025  ./cmd6
026
027  echo JOB01-ended
```

実行結果 1

ステップ外で停止し，joberrmode コマンドでエラー注入モードを有効にした場合の例を示します。行の右側に付加した番号に対する説明は，実行結果の下を参照してください。

```
[jobuser1@HOST01 joberrmode]$ adshexec -d joberrmode.ash                ←1.
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ち
ます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000004
(adshdb) b 2                                                            ←2.
KNAX7015-W 指定した行番号（”2”）にブレークポイントを設定できません。次に設定できる行に
ブレークポイントを設定します。
KNAX7018-I ブレークポイント ”1”: ファイル名=”joberrmode.ash” 行番号=3
(adshdb) b 17                                                           ←3.
KNAX7018-I ブレークポイント ”2”: ファイル名=”joberrmode.ash” 行番号=17
(adshdb) run                                                            ←4.
KNAX7007-I ジョブ定義スクリプトの実行を開始します。: /home/jobuser1/joberrmode/
joberrmode.ash
```

```
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000005
KNAX0091-I ADSH000005 ジョブが開始しました。
KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。

KNAX7018-I ブレークポイント ”1”: ファイル名=”joberrmode.ash” 行番号=3    ←5.
KNAX7032-I ジョブ定義スクリプトファイル（”joberrmode.ash”）の中で実行を停止しました。
3: #-adsh_step_start STEP001
現在位置: #-adsh_step_start STEP001
(adshdb) info status                                                  ←6.
joberrmode:off                                                        ←7.
(adshdb) joberrmode on                                                ←8.
KNAX7126-I エラー注入モードを ”on” に設定しました。
(adshdb) info status                                                  ←9.
joberrmode:on                                                        ←10.
(adshdb) c                                                           ←11.
KNAX7034-I ジョブ定義スクリプトの実行を継続します。
KNAX6508-I ADSH000005.STEP001 先行のジョブステップまたはコマンドがエラー終了したため，
ジョブステップをスキップしました。
KNAX0092-I ADSH000005.STEP002 ステップが開始しました。

KNAX7018-I ブレークポイント ”2”: ファイル名=”joberrmode.ash” 行番号=17  ←12.
KNAX7032-I ジョブ定義スクリプトファイル（”joberrmode.ash”）の中で実行を停止しました。
17: ./cmd4
現在位置: ./cmd4
(adshdb) joberrmode off
KNAX7127-E エラー注入モードの再設定に失敗しました。                     ←13.
(adshdb) c                                                           ←14.
KNAX7034-I ジョブ定義スクリプトの実行を継続します。
cmd4 start
cmd4 end
KNAX6116-I コマンド（./cmd4，行番号=17）が正常終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
KNAX6597-I ADSH000005.STEP002 ジョブステップが正常終了しました。rc=0 E-Time=4.666s C-
Time=0.000s
KNAX0101-E ADSH000005 ジョブを実行中にエラーが発生しました。
KNAX0098-I ADSH000005 ジョブが終了しました。rc=0 E-Time=15.203s C-Time=0.000s
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”/home/jobuser1/test6/spool/000005-ADSH000005/”

(adshdb) quit
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”/home/jobuser1/test6/spool/000004-ADSH000004/”
KNAX7999-I ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=0
```

実行結果の右側に付加した番号に従って解説します。

デバッグを開始する。

スクリプトの途中で停止させる。

エラー時に実行するステップ内で停止させる。

スクリプトを実行する。

最初のブレークポイントで停止する。

ステータスを表示する。

エラー注入モードは無効。

エラー注入モードを有効にする。

ステータスを表示する。

エラー注入モードは有効。

デバッグを再開する。

エラー時に実行するステップ内で停止する。

エラー注入モードの切り替えはできない。

デバッグを再開する。

実行結果 2

ステップ内で停止し，joberrmode コマンドでエラー注入モードを有効にした場合の例を示します。行の右側に付加した番号に対する説明は，実行結果の下を参照してください。

```
[jobuser1@HOST01 joberrmode]$ adshexec -d joberrmode.ash                ←1.
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ち
ます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000006
(adshdb) b 5                                                            ←2.
KNAX7018-I ブレークポイント ”1”: ファイル名=”joberrmode.ash” 行番号=5
(adshdb) b 17                                                           ←3.
KNAX7018-I ブレークポイント ”2”: ファイル名=”joberrmode.ash” 行番号=17
(adshdb) run                                                            ←4.
KNAX7007-I ジョブ定義スクリプトの実行を開始します。: /home/joberrmode/joberrmode/
joberrmode.ash

KNAX0724-I ジョブ識別子を割り当てました。Jobid=000007
KNAX0091-I ADSH000007 ジョブが開始しました。
KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
KNAX0092-I ADSH000007.STEP001 ステップが開始しました。

KNAX7018-I ブレークポイント ”1”: ファイル名=”joberrmode.ash” 行番号=5    ←5.
KNAX7032-I ジョブ定義スクリプトファイル（”joberrmode.ash”）の中で実行を停止しました。
5: ./cmd1
現在位置: ./cmd1
(adshdb) info status                                                    ←6.
joberrmode:off                                                          ←7.
(adshdb) joberrmode on                                                  ←8.
KNAX7126-I エラー注入モードを ”on” に設定しました。
(adshdb) info status                                                    ←9.
joberrmode:on                                                          ←10.
(adshdb) c                                                             ←11.
KNAX7034-I ジョブ定義スクリプトの実行を継続します。
cmd2 start
cmd2 end
KNAX6116-I コマンド（./cmd2, 行番号=9）が正常終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
KNAX6596-E ADSH000007.STEP001 ジョブステップがエラー終了しました。rc=0 E-Time=13.568s
C-Time=0.010s
KNAX0092-I ADSH000007.STEP002 ステップが開始しました。

KNAX7018-I ブレークポイント ”2”: ファイル名=”joberrmode.ash” 行番号=17  ←12.
KNAX7032-I ジョブ定義スクリプトファイル（”joberrmode.ash”）の中で実行を停止しました。
17: ./cmd4
現在位置: ./cmd4
(adshdb) joberrmode off
```

```
KNAX7127-E エラー注入モードの再設定に失敗しました。                    ←13.
(adshdb) c                                                          ←14.
KNAX7034-I ジョブ定義スクリプトの実行を継続します。
cmd4 start
cmd4 end
KNAX6116-I コマンド（./cmd4, 行番号=17）が正常終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
KNAX6597-I ADSH000007.STEP002 ジョブステップが正常終了しました。rc=0 E-Time=15.496s C-
Time=0.000s
KNAX0101-E ADSH000007 ジョブを実行中にエラーが発生しました。
KNAX0098-I ADSH000007 ジョブが終了しました。rc=0 E-Time=29.066s C-Time=0.010s
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory="/home/joberrmode/test6/spool/000007-ADSH000007/"
```

実行結果の右側に付加した番号に従って解説します。

デバッグを開始する。

ステップ内のスクリプトの途中で止める。

エラー時に実行するステップ内で止める。

スクリプトを実行する。

最初のブレークポイントで停止する。

ステータスを表示する。

エラー注入モードは無効。

エラー注入モードを有効にする。

ステータスを表示する。

エラー注入モードは有効。

デバッグを再開する。

エラー時に実行するステップ内で停止する。

エラー注入モードの切り替えはできない。

デバッグを再開する。

実行例 2

次に示すジョブ定義スクリプトを実行した場合について説明します。左側の行番号は，実行結果に出力される行番号との対応づけのため記載しています。

```
001  #-adsh_job JOB001
002  #-adsh_step_start STEP001 -onError cont  ←STEP001開始 -onError cont指定
003   ./cmd1                                  ←ステップ正常ブロック部
004   ./cmd2
005   ./cmd3
006  #-adsh_step_error
007   ./cmd4                                  ←ステップエラーブロック部
008   ./cmd5
009  #-adsh_step_end
010   ./cmd6                                  ←ステップ外のコマンド
```

実行結果

ステップ内（4 行目）で停止し，joberrmode コマンドを投入して実行した場合の例を示します。行の右側に付加した番号に対する説明は，実行結果の下を参照してください。

```
[jobuser1@HOST01 joberrmode]$ adshexec -d test_cont.ash                ←1.
KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ち
ます。
KNAX0724-I ジョブ識別子を割り当てました。Jobid=000008
(adshdb) b 4                                                           ←2.
KNAX7018-I ブレークポイント ”1”: ファイル名=”test_cont.ash” 行番号=4
(adshdb) r                                                             ←3.
KNAX7007-I ジョブ定義スクリプトの実行を開始します。: /home/jobuser1/joberrmode/
test_cont.ash

KNAX0724-I ジョブ識別子を割り当てました。Jobid=000009
KNAX0091-I JOB001 ジョブが開始しました。
KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
KNAX0092-I JOB001.STEP001 ステップが開始しました。
cmd1 start
cmd1 end
KNAX6116-I コマンド（./cmd1, 行番号=3）が正常終了しました。rc=0 E-Time=0.001s C-
Time=0.000s

KNAX7018-I ブレークポイント ”1”: ファイル名=”test_cont.ash” 行番号=4     ←4.
KNAX7032-I ジョブ定義スクリプトファイル（”test_cont.ash”）の中で実行を停止しました。
4:  ./cmd2
現在位置: ./cmd2
(adshdb) info status                                                   ←5.
joberrmode:off                                                         ←6.
(adshdb) jem on                                                        ←7.
KNAX7126-I エラー注入モードを ”on” に設定しました。
(adshdb) info status                                                   ←8.
joberrmode:on                                                          ←9.
(adshdb) c                                                            ←10.
KNAX7034-I ジョブ定義スクリプトの実行を継続します。
cmd4 start                                                            ←11.
cmd4 end
KNAX6116-I コマンド（./cmd4, 行番号=7）が正常終了しました。rc=0 E-Time=0.002s C-
Time=0.000s
cmd5 start
cmd5 end
KNAX6116-I コマンド（./cmd5, 行番号=8）が正常終了しました。rc=0 E-Time=0.001s C-
Time=0.000s
KNAX6596-E JOB001.STEP001 ジョブステップがエラー終了しました。rc=0 E-Time=34.884s C-
Time=0.000s
KNAX0101-E JOB001 ジョブを実行中にエラーが発生しました。
KNAX0098-I JOB001 ジョブが終了しました。rc=0 E-Time=34.886s C-Time=0.000s
KNAX6380-I ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job
directory=”/home/jobuser1/test6/spool/000009-JOB001/”
```

実行結果の右側に付加した番号に従って解説します。

デバッグを開始する。

ステップ内のスクリプトの途中で止める。

スクリプトを実行する。

最初のブレークポイントで停止する。

ステータスを表示する。

エラー注入モードは無効。

エラー注入モードを有効にする。
ステータスを表示する。
エラー注入モードは有効。
デバッグを再開する。
ステップエラーブロック部のコマンドを実行する。

6.2.22　変数の値を設定する（set コマンド）

シェル変数の値を設定するコマンドは，set コマンドです。引数に代入式を指定することで，その式を評価して変数の値を設定できます。set コマンドの短縮形はありません。set コマンドの形式を次に示します。

```
set 代入式
```

代入式の形式を次に示します。

```
変数名= {変数名|数値|”文字列”}
```

- **変数名**（左辺）：シェル変数名です。指定した変数に右辺の値が代入されます。
- **変数名**（右辺）：シェル変数名です。指定した変数の値が左辺の変数に代入されます。
- **数値**：整数値です。指定した整数値が左辺の変数に代入されます。
- **文字列**：文字列です。指定した文字列が左辺の変数に代入されます。

set コマンドを実行した場合の動作を次に示します。

set コマンドの後ろに引数を指定した場合

- 代入式
  指定した代入式に応じて変数の値を設定します。
- 作成されていない変数名が含まれる代入式
  エラーメッセージが出力されます。
- 代入式以外
  エラーメッセージが出力されます。

set コマンドの後ろに引数を指定しない場合

エラーメッセージが出力されます。

注意事項

- run コマンドによるジョブ定義スクリプト実行中以外では，変数情報が設定されていないため，エラーとなります。
- 文字列を指定する場合は，ダブルクォーテーションで囲んでください。文字列の中にダブルクォーテーションを使用するときは，¥を付けて¥"と指定します。また，¥"は¥¥¥"と指定します。

- 代入式の中で最初に出現する等号（=）以降の引数を，変数名および数値または文字列として処理します。
- 変数名に配列を指定する場合は，要素ごとに指定します。
- 変数名の指定に$を付けないでください。
- 数値の指定時に signed long 型の範囲を超えた数値を入力した場合，signed long 型の上限値または下限値に丸めて扱います。
- 代入式の右辺に指定した変数に格納されている数値については，値を丸めることなくそのまま左辺に代入します。
- スクリプト制御文の for 文の実行前で停止している場合，for 文の wordlists に指定した変数の値は固定です。for 文の実行によって代入される変数の値を書き換えたいときは，for 文の do 以降で最初に停止した位置で，set コマンドを使用して変数の値を書き換える，または for 文に到達する前に，set コマンドを使用して wordlists の変数の値を書き換えてください。例を次に示します。

例 1

```
1: a=1
2: b=2
3: date
4: for num in $a $b
5: do
6:  echo $num  ←numの値を書き換えたい場合，6行目の実行前に行う
7:  pwd
8: done
```

例 2

```
1: a=1
2: b=2
3: date        ←$aおよび$bの値をfor文で代入される変数に
                 反映する場合，3行目の実行前に行う
4: for num in $a $b
5: do
6:  echo $num
7:  pwd
8: done
```

使用例

数値を代入する場合，変数に対して typeset コマンドの-i を指定し，整数型として宣言しておく必要があります。

変数「a」に数値「10」を代入する場合

```
(adshdb) set a=10
```

変数「b」に文字列「test」を代入する場合

```
(adshdb) set b="test"
```

変数「c」に変数「a」の値を代入する場合

```
(adshdb) set c=a
```

変数「d[5]」（配列）に数値「1」を代入する場合

```
(adshdb) set d[5]=1
```

「e[5][1]」（配列）に数値「2」を代入する場合

```
(adshdb) set e[5][1]=2
```

6.2.23　変数の値を表示する（print コマンド）

ジョブ定義スクリプト内の変数の値を表示するコマンドは，print コマンドです。引数に変数名を指定することで，その変数の値を表示できます。print コマンドの短縮形は"p"です。print コマンドの形式を次に示します。

```
print 変数名
```

print コマンドを実行した場合の動作を次に示します。

print コマンドの後ろに引数を指定した場合

変数が定義されているときは，指定した変数の値を表示します。表示形式を次に示します。

```
変数値
```

- **変数値**：指定した変数の値です。値がない場合，<No value>と表示します。

変数が定義されていないときは，エラーメッセージが出力されます。

print コマンドの後ろに引数を指定しない場合

エラーメッセージが出力されます。

注意事項

- run コマンドによるジョブ定義スクリプト実行中以外では，変数情報が設定されていないため，エラーとなります。
- 変数名に配列を指定する場合は，要素ごとに指定します。
- 変数名の指定に$を付けないでください。

使用例

変数「a」の値を表示する場合

```
(adshdb) print a
```

変数「b[1]」（配列）の値を表示する場合

```
(adshdb) print b[1]
```

変数「c[0][1]」（2 次元配列）の値を表示する場合

```
(adshdb) print c[0][1]
```

6.2.24　バックトレースを表示する（where コマンド）

バックトレースとは，実行中のジョブ定義スクリプトが現在停止している位置にどのようにして到達したかを示す情報です。バックトレースは，各フレームによって表現されます。フレームとは，ある関数に対する 1 回の呼び出しに関連するデータのことです。関数を呼び出すとフレームは 1 個追加され，関数を終了するとフレームは 1 個削除されます。各フレームに，最も内側のフレームから順に番号を 0 から割り当てます。最も内側のフレームとは，現在実行中の関数のことです。バックトレースを表示するコマンドは，where コマンドです。where コマンドの短縮形は"whe"です。where コマンドの形式を次に示します。

```
where [ フレーム番号]
```

引数にフレームの番号を指定すると，最も内側のフレームから指定した番号までの間で存在するフレームの情報を表示します。引数を省略すると，すべてのフレーム情報を内側のフレームから表示します。表示形式を次に示します。

```
Num  Function  File:Line
フレーム番号  関数名  ファイル名:行番号
…
```

- **フレーム番号**：フレームの番号です。フレーム番号を 0 番から昇順に表示します。左詰めで 3 桁まで表示できます。
- **関数名**：フレームに対応する関数名に，関数を呼び出したジョブ定義スクリプトファイル名を付けた名称です。関数が呼び出されていない状態の場合，関数名を<main>として表示します。63 バイトまで表示できます。
- **ファイル名**：フレームに対応するジョブ定義スクリプトファイル名です。フレーム番号が 0 の場合は，現在停止中のファイル名を示します。フレーム番号が 1 以上の場合は，新しい関数を呼び出すときのファイル名を示します。
- **行番号**：フレームに対応する行番号です。フレーム番号が 0 の場合は，現在停止中の行番号を示します。フレーム番号が 1 以上の場合は，新しい関数を呼び出すときの行番号を示します。

  ジョブ定義スクリプト終端で停止している場合は<EOF>と表示します。trap コマンドの action を実行中に停止している場合は<Trap action>と表示します。

where コマンドを実行した場合の動作を次に示します。

where コマンドの後ろに引数を指定しない場合

ジョブ定義スクリプトが実行中で停止している状態で where コマンドを実行すると，すべてのフレーム情報を内側のフレームから順に表示します。

ジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

where コマンドの後ろに引数を指定した場合

ジョブ定義スクリプトが実行されている状態で where コマンドを実行すると，正しいフレーム番号を指定したときは，内側のフレームから指定したフレーム番号までのフレーム情報を表示します。

番号以外を指定したときおよびジョブ定義スクリプトが実行されていないときは，エラーメッセージが出力されます。

注意事項

- フレーム番号は 0 以上の整数を入力してください。先頭に「+」は付けません。
- フレーム番号に int 型の範囲を超えた数値を入力した場合，int 型の上限値に丸めて扱います。
- 表示できるフレーム数の上限は 255 個です。引数に 255 以上の番号を入力してもそれ以上の番号のフレームは表示されません。256 個以上フレームが存在する場合は，フレーム情報の下にメッセージが表示されます。

使用例

sample.ash の 12 行目で funcA を呼び出し，そのあと 9 行目で funcB を呼び出して test.ash の 12 行目で停止しているときに，where コマンドを実行します。

sample.ash

```
5: #-adsh_script test.ash
6:
7: funcA(){
8:   num=10
9:   funcB
10: }
11:
12: funcA
```

test.ash

```
10: funcB(){
11:   val=5
12:   num=20
13: }
```

```
Num  Function                File:Line
0    funcB (in sample.ash)   test.ash:12
1    funcA (in sample.ash)   sample.ash:9
2    <main>                  sample.ash:12
```

6.2.25　ソースファイルを表示する（list コマンド）

ソースファイルを表示するコマンドは，list コマンドです。list コマンドの短縮形は"l"です。list コマンドの形式を次に示します。

ファイル名を指定しない場合

```
list [ 行番号]
```

引数を省略すると，現在の行に停止して最初に list コマンドを実行した場合は，現在停止中の行の 5 行前から行番号付きで 11 行表示します。停止してから 2 回目以降の入力の場合は，前回表示した最終行の次の行から 11 行表示します。

ファイル名を指定する場合

```
list ジョブ定義スクリプトファイル名:行番号
```

引数に":"を使用すると，ジョブ定義スクリプトファイル名を指定して表示できます。

行番号を指定すると，その行の 5 行前から行番号付きで 11 行表示します。表示形式を次に示します。

```
行番号: ソースファイル行
…
```

- **行番号**：ソースファイルの行番号です。
- **ソースファイル行**：ソースファイルの行の内容です。

list コマンドを実行した場合の動作を次に示します。

list コマンドの後ろに引数を指定しない場合

現在の行に停止して最初に list コマンドを実行した場合は，現在停止中の行の 5 行前からソースファイルを行番号付きで 11 行表示します。

2 回目以降に list コマンドを実行した場合は，前回表示したファイルの最終行の次の行から 11 行表示します。

list コマンドの後ろに引数を指定した場合

存在する行番号を指定したときは，指定した行の 5 行前からソースファイルを行番号付きで 11 行表示します。

存在しない行番号を指定したときおよび上記以外のときは，エラーメッセージが出力されます。

注意事項

- ファイル名を指定する場合には，バッチジョブを実行するコマンドに指定したファイルまたはスクリプト拡張コマンドの#-adsh_script に指定したファイルを指定してください。
- ファイル名の指定は，最後に入力された「:」の前までの文字列をファイル名と見なします。
- 引数の有無に関係なく，ソースファイルの表示範囲に 1 より小さい行が含まれる場合，1 行目から 11 行表示します。
- 引数の有無に関係なく，ソースファイルの表示範囲に最終行より大きい行が含まれる場合，最終行の 10 行前から最終行までの 11 行を表示します。
- 行番号は 0 以上の整数を入力してください。先頭に「+」は付けません。
- 行番号の指定時に int 型の範囲を超えた数値を入力した場合，int 型の上限値に丸めて扱います。
- デバッグ実行中のジョブ定義スクリプトが次に示すどちらかの状態の場合は，list コマンドを引数を指定しないで実行するとエラーとなります。ただし，引数を指定して実行したあとは，続きを表示できます。

  ・ジョブ定義スクリプト終端(EOF)で停止している

・trap コマンドの action を実行中に停止している

出力例

20 行目で停止しているときに引数を省略して list コマンドを実行します。

```
15: echo ”start”
16:
17: a=1
18: while [[ $a -ne 10 ]]
19: do
20:   echo $a
21:   let a+=1
22: done
23:
24: pwd
25: echo ”end”
```

6.2.26　ディレクトリを移動する（cd コマンド）

デバッガの作業ディレクトリを移動するコマンドは，cd コマンドです。cd コマンドの短縮形はありません。cd コマンドの形式を次に示します。

```
cd ディレクトリパス名
```

引数にディレクトリパス名を指定すると，作業ディレクトリを移動できます。

cd コマンドを実行した場合の動作を次に示します。

cd コマンドの後ろに引数を指定した場合

指定したディレクトリにデバッガの作業ディレクトリを移動します。移動先のディレクトリの絶対パスを表示します。

ただし，ジョブ定義スクリプトが実行されている場合は，デバッガの作業ディレクトリだけを移動し，実行中のジョブ定義スクリプトのカレントディレクトリは変更しません。

なお，実行権限のないディレクトリパス名を指定した場合および存在しないディレクトリパス名を指定した場合など，デバッガの作業ディレクトリを移動できなかったときは，エラーメッセージが出力されます。

cd コマンドの後ろに引数を指定しない場合

エラーメッセージが出力されます。

使用例

cd コマンドで作業ディレクトリを移動します。そのあとに，exec コマンドを使用して外部コマンドを実行し，移動先のディレクトリに存在するファイルの内容を出力します。

```
(adshdb) cd work
KNAX7048-I ディレクトリパス（/home/xxx/work）に作業ディレクトリを移動しました。
(adshdb) exec cat test.txt
```

```
aaa bbb ccc
(adshdb)
```

6.2.27　ログインシェルを起動する（exec コマンド）

デバッグ中にログインシェルを起動するコマンドは，exec コマンドです。ログインシェルとは，シェル変数 SHELL に設定されているシェルを示します。exec コマンドの短縮形は"ex"です。exec コマンドの形式を次に示します。

```
exec [ ログインシェルに渡す引数] …
```

ログインシェルに渡す引数を指定すると，指定した引数をログインシェルに渡して実行します。引数を省略すると，ログインシェルを起動します。

注意事項

- 引数に&が含まれている場合，エラーメッセージが出力されます。
- バックグラウンド実行以外の目的で引数に&を使用する場合，代わりに¥&で指定します。
- 引数の個数に上限はありません。

使用例

exec コマンドでシェルの ls コマンドを実行します。

```
(adshdb) exec ls
aaa.txt  bbb.log  bin
```

6.2.28　ヘルプを表示する（help コマンド）

デバッガコマンドのヘルプを表示するコマンドは，help コマンドです。help コマンドの短縮形は"h"です。help コマンドの形式を次に示します。

```
help [ コマンド名]
```

引数にコマンド名を指定すると，そのコマンドの説明を表示します。引数を省略すると，すべてのコマンド名を表示します。また，引数に指定するコマンド名は短縮形で指定できます。

help コマンドを実行した場合の動作を次に示します。

help コマンドの後ろに引数を指定しない場合

すべてのコマンド名を表示します。表示形式を次に示します。

```
使用可能なコマンド一覧:
break      cd         continue   delete     exec
finish     help       info       joberrmode kill
```

```
list       next       print      quit       return
run        set        signal     step       watch
where
```

help コマンドの後ろに引数を指定した場合

コマンド名を指定したときは，指定したコマンドの使用方法を表示します。

存在しないコマンドを指定したときは，エラーメッセージが出力されます。

7 環境ファイルで設定するパラメーター

環境ファイルにパラメーターを設定すると，終了コード，カバレージ，システム実行ログ，ディレクトリのパス，環境変数などについて定義できます。この章では，パラメーターの記述形式と詳細について説明します。

7.1　環境ファイルの記述形式

環境ファイルで設定するパラメーターの記述形式について説明します。

パラメーターには次の種類があります。

パラメーターの種類	定義内容
環境設定パラメーター	終了コード，カバレージ，システム実行ログ，ディレクトリのパスなどを定義します。
export パラメーター	環境変数を定義します。
条件パラメーター	物理ホストまたは特定の論理ホストだけで有効とする環境設定パラメーターおよび export パラメーターを指定します。

環境ファイルの 1 行の長さは，コメントや区切り文字も含めて 4,092 バイト以内としてください。4,092 バイトを超えると解析エラーとなります。また，環境ファイルには，コメント内も含めて&（アンパーサンド）を記載しないでください。

環境ファイルで設定するパラメーターは，次の点に注意してください。

- 行の途中に NULL（"0x00"または C 言語での"¥0"）が混入している場合，ジョブコントローラはその行で NULL が現れる部分までを 1 行と見なします。その行で NULL のあとにほかの文字列があっても無視されます。不正な実行結果や実行時エラーの要因となるため，NULL を記述しないようにしてください。
- 環境ファイルのエンコーディングと，ジョブ定義スクリプトを実行する環境の LANG 環境変数の値は一致させてください。
- #の後ろが"-adsh_conf△$_1$"でない場合は，コメントになります。

7.1.1　パラメーターの記述形式

(1)　環境設定パラメーターの記述形式

環境設定パラメーターの記述形式を次に示します。

```
△0#-adsh_conf△1パラメーター△1値
```

- #-adsh_conf に続いてパラメーターを 1 行で記述します。
- パラメーターの値の後ろには何も記述しないでください。
- スペースを含む値をパラメーターに指定する場合は，その値を"（ダブルクォーテーション）で囲んでください。そのほかのエスケープ文字は使用できません。

(2) export パラメーターの記述形式

export パラメーターの記述形式を次に示します。

```
△0export△1環境変数名=環境変数値
```

- export パラメーターには，1 行で 1 個の環境変数を記述します。
- 環境変数 PATH 以外の環境変数を参照することはできません。

  例えば，次のように指定しても，環境変数 HOME の内容は展開されないで，「${HOME}」という文字列がそのまま環境変数 NEWHOME に設定されます。

  ```
  export NEWHOME=${HOME}
  ```

- スペースを含む値を指定する場合は"（ダブルクォーテーション）または'（シングルクォーテーション）で囲んでください。
- ¥はエスケープ文字として動作します（'（シングルクォーテーション）で囲まれた範囲は通常の文字として扱われます）。

(3) 条件パラメーターの記述形式

条件パラメーターの記述形式を次に示します。

```
△0#-adsh_conf△1 [phost_start | lhost_start△1 ホスト名]
環境設定パラメーターまたはexportパラメーター
      :
△0#-adsh_conf△1 [phost_end | lhost_end]
```

- 環境設定パラメーターと export パラメーターは複数記述できます。
- 物理ホストまたは論理ホストだけで有効とする環境設定パラメーターおよび export パラメーターを設定する場合，前後の行を条件パラメーターで囲んで設定します。
- パラメーターの値の後ろには何も記述しないでください。
- 条件パラメーターは複数指定できます。ただし，ネストした指定はできません（例 2 を参照のこと）。

例 1：複数指定する場合

```
#-adsh_conf phost_start
export HOME=/home/phost
#-adsh_conf phost_end

#-adsh_conf phost_start
export TEMP=/tmp
#-adsh_conf phost_end
```

例 2：ネストした指定の場合（エラーになる）

```
#-adsh_conf phost_start
export HOME=/home/phost
```

```
#-adsh_conf phost_start
export TEMP=/tmp
#-adsh_conf phost_end
#-adsh_conf phost_end
```

7.1.2　コメントの記述形式

コメントの記述形式を次に示します。

△$_0$#<「-adsh_conf」で開始しない任意文字列>

#に続いて「-adsh_conf△$_1$」でない文字列を記述すると，#以降から行末までをコメントとして扱います。

7.2　パラメーターの一覧

7.2.1　環境設定パラメーターの一覧

(1)　環境設定パラメーターの定義

環境設定パラメーターはシステム環境ファイルおよびジョブ環境ファイルに定義します。これらのファイルについての説明は「2.6.1　環境ファイルを設定する」を参照してください。

システム環境ファイルとジョブ環境ファイルに指定できるパラメーターは同じです。システム環境ファイルとジョブ環境ファイルの指定の有無によって，適用される指定値は次のようになります。

システム環境ファイル	ジョブ環境ファイル	
	指定なし	指定あり
指定なし	各パラメーターの省略値	ジョブ環境ファイルの指定値
指定あり	システム環境ファイルの指定値	パラメーターの仕様に従う※

注※　同じパラメーターを指定した場合の扱いは次のようになります。

- 指定個数の上限値が 1 の環境設定パラメーター
  ジョブ環境ファイルの指定が優先されます。
- 指定個数の上限値が 1 以外の環境設定パラメーター
  ジョブ環境ファイルの指定を先頭にして両方の指定をマージします。
  例
  ＜システム環境ファイルの指定＞
  ```
  #-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥abc
  #-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥def
  ```
  ＜ジョブ環境ファイルの指定＞
  ```
  #-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥ghi
  #-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥def
  #-adsh_conf PATH_CONV /jp1as/kkk c:¥¥jp1as¥¥kkk
  ```
  ＜解析結果は次の内容と同等となる＞
  ```
  #-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥ghi …1.
  #-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥def …1.
  #-adsh_conf PATH_CONV /jp1as/kkk c:¥¥jp1as¥¥kkk …1.
  #-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥abc …2.
  #-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥def …2.
  ```

注

1.：ジョブ環境ファイルの指定

2.：システム環境ファイルの指定

(2)　環境設定パラメーターの種類

JP1/Advanced Shell の環境ファイルに設定する環境設定パラメーターを次の表に示します。これらのパラメーターの指定は任意です。

表 7-1　JP1/Advanced Shell の環境ファイルに設定する環境設定パラメーター

パラメーター名	定義内容	指定個数の上限値※1	子孫ジョブ起動時の変更可否
ADSHCMD_RC_ERROR	スクリプト拡張コマンドが失敗したときの終了コードを定義します。	1	○
ADSHCMD_RC_SUCCESS	スクリプト拡張コマンドが成功したときの終了コードを定義します。	1	○
ASC_FILE	カバレージ採取の一括有効化機能で使用する蓄積ファイル名の生成規則を定義します。	1	－
BATCH_CVR	カバレージ採取の一括有効化機能を有効にします。	1	－
CHILDJOB_EXT	子孫ジョブとして実行するジョブ定義スクリプトファイルの拡張子を定義します。	255	○
CHILDJOB_PGM	子孫ジョブとして実行するように読み替えるプログラムパスを定義します。	255	○
CHILDJOB_SHEBANG	子孫ジョブとして実行するジョブ定義スクリプトファイルの実行プログラムパスを定義します。	255	○
CMDRC_CMDGRP_CHECK	関数の終了コードに従ってジョブおよびジョブステップのエラー判定をするかどうかを定義します。	1	○
CMDRC_THRESHOLD_DEFINE	コマンドの終了コードのしきい値を定義します。シェルスクリプトや子孫ジョブの終了コードに対しても定義できます。	無制限※3	○
CMDRC_THRESHOLD_USE_PRESET	UNIX 互換コマンドの終了コードのしきい値を定義します。	1	○
CMDSUB_PROCESS 【Windows 限定】	コマンド置換の動作を定義します。	1	○
COMMAND_CONV_ARG	コマンド実行時にジョブ定義スクリプト中の引数を変換する規則を定義します。	255	○
COMPATIBLE_CMD_EXEC	外部コマンドの起動方法を定義します。	1	○

パラメーター名	定義内容	指定個数の上限値※1	子孫ジョブ起動時の変更可否
COMPATIBLE_CMDSUB 【UNIX 限定】	コマンド置換の動作を定義します。	1	○
ESCAPE_SEQ_ECHO_DEFAULT	エスケープ文字関連のオプション省略時のecho コマンドの動作を定義します。	1	○
ESCAPE_SEQ_ECHO_HEX	16 進数表記の ASCII コード文字をエスケープ文字として解釈するか定義します。	1	○
EVENT_COLLECT	ジョブ定義スクリプト稼働実績情報取得機能を有効とするかどうかを定義します。	1	○
HOSTNAME_JP1IM_MANAGER※2	ユーザー応答機能で，JP1 イベントの送信先である JP1/IM - Manager が稼働している運用管理サーバを指定します。	1	×
INIT_SCRIPT_READ	ジョブコントローラ起動時に初期設定スクリプトファイルを読み込み，実行するかどうかを指定します。	1	−
JOBEXECLOG_PRINT	ジョブ終了時に標準エラー出力へ出力するジョブ実行ログの内容を定義します。	1	−
JOBLOG_SUPPRESS_MSG	ジョブ実行ログへの出力を抑止するメッセージを定義します。	無制限※3	○
KSH_ENV_READ	シェル変数ENV を読み込むかどうかを定義します。	1	○
LOG_DIR※4	システム実行ログを出力するディレクトリのパス名を定義します。	1	○
LOG_FILE_CNT※5	システム実行ログをバックアップする面数を定義します。	1	○
LOG_FILE_SIZE※5	システム実行ログを出力するファイルサイズを定義します。	1	○
OUTPUT_MODE_CHILD	子孫ジョブの終了時に，ジョブ実行ログを標準エラー出力へ出力するかどうかを定義します。	1	○
OUTPUT_MODE_ROOT	ルートジョブの終了時に，ジョブ実行ログを標準エラー出力へ出力するかどうかを定義します。	1	−
OUTPUT_STDOUT	ルートジョブの標準出力の出力先を定義します。	1	−
PATH_CONV	変換前・変換後のパス名を定義します。	255	○
PATH_CONV_ACCESS	変換前・変換後のパス名を定義します。この変換はファイルの入出力時に実行されます。	255	○
PATH_CONV_ENABLE	パス変換機能を有効にします。	1	○
PATH_CONV_NOVAR	パス名を扱わないシェル変数を定義します。	無制限※3	○

パラメーター名	定義内容	指定個数の上限値※1	子孫ジョブ起動時の変更可否
PATH_CONV_RULE 【Windows 限定】	パス変換ルールを定義します。	1	○
PATH_CONV_VAR	パス名を扱うシェル変数を定義します。	無制限※3	○
PERMISSION_SPOOLJOB_DIR 【UNIX 限定】	スプールジョブディレクトリのパーミッションを定義します。	1	－
PERMISSION_SPOOLJOB_FILE 【UNIX 限定】	スプールジョブディレクトリ下のファイルのパーミッションを定義します。	1	－
PIPE_CMD_LAST	パイプの最終コマンドの実行プロセスを定義します。	1	○
SPOOL_DIR※2，※4，※6	スプールルートディレクトリのパス名を定義します。	1	×
SPOOLJOB_CHILDJOB	子孫ジョブの終了時に，スプールジョブを削除するか，ルートジョブのスプールジョブへマージするかを定義します。	1	△
SPOOLJOB_CREATE	ジョブ定義スクリプトの実行時にスプールジョブを作成するかどうかを定義します。	1	△
TEMP_FILE_DIR※4	一時ファイルを格納するディレクトリのパス名を定義します。	1	○
TRACE_DIR※4	トレースを出力するディレクトリのパス名を定義します。	1	○
TRACE_FILE_CNT※7	トレースを出力する面数を定義します。	1	○
TRACE_FILE_SIZE※7	トレースを出力するファイルサイズを定義します。	1	○
TRACE_LEVEL	トレースを出力するレベルを定義します。	1	○
TRAP_ACTION_SIGTERM	ジョブコントローラが強制終了要求を受けたときの動作を定義します。	1	○
UMASK_INHERIT	ジョブ定義スクリプト実行開始時に親プロセスのファイルモード作成マスクを継承するかどうかを指定します。	1	○
UNSUPPORT_TEST 【Windows 限定】	サポートしていない条件式を実行した場合の動作を定義します。	条件ごとに 1	○
USERREPLY_DEBUG_DESTINATION	ユーザー応答機能で，デバッグ実行時の事象通知メッセージと応答要求メッセージの入出力先を定義します。	1	×

パラメーター名	定義内容	指定個数の上限値※1	子孫ジョブ起動時の変更可否
USERREPLY_JP1EVENT_INTERVAL※2	ユーザー応答機能で，adshecho コマンドや adshread コマンドによる JP1 イベントの最小発行間隔を定義します。	1	×
USERREPLY_WAIT_MAXCOUNT※2	ユーザー応答機能で，物理ホストまたは論理ホストごとに，応答要求メッセージの最大同時出力数を定義します。	1	×
VAR_ENV_NAME_LOWERCASE 【Windows 限定】	小文字が含まれる環境変数名を有効にするかどうかを定義します。	1	○
VAR_SHELL_FUNCINFO	シェル関数の情報を管理する配列を定義します。	1	○
VAR_SHELL_GETLENGTH	変数値の長さを参照する場合の長さの単位を定義します。	1	○

（凡例）

○：ルートジョブ起動時の設定値は子孫ジョブ起動時に変更しても動作しますが，推奨しません。

△：ルートジョブ起動時の設定値は子孫ジョブ起動時に変更しても有効になりません。

×：ルートジョブ起動時の設定値は子孫ジョブ起動時に変更しないでください。正しく動作しないおそれがあります。

－：子孫ジョブには該当しません。

注※1

システム環境ファイルとジョブ環境ファイルの両方の指定を有効とするパラメーターの場合，両方のファイルの指定数の合計がこの上限値を超えないよう指定してください。

各パラメーターでシステム環境ファイルとジョブ環境ファイルの両方が指定できるかどうかは，各パラメーターの説明を参照してください。

注※2

ユーザー応答機能を利用する場合，これらのパラメーターはシステム環境ファイルに指定する必要があります。また，システム環境ファイルを変更した場合，ユーザー応答機能管理デーモン・サービスを再起動してください。

それに加えて，パラメーターの指定に関しては次の点に注意してください。

- これらのパラメーターはジョブ環境ファイルに指定しないでください。次の問題が発生するおそれがあります。

  HOSTNAME_JP1IM_MANAGER：adshchmsg コマンドから応答を入力した場合や応答をキャンセルした場合，ユーザー応答機能管理デーモン・サービスの停止時に，応答待ちイベントが JP1/IM - View の滞留から解除されません。

  USERREPLY_WAIT_MAXCOUNT：ジョブ環境ファイルの指定は無視されます。

  USERREPLY_JP1EVENT_INTERVAL：JP1/IM - View に負荷が掛かります。

- 次のパラメーターをシステム環境ファイルとジョブ環境ファイルで異なる値を指定した場合は，次の問題が発生するおそれがあります。ユーザー応答機能を使用するときには，次のパラメーターはジョブ環境ファイルに指定しないでください。

  SPOOL_DIR：応答要求メッセージの出力に失敗します。

注※3

使用できるメモリ量で制限されます。

注※4

該当するパラメーターで指定するディレクトリを別々に分けることで，同一ホスト内で複数環境を使い分けられます。

注※5

複数のユーザーが同じファイルにシステム実行ログを出力している場合には，LOG_FILE_CNT と LOG_FILE_SIZE は，最後にシステム実行ログの出力を開始したユーザーの設定値が有効になります。

注※6

クラスタ運用で待機系のホストに情報を引き継ぎたい場合は，引き継ぐディレクトリをホスト間で共用します。その場合は，少なくともこのパラメーターのディレクトリをホスト間で共用します。

注※7

複数のユーザーが同じファイルにトレースログを出力している場合には，TRACE_FILE_CNT と TRACE_FILE_SIZE は，それぞれ，より大きい値を指定したユーザーの指定が有効になります。

また，環境ファイルでTRACE_FILE_CNT とTRACE_FILE_SIZE を変更した場合は，既存のトレースファイルの面数およびファイルサイズの設定値と比較して，それぞれ，より大きい値を指定したユーザーの指定が有効になります。

トレースファイルの面数およびファイルサイズを小さくする場合は，トレースフォルダにあるファイルをすべて削除してください。トレースフォルダにあるファイルをすべて削除するときは，同じトレースファイルにトレースを出力しているジョブがないことを確認してから行ってください。

7.2.2　export パラメーター

(1)　export パラメーターの定義

export パラメーターはシステム環境ファイルおよびジョブ環境ファイルに定義します。これらのファイルについての説明は「2.6.1　環境ファイルを設定する」を参照してください。

システム環境ファイルとジョブ環境ファイルの両方に指定した場合の扱いは次のようになります。

- 子孫ジョブの起動時にも環境ファイルを解析処理します。これによって，子孫ジョブ起動時に export パラメーターで指定した値が環境変数に再度設定されます。
- システム環境ファイルとジョブ環境ファイルの両方が有効であり，システム環境ファイル，ジョブ環境ファイルの順番で実行します。

例

＜システム環境ファイル＞

```
export A=s1
export B=s2
export A=s3
```

＜ジョブ環境ファイル＞

```
export C=j1
export B=j2
```

＜次の順に実行します＞

```
export A=s1
export B=s2
export A=s3
export C=j1
export B=j2
```

- PATH 環境変数に任意のパスを追加する場合の例を示します。

```
export   PATH='d:¥user¥prg;${PATH}'
export   PATH='/user/prg:${PATH}'
```

PATH 環境変数にパスを追加する場合，環境変数に指定できる文字列の上限サイズを超えることがあります。これは，ルートジョブ開始時，および子孫ジョブ開始時にパスが追加されて，PATH 環境変数に指定する文字列が長くなるためです。このため，子孫ジョブ機能と併用する場合は上限サイズを超えないように注意が必要です。環境変数に格納できる文字列の上限サイズは 32,766 バイトです。

Windows では PATH 環境変数に長大な文字列を格納した場合，上限サイズに達していなくても，OS の API でエラーが発生することがあります。このため，上限サイズに迫るような文字列が PATH 環境変数に格納されないようにしてください。

上限サイズを超えてしまう場合には，ルートジョブの環境ファイルと子孫ジョブの環境ファイルを分けて，ルートジョブの環境ファイルだけに PATH 環境変数にパスを追加してください。

(2) export パラメーターの一覧

export パラメーターの定義条件を次に示します。このパラメーターの指定は任意です。このパラメーターはジョブコントローラだけが使用します。

パラメーター名	定義内容	指定個数の上限値
export	環境ファイルを使用するジョブコントローラ起動時に有効としたい環境変数を定義します。	無制限

7.2.3　条件パラメーターの一覧

(1)　条件パラメーターの定義

論理ホスト専用，または物理ホスト専用の環境設定パラメーターおよび export パラメーターを定義する場合は，条件パラメーターで囲んで定義します。

条件パラメーター外に記述してあるパラメーターは，すべてのホストで有効になります。条件パラメーターによって有効になったパラメーターと，条件パラメーター外で有効となった同一のパラメーターは，重複指定と見なされ，指定可能数の上限を超えていた場合はエラーになります。

条件パラメーターを次のように定義した場合について説明します。

```
(パラメーター群A)
#-adsh_conf lhost_start HOST01
(パラメーター群B)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(パラメーター群C)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(パラメーター群D)
#-adsh_conf phost_end
#-adsh_conf lhost_start HOST01
(パラメーター群E)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(パラメーター群F)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(パラメーター群G)
#-adsh_conf phost_end
(パラメーター群H)
```

この定義の場合，各ホストで実行されるパラメーター群は次のとおりです。

論理ホスト HOST01 で実行されるパラメーター群

```
(パラメーター群A)
(パラメーター群B)
(パラメーター群E)
(パラメーター群H)
```

論理ホスト HOST02 で実行されるパラメーター群

```
(パラメーター群A)
(パラメーター群C)
(パラメーター群F)
(パラメーター群H)
```

物理ホストで実行されるパラメーター群

```
(パラメーター群A)
(パラメーター群D)
(パラメーター群G)
(パラメーター群H)
```

条件パラメーターはシステム環境ファイルおよびジョブ環境ファイルに定義します。これらのファイルについての説明は「2.6.1　環境ファイルを設定する」を参照してください。

システム環境ファイルとジョブ環境ファイルに指定できるパラメーターは同じですが，同じパラメーターを指定した場合の扱いは次のようになります。

- システム環境ファイルとジョブ環境ファイルの両方が有効になります。
- 条件が一致して有効になったパラメーターはそれぞれのパラメーターの規則が適用されます。

例

＜システム環境ファイル＞

```
#-adsh_conf lhost_start host01
#-adsh_conf TEMP_FILE_DIR /jp1as/temp
#-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥sys1
#-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥sys2
#-adsh_conf lhost_end
```

＜ジョブ環境ファイル＞

```
#-adsh_conf lhost_start host01
#-adsh_conf TEMP_FILE_DIR /home/temp
#-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥job1
#-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥job2
#-adsh_conf PATH_CONV /jp1as/kkk c:¥¥jp1as¥¥job3
#-adsh_conf lhost_end
```

＜論理ホスト host01 でジョブを実行した場合の結果は次の内容と同等です＞

```
#-adsh_conf TEMP_FILE_DIR /home/temp
#-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥job1
#-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥job2
#-adsh_conf PATH_CONV /jp1as/kkk c:¥¥jp1as¥¥job3
#-adsh_conf PATH_CONV /jp1as/abc c:¥¥jp1as¥¥sys1
#-adsh_conf PATH_CONV /jp1as/def c:¥¥jp1as¥¥sys2
```

(2)　条件パラメーターの一覧

条件パラメーターを次の表に示します。これらのパラメーターの指定は任意です。

表 7-2　JP1/Advanced Shell の環境ファイルに設定する条件パラメーター

パラメーター名	定義内容	指定個数の上限値
lhost_start	特定の論理ホストだけに有効とする環境設定パラメーターおよび export パラメーターを開始します。 対象となる論理ホスト名もあわせて定義します。	無制限
lhost_end	lhost_start によって開始された，論理ホスト用の環境設定パラメーターと export パラメーターを終了します。このパラメーターは，lhost_start パラメーターと必ず対になるように定義します。	
phost_start	物理ホストだけに有効とする環境設定パラメーターおよび export パラメーターを開始します。	
phost_end	phost_start によって開始された，物理ホスト用の環境設定パラメーターと export パラメーターを終了します。このパラメーターは，phost_start パラメーターと必ず対になるように定義します。	

(3)　条件パラメーターの定義例

環境ファイルで設定するパラメーターの定義例を示します。

(a)　システム環境ファイルとジョブ環境ファイルの定義例

システム環境ファイルとジョブ環境ファイルの関係を定義例で説明します。

- 単一ホストで運用する場合

  システムのデフォルトとしてシステム環境ファイルを定義します。ここでは次の内容を定義します。

  - JP1 イベントの発行先としてホスト名「HostJp1IM」を定義する。
  - KNAX6110-I，KNAX6111-I メッセージの出力を抑止する。

  この場合のシステム環境ファイルの定義例は次のとおりです。

```
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6110-I
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6111-I
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM
```

  ジョブごとの設定変更をジョブ環境ファイルで定義します。特定のジョブでジョブ定義スクリプト中の引数を変換することや，スクリプト拡張コマンド失敗時の終了コードを定義することができます。

  ジョブ環境ファイルの定義例は次のとおりです。

```
#-adsh_conf  ADSHCMD_RC_ERROR  8
#-adsh_conf  COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp
```

- 論理ホスト HOST01，HOST02 を同時に運用する場合

  システムのデフォルトとしてシステム環境ファイルを定義します。ここでは次の内容を定義します。

  - KNAX6110-I，KNAX6111-I メッセージの出力を抑止する。

- SPOOL_DIR，LOG_DIR，TRACE_DIR，TEMP_FILE_DIR，HOSTNAME_JP1IM_MANAGER パラメーターを論理ホストごとに別々の定義にして実行環境を分ける。
- 環境変数 ABC の値を論理ホストごとに分ける。

この場合のシステム環境ファイルの定義例は次のとおりです。

```
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6110-I
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6111-I
#-adsh_conf  lhost_start HOST01
#-adsh_conf  SPOOL_DIR  /jp1as/host01/spool
#-adsh_conf  LOG_DIR    /jp1as/host01/log
#-adsh_conf  TRACE_DIR  /jp1as/host01/trace
#-adsh_conf  TEMP_FILE_DIR /jp1as/host01/temp
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM01
#-adsh_conf  lhost_end
#-adsh_conf  lhost_start HOST02
#-adsh_conf  SPOOL_DIR  /jp1as/host02/spool
#-adsh_conf  LOG_DIR    /jp1as/host02/log
#-adsh_conf  TRACE_DIR  /jp1as/host02/trace
#-adsh_conf  TEMP_FILE_DIR /jp1as/host02/temp
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM02
#-adsh_conf  lhost_end
#-adsh_conf  lhost_start HOST01
export  ABC=/jp1as/host01/abc
#-adsh_conf  lhost_end
#-adsh_conf  lhost_start HOST02
export  ABC=/jp1as/host02/abc
#-adsh_conf  lhost_end
```

ジョブごとの設定変更をジョブ環境ファイルで定義します。特定のジョブでジョブ定義スクリプト中の引数を論理ホストごとに異なる値に変換することや，スクリプト拡張コマンド失敗時の終了コードを定義することができます。

ジョブ環境ファイルの定義例は次のとおりです。

```
#-adsh_conf ADSHCMD_RC_ERROR  8
#-adsh_conf lhost_start HOST01
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp01
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp02
#-adsh_conf lhost_end
```

- 通常は論理ホスト HOST01 で運用し，一時的に物理ホストで運用する場合

  システムのデフォルトとしてシステム環境ファイルを定義します。ここでは次の内容を定義します。

  - KNAX6110-I，KNAX6111-I メッセージの出力を抑止する。
  - SPOOL_DIR，LOG_DIR，TRACE_DIR，TEMP_FILE_DIR，HOSTNAME_JP1IM_MANAGER パラメーターを論理ホストと物理ホストで別々の定義にして実行環境を分ける。
  - 物理ホストの SPOOL_DIR，LOG_DIR，TRACE_DIR，TEMP_FILE_DIR パラメーターはデフォルトのディレクトリを使用する。

- 環境変数 ABC の値を論理ホストと物理ホストで分ける。

この場合のシステム環境ファイルの定義例は次のとおりです。

```
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6110-I
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6111-I
#-adsh_conf  lhost_start HOST01
#-adsh_conf  SPOOL_DIR  /jp1as/host01/spool
#-adsh_conf  LOG_DIR    /jp1as/host01/log
#-adsh_conf  TRACE_DIR  /jp1as/host01/trace
#-adsh_conf  TEMP_FILE_DIR /jp1as/host01/temp
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM01
#-adsh_conf  lhost_end
#-adsh_conf  phost_start
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM01
#-adsh_conf  phost_end
#-adsh_conf  lhost_start HOST01
export  ABC=/jp1as/host01/abc
#-adsh_conf  lhost_end
#-adsh_conf  phost_start
export  ABC=/jp1as/abc
#-adsh_conf  phost_end
```

ジョブごとの設定変更をジョブ環境ファイルで定義します。特定のジョブでジョブ定義スクリプト中の引数を論理ホストと物理ホストで異なる値に変換することや，スクリプト拡張コマンド失敗時の終了コードを定義することができます。

ジョブ環境ファイルの定義例は次のとおりです。

```
#-adsh_conf ADSHCMD_RC_ERROR  8
#-adsh_conf lhost_start HOST01
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp01
#-adsh_conf lhost_end
#-adsh_conf phost_start
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp00
#-adsh_conf phost_end
```

(b) 条件パラメーターの定義例

条件パラメーターの定義内容と，各ホストに適用されるパラメーターについて定義例で説明します。

例えば，条件パラメーターを次のように定義した場合について説明します。

```
(パラメーター群A)
#-adsh_conf lhost_start HOST01
(パラメーター群B)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(パラメーター群C)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(パラメーター群D)
#-adsh_conf phost_end
#-adsh_conf lhost_start HOST01
(パラメーター群E)
#-adsh_conf lhost_end
```

```
#-adsh_conf lhost_start HOST02
(パラメーター群F)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(パラメーター群G)
#-adsh_conf phost_end
(パラメーター群H)
```

この定義の場合，各ホストで実行されるパラメーター群は次のとおりです。

論理ホスト HOST01 で実行されるパラメーター群

```
(パラメーター群A)
(パラメーター群B)
(パラメーター群E)
(パラメーター群H)
```

論理ホスト HOST02 で実行されるパラメーター群

```
(パラメーター群A)
(パラメーター群C)
(パラメーター群F)
(パラメーター群H)
```

物理ホストで実行されるパラメーター群

```
(パラメーター群A)
(パラメーター群D)
(パラメーター群G)
(パラメーター群H)
```

7.3　環境設定パラメーター

JP1/Advanced Shell の環境ファイルに設定するパラメーターについて説明します。

7.3.1　ADSHCMD_RC_ERROR パラメーター（スクリプト拡張コマンド失敗時の終了コードを定義する）

形式

```
#-adsh_conf ADSHCMD_RC_ERROR 終了コード
```

機能

スクリプト拡張コマンドが失敗したときの終了コードを定義します。

オペランド

終了コード　～<符号なし整数>((0～255))《1》

スクリプト拡張コマンドが失敗したときの終了コードを指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.2　ADSHCMD_RC_SUCCESS パラメーター（スクリプト拡張コマンド成功時の終了コードを定義する）

形式

```
#-adsh_conf ADSHCMD_RC_SUCCESS 終了コード
```

機能

スクリプト拡張コマンドが成功したときの終了コードを定義します。

オペランド

終了コード　～<符号なし整数>((0～255))《0》

スクリプト拡張コマンドが成功したときの終了コードを指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.3　ASC_FILE パラメーター（蓄積ファイル名の生成規則を定義する）

形式

```
#-adsh_conf ASC_FILE ファイル名規則
```

機能

カバレージ採取の一括有効化機能で使用する蓄積ファイル名の生成規則を定義します。

オペランド

ファイル名規則

Windows の場合　～<任意文字列>((1～247 バイト))

UNIX の場合　～<任意文字列>((1～1,023 バイト))

置換位置を指定したパス名を指定します。

置換位置は*で指定し，*はジョブ定義スクリプト名で置換します。この場合，ジョブ定義スクリプトのファイル拡張子を除きます。置換位置の先頭から最初に検索した*を置換の対象とします。それ以降に指定しても置換の対象とはなりません。

置換位置の指定がない場合は，置換しないでそのままファイル名になります。

置換の有無に関係なく，結果の値がパス名として正しくない場合はエラーとなります。また，このパラメーターを省略した場合は，adshexec コマンドで-o オプションを指定しなかったときと同様の規則でファイル名が決定します。

変換後のファイル名が，adshexec コマンドで規定された asc ファイルのパス名の長さの上限を超えないように，ファイル名規則を設定してください。上限を超えると，adshexec コマンド実行時にエラーとなります。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- BATCH_CVR パラメーターのオペランドに YES の指定がない場合は無効となります。
- ファイル名に.（ドット）で始まる名称を使用しないでください。
- ファイル名に予約デバイス名（CON や AUX，NUL など）は使用しないでください。【Windows 限定】
- ファイル名に NTFS のストリームは使用しないでください。【Windows 限定】

使用例

- カバレージ採取の一括有効化機能を有効にし，蓄積ファイル名の生成規則を定義します。

```
#-adsh_conf BATCH_CVR YES
#-adsh_conf ASC_FILE ./cvrg/ver001-*
```

この場合，「adshexec sample.ash」の実行は，「adshexec -t -o ./cvrg/ver001-sample sample.ash」の実行と同じとなります。

7.3.4 BATCH_CVR パラメーター（カバレージ採取の一括有効化機能を有効にする）

形式

```
#-adsh_conf BATCH_CVR YES
```

機能

カバレージ採取の一括有効化機能を有効にします。

オペランド

YES

カバレージ採取の一括有効化機能を有効にします。この機能を有効にした場合，adshexec コマンドで-t オプションを指定した場合と同じ効果があります。

adshexec コマンドの-f オプションを指定しない場合と同様に，ジョブ定義スクリプトファイルとバックアップ情報に差分があったときは，カバレージを採取しません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- 次の場合に実行するとカバレージ採取の一括有効化機能は無効となります。
  - adshexec コマンドに-v オプションまたは-c オプションの指定がある場合
  - JP1/Advanced Shell - Developer で実行する場合
- ASC_FILE パラメーターの指定を有効にする場合，このパラメーターの指定が必要です。
- adshexec コマンドに-t の指定がある場合は，エラーメッセージを出力して，コマンドは RC=1 で終了します。
- すでにカバレージ情報ファイルがある状態で，ジョブ定義スクリプトファイルを変更すると，カバレージ情報を採取できません。この場合，次のどちらかを実施してください。

- ジョブ定義スクリプトを変更した asc ファイルを削除します。
- 環境ファイルで，asc ファイルの作成ディレクトリを変更します。

7.3.5 CHILDJOB_EXT パラメーター（子孫ジョブとして実行するジョブ定義スクリプトファイルの拡張子を定義する）

形式

```
#-adsh_conf CHILDJOB_EXT 拡張子
```

機能

子孫ジョブとして実行するジョブ定義スクリプトファイルの拡張子を定義します。

ジョブ定義スクリプト中に，このパラメーターで定義した拡張子を持つほかのジョブ定義スクリプトファイルをコマンド名として指定した場合，KNAX6832-I メッセージをジョブ実行ログに出力し，JP1/Advanced Shell のジョブ定義スクリプトと解釈して子孫ジョブとして実行します。なお，KNAX6832-I メッセージは，JOBLOG_SUPPRESS_MSG パラメーターでジョブ実行ログへの出力を抑止できます。

オペランド

拡張子　～<パス名>((1～245 バイト))

子孫ジョブとして実行するジョブ定義スクリプトファイルの拡張子を指定します。"（ダブルクォーテーション）で囲んだ値に空文字だけを指定することはできません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。ただし，システム環境ファイルとジョブ環境ファイルの定義の合計が 255 個を超えた場合，エラーになります。
- オペランドに指定する拡張子には.（ドット）を含まない値を指定します。
- オペランドに指定する拡張子に/（スラッシュ）だけを指定した場合，パラメーター解析時にエラー終了します。
- オペランドに指定する拡張子に，Windows が定める実行可能拡張子である exe，bat，cmd，または com を指定しないでください。指定した場合はほかの拡張子と同様に，該当するファイルを子孫ジョブとして実行します。【Windows 限定】
- ジョブ実行ログに出力されるコマンド名は，子孫ジョブとして実行したスクリプトファイルのパスとなります。
- このパラメーターによって実行可能なファイルに対して，ファイル属性を評価する演算子"-x"を使用して評価すると，判定は真となります。【Windows 限定】

- 同じ拡張子を重複して指定してもエラーにはなりません。
- シェル標準コマンドの exec コマンド，command コマンド，eval コマンド，およびスクリプト予約語コマンドの time コマンドの引数にファイルを指定した場合も，このパラメーターによる実行の対象となります。
- Windows 版の場合，英大文字と英小文字を区別しません。UNIX 版の場合，英大文字と英小文字を区別します。
- 拡張子がないファイルを子孫ジョブとして実行する場合は，CHILDJOB_PGM パラメーターまたは CHILDJOB_SHEBANG パラメーターを使用してください。
- このパラメーターは，変数置換やエイリアスの解決後のコマンド名に対して適用されます。

使用例

拡張子が ash，sh のファイルをコマンド名として指定した場合に，子孫ジョブとして実行したいときの例を次に示します。

```
#-adsh_conf CHILDJOB_EXT ash
#-adsh_conf CHILDJOB_EXT sh
```

7.3.6 CHILDJOB_PGM パラメーター（子孫ジョブとして実行する指定を定義する）

形式

```
#-adsh_conf CHILDJOB_PGM プログラムパス名 ［実行プログラムの引数］
```

機能

ジョブ定義スクリプト中に「**プログラムパス名** ［**実行プログラムの引数**］ファイル」という指定が現れた場合，KNAX6830-I メッセージをジョブ実行ログに出力し，ファイルをジョブ定義スクリプトと解釈し，子孫ジョブとして実行します。なお，KNAX6830-I メッセージは，JOBLOG_SUPPRESS_MSG パラメーターでジョブ実行ログへの出力を抑止できます。

例を次に示します。

パラメーターの指定例

```
#-adsh_conf CHILDJOB_PGM sh -x
```

ジョブ定義スクリプトの内容

```
sh -x $HOME/script/test.ash
```

この例の場合，「$HOME/script/test.ash」をジョブ定義スクリプトと解釈し，子孫ジョブとして実行します。

指定できるオペランドの数は，プログラムパス名と実行プログラムの引数の合計で，64 個までです。65 個以上指定した場合，パラメーター解析時にエラー終了します。

Windows 版では，環境ファイルに CHILDJOB_PGM パラメーターを定義しなくても，プログラムパスに「adshscripttool -exec」が指定された CHILDJOB_PGM パラメーターが定義されています。adshscripttool コマンドについては，「adshscripttool コマンド（ジョブ定義スクリプトの作成を支援する）【Windows 限定】」を参照してください。

オペランド

プログラムパス名　～<パス名>((1～1,023 バイト))

子孫ジョブとして実行するための起動プログラムに読み替える前のプログラムパスを定義します。"（ダブルクォーテーション）で囲んだ値に空文字だけを指定することはできません。また，「¥」はエスケープ文字として扱いません。

実行プログラムの引数　～<任意文字列>((1～1,023 バイト))

子孫ジョブとして実行するための起動プログラムに読み替える前のプログラムパスの引数を定義します。スペースおよびタブ文字で区切ることで，複数指定できます。また，「¥」はエスケープ文字として扱いません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。ただし，システム環境ファイルとジョブ環境ファイルの定義の合計が 255 個を超えた場合，エラーになります。
- このパラメーターと PATH_CONV パラメーターで，同じオペランドを定義した場合，PATH_CONV パラメーターによる変換が先に実施されます。例を次に示します。

  環境ファイルの内容

  ```
  #-adsh_conf PATH_CONV_ENABLE / :                ←1.
  #-adsh_conf PATH_CONV /usr/bin C:¥¥usr¥¥bin     ←2.
  #-adsh_conf CHILDJOB_PGM /usr/bin/ksh           ←3.
  ```

  ジョブ定義スクリプトの内容

  ```
  "/usr/bin/ksh" C:¥¥script¥¥test.ash
  ```

  この例では，1.および 2.が先に実施され，「/usr/bin/ksh」が「C:¥¥usr¥¥bin¥¥ksh」に変換されます。その後，3.によって「/usr/bin/ksh」が子孫ジョブとして実行するように読み替えるプログラムパスと解釈されます。しかし，パス変換済みのジョブ定義スクリプトの中には一致する文字列が存在しないため，「C:¥¥script¥¥test.ash」は子孫ジョブとして実行されません。
- このパラメーターと COMMAND_CONV_ARG パラメーターで，同じオペランドを定義した場合，COMMAND_CONV_ARG パラメーターによる変換が先に実施されます。

- このパラメーターは，変数置換やエイリアスの解決後の文字列に対して適用されます。変数置換の例を次に示します。

  環境ファイルの内容

  ```
  #-adsh_conf CHILDJOB_PGM /usr/bin/ksh
  ```

  ジョブ定義スクリプトの内容（ここでは変数 SHELL の値を"/usr/bin/ksh"と仮定）

  ```
  $SHELL /home/usr/test.ash
  ```

  この例では，「$SHELL」が「/usr/bin/ksh」として解決されたあとに，パラメーターの定義に従って読み替えられるため，「/home/usr/test.ash」は子孫ジョブとして実行されます。

- このパラメーターを使用して子孫ジョブを実行した場合，ジョブ実行ログに出力されるコマンド名は，JP1/Advanced Shell のコマンド（adshexec コマンドまたは adshexecsub コマンド）となります。

  ただし，コマンドを次の書式で実行した場合，ジョブ実行ログに出力されるコマンド名は，このパラメーターが適用される前のプログラムパス名となります。

  - パイプ（|）による別プロセスの実行
  - コマンド置換（$(), ``）による別プロセスの実行
  - |&によるバックグラウンドプロセスの実行
  - 単一コマンドのグループ化によるサブシェルの実行
  - &によるバックグラウンドの実行

- シェル標準コマンドの exec コマンド，command コマンド，eval コマンド，およびスクリプト予約語コマンドの time コマンドの引数に指定したプログラムパスも，このパラメーターが適用される対象となります。
- このパラメーターは，ジョブ定義スクリプト中のシェル標準コマンド，シェル拡張コマンド，関数，外部コマンドに対して適用されます。
- パス名，および実行プログラムの引数は，環境ファイルの 1 行の長さの上限の範囲で指定してください。

使用例

ジョブ定義スクリプトに指定された「$HOME/script/test.ash」を子孫ジョブとして実行させる場合を例に，ジョブ定義スクリプトと環境ファイルの指定内容を次に示します。CHILDJOB_PGM パラメーターで指定した内容を下線で示します。

例 1

環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /bin/sh
```

ジョブ定義スクリプトの内容

```
/bin/sh $HOME/script/test.ash
```

例 2

環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /opt/jp1as/bin/adshexec
```

ジョブ定義スクリプトの内容

```
/opt/jp1as/bin/adshexec $HOME/script/test.ash
```

例 3

環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM sh -x
```

ジョブ定義スクリプトの内容

```
sh -x $HOME/script/test.ash
```

例 4

環境ファイルの内容

```
#-adsh_conf CHILDJOB_PGM /usr/bin/env ksh
```

ジョブ定義スクリプトの内容

```
/usr/bin/env ksh $HOME/script/test.ash
```

7.3.7 CHILDJOB_SHEBANG パラメーター（子孫ジョブとして実行するジョブ定義スクリプトファイルの実行プログラムパスを定義する）

形式

```
#-adsh_conf CHILDJOB_SHEBANG パス名
```

機能

子孫ジョブとして実行するジョブ定義スクリプトファイルに指定された，「#!」に続く実行プログラムパスを定義します。「#!」の直後から行末までをパス名の比較対象とします。

ジョブ定義スクリプト中にほかのジョブ定義スクリプトファイルをコマンド名として指定した場合，このパラメーターで指定したパス名がジョブ定義スクリプトのファイルの 1 行目に「#!＋パス名」と記述されていれば，KNAX6831-I メッセージをジョブ実行ログに出力し，ジョブ定義スクリプトファイルを子孫ジョブとして実行します。KNAX6831-I メッセージは，JOBLOG_SUPPRESS_MSG パラメーターでジョブ実行ログへの出力を抑止できます。

なお，CHILDJOB_SHEBANG パラメーターには，次の 2 つがデフォルトで定義されています。

デフォルト定義	子孫ジョブ起動時の出力モード
/opt/jp1as/bin/adshexec	OUTPUT_MODE_CHILD パラメーターの指定に従って動作します。
/opt/jp1as/bin/adshexec -mMINIMUM	最小出力モードで動作します。

CHILDJOB_SHEBANG パラメーターのデフォルト定義については，「(b)　パラメーターのデフォルト定義で子孫ジョブを実行する方法」を参照してください。

オペランド

パス名　～<任意文字列>((1～1,023 バイト))

子孫ジョブとして実行するジョブ定義スクリプトファイルに指定した「#!」で開始する実行プログラムパスを定義します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。ただし，システム環境ファイルとジョブ環境ファイルの定義の合計が 255 個を超えた場合，エラーになります。
- オペランドに指定するパス名には，先頭の「#!」を含まない値を指定します。
- オペランドに指定するパス名に「/」（スラッシュ）だけ指定した場合，パラメーター解析時にエラー終了します。
- ジョブ実行ログに出力されるコマンド名は，直接指定したスクリプトファイルのパスとなります。
- このパラメーターによって実行可能なファイルに対して，ファイル属性を評価する演算子"-x"を使用して評価すると，判定は真となります。【Windows 限定】
- 同じ実行プログラムパスを重複して指定してもエラーにはなりません。
- シェル標準コマンドの exec コマンド，command コマンド，eval コマンド，およびスクリプト予約語コマンドの time コマンドの引数にファイルを指定した場合も，このパラメーターによる実行の対象となります。

使用例

先頭に特定の記述があるジョブ定義スクリプトファイルを子孫ジョブとして実行する例を次に示します。

- 先頭に「#!/bin/sh」または「#!/bin/ksh」が記述されたファイルを子孫ジョブとして実行する例

```
#-adsh_conf  CHILDJOB_SHEBANG  /bin/sh
#-adsh_conf  CHILDJOB_SHEBANG  /bin/ksh
```

- 先頭に「#!/bin/ksh -x」が記述されたファイルを子孫ジョブとして実行する例

```
#-adsh_conf  CHILDJOB_SHEBANG  ”/bin/ksh -x”
```

- 先頭に「#!/usr/bin/env ksh」が記述されたファイルを子孫ジョブとして実行する例

```
#-adsh_conf  CHILDJOB_SHEBANG  "/usr/bin/env ksh"
```

7.3.8 CMDRC_CMDGRP_CHECK パラメーター（関数の終了コードに従ってジョブおよびジョブステップのエラー判定をする）

形式

```
#-adsh_conf CMDRC_CMDGRP_CHECK  {FUNCTION|NONE}
```

機能

関数の終了コードに従ってジョブおよびジョブステップのエラー判定をするかどうかを定義します。

オペランド

FUNCTION

関数の終了コードに従ってジョブおよびジョブステップのエラー判定をし，関数内のコマンドの終了コードに従ってジョブおよびジョブステップのエラー判定をしないようにします。また，CMDRC_THRESHOLD_DEFINE パラメータ，#-adsh_rc_ignore コマンド，adshcmdrc コマンド，#-adsh_step_start コマンドの-successRC 属性によって，関数の終了コードしきい値を変更できるようになります。

NONE

関数内のコマンドの終了コードに従って，関数が定義されているジョブおよびジョブステップのエラー判定をするようにします。関数の終了コードに従ったジョブおよびジョブステップのエラー判定はしません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメータが指定された場合，ジョブ環境ファイルの定義が有効になります。
- オペランドに FUNCTION を指定した場合は，関数内のコマンドのメッセージは出力されません。
- 関数実行中に，シグナルまたは強制終了要求を受けて実行された trap コマンドのアクションは，関数内で実行されたと見なされます。このため，関数内で実行された trap コマンドのアクションは，オペランドの値に従って，ジョブおよびジョブステップのエラー判定の変更対象になります。
- 関数実行中にバックグラウンド実行したコマンドは，関数内から実行したとしても，関数内のコマンドとは見なしません。

7.3.9 CMDRC_THRESHOLD_DEFINE パラメーター（コマンドの終了コードのしきい値を定義する）

形式

```
#-adsh_conf CMDRC_THRESHOLD_DEFINE コマンド名 しきい値
```

機能

ジョブ定義スクリプトから実行されるコマンドの終了コードが 0 でなくても正常終了と見なしたい場合，対象となるコマンド名と，しきい値となる値を定義します。これによって，コマンドの終了コードがしきい値以下の場合が正常終了となります。

ただし，コマンドがシグナルを受信して終了した場合は，指定に関係なくコマンドがエラー終了します。

CMDRC_THRESHOLD_USE_PRESET パラメーターの ENABLE の指定によってしきい値が定義された UNIX 互換コマンドに対しても，CMDRC_THRESHOLD_DEFINE パラメーターでしきい値を変更できます。

オペランド

コマンド名　～<コマンド名>((1～255 バイト))

終了コードのしきい値を定義するコマンドの名称を指定します。Windows の場合は拡張子付きの指定もできます。コマンドパスは指定できません。

指定できるコマンドの種類を次に示します。これ以外のコマンドも，別プロセスで実行（パイプ，コマンド置換，|&，&を使用）した場合は対象になります。

- 外部コマンド
- UNIX 互換コマンド
- シェル運用コマンド
- コマンドとして実行したスクリプト（#!によって実行）
- シェルスクリプト
- 子孫ジョブ

Windows でコマンド名の拡張子の指定を省略すると，指定した名称と同じ名称のコマンドやバッチファイルが，拡張子に関係なくしきい値の管理対象となります。

Windows でスペースを含むコマンド名を指定する場合は，"（ダブルクォーテーション）で囲んでください。

しきい値　～<整数>((-1～255))

終了コードで正常終了と見なすしきい値を定義します。ここで指定したしきい値より終了コードが大きい場合，エラー終了と見なします。

-1 を指定した場合，実行結果は常にエラー終了します。

255 を指定した場合，実行結果は常に正常終了します。

注意事項

- このパラメーターは，変数置換やエイリアスの解決後のコマンド名に対して適用されます。
- このパラメーターによって設定されたコマンドのしきい値，および実行したコマンドの終了コードとの判定によって決定したコマンドの動作は，ジョブ定義スクリプトの#-adsh_rc_ignore コマンドや，#-adsh_step_start コマンドの successRC 属性の設定値が優先します。
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義された場合，両方の定義が有効になります。ただし，同じコマンドに対して異なるしきい値が定義されていた場合は，ジョブ環境ファイルに最後に定義された方が有効になります。
- パラメーターの指定数に上限はありませんが，多数指定するとジョブ定義スクリプトの実行性能に影響が及ぶため，必要のないコマンドの定義はしないでください。
- 子孫ジョブとして実行するジョブ定義スクリプトの終了コードのしきい値を定義したい場合，子孫ジョブの定義には CHILDJOB_EXT パラメーターまたは CHILDJOB_SHEBANG パラメーターを使用してください。

  CHILDJOB_PGM パラメーターで子孫ジョブを定義した場合，終了コードのしきい値の対象として見なされません。
- Windows の場合，コマンド名には拡張子を含めた指定および拡張子を省略した指定ができます。

  ただし，同じコマンド名に対して拡張子を含めた指定と拡張子を省略した指定をしても，同じコマンド名の指定とは見なされません。
  - このパラメーターで同じコマンド名に対して，拡張子を含めた指定と拡張子を省略した指定をした場合，先に指定された方が有効です。
  - CMDRC_THRESHOLD_USE_PRESET パラメーターで ENABLE を指定して仮定されるコマンド名は拡張子を含めた名称となるため，このパラメーターをしきい値の変更の目的で指定する場合，拡張子を含めた指定としなければなりません。拡張子を省略して指定した場合は，別コマンドとして登録されます。
- CMDRC_THRESHOLD_USE_PRESET パラメーターと CMDRC_THRESHOLD_DEFINE パラメーターは，指定の順番に関係なく CMDRC_THRESHOLD_USE_PRESET パラメーターが先に処理されます。
- CMDRC_CMDGRP_CHECK パラメータに FUNCTION を指定した場合，オペランドのコマンド名に関数名を指定することで，関数に対して正常終了とみなす終了コードしきい値を定義できます。

  CMDRC_CMDGRP_CHECK パラメータを指定しない場合，または CMDRC_CMDGRP_CHECK パラメータに NONE を指定した場合，オペランドのコマンド名に関数名を指定しても，同名のコマンド名が指定されたとして処理されます。

使用例

- 子孫ジョブ CHILD_EXEC1.sh に対し，終了コードのしきい値を 10（終了コードが 10 以下で正常終了）と定義します。

```
#-adsh_conf  CHILDJOB_EXT sh
#-adsh_conf  CMDRC_THRESHOLD_DEFINE CHILD_EXEC1.sh 10
```

- UNIX 互換コマンドの終了コードのしきい値として 1 を一括定義します。また，それ以外の UNIX 互換コマンドである acmd，bcmd，ccmd コマンドのしきい値として 20 を定義します。

```
#-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
#-adsh_conf CMDRC_THRESHOLD_DEFINE acmd 20
#-adsh_conf CMDRC_THRESHOLD_DEFINE bcmd 20
#-adsh_conf CMDRC_THRESHOLD_DEFINE ccmd 20
```

- 【Windows 限定】コマンド名「acmd.exe」の終了コードのしきい値として 10 を定義し，総称名「acmd」のしきい値として 20 を定義します。

```
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 10
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd 20
```

  この定義順の場合，コマンド「acmd.exe」のしきい値として 10 が適用され，拡張子が「.exe」以外のコマンド（acmd.bat など）のしきい値として 20 が適用されます。

- 【Windows 限定】コマンドの総称名「acmd」のしきい値として 20 を定義し，コマンド名「acmd.exe」の終了コードのしきい値として 10 を定義します。

  名称「acmd」のコマンド（acmd.exe や acmd.bat など）の終了コードのしきい値として 20 が適用されます。

```
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd 20
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 10
```

  この定義順の場合，最初に記述されているコマンドの総称名「acmd」に対する定義がコマンド「acmd.exe」に対しても優先的に適用されるため，コマンド「acmd.exe」のしきい値も 20 になります。

- 同じパラメーターにコマンド名「acmd.exe」を 2 つ指定します。しきい値はそれぞれ 30 と 20 を定義します。

```
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 30
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 20
```

  この場合，最後に指定したパラメーターのしきい値の 20 が適用されます。

- コマンド名「acmd.exe」をシステム環境ファイルにはしきい値 10 で指定し，ジョブ環境ファイルにはしきい値 20 でそれぞれ指定します。

  ＜システム環境ファイルの指定＞

```
#-adsh_conf CMDRC_THRESHOLD_DEFINE acmd.exe 10
```

  ＜ジョブ環境ファイルの指定＞

```
#-adsh_conf CMDRC_THRESHOLD_DEFINE acmd.exe 20
```

  この場合，ジョブ環境ファイルに指定したしきい値の 20 が適用されます。

- 【Windows 限定】UNIX 互換コマンドの終了コードのしきい値として 1 を一括定義したのち，cmp.exe コマンドだけしきい値を 2 に変更します。

```
#-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
#-adsh_conf CMDRC_THRESHOLD_DEFINE cmp.exe 2
```

- 【Windows 限定】次のように，CMDRC_THRESHOLD_USE_PRESET パラメーターでしきい値として 1 が仮定されたコマンド「cmp.exe」に対して，CMDRC_THRESHOLD_DEFINE パラメーターで「cmp 2」と指定しても，しきい値を 1 から 2 に変更することはできません。

  ```
  #-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
  #-adsh_conf CMDRC_THRESHOLD_DEFINE cmp 2
  ```

  この場合，「cmp.exe」と「cmp」は別々のコマンドとして管理され，それぞれのしきい値が有効となります。

- CMDRC_THRESHOLD_USE_PRESET パラメーターと CMDRC_THRESHOLD_DEFINE パラメーターを次のように異なる環境ファイルで指定した場合，コマンド「cmp.exe」のしきい値は，CMDRC_THRESHOLD_DEFINE パラメーターに指定したしきい値である 2 が適用されます。

  ＜システム環境ファイルの指定＞

  ```
  #-adsh_conf CMDRC_THRESHOLD_DEFINE cmp.exe 2
  ```

  ＜ジョブ環境ファイルの指定＞

  ```
  #-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
  ```

7.3.10 CMDRC_THRESHOLD_USE_PRESET パラメーター（UNIX 互換コマンドの終了コードのしきい値を定義する）

形式

```
#-adsh_conf CMDRC_THRESHOLD_USE_PRESET  {ENABLE|DISABLE}
```

機能

次に示すすべての UNIX 互換コマンドに対し，正常終了と見なす終了コードのしきい値を一括で定義します。定義できるしきい値は 0 または 1 です。

- cmp コマンド
- diff コマンド
- egrep コマンド
- expr コマンド
- grep コマンド
- sort コマンド

コマンドごとに異なるしきい値を設定したい場合は，CMDRC_THRESHOLD_DEFINE パラメーターでしきい値を定義してください。

なお，このパラメーターで定義されたコマンド名は，Windows では拡張子付きで登録されます。例えば，cmp コマンドは OS によって次のように登録されます。

- Windows 版の場合：cmp.exe
- UNIX 版の場合：cmp

そのため，CMDRC_THRESHOLD_DEFINE パラメーターでしきい値を変更する場合は，拡張子付きでコマンド名を定義してください。

オペランド

ENABLE

終了コードのしきい値として 1 を定義します。これによって，終了コードが 1 のコマンドも正常終了と見なされます。

また，対象となる UNIX 互換コマンドと同じ名称を持つコマンドも，しきい値として 1 が定義されます。

DISABLE

終了コードのしきい値として 0 を定義します。

注意事項

- このパラメーターは，変数置換やエイリアスの解決後のコマンド名に対して適用されます。
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- ENABLE を指定して設定されたコマンドのしきい値，および実行したコマンドの終了コードとの判定によって決定したコマンドの動作は，ジョブ定義スクリプトの#-adsh_rc_ignore コマンドや，#-adsh_step_start コマンドの successRC 属性の設定値が優先します。
- CMDRC_THRESHOLD_USE_PRESET パラメーターと CMDRC_THRESHOLD_DEFINE パラメーターは，指定の順番に関係なく CMDRC_THRESHOLD_USE_PRESET パラメーターが先に処理されます。

使用例

- 対象となる UNIX 互換コマンドの終了コードが 1 でも，正常終了として扱いたいときの例を次に示します。

```
#-adsh_conf  CMDRC_THRESHOLD_USE_PRESET ENABLE
```

7.3.11 CMDSUB_PROCESS パラメーター（コマンド置換の実行プロセスを定義する）【 Windows 限定】

形式

```
#-adsh_conf CMDSUB_PROCESS {CURRENT | OTHER}
```

機能

コマンド置換の実行プロセスを定義します。

コマンド置換で変数の内容を更新しますが，コマンド置換実行前の内容をコマンド置換終了後に再利用したい場合は，OTHER を指定してください。

OTHER を指定した場合の例を次に示します（CBL_SYSUT は CBLUAPx の共通インターフェース用変数と仮定します）。

- ジョブ定義スクリプトの内容

```
funcA(){
  CBL_SYSUT=/file2
  CBLUAP2
}
CBL_SYSUT=/file1
VAL1=$(CBLUAP1)
VAL2=$(funcA)
VAL3=$(CBLUAP3)
```

この場合，CBLUAP1 と CBLUAP3 の実行時のシェル変数 CBL_SYSUT には"/file1"，CBLUAP2 の実行時のシェル変数 CBL_SYSUT には"/file2"が格納されています。

オペランド

CURRENT

次に示すコマンドをコマンド置換に指定した場合に，コマンド置換をカレントプロセスで実行します。

- シェル標準コマンド
- 代入式
- スクリプト制御文
- シェル拡張コマンド
- スクリプト予約語コマンド
- 関数

上記以外の場合，別プロセスで動作します。また，データの受け渡しのために一時ファイルを使用します。

OTHER

コマンド置換を別プロセスで実行します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。
- OTHER を指定した場合，CURRENT に比べて実行時間が大きくなる場合があります。そのため，CURRENT から OTHER に切り替える際は，ジョブの実行時間を検証してください。なお，スクリプト開発部品やスクリプト形式の UNIX 互換コマンドには，スクリプト内でコマンド置換を使用しているものがあります。スクリプト開発部品やスクリプト形式の UNIX 互換コマンドを実行する場合は，実行時間の長大化を防ぐために CURRENT を指定しておくことを推奨します。
- 次のすべての条件が重なった命令を記述すると，コマンド置換の実行結果を正しく得ることができない場合があります。
  - CMDSUB_PROCESS パラメーターに CURRENT を指定している。
  - コマンド置換処理の中でパイプを使用している。
  - パイプで繋がれた最後のコマンドをバックグラウンド実行している。
    例: `cmd1 | { cmd2; cmd3 & }`

7.3.12 COMMAND_CONV_ARG パラメーター（コマンド実行時にジョブ定義スクリプト中の引数を変換する規則を定義する）

形式

```
#-adsh_conf COMMAND_CONV_ARG コマンド引数1 コマンド引数2
```

機能

ジョブ定義スクリプト中のシェル標準コマンド，シェル拡張コマンド，関数，スクリプト拡張コマンド，スクリプト予約語コマンド，外部コマンド，およびユーザープログラムの引数を変換する規則を定義します。

ジョブ定義スクリプトの実行時にコマンドの引数がコマンド引数 1 と完全一致する場合，コマンド引数 2 に変換します。コマンド引数 1 およびコマンド引数 2 は必ず指定します。

同じコマンド引数に異なる規則を定義した場合は，先に定義した規則が有効になります。

.（ドット）コマンド，および#-adsh_script コマンドに指定した外部スクリプトも，実行時に定義に合致していた場合，引数が変換されます。

変換結果は，KNAX6804-I メッセージ，または KNAX6806-I メッセージでジョブ実行ログに出力します。

オペランド

コマンド引数 1　～<任意文字列>((1～247 バイト))

変換前のコマンド引数を指定します。スペースを含むコマンド引数を指定する場合は，"（ダブルクォーテーション）で囲む必要があります。ただし，"で囲んだ値にスペース，タブ文字または空文字だけを指定できません。なお，次の文字は使用できません。

＊　？　＜　＞　｜　｀（バッククォーテーション）　$

コマンド引数 1 を指定しなかった場合，またはコマンド引数 1 に不当な値を指定した場合は，パラメーター解析時にエラー終了します。

コマンド引数 2　～<任意文字列>((1～247 バイト))

変換後のコマンド引数を指定します。スペースを含むコマンド引数を指定する場合は，"（ダブルクォーテーション）で囲む必要があります。ただし，"で囲んだ値にスペース，タブ文字または空文字だけを指定できません。なお，次の文字は使用できません。

＊　？　＜　＞　｜　｀（バッククォーテーション）　$

コマンド引数 2 を指定しなかった場合，またはコマンド引数 2 に不当な値を指定した場合は，パラメーター解析時にエラー終了します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。ただし，システム環境ファイルとジョブ環境ファイルの定義の合計が 255 個を超えた場合，エラーになります。
- 次の例のように，変換後のコマンド引数と後続のパラメーターの変換前のコマンド引数が同じ場合，規則に合致する引数を持つ eval コマンドを実行すると，eval コマンド実行時とその引数のコマンドの実行時とで，2 回の変換が実行されます。

  （例）

  環境ファイルの内容

  ```
  #-adsh_conf COMMAND_CONV_ARG /tmp /var/tmp        ←1.
  #-adsh_conf COMMAND_CONV_ARG /var/tmp /jp1as/tmp  ←2.
  ```

  ジョブ定義スクリプトの内容

  ```
  eval cd /tmp
  ```

  この場合，環境ファイルのそれぞれの行で次のように変換されます。

  1. eval コマンドが実行され，「/tmp」が「/var/tmp」に変換される。
  2. cd コマンドが実行され，「/var/tmp」が「/jp1as/tmp」に変換される。

- このパラメーターの変換後の引数に，文字列区切りである IFS シェル変数と一致する文字が含まれた場合でも，変換後の引数は 1 つの文字列の引数として解釈されます。そのため，パラメーターの変換結果

の文字列にメタキャラクタが含まれていて，実行するコマンドが eval コマンド以外の場合，その文字はメタキャラクタではなく一般的な文字として扱われます。

環境ファイルの内容（IFS シェル変数はスペースとする）

```
#-adsh_conf COMMAND_CONV_ARG ”D:¥JP1AS” ”C:¥¥Documents and Settings” ←引数の変換規則1
#-adsh_conf COMMAND_CONV_ARG ”A=1” ”A=1 &”                            ←引数の変換規則2
```

ジョブ定義スクリプトの内容

```
cd ”D:¥JP1AS”       ←1.
readonly A=1        ←2.
eval cd ”D:¥JP1AS”  ←3.
eval readonly A=1   ←4.
```

1. 変換規則 1 に従って次のように変換され，実行される。

   ```
   cd ”C:¥¥Documents and Settings”
   ```

2. 変換規則 2 に従って変換するが，次のように文字列が区切られる。

   ```
   readonly    A=1 &
   ```

3. eval コマンドの実行時に，変換規則 1 に従って次のように変換される。

   ```
   eval cd ”C:¥¥Documents and Settings”
   ```

   しかし，cd コマンド実行時に次のように引数が区切られ解釈される。

   ```
   cd    C:¥Documents    and    Settings
   ```

4. eval コマンドの実行時に変換規則 2 に従って次のように変換される。

   ```
   eval readonly A=1 &
   ```

   しかし，readonly コマンドの実行時に次のように引数が区切られ解釈される。

   ```
   readonly    A=1    &
   ```

- このパラメーターによる変換は，変数置換およびファイル名置換が解決した内容で実施されます。このため，ワイルドカードを含んだ文字列を引数に指定した場合，ワイルドカードによる置き換えが解決した内容をコマンドの引数として認識します。
- このパラメーターを test コマンドに適用した場合，形式の違いによってコマンド実行時の引数の文字列の解釈が異なります。[[ ]]の形式では，配列の要素に変数の置き換えを指定した場合，このパラメーターによる変換の対象外となります。このため，test コマンドを使用して，このパラメーターによるコマンドの引数を変換する場合は，test または[ ]による形式の使用を推奨します。

  (例)

  環境ファイルの内容

  ```
  #-adsh_conf PATH_CONV_ENABLE ¥¥ :
  #-adsh_conf COMMAND_CONV_ARG ”/tmp2” ”/tmp”   ←引数の変換規則1
  #-adsh_conf COMMAND_CONV_ARG ”ARY[1]” ”/tmp”  ←引数の変換規則2
  ```

  ジョブ定義スクリプトの内容

  ```
  ARY[0]=”/var”          ←配列ARYに”/var”, ”/tmp2”, ”/home”を格納
  ARY[1]=”/tmp2”
  ARY[2]=”/home”
  ```

```
id1=1
[[ -d ${ARY[$id1]} ]] ←${ARY[$id1]}を”/tmp2”まで置換。
                        このため，引数の変換規則1で”/tmp”に変換
[[ -d   ARY[$id1] ]]  ←置換しないで”ARY[$id1]”が引数となる。
                        ”$”は指定できないため，変換対象外
[ -d ${ARY[$id1]} ]   ←${ARY[$id1]}を”/tmp2”まで置換。
                        置換した結果を引数の変換規則1で”/tmp”に変換
[ -d   ARY[$id1] ]    ←${ARY[$id1]}を”ARY[1]”まで置換。
                        このため，引数の変換規則1で”/tmp”に変換
```

- 次に示すコマンドを実行した場合，引数に指定されたコマンドが実際には動作しますが，このパラメーターは引数に指定されたコマンドに対しても比較・変換を実行します。
  - builtin コマンド
  - command コマンド
  - eval コマンド
  - exec コマンド
  - time 予約語コマンド

  次の例では，builtin コマンドの引数に指定された pwd が実際には動作しますが，環境ファイルの内容に従って readonly コマンドが実行されます。

  環境ファイルの内容

  ```
  #-adsh_conf COMMAND_CONV_ARG pwd readonly
  ```

  ジョブ定義スクリプトの内容

  ```
  builtin pwd
  ```

- 変換する規則を定義した順に検索し，最初に変換条件に合致したものだけが適用されます。
- このパラメーターと PATH_CONV パラメーターで同じパス名に対して変換を定義した場合，PATH_CONV パラメーターによる変換が先に実施されます。PATH_CONV パラメーターで変換されたパス名を COMMAND_CONV_ARG パラメーターでさらに変換する場合は，PATH_CONV パラメーターで変換されたパス名を指定してください。
- このパラメーターではコマンドを実行するたびに，すべての引数に対して走査します。そのため，大量に設定するとジョブ定義スクリプトの実行時間に影響を及ぼす場合があります。注意してください。
- コマンド引数 2 に，メタキャラクタではなく文字列「¥」そのものを含む文字列を指定する場合，「¥」を「¥¥」と記述してください。

使用例

- Windows 用に作成したジョブ定義スクリプトを UNIX で実行するため，"C:¥temp"を"/tmp"に変換します。

  ```
  #-adsh_conf PATH_CONV_ENABLE ¥¥ ;
  #-adsh_conf COMMAND_CONV_ARG ”C:¥temp” /tmp
  ```

- UNIX 用に作成したジョブ定義スクリプトを Windows で実行するため，"/tmp"を"C:¥temp"に変換します。

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf COMMAND_CONV_ARG /tmp ”C:¥¥temp”
```

7.3.13 COMPATIBLE_CMD_EXEC パラメーター（外部コマンドの起動方法を定義する）【Windows 限定】

形式

```
#-adsh_conf COMPATIBLE_CMD_EXEC V10
```

機能

このパラメーターは互換性維持を目的としています。次の条件を満たすユーザーは定義が必要かどうか検討してください。

- JP1/Advanced Shell を 11-00 より前から使用し，11-00 以降にバージョンアップした。

このパラメーターを定義することで外部コマンドの起動方法を選択できます。

JP1/Advanced Shell は 11-00 で外部コマンドの起動方法を変更しています。ただし，この変更によって，外部コマンドによっては JP1/Advanced Shell 11-00 より前のバージョンでは正常終了していたコマンドがエラー終了する場合があります。

11-00 より前のバージョンで作成したジョブ定義スクリプトの内容は変更しないで，外部コマンドを 11-00 より前のバージョンとの互換性を保持した方式で起動したい場合は，オペランドに V10 を指定し，このパラメーターを環境ファイルに指定することを推奨します。

これによって 11-00 より前のバージョンと同じ方式で外部コマンドを起動します。オペランドに V10 を指定した場合，外部コマンド実行時に次の処理をします。

COMPATIBLE_CMD_EXEC パラメーターに V10 を指定した場合の動作

- 引数内の"（ダブルクォーテーション）の前の「¥」を「¥¥」に変換する。
- 引数内の"（ダブルクォーテーション）の前の「¥」を付与する。
- 引数を"（ダブルクォーテーション）で囲む。

上記の処理をスキップしたい場合，command コマンドに-w オプションを指定して，外部コマンドを起動してください。

ただし，command コマンドに-w オプションを指定して外部コマンドを起動した場合は，11-00 より前のバージョンとの互換性を保持した方式で外部コマンドを起動します。

オペランド

V10

上記の処理をし，11-00 より前のバージョンとの互換性を保持した方法で外部コマンドを起動します。

注意事項

- オペランドに V10 を指定した環境でバッチファイルを実行する場合，制限事項があります。制限の内容については，「5.1.11　外部コマンドの指定」を参照してください。
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。

7.3.14　COMPATIBLE_CMDSUB パラメーター（コマンド置換の動作を定義する）【UNIX 限定】

形式

```
#-adsh_conf COMPATIBLE_CMDSUB V10
```

機能

このパラメーターは互換性維持を目的としています。次の条件を満たすユーザーは定義が必要かどうか検討してください。

- JP1/Advanced Shell を 11-00 より前から使用し，11-00 以降にバージョンアップした。

オペランド

V10

PIPE_CMD_LAST パラメーターに CURRENT を指定した場合，パイプの最終コマンドをカレントプロセスで実行します。また，データの受け渡しのために一時ファイルを使用します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。

- 次のすべての条件が重なった命令を記述すると，コマンド置換の実行結果を正しく得ることができない場合があります。
  - COMPATIBLE_CMDSUB パラメーターを指定している。
  - PIPE_CMD_LAST パラメーターに CURRENT を指定している。
  - コマンド置換処理の中でパイプを使用している。
  - パイプでつながれた最後のコマンドをバックグラウンド実行している（例：`cmd1 | { cmd2, cmd3 & }`）。

7.3.15 ESCAPE_SEQ_ECHO_DEFAULT パラメーター（エスケープ文字関連のオプション省略時の echo コマンドの動作を定義する）

形式

```
#-adsh_conf ESCAPE_SEQ_ECHO_DEFAULT  {YES|NO}
```

機能

エスケープ文字関連のオプション（-E または-e）が省略された場合のecho コマンドのエスケープ文字に対する動作を定義します。

このパラメーターにYES を指定すると，echo コマンドで-e オプションを指定しなくてもエスケープ文字が解釈されます。NO を指定すると，エスケープ文字の解釈はecho コマンドでの指定に従います。

echo コマンドに-E オプションまたは-e オプションが指定された場合，このパラメーターの指定は無視されます。

オペランド

YES

-E オプションと-e オプションの両方が省略された場合，echo コマンドはエスケープ文字を解釈します。

NO

-E オプションと-e オプションの両方が省略された場合，echo コマンドはエスケープ文字を解釈しません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。

- 1～2 桁の 16 進数で表された ASCII コードの文字をエスケープ文字として解釈したい場合，ESCAPE_SEQ_ECHO_HEX パラメーターにもYES を指定してください。

使用例

- echo コマンドで-e オプションを指定しなくてもエスケープ文字を解釈します。

```
#-adsh_conf ESCAPE_SEQ_ECHO_DEFAULT YES
```

7.3.16 ESCAPE_SEQ_ECHO_HEX パラメーター（16 進数表記の ASCII コード文字をエスケープ文字として解釈するかを定義する）

形式

```
#-adsh_conf ESCAPE_SEQ_ECHO_HEX  {YES|NO}
```

機能

echo コマンドで，16 進数表記の ASCII コード文字をエスケープ文字として解釈するかどうかを定義します。このパラメーターは次のどちらかの条件を満たした場合に有効になります。

- echo コマンドに-e オプションが指定されている。
- ESCAPE_SEQ_ECHO_DEFAULT パラメーターに YES が指定されていて，かつ echo コマンドに-e オプションまたは-E オプションが指定されていない。

オペランド

YES

1～2 桁の 16 進数で表された ASCII コードの文字をエスケープ文字として解釈します。

NO

1～2 桁の 16 進数で表された ASCII コードの文字をエスケープ文字として解釈しません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。
- ASCII コードの範囲外の値を echo コマンドの引数に指定した場合，出力される内容は端末に指定された文字コードに従います。そのため，印字可能文字でない場合は，正しく出力されないことがあります。

使用例

- echo コマンドで 1～2 桁の 16 進数で表された ASCII コードの文字をエスケープ文字として解釈したい場合の例を次に示します。
  - 環境ファイルの内容

```
#-adsh_conf ESCAPE_SEQ_ECHO_HEX YES
```

  - ジョブ定義スクリプトの内容

```
STR="¥x48¥x49¥x54¥x41¥x43¥x48¥x49"
echo -e $STR
```

  - 標準出力に出力される内容

```
HITACHI
```

- echo コマンドで 1～2 桁の 16 進数で表された ASCII コードの文字をエスケープ文字として解釈しない場合の例を次に示します。
  - 環境ファイルの内容

```
#-adsh_conf ESCAPE_SEQ_ECHO_DEFAULT YES
#-adsh_conf ESCAPE_SEQ_ECHO_HEX NO
```

  - ジョブ定義スクリプトの内容

```
STR="¥t¥x48¥x49¥x54¥x41¥x43¥x48¥x49"
echo $STR
```

  - 標準出力に出力される内容

```
<タブ文字>¥x48¥x49¥x54¥x41¥x43¥x48¥x49
```

7.3.17 EVENT_COLLECT パラメーター（ジョブ定義スクリプト稼働実績情報取得機能の有効／無効を指定する）

形式

```
#-adsh_conf EVENT_COLLECT  {YES|NO}
```

機能

ジョブ定義スクリプト稼働実績情報取得機能を有効とするか，無効とするかを指定します。ジョブ定義スクリプト稼働実績情報取得機能を無効とした場合，スプールディレクトリにイベントファイルは作成されません。

JP1/Advanced Shell - Developer では，このパラメーターの指定は無視され，イベントファイルは作成されません。

CUI のデバッガを使用している場合も，このパラメーターの指定は無視され，イベントファイルは作成されません。

オペランド

YES

ジョブ定義スクリプト稼働実績情報取得機能を有効とします。

ジョブ定義スクリプト稼働実績情報を収集し，イベントファイルに出力します。

NO

ジョブ定義スクリプト稼働実績情報取得機能を無効とします。

イベントファイルを作成しません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。

使用例

- ジョブ定義スクリプト稼働実績情報取得機能を無効とします。

```
#-adsh_conf EVENT_COLLECT NO
```

7.3.18　export パラメーター（環境変数を定義する）

形式

```
export 環境変数名=環境変数値
```

機能

ジョブ定義スクリプトの実行時に有効としたい環境変数を定義します。

引数

環境変数名　～<環境変数名>((1～255 バイト))

設定する環境変数名を指定します。

Windows の場合，環境変数の名前は，環境設定パラメーター VAR_ENV_NAME_LOWERCASE に DISABLE を指定すると，大文字のシェル変数名として取り込まれます。

環境変数値　～<任意文字列>((0～1 行に記述できるまで))

環境変数に設定する値を指定します。

環境変数値にスペースを含む場合などは，ダブルクォーテーション（"），シングルクォーテーション（'）で囲むか，またはエスケープ文字（¥）を使用してスペースをエスケープすることで指定できます。ダブルクォーテーション（"）で囲まれた文字列に「¥」を指定すると，その後ろに指定された文字に関係なく，すべてエスケープ文字として扱われます。そのため，ダブルクォーテーション（"）で囲まれた文字列に「¥」を指定する場合は，「¥¥」と指定してください。

任意文字列中に${PATH}を記述することで，その時点の PATH 環境変数値を挿入できます。挿入したい個所に次の形式で指定します。

```
export 環境変数名=[任意文字列]${PATH}[任意文字列]
```

${PATH}による PATH 環境変数値は，ダブルクォーテーション（"），シングルクォーテーション（'），またはエスケープ文字（¥）の指定に関係なく，そのまま挿入されます。PATH 環境変数値が挿入された結果の文字列全体に対してダブルクォーテーション（"），シングルクォーテーション（'），またはエスケープ文字（¥）の指定が有効になり，最終的に環境変数に値が設定されます。

Windows の PATH 環境変数を定義する場合などは，PATH 環境変数値が挿入された結果の空白や¥が正しく解釈されるよう，シングルクォーテーション（'）で囲んでください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，次の順で環境変数が設定されます。
  - システム環境ファイルでの設定
  - ジョブ環境ファイルでの設定
- export パラメーターで LANG 環境変数の値を設定しても，この環境設定パラメーターを読み込んだ adshexec コマンド自身のプロセスのロケールは変更されません。【UNIX 限定】
- 環境変数に格納できる文字列の上限サイズは 32,766 バイトです。Windows では PATH 環境変数に長大な文字列を格納した場合，上限サイズに達していなくても，OS の API でエラーが発生することがあります。このため，上限サイズに迫るような文字列が PATH 環境変数に格納されないようにしてください。

使用例

- 環境変数 AAA に環境変数の値 BBB を設定します。

```
export AAA=BBB
```

- 環境変数 AAA に環境変数の値 BBB を設定するときの誤りの例です。

```
AAA=BBB
export AAA
```

- 既存の PATH 環境変数に/opt/jp1as/bin を追加します。

```
export PATH=/opt/jp1as/bin:${PATH}
```

パス区切り文字には「:」（コロン）を使用します。

- 既存の PATH 環境変数に C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin を追加します。

```
export PATH='C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin;${PATH}'
```

パス区切り文字には「;」（セミコロン）を使用します。

7.3.19 INIT_SCRIPT_READ パラメーター（初期設定スクリプトファイルを読み込み，実行するかどうかを定義する）

形式

```
#-adsh_conf INIT_SCRIPT_READ {YES | NO}
```

機能

ジョブコントローラ起動時に初期設定スクリプトファイルを読み込み，実行するかどうかを指定します。

オペランド

YES

ジョブコントローラは，ルートジョブを実行する直前に初期設定スクリプトファイルを読み込み，実行します。

ただし，次のような場合は，初期設定スクリプトファイルを読み込まないでエラー終了します。

- 初期設定スクリプトファイルが存在しない。
- 初期設定スクリプトファイルに読み込み権限が割り当てられていない。

NO

ジョブコントローラは，初期設定スクリプトファイルを読み込みません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.20 HOSTNAME_JP1IM_MANAGER パラメーター（JP1 イベントの送信先である JP1/IM - Manager が稼働している運用管理サーバを指定する）

形式

```
#-adsh_conf HOSTNAME_JP1IM_MANAGER JP1/IM - Managerが稼働している運用管理サーバのホスト名
```

機能

ユーザー応答機能を使用する場合に，JP1 イベントの送信先として，JP1/IM - Manager が稼働している運用管理サーバのホスト名を指定します。

オペランド

JP1/IM - Manager が稼働する運用管理サーバのホスト名　～<記号名称>((1～255 文字))《JP1/Advanced Shell が稼働するバッチ業務サーバの物理ホスト名》

JP1/IM - Manager が稼働する運用管理サーバのホスト名を指定します。指定されたホストでは，ユーザー応答機能に関する次の作業ができるようになります。

- JP1/IM - Manager に接続した JP1/IM - View から，JP1 イベントを確認できます。
- 応答待ちイベントに対して応答できます。

ホスト名の指定では次の点に注意してください。

- JP1/IM - Manager が稼働していないホストを指定した場合，JP1/Advanced Shell が出力する JP1 イベントは JP1/IM - View に表示されません。そのため，adshread コマンドは応答待ち状態のままとなります。その場合，このパラメーターで指定したホスト上で JP1/IM - Manager を起動するか，adshchmsg コマンドを実行して手動で応答してください。
- JP1/IM - Manager が稼働する運用管理サーバは，ホスト名で指定してください。IP アドレスで指定した場合，JP1/IM に対して JP1 イベントは発行されますが，応答待ちイベントとして扱われません。また，指定したホスト名は名前解決されているか確認してください。

このパラメーターを省略した場合は，JP1/Advanced Shell が稼働するバッチ業務サーバで hostname コマンドを実行したときに表示されるホスト名が仮定されます。

注意事項

- このパラメーターはジョブ環境ファイルに指定しないでください。
- このパラメーターを指定する場合は，物理ホストまたは論理ホストごとに応答要求メッセージの最大同時出力数を決定し，USERREPLY_WAIT_MAXCOUNT パラメーターに指定してください。
- このパラメーターで指定した運用管理サーバ上で，JP1/Advanced Shell が動作するバッチ業務サーバのホスト名が名前解決できることを確認してください。

- ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行した場合，このパラメーターの指定は無視されます。

7.3.21 JOBEXECLOG_PRINT パラメーター（ジョブ終了時に標準エラー出力へ出力するジョブ実行ログの内容を定義する）

形式

```
#-adsh_conf JOBEXECLOG_PRINT  {JOBLOG SCRIPT STDERR|STDERR}
```

機能

ジョブ終了時に標準エラー出力へ出力するジョブ実行ログの内容を定義します。このパラメーターで定義した内容が，adshexec コマンドを実行した端末画面や，JP1/AJS - View の［実行結果詳細］ダイアログボックスなどに表示されます。

JP1/Advanced Shell - Developer または adshexec -d コマンドによるデバッグ実行では，このパラメーターの指定に関係なく，ジョブ終了時にはジョブ実行ログの内容は標準エラー出力へ出力されません。

オペランド

JOBLOG SCRIPT STDERR

ジョブ終了時に次の内容を標準エラー出力へ出力します。

- JOBLOG の内容（コマンドの実行結果やファイルの割り当て結果など，ジョブの動作状況を示すメッセージ）

  ルートジョブの標準エラー出力へ出力される，子孫ジョブの JOBLOG の内容も含まれます。
- ジョブ定義スクリプト
- ジョブ実行中の標準エラー出力の内容

STDERR

ジョブ終了時に標準エラー出力へ出力する内容は，ジョブ実行中の標準エラー出力だけとします。

標準エラー出力には，もともと出力されている情報メッセージに加えて，メッセージ種別が I のメッセージ（通常は JOBLOG ファイルに出力される情報メッセージ）以外のメッセージも出力されます。

ルートジョブの標準エラー出力へ出力される子孫ジョブの JOBLOG の内容は出力されません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターに対して，同一環境ファイル内で次のどれかの指定をした場合，JP1/Advanced Shell のジョブコントローラはジョブを実行しないで，エラー終了します。

- このパラメーターを 2 つ以上指定した場合
- オペランドを 1 つも指定しなかった場合
- オペランドの数が 3 つよりも多い場合
- 「JOBLOG SCRIPT STDERR」または「STDERR」以外のオペランドを指定した場合
- 「JOBLOG SCRIPT STDERR」オペランドの指定順序が誤っている場合

- ジョブが簡潔出力モードまたは最小出力モードで動作する場合は，このパラメーターの指定に関係なく，ジョブ終了時にジョブ実行ログを標準エラー出力へ出力しません。

使用例

- ジョブ実行中の標準エラー出力の内容だけを標準エラー出力に出力します。

```
#-adsh_conf JOBEXECLOG_PRINT STDERR
```

7.3.22 JOBLOG_SUPPRESS_MSG パラメーター（ジョブ実行ログへ出力させないメッセージを定義する）

形式

```
#-adsh_conf JOBLOG_SUPPRESS_MSG メッセージID
```

機能

ジョブ実行ログへ出力させないメッセージのメッセージ ID を指定します。

メッセージ ID を複数指定する場合は，メッセージ ID の数だけこのパラメーターを指定してください。パラメーターの指定順序は任意です。

同じメッセージ ID を複数回指定しても，エラーにはなりません。1 回指定したものと見なされます。

このパラメーターの指定とメッセージの出力の関係を次の表に示します。

項番	メッセージの出力先	抑止するメッセージ ID の指定	
		指定あり	指定なし
1	スプールのジョブ実行ログのファイル	×	○
2	デバッガのコンソール	×	○
3	エディタのコンソール	×	○
4	システム実行ログ	×	○
5	上記以外の出力先（標準出力など）	×	×

（凡例）

○：メッセージを出力します。

×：指定したメッセージは出力しません。

オペランド

メッセージ ID　～((10 バイト))

ジョブ実行ログへの出力を抑止するメッセージのメッセージ ID を指定します。指定できるメッセージ ID は次のとおりです。

抑止できるメッセージの内容		指定できるメッセージ ID
組み込みコマンド	シェル変数更新	KNAX6110-I KNAX6111-I KNAX6120-I KNAX6121-I
	シェル標準コマンド	KNAX6112-I KNAX6113-I KNAX6122-I KNAX6123-I
外部コマンド	実行形式ファイル	KNAX6116-I KNAX6117-I KNAX6126-I KNAX6127-I
	スクリプトファイル	
シェル拡張コマンド		KNAX6114-I KNAX6115-I KNAX6124-I KNAX6125-I
子孫ジョブの実行に関するパラメーター	CHILDJOB_PGM パラメーター	KNAX6830-I
	CHILDJOB_SHEBANG パラメーター	KNAX6831-I
	CHILDJOB_EXT パラメーター	KNAX6832-I
関数（CMDRC_CMDGRP_CHECK パラメーターに FUNCTION を指定した場合）		KNAX6118-I KNAX6119-I KNAX6128-I KNAX6129-I

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。

使用例

- メッセージ KNAX6110-I，KNAX6111-I をジョブ実行ログに出力しないようにします。

```
#-adsh_conf JOBLOG_SUPPRESS_MSG KNAX6110-I
#-adsh_conf JOBLOG_SUPPRESS_MSG KNAX6111-I
```

7.3.23 KSH_ENV_READ パラメーター（シェル変数 ENV を読み込むかどうかを定義する）

形式

Windows，Linux の場合

```
#-adsh_conf KSH_ENV_READ  {YES|NO}
```

AIX，HP-UX，Solaris の場合

```
#-adsh_conf KSH_ENV_READ  {YES|NO}
```

機能

ジョブコントローラ起動時にシェル変数 ENV を読み込み，変数の値が示すスクリプトファイルを実行するかどうかを指定します。

オペランド

YES

ジョブコントローラ起動時にシェル変数 ENV を読み込みます。

NO

ジョブコントローラ起動時にシェル変数 ENV を読み込みません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.24 LOG_DIR パラメーター（システム実行ログ出力ディレクトリのパス名を定義する）

形式

```
#-adsh_conf LOG_DIR パス名
```

機能

多重実行するジョブのメッセージをシステム実行ログとして 1 つのファイルに集めます。このシステム実行ログを出力するディレクトリのパス名を定義します。

オペランド

パス名

Windows の実行環境の場合　～<パス名>((1～128 バイト))《共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥log》

Windows の開発環境の場合　～<パス名>((1～128 バイト))《共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥log》

UNIX の場合　～<パス名>((1～512 バイト))《/opt/jp1as/log》

システム実行ログを出力するディレクトリのパス名を指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターに，JP1/Advanced Shell でサポートしないファイルシステム上のディレクトリを指定しないでください。JP1/Advanced Shell でサポートしないファイルシステムの詳細については，「(2)　ファイルシステム」を参照してください。
- パス名に&，(,)，[,]，{,}，^，＝，;，!，'，＋，,，`，～，#，%の記号を含まないでください。これらの記号を含む場合，正常に動作しません。

補足

- 該当するパラメーターで指定するディレクトリを別々に分けることで，同一ホスト内で複数環境を使い分けられます。

7.3.25 LOG_FILE_CNT パラメーター（システム実行ログをバックアップする面数を定義する）

形式

```
#-adsh_conf LOG_FILE_CNT 面数
```

機能

システム実行ログをバックアップする面数を定義します。

オペランド

面数　～<符号なし整数>((1～64))《4》

システム実行ログをバックアップする面数を指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。ただし，LOG_DIR パラメーターの定義がシステム環境ファイルの指定と同一の場合，システム環境ファイルと異なる値を指定するとエラーになります（システム環境ファイルの指定がデフォルト値の場合でも，相違があればエラーになります）。

補足

- 複数のユーザーが同じファイルにシステム実行ログを出力している場合には，LOG_FILE_CNT と LOG_FILE_SIZE は，最後にシステム実行ログの出力を開始したユーザーの設定値が有効になります。同じファイルにシステム実行ログを出力するユーザー同士は，同じ設定値にすることを推奨します。

7.3.26 LOG_FILE_SIZE パラメーター（システム実行ログを出力するファイルサイズを定義する）

形式

```
#-adsh_conf LOG_FILE_SIZE ファイルサイズ
```

機能

システム実行ログを出力するファイルサイズを定義します。

オペランド

ファイルサイズ　～<符号なし整数>((1～16))《2》

システム実行ログを出力するファイルサイズを MB で指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。ただし，LOG_DIR パラメーターの定義がシステム環境ファイルの指定と同一の場合，システム環境ファイルと異なる値を指定するとエラーになります（システム環境ファイルの指定がデフォルト値の場合でも，相違があればエラーになります）。

補足

- 複数のユーザーが同じファイルにシステム実行ログを出力している場合には，LOG_FILE_CNT と LOG_FILE_SIZE は，最後にシステム実行ログの出力を開始したユーザーの設定値が有効になります。同じファイルにシステム実行ログを出力するユーザー同士は，同じ設定値にすることを推奨します。

7.3.27　OUTPUT_MODE_CHILD パラメーター（子孫ジョブの実行結果の出力情報に関する出力方式を定義する）

形式

```
#-adsh_conf OUTPUT_MODE_CHILD  {EXTENDED|SIMPLE|MINIMUM}
```

機能

子孫ジョブの実行結果を次のどの方式で出力するか選択します。

- 拡張出力モード（デフォルト）
- 簡潔出力モード
- 最小出力モード

簡潔出力モードを選択すると情報メッセージなどの出力を抑止でき，コマンドの実行結果だけが出力されるため，ジョブの実行結果をほかのプログラムで利用しやすくなります。最小出力モードを選択すると，簡潔出力モードよりさらに多くのメッセージの出力を抑止できます。

なお，標準出力と標準エラー出力の出力先は，子孫ジョブ起動時の出力先を引き継ぎます。

子孫ジョブを起動する adshexec コマンドに-m オプションを指定してジョブを実行した場合，このパラメーターの指定よりも-m オプションの指定が優先されます。

各出力モードの詳細は次の説明を参照してください。

- 「2.6.8　ジョブ実行結果とログの出力情報を定義する」
- 「3.4.1　標準出力，標準エラー出力の出力先に関する指定」
- 「3.4.2　ジョブの実行結果をスプールに出力する」
- 「3.4.4　ジョブ実行ログへの情報メッセージと警告メッセージの出力を抑止する」
- 「3.5.1　ジョブの種類ごとのジョブ実行ログの出力内容」
- 「3.5.4　ジョブ実行ログの出力例（簡潔出力モードまたは最小出力モードを選択した場合）」
- 「12.2　メッセージの出力先」

オペランド

EXTENDED

拡張出力モードを選択します。子孫ジョブの実行後に出力される情報は次のようになります。

- 子孫ジョブ終了時にジョブ実行ログは標準エラー出力に出力されます。
- ジョブコントローラの標準エラー出力と標準出力への出力メッセージはすべて出力されます。

SIMPLE

簡潔出力モードを選択します。子孫ジョブの実行後に出力される情報は次のようになります。

- 子孫ジョブ終了時にジョブ実行ログは標準エラー出力に出力されません。
- ジョブコントローラの標準エラー出力と標準出力へ出力されるメッセージは，エラーメッセージだけとなります。また，JOBLOG に出力されるエラーメッセージは標準エラー出力に出力されます。

MINIMUM

最小出力モードを選択します。子孫ジョブの実行後に出力される情報は次のようになります。

- 子孫ジョブ終了時にジョブ実行ログは標準エラー出力に出力されません。
- ジョブコントローラの標準エラー出力と標準出力へ出力されるメッセージは，コマンド，ジョブステップ，ジョブの終了コードの通知を除くエラーメッセージだけとなります。このほかに，シグナル，イベント受信を通知するメッセージも抑止されます。簡潔出力モードと異なり，抑止されるメッセージはスプールジョブディレクトリ下の JOBLOG にも出力されません。

  また，JOBLOG に出力されるエラーメッセージは標準エラー出力に出力されます。

環境設定パラメーターの解析が終了するまでは，親プロセスの出力モードを引き継いで動作します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.28 OUTPUT_MODE_ROOT パラメーター（ルートジョブの実行結果の出力情報に関する出力方式を定義する）

形式

```
#-adsh_conf OUTPUT_MODE_ROOT  {EXTENDED|SIMPLE|MINIMUM}
```

機能

ルートジョブの実行結果を次のどの方式で出力するか選択します。

- 拡張出力モード（デフォルト）
- 簡潔出力モード
- 最小出力モード

簡潔出力モードを選択すると情報メッセージなどの出力を抑止でき，コマンドの実行結果だけが出力されるため，ジョブの実行結果をほかのプログラムで利用しやすくなります。最小出力モードを選択すると，簡潔出力モードよりさらに多くのメッセージの出力を抑止できます。

ルートジョブを起動する adshexec コマンドに-m オプションを指定してジョブを実行した場合，このパラメーターの指定よりも-m オプションの指定が優先されます。

オペランド

EXTENDED

拡張出力モードを選択します。ルートジョブの実行後に出力される情報は次のようになります。

- 標準出力と標準エラー出力はスプールのファイルにリダイレクトされます。
- ジョブ終了時にジョブ実行ログは標準エラー出力に出力されます。
- ジョブコントローラの標準エラー出力と標準出力への出力メッセージはすべて出力されます。
- デバッグ実行時は，JOBLOG が標準エラー出力にタイムリーに出力されます。

SIMPLE

簡潔出力モードを選択します。ルートジョブの実行後に出力される情報は次のようになります。

- ジョブ終了時にジョブ実行ログは標準エラー出力に出力されません。
- 標準出力と標準エラー出力の出力先は，起動時の出力先を引き継ぎます。
- ジョブコントローラの標準エラー出力と標準出力へ出力されるメッセージは，エラーメッセージだけとなります。また，JOBLOG に出力されるエラーメッセージは標準エラー出力に出力されます。
- デバッグ実行時は，エラーメッセージを除いて JOBLOG を標準エラー出力に出力しません。ジョブコントローラの標準エラー出力と標準出力へのメッセージを出力します。デバッグ終了時のエラーメッセージ以外のメッセージは出力しません。

MINIMUM

最小出力モードを選択します。ルートジョブの実行後に出力される情報は次のようになります。

- ジョブ終了時にジョブ実行ログは標準エラー出力に出力されません。
- 標準出力と標準エラー出力の出力先は，起動時の出力先を引き継ぎます。
- ジョブコントローラの標準エラー出力と標準出力へ出力されるメッセージは，コマンド，ジョブステップ，ジョブの終了コードの通知を除くエラーメッセージだけとなります。このほかに，シグナル，イベント受信を通知するメッセージも抑止されます。簡潔出力モードと異なり，抑止されるメッセージはスプールジョブディレクトリ下の JOBLOG にも出力されません。

  また，JOBLOG に出力されるエラーメッセージは標準エラー出力に出力されます。
- デバッグ実行時は，エラーメッセージを除いて JOBLOG を標準エラー出力に出力しません。ジョブコントローラの標準エラー出力と標準出力へのメッセージを出力します。デバッグ終了時のエラーメッセージ以外のメッセージは出力しません。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- SIMPLE または MINIMUM を指定した場合は，OUTPUT_STDOUT パラメーターで SPOOL を指定するか，adshexec -s SPOOL と指定してコマンドを実行しても，標準出力はスプールのファイルにリダイレクトされません。

7.3.29 OUTPUT_STDOUT パラメーター（ルートジョブの出力先を定義する）

形式

```
#-adsh_conf OUTPUT_STDOUT {SPOOL|PARENT}
```

機能

ルートジョブの標準出力の出力先を定義します。子孫ジョブは，このオプションに PARENT が指定されたものとして動作します。このオプションを省略した場合，ルートジョブは SPOOL が指定されたものとして動作します。

adshexec コマンドに-s オプションを指定してジョブを実行した場合，このパラメーターの指定よりも-s オプションの指定が優先されます。

ルートジョブが簡潔出力モードまたは最小出力モードで動作（OUTPUT_MODE_ROOT パラメーターまたは adshexec コマンドの-m オプションで指定）する場合は，OUTPUT_STDOUT パラメーターまたは adshexec コマンドの-s オプションの指定に関係なく，プロセス起動時の標準出力を継承します。

オペランド

{SPOOL|PARENT}

ルートジョブの標準出力の出力先として，次のどちらかを指定します。

- SPOOL
  ルートジョブの標準出力をスプール内のファイルに出力します。
- PARENT
  ルートジョブの標準出力を，プロセス起動時に親プロセスから継承した出力先に出力します。親プロセスで出力先をリダイレクトしていない場合は，親プロセスと同じ出力先に出力します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.30　PATH_CONV パラメーター（パス変換内容を定義する）

形式

```
#-adsh_conf PATH_CONV パス名1 パス名2
```

機能

ジョブ定義スクリプト中の変換前・変換後のパス名を定義します。

ジョブ定義スクリプトの実行時に，パス区切り文字（PATH_CONV_ENABLE パラメーターで定義）で区切られた文字列の中でパス名 1 と前方一致する文字列をパス名 2 に置換します。また，PATH_CONV_ENABLE パラメーターで定義したパス区切り文字およびディレクトリ区切り文字も変換対象になります。

なお，Windows では PATH_CONV_RULE パラメーターで選択するパス変換ルールによって変換結果が異なります。詳細については，「PATH_CONV_RULE パラメーター（パス変換ルールを定義する）【Windows 限定】」を参照してください。

PATH_CONV_ENABLE パラメーターが定義されていない場合，このパラメーターは無効となります。また，このパラメーターを複数定義した場合，環境ファイルの先頭から調べて，最初に変換条件に合致した定義が適用されます。

オペランド

パス名 1　～<パス名>((1～247 バイト))

変換前のパスを指定します。スペースを含む値を指定する場合は，"（ダブルクォーテーション）で囲んでください。

パス名 1 には，Windows の場合は UNIX のパス，UNIX の場合は Windows のパスを指定できます。JP1/Advanced Shell では「¥」をエスケープ文字として扱うため，「¥」を指定する場合は，「¥¥」と指定してください。なお，次の文字は使用できません。

＊　？　＜　＞　｜　｀（バッククォーテーション）　$

パス名にはディレクトリ区切り文字が含まれている必要があります。ディレクトリ区切り文字は次の文字を指定してください。

- Windows の場合は「/」を指定します。
- UNIX の場合は「¥¥」を使用します。

パス名 2　～<パス名>((1～247 バイト))

変換後のパスを指定します。スペースを含む値を指定する場合は，"（ダブルクォーテーション）で囲んでください。JP1/Advanced Shell では「¥」をエスケープ文字として扱うため，「¥」を指定する場合は，「¥¥」と指定してください。なお，次の文字は使用できません。

＊　？　＜　＞　｜　｀（バッククォーテーション）　$

パス名にはディレクトリ区切り文字が含まれている必要があります。ディレクトリ区切り文字は次の文字を指定してください。

- Windows で実行する場合は，「¥¥」を指定します。
- UNIX で実行する場合は，「/」を使用します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。ただし，システム環境ファイルとジョブ環境ファイルの定義の合計が 255 個を超えた場合，エラーになります。
- このパラメーターによる変換は行ごとに実施されます。このため，ジョブ定義スクリプト中のパス名の該当する個所に改行が含まれている場合，正しく変換されません。
- コメント内の文字列も変換されます。
- 変換する規則を定義した順に検索し，最初に変換条件に合致したものだけが適用されます。
- SPOOLJOB_CHILDJOB パラメーターに DELETE を指定した場合，子孫ジョブとして実行するジョブ定義スクリプトにはスクリプトイメージが出力されません。そのため，子孫ジョブで実行したジョブ定義スクリプトに対して，このパラメーターで定義した変換規則に基づいて変換した場合，変換結果は出力されないので注意してください。
- パス変換ルール 2 の場合でも，「"（ダブルクォーテーション）」で囲んだ範囲内に「'（シングルクォーテーション）」を入れ子として指定することはできません。指定するとパス変換の対象となるので注意してください。

使用例

- UNIX 用に作成したジョブ定義スクリプトファイルを Windows で実行する場合に指定します。

```
#-adsh_conf PATH_CONV /home/hitachi "C:¥¥hitachi"
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /tmp/jp1as "D:¥¥jp1as_tmp"
#-adsh_conf PATH_CONV /tmp "C:¥¥temp"
```

- Windows 用に作成したジョブ定義スクリプトファイルを UNIX で実行する場合に指定します。

```
#-adsh_conf PATH_CONV "C:¥¥hitachi" /home/hitachi
#-adsh_conf PATH_CONV_ENABLE ¥¥ ;
#-adsh_conf PATH_CONV "D:¥¥jp1as_tmp" /tmp/jp1as
#-adsh_conf PATH_CONV "C:¥¥temp" /tmp
```

7.3.31 PATH_CONV_ACCESS パラメーター（ファイル入出力時のパス変換内容を定義する）

形式

```
#-adsh_conf PATH_CONV_ACCESS パス名1 パス名2
```

機能

ファイルの入出力時にジョブ定義スクリプト中のファイルパス名を変換するため，変換前・変換後のパス名を定義します。

ジョブ定義スクリプトを実行して入出力が発生した場合に，入出力を行うファイルのファイルパス名がパス名 1 と完全一致するときは，パス名 2 に変換します。パス名 1 およびパス名 2 は必ず指定します。

同じファイルパス名に異なる規則を定義した場合は，先に定義した規則が有効になります。

.（ドット）コマンド，および#-adsh_script コマンドに指定した外部スクリプトについては，COMMAND_CONV_ARG パラメーターの変換対象となります。COMMAND_CONV_ARG パラメーターの詳細については，「COMMAND_CONV_ARG パラメーター（コマンド実行時にジョブ定義スクリプト中の引数を変換する規則を定義する）」を参照してください。

変換結果は，KNAX6803-I メッセージ，または KNAX6805-I メッセージでジョブ実行ログに出力します。

環境ファイルに PATH_CONV_ENABLE パラメーターが定義されていない場合は，PATH_CONV_ACCESS パラメーターは有効になりません。

オペランド

パス名 1　～<パス名>((1～247 バイト))

変換前のファイルパスを指定します。スペースを含むファイルパスを指定する場合は，"（ダブルクォーテーション）で囲む必要があります。ただし，"で囲んだ値にスペース，タブ文字または空文字だけを指定できません。なお，次の文字は使用できません。

＊　？　＜　＞　｜　｀（バッククォーテーション）　$

パス名 1 を指定しなかった場合，またはパス名 1 に不当な値を指定した場合は，パラメーター解析時にエラー終了します。

パス名 2　～<パス名>((1～247 バイト))

変換後のファイルパスを指定します。スペースを含むファイルパスを指定する場合は，"（ダブルクォーテーション）で囲む必要があります。ただし，"で囲んだ値にスペース，タブ文字または空文字だけを指定できません。なお，次の文字は使用できません。

＊　？　＜　＞　｜　｀（バッククォーテーション）　$

パス名 2 を指定しなかった場合，またはパス名 2 に不当な値を指定した場合は，パラメーター解析時にエラー終了します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，両方の定義が有効になります。ただし，システム環境ファイルとジョブ環境ファイルの定義の合計が 255 個を超えた場合，エラーになります。
- このパラメーターによる変換は，変数置換およびファイル名置換が解決した内容で実施されます。
- 変換する規則を定義した順に検索し，最初に変換条件に合致したものだけが適用されます。
- このパラメーターと PATH_CONV パラメーターで同じパス名に対して変換を定義した場合，PATH_CONV パラメーターによる変換が先に実施されます。PATH_CONV パラメーターで変換されたパス名を PATH_CONV_ACCESS パラメーターでさらに変換する場合は，PATH_CONV パラメーターで変換されたパス名を指定してください。
- パス名 2 に，メタキャラクタではなく文字列「¥」そのものを含む文字列を指定する場合，「¥」を「¥¥」と記述してください。

使用例

- UNIX 用に作成したジョブ定義スクリプトを Windows で実行するため，"/dev/null"を"nul"に変換します。

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_ACCESS /dev/null nul
```

- Windows 用に作成したジョブ定義スクリプトを UNIX で実行するため，"nul"を"/dev/null"に変換します。

```
#-adsh_conf PATH_CONV_ENABLE ¥¥ ;
#-adsh_conf PATH_CONV_ACCESS nul /dev/null
```

7.3.32　PATH_CONV_ENABLE パラメーター（パス変換機能を有効にする）

形式

```
#-adsh_conf PATH_CONV_ENABLE ディレクトリ区切り文字 パス区切り文字
```

機能

パス変換機能を有効にします。すでに有効になっていた場合，メッセージを出力し，処理を終了します。

オペランド

ディレクトリ区切り文字　～((1 または 2 バイト))

パス変換機能で変換する前のパス名のディレクトリ区切り文字を指定します。「/」または「¥¥」を指定できます。

パス区切り文字　～((1 バイト))

パス変換機能で変換する前のパス名のパス区切り文字を指定します。「:」（コロン），または「;」（セミコロン）を指定できます。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

使用例

- Windows で実行する場合に指定します。

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /tmp "C:¥¥temp"
```

- UNIX で実行する場合に指定します。

```
#-adsh_conf PATH_CONV_ENABLE ¥¥ ;
#-adsh_conf PATH_CONV "C:¥¥temp" /tmp
```

7.3.33 PATH_CONV_NOVAR パラメーター（パス名を扱わないシェル変数を定義する）

形式

```
#-adsh_conf PATH_CONV_NOVAR シェル変数名 [シェル変数名 …]
```

機能

パス名を扱わないシェル変数名を定義します。

オペランド

シェル変数名～<[*][文字列][*]>((1～255))

パス名を扱わないシェル変数名を定義します。指定できるシェル変数名の個数に上限はありません（環境設定ファイルの 1 行の長さの上限は 4,092 バイトであり，1 行に記述できないときは，複数のパラメーターに分けて記述します）。

同一シェル変数名を重複指定してもエラーになりません。

シェル変数名は*を使用して総称変数名を指定できます。

*だけは，すべてのシェル変数名を示します。

*は 0 文字以上の文字を示します。

文字列*文字列の形式は指定できません。

**は指定できません。

シェル変数名 dirA001,dirA002,A001,A002,A0,C0B0D0 がある場合の指定例を次に示します。

指定例	対象となるシェル変数
#-adsh_conf PATH_CONV_NOVAR *	dirA001,dirA002,A001,A002,A0,C0B0D0 （すべて）
#-adsh_conf PATH_CONV_NOVAR *A001	dirA001,A001
#-adsh_conf PATH_CONV_NOVAR *B0*	C0B0D0
#-adsh_conf PATH_CONV_NOVAR dir*	dirA001,dirA002
#-adsh_conf PATH_CONV_NOVAR dirA001 #-adsh_conf PATH_CONV_NOVAR A002	dirA001,A002
#-adsh_conf PATH_CONV_NOVAR dirA001 A002	dirA001,A002

この機能が有効になるのは PATH_CONV_ENABLE 環境設定パラメーターの指定がある場合だけです。ただし，PATH_CONV_ENABLE 環境設定パラメーターの指定がなくてもエラーにはなりません。

このパラメーターの指定は PATH_CONV_VAR 環境設定パラメーターと指定順でマージされ，後の指定が優先します。同一パラメーター内で複数のシェル変数名を指定した場合でも後の指定が優先します。

使用例

dir で始まる名称のシェル変数はパス名を扱うシェル変数として定義しますが，dirno で始まる名称のシェル変数はパス名を扱わないシェル変数として定義します。

```
#-adsh_conf PATH_CONV_VAR dir*
#-adsh_conf PATH_CONV_NOVAR dirno*
```

7.3.34 PATH_CONV_RULE パラメーター（パス変換ルールを定義する）【Windows 限定】

形式

```
#-adsh_conf PATH_CONV_RULE  {1|2}
```

機能

パス変換ルールを選択します。指定を省略した場合はパス変換ルール 1 が設定されます。

また，PATH_CONV_ENABLE パラメーターでパス区切り文字が定義されている場合，区切られた範囲が変換対象となります。PATH_CONV_ENABLE パラメーターで定義された区切り文字は，ジョブ定義スクリプトを実行した OS で使用される区切り文字へ変換されます。

オペランド

1

パス変換ルール 1 を選択します。

「"（ダブルクォーテーション）」で囲まれた範囲だけが変換対象となります。

2

パス変換ルール 2 を選択します。

項目「パス変換ルール 2 で使用する区切り文字」に示す区切り文字で区切られた文字列が変換対象となります。ここで区切られた文字列は，パス区切り文字（PATH_CONV_ENABLE パラメーターで定義）でさらに区切られてから変換されます。ただし，' '内と${ }内の文字は変換されません。

変換個所が「"（ダブルクォーテーション）」で囲まれていない場合，変換結果が「"（ダブルクォーテーション）」で囲まれます。それに加えて次に示す範囲も「"（ダブルクォーテーション）」で囲まれます。

- パス区切り文字

  「";"」に変換されます。

- 「$**シェル変数名**」

  #-adsh_path_var コマンドまたは PATH_CONV_VAR パラメーターで指定していないシェル変数も含みます。PATH_CONV_NOVAR パラメーターで指定したシェル変数も含みます。

「$」の後ろから，英数字（先頭文字の場合は英字）と「_（アンダースコア）」のどちらでもない文字が現れるまでの範囲をシェル変数名と見なします。シェル変数名がない場合は「$」だけを「"（ダブルクォーテーション）」で囲むことはしません。

- 「${」から「}」までの範囲

  #-adsh_path_var コマンドまたは PATH_CONV_VAR パラメーターで指定していないシェル変数も含みます。PATH_CONV_NOVAR パラメーターで指定したシェル変数も含みます。

パス変換ルール 2 で使用する区切り文字

パス変換ルール 2 で使用する区切り文字と，区切り文字が有効/無効となる位置を次に示します。

区切り文字	区切り文字の位置						
	' '内	" "内※1	` `内※1	¥直後の 1 文字	$( )内※1	${ }内※1	その他
|	×	×	○	×	○	※2	○
&	×	×	○	×	○	※2	○
;	×	×	○	×	○	※2	○
<	×	×	○	×	○	※2	○
>	×	×	○	×	○	※2	○
(	×	※3	○	×	○	※2	○
)	×	×	○	×	○	※2	○
`	×	○	○	×	○	※2	○
'	○	×	○	×	○	※2	○
"	×	○	○	×	○	※2	○
#	×	×	○	×	○	※2	○
=	×	×	○	×	○	※2	○
スペース（タブ記号を含む）	×	×	○	×	○	※2	○
改行コード	×	×	○	×	○	※2	○

（凡例）

○：区切り文字が有効となります。

×：区切り文字が無効となります。

注※1

囲んだ範囲内に別の範囲を入れ子にできます。入れ子にできる組み合わせを次に示します。

範囲	入れ子とする範囲					
	' '	" "	` `	¥直後の 1 文字	$( )	${ }
" "内	×	×	○	○	○	○
` `内	○	○	×	○	○	○
$( )内	○	○	○	○	○	○
${ }内	○	○	○	○	○	○

(凡例)

○：入れ子にできます。

×：入れ子にできません。

範囲の終わりを示す文字が入れ子とする範囲にある場合は，範囲の終わりとは見なしません。例えば，「"¥""」と指定した場合，2 つ目の「"」は「¥直後」の範囲に含まれるため，1 つ目の「"」から始まる「"内」の範囲の終わりとは見なしません。

また，行末までに範囲の終わりを示す文字がない場合は，行末までを範囲内と見なします。

注※2

「${ }」を入れ子にしている範囲と同じです。

例えば，「` `内」なら「|」は有効となり，「" "内」なら無効となります。

注※3

「$(」の「(」だけ有効です。それ以外の「(」は無効となります。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- パス変換ルール 2 を選択すると，パス変換ルール 1 よりも変換範囲が広いため，期待どおりの変換結果が得られない場合があります。ジョブ定義スクリプトを実行する前に構文チェック機能を使用し，生成されたスクリプトイメージでパスの変換結果を確認してください。変換結果が適切でない場合は，パス変換ルールを変更するか，ジョブ定義スクリプトを変更して再実行してください。
- パス変換ルール 2 を選択すると，変数置換の変数やパターンは変換対象となりません。例を次に示します。

  環境設定パラメーター

  ```
  #-adsh_conf  PATH_CONV_ENABLE  / :
  #-adsh_conf  PATH_CONV_RULE  2
  #-adsh_conf  PATH_CONV  /tmp  d:¥¥temp
  ```

  変換前のジョブ定義スクリプト

  ```
  #-adsh_path_var  DIR
  AA=${DIR:-/tmp}
  ```

変換後のジョブ定義スクリプト

```
#-adsh_path_var  DIR
AA=”${DIR:-/tmp}”
```

この例では#-adsh_path_var コマンドで変数が定義されているため，${変数名}の部分が「"（ダブルクォーテーション）」で囲まれていますが，/tmp は変換されていません。これを回避するには次のように/tmp を変数として指定するなど，ジョブ定義スクリプトを変更してください。

```
#-adsh_path_var  DIR
BB=/tmp
AA=${DIR:-$BB}
```

- パス変換ルール 2 では文字列置換として変換されます。そのため，同じパスを指していても指定文字列が異なれば同じ変換結果にならないことがあります。
- パス変換ルール 2 ではヒアドキュメントのドキュメント部分も変換されるため，プログラムが処理できないデータへ変換されないよう注意してください。対応方法を次に示します。
  - 変換規則を工夫する
  - 変数置換を利用して変換されないようにする

  これらの方法で対応できない場合は，パス変換ルール 1 へ変更するか，ヒアドキュメントの外部ファイル化を検討してください。

  ヒアドキュメントの変換例を次に示します。

  環境設定パラメーター

```
#-adsh_conf  PATH_CONV_ENABLE / :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV  /home/user001  d:¥¥home¥¥user001
```

  変換前のジョブ定義スクリプト

```
uap << EOF
IN=/home/user001/infile
FTP=/home/user001/ftp/outfile
EOF
```

  変換後のジョブ定義スクリプト

```
uap << EOF
IN=”d:¥¥home¥¥user001”¥¥infile
FTP=”d:¥¥home¥¥user001”¥¥ftp¥¥outfile
EOF
```

  これによって，ヒアドキュメント内のデータは次のように変換されます。

```
IN=”d:¥¥home¥¥user001”¥¥infile
FTP=”d:¥¥home¥¥user001”¥¥ftp¥¥outfile
```

  変換後のパスにはダブルクォーテーションが付加されているため，ユーザープログラムがダブルクォーテーションを適切に処理できなければ，ユーザープログラムは正しく動作しません。また，ftp コマン

ドでリモートサイトのパスを指定する場合など，パス名を変換したくないことがあります。この場合，パス名部分を変数に置き換えてヒアドキュメント外に出すなどの変更をしてください。

例えば，IN=/home/user001/infile のパスは変換したいが，FTP=/home/user001/ftp/outfile のパスを変換したくない場合は，次のように変更することで対応できます。

```
VARIN=/home/user001/infile
VARFTP='/home/user001/ftp/outfile'
uap << EOF
IN=$VARIN
FTP=$VARFTP
EOF
```

- 次の文字はパス変換ルールに合致すると変換されるため，注意してください。
  - 演算子
  - /で始まるコマンドのオプション文字
  - コマンドの引数に指定された，パス名でないディレクトリ区切り文字を含んだデータ
  - 三項演算子

三項演算子が変換された例を次に示します。

環境設定パラメーター

```
#-adsh_conf  PATH_CONV_ENABLE  /  :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV  A2/A1  A2¥¥A1
```

変換前のジョブ定義スクリプト

```
#-adsh_job JOB001
A1=10
A2=5
((BB=A1>A2?A1/A2:A2/A1))
```

変換後のジョブ定義スクリプト

```
#-adsh_job JOB001
A1=10
A2=5
((BB=A1>A2?A1¥¥A2";""A2¥¥A1"))
```

この例では，((BB=A1>A2?A1/A2:A2/A1))の最後の A2/A1 が変換ルールに一致して変換されています。これを防ぐには，次のどちらかの方法でジョブ定義スクリプトを変更してください。

・変数を「$**シェル変数名**」の形式で指定する。

```
  ((BB=$A1>$A2?$A1/$A2:$A2/$A1))
```

・if 文で記述し直す。

```
  if((A1>A2)) then
  ((BB=$A1/$A2))
  else
```

```
  ((BB=$A2/$A1))
  fi
```

- パス変換ルール 2 では，変換規則に一致した文字列に${**配列名**[*]}形式の配列名が含まれる場合，ダブルクォーテーションで囲まれます。また，変換結果である"${**配列名**[*]}"は，各要素が IFS シェル変数の値で区切られるため，変換前と結果が異なる場合があるので注意してください。
- パス変換ルール 2 では，「cd work」のようにディレクトリ区切り文字が含まれないパスは変換できません。この場合は，ジョブ定義スクリプトの記述を変更するか，ディレクトリ名を変更する変換は指定しないでください。
- パス変換ルール 2 では，「**コマンド** -p **パス名**」など，オプションの値としてパス名を指定するような場合，オプション文字とパス名の間はスペースで区切ってください。
- パス変換ルール 2 では，逆引用符形式のコマンド置換でコマンドに含まれるパスを変換しても，期待どおりの動作となりません。例を次に示します。

  環境設定パラメーター

  ```
  #-adsh_conf  PATH_CONV_ENABLE / :
  #-adsh_conf  PATH_CONV_RULE  2
  #-adsh_conf  PATH_CONV  /home/user001  d:¥¥home¥¥user001
  ```

  変換前のジョブ定義スクリプト

  ```
  cat file | grep `cat /home/user001/text`
  ```

  変換後のジョブ定義スクリプト

  ```
  cat file | grep `cat ”d:¥¥home¥¥user001”¥¥text`
  ```

  この例では，コマンド置換のコマンドで使用する¥はメタキャラクタとして処理されるため，最終的には¥¥が消去され，期待どおりの動作となりません。

  この場合，次のように$()形式のコマンド置換に変更してください。

  ```
  cat file | grep $(cat /home/user001/text)
  ```

- パス変換ルール 2 で，/dev/null を Windows の NUL デバイスに変更したい場合は，COMMAND_CONV_ARG パラメーターと PATH_CONV_ACCESS パラメーターを使用してください。
- パス変換ルール 2 で，パスを次のように記述した場合，正しく変換できません。

  環境設定パラメーター

  ```
  #-adsh_conf  PATH_CONV_ENABLE  / :
  #-adsh_conf  PATH_CONV_RULE  2
  ```

  変換前のスクリプト例

  ```
  #-adsh_path_var  homedir

  INPUT1=${homedir}”/test/data”
  INPUT2=”${homedir}”/test/data
  ```

パス変換は区切り文字で区切られた文字列単位に実施されるため，${homedir}/test/data がパス名であったとしても，ダブルクォーテーションによって${homedir}と/test/data の 2 つの文字列として別々に変換処理されます。そのため，次のような変換結果を期待しても期待どおりの変換とはなりません。

期待する変換結果

```
#-adsh_path_var  homedir

INPUT1="${homedir}""¥¥test¥¥data"
INPUT2="${homedir}"¥¥test¥¥data
```

実際の変換結果

```
#-adsh_path_var  homedir

INPUT1="${homedir}""/test/data"
INPUT2="${homedir}"/test/data
```

この場合は次のように記述を変更してください。

```
#-adsh_path_var  homedir

INPUT1="${homedir}/test/data"
INPUT2="${homedir}/test/data"
```

使用例

パラメーターの設定例を次に示します。ジョブ定義スクリプトの変換例は，「2.6.2　パス名を変換する」を参照してください。

- パス変換ルール 1 でパスを変換します。

```
#-adsh_conf  PATH_CONV_ENABLE  / :
#-adsh_conf  PATH_CONV_RULE  1
#-adsh_conf  PATH_CONV  /home/user001  d:¥¥home¥¥user001
```

- パス変換ルール 2 でパスを変換します。

```
#-adsh_conf  PATH_CONV_ENABLE  / :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV /home/user01  d:¥¥home¥¥user01
#-adsh_conf  PATH_CONV BB/AA BB¥¥AA
```

7.3.35　PATH_CONV_VAR パラメーター（パス名を扱うシェル変数を定義する）

形式

```
#-adsh_conf PATH_CONV_VAR シェル変数名 [シェル変数名 …]
```

機能

パス名を扱うシェル変数名を定義します。パス名を扱うシェル変数の周りの区切り文字はパス変換機能で変換されます。

オペランド

シェル変数名～<[*][文字列][*]>((1～255))

パス名を扱うシェル変数名を定義します。指定できるシェル変数名の個数に上限はありません（環境設定ファイルの1行の長さの上限は4,092バイトであり，1行に記述できないときは，複数のパラメーターに分けて記述します）。

同一シェル変数名を重複指定してもエラーとなりません。

シェル変数名は*を使用して総称変数名を指定することができます。

*だけは，すべてのシェル変数名を示します。

*は0文字以上の文字を示します。

文字列*文字列の形式は指定できません。

**は指定できません。

シェル変数名dirA001,dirA002,A001,A002,A0,C0B0D0がある場合の指定例を次に示します。

指定例	対象となるシェル変数
#-adsh_conf PATH_CONV_VAR *	dirA001,dirA002,A001,A002,A0,C0B0D0 (すべて)
#-adsh_conf PATH_CONV_VAR *A001	dirA001,A001
#-adsh_conf PATH_CONV_VAR *B0*	C0B0D0
#-adsh_conf PATH_CONV_VAR dir*	dirA001,dirA002
#-adsh_conf PATH_CONV_VAR dirA001 #-adsh_conf PATH_CONV_VAR A002	dirA001,A002
#-adsh_conf PATH_CONV_VAR dirA001 A002	dirA001,A002

この機能が有効になるのはPATH_CONV_ENABLE環境設定パラメーターの指定がある場合だけです。ただし，PATH_CONV_ENABLE環境設定パラメーターの指定がなくてもエラーにはなりません。

このパラメーターの指定はPATH_CONV_NOVAR環境設定パラメーターと指定順でマージされ，後の指定が優先します。同一パラメーター内で複数のシェル変数名を指定した場合でも後の指定が優先します。

注意事項

システム環境ファイルとジョブ環境ファイルの両方に指定がある場合，両方をマージします。また，システム環境ファイルよりもジョブ環境ファイルの指定が優先します。

使用例

dir で始まる名称のシェル変数はパス名を扱うシェル変数として定義しますが，dirno で始まる名称のシェル変数はパス名を扱わないシェル変数として定義します。

```
#-adsh_conf PATH_CONV_VAR dir*
#-adsh_conf PATH_CONV_NOVAR dirno*
```

7.3.36 PERMISSION_SPOOLJOB_DIR パラメーター（スプールジョブディレクトリのパーミッションを定義する）【UNIX 限定】

形式

```
#-adsh_conf PERMISSION_SPOOLJOB_DIR パーミッション
```

機能

スプールジョブディレクトリのパーミッションを，ジョブ終了時に変更する場合，変更後のパーミッションを定義します。

このパラメーターを指定しない場合，スプールジョブディレクトリのパーミッションは 0700 になります。

オペランド

パーミッション　～＜ 4 桁の 8 進数＞((0000～1777))

パーミッションを指定します。ジョブ終了時に，このオペランドで指定したスプールジョブディレクトリのパーミッションが設定されます。

注意事項

- 1 つの環境ファイルの中でこのパラメーターに対して次の指定をした場合，ジョブは実行されないでエラー終了します。
  - パラメーターを 2 つ以上指定した場合
  - オペランドを 1 つも指定しなかった場合
  - オペランドの数が 2 つよりも多い場合
  - 8 進数以外の数値，またはパーミッションで指定できる値を超える値を指定した場合
  - 4 桁以外の値を指定した場合
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

使用例

- スプールジョブディレクトリのパーミッションを 0750 にします。

```
#-adsh_conf PERMISSION_SPOOLJOB_DIR 0750
```

7.3.37 PERMISSION_SPOOLJOB_FILE パラメーター（スプールジョブディレクトリ下のファイルのパーミッションを定義する）【UNIX 限定】

形式

```
#-adsh_conf PERMISSION_SPOOLJOB_FILE パーミッション
```

機能

ジョブ終了時にスプールジョブディレクトリ下のファイルのパーミッションを変更する場合，変更後のパーミッションを定義します。

このパラメーターを指定すると，スプールジョブディレクトリ下のディレクトリのパーミッションも変更されます。ただし，スプールジョブディレクトリ下のディレクトリの下にあるファイルやディレクトリのパーミッションは変更されません。

このパラメーターを指定しない場合，スプールジョブディレクトリ下に作成されるファイルのパーミッションは次のようになります。

- .DBG ファイル：パーミッションは 666 になります。
- #adsh_spoolfile コマンドで割り当てるファイル：パーミッションは，ファイルを作成するコマンドやプログラムの指定に従います。
- 上記以外のファイル：パーミッションは 600 になります。

オペランド

パーミッション　～＜3 桁の 8 進数＞((000～777))

パーミッションを指定します。スプールジョブディレクトリ下のファイルのパーミッションは，ジョブ終了時に，このオペランドで指定した値に設定されます。

注意事項

- 1 つの環境ファイルの中でこのパラメーターに対して次の指定をした場合，ジョブは実行されないでエラー終了します。
  - パラメーターを 2 つ以上指定した場合
  - オペランドを 1 つも指定しなかった場合

- オペランドの数が 2 つよりも多い場合
- 8 進数以外の数値，またはパーミッションで指定できる値を超える値を指定した場合
- 3 桁以外の値を指定した場合

- adshhk コマンドと adshevtout コマンドでは，スプールジョブディレクトリおよびスプールジョブディレクトリ下のファイルを操作します。パーミッションを変更する際は，これらのコマンドを実行するユーザーの権限を考慮してください。
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを指定すると，スプールジョブディレクトリ下のファイルの setuid ビット，および setgid ビットは削除されます。

使用例

- スプールジョブディレクトリ下のファイルのパーミッションを 744 にします。

```
#-adsh_conf PERMISSION_SPOOLJOB_FILE 744
```

7.3.38 PIPE_CMD_LAST パラメーター（パイプの最終コマンドの実行プロセスを定義する）

形式

UNIX 版

```
#-adsh_conf PIPE_CMD_LAST  {CURRENT|OTHER}
```

Windows 版

```
#-adsh_conf PIPE_CMD_LAST  {CURRENT|OTHER|SEQUENTIAL}
```

機能

パイプラインの最終コマンドをカレントプロセスで実行するかどうかを定義します。

次のようにパイプラインの最終コマンドで変数の内容を更新し，パイプライン以降のコマンドで更新した内容を使用する場合は，CURRENT を指定してください。

ジョブ定義スクリプトの内容

```
typeset -i CNT=0
cat INFILE | while read STR
do
  echo ”$STR”
  let CNT=CNT+1
```

```
done
echo ”Line count is $CNT.”
```

この結果，while 文終了後のシェル変数 CNT には，read コマンドで読み込まれた行数（INFILE の行数）が格納されます。

一方，次のようにパイプラインの最終コマンドで変数の内容を更新するが，パイプライン実行前の変数の内容で，パイプライン終了後に再度利用したい場合は OTHER を指定してください（CBL_SYSUT1，CBL_SYSUT2 は CBLUAPx の共通インターフェース用変数と仮定します）。

ジョブ定義スクリプトの内容

```
CBL_SYSUT1=/file1
CBL_SYSUT2=/file2
CBLUAP1
cat INFILE | while read DIR
do
CBL_SYSUT1=`cmd1 y`
CBL_SYSUT2=`cmd2 y`
CBLUAP2
done
CBLUAP3
```

この場合，while 文終了後のシェル変数 CBL_SYSUT1 には"/file1"，CBL_SYSUT2 には"/file2"が格納されています。

このパラメーターは同一の環境ファイル内で複数回指定することはできません。複数回指定した場合は，エラーメッセージが出力されてジョブは終了します。

オペランド

CURRENT

パイプラインの全コマンドを非同期実行します。パイプラインの最終コマンドが次に示すコマンドの場合，カレントプロセスで動作します。

- シェル標準コマンド
- 代入式
- スクリプト制御文
- シェル拡張コマンド
- スクリプト予約語コマンド
- 関数

パイプラインの最終コマンドが上記以外の場合，別プロセスで動作します。

OTHER

パイプラインの全コマンドを非同期実行します。パイプラインの最終コマンドは別プロセスで動作します。

SEQUENTIAL【Windows 限定】

パイプラインの全コマンドをカレントプロセスで逐次実行します。また，コマンド間のデータの受け渡しに一時ファイルを使用します。

これは JP1/Advanced Shell 11-00 より前の Windows 版と同様の動作です。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- Windows 版で，OTHER または CURRENT を指定した場合，SEQUENTIAL に比べて実行時間が長くなる場合があります。そのため，SEQUENTIAL から OTHER または CURRENT に切り替える際は，ジョブの実行時間を検証してください。

使用例

次に示す入力ファイルとジョブ定義スクリプトを使って，PIPE_CMD_LAST パラメーターを指定した場合の実行結果の違いを示します。

- 例 1

  ジョブ定義スクリプトの内容

```
STR=”abcdefg”
echo ”ABCDEFG” | read STR
echo $STR
```

  標準出力への出力結果（PIPE_CMD_LAST パラメーターに CURRENT または SEQUENTIAL を指定した場合）

```
ABCDEFG
```

  標準出力への出力結果（PIPE_CMD_LAST パラメーターに OTHER を指定した場合）

```
abcdefg
```

- 例 2

  ジョブ定義スクリプトの内容

```
A=1
echo ”Hello World” | A=10
echo $A
```

  標準出力への出力結果（PIPE_CMD_LAST パラメーターに CURRENT または SEQUENTIAL を指定した場合）

```
10
```

  標準出力への出力結果（PIPE_CMD_LAST パラメーターに OTHER を指定した場合）

```
1
```

- 例 3

入力ファイル「INFILE」の内容

```
user1,500,Tomato
user2,1000,Tomato
user3,300,Lettuce
user1,450,Cabbage
user1,250,Orange
```

ジョブ定義スクリプトの内容

```
typeset -i cnt=1
cat INFILE | grep user1 | while read NAME
do
  if [ $cnt -ge 3 ]; then
    break
  fi
  echo "$cnt = $NAME"
  let cnt=cnt+1
done
echo $cnt
```

while から done まではパイプの最終コマンドとして扱われます。

標準出力への出力結果（PIPE_CMD_LAST パラメーターに CURRENT または SEQUENTIAL を指定した場合）

```
1 = user1,500,Tomato
2 = user1,450,Cabbage
3
```

標準出力への出力結果（PIPE_CMD_LAST パラメーターに OTHER を指定した場合）

```
1 = user1,500,Tomato
2 = user1,450,Cabbage
1
```

7.3.39 SPOOL_DIR パラメーター（スプールルートディレクトリのパス名を定義する）

形式

```
#-adsh_conf SPOOL_DIR パス名
```

機能

バッチジョブの実行結果（ジョブ実行ログおよびジョブステップのプログラムが出力したデータファイル）をジョブごとにディレクトリを作成して出力するスプールルートディレクトリのパス名を定義します。

オペランド

パス名

Windows の実行環境の場合　～<パス名>((1～128 バイト))《共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥spool》

Windows の開発環境の場合　～<パス名>((1～128 バイト))《共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥spool》

UNIX の場合　～<パス名>((1～128 バイト))《/var/opt/jp1as/spool》

スプールルートディレクトリのパス名を指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- ユーザー応答機能を使用する場合，次の指定をすると正しく動作しません。
  - ジョブ環境ファイルにこのパラメーターを指定した場合
  - パス名にマルチバイト文字を指定した場合
- このパラメーターに，JP1/Advanced Shell でサポートしないファイルシステム上のディレクトリを指定しないでください。JP1/Advanced Shell でサポートしないファイルシステムの詳細については，「(2)　ファイルシステム」を参照してください。
- パス名に&，(，)，[，]，{，}，^，＝，；，!，'，＋，，，`，～，#，%の記号を含まないでください。これらの記号を含む場合，正常に動作しません。

補足

- 該当するパラメーターで指定するディレクトリを別々に分けることで，同一ホスト内で複数環境を使い分けられます。
- クラスタ運用で待機系のホストに情報を引き継ぎたい場合は，引き継ぐディレクトリをホスト間で共用します。その場合は，少なくともこのパラメーターのディレクトリをホスト間で共用します。

7.3.40　SPOOLJOB_CHILDJOB パラメーター（子孫ジョブのスプールジョブの扱いを定義する）

形式

```
#-adsh_conf SPOOLJOB_CHILDJOB  {MERGE|DELETE}
```

機能

子孫ジョブの終了時に，子孫ジョブのスプールジョブを削除するか，ルートジョブのスプールジョブにマージするかを選択します。

オペランド

MERGE

子孫ジョブの終了時に，子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージします。これによって次のように動作します。

- 子孫ジョブのジョブ実行ログがルートジョブのジョブ実行ログにマージされ，子孫ジョブが終了した順に出力されます。
- ルートジョブの JOBLOG と SCRIPT は，それぞれ子孫ジョブの JOBLOG と SCRIPT をマージした内容で作成されます。
- ルートジョブと子孫ジョブの出力内容が判別できる形式で次の個所へ出力されます。

  通常実行の場合：標準エラー出力（STDERR，**ステップ番号_ステップ名**_STDERR）

  デバッグ実行の場合：端末画面上の標準出力と標準エラー出力
- 子孫ジョブ実行時に#-adsh_spoolfile コマンドで割り当てるプログラム出力データファイルは，ルートジョブのスプールジョブディレクトリに次のファイル名で作成されます。

#-adsh_spoolfile コマンドの実行個所	割り当てるファイル名（Windows では末尾に拡張子「.sysout」が付与される）
子孫ジョブのジョブステップ外の場合	C**子孫ジョブ起動順序通し番号**_0000_**ジョブ名**_**ファイル環境変数定義名通し番号**_**ファイル環境変数定義名**
子孫ジョブのジョブステップ内の場合	C**子孫ジョブ起動順序通し番号**_**ステップ番号**_**ステップ名**_**ファイル環境変数定義名通し番号**_**ファイル環境変数定義名**

同じジョブ定義スクリプトを子孫ジョブとして複数回実行した場合，実行した数だけ SCRIPT を出力します。

スプールジョブディレクトリの作成方法については「3.4.2　ジョブの実行結果をスプールに出力する」を参照してください。ジョブ実行ログの出力形式については，「(3)　子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージした場合」を参照してください。

MERGE を指定した場合，1 個のルートジョブから起動できる子孫ジョブは，子孫ジョブから起動する子孫ジョブを含め，9,999,999 個までとなります。それを超えると子孫ジョブはエラー終了します。ただし，これより先に OS が定めるプロセス数やファイル数などの上限値に達した場合には，OS のエラー処理に従います。

MERGE を指定した場合のジョブ実行ログの出力例については，「3.5.3　ジョブ実行ログの出力例（子孫ジョブのスプールジョブを削除した場合）」を参照してください。

DELETE

子孫ジョブの終了時に，子孫ジョブのスプールジョブを削除します。

子孫ジョブのジョブ実行ログのうち，JOBLOG の内容だけを標準エラー出力へ出力します。

DELETE を指定した場合のジョブ実行ログの出力例については，「3.5.3　ジョブ実行ログの出力例（子孫ジョブのスプールジョブを削除した場合）」を参照してください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- MERGE を指定した場合，子孫ジョブを非同期実行すると，JOBLOG のマージ順序と SCRIPT のマージ順序とが一致しないことがあります。

  例えば，ルートジョブで子孫ジョブ A と子孫ジョブ B を非同期実行した場合に，JOBLOG は「子孫ジョブ B→子孫ジョブ A」の順序でマージされ，SCRIPT は「子孫ジョブ A→子孫ジョブ B」の順序でマージされることがあります。
- MERGE を指定した場合，ジョブ定義スクリプト内のコマンドを非同期実行すると，子孫ジョブの標準出力と標準エラー出力を示すジョブ実行ログの範囲に，非同期実行したコマンドの標準出力や標準エラー出力が混在することがあります。
- MERGE を指定した場合，起動した子孫ジョブで，ジョブ定義スクリプトを解析する前の初期設定処理でエラーが発生すると，マージ処理は実行されません。
- 子孫ジョブでは，ルートジョブの起動時に読み込んだ環境ファイルの SPOOLJOB_CHILDJOB パラメーターの値が指定されたものとして動作します。ルートジョブのジョブ環境ファイルと，子孫ジョブのジョブ環境ファイルとで SPOOLJOB_CHILDJOB パラメーターの値が異なっていても，無視して動作します。
- 文法チェックモードでジョブを実行した場合，マージ処理は実行されません。

使用例

- 子孫ジョブの終了時に，子孫ジョブのスプールジョブをルートジョブのスプールジョブにマージします。

```
#-adsh_conf SPOOLJOB_CHILDJOB MERGE
#-adsh_conf CHILDJOB_SHEBANG /bin/sh
```

7.3.41　SPOOLJOB_CREATE パラメーター（スプールジョブの作成要否を選択する）

形式

```
#-adsh_conf SPOOLJOB_CREATE  {YES|NO}
```

機能

ジョブ定義スクリプトの実行時にスプールジョブを作成するかどうかを選択します。

子孫ジョブに対してはこのパラメーターの指定は無効となり，ルートジョブの指定が引き継がれます。

オペランド

YES

スプールジョブディレクトリを作成します。

NO

スプールジョブディレクトリを作成しません。この場合の動作については，「(a)　スプールジョブ作成抑止機能の使用有無を決定する」を参照してください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- adshexec -r コマンドまたは adshscripttool -r コマンドの実行時は，このパラメーターの指定に関係なく，ジョブ定義スクリプトファイルの一時ファイルは作成されません。この場合，メッセージなどのジョブ定義スクリプトファイルの出力部分には「-r CMDLINE」と出力されます。
- CUI デバッガの使用時は，DBG ファイル「**一時ファイルディレクトリ**/ADSH_DBG_**プロセス ID**_**ジョブ識別子**」が一時的に作成され，デバッグ終了時に削除されます。

7.3.42　TEMP_FILE_DIR パラメーター（一時ファイルディレクトリのパス名を定義する）

形式

```
#-adsh_conf TEMP_FILE_DIR パス名
```

機能

一時ファイルを格納するディレクトリのパス名を定義します。

一時ファイルとはバッチジョブ中で作成され，バッチジョブ終了時にはすべて消去する暫定的に利用するファイルのことです。

オペランド

パス名

Windows の実行環境の場合　～<パス名>((1～128 バイト))《**共有ドキュメントフォルダ**¥Hitachi¥JP1AS¥JP1ASE¥temp》

Windows の開発環境の場合　～<パス名>((1～128 バイト))《**共有ドキュメントフォルダ**¥Hitachi¥JP1AS¥JP1ASD¥temp》

UNIX の場合　～<パス名>((1～512 バイト))《/var/opt/jp1as/temp》

一時ファイルを格納するディレクトリのパス名を指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターに，JP1/Advanced Shell でサポートしないファイルシステム上のディレクトリを指定しないでください。JP1/Advanced Shell でサポートしないファイルシステムの詳細については，「(2)　ファイルシステム」を参照してください。

補足

- 該当するパラメーターで指定するディレクトリを別々に分けることで，同一ホスト内で複数環境を使い分けられます。

7.3.43　TRACE_DIR パラメーター（トレースを出力するディレクトリのパス名を定義する）

形式

```
#-adsh_conf TRACE_DIR パス名
```

機能

トレースを出力するディレクトリのパス名を定義します。

オペランド

パス名

Windows の実行環境の場合　～<パス名>((1～128 バイト))《共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥trace》

Windows の開発環境の場合　～<パス名>((1～128 バイト))《共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥trace》

UNIX の場合　～<パス名>((1～512 バイト))《/opt/jp1as/trace》

トレースを出力するディレクトリのパス名を指定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

- このパラメーターに，JP1/Advanced Shell でサポートしないファイルシステム上のディレクトリを指定しないでください。JP1/Advanced Shell でサポートしないファイルシステムの詳細については，「(2)　ファイルシステム」を参照してください。
- パス名に&，(,)，[,]，{,}，^，＝，;，!，'，＋，,，`，～，#，%の記号を含まないでください。これらの記号を含む場合，正常に動作しません。

補足

- 該当するパラメーターで指定するディレクトリを別々に分けることで，同一ホスト内で複数環境を使い分けられます。

7.3.44　TRACE_FILE_CNT パラメーター（トレース面数を定義する）

形式

```
#-adsh_conf TRACE_FILE_CNT 面数
```

機能

トレースを出力する面数を定義します。

オペランド

面数　～<符号なし整数>((1～64))《4》

トレースを出力する面数を指定します。通常は 4 を指定してください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。ただし，TRACE_DIR パラメーターの定義がシステム環境ファイルの指定と同一の場合，システム環境ファイルと異なる値を指定するとエラーになります（システム環境ファイルの指定がデフォルト値の場合でも，相違があればエラーになります）。

補足

- 複数のユーザーが同じファイルにトレースログを出力している場合には，TRACE_FILE_CNT は，より大きい値を指定したユーザーの指定が有効になります。

  また，環境ファイルで TRACE_FILE_CNT を変更した場合は，既存のトレースファイルの面数の設定値と比較して，より大きい値を指定したユーザーの指定が有効になります。

  トレースファイルの面数を小さくする場合は，トレースフォルダにあるファイルをすべて削除してください。トレースフォルダにあるファイルをすべて削除するときは，同じトレースファイルにトレースを出力しているジョブがないことを確認してから行ってください。

同じファイルにトレースを出力するユーザー同士は，同じ設定値にすることを推奨します。

7.3.45 TRACE_FILE_SIZE パラメーター（トレースファイルサイズを定義する）

形式

```
#-adsh_conf TRACE_FILE_SIZE ファイルサイズ
```

機能

トレースを出力するファイルサイズを定義します。

オペランド

ファイルサイズ　～<符号なし整数>((1～16))《2》

トレースを出力するファイルサイズを MB で指定します。通常は 2 を指定してください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。ただし，TRACE_DIR パラメーターの定義がシステム環境ファイルの指定と同一の場合，システム環境ファイルと異なる値を指定するとエラーになります（システム環境ファイルの指定がデフォルト値の場合でも，相違があればエラーになります）。

補足

- 複数のユーザーが同じファイルにトレースを出力している場合には，TRACE_FILE_SIZE は，より大きい値を指定したユーザーの指定が有効になります。

  また，環境ファイルで TRACE_FILE_SIZE を変更した場合は，既存のファイルサイズの設定値と比較して，より大きい値を指定したユーザーの指定が有効になります。

  ファイルサイズを小さくする場合は，トレースフォルダにあるファイルをすべて削除してください。トレースフォルダにあるファイルをすべて削除するときは，同じトレースファイルにトレースを出力しているジョブがないことを確認してから行ってください。

  同じファイルにトレースを出力するユーザー同士は，同じ設定値にすることを推奨します。

7.3.46　TRACE_LEVEL パラメーター（トレース出力レベルを定義する）

形式

```
#-adsh_conf TRACE_LEVEL トレースレベル
```

機能

トレースを出力するレベルを定義します。

オペランド

トレースレベル　～<符号なし整数>((0，10，20，30))《0》

トレースを出力するレベルを指定します。数値が大きいほど詳細なトレースを出力します。通常は 0 を指定してください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.47　TRAP_ACTION_SIGTERM パラメーター（ジョブコントローラが強制終了要求を受けたときの動作を定義する）

形式

UNIX 版

```
#-adsh_conf TRAP_ACTION_SIGTERM {DISABLE|TERM|CONT|AUTO}
```

Windows 版

```
#-adsh_conf TRAP_ACTION_SIGTERM {DISABLE|TERM}
```

機能

ジョブコントローラが強制終了要求を受けたときに実行する動作を，trap コマンドによって定義できるかどうかを指定します。また，trap コマンドで定義した動作を実行した後のジョブコントローラの動作を指定します。ここでの強制終了要求とは，JP1/AJS - View からの強制終了操作，UNIX の kill コマンドによる SIGTERM シグナル送信，Windows の taskkill コマンドによる強制終了（TerminateProcess などによるプロセス即時終了）などを示します。

ジョブ定義スクリプト実行中に強制終了要求を受けた場合，ジョブコントローラはオペランドの指定に従って動作します。

このパラメーターの指定を省略すると，ジョブコントローラは DISABLE が指定されたと解釈して動作します。

trap コマンドについては，「9.3　シェル標準コマンド」の「trap コマンド（シグナルや強制終了要求を受けたときの動作を設定する）」を参照してください。

オペランド

DISABLE

強制終了要求を受けたときに実行する動作を trap コマンドで定義できません。ジョブコントローラは強制終了要求を受けた場合，後続のコマンドを実行しないでエラー終了します。

TERM

強制終了要求を受けたときに実行する動作を trap コマンドで定義できます。

ジョブコントローラは強制終了要求を受けた場合，trap コマンドで定義した動作を実行した後，後続のコマンドを実行しないでエラー終了します。

ただし，UNIX 版では 2 回目の強制終了要求を受けた場合，ジョブコントローラの後処理を実行しないでジョブが即時終了します。

CONT【UNIX 限定】

強制終了要求を受けたときに実行する動作を trap コマンドで定義できます。

ジョブコントローラは強制終了要求を受けた場合，trap コマンドで定義した動作に従います。また，強制終了要求を複数回受けても処理を継続します。

ただし，このオペランドは JP1/AJS から起動したジョブには指定できません。指定した場合，ジョブコントローラは環境ファイルの解析時に KNAX0474-E メッセージを出力してエラー終了します。

AUTO【UNIX 限定】

ジョブコントローラは，ジョブの起動方法に応じて TERM または CONT が指定されたと解釈します。ジョブの起動方法に関わらず同じ環境ファイルを共通的に使用したい場合は，このオペランドを指定してください。

対象のジョブ	動作
JP1/AJS から起動したジョブ	TERM が指定されたと解釈して動作します。
JP1/AJS 以外から起動したジョブ	CONT が指定されたと解釈して動作します。

なお，次のどれかの条件を満たした場合に，JP1/AJS から起動したジョブとみなします。

1. JP1/Advanced Shell のカスタムジョブから起動したジョブ
2. 環境変数 AJS_BJEX_STOP に TERM を設定した状態で起動したジョブ
3. 1.および 2.から起動した子孫ジョブ

オペランドごとの強制終了要求を受けたときの動作の詳細については「3.11　ジョブを強制終了する」を参照してください。

注意事項

【UNIX，Windows 共通】

- trap コマンドで強制終了要求を受けたときの動作を定義したジョブが強制終了要求を受けると，強制終了要求を受けたときの動作が完了するまでジョブは終了しません。そのため，長時間実行するコマンドや終了しない動作を定義するときはその点について十分に注意してください。
- ジョブ定義スクリプト内で trap コマンドによる動作定義をしても，ジョブ正常終了時の通常の後処理中は有効になりません。この間に強制終了要求を受けた場合，ジョブコントローラは trap コマンドによる動作定義がないものと見なして動作します。
- trap コマンドで定義した動作がエラー終了したり，exit コマンドでジョブを打ち切ると，ジョブは trap コマンドで定義した動作内で最後に実行したコマンドの終了コードで終了します。しかし，このパラメーターに TERM を指定した場合，trap コマンドで定義した動作の終了コードはジョブ，およびジョブステップの終了コードには反映されないで，強制終了要求を受けたときの終了コードで終了します。
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

【UNIX 限定】

- 次のどれかの条件を満たす場合，強制終了要求を受け，子孫プロセスの終了，trap コマンドで定義した動作の実行，一時ファイルの削除などの処理中に 2 回目の強制終了要求を受けると，作成した一時ファイルが削除されないで残るなど，後処理が完了しないことがあります。この場合は手動で一時ファイルを削除するなどの対処をしてください。
  - ・オペランドに DISABLE を指定している。
  - ・オペランドに TERM を指定している。
  - ・オペランドに CONT を指定し，かつ trap コマンドによる動作定義がない。

7.3.48 UMASK_INHERIT パラメーター（ジョブ定義スクリプト実行開始時のファイルモード作成マスクについて定義する）【UNIX 限定】

形式

```
#-adsh_conf UMASK_INHERIT  {YES|NO}
```

機能

ジョブ定義スクリプト実行開始時に，親プロセスのファイルモード作成マスクを継承するかどうかを指定します。

オペランド

YES

ジョブコントローラ起動時のファイルモード作成マスクを継承します。

YES を指定した場合，umask の影響を受けるファイルおよびディレクトリのアクセス権限に影響するため，注意してください。

NO

ジョブコントローラ起動時のファイルモード作成マスクを継承しないで，ファイルモード作成マスクを 0 に変更します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

7.3.49 UNSUPPORT_TEST パラメーター（サポートしていない条件式の実行時の動作を定義する）【Windows 限定】

形式

```
#-adsh_conf UNSUPPORT_TEST  {h|G|L|O|ef}   {ERR|TRUE|FALSE}
```

機能

Windows 環境の JP1/Advanced Shell でサポートしていない条件式を判定した場合の動作を指定します。

- 演算子「-G」を使った条件式
- 演算子「-O」を使った条件式

なお，次の条件式は JP1/Advanced Shell ではサポートしていますが，11-00 より前のバージョンではサポートしていなかったため，オペランドとして提供しています。

- 演算子「-h」を使った条件式
- 演算子「-L」を使った条件式
- 演算子「-ef」を使った条件式

このパラメーターで動作を指定していないときの動作は，条件式によって異なります。

-G，-O	ERR を指定したときと同じ動作となります。
-h，-L，-ef	条件判定をして結果を返します。ただし，環境変数 ADSH_LINK_SUPPORT に L0 を指定した場合は，ERR を指定したときと同じ動作となります。

条件式については，「5.2.2　条件式」を参照してください。

オペランド

{h|G|L|O|ef}

判定した場合の動作を定義する条件式を指定します。

- h
  演算子「-h」を使った条件式を示します。
- G
  演算子「-G」を使った条件式を示します。
- L
  演算子「-L」を使った条件式を示します。
- O
  演算子「-O」を使った条件式を示します。
- ef
  演算子「-ef」を使った条件式を示します。

{ERR|TRUE|FALSE}

条件式を判定した場合の動作を指定します。

- ERR
  エラーメッセージを出力し，ジョブを終了します。
- TRUE
  インフォメーションメッセージを出力し，常に正しいものと判定します。
- FALSE
  インフォメーションメッセージを出力し，常に誤りと判定します。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，指定した条件単位で，ジョブ環境ファイルでの定義が有効になります。

7.3.50　USERREPLY_DEBUG_DESTINATION パラメーター（デバッグ実行時の事象通知メッセージと応答要求メッセージの入出力先を指定する）

形式

```
#-adsh_conf USERREPLY_DEBUG_DESTINATION  [JP1EVENT|CONSOLE]
```

機能

ユーザー応答機能で，adshecho コマンドまたは adshread コマンドを使用したジョブ定義スクリプトをデバッグする場合に，事象通知メッセージおよび応答要求メッセージの入出力先を指定します。

オペランド

JP1EVENT

事象通知メッセージおよび応答要求メッセージを JP1 イベントとして発行します。

CONSOLE

事象通知メッセージおよび応答要求メッセージの入出力先を標準入出力にします。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- デバッグ実行時以外にこのパラメーターに CONSOLE を指定して実行すると，指定は無視されます。この場合，事象通知メッセージおよび応答要求メッセージは JP1 イベントとして発行されます。

7.3.51 USERREPLY_JP1EVENT_INTERVAL パラメーター（JP1 イベントの最小発行間隔を指定する）

形式

```
#-adsh_conf USERREPLY_JP1EVENT_INTERVAL 最小発行間隔
```

機能

ユーザー応答機能で発行する JP1 イベントの最小発行間隔を設定します。

前回の JP1 イベントが発行された時刻から，ここで定義した時間が経過するまで，次の JP1 イベントの発行を待つことで，流量制御を行います。

オペランド

最小発行間隔　～<符号なし整数>((100～100000))《500》

前回の JP1 イベントの発行時刻からの待ち時間をミリ秒単位で指定します。ただし，JP1/IM - Manager に過大な負荷が掛かることを避けるため，通常は 500 以上の値を指定してください。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行した場合，このパラメーターの指定は無視されます。

7.3.52 USERREPLY_WAIT_MAXCOUNT パラメーター（物理ホストまたは論理ホストごとに応答要求メッセージの最大同時出力数を指定する）

形式

```
#-adsh_conf USERREPLY_WAIT_MAXCOUNT 応答要求メッセージの最大同時出力数
```

機能

ユーザー応答機能を使用する場合，物理ホストまたは論理ホストごとに，応答要求メッセージの同時出力数の上限を指定します。

オペランド

応答要求メッセージの最大同時出力数　～<符号なし整数>((1～100))《5》

物理ホストまたは論理ホストごとの応答要求メッセージの同時出力数の上限を指定します。これによって，物理ホストまたは論理ホストごとに同時に存在できる応答要求メッセージの数を制限します。

JP1/IM - View に滞留できる応答待ちイベントは 2,000 件までです。そのため，応答待ちイベントを送信するホストでは，次の計算式を満たす値を指定してください。

```
JP1イベントの送信先（HOSTNAME_JP1IM_MANAGERパラメーターの指定）が同一である，応答待ちイベントを送信する全ホストの応答要求メッセージの最大同時出力数の合計値
＋他製品の応答待ちイベント滞留数の合計値
 ＜ 2000
```

注意事項

- このパラメーターはジョブ環境ファイルに指定しないでください。このパラメーターをシステム環境ファイルとジョブ環境ファイルの両方に指定した場合，ジョブ環境ファイルの指定は無視されます。
- 指定値を超える数の応答待ちイベントを出力しようとすると，共有メモリに空きが生じるまで待ち状態になります。共有メモリの空きは 3 分ごとに 3 回まで確認され，それでも空きがなかった場合はエラーとなります。そのため，複数のジョブから同時に応答要求メッセージが出力される場合は，指定値を超えないように値を設定してください。
- ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行した場合，このパラメーターの指定は無視されます。

7.3.53 VAR_ENV_NAME_LOWERCASE パラメーター（環境変数名の小文字の使用可否を指定する）【Windows 限定】

形式

```
VAR_ENV_NAME_LOWERCASE  {ENABLE|DISABLE}
```

機能

小文字を含む環境変数名を有効にするかどうかを指定します。

オペランド

ENABLE

小文字を含む環境変数名を有効とします。

なお，同じスペルで大文字・小文字だけが異なる環境変数名は，Windows の環境変数としては区別されませんが，シェル変数としては別の変数と認識されます。これによる混乱を避けるため，同じスペルのシェル変数は大文字・小文字を一致させることを推奨します。

DISABLE

小文字を含む環境変数名を無効とします。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。

使用例

シェル変数名「SAMPLE01」「sample01」を定義したジョブ定義スクリプトと，バッチファイルを次のように定義した場合について，実行例を示します。

ジョブ定義スクリプト「envsample.ash」の内容

```
export SAMPLE01=large
export sample01=small
.¥¥envsample.bat | "${ADSH_DIR_CMD}grep" -i "SAMPLE01" 1>&2
echo "*** Shell variables ***" >&2
echo "SAMPLE01=$SAMPLE01" >&2
echo "sample01=$sample01" >&2
```

バッチファイル「envsample.bat」の内容

```
set
```

- VAR_ENV_NAME_LOWERCASE パラメーターに DISABLE（小文字の環境変数名を無効とする）を指定した場合

環境ファイルの内容

```
#-adsh_conf VAR_ENV_NAME_LOWERCASE DISABLE
#-adsh_conf OUTPUT_MODE_ROOT  SIMPLE
#-adsh_conf OUTPUT_MODE_CHILD SIMPLE
```

実行結果

```
D:¥home>"C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin¥adshexec" envsample.ash
KNAX6712-E 現在のプラットフォームでは，名前がすべて大文字でない変数（"sample01"）はエクスポートできません。 filename="D:¥home¥envsample.ash" line=2
KNAX6521-E コマンド（export，行番号=2）がエラー終了しました。rc=1 E-Time=0.015s C-Time=0.000s
KNAX0101-E ADSH001150 ジョブを実行中にエラーが発生しました。

D:¥home>
```

この例では，「SAMPLE01」のあとに実行した「sample01」のエクスポートに失敗しています。

- VAR_ENV_NAME_LOWERCASE パラメーターに ENABLE（小文字の環境変数名を有効とする）を指定した場合

環境ファイルの内容

```
#-adsh_conf VAR_ENV_NAME_LOWERCASE ENABLE
#-adsh_conf OUTPUT_MODE_ROOT  SIMPLE
#-adsh_conf OUTPUT_MODE_CHILD SIMPLE
```

実行結果

```
D:¥home>"C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥bin¥adshexec" envsample.ash
sample01=small
*** Shell variables ***
SAMPLE01=large
sample01=small

D:¥home>
```

この例では，小文字のシェル変数をエクスポートできるようになり，最後にエクスポートしたシェル変数「sample01」が環境変数にエクスポートされます。また，シェル変数「SAMPLE01」「sample01」は別々の値を持ちます。

7.3.54 VAR_SHELL_FUNCINFO パラメーター（関数情報配列の使用有無を選択する）

形式

```
#-adsh_conf VAR_SHELL_FUNCINFO  {TYPE_A|TYPE_B|NONE}
```

機能

関数情報配列を使用するかどうかを定義します。関数情報配列は，adshexec コマンドが実行している関数の情報を格納する一次元配列です。

関数情報配列には次の 3 種類があります。関数情報配列については，「5.5.3　関数情報配列」を参照してください。

- 呼び出し関数名称配列
- 関数呼び出し行番号配列
- 関数定義スクリプトファイル名配列

オペランド

TYPE_A

関数情報配列を次に示す「ADSH_」で始まる名称で作成します。

配列の種類	配列名
呼び出し関数名称配列	ADSH_FUNCNAME
関数呼び出し行番号配列	ADSH_LINENO
関数定義スクリプトファイル名配列	ADSH_SOURCE

TYPE_B

関数情報配列を次に示す bash と同じ配列名で作成します。

配列の種類	配列名
呼び出し関数名称配列	FUNCNAME
関数呼び出し行番号配列	BASH_LINENO
関数定義スクリプトファイル名配列	BASH_SOURCE

NONE

関数情報配列を作成しません。

そのため，TYPE_A や TYPE_B の指定時に作成される配列を通常の配列として使用できます。

注意事項

- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，エラーとなります。
- 関数情報配列は読み込み専用です。読み込み専用の解除，値の更新，unset による無効化はできません。
- 関数情報配列は関数内ローカル変数として定義することはできません。関数内での属性の変更や，関数内ローカル変数の定義はしないでください。

- 関数情報配列は#-adsh_step_start コマンドの stepVar 属性に指定することはできません。
- ルートジョブと子孫ジョブでパラメーターの指定値が異なる場合，作成される配列はジョブ起動時のパラメーターの指定値に従います。

使用例

VAR_SHELL_FUNCINFO パラメーターに TYPE_A を指定した環境で次のジョブ定義スクリプトを実行した場合を例に説明します。

/home/user/script/adsh_func.ash

```
0001 : func1(){                ← 関数func1の定義
0002 :  function func1_2 {     ← 関数func1_2の定義
0003 :    echo "in func1_2"
0004 :    func2                ← 関数func1_2から関数func2を呼び出す
0005 :  }
0006 :  echo "in func1"
0007 :  . ./func2.ash     ← 関数func2を定義した外部スクリプトを.(ドット)で読み込む
0008 :  func1_2            ← 関数func1から関数func1_2を呼び出す
0009 : }
0010 :
0011 : echo "main_script start"
0012 : export FPATH=`pwd`     ← シェル変数FPATHにカレントの作業ディレクトリを格納
0013 : #-adsh_script ./func3.ash ← 関数func3を定義した外部スクリプトを#-adsh_scriptで読
み込む
0014 : autoload func4         ← 関数func4のオートロード機能を有効にする
0015 :
0016 : func1                  ← 関数func1を呼び出す
0017 : echo "main_script end"
```

/home/user/script/func2.ash

```
0001 : func2(){
0002 :   echo "in func2"
0003 :   func3
0004 : }
```

/home/user/script/func3.ash

```
0001 : func3(){
0002 :   echo "in func3"
0003 :   func4
0004 : }
```

/home/user/script/func4

```
0001 : func4(){
0002 :   echo "in func4"
0003 :   cnt=0
0004 :   for cnt in 0 1 2 3 4 5
0005 :   do
0006 :     echo "ADSH_FUNCNAME[$cnt] = ${ADSH_FUNCNAME[$cnt]}"
0007 :     echo "ADSH_LINENO[$cnt]   = ${ADSH_LINENO[$cnt]}"
0008 :     echo "ADSH_SOURCE[$cnt]   = ${ADSH_SOURCE[$cnt]}"
```

```
0009 :   done
0010 : }
```

echo コマンドの行で各関数のスタック情報が出力されます。

関数 func4()実行時点の関数情報配列の値を次に示します。

要素番号	配列名		
	ADSH_FUNCNAME	ADSH_LINENO	ADSH_SOURCE
0	func4	3	/home/user/script/func4
1	func3	3	/home/user/script/func3.ash
2	func2	4	/home/user/script/func2.ash
3	func1_2	8	/home/user/script/adsh_func.ash
4	func1	16	/home/user/script/adsh_func.ash
5	main	0	/home/user/script/adsh_func.ash

なお，VAR_SHELL_FUNCINFO パラメーターに TYPE_B を指定した環境で実行する場合は，ジョブ定義スクリプト「/home/user/script/func4」の内容を次のように変更してください。

```
0001 : func4(){
0002 :   echo "in func4"
0003 :   cnt=0
0004 :   for cnt in 0 1 2 3 4 5
0005 :   do
0006 :     echo "FUNCNAME[$cnt]      = ${FUNCNAME[$cnt]}"
0007 :     echo "BASH_LINENO[$cnt]   = ${BASH_LINENO[$cnt]}"
0008 :     echo "BASH_SOURCE[$cnt]   = ${BASH_SOURCE[$cnt]}"
0009 :   done
0010 : }
```

7.3.55 VAR_SHELL_GETLENGTH パラメーター（${#variable}書式で置換される変数値の長さの単位を定義する）

形式

```
VAR_SHELL_GETLENGTH  {BYTE|CHARACTER}
```

機能

${#variable}書式で置換される変数値の長さの単位を定義します。

オペランド

BYTE

${#variable}書式の variable に格納されている値の長さをバイト数で置換します。

CHARACTER

${#variable}書式の variable に格納されている値の長さを文字数で置換します。

使用例

文字列「abcdef 英字」が設定されている変数 CVAL の値の長さを求めるため,「echo ${#CVAL}」を実行した例を次に示します。

- VAR_SHELL_GETLENGTH パラメーターに BYTE（バイト数で置換）を設定するか, VAR_SHELL_GETLENGTH パラメーターの指定を省略した場合

  マルチバイト文字は実行環境によって長さが異なります。Linux の UTF-8 環境で実行した場合の例を次に示します。

  ```
  CVAL=abcdef英字
  echo ${#CVAL}
  ```

  「abcdef 英字」を 12 バイトと解釈し,「12」が標準出力に出力されます。

- VAR_SHELL_GETLENGTH パラメーターに CHARACTER（文字数で置換）を設定した場合

  ```
  CVAL=abcdef英字
  echo ${#CVAL}
  ```

  「abcdef 英字」は 8 文字のため,「8」が標準出力に出力されます。

7.4　条件パラメーター

論理ホストまたは物理ホストだけで有効となる環境設定パラメーターおよび export パラメーターを設定するには，前後の行を条件パラメーターで囲んで設定します。条件パラメーターについて説明します。

7.4.1　lhost_start パラメーター，lhost_end パラメーター（論理ホストだけで有効なパラメーターを定義する）

形式

```
#-adsh_conf lhost_start 論理ホスト名
指定した論理ホストだけで有効とする環境設定パラメーターまたはexportパラメーター
                        :
#-adsh_conf lhost_end
```

機能

特定の論理ホストだけで有効とする環境設定パラメーターまたは export パラメーターがある場合，lhost_start パラメーターと lhost_end パラメーターで囲んで定義します。lhost_start パラメーターと lhost_end パラメーターは対になるよう定義してください。

- lhost_start パラメーター

  指定した論理ホストだけで有効とする環境設定パラメーターまたは export パラメーターの指定を開始します。対象となる論理ホスト名もあわせて定義します。

- lhost_end パラメーター

  指定した論理ホストだけで有効とする環境設定パラメーターまたは export パラメーターの指定を終了します。

同じ論理ホストに対して条件パラメーターを複数定義してもエラーにはなりません。それぞれに指定したパラメーター群が有効になります。

オペランド

論理ホスト名　～<任意文字列>((1～255 バイト))

論理ホスト名を定義します。同一論理ホストに対する定義が複数回あってもエラーにはなりません。それぞれに指定したパラメーター群が有効になります。

Windows の場合，196 バイトを超える論理ホスト名は指定できません。また，論理ホスト名が 63 バイトを超えると動作しない場合があるため，63 バイト以下で指定することをお勧めします。

7.4.2 phost_start パラメーター，phost_end パラメーター（物理ホストだけで有効なパラメーターを定義する）

形式

```
#-adsh_conf phost_start
物理ホストだけで有効とする環境設定パラメーターまたはexportパラメーター
                    :
#-adsh_conf phost_end
```

機能

物理ホストだけで有効とする環境設定パラメーターまたは export パラメーターがある場合，phost_start パラメーターと phost_end パラメーターで囲んで定義します。phost_start パラメーターと phost_end パラメーターは対になるよう定義してください。

- phost_start パラメーター
  物理ホストだけで有効とする環境設定パラメーターまたは export パラメーターの指定を開始する行の前の行に定義します。
- phost_end パラメーター
  物理ホストだけで有効とする環境設定パラメーターまたは export パラメーターの指定を終了する行の後ろの行に定義します。

物理ホストの定義が複数回あってもエラーにはなりません。それぞれに指定したパラメーター群が有効になります。

8 運用時に使用するコマンド

この章では，JP1/Advanced Shell のシェル運用コマンドおよび UNIX 互換コマンドについて説明します。

8.1　コマンドの記述形式

シェル運用コマンドと UNIX 互換コマンドの記述形式を次に示します。

△$_0$**コマンド名**［△$_1$**オプション**］…［△$_1$**オプション**］［△$_1$**オペランド**］

コマンド名の後ろは，最初にオプションを指定し，次にオペランドを指定します。オペランドとは，オプション名とオプション値のほかにコマンドに指定できる引数のことです。オプションの前にオペランドを指定した場合は，指定内容をすべてオペランドとして処理します。

- オプションを複数指定する場合，指定順序は任意です。
- 不当なオプション，または指定できる範囲外のオプション値を指定した場合，エラーとなります。
- オプション名にマルチバイト文字は使用できません。

コマンドでのファイルのパス名の指定については，「8.1.3　ファイルのパス名」を参照してください。

端末から標準入力へキーボード入力をするコマンドを実行する場合，入力を完了するには次の操作をしてください。

EOF まで入力する場合

Windows では Enter を入力後に Ctrl+Z を入力して，さらに Enter を入力します。

UNIX では Ctrl+D を入力します。

1 行入力する場合

Enter を入力します。

8.1.1　シェル運用コマンドと UNIX 互換コマンド（スクリプト形式）【Windows 限定】のコマンドの記述規則

シェル運用コマンドと UNIX 互換コマンド（スクリプト形式）【Windows 限定】の指定規則を次に示します。

- オプション値のないオプションは連続して指定できます（例：「-a -b -c」と「-abc」は同じです）。その場合，最後のオプションにはオプション値を指定できます（例：「-abc xyz」の「xyz」は，オプション-c の値となります）。

8.1.2　UNIX 互換コマンドの記述規則

スクリプト形式を除く UNIX 互換コマンドには，次の記述規則があります。

- オプションは，ショートオプションとロングオプションに分類されます。

- 連続する 2 個のハイフン（-）だけを指定すると，オプションの指定の終わりを示します。連続する 2 個のハイフン（-）以降に指定した文字列（オプションの指定も含む）は，すべてオペランドとして処理されます。

ショートオプション，ロングオプション，およびコマンドごとの記述規則を次に説明します。

(1)　ショートオプションの指定形式

ショートオプションは，1 個のハイフン（-）の後ろに規定の文字が 1 つ続くオプションです。

ショートオプションの指定形式を次に示します。オプション値を省略できるかどうかは，オプションによって異なります。

```
-ショートオプション名［△0オプション値］
```

ショートオプションを指定する場合の規則を次に示します。

- オプション値のないオプション名は連続して指定できます（例：「-a -b -c」と「-abc」は同じです）。その場合，最後のオプションにはオプション値を指定できます（例：「-abc xyz」の「xyz」は，オプション-c の値となります）。
- オプションによっては，オプション値の前にスペースを指定できません。このオプションでスペースを指定すると，オプション値の部分がオペランドと見なされます。

  この章のコマンド説明の「形式」で，オプション値の前にスペースがないオプションは，スペースを指定できないことを示します。

(2)　ロングオプションの指定形式

ロングオプションは，連続する 2 個のハイフン（-）の後ろに規定の文字列が続くオプションです。

ロングオプションの指定形式を次に示します。オプション値を省略できるかどうかはオプションによって異なります。

```
--ロングオプション名［=オプション値］
```

オプション値のあるオプションは次の形式でも指定できます。ただし，オプション値を省略できるオプションの場合は，ロングオプション名とオプション値の区切りは=（イコール）でなければなりません。=（イコール）でなくスペースを指定すると，オプション値の部分がオペランドと見なされます。

```
--ロングオプション名△1オプション値
```

ロングオプション名とオプション値は短縮して指定できません。例えば，cut コマンドの--characters オプションを「--char リスト」と指定したり，ls コマンドの--format オプションで表示形式 long を「--format=l」と指定したりすると，エラーになります。

重要

ショートオプションに付随するロングオプションがある場合，このマニュアルではコマンド説明の「引数」の説明個所以外はショートオプションだけを記載しています。なお，付随するショートオプションがない場合はロングオプションを記述しています。コマンド説明の「形式」でのロングオプション名の記述位置は，他オプションとの関連がない場合は，すべてのショートオプション名の後ろに記述しています。

例えば，diff コマンドの-y オプションに付随するロングオプション「--side-by-side」は，コマンド説明の「形式」には記述していません。一方，ロングオプション「--suppress-common-lines」にはショートオプションが存在しないため，「形式」に記述しています。

(3) 各コマンドの注意事項

- cp，cut，date，diff，gunzip，gzip，ln，ls，expand，mv，stat コマンドは，オプションとオペランドの指定順序は任意です。オペランドの後ろにオプションを指定しても，そのオプションはオプションとして処理されます。

  次のどちらかの環境変数を設定した場合，オペランドはコマンドラインにすべてのオプションを指定したあとに指定する必要があります。オペランドの後ろにオプションを指定しても，オペランドと見なされます。これらの環境変数については，「2.5　環境変数を設定する」を参照してください。

  - 環境変数POSIXLY_CORRECT
  - 環境変数ADSH_CMD_ARGORDER=seq

- awk，find，getopt，tr，xargs コマンドでオプションとオペランドを指定する場合，すべてのオプションを指定したあとにオペランドを指定する必要があります。

- 次のコマンドでオプションとオペランドを指定する場合，Linux ではオプションとオペランドの指定順序は任意ですが，AIX，HP-UX，Solaris，Windows では，オプションを指定したあとにオペランドを指定する必要があります。

  cat，cmp，egrep，grep，head，mkdir，paste，rm，sed，sort，split，tail，touch，uniq，wc，which

  なお，Linux で環境変数POSIXLY_CORRECT を設定することで，AIX，HP-UX，Solaris，Windows と同じ動作にすることができます。

  環境変数POSIXLY_CORRECT については，「2.5　環境変数を設定する」を参照してください。

8.1.3　ファイルのパス名

ファイルは，絶対パス名と相対パス名で指定できます。

コマンドによっては，相対パス名を絶対パス名に変換して処理しています。このため，コマンドで指定するファイルのパス名の長さを，絶対パス名の長さで認識している場合があります。

コマンドで指定したファイルのパス名に対する絶対パス名の例を次に示します。

(1)　ファイル名の指定が絶対パス名の場合

指定されたファイル名が絶対パス名です。

例

```
/dir1/test1.asc
```

(2)　ファイル名の指定が相対パス名の場合

コマンドを実行したときのカレントディレクトリと，指定されたファイルのパス名の連結が絶対パス名となります。

例

```
コマンドを実行したときのカレントディレクトリ：/home/user1
ファイルのパス名：dir2/test2.asc
ファイルの絶対パス名：/home/user1/dir2/test2.asc
```

8.2　コマンドの一覧

8.2.1　シェル運用コマンドの一覧

シェル運用コマンドは，シェルまたはコマンドプロンプトから実行できるコマンドです。シェル運用コマンドには，adshexec コマンド（バッチジョブを実行するコマンド）などがあります。

JP1/Advanced Shell のシェル運用コマンドを次の表に示します。

表 8-1　シェル運用コマンド

コマンド名	機能	コマンドの格納場所
adshappagent 【Windows 実行環境限定】	アプリケーション実行エージェントを起動し，adshappexec コマンドの実行を監視して，adshappexec コマンドで指定した実行アプリケーションを起動します。	*インストール先フォルダ*¥JP1ASE¥bin
adshappexec 【Windows 実行環境限定】	GUI アプリケーションを実行します。	*インストール先フォルダ*¥JP1ASE¥bin
adshchmsg	障害発生時に，応答要求メッセージに対して手動で応答する場合に使用するコマンドです。	【Windows の実行環境の場合】 *インストール先フォルダ*¥JP1ASE¥bin 【Windows の開発環境の場合】 *インストール先フォルダ*¥JP1ASD¥bin 【UNIX の場合】 /opt/jp1as/sbin
adshcvmerg	マージする 2 つのasc ファイルを入力としてカバレージ情報をマージし，指定したパスのファイルに出力します。	【Windows の実行環境の場合】 *インストール先フォルダ*¥JP1ASE¥bin 【Windows の開発環境の場合】 *インストール先フォルダ*¥JP1ASD¥bin 【UNIX の場合】 /opt/jp1as/bin
adshcvshow	カバレージ情報ファイルを入力として読み込み，カバレージ情報を表示します。	
adshevtout※	ジョブ定義スクリプトの稼働実績情報を出力します。	
adshexec※	ジョブ定義スクリプトファイルを入力として読み込み，ジョブ定義スクリプトの内容に従ってバッチジョブを実行する，ジョブコントローラと呼ばれるプロセスを起動します。	
adshfile	ジョブステップ終了時またはジョブ終了時に，後処理をするファイルを割り当てます。	
adshhk	スプールルートディレクトリ名と日数を指定して，スプールジョブを削除します。	
adshjava	Java アプリケーションクラス名で指定された Java のバッチアプリケーションを uCosminexus Application Server で実行します。	

コマンド名	機能	コマンドの格納場所
【Windows，Linux，AIX，HP-UX 限定】	Java アプリケーションクラス名で指定された Java のバッチアプリケーションを uCosminexus Application Server で実行します。	【Windows の実行環境の場合】 *インストール先フォルダ*¥JP1ASE¥bin 【Windows の開発環境の場合】 *インストール先フォルダ*¥JP1ASD¥bin 【UNIX の場合】 /opt/jp1as/bin
adshlsmsg	障害発生時に，応答要求メッセージの一覧を表示するコマンドです。	【Windows の実行環境の場合】 *インストール先フォルダ*¥JP1ASE¥bin 【Windows の開発環境の場合】 *インストール先フォルダ*¥JP1ASD¥bin 【UNIX の場合】 /opt/jp1as/sbin
adshmdctl【UNIX 限定】	ユーザー応答機能のための共有メモリを管理するデーモンを起動および停止するコマンドです。	/opt/jp1as/sbin
adshmsvcd【Windows 開発環境限定】	ユーザー応答機能のための共有メモリを管理するサービスプログラムです。開発環境で使用できます。	*インストール先フォルダ*¥JP1ASD¥bin
adshmsvce【Windows 実行環境限定】	ユーザー応答機能のための共有メモリを管理するサービスプログラムです。実行環境で使用できます。	*インストール先フォルダ*¥JP1ASE¥bin

注※

Windows の実行環境および UNIX でだけ使用できます。

8.2.2　UNIX 互換コマンドの一覧

UNIX 互換コマンドには，実行ファイル形式で提供するコマンドとスクリプト形式で提供するコマンドがあります。

- 実行ファイル形式で提供するコマンド

  Windows および UNIX で共通の JP1/Advanced Shell のコマンドが使用できます。

  コマンドの指定方法については「8.4　UNIX 互換コマンド」を参照してください。

- スクリプト形式で提供するコマンド【Windows 限定】

UNIX の機能に依存するコマンドを Windows で実現するため，Windows 提供の機能を使用して UNIX の OS 標準のコマンドの一部機能を実現したスクリプト形式のコマンドを使用できます。

コマンドの指定方法については「8.5　UNIX 互換コマンド（スクリプト形式）【Windows 限定】」を参照してください。

(1)　実行ファイル形式で提供するコマンド

UNIX 互換コマンドのうち，実行ファイル形式で提供するコマンドは，ジョブ定義スクリプト中で実行できます。また，Windows のコマンドプロンプトおよび UNIX のシェルからも実行できます。

UNIX 互換コマンドのうち，実行ファイル形式で提供するコマンドの格納場所は次のとおりです。

- Windows の実行環境の場合
  インストール先フォルダ¥JP1ASE¥cmd
- Windows の開発環境の場合
  インストール先フォルダ¥JP1ASD¥cmd
- UNIX の場合
  /opt/jp1as/cmd

UNIX 互換コマンドのうち，実行ファイル形式で提供するコマンドには，ファイルシステムなど OS 差異の大きい制御に関して制限事項があります。また，Windows 限定でオーナーやグループ，アクセス権限などに制限事項があります。

使用できる実行ファイル形式の UNIX 互換コマンドと制限事項を次の表に示します。

表 8-2　UNIX 互換コマンド（実行ファイル形式）

コマンド名	機能概要	制限事項※
awk	テキストの加工やパターン処理をします。	• Windows の場合，system 関数で実行するコマンドの引数にワイルドカードを含むファイル名やディレクトリ名を指定しても，ワイルドカードは展開されません。 • Windows の場合，getline 関数，print 関数およびprintf 関数でパイプによって接続するコマンドの引数にワイルドカードを含むファイル名やディレクトリ名を指定しても，ワイルドカードは展開されません。
basename	パス名からファイル名部分の文字列を取り出し標準出力に出力します。	制限事項はありません。
cat	ファイルの内容を標準出力に出力します。	制限事項はありません。
cmp	バイナリファイルの内容を比較します。	制限事項はありません。
cp	ファイルまたはディレクトリをコピーします。	Windows の場合，-p オプションで保持されるのはコピー元ファイルの更新日時およびファイルアクセス日時だけとなります。ディレクトリの情報は保持しません。
cut	各行の選択範囲を標準出力に表示します。	制限事項はありません。

コマンド名	機能概要	制限事項※
date	システムの日付と時刻を表示します。設定はできません。	-a オプション（時刻設定）は使用できません。
diff	2 つのファイルを比較します。	制限事項はありません。
dirname	ファイルパス名規則に従った文字列からファイル名を取り除いた残りのディレクトリパス名を取り出し，標準出力に出力します。	制限事項はありません。
egrep	ファイル内の文字を検索します。指定されたパターンは拡張された正規表現として扱われます。 grep コマンドに-E オプションを指定した場合と同じ動作です。	制限事項はありません。
expand	タブストップでそろえられている行をタブ文字からスペース文字に置き換えて，標準出力に出力します。	制限事項はありません。
expr	式を評価します。	制限事項はありません。
find	ディレクトリ内のファイルを検索します。	制限事項はありません。
getopt	シェルスクリプトで容易に構文解析できるよう，コマンドラインのオプションを分解します。	制限事項はありません。
grep	ファイル内の文字を検索します。	制限事項はありません。
gunzip	圧縮されたファイルを伸長します。	Windows の場合，圧縮されたファイルの所有者，ACL およびファイル属性は，伸長したファイルには引き継がれません。
gzip	ファイルを圧縮，または圧縮されたファイルを伸長します。	• Windows の場合，圧縮するファイルの所有者，ACL およびファイル属性は，圧縮したファイルには引き継がれません。 • Windows の場合，圧縮されたファイルの所有者，ACL およびファイル属性は，伸長したファイルには引き継がれません。
head	ファイルの最初の部分を表示します。	制限事項はありません。
hostname	ホスト名を表示します。設定はできません。	制限事項はありません。
ln	ファイル，またはディレクトリへのリンクファイルを作成します。	UNIX の場合，次に対するハードリンクは作成できません。 • ディレクトリ • 存在しないファイル • 異なるファイルシステムのファイル Windows の場合，次に対するハードリンクは作成できません。 • ディレクトリ • ディレクトリへのシンボリックリンク • 存在しないファイル • 異なるドライブレターのファイル

コマンド名	機能概要	制限事項※
ln	ファイル，またはディレクトリへのリンクファイルを作成します。	Windows の場合，NTFS 以外のファイルシステムにリンクファイルを作成できません。 Windows の場合，引数ターゲット，またはターゲットディレクトリには UNC 形式を指定できません。
ls	ファイルまたはディレクトリの内容を表示します。	• Windows の場合，-l オプションでグループまたはファイルのオーナー以外のアクセス権を表示できません。 • Windows の場合，TZ 環境変数は表示に影響しません。コントロールパネルの［日付と時刻のプロパティ］ダイアログボックスで設定したタイムゾーンが使用されます。
mkdir	ディレクトリを作成します。	Windows の場合，-m オプションは無視されるため，モード設定はできません。
mv	ファイルまたはディレクトリを移動します。ファイル名またはディレクトリ名も変更できます。	• Windows の場合，オーバーライド時にオーナーのアクセス権以外は表示できません。 • Windows の場合，cp コマンド相当の処理をするケースで，オーナーの変更をサポートしません。オーナー/グループ/モードは保持されません。
paste	複数のファイルを行単位に連結して標準出力に出力します。	制限事項はありません。
printf	値や文字列を書式に従って変換し，標準出力に出力します。	制限事項はありません。
rm	ファイルまたはディレクトリを削除します。	Windows の場合，オーバーライド時にオーナーのアクセス権以外は表示できません。
rmdir	空のディレクトリを削除します。	制限事項はありません。
sed	テキスト中の文字列を置換します。	制限事項はありません。
sleep	指定された時間だけ停止します。	制限事項はありません。
sort	テキストファイルをソートします。	制限事項はありません。
split	ファイルを分割します。	制限事項はありません。
stat	ファイルまたはディレクトリの状態を標準出力に出力します。	• Windows の場合，ファイルの所有者以外のパーミッションを表示できません。 • Windows の場合，ファイルのブロック数，ブロックサイズの情報は常に0 を表示します。 • Windows の場合，デバイス番号はドライブ番号を表示します。 • Windows の場合，所有者のユーザー ID とグループ ID は常に 0 を表示します。 • Windows の場合，所有者のグループ名は常に「. . . 」を表示します。 • Windows の場合，i ノード番号は常に0 を表示します。 • Windows の場合，ディレクトリの合計サイズは常に0 を表示します。

コマンド名	機能概要	制限事項※
stat	ファイルまたはディレクトリの状態を標準出力に出力します。	• Windows の場合，メジャーデバイス番号とマイナーデバイス番号は常に0 を表示します。 • Windows の場合，ファイルの最終アクセス日時とファイル情報の最終変更日時はファイルの最終修正日時と同じ日時を表示します。 • Windows の場合，通常ファイル，ディレクトリ，およびシンボリックリンク以外のファイル種別を表示できません。
tail	ファイルの最後の部分を表示します。	制限事項はありません。
tar	ファイルまたはディレクトリをアーカイブに格納，およびアーカイブから抽出，表示を行います。	• Windows の場合，NTFS 以外のファイルシステムにリンクファイルは抽出できません。 • Windows の場合，シンボリックリンク先が存在しない状態ではシンボリックリンクを抽出できません。 • Windows の場合，-p オプションは使用できません。 • Windows の場合，アーカイブからの抽出時にファイルに対してオーナー/グループ/パーミッションは設定できません。
touch	ファイルの最終アクセス日時と最終修正日時を変更します。	• Windows の場合，最終アクセス日時は変更できません。 • Windows の場合，ディレクトリの最終修正日時は変更できません。 • Windows の場合，環境変数TZ に設定したタイムゾーンと［日付と時刻のプロパティ］に設定したタイムゾーンを同じ値にする必要があります。 • Windows の場合，実際にファイルに設定される修正時刻の精度はファイルシステムの仕様に依存します。
tr	標準入力から入力された文字列を，1 バイトごとに置換または削除しながら標準出力に出力します。	制限事項はありません。
uname	OS またはハードウェアの情報を表示します。	制限事項はありません。
uniq	ソートされたファイルから重複した行を削除します。	制限事項はありません。
wc	ファイルのバイト，行，文字および単語をカウントします。	制限事項はありません。
which	実行する外部コマンドのコマンドパスを環境変数PATH に設定されているコマンド検索パスから求めます。	Windows の場合，which コマンドの説明個所に記載されているパス検索規則に該当する外部コマンドだけがコマンドパスの出力対象となります。
xargs	標準入力からコマンド引数を入力し，コマンドラインを生成して実行します。	制限事項はありません。

注※

UNIX 互換コマンドのうち，実行ファイル形式で提供するすべてのコマンドに共通する制限事項を次に示します。

- Windows の場合，UNIX 互換コマンドをコマンドプロンプトから実行するとき，ワイルドカードは展開されません。ジョブ定義スクリプトファイルに記述した場合は展開されます。
- 出力されるメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。
- Windows の場合，コマンドプロンプトからコマンドを実行するときは，ダブルクォーテーションを使用する必要があります。
- 使用可能なファイルには制限があります。詳細は「2.2.3　JP1/Advanced Shell で使用するファイル」を参照してください。
- コマンドで生成するパス名およびジョブ定義スクリプトファイルに記述したファイルには，パス変換機能は無効です。
- 次のプログラムを実行する機能で GUI を操作するアプリケーションを実行した場合，バッチジョブ実行時に処理が停止することがあります。

  ・awk コマンドのsystem 関数

  ・awk コマンドの書式 *コマンド名* | getline [*変数名*]

  ・awk コマンドの書式 print [*式*[, ...]] | *コマンド名*

  ・find コマンドの-exec プライマリおよび-ok プライマリ

  ・xargs コマンド

  また，adshexec コマンドを実行すると，新たにジョブが生成されます。

UNIX 互換コマンドのうち，実行ファイル形式でハードリンク，シンボリックリンクを使用するときの制限事項を次に示します。

- find コマンド，ls コマンド，stat コマンドでディレクトリへのハードリンク数を出力すると，UNIX 版と Windows 版で値が異なることがあります。
- Windows 版では 11-00 でハードリンク，シンボリックリンクに対応したため，11-00 より前と以降で一部の UNIX 互換コマンドの出力形式が異なります。ハードリンク，シンボリックリンクを使用しないで，かつ出力形式を 11-00 より前に戻したい場合は環境変数ADSH_LINK_SUPPORT にL0 を指定してください。環境変数ADSH_LINK_SUPPORT の詳細は「2.2.7　ハードリンク，シンボリックリンクを使用する」を参照してください。
- 環境変数ADSH_LINK_SUPPORT にL0，L1 以外の値を指定して UNIX 互換コマンドを実行するとコマンドは終了コード255 以上でエラー終了します。

(2)　スクリプト形式で提供するコマンド【Windows 限定】

UNIX 互換コマンドのうち，スクリプト形式で提供するコマンドは，ジョブ定義スクリプト中だけで実行できます。

UNIX 互換コマンドのうち，スクリプト形式で提供するコマンドのサンプルスクリプトファイルの格納場所は次のとおりです。

- Windows の実行環境の場合

インストール先フォルダ¥JP1ASE¥sample

- Windows の開発環境の場合

  インストール先フォルダ¥JP1ASD¥sample

UNIX 互換コマンドのうち，スクリプト形式で提供するコマンドは，JP1/Advanced Shell が提供するサンプルスクリプトファイルを「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」に示す事前準備をしてから使用してください。

使用できるスクリプト形式の UNIX 互換コマンドを次の表に示します。なお，提供するサンプルスクリプトファイルは Windows 限定です。UNIX でこれらのコマンドを使用する場合，OS 提供のコマンドを使用してください。

表 8-3　UNIX 互換コマンド（スクリプト形式）

コマンド名	機能概要
chmod	ファイルまたはフォルダのアクセス権を変更します。
su	su コマンドに指定されたプログラムを実行します。
who	現在ログインしているユーザーの情報を表示します。

UNIX 互換コマンドのうち，スクリプト形式でハードリンク，シンボリックリンクを使用するときの制限事項を次に示します。

- 環境変数 ADSH_LINK_SUPPORT に L0，L1 以外の値を指定し，UNIX 互換コマンドを実行するとコマンドは終了コード 255 以上でエラー終了します。

8.3　シェル運用コマンド

8.3.1　adshappagent コマンド（アプリケーション実行エージェント起動コマンド）【Windows 実行環境限定】

形式

```
adshappagent [-q] [-c]
```

機能

アプリケーション実行エージェントを起動し，adshappexec コマンドの実行を監視して，adshappexec コマンドで指定した実行アプリケーションを起動します。

adshappagent コマンドは adshappexec コマンドを実行するユーザー（JP1/Base のマッピング OS ユーザー）ごとに実行する必要があります。

使用するユーザーには Windows の管理ツールの［ローカルセキュリティポリシー］-［ローカルポリシー］-［ユーザー権利の割り当て］-［グローバルオブジェクトの作成］権限を付与する必要があります。

セキュリティ上の問題などから「グローバルオブジェクトの作成」権限を有効にできない場合は，「グローバルオブジェクトの作成」権限を有効にできるユーザーでアプリケーション実行エージェント機能を使用してください。

Windows のスタートアップに登録，解除を画面操作でできる機能も提供します。この機能を使用し，ログオン時に自動的に起動する運用とすることを推奨します。

Windows の［スタート］メニューから，［すべてのプログラム］－［Advanced Shell］－［アプリケーション実行エージェント］を選択すると，タスクバーの通知領域に［アプリケーション実行エージェント］アイコンが表示されます。

［アプリケーション実行エージェント］アイコンにカーソルを当てると，次のようにアイコンの説明が表示されます。

アイコンは次のように状態が変わります。

（アイコン色 白）実行アプリケーション未起動。

（アイコン色 緑）実行アプリケーション実行中。

（アイコン色 黄）実行アプリケーション強制終了。

起動中の実行アプリケーションの情報を参照する場合は，adshappagent または adshappexec コマンドの引数-c を使用してください。ジョブ定義スクリプト内などからは情報を取得できません。

引数

引数の指定なし

アプリケーション実行エージェントを起動します。

-q

アプリケーション実行エージェントを終了します。

クラスタ連携の系切り替えをする場合など，アプリケーション実行エージェントを終了させる場合，この引数を使用します。

-c

アプリケーション実行エージェント機能共有メモリの内容を標準出力に出力します。

この引数は adshcollect コマンドで使用します。

使用する場合にはファイルにリダイレクトする必要があります。

コンソールには表示しません。

また，取得する情報はログインユーザーごとの情報であり，実行したログインユーザーで取得する必要があります。

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了

注意事項

- adshappagent コマンドは，adshappexec コマンド実行前に実行しておく必要があります。
- adshappexec コマンドで実行アプリケーションの終了を待つ設定（-w）をした場合は，実行アプリケーションの終了までアプリケーション実行エージェントを終了しないでください。終了した場合，起動した実行アプリケーションは起動したままとなり，実行アプリケーションを起動したジョブはエラーとなります。
- ドメインユーザーとドメイン以外のユーザーもアプリケーション実行エージェント機能では別ユーザーとして扱います。

- アンインストール時，およびバージョンアップインストール時にはアプリケーション実行エージェントを終了し，アプリケーション実行エージェントをスタートアップに登録したユーザーでログインし，スタートアップからアプリケーション実行エージェントを削除する必要があります。スタートアップから削除しない状態でアンインストールした場合は，再度 JP1/Advanced Shell をインストールし，スタートアップが残っているユーザーでログインし，スタートアップの解除をする必要があります。
- アプリケーション実行エージェントプログラム実行中に系切り替えをした場合，系切り替え前の情報を系切り替え後に引き継ぎません。-q オプションでアプリケーション実行エージェントを終了し，系切り替え実施後にアプリケーション実行エージェントを起動してください。
- 同一のユーザーに対してアプリケーション実行エージェントは一つしか起動できません。複数起動した場合，エラーとなります。
- adshappagent コマンドはログオン空間で動作させる必要があります。サービス空間で動作する JP1/AJS のジョブの延長で動作させないでください。動作させた場合，対話型の実行アプリケーションは操作できません。
- 引数-q および引数-c はほかの引数と同時に指定しないでください。同時に指定した場合はエラーとなります。

8.3.2 adshappexec コマンド（GUI アプリケーション実行コマンド）【Windows 実行環境】

形式

```
adshappexec [-m] [-d ワークフォルダ] [-v 表示名] {-w 実行アプリケーション名 | -n 実行アプリケーション名} [-- 引数1 引数2...]
```

機能

アプリケーション実行エージェントに実行アプリケーションの起動を要求します。JP1/AJS の PC ジョブ定義の実行ファイル名などに指定して動作させます。

実行アプリケーションの戻り値は，標準出力，およびメッセージ（JP1/AJS の実行結果詳細）に出力します。ジョブの戻り値として後続ジョブで使用する場合はコマンド置換で変数に格納します。

引数

--m

標準エラー出力へのメッセージの出力を抑止します。

標準入出力を使用できない環境で使用します。

コマンドの引数指定エラー，およびライセンスチェックエラーは，-m オプションを指定しても出力されます。

-d ワークフォルダ ～<パス名>((1～247 バイト))

実行アプリケーション実行時のワークフォルダを指定します。

ワークフォルダを指定しなかった場合は，adshappexec コマンド実行時のカレントパスで実行アプリケーションが動作します。

ワークフォルダはスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

-v 表示名 ～<パス名>((1～247 バイト))

［アプリケーション実行エージェント］アイコンを左クリックしたときに表示される表示名を指定します。

表示名はスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

表示名を省略した場合には，実行アプリケーション名を出力します。

複数の実行アプリケーションを動作させたときにアプリケーションを区別するためにこの引数を指定することを推奨します。

-w 実行アプリケーション名 ～<パス名>((1～247 バイト))

実行アプリケーションの終了まで実行を終了しません。

実行アプリケーション名は，実行アプリケーションのファイル名を指定します。

実行アプリケーション名はスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

-n 実行アプリケーション名 ～<パス名>((1～247 バイト))

実行アプリケーションの終了を待たずに終了します。

この引数を使用した場合，同時実行数の制限を受けずに起動できます。

実行アプリケーション名は，実行アプリケーションのファイル名を指定します。

実行アプリケーション名はスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

-- 引数 1 引数 2... ～<引数>((1～1023 バイト))

「--」のあとに実行アプリケーションの実行時に指定するパラメーターを指定します。

関連付けを行った実行アプリケーションを指定した場合には引数を指定しないでください。

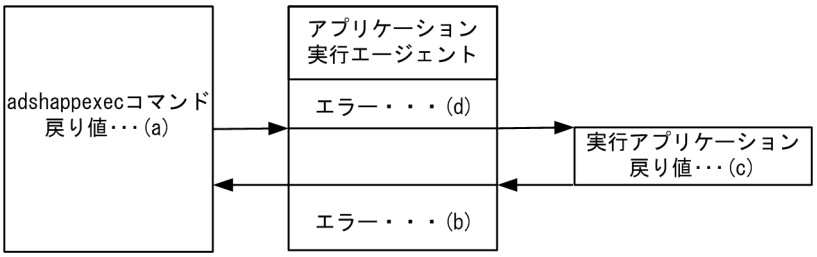
引数は，合計の引数長が 1,023 バイト以内であれば，いくつでも指定できます。

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了

adshappexec コマンドでは，3 つのプロセスでの終了コードがあります。プロセスごとの終了コードの扱いは次のようにします。

図 8-1　プロセスごとの終了コードの扱い



エラー発生個所での戻り値の出力は，次のとおりです。

(a) adshappexec コマンド処理でのエラー

コマンドの戻り値として設定します。標準エラー出力（JP1/AJS - View からの実行の場合実行結果詳細）にメッセージを出力します。

(b) アプリケーション実行エージェントでの adshappexec コマンドからの要求処理中のエラー

コマンドの戻り値として設定します。標準エラー出力（JP1/AJS - View からの実行の場合実行結果詳細）には，アプリケーション実行エージェントでエラー出力します。

(c) 実行アプリケーションの戻り値

標準出力に出力します。また，標準エラー出力（JP1/AJS - View からの実行の場合実行結果詳細）には，実行アプリケーションの戻り値を出力します。

実行アプリケーションの戻り値で 0 以外の戻り値を返しても，adshappexec コマンドは異常終了しません。

実行アプリケーションの戻り値を確認するにはメッセージを確認するか，標準出力の内容を確認してください。

(d) アプリケーション実行エージェントでの adshappexec コマンドからの要求処理中以外のエラー

アプリケーション実行エージェントのエラーとして，adshappagent コマンドの戻り値として設定します。

エラーの発生個所によって，メッセージをメッセージボックスで出力，またはアプリケーション実行エージェント機能ログに出力します。

注意事項

- adshappexec コマンドは，アプリケーション実行エージェントが起動している状態で実行する必要があります。起動する前に実行した場合，エラーになります。また，adshappexec コマンド実行中にアプリケーション実行エージェントを終了した場合，エラーになります。
- adshappexec コマンドの-w 引数と-n 引数は必ず指定してください。

  次の場合，最後に指定したものが有効になります。

  - -w オプションと-r オプションを同時に指定した場合
  - -w オプションを複数指定した場合
  - -r オプションを複数指定した場合

- JP1/AJS の PC ジョブの設定項目である環境変数，および環境変数ファイル名は実行アプリケーションに引き継ぎません。また，実行アプリケーションは実行ユーザーの環境変数を使用します。
- adshappexec コマンドの引数-w を指定した実行アプリケーションの最大同時起動数は 5 つです。5 つを超える数の実行アプリケーションを起動した場合，起動中の実行アプリケーションの終了を待ちます。

  実行アプリケーションの終了を待った場合，実行中の実行アプリケーションが終了してから終了待ちの実行アプリケーションが起動するまで 1 分以上の時間がかかる場合があります。
- 同じオプションを複数指定した場合，最後の指定が有効になります。
- 次の仕様のアプリケーションを-w 引数で実行した場合，指定したファイルを閉じても adshappexec コマンドが終了しないことがあります。
  - 実行アプリケーションに指定したファイルを閉じても，アプリケーション自体が終了しない

  このような場合は，アプリケーション自体が終了すれば，adshappexec コマンドも終了します。

  例：実行アプリケーションとして指定した Excel ファイルだけ閉じて，Excel 自体は終了させていない場合
- 実行アプリケーションのプロセス起動に関する仕様によっては，-w 引数を指定しても，アプリケーションの終了を待たずに adshappexec コマンドが終了することがあります。

  次に示す場合があります。

  (1)KNAX7259-W が出力される場合

  例：Excel ファイルを実行アプリケーションとして実行する前から，すでに Excel 自体が動作していた場合

  この場合は，Excel が起動していない状態で運用するか，-w 引数を使用しない運用としてください。

  (2)KNAX7259-W が出力されない場合

  例：explorer.exe を実行アプリケーションとして実行した場合

  この場合は，-w 引数を使用しない運用としてください。
- 強制終了時は次のことに注意してください。
  - adshappexec コマンド実行中にジョブを強制終了した場合

    起動した実行アプリケーションは起動したままとなります。
  - TRAP_ACTION_SIGTERM パラメーターで TERM を指定した場合

    trap コマンドによる動作定義に adshappexec コマンドを指定しないでください。

    対話操作をする実行アプリケーションを指定した場合，応答待ちになりジョブ定義スクリプトが終了しない可能性があります。

使用例

- 実行アプリケーションを実行し，終了を待つ場合

```
adshappexec -w “c:¥appfolder¥app.exe “・・・1
echo app終了・・・2
```

  c:¥appfolder¥app.exe を実行し（処理 1），終了待ちします（処理 2）。

- 実行アプリケーションを実行し，終了を待たない場合

```
adshappexec -n  “c:¥appfolder¥app.exe “・・・1
echo app起動・・・2
```

c:¥appfolder¥app.exe を実行し（処理 1），終了を待たないで処理 2 を実行します。

- シェル拡張コマンドを使用した実行アプリケーションの戻り値を判定する場合

```
exeapprc=$(adshappexec -w c:¥¥appfolder¥¥app.exe)
if [[ $? != 0 ]] ・・・1
then
  echo adshappexec error
  exit
fi
echo $exeapprc
if [[ $exeapprc != 0 ]] ・・・2
then
  echo app.exe error
  exit
fi
```

処理 1 は，adshappexec コマンドの戻り値を判定します。

処理 2 は，c:¥¥appfolder¥¥app.exe(実行アプリケーション)の戻り値を判定します。

8.3.3 adshchmsg コマンド（障害発生時に，応答要求メッセージに対して手動で応答する）

形式

```
adshchmsg  [-h 論理ホスト名] -n 応答要求メッセージ番号 {-r 応答|-d}
```

機能

障害発生時に共有メモリ上にある応答要求メッセージに対して応答を入力します。または，応答要求メッセージの応答待ち状態をキャンセルします。

応答要求メッセージの応答要求メッセージ番号を-n オプションに指定し，それに対する応答を-r オプションで指定します。応答要求メッセージの応答待ち状態をキャンセルする場合は，応答要求メッセージの応答要求メッセージ番号を-n オプションに指定し，それに対するキャンセル要求として-d オプションを指定します。

実行環境の場合，このコマンドは，JP1/Advanced Shell がインストールされているマシンの管理者権限を持つ root または Administrators が実行できます。開発環境の場合，このコマンドは一般ユーザーでも実行できます。

引数

-h 論理ホスト名　～<論理ホスト名>((1～255 バイト))

論理ホスト環境で運用している場合，このコマンドを実行する論理ホストのホスト名を指定します。

Windows の場合，196 バイトを超える論理ホスト名は指定できません。また，論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

-n 応答要求メッセージ番号　～<符号なし整数>((1～2147483647))

応答を入力したい応答要求メッセージまたは応答待ち状態をキャンセルしたい応答要求メッセージの応答要求メッセージ番号を指定します。応答要求メッセージ番号には，adshlsmsg コマンドで表示される応答要求メッセージの番号を指定します。

-r オプションまたは-d オプションの指定がない場合はエラーになります。

応答要求メッセージ番号の指定をしなかった場合，その次に指定されたオプションが引数として扱われエラーになります。

-r 応答　～< ASCII 文字列>((0～512 バイト))

応答待ちイベントの発行元に入力する応答を指定します。スペースを含む応答を入力する場合は"（ダブルクォーテーション）で囲みます。

-n オプションの指定がない場合はエラーになります。

512 バイトを超える文字列を指定した場合は，512 バイトまでを応答として扱います。また，改行を含む文字列が指定された場合，改行以降の文字列は無視されます。

-d

-n オプションで指定した応答要求メッセージの応答待ち状態をキャンセルします。-n オプションの指定がない場合はエラーになります。

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了

注意事項

- 次の場合，最後に指定したものが有効になります。
  - -r オプションと-d オプションを同時に指定した場合
  - -r オプションを複数指定した場合
  - -n オプションを複数指定した場合

8.3.4　adshcvmerg コマンド（カバレージ情報をマージする）

形式

```
adshcvmerg -o 出力するascファイルのパス名 ベースとなるascファイルのパス名 マージするascファイルのパス名
```

機能

ベースとなる asc ファイルとマージする asc ファイルのカバレージ情報をマージして，出力する asc ファイルに出力します。

マージする入力ファイルが同一ファイルである場合，コマンドエラーになります。入力ファイルが同一であるかどうかは，絶対パスを含めて同一ファイル名であるかどうかで判定します。入力ファイルの内容が同一であるかどうかでは判定しません。

引数

Windows の場合，英大文字，英小文字の相違を区別しません。UNIX の場合，英大文字，英小文字の相違を区別します。

-o 出力する asc ファイルのパス名

Windows の場合　～<パス名>((1～229 バイト))

UNIX の場合　～<パス名>((1～1,005 バイト))

マージした結果を出力するファイルのパス名を指定します。

ベースとなる asc ファイルのパス名

Windows の場合　～<パス名>((1～229 バイト))

UNIX の場合　～<パス名>((1～1,005 バイト))

ベースとなるファイルのパス名を指定します。

マージする asc ファイルのパス名

Windows の場合　～<パス名>((1～229 バイト))

UNIX の場合　～<パス名>((1～1,005 バイト))

マージするファイルのパス名を指定します。

終了コード

終了コード	意味
0	正常終了
1	asc ファイルの読み込みの途中で，ファイルの終了を検出しました。asc ファイルが異常です。
2	ファイルのロック解除でエラーが発生しました。
3	コマンドラインの指定に誤りがあります。

終了コード	意味
4	環境変数の設定に誤りがあります。 • 環境変数 LANG に設定している文字エンコーディングに対応していません。
5	次に示す 2 つのジョブ定義スクリプトが異なります。 • ベース asc ファイルのカバレージ情報を蓄積したときのジョブ定義スクリプト • マージ asc ファイルのカバレージ情報を蓄積したときのジョブ定義スクリプト
6	ファイルのオープン処理でエラーが発生しました。 • ファイルの種類が適切でない場合も，このエラーとなります。
7	ファイルのロックでエラーが発生しました。
8	ファイル名の変更処理でエラーが発生しました。
9	ファイルの入出力でエラーが発生しました。
10	メモリ不足が発生しました。
11	メッセージ出力処理でエラーが発生しました。
12	標準エラー出力への出力処理でエラーが発生しました。
13	内部処理矛盾を検出しました。
14	asc ファイルのデータ形式の誤りを検出しました。asc ファイルが不当です。
15	日時の取得でエラーが発生しました。
16	ジョブ定義スクリプトファイルの情報取得でエラーが発生しました。
17	コマンドで asc ファイルを処理できません。 asc ファイルは，異なるバージョンで作成されたものです。
19	コマンドの処理中に，OS とのやり取りでエラーが発生しました。

注意事項

- ベース asc ファイル，およびマージ asc ファイルのカバレージ情報を採取したときのジョブ定義スクリプトが同一の場合にだけ，マージできます。ジョブ定義スクリプトが異なる場合は，コマンドエラーとなります。
- 出力 asc ファイルに，ベース asc ファイルまたはマージ asc ファイルと同一のファイルを指定すると，コマンドエラーとなります。出力 asc ファイルは，ベース asc ファイルおよびマージ asc ファイルとは異なるファイルを指定してください。
- ベース asc ファイルおよびマージ asc ファイルに同一のファイルを指定すると，コマンドエラーとなります。
- ベース asc ファイルおよびマージ asc ファイルに指定したファイルが，同一であるかどうかは，指定したファイルの絶対パス名で判断します。絶対パス名が同一である場合，同一のファイルと判断します。

使用例

- JOB_user1.asc と JOB_user2.asc のカバレージ情報をマージして JOB_user3.asc に出力します。

```
adshcvmerg -o JOB_user3.asc JOB_user1.asc JOB_user2.asc
```

8.3.5 adshcvshow コマンド（カバレージ情報を表示する）

形式

```
adshcvshow { [ -l n1 [- [n2] ] [,n3 [- [n4] ] ] ] ...] |-s} ascファイルのパス名
```

機能

引数に指定した asc ファイルのカバレージ情報を表示します。

引数

-l n1 [- [n2]] [,n3 [- [n4]]] …

カバレージ情報を表示する範囲のジョブ定義スクリプトの行番号を指定します。

n1 [- [n2]] の形式で範囲を指定します。n1-は，行番号 n1 の行から最終行までの範囲を意味します。範囲は，「,」で区切ると複数指定できます。

- n1：表示する範囲の開始行の行番号です。
- n2：表示する範囲の終了行の行番号です。

このオプションを指定しない場合，ジョブ定義スクリプトの全行を表示範囲とします。

行の指定の形式は「,」で複数指定し，「-」で範囲を指定します。「-」の後ろに数字を指定しない場合は，前の値から最終行までが範囲になります。

-s

バックアップしているジョブ定義スクリプトファイルの内容を表示します。

このオプションは，asc ファイルがどのジョブ定義スクリプトファイルと対応しているかを調べる場合や差分を調べる場合に使用します。

asc ファイルのパス名

Windows の場合　～<パス名>((1～229 バイト))

UNIX の場合　～<パス名>((1～1,005 バイト))

表示するカバレージ情報が格納された asc ファイルのパス名を指定します。

終了コード

終了コード	意味
0	正常終了
1	asc ファイルの読み込みの途中で，ファイルの終了を検出しました。asc ファイルが異常です。
2	ファイルのロック解除でエラーが発生しました。
3	コマンドラインの指定に誤りがあります。
4	環境変数の設定に誤りがあります。 • 環境変数 LANG に設定している文字エンコーディングに対応していません。
6	ファイルのオープン処理でエラーが発生しました。 • ファイルの種類が適切でない場合も，このエラーとなります。
7	ファイルのロックでエラーが発生しました。
8	ファイル名の変更処理でエラーが発生しました。
9	ファイルの入出力でエラーが発生しました。
10	メモリ不足が発生しました。
11	メッセージ出力処理でエラーが発生しました。
12	標準エラー出力への出力処理でエラーが発生しました。
13	内部処理矛盾を検出しました。
14	asc ファイルのデータ形式の誤りを検出しました。asc ファイルが不当です。
15	日時の取得でエラーが発生しました。
16	ジョブ定義スクリプトファイルの情報取得でエラーが発生しました。
17	コマンドで asc ファイルを処理できません。 asc ファイルは，異なるバージョンで作成されたものです。
19	コマンドの処理中に，OS とのやり取りでエラーが発生しました。

注意事項

- -s オプションと-l オプションは同時に指定した場合，エラーになります。
- 行の範囲指定で，終了行の行番号が開始行の行番号より小さい場合，エラーとなります。
- 開始行の行番号と終了行の行番号が等しい場合は，その行だけが範囲となります。
- 開始行の行番号がジョブ定義スクリプトファイルの行数より大きい場合は，無視されます。
- 終了行の行番号がジョブ定義スクリプトファイルの行数より大きい場合，終了行を最終行とします。
- 範囲が重複している場合，範囲の和と解釈します。例えば，-l 1-10,5-20 の場合，-l 1-20 と同じになります。
- 範囲の形式が誤っている場合，エラーになります（例：1-10-20）。

- 行指定で０行目を指定した場合，エラーになります。
- -l オプションを指定している場合，Total information 以降の行は出力しません。

使用例

- １行目～10 行目，15 行目および 21 行目～最終行目のカバレージ情報を表示します。

```
adshcvshow -l 1-10,15,21-  JOB_user1.asc
```

- ２行目～８行目のカバレージ情報を表示します。

```
adshcvshow -l 2-6,4-8 JOB_user1.asc
```

- ジョブ定義スクリプトファイルが９行である場合，何も表示されません。

```
adshcvshow -l 10-15 JOB_user1.asc
```

- ジョブ定義スクリプトファイルが９行である場合，２行目～４行目のカバレージ情報を表示します。

```
adshcvshow -l 10-15,2-4 JOB_user1.asc
```

8.3.6 adshevtout コマンド（ジョブ定義スクリプトの稼働実績情報を出力する）

形式

```
adshevtout  [-s ジョブの実行開始日時の下限]
            [-e ジョブの実行開始日時の上限]
            [-c JP1/AJSのスケジューラーサービス名]
            [-r JP1/AJSのルートジョブネット名]
            [-k JP1/AJSのジョブ実行ID]
            [-n JP1/AJSのジョブ番号]
            [-g JP1/AJSのジョブ名]
            [-u JP1/Advanced Shellの実行ユーザー名]
            [-p ジョブ定義スクリプトファイルのパス名]
            [-i JP1/Advanced Shellのジョブ識別子]
            [-j スプールジョブ名]
            [-t]
            [-d]
            [-m]
            [-z]
            [-h 論理ホスト名]
```

機能

指定された条件に該当するジョブのジョブ定義スクリプト稼働実績情報をイベントファイルから検索し，CSV 形式で出力します。出力先は標準出力（stdout）です。

このコマンドは，JP1/Advanced Shell - Developer では使用できません。

出力条件の指定

次に示す引数で，ジョブ定義スクリプト稼働実績情報を出力するジョブを指定します。

```
[-s ジョブの実行開始日時の下限]
[-e ジョブの実行開始日時の上限]
[-c JP1/AJSのスケジューラーサービス名]
[-r JP1/AJSのルートジョブネット名]
[-k JP1/AJSのジョブ実行ID]
[-n JP1/AJSのジョブ番号]
[-g JP1/AJSのジョブ名]
[-u JP1/Advanced Shellの実行ユーザー名]
[-p ジョブ定義スクリプトファイルのパス名]
[-i JP1/Advanced Shellのジョブ識別子]
[-j スプールジョブ名]
```

出力条件を複数指定した場合は，すべての条件を満たすジョブのジョブ定義スクリプト稼働実績情報を出力します。

出力条件を指定しない場合，物理ホストまたは指定された論理ホストのすべてのジョブのジョブ定義スクリプト稼働実績情報を出力します。

出力条件としてジョブの属性（起動日時，ジョブ識別子，ジョブ名など）を指定した場合，ルートジョブの属性値で判定します。

すべての出力条件を満たすジョブに属するルートジョブ，すべての子孫ジョブのジョブ定義スクリプト稼働実績情報を出力します。ルートジョブまたは特定の子孫ジョブに限定してジョブ定義スクリプト稼働実績情報を出力することはできません。

ジョブ定義スクリプト稼働実績情報の出力内容

ジョブ定義スクリプト稼働実績情報は，標準の場合，1 行目にはヘッダ情報，2 行目以降に稼働実績情報やメッセージが出力されます。

adshevtout コマンドの次に示す引数指定で出力情報を選択できます。

```
[-t]
[-d]
[-m]
[-z]
```

稼働実績情報の出力例は，「3.7.9　ジョブ定義スクリプトの稼働実績情報の出力内容」を参照してください。

次の引数を指定した場合，論理ホストで実行したジョブの稼働実績情報を出力します。

```
[-h 論理ホスト名]
```

この引数を指定しない場合，物理ホストで実行したジョブの稼働実績情報を出力します。

引数

-s *ジョブの実行開始日時の下限*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，ジョブの実行開始日時の下限を指定します。

日時の指定形式については，項目「ジョブの実行開始日時」を参照してください。

指定を省略した場合，出力するジョブの実行開始日時の下限は制限されません。

-e *ジョブの実行開始日時の上限*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，ジョブの実行開始日時の上限を指定します。

日時の指定形式については，項目「ジョブの実行開始日時」を参照してください。

指定を省略した場合，出力するジョブの実行開始日時の上限は制限されません。

-c *JP1/AJS のスケジューラーサービス名*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，JP1/AJS のスケジューラーサービス名を指定します。

次の 2 つが文字列として一致するジョブが出力できます。

- この引数で指定したスケジューラーサービス名
- ジョブを起動した JP1/AJS のスケジューラーサービス名（JP1/AJS がジョブを起動したときに設定した環境変数AJS_AJSCONF の値）

-r *JP1/AJS のルートジョブネット名*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，JP1/AJS のルートジョブネット名を指定します。

次の 2 つが文字列として一致するジョブが出力できます。

- この引数で指定したルートジョブネット名
- ジョブを起動したときの JP1/AJS のルートジョブネット名（JP1/AJS がジョブを起動するときに設定した環境変数AJSNETNAME の値）

-k *JP1/AJS のジョブ実行 ID*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，JP1/AJS のジョブ実行 ID を指定します。

次の 2 つが文字列として一致するジョブが出力できます。

- この引数で指定したジョブ実行 ID
- ジョブを起動したときの JP1/AJS のジョブ実行 ID（JP1/AJS がジョブを起動したときに設定した環境変数AJSEXECID の値）

-n *JP1/AJS のジョブ番号*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，JP1/AJS のジョブ番号を指定します。

次の 2 つが文字列として一致するジョブが出力できます。

- この引数で指定したジョブ番号
- ジョブを起動したときのJP1/AJSのジョブ番号（JP1/AJSがジョブを起動するときに設定する環境変数JP1JobIDの値）

例えば，ジョブ番号「0000012345」のジョブを指定する場合，「-n 0000012345」と指定する必要があります。「-n 12345」と指定すると，ジョブ番号は一致しないと判定します。

ジョブ番号の形式はプラットフォーム間で異なる場合があります。

ジョブ番号の詳細は，JP1/AJSのマニュアルを参照してください。

-g *JP1/AJSのジョブ名*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，JP1/AJSのジョブ名を指定します。

次の2つが文字列として一致するジョブが出力できます。

- この引数で指定したジョブ名
- ジョブを起動したときのJP1/AJSのジョブ名（JP1/AJSがジョブを起動するときに設定する環境変数AJSJOBNAMEの値）

-u *JP1/Advanced Shellの実行ユーザー名*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，ジョブを実行したadshexecコマンドを実行したプロセスの実行ユーザー名を指定します。

次の2つが文字列として一致するジョブが出力できます。

- この引数で指定したユーザー名
- ジョブを実行したadshexecコマンドのプロセスのユーザー名

-p *ジョブ定義スクリプトファイルのパス名*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，ジョブを実行するときにadshexecコマンドに指定したジョブ定義スクリプトファイルのパス名を指定します。

次の2つが文字列として一致するジョブが出力できます。

- この引数で指定したパス名
- adshexecコマンドに指定されたジョブ定義スクリプトファイルのパス名

同一のジョブ定義スクリプトファイルのパスとして解釈できるパス名であっても，文字列として一致しない場合，出力するジョブではないと判断します。

パス名が一致しないと判断する例を次に示します。

adshexecコマンドの実行時のカレントディレクトリが/home/user1である場合

adshexecコマンドに指定したパス名：./test1.ash

adshevtoutコマンドに指定したパス名：/home/user1/test1.ash

-i *JP1/Advanced Shellのジョブ識別子*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，JP1/Advanced Shellのジョブ識別子を指定します。

次の2つが文字列として一致するジョブが出力できます。

- この引数で指定したジョブ識別子
- ジョブの JP1/Advanced Shell のジョブ識別子（環境変数ADSH_JOBID の値）

例えば，ジョブ識別子「000001」のジョブを指定する場合は，「-i 000001」と指定する必要があります。先頭の 0 は省略できません。

-j *スプールジョブ名*

ジョブ定義スクリプト稼働実績情報を出力するジョブの条件として，スプールジョブディレクトリのスプールジョブ名を指定します。このスプールジョブ名はジョブの JP1/Advanced Shell のジョブ名を指定します。ただし，シェル変数ADSH_SPOOL_JOBNAME でスプールジョブディレクトリのスプールジョブ名を指定した場合は，シェル変数ADSH_SPOOL_JOBNAME の値を指定します。

次の 2 つが文字列として一致するジョブが出力できます。

- この引数で指定した*スプールジョブ名*
- シェル変数ADSH_SPOOL_JOBNAME の指定がない場合は JP1/Advanced Shell のジョブ名（環境変数ADSH_JOB_NAME の値），シェル変数ADSH_SPOOL_JOBNAME の指定がある場合はシェル変数ADSH_SPOOL_JOBNAME で指定したスプールジョブディレクトリのスプールジョブ名

-t

ジョブ定義スクリプト稼働実績情報を出力する際，先頭にヘッダ情報を出力しないことを指定します。

-d

ジョブ定義スクリプト稼働実績情報を出力しないことを指定します。

ヘッダ情報だけを出力したい場合に使用します。

-m

ジョブ定義スクリプト稼働実績情報として，メッセージだけ出力することを指定します。

-z

環境変数の情報をジョブ定義スクリプト稼働実績情報へ出力しないことを指定します。

-h *論理ホスト名*

ジョブ定義スクリプト稼働実績情報を出力するジョブを実行した論理ホスト名を指定します。

adshevtout コマンドは，指定された論理ホストに対応するスプールにあるイベントファイルからジョブ定義スクリプト稼働実績情報を出力します。

Windows の場合，196 バイトを超える論理ホスト名は指定できません。また，論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

この引数は実行環境を指定する引数です。ジョブ定義スクリプト稼働実績情報の出力するジョブの条件を指定する引数ではありません。

この引数に指定した論理ホスト名が環境ファイルに定義されていない場合，-h の指定を無視します。

終了コード

終了コード	意味
0	正常終了

終了コード	意味
1	コマンドラインの指定に誤りがあります。
2	環境変数の設定に誤りがあります。
3	環境変数LANG に設定している文字エンコーディングに対応していません。
4	処理をスキップしたイベントファイルがあります。 原因については，コマンドが出力したメッセージを確認してください。
5	スプールの参照で入出力エラーが発生し，参照できませんでした。
6	ほかのコマンドがアクセスしているため，スプールを参照できませんでした。
7	メッセージ出力処理でエラーが発生しました。
8	標準出力への出力処理でエラーが発生しました。
10	日時の取得でエラーが発生しました。
11	メモリ不足が発生しました。
12	内部処理矛盾を検出しました。
13	初期化処理でエラーが発生しました。

終了コードは基本的に，コマンド実行中に発生した事象に対する終了コードの中の最大値となります。ただし，終了コード4 は，それ以外の事象が発生しなかった場合だけ出力されます。

同一引数の複数指定

同一の引数を複数回指定した場合，最後の指定を有効とします。

（例）

次の場合，-s 19900401 が指定されていると解釈します。

```
adshevtout -s 20120411  -s 19900401
```

異なる引数の組み合わせ

出力情報を選択する引数が組み合わせて指定された場合，引数は次の表の優先順位に従って解釈されます。

引数の優先順位※	引数	機能
1	-d	稼働実績情報を出力しない
2	-m	メッセージだけを出力する
3	-z	環境変数の情報を出力しない

注※

数値が小さいほど優先順位が高くなります。

優先順位の高い引数が指定されている場合，優先順位の低い引数は無視されます。

指定されている引数の形式が正しくない場合は，引数の優先順位に関係なく，コマンドエラーとなります。

ジョブの実行開始日時

ジョブの実行開始日時の下限，上限の日時は次の 3 とおりの形式で指定できます。

YYYY に指定できる西暦年の範囲は 1970～2038 です。

表 8-4　ジョブの実行開始日時の下限，上限の指定と解釈

日時の指定形式	指定の解釈
YYYYMMDD,*hhmmss*	年月日と時分秒を指定する形式です。 上限の場合，指定された時分秒＋ 1 秒と解釈します。時分秒を「235959」と指定した場合，指定された年月日の翌日の 00:00:00 と解釈します。
YYYYMMDD	年月日だけを指定する形式です。 時分秒は次のように解釈します。 下限の場合 　指定された年月日の 00:00:00 と解釈します。 上限の場合 　指定された年月日の翌日の 00:00:00 と解釈します。
,*hhmmss*	時分秒だけを指定する形式です。 年月日はコマンドの実行日と解釈します。 上限の場合，指定された時分秒＋ 1 秒と解釈します。時分秒を「235959」と指定した場合，コマンドの実行日の翌日の 00:00:00 と解釈します。

日時の解釈の例を次の表に示します。表の項番 5～7 は，adshevtout コマンドを 2012 年 10 月 23 日に実行した場合の例です。

項番	コマンドでの指定	コマンドによって解釈される日時	
		実行開始日時の下限の場合	実行開始日時の上限の場合
1	20120501,000000	2012 年 05 月 01 日 00 時 00 分 00 秒	2012 年 05 月 01 日 00 時 00 分 01 秒
2	20120501,100000	2012 年 05 月 01 日 10 時 00 分 00 秒	2012 年 05 月 01 日 10 時 00 分 01 秒
3	20120501,235959	2012 年 05 月 01 日 23 時 59 分 59 秒	2012 年 05 月 02 日 00 時 00 分 00 秒
4	20120501	2012 年 05 月 01 日 00 時 00 分 00 秒	2012 年 05 月 02 日 00 時 00 分 00 秒
5	,000000	2012 年 10 月 23 日 00 時 00 分 00 秒	2012 年 10 月 23 日 00 時 00 分 01 秒
6	,100000	2012 年 10 月 23 日 10 時 00 分 00 秒	2012 年 10 月 23 日 10 時 00 分 01 秒
7	,235959	2012 年 10 月 23 日 23 時 59 分 59 秒	2012 年 10 月 24 日 00 時 00 分 00 秒

コマンドに指定された日時は，コマンド実行時の環境変数TZ に設定されたタイムゾーンに応じて解釈します。

コマンドで，文字列として同じ日時を指定しても，環境変数TZ のタイムゾーンが異なると，コマンドが解釈した日時は異なります。注意してください。

指定できる日時は次のとおりです。

- 協定世界時（UTC）で表現した場合
  1970 年 1 月 1 日 00:00:00～2038 年 1 月 19 日 03:14:07
- 日本標準時（UTC+9）で表現した場合
  1970 年 1 月 1 日 09:00:00～2038 年 1 月 19 日 12:14:07

タイムゾーンが上記以外の場合，指定できる日時の範囲の表現は，使用しているタイムゾーンに応じて変わります。

また，各 OS が提供する時刻関連の関数の実装の相違で，上記の範囲の日時であってもエラーとなる場合があります。次の場合はエラーとなります。

- 不当な日時が指定された場合
- 実行開始日時の下限が，実行開始日時の上限よりあとの場合

環境変数TZ

環境変数TZ に設定されたタイムゾーンは，コマンド実行時に次に示す日時の解釈，日時の表現で参照します。

- コマンドの引数に指定された日時の解釈
- ジョブ定義スクリプト稼働実績情報内の日時の表現

タイムゾーンと日時の表現の関係は，「3.7.3　稼働実績情報の日時とタイムゾーンの関係」を参照してください。

環境変数TZ は次に示す POSIX 形式で，符号に注意して指定してください。

（例）

```
export TZ=JST-9
```

環境変数TZ は，次のような Time Zone Database 形式では設定しないでください。

（使用できない例）

```
export TZ=Asia/Tokyo
export TZ=Japan
```

環境変数TZ に，サマータイムの情報を設定して，adshevtout コマンドを実行しないでください。adshevtout コマンドはサマータイムには対応していません。

環境変数TZ の設定の詳細については，使用している OS の仕様を参照してください。

ジョブの実行開始日時の下限

-s でジョブの実行開始日時の下限が指定された場合，ジョブ定義スクリプト稼働実績情報を出力するジョブは，次に示す条件が成立する必要があります。

$ts \leqq tj$

tj：ジョブの実行開始日時

ts：adshevtout コマンドが解釈したジョブの実行開始日時の下限

ジョブの実行開始日時の上限

-e でジョブの実行開始日時の上限が指定された場合，ジョブ定義スクリプト稼働実績情報を出力するジョブは，次に示す条件が成立する必要があります。

$tj < te$

tj：ジョブの実行開始日時

te：adshevtout コマンドが解釈したジョブの実行開始日時の上限

出力できるジョブ定義スクリプト稼働実績情報

- 出力できるジョブ定義スクリプト稼働実績情報は，コマンドを実行したユーザーがアクセスできるイベントファイルに格納されているものです。コマンドを実行したユーザーがアクセスできない場合，メッセージを出力し，アクセスできないイベントファイルにあるジョブ定義スクリプト稼働実績情報を出力しません。
- 実行中のジョブ，adshexec コマンドがエラー終了したジョブのジョブ定義スクリプト稼働実績情報は出力しません。具体的には，スプールジョブのディレクトリ名が次の形式であるジョブのジョブ定義スクリプト稼働実績情報は出力しません。
  - 「*ジョブ識別子*」
  - 「*ジョブ識別子*-」
- 削除中のジョブのジョブ定義スクリプト稼働実績情報は出力しません。

  スプールジョブのディレクトリの直下に次に示すスプールジョブ管理ファイルが存在しない場合，ジョブは削除中と判断します。
  - UNIX の場合：.sysout
  - Windows の場合：sysout.ini

adshhk コマンドとの同時実行

- adshevtout コマンドとadshhk コマンド（スプールジョブの削除）は，スプールディレクトリを排他制御します。
- 同一のスプールディレクトリに対して，adshevtout コマンドとadshhk コマンドは同時に実行できません。
- 同一のスプールディレクトリに対して，複数のadshevtout コマンドは同時に実行できます。

- スプールディレクトリが排他制御されているためにadshevtout コマンドが実行できない場合，エラーメッセージを出力して処理を終了します。

注意事項

- 出力されるジョブ定義スクリプト稼働実績情報のサイズが大きい場合，コマンドの出力条件を日時単位などで分けて実行し，出力される稼働実績情報を分割してください。

使用例

- JP1/Advanced Shell のジョブ識別子 000100 のジョブ定義スクリプト稼働実績情報をファイルout.csv に出力します。

```
adshevtout -i 000100  >  out.csv
```

8.3.7　adshexec コマンド（バッチジョブを実行する）

形式

通常実行する場合

```
adshexec [-v] [-c] [-m {EXTENDED|SIMPLE|MINIMUM} ]
         [-t [-f] [-o ascファイルのパス名] ] [-h 論理ホスト名]
         [-s {SPOOL|PARENT} ] [-x]
         {-r コマンドライン|ジョブ定義スクリプトファイルのパス名}
         [実行時パラメーター]
```

デバッガモードで起動する場合【UNIX 限定】

```
adshexec -d [-v] [-c] [-m {EXTENDED|SIMPLE|MINIMUM} ]
         [-t [-f] [-o ascファイルのパス名] ] [-h 論理ホスト名]
         [-x] ジョブ定義スクリプトファイルのパス名
```

機能

ジョブコントローラを起動して，引数に指定したジョブ定義スクリプトファイルのバッチジョブを実行します。ジョブ定義スクリプトファイルに記述するコマンドを-r オプションに直接指定して実行することもできます。

このコマンドの引数は位置パラメーターよりも前に指定してください。

引数

-d【UNIX 限定】

ジョブコントローラをデバッガモードで起動します。UNIX 環境で使用できます。

デバッガモードではメモリ上にカバレージ情報を蓄積します。run コマンドを実行するごとに，継続蓄積になります。メモリ上に蓄積したカバレージ情報は，info coverage コマンドで表示できます。

-t オプションの指定がない場合，quit コマンドでデバッガを終了すると，メモリ上に蓄積したカバレージ情報を破棄します。

このオプションを指定した場合，ジョブ定義スクリプト稼働実績情報を採取しません。

-v

バージョン情報を表示します。バッチジョブは実行しません。

-c

ジョブ定義スクリプトファイルの文法チェックをします。実施するのは文法チェックだけで，バッチジョブは実行しません。

-m {EXTENDED|SIMPLE|MINIMUM}

起動するジョブの標準出力および標準出力エラーの出力方式を指定します。出力モードについては，「3.4.4　ジョブ実行ログへの情報メッセージと警告メッセージの出力を抑止する」を参照してください。

- EXTENDED

  拡張出力モードで動作します。

- SIMPLE

  簡潔出力モードで動作します。

- MINIMUM

  最小出力モードで動作します。

このオプションを省略した場合，OUTPUT_MODE_ROOT パラメーターとOUTPUT_MODE_CHILD パラメーターの指定に従います。

この指定はadshexec コマンドで起動するジョブだけに有効で，そのジョブからさらに起動されるジョブには継承されません。別途起動するジョブに対しては，その際実行されるadshexec コマンドに-m オプションを再度指定する必要があります。

ルートジョブが簡潔出力モードまたは最小出力モードで動作する場合，-s オプションを指定するか，OUTPUT_STDOUT 環境設定パラメーターでSPOOL を指定しても，標準出力はスプールのファイルにリダイレクトしません。

-r オプションを使用する場合に，ジョブ実行ログが出力結果に混在しないようにするには，-m SIMPLE または-m MINIMUM を同時に指定してください。

-t

カバレージ情報を蓄積してバッチジョブを実行します。蓄積したカバレージ情報は，コマンド終了時にasc ファイルに出力します。

UNIX でデバッガモードで起動する場合，-t オプションを指定しないときは，メモリ上だけでカバレージ情報を採取します。カバレージ情報は，デバッガのinfo coverage コマンドで表示できます。quit コマンドでデバッガを終了すると，採取したカバレージ情報を破棄します。

-f

すでに採取しているカバレージ情報がある場合，実行するジョブ定義スクリプトファイルとバックアップ情報に差分があったときに，asc ファイルを上書きするオプションを指定します。-f オプションを指定するには，-t オプションの指定が必要です。

-f オプションを指定した場合

バックアップ情報を破棄して，新規にカバレージ情報を採取します。

採取しているカバレージ情報がない場合，新規に採取したカバレージ情報をasc ファイルに出力します。

-f オプションを指定しない場合

ジョブ定義スクリプトファイルを実行しないで，コマンドエラーとなります。asc ファイルは更新しません。

-o *asc ファイルのパス名*

Windows の場合　～<*パス名*>((1～229 バイト))

UNIX の場合　～<*パス名*>((1～1,005 バイト))

カバレージ情報の蓄積時に，asc ファイルのパス名を任意に変更できます。-o オプションを指定するには，-t オプションの指定が必要です。

このオプションを省略すると，adshexec コマンドを実行したときのカレントディレクトリのasc ファイルを指定したと解釈されます。asc ファイルのファイル名を次に示します。

拡張子を除いたジョブ定義スクリプト名＋_（アンダースコア）＋*ユーザー名*＋「.asc」

例えば，次の条件でadshexec コマンドを実行したとします。

- adshexec コマンドを実行したときのカレントディレクトリ：/home/user1/test
- ユーザー名：user1
- ジョブ定義スクリプト名：script1.ash

このとき，-o オプションを指定しない場合のasc ファイル名は次のようになります。

/home/user1/test/script1_user1.asc

-h *論理ホスト名*　～<*論理ホスト名*>((1～255 バイト))

論理ホストで実行する場合の論理ホスト名を指定します。Windows の場合，196 バイトを超える論理ホスト名は指定できません。また，論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

*論理ホスト名*の部分に空文字列を指定すると，JP1_HOSTNAME 環境変数の設定値が使用されます。JP1_HOSTNAME 環境変数が設定されていない場合は，KNAX0220-E メッセージを出力して終了します。JP1_HOSTNAME 環境変数については，マニュアル「JP1/Base 運用ガイド」を参照してください。

物理ホストで実行する場合は，このオプションを指定しないでください。

-s {SPOOL|PARENT}

ルートジョブの標準出力の出力先を指定します。子孫ジョブは，このオプションにPARENT が指定されたものとして動作します。

このオプションを省略した場合，ルートジョブはパラメーターOUTPUT_STDOUTの指定に従って動作します。ルートジョブが簡潔出力モードまたは最小出力モードで動作する場合は，このオプションの指定に関係なく，標準出力はスプールのファイルにリダイレクトしません。

- SPOOL

  ルートジョブの標準出力をスプール内のファイルに出力します。

- PARENT

  ルートジョブの標準出力を，プロセス起動時に親プロセスから継承した出力先に出力します。親プロセスで出力先をリダイレクトしていない場合は，親プロセスと同じ出力先に出力します。

-x

シェルオプションxtraceを有効にします。

なお，ジョブ定義スクリプト中で「set +x」や「set +o xtrace」を実行することで，シェルオプションxtraceを無効にできます。

-r *コマンドライン*

ジョブで実行する内容をコマンドラインに指定します。*コマンドライン*にはシェル標準コマンドやUNIX互換コマンドなど，ジョブ定義スクリプトファイルに記述できるコマンドを指定できます。*コマンドライン*は，ジョブ定義スクリプトの1行の長さが上限（8,191バイト）を超えない長さで指定してください。

-vオプションと同時に指定した場合，-vオプションが有効となります。

【UNIXの場合】

-c，-d，または-tオプションと同時には指定できません。同時に指定した場合はエラーとなります。

【WINDOWSの場合】

-cおよび-tオプションと同時には指定できません。同時に指定するとエラーとなります。

詳細については，「3.2.4　ジョブで実行する内容をコマンドラインに指定する」を参照してください。

ジョブ定義スクリプトファイルのパス名

Windowsの場合　～<*パス名*>((1～247バイト))

UNIXの場合　～<*パス名*>((1～1,023バイト))

ジョブ定義スクリプトファイルのパス名を指定します。

実行時パラメーター　～<*任意文字列*>((1～1,022バイト))

ジョブ定義スクリプトの位置パラメーターに格納する値を指定します。スペースを実行時パラメーターとして指定する場合は，その文字列を”（ダブルクォーテーション）で囲んでください。

終了コード

契機	終了コード	
	-cオプションがない場合	-cオプションがある場合
通常実行でexitコマンドまたは関数外のreturnコマンドを実行した	コマンドに指定した終了コード	－

契機	終了コード	
	-c オプションがない場合	-c オプションがある場合
通常実行でexec コマンドの引数に外部コマンドを指定して実行した	引数に指定した外部コマンドの終了コード	－
通常実行でジョブ定義スクリプトファイルの末尾までジョブ定義スクリプトを実行した	最後に実行した，シェル標準コマンドまたはスクリプト拡張コマンドの終了コード	－
デバッガモードでジョブコントローラにエラーがない	0	－
ジョブ定義スクリプトファイル，および初期設定スクリプトファイルに文法エラーがない	ジョブ定義スクリプトを実行し，次の終了コードになる • コマンドに指定した終了コード • 最後に実行した，シェル標準コマンドまたはスクリプト拡張コマンドの終了コード	0
ジョブ定義スクリプトファイルに文法エラーがある	1，または環境変数ADSH_JOBRC_FATAL に設定した値	1，または環境変数ADSH_JOBRC_FATAL に設定した値
初期設定スクリプトファイルに文法エラーがある	1，または環境変数ADSH_JOBRC_FATAL に設定した値	－
環境ファイル読み込みエラーなど，ジョブ定義スクリプト実行中のエラーを除くジョブコントローラのエラーが発生した	1，または環境変数ADSH_JOBRC_FATAL に設定した値	1，または環境変数ADSH_JOBRC_FATAL に設定した値
初期設定スクリプトファイルが存在しない。またはジョブコントローラが初期設定スクリプトファイルを実行するために必要な権限が初期設定スクリプトファイルに割り当てられていない。	1，または環境変数ADSH_JOBRC_FATAL に設定した値	－
JP1/AJS の強制終了操作を実行した。※【UNIX 限定】	143	143
ジョブコントローラプロセスがシグナルを受信して終了した【UNIX 限定】	128 ＋*シグナル番号*	128 ＋*シグナル番号*
ジョブコントローラプロセスが，JP1/AJS や Windows のタスクマネージャーなど外部から強制終了された【Windows 限定】	ジョブコントローラを強制終了したプログラムが指定した終了コード	ジョブコントローラを強制終了したプログラムが指定した終了コード
OS に起因する要因でジョブコントローラの起動が失敗した【Windows 限定】	1～3	1～3
環境変数ADSH_JOBRC_FATAL の解析エラーが発生した	255	255
環境変数ADSH_LINK_SUPPORT に不当な値が指定された【Windows 限定】	255	255

（凡例）

－：ジョブ定義スクリプトを実行しないため該当しません。

注※

UNIX ジョブの「コマンド文」でジョブを定義している場合，JP1/AJS で参照できるジョブの終了コードは-1 になります。

注意事項

- 同じオプションを複数指定した場合，最後の指定が有効になります。
- オプション指定が-v や-c と同時指定の場合，優先度が高い順に有効になります。優先順位は，-v，-c，その他のオプションの順になります。優先度が低いものは無視されます。

  例

  -d オプションは無視され，-v だけが有効になります。

```
$ adshexec -v -d MyShell.ash
```

- 次の場合に，-o オプションを指定しないでカバレージ情報を蓄積したときは，asc ファイルのファイル名が重複します。asc ファイル名が重複しないようにジョブ定義スクリプトのファイル名またはasc ファイルのファイル名を変更してください。
  - 拡張子だけ異なるファイル名の，複数のジョブ定義スクリプトのカバレージ情報を蓄積した場合

    例：sc.1 およびsc.2
  - 別ディレクトリにある同じファイル名のジョブ定義スクリプトのカバレージ情報を蓄積した場合

    例：/dir1/sc1 および/dir2/sc1
- ファイル名として.（ドット）から始まるファイル名を指定しないでください。
- ファイル名に予約デバイス名（CON やAUX，NUL など）は使用しないでください。【Windows 限定】
- ファイル名に NTFS のストリームは使用しないでください。【Windows 限定】
- -d オプションを指定した場合，実行時パラメーターは指定できません。実行時パラメーターは，run コマンドの引数に指定してください。
- asc ファイルのアクセス権限は次のとおりに設定されます。
  - ファイルの所有者（作成者）には，umask の指定に関係なく，r（読み込み）またはw（書き込み）のアクセス権限が与えられます。グループ，一般のアクセス権限は，コマンドが起動されたときの umask の指定に従って設定されます。【UNIX 限定】
  - asc ファイルには，実行ユーザー自身に対して原則，フルコントロールのアクセス許可を与えられます。しかし，実際のファイルのアクセス許可は，Windows のアクセス許可の継承（上位ディレクトリでのアクセス許可の継承）の影響を受けます。また，そのほかのユーザーに対するアクセス許可は，Windows のアクセス許可の継承（上位ディレクトリでのアクセス許可の継承）に従います。【Windows 限定】
- 子プロセスとして生成したadshexec コマンドにファイルディスクリプタが引き継がれないで，クローズされた状態になります。例えば，子プロセスのジョブ定義スクリプト内で，親プロセスがオープンし

ていたファイルディスクリプタに対して再度オープンしないで入出力を行おうとするとエラーになります。標準出力と標準エラー出力は再度オープンされた状態になります。【Windows 限定】

使用例

- 文法チェックモードでジョブコントローラを起動します。

```
adshexec -c /home/user/shell/JOB.ash
```

- デバッガモードでジョブコントローラを起動します。

```
adshexec -d /home/user/shell/JOB.ash
```

- バッチジョブは実行しないで，ジョブコントローラのバージョン情報を表示します。

```
adshexec -v
```

- ジョブ定義スクリプトの位置パラメーターに渡す実行時パラメーターを指定してジョブコントローラを起動します。

```
adshexec /home/user/shell/JOB.ash parm1 parm2
```

- カバレージ情報を採取します。

```
adshexec -t /home/user/shell/JOB.ash
```

- ジョブ定義スクリプトファイルの内容が異なる場合，これまでに採取したカバレージ情報を破棄し，新規にカバレージ情報を採取します。

```
adshexec -t -f /home/user/shell/JOB.ash
```

- 採取したカバレージ情報を格納するasc ファイルを/home/user/JOB.asc とします。

```
adshexec -t -o /home/user/JOB.asc /home/user/shell/JOB.ash
```

- シェルオプションxtrace を有効にしてジョブコントローラを起動します。

```
adshexec -x /home/user/shell/JOB.ash
```

- -r オプションのコマンドラインにジョブで実行するコマンドを指定してジョブコントローラを起動します。

```
adshexec -r "ls *"
```

- -r オプションのコマンドラインで参照する位置パラメーターを実行時パラメーターに指定してジョブコントローラを起動します。

  ジョブ定義スクリプトファイルに記述する場合の例を次に示します。コマンドラインに位置パラメーターを指定するため，シングルクォーテーションで囲む必要があります。

```
adshexec -r 'cat $1 | grep $2' file.txt abc
```

- ファイルパスを扱うコマンドを-r オプションのコマンドラインに指定してジョブコントローラを起動します。

ジョブ定義スクリプトファイルに記述する場合の例を次に示します。

```
adshexec -r 'cat "C:¥¥Documents and Settings¥¥user001¥¥file.txt"'
```

8.3.8 adshfile コマンド（通常ファイルの割り当ておよび後処理を指定する）

形式

```
adshfile   [-s {step|job} ]  [-n {del|keep} ]  [-a {del|keep} ]
           [-c {exist|no} ]  ファイルパス名
```

機能

このシェル運用コマンドでは，通常ファイルの割り当て，通常ファイルの存在有無の確認および後処理を指定します。JP1/Advanced Shell のジョブコントローラで実行したジョブ定義スクリプト内に記述した場合に有効です。通常ファイルの割り当ては，64 個まで指定できます。通常ファイルの割り当て，後処理，#-adsh_file コマンドとの差異については，「5.9.1　通常ファイルの割り当ておよび後処理をする」を参照してください。

このコマンドで割り当てた通常ファイルは，#-adsh_file コマンドで割り当てた通常ファイルとは別に管理され，後処理は adshfile コマンド，#-adsh_file コマンドの順に実行されます。そのため，両方のコマンドで同じファイルを割り当てると，ファイルの後処理が二重に実行されることになり，エラーが発生する場合があるので注意してください。

このコマンドは次の点に注意して実行してください。

- コマンドの非同期実行はしないでください。
- このコマンドは別プロセスで実行しないでください。
- スプールジョブ作成抑止機能を使用した場合，このコマンドは実行できません。

同じオプションを重複して指定した場合，最後の指定が有効になります。

引数

-s {step|job}

ファイルの後処理の契機を指定します。

- step
  ファイルの後処理をジョブステップ終了時に実行します。
- job
  ファイルの後処理をジョブ終了時に実行します。

コマンドの発行はこのオプションの指定に関係なく，ジョブとジョブステップ内のどちらでもできますが，登録したファイルの後処理は次のジョブステップの終了時またはジョブ終了時に実行されます。また，step を指定してファイルを登録したあとに，ジョブステップ終了のタイミングがなかった場合は，後処理はジョブ終了時に実行されます。

子孫ジョブでこのオプションを指定した場合，その子孫ジョブ内でこれらの動作が実行されます。

-n {del|keep}

該当するジョブステップまたはジョブが正常終了した場合の後処理を指定します。

- del
  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除します。
- keep
  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除しません。

-a {del|keep}

該当するジョブステップまたはジョブがエラー終了した場合の後処理を指定します。

- del
  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除します。
- keep
  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除しません。

adshfile コマンドのファイル割り当て処理中にエラーとなった場合，次の動作となります。

- 該当する adshfile コマンドで指定した通常ファイルに対する後処理は実行されない
- 事前に adshfile コマンドで割り当てた通常ファイルに対しては，それぞれの adshfile コマンドの-a オプションに従った後処理が実行される

-c {exist|no}

ファイルパスの存在をチェックするかどうかを指定します。

- exist
  ファイルパスが存在するかチェックします。
  ファイルパスが存在する場合だけ後処理を登録します。
  ファイルパスが存在しない場合，コマンドはエラー終了します。
- no
  ファイルパスの存在はチェックしません。

ファイルパス

Windows の場合　～<パス名>((1～247 バイト))

UNIX の場合　～<パス名>((1～1,023 バイト))

割り当てる通常ファイルのパスを指定します。

相対パスと絶対パスのどちらでも指定できますが，相対パスは絶対パスへ変換されたあとのファイルパス名が上限長を超えるとエラーになります。

終了コード

終了コード	意味
0	正常終了
99	エラー終了

注意事項

#-adsh_file コマンドとは異なり，ジョブ実行ログに割り当て結果を示すメッセージは出力されません。adshfile コマンド実行後に，障害調査などの目的で adshfile コマンドの引数に指定したファイルパス名を確認する手順は次のとおりです。

1. 環境ファイルの EVENT_COLLECT パラメーターに YES を設定し，稼働実績情報の取得を設定します。
2. ジョブ定義スクリプトで adshfile コマンドを実行します。
   稼働実績情報が採取されます。
3. adshevtout コマンドを実行し，稼働実績情報を出力します。
4. 稼働実績情報の 1 列目の"EvtName"（稼働実績情報レコードの種類を示す）部分が"command"（コマンド情報を示す）の行を参照します。

なお，ジョブ定義スクリプトの稼働実績情報の採取は，実行環境の通常モードだけで実行できます。実行環境のデバッガモードおよび開発環境では採取できません。

8.3.9 adshhk コマンド（スプールジョブを削除する）

形式

adshhk *対象リストファイル名 レポートファイル名 ログファイル名* [*日数*]

機能

*対象リストファイル名*に指定した対象リストファイルに従って，スプールジョブを削除します。実行結果は*レポートファイル名*に指定したファイルに csv 形式で出力します。また，エラーメッセージなど実行時に出力するメッセージは，*ログファイル名*に指定したファイルに出力します。

このコマンドはadshevtout コマンドと，スプールディレクトリ単位で排他制御を実施します。adshhk コマンドがスプールディレクトリのロックを確保できない場合は，KNAX4425-E メッセージを出力して，該当するスプールディレクトリの処理をスキップします。

引数

対象リストファイル名

削除対象を指定した対象リストファイルのファイル名を指定します。

対象リストファイルには，削除対象のスプールディレクトリのスプールルートディレクトリ名と日数を指定しておきます。指定した日数（adshhk コマンド実行日の前日を基点にしてカウントした日数）以前に実行したスプールジョブが，指定したスプールルートディレクトリから削除されます。

対象リストファイルはテキストファイル形式で，複数行記述できます。各行は先頭から記述し，1 行には改行コードを含み，4,095 バイト以内で記述してください。また，指定値は”（ダブルクォーテーション）で囲んでください。

対象リストファイルの形式を次に示します。

```
”スプールルートディレクトリ名”［,”日数”］
```

各項目の指定内容を次に示します。

スプールルートディレクトリ名　～<*パス名*>((1～128 バイト))

スプールジョブを削除するスプールルートディレクトリ名を記述します。フルパスで記述することを推奨します。

日数　～<*符号なし整数*>((1～999))

指定した日数（adshhk コマンド実行日の前日を基点にしてカウントした日数）以前に実行したバッチジョブのスプールジョブディレクトリを削除します。省略した場合は，adshhk コマンドで指定した*日数*になります。両方に日数の指定がない場合は，その行の指定はエラーとなり，後続行の処理をします。

「””」と指定したときは，*日数*を省略したと解釈します。

レポートファイル名

実行結果を出力するファイル名を指定します。レポートファイルは csv 形式で出力します。指定したファイルが存在しない場合は新規に作成し，すでに存在する場合はそのファイルの内容を上書きします。

レポートファイルのアクセス権限は次のように設定されます。

- Windows の場合：出力先フォルダの設定に従います。
- UNIX の場合：600

レポートファイルの出力例は，「3.9　スプールジョブを削除する」を参照してください。

ログファイル名

エラーメッセージなどを出力するファイル名を指定します。指定したファイルが存在しない場合は新規に作成し，すでに存在する場合はそのファイルの内容を上書きします。

ログファイルのアクセス権限は次のように設定されます。

- Windows の場合：出力先フォルダの設定に従います。
- UNIX の場合：600

日数　～<*符号なし整数*>((1～999))

指定した日数（adshhk コマンド実行日の前日を基点にしてカウントした日数）以前に実行したバッチジョブのスプールジョブディレクトリを削除します。この引数は，対象リストファイル名に指定した*日数*よりも優先します。省略した場合は対象リストファイル名に指定した*日数*になります。

この引数の指定を省略した場合は，必ず対象リストファイルに*日数*を指定してください。

終了コード

終了コード	意味
0	正常終了
1	エラー終了
2	スプールディレクトリがほかのプログラムで処理中のため削除に失敗 （ただし，ほかのエラーが発生していると 2 以外になる）
253	標準エラー出力でエラー発生

注意事項

- コマンドを実行したユーザーに削除権限があるスプールジョブだけが削除の対象となります。削除権限がないスプールジョブは削除の失敗をレポートします。全ユーザーのスプールジョブを削除対象にしたい場合は，すべてのスプールジョブに削除権限があるユーザーで実行してください。
- スプールジョブディレクトリの下に作成したファイルは，そのバッチジョブが作成したファイルかどうかに関係なく，削除権限があれば削除します。
- スプールジョブディレクトリの下にサブディレクトリが作成されている場合，削除に失敗することがあります。
- ジョブ実行開始の日付がわからない場合は削除しません（エラー扱いとします）。
- スプールジョブディレクトリが「*ジョブ識別子-スプールジョブ名*」または「*ジョブ識別子-*」の形式のスプールジョブだけ削除します。ジョブ識別子の後ろに「-」が付いていないスプールジョブディレクトリは，バッチジョブが実行中である場合や，「3.11.1　ジョブの強制終了の方法」に示す方法以外の手段で不当に終了させられた場合などの状態を示しているため，実際の状態に関係なく削除しません。
- 削除処理中にエラーが発生した場合，そのスプールジョブの削除処理は途中まで進んでいる可能性があります。
- レポートファイルに出力された結果はジョブ番号の順で出力されません。必要に応じてソートプログラムなどでソートしてください。
- adshevtout コマンド（ジョブ定義スクリプトの稼働実績情報の出力）で処理中のスプールディレクトリを指定した場合，スプールジョブは削除されません。
- 削除するスプールジョブはスプールジョブ管理ファイルから削除するため，削除処理中に処理を中断すると，adshhk コマンドを再実行しても削除されません。その場合，削除に失敗したスプールジョブは手作業で削除してください。

- 削除に失敗したスプールジョブを手作業で削除する場合には，削除したいスプールジョブディレクトリ，およびその下位に存在するすべてのファイルをrm -r コマンドなどで削除してください。なお，削除するときには，スプールジョブディレクトリやその下位のファイルの作成日付などを参考に，ジョブが終了していることを確認して削除してください。そのため，運用停止後に削除することを推奨します。

使用例

- 次のバッチジョブを削除します。

  /home/user001/jp1as/spool ディレクトリの 7 日以上前に実行したバッチジョブ

  /home/user999/jp1as/spool ディレクトリの 30 日以上前に実行したバッチジョブ

  事前に対象リストファイル「/home/kanrisya/hk/target」に次の内容を記述します。

```
"/home/user001/jp1as/spool","7"
"/home/user999/jp1as/spool","30"
```

  この場合，次のコマンドを実行します。レポートは/home/kanrisya/hk/result.csv ファイルに保管します。

```
adshhk /home/kanrisya/hk/target  /home/kanrisya/hk/result.csv  /home/kanrisya/hk/
result.log
```

  「3.9　スプールジョブを削除する」を参照してください。

8.3.10　adshjava コマンド（Java のバッチアプリケーションを実行する）【Windows，Linux，AIX，HP-UX 限定】

形式

uCosminexus Application Server のスケジューリング機能を使用する場合

```
adshjava  [-grp スケジュールグループ名]
           -java  [ [Javaオプション] ...]  Javaアプリケーションクラス名
            [ [mainメソッドに渡す引数] ...
```

uCosminexus Application Server のスケジューリング機能を使用しない場合

```
adshjava -srv バッチサーバ名
         -java  [ [Javaオプション] ...]  Javaアプリケーションクラス名
            [ [mainメソッドに渡す引数] ...]
```

機能

adshjava コマンドの指定に従って，uCosminexus Application Server と連携して Java のバッチアプリケーションを実行します。このコマンドの指定内容は，uCosminexus Application Server の cjexecjob コマンドの引数に指定されて cjexecjob コマンドが実行されます。ジョブの強制終了を検知したときには，adshjava コマンドが cjkilljob コマンドを実行して Java のバッチアプリケーションを自動的に停止します。

JP1/Advanced Shell のジョブが強制終了すると，Java のバッチアプリケーションも強制終了されます。

強制終了時に Java のバッチアプリケーションを自動的に強制停止できる条件を次に示します。

表 8-5　強制終了時に Java のバッチアプリケーションを自動的に強制停止できる条件

強制停止できる条件（2 回目以降を除く）	Windows	Linux，AIX，HP-UX
JP1/AJS からの強制終了	○	○
Ctrl+C	○	○
Ctrl+break	×	○
コンソールを閉じる	×	○
adshexec コマンドプロセスへの終了シグナル※1 送信	－	○
adshjava コマンドプロセスへの SIGTERM 送信	－	○
adshjava コマンドプロセスへの SIGINT 送信	－	○
adshexec のデバッガの quit コマンドで中断	－※2	－※2

（凡例）

○：強制停止できます。

×：強制停止できません。

－：該当しない

注※1

終了シグナルとは，adshexec コマンドが終了対象とするシグナルのことです。終了対象とするシグナルについては，「3.11.2　シグナル受信時の動作【UNIX 限定】」を参照してください。

注※2

quit コマンドによって，バックグラウンド実行するコマンドの強制停止はできません。

Java のバッチアプリケーションの実行方法として，uCosminexus Application Server のスケジューリング機能の使用有無を選択できます。

- スケジューリング機能を使用しない場合

  Java のバッチアプリケーションは，指定したバッチサーバで実行されます。

- スケジューリング機能を使用する場合

  Java のバッチアプリケーションは，Component Transaction Monitor によって振り分けられたバッチサーバで実行されます。この場合，Java のバッチアプリケーションのジョブ ID は次の形式で生成します。

ADSH時刻プロセスID

時刻

1970 年 1 月 1 日午前 0 時からの経過秒数とナノ秒数を 16 進数で示した 16 文字の英数字。英字は大文字。

プロセス ID

16 進数で示した 8 桁のプロセス ID（コマンドのプロセス ID)。英字は大文字。

このコマンドを実行できるユーザーは次のとおりです。詳細については，マニュアル「Cosminexus V9 アプリケーションサーバ リファレンス コマンド編」の cjexecjob コマンドまたは cjkilljob コマンドの記述を参照してください。権限がない場合，cjexecjob コマンドまたは cjkilljob コマンドがエラー終了します。

- Windows の場合
  Administrators 権限または管理者特権を持つユーザー
- Linux，AIX，HP-UX の場合
  バッチサーバを起動した Component Container 管理者，またはスーパーユーザー

引数

-grp スケジュールグループ名

uCosminexus Application Server の Component Transaction Monitor が，Java のバッチアプリケーションの実行をスケジューリングするときに割り当てるバッチサーバのグループ名を指定します。バッチサーバのグループ名は 63 バイト以内で指定します。

このオプションは，-java オプションよりも前に指定してください。 スケジュールグループ名の長さはチェックされますが，それ以外のチェックはされないで，cjexecjob コマンド，cjkilljob コマンドの引数に指定されます。

-srv バッチサーバ名

Java のバッチアプリケーションを実行する，uCosminexus Application Server のバッチサーバ名を 255 バイト以内で指定します。

このオプションは，-java オプションよりも前に指定してください。

バッチサーバ名の長さはチェックされますが，それ以外のチェックはされないで，cjexecjob コマンド，cjkilljob コマンドの引数に指定されます。

-java

このオプション以降の指定内容が，そのまま uCosminexus Application Server の cjexecjob コマンドに渡されます。

このオプション以降に指定した内容は，1 つも指定がない場合を除いて adshjava コマンドではエラーチェックされません。cjexecjob コマンドでのチェック処理に任せます。

Java オプション

JavaVM の起動オプションを指定します。指定方法については，マニュアル「Cosminexus V9 アプリケーションサーバ リファレンス コマンド編」を参照してください。

#-adsh_file コマンド，#-adsh_file_temp コマンド，または#-adsh_spoolfile コマンドで割り当てたファイルは，このオプションで「-D **システムプロパティ名**=${**ファイル定義名**}」を記述することで，システムプロパティ経由でファイル名を Java のバッチアプリケーションに渡せます。

Java アプリケーションクラス名

パッケージ名を含めた Java アプリケーションのクラス名を指定します。Java アプリケーションクラス名については，マニュアル「Cosminexus V9 アプリケーションサーバ リファレンス コマンド編」の cjexecjob コマンドの記述個所を参照してください。

main メソッドに渡す引数

Java アプリケーションの main メソッドに渡す引数を指定します。main メソッドに渡す引数については，マニュアル「Cosminexus V9 アプリケーションサーバ リファレンス コマンド編」の cjexecjob コマンドの記述個所を参照してください。

終了コード

終了コード	発生条件	意味
0※1	public static int main(String[])メソッドから返された値が 0	public static int main(String[])メソッドを使用した場合に返されます。
	System.exit()，Runtime.halt()，Runtime.exit()の引数に指定した値が 0	System.exit()，Runtime.halt()，Runtime.exit()を使用した場合に返されます。
	cjexecjob コマンドの戻り値が 0	public static void main(String[])メソッドを使用して正常終了しました。
1※2	adshjava コマンドの処理の要因で終了	adshjava コマンドが強制終了しました。
2※2	adshjava コマンドの処理の要因で終了	adshjava コマンドの処理でエラーが発生しました。 強制終了処理で cjkilljob コマンドの起動に失敗すると 1 になることがあります。
public static int main(String[])メソッドから return した値+15※1※3	public static int main(String[])メソッドから返された値が 1 以上	public static int main(String[])メソッドを使用した場合に返されます。
System.exit()，Runtime.halt()，Runtime.exit()の引数に指定した値+15※1※3	System.exit()，Runtime.halt()，Runtime.exit()の引数に指定した値が 1 以上	System.exit()，Runtime.halt()，Runtime.exit()を使用した場合に返されます。
cjexecjob コマンドの戻り値＋ 15※1※3	cjexecjob コマンドの戻り値が 1	Java のバッチアプリケーションの実行に失敗しました。または，Java のバッチアプリケーションを強制終了しました。
public static int main(String[])メソッドから return した値※1※4	public static int main(String[])メソッドから返された値が負数（Windows 限	public static int main(String[])メソッドを使用した場合に返されます。

終了コード	発生条件	意味
public static int main(String[])メソッドから return した値※1※4	定。UNIX は下 8 ビットが RC として採用される）	public static int main(String[])メソッドを使用した場合に返されます。
System.exit()，Runtime.halt()，または Runtime.exit()の引数に指定した値※1※4	System.exit()，Runtime.halt()，または Runtime.exit()の引数に指定した値が負数（Windows 限定。UNIX は下 8 ビットが RC として採用される）	System.exit()，Runtime.halt()，または Runtime.exit()を使用した場合に返されます。
17	cjexecjob コマンドの起動に失敗	cjexecjob コマンドの起動が失敗した場合に返されます。

注※1

uCosminexus Application Server の cjexecjob コマンドの戻り値です。

注※2

adshjava コマンドの終了コードです。

注※3

終了コードが 255 を超える場合は，255 に変換されます。

注※4

戻り値が負の値の場合は，ジョブコントローラによって 255 に変換されます。

注意事項

- コマンドを実行する前に，スケジューリング機能の使用有無に応じたバッチサーバを起動しておく必要があります。
- このコマンドは uCosminexus Application Server の cjexecjob コマンド，cjkilljob コマンドを使用します。そのため，cjexecjob コマンド，cjkilljob コマンドが実行できる環境で実行する必要があります。
- このコマンドをバックグラウンドで実行しても，adshexec コマンドはこのコマンドの終了を待って終了します。
- cjexecjob コマンドおよび cjkilljob コマンドに渡す引数は，スケジュールグループ名とバッチサーバ名の長さだけをチェックして，そのまま cjexecjob コマンドまたは cjkilljob コマンドに渡されます。そのため，不当な値を指定すると，cjexecjob コマンドまたは cjkilljob コマンドがエラーになります。
- Windows では，ユーザーが事前に cjexecjob コマンドおよび cjkilljob コマンドのパスを PATH 環境変数に設定しておく必要があります。
- このコマンドは，adshexec コマンドのジョブ定義スクリプトから実行してください。adshexec コマンドのジョブ定義スクリプトから実行しないと，UNIX ではジョブの強制終了時に Java のバッチアプリケーションの停止処理ができません。また，Windows では実行できません。
- -grp オプションと-srv オプションを同時に指定した場合，コマンドの解析エラーになります。

- Java のバッチアプリケーションはバッチサーバで非同期に実行します。このため，Java のバッチアプリケーションの実行結果（標準出力，標準エラー出力）は，adshjava コマンドの実行結果として参照できません。
- Java のバッチアプリケーションの作成時の注意事項については，マニュアル「Cosminexus V9 アプリケーションサーバ 機能解説 拡張編」を参照してください。
- TRAP_ACTION_SIGTERM パラメーターで TERM を指定した場合，または UNIX 版で AUTO を指定して JP1/AJS からジョブを起動した場合は，trap コマンドによる動作定義に adshjava コマンドを指定しないでください。

使用例

- スケジューリング機能を使用する場合

```
#-adsh_file INPUT ”/files/file01”
#-adsh_file_temp TMP001
#-adsh_spoolfile SYSLIST

adshjava -grp JOBGROUP -java -DINPUT=${INPUT} -DTMP001=${TMP001} -DSYSLIST=${SYSLIST}
com.hitachi.mypackage.MyBatchApp
```

この例では，#-adsh_file，#-adsh_file_temp，および#-adsh_spoolfile で割り当てたファイル名をシステムプロパティで Java のバッチアプリケーションに渡しています。#-adsh_file，#-adsh_file_temp，および#-adsh_spoolfile で割り当てるファイル名は，ファイル環境変数定義名に指定した環境変数に設定されます。

この adshjava コマンドを実行すると，次の cjexecjob コマンドが実行されます。

```
cjexecjob JOBGROUP -jobID ジョブID -DINPUT=${INPUT} -DTMP001=${TMP001} -DSYSLIST=$
{SYSLIST} com.hitachi.mypackage.MyBatchApp
```

ジョブ ID は adshjava コマンドが生成したジョブ ID です。

${**環境変数名**}はそれぞれファイル名を示します。

- スケジューリング機能を使用しない場合

```
#-adsh_file INPUT ”/files/file01”
#-adsh_file_temp TMP001
#-adsh_spoolfile SYSLIST

adshjava -srv MyBatchServer -java -DINPUT=${INPUT} -DTMP001=${TMP001} -DSYSLIST=$
{SYSLIST} com.hitachi.mypackage.MyBatchApp
```

#-adsh_file，#-adsh_file_temp，および#-adsh_spoolfile で割り当てたファイル名をシステムプロパティで Java のバッチアプリケーションに渡しています。#-adsh_file，#-adsh_file_temp，および#-adsh_spoolfile で割り当てるファイル名は，ファイル環境変数定義名に指定した環境変数に設定されます。

この adshjava コマンドを実行すると，次の cjexecjob コマンドが実行されます。

```
cjexecjob MyBatchServer -DINPUT=${INPUT} -DTMP001=${TMP001} -DSYSLIST=${SYSLIST}
com.hitachi.mypackage.MyBatchApp
```

${環境変数名}はそれぞれファイル名を示します。

8.3.11 adshlsmsg コマンド（障害発生時に，応答要求メッセージの一覧を表示する）

形式

```
adshlsmsg [-h 論理ホスト名] [-n 応答要求メッセージ番号]
```

機能

共有メモリ上にある応答待ち状態の応答要求メッセージ，および受信待ち状態の応答要求メッセージとそれに対する応答の一覧を表示します。

実行環境の場合，このコマンドは，JP1/Advanced Shell がインストールされているマシンの管理者権限を持つ root または Administrators が実行します。開発環境の場合，このコマンドは一般ユーザーでも実行できます。

引数

-h 論理ホスト名　～<論理ホスト名>((1～255 バイト))

論理ホスト環境で運用している場合，このコマンドを実行する論理ホスト名を指定します。

Windows の場合，196 バイトを超える論理ホスト名は指定できません。また，論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

-n 応答要求メッセージ番号　～<符号なし整数>((1～2147483647))

応答待ち状態の応答要求メッセージ，および受信待ち状態の応答要求メッセージとそれに対する応答を表示したい応答要求メッセージ番号を指定します。

-n だけ指定して応答要求メッセージ番号の指定を省略すると，その次に指定されたオプションが引数として扱われます。

指定を省略した場合は，共有メモリ上にある応答待ち状態の応答要求メッセージ，および受信待ち状態の応答要求メッセージとそれに対する応答がすべて表示されます。

-n オプションを複数指定した場合は，最後に指定した内容が有効になります。

出力項目

コマンド実行結果に出力されるヘッダと，各項目に出力される内容は次のとおりです。

- MESSAGE-NO

  10 桁の 10 進数値の応答要求メッセージ番号です。このコマンドの-n オプション，または adshchmsg コマンドの-n オプションで指定する値です。

- STATUS

応答要求メッセージの状態として次の内容が出力されます。

- Wait：応答待ち状態の応答要求メッセージ
- Set：受信待ち状態の応答要求メッセージ

- JOBID
  adshread コマンドを発行したジョブ定義スクリプトのジョブ識別子を示す，6 桁の整数値です。
- LINENO
  adshread コマンドを実行したジョブ定義スクリプトの行番号です。
- DATE/TIME
  応答要求メッセージを出力した時刻です（ローカルタイム）。
- MESSAGE or RESPONSE
  次の内容が出力されます。
  - msg=：応答要求メッセージ本体
  - res=：応答内容（受信待ち状態の場合だけ表示される）

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了

使用例

応答待ち状態の応答要求メッセージ，および受信待ち状態の応答要求メッセージとその一覧を表示する場合の出力例を次に示します。

```
$ adshlsmsg
 MESSAGE-NO  STATUS JOBID      LINENO DATE/TIME         MESSAGE or RESPONSE
[0000017622] [Wait] 000228         20 12/05/24 18:28:00 msg=STOP
[0000017626] [Set ] 000229        136 12/05/24 18:28:10 msg=処理を続行しますか(Y/N)
[0000017626] [Set ] 000229        136 12/05/24 18:28:10 res=Y
```

8.3.12 adshmdctl コマンド（ユーザー応答機能管理デーモンを起動および停止する）【UNIX 限定】

形式

```
adshmdctl  [-h 論理ホスト名]
           {start  [reuse] |stop|status|conftest  [環境ファイル名] |help}
```

機能

ユーザー応答機能管理デーモンを起動または停止します。ユーザー応答機能管理デーモンは，ユーザー応答機能のための共有メモリを管理します。

ユーザー応答機能の応答要求メッセージの情報は共有メモリに格納され，通常はユーザー応答機能管理デーモン停止時に解放されます。ユーザー応答機能管理デーモンが障害によって共有メモリを解放しないまま終了した場合は，このコマンドを次の手順で実行して共有メモリを解放してください。

1. このコマンドに start reuse オプションを指定して実行する
2. このコマンドに stop オプションを指定して実行し，ユーザー応答機能管理デーモンを停止する

引数

-h 論理ホスト名　～<論理ホスト名>((1～255 バイト))

論理ホスト環境で運用している場合に，このコマンドを実行させる論理ホスト名を指定します。

論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

論理ホスト名の指定を省略すると，その次に指定されたオプションが引数として扱われます。

start ［reuse］

ユーザー応答機能管理デーモンを起動します。

reuse を指定すると，ユーザー応答機能の応答要求メッセージの情報をそのまま使用します。

ユーザー応答機能の応答要求メッセージの情報は共有メモリに格納され，通常はユーザー応答機能管理デーモンの停止時に解放されます。ユーザー応答機能管理デーモンが何らかの障害で共有メモリを解放しないで終了した場合，reuse オプションを指定して起動したあと，adshmdctl コマンドに stop オプションを指定して終了することで，共有メモリを解放します。

stop

ユーザー応答機能管理デーモンを停止します。

未応答の応答要求メッセージがある場合，応答はキャンセルされます。

status

ユーザー応答機能管理デーモンの動作状態を次の終了コードで返します。

- ユーザー応答機能管理デーモンが動作している場合：0
- ユーザー応答機能管理デーモンが動作していない場合：1

conftest ［環境ファイル名］

指定した環境ファイルのパラメーターをチェックします。結果は標準出力に出力します。

環境ファイル名の指定を省略すると，システム環境ファイルが参照されます。

help

adshmdctl コマンドのヘルプを表示します。

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了

注意事項

- ユーザー応答機能管理デーモンは，root で起動してください。
- ユーザー応答機能管理デーモンの動作中は，システム環境ファイルを変更しないでください。
- adshmdctl コマンドは，環境変数 LANG に C を指定してユーザー応答機能管理デーモンを起動します。そのため，そのあと出力されるメッセージや JP1 イベントは英語で出力されます。

8.3.13 adshmsvcd コマンド（開発環境でユーザー応答機能管理サービスを登録する）【Windows 限定】

形式

```
adshmsvcd [-install [-lhostname 論理ホスト名]]
```

機能

ユーザー応答機能管理サービス（adshmsvcd）を登録します。ユーザー応答機能管理サービスは，ユーザー応答機能のための共有メモリを管理します。このコマンドは，Windows の開発環境だけで実行できます。

引数

-install

ユーザー応答機能管理サービスを登録します。

ユーザー応答機能管理サービスは，JP1/Advanced Shell のセットアップ時に自動的に登録されます。ただし，登録したレジストリ情報を消去してしまった場合は，このオプションを使って手動で再登録する必要があります。

-lhostname 論理ホスト名　～<論理ホスト名>((1～196 バイト))

論理ホスト環境で運用している場合に，このコマンドを実行する論理ホスト名を指定します。

論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了

注意事項

- オプションを指定しなかったり，指定できるオプション以外を指定して実行したりした場合でも，コマンドはエラーにならないで終了します。この場合，レジストリ情報は更新されません。

8.3.14 adshmsvce コマンド（実行環境でユーザー応答機能管理サービスを登録する）【Windows 限定】

形式

```
adshmsvce [-install [-lhostname 論理ホスト名]]
```

機能

ユーザー応答機能管理サービス（adshmsvce）を登録します。ユーザー応答機能管理サービスは，ユーザー応答機能のための共有メモリを管理します。このコマンドは，Windows の実行環境だけで実行できます。

引数

-install

ユーザー応答機能管理サービスを登録します。

ユーザー応答機能管理サービスは，JP1/Advanced Shell をセットアップしたときに自動的に登録されます。ただし，登録したレジストリ情報を消去してしまった場合は，このオプションを使って手動で再登録する必要があります。

-lhostname 論理ホスト名　～<論理ホスト名>((1～196 バイト))

論理ホスト環境で運用している場合に，このコマンドを実行する論理ホスト名を指定します。

論理ホスト名の長さは 63 バイト以下を推奨します。63 バイトを超える名称を指定すると，動作しないことがあります。

終了コード

終了コード	意味
0	正常終了

終了コード	意味
0 以外	エラー終了

注意事項

- オプションを指定しなかったり，指定できるオプション以外を指定して実行したりした場合でも，コマンドはエラーにならないで終了します。この場合，レジストリ情報は更新されません。

8.4 UNIX 互換コマンド

この節では各 UNIX 互換コマンドの指定方法を説明しています。全般的な注意事項を次に示します。

UNIX 互換コマンドで使用できる正規表現

基本の正規表現および拡張された正規表現の 2 つを使用できます。基本の正規表現が使用できるコマンドを次に示します。

- expr コマンド
- grep コマンド（-G オプションを指定した場合）
- sed コマンド（-E オプションを指定しない場合）

拡張された正規表現が使用できるコマンドを次に示します。

- awk コマンド
- egrep コマンド
- grep コマンド（-E オプションを指定した場合）
- sed コマンド（-E オプションを指定した場合）

また，正規表現には次の表に示す文字を使用できます。次の表に示す文字を正規表現に使用すると，UNIX 互換コマンドはメタキャラクタと解釈します。

表 8-6　正規表現で使用できるメタキャラクタの違い

メタキャラクタ	意味	基本の正規表現	拡張された正規表現
*	0 回以上の繰り返し	○	○
¥+	1 回以上の繰り返し	○	×
+	1 回以上の繰り返し	×	○
.	1 文字	○	○
¥?	直前にある正規表現	○	×
?	直前にある正規表現	×	○
^	行頭	○	○
$	末尾	○	○
¥|	選択	○	×
|	選択	×	○
[char-list]	範囲指定	○	○
¥(regexp¥)	一まとめ	○	×
(regexp)	一まとめ	×	○
¥{n,m¥}	*n* 回以上 *m* 回以下の繰り返し	○	×

メタキャラクタ	意味	基本の正規表現	拡張された正規表現
{n,m}	*n* 回以上 *m* 回以下の繰り返し	×	○
¥{n¥}	*n* 回	○	×
{n}	*n* 回	×	○
¥{n,¥}	*n* 回以上	○	×
{n,}	*n* 回以上	×	○

（凡例）

○：使用できます。

×：使用できません。

コマンドの使用例について

- 各 UNIX 互換コマンドの使用例は，一部を除いて Windows で実行した場合の例を示しています。
- コマンドがインストールされているディレクトリへのパスが，環境変数ADSH_OSCMD_DIR に格納されていると仮定しています。

8.4.1　awk コマンド（テキストの加工やパターン処理をする）

形式

```
awk [-F 入力フィールドセパレータ] [-v 変数名=変数値] ...
    [-f スクリプトファイルのパス名|スクリプト]
    [ [対象パス名...] | [組み込み変数名=変数値...] ] ...
```

機能

テキストファイル内の各行（以降，レコードと呼びます）に対して特定のパターンに一致する行を検索し，一致した行に対して指定した処理をします。

引数

-F 入力フィールドセパレータ

入力フィールドセパレータの値を指定します。この指定値が awk コマンドの組み込み変数 FS の値となります。

-v 変数名=変数値

変数名とその値を指定します。変数名とその値は，-f オプションに指定したスクリプトファイルまたは引数に指定したスクリプトに渡されます。複数個指定できます。同じ変数名を複数回指定した場合，最後に指定した値が設定されます。

-f スクリプトファイルのパス名

入力ファイルを検索するパターンおよびパターンと一致したレコードに対しての処理命令を記述したファイル（スクリプトファイル）のパス名を指定します。

- パス名に「-」を指定した場合は，標準入力から入力します。
- -f オプションは 19 個まで指定できます。

スクリプト

入力ファイルを検索するパターンおよびパターンと一致したレコードに対しての処理命令を，引数として指定します。

対象パス名

処理対象とするファイルのパス名を指定します。複数指定できます。

パス名を指定しない，またはパス名に「-」を指定した場合は，標準入力から入力します。なお，BEGIN パターンだけの実行の場合は，指定したファイルまたは標準入力からレコードは入力しません。

組み込み変数名=変数値

組み込み変数名およびその値を指定します。変数名とその値は，-f オプションに指定したスクリプトファイルまたは引数に指定したスクリプトに渡されます。

- 組み込み変数の説明に記述されていない名称を指定した場合は，-v オプションと同じになります。
- すべての対象パス名の前に指定した場合は，BEGIN パターン処理を除くすべてのファイル処理と END パターン処理で有効となります。
- すべての対象パス名のあとに指定した場合は，END パターン処理だけで有効となります。
- 対象パス名の間に指定した場合は，この指定以降のパス名の処理と END パターン処理で有効となります。

スクリプト（パターンおよびアクション）

awk コマンドで実行するスクリプトの記述形式を次に示します。

```
[パターン] [{ [アクション] }]
```

パターンには，入力ファイルを検索するパターンを記述します。パターンに記述できる内容については，「パターンの種類」を参照してください。アクションには，パターンと一致したレコードに対しての処理命令を記述します。

入力ファイルから 1 レコードを入力するたびにパターンと比較し，パターンに一致した場合にアクションが実行されます。パターンを省略した場合は，すべてのレコードがアクション実行の対象となります。

アクションは，パターンと一致したレコードに対する制御文や関数を使用した処理を記述します。アクションには，制御文，組み込み関数，ユーザー定義関数，変数，または演算子を指定して動作を記述できます。処理は複数の文を記述でき，各文は改行またはセミコロンで区切ります。アクションを省略した場合，レコードの内容を標準出力に出力します。なお，{}で囲んだ部分のアクションだけを省略した場合は，パターンと一致したレコードに対する処理は行われません。

コメントを記述する場合は，コメントの前に「#」を記述します。「#」以降から行末までをコメントとして扱います。

レコードとフィールド

レコードとは，入力レコードセパレータで分割した単位のことです。入力レコードセパレータの値は改行文字です。Windows の場合，［CR］＋［LF］または［LF］が改行文字と見なされます。UNIX の場合，［LF］が改行文字と見なされます。なお，UNIX の場合，入力ファイルの各レコードが［CR］＋［LF］で区切られているときは，分割されたレコードには［CR］が含まれます。

入力レコードセパレータは，組み込み変数 RS にレコードの区切りを示す 1 バイトの文字を設定することによって変更できます。文字列を指定した場合は，先頭の 1 文字を設定します。

レコードはフィールドセパレータによって，フィールドと呼ばれる単位に分割されます。フィールドセパレータのデフォルト値はスペースです。フィールドセパレータは-f オプションまたは組み込み変数 FS にフィールドの区切りを示す文字列を設定することで変更できます。

アクションには，入力情報として入力ファイルから現在読み込んでいるレコードの内容およびレコードの各フィールドの値が渡されます。レコードの内容はフィールド変数の$0 に格納されます。各フィールドの値は，レコードの最初のフィールドがフィールド変数$1，2 番目のフィールドがフィールド変数$2 というように順に格納されます。

パターンの種類

入力ファイルを検索するパターンには次の指定ができます。

文字列

フィールドまたはレコードから検索したい文字列をスラッシュ(/)で囲みます。指定する文字列には正規表現を使用できます。スラッシュ(/)自体を検索したい場合は，エスケープ文字(¥)を使用します。「/¥//」と指定します。

検索したい文字列に「hitachi」を指定する場合の例を次に示します。

```
/hitachi/{
(アクション)
}
```

関係式

関係演算子（>，>=，<，<=，==，!=）を使用し，特定のフィールドに対して比較をします。指定例を次に示します。

レコードの 2 番目のフィールドの内容が「hitachi」の場合にアクションを実行します。

```
$2 == ”hitachi”{
(アクション)
}
```

パターンの組み合わせ

複数のパターンを組み合わせてアクション実行条件を記述します。使用できる組み合わせを次の表に示します。

書式	説明
パターン1 && **パターン2**	論理積の演算子で，パターン 1 およびパターン 2 に一致するレコードをアクションの実行対象とします。
パターン1 || **パターン2**	論理和の演算子で，パターン 1 またはパターン 2 に一致するレコードをアクションの実行対象とします。
パターン1 ? **パターン2** : **パターン3**	三項演算子で，パターン 1 とパターン 2 に一致するレコードまたはパターン 3 に一致するレコードをアクションの実行対象とします。
! **パターン**	否定の演算子で，パターンに一致しないレコードをアクションの実行対象とします。
(**パターン**)	複数条件のパターンをグループ化します。
パターン1, **パターン2**	パターン 1 に一致するレコードから，パターン 2 に一致するレコードまでがアクションの実行範囲となります。なお，パターン 2 に指定したレコードがなく入力ファイルの終端に達した場合は，入力ファイルの最終レコードまでが範囲となります。ただし，複数の入力ファイルを指定した場合，次の入力ファイルでパターン 2 に一致するレコードを検索します。この書式は 50 個まで指定できます。

BEGIN

ファイル入力の開始前に実行するアクションに対するパターンです。アクションの記述を省略できません。また，ほかのパターンと組み合わせることはできません。なお，複数の入力ファイルを指定している場合は，最初のファイル入力開始前にアクションが実行されます。

END

ファイルの最後のレコードに対するアクション実行後，または exit 制御文で終了した場合に実行するアクションに対するパターンです。アクションの記述を省略できません。また，ほかのパターンと組み合わせることはできません。なお，複数の入力ファイルを指定している場合は，最後のファイルの最後のレコードに対するアクション実行後となります。

制御文

使用できる制御文を次の表に示します。if 文，while 文，for 文，do 文，break 文，continue 文，return 文の構文規則は C 言語と同じです。ただし，for 文の初期化式と再初期化式は 1 つの式だけ指定できます。

制御文	構文	内容
if 文	if (**条件式**) **処理** [else **処理**]	条件分岐します。
	if (**変数** in **配列**) **処理**[else **処理**]	変数に指定した添え字の配列要素が配列に存在するかどうかを判定します。
while 文	while (**条件式**) **処理**	条件が成立している間，繰り返します。
for 文	for (**初期化式**; **継続条件式**; **再初期化式**) **処理**	繰り返し実行します。

制御文	構文	内容
for 文	for (**変数** in **配列**) **処理**	各配列要素の添え字の値を順次取り出しながら処理します。添え字の値の取り出しは順不同です。
do 文	do **処理** while (**継続条件式**)	後判定によって条件が成立している間，繰り返します。
break 文	break	繰り返し処理を抜けます。
continue 文	continue	繰り返し処理を中断して，繰り返し処理の先頭に戻ります。
next 文	next	処理中の入力レコードに対して，この制御文以降の処理を停止し，次の入力レコードの処理を開始します。
nextfile 文	nextfile	処理中の入力ファイルに対して，この制御文以降の処理を停止し，次の入力ファイルの処理を開始します。
return 文	return [**expr**]	ユーザー定義関数を終了します。式 expr で指定した値を呼び出し元に返します。式 expr を指定しない場合は，0 がユーザー定義関数の戻り値となります。
delete 文	delete **配列**	配列を削除します。
	delete **配列**[**要素**]	配列の要素を削除します。
exit 文	exit [**expr**]	処理中のスクリプトの実行を停止します。 式 expr で指定した値をコマンドの終了コードとして返します。式 expr を指定しない場合は，0 がコマンドの終了コードとなります。式 expr で指定した値は符号付きの 4 バイトの数値として扱います。 Windows の場合，式 expr で指定した値がコマンドの終了コードとなります。UNIX の場合，式 expr で指定した値が 0～255 の範囲外のときは，値の下位 8 ビットがコマンドの終了コードとなります。JP1/Advanced Shell のジョブ定義スクリプトから実行する場合は，0～255 の範囲の値を指定してください。 Windows で JP1/Advanced Shell のジョブ定義スクリプトから実行する場合，式 expr で指定した値が 0～255 の範囲外のときは，コマンドの呼び出し元に返す終了コードは式 expr で指定した値とは異なります。JP1/Advanced Shell でのコマンドの終了コードの扱いについては「ジョブ，ジョブステップおよびコマンドの終了コード」を参照してください。

組み込み関数

使用できる組み込み関数を次に示します。

数学関数

使用できる数学関数を次の表に示します。

関数名	内容
atan2(**y**,**x**)	y/x の逆正接を返します。単位はラジアンです。引数が不足している場合は 1 を返し，警告メッセージを出力します。
cos(**x**)	x の余弦を返します。単位はラジアンです。
exp(**x**)	x の指数関数を返します。結果がオーバーフローまたはアンダーフローする場合は 1 を返し，警告メッセージを出力します。
int(**x**)	x の小数点以下を切り捨てて整数を返します。
log(**x**)	x の自然対数を返します。x が 0 または負の場合は 1 を返し，警告メッセージを出力します。
rand()	乱数 n を返します（0≦n ＜ 1 の範囲）。srand 関数でシード値を設定しない場合は，コマンドを実行するたびに同じ値を返します。
sin(**x**)	x の正弦を返します。単位はラジアンです。
sqrt(**x**)	x の平方根を返します。x が負の場合は 1 を返し，警告メッセージを出力します。
srand([**expr**])	式 expr を rand 関数用のシード値を設定して，設定したシード値を返します。expr を省略した場合は，時刻を基にしたシード値を設定します。

文字列関数

使用できる文字列関数を次の表に示します。マルチバイト文字は 1 文字として扱います。

関数名	内容
gsub(**r**, **t**[, **s**])	文字列 s 中のすべての正規表現 r を t に置き換えます。s を省略した場合，$0（レコード全体が格納されているフィールド変数）を置き換え対象とします。t に&を指定した場合は，&が一致した文字列に置き換えられます。終了コードとして，置き換えた回数を返します。
index(**s**,**t**)	文字列 s 中の文字列 t の位置を返します。文字列 t がなかった場合は 0 を返します。
length[([**s**])]	文字列 s の文字数を返します。s を指定しなかった場合は，$0（レコード全体が格納されているフィールド変数）の文字数を返します。
match(**s**, **r**)	文字列 s 中の正規表現 r が現れる位置を返します。正規表現 r がなかった場合には 0 を返します。また，RSTART 組み込み変数には正規表現 r に一致した文字列の位置が設定されます。不一致時は 0 になります。RLENGTH 組み込み変数には正規表現 r に一致した文字列の長さが設定されます。不一致時は-1 になります。
sprintf(**書式**, **式**[, ...])	書式に従って，式を整形した結果の文字列を返します。書式については出力書式の説明を参照してください。
split(**s**, **array**[, **fs**])	文字列 s をフィールドセパレータ fs によってフィールド分割し，配列 array に格納します。戻り値として配列の要素数を返します。分割した各フィールドの値は，配列 array に，array[1]，array[2]，…，array[戻り値]の順に格納されます。

関数名	内容
split(s, array[, fs])	フィールドセパレータ fs の指定を省略した場合は，フィールドセパレータとして組み込み変数 FS の値が使用されます。 フィールドセパレータ fs には文字列および正規表現を指定できます。また，フィールドセパレータ fs に文字指定なしを示す「""」を指定した場合は，1 文字ずつ分割されます。
sub(r, t[, s])	文字列 s 中に最初にあった正規表現 r を t に置き換えます。s を省略した場合，$0（レコード全体が格納されているフィールド変数）を置き換え対象とします。t に&を指定した場合は，&が一致した文字列に置き換えられます。正規表現 r があった場合は終了コードとして 1 を返します。正規表現 r がなかった場合は 0 を返します。
substr(s, m[, n])	文字列 s の m 番目の文字から最大 n 文字の部分文字列を返します。n を省略した場合は，m 番目以降のすべての文字列を返します。
tolower(str)	文字列 str 中のすべての英大文字を英小文字に変換した文字列を返します。
toupper(str)	文字列 str 中のすべての英小文字を英大文字に変換した文字列を返します。

ビット操作関数

使用できる操作関数を次の表に示します。x および y は符号付きの 4 バイトの数値として扱います。

関数名	内容
compl(x)	整数 x の 1 の補数を返します。
and(x, y)	整数 x と整数 y のビットごとの論理積を返します。
or(x, y)	整数 x と整数 y のビットごとの論理和を返します。
xor(x, y)	整数 x と整数 y のビットごとの排他的論理和を返します。
lshift(x, n)	整数 x をビット数 n 分，左にシフトした値を返します。ビットのシフトは算術シフトで行われるため，符号部分もシフト対象となります。
rshift(x, n)	整数 x をビット数 n 分，右にシフトした値を返します。ビットのシフトは算術シフトで行われるため，符号部分は右にシフトした時に補てんする符号として扱われます。

入出力関数

使用できる入出力関数を次に示します。

getline [**変数名**]

現在の入力ファイルから次のレコードを入力します。変数名が指定されている場合は変数名で指定した変数にレコードを入力し，組み込み変数の NR および FNR が設定されます。変数名の指定を省略した場合は，フィールド変数$0 にレコードを入力し，組み込み変数の NF，NR および FNR が設定されます。レコードの入力に成功すると 1 を返し，ファイルの終端に到達すると 0 を返し，エラーが発生すると-1 を返します。

getline [**変数名**] < **パス名**

パス名に指定したファイルから次のレコードを入力します。パス名は"（ダブルクォーテーション）で囲んで指定します。パス名の代わりにパス名を代入した変数の名称も指定できます。パス名に「-」を指定した場合は標準入力から入力します。

指定例を次に示します。

```
getline line < "file001.txt"
```

変数名が指定されている場合は変数名で指定した変数にレコードを入力します。変数名の指定を省略した場合は，フィールド変数$0 にレコードを入力し，組み込み変数の NF が設定されます。

指定したファイルは，getline 関数で最初にレコードを受け取るときにオープンし，awk コマンドが終了するまでオープンしたままとなります。このため，同じファイルに対して再度先頭レコードから入力を開始したい場合は，close 関数を実行する必要があります。

コマンド名 | getline [**変数名**]

コマンド名で指定したプログラムがパイプに出力したレコードを，getline 関数を使用して入力します。コマンド名は，実行するプログラムの名称とその引数の値を"（ダブルクォーテーション）で囲んで，コマンドライン形式で記述します。コマンド名の代わりにコマンド名を代入した変数の名称も指定できます。

変数名が指定されている場合は，変数名で指定した変数にレコードを入力します。変数名の指定を省略した場合は，フィールド変数$0 にレコードを入力し，組み込み変数の NF が設定されます。指定例を次に示します。

コマンドがパイプに出力した内容から 1 レコード分を入力し，フィールド変数$0 に代入します。

```
"cat -n file01.txt" | getline
```

コマンド名を変数に代入し，変数名と getline 関数を接続して実行します。

```
rtxt = "cat -n file01.txt"
rtxt | getline
```

指定したプログラムの出力を受け取るためのパイプ作成とプログラム実行は，「コマンド名 | getline [変数名]」実行時に行われます。同じコマンド名を複数回実行する場合，パイプ作成とプログラム実行が行われるのは最初に指定したコマンド名の実行のときだけです。作成されたパイプは awk コマンドが終了するまで存在します。このため，コマンド名で指定したプログラムを再実行したい場合は，close 関数を実行する必要があります。次に例を示します。

```
"cat -n file01.txt" | getline rec        →1.
"cat -n file01.txt" | getline rec        →2.
close("cat -n file01.txt")               →3.
"cat -n file01.txt" | getline rec        →4.
```

パイプの作成および cat コマンドの実行が行われ，cat コマンドがパイプに出力した 1 番目のレコードの内容が変数 rec に格納されます。

cat コマンドがパイプに出力した 2 番目のレコードの内容が変数 rec に格納されます。

パイプが閉じられます。

パイプの作成および cat コマンドの実行が行われ，cat コマンドがパイプに出力した 1 番目のレコードの内容が変数 rec に格納されます。なお，コマンド名の記述内容が print 関数および printf 関数で指定するコマンド名の記述内容と同じでも別のコマンド名の記述と見なされます。

print [式[, ...]]

式を標準出力に出力します。式を省略した場合は，現在の入力レコードを標準出力に出力します。式をコンマ（,）で区切って複数指定した場合，各式は組み込み変数 OFS の値で区切られます。出力レコードの終端には組み込み変数 ORS の値が出力されます。

print [式[, ...]] > パス名

式をパス名で指定したファイルに出力します。パス名は"（ダブルクォーテーション）で囲んで指定します。パス名の代わりにパス名を代入した変数の名称も指定できます。式を省略した場合は，現在の入力レコードをパス名で指定したファイルに出力します。式をコンマ（,）で区切って複数指定した場合，各式は組み込み変数 OFS の値で区切られます。出力レコードの終端には組み込み変数 ORS の値が出力されます。既存ファイルに追加書きで出力する場合は「>>パス名」で指定します。

指定したファイルは，print 関数で最初に出力するときにオープンし，awk コマンドが終了するまでオープンしたままとなります。このため，同じファイルに対して，ファイルの先頭から出力したい場合は close 関数を実行する必要があります。

print [式[, ...]] | コマンド名

式をパイプに出力し，コマンド名で指定したプログラムに渡します。

コマンド名は，実行するプログラムの名称とその引数の値を"（ダブルクォーテーション）で囲んで，コマンドライン形式で記述します。コマンド名の代わりにコマンド名を代入した変数の名称も指定できます。

式を省略した場合は，現在の入力レコードをパイプに出力します。式をコンマ（,）で区切って複数指定した場合，各式は組み込み変数 OFS の値で区切られます。出力レコードの終端には組み込み変数 ORS の値が出力されます。

指定したプログラムへ出力結果を渡すためのパイプ作成とプログラム実行は，「print [式[, ...]] | コマンド名」実行時に行われます。同じコマンド名を複数回実行する場合，パイプ作成とプログラム実行が行われるのは最初に指定したコマンド名の実行のときだけです。作成されたパイプは awk コマンドが終了するまで存在します。このため，コマンド名で指定したプログラムを再実行したい場合は，close 関数を実行する必要があります。コマンド名の記述内容が getline 関数で指定するコマンド名の記述内容と同じでも別のコマンド名の記述と見なされます。

printf 書式[,式[, ...]]

書式に従って標準出力に出力します。書式については出力書式の説明を参照してください。その他の内容は「print [式[, ...]]」を参照してください。

Windows の場合，改行文字を表す「¥n」を指定したとき，出力先には［CR］＋［LF］で出力されます。

printf 書式[,式[, ...]] >パス名

書式に従ってファイルに出力します。書式については出力書式の説明を参照してください。その他の内容は「print [式[, ...]] >パス名」を参照してください。

printf 書式[,式[, ...]] | コマンド名

書式に従ってパイプに出力します。書式については出力書式の説明を参照してください。その他の内容は「print [式[, ...]] | コマンド名」を参照してください。コマンド名の内容が print 関数に指定したコマンド名と同じ場合は，同じプログラムの実行に対して出力します。

close(**パス名|コマンド名**)

getline 関数，print 関数，printf 関数で使用したファイルまたは getline 関数，print 関数，printf 関数でコマンド名実行時に作成されたパイプをクローズします。

クローズに成功した場合は 0 を返します。失敗した場合は，ファイルのときは OS の close 関数の戻り値，パイプのときは OS の pclose 関数の戻り値を返します。

引数には getline 関数，print 関数，printf 関数で指定したパス名またはコマンド名を指定します。パス名は"（ダブルクォーテーション）で囲んで指定します。コマンド名は，実行するプログラムの名称とその引数の値を"（ダブルクォーテーション）で囲んで，コマンドライン形式で記述します。また，getline 関数，print 関数，printf 関数で指定したパス名やコマンド名が格納されている変数も指定できます。なお，close 関数に記述するパス名やコマンド名は，getline 関数，print 関数，printf 関数に記述したパス名やコマンド名の文字列と文字数も含めて同じにしてください。パス名やコマンド名の文字列が異なる場合，別のパス名または別のコマンド名と見なされます。1 つの close 関数に指定したコマンド名が getline 関数，print 関数または printf 関数で指定したコマンド名と同じ場合は，getline 関数で作成されたパイプと print 関数または printf 関数で作成されたパイプの両方がクローズされます。指定例を次に示します。

```
print ”hitachi”  | ”cat -n”
close(”cat -n”)
```

fflush([**パス名|コマンド名**])

getline 関数，print 関数，printf 関数で使用したファイルまたは getline 関数，print 関数，printf 関数でプログラム実行時に作成されたパイプをフラッシュします。フラッシュに成功した場合は 0 を返します。失敗した場合は，OS の fflush 関数の戻り値を返します。

引数には getline 関数，print 関数，printf 関数で指定したパス名またはコマンド名を指定します。パス名は"（ダブルクォーテーション）で囲んで指定します。コマンド名は，実行するプログラムの名称とその引数の値を"（ダブルクォーテーション）で囲んで，コマンドライン形式で記述します。また，getline 関数，print 関数，printf 関数で指定したパス名やコマンド名が格納されている変数も指定できます。なお，fflush 関数に記述するパス名やコマンド名は，getline 関数，print 関数，printf 関数に記述したパス名やコマンド名の文字列と文字数も含めて同じにしてください。パス名やコマンド名の文字列が異なる場合，別のパス名または別のコマンド名と見なされます。

引数の指定を省略した場合は，すべてのファイルとパイプをフラッシュします。

汎用関数

使用できる汎用関数を次の表に示します。

関数名	内容
system(コマンド名)	コマンド名に指定したプログラムを実行し，実行したプログラムのステータスを返します。 コマンド名は，実行するプログラムの名称とその引数の値を"（ダブルクォーテーション）で囲んで，コマンドライン形式で記述します。コマンド名の代わりにコマンド名を代入した変数の名称も指定できます。 UNIX の場合，実行したプログラムのステータスは，プログラムが返す終了コードの下位 8 ビットの値となります。

ユーザー定義関数

組み込み関数以外に独自の関数を定義できます。構文を次に示します。

```
function|func name ( [param [,…] ] )  { statements }
```

関数名 name に使用できる文字は，英数字および_（アンダースコア）だけです。また，先頭文字は数字以外でなければなりません。

関数の引数 param にはユーザー定義の変数または配列の名称を指定できます。関数へ渡す引数は，ユーザー定義の変数は値渡しで，配列は参照渡しで行われます。

関数定義で指定した引数の数と，関数呼び出し時に指定した引数の数が異なっていても解析エラーにはなりません。ただし，関数呼び出し時に指定した引数の数が関数定義で指定した引数の数よりも多い場合は，警告メッセージが出力されます。関数の定義で指定した引数は，ローカル変数としますが，関数呼び出し時に指定した引数の数が関数定義で指定した引数の数よりも多い場合は，関数呼び出し時の余分な引数はグローバル変数と見なされます。

関数の定義で指定できる引数の数は 50 個までです。また，関数呼び出し時に指定できる引数の数も 50 個までです。なお，引数の数が 50 個以内かどうかは，関数の呼び出し時にチェックされます。

変数

スクリプトで使用する変数にはユーザー定義変数，フィールド変数，組み込み変数および配列があります。ユーザー定義変数と配列はスクリプト内で最初に使用したときに生成されます。なお，初期化されていない（演算や代入などの式で使用されていない）変数の初期値は，数値としては 0，文字列値としては NULL が格納されます。

変数の値の型は，変数が使用されるごとに，使用方法に合わせて数値または文字列に変化します。ただし，文字列が数字以外の場合は 0 の数値とします。例えば，次に示す 2 つの print 関数の出力結果は両方とも 7 が出力されます。

```
x = ”3” + ”4”
y = 3 + 4
print x
print y
```

変数ごとに説明します。

- ユーザー定義変数

  変数名に使用できる文字は，英数字と_（アンダースコア）だけです。先頭文字は数字以外でなければなりません。

- 配列

  配列名は，変数名のあとに[ ]（角括弧）で添え字を囲んで表します。添え字には数字だけでなく"（ダブルクォーテーション）で囲まれた文字列も使用できます。また，添え字をコンマで区切って複数記述し，多次元配列を表せます。

awk コマンドの多次元配列は，各次元の添え字の値を組み込み変数 SUBSEP の値で連結し，1 つの文字列の添え字として扱います。多次元配列は内部的には一次元配列として処理されます。例えば，組み込み変数 SUBSEP の値を「#」に変更したあとに配列を作成した場合，2 つの print 関数は同じ値を出力します。指定例を次に示します。print 関数の出力結果は両方とも「日立」になります。

```
SUBSEP = ”#”
arry[”あ”, ”か”, ”さ”] = ”日立”
print arry[”あ”, ”か”, ”さ”]
print arry[”あ#か#さ”]
```

- フィールド変数

入力レコードの内容を参照するために，次のフィールド変数があります。

変数名	内容
$0	入力ファイルから現在読み込んでいるレコードの内容を設定します。
$1,$2,...	現在読み込んでいるレコードの内容を順に設定します。 組み込み変数 FS の値によって分割された各フィールド値が，レコードの最初のフィールドに$1 変数，2 番目のフィールドに$2 変数というように設定されます。
$変数名	変数にフィールド番号を設定することで，$0 や$1 など直接フィールド番号を記述した場合と同じようにフィールドを参照できます。※ 関数や制御文で使用する場合，使用する前に変数にフィールド番号を設定する必要があります。
$(式)	フィールド番号を求める式（変数名+1 など）を式に指定することで，$0 や$1 など直接フィールド番号を記述した場合と同じようにフィールドを参照できます。

注※

次の場合，「print $a」および「print $1」は同じ内容が出力されます。

```
a = 1
print $a
print $1
```

- 組み込み変数

次の組み込み変数があります。

変数名	内容
ARGC	コマンドライン引数の個数です。オプション値およびスクリプト指定値は含みません。スクリプトおよび-v オプションで変更できます。-v オプションで ARGC に 0 を設定した場合，引数に対象パス名が指定されていない状態になります。
ARGV	コマンドライン引数の配列です。スクリプトで変更できます。引数に指定した対象パス名が格納されている要素に NULL を設定すると，入力ファイルからレコードの入力が行われません。-v オプションで指定した値は上書きされます。
CONVFMT※	数値を変換するときに使用する変換書式です。デフォルトは%.6g です。数値に小数部分がある場合に有効となります。
ENVIRON	実行時の環境変数の配列です。添え字は環境変数名です。環境変数名は"（ダブルクォーテーション）で囲んで指定します。環境変数名の代わりに環境変数名を代入した変数の名称も指定できます。

変数名	内容
FILENAME	現在の入力ファイル名です。標準入力からの入力の場合はファイル名に「-」が設定されます。
FNR	現在の入力ファイルの入力レコード数です。対象パス名で指定したファイルから 1 レコードを入力するたびに更新されます。また，「getline [変数名]」でレコードを入力するときにも更新されます。 複数の対象パス名を指定した場合，次の入力ファイルから入力を開始するときに 0 で初期化されます。
FS	フィールドセパレータです。デフォルトは 1 バイトのスペースです。正規表現を使用できます。フィールドセパレータを変更する場合，指定する文字列の長さは 99 バイト以下にしてください。また，フィールドセパレータ値が 1 バイトのスペースの場合は，1 バイトのスペースとタブ(¥t)によってフィールド分割されます。フィールドセパレータに値を設定しない場合は，1 文字ずつフィールド分割されます。 複数の文字を指定する場合は，正規表現となるため，¥はエスケープ文字として扱われます。-F オプションと同時に使用した場合は，この変数の値が優先されます。
NF	現在の入力レコードのフィールド数です。対象パス名で指定したファイルからレコードを入力すると格納されます。また，getline 関数でフィールド変数$0 にレコードを入力したときにも格納されます。 $NF と記述すると，現在の入力レコードの最終フィールドの値を参照できます。
NR	現在までに読み込んだ入力レコード数です。対象パス名で指定したファイルから 1 レコードを入力するたびに更新されます。また，「getline [変数名]」で次のレコードを入力するときにも更新されます。 複数の対象パス名を指定した場合は，レコードの入力が終了したファイルのレコード数も含まれます。
OFMT※	数値の出力書式です。デフォルトは%.6g です。数値に小数部分がある場合に有効となります。
OFS	出力フィールドセパレータです。デフォルトは 1 バイトのスペースです。
ORS	出力レコードセパレータです。デフォルトは改行文字（¥n）です。Windows の場合，改行文字（¥n）は［CR］＋［LF］で出力されます。
RLENGTH	match 関数で一致した文字列の長さです。一致しなかった場合は-1 が設定されます。
RS	入力レコードセパレータです。デフォルトは改行文字です。値を変更する場合，1 バイトの文字だけ設定できます。文字列やマルチバイト文字を指定した場合は，先頭 1 バイトを使用します。改行文字（¥n）を設定した場合，Windows は，入力ファイル中の［CR］＋［LF］または［LF］が対象となります。UNIX は，入力ファイル中の［LF］が対象となります。改行文字以外の値を設定した場合，Windows は，入力レコードに含まれる改行文字は［LF］です。UNIX は，入力ファイル中の改行文字が［CR］＋［LF］の場合は［CR］も含まれます。
RSTART	match 関数で一致した文字列の開始位置です。一致しなかった場合は 0 が設定されます。
SUBSEP	多次元配列のセパレータです。デフォルトは 0x1C です。

注※

指定できる変換指定子は，f，e，g，E，G だけです。それ以外の変換指定子を指定しないでください。

演算子

使用できる演算子を優先順位の低い順に次の表に示します。式の中で同じ優先順位の演算子は，式の左側に記述された演算子の優先順位が高くなります。

演算子	説明
=，+=，-=，*=，/=，%=，^=，**=	代入演算子です。
?:	三項演算子です。
||	論理 OR です。
&&	論理 AND です。
~，!~	正規表現の一致(~)と不一致(!~)です。
<，<=，>，>=，!=，==	関係演算子です。
スペース	文字列連結です。
+，-	加算および減算です。
*，/，%	乗算，除算および剰余です。
+，-，!	単項および論理否定です。
^，**	累乗です。
++，--	インクリメントおよびデクリメントです。

出力書式

printf 関数および sprintf 関数で，変換指定を示す%と共に指定する変換指定子を次の表に示します。

文字	説明
c	1 バイト文字です。
s	文字列です。
d	符号付き 10 進整数です。
i	
o	符号なし 8 進整数です。
x	符号なし 16 進整数です。10～15 の値には「abcdef」を使用します。
X	符号なし 16 進整数です。10～15 の値には「ABCDEF」を使用します。
u	符号なし 10 進整数です。
f	浮動小数点数です。[-]dddd.dddd 形式に変換します。
e	浮動小数点数です。[-]d.dddde[+-]dd[d]形式に変換します。
g	変換指定子 e または f で出力される符号付きの値のうち，指定された値および精度を表現できる短い方の書式です。末尾の 0 は出力されません。
E	浮動小数点数です。[-]d.ddddE[+-]dd[d]形式に変換します。
G	変換指定子 E または f で出力される符号付きの値のうち，指定された値および精度を表現できる短い方の書式です。末尾の 0 は出力されません。

文字	説明
%	%の文字です。

エスケープ文字

次の個所にはエスケープ文字を使用できます。

- -F オプションの入力フィールドセパレータ
- -v オプションの変数値
- 引数で指定する組み込み変数の変数値
- パターンおよび変数への代入などで指定する"（ダブルクォーテーション）で囲まれた文字列

使用できるエスケープ文字を次の表に示します。

エスケープ文字	意味
¥a	アラート文字（ベル）
¥b	バックスペース文字
¥f	フォームフィード文字（改ページ）
¥n	改行文字
¥r	復帰文字
¥t	タブ文字
¥v	垂直タブ文字
¥d,¥dd,¥ddd	1～3 桁の 8 進数で表された文字※1。0 を意味する数値は指定できません。
¥xhex	16 進値で表された文字（0～9，a～f，A～F）※1※2。0 を意味する数値は指定できません。
¥c	任意のリテラル文字（「¥"」の場合「"」）
¥¥	1 つのバックスラッシュ文字

注※1

スラッシュ（/）で囲んだパターンおよび正規表現に指定する場合，実行時の文字コード種別によっては指定できない値があります。

文字コード種別ごとに指定できる値を次の表に 16 進数値で示します。なお，次の値以外を指定した場合はエラー終了します。

文字コード種別	指定できる値（16 進数値）
シフト JIS	0x01-0x80，0xA0-0xDF，0xFD-0xFF
UTF-8	0x01-0xBF，0xFE-0xFF
EUC	0x01-0x8D，0x90-0xA0，0xFF

文字コード種別	指定できる値（16 進数値）
C	0x01-0xFF

注※2

ダブルクォーテーション（"）で囲んだ文字列に¥xhex を指定する場合，「¥x」から 16 進表記外の文字が出てくる前までの文字を 16 進表記文字とします。16 進表記文字が 98 文字を超える場合は 98 文字までとなります。ただし，16 進表記文字が 2 文字を超える場合は，16 進表記文字から 16 進値の変換結果は保証されません。

なお，-v オプションの変数値や，引数で指定する組み込み変数の変数値に指定した¥は，エスケープ文字として扱われます（シングルクォーテーションで囲んでいるケースを除く）。そのため，パス名を指定するときは注意が必要です。例を次に示します。

（例 1）正しいパス名（d:¥a¥b¥c）を awk コマンドのスクリプトに渡せない例

この例では，¥がエスケープ文字として処理されて¥が削除された結果，VAR001 には c:¥a¥b¥c が設定されます。

その後，awk コマンドの変数に設定される際には再び¥がエスケープ文字として処理され，最終的に VAR001 には c:abc が設定されます。

```
CCC01=”c:¥¥a¥¥b¥¥c”
awk -v VAR001=”${CCC01}” -f prog01.awk
```

同様に次の 2 つの例も正しいパス名を渡せません。

```
awk -v VAR001=c:¥¥a¥¥b¥¥c -f prog01.awk
awk -v VAR001=’c:¥a¥b¥c’ -f prog01.awk
```

（例 2）正しいパス名（d:¥a¥b¥c）を awk コマンドのスクリプトに渡せる例

この例では，CCC01，CCC02 の両方とも c:¥¥a¥¥b¥¥c が格納されます。

その後，awk コマンドの VAR001 へ格納される際には c:¥a¥b¥c となり，正しく処理できます。

```
CCC01=’c:¥¥a¥¥b¥¥c’
CCC02=”c:¥¥¥¥a¥¥¥¥b¥¥¥¥c”
awk -v VAR001=”${CCC01}” -f prog01.awk
awk -v VAR001=”${CCC02}” -f prog01.awk
```

同様に次の 2 つの例も正しいパス名を渡せます。

```
awk -v VAR001=c:¥¥¥¥a¥¥¥¥b¥¥¥¥c -f prog01.awk
awk -v VAR001=’c:¥¥a¥¥b¥¥c’ -f prog01.awk
```

特殊ファイル名

getline 関数で標準入力から入力したり，print 関数および printf 関数で標準出力，または標準エラー出力へ出力したりしたい場合に，それらの入力先および出力先を表す特殊ファイル名が使用できます。使用できる特殊ファイル名を次に示します。なお，特殊ファイル名を close 関数に指定しても無視されます。

特殊ファイル名	意味
/dev/stdin	標準入力
/dev/stdout	標準出力
/dev/stderr	標準エラー出力

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了
exit 文で指定した値	制御文の exit 文で指定したコマンドの終了コード

注意事項

- awk コマンドは，数値を内部的に倍精度浮動小数点数（8 バイト）として扱います。また，printf 関数および sprintf 関数の出力書式に，変換指定子 d，i，o，x，X，u を指定して出力および変換する場合は，4 バイトの符号付き整数で丸めます。このため，4 バイトの符号付き整数の範囲外の数値に，変換指定子 d，i，o，x，X，u を指定して出力または変換をした場合に誤差が発生します。この誤差は OS に依存します。
- getline 関数，print 関数および printf 関数で，同時にオープンできるファイルの最大数は 256 個です。同時にオープンできるファイルには，コマンド指定によって生成されるパイプも含みます。なお，UNIX の場合，OS 全体で同時にオープンできるファイルの最大値に達している，または ulimit でそのプロセスのファイルディスクリプタ数の最大値が制限されているなどの OS の設定値によっては，256 個より少なくなります。
- バイナリファイルからの入力およびバイナリデータの出力は，動作を保証しません。
- system 関数などによる外部プログラムの実行は，次のプログラムの引数として実行されます。このため，最大パス名長などの外部プログラム実行に関する仕様はそのプログラムの仕様に依存します。
  - Windows の場合

    環境変数 COMSPEC で指定したコマンドプロセッサの引数として実行されます。デフォルト値は cmd.exe です。なお，使用されるコマンドプロセッサは環境変数の COMSPEC と PATH によって決まります。
  - UNIX の場合

    シェルの引数として実行されます。OS の仕様によって，起動されるプログラムは異なる可能性があります。
- Windows の場合，getline 関数，print 関数または printf 関数で指定したコマンドからの入出力開始時に，デスクトップヒープ不足によって指定したコマンド実行でアプリケーションエラーとなる場合があります。このため，コマンド入出力が不要になったとき，close 関数でクローズするまたはデスクトップヒープ指定値を見直してください。

- Windows の場合，system 関数で実行するコマンドの引数にワイルドカードを含むファイル名やディレクトリ名を指定しても，ワイルドカードは展開されません。
- Windows の場合，getline 関数，print 関数，printf 関数でパイプによって接続するコマンドの引数にワイルドカードを含むファイル名やディレクトリ名を指定しても，ワイルドカードは展開されません。
- Windows の場合，system 関数のように外部プログラムを指定するところに，関連づけられているファイルのパス名を指定した場合，関連づけられているプログラムが起動します。バッチジョブで実行するときには注意が必要です。
- Windows の場合，入出力関数および汎用関数でパス名を指定するときは，次の指定をする必要があります。
- ディレクトリ区切り文字に「¥」を使用する場合は，「¥¥」で指定する必要があります。
  - system 関数のように外部プログラムを指定するところにスペースを含むパス名を指定する場合は，パス全体を「¥"」で囲む必要があります。
  - -F オプションの入力フィールドセパレータ，-v オプションの変数値，引数で指定する組み込み変数の変数値に指定した「¥」は，エスケープ文字を表す記号として扱われます。
- Windows の場合，system 関数などで生成したプロセスにファイルディスクリプタが引き継がれないで，クローズされた状態になります。例えば，親プロセスがオープンしていたファイルディスクリプタに対して再度オープンしないで入出力を行おうとするとエラーになります。ただし，標準入力，標準出力および標準エラー出力は再度オープンされた状態になります。
- Windows の場合，system 関数に指定したコマンド名にパスが含まれていないときは，コマンドプロンプトなどのコマンドプロセッサのパス検索順序で見つかったコマンドが実行されます。
- exit 文でコマンドの終了コードを返す場合，exit 文に指定した値によっては，コマンドの呼び出し元に返す終了コードと exit 文に指定した値が異なるときがあります。exit 文で指定するコマンドの終了コードの詳細については，exit 文の説明を参照してください。なお，awk コマンドでは exit 文で指定した値を 4 バイトの符号付き整数で丸めます。このため，4 バイトの符号付き整数の範囲外の数値を指定した場合は誤差が発生します。この誤差は OS に依存します。

使用例

- コマンドの引数に，レコードの検索するパターンおよびパターンに一致したレコード内の英小文字を英大文字に変換するアクションを指定します。入力ファイルは file01.txt です。

  file01.txt の内容

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk ”/hitachi/{print toupper($0)}” file01.txt
HITACHI GROUP01 TOKYO
HITACHI GROUP03 FUKUOKA
```

- 2 番目のフィールドが正規表現のパターンに一致するレコードを出力します。入力ファイルは file01a.txt です。

  file01a.txt の内容

```
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi grp0000 Fukuoka
hitachi group04 Hokkaido
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk ”$2 ~ /group/” file01a.txt
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi group04 Hokkaido
```

- -F オプションに入力フィールドセパレータとして#を指定して，3 番目のフィールドの内容を出力します。入力ファイルは file02.txt です。

  file02.txt の内容

```
hitachi#group01#Nagoya
HITACHI#group02#Hiroshima
hitachi#group03#Ooita
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -F”#” ”{print $3}” file02.txt
Nagoya
Hiroshima
Ooita
```

- コマンドの引数にアクションに渡す変数 padstr とその値を指定します。スクリプトファイルは prog01.awk です。入力ファイルは file03.txt です。

  prog01.awk の内容

```
# program name : prog01
{
count++
print padstr ” ” $0※
}
END{
    print ”total record : ” count
}
```

  注※

  -v オプションで指定した変数の値，スペース，入力レコードの内容を出力します。

  file03.txt の内容

```
group01 Tokyo
group02 Yokohama
group03 Fukuoka
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -v padstr="hitachi" -f prog01.awk file03.txt
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi group03 Fukuoka
total record : 3
```

- コマンドの引数に入力ファイル file02.txt に対する組み込み変数 FS の値として#を指定します。入力ファイルは file01.txt および file02.txt です。

  file01.txt の内容

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

  file02.txt の内容

```
hitachi#group01#Nagoya
HITACHI#group02#Hiroshima
hitachi#group03#Ooita
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk "{print $3}" file01.txt FS="#" file02.txt
Tokyo
Yokohama
Fukuoka
Hokkaido
Nagoya
Hiroshima
Ooita
```

- group03 を含むレコードから group06 を含むレコードまでをファイル file06.txt に出力します。スクリプトファイルは prog02.awk です。入力ファイルは file04.txt および file05.txt です。

  prog02.awk の内容

```
BEGIN{
    print "Extract record : group03 - group06" > "file06.txt"
}
/group03/,/group06/{ ※
    count++;
    print >> "file06.txt";

}
END{
    printf "total record : %03d¥n", count >> "file06.txt"
}
```

  注※

  group03 に一致するレコードから group06 に一致するレコードを処理対象とします。

  file04.txt の内容

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
```

```
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

file05.txt の内容

```
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog02.awk file04.txt file05.txt
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat file06.txt
Extract record : group03 - group06
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
hitachi group05 Nagoya
HITACHI group06 Hiroshima
total record : 004
```

- ファイル file04.txt の先頭レコードから group02 を含むレコードまでを出力します。また，ファイル file05.txt のすべてのレコードを出力します。スクリプトファイルは prog03.awk です。入力ファイルは直前の file04.txt および file05.txt です。

  prog03.awk の内容

```
{
    count++; print
    if ($2 == "group02") {
        nextfile※
    }
}
END{
    printf("total record : %03d¥n", count)
}
```

  注※

  2 番目のフィールドの内容が group02 の場合，次の入力ファイルの処理を開始します。

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog03.awk file04.txt file05.txt
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
total record : 005
```

- レコードの 2 番目のフィールドが group02 の場合，awk コマンドの実行を停止します。また，コマンドの終了コードとして，停止するまでに読み込んだレコード数を返します。スクリプトファイルは prog04.awk です。入力ファイルは file04.txt および file07.txt です。

  prog04.awk の内容

```
{
    print
```

```
if ($2 == ”group02”) {
        exit(NR)※
    }
}
END{
    printf(”total record : %03d¥n”, NR)
}
```

注※

現在までに読み込んだレコード数が格納されている組み込み変数 NR の値を終了コードとして，コマンドを終了します。

file04.txt の内容

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

file07.txt の内容

```
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
hitachi group03 Okinawa
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog04.awk file04.txt file07.txt
hitachi group01 Tokyo
HITACHI group02 Yokohama
total record : 002
```

- レコード中に最初に出てくる特定の文字列を，sub 関数を使用して変更します。スクリプトファイルは，prog05.awk です。入力ファイルは file08.txt です。

prog05.awk の内容

```
{
    if (sub(/日立/, ”&製作所”)※) {
        print
    } else {
            print ”未変換レコード : ” $0
    }
}
```

注※

レコード中の「日立」を「日立製作所」に置き換えます。

file08.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
田中 沖縄支店 田中グループ
日立 福岡支店 日立グループ
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog05.awk file08.txt
日立製作所 横浜支店 日立グループ
日立製作所 東京支店 日立グループ
未変換レコード : 田中 沖縄支店 田中グループ
日立製作所 福岡支店 日立グループ
```

- レコード中の特定の文字列を gsub 関数を使用してすべて変更します。スクリプトファイルは，prog06.awk です。入力ファイルは file09.txt です。

  prog06.awk の内容

```
{
    if (gsub(/日立/, "&製作所") ※) {
        print
    } else {
            print "未変換レコード : " $0
    }
}
```

  注※

  レコード中のすべての「日立」を「日立製作所」に置き換えます。

  file09.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
田中 沖縄支店 田中グループ
日立 福岡支店 日立グループ
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog06.awk file09.txt
日立製作所 横浜支店 日立製作所グループ
日立製作所 東京支店 日立製作所グループ
未変換レコード : 田中 沖縄支店 田中グループ
日立製作所 福岡支店 日立製作所グループ
```

- 特定の文字列の位置を index 関数によって求めます。スクリプトファイルは，prog07.awk です。

  prog07.awk の内容

```
BEGIN{
    str = "日立:hitachi"
    print "Column = " index(str, "hitachi")
}
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog07.awk
Column = 4
```

- 文字列の長さを length 関数によって求めます。スクリプトファイルは，prog08.awk です。

  prog08.awk の内容

```
BEGIN{
    str = ”日立:hitachi”
    print ”Length = ” length(str)
}
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog08.awk
Length = 10
```

- 特定の文字列の位置と文字列の長さを match 関数によって求めます。スクリプトファイルは，prog09.awk です。

  prog09.awk の内容

```
BEGIN{
    str = ”hitachi:日立製作所”
    print ”Column  = ” match(str, /製.所/)
    print ”RSTART  = ” RSTART
    print ”RLENGTH = ” RLENGTH
}
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog09.awk
Column  = 11
RSTART  = 11
RLENGTH = 3
```

- 文字列を特定の文字で分割し配列に格納します。スクリプトファイルは，prog10.awk です。

  prog10.awk の内容

```
BEGIN{
    str = ”日立#横浜支店#日立グループ”
    num = split(str, arry,”#”)
    for (i = 1; i <= num; i++ ) {
        print arry[i]
    }
}
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog10.awk
日立
横浜支店
日立グループ
```

- 特定の位置にある部分文字列を求めます。スクリプトファイルは，prog11.awk です。

  prog11.awk の内容

```
BEGIN{
    str = ”hitachi:日立製作所”
    rtnstr = substr(str, 11, 2)
    print ”SUBSTR = ” rtnstr
}
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog11.awk
SUBSTR = 製作
```

- 引数指定のファイル以外からレコードを入力します。スクリプトファイルは prog12.awk です。入力ファイルは file10.txt です。

  prog12.awk の内容

```
BEGIN{
    while ((getline rec < "file10.txt"※) > 0) {
        print rec
    }
}
```

  注※

  指定したファイル file10.txt からレコードを入力し，レコードの内容を変数 rec に格納します。

  file10.txt の内容

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog12.awk
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

- print 関数の出力内容をコマンドにパイプ経由で渡します。スクリプトファイルは prog13.awk です。入力ファイルは file11.txt です。

  prog13.awk の内容

```
BEGIN{
    cmd = "sort "
}
{
    if (sub(/group01/, $2)) {
        count++
        print | cmd※1
    }
}
END{
    close(cmd)※2
    print "total record : " count
}
```

  注※1

  変数 cmd に指定した sort コマンドにレコードの内容を渡します。

注※2

close 関数を実行することでパイプが閉じられ sort コマンドの実行が終了します。

file11.txt の内容

```
hitachi group01 003 tokyo
hitachi group02 001 yokohama
hitachi group03 001 fukuoka
hitachi group01 004 hokkaido
hitachi group01 001 nagoya
hitachi group02 001 hiroshima
hitachi group01 002 ooita
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog13.awk file11.txt
hitachi group01 001 nagoya
hitachi group01 002 ooita
hitachi group01 003 tokyo
hitachi group01 004 hokkaido
total record : 4
```

- 配列の要素を削除します。スクリプトファイルは prog14.awk です。

  prog14.awk の内容

```
BEGIN{
    arry[”福岡”]   = ”福岡”
    arry[”北海道”]   = ”札幌”
    arry[”神奈川”] = ”横浜”
    arry[”島根”]   = ”松江”
    for (key in arry) {
        printf(”  %*s : %s¥n”, 6, key,  arry[key])
    }
    print ”Deletes result of the array element”
    delete arry[”神奈川”]
    for (key in arry) {
        printf(”  %*s : %s¥n”, 6, key,  arry[key])
    }
}
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog14.awk
    福岡 : 福岡
  北海道 : 札幌
  神奈川 : 横浜
    島根 : 松江
Deletes result of the array element
    福岡 : 福岡
  北海道 : 札幌
    島根 : 松江
```

- プログラム開始時にディレクトリを作成し，作成したディレクトリ内のファイルにレコードの内容を出力します。スクリプトファイルは prog15.awk です。入力ファイルは file12.txt です。

  prog15.awk の内容

```
BEGIN{
    if ((rc = system("mkdir dir001")※1)) {
        printf("system func error rc = %x¥n", rc) > "/dev/stderr"※2
        exit(1)
    }
}
{
    print >> "dir001¥¥outfile.txt"
}
```

注※1

system 関数によって mkdir コマンドを実行してディレクトリを作成します。

注※2

system 関数の戻り値を標準エラー出力に出力します。

file12.txt の内容

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog15.awk file12.txt
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat dir001¥¥outfile.txt
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

- ユーザー定義関数を呼び出し，ユーザー定義関数の処理結果を出力します。スクリプトファイルは prog16.awk です。

  prog16.awk の内容

```
BEGIN{
a = 3
b = 4
result = func01(a, b, c)
print "func01 = " result
}
function func01(x,y){
x *= x
y *= y
return x + y
}
```

コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -f prog16.awk
awk: warning: function func01 called with 3 args, uses only 2※
 source line number 4
func01 = 25
```

注※

ユーザー定義関数呼び出し時の引数の個数が，関数の定義で記述した引数の個数よりも多いため，警告メッセージが出力されます。

- 制御文の記述で構文エラーがある場合のメッセージを表示します。スクリプトファイルは，prog17.awk です。

prog17.awk の内容

```
BEGIN{
    while ((getline rec < "file10.txt") > 0)) {※
        print rec
    }
}
```

注※

while 文の「(」と「)」の数が不一致となっています。

コマンドの実行結果を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥awk -f prog17.awk
awk: extra ) at source line 2 source file prog17.awk
 context is
            while ((getline rec < "file10.txt") > >>>  0)) <<<
awk: syntax error at source line 2 source file prog17.awk
awk: illegal statement at source line 2 source file prog17.awk
        extra )
```

- 組み込み関数の記述で書式が不正な場合のメッセージを表示します。スクリプトファイルは prog18.awk です。

prog18.awk の内容

```
BEGIN{
    str = "日立:hitachi"
    print "Column = " index(str)※
}
```

注※

index 関数の引数に検索する文字列の指定がありません。

コマンドの実行結果を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥awk -f prog18.awk
awk: syntax error at source line 3 source file prog18.awk
 context is
            print "Column = " >>>  index(str) <<<
awk: illegal statement at source line 3 source file prog18.awk
```

- ジョブコントローラの一時ファイル機能とパス変換機能を利用してファイルを入出力します。ジョブ定義スクリプトは adsh001.ash です。スクリプトファイルは prog19.awk です。入力ファイルは file12.txt です。

環境ファイルの一時ファイル機能とパス変換機能の指定内容

```
#-adsh_conf TEMP_FILE_DIR       "C:¥¥TEMP¥¥ADSH"
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV "/var/hitachi/jp1as/perm" "C:¥¥hitachi¥¥JP1AS¥¥perm"
```

adsh001.ash の内容

```
#-adsh_file_temp TEMP
#-adsh_step_start adsh001 -onError stop
"$ADSH_OSCMD_DIR/awk" -f prog19.awk "/var/hitachi/jp1as/perm/file12.txt"
#-adsh_step_error
exit 100
#-adsh_step_end
```

prog19.awk の内容

```
{
    print FILENAME,":",$0 > ENVIRON["TEMP"]
}
END{
while(getline var < ENVIRON["TEMP"])
    print var
}
```

file12.txt の内容

```
001 abc
002 efgh
003 ijklmnop
```

awk コマンドの出力内容（ジョブの標準出力に出力される内容）

```
C:¥hitachi¥JP1AS¥perm¥file12.txt : 001 abc
C:¥hitachi¥JP1AS¥perm¥file12.txt : 002 efgh
C:¥hitachi¥JP1AS¥perm¥file12.txt : 003 ijklmnop
```

- 引数が指定されていない場合のメッセージを表示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥awk
usage: awk [-F fs] [-v var=value] [-f progfile | prog]
        [[file ...] | [built-in-var=value ...]] ...
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥awk -z
awk: illegal option -- z
usage: awk [-F fs] [-v var=value] [-f progfile | prog]
      [[file ...] | [built-in-var=value ...]] ...
```

8.4.2　basename コマンド（パスからファイル名を取得する）

形式

```
basename　文字列［サフィックス］
```

機能

ファイルパス名規則に従った文字列からファイル名部分の文字列を取り出し，標準出力に出力します。

サフィックス（任意の文字による文字列）を指定すると，取り出した文字列の終端からサフィックスに一致する部分を削除します。

ファイル名部分の文字列の取り出し規則は次のとおりです。

- 指定された文字列の，ディレクトリ区切り文字で区切られた各要素から，最も右側にある要素を取り出します。
- UNIX の場合，/をディレクトリ区切り文字と見なします。Windows の場合，/と¥をディレクトリ区切り文字と見なします。
- Windows の場合，ドライブレターに続く:（コロン）も各要素の区切り文字として扱います。
- ディレクトリ区切り文字が連続している場合，1 つのディレクトリ区切り文字として扱います。
- 指定された文字列の終端がディレクトリ区切り文字の場合は，終端のディレクトリ区切り文字を取り除いたあとのファイル名部分の文字列を取り出します。
- 文字列にディレクトリ区切り文字がない場合，指定された文字列をそのまま取り出します。
- 文字列にディレクトリ区切り文字だけを指定した場合，ディレクトリ区切り文字を取り出します。

引数

文字列

ファイル名部分を取り出す文字列を指定します。

サフィックス

取り出したファイル名部分の終端から削除するサフィックスを指定します。指定したサフィックスが次の場合，取り出したファイル名部分の文字列をそのまま出力します。

- ファイル名部分の文字列の終端に一致しない。
- 取り出したファイル名部分の文字列と同じサフィックスを指定している。

終了コード

終了コード	意味
0	正常終了

終了コード	意味
1	エラー終了

注意事項

- このコマンドには指定できるオプションがありません。そのため，引数にオプションを指定した場合，そのオプションを，ファイル名部分を取り出す文字列またはサフィックスとして解釈します。

使用例

- パス名からファイル名部分の文字列を取り出す例を次に示します。

例 1

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename E:¥dir001¥file01.txt
file01.txt
```

例 2

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename /dir001
dir001
```

例 3

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename .¥file01.txt
file01.txt
```

例 4

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename E:¥dir001¥dir002¥
dir002
```

例 5

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename E:¥
E:
```

例 6

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename ¥¥server01¥
server01
```

例 7

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename ¥¥
¥
```

例 8

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename "C:¥Documents and Settings¥User01¥My Documents"
My Documents
```

例 9

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename C:file01.txt
file01.txt
```

- パス名から拡張子を除いたファイル名を取り出す例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥basename E:¥dir001¥file01.txt .txt
file01
```

8.4.3　cat コマンド（ファイルの内容を標準出力に出力する）

形式

```
cat [-b] [-n] [-s] [-u] [パス名 ...]
```

機能

ファイルの内容を標準出力に出力します。ファイルが複数ある場合は，連結して出力します。

引数

-b

空行以外のすべての行に番号を付けます。

-n

すべての行に番号を付けます。最初の行を 1 とします。行番号は 6 桁で表示します。6 桁で表示できない場合は順次，桁数を増やします。行番号の次には，タブを出力します。パス名が複数指定されている場合，パス名に指定されたファイルごとに行番号を付けて，連結して出力します。

-s

連続した空行を 1 つにします。

-u

UNIX の場合，出力時のバッファリングを抑止します。

Windows の場合，指定が無視されます。

パス名

連結して出力するファイルのパス名を指定します。パス名は複数指定できます。パス名を指定しない，またはパス名に「-」を指定した場合は，標準入力から入力します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 改行コードが［LF］の場合だけ空行と見なします。［CR］＋［LF］の行は空行と見なされないため，-b オプションおよび-s オプションは対象外になります。このため，Windows の通常ファイルの場合は，空行を含まないことになります。
- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。
- 標準出力をファイルに出力する場合，パス名に指定したファイルと同じファイルを指定すると，次のメッセージを出力してエラーになります。

```
cat: ファイル名: input file is output file
```

使用例

この使用例では，次の内容が記述された「abc.txt」「abcdex.txt」を基に，cat コマンドの実行結果を説明します。「△」はスペース，「→」はタブを表します。

- abc.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
cccccccccccccccc

    →       →       →

△△△△△△△△△△△△
dddddddddddd
```

- abcdex.txt

```
aaaaaaaaaaa
△△△△△△
△△△△△△
△△△△△△
△△△△△△
△△△△△△
bbbbbbbb





ccccccccccc




dddddddddddd
```

```
eeeeeeeeeeeeeeeee
   →       →
   →       →
```

cat コマンドを実行した結果表示に使用する入力ファイルの形式を次に示します。

- -b オプションを指定し，空行以外に行番号を付けます。

```
$ cat -b abc.txt
     1→aaaaaaaaaaa

     2→bbbbbbbb

     3→△△△△△△△
     4→cccccccccccccccc

     5→ →      →      →

     6→△△△△△△△△△△△
     7→ddddddddddddd
```

- -n オプションを指定し，すべての行に行番号を付けます。

```
$ cat -n abc.txt
     1→aaaaaaaaaaa
     2→
     3→bbbbbbbb
     4→
     5→
     6→cccccccccccccccc
     7→
     8→
     9→
    10→
    11→ddddddddddddd
```

- -s オプションを指定し，連続した空行を 1 つの空行として表示します。

```
$ cat -s abcdex.txt
aaaaaaaaaaa
△△△△△△
△△△△△△
△△△△△△
△△△△△△
△△△△△△
bbbbbbbb

ccccccccccc

ddddddddddddd

eeeeeeeeeeeeeeeee
```

```
→       →
→       →
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat -w
cat: illegal option -- w
usage: cat [-bnsu] [file ...]
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat file99
cat: file99: No such file or directory
```

8.4.4 cmp コマンド（バイナリファイルの内容を比較する）

形式

```
cmp [-l|-s] パス名1 パス名2 [比較開始位置1 [比較開始位置2] ]
```

機能

バイナリファイルを比較します。異なるバイト位置を表示できます。

引数

-l オプションおよび-s オプションを指定しない場合は，最初に検出した異なる場所を表示します。-l オプションおよび-s オプションを同時に指定した場合はエラーになります。

-l

違いのあるバイトのオフセット（10 進数）とその値（8 進数）を表示します。

-s

異なるかどうかを示す終了状態を返します。

パス名 1

比較元のパス名を指定します。パス名 1 に「-」を指定すると，標準入力から比較する内容を入力できます。

パス名 2

比較先のパス名を指定します。パス名 2 に「-」を指定すると，標準入力から比較する内容を入力できます。

比較開始位置 1

パス名 1 の比較を開始する位置（バイト）を指定します。

比較開始位置 2

パス名 2 の比較を開始する位置（バイト）を指定します。

終了コード

終了コード	意味
0	正常終了。ファイルは同一です。
1	正常終了。ファイルは異なっています。または，どちらかのファイルで先にファイルの終端（EOF）に到達しました。ファイルの終端に達した場合，メッセージ（cmp: EOF on ファイル名）を出力します。
2 以上	エラー終了

注意事項

- 改行コードが［CR］＋［LF］の場合，2 バイトと見なします。
- Windows の場合，ファイルおよび標準入力をバイナリモードで入力します。改行コードは変換しません。

使用例

この使用例では，次の内容が記述された「abc.txt」「abcd.txt」を基に，cmp コマンドの実行結果を説明します。「△」はスペース，「→」はタブを示します。

- abc.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
cccccccccccccccc

   →       →       →

△△△△△△△△△△△
dddddddddddd
```

- abcd.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
cccccccccccccccc
△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△△△△△△△△△△
dddddddddddd
```

```
eeeeeeeeeeeeeeeee
   →       →
   →       →
```

cmp コマンドを実行した結果表示に使用する入力ファイルの形式を次に示します。

- -l オプションを指定して，abc.txt と abcd.txt で違いのあるバイトのオフセット（10 進数）とその値（8 進数）を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp -l abc.txt abcd.txt
    49  12  40
    50  11  40
    51  11  40
    52  11  40
    53  12  40
    54  12  40
    65  40  12
    66  12  40
    67 144  40
    68 144  40
    69 144  40
    70 144  40
    71 144  40
    72 144  40
    73 144  40
    74 144  40
    75 144  40
    76 144  40
    77 144  40
    78 144  40
    79  12  40
cmp: EOF on abc.txt
```

- -s オプションを指定して，結果を表示しないで終了コードを返すようにします。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp -s abc.txt abcd.txt
```

- skip オプションを指定して，ファイルの比較を開始するバイトをそれぞれ 3 に設定します。上段に skip を指定しない場合を，下段に skip を 3 に設定した場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp abc.txt  abcd.txt
abc.txt abcd.txt differ: char 49, line 7

C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp abc.txt  abcd.txt  3 3
abc.txt abcd.txt differ: char 46, line 7
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp -w
cmp: illegal option -- w
usage: cmp [-l | -s] file1 file2 [skip1 [skip2]]
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp file99 file123
cmp: file99: No such file or directory
```

8.4.5 cp コマンド（ファイルまたはディレクトリをコピーする）

形式

```
cp [-f|-i] [-p] [-u] [-R|-r [-H|-L|-P] ] コピー元ファイル名 コピー先ファイル名
cp [-f|-i] [-p] [-u] [-R|-r [-H|-L|-P] ] コピー元 ... コピー先ディレクトリ名
```

機能

ファイルまたはディレクトリをコピーします。

引数

-f

--force

コピー先のファイルを上書きする場合に警告を出しません。-f，-i オプションは最後に指定されたオプションが有効になります。

-i

--interactive

コピー先のファイルを上書きする場合に警告を出し，応答を要求します。標準入力からの応答がy またはY の文字で始まっていればコピーします。それ以外の文字を応答したり，標準入力が使用できなかったりした場合は，処理を中断し，終了コード0 を返して終了します。

-f，-i オプションは最後に指定されたオプションが有効になります。

-p

--preserve

コピー元のファイルの属性を保存します。

Windows の場合，コピー元のファイルの最終修正日時および最終アクセス日時を保持します。ディレクトリの情報は保持しません。

UNIX の場合，コピー元のファイルの所有者，グループ，アクセス権，最終修正日時および最終アクセス日時を保持します。

-u

--update

ディレクトリ以外のファイルのコピーで，コピー先ファイルがすでに存在し，最終修正日時がコピー元と同じ，またはより新しい場合，コピーをしません。

シンボリックリンクのコピーでは，コピー先がシンボリックリンクの場合はコピー先のシンボリックリンク自身の最終修正日時で判断し，コピー先がファイルの場合はコピー先のファイルの最終修正日時で判断します。

ファイルの最終修正日時は，秒未満の値は切り捨てて判定します。

-R|-r

--recursive

ディレクトリを再帰的にコピーします。

Windows の場合，このオプションを指定して，コピー元に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。シンボリックリンクのリンク先のディレクトリをコピーしたい場合は，-H，または-L オプションと同時に指定してください。

-H

-R オプションまたは-r オプションと共に指定すると，コマンドライン上で指定したシンボリックリンクをたどります。

ツリー内をたどっている最中に見つけたシンボリックリンクのリンク先はたどりません。

このオプションはコピー元に対して適用されます。

-R オプションまたは-r オプションを指定しない場合は無視されます。また，-H オプション，-L オプションおよび-P オプションは最後に指定されたオプションが有効になります。

-L

--dereference

-R オプションまたは-r オプションと共に指定すると，遭遇したすべてのシンボリックリンクをたどります。

このオプションはコピー元に対して適用されます。

-R オプションまたは-r オプションを指定しない場合は無視されます。また，-H オプション，-L オプションおよび-P オプションは最後に指定されたオプションが有効になります。

-P

--no-dereference

-R オプションまたは-r オプションと共に指定すると，すべてのシンボリックリンクをたどりません。

このオプションはコピー元に対して適用されます。

-R オプションまたは-r オプションを指定しない場合は無視されます。また，-H オプション，-L オプションおよび-P オプションは最後に指定されたオプションが有効になります。

コピー元ファイル名

コピーするファイル名を指定します。

コピー先ファイル名

コピー先のファイル名を指定します。

コピー元

コピーするファイルまたはディレクトリを指定します。

コピー先ディレクトリ名

コピー先のディレクトリを指定します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- Windows の場合，コピー先のファイル名は，コピー元に指定したファイル名で作成されます。Windows では，ファイル名の英大文字を英小文字と扱ってコピーします。例えば，コピー対象のファイル名がA.txt の場合，cp a.txt tmpdir と実行すると，tmpdir の中のファイル名はa.txt になります。
- Windows の場合，ファイルをバイナリモードで入出力します。改行コードは変換しません。
- Windows の場合，-f オプションを指定してコピー先のファイルを上書きしようとすると，実行する Windows 環境の状態によって「Permission denied」を出力してエラー終了することがあります。
- UNIX の場合，一般ユーザーがcp コマンドの-p オプションでコピー元のファイルの属性を保存するとき，コピー元ファイルの所有者とcp コマンドの実行者が異なると，コピー元のファイルの所有者，グループ，およびアクセス権情報（setuid ビット，setgid ビット，スティッキービット）は保存しません。
- -u オプション指定によってコピー先が新しい（同じ場合を含む）場合，コピーしないときはエラーではなく，正常終了します。

使用例

- -i オプションを指定して，コピー先ファイルを上書きすることに対する応答を要求します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cp -i  file1.txt file2.txt
overwrite file2.txt? y

C:¥TEMP>%ADSH_OSCMD_DIR%¥cp -i file1.txt file2.txt
overwrite file2.txt? n
```

- オプションエラーのメッセージを表示します。

このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cp -w
cp: illegal option -- w
usage: cp [-fipu] [-Rr [-H | -L | -P]] source target
       cp [-fipu] [-Rr [-H | -L | -P]] source ... directory
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cp file99 file123
cp: file99: No such file or directory
```

8.4.6　cut コマンド（各行の選択範囲を標準出力に表示する）

形式

```
cut -b リスト [-n] [--output-delimiter=文字列] [パス名 ...]
cut -c リスト [--output-delimiter=文字列] [パス名 ...]
cut -f リスト [-s] [-d デリミタ] [--output-delimiter=文字列] [パス名 ...]
```

機能

各行の選択範囲を標準出力に表示します。それぞれのファイルまたはデフォルトの標準入力から，各行のリストに指定された部分を選択して標準出力に出力します。

引数

-b オプション，-c オプション，-f オプションのどれも指定しない場合は，usage を出力して終了します。

-b リスト

--bytes=リスト

動作を指定するオプションです。バイト位置で範囲指定します。リストには 1 から始まるバイト位置を指定します。複数回指定でき，指定した部分をすべてつなげて出力します。

--output-delimiter オプションを同時に指定すると，--output-delimiter オプションに指定した文字列でつなげて出力します。

-c リスト

--characters=リスト

動作を指定するオプションです。文字位置で範囲指定します。リストには 1 から始まる文字位置を指定します。複数回指定でき，指定した部分をすべてつなげて出力します。

--output-delimiter オプションを同時に指定すると，--output-delimiter オプションに指定した文字列でつなげて出力します。

-f リスト

--fields=リスト

動作を指定するオプションです。フィールド位置で範囲指定します。リストには区切り文字で区切られた 1 から始まるフィールド位置を指定します。複数回指定でき，指定した部分およびデリミタをすべてつなげて出力します。

選択されたフィールドは区切り文字で区切って表示します。区切り文字が存在しない行は，行全体を出力します。ただし，-s オプションを指定すると区切り文字が存在しない行は出力しません。

また，--output-delimiter オプションを指定することで，選択されたフィールドと共に出力する区切り文字を変更できます。

リスト

カラム位置または区切り文字で区切られたフィールド位置を指定できます。カラム位置は，1 から始まります。

選択範囲をコンマ，スペースまたはタブで区切ると，複数の選択範囲が指定できます。スペースまたはタブで区切る場合は，"（ダブルクォーテーション）で囲む必要があります。1 個の選択範囲は n，x-, -y，x-y のどれかを指定します。存在しない位置を指定してもエラーにはなりません。n，x，y はフィールドまたはカラム位置です。

- n：その位置だけを示します。
- x-：x の位置から最後までを示します。
- -y：先頭位置から y の位置までを示します。
- x-y：位置 x から位置 y を示します。x ＜ y となる必要があります。x ＞ y の場合，エラーメッセージが出力されます（cut: [-bcf] list: illegal list value）。

-n

マルチバイトを分割しません。-n を指定しない場合は，マルチバイト文字の途中でも分割します。

パス名

入力するパス名を指定します。パス名を省略するかハイフン（-）を指定すると，標準入力から入力します。

-s

--only-delimited

区切り文字が存在しない行は出力しません。-f オプションと共に指定しない場合，usage を表示して終了します。

-d デリミタ

--delimiter=デリミタ

デリミタで指定された先頭 1 文字をフィールド区切り文字にします。-d オプションを指定しない場合，タブが指定されたものとします。

-f オプションと共に指定しない場合，usage を表示して終了します。

--output-delimiter=文字列

-f オプションと共に指定した場合，出力するフィールドの区切り文字を指定した文字列に置き換えて出力します。

-b オプションまたは-c オプションと共に指定した場合，フィールド間を指定した文字列でつなげて出力します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- cut コマンドはテキストファイルを対象としています。バイナリファイルからの入力やバイナリデータの出力は，動作を保証しません。

使用例

この使用例では，次の内容が記述された「test.txt」を基に，cut コマンドの実行結果を説明します。

```
123:5678:abcdef:hijkl
field1:field2:field3:filed4
sssssssssssssssssssssss
```

cut コマンドを実行した結果表示に使用するファイルの形式を次に示します。

- 1 バイト目と 3 から 5 バイト目を出力します。

```
$ cut -b 1,3-5 test.txt
13:5
feld
ssss
```

- 1 から 4 文字目までを出力します。

```
$ cut -c -4 test.txt
123:
fiel
ssss
```

- 1 番目と 4 番目のフィールドを表示します。

```
$ cut -f 1,4 -d : test.txt
123:hijkl
field1:filed4
sssssssssssssssssssssss
```

- 1 バイト目と 3～5 バイト目を出力し，フィールド間に@:/の文字列を追加します。

```
$ cut -b 1,3-5 --output-delimiter="@:/" test.txt
1@:/3:5
f@:/eld
s@:/sss
```

- 1番目と4番目のフィールドを表示し，区切り文字を@:/の文字列に置き換えます。

```
$ cut -f 1,4 -d : --output-delimiter="@:/" test.txt
123@:/hijkl
field1@:/field4
sssssssssssssssssssssss
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windowsの例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cut -z
cut: illegal option -- z
usage: cut -b list [-n] [--output-delimiter=string] [file ...]
       cut -c list [--output-delimiter=string] [file ...]
       cut -f list [-s] [-d delim] [--output-delimiter=string] [file ...]
```

8.4.7 date コマンド（システムの日付と時刻を表示する）

形式

date [-u] [-d *日時情報指定文字列* | -r *経過秒*] [+*書式*]

機能

システムの日付と時刻を表示します。

引数

-u

--utc

--universal

UTC（世界協定時）の日付を表示します。

-d *日時情報指定文字列*

--date=*日時情報指定文字列*

表示する日時を日時情報指定文字列で指定します。複数指定した場合は最後の指定が有効になります。指定できる日時情報指定文字列については，「-d オプションで指定できる日時情報指定文字列」を参照してください。

-r *経過秒*

エポック（UTC の 1970 年 1 月 1 日 00:00:00）から，*経過秒*に指定した時間が経過した日時を表示します。*経過秒*に指定できる値は，-1009875600～2147440447 です。範囲外の値を指定した場合，出力する内容は保証できません。

+*書式*

日付と時刻の表示形式を書式指定コードで指定します。書式指定コードは OS の API である strftime 関数の書式指定コードが指定できます。指定できる書式指定コードについては，「指定できる書式指定コード」を参照してください。

この引数を指定していない場合は，日付と時刻の表示形式は「%Y/%m/%d %A %H:%M:%S %Z」になります。

指定できる書式指定コード

「+」で始まる引数には，OS の API である strftime 関数の書式指定コードが指定できます。この OS の API である strftime 関数で指定できる書式指定コードは，OS によって異なります。各 OS で指定できる書式指定コードについては，各 OS の strftime 関数についてのドキュメントを参照してください。

各 OS の strftime 関数と JP1/Advanced Shell 固有で共通して使用できる代表的な書式指定コードを次の表に示します。

書式指定コード	意味
%a	曜日の省略名
%A	曜日の正式名
%b	月の省略名
%B	月の正式名
%c	ロケールに対応する日付・時刻の表示
%d	10 進数で表す月の日付（01～31）
%H	24 時間表記で表す時間（00～23）
%I	12 時間表記で表す時間（01～12）
%j	10 進数で表す年の初めからの日数（001～366）
%m	10 進数で表す月（01～12）
%M	10 進数で表す分（00～59）
%p	現在のロケールの午前/午後
%S	10 進数で表す秒。表示される値の範囲は，うるう秒への対応の違いから，OS によって異なる
%s	エポック（UTC の 1970 年 1 月 1 日 00:00:00）からの経過秒数。 環境変数ADSH_CMDDATE_FORMAT で指定がない場合，Linux，AIX，HP-UX で使用できる（strftime 関数で処理する）。 環境変数ADSH_CMDDATE_FORMAT で指定した場合，すべての OS で使用できる（JP1/Advanced Shell 固有の変換をする）。

書式指定コード	意味
%U	10 進数で表す週の通し番号（00～53）。その年の最初の日曜日を週の最初の日とする
%w	10 進数で表す曜日（0～6，日曜日が0）
%W	10 進数で表す週の通し番号（00～53）。その年の最初の月曜日を週の最初の日とする
%x	現在のロケールの日付の表示
%X	現在のロケールの時刻の表示
%y	10 進数で表す西暦の下 2 桁（00～99）
%Y	10 進数で表す 4 桁の西暦
%Z	タイムゾーン名。タイムゾーンが不明な場合は表示しない
%%	%（パーセント）記号

-d オプションで指定できる日時情報指定文字列

日時情報指定文字列には，date コマンドで表示したい日時を次のように指定します。

- 絶対日時だけで指定
  指定した日時を表示します。
- 相対日時だけで指定
  現在の日時から移動した日時を表示します。
- 絶対日時と相対日時を組み合わせて指定
  絶対日時で指定した日時から，相対日時で指定した分だけ移動した日時を表示します。

日時情報指定文字列の要素は大文字でも小文字でも指定できます。UTC の 1970 年 1 月 1 日 0 時 0 分 0 秒より小さい値，または 2038 年 1 月 19 日 3 時 14 分 7 秒より大きい値を指定した場合，エラーメッセージ「date: Invalid date: *指定値*」を出力し，エラー終了します。ただし，AIX の場合，ローカルのタイムゾーンの 2038 年 1 月 19 日 3 時 14 分 7 秒より大きい値を指定すると，エラーメッセージ「date: Invalid date: *指定値*」を出力し，エラー終了します。また，日時情報指定文字列として空文字を指定した場合，現在の日付の 0 時 0 分 0 秒を表示します。

日時情報指定文字列はスペース区切りで指定します。ただし，文字，数字，符号の各種文字列で，異なる種類の文字列同士の場合はスペースなしで指定できます。例えば，文字と文字はスペースなしで指定できませんが，数字と文字など異なる種類の文字列同士の場合はスペースなしで指定できます。

日時情報指定文字列で指定できる要素と構文を次に示します。

- 絶対日時の指定
  絶対日時での日時情報指定文字列の要素を次の表に示します。

表 8-7　日時情報指定文字列（絶対日時の指定）の要素

種別	指定できる要素※	詳細
年	10 進数で表す 4 桁の西暦（*YYYY*）	1970 から2038 まで指定できます。
	10 進数で表す西暦の下 2 桁（*YY*）	00 から99 まで指定できます。 69 から99 までは 1900 年代と仮定され，00 から 68 までは 2000 年代と仮定されます。
月	月の名称（*MONTH*）	次の値が指定できます。 • January，Jan：1 月 • February，Feb：2 月 • March，Mar：3 月 • April，Apr：4 月 • May：5 月 • June，Jun：6 月 • July，Jul：7 月 • August，Aug：8 月 • September，Sept，Sep：9 月 • October，Oct：10 月 • November，Nov：11 月 • December，Dec：12 月
	10 進数で表す月（*MM*）	01 から12 まで指定できます。
日	10 進数で表す月の日付（*DD*）	01 から31 まで指定できます。
時	24 時間表記で表す時間（*hh*）	00 から23 まで指定できます。
	12 時間表記で表す時間（*hh*）	01 から12 まで指定できます。
分	10 進数で表す分（*mm*）	00 から59 まで指定できます。
秒	10 進数で表す秒（*ss*）	00 から59 まで指定できます。
午前・午後	午前・午後（am|a.m.|pm|p.m.）	次の値を時間の後ろに指定できます。 • am，a.m.：午前 • pm，p.m.：午後
タイムゾーン	タイムゾーン名（*ST*）	次の値が指定できます。 • UTC，UT：世界協定時 • GMT：グリニッジ標準時（UTC ＋ 0 時間） • WET：西ヨーロッパ時間（UTC ＋ 0 時間）

種別	指定できる要素※	詳細
タイムゾーン	タイムゾーン名（*ST*）	• AST：大西洋標準時（UTC - 4 時間） • EST：東部標準時（UTC - 5 時間） • CST：中部標準時（UTC - 6 時間） • MST：山岳部標準時（UTC - 7 時間） • PST：太平洋標準時（UTC - 8 時間） • HST：ハワイ標準時（UTC - 10 時間） • WAT：西アフリカ時間（UTC ＋ 1 時間） • CET：中央ヨーロッパ時間（UTC ＋ 1 時間） • MET：中央ヨーロッパ時間（UTC ＋ 1 時間） • CAT：中央アフリカ時間（UTC ＋ 2 時間） • EET：東ヨーロッパ時間（UTC ＋ 2 時間） • JST：日本標準時（UTC ＋ 9 時間） • GST：グアム標準時（UTC ＋ 10 時間） • NZST：ニュージーランド標準時（UTC ＋ 12 時間）
	ミリタリータイムゾーン（*ST*）	次の値が指定できます。 • A：UTC - 1 時間 • B：UTC - 2 時間 • C：UTC - 3 時間 • D：UTC - 4 時間 • E：UTC - 5 時間 • F：UTC - 6 時間 • G：UTC - 7 時間 • H：UTC - 8 時間 • I：UTC - 9 時間 • K：UTC - 10 時間 • L：UTC - 11 時間 • M：UTC - 12 時間 • N：UTC ＋ 1 時間 • O：UTC ＋ 2 時間 • P：UTC ＋ 3 時間 • Q：UTC ＋ 4 時間 • R：UTC ＋ 5 時間 • S：UTC ＋ 6 時間 • T：UTC ＋ 7 時間 • U：UTC ＋ 8 時間 • V：UTC ＋ 9 時間 • W：UTC ＋ 10 時間 • X：UTC ＋ 11 時間 • Y：UTC ＋ 12 時間 • Z：UTC

種別	指定できる要素※	詳細
タイムゾーン	UTC からの時間指定（+*hhmm* | -*hhmm* | +*hh*:*mm* | -*hh*:*mm*）	UTC からの時間を+*hhmm*，-*hhmm*，+*hh*:*mm* または-*hh*:*mm* で指定できます。*mm* および:*mm* は省略することができます。
タイムゾーン（夏時間）	タイムゾーン（夏時間）（*DT*）	次の値が指定できます。 • BST：英国夏時間（GMT ＋ 1 時間） • ADT：大西洋夏時間（AST ＋ 1 時間） • EDT：東部夏時間（EST ＋ 1 時間） • CDT：中部夏時間（CST ＋ 1 時間） • MDT：山岳部夏時間（MST ＋ 1 時間） • PDT：太平洋夏時間（PST ＋ 1 時間） • MEST：中央ヨーロッパ夏時間（MET ＋ 1 時間） • NZDT：ニュージーランド夏時間（NZST ＋ 1 時間）
デイライトセービングタイム	デイライトセービングタイム（*DST*）	デイライトセービングタイムとしてDST を指定できます。タイムゾーンと同時に指定することで，指定したタイムゾーンの標準時刻を，指定された日時やタイムゾーンに関係なく，常に 1 時間進めます。タイムゾーンの指定なしでDST を指定することはできません。

注※

括弧内の *YY* や *MONTH* などの記号は，次の表の構文と対応しています。

絶対日時での日時情報指定文字列の構文を次の表に示します。

表 8-8　日時情報指定文字列（絶対日時の指定）の構文

種別※	構文	詳細
日付の指定	[*YY*]*YYMMDD*	スペースなしで，年・月・日の順番で定義します。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。
	[*YYYY*/]*MM*/*DD*	「/」区切りで，年・月・日の順番で定義します。年を省略した場合，現在の年が仮定されます。 この構文の場合，*MM*，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	MM/*DD*[/[*YY*]*YY*]	「/」区切りで，月・日・年の順番で定義します。年を省略した場合，現在の年が仮定されます。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*MM*，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。ただし*MM* は 4 桁では指定できません。
	[*YY*]*YY*-*MM*-*DD*	「-」区切りで，年・月・日の順番で定義します。年の最初の 2 桁は省略できます。

種別※	構文	詳細
日付の指定	*[YY]YY-MM-DD*	この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*MM*，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	DD MONTH [[*YY*]*YY*]	スペース区切りで，日・月の正式名（または月の省略名）・年の順番で定義します。年を省略した場合，現在の年が仮定されます。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	MONTH DD [,[*YY*]*YY*]	月の正式名（または月の省略名）・日・年の順番で指定します。月の正式名（または月の省略名）と日をスペース区切りで，日と年を「,」区切りで定義できます。年を省略した場合，現在の年が仮定されます。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	DD[-]*MONTH*[[-][*YY*]*YY*]	「-」区切り，またはスペースなしで，日・月の正式名（または月の省略名）・年の順番で定義します。年を省略した場合，現在の年が仮定されます。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	MONTH-DD-[*YY*]*YY*	「-」区切りで，月の正式名（または月の省略名）・日・年の順番で定義します。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	MONTH DD [[*YY*]*YY*]	スペース区切りで，月の正式名（または月の省略名）・日・年の順番で定義します。年を省略した場合，現在の年が仮定されます。年の最初の 2 桁は省略できます。 この構文の場合，*YYYY* は前に 0 を付加した 5 桁以上の数値でも指定できます。また，*DD* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
時間の指定	*hh*[*mm*] *hh* [am｜a.m.｜pm｜p.m.]	スペースなしで，時・分の順番で定義します。 分の定義は省略できます。 時だけの定義のとき，スペース区切りでam(a.m.)，pm(p.m.)を定義できます。 am(a.m.)，pm(p.m.)を定義する場合，時の定義は 12 時間表記にしてください。 この構文の場合，*hh*[*mm*]の*hh* は 1 桁の数値でも指定できます。また，*hh* [am｜a.m.｜pm｜p.m.]の*hh* は 1 桁，または前に 0 を付加した 2 桁以上の数値を指定できます。
	hh:*mm*[:*ss*] [am｜a.m.｜pm｜p.m.]	「:」区切りで，時・分・秒の順番で定義します。 秒の定義は省略できます。

種別※	構文	詳細
時間の指定	*hh*:*mm*[:*ss*] [am｜a.m.｜pm｜p.m.]	時・分・秒の定義の後ろに，スペース区切りでam(a.m.)，pm(p.m.)を定義できます。 am(a.m.)，pm(p.m.)を定義する場合，時・分・秒の定義は12:59:59以内としてください。 この構文の場合，*hh*，*mm*，*ss* は1桁，または前に0を付けた2桁以上の数値を指定できます。
	hh:*mm*[:*ss*] [+*hh*[*mm*]｜-*hh*[*mm*]｜+*hh*[:*mm*]｜-*hh*[:*mm*]]	「:」区切りで，時・分・秒の順番で定義します。 秒の定義は省略できます。 時・分・秒の定義のあとに，スペース区切りで+*hhmm*，-*hhmm*，+*hh*:*mm* または-*hh*:*mm* を定義できます。 この構文の場合，*hh*，*mm*，*ss* は1桁，または前に0を付けた2桁以上の数値を指定できます。
タイムゾーン	*ST* [*DST*] *ST* [+*hh*[*mm*]｜-*hh*[*mm*]｜+*hh*[:*mm*]｜-*hh*[:*mm*]]	タイムゾーンを指定します。タイムゾーンの後ろにスペース区切りで，*DST*，+*hhmm*，-*hhmm*，+*hh*:*mm* または-*hh*:*mm* を定義できます。 この構文の場合，*hh*，*mm* は1桁，または前に0を付けた2桁以上の数値を指定できます。ただし，*hhmm* では*mm* は2桁で指定します。
	DT	タイムゾーン（夏時間）を定義します。タイムゾーン（夏時間）と共に，*DST*，+*hhmm*，-*hhmm*，+*hh*:*mm* または-*hh*:*mm* を定義することはできません。

注※

同じ種別は複数定義できませんが，異なる種別同士は組み合わせて定義できます。

日付だけ指定すると，時間は0時0分0秒が定義されます。

時（*hh*）だけ指定すると，現在の日付で分（*mm*）と秒（*ss*）が0で定義されます。

時間（*hh*）と分（*mm*）だけ指定すると，現在の日付で秒（*ss*）が0で定義されます。

- 相対日時の指定

相対日時での日時情報指定文字列の要素を次の表に示します。

表 8-9　日時情報指定文字列（相対日時の指定）の要素

指定できる要素※	詳細
年・月の移動（*DATE*）	次の値を指定します。 • year，years：年の移動 • month，months：月の移動 値の前に数値（*NUM*）を指定できます。数値（*NUM*）を省略した場合は1が仮定されます。
日の移動（*DATE*）	次の値を指定します。 • fortnight，fortnights：2週間（14日）の移動 • week，weeks：1週間（7日）の移動 • day，days：日の移動 • tomorrow：明日（1日後）

指定できる要素※	詳細
日の移動（*DATE*）	• yesterday：昨日（1 日前） • today：今日（0 日） • now：現在（0 日） 値の前に数値（*NUM*）を指定できます。数値（*NUM*）を省略した場合は1 が仮定されます。
時・分の移動（*DATE*）	次の値を指定します。 • hour, hours：時間の移動 • minute, min, minutes：分の移動 値の前に数値（*NUM*）を指定できます。数値（*NUM*）を省略した場合は1 が仮定されます。
秒の移動（*DATE*）	次の値を指定します。 • second, sec, seconds：秒の移動 値の前に数値（*NUM*）を指定できます。数値（*NUM*）を省略した場合は1 が仮定されます。
曜日の移動（*DAY*）	次の値を指定します。 • Monday, Mon：月曜 • Tuesday, Tue, Tues：火曜 • Wednesday, Wed, Wednes：水曜 • Thursday, Thu, Thur, Thurs：木曜 • Friday, Fri：金曜 • Saturday, Sat：土曜 • Sunday, Sun：日曜 値の前に数値（*NUM*）を指定し，*NUM* 回目の曜日を指定できます。数値（*NUM*）を省略した場合は1 が仮定されます。1 または数値（*NUM*）の指定がない場合，次に訪れる曜日を意味します。符号（+|-）または前後指定（ago）を指定することはできません。
数値指定（文字列）（*NUM*）	次の値を指定します。 • last：-1 • this：0 • next：1 • first：1 • third：3 • fourth：4 • fifth：5 • sixth：6 • seventh：7 • eighth：8 • ninth：9 • tenth：10 • eleventh：11 • twelfth：12 値の前に数値（*NUM*）は指定できません。
数値指定（数字）（*NUM*）	0 から2147483647 まで指定できます。
符号（+|-）	次の値を数値指定（数字）の前に指定できます。

指定できる要素※	詳細
符号（+|-）	• +：正 • -：負 符号の後ろに数値の指定がない場合，符号の指定は無視します。
前後指定（ago）	次の値を指定します。 • ago：前の「-」 ago の直前に指定された日時情報指定文字列の正負を逆（正なら負，負なら正）にします。

注※

括弧内の*NUM* や*DATE* などの記号は，次の表の構文と対応しています。

相対日時での日時情報指定文字列の構文を次の表に示します。「日時の移動」と「曜日の移動」は組み合わせて定義できます。

表 8-10　日時情報指定文字列（相対日時の指定）の構文

種別	構文	詳細
日時の移動	[[+ | -]*NUM*] *DATE* [ago]	現在の日時，または日時情報指定文字列（日時の指定）で指定した日時からの移動分を指定します。 スペース区切りで複数指定することもできます。 組み合わせ可能な要素については表「日時情報指定文字列（相対日時の指定）の要素」を参照してください。
曜日の移動	[*NUM*] *DAY* *DAY*[,]	曜日を指定します。複数指定することはできません。 また，曜日の前に数値（*NUM*）を指定すると，*NUM* 回目の曜日を定義できます。数値（*NUM*）の指定がない場合，次に訪れる曜日を意味します。 曜日の後ろに「,」またはスペースを指定することで，「日時の移動」の定義を続けて指定できます。

- その他の指定

  その他の日時情報指定文字列の要素を次の表に示します。

表 8-11　日時情報指定文字列（その他）の要素

指定できる要素	詳細
コメント	日時情報指定文字列の中に「(」と「)」で囲んで，コメントとして任意の文字列を指定します。「(」と「)」が適切に入れ子状になっている場合，中に指定された文字列は無視されます。 また，「(」だけが指定されている場合，「(」以降の指定はすべて無視されます。

「絶対日時の指定」および「相対日時の指定」は次の順序で算出されます。2.～4.の途中で，秒に換算した結果が0～2147483647 の範囲を超えた場合，最終結果に関係なくエラーになることがあります。

1. 現在の日時，または「絶対日時の指定」で指定された日時を求めます。
2. 1.に対して，「相対日時の指定」で指定された「曜日の移動」の結果を加算します。「絶対日時の指定」で「日付の指定」の指定がされた場合は，「曜日の移動」の指定があっても加算しません。

3. 2.に対して，「相対日時の指定」で指定された「年・月の移動」のすべての結果を加算・減算します。
4. 3.に対して，「相対日時の指定」で指定された「日の移動」，「時・分の移動」，「秒の移動」のすべての結果を加算・減算します。

2.～4.は計算対象の日時からの移動分を計算します。

例えば，現在の日時が 2014 年 4 月 30 日 10 時 10 分 10 秒のときに「date -d ”Fri, 1 year 1 month 1 day 1 hour 1 min 1 sec”」が指定された場合，次のように計算します。

1.で，現在の日時を求めます。
→2014 年 4 月 30 日（水曜日）10 時 10 分 10 秒になります。

2.で，2014 年 4 月 30 日（水曜日）から次の金曜日までの日数を足します。
→2014 年 5 月 2 日（金曜日）10 時 10 分 10 秒になります。

3.で，365 日と 2014 年の 5 月分の日数を足します。
→2015 年 6 月 2 日（月曜日）10 時 10 分 10 秒になります。

4.で，1 日分の日数と，1 時間 1 分 1 秒の時間を足します。
→2015 年 6 月 3 日（火曜日）11 時 11 分 11 秒になります。

日時情報指定文字列で複数の構文を組み合わせたときに注意が必要な指定

日時情報指定文字列で，「絶対日時の指定」および「相対日時の指定」を複数組み合わせたときに注意が必要な指定があります。

次に示すように，[*YY*]*YYMMDD* または *hh*[*mm*]の構文の後ろに符号付きの「相対日時の指定」を指定するとエラーになります。[*YY*]*YYMMDD* または *hh*[*mm*]の構文と符号付きの「相対日時の指定」を組み合わせて指定したい場合は，[*YY*]*YYMMDD* または *hh*[*mm*]を最後に指定してください。

```
C:¥TEMP>date -d ”20151112 -10 days”
date: Invalid date: 20151112 -10 days

C:¥TEMP>date -d ”-10 days 20151112”
2015/11/02 月曜日 00:00:00 JST

C:¥TEMP>date -d ”0955 -1 hours”
date: Invalid date: 0955 -1 hours

C:¥TEMP>date -d ”-1 hours 0955”
2016/09/28 水曜日 08:55:00 JST
```

文字，数字，符号の各種文字列で，異なる種類の文字列同士が隣り合う場合，スペースがあっても区切り文字として扱いません。このため，次の例の場合，「date -d ”10-November-15 days”」と同じ解釈をして，2015 年 11 月 10 日の 1 日後を表示します。後ろに指定する「相対日時の指定」の内容によって解釈が異なることがある「絶対日時の指定」を指定する場合は，「絶対日時の指定」を最後に指定してください。

```
C:¥TEMP>date -d ”10-November -15 days”
2015/11/11 水曜日 00:00:00 JST
```

次に示すように，*hh*:*mm*[:*ss*]およびST の後ろに符号付きの数字を指定した場合，符号付きの数字は+*hh*[*mm*]または-*hh*[*mm*]と解釈します。このため，次の例の場合，UTC-10 時間の 12 時 11 分 10 秒を JST に変換した日時に対して 1 時間進めた日時を表示します。

```
C:¥TEMP>date -d ”20151110 12:11:10 -10 hours”
2015/11/11 水曜日 08:11:10 JST
```

JP1/Advanced Shell 固有の共通書式指定コード

環境変数ADSH_CMDDATE_FORMAT で指定した書式指定コードは，strftime 関数を使用しないで，JP1/Advanced Shell が独自に編集します。これによって，OS 間のstrftime 関数の仕様差に影響されないで，共通の出力結果を得ることができます。

指定できる書式コードとその動作を次に示します。

環境変数ADSH_CMDDATE_FORMAT

JP1/Advanced Shell 固有に，変換処理は環境変数ADSH_CMDDATE_FORMAT で指定があったときだけ実行します。

環境変数名	指定値
ADSH_CMDDATE_FORMAT	s

環境変数ADSH_CMDDATE_FORMAT にs を指定した場合，エポック（UTC の 1970 年 1 月 1 日 00:00:00）からの経過秒数を出力します。s を指定した場合は以降に示す共通書式処理の範囲の動作となります。

書式の形式

%[*フラグ*][*フィールド幅*]

・*フラグ*～[_ | - | 0 | ^ | #]

_（アンダーバー）

数値の結果文字列のパディング（穴埋め）をスペース（空白文字）でします。

-（ハイフン）

数値の結果文字列をフィールドに左詰めに設定します。

フィールドの空きはスペース（空白文字）でパディングをします。

0

数値の結果文字列へのパディングを0 でします。

^

結果文字列中のアルファベット文字を大文字に変換します。

ただし，書式指定コードs では意味がないので無視します。

#

結果文字列の大文字・小文字を入れ替えます。

書式指定コードs では意味がないので無視します。

・*フィールド幅*

数値を出力するフィールドの幅を十進数で指定します。

パディングの指定がないときはスペース（空白文字）でパディングします。

書式の形式が正しくない場合は，この書式指定（当該書式の%からs まで）を文字として出力します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 「+」で始まる引数に指定した書式コードが，strftime 関数で有効な書式コード以外の場合は，指定された値をそのまま出力します。ただし，環境変数ADSH_CMDDATE_FORMAT で指定した書式指定コードは除きます。環境変数ADSH_CMDDATE_FORMAT で指定した書式指定コードの指定形式が不当な場合は，指定された値をそのまま出力します。
- Windows の場合，有効な書式コードと無効な書式コードを混在して指定すると，すべての書式コードを変換しないで，指定された値をそのまま出力します。
- UNIX の場合，有効な書式コードは変換出力し，無効な書式コードは指定された値を出力します。
- Windows の場合，環境変数TZ を設定するときは，環境変数TZ の値とコントロールパネルの［日付と時刻のプロパティ］ダイアログボックスで定義されているタイムゾーンを同じにしてください。同じでない場合，日時の表示が正しく行われない場合があります。
- 指定できる引数でない引数をコマンドラインに指定しても，その引数の指定は無視されます。
- 次の環境変数が設定されている場合は，「+」で始まる引数の後ろに指定したオプションは無視されます。
  - 環境変数POSIXLY_CORRECT
  - 環境変数ADSH_CMD_ARGORDER=seq

使用例

- オプションを指定しない場合のデフォルトを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date
2011/05/09 月曜日 02:03:05 JST
```

- -u オプションを指定して，UTC（世界協定時）の日付と時刻を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date -u
2011/05/08 日曜日 17:03:11 UTC
```

- -r オプションを指定して，エポックから，指定した秒が経過した日時を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date -r 1234567890
2009/02/14 土曜日 08:31:30 JST
```

- 「+」で始まるオペランドに，表示する日付と時刻の形式を指定します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date ”+%Y-%m-%d %H.%M.%S”
2011-05-09 02.10.02
```

- 今年の 12 月 12 日の日付を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date -d ”12/12”
2011/12/12 日曜日 00:00:00 JST
```

- 3 か月と 1 日後の日付を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date -d ”3 months 1 day”
2011/08/10 水曜日 00:00:00 JST
```

- 2 日前の日付を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date -d ”2 days ago”
2011/05/07 土曜日 00:00:00 JST
```

- 2011 年 5 月 1 日から 100 日後の日付を表示します。日時情報指定文字列を--date オプションで指定します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date --date=”1-May-2011 100 days”
2011/08/09 火曜日 00:00:00 JST
```

- Windows でエポック（UTC の 1970 年 1 月 1 日 00:00:00）からの経過秒数を表示します。

```
C:¥TEMP>set ADSH_CMDDATE_FORMAT=s
C:¥TEMP>%ADSH_OSCMD_DIR%¥date +%s
1435197101
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥date -a
date: illegal option -- a
usage: date [-u] [-d string | -r seconds] [+format]
```

8.4.8　diff コマンド（2 つのファイルや標準入力を比較する）

形式

```
diff [-a]  [-b]  [-i]  [-s]  [-w]
     [-c行数|-C 行数|-q|-u行数|-U 行数|
      -y [-W 出力幅]   [--suppress-common-lines] ]
     [-L ラベル]
     パス名1 パス名2
```

```
diff [-a] [-b] [-i] [-r] [-s] [-w]
     [-c行数|-C 行数|-q|-u行数|-U 行数|
      -y [-W 出力幅]  [--suppress-common-lines] ]
     [-L ラベル]
     ディレクトリ名1 ディレクトリ名2
```

機能

2 つのファイルや標準入力を比較します。

引数

-a

--text

ファイルをテキストと見なして比較します。

-b

--ignore-space-change

行中の 1 つ以上のスペースまたはタブを 1 個のスペースまたはタブとして比較します。-w オプションが指定されている場合は，-w オプションが有効となります。

-i

--ignore-case

英大文字と英小文字を区別しません。

-s

--report-identical-files

ファイルの内容が同じ場合，メッセージ（Files **パス名 1** and **パス名 2** are identical）を出力します。

-w

--ignore-all-space

行中のスペースおよびタブをすべて無視して比較します。-w オプションを指定すると，-b オプションの指定は無効になります。

-c 行数

-C 行数

--context［=行数］

比較するパス名を標準出力に出力します。行の追加，削除および変更を+，-，!の記号で出力します。

行数を指定した場合，差異の前後に指定した行数ずつ出力します。-c オプション，--context オプションに行数を指定しない場合，3 行ずつ出力します。

なお，-c オプションに行数を指定する場合，スペースを空けないで行数を指定してください。

-q

--brief

差異がある場合，メッセージ（Files **パス名 1** and **パス名 2** differ）だけを出力します。

-u 行数

-U 行数

--unified［=行数］

比較するパス名を標準出力に出力し，行の追加および削除を+，-の記号で出力します。差異が 1 つのセクションとして出力されます。

行数を指定した場合，差異の前後に指定した行数ずつ出力します。-u オプション，--unified オプションに行数を指定しない場合，3 行ずつ出力します。

なお，-u オプションに行数を指定する場合，スペースを空けないで行数を指定してください。

-L ラベル

--label=ラベル

-c，-u，-C，-U オプションが指定されている場合，パス名の代わりにラベルで指定したラベルを出力します。-L オプションを 1 つ指定した場合は，パス名 1 の代わりにラベルを出力します。-L オプションを 2 つ指定した場合は，パス名 1 とパス名 2 の代わりに指定された順序でラベルを出力します。

-y

--side-by-side

行の追加・削除・変更・改行コードの有無だけの差分を「>」「<」「|」「¥」「/」の記号で，差異のない行は記号なしで出力します。パス名 1 とパス名 2 の各行は 1 行にまとめて横並びに出力します。

まとめられた 1 行が 130 カラムを超える場合は，パス名 1 とパス名 2 の各行の長さを調整して出力します。

また，次のオプションと組み合わせて出力を変更することができます。

- -W オプション
- --suppress-common-lines オプション

-W 出力幅

--width=出力幅

1 行に出力する出力幅（カラム数）を変更できます。-y オプションと共に指定した場合に有効です。

--suppress-common-lines

差異のない行を出力しません。-y オプションと共に指定した場合に有効です。

パス名 1

比較元のパス名を指定します。

「-」を指定すると，比較する内容を標準入力から入力できます。また，標準入力から入力した内容を保存する一時ファイルが作成されます。一時ファイルの出力先ディレクトリは次のとおりです。

- UNIX の場合

  環境変数 TMPDIR に定義されたディレクトリに出力します。

  環境変数 TMPDIR が定義されていない場合は，/var/tmp に出力します。

- Windows の場合

  共通アプリケーションフォルダ¥HITACHI¥JP1AS¥misc に出力します。

パス名 2

比較先のパス名を指定します。

「-」を指定すると，比較する内容を標準入力から入力できます。また，標準入力から入力した内容を保存する一時ファイルが作成されます。一時ファイルの出力先ディレクトリは次のとおりです。

- UNIX の場合

  環境変数 TMPDIR に定義されたディレクトリに出力します。

  環境変数 TMPDIR が定義されていない場合は，/var/tmp に出力します。

- Windows の場合

  共通アプリケーションフォルダ¥HITACHI¥JP1AS¥misc に出力します。

-r

--recursive

ディレクトリ単位で比較した場合，サブディレクトリがあるときは，その配下も再帰的に検索して比較します。

ディレクトリ 1

比較元のディレクトリを指定します。ディレクトリ 1 とディレクトリ 2 のどちらか片方にパス名を指定した場合は，同じファイル名を別のディレクトリで検索して比較します。同じファイル名が存在しない場合はエラーメッセージ（diff: 比較したいパス名: No such file or directory）を出力します。

ディレクトリ 2

比較先のディレクトリを指定します。ディレクトリ 1 とディレクトリ 2 のどちらか片方にパス名を指定した場合は，同じファイル名を別のディレクトリで検索して比較します。同じファイル名が存在しない場合はエラーメッセージ（diff: 比較したいパス名: No such file or directory）を出力します。

出力形式

diff コマンドによる差異の表示形式には次に示す 3 つがあります。指定するオプションによって，どの出力形式になるかが決まります。

形式	意味
通常表示形式	-c，-C，-q，-u，-U，-y オプション指定時以外の表示形式です。2 つのファイルの差異を表示します。 2 つのファイルの差異の開始位置，終了位置および差異を表示します。2 つのファイルの差異の開始位置と終了位置の間の記号の意味を次に示します。 • a：追加 • d：削除 • c：変更 複数行にわたり差異がある場合は，差異開始行と差異終了行をコンマ（,）で区切って表示します。 差異はパス名 1 からの差分，パス名 2 からの差分の順に表示し，その間に「---」を表示します。差異の行頭の<は削除および変更された行を表し，>は追加および変更された行を表します。<と>の後ろにはスペースが 1 つ出力されます。
コンテキスト形式	-c，-C オプションを指定した場合の表示形式です。出力では差異がある行に加えて前後の変更されていない行も表示します。差異のない行を何行分表示するかは指定できます。デフォルトでは 3 行分表示します。 ヘッダには 2 つのファイルの情報を次のように表示します。 • 差異の固まりの境：15 個のアスタリスク（*） • 2 つのファイルの差異の開始位置，終了位置および差異 差異は，次のように表します。 • 行頭に+がある行：追加があった行 • 行頭にマイナス（-）がある行：削除があった行 • 行頭に!がある行：変更があった行 +，マイナス（-），!の後ろにはスペースが 1 つ出力されます。また，差分がない行の先頭にはスペースが 2 つ出力されます。 差異のある行が隣接する場合は 1 つの差異の固まりとして扱います。しかし，差異のある行が離れている場合は再度 15 個のアスタリスク（*）を表示し，差異を表示します。
ユニファイド形式	-u，-U オプションを指定した場合の表示形式です。出力はコンテキスト形式の出力を 1 つのセクションとして表示しています。差異のない行を何行分表示するかは指定できます。デフォルトでは 3 行分表示します。 ヘッダには 2 つのファイルの情報を次のように表示します。 • 2 つのアットマーク（@）で始まる行：2 つのファイルの差異の開始位置，終了位置および差異 差異は，次のように表示します。 • 行頭に+がある行：追加があった行 • 行頭にマイナス（-）がある行：削除があった行 +とマイナス（-）の後ろには，コンテキスト形式の場合と異なり，スペースは出力されません。また，差分がない行の先頭にはスペースが 1 つ出力されます。 変更があった行は，削除された行，追加された行として表されます。 差異のある行が隣接する場合は 1 つの差異の固まりとして扱います。しかし，差異のある行が離れている場合は再度 2 つのアットマーク（@）で始まる 2 つのファイルの差異の開始位置と終了位置を表示し，差異を表示します。
サイドバイサイド形式	-y オプションを指定した場合の表示形式です。出力はパス名 1 とパス名 2 のそれぞれの行を 1 行にまとめて横並びに表示します。デフォルトでは，出力する行は差異の有無に関係なくすべての行が対象です。まとめられた 1 行が 130 カラムを超える場合は，パス名 1 とパス名 2 のそれぞれの行は横並びで 130 カラム以内に表示できるように長さが調節されます。 差異は，パス名 2 の行の前に次の記号で示します。 • 「>」：追加された行 • 「<」：削除された行

形式	意味
サイドバイサイド形式	• 「|」：変更された行 • エスケープ文字(¥)：パス名 1 の行に改行がなかった行 • 「/」：パス名 2 の行に改行がなかった行 -y オプションは-W オプション，--suppress-common-lines オプションと組み合わせることで，1 行の出力幅の変更や，差異のない行の出力抑止ができます。

通常表示形式の例

通常表示形式の出力例を次に示します。

出力例

```
C:¥USR¥JP1¥oscmd¥bin>diff file1 file2
1c1,2                                ←1.
< aaaaaaaaaaa                        ←2.
---                                  ←3.
> aaAAAAAaaaa                        ←4.
> bbBBBBBbbbb                        ←4.
```

説明

1. file1 と file2 の差異がある位置を表します。file1 と file2 の間の記号の a は追加，d は削除，c は変更を意味します。記号の前には file1 の行番号が，記号の後には file2 の行番号が表示されます。複数行に渡って差異がある場合は，差異開始行と差異終了行をコンマ（,）で区切って表示します。
2. file1 の差異を表します。
3. file1 と file2 の差異の境目を表します。
4. file2 の差異を表します。

コンテキスト形式の例

出力例

```
C:¥USR¥JP1¥oscmd¥bin>diff -c file1 file2
*** file1       Thu May 12 20:17:54 2011  ←1.
--- file2       Thu May 12 20:18:29 2011  ←2.
***************                           ←3.
*** 1,5 ****                              ←4.
  aaaaaaaaaaa                            ←5.
! bbbbbbbb                                ←5.
  ccccccccccc                            ←5.
- ddddddddddd                            ←5.
  eeeeeeeee                              ←5.
--- 1,5 ----                             ←6.
  aaaaaaaaaaa                            ←7.
! bbbBBBbb                                ←7.
  ccccccccccc                            ←7.
  eeeeeeeee                              ←7.
+ fffffffffffffffff                      ←7.
```

説明

1. file1 のファイル情報として，ファイル名とファイルの最終修正日時を表示します。
2. file2 のファイル情報として，ファイル名とファイルの最終修正日時を表示します。
3. file1 と file2 の差異の固まりの境を 15 個のアスタリスク（*）で表示します。file1 と file2 の差異がある行が 3 行以上離れているときは，別の固まりとして再度この境を表示したあと，次の差異の情報を出力します。
4. file1 の差異の開始位置と終了位置をコンマ（,）で区切って表示します。
5. file1 の差異を表示します。プラス（+）は追加，マイナス（-）は削除，!は変更を意味します。
6. file2 の差異の開始位置と終了位置をコンマ（,）で区切って表示します。
7. file2 の差異を表示します。プラス（+）は追加，マイナス（-）は削除，!は変更を意味します。

ユニファイド形式の例

出力例

```
C:¥USR¥JP1¥oscmd¥bin>diff -u file1 file2
--- file1       Thu May 12 20:17:54 2011  ←1.
+++ file2       Thu May 12 20:18:29 2011  ←2.
@@ -1,5 +1,5 @@                           ←3.
 aaaaaaaaaaa                              ←4.
-bbbbbbbb                                 ←4.
+bbbBBBbb                                 ←4.
 ccccccccccc                              ←4.
-ddddddddddd                              ←4.
 eeeeeeeee                                ←4.
+fffffffffffffffff                        ←4.
```

説明

1. file1 のファイル情報として，ファイル名とファイルの最終修正日時を示します。
2. file2 のファイル情報として，ファイル名とファイルの最終修正日時を示します。
3. file1 と file2 の差異の開始位置と終了位置をコンマ（,）で区切って示します。先頭にマイナス（-）が付いている方が file1，先頭にプラス（+）が付いている方が file2 の差異の開始位置と終了位置を示します。
4. file1 と file2 の差異を一つのセクションとして示します。プラス（+）は file1 から file2 で追加された行を示します。マイナス（-）は file1 から file2 で削除された行を示します。変更部分は，削除された行，追加された行として表されます。

サイドバイサイド形式の例

出力例

```
C:¥USR¥JP1¥oscmd¥bin>diff -y file1 file2
a                                   a    ←1.
b                                 | b1   ←1.
```

```
c                                       c     ←1.
d                                     <       ←1.
e                                       e     ←1.
                                      > f     ←1.
g                                     ¥ g     ←1.
```

説明

1. file1 と file2 のそれぞれの行を 1 行にまとめて横並びに出力します。

   「>」は file1 から file2 で追加された行を示します。

   「<」は file1 から file2 で削除された行を示します。

   「|」は file1 から file2 で変更された行を示します。

   「¥」は file1 の行に改行がないことを示します。

   記号がない行は差異がない行です。

終了コード

終了コード	意味
0	ファイルは同一です。
1	ファイルは異なっています。
2 以上	エラー終了

注意事項

- -c オプション，-C オプション，-q オプション，-u オプション，-U および-y オプションは最後に指定したオプションが有効となります。
- ファイルの先頭から 8,192 バイト以内で表示できる 1 バイト文字，スペース，タブ，バックスペースおよびマルチバイト文字以外のデータが含まれている場合は，バイナリファイルと見なされます。
- ロケールと異なる文字コードのファイルはバイナリファイルと見なされます。
- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。
- パス名に「-」を指定した場合，端末から標準入力に入力している途中，および比較処理を実行している最中に diff コマンドの実行を中断すると，次の名前の一時ファイルが残ることがあります。この場合は，手動で一時ファイルを削除してください。

  【Windows の場合】

  diff.XXXXXX（XXXXXX は任意の 6 文字の文字列）

  【UNIX の場合】

  diffppppp.XXXXXXXX（ppppp は 5 桁以上のプロセス ID，XXXXXXXX は任意の 8 文字の文字列）

使用例

diff コマンドを実行した結果表示に使用する入力ファイルの形式を次に示します。「△」はスペース，「→」はタブを表します。

- abc.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
cccccccccccccccc

   →       →       →

△△△△△△△△△△△
dddddddddddd
△△△eeeeeeeeeeee
```

- abcd.txt

```
aaAAAAAaaaa

bbBBBbbb

△△△△△△△
cccccccccccccccc
△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△△△△△△△△△△
dddddddddddd
eeeeeeeeeeee
```

- wxy.txt

```
aaaaaaaaaaa

bbbbbbbb
xxxxxxxxxxxxxx

cccccccccccccccc
dddddddddddd
eeeeeeeeeeee
fffffffffffffff
ggggggggg
```

- wxyz.txt

```
aaaaaaaaaaa

bbbBBBbb
xxxxxxxxxxxxxx

cccccccccccccccc
dddddddddddd
```

```
fffffffffffffff
ggggggggg
hhhhhhhhhhhhhhhhhh
```

これらのファイルを基に，コマンドの実行結果を次に示します。

- オプションを指定しない場合のデフォルトを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff abc.txt abcd.txt
1c1
< aaaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
7,10c7,10
<
<    →    →     →
<
< △△△△△△△△△△△
---
> △△△△△△△△△△△△△△△△
> △△△△△△△△△△△△△△△△△△
> △△△△△△△△△△△△△△△
> △△△△△△△△△△△△△△△△△△△△△△△△△△△
12c12
< △△△eeeeeeeeeeee
---
> eeeeeeeeeeee
```

- -b オプションを指定し，スペースまたはタブの数の違いを無視します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -b abc.txt abcd.txt
1c1
< aaaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
12c12
< △△△eeeeeeeeeeee
---
> eeeeeeeeeeee
```

- -i オプションを指定し，英大文字と英小文字を区別しないで比較します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -i abc.txt abcd.txt
7,10c7,10
<
<    →    →     →
<
< △△△△△△△△△△△
---
```

```
> △△△△△△△△△△△△△△△△△△
> △△△△△△△△△△△△△△△△△△△△
> △△△△△△△△△△△△△△△△
> △△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△
12c12
< △△△eeeeeeeeeeee
---
> eeeeeeeeeeee
```

- -s オプションを指定し，ファイル内容が同一の場合も報告するようにします。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -s abc.txt abc.txt
Files abc.txt and abc.txt are identical
```

- -w オプションを指定し，行中のスペースおよびタブをすべて無視して比較します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -w abc.txt abcd.txt
1c1
< aaaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
```

- -q オプションを指定し，差異の内容は表示しないで，差異があるかどうかだけを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -q abc.txt abcd.txt
Files abc.txt and abcd.txt differ
```

- -c オプションを指定し，行の追加，削除および変更を+，-，!の記号で表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -c ..¥dir1¥wxy.txt ..¥dir1¥wxyz.txt
*** wxy.txt     Thu May 12 20:17:54 2011
--- wxyz.txt    Thu May 12 20:18:29 2011
***************
*** 1,10 ****
  aaaaaaaaaaa

! bbbbbbbb
  xxxxxxxxxxxxxx

  ccccccccccccccc
  ddddddddddddd
- eeeeeeeeeeee
  fffffffffffffff
  ggggggggg
--- 1,10 ----
  aaaaaaaaaaa

! bbbBBBbb
  xxxxxxxxxxxxxx

  ccccccccccccccc
  ddddddddddddd
  fffffffffffffff
```

```
  gggggggg
+ hhhhhhhhhhhhhhhhh
```

- -u オプションを指定し，行の追加および削除を+，-の記号で表示します。差異を 1 つのセクションとして表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -u ..¥dir1¥wxy.txt ..¥dir1¥wxyz.txt
--- wxy.txt     Thu May 12 20:17:54 2011
+++ wxyz.txt    Thu May 12 20:18:29 2011
@@ -1,10 +1,10 @@
 aaaaaaaaaaa

-bbbbbbbb
+bbbBBBbb
 xxxxxxxxxxxxxx

 ccccccccccccccc
 ddddddddddd
-eeeeeeeeeeee
 fffffffffffffff
 ggggggggg
+hhhhhhhhhhhhhhhhh
```

- -C オプションを指定し，差異の前後の 1 行を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -C1 wxy.txt wxyz.txt
*** wxy.txt     Thu May 12 20:17:54 2011
--- wxyz.txt    Thu May 12 20:18:29 2011
***************
*** 2,4 ****

! bbbbbbbb
  xxxxxxxxxxxxxx
--- 2,4 ----

! bbbBBBbb
  xxxxxxxxxxxxxx
***************
*** 7,10 ****
  ddddddddddd
- eeeeeeeeeeee
  fffffffffffffff
  ggggggggg
--- 7,10 ----
  ddddddddddd
  fffffffffffffff
  ggggggggg
+ hhhhhhhhhhhhhhhhh
```

- -U オプションを指定し，行の追加および削除を+，-の記号で表示します。差異を 1 つのセクションとして，差異の前後の 1 行を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -U1 wxy.txt wxyz.txt
--- wxy.txt     Thu May 12 20:17:54 2011
+++ wxyz.txt    Thu May 12 20:18:29 2011
@@ -2,3 +2,3 @@
```

```
-bbbbbbbb
+bbbBBBbb
 xxxxxxxxxxxxxx
@@ -7,4 +7,4 @@
 ddddddddddd
-eeeeeeeeeeee
 fffffffffffffff
 ggggggggg
+hhhhhhhhhhhhhhhhh
```

- -y オプションを指定し，行の追加，削除，変更を>，<，|の記号で表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -y wxy.txt wxyz.txt
aaaaaaaaaaa                         aaaaaaaaaaa

bbbbbbbb                          | bbbBBBbb
xxxxxxxxxxxxxx                      xxxxxxxxxxxxxx

ccccccccccccccc                     ccccccccccccccc
ddddddddddd                         ddddddddddd
eeeeeeeeeeee                      <
fffffffffffffff                     fffffffffffffff
ggggggggg                           ggggggggg
                                  > hhhhhhhhhhhhhhhhh
```

- -y オプションを指定してサイドバイサイド形式で表示する場合に，--suppress-common-lines オプションを指定して差異のない行を出力しないようにします。

```
C:¥TEMP>%ADSH_OSCMD_DIR%diff -y --suppress-common-lines wxy.txt wxyz.txt
bbbbbbbb                          | bbbBBBbb
eeeeeeeeeeee                      <
                                  > hhhhhhhhhhhhhhhhh
```

- -L オプションで指定したラベルで，比較元のファイル名を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -L name1 -c abc.txt abcd.txt
*** name1
--- abcd.txt    Thu May 12 20:36:44 2011
***************
*** 1,12 ****
! aaaaaaaaaaa

! bbbbbbbb

  △△△△△△△
  ccccccccccccccc
!
!   →    →     →
!
! △△△△△△△△△△△△
  ddddddddddd
! △△△eeeeeeeeeee
--- 1,12 ----
! aaAAAAAaaaa

! bbBBBbbb
```

```
  △△△△△△△
  cccccccccccccccc
! △△△△△△△△△△△△△△△△
! △△△△△△△△△△△△△△△△△△
! △△△△△△△△△△△△△△△
! △△△△△△△△△△△△△△△△△△△△△△△△△△△
  dddddddddddd
! eeeeeeeeeeee
```

- -a オプションを指定しないでバイナリファイルを比較した場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff binaryfile1 binaryfile2
Binary files binaryfile1 and binaryfile2 differ
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windowsの例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -z
diff: illegal option -- z
usage: diff [-abisw] [-c[number] | -C number | -q | -u[number] | -U number |
            -y [-W columns] [--suppress-common-lines]] [-L label] file1
            file2
       diff [-abirsw] [-c[number] | -C number | -q | -u[number] | -U number |
            -y [-W columns] [--suppress-common-lines]] [-L label] dir1 dir2
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff file99 file123
diff: file99: No such file or directory
```

8.4.9 dirname コマンド（パス名からディレクトリパス名部分の文字列を取り出す）

形式

```
dirname [文字列]
```

機能

ファイルパス名規則に従った文字列から，ファイル名部分を取り除いた残りのディレクトリパス名を取り出し，標準出力に出力します。

ディレクトリパス名の取り出し規則は次のとおりです。

- 指定された文字列のディレクトリ区切り文字で区切られた各要素のうち最も右側にある要素と，その要素の前にあるディレクトリ区切り文字を取り除いた，すべての要素を取り出します。なお，取り出したディレクトリパス名の途中に連続する区切り文字が存在しても，そのまま取り出します。

UNIX の場合は「/」がディレクトリ区切り文字と見なされ，Windows の場合は「/」と「¥」がディレクトリ区切り文字と見なされます。

- 指定された文字列の終端がディレクトリ区切り文字の場合は，終端のディレクトリ区切り文字を取り除いた上で最も右側にある要素を取り除いた，すべての文字列が取り出されます。
- 文字列にディレクトリ区切り文字がない場合や，文字列を指定していない場合は，カレントを意味する「.」（ピリオド）を出力します。
- 文字列にディレクトリ区切り文字だけを指定した場合は，ディレクトリ区切り文字を取り出します。
- Windows の場合，文字列の先頭 1 文字が英字で，次に「:」（コロン）が続いた場合，英字はドライブレターとして扱います。ドライブレターに続く「:」（コロン）も，各要素の区切り文字として扱います。
- Windows の場合，ルートディレクトリのディレクトリパスは，上記の規則に関係なく，次のように取り出されます。

パス名の開始文字列の形式	dirname コマンドの取り出し結果
ドライブレター:¥	ドライブレター:¥
ドライブレター:	ドライブレター:
¥¥サーバ名（UNC 名指定）	¥¥
¥¥?（サービス機能の不活性化指定）	¥¥
¥¥.（10 番以降のデバイス名指定）	¥¥

dirname コマンドの指定値と取り出し結果の例を次に示します。

dirname コマンドの指定値	取り出し結果
C:¥	C:¥
C:	C:
¥¥server01¥	¥¥
¥¥server01	¥¥
¥¥?¥	¥¥
¥¥?	¥¥
¥¥.¥	¥¥
¥¥.	¥¥
¥¥	¥¥
C:file001.txt	C:
C:¥file001.txt	C:¥

引数

文字列

ファイルパス名を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- このコマンドには指定できるオプションが存在しません。そのため，引数にオプションを指定した場合，そのオプションはディレクトリパス名部分を取り出す文字列として解釈します。

使用例

- パス名からディレクトリパス名部分の文字列を取り出す例を次に示します。

例 1

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname E:¥dir001¥file01.txt
E:¥dir001
```

例 2

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname /dir001
/
```

例 3

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname .¥file01.txt
.
```

例 4

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname E:¥dir001¥dir002¥
E:¥dir001
```

例 5

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname E:¥
E:¥
```

例 6

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname ¥¥server01¥
¥¥
```

例 7

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname ¥¥server01¥com
¥¥server01
```

例 8

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname ¥¥
¥¥
```

例 9

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname "C:¥Documents and Settings¥User01¥My Documents"
C:¥Documents and Settings¥User01
```

例 10

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname C:file01.txt
C:
```

例 11

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname C:¥file01.txt
C:¥
```

例 12

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥dirname file01.txt
.
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥> dirname /a/b /c/d
usage: dirname [string]
```

8.4.10　egrep コマンド（ファイル内の文字を検索する）

形式

```
egrep [-a] [-b] [-c] [-E] [-h] [-I] [-i] [-L] [-l] [-n]
      [-q] [-R] [-r] [-s] [-U] [-v] [-w] [-x]
      [-A 数値]  [-B 数値]  [-C [数値] ]
      [-e パターン]  [-f パターンファイルパス名]  [パターン]  [パス名 ...]
```

機能

ファイルの中から指定されたパターンを検索します。検索するパターンは拡張された正規表現として扱います。egrep コマンドの動作は-E オプションを指定した grep コマンドと同じです。

引数

-a

すべてのファイルを ASCII テキストファイルとして扱います。

-b

それぞれ一致した行の先頭にバイト単位のオフセットを出力します。

-c

選択された行数だけ標準出力に出力します。

-E

拡張された正規表現としてパターンを扱います（デフォルト値）。

-h

次のどちらかの指定をする場合，各出力行の先頭にファイル名を付けないようにします。

- -R または-r オプションを指定する
- 複数の検索対象パス名を指定する

-I

バイナリファイルを無視します。

-i

大文字と小文字を区別しません。

-L

パターンを含まないファイルの名前だけを標準出力に出力します。-L オプションと-l オプションは，最後に指定したオプションが有効となります。

-l

パターンを含むファイルの名前だけを標準出力に出力します。-L オプションと-l オプションは，最後に指定したオプションが有効となります。

-n

各出力行にファイルの相対的な行番号を出力します。-c オプション，-L オプション，-l オプション，および-q オプションを指定した場合は無視されます。

-q

標準出力には何も出力しません。終了コードだけを返します。

-R|-r

検索ディレクトリを再帰的に検索します。

なお，-L オプション，-l オプション，および-q オプションを指定しない場合は，各出力行の先頭にファイル名が付けられます。

Windows の場合，このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。

-s

読めないファイルや存在しないファイルは無視します。エラーメッセージを抑止します。

-U

バイナリファイルを検索します。ただし，出力はしません。

-v

パターンに一致しなかった行を出力します。

-w

指定文字列が単語として含まれている行を出力します。

単語とは英数字およびアンダースコア（_）から構成される文字列のことです。また，単語の前後はスペースなどの単語構成文字列以外の文字や，行頭または行末で区切られている必要があります。

-x

指定した文字列とファイルのすべての行を 1 行ごとに比較して完全に一致した場合に，一致した回数だけ指定した文字列を出力します。

-A 数値

数値で指定した行だけ，パターンにマッチした行のあとの行も出力します。

-B 数値

数値で指定した行だけ，パターンにマッチした行の前の行も出力します。

-C ［数値］

数値で指定した行だけ，パターンにマッチした行の前後の行も表示します。数値を省略した場合，前後 2 行を表示します。この場合，「-A 2 -B 2」と指定したときと同じになります。

-C オプションに数値を指定する場合は，-C オプションと数値の間にスペースを入れないでください。

パターン｜-e パターン

検索するパターンを指定します。-e オプションは複数指定できます。

-e オプションには，'-'で始まるパターンを指定できます。

-f パターンファイルパス名

検索するパターンを 1 行ごとにパターンファイルのパス名に指定します。パターンの指定がない場合はマッチしません。

［パス名 ...］

検索対象のパス名を指定します。複数指定ができます。パス名を指定しない場合は，検索対象の内容を標準入力から入力できます。ディレクトリ名の指定は，-R オプションまたは-r オプションを指定した場合に有効です。

なお，-L オプション，-l オプション，および-q オプションを指定しない場合は，各出力行の先頭にファイル名が付けられます。

終了コード

終了コード	意味
0	正常終了。 • パターンを含む行が存在します。 • -v が指定されている場合は，パターンを含まない行が存在します。
1	正常終了。 • パターンを含む行が存在しません。 • -v が指定されている場合は，パターンを含まない行が存在しません。
2 以上	エラー終了

注意事項

- ファイルの先頭から 8,192 バイト内に表示できる 1 バイト文字，スペース，タブ，バックスペースおよびマルチバイト文字以外のデータが含まれている場合は，バイナリファイルと見なされます。
- Windows のコマンドプロンプトから実行する場合，パターンをクォーテーションで囲むときは"（ダブルクォーテーション）を使用してください。
- ロケールと異なる文字コードのファイルはバイナリファイルと見なされます。
- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。
- 正規表現で使用する次のメタキャラクタを検索する場合は，直前にエスケープ文字（¥）を指定します。
  +，?，|，(,)，{,}

使用例

拡張された正規表現による検索の使用例を示します。オプションの使用例については grep コマンドの使用例を参照してください。

- 拡張された正規表現を示す「|」を使用して，文字列「AB」と「AD」のどちらかを含む行を検索します。入力ファイルは file01.txt です。

  file01.txt の内容

  ```
  AA
  AB
  AC
  AD
  AB|AD
  ```

  コマンドの実行結果を次に示します。

  ```
  C:¥TEMP>%ADSH_OSCMD_DIR%¥egrep ”AB|AD” file01.txt
  AB
  AD
  AB|AD
  ```

- 文字列「AB|AD」を含む行を検索します。「|」は拡張された正規表現として扱われるため，「|」の直前にエスケープ文字（¥）を指定します。入力ファイルは file01.txt です。

  file01.txt の内容

```
AA
AB
AC
AD
AB|AD
```

  コマンドの実行結果を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥egrep "AB¥|AD" file01.txt
AB|AD
```

8.4.11　expand コマンド（タブ文字をスペースに置き換える）

形式

```
expand [-タブリスト] [-t タブリスト] [パス名 ...]
```

機能

タブストップでそろえられている行をタブ文字からスペースに置き換えて標準出力に出力します。タブ文字の後ろにバックスペース文字が続いた場合，タブのカラム幅を減らして出力されます。

入力ファイル内の行は，改行文字で区切られたレコードを 1 つの行と見なします。Windows では［CR］＋［LF］または［LF］が改行文字と見なされ，UNIX では［LF］が改行文字と見なされます。なお，UNIX の場合，入力ファイルの各レコードが［CR］＋［LF］で区切られていると，変換後の出力行には［CR］が含まれます。

引数

-タブリスト

「-t タブリスト」と動作は同じです。

また，「-タブリスト」と「-t タブリスト」は同時に指定でき，その場合は複数指定となります。

-t タブリスト

--tabs=タブリスト

タブストップの位置を 1 以上の整数で指定します。このオプションを指定しない場合，タブストップのデフォルト値である 8 が適用され，タブストップの位置を 1 つ指定した場合と同じになります。

タブリストにタブストップの位置を 1 つ指定する場合と，複数指定する場合の動作は次のとおりです。

タブストップの位置を 1 つ指定した場合

タブストップごとの文字間隔として使用します。

これによって，1 つのタブストップに含まれるタブ文字をスペースへ置き換える際は，指定した文字間隔となるようスペースの数を調整されます。

タブストップの位置を複数指定した場合

タブストップのカラム位置として使用します。カラム位置は 0 から始まります。

タブストップの位置を複数指定する方法を次に示します。

・タブリストに，コンマまたはスペースで区切ってタブストップの位置を複数指定する

スペースで区切る場合は，"（ダブルクォーテーション）で囲む必要があります。

これによって，タブストップに含まれるタブ文字をスペースへ置き換える際は，指定したカラム位置となるようスペースの数を調整されます。タブリストで指定された数以降のタブストップの設定が必要な場合は，スペース 1 つに置き換えられます。

・タブリストにタブストップの位置を 1 つ指定し，オプションを複数指定する

・上記の 2 つの指定方法を組み合わせて指定する

タブストップは，入力行ごとに最初の指定値から設定されます。タブストップの位置は引数全体で昇順となるよう指定してください。

パス名

タブ文字をスペースに置き換えるファイルのパス名を指定します。パス名を指定しないか「-」を指定した場合は，標準入力から入力します。

複数のファイルを指定し，ファイルのどれかでオープンに失敗した場合は， エラーメッセージを出力して継続します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了 • パス名にオープンできないファイルが存在しました。
2	エラー終了（終了コード 1 のケースを除く）

注意事項

- expand コマンドはテキストファイルを対象としています。バイナリファイルから入力した場合や，バイナリデータに出力した場合は，動作を保証しません。

使用例

タブストップの位置を 1 つ指定する場合

タブストップを 1 つ指定した場合，タブストップごとの文字間隔として使用します。

例として，ファイル「file1」のタブストップに 6 を指定した例を次に示します。

ファイル「file1」の内容

次のファイル内容の「→」はタブ文字を示しています。

```
----+----+----+----+----+----+-----
a001 → a002 → a003
b001 → b002    →→    b003
c001 → c2   → c03
```

コマンド実行例

タブストップを 6 とし，ファイル file1 を指定しています。

```
$ expand -t 6 file1
```

実行結果

```
----+----+----+----+----+----+-----
a001  a002  a003
b001  b002        b003
c001  c2    c03
```

1 行目は指定値 6 の間隔でタブストップが設定されています。

2 行目の b002 と b003 の間はタブ文字が 2 つあるため，タブストップが 2 つ設定されています。

3 行目は文字列数が異なる c001，c2，c03 がタブストップ間隔 6 で設定されています。

タブストップの位置をタブリストで複数指定する場合

タブストップを複数指定した場合，指定した順にタブストップが設定されます。指定数を超えてタブストップの設定が必要な場合は，スペース 1 つと置き換えられます。

ファイル「file1」の内容

次のファイル内容の「→」はタブ文字を示しています。

```
----+----+----+----+----+----+-----
a001 → a002 → a003 → a004
```

コマンド実行例 1

タブリストを 6，16 とし，ファイル file1 を指定しています。

```
$ expand -t 6,16 file1
```

実行結果 1

```
----+----+----+----+----+----+-----
a001  a002      a003 a004
```

指定値の桁までスペースを設定して文字列を出力します。

1 つ目の指定値は 6 のため，a001 を出力して 6 桁までスペースを設定し，その後ろに a002 を出力しています。

同様に a002 を出力したあと，指定値である 16 桁までスペースを設定し a003 を出力しています。

タブリストにはそれ以降の指定値がないため，a003 と a004 の間にはスペースを 1 つ置いて a004 を出力しています。

コマンド実行例 2

タブリストを 2，16 とし，ファイル file1 を指定しています。

```
$ expand -t ”2 16” file1
```

実行結果 2

```
----+----+----+----+----+----+-----
a001            a002 a003 a004
```

指定値 1 の値が 2 のため，a001 を出力したあと 2 桁までスペースを設定しようとしますが，a001 を出力した時点で 2 桁を超えています。そのため，指定値「2」は無視し，次の値である 16 桁までスペースを設定して，a002 を出力しています。

タブリストにはそれ以降の指定値がないため，スペースを 1 つずつ設定して，その後ろの文字列を設定しています。

タブストップをオプションで複数指定する場合

［-t タブリスト］と［-タブリスト］を組み合わせたタブストップの複数指定には，幾つかの記述方法があります。次の指定例はタブストップを 2 と 16 で指定した場合の指定方法で，どの記述方法も同一です。

```
$ expand -t 2 -t 16 file1
$ expand -t 2 -16 file1
$ expand -2 -t 16 file1
$ expand -2 -16 file1
$ expand -t 2,16 file1
$ expand -2,16 file1
```

バックスペースを含んで入力する場合

タブ文字の後ろにバックスペース文字が続いた場合，タブのカラム幅を減算して出力されます。

ファイル「file1」の内容

次のファイル内容の「→」はタブ文字を示しています。

a003 の直前にバックスペース文字（^H）が存在しています。

```
----+----+----+----+----+----+-----
a001 → a002 → ^Ha003→a004
b001 → b002 → b003 → b004
```

コマンド実行例

タブストップをデフォルトの 8 で実行します。

```
$ expand file1
```

実行結果

```
----+----+----+----+----+----+-----
a001    a002   a003     a004
b001    b002    b003    b004
```

a003 の直前にバックスペース文字があるため，出力時に保存され，結果として a003 の出力桁は 17 桁ではなく 16 桁となります。

標準入力からの入力の場合

パス名を指定しないか「-」を指定する場合は，標準入力からの入力となります。

ファイル「file1」の内容

次のファイル内容の「→」はタブ文字を示しています。

```
----+----+----+----+----+----+-----
a001 → a002 → a003 → a004
```

コマンド実行例

file1 を標準入力から入力しています。

```
$ expand < file1
```

または

```
$ expand - < file1
```

実行結果

```
----+----+----+----+----+----+-----
a001    a002    a003    a004
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥> expand -z
expand: illegal option -- z
usage: expand [-tablist] [-t tablist] [file ...]
```

8.4.12　expr コマンド（式を評価する）

形式

```
expr 式
```

機能

式を評価して，標準出力に結果を書き込みます。式の要素はすべて別々の引数として指定します。

式は，数値・文字列・変数・式およびそれらと演算子の組み合わせで指定します。式の評価は文字列および整数として保持します。

引数

式

評価する式を指定します。演算子を優先順位の低い順に次に示します。同じ優先順位の演算子は，{ } で囲みコンマで区切って示します。引数が不正の場合，expr コマンドはエラーメッセージを出力して終了コード 2 を返します。expr1 と expr2 には式を指定します。

expr1| expr2

expr1 の評価が空文字列およびゼロではない場合，expr1 の評価を返します。expr1 が空文字列およびゼロの場合は，expr2 の評価を返します。expr2 も空文字の場合は，空文字を返します。

expr1 & expr2

どちらの式の評価も空文字列またはゼロではない場合，expr1 の評価を返します。それ以外の場合は，0 を返します。

expr1 {=，>，>=，<，<=，!=} expr2

両方の式の評価が整数の場合は，整数を比較した結果を返します。それ以外の場合は，ロケールで定義した照合順序で文字列を比較した結果を返します。結果は，指定された関係が真の場合は 1，偽の場合は 0 になります。

・=：左辺の値と右辺の値が等しい

・>：左辺の値が大きい

・>=：左辺の値が大きいか右辺の値と等しい

・<：左辺の値が小さい

・<=：左辺の値が小さいか右辺の値と等しい

・!=：左辺の値と右辺の値が等しくない

expr1 {+，-} expr2

両方の式の評価が整数値の場合，加算または減算の結果を返します。

整数値ではない場合，エラーメッセージ（expr: non-numeric argument）を出力します。

・+：加算

・-：減算

expr1 {*，/，%} expr2

両方の式の評価が整数値の場合，乗算，除算および剰余演算の結果を返します。整数値ではない場合，エラーメッセージ（expr: non-numeric argument）を出力します。除数がゼロの場合，エラーメッセージ（expr: division by zero）を出力します。

・*：乗算

・/：除算

・%：剰余

expr1 : expr2

expr2 が expr1 と一致するかどうかを評価します。

expr2 は正規表現で指定します。正規表現には，「^」がストリングの先頭に付加されます。

・expr2 にタグ付き正規表現が指定されている場合，（expr2 が expr1 と一致する場合）最初のタグ付き正規表現にマッチした文字列を返します。

・expr2 にタグ付き正規表現が指定されていない場合，（expr2 が expr1 と一致する場合）一致した文字数を返します。

・expr2 が expr1 と一致しない場合，および expr2 に正規表現が使用されている場合は空文字を返します。expr2 に正規表現が使用されていない場合は，0 を返します。

・expr2 の指定が空文字と一致する指定の場合，0 を返します。そのため，expr1 が空文字であることを判定する場合は，expr1 と expr2 の両方に同じ文字を付与して評価させる必要があります。つまり，「expr '' : '$'」はエラーであり，「expr X'' : 'X$'」などのように使用する必要があります。

length 文字列

指定した文字列の長さを返します。環境変数 ADSH_CMDEXPR_LENGTH については「2.5　環境変数を設定する」を参照してください。

・環境変数 ADSH_CMDEXPR_LENGTH=b が設定されている場合，length は演算子と判断され，その後ろに続く文字列の長さ（バイト数）を返します。

・環境変数 ADSH_CMDEXPR_LENGTH=c が設定されている場合，length は演算子と判断され，その後ろに続く文字列の長さ（文字数）を返します。

・環境変数 ADSH_CMDEXPR_LENGTH が設定されていない，または b，c 以外の値を設定した場合は，length は演算子として扱われません。

また，length 演算子には式を指定できます。式を指定する場合は，式全体を()（丸括弧）で囲む必要があります。

終了コード

終了コード	意味
0	正常終了。式は空文字列および 0 ではありません。
1	正常終了。式は空文字列または 0 です。
2	エラー終了。式は無効です。
3 以上	エラー終了 • メモリ不足などが発生しました。

注意事項

- 整数値は- 2147483648～2147483647 の範囲で保存します。それより大きな値を指定した場合は，32 ビットの 2 進数であふれた桁は無視して取り出されます。

- 演算子および括弧に指定する文字は，シェルによって解釈される文字を含むため，適切にエスケープする必要があります。式全体をダブルクォーテーション（"）で囲むと文字列として解釈されるため，個々の演算子をダブルクォーテーション（"）で囲む必要があります。
- このコマンドには指定可能なオプションが存在しません。そのため，引数にオプションを指定した場合，そのオプションを式として解釈します。

使用例

- 変数ａと変数ｂの演算をします。

```
$ a=2
$ b=3
$ x=`expr ¥( $a + $b ¥) ¥* 10`
$ echo $?
0
$ echo $x
50
$
```

- 変数ａ｜変数ｂの評価をします。

```
$ a=""
$ b="abcdef"
$ expr "$a" ¥| "$b"
abcdef
$
```

- パス名から拡張子を除いたファイル名を切り出します。

```
$ a='d:¥jp1as¥test.txt'
$ expr $a : '.*¥¥¥(.*¥)¥.'
test
$
```

- 変数に数字が含まれるかどうかを調べます。数字がない場合は０になります。

```
$ a='abcde12345kl'
$ b='abcdefg'
$ expr $a : '.*[0-9].*'
12
$ expr $b : '.*[0-9].*'
0
$
```

- 文字列「テスト文字列」の長さをバイト数で返します。

```
$ export ADSH_CMDEXPR_LENGTH=b
$ echo $LANG
ja_JP.UTF-8
$ expr length "テスト文字列"
18
```

- 文字列「テスト文字列」の長さを文字数で返します。

```
$ export ADSH_CMDEXPR_LENGTH=c
$ echo $LANG
ja_JP.UTF-8
$ expr length "テスト文字列"
6
```

- 文字列「teststring」の長さ（バイト数）に 2 を加算して返します。

```
$ export ADSH_CMDEXPR_LENGTH=b
$ echo $LANG
ja_JP.UTF-8
$ expr length teststring + 2
12
```

8.4.13 find コマンド（ディレクトリ内のファイルを検索する）

形式

find [-d] [-H] [-h] [-L] *パス名* [...] [*検索式*]

機能

検索を開始するパスをパス名に指定し，ディレクトリ階層をたどって，ファイルを検索します。検索の条件および検索したファイルの扱いを，検索式に指定できます。

引数

オプション，検索を開始するパス名および検索式を指定します。検索を開始するパス名は，find コマンドの引数のパス名で指定します。

オプションはハイフン（-）と共に 1 文字のオプション名を指定します。

-d

ディレクトリ内のファイルを階層の深いディレクトリから先に検索します。

-H

引数として指定したパス名がシンボリックリンクだった場合，リンク先が指定されたものとして処理します。リンク先が存在しない場合は，シンボリックリンクが指定されたとして処理します。検索中に遭遇したシンボリックリンクはリンク先を参照しません。-H オプション，-h オプションおよび-L オプションは，最後に指定したオプションが有効となります。

-h

シンボリックリンクは，すべてリンク先を参照して処理を継続します。リンク先が存在しない場合は，シンボリックリンクが指定されたとして処理します。-H オプション，-h オプションおよび-L オプションは，最後に指定したオプションが有効となります。

-L

シンボリックリンクは，すべてリンク先を参照して処理を継続します。リンク先が存在しない場合は，シンボリックリンクが指定されたとして処理します。-H オプション，-h オプションおよび-L オプションは，最後に指定したオプションが有効となります。

パス名

パス名を指定します。

検索式

検索式（expression）は，プライマリおよび演算子を指定します。

- プライマリ

-amin *時間差*

UNIX の場合，ファイルおよびディレクトリの最終アクセス日時と，find が実行を開始した日時の差が，ここで指定された*時間差*（単位：分）のときは真です。日時の差は，1 分未満は切り上げます。

*時間差*は，符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した時間差

・+を前置した場合：指定値より大きい

・-を前置した場合：指定値より小さい

*時間差*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647 となります。

*時間差*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*:*指定した文字列*: illegal numeric value）を出力します。

*時間差*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

Windows の場合，指定するとエラーとなります。

-anewer *パス名*

UNIX の場合，ファイルおよびディレクトリの最終アクセス日時が*パス名*より新しい場合は真です。

Windows の場合，指定するとエラーとなります。

-atime *時間差*

UNIX の場合，ファイルおよびディレクトリの最終アクセス日時と，find が実行を開始した日時の差がここで指定された*時間差*（単位：日）のときは真です。日時の差は，1 日未満は切り上げます。

*時間差*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した時間差

・+を前置した場合：指定値より大きい

・-を前置した場合：指定値より小さい

*時間差*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647 となります。

*時間差*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*:*指定した文字列*: illegal numeric value）を出力します。

*時間差*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

Windows の場合，指定するとエラーとなります。

-cmin *時間差*

UNIX の場合，ファイル情報の最終変更日時（書き込みが発生した，所有者，グループ，リンク数やモードなどが変更された日時）と，find が実行を開始した日時の差が，ここで指定された*時間差*（単位：分）のときは真です。日時の差は，1 分未満は切り上げます。

*時間差*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した時間差

・+を前置した場合：指定値より大きい

・-を前置した場合：指定値より小さい

*時間差*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647 となります。

*時間差*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

*時間差*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

Windows の場合，指定するとエラーとなります。

-cnewer *パス名*

UNIX の場合，ファイル情報の最終変更日時（書き込みが発生した，所有者，グループ，リンク数およびモードなどが最後に変更された日時）が，*パス名*で指定されたファイルより新しい場合は真です。

Windows の場合，指定するとエラーとなります。

-ctime *時間差*

UNIX の場合，ファイル情報の最終変更日時（書き込みが発生した，所有者，グループ，リンク数およびモードなどが最後に変更された日時）と，find が実行を開始した日時の差がここで指定された*時間差*（単位：日）のときは真です。日時の差は，1 日未満は切り上げます。

*時間差*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した時間差

・+を前置した場合：指定値より大きい

・-を前置した場合：指定値より小さい

*時間差*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647 となります。

*時間差*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

*時間差*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

Windows の場合，指定するとエラーとなります。

-depth

階層の深いディレクトリから検索します。ディレクトリ内のファイルを先に処理します。常に真です。

-empty

ファイルやディレクトリが空の場合は真です。

-exec *コマンドライン* ;

検索したファイルおよびディレクトリに対して処理をするコマンドラインを指定します。

・find コマンドを実行するシェルによっては，*，;（セミコロン）などの文字が展開されるため，”（ダブルクォーテーション）または’（シングルクォーテーション）で囲むか，エスケープ文字（¥）を使用する必要があります。

・*コマンドライン*は;（セミコロン）で区切ります。

・*コマンドライン*で指定したプログラムは，find が起動されたディレクトリをカレントディレクトリとして起動します。

・*コマンドライン*に{ }を指定すると，検索したファイルまたはディレクトリのパス名に置き換わります。パス名は，検索を開始するパスを絶対パスで指定した場合は絶対パスに，検索を開始するパスを相対パスで指定した場合は相対パスになります。

・*コマンドライン*で指定したプログラムが終了コード0 で終了した場合，真です。

-follow

常に真です。

シンボリックリンクは，すべてリンク先を参照して処理を継続します。リンク先が存在しない場合は，シンボリックリンクが指定されたとして処理します。

-group *グループ名*

Windows の場合，常に偽となります。

UNIX の場合，ファイルの属するグループが*グループ名*の場合は真です。*グループ名*が数字で，そのグループ名が存在しないときは，グループ ID と解釈します。

-iname *パターン*

-name オプションの説明を参照してください。ただし，英大文字と英小文字を区別しません。

-inum *番号*

Windows の場合，常に偽となります。

UNIX の場合，ファイルのinode 番号が指定した番号のときは真です。

*番号*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した番号

・+を前置した場合：指定値より大きい番号

・-を前置した場合：指定値より小さい番号

*番号*の範囲は9223372036854775807（0x7fffffffffffffff）までで，それ以上を指定しても9223372036854775807 となります。

*番号*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

*番号*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-links *リンク数*

ファイルのリンク数が指定したリンク数のときは真です。

*リンク数*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定したリンク数

・+を前置した場合：指定値より大きい数

・-を前置した場合：指定値より小さい数

*リンク数*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647となります。

*リンク数*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

*リンク数*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-ls

常に真です。

Windowsの場合は，ファイルパーミッション，ハードリンク数，所有者名，サイズ（バイト単位），最終修正日時，およびパス名を標準出力に出力します。ファイルがシンボリックリンクの場合は，リンク先のパス名が「->」のあとに表示されます。

UNIXの場合，inode番号，サイズ（512バイト単位），ファイルパーミッション，ハードリンク数，所有者名，グループ，サイズ（バイト単位），最終修正日時およびパス名を標準出力に出力します。ファイルがスペシャルファイルの場合は，サイズ（バイト単位）の代わりにメジャー番号およびマイナー番号を表示します。ファイルがシンボリックリンクの場合は，リンク先のパス名が「->」のあとに表示されます。

-maxdepth *深さ*

現在検索しているディレクトリの深さが，指定した深さより小さいまたは同じ場合には真です。最初に指定したディレクトリの深さは1です。

・*深さ*の指定範囲は，0から32767までです。0を指定すると，検索対象ディレクトリだけ（ディレクトリに格納されているファイルは対象外）となります。

・指定できる値より大きい値を指定するとエラーとなります（find: *指定値*: maxdepth value too large）。

・*深さ*に数値以外を指定した場合，エラーメッセージ（find: *指定した文字列*: *プライマリ*: value invalid）を出力します。

・*深さ*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-mindepth *深さ*

現在検索しているディレクトリの深さが指定した深さ以上の場合には真です。

・*深さ*の指定範囲は，0から32767までです。

・指定できる値より大きい値を指定してもエラーになりません。

・*深さ*に数値以外を指定した場合，0 が指定されたことになります。

・*深さ*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-mmin *時間差*

ファイルおよびディレクトリの最終修正日時と，find が実行を開始した日時の差が*時間差*で指定された分のときは真です。日時の差は，1 分未満は切り上げます。*時間差*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した時間差

・+を前置した場合：指定値より大きい

・-を前置した場合：指定値より小さい

*時間差*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647 となります。

・*時間差*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

・*時間差*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-mtime *時間差*

ファイルおよびディレクトリの最終修正日時と，find が実行を開始した日時の差が*時間差*で指定された日のときは真です。日時の差は，1 日未満は切り上げます。*時間差*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定した時間差

・+を前置した場合：指定値より大きい

・-を前置した場合：指定値より小さい

*時間差*の範囲は2147483647（0x7fffffff）までで，それ以上を指定しても2147483647 となります。

・*時間差*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

・*時間差*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-mount

常に真です。UNIX の場合，検索を開始したディレクトリのデバイス番号と異なるディレクトリは，検索しないようにします。

-name *パターン*

検索するファイル名またはディレクトリ名を*パターン*で指定します。検索したファイル名またはディレクトリ名が*パターン*に一致する場合は真です。

*パターン*は，文字とワイルドカードの組み合わせで指定します。ワイルドカードで使用する文字を指定するために，エスケープ文字（¥）を使用できます。また，ワイルドカードで使用する文字以外もエスケープ文字（¥）を使用できます。この場合，そのまま¥が無視されたように見えます。

ワイルドカードとして使用できる文字を次の表に示します。

ワイルドカード	意味
?	任意の 1 文字に合致します。
*	0 文字以上の文字列に合致します。
[…]	[ ]に囲まれた文字列のどれか 1 文字に合致します。[ ]に囲まれた文字列の先頭に!または^を付けた場合，[ ]に囲まれていない文字に合致します。-（ハイフン）で区切るとハイフンで区切られた，間にある任意の文字（その 2 文字も含む）に合致します。

ワイルドカード[ ]の記述例を次の表に示します。

記述例	意味
[!abc]	a，b，c の 3 つを除く文字と合致します。
[0-9]	0 から9 までのどれかに合致します。
[a-z]	英小文字に合致します。
[A-Z]	英大文字に合致します。
[0-9a-zA-Z]	英数字に合致します。

-newer *パス名*

現在のファイルおよびディレクトリが，*パス名*の最終修正日時より新しい場合は真です。

-nogroup

Windows の場合，常に偽となります。

UNIX の場合，現在のファイルが，存在しないグループに属している場合に真です。

-nouser

Windows の場合，常に偽となります。

UNIX の場合，現在のファイルの所有者が存在していないユーザーの場合に真です。

-ok *コマンドライン* ;

検索したファイルおよびディレクトリに対して処理をするコマンドラインを指定します。

・find コマンドを実行するシェルによっては，*，;（セミコロン）などの文字が展開されるため，”（ダブルクォーテーション）または’（シングルクォーテーション）で囲むか，エスケープ文字（¥）を使用する必要があります。

・*コマンドライン*は;（セミコロン）で区切ります。

・*コマンドライン*で指定したプログラムは，find が起動されたディレクトリをカレントディレクトリとして起動します。起動する前に，ユーザーに応答を求めます。標準入力からy が入力されない場合，コマンドラインを実行しないで，偽を返します。

・*コマンドライン*に{ }を指定すると，検索したファイルまたはディレクトリのパス名に置き換わります。パス名は，検索を開始するパス名を絶対パスで指定した場合は絶対パスに，検索を開始するパスを相対パスで指定した場合は相対パスになります。

・*コマンドライン*で指定したプログラムが終了コード0 で終了した場合，真です。

-path パターン

検索するファイル名またはディレクトリ名のパス名をパターンで指定します。検索したファイルまたはディレクトリのパス名がパターンに一致する場合は真です。

・パターンには，指定した文字とワイルドカードの組み合わせで指定します。ワイルドカードで使用する文字そのものを指定するために，エスケープ文字（¥）を使用できます。ワイルドカードで使用する文字以外に使用した場合，¥が無視されたように見えます。

・パターンの指定の詳細は，-name パターンの説明を参照してください。

-perm [-]パーミッション

UNIX の場合，パーミッションを 8 進数の数値またはシンボルで指定します。Windows でこの引数を指定するとエラー（find: -perm: unknown option）になります。

パーミッションをハイフン（-）に続いて指定した場合，ファイルまたはディレクトリのモードのうちパーミッションで指定された値が設定されていると真になります。ハイフンが指定されない場合は，パーミッションとファイルのモードが完全に一致したときに真になります。

パーミッションを数値で指定した場合，8 進数以外または 8 進数の07777（10 進数の4095）より大きな値を指定するとエラーとなります。

パーミッションをシンボルで指定した場合，何も指定されていない状態（数値表現での0）に対して設定，追加および削除をします。1 つまたは複数のシンボルで指定された結果が検索に使用されます。

シンボルは 3 つの部分から構成されます。次に示すシンボルを 1 つまたは複数指定します。複数指定する場合は，コンマ（,）でシンボル間を区切ります。

シンボル内の順序	指定できる値
1 つ目	アクセス権を設定する項目を指定します。複数同時に指定できます。 指定できる項目を次に示します。省略するとすべてのユーザーが仮定されます。 u：所有者 g：グループ o：その他 a：全ユーザー
2 つ目	モードに対する操作を指定します。1 つ目のシンボルで指定した項目に対して次の処理をします。 =：アクセス権の設定（上書き） +：アクセス権の追加 -：アクセス権の削除 設定，追加および削除する値は，3 つ目のシンボルで指定します。 3 つ目のシンボルに続いて 2 つ目および 3 つ目のシンボルを記述できます。3 つ目のシンボルは省略できます。
3 つ目	設定するアクセス権を指定します。複数同時に指定できます。指定できる値を次に示します。 r：読み取り w：書き込み x：実行 s：実行時にユーザーまたはグループ ID を設定する t：スティッキービット u：モードに現在設定されている所有者のアクセス権

シンボル内の順序	指定できる値
3つ目	g：モードに現在設定されているグループのアクセス権 o：モードに現在設定されているその他のアクセス権 省略するとアクセス権を設定する項目を消去します。消去した値を2つ目のシンボルに従って設定，追加および削除します。追加および削除だけでは値は変化しません。 s とt の指定は，1つ目でo だけを指定した場合には無視されます。

シンボルの指定例を次の表に示します。

-perm の指定値	同等の数値指定	説明
u=x,g=w	120	u に対してx を設定し，g に対してw を設定しています。
u=x,g=u	110	u に対してx を設定し，g に対してu と同じ値を設定しています。
u=x,=u	111	u に対して x を設定し，そのあとa（省略値）にu と同じ値を設定しています。
u=x,u=w	200	u に対してx を設定し，その後u に対してw を設定（上書き）しています。
u=x,u+w	300	u に対してx を設定し，その後u に対してw を追加しています。
ug=x	110	u とg に対してx を設定しています。
u=rw	600	u に対してr およびw を設定しています。
u=r+x	500	u に対してr を設定し，x を追加しています。
u=r=w	200	u に対してr を設定し，さらに w を設定（上書き）しています。
=x,u=	011	a（省略値）にx を設定し，u の設定を消去しています。
=	000	a（省略値）を消去しています。

-print

検索したファイルまたはディレクトリのパス名を標準出力に出力して改行します。常に真です。

-print0

検索したファイルまたはディレクトリのパス名とNULL（'¥0'）を標準出力に出力します。常に真です。

-prune

検索中に遭遇したディレクトリはたどらないようにします。常に真です。-d オプションが指定されている場合は無効となります。

-size *サイズ* [c]

ファイルのサイズが，指定したサイズブロック（512バイト単位に切り上げ）の場合は真です。*サイズ*のあとにc を指定するとバイト単位で評価します。

*サイズ*は符号を指定しないか，+または-の符号を付けた数値を指定することで，次のように扱われます。

・符号を指定しない場合：指定したサイズ

・+を前置した場合：指定値より大きいサイズ

・-を前置した場合：指定値より小さいサイズ

*サイズ*の範囲は9223372036854775807（0x7fffffffffffffff）までで，それ以上を指定しても9223372036854775807となります。

*サイズ*に数値以外を指定した場合，エラーメッセージ（find: *プライマリ*: *指定した文字列*: illegal numeric value）を出力します。

*サイズ*を指定しなかった場合，エラーメッセージ（find: *プライマリ*: requires additional arguments）を出力します。

-type *タイプ*

現在のファイルのタイプが指定した*タイプ*と等しい場合は真です。*タイプ*を次に示します。次のタイプ以外を指定した場合は，エラーメッセージ（find: -type: *指定した値*: unknown type）が出力されます。

・b：ブロック型スペシャルファイル（Windowsでは指定できません）

・c：キャラクタ型スペシャルファイル（Windowsでは指定できません）

・d：ディレクトリ

・f：通常ファイル

・l：シンボリックリンク

・p：FIFO（Windowsでは指定できません）

・s：ソケット（Windowsでは指定できません）

-user *ユーザー名*

Windowsの場合，ファイルの所有者が*ユーザー名*のときは真です。

UNIXの場合，ファイルの所有者が*ユーザー名*のときは真です。*ユーザー名*に数値を指定し，その所有者名が存在しないときはユーザーIDとして評価します。

-xdev

常に真です。UNIXの場合，検索を開始したディレクトリのデバイス番号と異なるディレクトリは，検索しないようにします。

- 演算子

プライマリは次の演算子と共に使用できます。優先度の高い順に示します。

(*検索式*)

括弧演算子内の検索式が条件を満たす場合，真です。

! *検索式*

!演算子に続く検索式が条件を満たす場合，偽です。

検索式 -and *検索式*|*検索式* -a *検索式*|*検索式* *検索式*

検索式を-and演算子もしくは-a演算子で接続する，または検索式を2つ並べると論理積となります。2つの検索式が真の場合，真です。最初の検索式が偽の場合，2つ目の検索式は評価されません。

検索式 -or *検索式*|*検索式* -o *検索式*

検索式を-or演算子または-o演算子で接続すると論理和になります。どちらかの検索式が真の場合，真です。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- find を実行するシェルによっては，セミコロンや括弧などにエスケープ文字（¥）を使用するか，シングルクォーテーション（'）またはダブルクォーテーション（"）で囲む必要があります。
- 検索したファイルやディレクトリの出力順序は，OS やファイルシステムによって異なります。そのため，複数プラットフォームでの動作の一貫性を期待する場合は，出力結果を sort する必要があります。
- Windows の場合，-exec プライマリなどで生成したプロセスにファイルディスクリプタが引き継がれないで，クローズされた状態になります。例えば，親プロセスがオープンしていたファイルディスクリプタに対して再度オープンしないで入出力を行おうとするとエラーになります。ただし，標準入力，標準出力および標準エラー出力は再度オープンされた状態になります。
- Windows の場合，-exec オプションに指定したプログラム名にパスが含まれていないときは，プログラムを実行する Windows API のパス検索順序で見つかったプログラムが実行されます。
- Windows の場合，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。検索を開始するパス名をシンボリックリンクのリンク先のディレクトリにしたい場合は，-H，-L オプションまたは，-follow プライマリを指定してください。

使用例

- 「.c」で終わる名称のファイルやディレクトリを表示します。

```
$ find . -name '*.c'
./test/a.c
./test/b.c
./test/c.c
./test/abc.c
$
```

- ファイルttt より古い，または所有者がroot ではないファイルやディレクトリを表示します。

```
$ ls -l
合計 0
-rw-rw-r-- 1 user1 group1 0 10月  7 10:12 a.c
-rw-rw-r-- 1 root  group1 0 10月  7 10:12 abc.c
-rw-rw-r-- 1 user1 group1 0 10月  7 10:12 b.c
-rw-rw-r-- 1 user1 group1 0 10月  7 10:10 c.c
-rw-rw-r-- 1 user1 group1 0 10月  7 10:11 ttt
$ find . ¥! ¥( -newer ttt -user root ¥)
.
./ttt
./b.c
./a.c
```

```
./c.c
$
```

- カレントディレクトリの下にある，ファイル名がドット(.)と1桁の数字で終わるファイルを表示します。ただし，command1ディレクトリはスキップします。

```
$ ls command1 command2
command1:
a1.txt  b1.txt  command1  command1.1  command1.c  command1.o  extern.h  obj

command2:
a2.txt  b2.txt  command2  command2.1  command2.c  command2.o  extern.h  obj
$ find . ! -path ’./command1/*’ -name ’*.[0-9]’
./command2/command2.1
$
```

- カレントディレクトリの下にある，すべての*.oファイルを削除します。

```
$ ls command1 command2
command1:
a1.txt  b1.txt  command1  command1.1  command1.c  command1.o  extern.h  obj

command2:
a2.txt  b2.txt  command2  command2.1  command2.c  command2.o  extern.h  obj
$ find . -name ’*.o’ -exec rm {} ¥;
$ ls command1 command2
command1:
a1.txt  b1.txt  command1  command1.1  command1.c  extern.h  obj

command2:
a2.txt  b2.txt  command2  command2.1  command2.c  extern.h  obj
$
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windowsの例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥find -w
find: illegal option -- w
usage: find [-dHhL] path ... [expression]
```

8.4.14 getoptコマンド（コマンドラインのオプションを解析する）

形式1

```
getopt 解析するオプション 解析される引数
```

形式2

```
getopt [オプション] [--] 解析するオプション 解析される引数
```

形式 3

```
getopt [オプション] -o 解析するショートオプション名 [オプション] [--] 解析される引数
```

機能

解析される引数に指定されたコマンドラインを**解析するオプション**の指定内容で解析し，結果を標準出力に出力します。ショートオプション，ロングオプションとも解析できます。これによって，シェルスクリプトでの構文解析が容易になります。

次のどちらかで指定された場合は，形式 1 で指定されていると判定されます。

- 引数の最初のパラメーターが「-」で始まっていない場合
- 環境変数 GETOPT_COMPATIBLE が設定されている場合

これ以外の場合は，-o オプションの指定があれば形式 3，指定がなければ形式 2 で指定されていると判定されます。

引数

解析するオプション

解析するオプションの文字列を指定します。

ショートオプションを指定する場合は次の点に注意してください。

- ショートオプションの先頭が「-」の場合，オペランドは出現個所で出力されます。ショートオプションの先頭が「+」の場合，オプションはオペランドの前に指定必須となります。

  ただし，形式 1 では先頭にある「-」「+」は無視されます。
- ショートオプションの先頭（「-」「+」の場合はその次の文字）が「:」の場合，解析される引数のオプションが，解析するオプションに指定されていなくてもエラーメッセージは出力されませんが，終了コードはエラーとなります。解析される引数は継続して解析されます。

なお，オプション文字またはオプション名の後ろに「:」または「::」を指定すると，次の内容を定義できます。

オプション文字またはオプション名の後ろに「:」を指定した場合

オプション値が必須なオプションであることを示します。

この場合，解析される引数には，オプション名とオプション値を次の書式で指定します。

・ショートオプション名とオプション値を続けて指定します。

```
$ getopt "xy:z" -yarg
-y arg --
```

・ショートオプション名とオプション値の間にスペースを入れて指定します。

```
$ getopt "xy:z" -y arg
-y arg --
```

・ロングオプション名とオプション値の間にスペースを入れて指定します。

```
$ getopt -o ”abc” -l xyz: -- --xyz nml
--xyz ’nml’ --
```

・「**ロングオプション名=オプション値**」の形式で指定します。

```
$ getopt -o ”abc” -l xyz: -- --xyz=nml
--xyz ’nml’ --
```

オプション文字またはオプション名の後ろに「::」を指定した場合

オプション値の指定が任意のオプションであることを示します。引数を指定する場合は，オプション文字またはオプション名の直後に指定する必要があります。

この場合，解析される引数には，オプション名とオプション値を次の書式で指定します。

・ショートオプションの場合

オプション名とオプション値を続けて指定します。

・ロングオプションの場合

「**ロングオプション名=オプション値**」の形式で指定します。

解析される引数

解析される引数として，オプション名，オプション値，オペランドを指定します。オプションの指定については，「8.1　コマンドの記述形式」を参照してください。

オプションの指定方法

ショートオプションの場合は「-」の後ろにショートオプション名を指定し，ロングオプションの場合は「--」の後ろにロングオプション名を指定します。「--」の後ろにロングオプション名が続かない場合は，その個所でオプションの解析を終了し，以降のパラメーターをオペランドと判断します。

ロングオプション名は短縮して指定できます。その場合，出力結果にはロングオプションの完全名が出力されます。例を次に示します。

```
$ getopt -o ”” -l ”longZ” -- --lo
 --longZ --
```

短縮指定の場合はほかのオプションと区別できるように指定してください。あいまいな指定をすると，**解析するオプション**で先に定義したオプション名と判断されます。

オプションとオペランドの指定順

デフォルトではオプションとオペランドの指定順に決まりはなく，オペランドの後ろに指定したオプションも解析されます。なお，次の環境変数を設定すると，オプションの指定順を設定できます。

・環境変数 ADSH_CMD_ARGORDER によるオプション指定順の設定

環境変数 ADSH_CMD_ARGORDER を設定すると，解析される引数に指定するオプションは，オペランドの前での指定が必須となります。

・環境変数 POSIXLY_CORRECT によるオプション指定順の設定

環境変数 POSIXLY_CORRECT を設定すると，解析される引数に指定するオプションは，オペランドの前での指定が必須となります。

環境変数 ADSH_CMD_ARGORDER に記述したコマンドのほかに，Linux 版のコマンドでも有効となります。

オプション

形式 2 と形式 3 では次のオプションを指定できます。

- -l ロングオプション名
- -longoptions=ロングオプション名

  解析するロングオプションを指定します。

  オプションを複数指定する場合は，「,」（コンマ）またはスペースで区切って一度に指定するか，このオプションで複数回指定します。
- -n プログラム名
- --name=プログラム名

  オプション解析のエラーメッセージで出力するコマンド名を「getopt」からここで指定したプログラム名へ変更します。
- -q
- --quiet

  解析される引数のオプション解析のエラーメッセージを抑止します。
- -Q
- --quiet-output

  解析結果の出力を抑止します。
- -u
- --unquoted

  形式 2，形式 3 の解析結果のオプション値とオペランドをクォーテーションで囲まないようにします。
- -o 解析するショートオプション名
- --options=ショートオプション名

  解析するショートオプションを指定します。

  複数回指定した場合，最後に指定した値が有効になります。

  ショートオプションに「W;」を指定し，解析される引数に「-W」指定のロングオプション名を指定した場合，ロングオプション名として解釈されます。

解析結果の出力

解析結果は，オプション名，オプション値，オペランドに分類され，標準出力に出力されます。

形式 2 または形式 3 で指定した場合は，オプション値およびオペランドはシングルクォーテーションで囲まれます。-u を指定した場合は囲まれません。

ショートオプションの場合は「-」とオプション名，ロングオプションの場合は「--」と完全なオプション名が，1 つのオプションとして出力されます。

オプション（オプション名・オプション値）とオペランドの間には，区切りとして「--」が出力されます。ただし，ショートオプションの先頭に「-」を指定した場合はこの限りではありません。

オプション解析でエラーが発生しても処理を継続し，後続のパラメーターのオプションを解析します。例を次に示します。

```
$ getopt ”xyz” -w -x
getopt: invalid option -- w
 -x --
```

終了コード

終了コード	意味
0	正常終了
1	エラー終了 • **解析される引数**に指定したオプションが，**解析するオプション**に定義されていません。
2	エラー終了 • getopt コマンドのオプションが不正です。
3	エラー終了 • 終了コード 1，2 以外のエラーです。

使用例

- コマンドを形式 1 で実行する例を示します。

```
$ getopt ”xy:z” -z -y arg1 arg2
 -z -y arg1 -- arg2
```

- コマンドを形式 2 で実行する例を示します。

```
$ getopt -q ”xy:z” -z -y arg1 arg2
 -z -y ’arg1’ -- ’arg2’
```

- コマンドを形式 3 で実行する例を示します。

```
$ getopt -o ”xy:z” -- -z -y arg1 arg2
 -z -y ’arg1’ -- ’arg2’
```

- 解析結果を位置パラメーターに設定する例を示します。

  実行ファイル

```
OPT=`getopt -o abc:d: -- -a -b`
eval set -- ”$OPT”
echo $1
echo $2
echo $3
```

  実行結果

```
-a
-b
--
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥> getopt -z
getopt: illegal option -- z
usage: getopt optstring parameters
       getopt [options] [--] optstring parameters
       getopt [options] -o optstring [options] [--] parameters
```

8.4.15　grep コマンド（ファイル内の文字を検索する）

形式

```
grep [-a] [-b] [-c] [-E] [-G] [-h] [-I] [-i] [-L] [-l] [-n]
     [-q] [-R] [-r] [-s] [-U] [-v] [-w] [-x]
     [-A 数値] [-B 数値] [-C [数値] ]
     [-e パターン] [-f パターンファイルパス名] [パターン] [パス名 ...]
```

機能

ファイル内の文字（指定したパターン）を検索します。

引数

-a

　すべてのファイルを ASCII テキストファイルとして扱います。

-b

　それぞれ一致した行の先頭にバイト単位のオフセットを出力します。

-c

　選択された行数だけ標準出力に出力します。

-E

　拡張された正規表現としてパターンを扱います。-E オプションおよび-G オプションは最後に指定したオプションが有効となります。

-G

　パターンを正規表現として扱います。デフォルト値です。-E オプションおよび-G オプションは最後に指定したオプションが有効となります。

-h

次のどちらかの指定をする場合，各出力行の先頭にファイル名を付けないようにします。

- -R または-r オプションを指定する
- 複数の検索対象パス名を指定する

-I

バイナリファイルを無視します。

-i

大文字と小文字を区別しません。

-L

パターンを含まないファイルの名前だけを標準出力に出力します。-L オプションおよび-l オプションは，最後に指定したオプションが有効となります。

-l

パターンを含むファイルの名前だけを標準出力に出力します。-L オプションおよび-l オプションは，最後に指定したオプションが有効となります。

-n

各出力行にファイルの相対的な行番号を表示します。-c オプション，-L オプション，-l オプションおよび-q オプションを指定した場合は無視されます。

-q

標準出力には何も出力しません。

-R|-r

検索ディレクトリを再帰的に検索します。

なお，-L オプション，-l オプション，および-q オプションを指定しない場合は，各出力行の先頭にファイル名が付けられます。

Windows の場合，このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。

-s

読めないファイルや存在しないファイルは無視します。エラーメッセージを抑止します。

-U

バイナリファイルを検索します。ただし，出力はしません。

-v

パターンに一致しなかった行を出力します。

-w

指定文字列が単語として含まれている行を出力します。

単語とは英数字およびアンダースコア（_）から構成される文字列のことです。また，単語の前後はスペースなどの単語構成文字列以外の文字や，行頭または行末で区切られている必要があります。

-x

指定した文字列とファイルのすべての行を 1 行ごとに比較して完全に一致した場合に，一致した回数だけ指定した文字列を出力します。

-A 数値

数値で指定した行だけ，パターンにマッチした行のあとの行も出力します。

-B 数値

数値で指定した行だけ，パターンにマッチした行の前の行も出力します。

-C ［数値］

数値で指定した行だけ，パターンにマッチした行の前後の行も出力します。数値を省略した場合，前後 2 行を表示します。この場合，「-A 2 -B 2」と指定したときと同じになります。

-C オプションに数値を指定する場合は，-C オプションと数値の間にスペースを入れないでください。

パターン｜-e パターン

検索するパターンを指定します。-e オプションは複数指定できます。

-e オプションには，'-'で始まるパターンを指定できます。

-f パターンファイルパス名

検索するパターンを 1 行ごとにパターンファイルのパス名に指定します。パターンの指定がない場合はマッチしません。

［パス名 ...］

検索対象のパス名を指定します。複数指定ができます。パス名を指定しない場合は，検索対象の内容を標準入力から入力できます。ディレクトリ名の指定は，-R オプションまたは-r オプションを指定した場合に有効です。

なお，-L オプション，-l オプション，および-q オプションを指定しない場合は，各出力行の先頭にファイル名が付けられます。

終了コード

終了コード	意味
0	正常終了。 • パターンを含む行が存在します。 • -v が指定されている場合は，パターンを含まない行が存在します。
1	正常終了。 • パターンを含む行が存在しません。 • -v が指定されている場合は，パターンを含まない行が存在しません。
2 以上	エラー終了

注意事項

- ファイルの先頭から 8,192 バイト内に表示できる 1 バイト文字，スペース，タブ，バックスペースおよびマルチバイト文字以外のデータが含まれている場合は，バイナリファイルと見なされます。

- Windows のコマンドプロンプトから実行する場合，パターンをクォーテーションで囲むときは"（ダブルクォーテーション）を使用してください。
- ロケールと異なる文字コードのファイルはバイナリファイルと見なされます。
- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。

使用例

- オプションを指定しない場合のデフォルトを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep ABCD test1.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]cccccccccc
555555555:ABCD:111111111
ABCD
ABCD_XYZ
0000<ABCD>0000
/* ABCD */
```

- 複数ファイルを指定して，オプションを指定しない場合のデフォルトを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep ABCD_ test1.txt test2.txt test3.txt test4.txt
test1.txt:ABCD_XYZ
test2.txt:ABCD_XYZ
test3.txt:ABCD_XYZ
test4.txt:ABCD_XYZ
```

- -h オプションを指定して，複数のファイルを検索したときにファイル名を付加しないで一致した行を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -h ABCD test1.txt test2.txt test3.txt test4.txt
ABCD_XYZ
ABCD_XYZ
ABCD_XYZ
ABCD_XYZ
```

- -b オプションを指定して，一致した行の先頭にバイト単位のオフセットを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -b ABCD test1.txt
77:ABCDEFGHIJKLMNOPQRSTUVWXYZ
104:77777777[ABCD]cccccccccc
133:555555555:ABCD:111111111
212:ABCD
256:ABCD_XYZ
301:0000<ABCD>0000
316:/* ABCD */
```

- -c オプションを指定して，一致した行数だけ表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -c ABCD test1.txt
7
```

- -i オプションを指定して，大文字と小文字を区別しない指定をした場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -i AbCd  test1.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
555555555:ABCD:111111111
abcdefghijklmnopqrstuvwxyz
ABCD
abcd
ABCD_XYZ
0000<ABCD>0000
/* ABCD */
```

- -L オプションを指定して，パターンを含まないファイル名だけを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -L ABC_  test1.txt test2.txt test3.txt test4.txt
test1.txt
test2.txt
test4.txt
```

- -l オプションを指定して，パターンを含むファイル名だけを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -l ABC_ test1.txt test2.txt test3.txt test4.txt
test3.txt
```

- -n オプションを指定して，各出力行にファイルの相対的な行番号を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -n ABCD test1.txt
4:ABCDEFGHIJKLMNOPQRSTUVWXYZ
5:77777777[ABCD]ccccccccc
7:555555555:ABCD:111111111
10:ABCD
14:ABCD_XYZ
17:0000<ABCD>0000
18:/* ABCD */
```

- -q オプションを指定して，標準出力に何も表示しない指定をした場合を表示します。上段は-q オプションを指定しない場合，下段は-q オプションを指定した場合です。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep ABCD_XYZ test1.txt
ABCD_XYZ

C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -q ABCD_XYZ test1.txt
```

- -R オプションを指定して，検索ディレクトリを再帰的に検索した場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -R ABCD C:¥USR¥data
C:¥USR¥data¥data_2¥data_3¥test3.txt:ABCDEFGHIJKLMNOPQRSTUVWXYZ
C:¥USR¥data¥data_2¥data_3¥test3.txt:ABCD333
C:¥USR¥data¥data_2¥data_3¥test3.txt:ABCD_AS
C:¥USR¥data¥data_2¥test2.txt:77777777[ABCD]ccccccccc
C:¥USR¥data¥data_2¥test2.txt:555555555:ABCD:111111111
C:¥USR¥data¥data_2¥test2.txt:ABCD222
C:¥USR¥data¥data_2¥test2.txt:ABCD_MM
C:¥USR¥data¥test0.txt:ABCD_1118
C:¥USR¥data¥test0.txt:ABCD_AS321
C:¥USR¥data¥test0.txt:0000<ABCD>0000
C:¥USR¥data¥test0.txt:/* ABCD */
```

- -s オプションを指定して，エラーメッセージを抑止した場合を表示します。上段は-s オプションを指定しない場合，下段は-s オプションを指定した場合です。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep ABCD  test5.txt
grep: test5.txt: No such file or directory

C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -s ABCD  test5.txt
```

- -w オプションを指定して，パターンが独立している場合だけ表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -w ABCD test1.txt
77777777[ABCD]ccccccccc
555555555:ABCD:111111111
ABCD
0000<ABCD>0000
/* ABCD */
```

- -x オプションを指定して，1 行に指定文字列だけある場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -x ABCD test1.txt
ABCD
```

- ファイル（file.txt）の内容で，-x オプションを指定した場合を表示します。
  - file.txt の内容

    ABABAB

    ACACACAC

    ABABAB

  次のコマンドを実行した場合，一致しないため何も表示されません。

```
grep -x ABA file.txt
```

  次のコマンドを実行した場合，ファイル（file.txt）の 1 行目と 3 行目が一致し，次のように表示されます。

```
grep -x ABABAB file.txt
ABABAB
ABABAB
```

- -A オプションで 3 を指定し，一致した行の後ろ 3 行も表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -A 3 XYZ  test1.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
-XYZ
555555555:ABCD:111111111
abababababababababababab
abcdefghijklmnopqrstuvwxyz
--
ABCD_XYZ
asasasasasasasasas01
ASASASASASASAS
0000<ABCD>0000
```

- -B オプションで 3 を指定し，一致した行の前 3 行も表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -B 3 XYZ  test1.txt
/*----------------------*/
ABABABABABABABABABABABAB
01234567890123456789 0
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]cccccccccc
-XYZ
--
JJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKK
abcd
ABCD_XYZ
```

- -C オプションを指定し，一致した行の前後 2 行も表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -C XYZ test1.txt
ABABABABABABABABABABABAB
01234567890123456789 0
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]cccccccccc
-XYZ
555555555:ABCD:111111111
abababababababababababab
--
KKKKKKKKKKKKKKKK
abcd
ABCD_XYZ
asasasasasasasasas01
ASASASASASASAS
```

- -e オプションで，-で始まるパターンを指定した場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -e ”-rw-” file01.txt
-rw------- user0001    12 May 12 17:19 a.txt
-rw------- user0001    79 May 12 20:36 abc.txt
-rw------- user0001   141 May 12 20:36 abcd.txt
-rw------- user0001    12 May 12 18:05 b.txt
-rw------- user0001   133 May 12 21:49 f01.txt
-rw------- user0001     0 May 12 19:42 ff
-rw------- user0001     0 May 12 20:54 ff.txt
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -d
grep: illegal option -- d
usage: grep [-abcEGhIiLlnqRrsUvwx] [-A num] [-B num] [-C[num]]
        [-e pattern] [-f file] [pattern] [file ...]
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep CHECK  file99
grep: file99: No such file or directory
```

8.4.16　gunzip コマンド（圧縮されたファイルを伸長する）

形式

【Windows 限定】

```
gunzip [-c] [-f] [-k] [-l] [-N] [-n] [-q] [-r] [-t] [-v]
       [-o 出力先パス名] [-S サフィックス] [対象パス名 ...]
```

【UNIX 限定】

```
gunzip [-a] [-c] [-f] [-k] [-l] [-N] [-n] [-q] [-r] [-t] [-v]
       [-o 出力先パス名] [-S サフィックス] [対象パス名 ...]
```

機能

圧縮されたファイルを伸長します。

このコマンドでは，次の操作ができます。

操作の種類	説明
伸長	圧縮ファイルを伸長します（以降，この操作を伸長操作と呼びます）。 この操作は，-d オプションを指定したgzip コマンドと同じです。 -l オプション，および-t オプションを指定しない場合，伸長操作をします。 伸長操作では，伸長したファイルを作成したあとに，圧縮ファイルを削除します。圧縮ファイルを削除させない方法については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「機能」を参照してください。
表示	-l オプションを指定することで，圧縮ファイルの情報を表示します（以降，この操作を表示操作と呼びます）。この操作は，-l オプションを指定したgzip コマンドと同じです。
検証	-t オプションを指定することで，圧縮ファイルを伸長しないで，圧縮ファイルに格納されている圧縮データの整合性を検証します（以降，この操作を検証操作と呼びます）。 この操作は，-t オプションを指定したgzip コマンドと同じです。

各操作の内容については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「機能」を参照してください。

環境変数GZIP にgunzip コマンドのオプションを設定することで，コマンド引数に指定しなくてもオプションを使用できます。環境変数GZIP については，「2.5　環境変数を設定する」を参照してください。

引数

操作オプションの指定

実行する操作を指定します。

次のオプションを指定しないでこのコマンドを実行した場合は，圧縮ファイルを伸長します（伸長操作）。

-l

--list

圧縮ファイルの情報を表示します（表示操作）。

圧縮する前のファイルサイズや圧縮率などの情報を表示します。

-v オプションを同時に指定すると，圧縮ファイルの詳細情報も表示します。

表示内容については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「圧縮ファイル情報の表示」を参照してください。

圧縮する前のファイルサイズの表示では，ファイルサイズが 4GB 以上の場合，値が正しく表示されません。正しいファイルサイズを表示したい場合は，-t オプションを同時に指定してください。ただし，-t オプションを同時に指定した場合は，表示されるまでに時間が掛かる場合があります。

-t

--test

圧縮ファイルを伸長しないで，圧縮ファイルに格納されている圧縮データの整合性を検証します（検証操作）。

-l オプションと同時に指定した場合は，圧縮データの整合性検証と，圧縮ファイル情報の表示の両方を行います。

ファイル情報の復元動作の指定

ファイル情報の復元動作を指定します。

-N オプションおよび-n オプションを同時に指定した場合，最後に指定したオプションが有効になります。-N オプション，-n オプションのどちらも指定しなかった場合，-n オプションが有効になります。-N オプションまたは-n オプションと，-t オプションを同時に指定した場合，指定を無視します。

なお，伸長操作での最終修正日時の扱いについては，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「ファイルの最終アクセス日時と最終修正日時の扱い」を参照してください。表示操作でのファイル名と最終修正日時の扱いについては，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「圧縮ファイル情報の表示」を参照してください。

-N

--name

【伸長操作の場合】

伸長するときに作成するファイル名を，圧縮ファイルに格納されているファイル名とします。また，圧縮ファイルに格納されている最終修正日時を，作成したファイルに設定します。圧縮ファイルに

ファイル名や最終修正日時が格納されていない場合は，-n オプションを指定したときと同じ動作をします。

【表示操作の場合】

表示する圧縮ファイル情報に，圧縮ファイルに格納されているファイル名と最終修正日時を表示します。

-n

--no-name

【伸長操作の場合】

伸長するときに作成するファイル名を，圧縮ファイル名から圧縮ファイルの拡張子を取り除いたファイル名とします。また，圧縮ファイルの最終修正日時を作成したファイルに設定します。

【表示操作の場合】

表示する圧縮ファイル情報に，圧縮ファイル名から圧縮ファイルの拡張子を取り除いたファイル名と，圧縮ファイルの最終修正日時を表示します。

入出力の指定

-c

--stdout

--to-stdout

【伸長操作の場合】

圧縮データの伸長結果を標準出力に出力します。

-l オプションまたは-t オプションと同時に指定した場合は指定を無視します。

-k

--keep

【伸長操作の場合】

伸長ファイルを作成したあと，圧縮ファイルを削除しません。

-l オプション，-t オプション，または-c オプションと同時に指定した場合は指定を無視します。

-r

--recursive

*対象パス名*に指定したパスがディレクトリの場合，ディレクトリおよびサブディレクトリ内の圧縮ファイルを検索します。

Windows の場合，-r オプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。

検索中に遭遇したファイルを圧縮ファイルとして扱うかどうかは，ファイル名の拡張子で決定します。圧縮ファイルとして扱うファイル名の拡張子については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「圧縮ファイルの拡張子の扱い」を参照してください。

ディレクトリのシンボリックリンクは，操作の種類や指定するオプションでリンク先をたどります。ディレクトリのシンボリックリンクを指定したときの動作については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「リンクファイルの扱い」を参照してください。

-o *出力先パス名*

--output=*出力先パス名*

【伸長操作の場合】

指定したパス名で伸長するファイルを作成します。また，伸長操作では，圧縮ファイルが削除されます。伸長操作で-N オプションを同時に指定した場合，作成する伸長ファイルの最終修正日時には圧縮ファイルに格納されている最終修正日時を使用します。ファイル名は*出力先パス名*に従います。

このオプションは，次のどれかと同時に指定するとエラー終了します。

- -c オプション，-t オプション，-l オプションまたは-r オプションを指定する。
- 圧縮ファイルを複数指定する。
- 圧縮ファイルを標準入力から入力する。

対象パス名

圧縮ファイルのパス名を指定します。

*対象パス名*に，圧縮ファイル以外のファイルを指定したときの扱いについては，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「圧縮ファイルの拡張子の扱い」を参照してください。

*対象パス名*は複数指定できます。

*対象パス名*の指定がない，または*対象パス名*に「-」を指定した場合は，標準入力から圧縮データを入力します。伸長操作では，標準入力から圧縮データを入力する場合，圧縮データの伸長結果を標準出力に出力します。なお，伸長操作では，標準入力が端末に関連づけられているとエラーとなります。

*対象パス名*に指定できるファイルの種類は次のとおりです。

- 伸長操作では，通常ファイルだけを指定できます。また，リンクファイルを指定できるかどうかは，指定するオプションによって異なります。リンクファイルを指定したときの動作については「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「リンクファイルの扱い」を参照してください。
- 表示操作，および検証操作では，通常ファイルとリンクファイルを指定できます。
- -r オプションを指定するときはディレクトリを指定できます。ディレクトリのシンボリックリンクを指定できるかどうかは，操作の種類や指定するオプションによって異なります。ディレクトリのシンボリックリンクを指定したときの動作については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「リンクファイルの扱い」を参照してください。

指定した*対象パス名*が，操作できるファイルの種類以外のときはエラーとなります。

その他の指定

-a

--ascii

【UNIX 限定】

OS が提供するgunzip コマンドとの互換のためのオプションです。

指定しても有効になりません。ただし，標準エラー出力にメッセージ（gunzip: Option -a is ignored on this system）が出力されます。

-f

--force

次の操作を許可します。

【伸長操作の場合】

- リンクファイルのリンク先をたどります。リンクファイルについては「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「リンクファイルの扱い」を参照してください。
- 作成する伸長後のファイルと同名のファイルがすでに存在する場合，応答要求を行わないで上書きします。
- 操作するファイルのデータを標準入力から入力するとき，標準入力が端末に関連づけられていても，入力を開始します。

-S *サフィックス*

--suffix=*サフィックス*

指定する*サフィックス*（任意の文字による文字列）を圧縮ファイルの拡張子として扱います。

【伸長操作の場合】

拡張子「.gz」，「.tgz」とともに，*サフィックス*が付いたファイルを圧縮ファイルと見なします。

指定できる*サフィックス*の長さの範囲は 1～30 バイトです。*サフィックス*にマルチバイト文字は使用できません。*サフィックス*を指定したときの動作については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「圧縮ファイルの拡張子の扱い」を参照してください。

-q

--quiet

メッセージの出力を抑止します。

ただし，次のメッセージは抑止の対象外です。

- オプション解析のエラーメッセージ

- 伸長操作の場合，伸長後のファイルと同名のファイルがすでに存在したとき，上書きするかどうかを確認するメッセージ

【表示操作の場合】

次の情報を表示しません。

- ヘッダ行
- 複数圧縮ファイルの合計情報

このオプションの後ろに-v オプションを指定した場合，このオプションの指定は取り消されます。

-v

--verbose

次の詳細情報を表示します。

- 伸長操作，および検証操作の場合，操作結果の詳細情報として「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の「表 8-12 圧縮操作，伸長操作，検証操作で出力する詳細情報」に示す内容を標準エラー出力に出力します。
- 表示操作の場合，表示する圧縮ファイル情報に，圧縮方式などの詳細情報も表示します。表示する内容については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の項目「圧縮ファイル情報の表示」を参照してください。
- -r オプションを指定してディレクトリ内のファイルを検索するとき，圧縮ファイル以外のファイルに対して，標準エラー出力に「gunzip: *ファイル名*: Unknown suffix: ignored」を出力します。

このオプションの後ろに-q オプションを指定した場合，このオプションの指定は取り消されます。

終了コード

終了コード	意味
0	正常終了
1	エラー終了 伸長操作の場合，伸長後のファイルは作成されていません。
2	エラー終了 伸長操作の場合，伸長後のファイルは作成されています。後処理でエラーが発生しました。エラーが発生した後処理の内容については，出力されるメッセージを参照してください。 なお，複数ファイルの操作で，終了コード1 と2 の両方のエラーが発生したときは，終了コードは1 が返ります。
3	エラー終了 不正なオプションを指定しました。

次のメッセージが出力される場合は，終了コード 1 になります。

- 「gunzip: *ファイル名*: Unknown suffix: ignored」

注意事項

gzip コマンドと共通の注意事項については，「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」を参照してください。gunzip コマンド固有の注意事項を次に示します。

- このコマンドは，JP1/Advanced Shell が提供するgzip コマンドで作成した圧縮ファイルの操作だけをサポートします。JP1/Advanced Shell が提供するgzip コマンド以外で作成した gzip 形式の圧縮ファイルをこのコマンドで操作した場合，正しく伸長，表示，または検証できないことがあります。
- Windows の場合，伸長する圧縮ファイルに読み取り専用属性が設定されていると，圧縮ファイルの削除で，ファイルごとに 1550 ミリ秒の待ち時間が発生します。ワイルドカード指定などによる複数ファイルの指定や，-r オプションを指定してディレクトリ内の複数のファイルを伸長するときは，事前に読み取り専用属性を解除してください。

使用例

gunzip コマンドでの伸長操作の使用例を示します。-l オプション，-t オプションおよびその他の伸長操作については「8.4.17　gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）」の使用例を参照してください。

- ファイルを伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gunzip file001.txt.gz

C:¥TEMP>%ADSH_OSCMD_DIR%ls
file001.txt
```

- -k オプションを使用して，圧縮ファイルを残したまま伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gunzip -k file001.txt.gz

C:¥TEMP>ls
file001.txt     file001.txt.gz
```

- -c オプションを使用して，圧縮ファイルを残したまま伸長します。また，伸長結果をfile003.txt ファイルに出力します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gunzip -c backup_file003.gz >file003.txt

C:¥TEMP>ls
backup_file003.gz file003.txt
```

- -N オプションを使用して，圧縮する前のファイル名で伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gunzip -N backup_file003.gz

C:¥TEMP>ls
file003.txt
```

- -S オプションを使用して，「.gz」以外の拡張子の圧縮ファイルを伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gunzip -S ".gzip" file004.txt.gzip
```

```
C:¥TEMP>ls
file004.txt
```

- 圧縮したアーカイブの伸長結果をtar コマンドに渡してアーカイブを展開します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gunzip -c DIR002.tar.gz | %ADSH_OSCMD_DIR%tar -xvf -
DIR002
DIR002¥DIR003
DIR002¥DIR003¥file004.txt
DIR002¥DIR003¥file005.txt
DIR002¥file001.txt
DIR002¥file002.txt
DIR002¥file003.txt
```

8.4.17 gzip コマンド（ファイルを圧縮，または圧縮されたファイルを伸長する）

形式

【Windows 限定】

```
gzip [-1|-2|-3|-4|-5|-6|-7|-8|-9] [-c] [-d] [-f] [-k] [-l] [-N] [-n] [-q]
     [-r] [-t] [-v] [-o 出力先パス名] [-S サフィックス] [対象パス名 ...]
```

【UNIX 限定】

```
gzip [-1|-2|-3|-4|-5|-6|-7|-8|-9] [-a] [-c] [-d] [-f] [-k] [-l] [-N] [-n] [-q]
     [-r] [-t] [-v] [-o 出力先パス名] [-S サフィックス] [対象パス名 ...]
```

機能

ファイルを圧縮，または圧縮されたファイルを伸長します。

このコマンドでは，次の操作ができます。

操作の種類	説明
圧縮	ファイルを圧縮します（以降，この操作を圧縮操作と呼びます）。 -d オプション，-l オプション，および-t オプションを指定しない場合，圧縮操作をします。 圧縮するときのファイル圧縮形式は gzip 形式です（以降，gzip 形式で圧縮したファイルを圧縮ファイルと呼びます）。 ファイルのデータを DEFLATE 圧縮方式※で圧縮します（以降，圧縮したファイルのデータを圧縮データと呼びます）。 圧縮データは，圧縮操作で作成する圧縮ファイルに格納されます。 圧縮操作では，圧縮するファイル名に拡張子「.gz」を付けた名称で圧縮ファイルを作成します。

操作の種類	説明
圧縮	圧縮操作では，圧縮ファイルを作成したあとに，圧縮するファイルを削除します。
伸長	-d オプションを指定することで，圧縮ファイルを伸長します（以降，この操作を伸長操作と呼びます）。 伸長できるファイル圧縮形式は gzip 形式だけです。また，ファイル名の終端に，拡張子「.gz」，または「.tgz」が付いたファイルを圧縮ファイルと見なします。gzip 形式で圧縮されていないファイルや，拡張子「.gz」，「.tgz」が付いていないファイルは，伸長できません。 伸長操作では，次のように生成した文字列をファイル名として，伸長したファイルを作成します。 • 圧縮ファイル名から拡張子「.gz」を取り除いた文字列 • 圧縮ファイル名の拡張子「.tgz」を「.tar」に変更した文字列 伸長操作では，伸長したファイルを作成したあとに，圧縮ファイルを削除します。
表示	-l オプションを指定することで，圧縮ファイルの情報を表示します（以降，この操作を表示操作と呼びます）。表示する圧縮ファイルの情報の詳細は，-l オプション，-v オプションの説明，および「圧縮ファイル情報の表示」を参照してください。
検証	-t オプションを指定することで，圧縮ファイルを伸長しないで，圧縮ファイルに格納されている圧縮データの整合性を検証します（以降，この操作を検証操作と呼びます）。

注※

gzip 形式の圧縮で用いられる圧縮方式

圧縮操作，および伸長操作で，圧縮するファイルや圧縮ファイルを削除したくない場合は，次のどちらかの方法で操作してください。

1. 圧縮結果，伸長結果を標準出力に出力する。

   次の操作では，ファイルを削除しないで，圧縮したファイルデータや，伸長したファイルデータを標準出力に出力します。必要に応じて，その出力をファイルにリダイレクトしたり，パイプで別のプログラムに渡したりしてください。

   - -c オプションを指定する。
   - 標準入力から圧縮するファイルや圧縮ファイルを入力する。

2. -k オプションを指定する。

   -k オプションを指定して実行すると次のように動作します。

   【圧縮操作の場合】

   圧縮ファイルを作成したあとも，圧縮するファイルを削除しません。

   【伸長操作の場合】

   伸長したファイルを作成したあとも，圧縮ファイルを削除しません。

圧縮操作，または伸長操作で，作成する圧縮ファイルや伸長後のファイルと同じ名前のファイルがすでに存在している場合は，上書きするかどうか問い合わせをします。標準入力からの応答文字がy またはY で始まっている場合，既存のファイルを削除して，圧縮ファイルや伸長後のファイルを作成します。応答文字がy およびY で始まっていない場合や，標準入力が使用できない場合は，操作対象ファイルの処理を中断します。問い合わせをしないで上書きを許可したい場合は，-f オプションを指定してください。

圧縮操作では，ディレクトリを圧縮できません。ディレクトリの構造を保持したまま圧縮したい場合は，tar コマンドでアーカイブを作成し，そのアーカイブを圧縮してください。また，tar コマンドの-z オプションを使用することで，アーカイブの作成と圧縮を同時にできます。tar コマンドの詳細については「8.4.34　tar コマンド（対象パス名をアーカイブに格納，およびアーカイブから抽出，表示する）」を参照してください。

検証操作で行う圧縮データの整合性検証は，伸長操作のときにも行われます。なお，圧縮ファイルの圧縮データ以外の値（圧縮方式などの情報）については，すべての操作で整合性を検証します。

環境変数GZIP にgzip コマンドのオプションを設定することで，コマンド引数に指定しなくてもオプションを使用できます。環境変数GZIP については，「2.5　環境変数を設定する」を参照してください。

引数

操作オプションの指定

実行する操作を指定します。

次のオプションを指定しないでこのコマンドを実行した場合は，ファイルを圧縮します（圧縮操作）。

-d

--decompress

--uncompress

圧縮ファイルを伸長します（伸長操作）。

-l オプションまたは-t オプションと同時に指定した場合は指定を無視します。

-l

--list

圧縮ファイルの情報を表示します（表示操作）。

圧縮する前のファイルサイズや圧縮率などの情報を表示します。

-v オプションを同時に指定すると，圧縮ファイルの詳細情報も表示します。

表示内容については，「圧縮ファイル情報の表示」を参照してください。

圧縮する前のファイルサイズの表示では，ファイルサイズが 4GB 以上の場合，値が正しく表示されません。正しいファイルサイズを表示したい場合は，-t オプションを同時に指定してください。ただし，-t オプションを同時に指定した場合は，表示されるまでに時間が掛かることがあります。

-t

--test

圧縮ファイルを伸長しないで，圧縮ファイルに格納されている圧縮データの整合性を検証します（検証操作）。

-l オプションと同時に指定した場合は，圧縮データの整合性検証と，圧縮ファイル情報の表示の両方を行います。

圧縮レベルの指定

-圧縮レベル

--fast

--best

圧縮するときの圧縮速度と圧縮率をレベルで指定します。

圧縮レベルは1 から9 の範囲で指定します。値の意味は次のとおりです。

- 圧縮速度

  1 が最も速く，9 が最も遅いです。

  数値が大きくなるに従って，圧縮速度は遅くなります。

- 圧縮率

  1 が最も低く，9 が最も高いです。

  数値が大きくなるに従って，圧縮率は高くなります。

このオプションを指定しない場合は，圧縮レベル6 で圧縮します。

--fast オプションは-1 と同じ圧縮レベルで圧縮します。

--best オプションは-9 と同じ圧縮レベルで圧縮します。

このオプションを複数指定した場合，最後に指定したオプションが有効になります。

-d オプション，-l オプション，または-t オプションと同時に指定した場合は指定を無視します。

ファイル情報の保存，復元動作の指定

ファイル情報の保存，復元動作を指定します。

-N オプションおよび-n オプションを同時に指定した場合，最後に指定したオプションが有効になります。-N オプションまたは-n オプションと，-t オプションを同時に指定した場合，指定を無視します。

なお，圧縮操作および伸長操作での最終修正日時の扱いについては，「ファイルの最終アクセス日時と最終修正日時の扱い」を参照してください。表示操作でのファイル名と最終修正日時の扱いについては，「圧縮ファイル情報の表示」を参照してください。

-N

--name

【圧縮操作の場合】

圧縮するファイルのファイル名と最終修正日時を圧縮ファイルに格納します。圧縮ファイルに格納するファイル名には，パス名から取り出したファイル名だけを格納します。なお，圧縮するファイルを標準入力から入力するときは，圧縮ファイル内の最終修正日時にコマンドの実行日時が格納されます。ただし，圧縮ファイルにファイル名は格納されません。

【伸長操作の場合】

伸長するときに作成するファイル名は，圧縮ファイルに格納されているファイル名とします。また，圧縮ファイルに格納されている最終修正日時を，作成したファイルに設定します。圧縮ファイルにファイル名や最終修正日時が格納されていない場合は，-n オプションを指定したときと同じ動作をします。

【表示操作の場合】

表示する圧縮ファイル情報に，圧縮ファイルに格納されているファイル名と最終修正日時を表示します。

圧縮操作ではこのオプションがデフォルトの動作です。

-n

--no-name

【圧縮操作の場合】

圧縮するファイルのファイル名と最終修正日時を圧縮ファイルに格納しません。

【伸長操作の場合】

伸長するときに作成するファイル名を，圧縮ファイル名から圧縮ファイルの拡張子を取り除いたファイル名とします。また，圧縮ファイルの最終修正日時を作成したファイルに設定します。

【表示操作の場合】

表示する圧縮ファイル情報に，圧縮ファイル名から圧縮ファイルの拡張子を取り除いたファイル名と，圧縮ファイルの最終修正日時を表示します。

伸長操作，表示操作では，このオプションがデフォルトの動作です。

入出力の指定

-c

--stdout

--to-stdout

【圧縮操作の場合】

圧縮したファイルデータを標準出力に出力します。

コマンド実行時に標準出力が端末に関連づけられている場合はエラーとなります。

このオプションを指定して複数のファイルを圧縮する場合，ファイル単位に圧縮と標準出力への出力を繰り返します。この動作は，cat コマンドなどで複数の圧縮ファイルを連結する（1 つの圧縮ファイルを作成する）場合と同じです。この動作で作られた圧縮ファイルの注意点については「注意事項」を参照してください。

【伸長操作の場合】

圧縮データの伸長結果を標準出力に出力します。

-l オプション，または-t オプションと同時に指定した場合は指定を無視します。

-k

--keep

【圧縮操作の場合】

圧縮ファイルを作成したあと，圧縮元ファイルを削除しません。

【伸長操作の場合】

伸長ファイルを作成したあと，圧縮ファイルを削除しません。

-l オプション，-t オプション，または-c オプションと同時に指定した場合は指定を無視します。

-r

--recursive

*対象パス名*に指定したパスがディレクトリの場合，ディレクトリおよびサブディレクトリを再帰的に検索します。

【圧縮操作の場合】

ディレクトリおよびサブディレクトリ内の圧縮されていないファイルを検索します。

【伸長操作，表示操作，検証操作の場合】

ディレクトリおよびサブディレクトリ内の圧縮ファイルを検索します。

Windows の場合，-r オプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。

検索中に遭遇したファイルを圧縮ファイルとして扱うかどうかは，ファイル名の拡張子で決定します。圧縮ファイルとして扱うファイル名の拡張子については，「圧縮ファイルの拡張子の扱い」を参照してください。

ディレクトリのシンボリックリンクは，操作の種類や指定するオプションでリンク先をたどります。ディレクトリのシンボリックリンクを指定したときの動作については，「リンクファイルの扱い」を参照してください。

-o *出力先パス名*

--output=*出力先パス名*

【圧縮操作の場合】

指定したパス名で圧縮ファイルを作成します。

また，圧縮操作では圧縮するファイルが削除されます。

【伸長操作の場合】

指定したパス名で伸長するファイルを作成します。また，伸長操作では，圧縮ファイルが削除されます。伸長操作で-N オプションを同時に指定した場合，圧縮ファイルに格納されている最終修正日時を使用します。ファイル名は*出力先パス名*に従います。

このオプションは，次のどれかと同時に指定するとエラー終了します。

- -c オプション，-t オプション，-l オプションまたは-r オプションを指定する。
- 圧縮するファイルや圧縮ファイルを複数指定する。
- 圧縮するファイルや圧縮ファイルを標準入力から入力する。

対象パス名

【圧縮操作の場合】

圧縮するファイルのパス名を指定します。

【伸長操作，表示操作，検証操作の場合】

圧縮ファイルのパス名を指定します。

圧縮操作で圧縮ファイルを指定したときや，伸長操作，表示操作，および検証操作で圧縮ファイル以外のファイルを指定したときの扱いについては，「圧縮ファイルの拡張子の扱い」を参照してください。

*対象パス名*は複数指定できます。

*対象パス名*に指定がない，または*対象パス名*として「-」を指定した場合は，圧縮するファイルデータまたは圧縮データを標準入力から入力します。圧縮操作，伸長操作では，標準入力から圧縮するファイルデータまたは圧縮データを入力する場合，圧縮したファイルデータや圧縮データの伸長結果を標準出力に出力します。なお，圧縮操作では，標準出力が端末に関連づけられているとエラーになります。伸長操作では，標準入力が端末に関連づけられているとエラーになります。

*対象パス名*に指定できるファイルの種類は次のとおりです。

- 圧縮操作および伸長操作では，通常ファイルだけを指定できます。また，リンクファイルを指定できるかどうかは，指定するオプションによって異なります。リンクファイルを指定したときの動作については，「リンクファイルの扱い」を参照してください。
- 表示操作および検証操作では，通常ファイルとリンクファイルを指定できます。
- -r オプションを指定するときはディレクトリを指定できます。ディレクトリのシンボリックリンクを指定できるかどうかは，操作の種類や指定するオプションによって異なります。ディレクトリのシンボリックリンクを指定したときの動作については，「リンクファイルの扱い」を参照してください。

指定した*対象パス名*が，操作できるファイルの種類以外のときはエラーとなります。

その他の指定

-a

--ascii

【UNIX 限定】

OS が提供するgzip コマンドとの互換のためのオプションです。

指定しても有効になりません。ただし，標準エラー出力にメッセージ（gzip: Option -a is ignored on this system）が出力されます。

-f

--force

次の操作を許可します。

【圧縮操作の場合】

- リンクファイルのリンク先をたどります。リンクファイルについては「リンクファイルの扱い」を参照してください。
- 作成する圧縮ファイルと同名のファイルがすでに存在する場合，応答要求を行わないで上書きします。
- -c オプションを指定しているときや，圧縮するファイルデータを標準入力から入力するときに，標準出力が端末に関連づけられていても，圧縮したファイルデータを出力します。

【伸長操作の場合】

- リンクファイルのリンク先をたどります。リンクファイルについては「リンクファイルの扱い」を参照してください。
- 作成する伸長後のファイルと同名のファイルがすでに存在する場合，応答要求を行わないで上書きします。
- 操作するファイルのデータを標準入力から入力するとき，標準入力が端末に関連づけられていても，入力を開始します。

-S *サフィックス*

--suffix=*サフィックス*

指定するサフィックス（任意の文字による文字列）を圧縮ファイルの拡張子として扱います。

【圧縮操作の場合】

拡張子「.gz」の代わりに，指定したサフィックスを付けて圧縮ファイルを作成します。

【伸長操作の場合】

拡張子「.gz」，「.tgz」とともに，サフィックスが付いたファイルを圧縮ファイルと見なします。

指定できるサフィックスの長さの範囲は 1～30 バイトです。サフィックスにマルチバイト文字は使用できません。サフィックスを指定したときの動作については，「圧縮ファイルの拡張子の扱い」を参照してください。

-q

--quiet

メッセージの出力を抑止します。

ただし，次のメッセージは抑止の対象外です。

- オプション解析のエラーメッセージ
- 圧縮操作の場合，圧縮ファイルと同名のファイルがすでに存在したとき，上書きするかどうかを確認するメッセージ
- 伸長操作の場合，伸長後のファイルと同名のファイルがすでに存在したとき，上書きするかどうかを確認するメッセージ

【表示操作の場合】

次の情報を表示しません。

- ヘッダ行
- 複数圧縮ファイルの合計情報

このオプションの後ろに-v オプションを指定した場合，このオプションの指定は取り消されます。

-v

--verbose

次の詳細情報を表示します。

- 圧縮操作，伸長操作，および検証操作の場合，操作結果の詳細情報として，「表 8-12 圧縮操作，伸長操作，検証操作で出力する詳細情報」に示す内容を標準エラー出力に出力します。
- 表示操作の場合，表示する圧縮ファイル情報に，圧縮方式などの詳細情報も表示します。表示する内容については，「圧縮ファイル情報の表示」を参照してください。
- -r オプションを指定してディレクトリ内のファイルを検索するとき，操作対象外のファイルに対して，次のメッセージを標準エラー出力に出力します。

  【圧縮操作の場合】

  圧縮ファイルに対して，「gzip: *ファイル名* already has *サフィックス* suffix -- unchanged」を出力します。

  【伸長操作，表示操作，検証操作の場合】

  圧縮ファイル以外のファイルに対して，「gzip: *ファイル名*: Unknown suffix: ignored」を出力します。

このオプションの後ろに-q オプションを指定した場合，このオプションの指定は取り消されます。

表 8-12　圧縮操作，伸長操作，検証操作で出力する詳細情報

操作の種類	詳細情報の内容※と説明（上段：内容，下段：説明）
圧縮	[*圧縮するファイル名*:]*圧縮率*%[-- replaced with *圧縮ファイル名*]
	圧縮するファイルの名称，圧縮率，および作成した圧縮ファイルの名称を出力します。

操作の種類	詳細情報の内容※と説明（上段：内容，下段：説明）
圧縮	-c オプションを指定したときは，圧縮ファイル名は出力されません。 標準入力から入力したときは，圧縮するファイル名と圧縮ファイル名は出力されません。
伸長	*圧縮ファイル名*:*圧縮率*%[-- replaced with *伸長後のファイル名*]
	圧縮ファイルの名称，圧縮率，および作成した伸長後のファイルの名称を出力します。 -c オプションを指定したときは，伸長後のファイル名は出力されません。 標準入力から入力したときは，詳細情報は出力されません。
検証	[*圧縮ファイル名*:] OK
	圧縮ファイルの名称と圧縮ファイルデータの整合性に問題がないことを示す「OK」を出力します。 標準入力から入力したときは，圧縮ファイル名は出力されません。

注※

- 「*圧縮するファイル名*: 」は「*圧縮するファイル名*:<タブ文字>」で出力されます。
- 「*圧縮ファイル名*: 」は「*圧縮ファイル名*:<タブ文字>」で出力されます。

終了コード

終了コード	意味
0	正常終了。
1	エラー終了。 圧縮操作，伸長操作の結果は次のとおりです。 【圧縮操作の場合】 圧縮ファイルは作成されていません。 【伸長操作の場合】 伸長後のファイルは作成されていません。
2	エラー終了。 圧縮操作，伸長操作の結果は次のとおりです。 【圧縮操作の場合】 圧縮ファイルは作成されています。後処理でエラーが発生しました。エラーが発生した後処理の内容については，出力されるメッセージを参照してください。 【伸長操作の場合】 伸長後のファイルは作成されています。後処理でエラーが発生しました。エラーが発生した後処理の内容については，出力されるメッセージを参照してください。 なお，複数ファイルの操作で，終了コード1 と2 の両方のエラーが発生したときは，終了コードは1 が返ります。

終了コード	意味
3	不正なオプションを指定しました。

次のメッセージが出力される場合は，終了コード 1 になります。

- 「gzip: *ファイル名* already has *サフィックス* suffix -- unchanged」
- 「gzip: *ファイル名*: Unknown suffix: ignored」

ファイルの最終アクセス日時と最終修正日時の扱い

圧縮操作，および伸長操作では，ファイルの最終アクセス日時と最終修正日時を次のように扱います。

- 圧縮操作

  圧縮操作では，通常，圧縮するファイルの最終アクセス日時と最終修正日時を圧縮ファイルに引き継ぎます。また，圧縮するファイルの最終修正日時を圧縮ファイルに格納します。

  最終アクセス日時と最終修正日時の扱いは，圧縮するファイルの入力元やオプションの指定によって，次の表に示す動作となります。

表 8-13　圧縮操作時の最終アクセス日時と最終修正日時の扱い

圧縮するファイルの入力元	圧縮結果の出力先	オプション※1	圧縮結果	
			圧縮ファイルの最終アクセス日時・最終修正日時	圧縮ファイルに格納される最終修正日時※2
ファイル	圧縮ファイル	-N	圧縮するファイルの最終アクセス日時・最終修正日時	圧縮するファイルの最終修正日時
		-n	圧縮するファイルの最終アクセス日時・最終修正日時	圧縮ファイルには最終修正日時が格納されません
	標準出力 (-c オプション指定)	-N	引き継ぎません※3	圧縮するファイルの最終修正日時※2
		-n	引き継ぎません※3	圧縮ファイルには最終修正日時が格納されません※2
標準入力	標準出力	-N	引き継ぎません※3	コマンドの実行日時※2
		-n	引き継ぎません※3	圧縮ファイルには最終修正日時が格納されません※2

注※1

-N オプション，-n オプションのどちらも指定しなかった場合，-N オプションが指定されたものとします。

注※2

圧縮結果の出力先が標準出力の場合，標準出力に出力した圧縮ファイルデータ内の最終修正日時を指します。

注※3

最終アクセス日時と最終修正日時の扱いは，圧縮結果の出力先の指定(リダイレクトやパイプなど)に従います。

なお，ファイルの最終アクセス日時と最終修正日時の設定に失敗した場合はエラーとなりますが，圧縮ファイルは作成され，圧縮するファイルは削除されます。

- 伸長操作

伸長操作では，通常，圧縮ファイルの最終アクセス日時と最終修正日時を，伸長したファイルに引き継ぎます。

最終修正日時の扱いは，オプションの指定や圧縮ファイルに最終修正日時が格納されているかどうかによって，次の表に示す動作となります。

表 8-14　伸長操作時の最終アクセス日時と最終修正日時の扱い

オプション※	圧縮ファイル内の最終修正日時の格納	伸長結果	
		伸長したファイルに設定する最終アクセス日時	伸長したファイルに設定する最終修正日時
-N	あり	圧縮ファイルの最終アクセス日時	圧縮ファイルに格納されている最終修正日時
	なし	圧縮ファイルの最終アクセス日時	圧縮ファイルの最終修正日時
-n	あり	圧縮ファイルの最終アクセス日時	圧縮ファイルの最終修正日時
	なし	圧縮ファイルの最終アクセス日時	圧縮ファイルの最終修正日時

注※

-N オプション，-n オプションのどちらも指定しなかった場合，-n オプションが指定されたものとします。

次のような伸長操作の場合，圧縮ファイルに格納されている最終修正日時は使用しません。

- 標準入力から圧縮ファイルを入力し，伸長結果を標準出力に出力する場合。
- -c オプションを指定して，伸長結果を標準出力に出力する場合。

この伸長操作での最終アクセス日時と最終修正日時の扱いは，伸長結果の出力先の指定（リダイレクトやパイプなど）に従います。

なお，ファイルの最終アクセス日時と最終修正日時の設定に失敗した場合はエラーとなりますが，伸長したファイルは作成され，圧縮ファイルは削除されます。

圧縮・伸長時のユーザー ID・グループ ID とパーミッションの扱い

UNIX の場合，ファイルのユーザー ID・グループ ID とパーミッションを次のように扱います。

【圧縮操作の場合】

圧縮するファイルのユーザー ID・グループ ID とパーミッションを圧縮ファイルに引き継ぎます。

【伸長操作の場合】

圧縮ファイルのユーザー ID・グループ ID とパーミッションを伸長したファイルに引き継ぎます。

また，ユーザー ID・グループ ID とパーミッションは次のように扱います。

- スティッキービット，setuid ビットおよび setgid ビットは引き継ぎません。
- ユーザー ID・グループ ID の引き継ぎに失敗してもエラーにはなりません。引き継ぎに失敗したときは，コマンド実行者のユーザー ID，グループ ID が設定されます。
- パーミッションの引き継ぎに失敗した場合はエラーとなります。しかし，圧縮ファイルや伸長したファイルは作成されます。また，圧縮するファイルや圧縮ファイルも削除されます。
- 標準入力から入力する場合や，-c オプションを指定して圧縮結果，伸長結果を標準出力に出力する場合，引き継がれません。ユーザー ID・グループ ID とパーミッションの扱いは，標準出力の出力先の指定(リダイレクトやパイプなど)に従います。

Windows の場合，ファイルの所有者，ACL およびファイル属性は引き継ぎません。

圧縮ファイルの拡張子の扱い

圧縮ファイルの拡張子の扱いについて説明します。なお，標準入力から圧縮するファイルや圧縮ファイルを入力する場合は，拡張子を意識する必要はありません。

1. 圧縮操作での圧縮済みファイルの扱い

   圧縮操作では，圧縮するファイルの名称に，拡張子「.gz」,「.tgz」または-S オプションに指定したサフィックスが付いている場合，圧縮済みファイルと見なして，圧縮しません。また，圧縮済みファイルに対して，メッセージ「gzip: *ファイル名* already has *サフィックス* suffix -- unchanged」を出力します。

   -r オプションを指定し，ディレクトリ内を検索して見つかった圧縮済みファイルに対しては，圧縮しません。また，メッセージ「gzip: *ファイル名* already has *サフィックス* suffix -- unchanged」も出力しません。なお，-v オプションを指定しているときはメッセージを出力します。

   -c オプションを指定したときは，圧縮済みファイルも圧縮します。

2. 伸長，表示，検証対象となる圧縮ファイルの拡張子

   伸長操作，検証操作，表示操作では，圧縮ファイルの拡張子の扱いは次のとおりです。

   - 引数に圧縮ファイル名を指定するとき

     【伸長操作の場合】

     圧縮ファイル名に，拡張子「.gz」,「.tgz」，または-S オプションに指定したサフィックスが付いているとき，圧縮ファイルと見なして，伸長します。それ以外のファイルの場合は，伸長しません。

また，指定されたファイルに対してメッセージ「gzip: *ファイル名*: Unknown suffix: ignored」を出力します。

【表示操作，検証操作の場合】

圧縮ファイル名に，拡張子「.gz」,「.tgz」および-S オプションに指定したサフィックスが付いていなくても操作します。

- -r オプション指定によりディレクトリ内を検索するとき

  ディレクトリ内に存在する，拡張子「.gz」,「.tgz」および-S オプションに指定したサフィックスが付いているファイルに対して操作します。拡張子「.gz」,「.tgz」または-S オプションに指定したサフィックスが付いていないファイルに対しては，操作しません。また，メッセージ「gzip: *ファイル名*: Unknown suffix: ignored」も出力しません。なお，-v オプションを指定しているときは，メッセージを出力します。

  なお，-c オプションを指定したときは，伸長操作ではファイル名に拡張子「.gz」,「.tgz」および-S オプションに指定したサフィックスが付いていなくても伸長を試みます。

3. 拡張子の文字種別の扱い

   圧縮ファイルの拡張子の確認では，拡張子「.gz」,「.tgz」および-S オプションに指定したサフィックスは，英大文字と英小文字を区別しません。例えば，圧縮するファイルの名称が，「file.GZ」でも圧縮ファイルと見なします。

4. 圧縮ファイル名の拡張子の補完

   伸長操作，表示操作，および検証操作では，引数に圧縮ファイル名を指定するとき，次の指定ができます。

   - 圧縮ファイルの拡張子が「.gz」の場合，「.gz」の記述を省略できます。
   - 圧縮ファイル名の終端が-S オプションに指定したサフィックスの場合，サフィックスの記述を省略できます。

   拡張子「.gz」や-S オプションに指定したサフィックスは，コマンドで補完します。

   ただし，拡張子が「.tgz」の圧縮ファイルは，「.tgz」の記述を省略できません。

   引数に指定した圧縮ファイル名に拡張子「.gz」が補完される例を次に示します。

```
$ ls
file1.txt.gz
$ gzip -d file1.txt
$ ls
file1.txt
```

なお，操作するディレクトリ内に，複数のファイルが格納されているときは，次のように動作します。

- 2 つの圧縮ファイルの名称の違いが，拡張子「.gz」と-S オプションに指定したサフィックスだけのときは，-S オプションのサフィックスが付いた圧縮ファイルが操作対象です。
- 拡張子「.gz」や-S オプションに指定したサフィックスが付いているファイルと付いていないファイルが存在するときは，付いていないファイルが操作対象です。

拡張子の補完では，拡張子「.gz」および-S オプションに指定したサフィックスの英大文字，英小文字の扱いは，OS によって異なります。

【UNIX の場合】

英大文字，英小文字の相違を区別します。

- 拡張子「.gz」

  英小文字の「.gz」が付いた圧縮ファイルが補完対象です。

  「.GZ」,「.gZ」および「.Gz」が付いた圧縮ファイルは補完の対象になりません。

- -S オプションに指定したサフィックス

  -S オプションに指定したサフィックスが付いた圧縮ファイルが補完対象です。

  指定したサフィックスと，英大文字および英小文字の構成だけが異なるサフィックスが付いた圧縮ファイルは，補完の対象になりません。

【Windows の場合】

英大文字，英小文字の相違を区別しません。

- 拡張子「.gz」

  英小文字の「.gz」,「.GZ」,「.gZ」および「.Gz」が付いた圧縮ファイルが補完対象です。

- -S オプションに指定したサフィックス

  -S オプションに指定したサフィックスが付いた圧縮ファイルが補完対象です。

  指定したサフィックスと，英大文字および英小文字の構成だけが異なるサフィックスが付いた圧縮ファイルも補完対象です。

圧縮ファイル情報の表示

圧縮ファイル情報の表示では，最初にヘッダ行として項目名を表示して，次の行に圧縮ファイル情報を表示します。ヘッダ行の表示は，-q オプションを使用して抑止できます。

1. 表示する項目

   次の項目（列）を表示します。項目「method」,「crc」,「date」,「time」は，-v オプションを指定したときに表示します。

表 8-15　圧縮ファイル情報の表示項目（列）

項目名	内容
method	圧縮方式。次の値を表示します。 defla：圧縮データは，DEFLATE 圧縮方式で圧縮されています。
crc	圧縮前のファイルデータの巡回冗長検査（CRC）値。8 桁の 16 進数で表示します この値を使用して，圧縮されたファイルデータの整合性を検証します。

項目名	内容
crc	サイズが0 のファイルを圧縮しているときは，00000000 を表示します。 標準入力から圧縮ファイルを入力したときは，ffffffff を表示します※1。
date	伸長後のファイルに設定されるファイルの最終修正日時の月日。年は表示されません。 表示する内容は，圧縮ファイルの入力元や指定するオプションによって異なります。 詳細は「表 8-16 「date」,「time」項目の表示内容」を参照してください。
time	伸長後のファイルに設定されるファイルの最終修正日時の時分。秒は表示されません。 表示する内容は，圧縮ファイルの入力元や指定するオプションによって異なります。 詳細は「表 8-16 「date」,「time」項目の表示内容」を参照してください。
compressed	圧縮ファイルのサイズ。 標準入力から圧縮ファイルを入力したときは，-1 を表示します※1。
uncompressed	圧縮する前のファイルのサイズ。 標準入力から圧縮ファイルを入力したときは，-1 を表示します※1。 圧縮する前のファイルのサイズが 4GB 以上の場合，値が正しく表示されません※2。
ratio	圧縮率（%）。小数点第 1 位までを表示します。 圧縮率は，圧縮する前のファイルのサイズと，圧縮ファイル内の圧縮データの長さを元に算出します。なお，圧縮データの長さは，圧縮ファイルのサイズとは異なります。 「compressed」項目や「uncompressed」項目に-1 が表示されるときは，「0.0%」を表示します。 サイズが小さいファイルや，圧縮効果が小さいファイルを圧縮したときは，圧縮率はマイナスになる場合があります。
uncompressed_name	伸長後のファイル名。圧縮ファイル名をディレクトリからのパス名で指定しているときは，パス名のディレクトリ部分と伸長後のファイル名によって生成したパス名を表示します。 標準入力から圧縮ファイルを入力したときは，「stdout」を表示します。 表示する内容は，圧縮ファイルの入力元や指定するオプションによって異なります。 詳細は「表 8-17 「uncompressed_name」項目の表示内容」を参照してください。

注※1

-t オプションを指定すると，標準入力から入力する場合も，圧縮ファイルから入力したときと同じ内容を表示できます。ただし，圧縮ファイルのサイズによっては，表示されるまでに時間が掛かる場合があります。

注※2

-t オプションを使用して，4GB 以上のサイズを表示できます。ただし，表示されるまでに時間が掛かる場合があります。

表 8-16 「date」，「time」項目の表示内容

圧縮ファイルの入力元	圧縮ファイル内の最終修正日時の格納※1	指定するオプション※2	表示する最終修正日時の月日と時分
ファイル	あり	-N	圧縮ファイル内に格納されている最終修正日時（圧縮する前のファイルの最終修正日時）
		-n	圧縮ファイル自身の最終修正日時
	なし	-N	圧縮ファイル自身の最終修正日時
		-n	圧縮ファイル自身の最終修正日時
標準入力	あり	-N	圧縮ファイル内に格納されている最終修正日時（圧縮する前のファイルの最終修正日時）
		-n	コマンドの実行日時
	なし	-N	コマンドの実行日時
		-n	コマンドの実行日時

注※1

圧縮ファイル内に最終修正日時が格納されるかどうかは，圧縮操作での圧縮するファイルの入力方法やオプションの指定によって決まります。圧縮操作での最終修正日時の扱いについては，「ファイルの最終アクセス日時と最終修正日時の扱い」を参照してください。

注※2

-N オプション，-n オプションのどちらも指定しなかった場合，-n オプションが指定されたものとします。

表 8-17 「uncompressed_name」項目の表示内容

圧縮ファイルの入力元	圧縮ファイル内のファイル名の格納※1	指定するオプション※2	表示する伸長後のファイル名
ファイル	あり	-N	圧縮ファイルに格納されているファイル名（圧縮する前のファイルの名称）

圧縮ファイルの入力元	圧縮ファイル内のファイル名の格納※1	指定するオプション※2	表示する伸長後のファイル名
ファイル	あり	-n	圧縮ファイル名から生成（圧縮ファイルの拡張子を取り除くなど）したファイル名
	なし	-N	圧縮ファイル名から生成（圧縮ファイルの拡張子を取り除くなど）したファイル名
		-n	圧縮ファイル名から生成（圧縮ファイルの拡張子を取り除くなど）したファイル名
標準入力	あり	-N	圧縮ファイルに格納されているファイル名（圧縮する前のファイルの名称）
		-n	「stdout」
	なし	-N	「stdout」
		-n	「stdout」

注※1

圧縮ファイル内にファイル名が格納されるかどうかは，圧縮操作での圧縮するファイルの入力方法や，-N オプション，-n オプションの指定によって決まります。圧縮ファイル内に格納されるファイル名については，引数-N オプション，-n オプションの説明を参照してください。

注※2

-N オプション，-n オプションのどちらも指定しなかった場合，-n オプションが指定されたものとします。

2. 複数圧縮ファイルの合計情報の表示

引数に複数の圧縮ファイルを指定するときや，-r オプションでディレクトリ内のすべての圧縮ファイルを対象とするときは，ファイル単位に圧縮ファイル情報を表示したあと，最終行に複数圧縮ファイルの合計情報を表示します。表示内容を「表 8-18 複数圧縮ファイルの合計情報の表示内容」に示します。

この表示は，-q オプションを使用して抑止できます。また，表示する複数ファイルに，標準入力からの入力が含まれる（「compressed」項目や「uncompressed」項目に-1 が表示される）場合，表示されません。

表 8-18　複数圧縮ファイルの合計情報の表示内容

項目名	内容
compressed	圧縮ファイルのサイズの合計値。
uncompressed	圧縮する前のファイルのサイズの合計値。
ratio	圧縮ファイルのサイズの合計値と，圧縮ファイルデータの長さの合計値を元に計算した圧縮率。

項目名	内容
uncompressed_name	「(totals)」。

リンクファイルの扱い

このコマンドでは，リンクファイル（シンボリックリンクおよびハードリンク）を次のように扱います。

【圧縮操作，伸長操作の場合】

圧縮操作，および伸長操作では，圧縮するファイルや圧縮ファイルのリンクファイルは操作できません。ただし，-f オプションを指定すると，リンクファイルを扱えます。-f オプションを指定したときのリンクファイル操作時の動作は次のとおりです。

操作の種類	リンクファイル操作時の動作
圧縮	• リンク先のファイルのデータを圧縮します。 • ファイル名から圧縮ファイル名を生成するときは，リンクファイル名を使用します。 • 圧縮ファイルに格納するファイル名には，リンクファイル名を格納します。 • 圧縮後には，リンクファイルを削除します。
伸長	• リンク先のファイルの圧縮データを伸長します。 • ファイル名から伸長するファイル名を生成するときは，リンクファイル名を使用します。 • 伸長後には，リンクファイルを削除します。

なお，圧縮後や伸長後にリンクファイルを削除したくない場合は-k オプションを指定してください。

-c オプションを指定する場合は，リンクファイルを扱えます。

-r オプションでディレクトリ内を検索するとき，ディレクトリへのシンボリックリンクを扱えます。ただし，-r オプションと同時に，-f オプションを指定してください。-f オプションを指定すると，ディレクトリへのシンボリックリンクのリンク先をたどります。また，ディレクトリを再帰的に検索したときも，遭遇したディレクトリのシンボリックリンクのリンク先をたどります。なお，-c オプションを指定したときも，ディレクトリのシンボリックリンクのリンク先をたどります。

【表示操作，検証操作の場合】

表示操作，および検証操作では，リンクファイルを扱えます。圧縮ファイルにリンクファイルを指定した場合，リンク先をたどり，リンク先の圧縮ファイルを参照して，表示や検証をします。ただし，表示操作で表示する伸長後のファイル名には，リンク先の圧縮ファイル名ではなく，リンクファイル名から生成した名称を表示します。

-r オプションでディレクトリ内を検索するとき，ディレクトリへのシンボリックリンクはリンク先をたどります。また，ディレクトリを再帰的に検索したときも，遭遇したディレクトリのシンボリックリンクのリンク先をたどります。

注意事項

- このコマンドは，このコマンド自身が作成した圧縮ファイルの操作だけをサポートします。次の操作は正常に行われない場合があります。
  - 他プログラム（OS 提供の`gzip`コマンドなど）で作成した gzip 形式の圧縮ファイルを，JP1/Advanced Shell 提供の`gzip`コマンドで操作する。
  - JP1/Advanced Shell 提供の`gzip`コマンドで作成した圧縮ファイルを，gzip 形式の圧縮ファイルを扱えるプログラム（OS 提供の`gzip`コマンドなど）で操作する。
- サイズが小さいファイルや，圧縮効果が小さいファイルを圧縮した場合，圧縮ファイルのファイルサイズが圧縮前のファイルサイズより大きくなることがあります。
- `-N`オプションを指定して伸長操作または表示操作をする場合に，圧縮ファイルに格納されているファイル名が次のときは，OS やエンコーディングが圧縮時と異なると，伸長や表示が正常に行われないおそれがあります。
  - マルチバイト文字などのエンコーディングに依存する文字を含んでいる。
  - ファイル名に OS 固有の指定可能文字を含んでいる（例：UNIX で¥をファイル名の文字として使用する）。
- 圧縮するファイルのパス名やファイル名の長さが次の値を超えるときは，圧縮ファイルの作成に失敗します。この場合，`-o`オプションや`-c`オプションを使用して圧縮ファイルを作成してください。
  - 「*OS が定める最大パス名長 － 圧縮ファイルの拡張子の長さ*」
  - 「*OS が定める最大ファイル名長 － 圧縮ファイルの拡張子の長さ*」
- 次のどれかに該当する場合，`-N`オプションを指定して伸長操作をすると，伸長後のファイルの作成に失敗します。
  - 圧縮ファイルに格納されているファイル名の長さが，OS が定める最大ファイル名長を超えている。
  - 圧縮ファイルに格納されているファイル名を元に生成したパス名の長さが，OS が定める最大パス名長を超えている。
- 圧縮ファイルへの最終修正日時に格納できる日時の範囲は，UTC（協定世界時）の 1970 年 1 月 1 日 0 時 0 分 1 秒～2038 年 1 月 19 日 3 時 14 分 7 秒です。範囲外のときは格納されません。
- 圧縮操作中にコマンドの実行を中断すると，圧縮途中の圧縮ファイルが残ることがあります。
- 伸長操作中にコマンドの実行を中断すると，伸長途中の伸長ファイルが残ることがあります。
- 圧縮するファイルに対して削除権限がない場合，圧縮操作で行う圧縮するファイルの削除が失敗します。圧縮するファイルの削除に失敗しても，圧縮ファイルは作成されています。
- 圧縮ファイルに対して削除権限がない場合，伸長操作で行う圧縮ファイルの削除が失敗します。圧縮ファイルの削除に失敗しても，伸長後のファイルは作成されています。
- すでに存在するファイルと同名の圧縮ファイル，伸長後のファイルを上書きで作成する場合，最初に既存のファイルを削除します。そのため，処理中にエラーが発生すると，既存のファイルは削除されている場合があります。

- 操作対象のファイルと出力先のファイルの実体が同じ（シンボリックリンク先が同じ，またはハードリンク先が同じ）場合，エラーになります。
- -c オプションを使用して複数のファイルから作成した圧縮ファイルや，cat コマンドなどで複数の圧縮ファイルを連結して作成した圧縮ファイルは，次のように伸長操作，または表示操作が行われます。
  - 複数のファイルのデータが出力された伸長後のファイルが 1 つ作成されます。圧縮前のそれぞれのファイルごとに，伸長後のファイルが作成されることはありません。
  - -N オプションを指定して伸長する場合，圧縮ファイル作成時に最初に指定したファイルの圧縮ファイル情報中のファイル名で，伸長後のファイルが作成されます。
  - 表示操作では，圧縮ファイル作成時に最後に指定したファイルのサイズが，圧縮する前のファイルのサイズとして表示されます。また，圧縮率はその値を元に算出されます。なお，-t オプションを指定した場合は，すべてのファイルサイズの合計値が表示されます。
- Windows の場合，圧縮するファイル名や圧縮ファイル名にショートネームを指定したとき，ショートネームをファイル名とした圧縮ファイルや伸長後のファイルが作成されます。
- Windows の場合，ファイル名の英大文字と英小文字を区別しません。このため，次の動作となります。
  - 圧縮ファイル名や伸長後のファイル名と同じスペルで英大文字，英小文字だけが異なるファイルは，同一ファイルと見なします。例えば，作成する圧縮ファイルや伸長後のファイルと同じスペルで，大文字・小文字だけが異なる名称のファイルは，同名のファイルがすでに存在していると見なします。
  - 圧縮するファイル名や圧縮ファイル名を，同じスペルで大文字・小文字だけを替えて引数に指定した場合，引数に指定したファイル名で圧縮ファイルや伸長後のファイルが作成されます。
- Windows の場合，操作対象のファイルや上書きする既存のファイルが別のプログラムで使用されていると，そのファイルの削除に失敗することがあります。
- Windows の場合，圧縮するファイルや伸長する圧縮ファイルに読み取り専用属性が設定されていると，圧縮するファイルの削除や圧縮ファイルの削除で，ファイルごとに 1550 ミリ秒の待ち時間が発生します。ワイルドカード指定などによる複数ファイルの指定や，-r オプションを指定してディレクトリ内の複数のファイルを圧縮・伸長するときは，事前に読み取り専用属性を解除してください。

使用例

- ファイルを圧縮します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip file001.txt

C:¥TEMP>%ADSH_OSCMD_DIR%ls
file001.txt.gz
```

- -k オプションを使用して，ファイルを残したまま圧縮します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -k file001.txt

C:¥TEMP>%ADSH_OSCMD_DIR%ls
file001.txt     file001.txt.gz
```

- -o オプションを使用して，コマンドで生成する圧縮ファイル名とは別の名称で，圧縮ファイルを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -o save_file002.gz  file002.txt

C:¥TEMP>ls
save_file002.gz
```

- -c オプションを使用して，ファイルを残したまま圧縮します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -c file003.txt > backup_file003.gz

C:¥TEMP>ls
backup_file003.gz file003.txt
```

- -S オプションを使用して，「.gz」以外の拡張子の圧縮ファイルを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -S ".gzip" file004.txt

C:¥TEMP>ls
file004.txt.gzip
```

- -r オプションを使用して，ディレクトリ内のファイルを圧縮します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -r DIR002

C:¥TEMP>ls -R DIR002
DIR003          file001.txt.gz  file002.txt.gz  file003.txt.gz

DIR002¥DIR003:
file004.txt.gz  file005.txt.gz
```

- -v オプションを使用して，圧縮するときに圧縮率と圧縮ファイル名を出力します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -v file005.txt
file005.txt:    26.5% -- replaced with file005.txt.gz
```

なお，圧縮率と圧縮ファイル名は標準エラー出力に出力されます。

- 圧縮ファイルを伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d file001.txt.gz

C:¥TEMP>%ADSH_OSCMD_DIR%ls
file001.txt
```

- -o オプションを使用して，コマンドで生成する伸長後のファイル名とは別の名称で，伸長したファイルを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d -o FILE002.txt save_file002.gz

C:¥TEMP>ls
FILE002.txt
```

- -c オプションを使用して，圧縮ファイルを残したまま伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d -c backup_file003.gz >file003.txt

C:¥TEMP>ls
backup_file003.gz file003.txt
```

- -N オプションを使用して，圧縮する前のファイル名で伸長します。

```
C:¥DIR001>%ADSH_OSCMD_DIR%gzip -d -N backup_file003.gz

C:¥DIR001>ls
file003.txt
```

- -S オプションを使用して，「.gz」以外の拡張子の圧縮ファイルを伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d -S ".gzip" file004.txt.gzip

C:¥TEMP>ls
file004.txt
```

- -r オプションを使用して，ディレクトリ内の圧縮ファイルを伸長します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d -r DIR002

C:¥TEMP>ls -R DIR002
DIR003       file001.txt  file002.txt  file003.txt

DIR002¥DIR003:
file004.txt  file005.txt
```

- -v オプションを使用して，伸長するときに伸長率と伸長後のファイル名を出力します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d -v file005.txt.gz
file005.txt.gz:    26.5% -- replaced with file005.txt
```

なお，伸長率と伸長後のファイル名は標準エラー出力に出力されます。

- 圧縮ファイル情報を表示します。

```
C:¥TEMP>gzip -l file001.txt.gz
         compressed        uncompressed  ratio uncompressed_name
           25025357            34040107  26.5% file001.txt
```

- -v オプションを使用して，詳細な圧縮ファイル情報を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -l -v file001.txt.gz
method crc      date   time           compressed        uncompressed  ratio
uncompressed_name
defla  fe65cbfa Dec 23 16:52          25025357              34040107  26.5%  file001.txt
```

- 複数の圧縮ファイルの圧縮ファイル情報を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -l -v file001.txt.gz  file002.txt.gz  file003.txt.gz
file004.txt.gz  file005.txt.gz
method crc      date   time           compressed        uncompressed  ratio
uncompressed_name
defla  5d262330 Dec 23 17:12            16142040            17838735   9.5% file001.txt
defla  0e523a79 Dec 23 17:12            18824484            20542848   8.4% file002.txt
```

```
defla  1cb8a527 Dec 23 17:12            12476069            17673807  29.4% file003.txt
defla  b167ed5d Dec 23 17:12            19489411            22390086  13.0% file004.txt
defla  9b4d3435 Dec 23 17:12            19631128            22513931  12.8% file005.txt
                                        86563132           100959407  14.3% (totals)
```

- 圧縮ファイルの整合性を検証します。また，-v オプションを指定して検証結果を出力します。

```
C:¥TEMP>gzip -t -v file001.txt.gz
file001.txt.gz:  OK
```

なお，検証結果は標準エラー出力に出力されます。

- tar コマンドのアーカイブ結果を圧縮します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%tar -cvf - DIR002 | %ADSH_OSCMD_DIR%gzip > DIR002.tar.gz
DIR002
DIR002/DIR003
DIR002/DIR003/file004.txt
DIR002/DIR003/file005.txt
DIR002/file001.txt
DIR002/file002.txt
DIR002/file003.txt
```

- 圧縮したアーカイブの伸長結果をtar コマンドに渡してアーカイブを展開します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%gzip -d -c DIR002.tar.gz | %ADSH_OSCMD_DIR%tar -xvf -
DIR002
DIR002¥DIR003
DIR002¥DIR003¥file004.txt
DIR002¥DIR003¥file005.txt
DIR002¥file001.txt
DIR002¥file002.txt
DIR002¥file003.txt
```

8.4.18　head コマンド（ファイルの最初の部分を表示する）

形式

```
head [-行数|-n 行数] [パス名 ...]
```

機能

ファイルの最初の数行を表示します。ファイルの最初の部分からそれぞれ指定した行数を標準出力に出力します。ファイルの指定がない場合，標準入力から入力します。行数を省略した場合は，10 行を仮定します。

引数

-行数 |-n 行数 ～＜10 進数＞((1～2147483647))

標準出力へ出力する，入力ファイルの最初の行数をそれぞれ指定します。0 以下または 2147483647 より大きい値を指定すると，エラーメッセージ（head: line count too small: 指定値／head: line count too large: 指定値）を出力します。

パス名

入力ファイル名を指定します。

- 省略時は標準入力から入力します。
- 複数ファイルを指定できます。複数ファイルを指定した場合，それぞれのファイルを識別するために，次に示す文字をそれぞれのファイルの出力に先行して出力します。2 つ目以降のファイルの場合，改行のあとに次に示す文字を出力します。

  ==> **ファイル名** <==
- 複数ファイルを指定して実行した場合はすべてのファイルを処理し，オープンに失敗したファイルが 1 つでもあると終了コード 1 で終了します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

head コマンドを実行した結果表示に使用するファイルの形式を次に示します。

- test1.txt

```
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
```

- test2.txt

```
0001:test2.txt
0002:test2.txt
0003:test2.txt
0004:test2.txt
0005:test2.txt
0006:test2.txt
```

```
0007:test2.txt
0008:test2.txt
0009:test2.txt
0010:test2.txt
```

これらのファイルを基にコマンドの実行結果を示します。

- test1.txt および test2.txt ファイルの最初の 2 行を表示します。

```
$ head -2 test1.txt test2.txt
==> test1.txt <==
0001:test1.txt
0002:test1.txt

==> test2.txt <==
0001:test2.txt
0002:test2.txt
$
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥head -d
head: illegal option -- d
usage: head [-count | -n count] [file ...]
```

8.4.19　hostname コマンド（ホスト名を表示する）

形式

```
hostname
```

機能

現在のホスト名を表示します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- このコマンドには指定可能な引数が存在しません。引数が指定された場合でも無視して処理を実行します。

使用例

- 現在のホストシステムの名称を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥hostname
HOST01
```

8.4.20 ln コマンド（ファイル，またはディレクトリへのリンクファイルを作成する）

形式

形式 1

```
ln [-f] [-i] [-L] [-n] [-P] [-s] [-T] [-v] [--follow={yes | no}]
   [--no-exist-directory] [--no-exist-file] リンク先パス名 ... [ターゲット]
```

形式 2

```
ln [-f] [-i] [-L] [-n] [-P] [-s] [-v] [--follow={yes | no}]
   [--no-exist-directory] [--no-exist-file] -t ターゲットディレクトリ名 リンク先パス
名 ...
```

機能

ファイル，またはディレクトリへのリンクファイルを作成します。-s オプションの指定に従って，ハードリンク，またはシンボリックリンクを作成します。

ln コマンドでシンボリックリンクを作成する場合は，シンボリックリンク作成権限を持つユーザーで実行してください。シンボリックリンク作成権限を持たないユーザーではシンボリックリンクを作成できません。また，ユーザーアカウント制御（UAC）が有効な環境では，シンボリックリンク作成権限を持つユーザーでもシンボリックリンクを作成できません。UAC が有効な環境でシンボリックリンクを作成する場合は，シンボリックリンク作成権限を持つユーザーに管理者特権を持たせてください。

形式 1 の場合，引数ターゲットに指定したパスにリンクファイルを作成します。引数ターゲットがディレクトリの場合はディレクトリ内にリンクファイルを作成します。

形式 2 の場合，ターゲットディレクトリ内にリンクファイルを作成します。

引数

-f

--force

引数ターゲットに指定したファイルがすでに存在する場合，既存のファイルを削除してからリンクファイルを作成します。

-i オプションと同時に指定した場合は，-i オプションの指定が有効となります。

ただし，環境変数 ADSH_CMDLN_OPT_I_F に LAST を指定することで最後に指定したオプションを有効にできます。

-i

--interactive

引数ターゲットに指定したファイルがすでにある場合は，削除するかどうか問い合わせをします。標準入力からの応答文字が y または Y で始まっていれば，既存のファイルを削除しリンクファイルを作成します。それ以外の文字を応答したり，標準入力が使用できない場合は処理を中断します。

-f オプションと同時に指定した場合，-i オプションの指定が有効となります。

ただし，環境変数 ADSH_CMDLN_OPT_I_F に LAST を指定することで最後に指定したオプションを有効にできます。

-L

--logical

引数リンク先パス名にシンボリックリンクを指定した場合，シンボリックリンクをたどり，リンク先が示すファイルの実体に対するハードリンクを作成します。ただし，リンク先が示すファイルが存在しない場合は作成しません。

このオプションはハードリンク作成時に有効なオプションです。-s オプションと同時に指定した場合は指定を無視します。

-L オプション，-P オプションのどちらも指定しなかった場合，-L オプションが指定されたものとします。

-L オプションと-P オプションを同時に指定した場合，最後に指定したオプションが有効となります。

-n

--no-dereference

意味を持たないオプションです。指定を無視します。

--follow={yes | no}

このオプションは，引数ターゲットに，ディレクトリに対するシンボリックリンクを指定したときの動作を選択するオプションです。

yes を指定した場合，引数ターゲットに指定したディレクトリに対するシンボリックリンクをたどり，リンク先のディレクトリにリンクファイルを作成します。

no を指定した場合，引数ターゲットに指定したディレクトリに対するシンボリックリンクをたどらず，リンクファイル名として扱います。

--follow オプションの指定がない場合は，yes が指定されたものとします。

-P

--physical

- Windows，Linux，Solaris の場合

  引数リンク先パス名にシンボリックリンクを指定した場合，シンボリックリンクをたどらず，シンボリックリンク自身のハードリンクを作成します。

  このオプションはハードリンク作成時に有効なオプションです。-s オプションと同時に指定した場合は指定を無視します。

  -L オプション，-P オプションのどちらも指定をしなかった場合，-L オプションが指定されたものとします。

  -L オプションと-P オプションを同時に指定した場合，最後に指定したオプションが有効となります。

- AIX，HP-UX の場合

  指定を無視し，-L オプションを指定した場合と同じ動作となります。

-s

--symbolic

引数リンク先パス名に指定したファイル，またはディレクトリに対してシンボリックリンクを作成します。

-s オプションの指定がない場合はハードリンクを作成します。ただし，次のハードリンクは作成できません。

- 引数リンク先パス名に指定した対象が存在しない
- ディレクトリに対するハードリンク
- UNIX の場合：異なるファイルシステムのファイル
- Windows の場合：異なるドライブレターのファイル

-s オプションの指定がある場合はシンボリックリンクを作成します。

UNIX の場合

引数リンク先パス名に指定した対象の有無にかかわらず作成できます。

Windows

引数リンク先パス名に指定した対象が存在しない場合，作成できません。引数リンク先パス名に指定した対象が存在しない状態でシンボリックリンクを作成する場合は--no-exist-directory オプション，または--no-exist-file オプションを指定してください。

作成したハードリンクはリンク先のアクセス権限を引き継ぎます。

作成したシンボリックリンクのアクセス権限は次のとおりです。

- UNIX の場合：全ユーザーに対してフルコントロール
- Windows：作成するディレクトリのアクセス権限を継承した上で，Everyone にフルコントロール

-t ターゲットディレクトリ名

--target-directory=ターゲットディレクトリ名

リンクファイルを作成するディレクトリを指定します。次のどれかの場合，エラー終了します。

- -T オプションと同時に指定した。
- ターゲットディレクトリ名にディレクトリ，またはディレクトリに対するシンボリックリンクのどちらでもないパスを指定した。
- 存在しないディレクトリをターゲットディレクトリ名に指定した。
- このオプションを複数指定した。

-T

--no-target-directory

引数ターゲットに指定したディレクトリに対するシンボリックリンクをたどらず，リンクファイル名として扱います。

次のどちらかを満たす場合，エラー終了します。

- 引数ターゲットに指定した内容と同名のディレクトリが存在した。
- -t オプションと同時に指定した。

-v

--verbose

引数リンク先パス名と，作成するリンクファイルを標準出力に出力します。出力形式は次のとおりです。

- ハードリンク作成時

  `引数リンク先パス名' => `作成するリンクファイル'
- シンボリックリンク作成時

  `引数リンク先パス名' -> `作成するリンクファイル'

--no-exist-directory

Windows の場合

このオプションを指定すると，シンボリックリンク作成時に，引数リンク先パス名に指定したファイル，ディレクトリが存在しない場合，ディレクトリに対するシンボリックリンクを作成します。

シンボリックリンクが指すディレクトリが存在しない状態で，ディレクトリに対するシンボリックリンクを先行して作成したい場合は，このオプションを指定してください。

なお，次のどちらかを満たす場合，このオプションの指定を無視します。

・引数リンク先パス名に指定したファイル，ディレクトリが存在する。

・-s オプションが指定されていない。

--no-exist-directory オプションと--no-exist-file オプションを同時に指定した場合は，最後に指定したオプションが有効になります。

UNIX の場合

指定を無視します。

--no-exist-file

Windows の場合

このオプションを指定すると，シンボリックリンク作成時に，引数リンク先パス名に指定したファイル，ディレクトリが存在しない場合，ファイルに対するシンボリックリンクを作成します。

シンボリックリンクが指すファイルが存在しない状態で，ファイルに対するシンボリックリンクを先行して作成したい場合は，このオプションを指定してください。

なお，次のどちらかを満たす場合，このオプションの指定を無視します。

・引数リンク先パス名に指定したファイル，ディレクトリが存在する。

・-s オプションが指定されていない。

--no-exist-file オプションと--no-exist-directory オプションを同時に指定した場合は，最後に指定したオプションが有効になります。

UNIX の場合

指定を無視します。

リンク先パス名

作成するリンクファイルが指すファイルパス名，またはディレクトリパス名を指定します。リンク先パス名は複数指定できます。リンク先パス名を 2 つ以上指定する場合は，引数ターゲットにディレクトリを指定する，または形式 2 によって-t オプションでディレクトリを指定してください。

ターゲット

作成するリンクファイル名，またはリンクファイルを作成するディレクトリ名を指定します。

リンクファイル名を指定した場合，指定したファイル名でリンクファイルを作成します。リンクファイルを作成するディレクトリ名を指定した場合，引数リンク先パス名に指定したファイルと同じ名前のリンクファイルを，指定したディレクトリ配下に作成します。

ターゲットの指定を省略した場合，カレントディレクトリにリンクファイルを作成します。

ディレクトリに対するシンボリックリンクを指定した場合，シンボリックリンクをたどり，リンク先のディレクトリにシンボリックリンクを作成します。ただし，次のどれかを指定することでシンボリックリンクをたどらないでリンクファイル名として扱うことができます。

- -T オプション
- --follow オプションに no を指定する
- 環境変数 ADSH_CMDLN_FOLLOW に NO を指定する

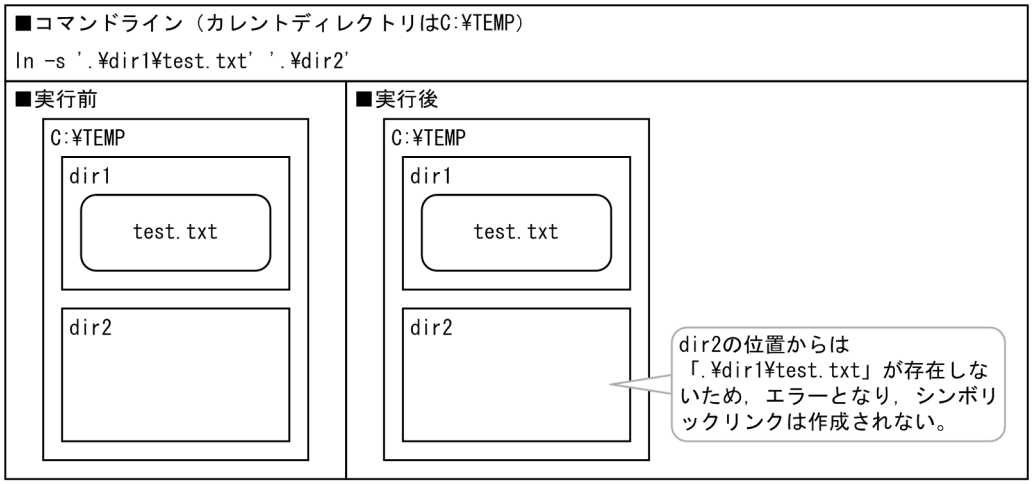
終了コード

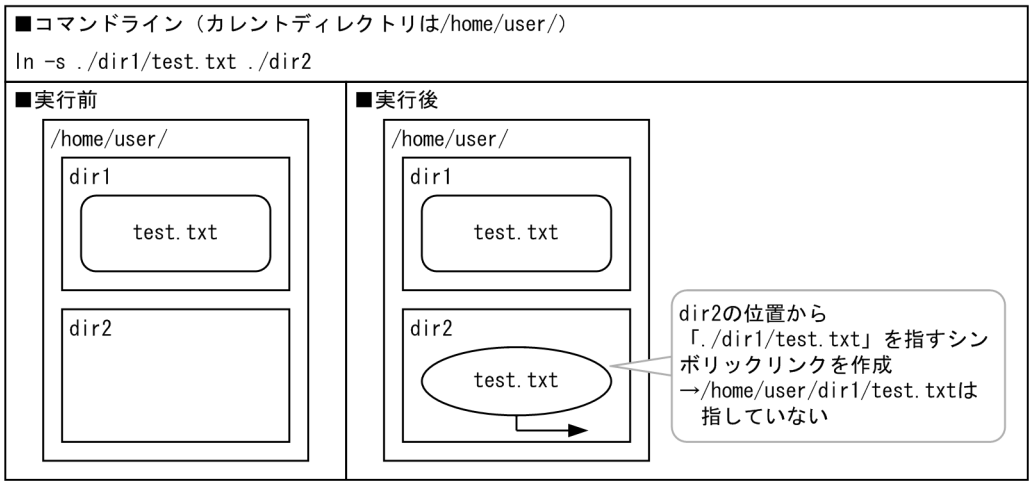
終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

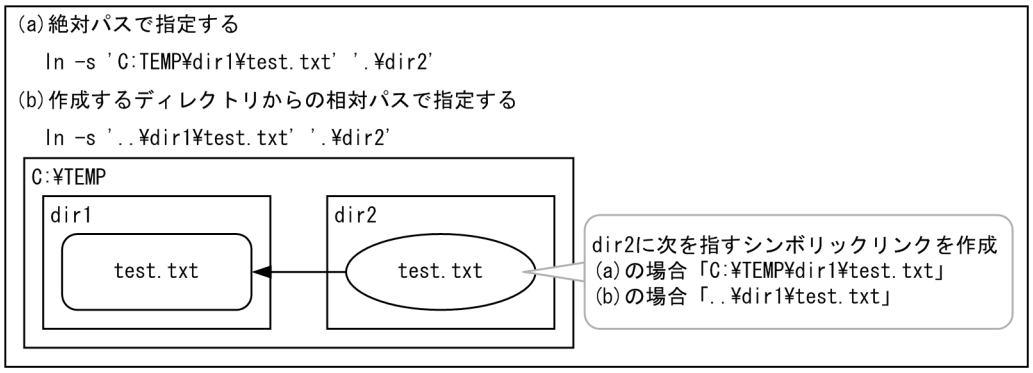
プラットフォーム共通の注意事項

- 1 つのファイルに対して複数のハードリンクを作成できます。しかし，OS，またはファイルシステムによって，1 つのファイルに対して作成できるハードリンク数には上限があります。そのため，上限を超えた場合，ハードリンクの作成に失敗します。
- 引数リンク先パス名，または引数ターゲットに指定したシンボリックリンクをたどる際にシンボリックリンクのネストの回数が OS の上限を超えるとエラーとなり，シンボリックリンクの作成に失敗します。
- 引数リンク先パス名，引数ターゲットに指定するファイルパスには，実行する OS に合わせたディレクトリ区切り文字を指定してください。ディレクトリ区切り文字の指定を誤った場合，リンク先をたどることができない場合があります。JP1/Advanced Shell で使用できるディレクトリ区切り文字の詳細は「2.2.3(1) JP1/Advanced Shell で使用するファイルの一覧」の「ファイルとパスの指定に関する注意事項」に関する説明を参照してください。
- 引数リンク先パス名を相対パスで指定すると，シンボリックリンクは自身が存在するディレクトリを起点として，その位置からリンク先を指すことになります。そのため，カレントディレクトリとシンボリックリンクを作成するディレクトリが異なる場合は注意してください。例を次に示します。





このため，カレントディレクトリとシンボリックリンクを作成するディレクトリが異なる場合は，引数リンク先パス名を絶対パスで指定するか，シンボリックリンクを作成するディレクトリからの相対パスで指定してください。



- 作成するリンクファイルと同名のファイル，ディレクトリが存在する環境で，-f オプションを指定，または-i オプションの応答で y,または Y を返した場合，ln コマンドはリンクファイルを作成するディレクトリに一時ファイルを作成します。このため，ln コマンドの実行を中断すると，一時ファイルが残る場合があります。この場合は手動で削除してください。
- リンクファイルを作成するディレクトリパス名の長さによっては一時ファイルの作成に失敗することがあります。その際は，既に存在する同名のファイルを削除，または名称を変更するか，作成するリンクファイル名を変更してください。
- 引数ターゲットにディレクトリに対するシンボリックリンクを指定した場合の解釈は，-T オプション,--follow オプション，環境変数 ADSH_CMDLN_FOLLOW の指定によって切り替えられます。同時に指定した場合の優先順位は-T オプション，--follow オプション，環境変数 ADSH_CMDLN_FOLLOW の順となります。
- -i オプションと-f オプションを同時に指定した場合，常に-i オプションの指定が有効になります。ただし，環境変数 ADSH_CMDLN_OPT_I_F に LAST を指定することで最後に指定したオプションを有効にすることができます。

UNIX 版限定の注意事項

次に対するハードリンクを作成することはできません。

- ディレクトリ
- 存在しないファイル
- 異なるファイルシステムのファイル

Windows 版限定の注意事項

- 次に対するハードリンクを作成することはできません。
  - ディレクトリ
  - 存在しないファイル
  - 異なるドライブレターのファイル

- NTFS 以外のファイルシステムにリンクファイルを作成できません。
- ln コマンドで作成したシンボリックリンクには，作成したディレクトリのアクセス権限を継承した上で，Everyone にフルコントロールのアクセス権限を付加します。
- 実行ファイルに対してハードリンク，またはシンボリックリンクを作成する場合は，拡張子「.bat」，「.com」，「.cmd」，または「.exe」を付加してください。また，ジョブ定義スクリプト内にも拡張子を付加したリンクファイル名を記述してください。
- 引数リンク先パス名に存在しないファイル，またはディレクトリを指定してシンボリックリンクを作成しようとすると，エラーとなりシンボリックリンクは作成されません。存在しないファイル，またはディレクトリに対してシンボリックリンクを作成する場合は，--no-exist-directory，--no-exist-file オプションを指定してください。
- 引数ターゲット，またはターゲットディレクトリには UNC 形式を指定できません。指定した場合はエラー終了します。また，引数リンク先パス名に UNC 形式を指定して，ハードリンクを作成するとエラー終了します。
- -f オプション指定時，または-f オプションを指定して y，Y を応答した場合であっても，すでに存在するファイルのアクセス権限によっては削除できないで，新しいリンクファイルの作成に失敗する場合があります。
- シンボリックリンク作成時にフルコントロールの割り当てに失敗した場合，不当なシンボリックリンクが残る場合があります。
- ファイルシステムの保護を目的とした製品をインストールしている環境では，シンボリックリンクを使用できない場合があります。ファイルシステムの保護を目的とした製品をインストールしている環境で，このコマンドを使用した場合，「Failed to rename a temporary file」というメッセージを出力し，エラー終了することがあります。また，このメッセージを出力した場合，一時ファイルが残ることがあります。この場合，JP1/Advanced Shell が提供する rm コマンドなどを使用して一時ファイルを削除し，シンボリックリンクを再作成してください。

使用例

ファイルへのハードリンクを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%ln test.txt hlink.txt

C:¥TEMP>%ADSH_OSCMD_DIR%ls -l
total 714
-rw------- 2 Administrators  357 Jun 01 15:05 hlink.txt
-rw------- 2 Administrators  357 Jun 01 15:05 test.txt
```

引数ターゲットを省略してファイルへのハードリンクを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%ln testdir¥new.txt

C:¥TEMP>%ADSH_OSCMD_DIR%ls -l
total 3572
-rw------- 2 Administrators  3572 Jun 04 11:19 new.txt
drwx------ 1 Administrators       Jun 04 11:19 testdir
```

```
C:¥TEMP>%ADSH_OSCMD_DIR%ls -l testdir
total 3572
-rw------- 2 70247321 3572 Jun 04 11:19 new.txt
```

ファイルへのシンボリックリンクを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%ln -s test.txt slink.txt

C:¥TEMP>%ADSH_OSCMD_DIR%ls -l
total 357
lrw------- 1 Administrators    0 Jun 01 15:07 slink.txt -> test.txt
-rw------- 1 Administrators  357 Jun 01 15:05 test.txt
```

ディレクトリへのシンボリックリンクを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%ln -s testdir slinkdir

C:¥TEMP>%ADSH_OSCMD_DIR%ls -l
total 0
lrw------- 1 Administrators  0 Jun 01 15:08 slinkdir -> testdir
drwx------ 1 Administrators    Jun 01 15:05 testdir
```

オプションエラーのメッセージを表示します。このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%ln -z
ln: illegal option -- z
usage: ln [-f] [-i] [-L] [-n] [-P] [-s] [-T] [-v] [--follow={yes | no}]
          [--no-exist-directory] [--no-exist-file] linkpath ... [target]
       ln [-f] [-i] [-L] [-n] [-P] [-s] [-v] [--follow={yes | no}]
          [--no-exist-directory] [--no-exist-file] -t target_directory linkpath ...
```

8.4.21　ls コマンド（ファイルまたはディレクトリの内容を表示する）

形式

```
ls [-1] [-A] [-a] [-C] [-c] [-d] [-F] [-f] [-g] [-h] [-i] [-k]
   [-L] [-l] [-m] [-n] [-p] [-q] [-R] [-r] [-S] [-s] [-T] [-t]
   [-u] [-x]
   [--format=表示形式] [--full-time]
   [--indicator-style=ファイル種別様式] [--sort=ソートキー]
   [--time=ファイル日時種別]
   [パス名 ...]
```

機能

ディレクトリの内容を表示します。ディレクトリの内容は標準出力に出力されます。出力内容のうちパーミッションの表示内容は次のようになります。

- 最初の 1 文字は，対象の種類として次の内容が表示されます。

  -：通常ファイル

  b：ブロック型スペシャルファイル

  c：キャラクタ型スペシャルファイル

  d：ディレクトリ

  l：シンボリックリンク

  p：FIFO

  s：ソケット

- 以降の 9 文字は，3 文字ごとの 3 つのセットとして解釈され，所有者，グループ，その他のユーザーのパーミッションを表示します。Windows の場合は所有者だけのパーミッションを表示します。

パーミッション内の順序	表示される文字※	権限
1 つ目	r	所有者による読み取り
2 つ目	w	所有者による書き込み
3 つ目	x	所有者による実行
	s	所有者によるセットユーザー ID またはセットグループ ID/実行
	S	所有者によるセットユーザー ID またはセットグループ ID/非実行
4 つ目	r	グループによる読み取り
5 つ目	w	グループによる書き込み
6 つ目	x	グループによる実行
	s	グループによるセットユーザー ID またはセットグループ ID/実行
	S	グループによるセットユーザー ID またはセットグループ ID/非実行
7 つ目	r	その他のユーザーによる読み取り
8 つ目	w	その他のユーザーによる書き込み
9 つ目	x	その他のユーザーによる実行
	t	その他のユーザーによるスティッキービット/実行
	T	その他のユーザーによるスティッキービット/非実行

注※

表示される文字の意味は次のとおりです。

文字	意味
-	対応するパーミッションが与えられていない
r	Windows の場合，ファイルまたはディレクトリが存在する UNIX の場合，読み取り権限が与えられている
w	Windows の場合，読み取り専用属性が設定されていない UNIX の場合，書き込み権限が与えられている
x	Windows の場合，次のどれかに該当する • 拡張子が「.com」,「.exe」,「.cmd」または「.bat」である • ディレクトリである UNIX の場合，実行権限が与えられている
s	セットユーザー ID またはセットグループ ID が与えられていて，実行権限が与えられている【UNIX 限定】
S	セットユーザー ID またはセットグループ ID が与えられていて，実行権限が与えられていない【UNIX 限定】
t	スティッキービットが与えられていて，実行権限が与えられている【UNIX 限定】
T	スティッキービットが与えられていて，実行権限が与えられていない【UNIX 限定】

-g オプション，-l オプション，-n オプション，--full-time オプションを指定すると，ロングフォーマットの出力形式となります。

ロングフォーマットとは，ファイル名やディレクトリ名の出力だけでなく，ファイルやディレクトリの詳細な情報を出力する形式です。また，ロングフォーマット形式に-h オプション，-T オプション，-u オプションを組み合わせると，各項目の出力形式を変更できます。

引数

-1

--format=single-column

1 行に 1 エントリ（1 列）で出力します。

-A

--almost-all

「.」および「..」を除いてすべてのエントリを出力します。

-a

--all

「.」で始まるファイル名およびディレクトリ名を含めて出力します。

-C

--format=vertical

縦方向にソートして，複数列で出力します。端末出力時のデフォルトです。

-c

--time=ctime

--time=status

ソート（-t オプション）やリスト出力（-g オプション，-l オプション，-n オプション，--full-time オプション）を指定した場合，ファイルの最終修正日時ではなく，ファイル情報の最終変更日時を使います。

Windows の場合，指定が無視されます。

-d

--directory

ディレクトリの内容を表示しないで，ディレクトリ名を出力します。

Windows の場合，このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。シンボリックリンクのリンク先のディレクトリの内容を出力したい場合は，-L オプションと同時に指定してください。

-F

--classify

--indicator-style=classify

ディレクトリ名の後ろに「/」，実行可能ファイルの後ろに「*」，シンボリックリンクの後ろに「@」，FIFO 名の後ろに「|」，ソケットの後ろに「=」を出力します。Windows の場合，指定が無視されます。

-f

--sort=none

ソートをしないで出力します。

-g

ロングフォーマットで出力しますが，所有者は出力しません。

Windows の場合，このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。シンボリックリンクのリンク先のディレクトリの内容を出力したい場合は，-L オプションと同時に指定してください。

-h

--human-readable

ロングフォーマット使用時に，ファイルサイズを 2 のべき乗で割って小数点第 2 位を四捨五入した値をファイルサイズとして出力します。ファイルサイズには，サイズ文字（M：1048576，K：1024）が付加されます。

ディレクトリ内にスペシャルファイルが存在する場合，-h オプションは無視されます。

-i

--inode

UNIX の場合，ファイルごとに inode 番号を出力します。

Windows の場合，常に 0 を出力します。

-k

UNIX の場合，-l オプション，-g オプション，-s オプションおよび--full-time オプションで出力するディレクトリの合計ブロック数と，-s オプションで表示するファイルサイズを KB 単位で出力します。

Windows の場合，-s オプションで表示するファイルサイズを KB 単位で出力します。

-L

--dereference

シンボリックリンクではなく，参照しているファイルの情報を出力します。

-l

--format=long

--format=verbose

次の項目をロングフォーマットで出力します。日時を完全な形式で出力するには--full-time オプションを使用してください。

- UNIX の場合

  アクセス権，リンク数，所有者名，グループ名，サイズ，最終修正日時，ファイル名またはディレクトリ名。ただし，表示対象がディレクトリの場合は，「.」や「..」も含めて，そのディレクトリ直下に存在するディレクトリの総数を表示します。

  ファイルがシンボリックリンクの場合は，リンク先のパス名を「->」のあとに表示します。

- Windows の場合

  ファイル所有者のアクセス権，リンク数，所有者名，サイズ（ディレクトリの場合は表示しません），最終修正日時，ファイル名またはディレクトリ名。

  ファイルがシンボリックリンクの場合は，リンク先のパス名を「->」のあとに表示します。

  このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。シンボリックリンクのリンク先のディレクトリの内容を出力したい場合は，-L オプションと同時に指定してください。

-m

--format=commas

ファイル名をコンマ（,）で区切って出力します。

-n

--numeric-uid-gid

- UNIX の場合

  ユーザー名，グループ名の代わりにユーザー ID，グループ ID を出力します。

- Windows の場合

  ユーザー ID に 0 を出力します。また，グループ ID は出力しません。

  このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。シンボリックリンクのリンク先のディレクトリの内容を出力したい場合は，-L オプションと同時に指定してください。

-p

--indicator-style=slash

ディレクトリ名の直後に「/」を出力します。

-q

--hide-control-chars

ファイル名に表示できない文字が使われていた場合，代わりに「?」を出力します。端末出力時のデフォルトです。

-R

--recursive

サブディレクトリを再帰的に出力します。

-r

--reverse

逆順にソートして出力します。

-S

--sort=size

サイズでソートして，最も大きいファイルを先頭に出力します。

-s

--size

UNIX の場合，ファイルのブロック数を出力します。-k オプションおよび環境変数 BLOCKSIZE が定義されていない場合は，512 バイトのブロック単位で切り上げて出力します。

Windows の場合，ブロック数は常に 0 と出力されます。

-T

月，日，時間，分，秒，年を含む日時情報を出力します。-g オプション，-l オプションまたは-n オプションのどれかと同時に指定した場合に有効となります。

-t

--sort=time

ファイルの最終修正日時でソートします。最新の修正が先頭になります。

-u

--time=atime

--time=access

--time=use

ソート（-t オプション）およびリスト表示（-g オプション，-l オプション，-n オプション，--full-time オプション）の場合，ファイルの最終修正日時ではなくファイルの最終アクセス日時を使用します。

Windows の場合，指定が無視されます。

-x

--format=across

--format=horizontal

-C と同様に複数列で出力しますが，横方向にソートして出力します。

-format=表示形式

ファイルまたはディレクトリの内容を表示する場合の表示形式を指定します。

表示形式に次の値を指定できます。--format オプションを複数指定した場合は，最後の指定が有効になります。

across または horizontal

横方向にソートして，複数列で出力します。-x オプションと同じです。

commas

ファイル名をコンマ（,）で区切って出力します。-m オプションと同じです。

long または verbose

ロングフォーマットで表示します。-l オプションと同じです。

single-column

1 行に 1 エントリ（1 列）で出力します。-1 オプションと同じです。

vertical

縦方向にソートして，複数列で出力します。-C オプションと同じです。

--full-time

-l オプションと同様の項目を出力します。ただし，日時に関する情報は標準の省略形式ではなく，完全な形式で出力します。

日時部分の出力形式は次のとおりです。

YYYY-MM-DD hh:mm:ss.nnnnnnnnn +/-hhmm

YYYY：西暦年

MM：月

DD：日

hh：時

mm：分

ss：秒

nnnnnnnnn：1 秒未満の日時。常に 000000000 と出力します。

+/-hhmm：タイムゾーン。UTC からの時差を示します。

Windows の場合，このオプションを指定して，パス名に末尾がディレクトリ区切り文字のディレクトリへのシンボリックリンクを指定しても末尾のディレクトリ区切り文字が無視されます。シンボリックリンクのリンク先のディレクトリの内容を出力したい場合は，-L オプションと同時に指定してください。

--indicator-style=ファイル種別様式

表示するファイルの種別を示す情報の様式を指定します。

ファイル種別様式には次の値を指定できます。

classify

ファイルの種別を表す文字をファイル名の直後に出力します。ディレクトリ名の場合は直後に「/」を表示します。-F オプションと同じです。

ファイルの種別を表す文字の内容は，-F オプションの説明を参照してください。

slash

ディレクトリ名の直後に「/」を表示します。-p オプションと同じです。

--indicator-style=classify と--indicator-style=slash を同時に指定した場合，classify の指定が有効になります。

Windows の場合，classify を指定しても無視されます。

--sort=ソートキー

複数のファイルを表示する場合に，ソートキーに指定したファイル情報をキーにしてソートして表示します。--sort オプションを複数指定すると，最後の指定が有効になります。

ソートキーには次の値を指定できます。

size

ファイルのサイズでソートします。-S オプションと同じです。

time

ファイルの最終修正日時でソートします。-t オプションと同じです。--time オプションを使用して，ファイルの最終アクセス日時や最終変更日時でのソートもできます。

none

ソートしないで出力します。-f オプションと同じです。

--time=ファイル日時種別

ソート（-t）やリスト表示（-g オプション，-l オプション，-n オプション，--full-time オプション）で使用するファイル日時は，ここで指定した種類の日時が適用されます。--time オプションを複数指定すると，最後の指定が有効になります。Windows の場合，--time オプションを指定しても無視されます。

ファイル日時には次の値を指定できます。

atime，access，または use

ファイルの最終アクセス日時を使用します。-u オプションと同じです。

ctime または status

ファイル情報の最終変更日時を使用します。-c オプションと同じです。

パス名

出力するファイル名またはディレクトリ名を指定します。複数指定することもできます。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- -1 オプション，-C オプション，-l オプション，-m オプション，-x オプションおよび--full-time オプションは，最後に指定したオプションが有効となります。

  ただし，-l オプションと--full-time オプションを同時に指定した場合は，--full-time オプションが有効となります。

- -A オプションの指定の有無に関係なく，常に「.」または「..」を除いたすべてのエントリ（「.」で始まるエントリを含む）が出力対象となります。
- ブロックサイズのデフォルトは 512 バイトです。
- ファイルの日時がコマンド実行日時より 182 日（約半年）以上前の場合や，182 日以上先の場合は，日時の代わりに年を表示します。

  ただし，--full-time オプションを指定した場合はこの限りではありません。

- UNIX の場合，ユーザー名が取得できないときはユーザー ID，グループ名が取得できないときはグループ ID を表示します。

- Windows の場合，ユーザー名が取得できないときは「...」と表示します。
- Windows の場合，ディレクトリ内のファイルサイズ合計はバイト単位で表示します。
- Windows 上で隠しファイル属性の場合も表示対象となります。
- このコマンドは，次の環境変数が有効になります。
  - 環境変数 COLUMNS

    -C オプションの指定による，複数列で出力したときの 1 行当たりの出力幅を定義します。
  - 環境変数 BLOCKSIZE

    UNIX の場合，-s オプションの指定で表示する，ブロック数の 1 ブロック当たりのサイズを定義します。環境変数 BLOCKSIZE には 512 から 1GB（1024×1024×1024）まで指定できます。範囲外の値を指定した場合は次のように処理し，警告メッセージを標準エラー出力に出力して，後続の処理を続行します。

    ・環境変数 BLOCKSIZE に 512 より小さい値を指定した場合

    ブロックサイズを 512 バイトとします。

    ・環境変数 BLOCKSIZE に 1G（1024×1024×1024）より大きい値を指定した場合

    ブロックサイズを 1G（1024×1024×1024）バイトとします。

    環境変数 BLOCKSIZE でブロックサイズを変更する場合は，512 の倍数を指定してください。512 の倍数でない場合，余りは切り捨てられます。例えば，1,500 バイトが定義されている場合，1,024 バイトとして扱います。数値の後ろに，何倍であるかを示すサイズ文字（G（1024×1024×1024），M（1024×1024），K（1024））を指定できます。数値とサイズ文字以外を指定した場合，ブロックサイズを 512 バイトとして，警告メッセージを標準エラー出力に出力し，後続の処理を続行します。
  - 環境変数 TZ

    UNIX の場合，日付時刻の表示に使用されるタイムゾーンを定義します。

    Windows の場合，日付時刻の表示は，コントロールパネルの「日付と時刻のプロパティ」ダイアログボックスで定義されているタイムゾーンが使用されます。環境変数 TZ の値は影響しません。

    ただし，--full-time オプションで表示するタイムゾーンは，環境変数 TZ の値とコントロールパネルの「日付と時刻のプロパティ」ダイアログボックスで定義されているタイムゾーンを使用するため，環境変数 TZ の値とコントロールパネルの「日付と時刻のプロパティ」ダイアログボックスのタイムゾーンは同じにしてください。同じでない場合，--full-time オプションで表示されるタイムゾーンは正しく表示されません。
- Windows の場合，ドライブレターを指定してディレクトリを参照すると，指定の仕方によってはコマンドを実行しているカレントディレクトリを参照します。

  次に示すフォルダ構成を例に指定例を説明します。

```
カレントドライブ        別ドライブ
D:¥                     E:¥
  |                       |
  ├X                      ├R
  ├Y                      ├S
  └Z                      └T
    |                       |
```

```
    ├file1                  ├fileA
    ├file2                  ├fileB
    └file3                  └fileC
```

ls コマンドにカレントドライブを「D:¥」でなく「D:」と指定すると，コマンドを実行したカレントディレクトリ（D:¥Z）配下の情報を表示します。

```
D:¥Z>ls -l D:
total 462
-rw------- ouser001  154 Jun 02 15:23 file1
-rw------- ouser001  154 Jun 02 15:23 file2
-rw------- ouser001  154 Jun 02 15:23 file3
```

カレントドライブ（D:¥）を指定すると，指定したドライブレターの直下（D:¥）の情報を表示します。

```
D:¥Z>ls -l D:¥
total 0
drwx------ ouser001   Jun 02 15:22 X
drwx------ ouser001   Jun 02 15:23 Y
drwx------ ouser001   Jun 02 15:25 Z
```

別ドライブ（E:）を指定すると，指定したドライブレターの直下（E:¥）の情報を表示します。

```
D:¥Z>ls -l E:
total 0
drwx------ ouser001   Jun 02 15:24 R
drwx------ ouser001   Jun 02 15:24 S
drwx------ ouser001   Jun 02 15:25 T
```

別ドライブ（E:¥）を指定すると，指定したドライブレターの直下（E:¥）の情報を表示します。

```
D:¥Z>ls -l E:¥
total 0
drwx------ ouser001   Jun 02 15:24 R
drwx------ ouser001   Jun 02 15:24 S
drwx------ ouser001   Jun 02 15:25 T
```

使用例

- オプションを指定しない場合，カレントディレクトリのファイルを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls
HARDLINK.txt        TestLog             test_result.txt
SYMLINK.txt         test_data.txt       uap.exe
```

- -1 オプションを指定して，1 列で表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -1
HARDLINK.txt
SYMLINK.txt
TestLog
test_data.txt
test_result.txt
uap.exe
```

- -A オプションを指定して，「.」および「..」を除いたすべてのエントリを表示します。Windows の場合は，-A オプションの指定に関係なく「.」で始まるエントリを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -A
HARDLINK.txt        TestLog             test_result.txt
SYMLINK.txt         test_data.txt       uap.exe
```

- -a オプションを指定して，「.」で始まるディレクトリを含めて表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -a
.                   SYMLINK.txt         test_result.txt
..                  TestLog             uap.exe
HARDLINK.txt        test_data.txt
```

- -C オプションを指定し，縦方向にソートして複数列で表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -C
HARDLINK.txt        TestLog             test_result.txt
SYMLINK.txt         test_data.txt       uap.exe
```

- -f オプションを指定して，ソートをしないで表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -f
HARDLINK.txt        TestLog             test_result.txt
SYMLINK.txt         test_data.txt       uap.exe
```

- -g オプションを指定し，所有者表示なしのロングフォーマットで表示します。Windows の場合はグループ名を表示しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -g
total 337744
-rw------- 2 102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1      0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1        Jul 06 16:58 TestLog
-rw------- 1 102000 Jul 06 16:20 test_data.txt
-rw------- 2 102000 Jul 06 16:26 test_result.txt
-rwx------ 1  31744 Jun 12 16:23 uap.exe
```

- -h オプションをロングフォーマットと共に指定して，ファイルサイズにサイズ文字を付加します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lh
total 337744
-rw------- 2 Administrators   99.6K Jul 06 16:26 HARDLINK.txt
lrw------- 1 Administrators      0B Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58 TestLog
-rw------- 1 Administrators   99.6K Jul 06 16:20 test_data.txt
-rw------- 2 Administrators   99.6K Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators   31.0K Jun 12 16:23 uap.exe
```

- -i オプションを指定して，ファイルごとに inode 番号を表示します。Windows の場合は inode 番号に 0 を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -i
0 HARDLINK.txt      0 TestLog           0 test_result.txt
0 SYMLINK.txt       0 test_data.txt     0 uap.exe

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -il
```

```
total 337744
0 -rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
0 lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
0 drwx------ 1 Administrators         Jul 06 16:58 TestLog
0 -rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
0 -rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
0 -rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

- -l オプションを指定して，ロングフォーマットで表示します。Windows の場合は，所有者のアクセス権限だけ表示します。グループ名，ディレクトリサイズは表示しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -l
total 337744
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58 TestLog
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

- -l オプションのリスト表示で-c オプションを指定して，ファイルの最終修正日時ではなくファイル情報の最終変更日時を表示します。Windows の場合は，-c オプションの指定を無視してファイルの最終修正日時を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lc
total 337744
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58 TestLog
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

- -l オプションのリスト表示で-u オプションを指定して，ファイルの最終修正日時ではなくファイルの最終アクセス日時を表示します。Windows の場合は，-u オプションの指定を無視してファイルの最終修正日時を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lu
total 337744
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58 TestLog
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

- -m オプションを指定して，ストリーム出力形式でコンマで区切って表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -m
HARDLINK.txt, SYMLINK.txt, TestLog, test_data.txt,
test_result.txt, uap.exe
```

- -n オプションを指定して，ユーザー名，グループ名の代わりにユーザー ID，グループ ID を表示します。Windows の場合はユーザー ID に 0 を表示します。また，グループ ID を表示しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -n
total 337744
-rw------- 2 0  102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1 0       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 0         Jul 06 16:58 TestLog
-rw------- 1 0  102000 Jul 06 16:20 test_data.txt
-rw------- 2 0  102000 Jul 06 16:26 test_result.txt
-rwx------ 1 0   31744 Jun 12 16:23 uap.exe
```

- -p オプションを指定して，ディレクトリの後ろに「/」を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -p
HARDLINK.txt        TestLog/            test_result.txt
SYMLINK.txt         test_data.txt       uap.exe

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -alp
total 337744
drwx------ 1 Administrators           Jul 06 16:29 ./
drwx------ 1 TrustedInstaller         Jan 01  1980 ../
-rw------- 2 Administrators    102000 Jul 06 16:26 HARDLINK.txt
lrw------- 1 Administrators         0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators           Jul 06 16:58 TestLog/
-rw------- 1 Administrators    102000 Jul 06 16:20 test_data.txt
-rw------- 2 Administrators    102000 Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators     31744 Jun 12 16:23 uap.exe
```

- -q オプションを指定して，表示できない文字を「?」で表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -q ..¥dir1
.sub1     file2.txt sub4      wc2.c     wc4.c
.sub2     sub3      wc1.c     wc3.c     ????.txt
```

- -R オプションを指定して，サブディレクトリを再帰的に表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -R ..¥dir1
.sub1     file2.txt sub4      wc2.c     wc4.c
.sub2     sub3      wc1.c     wc3.c     ????.txt

..¥dir1¥.sub1:

..¥dir1¥.sub2:

..¥dir1¥sub3:

..¥dir1¥sub4:
```

- -r オプションを指定して，逆順にソートして表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -r
uap.exe             test_data.txt       SYMLINK.txt
test_result.txt     TestLog             HARDLINK.txt

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -rl
total 337744
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
```

```
drwx------ 1 Administrators         Jul 06 16:58 TestLog
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
```

- -S オプションを指定して，サイズでソートして最も大きいファイルを先頭に表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -S
HARDLINK.txt        test_result.txt     SYMLINK.txt
test_data.txt       uap.exe             TestLog

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lS
total 337744
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58 TestLog
```

- -s オプションを指定して，ファイルのブロック数を表示します。Windows の場合は 0 です。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -sl
total 337744
0 -rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
0 lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
0 drwx------ 1 Administrators         Jul 06 16:58 TestLog
0 -rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
0 -rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
0 -rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

- -T オプションを指定して，月，日，時間，分，秒，年などの時間情報を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lT
total 337744
-rw------- 2 Administrators  102000 Jul 06 16:26:40 2015 HARDLINK.txt
lrw------- 1 Administrators       0 Jul 06 16:27:11 2015 SYMLINK.txt -> .¥test_data.txt
drwx------ 1 Administrators         Jul 06 16:58:21 2015 TestLog
-rw------- 1 Administrators  102000 Jul 06 16:20:28 2015 test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26:40 2015 test_result.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23:18 2015 uap.exe
```

- -t オプションを指定して，ファイルの最終修正日時でソートします。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -t
TestLog             HARDLINK.txt        test_data.txt
SYMLINK.txt         test_result.txt     uap.exe

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lt
total 337744
drwx------ 1 Administrators         Jul 06 16:58 TestLog
lrw------- 1 Administrators       0 Jul 06 16:27 SYMLINK.txt -> .¥test_data.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 HARDLINK.txt
-rw------- 2 Administrators  102000 Jul 06 16:26 test_result.txt
-rw------- 1 Administrators  102000 Jul 06 16:20 test_data.txt
-rwx------ 1 Administrators   31744 Jun 12 16:23 uap.exe
```

- -x オプションを指定し，横方向にソートして複数列で表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -x
HARDLINK.txt        SYMLINK.txt         TestLog
test_data.txt       test_result.txt     uap.exe
```

- --full-time オプションを指定し，日時を完全な形式でロングフォーマット表示します。

```
C:¥Program Files¥HITACHI¥JP1AS¥JP1ASE¥cmd>ls --full-time
total 2638901
-rwx------ 1 SYSTEM  327168 2014-01-10 19:47:42.000000000 +0900 awk.exe
-rwx------ 1 SYSTEM   10240 2014-01-10 19:45:32.000000000 +0900 basename.exe
-rwx------ 1 SYSTEM   12800 2014-01-10 19:48:44.000000000 +0900 cat.exe
-rwx------ 1 SYSTEM   11264 2014-01-10 19:48:44.000000000 +0900 cmp.exe
-rwx------ 1 SYSTEM   19968 2014-01-10 19:48:40.000000000 +0900 cp.exe
-rwx------ 1 SYSTEM   14848 2014-01-10 19:48:04.000000000 +0900 cut.exe
-rwx------ 1 SYSTEM   10240 2014-01-10 19:48:36.000000000 +0900 date.exe
-rwx------ 1 SYSTEM  237056 2014-01-10 19:48:14.000000000 +0900 diff.exe
-rwx------ 1 SYSTEM  224256 2014-01-10 19:45:28.000000000 +0900 egrep.exe
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥>ls -z
ls: illegal option -- z
usage: ls [-1AaCcdFfghikLlmnpqRrSsTtux] [--format=word] [--full-time]
          [--indicator-style=word] [--sort=word] [--time=word] [file ...]
```

8.4.22　mkdir コマンド（ディレクトリを作成する）

形式

```
mkdir [-p] [-m パーミッション] ディレクトリ ...
```

機能

ディレクトリを作成します。

引数

-p

必要に応じて，存在しない中間のディレクトリを作成します。

-m パーミッション

UNIX の場合，作成したディレクトリにパーミッションを設定します。umask 値は反映されません。

パーミッションは 8 進数の数値またはシンボルを指定します。

数値を指定した場合，8 進数以外または 8 進数の 07777（10 進数の 4095）より大きな値を指定するとエラーとなります。

シンボルを指定した場合，何も指定されていない状態（数値表現での 0）に対して設定，追加および削除をします。1 つまたは複数のシンボルで指定された結果が検索に使用されます。

シンボルは 3 つの部分から構成されます。次に示すシンボルを 1 つまたは複数指定します。複数指定する場合は，コンマ（,）でシンボル間を区切ります。

シンボル内の順序	指定できる値
1 つ目	アクセス権を設定する項目を指定します。複数同時に指定できます。指定できる項目を次に示します。省略するとすべてのユーザーが仮定されます。 • u：所有者 • g：グループ • o：その他 • a：全ユーザー
2 つ目	モードに対する操作を指定します。1 つ目のシンボルで指定した項目に対して次の処理をします。 • ＝：アクセス権の設定（上書き） • ＋：アクセス権の追加 • -：アクセス権の削除 設定，追加および削除する値は，3 つ目のシンボルで指定します。 3 つ目のシンボルに続けて 2 つ目および 3 つ目のシンボルを記述できます。このとき 3 つ目のシンボルは省略できます。
3 つ目	設定するアクセス権を指定します。複数同時に指定できます。指定できる値を次に示します。 • r：読み取り • w：書き込み • x：実行 • s：実行時にユーザーまたはグループ ID を設定する • t：スティッキービット • u：モードに現在設定されている所有者のアクセス権 • g：モードに現在設定されているグループのアクセス権 • o：モードに現在設定されているその他のアクセス権 省略するとアクセス権を設定する項目を消去します。消去した値を 2 つ目のシンボルに従って設定，追加および削除します。2 つ目で設定する追加および削除では，3 つ目で設定する値は変化しません。 s と t の指定は，1 つ目で o だけを指定した場合には無視されます。

シンボルの指定例を次の表に示します。

-perm の指定値	同等の数値指定	説明
u=x,g=w	120	u に対して x を設定し，g に対して w を設定しています。
u=x,g=u	110	u に対して x を設定し，g に対して u と同じ値を設定しています。
u=x,=u	111	u に対して x を設定し，そのあと a（省略値）に u と同じ値を設定しています。
u=x,u=w	200	u に対して x を設定し，そのあと u に対して w を設定（上書き）しています。
u=x,u+w	300	u に対して x を設定し，そのあと u に対して w を追加しています。

-perm の指定値	同等の数値指定	説明
ug=x	110	u と g に対して x を設定しています。
u=rw	600	u に対して r および w を設定しています。
u=r+x	500	u に対して r を設定し，x を追加しています。
u=r=w	200	u に対して r を設定し，さらに w を設定（上書き）しています。
=x,u=	011	a（省略値）に x を設定し，u の設定を消去しています。
=	000	a（省略値）を消去しています。

Windows の場合，指定は無視されます。

ディレクトリ

作成するディレクトリ名を指定します。ディレクトリ名は複数指定できます。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- Windows の場合，-m オプションは無視されます。モードの指定はできません。

使用例

- C:¥USR¥JP1 ディレクトリ下に Dir2 ディレクトリを作成します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mkdir C:¥USR¥JP1¥Dir2
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mkdir -w
mkdir: illegal option -- w
usage: mkdir [-p] [-m mode] directory ...
```

8.4.23 mv コマンド（ファイルまたはディレクトリを移動する）

形式

```
mv [-f] [-i] [-u] 移動元 移動先
mv [-f] [-i] [-u] 移動元 ... 移動先ディレクトリ
```

機能

ファイルまたはディレクトリを移動します。ファイル名またはディレクトリ名も変更できます。

引数

-f

--force

確認しないでパスを上書きします。-i オプションの前に指定すると無視されます。

-i

--interactive

上書きする場合に確認します。標準入力からy またはY を応答すると，上書きします。-f オプションの前に指定すると無視されます。

-u

--update

ディレクトリ以外のファイルの移動で，移動先ファイルがすでに存在し，ファイルの最終修正日時が移動元より新しい場合（同じ場合を含む），移動しません。ファイルの最終修正日時は，秒未満の値は切り捨てて判定します。

移動元

移動するパス名を指定します。移動元には，複数のパス名を指定できます。

移動先

移動先のパス名を指定します。移動元，移動先にパス名を指定した場合，ファイル名またはディレクトリ名を変更することもできます。

移動先ディレクトリ

移動先のディレクトリ名を指定します。移動元に複数のパス名を指定した場合，複数のファイルやディレクトリを移動できます。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- -i オプションと-f オプションは最後に指定されたオプションが有効となります。
- Windows の場合，オーバーライド時にオーナーのアクセス権以外は表示しません。
  表示するパーミッションの詳細については，「8.4.21　ls コマンド（ファイルまたはディレクトリの内容を表示する）」を参照してください。
- Windows の場合，グループおよびモードは保持されません。
- Windows の場合，移動先のファイル名は移動元に指定したファイル名で作成されます。また，Windows の場合，ファイル名の英大文字は英小文字に置き換えられます。例えば，移動対象のファイル名がA.txt の場合，mv a.txt tmpdir と実行すると，tmpdir の中のファイル名はa.txt になります。
- Windows の場合，ファイルをバイナリモードで入出力します。改行コードは変換しません。
- Windows の場合，移動先に移動元と同じ名称のファイルが存在する状態でファイルの移動をしようとすると，実行する Windows 環境の状態によって「Permission denied」を出力してエラー終了することがあります。
- Windows の場合，別ドライブへ移動するときに，移動元ファイルに読み取り専用属性が設定されていると，ファイルごとに 1550 ミリ秒の待ち時間が発生します。ワイルドカード指定などによる複数ファイルの指定や，ディレクトリ内の複数のファイルを移動するときは，事前に読み取り専用属性を解除してください。
- UNIX の場合，mv コマンドでファイルおよびディレクトリを移動したとき，次の条件をすべて満たすと，移動後のファイルおよびディレクトリの所有者はmv コマンドの実行者になります。
  - 一般ユーザーがmv コマンドを実行した。
  - 移動元ファイルの所有者がmv コマンドの実行者と異なる。
  - 移動元と移動先のファイルシステムが異なる。

  また，次の情報は引き継がれません。
  - 移動元ファイルに設定されていた setuid ビットと setgid ビットのアクセス権情報
  - 移動元ディレクトリに設定されていた setuid ビット，setgid ビット，スティッキービットのアクセス権情報
  - -u オプション指定で，移動先が新しい（同じ場合を含む）ため移動しない場合は，エラーではなく，正常終了します。

使用例

- -i オプションを指定して，移動先ファイルに上書きするかどうかを確認します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mv -i ..¥dir1¥file1.txt  ..¥dir1¥file2.txt
overwrite ..¥dir1¥file2.txt?
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mv -w
mv: illegal option -- w
usage: mv [-fiu] source target
       mv [-fiu] source ... directory
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mv file3.txt file4.txt
mv: file3.txt: No such file or directory
```

8.4.24　paste コマンド（複数のファイルを行単位で連結する）

形式

```
paste [-s] [-d リスト] [パス名 ...]
```

機能

複数のファイルを行単位に連結して標準出力に出力します。ファイル内のすべての行を 1 行に結合してから複数のファイルを連結することもできます。

引数

-s

ファイル内のすべての行を区切り文字によって結合して 1 つの行にします。

-s オプションを指定しない場合は，各ファイルの同じ行番号の行を区切り文字で結合します。

-d リスト

結合する行と行の間に挿入する区切り文字をリストに指定します。-d オプションを指定しない場合，タブ文字が指定されたものとします。

スペースまたはタブを指定する場合は，"（ダブルクォーテーション）で囲んでください。

区切り文字には次の特殊文字も指定できます。

使用できる特殊文字	意味	備考
¥n	改行文字	Windows の場合，[CR] ＋ [LF] が改行文字となります。

使用できる特殊文字	意味	備考
¥n	改行文字	UNIX の場合，［LF］が改行文字となります。
¥t	タブ文字	－
¥¥	1 つのバックスラッシュ文字	－
¥0	空文字列	長さ 0 の文字列（""）です。結合する行と行の間に区切り文字を挿入しません。

（凡例）

－：該当なし

特殊文字を指定する場合，paste コマンドを実行するシェルによってはエスケープ文字として扱われるため，特殊文字をクォーテーション（"または'）で囲む必要があります。特殊文字以外の文字の直前に「¥」を指定した場合，「¥」は無視され，「¥」の直後の文字が区切り文字として使用されます。また，「¥」だけを指定した場合はエラーで終了します。

区切り文字は複数指定できます。区切り文字を複数指定した場合の動作を次に示します。

- 行を結合するたびに区切り文字を取り出して，行と行の間に挿入します。区切り文字は先頭の区切り文字から順に取り出されます。
- -s オプションが指定されていない場合，結合した行を出力したときは，再び先頭の区切り文字から順に取り出されます。
- -s オプションが指定されている場合，ファイル内のすべての行を結合して出力したときは，再び先頭の区切り文字から順に取り出されます。
- 区切り文字を取り出すときに，-d オプションに指定した区切り文字を使い切っていた場合は再び先頭の区切り文字から順に取り出されます。

パス名

連結して出力するファイルのパス名を指定します。パス名を指定しない，またはパス名に「-」を指定した場合は，標準入力から入力します。

パス名，および「-」は複数指定できます。パス名と「-」を混在して指定することもできます。

複数のファイルを指定した場合，オープンに失敗したファイルがあると次のように動作します。

- -s オプションが指定されていないときはエラーメッセージを出力し，終了コード 1 でエラー終了します。標準出力には何も出力されません。
- -s オプションが指定されているときはオープンに失敗したファイル対してエラーメッセージを出力し，処理を続行します。すべてのファイルに対して処理をしたあと終了コード 1 で終了します。

パス名を 1 つだけ指定した場合は次のように動作します。

- -s オプションが指定されていないときは行の出力だけが行われます。
- -s オプションが指定されているときはファイル内のすべての行を結合して出力します。

行の入力と出力

入力ファイル内の行は，改行文字で区切られたレコードが 1 つの行と見なされます。

- Windows の場合，［CR］＋［LF］または［LF］が改行文字と見なされます。
- UNIX の場合，［LF］が改行文字と見なされます。
  なお，入力ファイルの各レコードが［CR］＋［LF］で区切られている場合は，連結後の行には［CR］が含まれます。

連結後に出力される行の終わりには改行文字が出力されます。出力される改行文字は次のとおりです。

- Windows の場合，［CR］＋［LF］で出力されます。
- UNIX の場合，［LF］で出力されます。

行単位の連結（-s オプションが指定されていない場合）

各ファイルの同じ行番号の行を区切り文字によって結合し，結合した結果を 1 つの行として出力します。なお，同じ行番号の行が空行の場合，行の内容は空文字列としてほかのファイルの行と結合されます。

各ファイルから同じ行番号の行を入力するときに，一部のファイルの入力でファイルの終端に達した場合，そのファイルの行の内容は空文字列としてほかのファイルの行と結合されます。

file1，file2，file3，file4 を行単位に連結する例を次に示します。

file1 の内容

```
a001
a002
```

file2 の内容

```
b001
（空行）
b003
```

file3 の内容

```
c001
c002
c003
c004
```

file4 の内容

```
d001
```

file1，file2，file3，file4 を行単位に連結するコマンド

```
$ paste file1 file2 file3 file4
```

このとき，次のように連結します。「→」は区切り文字のタブ文字を表します。

```
a001→b001→c001→d001   1.
a002→→    c002→       2.
→    b003→c003→       3.
→    →    c004→       4.
```

1. file1，file2，file3，file4 の 1 行目の内容を結合して出力します。行と行の間にはタブ文字を挿入します。
2. 次の値をタブ文字によって結合し出力します。
   - file1 の 2 行目の内容
   - file2 の 2 行目は空行のため空文字列
   - file3 の 2 行目の内容
   - file4 はファイル終端のため空文字列

   空文字列は長さ 0 の文字列であるため，実際には「file1 の 2 行目の内容＋タブ文字＋タブ文字＋ file3 の 2 行目の内容＋タブ文字」が出力されます。
3. 次の値をタブ文字によって結合し出力します。
   - file1 はファイル終端のため空文字列
   - file2 の 3 行目の内容
   - file3 の 3 行目の内容
   - file4 はファイル終端のため空文字列

   実際には「タブ文字＋ file2 の 3 行目＋タブ文字＋ file3 の 3 行目＋タブ文字」が出力されます。
4. 次の値をタブ文字によって結合し出力します。
   - file1 はファイル終端のため空文字列
   - file2 はファイル終端のため空文字列
   - file3 の 4 行目の内容
   - file4 はファイル終端のため空文字列

   実際には「タブ文字＋タブ文字＋ file3 の 4 行目＋タブ文字」が出力されます。

なお，一部のファイルが空のファイルの場合もそのファイルの行の内容は空文字列としてほかのファイルの行と結合されます。ただし，引数に指定したファイルすべてが空のファイルの場合は行は出力されません。

ファイル内の行の結合（-s オプションが指定されている場合）

1 つのファイル内のすべての行を区切り文字によって結合し，1 つの行にしてからほかのファイルと連結します。なお，ファイルが空の場合は改行文字だけを出力します。引数に指定したファイルすべてが空のファイルの場合もファイルごとに改行文字が出力されます。

file1，file2，file3，file4 を連結する例を次に示します。

file1 の内容

```
a001
a002
```

file2 の内容

```
空のファイル
```

file3 の内容

```
c001
c002
c003
c004
```

file4 の内容

```
d001
```

file1，file2，file3，file4 を連結するコマンド

```
$ paste -s file1 file2 file3 file4
```

このとき，次のように連結します。「→」は区切り文字のタブ文字を表します。

```
a001→a002                1.
                         2.
c001→c002→c003→c004      3.
d001                     4.
```

1. file1 のすべての行を区切り文字で結合して出力します。
2. file2 は空のファイルのため改行文字だけを出力します。
3. file3 のすべての行を区切り文字で結合して出力します。
4. file4 は 1 行だけのため，行の内容だけ出力します。

標準入力から入力した行の結合

標準入力から入力した行の結合について説明します。

行単位の結合（-s オプションが指定されていない場合）

標準入力から 1 行だけ入力し，ほかのファイルの行と結合します。「-」を複数指定した場合は「-」を指定した順に標準入力から 1 行ずつ入力し，それぞれ入力した行を結合します。

ファイルと標準入力の内容を連結する場合，標準入力からの入力は連結するファイルの終端に達するまで行われます。このため，標準入力から行を入力するときに EOF が入力されると，空文字列としてファイルの行と結合されます。

file1 を標準入力から入力し file2 と行単位に連結する例を次に示します。

file1 の内容

```
a001
a002
a003
a004
a005
```

file2 の内容

```
b001
b002
b003
b004
```

file1，file2 を行単位に連結するコマンド

```
$ cat file1 | paste - file2 -
```

行は次の順に結合されます。

1. file1 から入力した行（標準入力から入力）
2. file2 から入力した行
3. file1 から入力した行（標準入力から入力）

このとき，次のように連結します。「→」は区切り文字のタブ文字を表します。

```
a001→b001→a002     1.
a003→b002→a004     2.
a005→b003→         3.
→    b004→         4.
```

1. 次の値をタブ文字によって結合し出力します。
   - 標準入力から入力した行の内容（file1 の 1 行目の内容）
   - file2 の 1 行目の内容
   - 標準入力から入力した行の内容（file1 の 2 行目の内容）
2. 次の値をタブ文字によって結合し出力します。
   - 標準入力から入力した行の内容（file1 の 3 行目の内容）
   - file2 の 2 行目の内容
   - 標準入力から入力した行の内容（file1 の 4 行目の内容）
3. 次の値をタブ文字によって結合し出力します。
   - 標準入力から入力した行の内容（file1 の 5 行目の内容）
   - file2 の 3 行目の内容
   - 標準入力からは EOF を入力した（標準入力に出力するファイルが終端に到達している）ため空文字列
4. 次の値をタブ文字によって結合し出力します。

・標準入力からは EOF を入力した（標準入力に出力するファイルが終端に到達している）ため空文字列

・file2 の 4 行目の内容

・標準入力からは EOF を入力した（標準入力に出力するファイルが終端に到達している）ため空文字列

ファイル内の行の結合（-s オプションが指定されている場合）

標準入力から EOF を入力するまで行の入力を繰り返し，標準入力から入力したすべての行を区切り文字で結合します。結合した行をほかのファイルと連結します。file1 を標準入力から入力し file2 と連結する例を次に示します。

file1 の内容

```
a001
a002
a003
```

file2 の内容

```
b001
b002
```

file1，file2 を連結するコマンド（file1 を標準入力から入力）

```
$ cat file1 | paste -s - file2
```

このとき，次のように連結します。「→」は区切り文字のタブ文字を表します。

```
a001→a002→a003    1.
b001→b002         2.
```

1. 標準入力から入力したすべての行（file1 のすべての行）を区切り文字で結合して出力します。
2. file2 から入力したすべての行を区切り文字で結合して出力します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了 • 引数に指定したファイルのうち，オープンに失敗したファイルがありました。 -s オプションが指定されていない場合は，ファイルは連結されません。 -s オプションが指定されている場合は，ファイルのオープン失敗時にエラーメッセージを出力し，次に指定されているファイルを処理します。
2	エラー終了 • 不正なオプションを指定しました。
3	エラー終了 • メモリ不足が発生したなどの処理を続行できないエラーが発生しました。

注意事項

- paste コマンドはテキストファイルを対象としています。バイナリファイルからの入力やバイナリデータの出力は，動作を保証しません。
- -s オプションを指定しない場合，引数に指定した複数のファイルを同時にオープンします。UNIX の場合，OS 全体で同時にオープンできるファイルの最大値，または ulimit でそのプロセスのファイルディスクリプタ数の最大値制限されているなどの OS の設定値によっては，ファイルをオープンするときにエラーが発生するおそれがあります。

使用例

- 複数のファイルを行単位に連結します。結合する行と行の間に挿入する区切り文字はタブ文字とします。出力結果の「→」はタブ文字を表します。

  入力ファイル（file01）の内容

  ```
  a001
  a002
  a003
  ```

  入力ファイル（file02）の内容

  ```
  b001
  b002
  b003
  ```

  入力ファイル（file03）の内容

  ```
  c001
  c002
  c003
  ```

  このときのコマンド指定と実行結果を次に示します。

  ```
  $ paste file01 file02 file03
  a001→b001→c001
  a002→b002→c002
  a003→b003→c003
  ```

- 複数のファイルを行単位に連結します。結合する行と行の間には区切り文字「=」と「%」を順に挿入します。

  入力ファイル（file01）の内容

  ```
  a001
  a002
  a003
  ```

  入力ファイル（file02）の内容

  ```
  b001
  b002
  b003
  ```

入力ファイル（file03）の内容

```
c001
c002
c003
```

入力ファイル（file04）の内容

```
d001
d002
d003
```

このときのコマンド指定と実行結果を次に示します。

```
$ paste -d ”=%” file01 file02 file03 file04
a001=b001%c001=d001
a002=b002%c002=d002
a003=b003%c003=d003
```

- 複数のファイルを行単位に連結します。結合する行と行の間には区切り文字「=」,「%」および「@」を順に挿入します。

入力ファイル（file01）の内容

```
a001
a002
a003
```

入力ファイル（file02）の内容

```
b001
b002
b003
```

入力ファイル（file03）の内容

```
c001
c002
c003
```

このときのコマンド指定と実行結果を次に示します。

```
$ paste -d ”=%@” file01 file02 file03
a001=b001%c001 ※
a002=b002%c002
a003=b003%c003
```

注※

同じ行番号に対する行の連結回数は次のように 2 回です。

・file01 の 1 行と file02 の 1 行を連結

・file01 と file02 の行の連結結果に対して file03 の 1 行を連結

このため，-d オプションに指定した区切り文字「@」は使用されません。

- ls コマンドで表示されるファイル名一覧を標準入力から入力し 4 列で出力します。ファイル名とファイル名の間には区切り文字「,」を挿入します。

```
$ ls
a001  a002  a003  a004  b001  b002  b003  b004  c001  c002
$ ls | paste -d ”,” - - - -
a001,a002,a003,a004
b001,b002,b003,b004
c001,c002,,
```

- ファイル単位にファイル内の全行を結合して 1 行としたあと，各ファイルを連結します。結合する行と行の間には区切り文字「=」,「%」および「@」を順に挿入します。

  入力ファイル（file01）の内容

```
a001
a002
a003
```

  入力ファイル（file02）の内容

```
b001
b002
b003
b004
```

  入力ファイル（file03）の内容

```
c001
c002
c003
c004
c005
```

  入力ファイル（file04）の内容

```
d001
d002
```

  このときのコマンド指定と実行結果を次に示します。

```
$ paste -s -d ”=%@” file01 file02 file03 file04
a001=a002%a003
b001=b002%b003@b004
c001=c002%c003@c004=c005
d001=d002
```

- 複数のファイルを行単位に連結する場合に，存在しないファイルを指定したときのメッセージを次に示します。

```
$ paste file01 file02 file03
paste: file02: No such file or directory
```

- ファイル内の全行を結合し，複数のファイルを連結します。入力ファイルとして存在しないファイル file01 と file03 が指定されています。

  入力ファイル（file02）の内容

```
b001
b002
```

```
b003
b004
```

入力ファイル（file04）の内容

```
d001
d002
```

このときのコマンド指定と実行結果を次に示します。

```
$ paste -s -d ”=%@” file01 file02 file03 file04
paste: file01: No such file or directory※
b001=b002%b003@b004
paste: file03: No such file or directory※
d001=d002
```

注※

標準エラー出力に出力される内容

8.4.25 printf コマンド（書式の引数を書式に従って変換し，標準出力に出力する）

形式

printf *書式* [*書式の引数* ...]

機能

*書式の引数*を*書式*に従って変換し，標準出力に出力します。

引数

書式

*書式*は，次の 3 種類の文字列で構成されます。

- 標準出力に出力する文字
- 変換指定
- エスケープ文字

変換指定は変換指定を示す%の後ろに，フラグ文字，最小フィールド幅，精度，変換指定子の順で指定します。フラグ文字，最小フィールド幅，精度の指定は省略できます。

- 変換指定

%	変換指定を示します。

- フラグ文字列

  変換指定を示す%の後ろに次のフラグ文字を指定できます。この項目は指定を省略できます。

フラグ文字	意味
-	変換の結果はフィールド内で左そろえになります。 このフラグ文字を指定していない場合，右そろえになります。
+	符号付き変換の結果に符号+，または-が付きます。 このフラグ文字を指定していない場合，変換の結果が負の値の時だけ-が付きます。
スペース	符号付き変換の結果で，正の数値の前にスペースが付きます。 +フラグ文字と同時に指定した場合は，+フラグ文字が優先されます。
#	以下の変換指定子を指定した時に，変換の結果を別の形式で出力します。 • o 変換の場合，変換結果の先頭に0 が付きます(変換の結果の0 は除きます)。 • x，X 変換の場合，変換結果の先頭に0x または0X が付きます(変換の結果の0 は除きます)。 • e，E，f，g，G 変換の場合，小数点を含まない値を指定しても，必ず小数点を付けて出力します。 • g とG 変換の場合，末尾の0 は変換結果から削除されません。
0	d，i，o，u，x，X，e，E，f，g およびG 変換の場合，変換結果のフィールド内の左側のスペースを 0 で埋めます。 -フラグ文字，もしくは変換指定子d，i，o，u，x，X の精度と共に指定した場合は，0 フラグ文字は無視されます。

- 最小フィールド幅

  最小フィールド幅を 10 進数で指定します。最小フィールド幅は0～2147483647 の範囲で指定できます。この項目は指定を省略できます。変換された値の文字数がフィールドよりも少ない場合，フィールドの左側をスペースで埋めます。ただし，左そろえのフラグ文字が指定されている場合は右側をスペースで埋めます。変換結果がフィールド幅よりも広い場合，フィールドは変換結果が入る幅に広げられます。アスタリスク(*)を指定した場合，*書式の引数*に指定した値を最小フィールド幅として使用します。

- 精度

  精度は，ピリオド(.)とそれに続く 10 進数で指定します。UNIX の場合は0～2147483647 の範囲で，Windows の場合は0～512 の範囲で指定できます。この項目は指定を省略できます。ピリオド(.)だけを指定した場合は0 が指定されたと仮定します。d，i，o，u，x またはX 変換では，表示する最小の桁数を指定します。e，E およびf 変換では表示する小数点以下の桁数を指定します。g，G 変換では最大有効桁数を指定します。s 変換では表示する文字列の最大バイト数を指定します。値の代わりにアスタリスク(*)を指定した場合，*書式の引数*に指定した値を使用します。

- 変換指定子

変換指定子	意味
d，i	符号付き 10 進数表記に変換します。 精度は表示する最小桁数を指定します。変換後の値が精度で指定した桁数に満たない場合は，値の先頭に0 が付きます。デフォルトの精度は1 です。 精度に0 を指定し，値0 を変換した場合，空文字を出力します。
o	符号なし 8 進数表記に変換します。

変換指定子	意味
o	精度は表示する最小桁数を指定します。変換後の値が精度で指定した桁数に満たない場合は，値の先頭に0 が付きます。デフォルトの精度は1 です。 精度に0 を指定し，値0 を変換した場合，空文字を出力します。
u	符号なし 10 進数表記に変換します。 精度は表示する最小桁数を指定します。変換後の値が精度で指定した桁数に満たない場合は，値の先頭に0 が付きます。デフォルトの精度は1 です。 精度に0 を指定し，値0 を変換した場合，空文字を出力します。
x，X	16 進数表記に変換します。x を指定した場合は小文字(abcdef)で表記し，X を指定した場合は大文字(ABCDEF)で表記します。 精度は表示する最小桁数を指定します。変換後の値が精度で指定した桁数に満たない場合は， 値の先頭に0 が付きます。デフォルトの精度は1 です。 精度に0 を指定し，値0 を変換した場合，空文字を出力します。
e，E	浮動小数点数を[-]d.ddddde± dd[d]の 10 進数スタイルに変換します。 e を指定した場合は小文字で表記し，E を指定した場合は大文字で表記します。 小数点の前は 1 桁です。また，小数点以下の桁数は，精度を指定していない場合は 6 桁，精度を指定した場合は指定した桁数です。また，精度に0 を指定した場合，小数点以下は出力されません。
f	浮動小数点数を[-]dddd.dddd の 10 進数スタイルに変換します。 小数点の前は 1 桁以上表示されます。また，小数点以下の桁数は，精度を指定していない場合は 6 桁，精度を指定した場合は指定した桁数です。
g，G	浮動小数点数をスタイルf またはe，E の形式で表示します。 表示するスタイルは変換される値によって異なります。 表示する桁数は，精度を指定していない場合は 6 桁，精度を指定した場合は指定した桁数となります。末尾の0 は変換結果から削除されます。
c	*書式の引数*の最初の 1 バイトを表示します。
s	*書式の引数*を文字列と解釈して表示します。 精度を指定していない場合は文字列すべてを表示し，精度を指定した場合は指定した値のバイト数分を表示します。 ただし，OS の仕様で，精度を指定して出力する区切りがマルチバイト文字の途中の場合，出力するバイト数は指定値より少なくなることがあります。
%	%を表示します。 フラグ文字，最小フィールド幅，精度の指定はできません。
b	*書式の引数*を文字列と解釈して表示します。 文字列に含まれているエスケープ文字も文字列として解釈し，変換表示します。 ただし，*書式の引数*の文字列に¥c が現れた場合は，それ以降の変換は表示されません。 フラグ文字，最小フィールド幅，精度の指定はできません。

- エスケープ文字

  使用できるエスケープ文字を次の表に示します。

エスケープ文字	意味
¥a	アラート文字(ベル)
¥b	バックスペース文字
¥f	フォームフィード文字(改ページ)
¥n	改行文字
¥r	復帰文字
¥t	タブ文字
¥v	垂直タブ文字
¥d，¥dd，¥ddd	1～3 桁の 8 進数で表された ASCII コードの文字(0～7)
¥xhex	1～2 桁の 16 進数で表された ASCII コードの文字(0～9，a～f，A～F)
¥¥	¥の表示
¥’	’の表示
¥”	”の表示

上記以外で文字列に¥が含まれる場合は，¥も出力されます。

書式の引数

- *書式*に指定した変換指定の数より，*書式の引数*に指定した数が多い場合は，*書式*を繰り返し使用します。余分な*書式*は0 またはNULL で評価されます。

  例

```
$ printf ”%x %d ” 123 456 789
7b 456 315 0
```

- 数値変換の場合，*書式の引数*で文字の前にシングルクォーテーション(’)，またはダブルクォーテーション(”)を付加すると ASCII コードで出力されます。

  例

```
$ printf ”%x %x” ¥’a ¥”b
61 62
```

- 変換指定子d，i，o，u，x，X の*書式の引数*に数値を指定する場合，8 進数(0 指定)，10 進数，16 進数(0x 指定)が指定できます。

  例

```
$ printf ”%d %d %d” 010 10 0x10
8 10 16
```

終了コード

終了コード	意味
0	正常終了。
1	エラー終了。

注意事項

- 改行文字は，Windows の場合は［CR］＋［LF］，UNIX の場合は［LF］で出力されます。
- printf コマンドは，変換指定子d，i，o，u，x，X を指定して出力および変換する場合，4 バイトの整数で扱います。変換指定子e，E，f，g，G を指定して出力および変換する場合は倍精度浮動小数点数（8 バイト）として扱います。このため，変換結果に誤差が発生します。この誤差は OS に依存します。

使用例

- フラグ文字に#0，最小フィールド幅に10 を指定して，16 数変換した内容を出力します。

```
$ ./printf "%#010x¥n" 123
0x0000007b
```

- フラグ文字に#+を指定して 10 進数変換した内容を出力します。

```
$ ./printf "%#+d¥n" 123
+123
```

- 精度に8 を指定して浮動小数点を f 形式に変換した内容を出力します。

```
$ ./printf "%.8f¥n" 123.456
123.45600000
```

- 指定した浮動小数点を e 形式に変換した内容を出力します。

```
$ ./printf "%e¥n" 123.456
1.234560e+02
```

- 文字列abcdef のうち，1 バイト分を出力します。

```
$ ./printf "%c¥n" abcdef
a
```

- 精度に3 を指定し，文字列abcdef を出力します。

```
$ ./printf "%.3s¥n" abcdef
abc
```

- *書式*にエスケープ文字(¥t)を指定して，*書式の引数*に指定された内容を出力します。

```
$ ./printf "%s %d¥txyz¥n" abc 123
abc 123 xyz
```

- フラグ文字に0，精度に*(アスタリスク)を指定して出力します。

この場合，*(アスタリスク)は5 に置き換えられます。

```
$ ./printf "%0*d" 5 123
00123
```

8.4.26　rm コマンド（ファイルまたはディレクトリを削除する）

形式

```
rm [-d] [-f] [-i] [-R] [-r] パス名 ...
```

機能

ファイルまたはディレクトリを削除します。

引数

-d

ファイルまたはディレクトリを削除します。ディレクトリの場合，ディレクトリごと削除します。

-f

ファイルを削除します。存在しないファイルは無視されます。削除するかどうかは問い合わせません。-i オプションの前に指定すると無視されます。

-i

ファイルを削除する前に確認します。標準入力から y または Y を応答すると削除します。-f オプションの前に指定すると無視されます。

-R|-r

再帰的にディレクトリツリーを削除します。

パス名

削除するパス名を指定します。複数指定できます。

終了コード

終了コード	意味
0	正常終了 • 指定したファイルやディレクトリの削除に成功しました。 • -f オプションを指定した場合は，指定したファイルのうち，存在するファイルの削除に成功しました。
1 以上	エラー終了

注意事項

- Windows の場合，write 権限がなく削除を確認するときは，オーナーのアクセス権以外は表示しません。表示するパーミッションの詳細については，「8.4.21　ls コマンド（ファイルまたはディレクトリの内容を表示する）」を参照してください。
- -f オプションおよび-i オプションは最後に指定したオプションが有効となります。
- rm コマンドの引数には削除する権限が割り当たっているファイル，ディレクトリを指定してください。削除する権限が割り当たっていないファイル，ディレクトリを指定すると削除に失敗する場合があります。【Windows 版】

使用例

- -i オプションを指定して，ファイルを削除するかどうか確認します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rm -i file2.txt
remove file2.txt?
```

- ファイルがない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rm c.txt
rm: c.txt: No such file or directory
```

- -d オプションを使用しないでディレクトリを削除しようとした場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rm dir8
rm: dir8: is a directory
```

- 削除しようとしたディレクトリにファイルが存在した場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rm -d dir8
rm: dir8: Directory not empty
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rm -w
rm: illegal option -- w
usage: rm [-dfiRr] file ...
```

8.4.27　rmdir コマンド（空のディレクトリを削除する）

形式

```
rmdir ディレクトリ名 ...
```

機能

空のディレクトリを削除します。

引数

ディレクトリ名

削除するディレクトリを指定します。

終了コード

終了コード	意味
0	正常終了 • ディレクトリの削除に成功しました。
1 以上	エラー終了

注意事項

- このコマンドには指定可能なオプションが存在しません。そのため，引数にオプションを指定した場合，そのオプションをディレクトリとして解釈します。

使用例

- D:¥temp¥dir1 の dir1 ディレクトリを削除します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rmdir  D:¥temp¥dir1
```

- ディレクトリが空ではない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rmdir dir8
rmdir: dir8: Directory not empty
```

- 削除するディレクトリが指定されていない場合のエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rmdir
usage: rmdir directory ...
```

8.4.28　sed コマンド（テキスト中の文字列を置換する）

形式

```
sed [-a] [-E] [-n] [-r] [-u] コマンド [入力ファイルパス名...]
sed [-a] [-E] [-n] [-r] [-u] [-e コマンド] ... [-f スクリプトファイルパス名] ... [入力ファイルパス名...]
```

機能

ファイルや標準入力のテキストの文字列を置換して標準出力に出力します。

引数

-a

編集コマンドの解析エラーが発生した場合に，ファイルが作成されないようにしたり，既存ファイルが空にならないようにしたりするときに指定します。w コマンドまたは s コマンドの w フラグで指定するパターンスペース出力ファイルの作成方法を定義します。パターンスペース出力ファイルは，w コマンド，または w フラグ指定の s コマンドが適用されるときに作成されます。-a オプションを指定しない場合は，w コマンド，または s コマンドの w フラグ解析時にパターンスペース出力ファイルを作成します。

-E|-r

コマンドで指定するパターンを，拡張した正規表現として扱います。どちらのオプションを指定しても同じように実行されます。

-n

標準出力へのパターンスペースの出力を抑止します。p コマンドまたは P コマンドによる出力以外はパターンスペースの標準出力への出力は行いません。

-u

Windows の場合，標準出力への実行結果出力時のバッファリングを抑止します。

UNIX の場合，標準出力への実行結果出力時のバッファリングをレコード単位にします。

コマンド|-e コマンド

入力ファイルを編集するコマンドを指定します。-e オプションは複数指定できます。-e オプションを複数指定する場合，コマンドの実行順序は指定された順番です。-f オプションを指定しない場合は，-e オプションの指定を省略できます。

-f スクリプトファイルパス名

スクリプトファイルのパス名を指定します。スクリプトファイルには，入力ファイルのレコードを編集する編集コマンドを記述します。-f オプションは複数指定できます。-f オプションを複数指定する場合，または-e オプションと組み合わせて指定する場合，コマンドの実行順序は指定された順番です。

入力ファイルパス名

編集する入力ファイルのパス名を指定します。複数指定できます。パス名を指定しない場合は，標準入力から入力します。ファイルを複数指定した場合，処理中のファイルが終端に到達したときに次のファイルをオープンしてレコードを入力します。

編集コマンドの記述形式

入力ファイルを編集する編集コマンドの記述形式を次に示します。

```
[address [,address] ] command [arguments]
```

address には，編集対象のレコードを特定するためのアドレス（レコードの行番号，または検索パターン文字列）を指定します。

command には，入力レコードに適用する編集コマンドを指定します。

arguments には，編集コマンドに渡す引数を指定します。

入力ファイルから 1 レコードを入力するたびにアドレスで指定したレコードの行番号，または検索パターン文字列と比較し，一致した場合に編集コマンドが実行されます。アドレスを省略した場合はすべてのレコードが編集の対象となります。編集コマンドの実行結果は標準出力に出力されます。

アドレス

入力ファイル内の編集対象となるレコードを特定するためのアドレスです。

- 行番号

  入力ファイルの先頭レコードを 1 とした行番号を指定します。最終レコードは「$」を使うこともできます。複数の入力ファイルを指定した場合は通し番号になります。なお，行番号（アドレス範囲指定時は開始行番号）に 0 を指定した場合，編集コマンドは入力レコードに適用されません。

- 検索パターン文字列

  レコード内の文字列に一致させる検索パターン文字列を「/」で囲んで指定します。検索パターン文字列には正規表現が指定できます。指定例を次に示します。

  ```
  /abc/w file
  ```

  検索パターン文字列を囲む「/」は，「¥」と改行文字以外のすべての 1 バイト文字のどれかに変更できます。「/」以外の文字を区切り文字として使用する場合は，先頭の区切り文字の前に「¥」を記述します。

  指定例を次に示します。検索パターン文字列を囲む「/」を「#」に変更して指定します。

  ```
  ¥#abc#w file
  ```

- アドレス範囲指定

  「address，address」を指定することで編集対象となるレコードの範囲が指定できます。1 番目の address に一致するレコードから，2 番目の address に一致するレコードまでが編集コマンドの実行範囲となります。

  範囲は，次のように指定できます。

  - 2 つの行番号による範囲
  - 2 つの検索パターン文字列による範囲
  - 行番号と検索パターン文字列の組み合わせによる範囲

  2 つの行番号による範囲の指定例を次に示します。

  ```
  5,20w outfile
  ```

  1 番目の address で指定した行番号のレコードから，2 番目の address に指定した行番号のレコードまでが編集コマンドの実行範囲となります。

なお，1 番目の address で指定した行番号が 2 番目の address で指定した行番号より大きい場合（1 番目の address ＞ 2 番目の address），1 番目の address で指定した行番号のレコードだけが編集コマンドの実行対象となります。

2 つの検索パターン文字列による範囲の指定例を次に示します。

```
/abc/,/xyz/w file
```

1 番目の address の検索パターン文字列に一致するレコードから，2 番目の address の検索パターン文字列に一致するレコードまでが編集コマンドの実行範囲となります。

なお，2 番目の address で指定した検索パターン文字列を含むレコードがなく入力ファイルの終端に達した場合は，入力ファイルの最終行までが範囲となります。ただし，複数の入力ファイルを指定した場合，次の入力ファイルで 2 番目の address で指定した検索パターン文字列に一致するレコードを検索します。

1 番目の address が検索パターン文字列で 2 番目の address が行番号の場合，検索パターンに一致したレコードの行番号が 2 番目の address の行番号より大きい場合（1 番目の address ＞ 2 番目の address），検索パターンに一致したレコードだけが編集コマンドの実行対象となります。

パターンスペースとホールドスペース

sed コマンドにはパターンスペース，ホールドスペースと呼ばれるテキスト編集用の作業領域があります。

パターンスペースには入力ファイルから入力したレコードが格納されます。

パターンスペースは，次に示す sed コマンドの処理の流れで使用されます。

1. 入力ファイルから改行コードで分割された 1 レコードを入力します。

   Windows では，［CR］＋［LF］または［LF］です。UNIX では，［LF］だけです。UNIX の場合，入力ファイルの改行コードが［CR］＋［LF］のときは，パターンスペースに［CR］が格納されます。
2. 入力レコードの内容をパターンスペースにコピーします。
3. パターンスペースがアドレスで指定した行番号，またはパターンスペースの内容に検索パターン文字列が一致する場合に編集コマンドを実行します。

   実行するコマンドが D コマンドの場合に，D コマンド実行後にパターンスペースの内容が残っているときは，手順 1 と手順 2 は処理されません。
4. パターンスペースの内容を標準出力に出力します。

   ただし，-n オプションが指定されている場合は処理されません。
5. パターンスペースの内容を消去します。

ホールドスペースはパターンスペースの内容を退避したり，ホールドスペースの内容をパターンスペースに戻したりする一時的な作業領域として使用できます。

編集コマンド

sed コマンドで使用できる編集コマンドを次に示します。

[address［,address]］{command-list}

入力レコードに適用する複数の編集コマンドをグループ化します。各編集コマンドは改行または;（セミコロン）で区切ります。なお，最後に記述した編集コマンドと同じ行に「}」を記述する場合は，コマンド名の後ろに「;」を記述する必要があります。

[address]a¥（改行）

text

次の入力レコードを読み込む前に text に記述したテキストを標準出力に出力します。複数レコードを出力する場合は，改行の直前に「¥」を記述します。

指定例を次に示します。次の入力レコードを読み込む前に 2 レコードを標準出力に出力します。

```
a¥（改行）
テキスト1¥（改行）
テキスト2
```

[address[,address]]b[label]

指定したラベル label が定義されている「:label」コマンドに分岐します。label の指定を省略した場合は，スクリプト記述の末尾に分岐します。

[address[,address]]c¥（改行）

text

パターンスペースの内容を削除します。アドレス未指定または 1 個のアドレスが指定されている場合は，text に記述したテキストを標準出力に出力します。アドレス 2 個指定の場合は，選択された範囲の最終レコードを処理したあとに text に記述したテキストを標準出力に出力します。複数レコードを出力する場合は，改行の直前に「¥」を記述します。

また，c コマンドの後ろに記述されているコマンドは実行されないで，次の入力レコードを読み込んで先頭のコマンドから実行が開始されます。

[address[,address]]d

パターンスペースの内容を削除します。削除するパターンスペースの内容は標準出力には出力されません。また，d コマンドの後ろに記述されているコマンドは実行されないで，次の入力レコードを読み込んで先頭のコマンドから実行が開始されます。

[address[,address]]D

パターンスペースに複数レコードが格納されている場合に，最初の改行までを削除します。パターンスペースの内容は標準出力には出力されません。また，D コマンドの後ろに記述されているコマンドは実行されないで，先頭のコマンドから実行が開始されます。

D コマンドを実行した結果，パターンスペースの内容がなくなった場合は，次の入力レコードを読み込んで先頭のコマンドから実行が開始されます。

[address[,address]]g

ホールドスペースの内容をパターンスペースにコピーします。コピー前のパターンスペースの内容は破棄されます。

[address[,address]]G

ホールドスペースの内容をパターンスペースに追加します。追加前にパターンスペースに格納されているレコードとは改行文字で区切られます。

[address[,address]]h

パターンスペースの内容をホールドスペースにコピーします。コピー前のホールドスペースの内容は破棄されます。

[address[,address]]H

パターンスペースの内容をホールドスペースに追加します。追加前にホールドスペースに格納されているレコードとは改行文字で区切られます。

[address]i¥（改行）

text

現在の入力レコードをパターンスペースに格納する前に text に記述したテキストを標準出力に出力します。複数レコードを出力する場合は，改行の直前に「¥」を記述します。

[address[,address]]l

パターンスペースの内容を標準出力に出力します。1 バイト文字（0x20～0x7e の範囲），スペースおよびマルチバイト文字以外のデータは，各バイトごとに「¥」に続いて 3 桁の 8 進数で出力します。また，「¥」は「¥¥」として出力され，次の表に示す制御コードはエスケープ文字として出力されます。

制御コード	出力されるエスケープ文字
アラート文字（ベル）	¥a
バックスペース文字	¥b
フォームフィード文字（改ページ）	¥f
改行文字。なお，行（複数行の場合は最終行）の終端の改行文字は出力されません。	¥n
復帰文字	¥r
タブ文字	¥t
垂直タブ文字	¥v

1 レコードの出力幅は，次の優先順位で値が決まります。

1. 環境変数 COLUMNS の値
2. コンソールへの出力の場合はコンソール画面の幅
3. 半角文字で 60 文字

各レコードの終わりには$記号が出力されます。なお，1 レコードが出力幅を超える場合は折り返して出力されます。折り返し部分には「¥」が出力されます。

[address[,address]]n

入力ファイルから次の入力レコードを読み込んでパターンスペースに格納し，現在の内容を標準出力に出力します。現在の行番号には 1 が加算されます。-n オプション指定時は，パターンスペースの現在の内容は標準出力に出力されません。

[address[,address]]N

入力ファイルから次の入力レコードを読み込んでパターンスペースに追加します。追加前にパターンスペースに格納されているレコードとは改行文字で区切られます。現在の行番号には 1 が加算されます。

[address[,address]]p

パターンスペースの内容を標準出力に出力します。

[address[,address]]P

パターンスペースに複数レコードが格納されている場合に，最初の改行までの内容を出力します。パターンスペースに 1 レコードしか格納されていない場合は p コマンドの場合と同じです。

[address]q

スクリプトの処理を終了します。この記述以降はコマンドの実行および入力レコードの入力はしません。-n オプションが指定されていない場合，終了時にパターンスペースの内容を標準出力に出力します。また，a または r コマンドで追加されたレコードがあるときはそのレコードが出力されます。

[address]r **パス名**

次の入力レコードの入力前にパス名で指定したファイルの内容を標準出力に出力します。パス名で指定したファイルの入力でエラーが発生しても無視されます。

Windows の場合，ファイル中の改行コードは［CR］＋［LF］で出力されます。

UNIX の場合，ファイル中の改行コードをそのまま出力します。

[address[,address]]s/pattern/replacement/flags

パターンスペース内のパターン pattern に最初に一致する文字列を置き換え文字列 replacement に置き換えます。「s」，pattern および replacement の区切り文字である「/」を，「¥」と改行文字以外のすべての 1 バイト文字のどれかに変更できます。区切り文字を pattern，replacement の文字に含めたい場合は，pattern，replacement 内の区切り文字の前に「¥」を記述します。

パターンには正規表現が指定できます。

また，置き換え文字列 replacement には次の文字が指定できます。

- 「&」を指定した場合は，「&」がパターンに一致した文字列に置き換わります。「&」を置き換える文字として扱いたい場合は「&」の前に「¥」を記述します。
- 「¥N」(N:1～9 の数字)を指定した場合は，「¥N」がパターンの()（丸括弧）で囲まれたタグ付き正規表現に一致する文字列に置き換わります。「¥」の後ろの数字はパターン中のタグ付き正規表現文字列の順番を示します。
- 改行を含める場合は改行の直前に「¥」を記述します。

flags に指定できるフラグには次の値があります。なお，フラグは省略でき，また 1 個以上指定できます。

N

パターンスペースで N 回目に一致したパターンだけを置き換えます。

g

パターンスペースの最初にパターンに一致した文字列の置き換えだけでなく，パターンスペース内のパターンに一致したすべての文字列を置き換えます。

p

置き換えを実行した場合，置き換え後のパターンスペースの内容を標準出力に出力します。

w **パス名**

置き換えを実行した場合，置き換え後のパターンスペースの内容をパス名で指定したファイルに追加します。指定したファイルがある場合は，次のようになります。

・-a オプションを指定しないとき

置き換えの有無に関係なく sed コマンド実行前の内容は破棄されます。

・-a オプションを指定するとき

置き換えが実行されると sed コマンド実行前の内容は破棄されます。

Windows の場合，ファイルに出力される改行コードは［CR］＋［LF］で出力されます。

[address[,address]]t[label]

入力レコードが読み込まれてから，または直前に実行された t コマンド以降で s コマンドによる置き換えが行われた場合に，指定したラベル label が定義されている:（コロン）コマンドに分岐します。label の指定を省略した場合は，スクリプト記述の末尾に分岐します。

[address[,address]]w **パス名**

パターンスペースの内容をパス名で指定したファイルに追加します。指定したファイルがある場合は，次のようになります。

- -a オプションを指定しないとき

  アドレス address が一致しているかどうかに関係なく sed コマンド実行前の内容は破棄されます。

- -a オプションを指定するとき

  アドレス address に一致すると sed コマンド実行前の内容は破棄されます。

Windows の場合，ファイルに出力される改行コードは［CR］＋［LF］で出力されます。

[address[,address]]x

パターンスペースの内容とホールドスペースの内容を交換します。

[address[,address]]y/string1/string2/

パターンスペースの内容に対して，文字列 string1 に指定した文字ごとに検索と置き換えを行います。置き換える文字は，文字列 string1 の各文字に対応する位置にある文字列 string2 中の文字で置き換えます。

string1 と string2 の文字数は同じにする必要があります。

string1 または string2 に改行文字を指定する場合は¥n を指定します。「y」，string1 および string2 の区切り文字である「/」を，「¥」と改行文字以外のすべての 1 バイト文字のどれかに変更できます。

[address[,address]]!command または[address[,address]]!{command-list}

コマンドまたは，グループ化したコマンド群をアドレス address で選択されないレコードに適用します。

:label

b および t コマンドに指定した分岐先のラベルを定義します。:（コロン）コマンド自体は処理をしません。

[address]=

現在の行番号を 1 レコードとして標準出力に出力します。

(空行)

空行は無視されます。

#

#以降はコメントとして扱われます。なお，スクリプトファイルの先頭レコードの 1 カラム目に「#n」を記述した場合は，-n オプションを指定した場合の動作となります。

エスケープ文字

アドレスの検索パターン，a コマンドのテキスト，c コマンドのテキスト，i コマンドのテキスト，s コマンドのパターンと置き換え文字列，および y コマンドの検索文字と置き換え文字には次のエスケープ文字を使用できます。

エスケープ文字	意味
¥a	アラート文字（ベル）
¥b	バックスペース文字※1
¥f	フォームフィード文字（改ページ）
¥n	改行文字※2
¥r	復帰文字
¥t	タブ文字
¥v	垂直タブ文字
¥xhex	1～2 桁の 16 進値で表された文字（0～9，a～f，A～F）※3
¥c	任意のリテラル文字(「¥"」なら「"」)
¥¥	1 つのバックスラッシュ文字

注※1

アドレスの検索パターン，s コマンドのパターンに指定した場合，正規表現演算子の¥b として扱われます。なお，[ ]で囲んだ文字集合内に指定した場合はバックスペース文字として扱われます。

注※2

Windows の場合，a コマンドのテキスト，c コマンドのテキストおよび i コマンドのテキストで指定すると，出力時に［CR］＋［LF］で出力されます。

注※3

パターンに指定する場合，実行時の文字コード種別によっては指定できない値があります。文字コード種別ごとに指定できる値を 16 進数値で示します。なお，次の値以外を指定した場合はエラー終了します。

・文字コード種別：シフト JIS

0x01-0x80,0xA0-0xDF,0xFD-0xFF

・文字コード種別：UTF-8

0x01-0xBF,0xFE-0xFF

・文字コード種別：EUC

0x01-0x8D,0x90-0xA0,0xFF

・文字コード種別：C

0x01-0xFF

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

- ファイルの 1 レコード目から 3 レコード目を削除する d コマンドを指定します。入力ファイルは，file01.txt です。

  file01.txt の内容

  ```
  hitachi group01 Tokyo
  HITACHI group02 Yokohama
  hitachi group03 Fukuoka
  HITACHI group04 Hokkaido
  HITACHI group05 Ooita
  HITACHI group06 Hiroshima
  ```

  コマンドの実行例を次に示します。

  ```
  C:¥DIR>%ADSH_OSCMD_DIR%¥sed ”1,3d” file01.txt
  HITACHI group04 Hokkaido
  HITACHI group05 Ooita
  HITACHI group06 Hiroshima
  ```

- 検索パターンに一致したレコードに対して，i コマンドで一致したレコードの前に 2 レコード追加し，a コマンドで一致したレコードの前に 1 レコード追加します。検索パターンに一致しないレコードは c コマンドで別のレコードに置き換えます。スクリプトファイルは，scpt01.sed です。入力ファイルは，file02.txt です。

  scpt01.sed の内容

```
/file/{
i¥
<FILE-LINE>¥
 [FILE-BEGIN]
a¥
 [FILE-END]
}
/file/!{
c¥
<DIR-LINE>
}
```

file02.txt の内容

```
The file path used by trace is invalid.
Don't know current directory.
Input asc file is the same as output asc file.
Cannot change directory.
Merging two asc files is started.
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -f scpt01.sed file02.txt
<FILE-LINE>
 [FILE-BEGIN]
The file path used by trace is invalid.
 [FILE-END]
<DIR-LINE>
<FILE-LINE>
 [FILE-BEGIN]
Input asc file is the same as output asc file.
 [FILE-END]
<DIR-LINE>
<FILE-LINE>
 [FILE-BEGIN]
Merging two asc files is started.
 [FILE-END]
```

- パターンに最初に一致する文字列を置き換えます。入力ファイルは，file03.txt です。

  file03.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
日立 沖縄支店 日立グループ
日立 福岡支店 日立グループ
```

  コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "s/日立/&製作所/" file03.txt
日立製作所 横浜支店 日立グループ
日立製作所 東京支店 日立グループ
日立製作所 沖縄支店 日立グループ
日立製作所 福岡支店 日立グループ
```

- パターンに一致する文字列を整形して置き換えます。入力ファイルは，file04.txt です。

  file04.txt の内容

```
日立 横浜支店    日立グループ
日立 東京支店    日立グループ
日立 沖縄支店    日立グループ
日立 北海道支店 日立グループ
日立 福岡支店    日立グループ
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "s/¥(日立 ¥)¥(.*¥)¥(支店¥)/¥1¥3名:¥2/" file04.txt
日立 支店名:横浜    日立グループ
日立 支店名:東京    日立グループ
日立 支店名:沖縄    日立グループ
日立 支店名:北海道 日立グループ
日立 支店名:福岡    日立グループ
```

- パターンに 2 回目に一致する文字列を置き換えます。入力ファイルは，file05.txt です。

  file05.txt の内容

```
日立 横浜支店 日立グループ 日立製作所
日立 東京支店 日立グループ 日立製作所
日立 沖縄支店 日立グループ 日立製作所
日立 福岡支店 日立グループ 日立製作所
```

  コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "s/日立/&製作所/2" file05.txt
日立 横浜支店 日立製作所グループ 日立製作所
日立 東京支店 日立製作所グループ 日立製作所
日立 沖縄支店 日立製作所グループ 日立製作所
日立 福岡支店 日立製作所グループ 日立製作所
```

- 特定範囲のレコードのパターンに一致するすべての文字列を置き換えます。入力ファイルは，file06.txt です。

  file06.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
日立 沖縄支店 日立グループ
日立 福岡支店 日立グループ
```

  コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "/東京/,/沖縄/s/日立/&製作所/g" file06.txt
日立 横浜支店 日立グループ
日立製作所 東京支店 日立製作所グループ
日立製作所 沖縄支店 日立製作所グループ
日立 福岡支店 日立グループ
```

- s コマンドのフラグに p を指定し，文字列を置き換えたレコードを標準出力に出力します。入力ファイルは，file07.txt です。

  file07.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
```

```
日立 沖縄支店 日立グループ
日立 福岡支店 日立グループ
```

コマンドの実行例を次に示します。

-n オプションを指定した場合

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -n "/東京/,/沖縄/s/日立/&製作所/gp" file07.txt
日立製作所 東京支店 日立製作所グループ
日立製作所 沖縄支店 日立製作所グループ
```

-n オプションを指定しない場合

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "/東京/,/沖縄/s/日立/&製作所/gp" file07.txt
日立 横浜支店 日立グループ
日立製作所 東京支店 日立製作所グループ
日立製作所 東京支店 日立製作所グループ
日立製作所 沖縄支店 日立製作所グループ
日立製作所 沖縄支店 日立製作所グループ
日立 福岡支店 日立グループ
```

- s コマンドのフラグに w を指定し，文字列を置き換えたレコードをファイルに出力します。入力ファイルは，file08.txt です。

  file08.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
日立 沖縄支店 日立グループ
日立 福岡支店 日立グループ
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -n "/東京/,/沖縄/s/日立/&製作所/gw dir¥¥out.txt" file08.txt
C:¥DIR>%ADSH_OSCMD_DIR%¥cat dir¥out.txt
日立製作所 東京支店 日立製作所グループ
日立製作所 沖縄支店 日立製作所グループ
```

- 特定範囲のレコード以外のレコードをファイルに出力します。入力ファイルは，file09.txt です。

  file09.txt の内容

```
日立 横浜支店 日立グループ
日立 東京支店 日立グループ
日立 北海道支店 日立グループ
日立 沖縄支店 日立グループ
日立 福岡支店 日立グループ
```

コマンドの実行例を次に示します。

```
C:¥DIR>sed -n "/東京/,/沖縄/!w dir¥¥out.txt" file09.txt
C:¥DIR>cat dir¥out.txt
日立 横浜支店 日立グループ
日立 福岡支店 日立グループ
```

- y コマンドで文字を置き換えます。入力ファイルは，file10.txt です。

  file10.txt の内容

```
あ い う え お あ い う
お え う い あ
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "y/あいうえお/アiウeオ/" file10.txt
ア i ウ e オ ア i ウ
オ e ウ i ア
```

- 検索パターンに一致したレコードと行番号を標準出力に出力します。スクリプトファイルの先頭レコードに「#n」を記述し，検索パターンに一致しないレコードは出力しません。スクリプトファイルは，scpt02.sed です。入力ファイルは，prog01.awk です。

  scpt02.sed の内容

```
#n
/ print/{
=
p
}
```

  prog01.awk の内容

```
BEGIN{
    print "Extract record : group03 - group06" > "file06.txt"
}
/group03/,/group06/{  ※
    count++;
    print >> "file06.txt";

}
END{
    printf "total record : %03d¥n", count >> "file06.txt"
}
```

  注※

  group03 に一致するレコードから group06 に一致するレコードを処理対象とします。

  コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -f scpt02.sed prog01.awk
2
    print "Extract record : group03 - group06" > "file06.txt"
6
    print >> "file06.txt";
10
    printf "total record : %03d¥n", count >> "file06.txt"
```

- l コマンドで印字できない文字とエスケープ文字を可視化して出力します。入力ファイルは，file11.txt です。

  file11.txt の内容

```
日立(タブ)横浜¥支店(タブ)日立グループ
日立(タブ)東京¥支店(タブ)日立グループ
日立(タブ)福岡¥支店(タブ)日立(0x12)※グループ
```

注※

1 バイトのデータです。

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -n ”l” file11.txt
日立¥t横浜¥¥支店¥t日立グループ$
日立¥t東京¥¥支店¥t日立グループ$
日立¥t福岡¥¥支店¥t日立¥022グループ$
```

- 検索パターンに一致したレコードの位置に r コマンドで指定したファイル内のレコードを出力します。d コマンドで検索パターンに一致したレコードを削除します。スクリプトファイルは，scpt03.sed です。入力ファイルは，prog02.awk，および header.txt です。

scpt03.sed の内容

```
/^<Header>/{
r header.txt
d
}
```

prog02.awk の内容

```
#################################################################
<Header>
#################################################################
BEGIN{
    str = ”日立#横浜支店#日立グループ”
    num = split(str, arry,”#”)
    for (i = 1; i <= num; i++ ) {
        print arry[i]
    }
}
```

header.txt の内容

```
# Sample program
# Hitachi group list
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -f scpt03.sed prog02.awk
#################################################################
# Sample program
# Hitachi group list
#################################################################
BEGIN{
    str = ”日立#横浜支店#日立グループ”
    num = split(str, arry,”#”)
    for (i = 1; i <= num; i++ ) {
        print arry[i]
    }
}
```

- ファイルからレコードブロックを抽出します。スクリプトファイルは，scpt04.sed です。入力ファイルは，file12.txt です。

scpt04.sed の内容

```
/^Error01/{
:LOOP
      n※1
      /Error/{
          /^Error01/b LOOP※2
          /^Error01/!d
      }
      b LOOP※2
}
d
```

注※1

現在のパターンスペースの内容を標準出力に出力し，次のレコードを入力します。

注※2

次のレコードを入力するために n コマンドを実行する LOOP ラベルに分岐します。

file12.txt の内容

```
Error01001
The file path used by trace is invalid.
Error02001
Don't know current directory.
Error01002
Unable to get the date for the start of the job execution.
Spool job was not deleted.
Error01003
Cannot change directory.
Error02002
Asc file name size is exceeded limits
for batch coverage function.
Error01004
Failed to get the current time.
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -f scpt04.sed file12.txt
Error01001
The file path used by trace is invalid.
Error01002
Unable to get the date for the start of the job execution.
Spool job was not deleted.
Error01003
Cannot change directory.
Error01004
Failed to get the current time.
```

- q コマンドでパターンに一致したレコードを入力した場合にスクリプトを終了します。入力ファイルは，file13.txt です。

file13.txt の内容

```
Error01001
The file path used by trace is invalid.
Error02001
```

```
Don't know current directory.
Error01002
Unable to get the date for the start of the job execution.
Spool job was not deleted.
Error01003
Cannot change directory.
```

コマンドの実行例を次に示します。

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed "/Error01002/q" file13.txt
Error01001
The file path used by trace is invalid.
Error02001
Don't know current directory.
Error01002
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sed -x
sed: illegal option -- x
usage: sed [-aEnru] command [file ...]
       sed [-aEnru] [-e command] ... [-f command_file] ... [file ...]
```

8.4.29　sleep コマンド（指定された時間だけ停止する）

形式

```
sleep 秒数
```

機能

指定した時間だけ実行を停止します。

引数

秒数

実行を停止する時間を秒単位で指定します。数字以外を指定すると usage が表示されます。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

- 5 秒間実行を停止します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sleep  5
```

- seconds に数字以外を指定した場合を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sleep poipoi
usage: sleep seconds
```

8.4.30　sort コマンド（テキストファイルをソートする）

形式

```
sort [-c|-m] [-b] [-f] [-n] [-r] [-u] [-z]
     [-k 開始位置 [,終了位置] ] [-o 出力先パス名]
     [-T 一時ファイルディレクトリ] [-t フィールド区切り文字]
     [入力パス名 ...]
```

機能

ファイルや標準入力から入力して，次のどれかの処理を実施します。実行結果を標準出力に出力します。

- ソート
- マージ
- ソートされているかのチェック

引数

動作モードの指定

動作モードの指定をするオプションを省略すると，ソートします。ソートを昇順にするか降順にするかは，-r オプションの有無で指定します。

-c

指定したファイルに対してソートされているかどうかチェックします。チェック機能は，1 つのファイルが正しくソートされているかどうかを判定します。

ソートされている場合は終了コード 0 で終了します。ソートされていない場合は，標準エラー出力にメッセージ（sort: found disorder: フィールドの内容）を出力し，終了コード 1 で終了します。

このオプションを指定した場合，入力ファイルを 2 個以上指定するとエラーとなります（sort: too many input files for the -c option）。-u オプション以外のオプションと同時に指定した場合，このオプションを最優先します。複数回指定してもエラーになりません。

-c オプションを指定しない場合はソートします。

-m

入力ファイルはソートされていると仮定してマージだけします。-c オプションが同時に指定されている場合，-m オプションは無視されます。複数回指定してもエラーになりません。

-m オプションを指定しない場合はソートします。

入出力の指定

-o 出力先パス名

標準出力の代わりに出力先パス名に指定したファイルに出力します。

出力先のファイルが存在しない場合は新規に作成します。UNIX の場合，新規に作成したファイルのパーミッションは umask に従って設定します。

ファイルが存在する場合は中間ファイルにいったん出力してから，元のファイルを削除して中間ファイルを出力ファイル名に変更します。この中間ファイルの出力先は入力ファイルと同一ディレクトリに作成します。UNIX の場合，ファイルのパーミッションは umask に従って新たに設定します。

このオプションを複数指定した場合，最後に指定したオプションが有効になります。

UNIX の場合，出力先パス名に/dev/stdout（Windows の場合も小文字で/dev/stdout と記述する）を指定したとき，標準出力を使用します。

出力先パス名にハードリンク，またはシンボリックリンクを指定した場合，リンクを削除し，新規のファイルが作成されます。

-T 一時ファイルディレクトリ

sort コマンドで内部処理として使用する一時ファイルを作成するディレクトリとして，一時ファイルディレクトリで指定したディレクトリを使用します。

一時ファイルとは，ソートおよびマージ処理でメモリ上だけで処理ができない場合に使用する作業用のファイルのことです。

このオプションを複数指定した場合，最後の指定が有効になります。

このオプションを省略した場合，次のディレクトリを使用します。

Windows の場合，**共通アプリケーションフォルダ**¥HITACHI¥JP1AS¥misc を使用します。

UNIX の場合，環境変数 TMPDIR に定義されたディレクトリを使用します。環境変数 TMPDIR が定義されていないときは，/var/tmp を使用します。

入力パス名

入力するファイルを指定します。入力パス名に指定がない，または入力パス名として「-」を指定した場合は，標準入力から入力します。/dev/stdin（Windows の場合も小文字で/dev/stdin と記述する）を指定した場合，標準入力を使用します。

ソートキーに対する指定

-b

-k オプションで指定した開始位置および終了位置に対して，行頭のスペースを無視してソートキーの位置を決めます。-k オプションでソートキーを指定した場合に，-b オプションは有効です。-b オプションは-k オプションよりあとに記述できません。

-f

小文字を大文字と見なしてソートします。複数回指定してもエラーになりません。

-n

先頭の数値文字列を数値としてソートします。

-f オプションよりも-n オプションが優先されます。このオプションは複数回指定できます。

数値の扱いを次に示します。

・アスキー文字の 0(0x30)～9(0x39)で構成される文字列です。

・先頭のスペース(0x20 および 0x09)ならびにゼロ(0x30)は無視します。

・数値にマイナス記号(0x2d)を前に置いてもかまいません。

・数値の小数点は 1 つまで指定できます。

・数値に，整数部分の桁区切り文字を含んでもかまいません。

・小数点および整数部分の桁区切り文字はロケールによって異なります。主に小数点はピリオド(.)，整数部分の桁区切り文字はコンマ(,)に定義されています。

・数値文字列がない場合は 0 として扱います。

・整数の桁数，または小数点以下の値で小数点以下の桁数が 61 以上になる数値をソートキーとして指定しないでください。

-r

-r オプションを指定すると降順に出力します。このオプションを指定しない場合，昇順に出力します。このオプションは複数回指定できます。

フィールド区切りの指定

-t フィールド区切り文字

フィールド区切り文字を指定します。ソートキーのオフセットを決める場合，フィールド区切り文字はフィールドの一部と見なされません。また，連続するフィールド区切り文字は空のフィールドとなります。レコードの区切り文字と同じ文字は指定できません。

-t オプションを指定していない場合，デフォルトのフィールド区切りは連続したスペースとなり，スペースと非スペースの間でフィールドを区切ります。連続したスペースが空のフィールドを区切ることはありません。先頭のスペースは，ソートキーのオフセットを決めるときにフィールドの一部と見なされます。

フィールド区切りとして複数文字指定した場合，およびマルチバイト文字を指定した場合は，先頭 1 バイトをフィールド区切りと見なします。レコード区切りと同じバイト値を指定できません。

-t オプションにフィールド区切り文字の指定をしなかった場合，直後に指定したオプションおよびファイル名がフィールド区切り文字として処理されます。そのため，区切り文字を指定するようにしてください。このオプションは複数指定するとエラーとなります（sort: multiple field delimiters)。

ソートキーの指定

-k 開始位置［,終了位置］

ソートキーの開始位置および終了位置を指定します。複数指定すると，最初のソートキーが等しい場合に次のソートキーで比較できます。

開始位置より終了位置の指定が大きい場合または指定したフィールドが存在しない場合，ソートキーの指定はないものとして扱われ，このソートキーの比較は等しいと判断されます。

開始位置および終了位置は，次の形式で指定します。

```
フィールド位置［.インデント］［bfnr］
```

・フィールド位置

レコード内のフィールドの位置を指定します。数値以外を指定するとエラーとなります（sort: missing field number）。負の値を指定するとエラーとなります（sort: field numbers must be positive）。

開始位置には 0 を指定できません。

終了位置に 0 を指定した場合，レコードの末尾までを仮定します。

フィールド位置には int 型の最大値まで指定できます。それ以上を指定すると，オーバーフローが発生して予測しない値になることがあるため，指定しないでください。

フィールド位置の終了位置に 0 を指定した場合，次のインデントで説明する終了位置の指定はできません。

・インデント

フィールドの中のオフセットを指定します。数値以外または負の値を指定するとエラーとなります（sort: missing offset）。

インデントの単位はバイトで，マルチバイトの途中を指定するとそのバイト位置から評価します。

開始位置のインデントには 0 を指定できません。

終了位置のインデントには 0 を指定できますが，0 を指定するとインデントの指定がないと見なされます。

インデントは int 型の最大値まで指定できます。それ以上を指定すると，オーバーフローが発生して予測しない値になることがあるため，指定しないでください。

フィールド位置の開始位置でインデントを省略した場合，フィールドの先頭バイト位置となります。

フィールド位置の終了位置でインデントを省略した場合，フィールドの最終バイト位置となります。

・ソートキーに指定するオプション

ソートキーは，b オプション，f オプション，n オプション，r オプションで指定できます。

b オプションでは，前のスペースを無視してソートキーの位置を決めます。

f オプションでは，小文字を大文字と見なしてソートします。

n オプションでは，先頭の数値文字列を数値としてソートします。

r オプションでは，降順にソートします。

開始位置に指定した b オプションは開始位置にだけ有効です。また，終了位置に指定した b オプションは終了位置にだけ有効です。終了位置にインデントの指定がない場合は，b オプションは無効になります。-b オプション以外のオプションは，開始位置または終了位置のどちらにも指定でき，意味は同じです。

その他の指定

-u

同一のソートキーであるレコードが複数あった場合は，どれか 1 レコードだけ出力します。-c オプションと同時に指定すると，ソートキーが同じレコードがないかどうかをチェックします。複数回指定してもエラーになりません。

-z

レコードの区切りを変更するオプションです。レコードの区切りとして NULL(0x00)を使用します。このオプションは複数指定するとエラーとなります（sort: multiple field delimiters）。

Windows の場合，入力データの改行コードは，入力時に削除され，出力時に追加されます。そのため，バイナリファイルを入力しないでください。

ソート機能

ソート機能は，1 つまたは複数のファイルを読み込み，1 つまたは複数のソートキーを比較します。ソートキーは，-k オプションで指定し，1 つまたは複数のフィールドで指定します。フィールドは，-t オプションで指定するフィールド区切り文字でレコードを区切ります。

ソートキーの指定がない場合，デフォルトではレコード全体を 1 つのソートキーと見なします。ソートキー同士はバイトごとに比較します。

ソートキーが複数ある場合，先頭に指定したソートキーを比較し，一致したときは次のソートキーを不一致になるまで順次比較します。

ソートキーがすべて一致した場合，レコード全体をバイト単位で比較して出力します。-r オプションの指定がない場合は昇順に，-r オプションの指定がある場合は降順に出力します。

ソートキーに対するオプションの指定方法

ソートキーに対するオプションは 2 つに分類されます。1 つまたは複数のキーを指定した場合にそれぞれに有効となるグローバルオプションと，-k オプションに指定するローカルオプションです。

sort コマンドで指定する-fnrb オプションはグローバルオプションです。また，sort コマンドの-k オプションに指定する fnbr をローカルオプションと呼びます。グローバルオプションは-k オプションより後ろに指定できません。

各オプションをグローバルオプションとローカルオプションで指定した場合の動作を次に示します。

オプション	動作
b	グローバルオプションの指定は，-k オプションに指定する開始位置および終了位置の両方に有効になります。終了位置にインデント指定がない場合，または終了位置のインデントに 0 を指定した場合は，終了位置に対する指定は無効となります。 -k オプションを指定しない場合は，-b オプションの指定は無効となります。
f|n|r	ローカルオプションに指定がある場合，グローバルオプションの指定を無視します。

グローバルオプションの指定例を次に示します。

```
-bfnr -k 1,1 -k 2,2
```

1 番目と 2 番目のフィールドは-bfnr オプションが有効になります。1 番目と 2 番目のフィールドに次のように指定します。

- -b オプション：前のスペースを無視してソートキーの位置を決めます。
- -f オプション：小文字を大文字と見なして比較します。-n オプションの指定によって無効となります。
- -n オプション：先頭の数値文字列を数値として比較します。
- -r オプション：降順となるように比較します。

グローバルオプション-b の指定がない場合のソートキーの範囲を次に説明します。

-k 1
: 1 番目のフィールドからレコードの末尾までをソートキーとします。

-k 1,1
: 1 番目のフィールド全体をソートキーとします。

-k 1,5
: 1 番目のフィールドの先頭バイトから，5 番目のフィールドの最終バイトをソートキーとします。

-k 1.2,5.11
: 1 番目のフィールドの 2 バイト目から 5 番目のフィールドの 11 バイト目までをソートキーとします。

-k 2,1
: 2 番目から 1 番目のフィールドの指定ですが，大小関係が逆転しており，ソートキーは比較されません。

-k 2.1b,5.1b
: 2 番目のフィールドの先頭のスペースを除いた 1 バイト目から，5 番目のフィールドの先頭スペース文字を除いた 1 バイト目までをソートキーとします。

-k 2.1b,5.0b
: 2 番目のフィールドの先頭のスペースを除いた 1 バイト目から，5 番目のフィールドの最終バイトまでをソートキーとします。

マージ機能

マージ機能はソート済みの各入力ファイル間のレコードを比較してデータを整列し，結合します。実際にはソートされていない場合でも，ソートしたと仮定して動作します。file1 および file2 をマージする例を次に示します。

file1

```
AAA
DDD
```

file2

```
BBB
AAA
```

file1 および file2 をマージするコマンド

```
# sort -m file1 file2
```

この場合，次のように結合します。

```
AAA   (file1の1行目のレコード)     ←file1:1行目とfile2の1行目を比較した結果
BBB   (file2の1行目のレコード)     ←file1:2行目とfile2の1行目を比較した結果
AAA   (file2の2行目のレコード)     ←file1:2行目とfile2の2行目を比較した結果
DDD   (file1の2行目のレコード)     ←file2:3行目がないためfile1の2行目
```

小文字を大文字と見なすオプション（-f オプション）

レコードをソートする場合の例を次に示します。

file1

```
a:B
A:b
```

大文字と見なさないソートコマンド

```
$ sort -t : -k 2,2 file1
```

「:」で区切った 2 つ目のフィールドをキーにソートします。

大文字と見なさないソートコマンドの出力

```
a:B
A:b
```

b より B が小さいため，「a:B」が先に出力されます。

大文字と見なすソートコマンド

```
$ sort -f -t : -k 2,2 file1
```

-f オプションを追加します。

大文字と見なすソートコマンドの出力

```
A:b
a:B
```

2 つ目のフィールドを-f オプションの指定に従って大文字と見なして比較するため，同じ値のソートキーと評価します。ソートキーが同じ値であるため，レコード全体をバイトで比較します。この結果，a より A が小さいため「A:b」が先に出力されます。

終了コード

終了コード	意味
0	正常終了
1	正常終了 • 入力したデータはソートされていません（-c オプションを指定した場合）。 • 重複するキーが存在します（-c オプションと-u オプションを指定した場合）。
2	エラー終了

注意事項

- メモリ上で処理ができない場合，一時ファイルを使用して処理をします。一時ファイルの使用時にディスク容量が不足した場合，次のメッセージを出力してエラーになります。

  ```
  sort: fwrite: No space left on device
  ```

  上記のメッセージが出力された場合は，-T オプションを使用して，容量に十分な空きがあるディスクを指定してください。

- sort コマンドの実行を中断すると，-o オプションで指定したファイルが存在するディレクトリに中間ファイルが残ることがあります。この場合は手動で削除してください。-o オプションを指定しない場合も一時ファイルを使用すると，実行中断などで一時ファイルが残ることがあるため，手動で削除してください。
- sort コマンドでのスペースとは，¥t（タブ）とスペース文字(0x20)のことです。また，-z オプションを指定すると¥n（改行）もスペースと見なされます。
- 入力ファイルの最終レコードにレコード区切り文字がなくても，ソートおよびマージの結果にはレコード区切り文字が付加されます。
- 入力の改行コードは<CR><LF>または<LF>で処理できますが，UNIX の場合，＜ CR ＞はデータとして扱います。また，出力結果は入力ファイルの改行コードの形式に関係なく，プラットフォームの改行コードに従った改行コードになります。
- レコードが格納できない場合，格納できるまで拡張します。メモリが確保できない場合，エラーとなります。
- ソートで使用するバッファは 16MB です。このバッファでソートを継続できない場合，一時ファイルを作成します。そのため，大容量データではこのコマンドを使用しないことを推奨します。
- ソート・マージ処理でメモリ上だけで処理ができない場合，sort コマンドの処理を中断すると，次の名前の一時ファイルが残ることがあります。この場合は手動で一時ファイルを削除してください。

  【Windows の場合】

  sor**uuuu**.tmp（**uuuu** は任意の 16 進文字列）

  【UNIX の場合】

  sort**ppppp**.**XXXXXX**（**ppppp** は 5 桁以上のプロセス ID，**XXXXXX** は任意の 6 文字の文字列）

- -o オプションを指定した場合，sort コマンドの処理を中断すると，次のファイル名の中間ファイルが残ることがあります。この場合は手動で中間ファイルを削除してください。

  【Windows の場合】

  出力先パス名の先頭3文字uuuu.tmp（**uuuu** は任意の 16 進文字列）

  【UNIX の場合】

  出力先ファイル名ppppXXXXXX（**ppppp** は 5 桁以上のプロセス ID，**XXXXXX** は任意の 6 文字の文字列）

- -o オプションに同じ出力先パス名を指定して多重実行すると，エラー終了することがあります。この場合の動作は保証できません。

使用例

sort コマンドを実行した結果表示に使用するファイルの形式を次に示します。

- file1

```
yyyy:101
tttt:8
ppppppp:14
```

- file2

```
cccccc:101
ggggg:31
rrrrrrrr:5
mmmmmmm:14
```

コマンドの実行例を次に示します。

- 2 つのテキストファイルを合わせて並べ替えます。

```
$ sort file1 file2
cccccc:101
ggggg:31
mmmmmmm:14
ppppppp:14
rrrrrrrr:5
tttt:8
yyyy:101
```

- 2 つのテキストファイルを合わせて，数値部分の降順で並べ替えます。

```
$ sort -t: -n -r -k 2 file1 file2
yyyy:101
cccccc:101
ggggg:31
ppppppp:14
mmmmmmm:14
tttt:8
rrrrrrrr:5
```

- 1 番目のフィールドをソートキーとして，3 つのファイルをマージします。

```
$ cat s1.txt
AAA s1
DDD s1

$ cat s2.txt
BBB s2
AAA s2

$ cat s3.txt
CCC s3
111 s3

$ sort -m -k 1,1 s1.txt s2.txt s3.txt
AAA s1
BBB s2
AAA s2

CCC s3
111 s3

DDD s1

$
```

- キーが同じデータをソートします。

```
$ cat zr1.txt
aaa:999
$ cat zr2.txt
bbb:999

$ sort -k 2,2 -t : zr2.txt zr1.txt

aaa:999
bbb:999
$
```

- 1 番目のフィールドは数値として，2 番目のフィールドは文字列としてソートします。

  入力コマンド

```
sort -t : -k 1n,1  -k 2,2
```

  入力データ

```
0010:aaa
10:AAA
-1:aaa
-1.00:ZZZ
1:zzz
```

  実行結果

```
-1.00:ZZZ
-1:aaa
1:zzz
```

```
10:AAA
0010:aaa
```

- 3 番目のフィールドの先頭から行末までの小文字を大文字として評価し，2 番目のフィールドは降順に評価してソートします。2 番目のフィールドにはローカルオプションを指定しているため，グローバルオプションの指定は有効ではなく，小文字は小文字として評価します。

  入力コマンド

```
sort -t : -f -k 3  -k 2,2r
```

  入力データ

```
aaa:aaa:cccc
aaa:AAA:cccc
aaa:aaa:AAAA
aaa:AAA:aaaa
aaa:aaa:BBBB
aaa:AAA:bbbb
```

  実行結果

```
aaa:aaa:AAAA
aaa:AAA:aaaa
aaa:aaa:BBBB
aaa:AAA:bbbb
aaa:aaa:cccc
aaa:AAA:cccc
```

- オプションエラーのメッセージを表示します。

  Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sort -w
sort: illegal option -- w
usage: sort [-cm][-bfnruz] [-k field1[,field2]] [-o output]
       [-T dir] [-t char] [file ...]
```

  Linux の例

```
$ sort -w
sort: invalid option -- w
usage: sort [-cm][-bfnruz] [-k field1[,field2]] [-o output]
        [-T dir] [-t char] [file ...]
```

  AIX の例

```
$ sort -w
sort: illegal option -- w
usage: sort [-cm][-bfnruz] [-k field1[,field2]] [-o output]
        [-T dir] [-t char] [file ...]
```

- 入力ファイルにディレクトリを指定した場合のメッセージを表示します。

```
$ ./sort dir01
sort: dir01: Is a directory
```

- 入力ファイルとして存在しないファイルを指定した場合のメッセージを表示します。

```
$ ./sort xxxx
sort: xxxx: No such file or directory
```

- 存在しない一時ファイルディレクトリを指定した場合のメッセージを表示します。

  Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sort -mTxxx s0.txt s0.txt s0.txt s0.txt s0.txt s0.t
xt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
sort: xxx¥sort: The directory name is invalid.
```

  Linux の例

```
$ ./sort -mT xxxx s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
sort: xxxx/sort.SDm1yr: No such file or directory
```

  AIX の例

```
$ ./sort -mT xxxx s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
sort: xxxx/sort.XXXXXX: No such file or directory
```

- 不当なフィールド位置を指定した場合のメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sort -k xx
sort: missing field number
```

- 不当なフィールド位置を指定した場合のメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sort -k 0 s0.txt
sort: field numbers must be positive
```

- 不当なインデントをフィールド位置に指定した場合のメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sort -k 1.0 s0.txt
sort: illegal offset
```

8.4.31　split コマンド（ファイルを分割する）

形式

```
split [-a サフィックス長]
      [-b バイト数 [k|m] |-l 行数] [入力パス名 [プリフィックス] ]
```

機能

ファイルや標準入力の内容を分割して，ファイルに出力します。

引数

-a サフィックス長

分割後にファイルの名前に付けるサフィックスの長さを指定します。

1 から 254 の範囲で指定します。範囲外の値を指定した場合，または数値以外を指定した場合はエラーとなります（split: 指定値: too small／split: 指定値: too large／split: 指定値: invalid）。デフォルトは 2 です。複数回指定できますが，最後に指定した値が有効となります。

-b バイト数 [k|m]

ファイルをデータサイズで分割する場合のサイズをバイトで指定します。-l オプションと同時に指定した場合はエラーとなり，usage が表示されます。

- k：キロバイト単位の値になります（1k=1,024 バイト）。
- m：メガバイト単位の値になります（1m=1,048,576 バイト）。

複数回指定できますが，最後に指定した値が有効となります。

-l 行数

行数でファイルを分割する場合に，行数を指定します。-b オプションと同時に指定した場合はエラーとなり，usage が表示されます。-b オプションおよび-l オプションを指定しない場合は，1000 行が指定されたものとします。

入力パス名

入力するファイル名を指定します。省略時は標準入力を仮定します。

プリフィックス

分割後にファイルの名前に付けるプリフィックスとして使用します。

分割後のファイルの名前は次のように決定します。

プリフィックス＋サフィックス

プリフィックスは指定がある場合は，その文字列を使用します。指定がない場合は，「x」，「y」，「z」の順番に使用されます。

サフィックスは，a～z を組み合わせた文字列を，サフィックス長で指定された長さ分使用します。サフィックスは文字コード順に使用されます。

例：2 バイトの場合，aa，ab，ac，…，az，ba，bb，…となります。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 出力ファイルが入力ファイルと同じ場合，上書きをします。パス名が同じにならないようにプリフィックスを指定するか，または入力するファイルをカレントディレクトリとは別のディレクトリに移動してください。
- 分割後のファイル名が不足する場合は，エラーとなります（split: too many files）。作成したファイルは削除しないで終了します。この場合，サフィックス長を大きく指定する，またはバイト数および行数を大きくしてください。
- 分割後のファイル名の長さがシステムの上限を超えた場合，次のメッセージを出力してエラーになります。

  【Windows の場合】

  ```
  split : ファイル名 : No such file or directory
  ```

  【UNIX の場合】

  ```
  split : ファイル名 : File name too long
  ```

- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。

使用例

- test1.txt ファイルを 2 行単位で分割します。

```
$ ls
test1.txt
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ split -l2 test1.txt
$ ls
test1.txt  xaa  xab  xac  xad  xae
$ cat xaa
0001:test1.txt
0002:test1.txt
$ cat xab
0003:test1.txt
0004:test1.txt
$ cat xac
0005:test1.txt
0006:test1.txt
$ cat xad
0007:test1.txt
```

```
0008:test1.txt
$ cat xae
0009:test1.txt
0010:test1.txt
$
```

- test1.txt ファイルを 40 バイト単位で分割します。

```
$ ls
test1.txt
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ split -a 5 -b 40 test1.txt new
$ ls
newaaaaa  newaaaab  newaaaac  newaaaad  test1.txt
$ cat newaaaaa
0001:test1.txt
0002:test1.txt
0003:test1$
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥split -z
split: illegal option -- z
usage: split [-a suffix_length]
             [-b byte_count[k|m] | -l line_count] [file [name]]
```

8.4.32 stat コマンド（ファイルまたはディレクトリの状態を標準出力に出力する）

形式

```
stat [-L] [-c 書式] [-t] パス名 ...
```

機能

ファイルまたはディレクトリの状態を標準出力に出力します。パス名にシンボリックリンクファイルが指定された場合，リンクをたどらないでシンボリックリンクファイルの状態を表示します。

引数

-L

--dereference

パス名にシンボリックリンクファイルを指定した場合，リンクをたどった先のファイルまたはディレクトリの状態を表示します。

-c 書式

--format=書式

ファイルまたはディレクトリの状態を書式に従った形式で表示します。書式には，書式指定コードおよび任意の文字列を指定できます。このオプションを指定した場合の表示形式および書式指定コードについては，項目「表示形式」の「独自の表示形式」を参照してください。指定可能な書式指定コード以外を指定した場合，標準エラー出力に警告メッセージを出力し，標準出力に「?」を出力して，後続の処理を続行します。

-t オプションと同時に指定した場合，このオプションが優先されます。

-t

--terse

簡潔な表示形式で情報を出力します。簡潔な表示形式については，項目「表示形式」の「簡潔な表示形式」を参照してください。

パス名

状態を表示するファイル名またはディレクトリ名を指定します。

パス名を複数指定した場合は，ファイルまたはディレクトリの状態を縦に連続して表示します。複数指定して実行した場合はすべてのファイルまたはディレクトリを処理し，1 つでも状態の表示に失敗したファイルまたはディレクトリがあると，終了コード 1 で終了します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

表示形式

ファイル情報の表示形式には，通常表示形式，簡潔な表示形式，独自の表示形式の 3 種類があります。どの表示形式になるかは指定するオプションによって決まります。

通常表示形式

オプションを指定しないで実行した場合の表示形式です。次のファイル情報をラベル付きで表示します。

出力情報	ラベル
クォートされたファイル名 シンボリックリンクの場合は，シンボリックリンクの参照先も表示します。	File:
合計サイズ	Size:
割り当てられたブロック数	Blocks:
ファイルシステム I/O 操作での最適なブロックサイズ	IO Block:
ファイルの種類 表示内容は，項目「表示形式」の「ファイルの種類の表示内容」を参照してください。	－
デバイス番号 「**デバイス番号の16進数表記**h/**デバイス番号の10進数表記**d」の形式で表示します。 デバイスファイルではない場合は表示しません。	Device:
i ノード番号	Inode:
ハードリンクの数	Links:
デバイスファイルの種類 「**メジャーデバイス番号**，**マイナーデバイス番号**」の形式で表示します。デバイスファイルではない場合は表示しません。	Device type:
パーミッション 「**パーミッションの8進数表記**/**パーミッションの文字列表記**」の形式で表示します。	Access:
所有者のユーザー情報 「**所有者のユーザーID**/**所有者のユーザー名**」の形式で表示します。	Uid:
所有者のグループ情報 「**所有者のグループID**/**所有者のグループ名**」の形式で表示します。	Gid:
ファイルの最終アクセス日時	Access:
ファイルの最終修正日時	Modify:
ファイル情報の最終変更日時	Change:

（凡例）

－：ラベルがないことを示します。

簡潔な表示形式

-t オプションを指定した場合の表示形式です。次のファイル情報をスペース区切りで連続して表示します。

- ファイル名
- 合計サイズ
- 割り当てられたブロック数
- raw モードの 16 進数表記

- 所有者のユーザー ID
- 所有者のグループ ID
- デバイス番号の 16 進数表記
- i ノード番号
- ハードリンクの数
- メジャーデバイス番号
- マイナーデバイス番号
- ファイルの最終アクセス日時（エポックからの秒数）
- ファイルの最終修正日時（エポックからの秒数）
- ファイル情報の最終変更日時（エポックからの秒数）
- ファイルシステム I/O 操作での最適なブロックサイズ

独自の表示形式

-c オプションを指定した場合の表示形式です。書式指定コードおよび任意の文字列を組み合わせて，独自の表示形式を指定できます。また，書式指定コードの%の後ろには，フラグ文字，フィールド幅，精度も定義できます。

- 書式指定コード
  指定できる書式指定コードを次の表に示します。

書式指定コード	意味
%a	パーミッションの 8 進数表記 Windows の場合，所有者だけのパーミッションを表示します。
%A	パーミッションの文字列表記 Windows の場合，所有者だけのパーミッションを表示します。
%b	割り当てられたブロック数 Windows の場合，常に 0 を表示します。
%B	各ブロックの大きさ（バイト単位） Windows の場合，常に 0 を表示します。
%d	デバイス番号の 10 進数表記 Windows の場合，ドライブ番号を表示しますが，次の場合は表示が異なります。 • フルパスにしたときにドライブレターに続く「:」がないパスの場合 デバイス番号として「-」を表示して，後続の処理を続行します。 • デバイス番号取得処理でエラーが発生した場合 警告メッセージを標準エラー出力に出力し，デバイス番号として「?」を表示して，後続の処理を続行します。
%D	デバイス番号の 16 進数表記 Windows の場合，ドライブ番号を表示しますが，次の場合は表示が異なります。

書式指定コード	意味
%D	• フルパスにしたときにドライブレターに続く「:」がないパスの場合 デバイス番号として「-」を表示して，後続の処理を続行します。 • デバイス番号取得処理でエラーが発生した場合 警告メッセージを標準エラー出力に出力し，デバイス番号として「?」を表示して，後続の処理を続行します。
%f	raw モードの 16 進数表記 Windows の場合，所有者だけのパーミッションを表示します。
%F	ファイルの種類 表示内容は，項目「表示形式」の「ファイルの種類の表示内容」を参照してください。
%g	所有者のグループ ID Windows の場合，常に 0 を表示します。
%G	所有者のグループ名 Windows の場合，常に「...」を表示します。 UNIX の場合，所有者のグループ名を取得できないときは所有者のグループ ID を表示して，後続の処理を続行します。
%h	ハードリンクの数
%i	i ノード番号 Windows の場合，常に 0 を表示します。
%n	ファイル名
%N	クォーテーションで囲まれたファイル名 シンボリックリンクの場合は参照先ファイル名も表示します。 参照先ファイル名の取得に失敗したときは，警告メッセージを標準エラー出力に出力し，参照先ファイル名は表示しないで，後続の処理を続行します。
%o	ファイルシステム I/O 操作での最適なブロックサイズ Windows の場合，常に 0 を表示します。
%s	合計サイズ（バイト単位） Windows の場合，ディレクトリの合計サイズは常に 0 を表示します。 UNIX の場合，デバイスファイルの合計サイズは常に 0 を表示します。
%t	メジャーデバイス番号の 16 進数表記 Windows の場合，常に 0 を表示します。
%T	マイナーデバイス番号の 16 進数表記 Windows の場合，常に 0 を表示します。
%u	所有者のユーザー ID Windows の場合，常に 0 を表示します。
%U	所有者のユーザー名 Windows の場合，所有者のユーザー名が取得できないときは「...」を表示して，後続の処理を続行します。 UNIX の場合，所有者のユーザー名が取得できないときは所有者のユーザー ID を表示して，後続の処理を続行します。

書式指定コード	意味
%x	ファイルの最終アクセス日時※ Windows の場合，ファイルの最終修正日時と同じ情報を表示します。 ファイルの最終アクセス日時の表示に失敗した場合，警告メッセージを標準エラー出力に出力し，ファイルの最終アクセス日時の表示は「?」にして，後続の処理を続行します。
%X	ファイルの最終アクセス日時のエポック（UTC の 1970 年 1 月 1 日 00:00:00）からの秒数 Windows の場合，ファイルの最終修正日時と同じ情報を表示します。
%y	ファイルの最終修正日時※ ファイルの最終修正日時の表示に失敗した場合，警告メッセージを標準エラー出力に出力し，ファイルの最終修正日時の表示は「?」にして，後続の処理を続行します。
%Y	ファイルの最終修正日時のエポック（UTC の 1970 年 1 月 1 日 00:00:00）からの秒数
%z	ファイル情報の最終変更日時※ Windows の場合，ファイルの最終修正日時と同じ情報を表示します。 ファイル情報の最終変更日時の表示に失敗した場合，警告メッセージを標準エラー出力に出力し，ファイル情報の最終変更日時の表示は「?」にして，後続の処理を続行します。
%Z	ファイル情報の最終変更日時のエポック（UTC の 1970 年 1 月 1 日 00:00:00）からの秒数 Windows の場合，ファイルの最終修正日時と同じ情報を表示します。
%%	%（パーセント）記号

注※

「ファイルの最終アクセス日時」,「ファイルの最終修正日時」,「ファイル情報の最終変更日時」で出力する日時の出力形式は次のとおりです。

YYYY-MM-DD hh:mm:ss.nnnnnnnnn +/-hhmm

YYYY：西暦年

MM：月

DD：日

hh：時

mm：分

ss：秒

nnnnnnnnn：1 秒未満の日時。常に 000000000 と出力します。

+/-hhmm：タイムゾーン。UTC からの時差を示します。

- フラグ文字

書式指定コードの%の後ろに次のフラグ文字を指定できます。省略することもできます。

フラグ文字	内容
#	0 以外の 8 進数表記に対しては前に 0 を付けます。 0 以外の 16 進数表記に対しては前に 0x を付けます。
-	出力文字列をフィールドの左にそろえます。

フラグ文字	内容
+	数値の正負を示す記号（+/-）を常に表示します。 符号なし整数として定義されているファイル情報に対しては，この指定は無視されます。
スペース	符号付き整数として定義されているファイル情報に対して，正の数字の前にスペースを表示します。 +と同時に指定した場合は，+が優先されます。
0	フィールドの左側をスペースの代わりに 0 で埋めます。

- フィールド幅

  書式指定コードの%またはフラグ文字の後ろに数値を指定することで，最小のフィールド幅を定義します。フィールド幅は 0～2147483647 の範囲で指定できます。省略することもできます。

- 精度

  書式指定コードの%またはフラグ文字の後ろに，ピリオド（.）と次に示す数値を定義します。精度は，UNIX の場合は 0～2147483647 の範囲で，Windows の場合は 0～512 の範囲で指定できます。省略することもできます。

  - ファイル情報が文字列の場合

    表示する最大長を定義します。

  - ファイル情報が数値の場合

    最小桁数を定義します。

ファイルの種類の表示内容

表示されるファイルの種類と，その意味は次のとおりです。

ファイルの種類	意味
regular file	通常ファイル
directory	ディレクトリ
symbolic link	シンボリックリンク
fifo	FIFO【UNIX 限定】
socket	ソケット【UNIX 限定】
block special file	ブロック型スペシャルファイル【UNIX 限定】
character special file	キャラクタ型スペシャルファイル【UNIX 限定】
unknown file	不明なファイル（上記以外のファイル）【UNIX 限定】

注意事項

- Windows の場合，「通常ファイル」，「ディレクトリ」，「シンボリックリンク」以外については，通常ファイルまたはディレクトリとして扱います。

UNIX の場合，「通常ファイル」，「ディレクトリ」，「シンボリックリンク」，「FIFO」，「ソケット」，「ブロック型スペシャルファイル」，「キャラクタ型スペシャルファイル」以外は不明なファイルとして扱います。

- Windows の場合，日付と時刻の表示には「日付と時刻のプロパティ」で定義されているタイムゾーンが使用されます。環境変数 TZ の値は影響しません。

  ただし，タイムゾーンの表示は環境変数 TZ の値とコントロールパネルの「日付と時刻のプロパティ」ダイアログボックスで定義されているタイムゾーンを使用するため，環境変数 TZ の値とコントロールパネルの「日付と時刻のプロパティ」ダイアログボックスで定義されているタイムゾーンは同じにしてください。同じでない場合，「ファイルの最終アクセス日時」，「ファイルの最終修正日時」，「ファイル情報の最終更新日時」で表示されるタイムゾーンが正しく表示されません。

- UNIX の場合，ブロックサイズのデフォルトは 512 バイトです。ブロックサイズは環境変数 BLOCKSIZE で変更できます。

  環境変数 BLOCKSIZE は 512 から 1G（1024×1024×1024）の範囲で指定できます。範囲外の値を指定した場合は次のように処理し，警告メッセージを標準エラー出力に出力して，後続の処理を続行します。

  - 環境変数 BLOCKSIZE に 512 より小さい値を指定した場合

    ブロックサイズを 512 バイトとします。

  - 環境変数 BLOCKSIZE に 1G（1024×1024×1024）より大きい値を指定した場合

    ブロックサイズを 1G（1024×1024×1024）バイトとします。

  環境変数 BLOCKSIZE でブロックサイズを変更する場合は，512 の倍数を指定してください。512 の倍数でない場合，余りは切り捨てられます。例えば，1,500 バイトが定義されている場合，1,024 バイトとして扱います。

  数字の後ろには，何倍であるかを示すサイズ文字（G（1024×1024×1024），M（1024×1024），K（1024））を指定できます。数値とサイズ文字以外を指定した場合，ブロックサイズを 512 バイトとして，警告メッセージを標準エラー出力に出力し，後続の処理を続行します。

- Solaris の場合，ディレクトリ内のファイルの「割り当てられたブロック数」を出力する際，間接ブロックを含むブロックの総数を表示します。また，ハードリンクされたファイルがある場合，「割り当てられたブロック数」は正しく表示されません。

使用例

- ファイルの状態を通常表示形式で表示します。

  Windows の場合

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥stat .¥test.txt
  File: `.¥test.txt'
  Size: 7               Blocks: 0          IO Block: 0      regular file
Device: 0003h/00003d    Inode: 0           Links: 1
Access: (0600/-rw-------)  Uid: (    0/   user1)   Gid: (    0/     ...)
Access: 2014-02-20 10:31:28.000000000 +0900
Modify: 2014-02-20 10:31:33.000000000 +0900
Change: 2014-02-20 10:31:28.000000000 +0900
```

UNIX の場合

```
$ stat ./test.txt
  File: `./test.txt'
  Size: 4               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 688407      Links: 2
Access: (0644/-rw-r--r--)  Uid: (  501/   user1)   Gid: (  502/  group1)
Access: 2014-02-11 18:35:52.000000000 +0900
Modify: 2014-02-11 18:35:52.000000000 +0900
Change: 2014-02-18 16:08:39.000000000 +0900
```

- ファイルのサイズだけを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥stat -c %s .¥test.txt
7
```

- 複数ファイルの状態を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥stat .¥test.txt .¥test1.txt
  File: `.¥test.txt'
  Size: 7               Blocks: 0          IO Block: 0      regular file
Device: 0003h/00003d    Inode: 0           Links: 1
Access: (0600/-rw-------)  Uid: (    0/   user1)   Gid: (    0/     ...)
Access: 2014-02-20 10:31:28.000000000 +0900
Modify: 2014-02-20 10:31:33.000000000 +0900
Change: 2014-02-20 10:31:28.000000000 +0900
  File: '.¥test1.txt'
  Size: 7               Blocks: 0          IO Block: 0      regular file
Device: 0003h/00003d    Inode: 0           Links: 1
Access: (0600/-rw-------)  Uid: (    0/   user1)   Gid: (    0/     ...)
Access: 2014-02-20 14:34:01.000000000 +0900
Modify: 2014-02-20 14:34:01.000000000 +0900
Change: 2014-02-20 14:34:01.000000000 +0900
```

- ファイルの状態を独自の表示形式で出力します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥stat -c "Filename : %n" .¥test.txt
Filename : .¥test.txt
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥stat -z
stat: illegal option -- z
usage: stat [-L] [-c format] [-t] file ...
```

8.4.33 tail コマンド（ファイルの最後の部分を表示する）

形式

```
tail [-r] [-b ブロック数|-c バイト数|-n 行数|-行数] [パス名 ...]
```

機能

ファイルや標準入力の最後の部分を標準出力に出力します。デフォルトは標準入力です。入力のバイト単位，行単位またはブロック単位の位置から表示されます。指定した表示範囲のデータが存在しない場合でもエラーになりません。表示できるデータは表示されます。

引数

+符号のある数は入力の先頭からの位置を示します。例えば，-c +2 は入力の先頭から 2 バイト目で表示を始めます。

-符号がある数字または符号のない数字は最後からの位置を示します。例えば，-n 2 は入力の最後から 2 行目を示します。デフォルトは，-n 10 または入力の最後の 10 行です。

-r

行単位で逆順に表示します。

-b オプションと共に指定した場合，ファイルの終端から，-b オプションに指定したブロック数分戻った所まで，行単位にファイルの終端から出力します。表示開始位置がマルチバイト文字の中間の場合，文字化けすることがあります。

-c オプションと共に指定した場合，ファイルの終端から，-c オプションに指定したバイト数分戻った所まで，行単位にファイルの終端から出力します。表示開始位置がマルチバイト文字の中間の場合，文字化けすることがあります。

-n オプションまたは行数と共に指定した場合，ファイルの終端から，-n オプションまたは行数で指定した行数分戻った所まで，行単位にファイルの終端から出力します。

-r オプションだけ指定した場合，入力したすべての行を，行単位にファイルの終端から出力します。複数回指定してもエラーになりません。

-b ブロック数

-r オプションを指定しない場合，1 ブロック（512 バイト）単位で表示開始位置を指定します。

符号がない場合またはマイナス（-）符号がある場合，最後からの位置を示します。プラス（+）符号がある場合，入力の先頭からの表示位置を示します。

ブロック数の指定を省略した場合は，エラーメッセージ（tail: option requires an argument - オプション）と usage が出力されます。ブロック数に数値以外を指定した場合は，エラーメッセージ（tail: illegal offset -- 指定文字列）が出力されます。

表示開始位置がマルチバイト文字の中間の場合，文字化けすることがあります。また，改行コードはバイト数にカウントされます。例えば Windows の場合，行末が<LF>のときは 1 バイト，<CR><LF>のときは 2 バイトでカウントします。複数回指定すると usage が出力されます。

-c バイト数

-r オプションを指定しない場合，バイト数で表示開始位置を指定します。

符号がない場合またはマイナス（-）符号がある場合，最後からの位置を示します。プラス（+）符号がある場合，入力の先頭からの表示位置を示します。

バイト数の指定を省略した場合は，エラーメッセージ（tail: option requires an argument - オプション）と usage が出力されます。

バイト数に数値以外を指定した場合は，エラーメッセージ（tail: illegal offset -- 指定文字列）が出力されます。

表示開始位置がマルチバイト文字の中間の場合，文字化けすることがあります。また，改行コードはバイト数にカウントされます。例えば Windows の場合，行末が<LF>のときは 1 バイト，<CR><LF>のときは 2 バイトでカウントします。複数回指定すると usage が出力されます。

-n 行数|-行数

-r オプションを指定しない場合，行単位で表示開始位置を指定します。

符号がない場合またはマイナス（-）符号がある場合，最後からの位置を示します。プラス（+）符号がある場合，入力の先頭からの表示位置を示します。

行数の指定を省略した場合は，エラーメッセージ（tail: option requires an argument - オプション）と usage が出力されます。行数に数値以外を指定した場合は，エラーメッセージ（tail: illegal offset -- 指定文字列）が出力されます。複数回指定すると usage が出力されます。

パス名

入力ファイルを指定します。指定がない場合は標準入力から入力します。入力ファイルは複数指定できます。複数ファイルを指定した場合，それぞれのファイルを識別するために，次に示す文字をそれぞれのファイルの出力に先行して出力します。2 つ目以降のファイルの場合，改行のあとに次に示す文字を出力します。

==> ファイル名 <==

複数ファイルを指定して実行した場合，すべてのファイルに対して処理をしたあと，オープンに失敗したファイルが 1 つでもあると終了コード 1 で終了します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 指定した表示範囲のデータが存在しない場合，エラーになりません。表示できるデータは表示します。
- -r オプションを指定しない場合，かつ-b オプション，-c オプション，-n オプションを指定しない場合，-n に 10 が指定されます。
- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。

使用例

- test1.txt と test2.txt ファイルの最後の 2 行を表示します。

```
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ cat test2.txt
0001:test2.txt
0002:test2.txt
0003:test2.txt
0004:test2.txt
0005:test2.txt
0006:test2.txt
0007:test2.txt
0008:test2.txt
0009:test2.txt
0010:test2.txt
$ tail -n2 test1.txt test2.txt
==> test1.txt <==
0009:test1.txt
0010:test1.txt

==> test2.txt <==
0009:test2.txt
0010:test2.txt
$
```

- test1.txt ファイルの先頭から 5 行目以降を表示します。

```
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ tail -n+5 test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$
```

- -r オプションを指定した場合の例 1 を表示します。

```
$ cat zztt1.txt
1:0001:zzzz:
2:0001:aaaa:
3:0001:JJJJ:
4:0001:cccc:
5:0001:cccc:
6:0001:cccc:
7:0001:cccc:
8:0001:cccc:
9:0001:cccc:
10:0001:cccc:
11:0001:cccc:
12:0001:cccc:
$ tail -r -n  2 zztt1.txt
12:0001:cccc:
11:0001:cccc:
$ tail -r  zztt1.txt     (10行ではなく全行表示)
12:0001:cccc:
11:0001:cccc:
10:0001:cccc:
9:0001:cccc:
8:0001:cccc:
7:0001:cccc:
6:0001:cccc:
5:0001:cccc:
4:0001:cccc:
3:0001:JJJJ:
2:0001:aaaa:
1:0001:zzzz:
$
```

- -r オプションを指定した場合の例 2 を表示します。

```
$ cat block.txt  --->1行100バイト+改行コード(¥n)で101行のデータ
0000000000:1234567890123(中略)789012345678901234567890123456 78T
00001xxx00:1234567890123(中略)789012345678901234567890123456 78T
00002xxx00:1234567890123(中略)789012345678901234567890123456 78T
    (中略)
00098xxx00:1234567890123(中略)789012345678901234567890123456 78T
00099xxx00:1234567890123(中略)789012345678901234567890123456 78T
00100xxx00:1234567890123(中略)789012345678901234567890123456 78T
$ tail -b 1 block.txt
45678T
00096xxx00:1234567890123(中略)789012345678901234567890123456 78T
00097xxx00:1234567890123(中略)789012345678901234567890123456 78T
00098xxx00:1234567890123(中略)789012345678901234567890123456 78T
00099xxx00:1234567890123(中略)789012345678901234567890123456 78T
00100xxx00:1234567890123(中略)789012345678901234567890123456 78T
$ tail -rb 1 block.txt
00100xxx00:1234567890123(中略)789012345678901234567890123456 78T
00099xxx00:1234567890123(中略)789012345678901234567890123456 78T
00098xxx00:1234567890123(中略)789012345678901234567890123456 78T
00097xxx00:1234567890123(中略)789012345678901234567890123456 78T
00096xxx00:1234567890123(中略)789012345678901234567890123456 78T
45678T
$ tail -c 110 block.txt
2345678T
```

```
00100xxx00:1234567890123(中略)789012345678901234567890123456 78T
$ tail -rc 110 block.txt
00100xxx00:1234567890123(中略)789012345678901234567890123456 78T
2345678T
$
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tail -z
tail: illegal option -- z
usage: tail [-r] [-b number | -c number | -n number | -number] [file ...]
```

8.4.34 tar コマンド（対象パス名をアーカイブに格納，およびアーカイブから抽出，表示する）

形式

【Windows 限定】

```
tar {-c|-r|-t|-u|-x} [-b ブロック化係数] [-d] [-f アーカイブ] [-h] [-m] [-P]
    [-V] [-v] [-X除外リストファイル] [-z] [--cmdrc0] [--compatible=種別]
    [-C ディレクトリ] [-T ファイル] [対象パス名...]

tar {c|r|t|u|x}[b][d][f][h][m][P][V][v][X][z] [ブロック化係数] [アーカイブ]
    [除外リストファイル] [--cmdrc0] [--compatible=種別] [-C ディレクトリ]
    [-T ファイル] [対象パス名...]
```

【UNIX 限定】

```
tar {-c|-r|-t|-u|-x} [-b ブロック化係数] [-d] [-f アーカイブ] [-h] [-m] [-P] [-p]
    [-V] [-v] [-X除外リストファイル] [-z] [--cmdrc0] [--compatible=種別]
    [-C ディレクトリ] [-T ファイル] [対象パス名...]

tar {c|r|t|u|x}[b][d][f][h][m][P][p][V][v][X][z] [ブロック化係数] [アーカイブ]
    [除外リストファイル] [--cmdrc0] [--compatible=種別] [-C ディレクトリ]
    [-T ファイル] [対象パス名...]
```

機能

対象パス名をアーカイブに格納，およびアーカイブから抽出，表示します。コマンドの動作は，1 つの機能文字（c，r，t，u，またはx）とその他の機能修飾子を指定することで決定します。

このコマンドで作成するアーカイブは ustar format 形式です。

引数

機能文字オプション

-c

--create

新しいアーカイブを作成します。アーカイブの先頭から対象パス名を格納します。対象パス名がディレクトリの場合，ディレクトリ下も格納します。

-r

--append

指定したアーカイブの終端に対象パス名を追加格納します。

-t

--list

アーカイブに格納されているファイル名一覧を表示します。

-v オプションと共に指定した場合，次の詳細情報も表示されます。

「*パーミッション uid/gid ファイルサイズ 日時 ファイル名*」

-u

--update

対象パス名がアーカイブ内の同名ファイルより新しい，またはアーカイブに格納されていないファイルを追加格納します。

-x

--extract

--get

アーカイブから対象パス名を抽出します。対象パス名がディレクトリの場合，ディレクトリ下も抽出します。

ファイルの抽出では次のように動作します。

- 既存のファイルが存在する場合は，既存のファイルを削除した上で新規作成します（既存のファイルのパーミッションは保持されません）。

  UNIX で-p オプションを指定していない場合，パーミッションはアーカイブに格納されているパーミッションとumask によって決定され，実行者が一般ユーザーの場合 setuid ビット，setgid ビット，スティッキービットは設定されません。

  すでにディレクトリが存在する場合，該当するディレクトリに抽出ファイルを復元します。
- UNIX で実行者がスーパーユーザーの場合，ユーザー，グループを抽出して設定します。
- Windows は，パーミッション，ユーザー，グループは抽出して設定されません。

機能修飾子オプション

-b *ブロック化係数*

--blocking-factor=*ブロック化係数*

ブロックサイズを*ブロック化係数**512 バイトとします。

デフォルト値は 20（ブロックサイズは 20*512 バイト）です。

*ブロック化係数*に指定できる範囲は 1～63 です。

-c オプションと共に指定した場合に有効となり，-t，-x オプションと共に指定した場合は意味のない指定となります。

-r，-u オプションと共に指定した場合は，アーカイブのサイズと指定した*ブロック化係数*で決定されます。

複数指定は，最後に指定したブロック化係数が有効となります。

指定できる範囲外を指定した場合，20 を仮定して動作します。

-d

アーカイブに格納する場合，スペシャルファイル，パイプファイルも対象とします。

このオプションはデフォルトで付加され，オプション有無による動作の違いはありません。

-f *アーカイブ*

--file=*アーカイブ*

アーカイブを指定します。指定できるアーカイブは通常ファイル，パイプファイル，シンボリックリンクファイルです。

このオプションを指定していない，または「-」（ハイフン）を指定した場合は，標準入出力を示します。

-r，-u オプションの場合，標準入出力とパイプファイルの組み合わせは使用できません。

このオプションを複数指定した場合，最後に指定したアーカイブが有効となります。

-h

--dereference

シンボリックリンクをたどり，ファイルまたはディレクトリとして扱います。

-c，-r，-u オプションと共に指定した場合に有効となり，-t，-x オプションと共に指定した場合は意味のない指定となります。

-m

--touch

ファイルの最終修正日時は抽出し設定しません。

-x オプションと共に指定した場合に有効となり，-c，-r，-u，-t オプションと共に指定した場合は意味のない指定となります。

Windows の場合は-m オプションの指定に関わらず，ディレクトリの最終修正日時は設定できません。

-P

--absolute-names

対象パス名のルートディレクトリを取り除きません。

-p

--same-permissions

--preserve-permissions

【UNIX 限定】

パーミッションを抽出して設定します。-x オプションと共に指定した場合に有効となり，-c，-r，-u，-t オプションと共に指定した場合は意味のない指定となります。

スーパーユーザーはデフォルトで動作します。

-V

アーカイブの詳細情報を表示する際，ファイルのタイプを示す英字も出力します。

このオプションはデフォルトで付加され，オプションの有無による動作の違いはありません。

-v

--verbose

詳細情報を出力します。

-z

--gzip

--gunzip

--ungzip

gzip 形式のアーカイブを操作します。

-c オプションと共に指定した場合，新しいアーカイブを gzip 形式で圧縮して作成します。

-x オプションと共に指定した場合，gzip 形式で圧縮されたアーカイブから対象パス名を抽出します。

-t オプションと共に指定した場合，gzip 形式で圧縮されたアーカイブに格納されているファイル名一覧を表示します。

gzip 形式のアーカイブを操作するとき，tar コマンドはgzip コマンドを呼び出します。tar コマンドから呼び出すgzip コマンドに，gzip コマンドのオプションを渡したい場合は，環境変数GZIP にgzip コマンドのオプションを設定してからtar コマンドを実行してください。環境変数GZIP の詳細については，「2.5　環境変数を設定する」を参照してください。

-z オプションは，-r オプション，または-u オプションと同時には指定できません。-r オプション，-u オプションと同時に指定した場合はエラー終了します。

-x オプション，または-t オプションと共に指定する場合に，アーカイブが gzip 形式で圧縮されていないときは，抽出や表示ができません。

なお，--gzip，--gunzip，および--ungzipオプションは，-zオプションのロングオプションという意味しかありません。例えば，-cオプションと--gunzipオプションを指定した場合も，新しいアーカイブをgzip形式で圧縮して作成します。

--cmdrc0

オプションの解析後にエラーが発生しても戻り値を必ず0とします。

--compatible=*種別*

*種別*には次の文字列を指定できます。

long-file-name

対象パス名をアーカイブに格納する場合，対象パス名長はディレクトリパス長155バイト，ファイル名100バイト，シンボリックリンクのリンク先のファイル名長100バイトのアーカイブ内のヘッダ領域の制限があります。

このオプションを使用した場合，制限を解除できます（対象パス名は3072バイトまで可能です。実際に作成できるパス名長は各OSの最大長に依存します）。

ただし，このオプションで作成したアーカイブは，ustar format形式のOS提供のtarコマンドと互換性はありません。

-c，-r，-uオプションと共に指定した場合に有効となり，-t，-xオプションと共に指定した場合は無視されます。

-C *ディレクトリ*

--directory=*ディレクトリ*

指定したディレクトリに移動して動作します。

対象パス名，-Tオプション，-Cオプションを組み合わせた場合，-Cオプションのあとに指定した対象パスが対象となります。

-Cオプションを複数指定する場合，2個目以降の-Cオプションは以前に指定した-Cオプションによってカレントディレクトリが移動していることを考慮する必要があります。

-T *ファイル*

-L *ファイル*

-I *ファイル*

--files-from= *ファイル*

格納，抽出，表示する対象パス名の一覧を指定ファイルから読み込みます。1行に1つの対象パス名を正確に記載します。スペース，空行もパス名とみなされます。

このオプションで複数指定した場合，どの指定も有効となります。

また，短いオプション-T，-L，-Iオプションはどの指定も同じ動作となります。

-xまたは-tオプションと組み合わせた場合，対象パス名は，ワイルドカードの組み合わせで指定できます。

ワイルドカードで使用する文字そのものを指定するために，¥でエスケープすることもでき，¥の指定は¥でエスケープする必要があります。

ワイルドカードとして使用できる文字を次に示します。

ワイルドカード	意味
?	任意の 1 文字に合致します。
*	0 文字以上の文字列に合致します。
[…]	[ ]に囲まれた文字列のどれか 1 文字に合致します。[ ]に囲まれた文字列の先頭に!または^を付けた場合，[ ]に囲まれていない文字に合致します。-（ハイフン）で区切るとハイフンで区切られた，間にある任意の文字（その 2 文字も含む）に合致します。

ワイルドカード[ ]の記述例を次の表に示します。

記述例	意味
[!abc]	a，b，c の 3 つを除く文字と合致します。
[0-9]	0 から9 までのどれかに合致します。
[a-z]	英小文字に合致します。
[A-Z]	英大文字に合致します。
[0-9a-zA-Z]	英数字に合致します。

-X *ファイル*

--exclude-from=*ファイル*

格納，抽出，表示を除外する対象パス名の一覧を指定ファイルから読み込みます。1 行に 1 つの対象パス名を記載します。対象パスは部分的なディレクトリパスの記載でも除外する対象となります。

スペース，空行もパス名とみなされます。

このオプションで指定した除外するファイル一覧は，-T，-L，-I，--files-from オプションで指定したファイル一覧より優先されます。

対象パス名は，ワイルドカードの組み合わせで指定できます。ワイルドカードで使用する文字そのものを指定するために，¥でエスケープすることもでき，¥の指定は¥でエスケープする必要があります。

ワイルドカードとして使用できる文字を次に示します。

ワイルドカード	意味
?	任意の 1 文字に合致します。
*	0 文字以上の文字列に合致します。
[…]	[ ]に囲まれた文字列のどれか 1 文字に合致します。[ ]に囲まれた文字列の先頭に!または^を付けた場合，[ ]に囲まれていない文字に合致します。-（ハイフン）で区切るとハイフンで区切られた，間にある任意の文字（その 2 文字も含む）に合致します。

ワイルドカード[ ]の記述例を次に示します。

記述例	意味
[!abc]	a，b，c の 3 つを除く文字と合致します。
[0-9]	0 から9 までのどれかに合致します。
[a-z]	英小文字に合致します。
[A-Z]	英大文字に合致します。
[0-9a-zA-Z]	英数字に合致します。

対象パス名

格納，抽出，表示するファイルまたはディレクトリのパス名を指定します。

格納時，対象パス名とアーカイブが同一である場合，格納されません。

対象パス名の長さの制限に関しては--compatible=long-file-name オプションを参照してください。

-x または-t オプションと組み合わせた場合，対象パス名は，ワイルドカードの組み合わせで指定することが出来ます。

ワイルドカードで使用する文字そのものを指定するために，¥でエスケープすることもでき，¥の指定は¥でエスケープする必要があります。

ワイルドカードとして使用できる文字を次に示します。

ワイルドカード	意味
?	任意の 1 文字に合致します。
*	0 文字以上の文字列に合致します。
[…]	[ ]に囲まれた文字列のどれか 1 文字に合致します。[ ]に囲まれた文字列の先頭に!または^を付けた場合，[ ]に囲まれていない文字に合致します。-（ハイフン）で区切るとハイフンで区切られた，間にある任意の文字（その 2 文字も含む）に合致します。

ワイルドカード[ ]の記述例を次に示します。

記述例	意味
[!abc]	a，b，c の 3 つを除く文字と合致します。
[0-9]	0 から9 までのどれかに合致します。
[a-z]	英小文字に合致します。
[A-Z]	英大文字に合致します。
[0-9a-zA-Z]	英数字に合致します。

格納，抽出する対象パス名のルートディレクトリの扱い

コマンドは，格納，抽出の操作をする際，対象パス名のルートディレクトリを取り除いて動作します。該当するルートディレクトリは次の表のとおりです。-P オプションを使用することで，対象パス名のルートディレクトリは取り除かないで動作することができます。

また，環境変数ADSH_CMDTAR_ROOTPATH を指定した場合，コマンドのデフォルト動作を「対象パス名のルートディレクトリは取り除く」から「対象パス名のルートディレクトリは取り除かない」に変更できます。

表 8-19　格納，および抽出で取り除かれる対象パス名のルートディレクトリ

OS	説明
UNIX	対象パス名を格納，抽出する際，先頭部分の次のルートディレクトリを取り除きます。 / ../
Windows	対象パス名を格納，抽出する際，先頭部分の次のルートディレクトリを取り除きます。 ¥ ..¥ *ドライブレター*:¥

- 環境変数ADSH_CMDTAR_ROOTPATH 指定なし

  コマンドのデフォルト動作はルートディレクトリを取り除いて格納，抽出します。ルートディレクトリを取り除きたくない場合は-P オプションを指定して実行します。

- 環境変数ADSH_CMDTAR_ROOTPATH 指定あり

  ADSH_CMDTAR_ROOTPATH=absolute を設定することで，コマンドのデフォルト動作はルートディレクトリを取り除かないで格納，抽出します。

ユーザー名，ユーザー ID，グループ名，グループ ID の扱い

アーカイブの作成，抽出，および表示でのユーザー，グループの扱いは次のとおりです。マルチプラットフォーム間でアーカイブを操作した場合，ユーザー，グループが異なることがあります。

アーカイブの状態	ユーザー ID/名	グループ ID/名
アーカイブ作成時	UNIX 該当ファイルの*ユーザー ID*，*ユーザー名*を格納。 Windows *ユーザー ID* は0 と，該当ファイルの*ユーザー名*を格納。	UNIX 該当ファイルの*グループ ID*，*グループ名*を格納。 Windows *グループ ID* は0，*グループ名*は設定しない。
アーカイブ抽出時	UNIX ・スーパーユーザー 抽出した*ユーザー名*から*ユーザー ID* を求めてファイルに設定する。求められない場合は抽出した*ユーザー ID* を設定。 ・一般ユーザー *実行ユーザー ID* または*実行ユーザー名*で設定。 Windows ログオンユーザー名。	UNIX ・スーパーユーザー 抽出した*グループ名*から*グループ ID* を求めてファイルに設定する。求められない場合は抽出した*グループ ID* を設定。 ・一般ユーザー *実行ユーザー名*に属する*グループ ID* または*グループ名*で設定。 Windows 該当なし。

アーカイブの状態	ユーザー ID/名	グループ ID/名
アーカイブ表示時	UNIX 抽出したユーザー名からユーザー *ID* を求めて表示する。求められない場合は，抽出したユーザー *ID* を表示。 Windows 抽出したユーザー *ID* を表示。	UNIX 抽出したグループ名からグループ *ID* を求めて表示する。求められない場合は，抽出したグループ *ID* を表示。 Windows 抽出したグループ *ID* を表示。

一時ファイルディレクトリ

このコマンドは，一時ファイルを作成することがあります。一時ファイルディレクトリは次のとおりです。

Windows の場合

共通アプリケーションフォルダ¥HITACHI¥JP1AS¥misc

UNIX の場合

環境変数TMPDIR に定義されたディレクトリ

環境変数TMPDIR が定義されていないときは，/var/tmp

終了コード

終了コード	意味
0	正常終了。
1	エラー終了。2 のケースを除く。
2	エラー終了。不正なオプションを指定。

注意事項

- 「-」なしショートオプションの形式で指定する場合，オプションとオペランドは「形式」で示した順で指定する必要があります。
- シンボリックリンクの場合は最終修正日時・最終アクセス日時を保持しません。
- 対象パス名をアーカイブに格納している途中にtar コマンドを中断すると，次の名前の一時ファイルが残ることがあります。この場合は手動で一時ファイルを削除する必要があります。

  【Windows の場合】

  tar.*XXXXXX*（*XXXXXX* は任意の 6 文字の文字列）

  【UNIX の場合】

  tar*ppppp*.*XXXXXXXX*（*ppppp* は 5 桁以上のプロセス ID，*XXXXXXXX* は任意の 8 文字の文字列）
- 抽出で未知の種別のファイルを検出した場合，通常ファイルとして処理を試みます。

Windows の場合，ブロックデバイスファイル，キャラクタデバイスファイル，パイプファイルの抽出は未知のファイル種別として判断されます。

(例)

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar xvf PIPE.tar
tar: PIPE/pp creating as regular file.
PIPE/pp
```

- アーカイブに同じ名前のパス名が複数存在する場合，最後のパス名でそれ以前のファイルを上書きします。
- Windows の UNC パスの格納，抽出はサポートしていません。
- Windows の場合，抽出するファイル名が大文字と小文字の違いだけのファイル名は同一ファイルとして扱われます。
- Windows の場合，ショートネーム指定は，ロングネームで格納，抽出および表示をします。ただし，シンボリックリンク先は指定されたファイル名で格納，抽出および表示をします。
- Windows の場合，ディレクトリ区切り文字を¥で使用している場合，格納，抽出は「/」に変換統一されます。
- Windows の場合，ジャンクションは実ファイルとして処理されます。
- Windows の場合，シンボリックリンクの抽出はユーザーにシンボリックリンク作成の権限が必要です。
- Windows の場合，NTFS 以外のファイルシステムにリンクファイルは抽出できません。
- Windows の場合，シンボリックリンク先が存在しない状態ではシンボリックリンクを抽出できません。
- UNIX で¥を含むファイルを格納し，Windows で抽出をする場合，¥はディレクトリ区切り文字として扱われます。
- Windows で格納し，UNIX で抽出する場合，Windows 固有のパス指定はディレクトリ階層が格納時と抽出時で異なることがあります。

(例)

UNIX ではドライブレターはディレクトリの一部と判断されます。

ドライブレターを使用した例を次に示します。

```
Windowsで格納
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar cvfP reg.tar G:¥DIR¥regfile
G:/DIR/regfile

UNIXで抽出
$ tar xvf reg.tar
G:/DIR/regfile
$ ls -lR ./G:
total 8
drwxrwxr-x  2 USER1  GROUP1  4096 Oct 30 15:59 DIR

./G:/DIR:
```

```
total 8
-rw-rw-r--  1 USER1  GROUP1  3 Oct 30 15:59 regfile
```

UNIX 上でドライブレターはディレクトリとして作成されます。

- UNIX の場合，スペシャルファイルの抽出は，スペシャルファイルの作成権限を持つユーザーで行ってください。
- このコマンドは，このコマンド自身が作成したアーカイブの操作だけをサポートします。アーカイブの形式の違いによって，各 OS 提供のtar コマンドと JP1/AS のtar コマンド間でアーカイブの操作をした場合，正しく格納，抽出および表示できないことがあります。
- 格納時，抽出時，表示時でエンコーディングが異なる場合，ファイル名が正しく抽出，表示できない場合があります。
- 8GB 以上のファイルをアーカイブに格納，抽出および表示はサポートしていません。
- UNIX の場合，2097152 以上のユーザー ID，グループ ID，メジャー番号，マイナー番号はサポートしていません。
- ユーザー名，グループ名が 32 文字以上のファイルはサポートしていません。

使用例

例 1　アーカイブの作成

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar cvf ar.tar ar
ar
ar/file1
ar/file2
```

例 2　アーカイブの抽出

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar xvf ar.tar ar
ar
ar¥file1
ar¥file2
```

例 3　アーカイブの表示

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar tvf ar.tar
drwx------ 1234/5678           0 Oct 30 15:52:24 2015 ar
-rw------- 1234/5678           0 Oct 30 15:52:24 2015 ar/file1
-rw------- 1234/5678           0 Oct 30 15:52:24 2015 ar/file2
```

例 4　アーカイブの追加格納（-r オプション時）

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar rvf ar.tar ar
tar: Reading archive to position at the end...done.
ar
ar/file1
ar/file2
```

例５　アーカイブの追加格納(-u オプション時)

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar uvf ar.tar ar
tar: Reading archive to position at the end...done.
ar/file2
```

ar/file2 はar.tar にすでに格納されていますが，格納するar/file2 はar.tar のファイルより新しい時刻のため格納されます。

例６ クロスプラットフォームでの格納と抽出

```
Windowsで格納
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar cvf reg.tar .¥regfile
./regfile

UNIXで抽出
$ tar xvf reg.tar
./regfile
```

例７　アーカイブ先を標準入出力にする

```
$ tar cvf - regfile > reg.tar
regfile
$ cat reg.tar | ./tar tvf -
-rw------- 1234/5678           5 Feb 17 16:47:28 2015 regfile
```

例８　Windows 固有のパス名で格納して UNIX で抽出する

```
Windowsで格納
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar cvfP reg.tar G:¥DIR¥regfile
G:/DIR/regfile

UNIXで抽出
$ ./tar xvf reg.tar
G:/DIR/regfile

$ ls -lR ./G:
total 8
drwxrwxr-x  2 USER1  GROUP1  4096 Oct 30 15:59 DIR

./G:/DIR:
total 8
-rw-rw-r--  1 USER1  GROUP1  3 Oct 30 15:59 regfile
```

UNIX での抽出は，G:もディレクトリとして抽出されます。

例９　スーパーユーザーで抽出

```
$ ./tar tvf reg.tar
-rw-rw-r-- 1234/5678           3 Oct 30 15:59:00 2015 regfile

スーパーユーザーで抽出
# ./tar xvf reg.tar
regfile
```

```
# ls -al regfile
-rw-rw-r--  1 USER1  GROUP1  3 Oct 30 15:59 regfile
```

スーパーユーザーで抽出した場合，ユーザー，グループも抽出し設定されます。

例 10　異なるパス名のファイルを同一の親ディレクトリとして格納

```
DIR1/SUB1/file1
DIR2/SUB1/file2
が存在し， DIR1，DI2と同階層でtarコマンドを実行

$ tar -c -v -f ar.tar -C DIR1 SUB1 -C ../DIR2 SUB1
SUB1
SUB1/file1
SUB1
SUB1/file2
```

例 11　gzip 形式のアーカイブの作成

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar cvzf ar.tar.gz ar
ar
ar/file1
ar/file2
```

例 12　gzip コマンドのオプションを環境変数GZIP に指定して，gzip 形式のアーカイブの作成

```
$ export GZIP=”-9 -q”
$ /opt/jp1as/cmd/tar cvzf ar.tar.gz ar
ar
ar/file1
ar/file2
```

例 13　gzip 形式のアーカイブからディレクトリの抽出

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar xvzf ar.tar.gz ar
ar
ar/file1
ar/file2
```

例 14　gzip 形式のアーカイブの表示

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥tar tvzf ar.tar.gz
drwx------ 0/0                0 Dec 09 19:44:24 2015 ar
-rw------- 0/0             2758 Dec 09 19:39:35 2015 ar/file1
-rw------- 0/0             5516 Dec 09 19:39:55 2015 ar/file2
```

8.4.35 touch コマンド（ファイルの最終アクセス日時と最終修正日時を変更する）

形式

```
touch [-a] [-c] [-f] [-m] [-r パス名] [-t 設定日時] パス名 ...
touch [-a] [-c] [-f] [-m] 設定日時 パス名 ...
```

機能

指定したファイルの最終アクセス日時と最終修正日時を変更します。Windows の場合はファイルの最終修正日時だけを変更します。

引数

変更する日時の種類

変更する日時の種類は-a オプションまたは-m オプションで指定します。-a オプションと-m オプションの両方とも指定していない場合，または，-a オプションおよび-m オプションの両方とも指定した場合は，ファイルの最終アクセス日時と最終修正日時の両方を変更します。ただし，Windows の場合は，実際にはファイルの最終アクセス日時は変更されません。

-a

ファイルの最終アクセス日時を変更します。

-m オプションを指定しないで-a オプションを指定した場合は，ファイルの最終アクセス日時だけを変更し，ファイルの最終修正日時は変更しません。

Windows の場合，-a オプションを指定しても実際にはファイルの最終アクセス日時は変更されませんが，引数に指定した設定日時の形式チェックと，-r オプションに指定したファイルの読み込みは実行されます。

-m

ファイルの最終修正日時を変更します。

-a オプションを指定しないで-m オプションを指定した場合は，ファイルの最終修正日時だけを変更し，ファイルの最終アクセス日時は変更しません。

設定する日時の指定

設定する日時は，-r オプション，-t オプションまたは MMDDhhmm[YY]形式の設定日時で設定します。これらのオプションの指定がない場合は，コマンドの実行日時が設定されます。

-t オプションおよび MMDDhhmm[YY]形式の設定日時に指定できる日時の範囲は，UTC（協定世界時）の 1970 年 1 月 1 日 0 時 0 分 0 秒～2038 年 1 月 19 日 3 時 14 分 7 秒です。指定した日時は，コマンド実行時のタイムゾーンによって解釈されます。

タイムゾーンが日本標準時（UTC+9）の場合に指定できる日時の範囲は，1970 年 1 月 1 日 9 時 0 分 0 秒～2038 年 1 月 19 日 12 時 14 分 7 秒です。ただし，AIX，Windows の場合，タイムゾーンが日本標準時（UTC+9）のときに指定できる日時の上限は 2038 年 1 月 19 日 3 時 14 分 7 秒です。

タイムゾーンは環境変数 TZ に設定されている値が使用されます。Windows で環境変数 TZ が設定されていない場合は，コントロールパネルの「日付と時刻のプロパティ」ダイアログボックスに設定されているタイムゾーンが使用されます。なお，Windows の場合，環境変数 TZ に設定したタイムゾーンと「日付と時刻のプロパティ」に設定したタイムゾーンを同じ値にする必要があります。

-r パス名

ファイルに設定する最終アクセス日時および最終修正日時をここで指定したファイルパスから取得します。取得したファイルの最終日時は，-a オプションが指定されている場合は最終アクセス日時，-m オプションが指定されている場合は最終修正日時として設定されます。

ディレクトリ名を指定すると，指定したディレクトリから最終アクセス日時，および最終修正日時を取得します。

オプションを複数回指定した場合，最後に指定した値が有効になります。

-r オプションと-t オプションを両方指定した場合は，最後に指定されたオプションが有効になります。

-t 設定日時

ファイルに設定する最終アクセス日時，最終修正日時に設定する日時を指定します。指定した日時は，-a オプションが指定されている場合は最終アクセス日時，-m オプションが指定されている場合は最終修正日時として設定されます。

オプションを複数回指定した場合，最後に指定した値が有効になります。

-r オプションと-t オプションを両方指定した場合は，最後に指定されたオプションが有効になります。

設定日時は次の形式で指定します。

```
[[CC]YY]MMDDhhmm[.SS]
```

CC

西暦年の上 2 桁を指定します。

YY

西暦年の下 2 桁を指定します。

CC を省略すると，CC には次の値が設定されます。

YY が 69～99 の場合，CC には 19 が設定されます。

YY が 00～68 の場合，CC には 20 が設定されます。

なお，CC と YY の両方の指定を省略した場合は，コマンド実行日時の西暦年が設定されます。

MM

月を 01～12 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

DD

日を 01～31 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

hh

時を 00～23 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

mm

分を 00～59 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

SS

秒を 00～61 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。秒の指定を省略した場合は 00 が設定されます。

なお，60～61 の値を指定して設定した場合，うるう秒への対応をしていないシステムでは，ls コマンドなどで日時を表示すると 60 のときは 1 秒，61 のときは 2 秒繰り上げられた日時が表示されます。

設定日時

ファイルの最終アクセス日時，最終修正日時に設定する日時を指定します。指定した日時は，-a オプションが指定されている場合は最終アクセス日時，-m オプションが指定されている場合は最終修正日時として設定されます。

-r オプションまたは-t オプションを指定した場合は，ファイル名として扱われます。

指定した日時の桁数が 8 桁と 10 桁のどちらでもない場合はファイル名として扱われ，ファイルが存在しない場合はファイルが作成されます。

設定日時は次の形式で指定します。

```
MMDDhhmm[YY]
```

MM

月を 01～12 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

DD

日を 01～31 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

hh

時を 00～23 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

mm

分を 00～59 の範囲の数値で指定します。1 桁の数値を指定する場合は先頭に 0 を指定してください。

YY

西暦年の下 2 桁を指定します。指定を省略した場合はコマンド実行日時の西暦年が設定されます。

なお，西暦年の上 2 桁には次の値が設定されます。

YY が 69～99 の場合：19

YY が 00～68 の場合：20

その他のオプション

パス名

最終アクセス日時，最終修正日時を変更するファイルのパス名を指定します。パス名は複数指定できます。

指定したパスが存在しない場合は 0 バイトのファイルを新規に作成します。

Windows の場合

ディレクトリの最終修正日時は変更できません。ディレクトリ名を指定するとエラーとなります。また，存在するファイルに対して最終修正日時を変更する場合は，読み取り権限と書き込み権限が必要です。

UNIX の場合

新規に作成したファイルのパーミッションは，umask に従って設定されます。ディレクトリ名を指定すると，ディレクトリの最終アクセス日時，最終修正日時が変更されます。

また，スーパーユーザー以外のユーザーが，存在するファイルに対して最終アクセス日時や最終修正日時を変更する場合は，次の権限が必要です。

・-t オプション，または MMDDhhmm[YY]形式の設定日時の指定あり

ファイルの所有者の権限が必要です。

・-t オプション，および MMDDhhmm[YY]形式の設定日時の指定なし

ファイルに対する書き込み権限が必要です。

-c

最終アクセス日時，最終修正日時を変更するファイルが存在しなかった場合，ファイルを作成しません。また，エラーメッセージは出力しません（エラー扱いとはしません）。

-f

OS が提供する touch コマンドとの互換のためのオプションです。指定しても無視されます。

ファイルに設定する最終アクセス日時と最終修正日時

日時を設定する-r オプション，-t オプション，または MMDDhhmm[YY]形式の設定日時と，変更する日時の種類を指定する-a オプションまたは-m オプションの指定によって，ファイルの最終アクセス日時と最終修正日時は次のように設定されます。

-r オプションを指定した場合

-r オプションに指定したファイルから取得した最終アクセス日時と最終修正日時は，パス名に指定したファイルの有無によって，次の表のように設定されます。

表 8-20　-r オプションを指定した場合に設定されるファイルの最終アクセス日時，最終修正日時

-a オプションと-m オプションの指定	パス名に指定されたファイルの有無	設定される最終アクセス日時		設定される最終修正日時
		Windows	UNIX	
-a だけを指定	存在する	－	◎	－
	存在しない	○	◎	○

-a オプションと-m オプションの指定	パス名に指定されたファイルの有無	設定される最終アクセス日時		設定される最終修正日時
		Windows	UNIX	
-m だけを指定	存在する	－		◎
	存在しない	○		◎
-a，-m の両方を指定， または -a，-m とも指定なし	存在する	－	◎	◎
	存在しない	○	◎	◎

（凡例）

◎：-r オプションに指定したファイルの対応する最終アクセス日時および最終修正日時が設定されます。

○：コマンドの実行日時が設定されます。

－：コマンド実行前のファイルの最終アクセス日時および最終修正日時がそのまま適用されます。

-t オプションまたは MMDDhhmm[YY]形式の設定日時を指定した場合

-t オプションまたは MMDDhhmm[YY]形式の設定日時で指定した日時は，**パス名**に指定したファイルの有無によって，次の表のように設定されます。

表 8-21　引数に設定日時を指定した場合に設定されるファイルの最終アクセス日時，最終修正日時

-a オプションと-m オプションの指定	パス名に指定されたファイルの有無	設定されるファイルの最終アクセス日時		設定されるファイルの最終修正日時
		Windows	UNIX	
-a だけを指定	存在する	－	◎	－
	存在しない	○	◎	○
-m だけを指定	存在する	－		◎
	存在しない	○		◎
-a，-m の両方を指定， または -a，-m とも指定なし	存在する	－	◎	◎
	存在しない	○	◎	◎

（凡例）

◎：-t オプションに指定した設定日時，または MMDDhhmm[YY]形式の設定日時が設定されます。

○：コマンドの実行日時が設定されます。

－：コマンド実行前のファイルの最終アクセス日時および最終修正日時がそのまま適用されます。

-r オプション，-t オプションおよび MMDDhhmm[YY]形式の設定日時を指定しないとき

-r オプション，-t オプションおよび MMDDhhmm[YY]形式の設定日時が指定されていない場合，**パス名**に指定したファイルの有無によって，次の表のように設定されます。

表 8-22　設定日時の取得および指定がない場合に設定されるファイルの最終アクセス日時，最終修正日時

-a オプションと-m オプションの指定	パス名に指定されたファイルの有無	設定されるファイルの最終アクセス日時		設定されるファイルの最終修正日時
		Windows	UNIX	
-a オプションだけを指定	存在する	－	○	－
	存在しない	○		○
-m オプションだけを指定	存在する	－		○
	存在しない	○		○
-a，-m の両方を指定， または -a，-m とも指定なし	存在する	－	○	○
	存在しない	○		○

（凡例）

○：コマンドを実行した日時

－：コマンド実行前のファイルの最終アクセス日時および最終修正日時がそのまま適用されます。

終了コード

終了コード	意味
0	正常終了。
1	エラー終了。 • 不正なオプションを指定しました。 • -t オプションに指定した設定日時，MMDDhhmm[YY]形式の設定日時が正しくありません。 • -r オプションに指定したファイルの読み込みに失敗しました。
2	エラー終了。 • ファイルの作成に失敗しました。 • ファイルの最終アクセス日時，最終修正日時の変更に失敗しました。 • 最終修正日時を変更するファイルのパス名にディレクトリを指定しました。【Windows 限定】 引数に複数のファイルが指定されているときは，後続のファイルを処理します。

注意事項

- UNIX の場合，-r オプションに指定するパス名，ファイルの最終アクセス日時と最終修正日時を変更するパス名にシンボリックリンク名を指定した場合，シンボリックリンク先のパスが対象となります。

  Windows の場合，-r オプションに指定するパス名，ファイルの最終修正日時を変更するパス名にシンボリックリンク名を指定した場合，シンボリックリンク先のパスが対象となります。

- -r オプションに指定したファイルの最終アクセス日時，最終修正日時が UTC（協定世界時）の 1970 年 1 月 1 日 0 時 0 分～2038 年 1 月 19 日 3 時 14 分の範囲外の場合，ファイルに設定される最終アクセス日時，最終修正日時の値は保証されません。

- Windows の場合，実際にファイルに設定される最終修正日時の精度はファイルシステムの仕様に依存します。例えば，FAT ファイルシステム上のファイルへの最終アクセス日時，最終修正日時は次のように設定されます。
  - 指定できる日時の範囲は，コマンド実行時のタイムゾーンに関係なく 1980 年 1 月 1 日 0 時 0 分 0 秒～2038 年 1 月 19 日 3 時 14 分 7 秒です。
  - ファイルの最終アクセス日時の時分秒（hhmm.ss）は設定されません。
  - ファイルの最終修正日時に指定した秒（ss）は 2 秒単位に繰り上げられて設定されます。

使用例

- ファイルを新規に作成します。

```
$ touch file001
```

  この場合，作成されたファイルの最終アクセス日時と最終修正日時は，コマンドを実行した日時となります。また，ファイルサイズは 0 で作成されます。

- 存在するファイルの最終アクセス日時と最終修正日時を-t オプションで指定した日時（2012 年 5 月 12 日 3 時 49 分 5 秒）に変更します。

```
$ touch -t 1205120349.05 file001
```

  コマンドを実行すると，ファイルの最終アクセス日時と最終修正日時は-t オプションで指定した日時となります。

- 存在するファイルの最終アクセス日時だけを-t オプションで指定した日時（2013 年 11 月 01 日 15 時 08 分 0 秒）に変更します。

```
$ touch -a -t 201311011508 file001
```

  コマンドを実行すると，ファイルの最終アクセス日時は-t オプションで指定した日時になります。ファイルの最終修正日時はコマンド実行前のままです。

- 存在するファイルの最終修正日時だけを MMDDhhmm[YY]形式の設定日時で指定した日時（2013 年 9 月 29 日 23 時 00 分 0 秒）に変更します。

```
$ touch -m 0929230013 file001
```

  コマンドを実行すると，ファイルの最終修正日時は MMDDhhmm[YY]形式の設定日時で指定した日時になります。ファイルの最終アクセス日時はコマンド実行前のままです。

- 複数のファイルに対して，ファイルの最終アクセス日時と最終修正日時を-t オプションで指定した日時へ変更します。-c オプションを指定することで，存在しないファイルは作成しないこととします。次の例では，存在しないファイルは file002 です。

```
$ touch -c -t 201311011508 file001 file002 file003
$ ls -lT *
-rw-r--r--  1 usr1  grp1  5 Nov  1 15:08:00 2013 file001
-rw-r--r--  1 usr1  rrp1  9 Nov  1 15:08:00 2013 file003
```

8.4.36 tr コマンド（標準入力から入力された文字列を，1 バイトごとに置換または削除しながら標準出力に出力する）

形式

形式 1

```
tr [-cst] [--check-multi-byte] 文字列1 文字列2
```

形式 2

```
tr -d [-c] [--check-multi-byte] 文字列1
```

形式 3

```
tr -s [-c] [--check-multi-byte] 文字列1
```

形式 4

```
tr -ds [-c] [--check-multi-byte] 文字列1 文字列2
```

機能

標準入力から入力された文字列を，1 バイトごとに置換または削除しながら標準出力に出力します。

形式 1 でコマンドが指定された場合，標準入力から入力された文字列を置換して，その結果を標準出力に出力します。置換の際，標準入力の文字列に含まれる*文字列 1* の 1 バイトごとの文字を，*文字列 2* の 1 バイトごとの文字に置換します。*文字列 1* が*文字列 2* よりも長い場合，*文字列 2* の最後の 1 バイトの文字が*文字列 1* と同じ長さになるまで繰り返されているものと見なします。

形式 2 でコマンドが指定された場合，標準入力から入力された文字列に含まれる，*文字列 1* の文字（1 バイト単位）を削除して，その結果を標準出力に出力します。

形式 3 でコマンドが指定された場合，標準入力から入力された文字列を置換して，その結果を標準出力に出力します。置換の際，標準入力の文字列に，*文字列 1* の文字（1 バイト単位）が連続して含まれていた場合に，連続する同じ文字を 1 文字に圧縮します。

形式 4 でコマンドが指定された場合，標準入力から入力された文字列を削除および置換して，その結果を標準出力に出力します。削除の際，標準入力の文字列に含まれる，*文字列 1* の文字（1 バイト単位）を削除します。また，*文字列 1* の文字を削除したあとの文字列に，*文字列 2* の文字（1 バイト単位）が連続して含まれていた場合に，連続する同じ文字を 1 文字に圧縮します。

引数

-c

--complement

文字列 1 を*文字列 1* の補集合（*文字列 1* に含まれない文字すべて）で置換します。

-d

--delete

標準入力の文字列から，*文字列 1* にある入力文字を削除します。

-s

--squeeze-repeats

すべての削除や置換が行われたあとの文字列に含まれる，連続する同じ文字（バイト単位）を 1 文字に圧縮します。置換する文字列は，形式 1 と形式 4 では*文字列 2*，形式 3 では*文字列 1* に指定した文字列（1 バイト単位）です。

-t

--truncate-set1

文字列 1 が*文字列 2* よりも長い場合に，*文字列 1* の文字を削除して，*文字列 2* と同じ長さに切り詰めてから置換します。

--check-multi-byte

文字列 1 または*文字列 2* にマルチバイト文字が含まれている場合，エラーにします。

文字列 1，*文字列 2*

標準入力から入力された文字列に対して，置換または削除する対象の文字列を指定します。文字列は，1 バイト単位で処理されます。

文字列 1，*文字列 2* の文字の集合を指定する場合，次の記述が利用できます。

記述形式	意味	
¥ooo	1～3 桁の 8 進数で表された ASCII コードの文字を指定します。	
文字 1-*文字 2*	範囲指定。文字（1 バイト）は ASCII 照合順序で指定します。	
エスケープ文字	¥a	アラート文字（ベル）
	¥b	バックスペース文字
	¥f	フォームフィード文字（改ページ）
	¥n	改行文字
	¥r	復帰文字
	¥t	タブ文字
	¥v	垂直タブ文字
[:*文字クラス*:]	*文字列 1* および形式 4 の*文字列 2* は以下の文字クラスを指定できます。形式 1 の*文字列 2* は，次の条件をすべて満たしたときに，lower またはupper が指定できます。	

記述形式	意味	
[:*文字クラス*:]	・-c オプションなし ・*文字列 1* の，*文字列 2* と同じ位置にlower またはupper を指定します。	
	alnum	英数字
	alpha	英字
	blank	空白文字(空白とタブ)
	cntrl	制御文字
	digit	数字
	graph	表示可能文字(スペースは含まない)
	lower	英小文字
	print	表示可能文字(スペースを含む)
	punct	句読点文字
	space	空白文字
	upper	英大文字
	xdigit	16 進数での数字
[*#*n*]	*#*で指定した文字（1 バイト）を*n* 個指定したと同じ意味です。形式 1 の*文字列 2* で使用します。*n* を省略した場合または0 の場合，*文字列 2* が*文字列 1* と同じ長さの値として解釈され，後ろに指定した文字は無視されます。*n* が 0 で始まる場合，8 進数として解釈され，その他は 10 進数と解釈されます。	

終了コード

終了コード	意味
0	正常終了。
1 以上	エラー終了。

注意事項

- この機能は 1 バイトごとに，文字を置換または削除するため，*文字列 1*，*文字列 2*，または標準入力から入力された文字列にマルチバイト文字が含まれている場合，予期しない出力結果になることがあります。マルチバイト文字を含む文字列を置換または削除する場合は，sed コマンドを使用してください。
- 入力文字が［CR］＋［LF］の改行の場合，［CR］および［LF］それぞれが処理の対象になります。

使用例

例 1　*文字列 1*，*文字列 2* の指定に従って，標準入力から入力された文字列を置換します。この例では，括弧の種類が異なる文字列を[]に統一しています。

file1 の内容

```
[apple],{banana}
[orange],<peach>,{cherry}
```

コマンドの実行結果

```
$ tr  '{}<>' '[][]' < file1
[apple],[banana]
[orange],[peach],[cherry]
```

例 2　*文字列 1* の補集合を*文字列 2* で置換し，連続している文字を 1 文字に圧縮します。この例では，標準入力から入力された文字列から英文字を洗い出しています。

file1 の内容

```
[apple],{banana}
[orange],<peach>,{cherry}
```

コマンドの実行結果

```
$ tr -s -c '[:alpha:]' '[¥n*]' < file1

apple
banana
orange
peach
cherry
```

例 3　*文字列 1* の指定に従って，標準入力から入力された文字列を削除します。この例では，括弧を削除しています。

file1 の内容

```
[apple],{banana}
[orange],<peach>,{cherry}
```

コマンドの実行結果

```
$ tr -d '{}<>[]' < file1
apple,banana
orange,peach,cherry
```

例 4　*文字列 1*，*文字列 2* の指定に従って，標準入力から入力された文字列を置換し，連続する文字を圧縮します。この例では，括弧を”,”に置き換え,”,”を 1 文字に圧縮しています。

file1 の内容

```
[apple],{banana}
[orange],<peach>,{cherry}
```

コマンドの実行結果

```
$ tr -s '{}<>[]' ',' < file1
,apple,banana,
,orange,peach,cherry,
```

例５　*文字列１*に従って，標準入力から入力された文字列を削除し，*文字列２*に従って連続する文字を圧縮します。この例では，括弧を削除し，”,”を圧縮しています。

file1の内容

```
[apple],{banana}
[orange],<>,{cherry}
```

コマンドの実行結果

```
$ tr -ds '{}<>[]' ',' < file1
apple,banana
orange,cherry
```

例６　*文字列１*を*文字列２*の長さに切り詰め，置換します。この例では，括弧“{”と“}”を”[”と”]”に置換します。”<”と”>”は，置換しません。

file1の内容

```
[apple],{banana}
[orange],<peach>,{cherry}
```

コマンドの実行結果

```
$ tr -t '{}<>' '[]' < file1
[apple],[banana]
[orange],<peach>,[cherry]
```

8.4.37　uname コマンド（OS またはハードウェアの情報を表示する）

形式

```
【Windows限定】uname [-a] [-m] [-n] [-r] [-s] [-v] [-w]
【UNIX限定】   uname [-a] [-m] [-n] [-r] [-s] [-v]
```

機能

OS，システムのホスト名，またはハードウェアの情報を標準出力に出力します。

引数

オプションを指定しないで実行した場合は，-s オプションを指定した場合と同じ動作になります。

-a

【Windows 限定】

-w オプションを指定しない場合は，次の情報を１行にまとめて順番に表示します。

・OS 名（常に「Windows」）

・ノード名

・OS の説明

・OS にインストールされている最新のサービスパック

・OS のバージョン

・マシン（ハードウェア）のタイプ

-w オプションとあわせて指定した場合，次の情報を 1 行ずつ順番に表示します。

・OS 名，その OS が組み込まれているフォルダ，およびその OS が組み込まれているディスクのパーティション情報

・ノード名

・OS のリリース（常に「unknown」）

・OS のバージョン

・マシン（ハードウェア）のタイプ

【UNIX 限定】

次の情報を 1 行にまとめて順番に表示します。

・OS 名

・ノード名

・OS のリリース

・OS のバージョン

・マシン（ハードウェア）のタイプ

-m

マシン（ハードウェア）のタイプを表示します。

-n

ノード名を表示します。

-r

OS のリリースを表示します。Windows の場合は，常に「unknown」と表示します。

-s

OS 名を表示します。

Windows の場合は，OS 名として次の内容を表示します。

- -w オプションを指定しない場合は，常に「Windows」と表示します。
- -w オプションとあわせて指定した場合は，OS 名，その OS が組み込まれているフォルダ，およびその OS が組み込まれているディスクのパーティション情報を表示します。

-v

OS のバージョンを表示します。

-w【Windows 限定】

このオプションを指定すると，uname コマンドが表示する情報が，JP1/Advanced Shell 10-01 以前のバージョンの形式で表示されます。-w オプションを指定した場合の表示内容を次に示します。

- 各オプションの情報を 1 行ごとに表示します。
- -a オプションと-s オプションによって表示される情報が変わります。詳細については各オプションの説明を参照してください。

また，このオプションだけを指定した場合は，-w オプションと-s オプションをあわせて指定した場合と同じ動作となります。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- Windows の場合，ファイルおよび標準入力，標準出力をバイナリモードで入出力します。改行コードは変換しません。
- Windows の場合，次に示すオプション以外のオプションでは，Administrators 権限を利用して情報を取得しているため，Administrators 権限を持つユーザーで使用してください。Administrators 権限を持たないユーザーがそれらのオプションを指定して uname コマンドを実行した場合はエラーになります。
  - -r オプション
  - -s オプション（-w オプションと一緒に指定しない場合）
- Windows の場合，uname コマンドは Windows の OS の機能を使用して OS およびハードウェアの情報を取得しているため，スクリプトを実行している間の PATH 環境変数には，Windows のシステムフォルダのパス情報が含まれている必要があります。そのため，PATH 環境変数に別のパス情報を追加したい場合は，次のように PATH 環境変数の情報を変更してください。

  例

  PATH="${PATH};C:¥¥home¥¥bin"

使用例

- オプションを指定しない場合のデフォルトを表示します。

  Windows の場合

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname
Windows
```

  UNIX の場合（Linux 上でコマンドを実行した場合）

```
$ /opt/jp1as/cmd/uname
Linux
```

- -a オプションを指定して，OS 環境の詳細な情報をすべて表示します。

Windows の場合（-w オプションを指定しない場合）

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -a
Windows MyMachine Microsoft Windows Server 2008 R2 Enterprise Service Pack 1 6.1.7601
x64-based PC
```

Windows の場合（-w オプションをあわせて指定した場合）

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -aw
Microsoft Windows Server 2008 R2 Enterprise|C:¥Windows|¥Device¥Harddisk0¥Partition2
 MyMachine
 unknown
 6.1.7601
 x64-based PC
```

UNIX の場合（Linux 上でコマンドを実行した場合）

```
$ /opt/jp1as/cmd/uname -a
Linux LINUX1 2.6.18-53.el5 #1 SMP Wed Oct 10 16:34:02 EDT 2007 i686
```

- -m オプションを指定して，マシンおよびハードウェアの名称を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -m
x64-based PC
```

- -n オプションを指定して，ノード名を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -n
MyMachine
```

- -r オプションを指定して，OS のリリースを表示します。次の例は Windows の例であり，常に「unknown」が表示されます。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -r
unknown
```

- -s オプションを表示して，OS 名を表示します。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -s
Windows
```

- -v オプションを指定して，OS のバージョンを表示します。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -v
6.1.7601
```

- 複数のオプションを組み合わせた場合は，-a オプションの表示順に従って，該当する情報を表示します。Windows の例を次に示します。この例では-v，-s の順にオプションを指定していますが，この場合は-s，-v の順で情報が表示されます。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -v -s
Windows 6.1.7601
```

- オプションエラーのメッセージを表示します。

このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windowsの例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -p
uname: illegal option -- p
usage: uname [-amnrsvw]
```

- Windowsの場合，Administrators権限を持たないユーザーで，Administrator権限が必要となるオプションを指定してunameコマンドを実行した場合は，次のようにエラーメッセージが出力されます。次の例は，-mオプションを指定した場合の表示内容です。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -m
mof ファイルを登録できませんでした。
WMIC.EXE を使用できるのは管理者グループのメンバだけです。
理由:Win32 エラー: アクセスが拒否されました。

unknown
```

8.4.38 uniqコマンド（ソートされたファイルから重複した行を削除する）

形式

```
uniq [-c] [-d] [-u] [入力パス名 [出力パス名] ]
```

機能

ファイル中の重複した行を1行にして出力します。なお，同一内容の行が連続している場合だけ重複行と見なします。

引数

オプションを省略した場合，-dオプションおよび-uオプションを指定したときと同じ動作となります。重複している行は1行だけ出力し，重複していない行はすべて出力します。

-c

各出力行の先頭に，同一行が続けて出現した回数およびスペースを1つ出力します。回数は4桁で表示します。4桁で表示できない場合は，順次桁数を増やします。回数の後ろには1文字のスペースを表示します。

-d

重複している行を1行だけ表示します。

-u

重複していない行を出力します。

入力パス名

入力対象のファイルを指定します。入力パス名を指定しない，または「-」を指定した場合は標準入力から入力します。

出力パス名

結果を出力するファイルを指定します。出力パス名を指定しない，または「-」を指定した場合は標準出力に出力します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 入力パス名と出力パス名に同じファイルを指定すると，ファイルは空になります。
- 比較できる 1 行の最大バイト数は 8192 バイトです。
- バイナリファイルからの入力およびバイナリデータの出力は，動作を保証しません。

使用例

uniq コマンドを実行した結果表示に使用するファイルの形式を次に示します。

- file1.txt

```
aaaa
aaaaaaa                         →重複している文字列
aaaaaaa                         →重複している文字列
bbbbbbb
bbbbbbbbbbb                     →重複している文字列
bbbbbbbbbbb                     →重複している文字列
bbbbbbbbbbb                     →重複している文字列
bbbbbbbbbbb                     →重複している文字列
bcbcbcbcbcb
ddddddddddddddddddd
dddddddddddddddddddddd
ddddddddddddddddddddddddd
dddddddddddddddddddddddddeee    →重複している文字列
ddddddddddddddddddddddddddeee   →重複している文字列
```

コマンドの実行例を次に示します。

- オプションを指定しない場合のデフォルトを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uniq file1.txt
aaaa
aaaaaaa
bbbbbbb
```

```
bbbbbbbbbbb
bcbcbcbcbcb
dddddddddddddddddddd
ddddddddddddddddddddddd
dddddddddddddddddddddddd
ddddddddddddddddddddddddeee

C:¥TEMP>
```

- -c オプションを指定して，同一行が続けて出現した回数と該当する行の内容を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uniq -c file1.txt
   1 aaaa
   2 aaaaaaa
   1 bbbbbbb
   4 bbbbbbbbbbb
   1 bcbcbcbcbcb
   1 ddddddddddddddddddd
   1 dddddddddddddddddddddd
   1 ddddddddddddddddddddddd
   2 ddddddddddddddddddddddddeee

C:¥TEMP>
```

- -d オプションを指定して，重複している行だけを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uniq -d file1.txt
aaaaaaa
bbbbbbbbbbb
ddddddddddddddddddddddddeee

C:¥TEMP>
```

- -u オプションを指定して，重複しない行だけを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uniq -u file1.txt
aaaa
bbbbbbb
bcbcbcbcbcb
ddddddddddddddddddd
ddddddddddddddddddddd
dddddddddddddddddddddd

C:¥TEMP>
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uniq -w
uniq: illegal option -- w
usage: uniq [-cdu] [input_file [output_file]]
```

8.4.39 wc コマンド（ファイルのバイト，行，文字および単語をカウントする）

形式

```
wc [-c] [-l] [-m] [-w] [パス名 ...]
```

機能

ファイルのバイト，行，文字または単語をカウントします。入力ファイルの行数・単語数・文字数・バイト数・ファイル名の順序に，オプションに指定された情報だけ表示します。

引数

-c
入力ファイルのバイト数を標準出力に出力します。

-l
入力ファイルの行数を標準出力に出力します。改行コードの数を行数とします。

-m
入力ファイルの文字数を標準出力に出力します。マルチバイト文字も 1 文字としてカウントします。

-w
入力ファイルの単語数を標準出力に出力します。単語は，スペース，タブおよび改行で区切られた文字列の数とします。

パス名
入力対象とするファイル名を指定します。パス名を指定しない，または「-」を指定した場合は標準入力から入力します。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- ロケールと異なる文字コードの文字は，無効または不完全な文字と見なされます。
- -c オプション，-l オプション，-m オプション，-w オプションのどれも指定しなかった場合，-c オプション，-l オプション，-w オプションが指定されたものとします。

- オプションの指定順序に関係なく，行数，単語数，マルチバイトの文字数，バイト数，ファイル名の順序で表示します。数値は 1 文字のスペースと 7 桁で表示します。7 桁で表示できない場合は，順次桁数を増やします。
- 無効，不完全なマルチバイト，ワイド文字，バイナリデータ，ロケールと異なる文字コードが含まれるファイルを入力するとエラーとなります（wc: binaryfile: Invalid or incomplete multibyte or wide character）。

使用例

- オプションを指定しない場合のデフォルトを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc a.txt b.txt
       5       5      55 a.txt
       4       4      44 b.txt
       9       9      99 total
```

- -c オプションを指定して，入力ファイルのバイト数を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -c a.txt
      55 a.txt
```

- -l オプションを指定して，入力ファイルの行数を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -l a.txt
       5 a.txt
```

- -m オプションを指定して，入力ファイルの文字数を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -m a.txt
      50 a.txt
```

- -w オプションを指定して，入力ファイルの単語数を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -w a.txt
       5 a.txt
```

- すべてのオプションを指定して，入力ファイルの行数，単語数，文字数およびバイト数を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -clmw a.txt
       5       5      50      55 a.txt
```

- オプションエラーのメッセージを表示します。

  このメッセージは，コマンドを実行するプラットフォームによって異なる場合があります。Windows の例を次に示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -z
wc: illegal option -- z
usage: wc [-clmw] [file ...]
```

- ファイル内に無効または不完全な文字がある場合にエラーメッセージを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc binaryfile
wc: binaryfile: Invalid or incomplete multibyte or wide character
```

無効または不完全な文字として，次のものがあります。

- 無効または不完全なマルチバイト，ワイド文字，バイナリデータ
- ロケールと異なる文字コードの文字

8.4.40 which コマンド（外部コマンドのパスを取得する）

形式

```
which [-a] コマンド名 ...
```

機能

環境変数PATH に設定されているコマンド検索パスから，実行する外部コマンドのコマンドパスを取得します。取得したコマンドパスは標準出力に出力します。

引数

-a

環境変数PATH に設定されているコマンド検索パスの中から，実行できるすべてのコマンドパスを取得して出力します。

-a オプションを指定しない場合は，最初に取得されたコマンドパスだけを出力します。

コマンド名

コマンドパスを取得したい外部コマンド名を指定します。複数指定できます。

指定した外部コマンドのコマンドパスが見つからなかった場合は，標準エラー出力に外部コマンドのコマンドパスが見つからなかったことを示すメッセージが出力されます。

コマンドパスの検索規則

外部コマンドのコマンドパスは次の規則で検索します。

Windows の場合

外部コマンドの検索対象パス

環境変数PATH に設定されているコマンド検索パスに対して，外部コマンドを検索します。環境変数PATH に複数のコマンドパスが設定されている場合は，先頭のコマンドパスから順番に検索します。なお，which コマンドの実行者に外部コマンド格納ディレクトリの読み込み権限がない場合，そのディレクトリはコマンドパスの検索対象にはなりません。

コマンドパスの出力対象となる外部コマンド

which コマンドの実行者に外部コマンド格納ディレクトリの読み込み権限があればコマンドパスを出力します。外部コマンドの実行権限の有無は判断されません。

コマンドパスの出力対象となる外部コマンドは拡張子が「.com」,「.exe」,「.cmd」,「.bat」の実行ファイルです。

指定した外部コマンドに拡張子が含まれない場合，外部コマンド名に環境変数PATHEXT に定義されている順に拡張子を付けて外部コマンドを検索します。対象となる拡張子は「.com」,「.exe」,「.cmd」,「.bat」です。詳細については，「5.1.11　外部コマンドの指定」を参照してください。

UNIX の場合

外部コマンドの検索対象パス

環境変数PATH に設定されているコマンド検索パスに対して外部コマンドを検索します。環境変数PATH に複数のコマンドパスが設定されている場合は，先頭のコマンドパスから順に検索します。なお，which コマンドの実行者に外部コマンド格納ディレクトリ（パスを構成するすべてのディレクトリ）の検索権限がない場合，そのディレクトリはコマンドパスの検索対象にはなりません。

コマンドパスの出力対象となる外部コマンド

which コマンドの実行者に外部コマンドの実行権限があれば，実行できる外部コマンドと判断し，コマンドパスを出力します。外部コマンドの実行権限がない場合，その外部コマンドはコマンドパスの出力対象にはなりません。

引数に指定したコマンド名にパスが含まれている場合

Windows の場合

which コマンドの実行者に外部コマンド格納ディレクトリの読み込み権限があればコマンドパス名を出力します。外部コマンドの実行権限の有無は判断されません。

外部コマンド格納ディレクトリの読み込み権限がない場合は外部コマンドのコマンドパスが見つからなかったことを示すメッセージを出力します。

コマンドパスの出力対象となる外部コマンドは拡張子が「.com」,「.exe」,「.cmd」,「.bat」の実行ファイルです。

指定した外部コマンドに拡張子が含まれない場合は，外部コマンド名に環境変数PATHEXT に定義されている順に拡張子を付けます。対象となる拡張子は「.com」,「.exe」,「.cmd」,「.bat」です。

指定した外部コマンドがシンボリックリンクの場合は，次のように動作します。

- シンボリックリンクとリンク先ファイルの両方の拡張子が「.com」,「.exe」,「.cmd」,「.bat」であることで実行権限があると判断します。
- シンボリックリンクに拡張子が含まれていない場合は，環境変数PATHEXT に定義されている順に拡張子を付けます。ただし，リンク先のファイルは拡張子の付加の対象外です。対象となる拡張子は「.com」,「.exe」,「.cmd」,「.bat」です。

UNIX の場合

which コマンドの実行者に外部コマンド格納ディレクトリ（パスを構成するすべてのディレクトリ）の検索権限と外部コマンドの実行権限がある場合，引数に指定したコマンド名を出力します。これらの権限がない場合は，外部コマンドのコマンドパスが見つからなかったことを示すメッセージを出力します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了。 外部コマンドのコマンドパスが見つかりませんでした。または，複数の外部コマンドを検索した際，コマンドパスが見つからなかった外部コマンドがありました。
2	エラー終了。 • 不正なオプションを指定しました。 • 環境変数PATH が定義されていません。 • Windows の場合，環境変数PATHEXT が定義されていません。

注意事項

- 環境変数PATH が定義されていない場合はエラー終了します。
- Windows の場合，環境変数PATHEXT が定義されていない場合はエラー終了します。
- 次の名前を引数のコマンド名に指定した場合，外部コマンドとして検索します。
  - alias コマンドで定義したエイリアス
  - 予約語，シェル標準コマンド，シェル拡張コマンド，または関数
- 引数にパスが含まれているコマンド名を指定した場合，パス名が次の環境設定パラメーターによる変換対象のときは，変換後のパス名が出力されます。
  - PATH_CONV パラメーター
  - COMMAND_CONV_ARG パラメーター
- Windows の場合，コマンドパスの検索規則に該当する外部コマンドだけがコマンドパスの出力対象となります。

  awk，find，およびxargs コマンドで外部コマンドを実行する場合は，その外部コマンドのパス検索規則は，次のように行われます。

外部コマンドの実行方法	パス検索規則
• awk コマンドのsystem 関数 • awk コマンドの書式 *コマンド名* | getline [*変数名*] • awk コマンドの書式 print [*式*[, …]] | *コマンド名*	コマンドプロンプトなどのコマンドプロセッサ実行のパス検索規則に従う
• find コマンドの-exec プライマリ • find コマンドの-ok プライマリ • xargs コマンド	プログラム実行を行う Windows API のパス検索規則に従う

  このため，上記のコマンドに指定したコマンド名を引数に指定したときに出力されるコマンドパスと，実行されるコマンドのパスが異なる場合があります。

使用例

- コマンド名pgm01.exe のコマンドパスを取得します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which pgm01.exe
C:¥Program Files¥Hitachi¥PP001¥pgm01.exe
```

- コマンド名pgm01 のコマンドパスを取得します。コマンド名には拡張子を指定していません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which pgm01
C:¥Program Files¥Hitachi¥PP001¥pgm01.exe
```

- -a オプションを指定して，コマンド名pgm01.exe のすべてのコマンドパスを取得します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which -a pgm01.exe
C:¥Program Files¥Hitachi¥PP001¥pgm01.exe
C:¥Program Files¥Hitachi¥PP002¥pgm01.exe
C:¥Program Files¥Hitachi¥PP003¥pgm01.exe
```

- コマンド名pgm02 のコマンドパスを取得します。なお，コマンド検索パスにpgm02 は存在しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which pgm02
which: no pgm02 in (C:¥WINDOWS¥system32;C:¥WINDOWS;C:¥Program Files¥Hitachi¥PP001
;C:¥Program Files¥Hitachi¥PP002;C:¥Program Files¥Hitachi¥PP003)
```

- コマンド名pgm01，pgm02，pgm03，pgm04 のコマンドパスを取得します。なお，コマンド検索パスにコマンド名pgm02，pgm04 は存在しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which pgm01 pgm02 pgm03 pgm04
C:¥Program Files¥Hitachi¥PP001¥pgm01.exe
which: no pgm02 in (C:¥WINDOWS¥system32;C:¥WINDOWS;C:¥Program Files¥Hitachi¥PP001
;C:¥Program Files¥Hitachi¥PP002;C:¥Program Files¥Hitachi¥PP003)
C:¥Program Files¥Hitachi¥PP001¥pgm03.exe
which: no pgm04 in (C:¥WINDOWS¥system32;C:¥WINDOWS;C:¥Program Files¥Hitachi¥PP001
;C:¥Program Files¥Hitachi¥PP002;C:¥Program Files¥Hitachi¥PP003)
```

- パスを含むコマンド名を指定して実行します。指定したプログラム名は存在します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which ”C:¥Program Files¥Hitachi¥PP001¥pgm01”
C:¥Program Files¥Hitachi¥PP001¥pgm01.exe
```

- パスを含むコマンド名を指定して実行します。指定したコマンド名は存在しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥which ”C:¥Program Files¥Hitachi¥PP001¥pgm02”
which: no pgm02 in (C:¥Program Files¥Hitachi¥PP001)
```

8.4.41　xargs コマンド（コマンドラインを生成して実行する）

形式

```
xargs  [-0] [-r] [-x] [-d デリミタ] [-n コマンド引数の最大個数]
       [-s コマンドラインの最大長] [--cmdrc-threshold=しきい値]
       [外部コマンド名 [外部コマンドのコマンド引数 ...]
```

機能

標準入力からコマンド引数を入力し，コマンドラインを生成して実行します。

コマンド引数の入力方法については，「コマンド引数の入力」を参照してください。

生成するコマンドラインの長さが，コマンドラインの最大長を超えるときは，コマンドラインに含めるコマンド引数の数を調整し，コマンドラインの生成と実行を繰り返します。コマンドラインの最大長については，「コマンドラインの最大長」を参照してください。

コマンド引数の入力

標準入力から入力するコマンド引数とは，標準入力から，コマンド引数の区切り文字で区切られた部分を選択した文字列です。

コマンド引数の区切り文字は，スペース（0x20），タブ文字（0x09），および改行文字です。Windows の場合，［CR］＋［LF］（0x0d0a）または［LF］（0x0a）が改行文字と見なされます。UNIX の場合，［LF］（0x0a）が改行文字と見なされます。これらの区切り文字が連続している場合，1 つの区切り文字として扱います。コマンド引数の区切り文字は，-0 オプションまたは-d オプションで変更できます。

コマンド引数の入力では，次の文字は特殊な意味を持ちます。

表 8-23　コマンド引数の入力で意味を持つ特殊文字

特殊文字	意味
’	’（シングルクォーテーション）で囲まれた文字列に含まれる次の文字は，コマンド引数に含まれる文字として扱います。 • コマンド引数の区切り文字（改行文字を除く） • その他の特殊文字
”	”（ダブルクォーテーション）で囲まれた文字列に含まれる次の文字は，コマンド引数に含まれる文字として扱います。 • コマンド引数の区切り文字（改行文字を除く） • その他の特殊文字
¥	¥（バックスラッシュ）の直後にある次の文字は，その特殊な意味を無効（エスケープ文字）にします。 • コマンド引数の区切り文字 • 特殊文字

また，コマンド引数の入力で，考慮する必要がある文字を次に示します。

- NULL(0x00)

コマンド引数の文字列に含めることはできません。また，次のオプションをどちらも指定しなかった場合は，標準入力からNULL(0x00)を入力すると，入力を終了します。

・-0 オプション

・オプション値に，「¥0」などのNULL(0x00)を表すデリミタを指定した-d オプション

- フォームフィード文字(0x0c)，復帰文字(0x0d)，および垂直タブ文字(0x0b)

  これらの文字が，区切り文字で区切られたコマンド引数の先頭に置かれる場合は，コマンド引数の文字列に含まれません。

  (例）フォームフィード文字は¥f，復帰文字は¥r，垂直タブ文字は¥v，スペースは△で示します。

  ・標準入力の内容

```
¥f¥f¥rABC¥vDEF△¥vGHI¥fJKL
```

  ・外部コマンドに渡されるコマンド引数配列(argv)の内容

コマンド引数配列(argv)	コマンド引数の内容
1 番目	*外部コマンド名*
2 番目	ABC¥vDEF
3 番目	GHI¥fJKL

コマンドラインの生成と実行は，次の要因で標準入力からコマンド引数が入力されなかったときにも行われます。

- 標準入力からの入力が 0 バイト
- コマンド引数の区切り文字だけを入力した

標準入力からコマンド引数が入力されなかった場合に，コマンドラインの生成と実行をしたくないときは，-r オプションを指定してください。

コマンド引数の入力時の注意点については，「注意事項」を参照してください。

コマンドラインの長さ

UNIX の場合，コマンドラインの長さの単位はバイトです。

Windows の場合，コマンドラインの長さの単位は文字です。マルチバイト文字は 1 文字として数えます。

コマンドラインの長さは，次の文字列の長さの合計です。文字列の長さには，文字列の終端を示すNULL（0x00）の長さを含みます。

- 外部コマンド名
- xargs コマンドの引数に指定した外部コマンド名のコマンド引数
- 標準入力から入力するコマンド引数

Windows の場合，文字列の内容によって，次の文字数を加算します。

文字列の内容	加算する文字数
空白文字（0x20）※またはタブ文字（0x09）※を含む	2

文字列の内容	加算する文字数
”（ダブルクォーテーション）※を含む	1
”（ダブルクォーテーション）※の前に「¥」※がある	・「¥」が 1 つの場合：1 ・「¥」が連続している場合：「¥」の個数
終端が「¥」※でかつ，空白文字（0x20）※を含む	・「¥」が 1 つの場合：1 ・「¥」が連続している場合：「¥」の個数

注※

「コマンド引数の入力で意味を持つ特殊文字」の表に示す特殊文字，-0 オプション，または-d オプションで，コマンド引数の文字列として扱う場合です。

コマンドラインの最大長

「コマンドラインの長さ」に示すコマンドラインの長さが，コマンドラインの最大長を超えるときは，コマンドラインに含めるコマンド引数の数を調整し，コマンドラインの生成と実行を繰り返します。

コマンドラインの最大長は，-s オプションを指定しない場合，デフォルト値が適用されます。デフォルトのコマンドラインの最大長は，システムで実行できるコマンドラインの最大長によって決定します。コマンドラインの最大長と，システムで実行できるコマンドラインの最大長について，次に説明します。

【UNIX の場合】

コマンドラインの最大長は，システムで実行できるコマンドラインの最大長によって，次の値となります。

条件	使用するコマンドラインの最大長
システムで実行できるコマンドラインの最大長 ≧ 131072	131072 バイト(128K バイト)
システムで実行できるコマンドラインの最大長 ＜ 131072	システムで実行できるコマンドラインの最大長

システムで実行できるコマンドラインの最大長は，次の算出式で求めた値です。

システムで実行できるコマンドラインの最大長※ ＝ A － E － 8192

（凡例）

A：システムのARG_MAX 値

E：外部コマンド実行時に設定される環境変数の合計サイズ

注※

小数点以下は切り捨てます。

重要

・64 ビット版の Linux の場合，ARG_MAX 値に 1G バイト以上の値が設定されているときは，1G－1 バイトをARG_MAX 値とします。

・Solaris の場合，32 ビットのプログラムに適用される値をARG_MAX 値とします。

なお，Solaris の場合，次の 1 と 2 の合計がコマンドラインの最大長を超えないように，コマンドラインを生成します。

1.「コマンドラインの長さ」のコマンドラインの長さ

2.　次の文字列ごとに 4 バイト

・外部コマンド名

・xargs コマンドの引数に指定した外部コマンドのコマンド引数

・標準入力から入力するコマンド引数

【Windows の場合】

コマンドラインの最大長は，32760 文字です。システムで実行できるコマンドラインの最大長も同じ値です。

なお，コマンドラインに含めるコマンド引数の数が 1 個の場合，コマンドラインの最大長を超えるときは，エラー終了します。

引数

-0

--null

NULL（0x00）を，コマンドライン引数の区切り文字に設定します。

標準入力からコマンド引数を入力するときに，NULL（0x00）で区切られた文字列を，1 つのコマンド引数として扱います。

デフォルトのコマンドライン引数の区切り文字や，「コマンド引数の入力で意味を持つ特殊文字」に示す特殊文字は，コマンド引数に含まれる文字として扱います。

なお，標準入力から入力する文字列で，NULL（0x00）の前にNULL（0x00）以外の文字がない場合でも，外部コマンドには文字列の終端を示すNULL（0x00）だけを格納したコマンド引数が渡されます。

（例）

NULL を「¥0」で示します。

- 標準入力の内容

```
abc¥0¥0def
```

- 外部コマンドに渡されるコマンド引数配列(argv)の内容

コマンド引数配列(argv)	コマンド引数の内容
1 番目	*外部コマンド名*¥0
2 番目	abc¥0
3 番目	¥0
4 番目	def¥0

-0 オプションおよび-d オプションを同時に指定した場合，最後に指定したオプションが有効になります。

-r

--no-run-if-empty

標準入力からコマンド引数を入力できなかった場合，外部コマンドを実行しません。コマンド引数の入力については，「コマンド引数の入力」を参照してください。

-x

--exit

生成したコマンドラインの長さが，コマンドラインの最大長を超えた場合，エラー終了させます。

このオプションは，-n オプションを同時に指定した場合に有効になります。-n オプションと同時に指定した場合の動作については，-n オプションの説明を参照してください。

-d *デリミタ*

--delimiter=*デリミタ*

指定した*デリミタ*を，コマンドライン引数の区切り文字に設定します。

標準入力からコマンド引数を入力するときに，*デリミタ*で区切られた文字列を，1 つのコマンド引数として扱います。

デフォルトのコマンドライン引数の区切り文字や，「コマンド引数の入力で意味を持つ特殊文字」の表に示す特殊文字は，コマンド引数に含まれる文字として扱います。

*デリミタ*には，1 バイトの文字だけ指定できます。指定した*デリミタ*の長さが 1 バイト以外のときは，エラー終了します。

また，次のエスケープ文字も指定できます。

エスケープ文字	意味
¥a	アラート文字(ベル)
¥b	バックスペース文字
¥f	フォームフィード文字(改ページ)
¥n	改行文字
¥r	復帰文字
¥t	タブ文字
¥v	垂直タブ文字
¥*d*，¥*dd*，¥*ddd*	1～3 桁の 8 進数で表された ASCII コードの文字(*d*：0～7)
¥*xh*，¥*xhh*	1～2 桁の 16 進数で表された ASCII コードの文字(*h*：0～9，a～f，A～F)

なお，標準入力から入力する文字列で，指定したデリミタの前にデリミタ以外の文字がない場合でも，外部コマンドには文字列の終端を示すNULL（¥0）だけを格納したコマンド引数が渡されます。

（例）NULL を「¥0」で示します。指定したデリミタを「X」で示します。

- 標準入力の内容

```
abcXXdef
```

- 外部コマンドに渡されるコマンド引数配列(argv)の内容

コマンド引数配列(argv)	コマンド引数の内容
1 番目	*外部コマンド名*¥0
2 番目	abc¥0
3 番目	¥0
4 番目	def¥0

-d オプションおよび-0 オプションを同時に指定した場合，最後に指定したオプションが有効になります。

-n *コマンド引数の最大個数*

--max-args=*コマンド引数の最大個数*

コマンドラインに含めるコマンド引数の最大個数を指定します。

指定するコマンド引数の最大個数には，次の引数の数は含まれません。

- 外部コマンド名
- xargs コマンドのコマンドラインに指定した外部コマンドのコマンド引数

コマンド引数の最大個数は，1～2147483647 の範囲で指定できます。最大個数分のコマンド引数を含めたコマンドラインの長さが，コマンドラインの最大長を超える場合は，最大長を超えないように，最大個数よりも少ない数のコマンド引数で生成します。

なお，-x オプションを同時に指定した場合は，最大個数分のコマンド引数を含めたコマンドラインの長さが，コマンドラインの最大長を超えると，エラー終了します。

-s *コマンドラインの最大長*

--max-chars=*コマンドラインの最大長*

生成するコマンドラインの最大長を指定します。

UNIX の場合，コマンドラインの長さの単位はバイトです。

Windows の場合，コマンドラインの長さの単位は文字です。マルチバイト文字は 1 文字として数えます。

最大長は，1 から「コマンドラインの最大長」に示す「システムで実行できるコマンドラインの最大長」の範囲で指定できます。

指定した最大長が，「コマンドラインの最大長」に示す「システムで実行できるコマンドラインの最大長」より大きい場合は，標準エラー出力にメッセージ「xargs: The value specified for the option -s exceeds the maximum length（*システムで実行できるコマンドラインの最大長*）of the command line in the system」を出力します。また，「システムで実行できるコマンドラインの最大長」が，コマンドラインの最大長に設定されます。

--cmdrc-threshold=*しきい値*

実行する外部コマンドの終了コードが0でない場合でも，正常終了と見なしたい場合，しきい値となる値を指定します。これによって，外部コマンドの終了コードが，しきい値以下の場合が正常終了となります。

しきい値に指定できる値は，UNIXの場合は0～254の範囲で，Windowsの場合は0～2147483647の範囲で指定できます。

しきい値の対象となるのは，xargsコマンドが終了コードに123を設定するときの外部コマンドの終了コードだけです。

外部コマンドの終了コードの扱いについては，「終了コード」を参照してください。

外部コマンド名

コマンドラインで実行する外部コマンド名を指定します。

*外部コマンド名*にパスが含まれていない場合は，OSのファイル実行関数（UNIXはexecvp関数，WindowsはCreateProcess関数）のパス検索順序で見つかった外部コマンドが実行されます。

*外部コマンド名*の指定を省略した場合，コマンド引数をそのまま標準出力に出力します。また，コマンド引数内のエスケープ文字もそのまま出力します。なお，「コマンドラインの長さ」のコマンドラインの長さには，外部コマンド名分の文字列の長さとして，5が加算されます。

外部コマンドのコマンド引数

コマンドラインで，外部コマンド名の後ろに追加するコマンド引数を指定します。

標準入力から入力したコマンド引数は，このコマンド引数の後ろに追加されます。複数回，コマンドラインの生成と実行が行われる場合は，それぞれのコマンドラインに，このコマンド引数が含められます。

終了コード

終了コード	意味
0	正常終了。
1	エラー終了。 xargsコマンド処理でエラーが発生しました。
123	【UNIXの場合】 外部コマンドが1～254（126，127，およびシグナル終了時の終了コードは除く）の終了コード※1で終了しました。 【Windowsの場合】 外部コマンドが1以上の終了コード※2で終了しました。 コマンドラインの生成と実行処理を複数回行う必要がある場合，処理は継続されます。
124	【UNIXの場合】 外部コマンドが255の終了コード※1で終了しました。 【Windowsの場合】 外部コマンドが0未満の終了コード※2で終了しました。

終了コード	意味
124	なお，0 未満の終了コードのうち，例外コードは除きます。例外コードについては，「5.8.8　ジョブ，ジョブステップおよびコマンドの終了コード」の「例外として扱う例外コードと意味」の表を参照してください。
125	【UNIX の場合】 外部コマンドがシグナル終了しました。 【Windows の場合】※3 外部コマンドで例外が発生して終了しました。例外の内容については，「5.8.8　ジョブ，ジョブステップおよびコマンドの終了コード」の「例外として扱う例外コードと意味」の表を参照してください。
126	外部コマンドを実行できませんでした。 UNIX の場合，外部コマンドの終了コードが126 のときは，この終了コードになります。
127	外部コマンドが見つかりませんでした。 UNIX の場合，外部コマンドの終了コードが127 のときは，この終了コードになります。

注※1

外部コマンドがリターンする値の下位 8 ビットを，終了コードとして扱います。

注※2

外部コマンドがリターンする値を，符号付きの整数として扱います。

注※3

TerminateProcess などによるプロセス即時終了で外部コマンドが終了した場合は，プロセス即時終了操作で設定される終了コードが，外部コマンドの終了コードになります。

注意事項

- 次のオプションの両方を指定しなかった場合，標準入力から入力する文字列に，NULL（0x00）が含まれているときは，NULL（0x00）の直前までの文字列で，コマンドラインを生成して実行します。また，NULL（0x00）以降の文字列は無視されます。
  - -0 オプション
  - オプション値に，「¥0」などのNULL（0x00）を表すデリミタを指定した-d オプション

  NULL（0x00）の扱いについては，「コマンド引数の入力」を参照してください。
- Windows の場合，標準入力からコマンド引数としてパス名を入力するときは，ディレクトリ区切り文字の¥（バックスラッシュ）や，UNC 形式のパス名を示す先頭の「¥¥」は，エスケープ文字として扱われます。このため，次のどれかを行ってください。
  - 「コマンド引数の入力で意味を持つ特殊文字」の表に示す特殊文字で，¥（バックスラッシュ）を文字として扱うようにします。
  - -0 オプションを指定します。さらに，標準入力内のパス名の区切りをNULL（0x00）にします。例えば，find コマンドが出力するファイル検索結果を，xargs コマンドで入力するときは，find コマンドの検索式に-print0 を指定します。

- -d オプションを指定します。さらに，標準入力内のパス名の区切りを，-d オプションに指定するデリミタにします。例えば，find コマンドが出力するファイル検索結果を，xargs コマンドで入力するときは，-d オプションに「¥n」（改行文字）を指定します。

- Windows の場合，特殊文字などを使用して，改行文字をコマンド引数に含めるときは，外部コマンドに渡される改行文字は［LF］（0x0a）です。
- Windows の場合，標準入力から0x1a コードを入力すると，入力を終了します。
- コマンドラインの実行では，外部コマンドの標準入力には，次のファイルが設定されます。
  - UNIX の場合：/dev/null ファイル
  - Windows の場合：NUL デバイス

  /dev/null ファイル，またはNUL デバイスがシステムに存在しない場合，標準エラー出力にメッセージ「xargs: Failed to open a null file when executing the external command *外部コマンド名* (error=*エラー詳細*)」を出力します。また，外部コマンドの標準入力をクローズしてから，コマンドラインを実行します。
- 外部コマンドからプログラムを実行する場合，外部コマンドに渡されたコマンド引数をプログラムに渡すときは，OS のコマンドラインの最大長を超えないようにしてください。OS のコマンドラインの最大長を超えた場合は，プログラムの実行に失敗します。
- 実行する外部コマンドが，0 以外の正常終了を示す終了コードを返しても，xargs コマンドの終了コードは123 になります。xargs コマンドの終了コードを0 にしたい場合は，--cmdrc-threshold オプションを使用してください。なお，外部コマンドの終了コードの扱いについては，「終了コード」を参照してください。
- 外部コマンドによっては，その外部コマンドのコマンドラインの最大長が，「コマンドラインの最大長」に示すデフォルトのコマンドラインの最大長より小さいときがあります。このような場合は，-s オプションに，外部コマンドのコマンドラインの最大長を指定して実行してください。
- 外部コマンドにコマンド引数の個数の上限がある場合は，-n オプションを使用して，コマンドラインに含めるコマンド引数の数が上限を超えないようにしてください。

使用例

find コマンドで検索したファイルのパス名を受け取り，rm コマンドのオペランドに指定して実行します。なお，ディレクトリ区切り文字をエスケープ文字として扱わないようにするため，次の指定をします。

- find コマンドの検索式に-print0 を指定。
- xargs コマンドのオプションに-0 を指定。

また，-r オプションを指定し，find コマンドでファイルが見つからなかったときに，rm コマンドを実行しないようにします。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls "C:¥Program TEMP"
cmd001.log   file001.tmp  file002.tmp  file003.tmp  file004.tmp  file005.tmp  file006.tmp

C:¥TEMP>%ADSH_OSCMD_DIR%¥find "C:¥Program TEMP" -name "*.tmp" -print0 | %ADSH_OSCMD_DIR%
```

```
¥xargs -0 -r %ADSH_OSCMD_DIR%¥rm -f

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls "C:¥Program TEMP"
cmd001.log
```

8.5　UNIX 互換コマンド（スクリプト形式）【Windows 限定】

次の UNIX 互換コマンドは，JP1/Advanced Shell が提供するサンプルスクリプトファイルを使用して実行します。提供するサンプルスクリプトファイルは Windows 限定です。UNIX では OS 提供のコマンドを使用してください。

表 8-24　サンプルスクリプトファイルで提供する UNIX 互換コマンド

コマンド名	サンプルスクリプトファイル名	機能概要
全コマンド	script_0	ジョブ定義スクリプトに記述されているコマンドの指定を無効にします。
chmod	script_chmod1	ファイルの読み取り専用属性の有効・無効を切り替えます。
	script_chmod2	ファイルまたはフォルダに対するパーミッションを数値で設定します。
	script_chmod3	ファイルまたはフォルダに対するパーミッションをシンボルまたは数値で設定します。
su	script_su1	実行ユーザーの権限でプログラムを実行します。
who	script_who1	ログインユーザーの情報をログに出力します。

サンプルスクリプトファイルを使うための詳細手順は，「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

8.5.1　chmod コマンド（ジョブ定義スクリプトに記述されている chmod コマンドの指定を無効にする）

形式

```
chmod ［オプション］［モード］［パス名］
```

このコマンドはサンプルスクリプトファイル script_0 を基に作成します。作成手順については「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

ジョブ定義スクリプトに記述されているすべての chmod コマンドとその引数の指定を無効にします。このコマンドを実行すると，常に終了コード 0 で正常終了します。

Windows ではログインユーザーごとにアクセス制御する運用とすると，ジョブ定義スクリプトの実行時のアクセス権変更が不要になる場合があります。このような場合，このコマンドを実行することで，ジョブ定義スクリプトのすべての chmod コマンドの定義を無効にできるため，UNIX から Windows に移行してきたジョブ定義スクリプトの修正が不要になります。

引数

オプション

指定を無視します。

モード

指定を無視します。

パス名

指定を無視します。

終了コード

終了コード	意味
0	正常終了

注意事項

- コマンド置換で引数に指定したコマンドは実行されるため，後続の処理に影響がある場合は，指定を見直してください。

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_0 を基に chmod コマンドが作成されていることを前提とします。

- ジョブ定義スクリプトの chmod コマンドの指定を無視します。この例で指定された chmod コマンドのオプションは実行されません。

```
chmod go-x test.txt
if [[ $? -ge 1 ]]; then     # chmodの終了コードは必ず0なので，そのまま処理を続行する。
  echo "chmod error." 1>&2
  exit 1
fi
```

8.5.2 chmod コマンド（ファイルの読み取り専用属性の有効・無効を切り替える）

形式

```
chmod [-fhR] モード パス名
```

このコマンドはサンプルスクリプトファイル script_chmod1 を基に作成します。作成手順については「(2) スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

ファイルの読み取り専用属性の有効・無効を切り替えます。

このコマンドは，ファイルの更新を抑止したい場合などに使用します。

引数

-f

指定を無視します。

-h

指定を無視します。

-R

指定を無視します。

モード

モードをシンボルまたは数値で指定して，読み取り専用属性の有効・無効を切り替えます。指定方法を次に示します。これ以外のモードが指定された場合，標準エラー出力へ「chmod: invalid file mode: モード」を出力し，アクセス許可は変更しません。

指定内容	シンボルでの指定	数値での指定
読み取り専用属性を無効にし，書き込みを可能にする場合 (adshscripttool -fmode -s w コマンドの実行結果が AAA または RRR の場合に相当)	+w を指定	u，g，o の書き込み権限のモードビットがすべて ON になるよう指定（777，666，333，222，733 など）
読み取り専用属性を有効にし，書き込みを禁止にする場合 (adshscripttool -fmode -s w コマンドの実行結果が DDD の場合に相当)	-w を指定	u，g，o の書き込み権限のモードビットがすべて OFF になるよう指定（555，444，111，000，511 など）

パス名

対象とするファイルを指定します。複数指定することもできます。フォルダは指定できません。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- ファイルの属性の変更権限がない場合，attrib コマンドが標準エラー出力にメッセージ「アクセスは拒否されました - **パス名**」を出力して attrib コマンドがエラーとなり，権限を変更できません。実行ユーザーに属性の変更権限を付与して使用してください。
- パス名にフォルダを指定した場合，メッセージ「chmod: cannot access [**パス名**]: change for the directory is not supported」を出力し，フォルダの読み取り専用属性は変更されないで，終了コード 1 で終了します。
- 引数パス名にシンボリックリンクを指定した場合，リンク先のファイルが変更の対象となります。シンボリックリンク自身を変更の対象とする場合は OS が提供するコマンドを使用してください。

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_chmod1 を基に chmod コマンドが作成されていることを前提とします。

- シンボル指定でファイルへの書き込みを禁止します。

```
chmod -w test.txt
```

- シンボル指定でファイルへの書き込みを許可します。

```
chmod +w test.txt
```

- 数値指定でファイルへの書き込みを禁止します。

```
chmod 444 test.txt
```

- 指定が許可されていないモードを指定します。

```
chmod -r test.txt
```

  この場合，標準エラー出力には次のように出力されます。

```
chmod: invalid file mode: -r
```

8.5.3　chmod コマンド（パーミッションを数値で設定する）

形式

```
chmod [-fhR] モード パス名
```

このコマンドはサンプルスクリプトファイル script_chmod2 を基に作成します。作成手順については「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

既存の ACL（Access Control List）を削除して，モードの数値の指定に従った ACL を再設定します。

このコマンドは，次のような設定をしたい場合などにアクセス権を数値で設定します。

- 「所有者」以外のユーザーの「書き込み」「読み取り」を抑止したい場合
- すべてのユーザーの「書き込み」または「読み取り」を許可したい場合
- 「所有者」を含むすべてのユーザーの「書き込み」を抑止したい場合

引数

-f

指定を無視します。

-h

指定を無視します。

-R

指定を無視します。

モード

モードの指定値と，それによって設定される各 ACE（Access Control Entry）のアクセス許可を次に示します。これ以外のモードが指定された場合，標準エラー出力へ「chmod: invalid file mode: モード」を出力し，アクセス許可は変更されません。

モードの指定値	設定されるアクセス許可
777	所有者：F，Everyone：F
766	所有者：F，Everyone：C
755	所有者：F，Everyone：R
744	所有者：F，Everyone：R
733	所有者：F，Everyone：W
722	所有者：F，Everyone：W
700	所有者：F
666	所有者：C，Everyone：C
655	所有者：C，Everyone：R
644	所有者：C，Everyone：R
633	所有者：C，Everyone：W
622	所有者：C，Everyone：W
600	所有者：C

モードの指定値	設定されるアクセス許可
555	所有者：R，Everyone：R
544	所有者：R，Everyone：R
533	所有者：R，Everyone：W
522	所有者：R，Everyone：W
500	所有者：R
444	所有者：R，Everyone：R
433	所有者：R，Everyone：W
422	所有者：R，Everyone：W
400	所有者：R
333	所有者：W，Everyone：W
322	所有者：W，Everyone：W
300	所有者：W
222	所有者：W，Everyone：W
200	所有者：W

（凡例）

表の F，C，R，W は，cacls コマンドの次のアクセス許可と対応しています。

F：フルコントロール

C：変更権限

R：読み取り権限

W：書き込み権限

モードでは，読み取り権限と書き込み権限を合わせたもの（モードビット 6 の場合）を変更権限として定義しています。すべての権限を合わせたもの（モードビット 7 の場合）をフルコントロールとして定義しています。

モードで，実行権限に相当する指定は無視します。したがって，モードビット 5 は 4 と同等，モードビット 3 は 2 と同等として定義しています。

パス名

対象とするファイルまたはフォルダを指定します。複数指定することもできます。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- このコマンドを実行すると，既存の ACL は削除され，モードの説明にある ACE の設定だけを実施します。必要なアカウントの ACE がある場合は，サンプルスクリプト内の cacls コマンドの定義に追加してください。
- 「その他のユーザー」のアカウントを Everyone にしているため，所有者のアクセス権がその他のユーザーのアクセス権より低い場合でも，所有者は Everyone のアクセス権でアクセスできるようになります。
- ファイルおよびフォルダのアクセス許可の変更権限がない場合，cacls コマンドが標準エラー出力にメッセージ「アクセスが拒否されました」を出力して cacls コマンドがエラーとなり，権限を変更できません。実行ユーザーにアクセス許可の変更権限を付与して使用してください。
- 引数パス名にシンボリックリンクを指定した場合，リンク先のファイル，ディレクトリが変更の対象となります。シンボリックリンク自身を変更の対象とする場合は OS が提供するコマンドを使用してください。

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_chmod2 を基に chmod コマンドが作成されていることを前提とします。

- 指定したファイルに対し，すべてのユーザーを読み取り可能にします。

```
chmod 444 test.txt
```

- 指定が許可されていないモードを指定します。

```
chmod 611 test.txt
```

この場合，標準エラー出力には次のように出力されます。

```
chmod: invalid file mode: 611
```

8.5.4　chmod コマンド（パーミッションをシンボルまたは数値で設定する）

形式

```
chmod [-fhR] モード パス名
```

このコマンドはサンプルスクリプトファイル script_chmod3 を基に作成します。作成手順については「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

所有者および Everyone 以外の ACE を削除して，モードのシンボルまたは数値の指定に従い，アクセス許可を変更または設定します。

このコマンドは，次のような設定をしたい場合などにアクセス権をシンボルまたは数値で設定します。

- chmod コマンドのモードがシンボルで指定されたジョブ定義スクリプトを Windows に移行したい場合
- 「所有者」またはすべてのユーザーのアクセス権を追加および抑止したい場合

引数

-f

指定を無視します。

-h

指定を無視します。

-R

指定を無視します。

モード

モードの指定値と，それによって設定される各 ACE のアクセス許可を次に示します。これ以外のモードが指定された場合，標準エラー出力へ「chmod: invalid file mode: **モード**」を出力し，アクセス許可は変更しません。

モードの指定値 （括弧内：adshscripttool -fmode コマンドの実行結果）	設定されるアクセス許可
u+r（A00000000）	所有者：R を追加
u+rw（AA0000000）	所有者：C を追加
u+rwx（AAA000000）	所有者：F を追加
u+w（0A0000000）	所有者：W を追加
u-rwx（DDD000000）	所有者の ACE を削除
u=r（RDD000000）	所有者：R に置き換え
u=rw（RRD000000）	所有者：C に置き換え
u=rwx（RRR000000）	所有者：F に置き換え
u=w（DRD000000）	所有者：W に置き換え
o+r（000000A00）	Everyone：R を追加
o+rw（000000AA0）	Everyone：C を追加
o+rwx（000000AAA）	Everyone：F を追加

モードの指定値 （括弧内：adshscripttool -fmode コマンドの実行結果）	設定されるアクセス許可
o+w（0000000A0）	Everyone：W を追加
o-rwx（000000DDD）	Everyone の ACE を削除
o=r（000000RDD）	Everyone：R に置き換え
o=rw（000000RRD）	Everyone：C に置き換え
o=rwx（000000RRR）	Everyone：F に置き換え
o=w（000000DRD）	Everyone：W に置き換え
+r / ugo+r（A00A00A00）	所有者：R，Everyone：R を追加
+rw / ugo+rw（AA0AA0AA0）	所有者：C，Everyone：C を追加
+rwx / ugo+rwx（AAAAAAAAA）	所有者：F，Everyone：F を追加
+w / ugo+w（0A00A00A0）	所有者：W，Everyone：W を追加
-rwx / ugo-rwx（DDDDDDDDD）	所有者，Everyone の ACE を削除
=r / ugo=r（RDDRDDRDD）	所有者：R，Everyone：R に置き換え
=rw / ugo=rw（RRDRRDRRD）	所有者：C，Everyone：C に置き換え
=rwx / ugo=rwx（RRRRRRRRR）	所有者：F，Everyone：F に置き換え
=w / ugo=w（DRDDRDDRD）	所有者：W，Everyone：W に置き換え
777（RRRRRRRRR）	所有者：F，Everyone：F に置き換え
766（RRRRRDRRD）	所有者：F，Everyone：C に置き換え
755（RRRRDRRDR）	所有者：F，Everyone：R に置き換え
744（RRRRDDRDD）	所有者：F，Everyone：R に置き換え
733（RRRDRRDRR）	所有者：F，Everyone：W に置き換え
722（RRRDRDDRD）	所有者：F，Everyone：W に置き換え
700（RRRDDDDDD）	所有者：F に置き換え
666（RRDRRDRRD）	所有者：C，Everyone：C に置き換え
655（RRDRDRRDR）	所有者：C，Everyone：R に置き換え
644（RRDRDDRDD）	所有者：C，Everyone：R に置き換え
633（RRDDRRDRR）	所有者：C，Everyone：W に置き換え
622（RRDDRDDRD）	所有者：C，Everyone：W に置き換え
600（RRDDDDDDD）	所有者：C に置き換え
555（RDRRDRRDR）	所有者：R，Everyone：R に置き換え
544（RDRRDDRDD）	所有者：R，Everyone：R に置き換え

モードの指定値 （括弧内：adshscripttool -fmode コマンドの実行結果）	設定されるアクセス許可
533（RDRDRRDRR）	所有者：R，Everyone：W に置き換え
522（RDRDRDDRD）	所有者：R，Everyone：W に置き換え
500（RDRDDDDDD）	所有者：R に置き換え
444（RDDRDDRDD）	所有者：R，Everyone：R に置き換え
433（RDDDRRDRR）	所有者：R，Everyone：W に置き換え
422（RDDDRDDRD）	所有者：R，Everyone：W に置き換え
400（RDDDDDDDD）	所有者：R に置き換え
333（DRRDRRDRR）	所有者：W，Everyone：W に置き換え
322（DRRDRDDRD）	所有者：W，Everyone：W に置き換え
300（DRRDDDDDD）	所有者：W に置き換え
222（DRDDRDDRD）	所有者：W，Everyone：W に置き換え
200（DRDDDDDDD）	所有者：W に置き換え

（凡例）

表の F，C，R，W は，cacls コマンドの次のアクセス許可と対応しています。

F：フルコントロール

C：変更権限

R：読み取り権限

W：書き込み権限

モードでは，読み取り権限と書き込み権限を合わせたもの（モードビット 6 の場合）を変更権限として定義しています。すべての権限を合わせたもの（モードビット 7 の場合）をフルコントロールとして定義しています。

モードで，実行権限に相当する指定は無視します。したがって，モードビット 5 は 4 と同等，モードビット 3 は 2 と同等として定義しています。

パス名

対象とするファイルまたはフォルダを指定します。複数指定することもできます。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- このコマンドを実行すると，所有者および Everyone 以外の ACE は削除され，モードの説明にある ACE の設定だけを実施します。必要なアカウントの ACE がある場合は，サンプルスクリプト内の cacls コマンドの定義に追加してください。
- 「その他のユーザー」のアカウントを Everyone にしているため，所有者のアクセス権がその他のユーザーのアクセス権より低い場合でも，所有者は Everyone のアクセス権でアクセスできるようになります。
- ファイルおよびフォルダのアクセス許可の変更権限がない場合，cacls コマンドが標準エラー出力にメッセージ「アクセスが拒否されました」を出力して cacls コマンドがエラーとなり，権限を変更できません。実行ユーザーにアクセス許可の変更権限を付与して使用してください。
- 引数パス名にシンボリックリンクを指定した場合，リンク先のファイル，ディレクトリが変更の対象となります。シンボリックリンク自身を変更の対象とする場合は OS が提供するコマンドを使用してください。

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_chmod3 を基に chmod コマンドが作成されていることを前提とします。

- その他のユーザーに書き込み権限を追加します。

```
chmod o+w test.txt
```

- 指定が許可されていないモードを指定します。

```
chmod g-w test.txt
```

この場合，標準エラー出力には次のように出力されます。

```
chmod: invalid file mode: g-w
```

8.5.5 su コマンド（ジョブ定義スクリプトに記述されている su コマンドの指定を無効にする）

形式

```
su [-] [ユーザー名] [引数...]
```

このコマンドはサンプルスクリプトファイル script_0 を基に作成します。作成手順については「(2) スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

ジョブ定義スクリプトに記述されているすべての su コマンドとその引数の指定を無効にします。このコマンドを実行すると，常に終了コード 0 で正常終了します。

UNIX で su コマンドを使ってサブシステムの起動・停止を実行していたジョブ定義スクリプトを Windows に移行したあと，Windows では別のシステムを利用してサブシステムの起動・停止を実行する運用としたときなどに，ジョブ定義スクリプトに記述されている su コマンドの処理が不要になる場合があります。このような場合，このコマンドを使用することで，ジョブ定義スクリプトのすべての su コマンドの定義を無効にできるため，UNIX から Windows に移行してきたジョブ定義スクリプトの修正が不要になります。

引数

-

指定を無視します。

ユーザー名

指定を無視します。

引数

指定を無視します。

終了コード

終了コード	意味
0	正常終了

注意事項

- コマンド置換で引数に指定したコマンドは実行されるため，後続の処理に影響がある場合は，指定を見直してください。

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_0 を基に su コマンドが作成されていることを前提とします。

- ジョブ定義スクリプトの su コマンドの指定を無視します。この例で指定された su コマンドのオプションは実行されません。

```
su - ${DBADMIN} -c 'export PDDIR=/home/db/db1; start -q'
if [[ $? -ge 1 ]]; then     # suの終了コードは必ず0なので，そのまま処理を続行する。
  echo "su error." 1>&2
  exit 1
fi
```

8.5.6　su コマンド（実行ユーザーの権限でプログラムを実行する）

形式

```
su　[-]　ユーザー名　{-c コマンドライン|スクリプトファイルのパス名}
　　[実行時パラメーター]
```

このコマンドはサンプルスクリプトファイル script_su1 を基に作成します。作成手順については「(2) スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

引数に指定されたコマンドを実行します。ユーザー名の指定は無視され，実行ユーザーの権限で実行されます。

既存のジョブ定義スクリプトに su コマンドが記述されている場合に，ジョブ定義スクリプトを書き換えないで Windows に移行できます。

引数

-

指定を無視します。

ユーザー名

指定を無視します。

-c コマンドライン

ジョブで実行するコマンドラインを指定します。

コマンドラインには，シェル運用コマンド，UNIX 互換コマンドなど，ジョブ定義スクリプトファイルに記述できるコマンドを指定できます。

スクリプトファイルのパス名

実行するスクリプトファイルのパス名を指定します。

実行時パラメーター

コマンドラインまたはスクリプトファイルのパス名の位置パラメーターに格納する値を指定します。スペースを実行時パラメーターとして指定する場合は，その文字列を"（ダブルクォーテーション）で囲んでください。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 実行ユーザーに必要な権限を与えてからこのコマンドを実行してください。

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_su1 を基に su コマンドが作成されていることを前提とします。

- 実行ユーザーに必要な権限を与え，引数-c に複数のコマンドを指定して実行します。

```
# 必要な権限を持ったユーザーからコマンドを実行
su - ${DBADMIN} -c 'export PDDIR=C:¥¥db¥¥db1; start -q'
```

- 実行ユーザーに必要な権限を与え，引数にスクリプトファイル名を指定して実行します。

```
# 必要な権限を持ったユーザーからジョブ定義スクリプトを実行
su - ${DBADMIN} '.¥¥DBSTART.ash'
```

8.5.7　who コマンド（ジョブ定義スクリプトに記述されている who コマンドの指定を無効にする）

形式

```
who [am i]
```

このコマンドはサンプルスクリプトファイル script_0 を基に作成します。作成手順については「(2)　スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

ジョブ定義スクリプトに記述されているすべての who コマンドとその引数の指定を無効にします。このコマンドを実行すると，常に終了コード 0 で正常終了します。

UNIX で who コマンドを使っていたが，Windows ではその情報が不要となったときなどに，ジョブ定義スクリプトに記述されている who コマンドの処理が不要になる場合があります。このような場合，このコマンドを使用することで，ジョブ定義スクリプトのすべての who コマンドの定義を無効にできるため，UNIX から Windows に移行してきたジョブ定義スクリプトの修正が不要になります。

引数

am i

指定を無視します。

終了コード

終了コード	意味
0	正常終了

注意事項

- コマンド置換で引数に指定したコマンドは実行されるため，後続の処理に影響がある場合は，指定を見直してください。

8.5.8　who コマンド（ログインユーザーの情報をログに出力する）

形式

```
who [am i]
```

このコマンドはサンプルスクリプトファイル script_who1 を基に作成します。作成手順については「(2) スクリプト形式の UNIX 互換コマンドを使うための準備【Windows 限定】」を参照してください。

機能

quser.exe コマンドまたは qwinsta.exe コマンドを起動します。

ジョブ定義スクリプトの実行時にシステムにログインしているユーザーの一覧をログに出力したい場合に指定します。

引数

am i

指定を無視します。

終了コード

終了コード	意味
0	正常終了
0 以外	エラー終了（quser.exe コマンドまたは qwinsta.exe コマンドの終了コード）

使用例

ジョブ定義スクリプトの定義例を次に示します。サンプルスクリプトファイル script_who1 を基に who コマンドが作成されていることを前提とします。

- 実行時のログインユーザーの一覧をログファイル「log.txt」に出力します。

```
who >>log.txt
```

9 ジョブ定義スクリプトのコマンドおよび制御文

この章では，ジョブ定義スクリプトファイルに使用するコマンドや制御文に関して，記述形式と詳細を説明します。

9.1　コマンドおよび制御文の記述形式

ジョブ定義スクリプトファイルには，次のコマンドおよび制御文が使用できます。

- シェル標準コマンド
- シェル拡張コマンド
- スクリプト拡張コマンド
- スクリプト制御文
- スクリプト予約語コマンド

なお，ジョブ定義スクリプトファイルを記述する際は，次の点に注意してください。

- 行の途中に NULL（"0x00"または C 言語での"¥0"）が混入している場合，ジョブコントローラはその行で NULL が現れる部分までを 1 行と見なします。その行で NULL のあとにほかの文字列があっても無視されます。不正な実行結果や実行時エラーの要因となるため，NULL を記述しないようにしてください。

  （例）

  - 入力行（"0x00"を「¥0」で示します）

    echo "test¥0null";echo "test after"

  - 出力例

    echo "test
- ジョブ定義スクリプトを見やすくするため，またはカバレージ情報が適切に表示されるようにするために，1 行にコマンドを 1 個ずつ記述することを推奨します。「;」を使用して，1 行に複数のコマンドを記述することはお勧めしません。

ジョブ定義スクリプトファイルに指定されたコマンドのカバレージ情報を採取する場合は，次の点に注意して記述してください。

- 1 行に記述するコマンドが 4 個以内の場合は，各コマンドを実行したかどうかの情報を表示できます。
- 1 行に記述するコマンドが 32 個以内の場合は，ジョブ定義スクリプト全体のコマンド数がすべて実行できたどうかを全体のコマンドで判断できます。

  1 行に記述するコマンドが 32 個を超える場合は，33 個目以降のコマンドはカバレージ情報を取得しません。ジョブ定義スクリプト内のすべてのコマンドを実行しても C0 実行比率が 100%になりません。
- if 文などのスクリプト制御文も，スクリプト制御文全体を 1 行に記述しないで，キーワード単位で改行することを推奨します。
- 次に示すキーワードは，単独で 1 行に記述してください（「fi;fi」のように記述しないでください）。同一の行に記述すると，カバレージ情報が正しく表示されません。
  - if 文の終了を示す fi
  - do のブロックの終了を示す done

- case 文の終了を示す esac

- カバレージ情報は，次の内容だけを出力します。
  - C0 情報の場合，行の先頭から C0 対象となる最初の 4 個のコマンド
  - C1 情報の場合，行の先頭から最初の 4 個の実行パス
- 1 行に複数のコマンドおよび実行パスを記述すると，すべてのカバレージ情報を表示しない場合があります。

  例 1

  ・1 行に複数のコマンドおよび実行パスを記述した場合

```
echo 1; echo 2; echo 3; echo 4; echo 5
```

  ・複数の行で，コマンドおよび実行パスを記述した場合

```
echo 1
echo 2
echo 3
echo 4
echo 5
```

  例 2

  ・1 行に複数のコマンドおよび実行パスを記述した場合

```
if true ;then echo 1 ;elif true ;then echo 2 ;elif true ;then echo 3 ;else echo 4 ;fi
```

  ・複数の行で，コマンドおよび実行パスを記述した場合

```
if true
then
  echo 1
elif true
then
  echo 2
elif true
then
  echo 3
else
  echo 4
fi
```

- 端末から標準入力へキーボード入力をするコマンドを実行する場合，入力を完了するには次の操作をしてください。
  - EOF まで入力する場合

    Windows では［Enter］を入力後に Ctrl+Z を入力して，さらに［Enter］を入力します。

    UNIX では Ctrl+D を入力します。
  - 1 行入力する場合

    ［Enter］を入力します。

コマンドおよび制御文の記述形式を次に示します。

9.1.1　シェル標準コマンドの記述形式

シェル標準コマンドの記述形式を次に示します。

△$_0$**コマンド名**［△$_1$**オプション**］…［△$_1$**オプション**］［△$_1$**オペランド**］

- 最初にオプションを指定し，次にオペランドを指定します。オペランドとは，オプション名とオプション値のほかにコマンドに指定できる引数のことです。オプションの前にオペランドを指定した場合は，指定内容をすべてオペランドとして処理します。
- オプションは「-**オプション名**［△$_1$ **値**］」の形式で指定します。オプションを複数指定する場合，指定順序は任意です。
- 値のないオプションは連続して指定できます（例：「-a -b -c」と「-abc」は同じです）。その場合，最後のオプションには値を指定できます（例：「-abc xyz」の「xyz」は，-c オプションの値となります）。
- 不当なオプション，または指定できる範囲外の値を指定した場合，エラーになります。
- オプション名にはマルチバイト文字は使用できません。

9.1.2　シェル拡張コマンドの記述形式

シェル拡張コマンドの記述形式を次に示します。

△$_0$**コマンド名**［△$_1$**オプション**］［△$_1$**オペランド**］

9.1.3　スクリプト拡張コマンドの記述形式

スクリプト拡張コマンドの記述形式を次に示します。

△$_0$**コマンド名**△$_1$**属性値**［…△$_1$**属性値**］［△$_1$-**属性名**△$_1$**属性値**［…△$_1$-**属性名**△$_1$**属性値**］］

- スクリプト拡張コマンドのコマンド名は，必ず「#-adsh_」で開始します。
- コマンド名のあとに，「**属性値**」のリストと，「-**属性名 属性値**」のリストを連続して記載します。
- 「**属性値**」のリストは順序に意味があるため，省略できません。「-**属性名 属性値**」のリストは順不同であり，省略できます。
- 「**属性値**」に「-」で開始する文字列を指定すると，「-**属性名**」の指定と見なされます。このため，「-」を「¥」，「"」，または「'」でエスケープしてください。
- スクリプト拡張コマンドと同一のコメントは記述できません。スクリプト拡張コマンドをコメントにする場合は，先頭にもう 1 つ「#」を書いてください。
- すべての記述でダブルクォーテーション「"」，シングルクォーテーション「'」，エスケープ文字「¥」が使用できます。

ただし，スクリプト拡張コマンドではダブルクォーテーションで囲まれた文字列中の「¥」は，直後の文字の種類に関係なくすべてエスケープ文字となります。ダブルクォーテーションで囲まれた文字列に「¥」を指定したい場合は，¥¥と記述してください。

- コマンド名，属性名および属性値（予約語の場合）は，すべて大文字と小文字を区別します。
- 属性値には環境変数名が指定でき，スクリプト起動前に設定していた値で置換できます。環境変数名を記述する場合，{ }で囲む必要があります。環境変数名は「(2)　文字セットの定義」で示す＜環境変数名＞の形式で，255 バイト以内で記述します。
- スクリプト拡張コマンドは各行の先頭に記述してください。また，コマンド名の後ろから改行コードまでに必ずスペースを指定します。スペース以外が存在すると構文解析エラーになります。
- コマンド区切り記号を指定して，同一行の 2 番目以降にスクリプト拡張コマンドを記述できません。記述すると構文解析エラーになります。
- 関数内にスクリプト拡張コマンドを記述できません。記述すると構文解析エラーになります。
- for 文，while 文，until 文のブロック内および関数定義内に，スクリプト拡張コマンドを記述できません。記述すると，実行前に文法エラーになります。
- 「.」で呼び出す外部スクリプトの中に，スクリプト拡張コマンドを記述できません。記述するとコメントとして扱われます。

スクリプト拡張コマンドを複数行に分けて記述する場合は，2 行目以降を次の形式で記述します。

```
#-adsh△1継続指定内容
```

- コマンド名および属性の区切り文字の個所だけ，継続行指定ができます。コマンド名，属性名および属性値の途中で継続行指定はできません。
- 継続行で文法エラーがある場合，エラーメッセージに表示される行番号は，そのスクリプト拡張コマンドの先頭行の行番号となります。

(1)　制限事項

- 継続する行を含めて，スクリプト拡張コマンドの 1 行は 8,191 バイト以下にしてください。
- 属性値を複数指定する場合はスペースまたはコンマで区切ります。コンマの間の値は省略できません。

(2)　文字セットの定義

属性値として使用できる文字セットの定義を次の表に示します。

表 9-1　属性値として使用できる文字セットの定義

構文要素	指定できる文字の内容	対象
＜記号名称＞	{＜英字＞|＜数字＞|@|#|_（アンダースコア）} ＋	ジョブ名など
＜環境変数名＞	{＜英字＞|_（アンダースコア）} {＜英字＞|_（アンダースコア）|＜数字＞} ＊	ファイル環境変数定義名など

構文要素	指定できる文字の内容	対象
＜パス名＞	OS のファイルパス名規則に従った文字列です。 「¥」はメタキャラクタ（エスケープ文字）として扱うため，Windows では次のように記述してください。メタキャラクタについては，「メタキャラクタ」を参照してください。 指定例：'C:¥test'または C:¥¥test など	パス名
＜任意文字列＞	任意の文字による文字列です。 次の範囲での利用を推奨します。 {＜英字＞|＜数字＞|@|#|_（アンダースコア）}＋	環境変数値など

9.1.4　スクリプト制御文の記述形式

スクリプト制御文の記述形式を次に示します。

△$_0$**制御文**［△$_1$**条件**］［△$_1$**予約語**［△$_1$**処理**］］…
　　［△$_1$**条件**］［△$_1$**予約語**［△$_1$**処理**］］
［△$_0$**制御文（終了）または予約語**］

条件，予約語，処理

条件，予約語，処理などを指定します。

9.1.5　スクリプト予約語コマンドの記述形式

スクリプト予約語コマンドの記述形式を次に示します。

△$_0$**コマンド名**［△$_1$**オプション**］…［△$_1$**オプション**］［△$_1$**オペランド**］

- 最初にオプションを指定し，次にオペランドを指定します。オペランドとは，オプション名とオプション値のほかにコマンドに指定できる引数のことです。オプションの前にオペランドを指定した場合は，指定内容をすべて任意名として処理します。
- オプションは「-**オプション名**［△$_1$ **値**］」の形式で指定します。オプションを複数指定する場合，指定順序は任意です。
- 値のないオプションは連続して指定できます（例：「-a -b -c」と「-abc」は同じです）。その場合，最後のオプションには値を指定できます（例：「-abc xyz」の「xyz」は，-c オプションの値となります）。
- 不当なオプション，または指定できる範囲外の値を指定した場合，エラーになります。
- オプション名にはマルチバイト文字は使用できません。

9.2　コマンドおよび制御文の一覧

シェル標準コマンド，シェル拡張コマンド，スクリプト拡張コマンド，スクリプト制御文，およびスクリプト予約語コマンドの概要を説明します。

9.2.1　シェル標準コマンドの一覧

シェル標準コマンドには，特殊組み込みコマンドと正規組み込みコマンドがあります。

表 9-2　特殊組み込みコマンドの一覧

コマンド名	機能概要
.	シェルスクリプトを実行します。
:	引数を展開し，終了コード 0 を返します。
break	繰り返し処理を抜けます。
continue	ループの処理を中断して，ループの先頭に戻ります。
eval	引数を 1 つにまとめて，コマンドとして実行します。
exec	指定されたコマンド実行して終了します。
exit	シェルを終了します。
export	シェル変数をエクスポートします。
readonly	変数の属性を読み込み専用に変更する，または読み込み専用の変数を表示します。
return	関数またはジョブ定義スクリプトから復帰します。
set	シェルオプションを設定する，配列を作成する，または変数の値を表示します。
shift	実行時パラメーターをシフトします。
trap	シグナルや強制終了要求を受けたときの動作を設定します。
typeset	変数や関数の属性と値を明示的に宣言します。
unset	変数の値と属性の設定を解除します。

表 9-3　正規組み込みコマンドの一覧

コマンド名	機能概要
alias	エイリアスを定義します。
builtin	組み込みコマンドを実行します。
cd	カレントディレクトリを移動します。
command	組み込みコマンドや外部コマンドを実行します。
echo	引数で指定した値を標準出力に出力します。

コマンド名	機能概要
false	終了コード 1 を返します。
getopts	引数を解析します。
kill	プロセスにシグナルを送信します。
let	算術式による数値計算を行って，評価します。
print	引数で指定した値を，標準出力に出力します。
pwd	カレントディレクトリのパスを表示します。
read	標準入力から読み込んでシェル変数に格納します。
test	条件式を判定します。
times	シェルが消費した CPU 時間を表示します。
true	終了コード 0 を返します。
ulimit【UNIX 限定】	システムリソースの上限を設定し，情報を表示します。
umask【UNIX 限定】	ファイルモード作成マスクを設定し，表示します。
unalias	エイリアス定義を無効にします。
wait	プロセスの完了を待ちます。
whence	指定された文字列をコマンドとした場合の解釈を表示します。

9.2.2　シェル拡張コマンドの一覧

シェル拡張コマンドの一覧を次の表に示します。すべて正規組み込みコマンドです。

表 9-4　シェル拡張コマンドの一覧

コマンド名	機能概要
adshappexec【Windows 実行環境限定】	アプリケーション実行エージェントに実行アプリケーションの起動要求をします。
adshappexec【Windows 開発環境限定】	開発環境でのデバッグのために，アプリケーション実行エージェントを経由しないで実行アプリケーションを起動します。
adshcmdrc	コマンドの終了コードのしきい値を定義します。
adshecho	ユーザー応答機能で，指定した事象通知メッセージを JP1 イベントとして発行します。
adshjoberr	ジョブおよびジョブステップにエラーを通知します。
adshmktemp	ほかと重ならない名前を持つファイルを作成します。
adshparsecsv	CSV データを解析します。

コマンド名	機能概要
adshparsejson	JSON データを解析します。
adshread	ユーザー応答機能で，指定した応答要求メッセージを応答待ちイベントとして発行します。
adshscripttool【Windows 限定】	ジョブ定義スクリプトを作成しやすくするための情報の取得や出力などを実行します。
adshvarconv	変数の値を変換します。

9.2.3　スクリプト拡張コマンドの一覧

スクリプト拡張コマンドの一覧およびジョブ内での指定個数の上限値を次の表に示します。

表 9-5　スクリプト拡張コマンドの一覧

コマンド名	機能概要	指定個数の上限値※
#-adsh_file	通常ファイルの割り当ておよび後処理をします。	4,095
#-adsh_file_temp	一時ファイルの割り当ておよび後処理をします。	4,095
#-adsh_job	ジョブ名を宣言します。	1
#-adsh_job_stop	ジョブの打ち切り条件を定義します。	1,023
#-adsh_path_var	パス名を扱うシェル変数を定義します。	1
#-adsh_rc_ignore	常に正常終了するコマンドを定義します。	1,023
#-adsh_script	実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルを呼び出します。	4,095
#-adsh_spoolfile	プログラム出力データファイルの割り当てをします。	ジョブ内：4,095 1 つのジョブステップ内：255 ジョブステップ外：255
#-adsh_step	ジョブステップを定義します。#-adsh_step_start，#-adsh_step_end および#-adsh_step_error コマンドがあります。	4,095

注※

ジョブ内の指定個数の上限は，adshexec コマンドの引数に指定したジョブ定義スクリプトファイルに定義されたスクリプトと，そのスクリプトから呼び出される#-adsh_script コマンドによる外部スクリプトで指定された個数の合計です。外部スクリプトには，ネストで呼び出しているスクリプトも含みます。

9.2.4　スクリプト制御文の一覧

スクリプト制御文の一覧を次の表に示します。

表 9-6　スクリプト制御文の一覧

制御文	機能概要
case	文字列の内容に応じて，幾つかある処理のうち，1 つを実行します。
for	値を順次変化させながら，同じ処理を繰り返し実行します。
if	条件に合わせて分岐することで，各条件に沿った処理を実行します。
until	条件が成立するまで，同じ処理を繰り返し実行します。
while	条件が成立している間，同じ処理を繰り返し実行します。

9.2.5　スクリプト予約語コマンドの一覧

スクリプト予約語コマンドの一覧を次の表に示します。

表 9-7　スクリプト予約語コマンドの一覧

コマンド名	機能概要
time	コマンドの実行時間を表示します。

9.3　シェル標準コマンド

シェル標準コマンドは，次の 2 つの組み込みコマンドに分類されます。組み込みコマンドはシェル本体に組み込まれたコマンドであり，シェル自身によって実行できます。

- 特殊組み込みコマンド
  コマンドの構文を誤るとコマンドを実行しているシェルが終了します。
- 正規組み込みコマンド
  コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

9.3.1　.コマンド（シェルスクリプトを実行する）

形式

```
. filename [args]
```

機能

カレントシェル上でシェルスクリプトを実行します。filename に指定されたシェルスクリプトをシェル環境で実行します。filename で指定されたシェルスクリプトのカレントディレクトリは，.（ドット）コマンドを使用したカレントシェルのカレントディレクトリと同じになります。

カレントシェル環境で実行するため，filename のシェルスクリプトが終了したあとも，filename のシェルスクリプト内で設定された変数および定義された関数を使用できます。また，filename のシェルスクリプト開始前に設定された変数および定義された関数を，filename のシェルスクリプト内で使用できます。ただし，filename に指定したシェルスクリプト内でスクリプト拡張コマンドは使用できません。スクリプト拡張コマンドが記述されていた場合は，コメントとして扱います。スクリプト拡張コマンドについては，「9.5　スクリプト拡張コマンド」を参照してください。

引数

filename

カレントシェル上で実行するシェルスクリプトのファイル名を指定します。

args

filename のシェルスクリプト内で使用する位置パラメーターを指定します。args の指定の有無または filename 内での位置パラメーターの変更の有無によって，位置パラメーターは次のように値が設定されます。

args の指定	filename 内の位置パラメーターの変更あり	filename 内の位置パラメーターの変更なし
あり	• filename 内の位置パラメーターの値は引数 args に指定した値になります。	左の説明と同じです。

args の指定	filename 内の位置パラメーターの変更あり	filename 内の位置パラメーターの変更なし
あり	• filename 終了後の位置パラメーターの値は．（ドット）コマンド実行直前の値になります。	左の説明と同じです。
なし	• filename 内の位置パラメーターの値は．（ドット）コマンド実行直前の値になります。 • filename 終了後の位置パラメーターの値は filename 内で変更した値になります。	• filename 内の位置パラメーターの値は．（ドット）コマンド実行直前の値になります。 • filename 終了後の位置パラメーターの値は．（ドット）コマンド実行直前の値になります。

終了コード

終了コード	意味
0～255	正常終了 • 実行したスクリプトの終了コードが設定されます。
0	正常終了 • 引数 filename を指定しないで実行しました。
1	エラー終了 • 引数 filename に通常ファイル以外を指定しました。 • 引数 filename に指定されたファイルを読み込むことができませんでした。

注意事項

- このコマンドで指定したシェルスクリプトは，スプールディレクトリのスクリプトイメージファイルには出力されません。実行履歴としてスクリプトイメージファイルに出力したい場合は，#-adsh_script コマンドを使用してください。
- このコマンドが正常終了した場合，コマンドの実行結果はジョブ実行ログに出力されません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用されません。呼び出した外部スクリプトの実行結果を参照してください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- カレントシェル上でシェルスクリプトを実行します。

```
. ./test.sh
```

9.3.2 :コマンド（引数を展開する）

形式

```
: [arguments]
```

機能

引数を展開し，終了コード 0 を返すコマンドです。

例えば if 文では，else 節や elif 節は省略できても then 節は省略できません。このとき then 節には，条件に合致した場合は何もしないことを示す:コマンドを次のように指定します。

```
if [ 条件式 ]; then
  :                    # 条件式の結果が真の場合，何もしない。
else
  cmd1                 # 条件式の結果が偽の場合，cmd1を実行する。
fi
```

引数

arguments

指定した arguments は次のように展開されます。

ジョブ定義スクリプトの指定例

```
set -x
NUMBER=1
: $NUMBER
```

標準エラー出力への出力結果

```
+ NUMBER=1
+ : 1                    # 変数NUMBERの展開結果が出力される
```

そのため，引数 arguments に変数置換を指定することで，変数に値が格納されているか確認し，格納されていない場合は値を代入する処理ができます。

ジョブ定義スクリプトの指定例

```
STRING01=ABC
: ${STRING01:=DEF}          # 変数STRING01はABCが格納されているため，そのまま
: ${STRING02:=GHI}          # 変数STRING02は未定義のため，GHIが代入される
echo $STRING01 $STRING02    # 標準出力に”ABC GHI”が出力される
```

ただし，引数 arguments に${variable:?word}または${variable?word}の書式で変数置換が指定されていて，variable に値が格納されていない場合は，終了コード 1 でエラー終了します。

なお，このコマンドには指定可能なオプションが存在しません。オプションが指定された場合でも無視して処理を実行します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

終了コード	意味
1	• 引数に${variable:?word}の書式による変数置換が指定されていて variable が定義されていない，または variable に値が格納されていません。 • 引数に${variable?word}の書式による変数置換が指定されていて variable が定義されていません。

注意事項

- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- 引数だけを展開します。

```
: $test
```

9.3.3　alias コマンド（エイリアスを定義する）

形式

```
alias [-p|-x|+p|+x] [name [=value] ...]
```

機能

エイリアスを定義または定義されているエイリアスを標準出力に出力します。引数を 1 つも指定しなかった場合は，現在定義されているエイリアスの名称と値を出力します。

引数

-p
　定義済みのエイリアスを，「alias **エイリアス名**=**値**」の書式で出力します。

-x|+x
　エクスポートされたエイリアスを定義または出力します。

+p
　定義済みのエイリアスを，「alias **エイリアス名**」の書式で出力します。

name
　定義または出力するエイリアス名を指定します。エイリアスを出力する場合は，定義されているエイリアス名を name に指定します。

value
　name に指定されたエイリアス名に設定する内容を指定します。エイリアスを定義する場合は，「**エイリアス名**=**値**」の書式で指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了 • 定義されていないエイリアスを出力しようとしました。

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- 定義されているエイリアス（functions）を出力します。

  ジョブ定義スクリプトの内容

```
alias functions='typeset -f'
alias functions
```

  実行ジョブの STDOUT ファイルの内容

```
********   実行ジョブのSTDOUTファイルの内容     ********
functions='typeset -f'
```

9.3.4　break コマンド（繰り返し処理を抜ける）

形式

```
break  [n]
```

機能

for 文や while 文などの繰り返し処理を，指定した数だけ抜けます。1 つも繰り返し処理に含まれていない状態で実行した場合，メッセージを出力し，正常終了します。

引数

n

繰り返し処理を抜ける数を 1 以上の整数で指定します。

n に指定した数の繰り返し処理を抜けます。**n** を指定しなかった場合，1 段外側の繰り返し処理を抜けます。

n に繰り返し処理の数より大きな値を指定し実行した場合，最上位の繰り返し処理まで抜けた上でメッセージを出力し，正常終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了 • n に 0 を指定しています。 • n に数字以外を指定しています。 • n に負の値，またはオプション文字列（-英数字）を指定しています。

注意事項

- n に 0 以下，または 2,147,483,647 より大きな値を指定すると桁あふれが発生し，あふれた桁を無視した値で動作します。1～2,147,483,647 の範囲内の値を指定してください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- 繰り返し処理を 2 段抜けます。

```
break 2
```

9.3.5　builtin コマンド（組み込みコマンドを実行する）

形式

```
builtin [command [args ...] ]
```

機能

引数を指定して組み込みコマンドを実行します。

引数

command

実行する組み込みコマンド名を指定します。引数 command を指定しなかった場合，正常終了します。

args

組み込みコマンドの引数を指定します。

終了コード

終了コード	意味
0	正常終了 • 組み込みコマンドが正常終了しました。
1	エラー終了 • 引数 command に組み込みコマンド以外を指定した，またはコマンドがエラーで終了しました。

注意事項

- このコマンドには指定可能なオプションが存在しません。そのため，引数にオプションを指定した場合，そのオプションを引数 command として解釈し，ジョブはエラー終了します。
- このコマンドの実行結果はジョブ実行ログに出力されません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用されません。呼び出したコマンドの実行結果を参照してください。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- builtin コマンドで組み込みコマンド pwd を実行します。

  ジョブ定義スクリプトの内容

```
cd /tmp
builtin pwd
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
/tmp
```

9.3.6　cd コマンド（カレントディレクトリを移動する）

形式 1

```
cd ［ディレクトリパス］
```

形式 2

```
cd old new
```

機能

カレントディレクトリを移動します。移動先を指定する方法として 2 つの形式が使用できます。

1 つ目の形式は，移動先のディレクトリパスを指定する方法です。CDPATH 変数が定義されている場合は，定義された位置から移動するディレクトリを特定します。CDPATH 変数が定義されていない場合は，カレントのディレクトリから移動するディレクトリを特定します。

2 つ目の形式は，カレントディレクトリパス名に含まれる文字列の中で，**old** と一致する文字列を**new** に置き換えたディレクトリパスに移動します。

引数

ディレクトリパス

移動するディレクトリパスを指定します。

ディレクトリパスを指定しなかった場合は，ユーザーのホームディレクトリ（HOME 変数）に移動します。ディレクトリパスにハイフン（-）を指定した場合は，直前の作業ディレクトリ（OLDPWD 変数）に移動します。

old

カレントディレクトリパス名に含まれる文字列の中で，置換対象となる文字列を指定します。

new

カレントディレクトリパス名に含まれる文字列 old に対して置換する文字列を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- Windows の場合，cd コマンドを実行するとディレクトリ区切り文字は「/」から「¥」に変換されます。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- HOME 変数が定義されていない場合，引数を指定しないで cd コマンドを実行するとエラーになります。
- ディレクトリパス名に UNC 形式の名称は指定できません。
- 形式 2 でカレントディレクトリパス名に複数含まれる文字列を置換対象として指定した場合，最初の文字列だけが置換の対象になります。例を次に示します。

  （例）カレントディレクトリパス名が/home/user/test/test の場合

  ```
  cd test tmp
  ```

  この場合，cd コマンドは/home/user/tmp/tmp ではなく，/home/user/tmp/test に移動しようとします。

使用例

- /var/log から/var/lib に移動します。

  ジョブ定義スクリプトの内容

```
pwd
cd log lib
pwd
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
/var/log
/var/lib
/var/lib
```

9.3.7　command コマンド（コマンドを実行する）

形式

【UNIX】

```
command  [-p]  [command  [args ...] ]
command  [-v|-V]  [-p]  [command  [command ...] ]
```

【Windows】

```
command  [-w]  [command  [args ...] ]
command  [-v|-V]  [command  [command ...] ]
```

機能

コマンドや組み込みコマンドを実行します。

args を引数として**command** に指定されたコマンドを実行します。

-v オプションを指定した場合，whence コマンドと同じコマンドのパス名を標準出力に出力します。-V オプションを指定した場合，whence -v コマンドと同じコマンドの解釈を標準出力に出力します。-v オプションと-V オプションを同時に指定した場合は，-V オプションが有効になります。

出力形式については，「9.3.35　whence コマンド（文字列をコマンドとした場合の解釈を表示する）」を参照してください。

引数

-p【UNIX 限定】

command に指定されたコマンドを標準パスで検索します。

-w【Windows 限定】

Windows で外部コマンドを実行する場合に次の処理をスキップします。

- 引数内の"（ダブルクォーテーション）の前の「¥」を「¥¥」に変換する処理
- 引数内の"（ダブルクォーテーション）の前の「¥」を付与する処理
- 引数を"（ダブルクォーテーション）で囲む処理

ただし，**command** に指定した文字列については，このオプションを指定した場合でも上記の処理を実施します。

-v

command に指定された文字列をコマンドとして扱った場合のコマンドパスを出力します。

-V

command に指定された文字列がコマンド，予約語，エイリアス，シェル標準コマンド，シェル拡張コマンドまたは関数かどうかを出力します。

command

実行するコマンド名またはコマンドとして扱う文字列を指定します。引数**command** を指定しなかった場合，何も実行しないで正常終了します。

args

command に指定したコマンドの引数を指定します。

終了コード

-v，または-V を指定しなかった場合

終了コード	意味
0	正常終了
127	エラー終了 • コマンドを特定できません。
上記以外	エラー終了 • コマンドの形式が不正か，またはコマンドがエラーで終了しました。

-v，または-V を指定した場合

終了コード	意味
0	正常終了
1	エラー終了。または command に指定したコマンドのどれかが見つかりませんでした。

注意事項

- 引数 command にシンボリックリンクを指定した場合，実行できるかどうかの判定では，シンボリックリンクとリンク先の両方の実行権限を判定します。【Windows 版】

- このコマンドが正常終了した場合，コマンドの実行結果はジョブ実行ログファイルに出力されません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用されません。標準出力および呼び出したコマンドの実行結果を参照してください。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- command コマンドで pwd コマンドを実行します。

  ジョブ定義スクリプトの内容

  ```
  command -p pwd
  ```

  実行ジョブの STDOUT ファイルの内容

  ```
  ********    実行ジョブのSTDOUTファイルの内容      ********
  /tmp
  ```

9.3.8 continue コマンド（繰り返し処理を中断して繰り返し処理の先頭に戻る）

形式

```
continue  [n]
```

機能

for 文や while 文などの繰り返し処理を中断して，繰り返し処理の先頭に戻ります。**n** には繰り返し処理を中断する数を指定します。1 つも繰り返し処理に含まれていない状態で実行した場合，メッセージを出力し，正常終了します。

引数

n

繰り返し処理を中断する数を 1 以上の整数で指定します。

n に指定した数の繰り返し処理を中断して，繰り返し処理の先頭に戻ります。**n** を指定しなかった場合，1 段分繰り返し処理を中断し，繰り返し処理の先頭に戻ります。

n に繰り返し処理の数より大きな値を指定した場合，最上位の繰り返し処理の先頭に戻った上でメッセージを出力し，正常終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了 • n に 0 を指定しました。 • n に数字以外の値を指定しました。 • n に負の数値またはオプション文字列（-英数字）を指定しました。

注意事項

- n に 0 以下，または 2,147,483,647 より大きな値を指定すると桁あふれが発生し，あふれた桁を無視した値で動作します。1～2,147,483,647 の範囲内の値を指定してください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- 2 段分の繰り返し処理を中断します。

```
continue 2
```

9.3.9　echo コマンド（引数で指定した内容を標準出力に出力する）

形式

```
echo [-n] [-e|-E] [args ...]
```

機能

引数で指定した内容を標準出力に出力します。

出力する場合は，¥で始まるエスケープ文字を置き換えます。エスケープ文字を置き換えたときの意味を次の表に示します。

エスケープ文字	意味
¥a	アラート文字（ベル）
¥b	バックスペース文字
¥c	行末の改行を抑止する（¥c の後ろに指定した文字は出力されない）
¥f	フォームフィード文字（改ページ）
¥n	改行文字

エスケープ文字	意味
¥r	復帰文字
¥t	タブ文字
¥v	垂直タブ文字
¥0nnn※1	1～3 桁の 8 進数で表された ASCII コードの文字（0～7）
¥xnn※2	1～2 桁の 16 進数で表された ASCII コードの文字（0～9，a～f，A～F）
¥¥	1 つのバックスラッシュ文字

注※1

指定した ASCII コード文字が 1 桁または 2 桁の場合，前に 0 を付けて 3 桁で指定しても，同じ意味として解釈されます。例えば，次の 3 つの指定は同じ意味として解釈され，アラート文字（ベル）が 3 回出力されます。

echo -e "¥07"

echo -e "¥007"

echo -e "¥0007"

注※2

環境設定パラメーター ESCAPE_SEQ_ECHO_HEX に YES を指定した場合だけ有効になります。ESCAPE_SEQ_ECHO_HEX パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「ESCAPE_SEQ_ECHO_HEX パラメーター（16 進数表記の ASCII コード文字をエスケープ文字として解釈するかを定義する）」を参照してください。

指定した ASCII コード文字が 1 桁の場合，前に 0 を付けて 2 桁で指定しても，同じ意味として解釈されます。例えば，次の 2 つの指定は同じ意味として解釈され，改行文字が 2 回出力されます。

echo -e "¥xA"

echo -e "¥x0A"

エスケープ文字を置き換えたい場合は，-e オプションの引数を次の例の 2.のように"（ダブルクォーテーション）または'（シングルクォーテーション）で囲んでください。クォーテーションの有無と，-e オプションや-E オプションの指定によるエスケープ文字の解釈を次の例に示します。

1. 次の例では，標準出力には「ta」と出力されます。

```
echo -e ¥ta
```

2. 次の例では，標準出力には「<タブ文字>a」と出力されます。

```
echo -e ”¥ta”
```

3. 次の例では，標準出力には「ta」と出力されます。

```
echo -E ¥ta
```

4. 次の例では，標準出力には「¥ta」と出力されます。

```
echo -E ”¥ta”
```

コマンドに指定したオプションと引数の解釈

echo コマンドは引数に指定された文字がすべて有効なオプション文字の場合，オプションと解釈します。次のように指定した場合，「eEn」はすべて有効なオプション文字のため，オプションと解釈します。

```
echo -eEn
```

しかし，1 文字でも無効なオプション文字が指定されていた場合は，引数 args として解釈します。次のように指定した場合，「a」は無効なオプション文字のため，引数 args と解釈し，「-eEna」を標準出力に出力します。

```
echo -eEna
```

また，次のように引数をクォーテーションで囲んだ場合，囲まれた文字列は 1 つの引数と解釈します。次のように指定した場合，スペースは無効なオプション文字のため，引数 args と解釈し，「-e a」を標準出力に出力します。

```
echo ”-e a”
```

エスケープ文字の解釈（-e オプションと-E オプション）

-e オプションと-E オプションの指定によって，エスケープ文字の解釈は次のようになります。

- -e オプションを指定すると，エスケープ文字が解釈されます。
- -E オプションを指定すると，エスケープ文字は解釈されません。
- -e，-E オプションの両方を指定すると，最後に指定したオプションに従って動作します。
- -e，-E オプションのどちらも指定しなかった場合は，環境設定パラメーター ESCAPE_SEQ_ECHO_DEFAULT の指定内容に従って動作します。
  ESCAPE_SEQ_ECHO_DEFAULT パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「ESCAPE_SEQ_ECHO_DEFAULT パラメーター（エスケープ文字関連のオプション省略時の echo コマンドの動作を定義する）」を参照してください。

引数

-n

出力の最後で改行しないで，標準出力に出力します。

-e

エスケープ文字を解釈します。解釈するエスケープ文字は環境設定パラメーター ESCAPE_SEQ_ECHO_HEX の指定に従います。エスケープ文字を解釈したい場合は，"（ダブルクォーテーション）または'（シングルクォーテーション）で囲んでください。

-E

エスケープ文字を解釈しません。

args

引数（出力する内容）を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- ASCII コードの 16 進数表記でエスケープ文字を指定すると，エスケープ文字を直接指定した場合と同じ結果となります。

  例えば次のように指定すると，どちらも「a<タブ文字>b」と出力されます。

  ```
  echo -e "a¥tb"
  echo -e "a¥x09b"
  ```

  また，次のように指定すると，out.txt にはどちらも「a<改行文字>b」が出力されますが，改行文字は CR+LF になります。【Windows 限定】

  ```
  echo -e "a¥nb" > out.txt
  echo -e "a¥x0ab" > out.txt
  ```

- ASCII コードの文字列でエスケープ文字を表す場合，ASCII コードの範囲外の値を指定すると，出力される内容は端末に指定された文字コードに従います。そのため，印字不可能文字の場合，正しく出力されないことがあります。
- 引数にパス名を指定する場合は，¥がエスケープ文字として置換されないよう，-E オプションの指定や ESCAPE_SEQ_ECHO_DEFAULT パラメーターに NO の指定をした環境で，echo コマンドを実行してください。

  例えば次のどのコマンドでも，パス名（d:¥a¥b¥c）は正しく出力されません。

  ```
  FILE="d:¥¥a¥¥b¥¥c"
  echo $FILE
  echo "$FILE"
  echo "d:¥¥a¥¥b¥¥c"
  echo 'd:¥a¥b¥c'
  ```

  次のコマンドでは，どれもパス名（d:¥a¥b¥c）が正しく出力されます。

  ```
  FILE="d:¥¥a¥¥b¥¥c"
  echo -E $FILE
  echo -E "$FILE"
  echo -E "d:¥¥a¥¥b¥¥c"
  echo -E 'd:¥a¥b¥c'
  ```

```
FILE="d:¥¥¥¥a¥¥¥¥b¥¥¥¥c"
echo $FILE
echo 'd:¥¥a¥¥b¥¥c'
echo d:¥¥¥¥a¥¥¥¥b¥¥¥¥c
```

使用例

- LANG 変数を出力します。

  ジョブ定義スクリプトの内容

  ```
  echo $LANG
  ```

  実行ジョブの STDOUT ファイルの内容

  ```
  ********    実行ジョブのSTDOUTファイルの内容      ********
  ja_JP.eucJP
  ```

9.3.10　eval コマンド（引数を 1 つにまとめてコマンドとして実行する）

形式

```
eval  [command [args ...] ]
```

機能

引数を 1 つにまとめて，コマンドとして実行します。引数として与えられた文字列をそのままコマンドとして実行します。

引数

command

実行するコマンドのコマンド名を指定します。引数**command** を指定しなかった場合，何も実行しないで正常終了します。

args

1 つにまとめて実行するコマンドの引数を指定します。

終了コード

終了コード	意味
0	正常終了
127	エラー終了 • コマンドを特定できません。
上記以外	エラー終了

終了コード	意味
上記以外	• コマンドの形式が不正か，またはコマンドがエラーで終了しました。

注意事項

- このコマンドの実行結果はジョブ実行ログに出力されません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用されません。呼び出したコマンドの実行結果を参照してください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- /home/adsh/script ディレクトリに移動します。

```
eval cd /home/adsh/script
```

9.3.11　exec コマンド（コマンドを実行して終了する）

形式

```
exec [command [args]...]
```

機能

指定されたコマンドを実行し，終了します。

引数に外部コマンドを指定した場合，そのコマンドを adshexec コマンドの子プロセスとして実行します。外部コマンドが完了するのを待ってから，一時ファイルの削除など，ジョブの後処理を行います。

入出力リダイレクト記号とリダイレクト先だけを指定すると，入出力リダイレクト記号に従って，入出力先を切り替えます。リダイレクトについては，「(8)　入出力リダイレクト」を参照してください。

引数

command

実行するコマンドのコマンド名を指定します。**command** に引数を指定しなかった場合，exec コマンドは何もしないで，ジョブ定義スクリプトの実行を継続します。

args

実行するコマンドの引数を指定します。

終了コード

終了コード	意味
0	正常終了
127	エラー終了 • コマンドを特定できません。
上記以外	エラー終了 • コマンドがエラーで終了しました。

注意事項

- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- ユーザープログラム UAP01 を実行し，ジョブを終了します。

```
exec UAP01
```

- 標準出力先を file01 に切り替えます。

```
exec > file01
```

9.3.12　exit コマンド（シェルを終了する）

形式

```
exit [n]
```

機能

シェルを終了します。このコマンドは，終了コードの値とは関係なく，コマンドの構文が正しいかどうかでコマンドの正常終了およびエラー終了を決定します。

引数を指定しない場合は，最後に実行したコマンドの終了コードをこのコマンドの終了コードとして正常終了します。引数に適切な数値を指定して実行した場合は，正常終了します。引数に数字以外の文字など不適切な値を指定して実行した場合は，エラー終了します。エラー終了のとき，コマンドの終了コードは 1 を返します。

ジョブステップエラーブロック内でこのコマンドを実行したときの動作を次に示します。

- 引数を指定して正常終了した場合は，引数に指定した値がジョブステップの終了コードになります。
- 引数を指定しないで正常終了した場合，または引数を指定してエラー終了した場合は，ジョブステップ正常ブロック内で最後に実行したコマンドの終了コードが，ジョブステップの終了コードになります。

引数

n　～<符号なし整数>((0～255))

シェル終了時の終了コードを指定します。**n** を指定しなかった場合，最後に実行したコマンドの終了コードを返してシェルを終了します。

n に 256 以上の値を指定した場合，**n** の値を 256 で割った余りを終了コードとして，正常終了します。

n に負の値を指定した場合，指定した値の 2 の補数を終了コードとして，正常終了します。

終了コード

終了コード	意味
0～255	正常終了 • **n** または最後に実行したコマンドの終了コードを返します。
1	エラー終了 • **n** に数字以外を指定しました。

注意事項

- **n** には負の値および 256 以上の値も指定できますが，JP1/Advanced Shell では 0～255 の範囲内の値を指定することを推奨します。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。
- 「&」やコマンド置換など，別プロセスで exit コマンドを実行する場合は，「5.1.7　別プロセスでの実行」に示す注意事項もあわせて参照してください。

使用例

- 終了コード 2 でシェルを終了します。

```
exit 2
```

9.3.13　export コマンド（シェル変数をエクスポートする）

形式

```
export [-p] [name [=value] ...]
```

機能

name に指定されたシェル変数をエクスポートします。なお，-p オプションと**name** を同時に指定した場合，**name** のエクスポートが優先されます。

すべてのオプションを指定しないで実行した場合，エクスポートされているすべての変数の変数名を標準出力に出力します。

引数

-p

エクスポートされているすべての変数を「export **変数名**=**値**」の書式で標準出力に出力します。

name

エクスポートする変数の名称を指定します。

Windows でシェル変数をエクスポートする場合，指定できる変数名は次のように異なります。

- VAR_ENV_NAME_LOWERCASE パラメーターに DISABLE を指定した場合

  シェル変数名に小文字が含まれているとエクスポートできないため，変数名に含まれる英字はすべて大文字にする必要があります。

  英小文字が含まれるシェル変数をエクスポートしようとすると，エラーメッセージを出力し，バッチジョブを終了します。

- VAR_ENV_NAME_LOWERCASE パラメーターに ENABLE を指定した場合

  シェル変数名に小文字が含まれていてもエクスポートできます。

  ただし，環境変数は大文字・小文字の区別はなく，最後にエクスポートした同じスペルのシェル変数が最終的な環境変数値となります。また，シェル変数はエクスポートの有無に関係なく，大文字・小文字の相違があれば別々の値を保持します。

name にはエクスポートする変数名および配列名を複数指定できます。**name** に配列名を指定した場合，配列を構成する全要素をエクスポートします。配列の 1 つの要素を指定した場合も，配列の全要素をエクスポートします。

name に未作成の変数を指定した場合，変数の作成とエクスポートを同時に行います。ただし，この場合は**value** を指定しないと**name** には改行文字が代入され，エクスポートされます。

name に指定されたシェル変数が読み込み専用属性でかつ**value** を指定した場合，エラー終了します。

value

name に指定された変数に代入する値を指定します。

name の後ろに=**value** を指定した場合，**name** への値の代入とエクスポートを同時に行います。**value** を指定しなかった場合，**name** に設定されている値でエクスポートされます。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- シェル変数 HOME に"/home/jp1as"を代入してエクスポートします。

```
export HOME=”/home/jp1as”
```

9.3.14　false コマンド（終了コード 1 を返す）

形式

```
false
```

機能

終了コード 1 を返します。

なお，このコマンドには指定可能なオプションが存在しません。オプションが指定された場合でも無視して処理を実行します。

終了コード

終了コード	意味
1	正常終了 • 常に 1 を返します。

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- このコマンドの実行結果は KNAX6113-I メッセージに出力されます。

使用例

- 終了コード 1 を設定します。

  ジョブ定義スクリプトの内容

```
false
echo $?
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
1
```

9.3.15　getopts コマンド（引数を解析する）

形式

```
getopts optstr name  [args ...]
```

機能

指定された引数を解析します。

引数

optstr

有効なオプション文字の文字列を指定します。文字の後ろにコロンが続く場合は，そのオプションがオプションの値を持つことを示します。

optstr には，コマンドラインまたは**args** に指定された引数の中で，有効なオプション文字とする文字列（-a と-b を有効オプションする場合は ab）を指定します。マルチバイト文字は使用できません。

引数が**optstr** と一致した場合，**name** には一致したオプションの文字を格納します。引数が**optstr** と一致しない場合は「?」を格納します。

オプションに値がある場合，オプション文字の後ろに「:」を指定します。「:」を指定すると，一致したオプションの値を OPTARG シェル変数に格納します。getopts コマンドは，OPTIND シェル変数に設定されている引数インデックス（初期値は 1）以降の引数を解析します。例えば，**args** に-a 10 と指定した場合は-a の位置がインデックス 1 となります。

name

getopts コマンドによって一致したオプション文字を格納する変数を指定します。

args

解析対象となる引数を指定します。このオプションを指定しない場合，コマンドラインの引数を解析します。

終了コード

終了コード	意味
0	正常終了
1	• 正常終了（オプションの終了を検出） • エラー終了

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- 「&」やコマンド置換など，別プロセスで getopts コマンドを実行する場合は，「5.1.7　別プロセスでの実行」に示す注意事項もあわせて参照してください。

使用例

- -b を有効オプションとして解析します。

  ジョブ定義スクリプトの内容

```
getopts b: name -b 10
echo $name
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
b
```

9.3.16　kill コマンド（シグナルを送信する）

形式

```
kill  [-s {signame|signum} ]  {pid|-pid} ...
kill  [-signame|-signum]  {pid|-pid} ...
```

機能

プロセスにシグナルを送信します。signame またはsignum に指定したシグナルを，pid に指定したプロセスに送信します。signame またはsignum を指定しなかった場合は，SIGTERM を送信します。シグナル名称には，シグナル名から先頭の「SIG」を除いた名称を指定してください（例：SIGINT であれば「INT」と指定する）。それぞれのシグナルの仕様については，使用している OS のマニュアルを参照してください。

引数

-s

　送信するシグナルをシグナル番号またはシグナル名称で指定します。

signame または-signame

　送信するシグナルのシグナル名称を signame に指定します。

signum または-signum

　送信するシグナルのシグナル番号を signum に指定します。

pid

シグナルを送信するプロセス ID を指定します。

-pid

プロセスグループに属するすべてのプロセスへシグナルを送信します。シグナルを送信するプロセスグループの ID を pid に指定します。Windows の場合，複数のプロセスにシグナルを送る pid に 0 以下の値を指定できません。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- Windows の場合，SIGKILL 以外のシグナルを指定したときは，エラーになります。
- Windows の場合，SIGKILL を指定すると TerminateProcess()によってプロセスを即時終了します。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- -signame または-signum による書式でデフォルトのシグナル（SIGTERM）をプロセスグループ（-pid）に送信する場合は，シグナルとして「--」を指定する必要があります。「--」を指定しないと，プロセスグループ（-pid）を-signum と解釈します。例を次に示します。

  プロセスグループ 14588 にデフォルトのシグナル（SIGTERM）を送信します。

  ```
  kill -- -14588
  ```

使用例

- UNIX でプロセス ID4725 に SIGINT を送信します。

  ```
  kill -INT 4725
  ```

- Windows でプロセス ID4725 を即時終了します。

  ```
  kill -KILL 4725
  ```

9.3.17　let コマンド（数値計算を行って評価する）

形式 1

```
let 算術式 [,算術式 ... ]
```

形式 2

```
((算術式))
```

機能

算術式による数値計算を行って評価します。

また，let コマンドの代わりに，「((算術式))」と記載することで let コマンドと同様に算術式を計算し，評価できます。

let コマンドは，コンマで区切ると算術式を複数指定できます。複数指定した場合，算術式は左から右へ順に計算します。そのため，コンマで区切って指定した算術式を条件式の判定に使用すると，最後に実行した算術式の結果に従って条件判定をします。また，コンマの前後にスペースが存在すると，算術エラーで終了します。演算を括弧でまとめると，演算の優先順位を変更できます。

算術式の詳細については「5.3　算術演算」，条件判定の詳細については「5.2　条件判定」を参照してください。

終了コード

終了コード	意味
0	正常終了 • 算術式の値が 0 以外です。
1	正常終了 • 算術式の値が 0 です。 • 算術式を指定しないで，(( ))を実行しました。
	エラー終了 • 算術式を指定しないで，let コマンドを実行しました。
2	エラー終了 • 算術エラー（ゼロ除算，算術式不正）です。

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- *，&，<，<<，>および>>などの算術演算子はメタキャラクタとして特別な意味を持っています。これらの文字を let コマンドで使用する場合は，メタキャラクタを無効にする必要があります。

  例　1 を 2 ビット左シフトした結果を変数 RC に設定します。

  ジョブ定義スクリプトの内容

  ```
  let "RC=1<<2"
  echo $RC
  ```

実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
4
```

- let コマンドには指定できるオプションがありません。そのため，let コマンドの引数に「-英字」を指定すると，オプションではなく変数名として解釈し動作します。

  例　引数に「-a」を指定すると「-3」と解釈し，終了コードは 0 になります。

  ジョブ定義スクリプトの内容

```
a=3
let -a
echo $?
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
0
```

- 「&」やコマンド置換など，別プロセスで let コマンドを実行する場合は，「5.1.7　別プロセスでの実行」に示す注意事項もあわせて参照してください。

使用例

- 3+4 を行ったあとに 2 を掛けます。

  ジョブ定義スクリプトの内容

```
let "VAR=2*(3+4)"
echo $VAR
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
14
```

- 1+2 の結果を変数 RC に設定します。

  ジョブ定義スクリプトの内容

```
((RC=1+2))
echo $RC
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
3
```

9.3.18　print コマンド（標準出力に出力する）

形式

```
print [-n|-p|-r] [-u [num] ] [--] [args]
```

機能

引数で指定した内容を標準出力に出力します。出力の最後に改行します。

出力する場合は，¥で始まるエスケープ文字を置き換えます。エスケープ文字を置き換えたときの意味を次の表に示します。

エスケープ文字	意味
¥a	アラート文字（ベル）
¥b	バックスペース文字
¥c	行末の改行を抑止する（¥c の後ろに指定した文字は出力されない）
¥f	フォームフィード文字（改ページ）
¥n	改行文字
¥r	復帰文字
¥t	タブ文字
¥v	垂直タブ文字
¥0nnn※	1～3 桁の 8 進数で表された ASCII コードの文字（0～7）
¥¥	1 つのバックスラッシュ文字

注※

指定した ASCII コード文字が 1 桁または 2 桁の場合，前に 0 を付けて 3 桁で指定しても，同じ意味として解釈されます。

-r オプションを指定した場合，エスケープ文字を無視します。

引数

-n

出力の最後で改行しないで，標準出力に出力します。

-p

標準出力ではなく，パイプを使ってバックグラウンドプロセスの標準入力に出力します。

-r

エスケープ文字を無視します。

-u [num]

ファイル識別子 num に出力します。num を指定しない場合，1 が指定されたものとします。

num では，出力先のファイル識別子または p を指定します。num に p を指定した場合，-p オプションを指定したときと同じになります。

--

オプション終端文字です。このオプション以降に指定したオプションは，args として解釈します。

args

引数（出力する内容）を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- ASCII コードの文字列でエスケープ文字を表す時に，ASCII コードの範囲外の値を指定すると，出力される内容は端末に指定された文字コードに従います。そのため，印字不可能文字の場合，正しく出力されないことがあります。

使用例

- 改行文字の付いた文字列 abc を出力します。

  ジョブ定義スクリプトの内容

```
print "abc¥n"
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
abc
```

- バックグラウンドプロセス（coproc.sh）の標準入力に文字列 abc を出力します。

```
coproc.sh |&
print -p abc
```

9.3.19　pwd コマンド（カレントディレクトリのパスを出力する）

形式

```
pwd　[-L|-P]
```

機能

カレントディレクトリのパスを標準出力に出力します。

引数

オプションをすべて指定しなかった場合は，-L オプションが指定されたときと同じになります。

-L

カレントディレクトリがシンボリックリンク（実際のファイルパスを格納したファイルを使ってリンクする）を含むパスの場合，シンボリックリンクを解決しない状態でパスを出力します。-L オプションおよび-P オプションの両方を指定した場合は，最後に指定したオプションに従います。

-P

カレントディレクトリがシンボリックリンクを含むパスの場合，シンボリックリンクを解決した状態でパスを出力します。-L オプションおよび-P オプションの両方を指定した場合は，最後に指定したオプションに従います。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- Windows の場合，pwd コマンドを実行するとディレクトリ区切り文字の「/」は「¥」で表示されます。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- カレントディレクトリのパスを出力します。

  ジョブ定義スクリプトの内容

  ```
  cd /tmp
  pwd
  ```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
/tmp
```

9.3.20　read コマンド（標準入力から読み込んで変数に格納する）

形式

```
read [-p] [-r] [-u [num] ] [varname ...]
```

機能

標準入力から読み込みを行います。標準入力から 1 行読み取り，読み込んだ内容を varname に指定したシェル変数に格納します。

引数

-p

パイプを使用して，バックグラウンドプロセスの出力から読み込みます。

-r

¥（バックスラッシュ）をエスケープ文字として扱いません。

read コマンドは読み込む内容に¥（バックスラッシュ）が含まれていた場合，エスケープ文字として扱います。行中に指定されていた場合は，¥（バックスラッシュ）の後ろがシェル変数 IFS で定義した区切り文字であっても区切り文字とは扱いません。行末に指定されていた場合は，次の行を継続して読み込みます。¥（バックスラッシュ）をエスケープ文字として扱わない場合は，-r オプションを使用してください。

-u [num]

ファイル識別子 num から読み込みます。num を指定しない場合，標準入力から読み込みます。

num では，読み込みを行うファイル識別子または p を指定します。num に p を指定した場合，-p オプションを指定したときと同じになります。

varname

読み込んだ内容を格納する変数名を指定します。

varname を複数指定した場合，入力行を IFS 変数を区切り文字としてフィールド分割し，分割したフィールドを varname に順次格納します（1 つ目の varname には入力行の最初のフィールドを格納し，2 つ目の varname には 2 つ目のフィールドを格納します）。

フィールド数が varname に指定した変数よりも多い場合は，最後に指定した変数に残りの全フィールドの値を格納します。

フィールド数が変数よりも少ない場合は，残りの変数に改行文字を格納します。

終了コード

終了コード	意味
0	正常終了
1	正常終了 • ファイルの終了（EOF）を検出しました。 エラー終了 • 上記以外です。

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- 「&」やコマンド置換など，別プロセスで read コマンドを実行する場合は，「5.1.7　別プロセスでの実行」に示す注意事項もあわせて参照してください。
- Windows の場合，エディタからのデバッグ実行でコンソールまたはコマンドプロンプトから入力などをするときは，ブレークポイントでの停止中や先行コマンドの実行中など，read コマンドが開始する前であってもキー入力が受け付けられます。その結果，入力内容を正しく読み込めなくなるため，read コマンドの開始前にキー入力をしないでください。

  read コマンドの開始前にキー入力した内容は，read コマンド開始時に表示されます。その場合は表示内容をすべて削除してから再度入力してください。
- read コマンドは，標準入力から読み込んだ文字列の改行コードが［LF］の場合，［LF］を取り除いた内容を変数に格納します。また，標準入力から読み込んだ文字列の改行コードが［CR］＋［LF］の場合，［CR］と［LF］を取り除いた内容を変数に格納します。

使用例

- ファイル string.txt の内容を読み込み，標準出力に出力します。

  ジョブ定義スクリプトの内容

```
while read LINE
do
 echo "$LINE"
done     < string.txt
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
HITACHI
JP1
Advanced Shell
```

- バックグラウンドプロセス（coproc.sh）の標準出力に出力された文字列を変数 NAME に読み込みます。

```
coproc.sh |&
read -p NAME
```

9.3.21 readonly コマンド（変数の属性を読み込み専用に変更する，または読み込み専用の変数を表示する）

形式

```
readonly [-p] [name [=value] ...]
```

機能

変数の属性を読み込み専用に変更する，または読み込み専用の変数を表示します。すべてのオプションを指定しないで実行した場合，読み込み専用のすべての変数名を標準出力に出力します。

読み込み専用に変更した変数の属性を再び書き込み可能に変更する場合は，typeset コマンドに+r オプションを指定してください。

引数

-p

読み込み専用属性のすべての変数を「readonly **変数名**=**値**」の書式で標準出力に出力します。ただし，-p オプションと**name** を同時に指定した場合，**name** の属性を読み込み専用に変更する方が優先されます。

name

属性を読み込み専用に変更する変数の名称を指定します。

name に指定された変数の属性を読み込み専用に変更します。**name** には変数名または配列名を複数指定できます。ただし，**name** に関数名を指定しても関数の属性は読み込み専用に変更されないで，関数名と同じ名称の変数が読み込み専用になります。

name に配列名を指定した場合，配列を構成する全要素を読み込み専用に変更します。配列の 1 つの要素を指定した場合も配列の全要素を読み込み専用に変更します。

name に未作成の変数を指定した場合，変数の作成と読み込み専用への属性の変更を同時に実行します。この場合，value を指定しないと**name** には改行文字が代入され，読み込み専用に変更されます。

name に指定された変数の属性がすでに読み込み専用の場合，何もしないで正常終了します。

value

name に指定された変数に代入する値を指定します。

name の後ろに=**value** を指定すると，**name** への値の代入と読み込み専用への変更を同時に行います。**value** を指定しなかった場合，**name** に設定されている値のまま属性を読み込み専用に変更します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- 変数 test の属性を読み込み専用に変更します。

```
readonly test
```

9.3.22　return コマンド（関数または外部スクリプトから復帰する）

形式

```
return [n]
```

機能

関数または外部スクリプトから復帰し，呼び出し元の処理を継続します。ただし，関数の外かつ外部スクリプトではない位置でこのコマンドを実行すると，シェルを終了します。

このコマンドは，終了コードの値とは関係なく，コマンドの構文が正しいかどうかでコマンドの正常終了およびエラー終了を決定します。

引数を指定しない場合は，最後に実行したコマンドの終了コードをこのコマンドの終了コードとして正常終了します。引数に適切な数値を指定して実行した場合は，正常終了します。引数に数字以外の文字など不適切な値を指定して実行した場合は，エラー終了します。エラー終了のとき，コマンドの終了コードは 1 を返します。

引数

n　～<符号なし整数>((0～255))

復帰時の終了コードを指定します。

n を指定しなかった場合，最後に実行したコマンドの終了コードを返して復帰します。

n に 256 以上の値を指定した場合，**n** の値を 256 で割った余りを終了コードとして，正常終了します。

n に負の値を指定した場合，指定した値の 2 の補数を終了コードとして，正常終了します。

終了コード

終了コード	意味
0～255	正常終了 • **n** または最後に実行したコマンドの終了コードを返します。
1	エラー終了

終了コード	意味
1	• n に数字以外を指定しました。

注意事項

- n には負の値および 256 以上の値も指定できますが，JP1/Advanced Shell では 0～255 の範囲内の値を指定することを推奨します。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。
- 「&」やコマンド置換など，別プロセスで return コマンドを実行する場合は，「5.1.7　別プロセスでの実行」に示す注意事項もあわせて参照してください。

使用例

- 関数または外部スクリプトから終了コード 2 で復帰し，呼び出し元の処理を継続します。

```
return 2
```

9.3.23　set コマンド（シェルオプションを設定する，配列を作成する，または変数の値を表示する）

形式

```
set  [-a|+a]  [-f|+f]  [-u|+u]  [-v|+v]  [-x|+x]
     [ {-o|+o}  [opt] ] ...
     [ {-A|+A} name]  [--]  [val ...]
     [ {-D|+D} name]  [--]  [{ val ... } ...]
```

機能

シェルオプションの設定，配列の作成，または変数の値を表示します。

このコマンドは，シェルオプションの設定と配列の作成を同時に指定できます。同時に指定する場合は，シェルオプション，配列の作成の順に指定してください。配列の作成を先に指定すると，-A オプションおよび-D オプション以降の内容をname およびval として解釈します。

引数

オプションは同時に複数指定できます。同じオプションを複数回指定した場合，最後に指定した内容が設定されます。

オプションを指定しないで実行すると，割り当てられているすべての変数が「変数名=値」の書式で標準出力に出力されます。

-a|+a

- -a：allexport オプションを有効にします。
- +a：allexport オプションを無効にします。

-f|+f

- -f：noglob オプションを有効にします。
- +f：noglob オプションを無効にします。

-u|+u

- -u：nounset オプションを有効にします。
- +u：nounset オプションを無効にします。

-v|+v

- -v：verbose オプションを有効にします。有効にすると，コマンドや制御文などの区分に関係なく，ファイルから入力した行をすべて出力します。出力する内容を次に示します。
  - ・コメント
  - ・存在しないコマンド
  - ・if 文や case 文の条件を満たさないため，実行されないコマンド
  - ・while 文や for 文のループに一度も入らないで，実行されなかったコマンド
  - ・run 属性によってスキップされたジョブステップ
- +v：verbose オプションを無効にします。

-x|+x

- -x：xtrace オプションを有効にします。
- +x：xtrace オプションを無効にします。

-o|+o

- -o：opt に指定されたシェルオプションを有効にします。また，現在設定されているシェルオプションの一覧を表示します。
- +o：opt に指定されたシェルオプションを無効にします。また，現在有効に設定されているシェルオプションをコマンドラインに入力できる書式で表示します。

opt

設定するシェルオプションの名称を指定します。指定できるシェルオプションの名称については，「5.6 シェルオプション」を参照してください。

-A|+A

1 次元配列に値を代入する場合に指定します。

-A オプションを指定し実行すると，name に指定した配列や変数がすでに存在している場合，name の内容を消去してから，name に指定した配列や変数へ val に指定した値を代入して作成します。val に指定した引数の数だけ，配列の要素を作成します。

+A オプションを指定し実行すると，name に指定した配列や変数が存在していても name の内容は消去しないで，name に指定した配列や変数へ val に指定した値を代入します。代入する要素数が既存の配列の要素数より少ない場合，代入されない配列要素の値は変更されません。name に指定した配列や変数がすでに存在しない場合は，-A オプションを指定した場合と同じ動作となります。

作成できる配列の要素数は 2 から 65,536 で，val の個数が 1 つの場合は配列ではなく変数を作成します。また，複数の配列を同時に作成できません。

-D|+D

2 次元配列に値を代入する場合に指定します。

-D オプションを指定し実行すると，name に指定した配列や変数がすでに存在している場合，name の内容を消去してから，name に指定した配列や変数へ val に指定した値を代入して作成します。val に指定した引数の数だけ，配列の要素を作成します。

+D オプションを指定し実行すると，name に指定した配列や変数が存在していても name の内容は消去しないで，name に指定した配列や変数へ val に指定した値を代入します。代入する要素数が既存の配列の要素数より少ない場合，代入されない配列要素の値は変更されません。name に指定した配列や変数がすでに存在しない場合は，-D オプションを指定した場合と同じ動作となります。

作成できる配列の要素数は 65,536×64 で，val の個数が 1 つの場合は配列ではなく変数を作成します。また，複数の配列を同時に作成できません。

name

割り当てる配列の名称を指定します。name に読み込み専用属性の変数を指定した場合，エラー終了します。

--

オプション終端文字です。このオプション以降に指定したオプションは，val として解釈します。

val

1 次元配列の要素に代入する値を指定します。val だけを指定し実行すると，val に指定された値は位置パラメーターに代入されます。val を複数指定した場合は，左から$1，$2...の順序で代入されます。

2 次元配列の要素に代入する値を指定する場合は，括弧 { } で囲んで指定する必要があります。括弧と val の間には 1 つ以上の空白文字を入れる必要があります。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。
- braceexpand オプションと noglob オプションを同時に指定した場合，noglob オプションが優先的に適用され，ブレース展開は無効になります。

使用例

- 1 次元配列 test を作成し，test[0]に a01，test[1]に a02，test[2]に a03 を代入します。

```
set -A test a01 a02 a03
```

- 2 次元配列 test2 を作成し，test[0][0]に a01，test[0][1]に a02，test[0][2]に a03 を代入します。

```
set -D test { a01 a02 a03 }
```

9.3.24　shift コマンド（実行時パラメーターをシフトする）

形式

```
shift [n]
```

機能

実行時パラメーターをシフトします。実行時パラメーターをシフトした場合，シフトの数だけ先頭から移動します。

引数

n

実行時パラメーターをシフトする数を指定します。n を指定した場合，n に指定した数だけ実行時パラメーターをシフトします。n を指定しなかった場合，引数を 1 シフトします。n に 0 を指定すると，実行時パラメーターはシフトされません。n に負の値，または数値以外を指定した場合，エラー終了します。実行時パラメーターの個数より大きい値を指定した場合，エラー終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- n に 0 を指定すると実行時パラメーターはシフトされません。実行時パラメーターを for 文や while 文を終了するための条件に使用する場合，shift コマンドの引数に 0 を指定しないでください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- 実行時パラメーターを 2 つシフトします。

```
shift 2
```

9.3.25　test コマンド（条件式を判定する）

形式 1

```
test 条件式
```

形式 2

```
[ 条件式 ]
```

形式 3

```
[[ 条件式 ]]
```

機能

条件式を判定します。条件判定の演算子を用いて記述した条件式を判定し，判定結果が真のときは 0 を返し，判定結果が偽のときは 1 を返します。条件式を指定しないで test コマンドおよび[ ]を実行した場合も 1 を返します。

条件式については，「5.2　条件判定」を参照してください。

終了コード

終了コード	意味
0	正常終了 • 条件式の判定結果が真です。
1	正常終了 • 条件式の判定結果が偽です。
2	エラー終了 • コマンドがエラー終了しました。

注意事項

- 「<」および「>」などの演算子はメタキャラクタとして特別な意味を持っています。これらの文字を test コマンドで使用する場合は，メタキャラクタを無効にする必要があります。
- [[ ]]の場合，[[と]]の間にある文字列に対して，ワイルドカードやファイル名置換は適用されません。例を次に示します。この例では，カレントディレクトリに「test.ash」「hhh」というファイルが存在すると仮定しています。

```
[[ -f *est.ash ]]          …(1)
[ -f *est.ash ]            …(2)
test -f *est.ash           …(3)
[[ -f ?(hhh) ]]            …(4)
[ -f ?(hhh) ]              …(5)
test -f ?(hhh)             …(6)
```

この例の(1)，(4)の場合，[[ ]]で囲まれたワイルドカードは適用されないため，該当するファイルは存在しないと解釈され，終了コードは 1 となります。

(2)，(3)，(5)，(6)の場合はワイルドカードが適用されるため，条件式の判定結果は真となり，終了コードは 0 となります。

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- 変数 arg1 と変数 arg2 の値が同じかどうか比較します。

```
test $arg1 -eq $arg2
```

9.3.26　times コマンド（シェルが消費した CPU 時間を出力する）

形式

```
times
```

機能

シェルとシェルから起動したプロセスの CPU 時間を標準出力に出力します。次に示す情報を出力します。

- シェルが消費したユーザー CPU 時間（秒）
- シェルが消費したシステム CPU 時間（秒）
- シェルから起動したプロセスが消費したユーザー CPU 時間（秒）の合計
- シェルから起動したプロセスが消費したシステム CPU 時間（秒）の合計

times コマンドの出力形式を次の表に示します。

出力形式※	内容
Shell: CPU時間 user CPU時間 system	シェルが消費した CPU 時間を，ユーザー CPU 時間，システム CPU 時間の順に出力します。
Kids: CPU時間 user CPU時間 system	シェルから起動したプロセスが消費した CPU 時間を，ユーザー CPU 時間，システム CPU 時間の順に出力します。

注※

CPU 時間は小数点第 2 位まで出力します。

なお，このコマンドは書式の判定は行わないで，不当なオプションが指定された場合でも無視して処理を実行します。

終了コード

終了コード	意味
0	正常終了

注意事項

- Windows の場合，子プロセスの CPU 時間に孫プロセスの CPU 時間を含みません。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- シェルとシェルから起動したプロセス（ps コマンド）の CPU 時間を出力します。

  ジョブ定義スクリプトの内容

```
ps > /dev/null
times
```

  実行ジョブの STDOUT ファイルの内容

```
********   実行ジョブのSTDOUTファイルの内容    ********
Shell:     0.01s user     0.02s system
Kids:      0.00s user     0.01s system
```

9.3.27 trap コマンド（シグナルや強制終了要求を受けたときの動作を設定する）

trap コマンドは UNIX 版と Windows 版で提供する機能が異なります。UNIX 版で提供する機能については，「(1) trap コマンド【UNIX 版】」を参照してください。Windows 版で提供する機能については，「(2) trap コマンド【Windows 版】」を参照してください。

(1) trap コマンド【UNIX 版】

形式

```
trap [action] [signal ...]
```

機能

シグナルを受け取ったときの動作を設定します。signal に指定されたシグナルをシェルが受け取ると，action に指定された動作を実行します。

引数を指定しないで実行した場合，シグナルに設定されている動作を次の形式で標準出力に出力します。

シグナル種別	出力形式
名称が定義されているシグナルの場合	trap -- action "先頭の SIG を除いたシグナル名"
名称が定義されていないシグナルの場合	trap -- action UNKNOWN SIGNAL

1 つのシグナル番号に複数の名称が定義されているシグナルに対する動作を次に示します。

【Linux 限定】

シグナル名称	別名称	trap コマンドによる action の設定	
SIGSYS	SIGUNUSED	SIGSYS	設定できます
		SIGUNUSED	設定できます

注

拡張機能では，SIGUNUSED ではなく，SIGSYS を主なシグナル名称として扱っているため，trap コマンドでも SIGSYS をシグナル名称として扱います。

【AIX 限定】

シグナル名称	別名称 1	別名称 2	trap コマンドによる action の設定	
SIGABRT	SIGLOST	SIGIOT	SIGABRT	設定できます
			SIGLOST	設定できます
			SIGIOT	設定できます
SIGIO	SIGPOLL	なし	SIGIO	設定できます
			SIGPOLL	設定できます

【HP-UX，Solaris 限定】

シグナル名称	別名称	trap コマンドによる action の設定	
SIGABRT	SIGIOT	SIGABRT	設定できます
		SIGIOT	設定できます
SIGIO	SIGPOLL	SIGIO	設定できます
		SIGPOLL	設定できます

また，1 つのシグナル番号に複数の名称が定義されているシグナルに対して，trap コマンドでaction を設定した場合，出力するシグナル名はどれか 1 つのシグナル名になります。

引数

action

指定されたシグナルを受け取ったときの動作を指定します。

action にハイフンを指定した場合，**signal** に一致する指定済みのトラップがリセットされ，デフォルトに戻ります。**action** を指定しないで，**signal**にシグナル番号を指定した場合も同様にデフォルトに戻ります。

action に""を指定した場合，**signal** に指定されたシグナルを無視（SIG_IGN）します。

ただし，SIGTERM については，action に""を指定しても，シグナルを無視（SIG_IGN）しません。

指定した場合，現在設定されている action を変更しないで，trap コマンドは正常終了します。

signal

trap の対象となるシグナルを指定します。

signal にはシグナル番号またはシグナル名称を指定します。シグナル名称には，シグナル名から先頭の「SIG」を除いた名称を指定してください（例：SIGINT であれば「INT」と指定する）。それぞれのシグナルの仕様については，使用している OS のマニュアルを参照してください。

なお，SIGTERM を指定した場合の動作は，環境設定パラメーター TRAP_ACTION_SIGTERM の指定に従います。詳細については，「7. 環境ファイルで設定するパラメーター」の「TRAP_ACTION_SIGTERM パラメーター（ジョブコントローラが強制終了要求を受けたときの動作を定義する）」を参照してください。

signal はスペースで区切って複数のシグナルを指定できます。また，**signal** には 0，"EXIT"または"ERR"を指定できます。

signal に 0 または"EXIT"を指定し，trap コマンドを実行した場合

シェル終了時に**action** に指定したコマンドを実行します。

signal に"ERR"を指定し，trap コマンドを実行した場合

trap コマンド以降に実行した次に示すコマンドが 0 以外の終了コードで完了すると，**action** に指定した動作を実行します。

・正規組み込みコマンド

・typeset コマンド

・関数内，および外部スクリプト内で書式不正によってエラーとなった return コマンド

AIX の場合，**signal** に SIGWAITING は指定できません。SIGWAITING を指定して実行した場合，エラー終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- trap コマンドは 0 より小さい値を引数signal に指定すると不当なシグナルと判断します。そのため，引数signal を数値で指定する場合は，0～シグナルの有効範囲内の値を指定してください。
- trap コマンドによる動作定義にシェル拡張コマンド adshread を指定すると，強制終了要求を受けても入力応答待ちとなりジョブが終了しなくなります。TRAP_ACTION_SIGTERM パラメーターのオペランドに TERM を指定した場合，または AUTO を指定して JP1/AJS からジョブを起動した場合は，trap コマンドによる動作定義にシェル拡張コマンド adshread を指定しないでください。
- trap コマンドによる動作定義にシェル運用コマンド adshjava を指定して強制終了すると，trap アクション内で Java のバッチアプリケーションを実行できますが，trap アクション実行中にさらに強制終了すると，adshjava コマンドは強制終了されません。このため，TRAP_ACTION_SIGTERM パラメーターのオペランドに TERM を指定した場合，または AUTO を指定して JP1/AJS からジョブを起動した場合は trap アクション内で adshjava コマンドは使用しないでください。ユーザー固有の後処理として Java のバッチアプリケーションを動かしたい場合，cjexecjob コマンドおよび cjkilljob コマンドを使用してください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。
- 引数action を省略し，引数signal にシグナル番号だけを指定すると trap コマンドは引数signal に指定されたアクションをリセットしデフォルトに戻ります。しかし，SIG を除いたシグナル名称だけを指定すると，trap コマンドは引数signal に指定されたアクションをリセットしないで終了します。

  例：trap 15

  →シグナル番号 15 のアクションをリセットし，デフォルトに戻ります。

  例：trap TERM

  →アクションをリセットしません。

使用例

- INT シグナルを受け取った場合，echo コマンドで'trapped.'を出力します。

```
trap 'echo trapped.' INT
```

- シグナルに設定されているaction を表示します。

  ジョブ定義スクリプトの内容

```
trap 'echo Hangup.' HUP
trap 'echo trapped.' INT
trap
```

  実行ジョブの STDOUT ファイルの内容

```
********   実行ジョブのSTDOUTファイルの内容     ********
trap -- 'echo Hangup.' HUP
trap -- 'echo trapped.' INT
```

(2) trap コマンド【Windows 版】

形式

```
trap [action] [method]
```

機能

強制終了要求を受けたときの動作を設定します。

TRAP_ACTION_SIGTERM パラメーターに TERM を指定した場合：

強制終了要求をジョブコントローラが受けたときの動作を設定できます。ジョブコントローラは**method**に指定された強制終了要求を受けると，**action** に指定された動作を実行します。

method に TERM でも 15 でもない強制終了要求を指定したとき，指定した**action** が設定されないで，メッセージ KNAX6718-I を出力して，終了コード 0 を返して終了します。

引数を指定しないで実行した場合，強制終了要求に対して設定されている動作を次の形式で標準出力に出力します。

出力形式
trap -- action "強制終了の方法を示す文字列"

TRAP_ACTION_SIGTERM パラメーターに DISABLE を指定した場合：

メッセージ KNAX6710-I を出力して，終了コード 0 を返して常に正常終了します。強制終了要求に対して処理はしません。

引数

action

強制終了要求を受けたときの動作を指定します。

action にハイフンを指定した場合，**method** に一致する指定済みの**action** がリセットされ，**method** に対しての動作定義（**action**）が無効になり，何も設定されていない状態に戻ります。**action** を指定しないで**method** に 15 を指定した場合も同様に，**method** に対しての動作定義（**action**）が無効になり，何も設定されていない状態に戻ります。

action に""を指定した場合，現在設定されている**action** を変更しないで，trap コマンドは正常終了します。

method

trap の対象となる強制終了の方法を指定します。

method には TERM または 15 を指定します。

TERM または 15

TerminateProcess などによるプロセス即時終了（例：JP1/AJS からの強制終了，taskkill コマンド）。

終了コード

終了コード	意味
0	正常終了

終了コード	意味
1	エラー終了

注意事項

- 「kill -KILL プロセス ID」を実行して任意のジョブの adshexec.exe を終了させると，終了させられたジョブの「trap action TERM」で定義された**action** が実行されます。
- TRAP_ACTION_SIGTERM パラメーターに TERM を指定した場合，trap コマンドにオプションを指定するとエラーになります。TRAP_ACTION_SIGTERM に TERM 以外を指定した場合，trap コマンドにオプションを指定してもエラーになりません。
- trap コマンドによる動作定義にシェル拡張コマンド adshread を指定すると，強制終了要求を受けても入力応答待ちとなり，ジョブが終了しなくなります。TRAP_ACTION_SIGTERM パラメーターのオペランドに TERM を指定した場合，trap コマンドによる動作定義にシェル拡張コマンド adshread を指定しないでください。
- trap コマンドによる動作定義にシェル運用コマンド adshjava を指定して強制終了すると，trap アクション内で Java のバッチアプリケーションを実行できますが，trap アクション実行中にさらに強制終了すると，adshjava コマンドは強制終了されません。このため，TRAP_ACTION_SIGTERM パラメーターのオペランドに TERM を指定した場合，または AUTO を指定して JP1/AJS からジョブを起動した場合は trap アクション内で adshjava コマンドは使用しないでください。ユーザー固有の後処理として Java のバッチアプリケーションを動かしたい場合，cjexecjob コマンドおよび cjkilljob コマンドを使用してください。
- この特殊組み込みコマンドは，コマンドの構文を誤ってエラー終了すると，コマンドを実行しているシェルが終了します。
- 引数**action** を省略し，引数**method** に 15 を指定すると，trap コマンドは TerminateProcess などによるプロセス即時終了に指定されたアクションをリセットし，何も設定されていない状態に戻ります。しかし，TERM だけを指定すると trap コマンドは TerminateProcess などによるプロセス即時終了に指定されたアクションをリセットしないで終了します。

  例：trap 15

  →TerminateProcess などによるプロセス即時終了に指定されたアクションをリセットし，何も設定されていない状態に戻ります。

  例：trap TERM

  →アクションをリセットしません。
- **method** に 2 つ以上の値が指定された場合，その値に TERM または 15 が一回以上含まれるときは，KNAX6718-I を出力しないで，TERM または 15 に対して**action** を設定します。それ以外の**method** に対しては**action** を設定しません。

  例：「trap date 28 15」を実行した場合，KNAX6718-I を出力しないで，15 に対する**action** として date コマンドを設定します。
- 強制終了要求を受けたときに，バックグラウンドで実行されているプロセスは，「trap action TERM」で定義された action の前に終了します。

使用例

- 強制終了要求を受けた場合，echo コマンドで'trapped.'を出力します。

```
trap 'echo trapped.' TERM
```

- 強制終了要求に対して設定されている**action** を表示します。**action** を設定する trap コマンドで**method** に 15 を指定しても，強制終了の方法には TERM が出力されます。

  ジョブ定義スクリプトの内容

```
trap 'echo trapped.' 15
trap
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
trap -- 'echo trapped.' TERM
```

9.3.28　true コマンド（終了コード 0 を返す）

形式

```
true
```

機能

終了コード 0 を返して正常に終了します。

なお，このコマンドには指定可能なオプションが存在しません。オプションが指定された場合でも無視して処理を実行します。

終了コード

終了コード	意味
0	正常終了 • 常に 0 を返します。

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- このコマンドの実行結果は KNAX6113-I メッセージに出力されます。

使用例

- 終了コード 0 を設定します。

  ジョブ定義スクリプトの内容

```
true
echo $?
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
0
```

9.3.29 typeset コマンド（変数や関数の属性と値を明示的に宣言する）

形式

```
typeset  [ {-L|+L}  [n] ]  [ {-R|+R}  [n] ]  [ {-Z|+Z}  [n] ]
         [-l|+l|-u|+u]  [ {-i|+i}  [n] ]  [-r|+r|-x|+x]
         [ {-f|+f}  [-t|+t]  [-u] ]
         [-p|+p]
         [--]  [name [=value] ...]
```

機能

変数や関数の属性と値を明示的に宣言します。*name* に指定された変数や関数の属性と値を明示的に宣言し，定義します。

このコマンドを関数内で実行すると，関数内で有効なローカル変数（関数内ローカル変数）として定義します。関数内ローカル変数を定義した場合，関数の完了時に値と属性を回復します。

このコマンドのオプションは文字列書式オプション，属性・型オプション，関数オプション，表示オプションの 4 種類に分類されます。

引数

オプションをハイフンで指定した場合，指定したオプションは有効に設定されます。オプションをプラスで指定した場合，指定したオプションは無効に設定されます。

[{-L|+L} [*n*]] [{-R|+R} [*n*]] [{-Z|+Z} [*n*]]

- -L|+L

  文字列書式オプションです。-L オプションでは，変数の内容に対して左詰めにします。+L オプションでは，-L オプションで指定した左詰めにする属性を解除します。

  変数に値を代入する際に *value* に指定された内容が領域長よりも短い場合は，*value* の最後から領域の終端までスペースを挿入します。*value* に指定された内容が領域長よりも長い場合は，*value* の先頭から領域長までが代入され，残りは切り捨てになります。

同時に-Z オプションを指定した場合，先行する 0 の削除もします。同時に-R オプションを指定した場合，あとに指定した方を設定します。

-R オプションで *name* を定義済みの場合は，右詰めの設定は無効になります。

- -R|+R

  文字列書式オプションです。-R オプションでは，変数の内容に対して右詰めにします。+R オプションでは，-R オプションで指定した右詰めにする属性を解除します。

  変数に値を代入する際に*value* に指定された内容が領域長よりも短い場合は，領域の先頭から *value* の先頭までスペースを挿入します。*value* に指定された内容が領域長よりも長い場合は，*value* の最後から領域長までが代入され，残りは切り捨てになります。同時に-L オプションを指定した場合，あとに指定した方を設定します。

  -L オプションで *name* を定義済みの場合は，左詰めの設定は無効になります。

- -Z|+Z

  文字列書式オプションです。-Z オプションでは，変数の内容に対してゼロ詰めにします。+Z オプションでは，-Z オプションで指定したゼロ詰めにする属性を解除します。

  -L オプションが設定されていない場合は右詰めになります。*value* に指定された内容の先頭文字が数字の場合は，領域の先頭から *value* の先頭までゼロ詰めになり，数字以外の文字の場合は，領域の先頭から *value* の先頭までスペースを挿入します。

- *n*

  n には *value* の領域長を指定します。*n* が0，または *n* を省略した場合は，*value* の長さを領域長にします。*n* に16385 以上を指定した場合は，エラーになります。

[-l|+l|-u|+u]

- -l|+l

  文字列書式オプションです。-l オプションでは，*name* に指定された変数に代入されている英字の大文字を小文字に変換します。変数に代入されている文字列に大文字と小文字が混在している場合，大文字だけを小文字に変換します。同時に-u オプションを指定した場合，あとに指定した方を設定します。

  +l オプションでは，-l オプションで指定した変数に代入されている，英字の大文字を小文字に変換する属性を解除します。

- -u|+u

  文字列書式オプションです。-u オプションでは，*name* に指定された変数に代入されている英字の小文字を大文字に変換します。変数に代入されている文字列に大文字と小文字が混在している場合，小文字だけを大文字に変換します。同時に-l オプションを指定した場合，あとに指定した方を設定します。

  +u オプションでは，-u オプションで指定した変数に代入されている，英字の小文字を大文字に変換する属性を解除します。

[{-i|+i} [*n*]] [-r|+r|-x|+x]

- -i|+i

属性・型オプションです。-i オプションでは，*name* に指定された変数の型を整数型として宣言します。*value* には代入する値を 10 進数で指定します。-i で 10 進数以外の基数を指定した場合，*name* に指定された変数の内容の先頭に'*基数#*'が付加されます。-Z オプションでゼロ詰めをしている場合は，'*基数#*'の先頭までをゼロ詰めにします。

+i オプションでは，*name* に指定された変数の整数型属性を解除します。

- *n*

  n には出力時に何進数で表示するかを指定します。*n* を省略または0 を指定し，かつ*name* が未定義の変数の場合，10 進数として扱われます。*n* を省略または0 を指定し，かつ*name* が定義済みの変数の場合，定義されている基数に従います。*n* に1 や17 以上を指定した場合は，エラーになります。

- -r|+r

  属性・型オプションです。-r オプションでは，*name* に指定された変数の属性を読み込み専用にします。属性を読み込み専用にすると，それ以降，変数の値および属性を変更できません。

  +r オプションでは，*name* に指定された変数の読み込み専用属性を解除します。

- -x|+x

  属性・型オプションです。-x オプションでは，*name* に指定された変数をエクスポートします。+x オプションでは，*name* に指定された変数のエクスポートを解除します。

  Windows で変数をエクスポートする場合，指定できる変数名は次のように異なります。

VAR_ENV_NAME_LOWERCASE パラメーターにDISABLE を指定した場合

シェル変数名に小文字が含まれているとエクスポートできないため，変数名に含まれる英字はすべて大文字にする必要があります。

英小文字を含む変数名を*name* に指定した場合，エラーメッセージを出力し，バッチジョブを終了します。

VAR_ENV_NAME_LOWERCASE パラメーターにENABLE を指定した場合

シェル変数名に小文字が含まれていてもエクスポートできます。

ただし，環境変数は大文字・小文字の区別はなく，最後にエクスポートした同じスペルのシェル変数が最終的な環境変数値となります。

{-f|+f} [-t|+t] [-u]

- -f|+f

  関数オプションです。-f オプションでは，*name* に指定された処理対象を変数ではなく関数として扱います。-f オプションを指定し実行した場合，*name* に指定された関数を標準出力に出力します。

  +f オプションを指定し実行した場合，関数を出力しません。-f オプションだけを指定し実行した場合，定義されているすべての関数を標準出力に出力します。

- -t|+t

  関数オプションです。-t オプションでは，*name* に指定された関数のトレースモードを有効にします。このオプションは，-f オプションと同時指定された場合に有効になります。

  +t オプションでは，*name* に指定された関数のトレースモードを無効にします。

- -u

関数オプションです。-u オプションでは，*name* に指定された関数に対してオートロード機能を有効にします。このオプションは，-f オプションと同時指定された場合に有効になります。

-p|+p

表示オプションです。-p オプションでは，定義されているすべての変数を「typeset *変数名=値*」の書式で標準出力に出力します。ただし，-p オプションと *name* を同時に指定した場合，*name* に指定された変数の属性の宣言が優先されます。

+p オプションでは，定義されているすべての変数を「typeset *変数名*」の書式で標準出力に出力します。

オプション指定なし

表示オプションです。すべてのオプションを指定しないで実行した場合，定義されているすべての変数を「typeset 宣言されている属性・型オプションの値 変数名」の書式で標準出力に出力します。ただし，属性・型オプションが宣言されていない場合は変数名を左に詰めて出力します。

name を指定しないで，オプションだけ指定

表示オプションです。指定したオプションの属性と等しい変数，および関数をすべて出力します。ハイフンで指定した場合，「*変数名=値*」または関数の内容が標準出力に出力されます。プラスで指定した場合，「*変数名*」または「*関数名*」が標準出力に出力されます。

--

オプション終端文字です。このオプション以降に指定したオプションは，val（変数）として解釈します。

name

属性や値を宣言する変数名，配列名，または関数名を複数指定します。

配列名を指定した場合，配列を構成する全要素が対象になります。配列の 1 つの要素を指定した場合も配列の全要素が対象になります。

name の後ろに=を指定すると，*name* への値の代入と属性の宣言を同時にできます。

name に指定された変数の属性が読み込み専用で，値を代入しようとした場合，エラー終了します。

value

name に代入する値を指定します。*value* を指定しなかった場合，*name* には改行文字が代入され，属性を変更します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- 引数 *n* には，同時に指定する引数ごとに設けられた範囲内の値を指定してください。

- マルチバイト文字を含む変数に対して，左詰めまたは右詰めで切り捨てられると，マルチバイト文字を構成する一部のデータが消失することがあります。この場合，文字として不完全なバイト列が代入されることがあります。
- 引数が指定できるオプションとその引数を 1 つのハイフンで同時に指定できません。8 進数の int 型で，かつ右詰め 16 桁として定義する場合は，「-i8 -R16」と指定します。「-i8R16」のように指定した場合は，エラー終了します。
- -f オプションと同時に-x オプションを指定した場合，-x オプションの指定は無視されます。typeset コマンドは-f オプションが指定された場合と同じ動作をします。
- -i オプションを指定して属性変更をする指定の場合，配列の一括定義（*配列名*=(…)または*配列名*[]=({…}…)の指定で定義する指定）はできません。配列の属性を一括変更する場合は，配列の一括定義後に-i オプションを指定したtypeset コマンドで定義済みの配列名を指定してください。

使用例

- 変数num の属性を左に詰めて，10 桁の整数型に変更します。

```
typeset -L10 -i num
```

- 関数func のトレースモードを有効にします。

```
typeset -ft func
```

9.3.30 ulimit コマンド（システムリソースの上限を設定する）【UNIX 限定】

形式

```
ulimit  [-H]  [-S]  [-a]  [-c]  [-d]  [-f]  [-l]  [-m]
        [-n]  [-p]  [-s]  [-t]  [limit]
```

機能

システムリソースの上限の設定および情報を標準出力に出力します。指定されたオプションに従って，システムリソースの上限値を設定し，出力します。

出力するリソース上限の出力形式を次の表に示します。

出力形式	内容
time(cpu-seconds) *上限値*	CPU 時間の上限
file(blocks) *上限値*	ファイルサイズの上限
coredump(blocks) *上限値*	コアダンプのファイルサイズの上限
data(kbytes) *上限値*	データ領域サイズの上限

出力形式	内容
stack(kbytes) *上限値*	スタック領域サイズの上限
lockedmem(kbytes) *上限値*	ロックされる物理メモリのメモリサイズの上限
memory(kbytes) *上限値*	使用される物理メモリのメモリサイズの上限
nofiles(descriptors) *上限値*	ファイルディスクリプタ数の上限
processes *上限値*	プロセス数の上限

引数

リソースを示すオプションを複数同時に指定した場合は，あとに指定したオプションが有効になります。

-H

ハードリミットを設定または出力します。-H とオプションを同時に指定した場合は，あとに指定したオプションが有効になります。

-S

ソフトリミットを設定または出力します。-H と-S オプションを同時に指定した場合は，あとに指定したオプションが有効になります。

-a

すべてのリソースの上限値を出力します。

-c

コアダンプのファイルサイズ上限を block 単位で設定または出力します。

-d

データ領域サイズの上限を KB 単位で設定または出力します。

-f

シェルまたはシェルから起動したプロセスが書き込むファイルの，ファイルサイズの上限を block 単位で設定または出力します。

-l【Linux 限定】

ロックされる物理メモリのメモリサイズの上限を KB 単位で設定または出力します。

-m【AIX，HP-UX，Linux 限定】

使用される物理メモリのメモリサイズの上限を KB 単位で設定または出力します。

-n

オープンされたファイルディスクリプタ数の上限を設定または出力します。

-p【Linux 限定】

ユーザー 1 人が起動できるプロセス数の上限を設定または出力します。

-s

スタック領域サイズの上限を KB 単位で設定または出力します。

-t

CPU 時間の上限を秒単位で設定または出力します。

limit

変更するリソースの上限値を指定します。unlimited を指定すると，上限なしで設定します。上限値には任意の数値を指定できますが，実際に有効となる上限値の仕様については，使用している OS のマニュアルを参照してください。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- このコマンドは次のどちらかに該当する場合，サポートしていないことを示すメッセージKNAX6710-Iを出力し，終了コード0 を返して常に正常終了します。
  - OS がサポートしていないオプションを指定した。
  - Windows 環境で実行した。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- adshexec コマンドはジョブ定義スクリプトの実行時に実行に必要なファイルを生成するため，ulimit コマンドで指定するファイルサイズの上限値が小さいと，SIGXFSZ シグナルを受信する場合があります。
- ハードリミットを増加させる場合は，管理者権限が必要です。
- 設定できるリソースの上限は実行環境や OS ごとに異なります。
- 変更するリソースによっては，設定できない値を指定すると，実行環境や OS によって異なる値が設定される場合があります。
- 引数limit には0～2,147,483,647 の範囲内の値を指定してください。引数limit に範囲外の値を指定した場合，期待した結果が得られない場合があります。

使用例

- すべてのリソースの上限値を出力します。

  ジョブ定義スクリプトの内容

```
ulimit -a
```

  実行ジョブのSTDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容     ********
time(cpu-seconds)    unlimited
file(blocks)         unlimited
```

```
coredump(blocks)     0
data(kbytes)         unlimited
stack(kbytes)        10240
lockedmem(kbytes)    32
memory(kbytes)       unlimited
nofiles(descriptors) 1024
processes            4096
```

9.3.31 umask コマンド（新規ファイル作成時のアクセス権を設定する）【UNIX 限定】

形式

```
umask [-S] [mask]
```

機能

新規ファイル作成時のアクセス権を設定します。**mask** には設定するファイルモード作成マスクを指定します。**mask** を指定しないで実行した場合，現在の umask 値を標準出力に出力します。

引数

-S

シンボリック形式で値を設定または出力します。

-S オプションを指定した場合，シンボリック形式でファイルモードの設定および出力をします。-S オプションを指定しなかった場合，8 進数で表したアクセス権の設定および出力をします。指定したアクセス権はファイル作成時に許可しないことを示します。

mask

ファイル作成時のファイルモードのデフォルト値に対する umask 値を指定します。**mask** には数値またはシンボリック形式を指定できます。シンボリック形式で指定する場合，[who][op][perm][, ...]の書式に従って指定します。コンマで区切ると複数指定できます。スペースは使用できません。

- who

  mask を設定する対象です。次の文字を 0 個以上指定します。

  u：ユーザー（所有者）用のパーミッションを修正します。

  g：グループ用のパーミッションを修正します。

  o：そのほかのパーミッションを修正します。

  a：全対象のパーミッションを修正します(a=ugo)。

  指定なし：全対象のパーミッションを修正します(a=ugo)。

- op

  mask の設定方法です。次の記号を 1 つ指定します。

＋：perm を who の既存のマスクに追加します。

-：perm を who の既存のマスクから削除します。

＝：who の既存のマスクを perm で置換します。

- perm

  ファイル作成時に許可する権限を指定します。次の文字を 0 個以上指定します。

  r：read 権限です。

  w：write 権限です。

  x：実行権限です。

  u：ユーザー用と同じ権限です。

  g：グループ用と同じ権限です。

  o：そのほかと同じ権限です。

  X：ugo のどれかに実行権限がある場合は，その実行権限になります。ugo のどれにも実行権限がない場合は，権限指定がありません。権限指定がない場合，op が＋または-のときは変更しません。op が＝のときは，who のマスクが解除されます。

  s：権限指定がありません。op が＋または-の場合は，変更しません。op が＝の場合は，who のマスクが解除されます。

  指定なし：権限指定がありません。op が＋または-の場合は，変更しません。op が＝の場合は，who のマスクが解除されます。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- Windows 環境でこのコマンドを実行すると，サポートしていないことを示すメッセージ KNAX6710-I を出力し，終了コード 0 を返して常に正常終了します。ファイルモード作成マスクは設定しません。
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。
- #-adsh_file_temp コマンドおよび adshmktemp コマンドを使用して作成した一時ファイルのパーミッションは，ファイルの所有者（作成者）の部分だけ umask 値の指定に従い，グループおよびその他のユーザーのアクセス権限部分は常に 0 になります。

使用例

- ファイル作成時にユーザー以外の全アクセス権限を設定しません。

```
umask 077
```

- ファイル作成時にユーザー以外の write 権限を設定しません。

```
umask u=rwx,go=rx
```

9.3.32　unalias コマンド（エイリアス定義を無効にする）

形式

```
unalias [-a] name [name...]
```

機能

エイリアス定義を無効します。name には定義を無効にするエイリアスの名称を指定します。スペースで区切ると複数のエイリアスを指定できます。定義されていないエイリアス名称をname に指定した場合，またはオプションや引数を指定しないで実行した場合は，終了コード 1 でエラー終了します。

引数

-a

　すべてのエイリアス定義を無効にします。

name

　定義を無効にするエイリアスを指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了。または，name に指定した名称のどれかがエイリアスとして定義されていません。

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- 定義されているエイリアス（functions）を無効にします。

```
unalias functions
```

9.3.33　unset コマンド（変数の値と属性の設定を解除する）

形式

```
unset [-f] name [ name ...]
```

機能

変数および関数の設定を解除します。name に指定された変数の設定を解除します。name は複数指定できます。-f オプションを指定し実行した場合，name を関数名として扱い，関数の定義を解除します。

引数

-f

name を関数名として扱い，関数の定義を解除します。

name

対象となる変数名または関数名を指定します。name に配列名も指定できます。

name に配列名を指定した場合，配列を構成する全要素の設定を解除します。1 つの要素の設定だけを解除する場合は，「配列名［要素番号］」または「配列名［要素番号］［要素番号］」をname に指定します。

配列要素番号には配列要素番号を示す数値のほか，@および*を指定できます。指定の組み合わせによる unset 対象範囲については，「(3)　配列の値の参照」を参照してください。

name に指定した変数の属性が読み取り専用の場合，エラー終了します。未定義の変数名および関数名をname に指定し実行すると，エラー終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了。または name に指定した名称のどれかが変数，もしくは関数として定義されていません。

注意事項

- このコマンドで LINENO，OPTARG，OPTIND，RANDOM，SECONDS などのシェル変数の設定を解除すると，再びこれらのシェル変数を定義しても，シェル変数が持つ特殊な意味は失われます。
- このコマンドで配列の要素の設定を個別に解除した場合，デバッガでは定義済みの変数として扱います。そのため，unset コマンドで配列の要素の設定を解除したあとも，デバッガコマンドで表示・設定の対象にできます。

  配列を構成する全要素の設定を解除した場合は，デバッガでは未定義の変数として扱うため，デバッガコマンドによる表示・設定の対象にできません。デバッガコマンドによる変数の値の表示・設定については，「6.　ジョブ定義スクリプトのデバッグ」を参照してください。
- この特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了します。

使用例

- unset コマンドで変数を解除します。

```
unset val
```

- unset コマンドで関数を解除します。

```
unset -f func
```

9.3.34　wait コマンド（子プロセスの完了を待つ）

形式

```
wait  [pid ...]
```

機能

子プロセスの完了を待ちます。**pid** には完了を待つ子プロセスのプロセス ID を 1 つ以上指定します。**pid** を指定しなかった場合は，実行中のすべての子プロセスの完了を待ちます。数値以外から始まる不当なプロセス ID を**pid** に指定した場合は，終了コード 127 で正常終了します。

ただし，次のように数値と数値以外を混在して指定し，かつ先頭が数値の場合，数値として解釈される位置までをプロセス ID と判断し，プロセスの完了を待ちます。

```
UAP &        # シェル変数!にはUAPを起動したプロセスが格納されます。
wait $!ABC   # waitコマンドはABCの直前までをプロセスIDと解釈し，完了を待ちます。
```

引数**pid** を指定した場合，wait コマンドは最後の完了を待ったプロセスのコマンドの終了コードで終了します。例を次に示します。

```
UAP1 &                    # UAP1は終了コード2
PID1=$!                   #
UAP2 &                    # UAP2は終了コード16
PID2=$!                   #
UAP3 &                    # UAP3は終了コード0
PID3=$!                   #
wait $PID1 $PID2 $PID3    # waitコマンドは終了コード0で終了します
```

引数

pid

完了を待つ子プロセスのプロセス ID を指定します。

終了コード

終了コード	意味
0	正常終了
127	正常終了 • pid に指定された子プロセスを特定できません。 • 実行中の子プロセス以外のプロセス ID を pid に指定しました。
上記以外	エラー終了

注意事項

- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- プロセス ID が 4848 の子プロセスの完了を待ちます。

```
wait 4848
```

9.3.35　whence コマンド（文字列をコマンドとした場合の解釈を表示する）

形式

```
whence [-p] [-v] name [name...]
```

機能

指定された文字列をコマンドとした場合の解釈を標準出力に出力します。オプションを指定しない場合は次の内容を出力します。

- name に指定された文字列がコマンドのときは，コマンドのパス名を出力します。
- name に指定された文字列がエイリアスのときは，エイリアスの値を出力します。
- name に指定された文字列が予約語，シェル標準コマンド，シェル拡張コマンドまたは関数のときは，name を出力します。
- 上記のどれにも該当しない場合は，何も出力しないで終了コード 1 で終了します。

-p オプションと-v オプションを同時に指定した場合は，name に指定された文字列をコマンドと解釈して出力します。引数name を指定しなかった場合は，終了コード 1 でエラー終了します。

引数

-p

name に指定された文字列をコマンドとした場合のコマンドパスを出力します。

-v

name に指定された文字列がコマンド，予約語，エイリアス，シェル標準コマンド，シェル拡張コマンドまたは関数かどうかを出力します。

出力内容を次の表に示します。

項番	出力内容	意味
1	name is a reserved word	name は予約語です。
2	name is a function	name は関数です。
3	name is a traced function	name はトレースモードが有効な関数です。 関数が未定義で，かつ関数のトレースモードが有効な場合は，項番 4 の内容が出力されます。
4	name is an undefined function	name は未定義の関数です。
5	name is an extended shell command	name はシェル拡張コマンドです。
6	name is a shell builtin	name は組み込みコマンドです。
7	name is a special shell builtin	name は特殊組み込みコマンドです。
8	name is a shell builtin not supported	name は JP1/Advanced Shell で提供しないコマンドです。
9	name is パス名	name はコマンドまたは実行できるファイルです。
10	name is an alias for 'エイリアスの値'	name はエイリアスです。
11	name is an exported alias for 'エイリアスの値'	name はエクスポートされたエイリアスです。
12	name not found	name はコマンド，予約語，エイリアス，シェル標準コマンド，シェル拡張コマンドまたは関数のどれも該当しません。

name

コマンドとして扱う文字列を指定します。引数 name を指定しなかった場合は，終了コード 1 でエラー終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了。または，name に指定したコマンドのどれかが見つかりません。

注意事項

- 引数 name にシンボリックリンクを指定した場合，実行可能かどうかの判定は，シンボリックリンクとリンク先の両方の実行権限を判定します。【Windows 版】
- この正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しません。

使用例

- pwd をコマンドとした場合のコマンドパスを出力します。

  ジョブ定義スクリプトの内容

```
whence -p pwd
```

  実行ジョブの STDOUT ファイルの内容

```
********    実行ジョブのSTDOUTファイルの内容      ********
/bin/pwd
```

9.4　シェル拡張コマンド

シェル拡張コマンドは，JP1/Advanced Shell 独自の組み込みコマンドです。組み込みコマンドはシェル本体に組み込まれたコマンドであり，シェル自身によって実行します。

シェル拡張コマンドは，コマンドの構文を誤っても，コマンドを実行しているシェルは終了しません。

9.4.1　adshappexec コマンド（GUI アプリケーション実行コマンド）【Windows 実行環境限定】

シェル運用コマンドの機能と同じです。このコマンドの機能については，「8.3.2　adshappexec コマンド（GUI アプリケーション実行コマンド）【Windows 実行環境】」を参照してください。

9.4.2　adshappexec コマンド（GUI アプリケーション実行コマンド）【Windows 開発環境限定】

形式

```
adshappexec [-m] [-d ワークフォルダ] [-v 表示名] {-w 実行アプリケーション名 | -n 実行アプリケーション名} [-- 引数1 引数2...]
```

機能

開発環境でのデバッグのために，アプリケーション実行エージェントを経由しないで実行アプリケーションを起動します。

実行アプリケーションが返す戻り値は標準出力に出力します。ジョブの戻り値として後続ジョブで使用する場合はコマンド置換で変数に格納します。

引数

-m

標準エラー出力へのメッセージの出力を抑止します。標準入出力を使用できない環境で使用します。

コマンドの引数指定エラー，およびライセンスチェックエラーは，-m オプションを指定しても出力されます。

-d ワークフォルダ ～<パス名>((1～247 バイト))

実行アプリケーション実行時のワークフォルダを指定します。

ワークフォルダを指定しなかった場合は，adshappexec コマンド実行時のカレントパスで動作します。

ワークフォルダはスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

-v 表示名 ～<パス名>((1～247 バイト))

アプリケーション実行エージェントアイコンを左クリックした時に表示される表示名を指定します。

表示名はスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

表示名を省略した場合には，実行アプリケーション名を出力します。

Windows 開発環境の場合，メッセージ出力にだけ使用します。

複数の実行アプリケーションを動作させたときにアプリケーションを区別するためにこの引数を指定することを推奨します。

-w 実行アプリケーション名 ～<パス名>((1～247 バイト))

実行アプリケーションの終了まで実行を終了しません。

実行アプリケーション名は，実行アプリケーションのファイル名を指定します。

実行アプリケーション名はスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

-n 実行アプリケーション名 ～<パス名>((1～247 バイト))

実行アプリケーションの終了を待たずに終了します。

実行アプリケーション名は，実行アプリケーションのファイル名を指定します。

実行アプリケーション名はスペースを含む場合，ジョブ定義スクリプトからの実行であれば「"」で囲むなどして，スペースを含めて指定してください。

-- 引数 1 引数 2... ～<引数>((1～1,023 バイト))

「--」のあとに実行アプリケーションの実行時に指定するパラメーターを指定します。

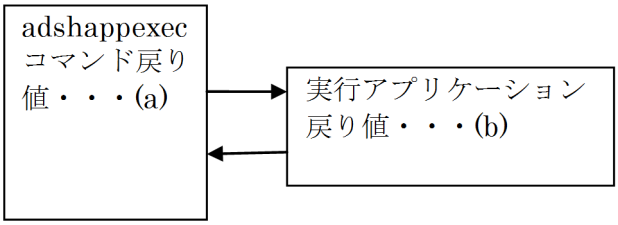
関連付けを行った実行アプリケーションを指定した場合には引数を指定しないでください。

引数は，合計の引数長が 1,023 バイト以内であれば，いくつでも指定できます。

終了コード

終了コード	意味
0	正常終了
0 以外	異常終了

adshappexec コマンドでは，次の 2 つのプロセスでの終了コードがあります。プロセスごとの終了コードの扱いは次のようになります。



エラー発生個所での戻り値の出力は次のとおりです

(a) adshappexec コマンド処理でのエラー

コマンドの戻り値として出力します。

また，標準エラー出力にメッセージを出力します。

(b) 実行アプリケーションの戻り値

標準出力に出力します。

また，標準エラー出力には，実行アプリケーションの戻り値を出力します。

実行アプリケーションの戻り値で 0 以外の戻り値を返しても，adshappexec コマンドは異常終了しません。

実行アプリケーションの戻り値を確認するにはメッセージを確認するか，標準出力の内容を確認してください。

注意事項

- adshappexec コマンドの-w 引数と-n 引数は必ず指定してください。

  次の場合，最後に指定したものが有効になります。

  ・-w オプションと-r オプションを同時に指定した場合

  ・-w オプションを複数指定した場合

  ・-r オプションを複数指定した場合

- 実行アプリケーションは実行ユーザーの環境変数を使用します。
- 同じオプションを複数指定した場合，最後の指定が有効になります。
- 次の仕様のアプリケーションを-w 引数で実行した場合，指定したファイルを閉じても adshappexec コマンドが終了しないことがあります。

  - 実行アプリケーションに指定したファイルを閉じても，アプリケーション自体が終了しない

  このような場合は，アプリケーション自体が終了すれば，adshappexec コマンドも終了します。

  例：実行アプリケーションとして指定した Excel ファイルだけ閉じて，Excel 自体は終了させていない場合

- 実行アプリケーションのプロセス起動に関する仕様によっては，-w 引数を指定しても，アプリケーションの終了を待たずに adshappexec コマンドが終了することがあります。

  次に示す場合があります。

(1)KNAX7259-W が出力される場合

例：Excel ファイルを実行アプリケーションとして実行する前から，すでに Excel 自体が動作していた場合

この場合は，Excel が起動していない状態で運用するか，-w 引数を使用しない運用としてください。

(2)KNAX7259-W が出力されない場合

例：explorer.exe を実行アプリケーションとして実行した場合

この場合は，-w 引数を使用しない運用としてください。

- 強制終了時は次のことに注意してください。
  - 実行アプリケーション動作中にエディタを使用してデバッグの中止，もしくは「×」ボタンでエディタを終了することはできません。

    実行アプリケーションの終了待ちとなります。
  - TRAP_ACTION_SIGTERM パラメーターで TERM を指定した場合，trap コマンドによる動作定義に adshappexec コマンドを指定しないでください。

    また，対話操作をする実行アプリケーションを指定した場合，応答待ちになり，ジョブ定義スクリプトが終了しない可能性があります。

9.4.3 adshcmdrc コマンド（コマンドの終了コードしきい値を定義する）

形式

```
adshcmdrc コマンド名 しきい値
```

機能

ジョブ定義スクリプトから実行されるコマンドの終了コードが 0 でない場合でも正常終了と見なしたい場合，対象となるコマンド名と，しきい値となる値を定義します。これによって，コマンドの終了コードがしきい値以下の場合が正常終了となります。引数に指定したコマンドがシグナルを受信して終了した場合は，この指定に関係なく，引数に指定したコマンドはエラー終了します。このコマンドは，4095 個まで指定できます。

このコマンドの定義の有効範囲を次に示します。

- コマンドを指定した個所以降のジョブ定義スクリプト実行で有効。
- 同じコマンドに対しての定義が 2 つ以上ある場合，対象のコマンドの個所に近い定義が有効。
- ジョブステップ外に指定した場合，ジョブ定義スクリプト全体に有効。
- ジョブステップ内に指定した場合，指定した個所以降からジョブステップ終了まで有効。

引数

コマンド名　～<コマンド名>((1～255 バイト))

正常終了と見なす終了コードを定義するコマンドの名称を指定します。Windows の場合は拡張子付きの指定もできます。コマンドのパスは指定できません。指定できるコマンドの種類を次に示します。これら以外のコマンドも，別プロセス実行（パイプ，コマンド置換，|&，&を使用）した場合は対象になります。

- 外部コマンド
- UNIX 互換コマンド
- シェル運用コマンド
- コマンドとして実行したスクリプト（#!によって実行）
- 子孫ジョブ
- 関数（CMDRC_CMDGRP_CHECK パラメータのオペランドに FUNCTION を指定したときだけ）

Windows でコマンド名の拡張子を省略すると，指定した名称と同じ名称のコマンドやバッチファイルが，拡張子に関係なくしきい値の管理対象になります。Windows でスペースを含むコマンド名を指定する場合は，"（ダブルクォーテーション）で囲んでください。

しきい値　～<整数>((-1～255))

終了コードで正常終了と見なすしきい値を定義します。ここで指定したしきい値より終了コードが大きい場合，エラー終了と見なします。

-1 を指定した場合，実行結果は常にエラー終了します。

255 を指定した場合，実行結果は常に正常終了します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- このコマンドは，変数置換やエイリアスの解決後のコマンド名に対して適用されます。
- 子孫ジョブとして実行するジョブ定義スクリプトに対して，正常終了と見なす終了コードしきい値を定義したい場合，子孫ジョブの定義には CHILDJOB_EXT パラメータまたは CHILDJOB_SHEBANG パラメータを使用してください。CHILDJOB_PGM パラメータで子孫ジョブを定義した場合，終了コードのしきい値の対象と見なされません。
- CMDRC_CMDGRP_CHECK パラメータに FUNCTION を指定した場合，引数のコマンド名に関数名を指定することで，関数に対して正常終了とみなす終了コードしきい値を定義できます。CMDRC_CMDGRP_CHECK パラメータを指定しない場合，または CMDRC_CMDGRP_CHECK パ

ラメータに NONE を指定した場合，引数のコマンド名に関数名を指定しても，同名のコマンド名が指定されたとして処理されます。

- CMDRC_CMDGRP_CHECK パラメータに FUNCTION を指定した場合，関数内に指定した adshcmdrc コマンドの定義は有効になりません。CMDRC_CMDGRP_CHECK パラメータを指定しない場合，または CMDRC_CMDGRP_CHECK パラメータに NONE を指定した場合，関数内に指定した adshcmdrc コマンドの定義は関数呼び出し元のスクリプトのコマンドに有効になります。
- adshcmdrc コマンドを別プロセス実行した場合，呼び出し元のスクリプトのコマンドには正常終了と見なす終了コードしきい値の定義は有効になりません。adshcmdrc コマンドは別プロセス実行しないようにしてください。
- adshcmdrc コマンドを子孫ジョブ内に定義して実行した場合，子孫ジョブ呼び出し元のスクリプトのコマンドには正常終了と見なす終了コードしきい値の定義は有効になりません。子孫ジョブ内のコマンドにだけ，正常終了と見なす終了コードしきい値の定義が有効になります。
- 指定個数の上限は，adshexec コマンドの引数に指定したジョブ定義スクリプトファイルに定義されたスクリプト，そのスクリプトから呼び出される.（ドット）コマンド，および#-adsh_script コマンドによる外部スクリプトで指定された個数の合計です。外部スクリプトには，ネストで呼び出しているスクリプトも含みます。

使用例

UAP が正常終了したと見なす終了コードのしきい値を定義します。次の例では，ジョブステップ STEP1 内で UAP が終了コード 1 以下で終了した場合，UAP は正常終了したと見なします。

```
#-adsh_step_start STEP1
  adshcmdrc UAP 1
  UAP data
#-adsh_step_end
```

9.4.4 adshecho コマンド（指定した事象通知メッセージを JP1 イベントとして発行する）

形式

```
adshecho [-d] 事象通知メッセージ
```

機能

指定された事象通知メッセージを JP1 イベントとして発行します。発行された JP1 イベントは，JP1/IM - View に表示されます。ただし，ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行した場合，事象通知メッセージを標準出力に表示します。

JP1 イベントは，前回の JP1 イベントの発行時刻から所定の時間（USERREPLY_JP1EVENT_INTERVAL パラメーターで指定）が経過するのを待って発行されます。USERREPLY_JP1EVENT_INTERVAL パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「USERREPLY_JP1EVENT_INTERVAL パラメーター（JP1 イベントの最小発行間隔を指定する）」を参照してください。

引数

-d

デバッグ実行時に事象通知メッセージの出力先を標準出力にします。デバッグ実行以外の場合，指定は無視されます。

「-」で始まる文字列は，すべて有効なオプション文字列が指定されている場合はオプションの指定として扱われます。不当なオプション文字列が指定された「-」で始まる文字列，または「-」で始まらない文字列が指定された場合は，その位置から事象通知メッセージとして扱われます。

事象通知メッセージ　～<任意文字列>((0～1,023 バイト))

JP1 イベントとして発行する事象通知メッセージを指定します。

事象通知メッセージの文字コードは，同一ホスト内で稼働する JP1/Base と合わせてください。文字コードが異なると文字化けします。

指定された事象通知メッセージは，コマンド「echo -E **事象通知メッセージ**」の実行時と同様の変換内容が JP1 イベントとして発行されます。複数の事象通知メッセージが指定された場合はエラー（KNAX7403-E）となります。

終了コード

終了コード	意味	対処	リトライの可否
0	正常終了	なし。	－
1	続行不可能なエラーが発生 • メモリ不足 • 内部矛盾を検出	システム管理者に連絡してください。	×
4	JP1 イベントの処理中にエラーが発生	メッセージに出力されたエラー情報を基に，「ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照して対処してください。	○
5	JP1 イベントの処理中にエラーが発生	メッセージに出力されたエラー情報を基に，「ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照して対処してください。	×
6	指定されたホストへの JP1 イベントの転送に失敗	次の点を確認してください。 • JP1/IM - Manager がインストールされているホストに，JP1/Base がインストールされているか • JP1/IM - Manager がインストールされているホストの JP1/Base のイベントサービスが起動しているか	○

終了コード	意味	対処	リトライの可否
6	指定されたホストへの JP1 イベントの転送に失敗	• JP1/Advanced Shell がインストールされているホストと，JP1/IM - Manager がインストールされているホストの間の，JP1/Base のコネクションが確立されているか	○
7	JP1/Base のライブラリが見つからない	JP1/Advanced Shell がインストールされているホストに，JP1/Base がインストールされているか確認してください。 JP1/Base がインストールされていてこの現象が発生した場合は，JP1/Base を再インストールしてください。	×
8	自ホストの JP1/Base のイベントサービスの接続に失敗	JP1/Advanced Shell がインストールされているホストで，JP1/Base のイベントサービスが起動しているか確認してください。	○
10	指定された形式が不正	コマンドの形式を確認してください。	×
128+シグナル番号【UNIX 限定】	adshecho コマンドがシグナルを受信して終了	ジョブがシグナルを受信して終了していることを確認してください。	×
200【Windows 限定】	adshecho コマンドが強制終了	ジョブが強制終了されていることを確認してください。	×

（凡例）

○：リトライ可

×：リトライ不可

－：対象外

注意事項

- このコマンドは別プロセスで実行しないでください。別プロセスで実行した場合，流量制御（USERREPLY_JP1EVENT_INTERVAL パラメーターで指定）が機能しません。
- このコマンドは，ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行する場合を除いて，JP1/Base や JP1/IM が存在しない環境では実行しないでください。

  実行すると次の問題が発生します。

  - JP1/Advanced Shell が稼働するホストに JP1/Base がインストールされていない場合，コマンドはエラー終了します。
  - JP1/Advanced Shell が稼働するホストの JP1/Base のイベントサービスが稼働していない場合，コマンドはエラー終了します。
  - HOSTNAME_JP1IM_MANAGER で指定されたホストに JP1/Base がインストールされていない，または JP1/Base のイベントサービスが稼働していない場合，コマンドはエラー終了します。
- HOSTNAME_JP1IM_MANAGER で指定されたホストの JP1/IM - Manager が稼働していない場合でも，HOSTNAME_JP1IM_MANAGER で指定されたホストの JP1/Base のイベントサービスに JP1 イベントが到達した時点で JP1 イベントの送信に成功したとしてコマンドは動作します。

- リトライ可能なエラーでコマンドが終了した場合は，そのコマンドを再実行することで成功する可能性があります。コマンドを再実行したい場合は，「3.8.5　adshecho コマンドまたは adshread コマンドがエラー終了した場合の対処」に記載しているジョブ定義スクリプトの記述例を参考として，そのコマンドを再実行するようにジョブ定義スクリプトを作成してください。
- このコマンドはパイプ記号と組み合わせて実行しないでください。
- このコマンドはリダイレクト記号と組み合わせて実行しないでください。

9.4.5　adshjoberr コマンド（ジョブおよびジョブステップにエラーを通知する）

形式

```
adshjoberr 終了コード
```

機能

指定した場所からジョブおよびジョブステップにエラーを通知できます。指定した場所がジョブステップ内またはジョブステップ外の場合，次のように動作します。

ジョブステップ内に指定した場合

ジョブステップのonError 属性の指定に関わらず，ジョブステップ正常ブロックの後続処理は実行しないで，ジョブステップエラーブロックが定義されていれば実行し，ジョブステップがエラー終了します。

ジョブステップ外に指定した場合

run 属性がabnormal またはalways の後続のジョブステップが定義されていれば実行し，ジョブがエラー終了します。

引数

終了コード　～<*整数*>((0～255))

ジョブまたはジョブステップに通知する終了コードを定義します。

終了コード

終了コード	意味
0～255	エラー終了 adshjoberr コマンドの引数に指定した終了コードをジョブまたはジョブステップに通知します。
1	エラー終了 続行不可能なエラーが発生しました。

終了コード	意味
200	エラー終了 コマンドラインの指定に誤りがあります。

注意事項

- adshjoberr コマンドを別プロセス実行すると，呼び出し元のスクリプトのジョブおよびジョブステップにエラーが通知されない場合があります。adshjoberr コマンドは別プロセス実行しないようにしてください。
- adshjoberr コマンドを子孫ジョブ内に定義して実行した場合，子孫ジョブおよび子孫ジョブ内のジョブステップにだけエラーが通知されます。子孫ジョブ呼び出し元のジョブまたはジョブステップにはエラーは通知されません。
- trap コマンドのアクション内でadshjoberr コマンドを実行した場合，実行したタイミングがジョブステップ外の場合はジョブに，ジョブステップ内の場合はジョブステップにエラーが通知されます。
- CMDRC_CMDGRP_CHECK パラメーターのオペランドにFUNCTION を定義した場合，関数内のコマンドはエラー判定対象外であるため，関数内に指定したadshjoberr コマンドから，呼び出し元のスクリプトのジョブおよびジョブステップにエラーが通知されません。CMDRC_CMDGRP_CHECK パラメーターのオペランドにFUNCTION を定義した場合，adshjoberr コマンドは関数内に定義しないようにしてください。
- adshjoberr コマンドは.env ファイル，初期設定スクリプトファイルに定義できません。

使用例

ファイルが存在しない場合，ジョブステップを終了コード 1 で異常終了させます。

ジョブ定義スクリプト

```
#-adsh_job J01

#-adsh_step_start S01
  cmd1
  if [[ ! -a /tmp/tempfile ]]; then
    echo ”tempfile not found”
    adshjoberr 1
  fi
#-adsh_step_end
```

実行結果

```
********  ジョブコントローラのメッセージ出力  ********
17:02:02 046277 KNAX0091-I J01 ジョブが開始しました。
17:02:02 046277 KNAX7901-I ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。
17:02:02 046277 KNAX7902-I ジョブコントローラは，”端末入力モード”で動作します。
17:02:02 046277 KNAX0092-I J01.S01 ステップが開始しました。
17:02:02 046277 KNAX6116-I コマンド（./cmd1，行番号=4）が正常終了しました。rc=0 E-Time=0.001s C-Time=0.000s
17:02:02 046277 KNAX6112-I コマンド（[[，行番号=5）が正常終了しました。rc=0 E-Time=0.000s C-Time=0.000s
```

```
17:02:02 046277 KNAX6112-I コマンド（echo，行番号=6）が正常終了しました。rc=0 E-
Time=0.000s C-Time=0.000s
17:02:02 046277 KNAX6150-E ジョブステップにエラーが通知されました。rc=1 line=7
17:02:02 046277 KNAX6596-E J01.S01 ジョブステップがエラー終了しました。rc=1 E-
Time=0.003s C-Time=0.000s
17:02:02 046277 KNAX0101-E J01 ジョブを実行中にエラーが発生しました。
17:02:02 046277 KNAX0098-I J01 ジョブが終了しました。rc=1 E-Time=0.004s C-Time=0.000s
```

9.4.6 adshmktemp コマンド（ほかと重ならない名前を持つファイルを作成する）

形式

```
adshmktemp  プリフィックス
```

機能

他と重ならない名前を持つファイルを作成し，そのパス名を標準出力に出力します。

作成されるファイルのパス名を次に示します。

一時ファイルディレクトリパス名 プリフィックス_ジョブ識別子_プロセス ID_時間情報_ファイル通し番号

一時ファイルディレクトリパス名

TEMP_FILE_DIR パラメーターによって定義されたディレクトリのパス名。

プリフィックス

引数に指定したプリフィックス。

ジョブ識別子

10 進数の 6 桁のジョブ識別子。6 桁未満の場合，前方に 0 を付加して 6 桁にする。

プロセス ID

10 進数のプロセス ID。5 桁未満の場合，前方に 0 を付加して 5 桁にする。5 桁以上の場合，そのままの値とする。

時間情報

時間情報を 16 進数で表した数値文字列。

ファイル通し番号

1 つのプロセス内で作成するファイルパス名の通し番号。0001 から 4095 までの 10 進数 4 桁の数値文字列。

1 つのプロセス内で，ファイル通し番号が 4095 を超えるまでファイルを作成できる。

UNIX の場合，作成するファイルのパーミッションは，ファイルの所有者（作成者）の部分だけ umask 値の指定に従い，グループおよびその他のユーザーのアクセス権限部分は常に 0 になります。

Windows の場合，権限の指定はありません。

引数

プリフィックス　～<任意文字列>((1～128 バイト))

ファイル名の先頭に付与する文字列を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- 作成したファイルパス名が OS の許容するパス名の最大バイト数を超える場合，エラー終了します。
- 引数のプリフィックスに次の文字を含む文字列を指定するとエラー終了します。
  UNIX の場合：「/」
  Windows の場合：「/」，「¥」，「:」
- 引数のプリフィックスにパス名として使用できない文字を指定しないでください。指定した場合，ファイルの作成に失敗してエラー終了します。

使用例

```
# UNIX版でTEMP_FILE_DIRパラメーターがデフォルト値の場合
adshmktemp testjob
# 以下のファイルが作成され，標準出力にパス名が出力される。
/var/opt/jp1as/temp/testjob_000150_12278_557539f8_0001

# Windows版でTEMP_FILE_DIRパラメーターに「E:¥temp」を指定した場合
adshmktemp testjob
# 以下のファイルが作成され，標準出力にパス名が出力される。
E:¥temp¥testjob_000030_00826_55265a48_0002

# ファイルをadshfileコマンドに指定し，echoコマンドで文字列を出力する。
# adshfileコマンドによって，ファイルはジョブ終了時に削除される。
tempfile=$(adshmktemp job1)
"${ADSH_DIR_BIN}adshfile" -n del -a del "$tempfile"
echo "OK" > "$tempfile"
```

9.4.7　adshparsecsv コマンド（CSV データを解析する）

形式

```
adshparsecsv [-e] 配列名
```

機能

このコマンドはスクリプト開発部品の中で使用されるコマンドです。そのため，CSV データを解析する場合は，スクリプト開発部品の CSV 操作の部品の使用を推奨します。

標準入力から読み込んだ CSV データを 2 次元配列に格納します。具体的には，カンマで区切られたデータを 2 次元配列の各要素の値として格納します。入力できる CSV データのサイズは，100KB 以下を対象としています。100KB より大きいサイズのデータを入力すると，ジョブの実行時間が長大化する可能性があります。

CSV データと 2 次元配列の要素の対応例を次に示します。

CSV データ(data.csv)

```
name,value,id
”Yokohama”,200,0001
”Kawasaki”,100,0002
```

2 次元配列(array)の要素と CSV データ(data.csv)の対応

```
array[0][0]=name
array[0][1]=value
array[0][2]=id
array[1][0]=”Yokohama”
array[1][1]=200
array[1][2]=0001
array[2][0]=”Kawasaki”
array[2][1]=100
array[2][2]=0002
```

引数

-e

CSV データに含まれるダブルクォートをメタキャラクタと解釈します。

配列名　～＜環境変数名＞

データを格納する配列名を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

使用例

```
CSVデータ(data.csv)
name,value,id
”Yokohama”,200,0001
”Kawasaki”,100,0002

# CSVデータを2次元配列に格納する。
adshparsecsv array < data.csv
echo ”${array[1][0]}”        #  「”Yokohama”」が出力される。
echo ”${array[1][1]}”        #  「200」が出力される。

# CSVデータのダブルクォートをメタキャラクタと解釈して2次元配列に格納する。
adshparsecsv -e array < data.csv
echo ”${array[1][0]}”        #  「Yokohama」が出力される。
echo ”${array[1][1]}”        #  「200」が出力される。
```

9.4.8　adshparsejson コマンド（JSON データを解析する）

形式

```
adshparsejson  名前
```

機能

このコマンドはスクリプト開発部品の中で使用されるコマンドです。そのため，JSON データを解析する場合は，スクリプト開発部品の JSON 操作の部品の使用を推奨します。

標準入力から読み込んだ JSON データから名前を検索し，完全一致した名前に対応する値を改行区切りで標準出力に出力します。

【UNIX 限定】

JP1/Advanced Shell が動作する環境の環境変数 LANG の値と異なるエンコーディングの JSON データを入力する場合，シェル変数 ADSH_PARSER_LANG に値を設定しておくことで，adshparsejson コマンドを実行する間はエンコーディングを統一して動作させることができます。シェル変数 ADSH_PARSER_LANG には，環境変数 LANG に指定可能な値を設定します。例えば，次の条件に当てはまる場合，エンコーディングを統一して動作させるためには，シェル変数 ADSH_PARSER_LANG に JA_JP.UTF-8 を設定してから adshparsejson コマンドを実行します。

- AIX で環境変数 LANG に SJIS を示す値(Ja_JP)が設定されている
- エンコーディングが UTF-8 の JSON データを入力する

引数

名前

検索する名前を指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- 一致する項目が見つからなかった場合，エラー終了します。
- 同じ名前のデータをネストしている場合，データの最も外側の名前に対応する値を出力します。

  例(test.json)：{"num":{"id":"0001","num":200}}

  上記データに対して num で検索すると，{"id":"0001","num":200}が出力されます。この場合，次のように指定して，データの内側の名前に対応する値(200)を出力できます。

  adshparsejson num < test.json | adshparsejson num

使用例

```
JSONデータ(data.json)
{ ”city”: [
    { ”name”:”Yokohama”, ”id”:”0001”, ”value”:{ ”A”:200, ”B”:100 } },
    { ”name”:”Kawasaki”, ”id”:”0002”, ”value”:{ ”A”:100, ”B”:300 } }
  ]
}

adshparsejson name < data.json
# 以下の内容が出力される。
”Yokohama”
”Kawasaki”

adshparsejson value < data.json
# 以下の内容が出力される。(ネストされているデータをそのまま出力する)
{ ”A”:200, ”B”:100 }
{ ”A”:100, ”B”:300 }

adshparsejson B < data.json
# 以下の内容が出力される。
100
300
```

9.4.9 adshread コマンド（指定した応答要求メッセージを応答待ちイベントとして発行する）

形式

```
adshread　[-d]　変数名　応答要求メッセージ
```

機能

指定された応答要求メッセージを応答待ちイベントとして発行します。発行された応答待ちイベントはJP1/IM - View に表示され，運用者は JP1/IM - View から応答を入力できます。入力された応答は，指定された変数に格納します。

ただし，ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行した場合，応答要求メッセージを標準出力に表示し，応答を標準入力から受け取ります。

JP1 イベントは，前回の JP1 イベントの発行時刻から所定の時間（USERREPLY_JP1EVENT_INTERVAL パラメーターで指定）が経過するのを待って発行されます。USERREPLY_JP1EVENT_INTERVAL パラメーターについては，「7. 環境ファイルで設定するパラメーター」の「USERREPLY_JP1EVENT_INTERVAL パラメーター（JP1 イベントの最小発行間隔を指定する）」を参照してください。

引数

-d

デバッグ実行時に，応答要求メッセージの出力先を標準出力にし，応答を標準入力から受け取ります。デバッグ実行以外の場合，指定は無視されます。

「-」で始まる文字列は，「-」以外の文字で始まる文字列が出現するまでの間，オプションの指定として扱われます。不当なオプションが指定された場合はオプションエラーとなります。

変数名　～<環境変数名>

運用者からの応答内容を格納するシェル変数を指定します。シェル変数は 1 つしか指定できません。シェル変数を複数指定すると，2 つ目以降は応答要求メッセージと解釈されます。

シェル変数が受け取れる文字列は 0～512 バイトの ASCII 文字列です。

ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行し，標準入力からの応答に 513 バイト以上の文字列が指定された場合，513 バイト以降の文字列は無視されます。また，改行を含む文字列が指定された場合，改行以降の文字列は無視されます。

変数に配列を指定する場合は，要素ごとに指定します。1 次元配列の要素数は 0～65,535 の間で指定できます。

2 次元配列の要素数は要素番号指定ごとに 0～65,535 の間で指定できます。

例

通常の変数指定：adshread ans“処理を続行しますか(Y/N)”

配列指定：adshread ans[1] “処理を続行しますか(Y/N)”

指定したシェル変数の属性に対する，このコマンドの実行結果を次の表に示します。

指定したシェル変数の属性	コマンドの実行結果
読み取り専用	KNAX6008-E を出力して終了します。
数値型	KNAX7404-E を出力して終了します。
文字列型/文字列型の配列	値を更新して終了します。
存在しない変数	文字型の変数を新規作成し，値を設定して終了します。
変数名が不正	KNAX6003-E を出力し終了します。
配列の要素数が範囲外の変数	KNAX6007-E を出力し終了します。

応答要求メッセージ　～<任意文字列>((0～1,023 バイト))

応答待ちイベントとして発行する応答要求メッセージを指定します。

応答要求メッセージの文字コードは，同一ホスト内で稼働する JP1/Base と合わせてください。文字コードが異なると文字化けします。

指定された応答要求メッセージは，コマンド「echo -E **応答要求メッセージ**」の実行時と同様の変換内容が応答待ちイベントとして発行されます。複数の応答要求メッセージが指定された場合はエラー（KNAX7403-E）となります。

終了コード

終了コード	意味	対処	リトライの可否
0	正常終了	なし。	－
1	次に示す続行不可能なエラーが発生 • メモリ不足 • 変数名として使用できない変数を検出 • 内部矛盾を検出	変数名として使用できない変数を指定している場合は，変数の指定を変更してください。 問題が解決しない場合は，システム管理者に連絡してください。	×
2	セマフォ（Mutex）・共有メモリの操作でエラーが発生	メッセージに出力されたエラー情報を基に，「ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照して対処してください。	×
3	共有メモリに空きがない	USERREPLY_WAIT_MAXCOUNT パラメーターの設定を見直してください。	○
4	JP1 イベントの処理中にエラーが発生	メッセージに出力されたエラー情報を基に，「ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照して対処してください。	○
5	JP1 イベントの処理中にエラーが発生	メッセージに出力されたエラー情報を基に，「ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照して対処してください。	×

終了コード	意味	対処	リトライの可否
6	指定されたホストへの JP1 イベントの転送に失敗	次の点を確認してください。 • JP1/IM - Manager がインストールされているホストに，JP1/Base がインストールされているか • JP1/IM - Manager がインストールされているホストの JP1/Base のイベントサービスが起動しているか • JP1/Advanced Shell がインストールされているホストと，JP1/IM - Manager がインストールされているホストの間の，JP1/Base のコネクションが確立されているか	○
7	JP1/Base のライブラリが見つからない	JP1/Advanced Shell がインストールされているホストに，JP1/Base がインストールされているか確認してください。 JP1/Base がインストールされていてこの現象が発生した場合は，JP1/Base を再インストールしてください。	×
8	自ホストの JP1/Base のイベントサービスの接続に失敗	JP1/Advanced Shell がインストールされているホストで，JP1/Base のイベントサービスが起動しているか確認してください。	○
10	指定された形式が不正	コマンドの形式を確認してください。	×
128+シグナル番号【UNIX 限定】	adshread コマンドがシグナルを受信して終了	ジョブがシグナルを受信して終了していることを確認してください。	×
200【Windows 限定】	adshread コマンドが強制終了	ジョブが強制終了されていることを確認してください。	×

（凡例）

○：リトライ可

×：リトライ不可

－：対象外

注意事項

- このコマンドは別プロセスで実行しないでください。別プロセスで実行した場合，流量制御（USERREPLY_JP1EVENT_INTERVAL パラメーターで指定）が機能しません。また，運用者からの応答は adshread コマンドを実行したジョブの指定された変数に格納されません。
- このコマンドはパイプを指定して実行しないでください。
- このコマンドにリダイレクトで値を受け取る処理を指定しないでください。
- ユーザー応答機能の入出力先に標準入出力を指定してデバッグ実行する場合を除いて，JP1/Base や JP1/IM が存在しない環境やユーザー応答機能管理デーモンまたはサービスが起動していない状態では実行しないでください。

  実行すると次の問題が発生します。

- JP1/Advanced Shell が稼働するホストに JP1/Base がインストールされていない場合，コマンドはエラー終了します。
- JP1/Advanced Shell が稼働するホストの JP1/Base のイベントサービスが稼働していない場合，コマンドはエラー終了します。
- HOSTNAME_JP1IM_MANAGER で指定されたホストに JP1/Base がインストールされていない，または JP1/Base のイベントサービスが稼働していない場合，コマンドはエラー終了します。
- HOSTNAME_JP1IM_MANAGER で指定されたホストの JP1/IM - Manager が稼働していない場合でも，HOSTNAME_JP1IM_MANAGER で指定されたホストの JP1/Base のイベントサービスに JP1 イベントが到達した時点で JP1 イベントの送信に成功したとしてコマンドは動作します。
- ユーザー応答機能管理デーモン・サービスが起動していない場合，コマンドはエラー終了します。

- リトライ可能なエラーでコマンドが終了した場合は，そのコマンドを再実行することで成功する可能性があります。コマンドを再実行したい場合は，「3.8.5　adshecho コマンドまたは adshread コマンドがエラー終了した場合の対処」に記載しているジョブ定義スクリプトの記述例を参考として，そのコマンドを再実行するようにジョブ定義スクリプトを作成してください。
- adshread コマンドが応答要求メッセージの応答待ちの状態で，ジョブを「3.11.1　ジョブの強制終了の方法」で示す以外の方法で即時終了した場合，共有メモリ上に応答要求メッセージの情報が残り，JP1/IM - View に応答待ちイベントが滞留したままになることがあります。その場合，adshchmsg コマンドの-d オプションで応答要求メッセージの応答待ち状態をキャンセルするか，ユーザー応答機能管理デーモン・サービスを再起動してください。
- TRAP_ACTION_SIGTERM パラメーターで TERM を指定した場合，または UNIX 版で AUTO を指定して JP1/AJS からジョブを起動した場合は，trap コマンドによる動作定義に adshread コマンドを指定しないでください。

使用例

- 応答要求メッセージを出力し，運用者からの応答に応じて処理を決定します。

```
adshread  ans "処理を続行しますか(Y/N)"

if [ "$ans" = "Y" ] ; then
     echo "処理を続行します。"
elif [ "$ans" = "N" ] ; then
     echo "処理を終了します。"
     exit 1
else
     echo "指定以外の応答が入力されました。処理を終了します。"
     exit 1

fi
```

9.4.10 adshscripttool コマンド（ジョブ定義スクリプトの作成を支援する）【Windows 限定】

形式

```
adshscripttool -fowner [-L] パス名
adshscripttool -fentry [-L] パス名
adshscripttool -fmode  [-s  {u|g|o|r|w|x} ]  モード
adshscripttool -exec  [-m  {SIMPLE|MINIMUM} ]
               {-r コマンドライン|ジョブ定義スクリプトファイルのパス名}
               [実行時パラメーター]
```

機能

ジョブ定義スクリプトを作成しやすくするための情報の取得や出力などを実行します。指定できる引数と用途を次に示します。

引数	実行内容	用途
-fowner	ファイルまたはフォルダの所有者名を出力します。	ファイルまたはフォルダの所有者のアクセス権を変更するために，ファイルまたはフォルダの所有者名を取得したい場合に使用します。
-fentry	ファイルまたはフォルダの ACL に登録されているアカウント名の一覧を出力します。	取得した ACL のアカウント情報に従って，ジョブ定義スクリプト内の cacls コマンドや attrib コマンドの実行を変更したい場合に使用します。
-fmode	chmod コマンドのモードとして指定されたシンボルまたは数値を解析して，「所有者」「グループ」「その他のユーザー」に対しての権限の変更内容を，ジョブ定義スクリプト中で使用しやすいように 9 桁の文字列として出力します。 -s オプションと同時に指定すると，-fmode オプション指定時の出力のうち，ugorwx の指定に対応する桁の文字だけを出力します。	シンボルまたは数値の指定内容に従って，ジョブ定義スクリプト内の cacls コマンドや attrib コマンドの実行を変更したい場合に使用します。
-exec	指定したコマンドラインまたはジョブ定義スクリプトを子孫ジョブとして実行します。	指定したコマンドラインまたはジョブ定義スクリプトを子孫ジョブとして実行したい場合に使用します。

引数

-L

情報の取得先を選択します。

パス名に指定したファイルが通常ファイルの場合，指定したファイルの情報を取得します。パス名に指定したファイルがシンボリックリンクの場合，-L オプションの指定に従います。

-L オプションが指定された場合，シンボリックリンクが指す実体の情報を取得します。

-L オプションが指定されなかった場合，シンボリックリンクの情報を取得します。

環境変数 ADSH_LINK_SUPPORT に L0 を指定した場合，-L オプションを使用できません。

-fowner

ファイルまたはフォルダの所有者名を標準出力に出力します。

出力する所有者名の形式は，「**ドメインまたはコンピュータ名¥ユーザー名**」または「**ユーザー名**」となります。

cacls コマンドで Creator Owner を定義しても，ファイルまたはフォルダの所有者とマッピングされない場合があります。cacls コマンドで所有者名を指定する前に，このオプションを指定して adshscripttool コマンドを実行し，所有者名を求めておいてください。

-fentry

ファイルまたはフォルダの ACL に登録されているアカウント名の一覧を「;（セミコロン）」区切りで標準出力に出力します。

出力するアカウント名の形式は「**ドメインまたはコンピュータ名¥ユーザー名**」または「**ユーザー名**」となります。

パス名

対象とするファイルまたはフォルダを指定します。

-fmode

chmod コマンドのモードとして指定されたシンボルまたは数値を解析し，「所有者」「グループ」「その他のユーザー」に対しての権限の変更内容を 9 桁の文字列として標準出力に出力します。

数値が指定された場合，モードビット ON に対応する値を R に，モードビット OFF に対応する値を D に設定します。エラー時は'E'だけを出力して終了します。

-fmode オプション指定時の adshscripttool コマンドの実行結果として出力される文字列の意味を桁ごとに次に示します。

左から数えた桁番号	意味
1	所有者の読み込み権限
2	所有者の書き込み権限
3	所有者の実行権限
4	グループの読み込み権限
5	グループの書き込み権限
6	グループの実行権限
7	その他のユーザーの読み込み権限
8	その他のユーザーの書き込み権限
9	その他のユーザーの実行権限

各桁には次の値が設定されます。

値	意味
A	追加（「+」が設定された）
D	削除（「-」が設定された）

値	意味
R	置き換え（「=」または数値が設定された）
0	指定されていない
E	adshscripttool -fmode コマンドの実行がエラー終了

-s {u|g|o|r|w|x}

-fmode オプション指定時の出力のうち，出力する桁を次の中から指定します。-fmode オプションの指定時に指定できます。

- u

  -fmode オプション指定時の出力のうち，1～3 桁目に対応します。

- g

  -fmode オプション指定時の出力のうち，4～6 桁目に対応します。

- o

  -fmode オプション指定時の出力のうち，7～9 桁目に対応します。

- r

  -fmode オプション指定時の出力のうち，1，4，7 桁目に対応します。

- w

  -fmode オプション指定時の出力のうち，2，5，8 桁目に対応します。

- x

  -fmode オプション指定時の出力のうち，3，6，9 桁目に対応します。

モード

8 桁の数値またはシンボルを指定します。-fmode オプションの指定時に指定できます。

- 数値で指定する場合

  8 進数で指定します。8 進数以外または 8 進数の 07777（10 進数の 4095）より大きな値が指定されるとエラーとなります。

- シンボルで指定する場合

  何も指定されていない状態（数値表現での 0）に対して設定・追加・削除を設定します。シンボルの指定結果が結果として出力されます。

  シンボルとして指定できる内容を次に示します。複数指定する場合は間をコンマ（,）で区切ってください。

シンボル内の順序	指定できる値
1 つ目	アクセス権を設定する項目を指定します。複数同時に指定できます。 指定できる項目を次に示します。省略するとすべてのユーザーが仮定されます。 u：所有者 g：グループ

シンボル内の順序	指定できる値
1 つ目	o：その他 a：全ユーザー
2 つ目	モードに対する操作を指定します。1 つ目のシンボルで指定された項目に対して次の処理をします。 ＝：アクセス権の設定（上書き） ＋：アクセス権の追加 -：アクセス権の削除 設定，追加および削除する値は，3 つ目のシンボルで指定します。 3 つ目のシンボルに続いて 2 つ目および 3 つ目のシンボルを記述できます。3 つ目のシンボルは省略できます。
3 つ目	設定するアクセス権を指定します。複数同時に指定できます。指定できる値を次に示します。 r：読み取り w：書き込み x：実行 省略するとアクセス権を設定する項目を消去します。消去した値を 2 つ目のシンボルに従って設定，追加および削除します。追加および削除だけでは値は変化しません。 s，t，u，g，o は指定しても無視されます。

-exec

-r オプションに指定したコマンドライン，または指定したジョブ定義スクリプトファイルを子孫ジョブとして実行します。

-m　{SIMPLE|MINIMUM}

標準出力，標準エラー出力への出力方式を指定します。-exec オプションの指定時に指定できます。簡潔出力モードと最小出力モードについては，「3.4.4　ジョブ実行ログへの情報メッセージと警告メッセージの出力を抑止する」を参照してください。

- SIMPLE

  実行する子孫ジョブを簡潔出力モードで動作します。

- MINIMUM

  実行する子孫ジョブを最小出力モードで動作します。

子孫ジョブ形式のサンプルスクリプトは最小出力モードで動作します。そのため，子孫ジョブ形式のサンプルスクリプト内でこのコマンドを使用する場合，-m オプションには MINIMUM の指定を推奨します。

-r コマンドライン

ジョブで実行する内容をコマンドラインに指定します。-exec オプションの指定時に指定できます。

-r オプションに指定した内容は，スプールジョブディレクトリにジョブ定義スクリプトファイルとして作成されません。そのため，メッセージなどのジョブ定義スクリプトファイル名の出力個所には，ジョブ定義スクリプトファイルの絶対パスではなく「"-r CMDLINE"」が出力されます。

コマンドラインの指定

コマンドラインには，シェル運用コマンドや UNIX 互換コマンドなど，ジョブ定義スクリプトに記載できるコマンドを指定できます。コマンドラインにシェル標準コマンドの pwd コマンドを指定する例を次に示します。

```
adshscripttool -exec -m MINIMUM -r pwd
```

また，コマンドセパレータによる複数コマンド記述など，ジョブ定義スクリプトファイルに記述する内容を指定できます。コマンドラインに複数のコマンドを指定する例を次に示します。

```
adshscripttool -exec -m MINIMUM -r ”export DATA=file01 ; pgm001”
```

コマンドラインにスペースを指定する場合，クォーテーション（「'」または「"」）で囲む必要があります。また，コマンドを実行するシェルによっては，コマンドラインに指定した$，*，；（セミコロン）などのメタキャラクタが展開されるため，クォーテーションで囲むか，エスケープ文字（¥）を使用する必要があります。

子孫ジョブの出力情報

-r オプションを指定した場合，ジョブコントローラが出力するメッセージテキストや，ジョブ定義スクリプトの稼働実績情報には，ジョブ定義スクリプトファイルのパス名として，ジョブ定義スクリプトファイルの絶対パスではなく「"-r CMDLINE"」が出力されます。

位置パラメーターの格納情報

-r オプションのコマンドラインに位置パラメーター$0 を指定する場合，$0 には「adshexec」が格納されます。

SPOOLJOB_CHILDJOB パラメーターとの関連

-r オプションを指定して実行する場合，SPOOLJOB_CHILDJOB パラメーターに MERGE を指定して実行したときに，ルートジョブのジョブ実行ログに出力される子孫ジョブのジョブ実行ログの開始と終了を表す記号には，ジョブ定義スクリプトファイルの絶対パスではなく次の値が出力されます。SPOOLJOB_CHILDJOB パラメーターに MERGE を指定した時の出力形式は「(3)　子孫ジョブのスプールジョブをルートジョブのスプールジョブへマージした場合」を参照してください。

・子孫ジョブの JOBLOG の開始を表す記号

「>>>>>> [JOBLOG] "-r CMDLINE"」

・子孫ジョブの JOBLOG の終了を表す記号

「<<<<<< [JOBLOG] "-r CMDLINE"」

・子孫ジョブの標準エラー出力の開始を表す記号（通常実行時）

「>>>>>> [STDERR] "-r CMDLINE"」

・子孫ジョブの標準エラー出力の終了を表す記号（通常実行時）

「<<<<<< [STDERR] "-r CMDLINE"」

・子孫ジョブの標準エラー出力および標準出力の開始を表す記号（デバッグ実行時）

「>>>>>> [STDERR,STDOUT] "-r CMDLINE"」

・子孫ジョブの標準エラー出力および標準出力の終了を表す記号（デバッグ実行時）

「<<<<<< [STDERR,STDOUT] "-r CMDLINE"」

また，スクリプトイメージファイルに出力されるジョブ定義スクリプトファイル名の部分にも「"-r CMDLINE"」が出力されます。

ジョブ定義スクリプトファイルのパス名　～<パス名>((1～247 バイト))

実行するジョブ定義スクリプトのファイルのパス名を指定します。-exec オプションの指定時に指定できます。

実行時パラメーター　～<任意文字列>((1～1,022 バイト))

-r オプションに指定したコマンドラインまたはジョブ定義スクリプトファイルの位置パラメーターに格納する値を指定します。-exec オプションの指定時に指定できます。

スペースを実行時パラメーターとして指定する場合は，その文字列を"（ダブルクォーテーション）で囲んでください。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

使用例

- -fowner オプションの指定例と，標準出力への出力結果を示します。

  ジョブ定義スクリプトの内容

```
adshscripttool -fowner test.txt
```

  標準出力への出力結果

```
MYPC¥user1
```

- -fentry オプションの指定例と，標準出力への出力結果を示します。

  ジョブ定義スクリプトの内容

```
adshscripttool -fentry test.txt
```

  標準出力への出力結果

```
BUILTIN¥Administrators;NT AUTHORITY¥SYSTEM;MYPC¥user1;BUILTIN¥Users
```

- -fmode オプションに「+w」を指定した例と，標準出力への出力結果を示します。

  ジョブ定義スクリプトの内容

```
adshscripttool -fmode +w
```

  標準出力への出力結果

```
0A00A00A0
```

- -fmode オプションに「ug-r」を指定した例と，標準出力への出力結果を示します。

ジョブ定義スクリプトの内容

```
adshscripttool -fmode ug-r
```

標準出力への出力結果

```
D00D00000
```

- -fmode オプションに「ug-w,u+w」を指定した例と，標準出力への出力結果を示します。

  ジョブ定義スクリプトの内容

```
adshscripttool -fmode ug-w,u+w
```

  標準出力への出力結果

```
0A00D0000
```

- -fmode オプションに「655」を指定した例と，標準出力への出力結果を示します。

  ジョブ定義スクリプトの内容

```
adshscripttool -fmode 655
```

  標準出力への出力結果

```
RRDRDRRDR
```

- -fmode オプションの指定例と，標準出力への出力結果を示します。この例では-s w 指定で「+w」を解析しています。

  ジョブ定義スクリプトの内容

```
adshscripttool -fmode -s w +w
```

  標準出力への出力結果（0A00A00A0 の結果のうち，2 桁目と 5 桁目と 8 桁目の結果を出力）

```
AAA
```

- -fmode オプションの指定例と，標準出力への出力結果を示します。この例では-s r 指定で「655」を解析しています。

  ジョブ定義スクリプトの内容

```
adshscripttool -fmode -s r 655
```

  標準出力への出力結果（RRDRDRRDR の結果のうち，1 桁目と 4 桁目と 7 桁目の結果を出力）

```
RRR
```

- -fmode オプションの指定例と，標準出力への出力結果を示します。この例では-s uor 指定で「655」を解析しています。

  ジョブ定義スクリプトの内容

```
adshscripttool -fmode -s uor 655
```

  標準出力への出力結果（RRDRDRRDR の結果のうち，1 桁目と 7 桁目の結果を出力）

```
RR
```

- シンボルの解析結果に従って cacls コマンドの実行を切り替えるよう，ジョブ定義スクリプトで指定する例を次に示します。

```
username=`adshscripttool -fowner "$1"`      # 所有者名を取得する
if [[ $? -ge 1 ]]                           # adshscripttool -fownerのエラー処理
then
  echo "adshscripttool -fowner error."
  return 1
fi
modebit=`adshscripttool -fmode $mode`       # モードを解析する(mode=u+w)
case $modebit in                            # modebitに0A0000000が格納されている
  "AA0000000" ) cacls "$1" /E /G $username:C ;;
  "0A0000000" ) cacls "$1" /E /G $username:W ;;     # このcaclsが実行される
          "E" ) echo "adshscripttool -fmode error." # adshscripttool -fmodeのエラー処理
                return 1 ;;
esac
```

- 所有者と Everyone 以外の ACE を削除するよう，ジョブ定義スクリプトで指定する例を次に示します。

```
IFS=¥;
username=`adshscripttool -fowner "$1"`     # 所有者名を取得する
if [[ $? -ge 1 ]]                          # adshscripttool -fownerのエラー処理
then
  echo "adshscripttool -fowner error."
  return 1
fi
set -A entry `adshscripttool -fentry $1`   # アカウント名一覧を取得
for i in "${entry[@]}"
do
  if ! [[ $i == "$username" || $i == "Everyone" ]]
  then
    cacls "$1" /E /R "$i"                  # 所有者とEveryone以外のACEを削除
  fi
done
```

- 数値の解析結果のうち，所有者の定義内容だけに従って cacls コマンドの実行を切り替えるよう，ジョブ定義スクリプトで指定する例を次に示します。

```
username=`adshscripttool -fowner "$1"`     # 所有者名を取得する
if [[ $? -ge 1 ]]                          # adshscripttool -fownerのエラー処理
then
  echo "adshscripttool -fowner error."
  return 1
fi
modebit=`adshscripttool -fmode -s u $mode` # モードを解析する(mode=644)
case $modebit in                           # modebitにRRD------が格納されている
  "RRR" ) cacls "$1" /P $username:F ;;
  "RRD" ) cacls "$1" /P $username:C ;;          # このcaclsが実行される
  "RDD" ) cacls "$1" /P $username:R ;;
  "DRD" ) cacls "$1" /P $username:W ;;
    "E" ) echo "adshscripttool -fmode error."   # adshscripttool -fmodeのエラー処理
          return 1 ;;
esac
```

- コマンドライン「export DBPATH=C:¥¥HOME¥¥DBUSER; start -q」を子孫ジョブとして実行する例を示します。

```
adshscripttool -exec -m MINIMUM -r 'export DBPATH=C:¥¥HOME¥¥DBUSER; start -q '
```

- ジョブ定義スクリプトファイル「ppstart.ash」を子孫ジョブとして実行する例を示します。

```
adshscripttool -exec -m MINIMUM ppstart.ash
```

注意事項

- -fowner オプションと-fentry オプションを指定する場合は，対象のファイルまたはフォルダの所有者が実行してください。所有者でない場合，エラーメッセージを出力してエラー終了することがあります。
- -fowner オプション，または-fentry オプションの前に-L オプションは指定できません。指定した場合，コマンドの解析エラーになります。
- -fmode オプションの前に-s オプションは指定できません。指定した場合，コマンドの解析エラーになります。
- 内部矛盾を検出した場合，実行しているシェルを終了します。内部矛盾以外のエラーの場合，実行しているシェルは終了しないで処理を続行します。
- -fentry オプションで各 ACE のセキュリティ情報を参照中にエラーになった場合，参照できた ACE のアカウント名までの結果を出力し，エラー終了します。
- -exec オプションで子孫ジョブを実行した場合，ジョブ実行ログに出力されるコマンド名は JP1/Advanced Shell のコマンド（adshexecsub コマンド）となります。

9.4.11　adshvarconv コマンド（変数の値を変換する）

形式

```
adshvarconv [-o] -p シェル変数名 [シェル変数名 ・・・]
adshvarconv [-o] [-c] -b 変換前文字列 [-a 変換後文字列]
                         シェル変数名 [シェル変数名 ・・・]
adshvarconv [-o] -i ¥の増加数 シェル変数名 [シェル変数名 ・・・]

【Windows限定】
adshvarconv [-o] [-u] -e 変換前エンコーディング 変換後エンコーディング
                         シェル変数名 [シェル変数名 ・・・]
```

機能

オプションに従って変数の値を変換します。

オプション	機能
-p	パス変換ルールに従ってシェル変数値を変換します。

オプション	機能
-b,-a,-c	シェル変数値の**変換前文字列**を**変換後文字列**に変換します。
-i	シェル変数値の¥を指定個数分増加します。
-e,-u	シェル変数値のコード変換を行います。【Windows 限定】

-o オプションについて次に示します。

-o オプション	出力先
無し	元のシェル変数値を変換後の値で更新します。
有	元のシェル変数値は変更せずに，変換後の値を標準出力に出力します。

引数

-p

パス変換ルール（PATH_CONV_ENABLE,PATH_CONV 環境設定パラメーター）に従って，シェル変数の値を変換します。adshvarconv コマンド実行時に変換します。このため，コマンド実行後に変数の値を変更した場合，その値は自動的に変換されません。この変更後の値がパス変換ルールに一致する値であっても，自動的に変換しません。

パス変換の動作は，変換ルールとは異なります。パス変換の動作を次に示します。

パス変換動作
PATH_CONV_ENABLE で指定したパス区切り文字で区切られた，それぞれのパスを環境設定パラメーター PATH_CONV で変換する。 注　環境設定パラメーターでは Windows のディレクトリ区切り記号は¥¥と記述しますが，このコマンドの変換では¥2 個を¥1 個に自動的に変更して変換します。
PATH_CONV 環境設定パラメーターで変換した場合，パス区切り文字，ディレクトリ区切り文字を該当 OS の区切り文字に変換します。 注　通常のパス変換機能では Windows のディレクトリ区切り記号は¥¥に変換しますが，このコマンドの変換では¥1 個に変換します。

パス変換機能が無効のとき，このコマンドは何もしません。

-b 変換前文字列 [-a 変換後文字列]

シェル変数値の**変換前文字列**を**変換後文字列**に変換します。-a を指定しない場合は，**変換前文字列**を削除します。

変換前文字列～<任意文字列>((1 から 256 バイト))

変換後文字列～<任意文字列>((1 から 256 バイト))

-c

パス変換機能が有効の場合だけ変換します。パス変換機能が無効のときは，このコマンドは何もしません。-c 指定がないときは，無条件に変換します。

-i ¥の増加数

シェル変数値の¥をそれぞれの文字に対して，指定個数分増加します。-i 1 と指定した場合，変換前の文字列が abc¥def¥¥ghi¥¥¥jkl であれば，変換後は abc¥¥def¥¥¥¥ghi¥¥¥¥¥¥jkl です。

¥の増加数～<3 桁の十進数>((1～256))

-e 変換前エンコーディング 変換後エンコーディング【Windows 限定】

シェル変数の値をコード変換します。

- **変換前エンコーディング**～{SJIS | UTF8}

  SJIS：エンコーディングが Shift-JIS であることを示します。

  UTF8：エンコーディングが UTF-8 であることを示します。

- **変換後エンコーディング**～{SJIS | UTF8}

  SJIS：エンコーディングが Shift-JIS であることを示します。

  UTF8：エンコーディングが UTF-8 であることを示します。

変換前エンコーディングと変換後エンコーディングの組み合わせの妥当性チェックはしません。

-u

コード変換時に変換できないコードがあった場合，エラー終了します。

-u オプションを指定しない場合，変換できない文字を「?」に変換して処理を続行します。

-o

変換後の値を標準出力に出力します。シェル変数値は更新しません。

-o オプションを指定する場合，複数のシェル変数名を指定できません。配列で複数要素を指定することもできません。

-e オプション以外の場合，変換をしない場合でも-o オプションがあるときは，変換前の変数の値を標準出力に出力します。

-e オプションでは変換できなかったときは-o オプションがあっても標準出力には結果を出力しません。

シェル変数名　～<シェル変数名>((1 から 256))

値を変更するシェル変数を指定します。長さの上限は，配列の場合，引数指定を含んだ文字列の長さの上限です。

配列を指定する場合は次の形式で指定します。

配列の要素番号は 0 から 65,535 まで指定できます。

配列の指定方法	説明
array[n]	n 番目の要素
array[@]	全要素
array[*]	全要素
array[]	array[0]と同一
array[n][m]	n 行 m 列の要素
array[n][@]	n 行目のすべての要素

配列の指定方法	説明
array[n][*]	n 行目のすべての要素
array[n][]	array[n][0]と同一
array[@][m]	m 列のすべての要素
array[@][@]	すべての要素
array[@][*]	すべての要素
array[@][]	すべての要素
array[*][m]	m 列のすべての要素
array[*][@]	すべての要素
array[*][*]	すべての要素
array[*][]	すべての要素
array[][m]	array[0][m]と同一
array[][@]	すべての要素
array[][*]	すべての要素
array[][]	array[0][0]と同一

終了コード

終了コード	意味
0	正常終了
1	• -e で変換できない文字がありました。(-u を指定した場合) • 変数が読み込み専用属性です。 • 変数が整数型属性です。 • 変数名が妥当ではありません。 • 変数名の指定がありません。 • -a があるときに-b がありません。 • 変換前文字列，変換後文字列の文字列長が正しくありません。 • 値が必要なオプションに値の指定がありません。 • オプションの組み合わせが誤っています。 • -o オプション指定時に複数の変数を指定しています。 • -o オプション指定時に配列要素に*または@を指定しています。 • -e でエンコーディングの指定が不当です。 • 不当なオプションを指定しています。 • 必要なオプションが指定されていません。 • 予期しないエラーが発生しました。

注意事項

- 同一変数を複数回指定すると 2 重に変換します。
- 配列変数で指定した要素が存在しないとき，エラーにしないで後続の変数の処理をします。
- 指定した変数名が存在しないとき，エラーにしないで後続の変数の処理をします。
- 右詰め属性などで値が変数格納時に加工されている場合，パス変換ルールによる変換では，属性によって加工された値を変換ルールと評価するため，変換対象にならないことがあります。
- 変換後の値は変数の属性によって加工されることがあるので注意が必要です。

  たとえば typeset -R10（右詰で領域長が 10 文字）の場合，変換後の値が領域長を超えると，仕様に従って切り捨てが発生します。また，先頭の空白を削除しても領域長より短い場合は先頭に空白が挿入されます。

使用例

- ジョブ定義スクリプト起動時のパラメーターにパスが指定されていて，それを PATH_CONV の変換ルールで変換したい場合。
  - 環境設定パラメーター

    ```
    #-adsh_conf PATH_CONV_ENABLE / :
    #-adsh_conf PATH_CONV_RULE 2
    #-adsh_conf PATH_CONV /home/user001 ”d:¥¥user001”
    ```

  - ジョブ定義スクリプト（sample.ash）

    ```
    infile=$1
    adshvarconv -p infile
    ”${ADSH_DIR_CMD}cat”  ${infile}
    ```

  - 実行例

    ```
    D:¥user001>cat d:¥user001¥zzzz.txt
    Data_zzzz.txt(d:¥user001¥zzzz.txtファイルの内容)

    D:¥user001>adshexec -m MINIMUM sample.ash  /home/user001/zzzz.txt
    Data_zzzz.txt(d:¥user001¥zzzz.txtファイルの内容)
    ```

- ファイルに記述されたパスを引数の値で変換する場合。
  - ジョブ定義スクリプト（sample.ash）

    ```
    while read LINE
    do
      adshvarconv -b ”/home/user1” -a ”$1” LINE
      adshvarconv -b ”/” -a ”¥¥” LINE
      echo -E ”$LINE” >&2
    done  < input.txt
    ```

  - input.txt

```
/home/user1/data001
/home/user1/data002
```

- 実行例

```
D:¥user001>adshexec -m MINIMUM sample.ash D:/home/winuser001
D:¥home¥winuser001¥data001
D:¥home¥winuser001¥data002
```

- 入力データから¥de の文字列を削除する場合。
  - ジョブ定義スクリプト（sample.ash）

```
echo -E 'abc¥de¥kkk' >test.txt
echo -E '123¥de¥kkk' >>test.txt
"${ADSH_DIR_CMD}cat" test.txt >&2
echo -E start_sed >&2
"${ADSH_DIR_CMD}cat" test.txt | "${ADSH_DIR_CMD}sed" -e 's/¥de//' >&2
echo -E start_adshvarconv >&2
while read -r LINE
do
adshvarconv -o -b '¥de' LINE >&2
done < test.txt
```

  - 実行結果

```
abc¥de¥kkk
123¥de¥kkk
start_sed
abc¥¥kkk
123¥¥kkk
start_adshvarconv
abc¥kkk
123¥kkk
```

  sed コマンドのパターンに指定した文字列の¥はエスケープ文字として扱われるため，正しく¥de の文字列の削除ができません。

- awk コマンドにパス名を渡す時に¥を 1 個増加して，エスケープ文字の処理で¥が消えないようにする場合。
  - ジョブ定義スクリプト（sample.ash）

```
aa=d:¥¥g1234z¥¥azzzz
echo -E 'hitachi' | "${ADSH_DIR_CMD}awk" -v VVV1=$aa '/hitachi/ { ¥
print "VVV1=" VVV1 ; ¥
}' >&2
adshvarconv -i 1 aa
echo -E 'hitachi' | "${ADSH_DIR_CMD}awk" -v VVV1=$aa '/hitachi/ { ¥
print "VVV1=" VVV1 ; ¥
}' >&2
```

  - 実行例

```
D:¥user001>adshexec -m MINIMUM sample.ash
VVV1=d:g1234zzzzz
VVV1=d:¥g1234z¥azzzz
```

- ファイルを read で読んで UTF-8 から Shift-JIS に変換する場合。
  - ジョブ定義スクリプトファイル（sample1.ash）

```
echo -E "--- before ---"
"${ADSH_DIR_CMD}ls" outdir
while read LINE
do
  "${ADSH_DIR_CMD}cp"  $(adshvarconv -o -e UTF8 SJIS  LINE)  outdir
done  < input.txt
echo -E "--- after ---"
"${ADSH_DIR_CMD}ls" outdir
```

  - input.txt ファイル(UTF-8)

```
東京.txt
京都.txt
```

  - 実行例

```
D:¥user001>adshexec -m MINIMUM sample.ash
--- before ---
--- after ---
京都.txt 東京.txt
```

9.5　スクリプト拡張コマンド

スクリプト拡張コマンドとは，ジョブ定義スクリプトファイルに記載する「#-adsh_」で始まるコマンドのことです。

ファイルの作成および環境変数への割り当て，ジョブ定義スクリプトまたはジョブステップ実行後のファイルの後処理や，ジョブ定義スクリプトのジョブ名を宣言できます。また，ジョブステップを定義してジョブの実行を制御したり，スクリプト拡張コマンドを記載した外部のスクリプトを呼び出したりできます。

スクリプト拡張コマンドの終了コードは，環境設定パラメーターの ADSHCMD_RC_ERROR パラメーターおよび ADSHCMD_RC_SUCCESS パラメーターで変更できます。ただし，次の場合の終了コードは変更できません。

- #-adsh_step_end コマンドでジョブステップ正常終了およびジョブステップエラー終了した場合
- #-adsh_script コマンドで正常終了した場合

環境設定パラメーターについては，「7.　環境ファイルで設定するパラメーター」を参照してください。

また，スクリプト拡張コマンドは別プロセスで実行しないでください。

9.5.1　#-adsh_file コマンド（通常ファイルの割り当ておよび後処理を指定する）

形式

```
#-adsh_file ファイル環境変数定義名 ファイルパス
            [-chk {exist|no} ]
            [-normal {del|keep} ] [-abnormal {del|keep} ]
```

機能

通常ファイルの割り当て，通常ファイルの存在有無の確認および後処理を指定します。通常ファイルの割り当ては，4,095 個まで指定できます。

通常ファイルの割り当て，後処理，adshfile コマンドとの機能差については，「5.9.1　通常ファイルの割り当ておよび後処理をする」を参照してください。

このコマンドで割り当てた通常ファイルは，adshfile コマンドで割り当てた通常ファイルとは別に管理され，後処理は adshfile コマンド，#-adsh_file コマンドの順に実行されます。そのため，両方のコマンドで同じファイルを割り当てると，ファイルの後処理が二重に実行されることになり，エラーが発生する場合があるので注意してください。

引数

ファイル環境変数定義名

～<環境変数名>((1～31 バイト))

割り当てる通常ファイルを識別するキーとなる，ファイル環境変数定義名を指定します。

Windows の場合，環境設定パラメーター VAR_ENV_NAME_LOWERCASE で ENABLE を指定していれば小文字を指定できます。DISABLE を指定していれば小文字は指定できません。

ファイルパス

Windows の場合　～<パス名>((1～247 バイト))

UNIX の場合　～<パス名>((1～1,023 バイト))

割り当てる通常ファイルのパスを指定します。

相対パスを指定した場合は絶対パスに変換されて実行されます。そのため，絶対パスに変換されたあとのパス長が，OS で規定されているパス長の上限を超えないよう注意してください。OS のパス長の上限を超えると実行時にエラーになります。

-chk {exist|no}

割り当てる通常ファイルの存在確認の有無を指定します。指定が省略されている場合，no が指定されたものとします。

- exist

  ファイルの有無を確認します。

- no または指定なし

  ファイルの有無を確認しません。

-normal {del|keep}

該当するジョブステップまたはジョブが正常終了した場合の後処理を指定します。指定が省略されている場合，keep が指定されたものとします。

- del

  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除します。

- keep

  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除しません。

-abnormal {del|keep}

該当するジョブステップまたはジョブがエラー終了した場合の後処理を指定します。指定が省略されている場合，keep が指定されたものとします。

- del

  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除します。

- keep

  該当するジョブステップまたはジョブ終了後，割り当てた通常ファイルを削除しません。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

9.5.2 #-adsh_file_temp コマンド（一時ファイルの割り当ておよび後処理をする）

形式

```
#-adsh_file_temp ファイル環境変数定義名 [ -id 一時ファイル識別名]
                 [-chk {create|exist} ]
                 [-normal {del|keep} ]
```

機能

ジョブ定義スクリプト内で一時的に使用するファイルの割り当ておよび後処理を指定します。一時ファイルの割り当ては，4,095 個まで指定できます。#-adsh_file_temp コマンドを使った一時ファイルの作成については，「5.9.2　一時ファイルの割り当ておよび後処理をする」を参照してください。

引数

ファイル環境変数定義名

～<環境変数名>((1～31 バイト))

割り当てた一時ファイルを識別するキーとなる，ファイル環境変数定義名を指定します。

Windows の場合，環境設定パラメーター VAR_ENV_NAME_LOWERCASE で ENABLE を指定していれば小文字を指定できます。DISABLE を指定していれば小文字は指定できません。

-id 一時ファイル識別名

～<記号名称>((1～31 バイト))

ジョブステップ内で作成した一時ファイルを後続ジョブステップで使用する場合，使用するファイルを特定するために，一時ファイル識別名を指定します。割り当てた一時ファイルを以降のジョブステップで使用しない場合，指定を省略できます。ジョブステップ外で割り当てる場合は指定できません。

一時ファイル識別名は，作成する一時ファイルごとに一意にしてください。先行ジョブステップで作成済みの一時ファイルと同じ識別名は指定できません。ただし，先行ジョブステップの後処理で削除済みのファイルの識別名は指定できます。

-chk {create|exist}

一時ファイルの割り当て方法を指定します。指定が省略されている場合，create が指定されたものとします。

- create
  割り当てる一時ファイルを新規に作成して割り当てます。ジョブコントローラがファイル名を生成し0バイトファイルを作成します。
- exist
  先行ジョブステップで作成した一時ファイルを割り当てる場合に指定します。ジョブステップ外で割り当てる場合，および一時ファイル識別名を省略した場合は指定できません。

-normal {del|keep}

一時ファイルの後処理を指定します。指定が省略されている場合，del が指定されたものとします。

- del
  該当するジョブステップまたはジョブ終了後，割り当てた一時ファイルを削除します。
- keep
  該当するジョブステップ終了時，割り当てた一時ファイルを削除しません。ジョブステップ外で割り当てる場合，および一時ファイル識別名を省略した場合は指定できません。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

9.5.3 #-adsh_job コマンド（ジョブ名を宣言する）

形式

```
#-adsh_job ジョブ名
```

機能

ジョブ定義スクリプトのジョブ名を宣言します。ジョブ名の宣言は，1行目または2行目のどちらかに1個指定できます。

引数

ジョブ名

～<記号名称>((1～31バイト))

ジョブを識別する情報の1つであるジョブ名を定義します。ジョブ名はジョブ実行ログなどにメッセージとして表示されるほか，ジョブコントローラが作成するファイル名の一部にも使用されます。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

9.5.4　#-adsh_job_stop コマンド（ジョブの打ち切り条件を定義する）

形式

```
#-adsh_job_stop 終了コード定義 [,終了コード定義 ...]
```

機能

ジョブステップ終了時に，ジョブを打ち切るかどうかを判断する条件を定義します。ジョブの打ち切り条件の定義は，1,023 個まで指定できます。

引数

終了コード定義［,終了コード定義］...

ジョブを打ち切ると判断するジョブステップの終了コードの値を定義します。

終了コード定義を「,」で区切って複数指定した場合，どれかの定義を満たしたときにジョブを打ち切ります。終了コード定義は 8 個まで指定できます。

終了コード定義

～<符号なし整数>((0～255))

・終了コード

終了コードの場合，ジョブを打ち切ります。

・終了コード 1:終了コード 2

終了コード 1 以上，終了コード 2 以下の場合，ジョブを打ち切ります。

・終了コード:

終了コード以上の場合，ジョブを打ち切ります。

・:終了コード

終了コード以下の場合，ジョブを打ち切ります。

・:

終了コード定義を無効にし，ジョブが打ち切られない状態にします。終了コードを「,」で区切って複数指定した場合，この形式が含まれている場合は文法エラーになります。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- このコマンドをジョブステップ内に記述した場合，構文解析エラーになります。
- ジョブを打ち切った場合，後続ジョブステップの run 属性値に関係なく，後続ジョブ定義スクリプトは一切実行しません。

9.5.5　#-adsh_path_var コマンド（パス名を扱うシェル変数を定義する）

形式

```
#-adsh_path_var シェル変数名 [, ... シェル変数名]
```

機能

パス名を扱うシェル変数を定義します。環境ファイルに PATH_CONV_ENABLE パラメーターが定義されている場合に有効となります。

環境ファイルの PATH_CONV_VAR パラメーターと PATH_CONV_NOVAR パラメーターの指定より優先します。

#-adsh_path_var コマンドは，次のどれかの場合だけ使用できます。

- 1 行目の「#!任意文字列」の次の行
- #-adsh_job コマンドの次の行
- 1 行目（継続行は指定できる）

このコマンドで指定したシェル変数をパス名の先頭に記述すると，そのパス名中の PATH_CONV_ENABLE パラメーターで定義したパス区切り文字とディレクトリ区切り文字が，実行先の OS のパス区切り文字とディレクトリ区切り文字に変換されます。

次の形式で指定したシェル変数と前方一致する記述を，パス名とみなして変換します。

- $シェル変数名
- ${シェル変数名}

変数名は完全一致で判定されるため，**シェル変数名**の後ろに英数字または「_（アンダーバー）」が付く場合は，対象のシェル変数ではないと判定されて変換されません。

変換後，変換対象の文字列に PATH_CONV_ENABLE パラメーターで定義されたパス区切り文字またはディレクトリ区切り文字が含まれる場合，その区切り文字はジョブ定義スクリプトの実行先の OS に合わせて変換されます。

PATH_CONV_RULE パラメーターでパス変換ルール 1 とパス変換ルール 2 のどちらを選択したかでパスの変換結果が異なります。パス変換ルールについては，「PATH_CONV_RULE パラメーター（パス変換ルールを定義する）【Windows 限定】」を参照してください。ジョブ定義スクリプトの変換例は，「2.6.2 パス名を変換する」を参照してください。

引数

シェル変数名

～<環境変数名>((1～255 バイト))

パス名を扱うシェル変数として定義するシェル変数の名称を指定します。シェル変数は 255 個まで指定できます。定義したシェル変数をジョブ定義スクリプト中で使用する場合，$**シェル変数名**または${**シェル変数名**}と記載します。

すでに設定されているシェル変数や，使用できないシェル変数名については，「5.5　シェル変数」を参照してください。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- このコマンドによる変換は行ごとに実施されます。このため，ジョブ定義スクリプト内で行が継続している場合，正しく変換されないことがあります。

  例として，ジョブ定義スクリプトに「$DIR1**改行コード**¥¥bar1¥¥…」と記述し，パス変換ルール 1 を指定して Linux 上で変換した場合を次に示します。

```
#-adsh_path_var DIR1,DIR2
echo foo > ”$DIR1            ←””の途中で改行されているため変換されない。
¥¥bar1¥¥”$DIR2¥¥bar2”bar3”  ←「$DIR2¥¥bar2」が「$DIR2/bar2」に変換される。
```

  同様に，ジョブ定義スクリプトに「$DIR1**改行コード**/bar1/…」と記述し，パス変換ルール 2 を指定して Windows 上で変換した場合の例を次に示します。

```
#-adsh_path_var DIR1,DIR2
echo foo > ’$DIR1
/bar1/”$DIR2/bar2”bar3’      ←「$DIR2/bar2」が「$DIR2¥¥bar2」に変換される。
```

- コメント内の文字列も変換されます。
- SPOOLJOB_CHILDJOB パラメーターに DELETE を指定した場合，子孫ジョブとして実行するジョブ定義スクリプトはスクリプトイメージが出力されません。そのため，子孫ジョブで実行したジョブ定義スクリプトに対して，このコマンドで定義した変換規則に基づいて変換した場合，変換結果は出力されないので注意してください。
- パス変換ルール 2 の場合でも，「"（ダブルクォーテーション）」で囲んだ範囲内に「'（シングルクォーテーション）」を入れ子として指定することはできません。指定するとパス変換の対象となるので注意してください。
- #-adsh_path_var コマンドで指定した変数は，別プロセスではパスを扱う変数として定義されません。別プロセスでもパスを扱う変数として定義したい場合は，PATH_CONV_VAR パラメーターと PATH_CONV_NOVAR パラメーターで指定してください。【Windows 限定】

9.5.6　#-adsh_rc_ignore コマンド（常に正常終了するコマンドを定義する）

形式

```
#-adsh_rc_ignore コマンド名 [,コマンド名 ...]
```

機能

このコマンドに引数として定義したコマンドは，終了コードに関係なく常に正常終了します。その場合，対象コマンドの終了コードはジョブステップの成功または失敗の判定に影響しません。常に正常終了するコマンドの定義は，1,023 個まで指定できます。

ただし，コマンドがシグナルを受信して終了した場合は，指定に関係なくコマンドがエラー終了します。定義方法については，「(2)　常に正常終了するコマンドを定義する」を参照してください。

このコマンドを指定した個所以降のジョブ定義スクリプト実行で有効となります。ジョブステップ外に指定した場合はジョブ定義スクリプト全体に有効で，ジョブステップ内に指定した場合はジョブステップ内だけで有効です。ジョブステップ内に指定した場合，指定した個所以降からジョブステップ終了まで有効となり，ジョブステップ外に指定した値は一時的に無効になります。また，ジョブステップ内に指定するまではジョブステップ外に指定した値が有効になります。

引数

コマンド名［,コマンド名 ...］

常に正常終了するコマンドを定義します。

コマンド名を「,」で区切って複数指定した場合，指定したすべてのコマンドに対して有効になります。コマンド名は，255 個まで指定できます。

- **コマンド名**

Windows の場合　～<パス名>((1～247 バイト))
UNIX の場合　～<パス名>((1～256 バイト))
コマンド名をベース名で指定します。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- コマンド名をベース名で指定します。ジョブステップ内で実行するコマンド名が重複する場合は，エイリアスやリンクを利用してベース名が重複しないようにしてください。
- このコマンドは，ジョブ定義スクリプトファイル中に 1,023 個まで指定できます。
- このコマンドは，スクリプト拡張コマンドに対して指定できません。スクリプト拡張コマンドの終了コードは，必ず 0 が正常終了で 1 がエラー終了であり，エラー終了の場合はジョブを続行できないためです。
- このコマンドをジョブステップエラーブロックに記載できません。
- KNAX6584-I メッセージを出力してバッチジョブを中断する場合，最後に実行したコマンドに対してはこのコマンドの指定は有効になりません。
- この#-adsh_rc_ignore コマンドを使って「5.1.7　別プロセスでの実行」に示す書式で実行されたコマンドを常に正常終了させる場合，文字列を置換する前のコマンド名に対するベース名をコマンドの引数に指定してください。
- CMDRC_CMDGRP_CHECK パラメータに FUNCTION を指定した場合，引数のコマンド名に関数名を指定することで，関数を常に正常終了にすることができます。CMDRC_CMDGRP_CHECK パラメータを指定しない場合，または CMDRC_CMDGRP_CHECK パラメータに NONE を指定した場合，引数のコマンド名に関数名を指定しても，同名のコマンド名が指定されたとして処理されます。

使用例

- grep の終了コードを無視します。

```
#-adsh_step_start STEP1
 #-adsh_rc_ignore grep
  UAP data|grep "TOTAL:"
#-adsh_step_end
```

9.5.7 #-adsh_script コマンド（実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルを呼び出す）

形式

```
#-adsh_script ジョブ定義スクリプトファイル名
```

機能

外部のジョブ定義スクリプトファイルのジョブコントローラ起動時点での内容を，現在実行中のジョブ定義スクリプトファイルに挿入します。実行中のジョブ定義スクリプトから外部のジョブ定義スクリプトファイルの呼び出しは，4,095 個まで指定できます。呼び出した外部スクリプトは呼び出し元のジョブ定義スクリプトに展開され，全体を 1 個のジョブ定義スクリプトとして解析し，実行します。

引数

ジョブ定義スクリプトファイル名

Windows の場合　～<*パス名*>((1～247 バイト))

UNIX の場合　～<*パス名*>((1～4,096 バイト))

展開するジョブ定義スクリプトファイルのパスを指定します。相対パスで指定した場合は，ジョブコントローラ起動時のカレントディレクトリからの相対パスとなります。

終了コード

終了コード	意味
呼び出した外部スクリプト内で最後に終了したコマンドの終了コード	正常終了
1	エラー終了

注

#-adsh_script コマンドの正常終了の終了コードは，環境設定パラメーターで変更できません。

#-adsh_script コマンドが正常終了となるかエラー終了となるかは，外部のジョブ定義スクリプトファイルが正しく挿入されたかどうかで決定します。挿入した外部のジョブ定義スクリプトファイル内で実行したコマンドが正常終了となったかエラー終了となったかの影響は受けません。

注意事項

- シェル標準コマンドの.（ドット）コマンドと次の個所が異なります。
  - .（ドット）コマンドは，ジョブ定義スクリプト実行処理時点での外部スクリプトの内容が実行されます。#-adsh_script コマンドは，ジョブ定義スクリプトの解析処理時点での外部スクリプトの内容が実行されます。ジョブ定義スクリプトの解析処理から実行処理の間に外部スクリプトの内容を変更しないでください。

- .（ドット）コマンドは外部スクリプト内にスクリプト拡張コマンドが記述されている場合，コメントと見なします。#-adsh_script コマンドは，外部スクリプト内にスクリプト拡張コマンドを記述し，実行できます。
- #-adsh_script コマンドで実行した外部スクリプトから，さらに#-adsh_script コマンドを実行する場合，同一の外部スクリプトを 2 回以上呼び出さないでください。
- .（ドット）コマンドで呼び出す外部スクリプトの内容は，スクリプトイメージに出力されません。#-adsh_script コマンドで呼び出す外部スクリプトの内容は，スクリプトイメージに出力されます。
- .（ドット）コマンドに相対パスで外部スクリプトを指定した場合，環境変数PATH の値を参照してパスを解決します。#-adsh_script コマンドに相対パスで外部スクリプトを指定した場合，環境変数PATH の値を参照しないで，adshexec 起動時のカレントディレクトリからの相対パスとして解釈します。
- .（ドット）コマンドはジョブ内で使用できる数に制限がありません。#-adsh_script コマンドは，ジョブ内で使用できる上限数が 4,095 個です。
- .（ドット）コマンドは外部スクリプトへの引数を指定できます。#-adsh_script コマンドは，外部スクリプトへの引数を指定できません。

- ファイル名として.（ドット）から始まるファイル名を指定しないでください。
- ファイル名に予約デバイス名（CON やAUX，NUL など）は使用しないでください。【Windows 限定】
- ファイル名に NTFS のストリームは使用しないでください。【Windows 限定】
- このコマンドが正常終了した場合，ジョブやジョブステップの正常終了またはエラー終了の判定には使用されません。呼び出した外部スクリプトの実行結果を参照してください。

9.5.8 #-adsh_spoolfile コマンド（プログラム出力データファイルの割り当てをする）

形式

```
#-adsh_spoolfile ファイル環境変数定義名
```

機能

プログラム出力データファイルを割り当てます。

出力ファイルの割り当ては，4,095 個まで指定できます。また，1 つのジョブステップ内およびジョブステップ外には 255 個まで指定できます。割り当て方法については，「5.9.3　プログラム出力データファイルの割り当てをする」を参照してください。

引数

ファイル環境変数定義名

～<環境変数名>((1～31 バイト))

割り当てたプログラム出力データファイルを識別するキーとなる，ファイル環境変数定義名を指定します。

Windows の場合，環境設定パラメーター VAR_ENV_NAME_LOWERCASE で ENABLE を指定していれば小文字を指定できます。DISABLE を指定していれば小文字は指定できません。

終了コード

終了コード	意味
0	正常終了
1	エラー終了

注意事項

- #-adsh_spoolfile コマンドで割り当てたパス名を利用して，スプールジョブディレクトリ下にディレクトリを生成しないでください。スプールジョブディレクトリ下にディレクトリが生成されていると，adshhk コマンドでスプールジョブの削除に失敗するなどの予期しない障害が発生することがあります。
- スプールジョブ作成抑止機能を使用した場合，#-adsh_spoolfile コマンドは使用できません。使用すると，KNAX6385-E メッセージを出力して終了します。

9.5.9 #-adsh_step_start コマンド，#-adsh_step_error コマンド，#-adsh_step_end コマンド（ジョブステップを定義する）

形式

```
#-adsh_step_start
     [ジョブステップ名]
     [-successRC 終了コード定義 [,終了コード定義 ...] ]
     [-stepVar シェル変数名 [,シェル変数名 ...] ]
     [-run {normal|abnormal|always} ]
     [-onError {cont|stop} ]

... ジョブステップ内の処理...（ジョブステップ正常ブロック）

 [#-adsh_step_error]

 [... ジョブステップエラー時の処理...（ジョブステップエラーブロック）]

#-adsh_step_end
```

機能

ジョブ定義スクリプトの一部分を，ジョブステップとしてグループ化します。ジョブステップとは，グループ化した一まとまりのコマンド群のことです。ジョブステップの定義は，それぞれ 4,095 個まで指定できます。

ジョブステップの使用方法については，「5.8.3　ジョブステップを定義する」を参照してください。

ジョブステップ内で実行するコマンドの正常・エラーの判定については，「5.8.8　ジョブ，ジョブステップおよびコマンドの終了コード」を参照してください。

ジョブステップ内でエラーが発生した場合の動作については，「5.8.10　ジョブ実行中にエラーが発生した場合の動作」を参照してください。

引数

ジョブステップ名

～<*環境変数名*>((1～31 バイト))

ジョブステップを識別する情報の 1 つであるジョブステップ名を定義します。ジョブステップ名はジョブ実行ログなどにメッセージとして表示されるほか，JP1/Advanced Shell が作成するファイル名の一部にも使用されます。

ジョブステップ名は，ジョブ内で重複できます。

-successRC *終了コード定義* [,*終了コード定義*] …

ジョブステップ正常ブロック内で実行するコマンドが正常終了したと見なす，コマンドの終了コードの値を定義します。定義を「,」で区切って複数指定した場合，どれかの定義を満たした場合に正常終了と見なします。

ただし，ステップ正常ブロック内で実行するコマンドがシグナル終了した場合は，この指定に関係なく，コマンドはエラー終了となります。

ステップ正常ブロック内で実行するコマンドが，successRC 属性で定義したコマンドの終了コードに該当しない終了コードを返している場合でも，#-adsh_rc_ignore コマンドで指定したコマンド名に該当すれば，successRC 属性の指定値に関係なく#-adsh_rc_ignore コマンドの指定が優先されます。

終了コード定義

～<*符号なし整数*>((0～255))

終了コード定義は 8 個まで指定できます。

・*終了コード*

*終了コード*に合致する場合，正常終了します。

・*終了コード 1*:*終了コード 2*

終了コード 1 以上，*終了コード 2* 以下の場合，正常終了します。

・*終了コード*:

*終了コード*以上の場合，正常終了します。

・:*終了コード*

*終了コード*以下の場合，正常終了します。

-stepVar *シェル変数名* [,*シェル変数名* …]

ジョブステップ内だけで有効なシェル変数を宣言します。シェル変数名は，コンマで区切って 32 個まで指定できます。

- *シェル変数名*

  ～<*環境変数名*>((1～255 バイト))

  ジョブステップ内だけで有効なシェル変数の名称を指定します。ただし，関数情報配列の名称，およびシェル変数ADSH_RC_EXTERNAL は指定できません。

-run {normal|abnormal|always}

先行ジョブステップや先行ジョブ定義スクリプト中のコマンドの状態によって，そのジョブステップを実行するかどうかを定義します。指定が省略されている場合，normal が指定されたものとします。

- normal

  先行ジョブステップの中にエラー終了したジョブステップが存在しない場合，または先行ジョブ定義スクリプト中にエラー終了したコマンドが存在しない場合，実行します。

- abnormal

  先行ジョブステップの中にエラー終了したジョブステップが存在する場合，または先行ジョブ定義スクリプト中にエラー終了したコマンドが存在する場合，実行します。

- always

  先行ジョブステップや先行ジョブ定義スクリプト中のコマンドの結果に関係なく，常に実行します。

-onError {cont|stop}

ジョブステップ正常ブロック内のコマンドがエラー終了したとき，ジョブステップ正常ブロック内の後続ジョブ定義スクリプトを実行しないでジョブステップエラーブロックへジャンプするか，ジョブステップ正常ブロック内の後続ジョブ定義スクリプトを実行するかを定義します。指定が省略されている場合，stop が指定されたものとします。

- cont

  ジョブステップ正常ブロック内の後続ジョブ定義スクリプトを実行します。

- stop

  ジョブステップ正常ブロック内の後続ジョブ定義スクリプトを実行しないで，ジョブステップエラーブロック内のジョブ定義スクリプトを実行します。

終了コード

#-adsh_step_start，#-adsh_step_error の場合

終了コード	意味
0	正常終了
1	エラー終了

#-adsh_step_end の場合

終了コード	意味
ジョブステップ正常ブロック内で最後に終了したコマンドの終了コード	ジョブステップ正常終了
	ジョブステップエラー終了
exit コマンドの引数	ジョブステップエラーブロック内で引数を指定したexit コマンドを実行して終了
1	#-adsh_step_end 自身のエラー終了

注

#-adsh_step_end コマンドのジョブステップ正常終了およびジョブステップエラー終了の終了コードは，環境設定パラメーターで変更できません。

注意事項

- ジョブステップを制御文（if，for，while，until，case）のブロック内に記述する場合は，#-adsh_step_start から#-adsh_step_end までを同一ブロック内に記述してください。記述しなかった場合は，実行前に文法エラーとなります。
- for 文，while 文，until 文のブロック内に，ジョブステップを定義しないでください。これらのブロック内に外部スクリプト展開がある場合，外部スクリプトにジョブステップを含むこともできません。含んだ場合，実行前に文法エラーとなります。
- if 文，case 文のブロック内にはジョブステップを定義できます。ただし，run 属性にabnormal または always を指定できません。
- ジョブステップ内にジョブステップを定義できません。
- #-adsh_rc_ignore コマンドなどを使用して，ジョブステップ正常ブロックで最後に実行したコマンドが0 以外で正常終了した場合，ジョブステップが正常終了してもジョブステップの終了コードが0 以外となることがあります。

  ジョブ定義スクリプト

```
#-adsh_rc_ignore cmdA
#-adsh_step_start S1 -onError cont
  cmdA    #rc=4となるコマンド
  cmdA    #rc=4となるコマンド
#-adsh_step_end
```

  実行ログ

```
  KNAX6117-I コマンド（/home/hitachi/bin/cmdA，行番号=3）が終了しました。rc=4 E-
Time=0.001s C-Time=0.000s
  KNAX6117-I コマンド（/home/hitachi/bin/cmdA，行番号=4）が終了しました。rc=4 E-
Time=0.001s C-Time=0.000s
  KNAX6597-I ADSH152256.S1 ジョブステップが正常終了しました。rc=4 E-Time=0.004s C-
Time=0.000s
```

- KNAX6584-I メッセージを出力してバッチジョブを中断する場合，最後に実行したコマンドに対しては，successRC 属性の指定は有効になりません。

- ジョブステップ正常ブロック内およびジョブステップエラーブロック内に関数を定義した場合，ジョブステップがrun属性によってスキップされても，定義した関数を使用できます。
- CMDRC_CMDGRP_CHECKパラメータの指定によって，関数と関数内のコマンドのエラー判定およびsuccessRC属性の定義は次のようになります。

CMDRC_CMDGRP_CHECKパラメータの指定	対象	定義可否	詳細
FUNCTION	関数	○	関数の終了コードとsuccessRC属性の定義に従い，ジョブステップのエラー判定をします。
	関数内のコマンド	×	関数内のコマンドの終了コードに従いジョブステップのエラー判定はされないで，常に正常終了したと見なされるため，ジョブステップのエラー判定とsuccessRC属性の定義の対象外となります。
NONEまたはパラメータの指定なし	関数	×	関数の終了コードに従いジョブステップのエラー判定はされないため，ジョブステップのエラー判定とsuccessRC属性の定義の対象外となります。
	関数内のコマンド	○	関数内のコマンドの終了コードとsuccessRC属性の定義に従い，ジョブステップのエラー判定をします。

（凡例）

○：対象が正常終了したと見なす終了コードの値をsuccessRC属性によって定義できる。

×：対象が正常終了したと見なす終了コードの値をsuccessRC属性によって定義できない。

使用例

- if制御文のブロック内に#-adsh_step_startを指定し，対応する#-adsh_step_endをブロックの外に指定するとエラーになります。

```
if [[ $a = $b ]]; then
  #-adsh_step_start S1
fi
  #-adsh_step_end
```

- while制御文のブロック内に，ジョブステップを定義するとエラーになります。

```
while [[ $a = $b ]] do
  #-adsh_step_start S1
  #-adsh_step_end
done
```

- if制御文のブロック内には，ジョブステップを定義できます。

```
if [[ $a = $b ]]; then
  #-adsh_step_start S1
```

```
  #-adsh_step_end
fi
```

9.6　スクリプト制御文

スクリプト制御文とは，ジョブ定義スクリプトに記述する制御文のことです。

ジョブ定義スクリプトは，制御文に記述された条件式の結果を基に，実行する処理を制御します。制御文を構成する予約語，処理の前には 0 個以上のスペースおよびタブ文字を挿入できます。

9.6.1　case 文（複数処理からの選択）

形式

```
case 式 in
        パターン1)処理a
                ;;
        パターン2)処理b
                ;;
        ...
        *)処理x
                ;;
esac
```

機能

文字列の内容に応じて複数ある処理のうち，1 つを実行する制御文です。

説明

in は case 文の処理の開始を意味し，esac は case 文の終了を意味します。一致するパターンが存在した場合，「)」から「;;」までに記述されている処理を実行します。1 つのパターンは「;;」で区切られ，パターンは複数記述できます。また，*パターンにはどのパターンにも一致しなかった場合の処理を記述します。パターンと一致しているかどうかの判定は，記述された順に行います。式の内容が複数のパターンに一致する場合は，最初に一致したパターンに記述された処理を実行します。

in を「{」，esac を「}」で記述できます。しかし，in の場合は esac を，「{」の場合は「}」を省略できません。それぞれの対応が合わない場合，構文不正でエラー終了します。

パターンには，ワイルドカードによる正規表現の指定ができます。

使用例

- パターンの終端を示す「;;」は，処理と同一行に記述できます。

```
case $cnt in
    0)
        echo "cnt is ZERO" ;;
    *)
```

```
        echo "cnt is not ZERO" ;;
esac
```

- パターン内の最後のコマンドがスクリプト拡張コマンドの場合，「;;」はスクリプト拡張コマンドの引数と解釈されるため，改行して記述します。

```
case $cnt in
    0)
        #-adsh_step_start STEP01
        echo "cnt is ZERO"
        #-adsh_step_end ;;    ←誤り。「;;」の前で改行する。
    *)
        #-adsh_step_start STEP01
        echo "cnt is not ZERO"
        #-adsh_step_end
        ;;
esac
```

9.6.2　for 文（繰り返し実行）

形式 1

```
for 変数 [in wordlists]
do
        処理
done
```

形式 2

```
for 変数 [in wordlists] ;do
        処理
done
```

機能

値を順次変化させながら，同じ処理を繰り返し実行する制御文です。

説明

先頭に for 文があり，do と done で終わります。ループの回数は wordlists の要素数で決定します。変数には wordlists の各要素が左から順に代入され，do から done の間に記述された処理を実行します。wordlists の各要素をすべて代入し終えると，for 文は終了します。

wordlists の各要素は，「要素 1 要素 2 ...要素 n」のようにスペースで指定します。

wordlists に変数を指定した場合，指定した変数の値を do から done の間に変更しても for 文の変数に代入される値は変更されません。

wordlists に「$@」と指定された場合，ジョブ定義スクリプトの引数を wordlists として使用します。また，in wordlists は省略できますが，in wordlists を省略した場合は wordlists に「$@」が指定された場合と同じ処理をします。

do を「{」，done を「}」で記述できます。しかし do の場合は done を，「{」の場合は「}」を省略できません。それぞれの対応が合わない場合，構文不正でエラー終了します。

wordlists の直後に「;」を付けた場合，継続して記述できます。

使用例

- 値を変えて表示を 3 回繰り返します。

```
for num in 1 2 3
do
    echo "num is $num"
done
```

9.6.3　if 文（条件分岐）

形式 1

```
if 条件1
then
        処理a
 [elif 条件2
then
        処理b]
 [else
        処理c]
fi
```

形式 2

```
if 条件1;then
        処理a
 [elif 条件2; then
        処理b]
 [else
        処理c]
fi
```

機能

ある条件を指定し，その結果が真(0)か偽（0 以外）のどちらかによって処理を分岐します。

説明

if 文で開始し，fi 文で終了します。条件には任意のコマンドまたは&&，||，( )，{ }などを使用し，複数のコマンドをまとめて指定するコマンドリストを記述します。コマンドまたはコマンドリストの終了コードが 0 の場合は then 節に進み，0 以外の場合は else 節または elif 節に進みます。

elif 節および else 節は省略できますが，then および fi は必ず指定してください。elif 節は複数指定できます。if に対応する then および fi が見つからない場合，構文不正でエラー終了します。

条件の直後に「;」を付けた場合，継続して記述できます。

使用例

- 値を 3 と比較して結果を表示します。

```
if [[ $num -eq 3 ]]
then
    echo "num = 3"
elif [[ $num -lt 3 ]]
then
    echo "num < 3"
else
    echo "num > 3"
fi
```

9.6.4　until 文（条件が成立するまでの繰り返し）

形式 1

```
until 条件
do
        処理
done
```

形式 2

```
until 条件;do
        処理
done
```

機能

条件が成立するまで，同じ処理を繰り返し実行する制御文です。

説明

先頭に until 文があり，do と done で終わります。条件には任意のコマンドまたは&&，||，( )，{ }などを使用し，複数のコマンドをまとめて指定するコマンドリストを記述します。条件に記述したコマンドやコマンドリストの実行による終了コードが 0 になるまで，do から done の間に記述された処理を繰り返し実行します。そのため，until 文から抜けるには，do から done の間の処理で条件が成立するよう状態を変化させる必要があります。また，until 文の先頭時点で条件が成立していた場合，処理は一度も実行されることなく終了します。

do および done は省略できません。do と done の対応が合わない場合，構文不正でエラー終了します。

条件の直後に「;」を付けた場合，継続して記述できます。

使用例

- 0 から 10 になるまで表示を繰り返します。

```
num=0
until [[ $num -eq 10 ]]
do
    echo "num is $num"
    ((num+=1))
done
```

9.6.5　while 文（条件が成立している間の繰り返し）

形式 1

```
while 条件
do
        処理
done
```

形式 2

```
while 条件;do
        処理
done
```

機能

条件が成立している間，同じ処理を繰り返し実行する制御文です。

説明

先頭に while 文があり，do と done で終わります。条件には任意のコマンドまたは&&，||，( )，{ }などを使用し，複数のコマンドをまとめて指定するコマンドリストを記述します。条件に記述したコマンドや

コマンドリストの実行による終了コードが 0 という条件を満たしている間，do から done の間に記述された処理を繰り返し実行します。そのため，while 文から抜けるには，do から done の間の処理で条件が不成立になるよう状態を変化させる必要があります。

do および done は省略できません。do と done の対応が合わない場合，構文不正でエラー終了します。

条件の直後に「;」を付けた場合，継続して記述できます。

使用例

- num の値が 0 から 9 の間，表示を繰り返します。

```
num=0
while [[ $num -ne 10 ]]
do
    echo ”num is $num”
    ((num+=1))
done
```

9.7　スクリプト予約語コマンド

スクリプト予約語コマンドとは，ジョブ定義スクリプトで予約語として使用できるコマンドのことです。

9.7.1　time コマンド（コマンドの実行時間を出力する）

形式

```
time [-p] [command]
```

機能

コマンドの実行時間を標準エラー出力に出力します。

command に指定したコマンドの実行時間を標準エラー出力に出力します。command を指定しない場合は，シェルの実行時間を出力します。

出力形式を次に示します。

- command を指定した場合

  ```
  commandの実行時間　commandのユーザーCPU時間　commandのシステムCPU時間
  ```

  Windows の場合，「commandのユーザーCPU時間」，「commandのシステムCPU時間」には，孫プロセスの CPU 時間は含まれません。

- command を指定しない場合

  ```
  シェル※のユーザーCPU時間　シェル※のシステムCPU時間
  ```

  注※　シェルから起動したプロセスも含みます。

  Windows の場合，「シェルのユーザーCPU時間」，「シェルのシステムCPU時間」には，孫プロセスの CPU 時間は含まれません。

引数

-p

command の実行時間，ユーザー CPU 時間，システム CPU 時間をそれぞれ改行して出力します。

command

実行時間および CPU 時間を出力したいコマンドの名称を指定します。

終了コード

終了コード	意味
command に指定したコマンドの終了コード command を指定しない場合は 0	正常終了

注意事項

- time コマンドの結果を標準エラー出力以外のファイルにリダイレクトできません。
- このコマンドの実行結果はジョブ実行ログファイルに出力されません。また，ジョブやジョブステップの正常終了またはエラー終了の判定にも使用されません。標準エラー出力に出力される実行時間，および呼び出したコマンドの実行結果を参照してください。

使用例

- コマンドの実行時間および CPU 時間を出力します。

  ジョブ定義スクリプトの内容

```
time date
```

  実行ジョブの STDOUT ファイルの内容

```
********   実行ジョブのSTDOUTファイルの内容     ********
2013/12/06 金曜日 13:16:21 JST
```

  実行ジョブの STDERR ファイルの内容

```
********   実行ジョブのSTDERRファイルの内容     ********
    0.00s real     0.00s user     0.00s system
```

10 スクリプト開発部品

この章では，スクリプト開発部品の記述形式と詳細を説明します。

10.1　スクリプト開発部品の記述形式

スクリプト開発部品の記述形式を次に示します。

△$_0$部品名［△$_1$オプション］…［△$_1$オプション］［△$_1$オペランド］

- 最初にオプションを指定し，次にオペランドを指定します。オペランドとは，オプション名とオプション値のほかにコマンドに指定できる引数のことです。オプションの前にオペランドを指定した場合は，指定内容をすべてオペランドとして処理します。
- オプションは「-オプション名［△1 値]」の形式で指定します。オプションを複数指定する場合，指定順序は任意です。
- 値のないオプションは連続して指定できます（例：「-a -b -c」と「-abc」は同じです）。その場合，最後のオプションには値を指定できます（例：「-abc xyz」の「xyz」は，-c オプションの値となります）。
- 不当なオプション，または指定できる範囲外の値を指定した場合，エラーになります。

オプション名にはマルチバイト文字は使用できません。

10.2　スクリプト開発部品の一覧

スクリプト開発部品の一覧を次の表に示します。

表 10-1　スクリプト開発部品の一覧

分類	部品名	機能概要
変数操作	getArrayIndex	配列の値をキーにした添え字取得
	isEmptyVar	変数の空文字判定
	isInitVar	変数の初期化判定
	sortArray	配列のデータのソート
文字列操作	deleteSpace	空白を削除した文字列の取得
	getStrLen	文字列の文字数取得
	getStrPos	文字列の位置取得
	isLowerStr	文字列の半角英小文字の判定
	isUpperStr	文字列の半角英大文字の判定
数値操作	isNumericStr	数値判定
日付操作	cmpDate	日付の比較
	getCalcDate	加減算した日付の取得
	getDate	現在の日付取得
	getDateDiff	日付の経過日数の取得
	getDay	日付から日の取得
	getHour	時刻から時の取得
	getMinute	時刻から分の取得
	getMonth	日付から月の取得
	getSecond	時刻から秒の取得
	getTime	現在の時刻取得
	getWeekday	日付から曜日の取得
	getYear	日付から年の取得
	isLeapYear	うるう年の判定
ファイル・ディレクトリ操作	getFileMTime	ファイルの日付と時刻取得
	getFileSize	ファイルのサイズ取得
	isDir	ディレクトリの存在有無判定
	isEmptyDir	ディレクトリの内容有無判定

分類	部品名	機能概要
ファイル・ディレクトリ操作	isFileOrDir	ファイル・ディレクトリの存在有無判定
	isNormalFile	通常ファイルの存在有無判定
CSV 操作	arrayToCsv	2 次元配列の値の CSV データ出力
	convCsvSep	CSV データの区切り文字の変換
	csvToArray	CSV データの 2 次元配列への格納
	getCsvColumn	CSV データの空白行を意識したカラム取得
	searchCsvColumn	CSV データの特定の列を対象とした検索によるレコード取得
JSON 操作	getJsonValue	JSON データの要素に対応する値の取得
XML 操作	getXmlAttrValue	XML データの要素の属性値の取得
	getXmlDecl	XML 宣言の取得
	getXmlElem	XML データの要素の内容の取得

10.3　スクリプト開発部品

スクリプト開発部品とは，JP1/Advanced Shell が提供する，関数形式のジョブ定義スクリプトです。空白を削除した文字列の取得や，日付の経過日数の取得，ファイルサイズの取得など，汎用的な処理を関数として呼び出すことができます。

スクリプト開発部品の格納場所を次に示します。

Windows の実行環境の場合

- インストール先フォルダ¥JP1ASE¥parts¥en
- インストール先フォルダ¥JP1ASE¥parts¥ja

Windows の開発環境の場合

- インストール先フォルダ¥JP1ASD¥parts¥en
- インストール先フォルダ¥JP1ASD¥parts¥ja

UNIX の実行環境の場合

- /opt/jp1as/parts/en
- /opt/jp1as/parts/ja

en ディレクトリ配下のスクリプト開発部品には英語のコメントが記述されています。ja ディレクトリ配下のスクリプト開発部品には日本語のコメントが記述されています。両者の違いはコメントだけであり，機能は同じです。

提供するスクリプト開発部品のファイルのエンコーディングを次に示します。

表 10-2　スクリプト開発部品のファイルのエンコーディング

OS	エンコーディング
Linux	UTF-8
AIX	
HP-UX	
Solaris	
Windows	SJIS

スクリプト開発部品を使うための詳細手順は，「2.6.22　スクリプト開発部品を使うための準備」を参照してください。

なお，スクリプト開発部品を改造した場合の動作は保証しません。

10.3.1 getArrayIndex（配列の値をキーにした添え字取得）

形式

```
getArrayIndex ［-d 要素番号］ 配列名　文字列
```

機能

引数に指定された配列から文字列を検索し，最初に完全一致した配列の添え字を返します。配列の 0 番目の要素から順に検索します。

引数

-d *要素番号*

指定した要素番号に対応する 2 次元配列から文字列を検索します。

配列名

添え字を取得する配列名を指定します。

文字列

検索する文字列を指定します。

標準出力への出力

添え字を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 一致する文字列が見つからなかった場合，エラー終了します。
- 引数の*要素番号*には0 から65535 までの数値を指定できますが，指定できる文字列かどうかのチェックはしません。
- 引数の*配列名*に「adsh」から始まる配列名を指定しないでください。指定すると，部品内部で使用している変数の名称と重複し，不当な出力結果となることがあります。
- 引数の*配列名*に指定できる文字列は，変数名として使用できる文字列と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
set -A input Tokyo Yokohama Fukuoka Nagoya
getArrayIndex input Tokyo          #「0」が出力される。
getArrayIndex input Yokohama       #「1」が出力される。

set -D input { Osaka Fukuoka Nagoya } { Tokyo Yokohama Chiba }
getArrayIndex -d 0 input Osaka     #「0」が出力される。
getArrayIndex -d 1 input Chiba     #「2」が出力される。
```

10.3.2 isEmptyVar（変数の空文字判定）

形式

isEmptyVar *変数名*

機能

引数に指定された変数の値が空かどうかを判定します。

次のどれかの場合，1 を出力します。

- 変数が定義されていない
- 変数に値が代入されていない
- 変数に空文字列が代入されている

次の場合，0 を出力します。

- 変数に 1 文字以上の値が代入されている

引数

変数名

判定する変数名を指定します。

配列を指定する場合，要素番号を含めて指定します（例：array[1]）。

標準出力への出力

1 または0。

終了コード

終了コード	意味
0	正常終了

終了コード	意味
1 以上	エラー終了

注意事項

- 引数の*変数名*に「adsh」から始まる変数名を指定しないでください。指定すると，部品内部で使用している変数の名称と重複し，不当な出力結果となることがあります。
- 引数の*変数名*に指定できる文字列は，変数名として使用できる文字列と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
typeset var1
isEmptyVar var1    # 「1」が出力される。
var1=""
isEmptyVar var1    # 「1」が出力される。
var1=100
isEmptyVar var1    # 「0」が出力される。

# 変数var1が空の場合，1でreturnする。
result=$(isEmptyVar var1)
if [[ $result -eq 1 ]]; then
  return 1
fi
```

10.3.3 isInitVar（変数の初期化判定）

形式

isInitVar *変数名*

機能

引数に指定された変数に値が代入されているかどうかを判定します。

次のどちらかの場合，1 を出力します。

- 変数に空文字列が代入されている
- 変数に 1 文字以上の値が代入されている

次のどちらかの場合，0 を出力します。

- 変数が定義されていない
- 変数に値が代入されていない

引数

変数名

判定する変数名を指定します。

配列を指定する場合，要素番号を含めて指定します（例：array[1]）。

標準出力への出力

1 または0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 引数の*変数名*に「adsh」から始まる変数名を指定しないでください。指定すると，部品内部で使用している変数の名称と重複し，不当な出力結果となることがあります。
- 引数の*変数名*に指定できる文字列は，変数名として使用できる文字列と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
typeset var1
isInitVar var1    # 「0」が出力される。
var1=""
isInitVar var1    # 「1」が出力される。
var1=100
isInitVar var1    # 「1」が出力される。

# 変数var1が初期化されていない場合，1でreturnする。
result=$(isInitVar var1)
if [[ $result -eq 0 ]]; then
  return 1
fi
```

10.3.4 sortArray（配列のデータのソート）

形式

sortArray [-d *要素番号*] [-n] [-r] *配列名1* *配列名2*

機能

引数に指定された配列の値をソートします。値に含まれる文字の大文字と小文字は区別されます。

引数

-d *要素番号*

指定した要素番号に対応する2次元配列の値をソートします。2次元配列のすべての要素の値をソートする場合は，要素番号に「@」を指定します。「@」を指定すると，2次元配列の1つ目の要素番号に対応する各配列内で要素の値をソートします。例えば，2×3の2次元配列array の場合，array[0][0]とarray[0][1]とarray[0][2]の間で要素の値をソートし，array[1][0]とarray[1][1]とarray[1][2]の間で要素の値をソートします。

このオプションを指定しない場合，配列を1次元配列と解釈してソートします。

-n

先頭の数値文字列を数値と解釈してソートします。

-r

降順にソートします。

このオプションを指定しない場合，昇順にソートします。

配列名1

ソート対象の配列名を指定します。

配列名2

ソートしたデータを格納する配列名を指定します。

標準出力への出力

なし

終了コード

終了コード	意味
0	正常終了
1以上	エラー終了

注意事項

- ソート対象の配列の値に改行が含まれていると，正しくソートされません。
- ソート対象の配列に値を持たない要素が含まれていると，その要素は削除されてソートされます。
- 引数の要素番号には「@」または0から65535までの数値を指定できますが，指定できる文字列かどうかのチェックはしません。

- 引数の*配列名 1* *配列名 2* に「adsh」から始まる配列名を指定しないでください。指定すると，部品内部で使用している変数の名称と重複し，不当な出力結果となることがあります。
- 引数の*配列名 1* と*配列名 2* に指定できる文字列は，変数名として使用できる文字列と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
set -A input Tokyo Yokohama Fukuoka Nagoya
sortArray input output
echo "${output[@]}"        #  「Fukuoka Nagoya Tokyo Yokohama」が出力される。

set -A input Tokyo Yokohama Fukuoka Nagoya
sortArray -r input output
echo "${output[@]}"        #  「Yokohama Tokyo Nagoya Fukuoka」が出力される。

set -A input -- -3 70 -50 8 100
sortArray -n input output
echo "${output[@]}"        #  「-50 -3 8 70 100」が出力される。

set -D input { Osaka Fukuoka Nagoya } { Tokyo Yokohama Chiba }
sortArray -d 1 input output
echo "${output[0][@]}"     #  「Osaka Fukuoka Nagoya」が出力される。
echo "${output[1][@]}"     #  「Chiba Tokyo Yokohama」が出力される。

set -D input { Osaka Fukuoka Nagoya } { Tokyo Yokohama Chiba }
sortArray -d @ input output
echo "${output[0][@]}"     #  「Fukuoka Nagoya Osaka」が出力される。
echo "${output[1][@]}"     #  「Chiba Tokyo Yokohama」が出力される。
```

10.3.5　deleteSpace（空白を削除した文字列の取得）

形式

```
deleteSpace  [-a] [-l] [-r]  文字列
```

機能

引数に指定された文字列から空白を削除した文字列を出力します。空白とは，正規表現[:space:]で示す文字であり，半角空白やタブなどを示します。

引数

-a

文字列に含まれるすべての空白を削除します。

-l

文字列の先頭の空白だけを削除します。

-r

文字列の末尾の空白だけを削除します。

上記オプションを何も指定しなかった場合は，文字列の先頭と末尾の空白を削除します。これは，-l オプションと-r オプションを同時に指定した場合とおなじです。

-a オプションとほかのオプションを同時に指定した場合，-a オプションを優先します。

文字列

空白を削除する文字列を指定します。

標準出力への出力

空白が削除された文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
deleteSpace ”  ab  cd  ”        # 「ab  cd」が出力される。
deleteSpace -l ”  ab  cd  ”     # 「ab  cd  」が出力される。
deleteSpace -r ”  ab  cd  ”     # 「  ab  cd」が出力される。
deleteSpace -a ”  ab  cd  ”     # 「abcd」が出力される。
```

10.3.6　getStrLen（文字列の文字数取得）

形式

```
getStrLen  文字列
```

機能

引数に指定された文字列の文字数を出力します。文字列に含まれる改行は 1 文字と解釈されます。

引数

文字列

文字数を取得する文字列を指定します。

標準出力への出力

文字数を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
getStrLen abcdefg     #「7」が出力される。
getStrLen あいう      #「3」が出力される。
```

10.3.7　getStrPos（文字列の位置取得）

形式

```
getStrPos  文字列1  文字列2 ［検索開始位置］
```

機能

引数に指定された文字列の中から文字列を検索し，最初に完全一致した文字列の位置（先頭からその位置までの文字数）を出力します。文字列が見つからなかった場合は 0 を出力します。

検索を開始する位置を指定できます。検索開始位置を省略すると，1 を仮定し，文字列の先頭から検索を開始します。

引数

文字列 1

位置を取得する文字列を指定します。

文字列 2

検索する文字列を指定します。

検索開始位置

検索を開始する位置を指定します。

標準出力への出力

文字数を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の検索開始位置には 1 以上の数値を指定できますが，指定できる文字列かどうかのチェックはしません。

使用例

```
getStrPos abcdefg c      #「3」が出力される。
getStrPos abcdefg cd     #「3」が出力される。
getStrPos abcdabcd c 2   #「3」が出力される。(2文字目以降で出現するcの位置)
getStrPos abcdabcd c 4   #「7」が出力される。(4文字目以降で出現するcの位置)
getStrPos abcdabcd cd    #「3」が出力される。
getStrPos ABCDabcd cd    #「7」が出力される。
getStrPos あいう う      #「3」が出力される。
getStrPos あiうeお e     #「4」が出力される。
getStrPos ab¥tcd ¥t      #「3」が出力される。
```

10.3.8　isLowerStr（文字列の半角英小文字の判定）

形式

```
isLowerStr  文字列
```

機能

引数に指定された文字列がすべて半角の英小文字であるかどうかを判定します。

次の場合，1 を出力します。

- すべての文字が半角の英小文字である

次の場合，0 を出力します。

- 半角の英小文字でない文字が含まれている

引数

文字列

判定する文字列を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
isLowerStr abc            #「1」が出力される。
isLowerStr ABC            #「0」が出力される。
isLowerStr aBc            #「0」が出力される。
isLowerStr あいう         #「0」が出力される。

# 変数var1の値に半角の英小文字でない文字が含まれている場合，1でreturnする。
result=$(isLowerStr "$var1")
if [[ $result -eq 0 ]]; then
  return 1
fi
```

10.3.9　isUpperStr（文字列の半角英大文字の判定）

形式

```
isUpperStr  文字列
```

機能

引数に指定された文字列がすべて半角の英大文字であるかどうかを判定します。

次の場合，1 を出力します。

- すべての文字が半角の英大文字である

次の場合，0 を出力します。

- 半角の英大文字でない文字が含まれている

引数

文字列

判定する文字列を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
isUpperStr ABC            #「1」が出力される。
isUpperStr abc            #「0」が出力される。
isUpperStr AbC            #「0」が出力される。
isUpperStr あいう         #「0」が出力される。

# 変数var1の値に半角の英大文字でない文字が含まれている場合，1でreturnする。
result=$(isUpperStr "$var1")
if [[ $result -eq 0 ]]; then
  return 1
fi
```

10.3.10　isNumericStr（数値判定）

形式

```
isNumericStr  文字列
```

機能

引数に指定された文字列が数値として評価できるかどうかを判定します。

数値とは，次の形式です（例：123，-100，001）

[-]＜数字＞＋

次の場合，1 を出力します。

- 数値として評価できる

次の場合，0 を出力します。

- 数値として評価できない

引数

文字列

判定する文字列を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
isNumericStr 123      #「1」が出力される。
isNumericStr -100     #「1」が出力される。
isNumericStr 001      #「1」が出力される。
isNumericStr +5       #「0」が出力される。
isNumericStr abc      #「0」が出力される。

# 変数var1の値が数値として評価できない場合，1でreturnする。
result=$(isNumericStr "$var1")
if [[ $result -eq 0 ]]; then
  return 1
fi
```

10.3.11　cmpDate（日付の比較）

形式

```
cmpDate  日付1  {eq|ne|ge|gt|le|lt}  日付2
```

機能

引数に指定された 2 つの日付を比較します。

次の場合，1 を出力します。

- 日付の比較結果が真である。

次の場合，0 を出力します。

- 日付の比較結果が偽である。

引数

日付 1

日付を示す文字列を指定します。

日付 2

日付を示す文字列を指定します。

日付には，date コマンドにおける絶対日時による日付の形式を指定できます（例：yyyy/mm/dd，yyyymm-dd，yyyymmdd，mm/dd/yyyy など）。

eq

日付 1 が日付 2 と等しいことを判定します。

ne

日付 1 が日付 2 と等しくないことを判定します。

ge

日付 1 が日付 2 以上であることを判定します。

gt

日付 1 が日付 2 より大きいことを判定します。

le

日付 1 が日付 2 以下であることを判定します。

lt

日付 1 が日付 2 より小さいことを判定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の**日付 1** と**日付 2** に指定できる文字列は，date コマンドにおける絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
cmpDate 2016/05/02 eq 2016/05/02     #「1」が出力される。
cmpDate 2016/05/02 ne 2016/05/02     #「0」が出力される。
cmpDate 20160505 ge 20160502         #「1」が出力される。
cmpDate 20160101 gt 20160502         #「0」が出力される。
cmpDate 20160505 le 20160502         #「0」が出力される。
cmpDate 20160101 lt 20160502         #「1」が出力される。

# 変数date1が示す日付と変数date2が示す日付が異なる場合，1でreturnする。
result=$(cmpDate "$date1" ne "$date2")
if [[ $result -eq 1 ]]; then
  return 1
fi
```

10.3.12　getCalcDate（加減算した日付の取得）

形式

```
getCalcDate  日付  [+|-]年数  [+|-]月数  [+|-]日数
```

機能

引数に指定された**日付**に，引数に指定された**年数・月数・日数**を加減算して求めた日付を yyyy/mm/dd の形式で出力します。

引数

日付

日付を示す文字列を指定します。

年数

年数を示す 0 以上の数値を指定します。

月数

月数を示す 0 以上の数値を指定します。

日数

日数を示す 0 以上の数値を指定します。

日付には，date コマンドにおける絶対日時による日付の形式を指定できます（例：yyyy/mm/dd, yyyy-mm-dd, yyyymmdd, mm/dd/yyyy など）。

標準出力への出力

加減算された日付を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 日数と月数を加減算した結果，日付がその月の末日を超えている場合，超えた分を次の月に加算して出力します。

  例えば，引数に「2015/03/31 0 -1 0」を指定すると，末日の「2015/02/28」から超えた 3 日分を次の月に加算し，「2015/03/03」を出力します。
- 引数の日付に指定できる文字列は，date コマンドにおける絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。
- 引数の年数と月数と日数には 0 以上の数値を指定できますが，指定できる文字列かどうかのチェックはしません。

使用例

```
getCalcDate 2016/05/02 +1 -4 18     #「2017/01/20」が出力される。
getCalcDate 2016/05/02 0 -6 0       #「2015/11/02」が出力される。
```

10.3.13　getDate（現在の日付取得）

形式

```
getDate
```

機能

現在の日付を yyyy/mm/dd の形式で出力します。

標準出力への出力

現在の日付を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
getDate       # 「2015/04/23」が出力される。
```

10.3.14　getDateDiff（日付の経過日数の取得）

形式

```
getDateDiff [-m|-y] 日付1 日付2
```

機能

引数に指定された 2 つの日付の差を出力します。

引数

-m

日付の差を月の単位で出力します。

例えば，日付 1 が 2016/05/10 で，日付 2 が 2016/06/10 の場合は 1 を出力し，日付 1 が 2016/05/10 で，日付 2 が 2016/06/09 の場合は 0 を出力します。

-y

日付の差を年の単位で出力します。

例えば，日付 1 が 2015/05/10 で，日付 2 が 2016/05/10 の場合は 1 を出力し，日付 1 が 2015/05/10 で，日付 2 が 2016/4/10 の場合は 0 を出力します。

なお，うるう年の場合は 366 日を 1 年と解釈し，うるう年ではない場合は 365 日を 1 年と解釈します。

上記オプションをどちらも指定しなかった場合は，日付の差を日の単位で出力します。上記オプションをどちらも指定した場合，エラー終了します。

日付 1

日付を示す文字列を指定します。

日付 2

日付を示す文字列を指定します。

日付には，date コマンドにおける絶対日時による日付の形式を指定できます（例：yyyy/mm/dd, yyyy-mm-dd, yyyymmdd, mm/dd/yyyy など）。

標準出力への出力

日付の差を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の日付 1 と日付 2 に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getDateDiff 2016/05/10  2016/06/10          #「31」が出力される。
getDateDiff -m 2016/05/10  2016/08/20       #「3」が出力される。
getDateDiff -y 2016/05/10  2018/06/10       #「2」が出力される。

# 現在の日付と変数date1が示す日付との差を出力する。
getDateDiff $("${ADSH_DIR_CMD}date" +%Y%m%d) "$date1"
```

10.3.15　getDay（日付から日の取得）

形式

```
getDay ［日付］
```

機能

引数に指定された日付の日を 01 から 31 までの 2 桁の文字列（数字）で出力します。引数を省略すると，現在の日付を仮定します。

引数

日付

日付を示す文字列を指定します。

日付には，date コマンドにおける絶対日時による日付の形式を指定できます（例：yyyy/mm/dd, yyyy-mm-dd, yyyymmdd, mm/dd/yyyy など）。

標準出力への出力

日を表す 2 桁の文字列（数字）

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の**日付**に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getDay 2016/05/02      # 「02」が出力される。
```

10.3.16　getHour（時刻から時の取得）

形式

```
getHour ［時刻］
```

機能

引数に指定された**時刻**の時を 00 から 23 までの 2 桁の文字列（数字）で出力します。引数を省略すると，現在の時刻を仮定します。

引数

時刻

時刻を示す文字列を指定します。

時刻には，date コマンドでの絶対日時による時刻の形式を指定できます（例：hh:mm:ss, hhmm など）。

標準出力への出力

時を表す 2 桁の文字列（数字）

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の**時刻**に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getHour 15:20:01      # 「15」が出力される。
```

10.3.17　getMinute（時刻から分の取得）

形式

```
getMinute ［時刻］
```

機能

引数に指定された時刻の分を 00 から 59 までの 2 桁の文字列（数字）で出力します。引数を省略すると，現在の時刻を仮定します。

引数

時刻

時刻を示す文字列を指定します。

時刻には，date コマンドでの絶対日時による時刻の形式を指定できます（例：hh:mm:ss, hhmm など）。

標準出力への出力

分を表す 2 桁の文字列（数字）

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の**時刻**に指定できる文字列は，date コマンドでの絶対日時による時刻の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getMinute 15:20:01      # 「20」が出力される。
```

10.3.18　getMonth（日付から月の取得）

形式

```
getMonth ［日付］
```

機能

引数に指定された日付の月を 01 から 12 までの 2 桁の文字列（数字）で出力します。引数を省略すると，現在の日付を仮定します。

引数

日付

　日付を示す文字列を指定します。

日付には，date コマンドにおける絶対日時による日付の形式を指定できます（例：yyyy/mm/dd, yyyy-mm-dd, yyyymmdd, mm/dd/yyyy など）。

標準出力への出力

月を表す 2 桁の文字列（数字）

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の日付に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getMonth 2016/05/02     # 「05」が出力される。
```

10.3.19　getSecond（時刻から秒の取得）

形式

```
getSecond ［時刻］
```

機能

引数に指定された時刻の秒を 2 桁の文字列（数字）で出力します。数字の範囲は，うるう秒への対応の違いから，OS によって異なります。引数を省略すると，現在の時刻を仮定します。

引数

時刻

　時刻を示す文字列を指定します。

時刻には，date コマンドでの絶対日時による時刻の形式を指定できます（例：hh:mm:ss, hhmm など）。

標準出力への出力

秒を表す 2 桁の文字列（数字）

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の**時刻**に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getSecond 15:20:01      # 「01」が出力される。
```

10.3.20　getTime（現在の時刻取得）

形式

```
getTime
```

機能

現在の時刻を hh:mm:ss の形式で出力します。

標準出力への出力

現在の時刻を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
getTime       # 「09:20:50」が出力される。
```

10.3.21　getWeekday（日付から曜日の取得）

形式

```
getWeekday [-l|-s] [日付]
```

機能

引数に指定された日付の曜日を文字（数字）または文字列で出力します。引数の日付を省略すると，現在の日付を仮定します。

引数

-l

　曜日の正式名を表す文字列を出力します。

-s

　曜日の省略名を表す文字列を出力します。

上記オプションをどちらも指定しなかった場合は，0（日曜）から6（土曜）までの 1 桁の文字（数字）を出力します。上記オプションをどちらも指定した場合，エラー終了します。

各出力内容は部品内のdate コマンドの動作に従います。

日付

日付を示す文字列を指定します。

日付には，date コマンドでの絶対日時による*日付*の形式を指定できます（例：*yyyy/mm/dd*，*yyyy-mm-dd*，*yyyymmdd*，*mm/dd/yyyy* など）。

標準出力への出力

曜日を表す 1 桁の文字（数字），または曜日を表す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の*日付*に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getWeekday 2016/05/02          #「1」が出力される。
getWeekday -l 2016/05/02       #「月曜日」が出力される。
getWeekday -s 2016/05/02       #「月」が出力される。
```

10.3.22　getYear（日付から年の取得）

形式

```
getYear ［日付］
```

機能

引数に指定された日付の年を 4 桁の文字列（数字）で出力します。引数を省略すると，現在の日付を仮定します。

引数

日付

日付を示す文字列を指定します。

日付には，date コマンドにおける絶対日時による日付の形式を指定できます（例：yyyy/mm/dd, yyyy-mm-dd, yyyymmdd, mm/dd/yyyy など）。

標準出力への出力

年を表す 4 桁の文字列（数字）。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の**日付**に指定できる文字列は，date コマンドでの絶対日時による日付の形式と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
getYear 2016/05/02      # 「2016」が出力される。
```

10.3.23　isLeapYear（うるう年の判定）

形式

```
isLeapYear  年
```

機能

引数に指定された年がうるう年かどうかを判定します。

次の場合，1 を出力します。

- うるう年である。

次の場合，0 を出力します。

- うるう年でない。

引数

年

年を示す文字列を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

引数の年には 0 以上の数値を指定できますが，指定できる文字列かどうかのチェックはしません。

使用例

```
isLeapYear 2016              #  「1」が出力される。
isLeapYear 2015              #  「0」が出力される。

# 変数var1の値がうるう年の場合，UAP1を実行する。
result=$(isLeapYear ”$var1”)
if [[ $result -eq 1 ]]; then
  UAP1
fi
```

10.3.24　getFileMTime（ファイル・ディレクトリの日付と時刻取得）

形式

```
getFileMTime  [-d] [-t]  パス名
```

機能

引数に指定されたファイルまたはディレクトリの更新日付および更新時刻を出力します。更新日付は yyyy/mm/dd の形式で出力します。更新時刻は hh:mm:ss の形式で出力します。

引数

-d

更新日付を出力します。

-t

更新時刻を出力します。

上記オプションをどちらも指定しなかった場合，または上記オプションをどちらも指定した場合，更新日付と更新時刻を空白区切りで 1 行にして出力します。

パス名

更新日付・更新時刻を取得するファイルまたはディレクトリのパス名を指定します。

標準出力への出力

ファイルまたはディレクトリの更新日付および更新時刻を示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
getFileMTime log.txt      #  「2015/04/23 09:20:50」が出力される。
getFileMTime -d log.txt   #  「2015/04/23」が出力される。
getFileMTime -t log.txt   #  「09:20:50」が出力される。
```

10.3.25　getFileSize（ファイルのサイズ取得）

形式

```
getFileSize  [-k|-m]  パス名
```

機能

引数に指定されたファイルのサイズを出力します。

引数

-k

サイズをキロバイト単位で出力します（1 キロバイト=1024 バイト）。

-m

サイズをメガバイト単位で出力します（1 メガバイト=1048576 バイト）。

上記オプションをどちらも指定しなかった場合，サイズをバイト単位で出力します。上記オプションをどちらも指定した場合，エラー終了します。

小数点以下の数値は切り上げてサイズを出力します。

パス名

サイズを取得するファイルのパス名を指定します。

標準出力への出力

ファイルのサイズを示す文字列。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

Windows の場合，ディレクトリのサイズは常に 0 を出力します。

UNIX の場合，デバイスファイルのサイズは常に 0 を出力します。

使用例

```
getFileSize log.txt      #「1279571」が出力される。
getFileSize -k log.txt   #「1250」が出力される。
getFileSize -m log.txt   #「2」が出力される。
```

10.3.26　isDir（ディレクトリの存在有無判定）

形式

```
isDir  ディレクトリパス名
```

機能

引数に指定されたディレクトリが存在するかどうかを判定します。なお，ファイルはディレクトリではないものと解釈されます。

次の場合，1 を出力します。

- ディレクトリが存在する。

次の場合，0 を出力します。

- ディレクトリが存在しない。

引数

ディレクトリパス名

判定するディレクトリのパス名を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
ディレクトリtestは存在し，ディレクトリprogは存在しない場合
isDir test          #「1」が出力される。
isDir prog          #「0」が出力される。

# ディレクトリdir1が存在しない場合，dir1を作成する。
result=$(isDir dir1)
if [[ $result -eq 0 ]]; then
  mkdir dir1
fi
```

10.3.27　isEmptyDir（ディレクトリの内容有無判定）

形式

```
isEmptyDir  ディレクトリパス名
```

機能

引数に指定されたディレクトリが空かどうかを判定します。

次の場合，1 を出力します。

- ディレクトリが空である。

次のどちらかの場合，0 を出力します。

- ディレクトリが空でない。
- ディレクトリが存在しない。

引数

ディレクトリパス名

判定するディレクトリのパス名を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
ディレクトリtestの配下にはファイルが存在し，ディレクトリtmpの配下には何も存在しない場合
isEmptyDir test         #「0」が出力される。
isEmptyDir tmp          #「1」が出力される。

# ディレクトリdir1が空の場合，dir1を削除する。
result=$(isEmptyDir dir1)
if [[ $result -eq 1 ]]; then
  rmdir dir1
fi
```

10.3.28　isFileOrDir（ファイル・ディレクトリの存在有無判定）

形式

```
isFileOrDir  パス名
```

機能

引数に指定されたファイルまたはディレクトリが存在するかどうかを判定します。

次の場合，1 を出力します。

- ファイルまたはディレクトリが存在する。

次の場合，0 を出力します。

- ファイルとディレクトリがどちらも存在しない。

引数

パス名

判定するファイルまたはディレクトリのパス名を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
ファイルlog.txtは存在し，ファイルtmp.txtは存在しない場合
isFileOrDir log.txt      # 「1」が出力される。
isFileOrDir tmp.txt      # 「0」が出力される。

ディレクトリtestは存在し，ディレクトリprogは存在しない場合
isFileOrDir test          # 「1」が出力される。
isFileOrDir prog          # 「0」が出力される。

# ファイルまたはディレクトリentry1が存在しない場合，1でreturnする。
result=$(isFileOrDir entry1)
if [[ $result -eq 0 ]]; then
  return 1
fi
```

10.3.29　isNormalFile（通常ファイルの存在有無判定）

形式

```
isNormalFile　ファイルパス名
```

機能

引数に指定された通常ファイルが存在するかどうかを判定します。なお，ディレクトリはファイルではないものと解釈されます。

次の場合，1 を出力します。

- 通常ファイルが存在する。

次の場合，0 を出力します。

- 通常ファイルが存在しない。

引数

ファイルパス名

判定するファイルのパス名を指定します。

標準出力への出力

1 または 0。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
通常ファイルlog.txtは存在し，通常ファイルtmp.txtは存在しない場合
isNormalFile log.txt      #「1」が出力される。
isNormalFile tmp.txt      #「0」が出力される。

ディレクトリtestは存在し，ディレクトリprogは存在しない場合
isNormalFile test          #「0」が出力される。
isNormalFile prog          #「0」が出力される。

# 通常ファイルfile1が存在しない場合，file1を作成する。
result=$(isNormalFile file1)
if [[ $result -eq 0 ]]; then
```

```
  touch file1
fi
```

10.3.30 arrayToCsv（2 次元配列の値の CSV データ出力）

形式

```
arrayToCsv [-i] 配列名
```

機能

引数に指定された 2 次元配列のデータを CSV データとして出力します。CSV データの各フィールドはダブルクォートで囲まれます。2 次元配列のデータと出力内容の対応例を次に示します。

2 次元配列（array）のデータ

```
array[0][0]=name
array[0][1]=value
array[0][2]=id
array[1][0]=Yokohama
array[1][1]=200
array[1][2]=1
array[2][0]=Kawasaki
array[2][1]=100
array[2][2]=2
```

2 次元配列（array）のデータを出力した場合の内容

```
”name”,”value”,”id”
”Yokohama”,”200”,”1”
”Kawasaki”,”100”,”2”
```

なお，2 次元配列のデータにダブルクォートが含まれている場合，ダブルクォートを 1 つ増やして出力します。

例

2 次元配列のデータ	出力されるデータ
a”b”c	”a””b””c”

引数

-i

フィールド値が数値の場合は，出力時にダブルクォートを付与しません。

数値とは，以下の形式である。例：123，-100，001

[-]<*数字*>＋

配列名

データを出力する 2 次元配列の名前を指定します。

標準出力への出力

CSV の形式にした 2 次元配列のデータ。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 引数の*配列名*に「adsh」から始まる配列名を指定しないでください。指定すると，部品内部で使用している変数の名称と重複し，不当な出力結果となることがあります。
- 引数の*配列名*に指定できる文字列は，変数名として使用できる文字列と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
set -D array { name value id } { Yokohama 200 1 } { Kawasaki 100 2 }
arrayToCsv array
# 以下の内容が出力される。
”name”,”value”,”id”
”Yokohama”,”200”,”1”
”Kawasaki”,”100”,”2”

set -D array { name value id } { Yokohama 200 1 } { Kawasaki 100 2 }
arrayToCsv -i array
# 以下の内容が出力される。
”name”,”value”,”id”
”Yokohama”,200,1
”Kawasaki”,100,2
```

10.3.31　convCsvSep（CSV データの区切り文字の変換）

形式

```
convCsvSep　区切り文字　［ファイルパス名］
```

機能

引数に指定された CSV ファイルの区切り文字（カンマ）を別の区切り文字に変換します。

引数

区切り文字

変換先の区切り文字を指定します。

ファイルパス名

CSV ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

入力できる CSV データのサイズは，100KB 以下を対象としています。100KB より大きいサイズのデータを入力すると，ジョブの実行時間が長大化する可能性があります。

標準出力への出力

区切り文字変換後の CSV データ。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

使用例

```
CSVデータ(data.csv)
name,value,id
Yokohama,200,1
Kawasaki,100,2

convCsvSep : data.csv
# 以下の内容が出力される。
name:value:id
Yokohama:200:1
Kawasaki:100:2
```

10.3.32　csvToArray（CSV データの 2 次元配列への格納）

形式

```
csvToArray 配列名 ［ファイルパス名］
```

機能

引数に指定された CSV ファイルのデータを 2 次元配列に格納します。具体的には，コンマで区切られたデータを 2 次元配列の各要素の値として格納します。CSV データと 2 次元配列の要素の対応例を次に示します。

```
CSVデータ(data.csv)
name,value,id
Yokohama,200,1
Kawasaki,100,2

2次元配列(array)の要素とCSVデータ(data.csv)の対応
array[0][0]=name
array[0][1]=value
array[0][2]=id
array[1][0]=Yokohama
array[1][1]=200
array[1][2]=1
array[2][0]=Kawasaki
array[2][1]=100
array[2][2]=2
```

なお，CSV データのフィールドの前後を囲むダブルクォートは，削除されて格納されます。また，フィールド内で 2 つ連続して記述されたダブルクォートは，1 つのダブルクォートと解釈して格納されます。

例

CSV データ	2 次元配列に格納されるデータ
”abc”	abc
”a””b””c”	a”b”c

引数

配列名

データを格納する配列名を指定します。

ファイルパス名

CSV ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

入力できる CSV データのサイズは，100KB 以下を対象としています。100KB より大きいサイズのデータを入力すると，ジョブの実行時間が長大化する可能性があります。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 引数の*配列名*に「adsh」から始まる配列名を指定しないでください。指定すると，部品内部で使用している変数の名称と重複し，不当な出力結果となることがあります。
- 引数の*配列名*に指定できる文字列は，変数名として使用できる文字列と同じですが，指定できる文字列かどうかのチェックはしません。

使用例

```
CSVデータ(data.csv)
name,value,id
Yokohama,200,1
Kawasaki,100,2

# CSVファイルのデータを2次元配列に格納する。
csvToArray array data.csv
echo "${array[1][0]}"        # 「Yokohama」が出力される。
echo "${array[1][1]}"        # 「200」が出力される。

# CSVファイルから1行目を除いたデータを2次元配列に格納する。
"${ADSH_DIR_CMD}awk" '{if(FNR!=1){print $0}}' data.csv | csvToArray array
echo "${array[0][0]}"        # 「Yokohama」が出力される。
echo "${array[0][1]}"        # 「200」が出力される。
```

10.3.33 getCsvColumn（CSV データの空白行を意識したカラム取得）

形式

```
getCsvColumn [-c カラム] [-d] [ファイルパス名]
```

機能

引数に指定された CSV ファイルのカラムを出力します。

引数

-c *カラム*

指定したカラムのデータを出力します。カラムには1 以上の数値を指定します。このオプションを指定しない場合，すべてのカラムのデータを出力します。

-d

空白行を削除して出力します。

ファイルパス名

CSV ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

入力できるCSVデータのサイズは，100KB以下を対象としています。100KBより大きいサイズのデータを入力すると，ジョブの実行時間が長大化する可能性があります。

終了コード

終了コード	意味
0	正常終了
1以上	エラー終了

注意事項

引数のカラムには1以上の数値を指定できますが，指定できる文字列かどうかのチェックはしません。

使用例

```
CSVデータ(data.csv)
name,value,id

Yokohama,200,1
Kawasaki,100,2

# 1カラム目のデータを出力する。
getCsvColumn -c 1 data.csv
# 以下の内容が出力される。
name

Yokohama
Kawasaki

# 1カラム目のデータから空白行を削除したデータを出力する。
getCsvColumn -c 1 -d data.csv
# 以下の内容が出力される。
name
Yokohama
Kawasaki

# 1行目を除いたデータから空白行を削除したデータを出力する。
”${ADSH_DIR_CMD}awk” ’{if(FNR!=1){print $0}}’ data.csv | getCsvColumn -d
# 以下の内容が出力される。
Yokohama,200,1
Kawasaki,100,2
```

10.3.34 searchCsvColumn（CSV データの特定の列を対象とした検索によるレコード取得）

形式

```
searchCsvColumn カラム 文字列 ［ファイルパス名］
```

機能

引数に指定された CSV ファイルからフィールドの値を検索し，一致したフィールドを含むすべてのレコードを出力します。

引数

カラム

指定したカラムのデータを検索の対象とします。一致したフィールドが存在する場合，そのフィールドを含むレコードを出力します。カラムには 1 以上の数値を指定してください。

文字列

検索するフィールドの値を指定します。拡張された正規表現を使用できます。ダブルクォートで囲まれたデータを検索する場合，ダブルクォートを¥でエスケープして指定します。

ファイルパス名

CSV ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

入力できる CSV データのサイズは，100KB 以下を対象としています。100KB より大きいサイズのデータを入力すると，ジョブの実行時間が長大化する可能性があります。

標準出力への出力

CSV データのレコード。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 一致するフィールドが見つからなかった場合，エラー終了します。
- 引数の**カラム**には 1 以上の数値を指定できますが，指定できる文字列かどうかのチェックはしません。

使用例

```
CSVデータ(data.csv)
name,value,id
”Yokohama”,200,100
”Kawasaki”,100,200
”Tokyo”,200,300

searchCsvColumn 2 200 data.csv
# 以下の内容が出力される。
”Yokohama”,200,100
”Tokyo”,200,300

searchCsvColumn 1 ¥”Kawasaki¥” data.csv
# 以下の内容が出力される。
”Kawasaki”,100,200

searchCsvColumn 3 1.* data.csv
#以下の内容が出力される。
”Yokohama”,200,100
```

10.3.35 getJsonValue（JSON データの名前に対応する値の取得）

形式

```
getJsonValue ［-e 文字コード］ 名前 ［ファイルパス名］
```

機能

引数に指定された JSON ファイルから名前を検索し，完全一致したすべての名前に対応する値を出力します。

引数

-e 文字コード

Windows 版

JSON ファイルの文字コードを指定します。指定できる値は「SJIS」または「UTF8」です。このオプションを指定しない場合，文字コードを UTF8 として扱います。

UNIX 版

JSON ファイルの文字コードを環境変数 LANG の値の形式で指定します。指定できる値は，「2.2.4 JP1/Advanced Shell を使用するときのエンコーディング」を参照してください。

このオプションを指定しない場合，JP1/Advanced Shell が動作する環境の環境変数 LANG の値の形式で文字コードを指定したと仮定します。

例えば，AIX で文字コード UTF-8 の JSON ファイルを扱う場合，-e オプションに「JA_JP」または「JA_JP.UTF-8」を指定します。

名前

検索する名前を指定します。

ファイルパス名

JSON ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

標準出力への出力

JSON データの名前に対応する値。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 一致する名前が見つからなかった場合，エラー終了します。
- 同じ名前のデータをネストしている場合，データの最も外側の名前に対応する値を出力します。

  例(test.json)：{"num":{"id":"0001","num":200}}

  上記データに対して num で検索すると，{"id":"0001","num":200}が出力されます。この場合，次のように指定して，データの内側の名前に対応する値(200)を出力できます。

  getJsonValue num test.json | getJsonValue num
- この部品では，adshmktemp コマンドを使用して一時ファイルを作成します。そのため，部品実行中に強制終了要求を受けると，一時ファイルが残ることがあります。この場合は，手動で一時ファイルを削除してください。一時ファイルは，環境ファイルの TEMP_FILE_DIR パラメーターで定義したディレクトリに次の命名規則で作成されます。

  ```
  getJsonValue_ジョブ識別子_プロセスID_時間情報_ファイル通し番号
  ```

使用例

```
JSONデータ(data.json)
{ "city": [
    { "name":"Yokohama", "id":"0001", "value":{ "A":200, "B":100 } },
    { "name":"Kawasaki", "id":"0002", "value":{ "A":100, "B":300 } }
  ]
}

getJsonValue name data.json
# 以下の内容が出力される。
"Yokohama"
"Kawasaki"

getJsonValue -e SJIS value data.json
```

```
# 以下の内容が出力される。
{ "A":200, "B":100 }
{ "A":100, "B":300 }
```

10.3.36 getXmlAttrValue（XML データの要素の属性値の取得）

形式

```
getXmlAttrValue [-e 文字コード] 要素名 属性名 [ファイルパス名]
```

機能

引数に指定された XML ファイルから要素の属性を検索し，完全一致したすべての属性の値を出力します。属性値に含まれる改行は削除して出力されます。

引数

-e 文字コード

Windows 版

XML ファイルの文字コードを指定します。指定できる値は「SJIS」または「UTF8」です。このオプションを指定しない場合，文字コードを UTF8 として扱います。

UNIX 版

XML ファイルの文字コードを環境変数 LANG の値の形式で指定します。指定できる値は，「2.2.4 JP1/Advanced Shell を使用するときのエンコーディング」を参照してください。

このオプションを指定しない場合，JP1/Advanced Shell が動作する環境の環境変数 LANG の値の形式で文字コードを指定したと仮定します。

例えば，AIX で文字コード UTF-8 の XML ファイルを扱う場合，-e オプションに「JA_JP」または「JA_JP.UTF-8」を指定します。

要素名

属性を持つ要素名を指定します。

属性名

検索する属性名を指定します。

ファイルパス名

XML ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

標準出力への出力

XML データの要素の属性の値。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 一致する要素や属性が見つからなかった場合，エラー終了します。
- 要素名や属性名に正規表現のメタキャラクタが含まれている場合，¥でエスケープして指定してください。

  例

  要素名「name+」の属性名「id*」を検索する場合，次のように指定する。

  ```
  getXmlAttrValue ’name¥+’ ’id¥*’ test.xml
  ```

- この部品では，adshmktemp コマンドを使用して一時ファイルを作成します。そのため，部品実行中に強制終了要求を受けると，一時ファイルが残ることがあります。この場合は，手動で一時ファイルを削除してください。一時ファイルは，環境ファイルの TEMP_FILE_DIR パラメーターで定義したディレクトリに次の命名規則で作成されます。

  ```
  getXmlAttrValue_ジョブ識別子_プロセスID_時間情報_ファイル通し番号
  ```

使用例

```
XMLデータ(data.xml)
<data>
  <city>
    <name id=”0001” value=”200”>Yokohama</name>
    <name id=”0002” value=”100”>Kawasaki</name>
  </city>
</data>

getXmlAttrValue name id data.xml
# 以下の内容が出力される。
0001
0002

getXmlAttrValue -e SJIS name value data.xml
# 以下の内容が出力される。
200
100
```

10.3.37　getXmlDecl（XML 宣言の取得）

形式

```
getXmlDecl　項目名　［ファイルパス名］
```

機能

引数に指定された XML ファイルから XML 宣言を検索し，指定された項目の値を出力します。

引数

項目名

取得する XML 宣言の項目名を指定します。

ファイルパス名

XML ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

標準出力への出力

XML 宣言の項目の値。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 一致する項目が見つからなかった場合，エラー終了します。
- 一致した項目の値が空文字列であった場合，エラー終了します。

使用例

```
XMLデータ(data.xml)
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<data>
…
</data>

getXmlDecl encoding data.xml    # 「UTF-8」が出力される。
```

10.3.38 getXmlElem（XML データの要素の内容の取得）

形式

```
getXmlElem ［-e 文字コード］ 要素名 ［ファイルパス名］
```

機能

引数に指定された XML ファイルから要素名を検索し，完全一致したすべての要素の内容を出力します。要素の内容に含まれる改行は削除して出力されます。

引数

-e 文字コード

Windows 版

XML ファイルの文字コードを指定します。指定できる値は「SJIS」または「UTF8」です。このオプションを指定しない場合，文字コードを UTF8 として扱います。

UNIX 版

XML ファイルの文字コードを環境変数 LANG の値の形式で指定します。指定できる値は，「2.2.4 JP1/Advanced Shell を使用するときのエンコーディング」を参照してください。

このオプションを指定しない場合，JP1/Advanced Shell が動作する環境の環境変数 LANG の値の形式で文字コードを指定したと仮定します。

例えば，AIX で文字コード UTF-8 の XML ファイルを扱う場合，-e オプションに「JA_JP」または「JA_JP.UTF-8」を指定します。

要素名

検索する要素名を指定します。

ファイルパス名

XML ファイルのパスを指定します。ファイルパス名を指定しない場合，標準入力から入力します。

標準出力への出力

XML データの要素の内容。

終了コード

終了コード	意味
0	正常終了
1 以上	エラー終了

注意事項

- 一致する要素が見つからなかった場合，エラー終了します。
- 出力する要素の内容には，記述された文字列すべてが含まれます。そのため，CDATA セクションなども出力されます。
- 要素の内容に含まれる「&」などの実体参照の記述はそのまま出力されます。

  要素名に正規表現のメタキャラクタが含まれている場合，¥でエスケープして指定してください。

  例

  要素名「name+」を検索する場合，次のように指定します。

  ```
  getXmlElem 'name¥+' test.xml
  ```

- 同名の要素をネストしている場合，要素の内容を正しく出力できません。

  例

  ```
  <value><value>100</value></value>
  ```

  ただし，ネストされた要素に別の要素を含んでいる場合，次のようにすることで，最も内側の要素の内容を出力できます。

  例（test.xml）

  ```
  <value><name><value>100</value></name></value>
  ```

  ```
  getXmlElem name test.xml | getXmlElem value
  ```

- この部品では，adshmktemp コマンドを使用して一時ファイルを作成します。そのため，部品実行中に強制終了要求を受けると，一時ファイルが残ることがあります。この場合は，手動で一時ファイルを削除してください。一時ファイルは，環境ファイルの TEMP_FILE_DIR パラメーターで定義したディレクトリに次の命名規則で作成されます。

  ```
  getXmlElem_ジョブ識別子_プロセスID_時間情報_ファイル通し番号
  ```

使用例

```
XMLデータ(data.xml)
<data>
  <city>
    <name id="0001">Yokohama</name>
    <name id="0002">Kawasaki</name>
  </city>
</data>

getXmlElem -e SJIS name data.xml
# 以下の内容が出力される。
Yokohama
Kawasaki

XMLデータ(data2.xml)
<name>
```

```
  <city>
    <name id="0001">Yokohama</name>
    <name id="0002">Kawasaki</name>
  </city>
</name>

getXmlElem city data2.xml | getXmlElem name
# 以下の内容が出力される。
Yokohama
Kawasaki
```

11 トラブルシューティング

トラブルシューティングとして，対処の手順，ログ情報の種類，必要な資料，資料の採取方法について説明します。

11.1　対処の手順

JP1/Advanced Shell でジョブ定義スクリプトを実行してエラー終了するなどのトラブルが発生した場合は，トラブルが発生したときの現象を確認します。

メッセージが出力されている場合は，メッセージの内容を確認します。各メッセージの出力要因と対処方法については，「12.　メッセージ」を参照してください。また，メッセージの出力要因に応じて，次のように対処してください。

- ジョブ定義スクリプトの問題の場合

  ジョブ定義スクリプトの問題を指摘するメッセージが出力された場合は，次のことを実施します。

  - 問題の調査・対処

    問題の調査結果を基に開発環境でジョブ定義スクリプトを修正し，デバッガで確認します。

  - 運用の実施

    再び実行環境で運用を実施します。

- システム管理者に連絡する必要がある問題の場合

  システム管理者に連絡する必要があるメッセージが出力された場合は，次のことを実施します。

  - 資料の採取

    トラブルの要因を調べるために資料の採取が必要です。「11.2　トラブル発生時に採取が必要な資料」を参照して，必要な資料を採取してください。

  - 問題の調査

    採取した資料を基に問題の要因を調査し，問題が発生している部分と問題の範囲を切り分けます。

- ユーザー応答機能の応答入力ができない場合

  - 問題の調査・対処

    「11.1.1　ユーザー応答機能使用時の障害対応」を参照して問題を調査し，対処してください。

11.1.1　ユーザー応答機能使用時の障害対応

応答要求メッセージに対しては JP1/IM - View から応答します。ただし，次のケースでは JP1/IM - View から応答を入力できません。

表 11-1　応答待ちイベントに対して応答が入力できないケース

項番	ケース	ユーザーへの通知
1	JP1/IM から応答を入力した際に JP1/Advanced Shell 側でエラーとなり，JP1/IM 側で応答の成功/失敗が不明な場合	電文不正メッセージなど。
2	滞留する応答待ちイベントが 2,000 件を超えて，応答待ちイベントの滞留が解除される場合	JP1/IM - View に KAVB0551-E が表示される。

項番	ケース	ユーザーへの通知
3	通信障害などで JP1/IM が使用できない場合	JP1/IM - View が使用できない状態。

応答要求メッセージは，JP1/Advanced Shell がインストールされているマシン上の共有メモリ上で管理しているため，JP1/Advanced Shell の管理者は，次のコマンドで応答要求メッセージの状況確認や応答ができます。

- 応答待ち状態にある応答要求メッセージの一覧表示

  adshlsmsg コマンドを使用して，運用者からの応答待ち状態にある応答要求メッセージの一覧を表示できます。詳細については，「8.3　シェル運用コマンド」の「adshlsmsg コマンド（障害発生時に，応答要求メッセージの一覧を表示する）」を参照してください。

- 応答待ち状態にある応答要求メッセージに対する手動応答とキャンセル

  adshchmsg コマンドを使用して，運用者からの応答待ち状態にある応答要求メッセージに対して応答を入力またはキャンセルできます。詳細については，「8.3　シェル運用コマンド」の「adshchmsg コマンド（障害発生時に，応答要求メッセージに対して手動で応答する）」を参照してください。

11.1.2　ルートジョブが子孫ジョブより先に終了した場合の注意事項

子孫ジョブからさらに実行する子孫ジョブがある場合，中間のジョブが UNIX の SIGKILL や Windows の TerminateProcess で即時終了すると，ルートジョブがすべての子孫ジョブの完了を待たないで終了することがあります。そのため，このような即時終了操作は実行しないでください。詳細については，「(4) 子孫ジョブからさらに実行する子孫ジョブがある場合の注意事項」を参照してください。

もし，この現象が発生した場合は，関連するルートジョブや子孫ジョブの実行結果を調査してください。なお，即時終了したジョブ以外のすべての子孫ジョブは，スプールジョブディレクトリが削除に失敗して残っているか，削除されていても JOBLOG の内容が標準エラー出力へ出力されているため，ログは失われません。

11.2　トラブル発生時に採取が必要な資料

トラブルが発生したときに採取が必要な資料を次の表に示します。

表 11-2　トラブルが発生したときに採取が必要な資料

種別	内容	採取する資料
ログ	JP1/Advanced Shell が出力するログ	• システム実行ログ • トレースログ • アプリケーション実行エージェント機能ログ【Windows 実行環境限定】
障害情報	システムが採取する障害情報	• dump ファイル【Windows 限定】 • core ファイル【UNIX 限定】
スプール情報	スプールを管理する情報	指定の環境ファイルおよびジョブ ID ファイル。.jobid または adsh.jobid のファイル名が該当します。
環境情報	システムの状態	• 基礎情報 • プロセス情報 • メモリ使用情報 • ファイル情報 • ネットワーク使用状況 • JP1 イベント情報 • エラーログ • アプリケーション実行エージェント機能情報【Windows 実行環境限定】
【UNIX 限定】 ユーザー応答機能管理デーモンの情報	ユーザー応答機能管理デーモンの起動・停止に関する情報	ユーザー応答機能管理デーモンの起動ログおよび pid ファイル

なお，JP1/Advanced Shell の adshcollect コマンドを使うと必要な資料を一括採取できます。adshcollect コマンドの詳細については，「11.3　資料の採取方法」を参照してください。

種別ごとに必要な資料の詳細を次に示します。なお，環境情報の詳細については，製品の内部情報であるため，記載しません。

11.2.1　ログ

採取が必要なログを次の表に示します。

表 11-3　採取が必要なログ

種類	内容	出力先
システム実行ログ	JP1/Advanced Shell の統括的な実行ログ	環境ファイルの LOG_DIR パラメーター※の指定に従って出力されます。
トレースログ	JP1/Advanced Shell の内部トレースログ	環境ファイルの TRACE_DIR パラメーター※の指定に従って出力されます。 カスタムジョブ，エディタおよび共通コマンドのトレースログは，システムの仕様に従って出力されます。
アプリケーション実行エージェント機能ログ【Windows 実行環境限定】	アプリケーション実行エージェント機能ログ	共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥appexec に出力されます。

注※

パラメーターのデフォルト値については，「11.3.1　adshcollect コマンド（資料を採取する）」を参照してください。

11.2.2　障害情報

採取が必要な障害情報を次の表に示します。障害情報は，保守情報を採取するための定義ファイルに DUMP または CORE を指定したときだけ採取されます。

表 11-4　採取が必要な障害情報

種類	内容	出力先
dump ファイル【Windows 限定】	ワトソンログなどが採取する障害情報	ワトソン博士などのデバッグツールを起動している場合，出力されます。ワトソン博士の場合，デフォルトでは次のディレクトリに障害情報が出力されます。 • 共通アプリケーションフォルダ¥Microsoft¥Dr Watson¥drwtsn32.log
core ファイル【UNIX 限定】	システムが採取する障害情報	各プロセスのエラー終了時，システムに設定された core ファイルの出力先ディレクトリに出力されます。設定がない場合は，adshexec 起動時のカレントディレクトリに出力されます。

11.2.3　スプール情報

採取が必要なスプール情報を次の表に示します。

表 11-5　採取が必要なスプール情報

種類	内容	出力先
環境ファイル	JP1/Advanced Shell の定義情報	環境変数 ADSH_ENV に作成した環境ファイルのパス adshcollect コマンドの-e オプションで指定した環境ファイル（カスタムジョブの［実行定義］ダイアログボックスの［ジョブ環境ファイル名］に定義したジョブ環境ファイル，およびエディタの［実行環境の設定］ダイアログボックスのジョブ環境ファイルに定義したジョブ環境ファイル）
スプールディレクトリ下のファイル	スプールに出力したバッチジョブの情報	ジョブ ID ファイルは，.jobid または adsh.jobid が該当します。

11.2.4　ユーザー応答機能管理デーモンの情報【UNIX 限定】

採取が必要なユーザー応答機能管理デーモンの情報を次の表に示します。

表 11-6　採取が必要なユーザー応答機能管理デーモンの情報

種類	内容	出力先
起動ログ	ユーザー応答機能管理デーモンの起動ログ	「/opt/jp1as/system」にユーザー応答機能管理デーモンの起動ログが出力されます。 詳細は「ジョブ実行結果とログの出力情報を定義する」を参照してください。
pid ファイル	ユーザー応答機能管理デーモンの pid ファイル	「/opt/jp1as/system」にユーザー応答機能管理デーモンの pid ファイルが出力されます。 pid ファイルは次のファイル名が該当します。 • 物理ホストのユーザー応答機能管理デーモンの場合 adshmd.pid • 論理ホストのユーザー応答機能管理デーモンの場合 adshmd_**論理ホスト名**.pid

11.3　資料の採取方法

JP1/Advanced Shell がエラー終了，無応答になった場合などに，システム管理者が障害調査を実施するためのコアダンプ（core ファイルまたは dump ファイル），ログなどの資料が必要となります。adshcollect コマンドを使用すると，これらの障害調査のための資料を一括して採取できます。

この節では，adshcollect コマンドの使用方法，保守情報を採取するための定義ファイルの設定，および環境ファイルの設定について説明します。adshcollect コマンドで採取する保守情報（資料）は，Windows と UNIX とで異なる場合があります。

11.3.1　adshcollect コマンド（資料を採取する）

形式

```
adshcollect 保守情報出力先ディレクトリ ［-f 定義ファイル名］
            ［-e 環境ファイル名］ ［-h 論理ホスト名］
```

機能

adshcollect コマンドによって，障害調査のための資料を一括して収集できます。adshcollect コマンドを実行する場合は，Windows のときはコマンドプロンプトから，UNIX のときはシェルから起動してください。

このコマンドは，障害発生時の障害情報を採取するため，実行ユーザーの権限で実行する必要があります。ただし，ユーザー応答機能の情報を採取する場合は，管理者権限で実行する必要があります。

adshcollect コマンドの使用手順を次に示します。

1. 障害が発生したときの環境ファイルを用意してください。

   障害発生後，環境ファイルを変更した場合は，環境ファイルを障害発生時の運用環境に合わせて書き換えてください。障害が発生したときに環境ファイルを使用していなかった場合は，用意は不要です。

   注意事項

   Windows 版の場合，環境ファイルに&（アンパーサンド）が記述されていると，adshcollect コマンドがエラー終了することがあります。

   ジョブ環境ファイルに&（アンパーサンド）が記述されている場合は，ジョブ環境ファイルをコピーして，コピーしたジョブ環境ファイルの&（アンパーサンド）を削除してください。さらに手順 3 で，コピーしたジョブ環境ファイルを-e オプションに指定してください。

   システム環境ファイルに&（アンパーサンド）が記述されている場合は，事前にシステム環境ファイルを別のディレクトリにコピーしてバックアップを作成したあと，&（アンパーサンド）を削除してください。

2. 定義ファイルを用意してください。

core ファイルまたは dump ファイルを採取する場合，任意の場所に定義ファイルを作成してください。core ファイルまたは dump ファイルを採取する必要がない場合は，作成は不要です。

3. adshcollect コマンドを実行します。

次のように引数を指定して adshcollect コマンドを実行してください。

実行時の注意点については「注意事項」を参照してください。

保守情報出力先ディレクトリ

指定したディレクトリに保守情報が作成されるため，次の点に注意してください。

・保守情報の出力先は書き込み可能であり，十分な空き容量があること

・JP1/Advanced Shell で使用しないディレクトリであること

環境ファイル名

手順 1 で用意した環境ファイルのパスを-e オプションまたは環境変数 ADSH_ENV に指定してください。手順 1 で環境ファイルを用意しなかった場合は，指定は不要です。

定義ファイル名

手順 2 で用意した定義ファイルのパスを-f オプションに指定してください。手順 2 で定義ファイルを用意しなかった場合は，指定は不要です。

論理ホスト名

障害が発生したときの環境が論理ホストの場合は，-h オプションに論理ホスト名を指定してください。障害が発生したときの環境が論理ホストではない場合は，指定は不要です。

引数

保守情報出力先ディレクトリ

【Windows 限定】

保守情報を格納したファイルを出力先ディレクトリに出力します。ディレクトリの名称は次の形式となります。

ADSH`yyyymmddhhmmss`

`yyyymmdd`：adshcollect コマンドを起動した日付

`hhmmss`：adshcollect コマンドを起動した 24 時間制のローカルタイムでの時刻

Windows の標準機能には UNIX の tar 相当の機能がないため，保守情報を提供する場合，このファイルをユーザーの圧縮ツールを使用して zip または lzh 形式などの一般的な形式に圧縮してください。

【UNIX 限定】

収集した情報を tar のアーカイブファイルとして出力する場合の，出力先のディレクトリを指定します。また，一時ファイルを必要とする場合は，このディレクトリに作成します。アーカイブファイルの名称は次の形式となります。

ADSH`yyyymmddhhmmss`.tar

`yyyymmdd`：adshcollect コマンドを起動した日付

`hhmmss`：adshcollect コマンドを起動した 24 時間制のローカルタイムでの時刻

保守情報を出力した圧縮ファイルのディスク使用量は次のとおりです。

システム実行ログ，トレースログの容量＋DUMPまたはCOREで指定したファイルの容量※

注※

Windows 環境の場合は DUMP ファイル，UNIX 環境の場合は CORE ファイルになります。

-f 定義ファイル名

採取する保守情報が定義された定義ファイルの名称を指定します。絶対パスまたはカレントディレクトリからの相対パスで指定します。設定内容は項目「定義ファイルと環境ファイルの設定」を参照してください。

定義ファイル名の指定は任意です。定義ファイル名の指定がない場合は，DUMP，CORE 相当の保守情報を採取しません。

-e 環境ファイル名

このオプションは，環境変数 ADSH_ENV に設定したファイルパスと別のファイルパスを指定したい場合に指定します。絶対パスまたはカレントディレクトリからの相対パスで指定します。

- このオプションの指定がない場合

  環境変数 ADSH_ENV に設定したファイルパスを環境ファイル名として扱います。

- このオプションと環境変数 ADSH_ENV の指定がない場合

  システム環境ファイルの内容に従って資料を採取します。

- このオプションの指定，環境変数 ADSH_ENV の指定，およびシステム環境ファイルがない場合

  SPOOL_DIR，LOG_DIR および TRACE_DIR はデフォルト値になります。

-h 論理ホスト名

障害情報を採取する論理ホスト名を指定します。指定された論理ホスト名を基に環境ファイルが解析されます。

「-h」だけ指定して引数の論理ホスト名を指定しなかった場合は，JP1_HOSTNAME 環境変数から論理ホスト名を取得します。そのとき JP1_HOSTNAME 環境変数の指定がなければ，usage を出力してエラー終了します。JP1_HOSTNAME 環境変数については，マニュアル「JP1/Base 運用ガイド」を参照してください。

定義ファイルと環境ファイルの設定

採取する情報を定義ファイルに定義し，採取情報の出力先を環境ファイルに定義します。

- 定義ファイルの定義

  定義ファイルは，「#-adsh_conf△1」に続いてキーワードと値をスペースで区切って記述します。値に指定するファイル名はすべて絶対パスで指定します。

  定義ファイルのキーワードと内容を次の表に示します。どのキーワードも指定は任意ですが，定義ファイルにはこれらのキーワード以外の内容（例えばコメントなど）は指定できません。また，どのキーワードにも値にワイルドカードは指定できません。

表 11-7　定義ファイルのキーワードと指定の関係

キーワード	指定内容	複数指定
DUMP【Windows 限定】	ワトソンログなど，Windows で採取したい dump ファイルを指定します。ワトソンログについては，Windows の資料を参照してください。 パスにスペースがある場合は，ダブルクォーテーションで囲んでください。	○ （16 個まで）
CORE【UNIX 限定】	core ファイルを障害情報として採取する必要があるとき，ファイルを格納しているディレクトリ名を指定します。指定したディレクトリ以下にある，名前の一部に「core」と付いたファイルを一括して採取します。	○

（凡例）

○：指定できます。

- 環境ファイルの定義

  環境ファイルのキーワードと内容を次の表に示します。どのキーワードも指定は任意です。キーワードの指定がない場合は，表の「パス名のデフォルト値」に示す情報を採取します。また，どのキーワードにも値にワイルドカードは指定できません。

表 11-8　環境ファイルのキーワードと指定の関係

キーワード（環境設定パラメーター）	指定内容	パス名のデフォルト値	複数指定
SPOOL_DIR	スプールルートディレクトリのパス名※	• 実行環境の場合【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥spool • 開発環境の場合【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥spool • 実行環境の場合【UNIX 限定】 /var/opt/jp1as/spool	×
LOG_DIR	システム実行ログ出力先ディレクトリのパス名※	• 実行環境の場合【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASE¥log • 開発環境の場合【Windows 限定】 共有ドキュメントフォルダ¥Hitachi¥JP1AS¥JP1ASD¥log • 実行環境の場合【UNIX 限定】 /opt/jp1as/log	×
TRACE_DIR	トレースログ出力先ディレクトリのパス名※	• 実行環境の場合【Windows 限定】 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥trace • 開発環境の場合【Windows 限定】 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥trace • 実行環境の場合【UNIX 限定】 /opt/jp1as/trace	×

（凡例）

×：指定できません。

注※

Windows のパスにスペースがある場合は，ダブルクォーテーションで囲んでください。

定義ファイルと環境ファイルの指定例

- Windows の場合

  定義ファイルの指定例を次に示します。

```
#-adsh_conf DUMP "C:¥Program Files¥Hitachi¥JP1AS¥JP1ASE¥dump"
```

  環境ファイルの指定例を次に示します。

```
#-adsh_conf SPOOL_DIR "C:¥Documents and Settings¥All Users¥Documents¥Hitachi¥JP1AS¥JP1ASE
¥spool"
#-adsh_conf LOG_DIR "C:¥Documents and Settings¥All Users¥Documents¥Hitachi¥JP1AS¥JP1ASE
¥log"
#-adsh_conf TRACE_DIR "C:¥Documents and Settings¥All Users¥Application Data¥Hitachi¥JP1AS
¥JP1ASE¥trace"
```

- UNIX の場合

  定義ファイルの指定例を次に示します。

```
#-adsh_conf  CORE  /home/user1/program1
```

  環境ファイルの指定例を次に示します。

```
#-adsh_conf  SPOOL_DIR  /var/opt/jp1as/spool
#-adsh_conf  LOG_DIR  /opt/jp1as/log
#-adsh_conf  TRACE_DIR  /opt/jp1as/trace
```

adshcollect コマンドで採取するファイルの一覧

adshcollect コマンドで採取するファイルと最大サイズは，次の表に示すように Windows と UNIX で異なります。

表 11-9　adshcollect コマンドで採取するファイルと最大サイズ【Windows 限定】

ファイルの種類	ファイル名	最大サイズ	採取
スプール管理ファイル	[環境ファイルのSPOOL_DIR※] ¥adsh.jobid	1KB 程度	○
システム実行ログ（JP1/Advanced Shell）	[環境ファイルのLOG_DIR※] ¥AdshLog.log [環境ファイルのLOG_DIR※] ¥AdshLog_n.log（n は面数）	[環境ファイルのLOG_FILE_SIZE] ×（n ＋ 1）MB	○
	[環境ファイルのLOG_DIR※] ¥AdshLog.conf	1KB 程度	○

ファイルの種類	ファイル名	最大サイズ	採取
JP1/Advanced Shell 内部処理の実行ログ	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥uxpl¥spool¥uxpllog［n］.txt（n は面数：最大 2 面）	5MB	○
	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥uxpl¥spool¥uxpllog［n］.txt（n は面数：最大 2 面）	5MB	○
	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥misc¥uxpl¥spool¥uxpllog［n］.txt（n は面数：最大 2 面）	5MB	○
トレースログ（JP1/Advanced Shell）	［環境ファイルのTRACE_DIR※］¥AdshTrace_［n］.log（n は面数：4 面固定）	［環境ファイルのTRACE_FILE_SIZE］×nMB	○
トレースログ（カスタムジョブ）	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASV¥trace¥AdshTrace_1.log	1MB	○
トレースログ（エディタ）	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥adshedit¥trace¥AdshTrace_1.log	1MB	○
トレースログ（JP1/Advanced Shell，JP1/Advanced Shell - Developer 共通コマンド）	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥misc¥trace¥AdshTrace_［n］.log（n は面数）	8MB	○
トレースログ（エディタ独自機能）	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASD¥adshedit¥trace¥adshedit.txt	ユーザー環境の設定による	○
アプリケーション実行エージェント機能ログ【実行環境限定】	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥ appexec¥APPEXECAGENT.log 共通アプリケーションフォルダ¥Hitachi¥JP1AS¥JP1ASE¥ appexec¥APPEXECAGENT n.log（nは面数：最大2面）	5MB 程度	○
dump ファイル	定義ファイルの DUMP 以下の dump ファイル	ユーザー環境の設定による	△
環境ファイル	環境変数 ADSH_ENV のファイルまたは-e オプションで指定したファイル	1KB 程度	△
システム環境ファイル	共通アプリケーションフォルダ¥Hitachi¥JP1AS¥製品名¥conf¥adshrc.ase	1KB 程度	△
マシンに設定されているホスト名	システムルートフォルダ¥system32¥drivers¥etc¥hosts	ユーザー環境の設定による	○
マシンに設定されているサービスポート	システムルートフォルダ¥system32¥drivers¥etc¥services	ユーザー環境の設定による	○
環境情報ファイル	ADSHTMPyyyymmddhhmmss.txt yyyymmdd：adshcollect コマンドの起動日 hhmmss：adshcollect コマンドの起動時刻	ユーザー環境の設定による	○

（凡例）

○：adshcollect コマンドによって必ず採取します。

△：adshcollect コマンドのオプション指定時に採取します。

注※

環境ファイルで変更できます。パス名のデフォルト値については，前記の表を参照してください。

表 11-10　adshcollect コマンドで採取するファイルと最大サイズ【UNIX 限定】

ファイルの種類	ファイル名	最大サイズ	採取
スプール	[環境ファイルのSPOOL_DIR※] /.jobid	1KB 程度	○
システム実行ログ	[環境ファイルのLOG_DIR※] /AdshLog.log [環境ファイルのLOG_DIR※] /AdshLog_ [n] .log（n は面数）	[環境ファイルのLOG_FILE_SIZE] ×(n + 1) MB	○
	[環境ファイルのLOG_DIR※] /AdshLog.conf	1KB 程度	○
トレースログ	[環境ファイルのTRACE_DIR※] /AdshTrace_ [n] .log（n は面数）	[環境ファイルのTRACE_FILE_SIZE] ×nMB	○
core ファイル	定義ファイルのキーワード「CORE」で採取した core ファイル	ユーザー環境の設定による	△
ユーザー応答機能管理デーモンの情報	/opt/jp1as/system 以下の起動ログおよび pid ファイル	1KB 程度×稼働するユーザー応答機能管理デーモンの数	○
環境ファイル	環境変数 ADSH_ENV のファイルまたは-e オプションで指定したファイル	1KB 程度	△
システム環境ファイル	/opt/jp1as/conf/adshrc.ase	1KB 程度	○
インストール済みの日立製品	/etc/.hitachi/pplistd/pplistd	ユーザー環境の設定による	○
環境変数	• AIX または Linux の場合 /etc/environment • HP-UX の場合 /etc/profile • Solaris の場合 /etc/skel/.profile	ユーザー環境の設定による	○
環境情報ファイル	ADSHTMPyyyymmddhhmmss.txt yyyymmdd：adshcollect コマンドの起動日 hhmmss：adshcollect コマンドの起動時刻	ユーザー環境の設定による	○
tar のログ	ADSHTARyyyymmddhhmmss.txt yyyymmdd：adshcollect コマンドの起動日 hhmmss：adshcollect コマンドの起動時刻	1KB 程度	○

（凡例）

○：adshcollect コマンドによって必ず採取します。

△：adshcollect コマンドのオプション指定時に採取します。

注※

環境ファイルで変更できます。パス名のデフォルト値については，前記の表を参照してください。

注意事項

- 保守情報出力先ディレクトリには，出力ファイルおよび一時ファイルを作成するため，空き領域を確保してください。また，保守情報出力先ディレクトリは書き込み可能にしてください。
- adshcollect コマンドの実行中に強制終了すると，一時ファイルが保守情報出力先ディレクトリに残る場合があります。このような場合は，一時ファイルを手動で削除してください。
- 【UNIX 限定】ユーザー応答機能を使用している場合は，root 権限を持つユーザーで adshcollect コマンドを実行してください。root 権限を持つユーザー以外で実行した場合は，ユーザー応答機能の情報を採取できません。
- 【Windows 限定】ユーザー応答機能を使用している場合は，Administrators 権限を持つユーザーで adshcollect コマンドを実行してください。Administrators 権限を持つユーザー以外で実行した場合は，ユーザー応答機能の情報を採取できません。
- 保守情報出力先ディレクトリのパス，環境ファイルのパス，定義ファイルのパス，SPOOL_DIR のパス，LOG_DIR のパス，TRACE_DIR のパス，DUMP のパス，CORE のパスおよび adshcollect コマンド実行時のカレントディレクトリのパスに次の記号を指定しないでください。

  & ( ) [ ] { } ^ = ; ! ' + , ` ～ # %
- adshcollect コマンドの引数の保守情報出力先ディレクトリ，定義ファイル名，環境ファイル名，論理ホスト名にオプションを指定した場合，そのオプションをディレクトリまたはファイル名，論理ホスト名として解釈します。

12 メッセージ

この章では，JP1/Advanced Shell が出力するメッセージとエラーの詳細について説明します。

12.1　メッセージの形式

JP1/Advanced Shell のメッセージの形式について説明します。

12.1.1　メッセージの出力形式

JP1/Advanced Shell が出力するメッセージの形式を次に示します。

```
KNAXnnnn-t メッセージテキスト
```

- KNAX
  メッセージプリフィックスです。JP1/Advanced Shell のメッセージであることを示します。
- **nnnn**
  メッセージ番号を示します。
- **t**
  メッセージ種別です。メッセージに対する処置の指標を示します。メッセージ種別には，次の表に示す種類があります。

表 12-1　メッセージ種別

メッセージ種別	種類	意味
E	エラー（Error）	• 各ライブラリ，コマンドまたはサーバの機能が働かない障害が起きたことを示します。 • 定義誤り，コマンドの引数の指定誤りによって，動作できないことを示します。
W	警告（Warning）	メッセージ出力後，処理は続けられます。
I	情報（Information）	ユーザーに情報を知らせます。

(1)　ジョブ実行ログに出力されるメッセージの出力形式

メッセージがジョブ実行ログに出力される場合，次のようにメッセージの前に時刻およびジョブ識別子が付加されます。

```
時刻 ジョブ識別子 KNAXnnnn-t メッセージテキスト
```

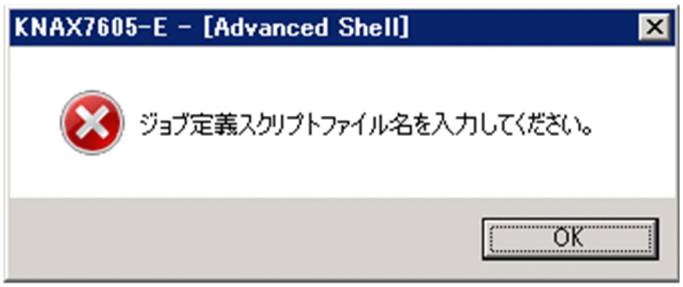
- **時刻**
  メッセージを出力した時刻を hh:mm:ss の形式で示します。
- **ジョブ識別子**
  メッセージを出力したジョブのジョブ識別子を 6 桁で示します。6 桁に満たない場合は，前方に 0 を付加して 6 桁にします。

(2) メッセージ用ダイアログボックスまたはエラーウィンドウに出力されるメッセージの出力形式

一部のメッセージは，次の図で示すメッセージ用ダイアログボックスまたはエラーウィンドウに出力されることがあります。

- メッセージ用ダイアログボックス

図 12-1　メッセージ用ダイアログボックス



メッセージ用ダイアログボックスでは，メッセージの内容によってメッセージの種類がアイコンで表示されます。メッセージ用ダイアログボックスでのアイコンの意味を次の表に示します。

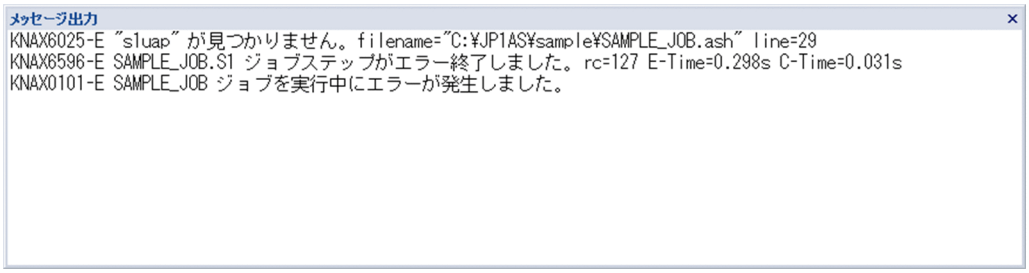
表 12-2　メッセージ用ダイアログボックスでのアイコンの意味

種類	アイコン	意味	ユーザーの対応方法
情報		処理中にユーザーに伝えるべき事象が発生したときに表示されます。	［OK］をクリックします。
質問		処理中にユーザーに問い合わせるべき事象が発生したことを伝え，二者択一の選択肢のどちらを実施するかを問い合わせます。	［はい］または［いいえ］を選択します。
警告		処理中にユーザーに警告すべき事象が発生したことを伝え，二者択一の選択肢のどちらを実施するかを問い合わせます。	［OK］または［キャンセル］を選択します。
エラー		処理中にエラーが発生したときに表示されます。	［OK］をクリックします。

- エラーウィンドウ

JP1/Advanced Shell エディタを使用している場合，次の図で示すエラーウィンドウにメッセージが表示されることもあります。

図 12-2　エラーウィンドウ



12.1.2　メッセージの記載形式

出力するメッセージの記載形式を次に示します。

メッセージテキスト中の**太字**で書かれている部分は，メッセージテキスト内で表示内容が変わる位置を示しています。メッセージ ID の右に【Windows 限定】または【UNIX 限定】が記載されている場合は，おのおのが Windows 環境または UNIX 環境だけで表示されます。

メッセージテキスト内に記載している**エラー詳細**の意味と対策方法については，「12.4　エラーの詳細」を参照してください。

メッセージはメッセージ ID 順に記載しています。記載形式の例を次に示します。

メッセージ ID　[【Windows 限定】|【UNIX 限定】]

　メッセージテキスト

　メッセージの説明文

　(S)

　　システムの処置を示します。

　(O)

　　メッセージが出力された場合の，開発者または運用者の対処を示します。

12.1.3　メッセージ番号の割り当て

JP1/Advanced Shell のメッセージプリフィックス KNAX の後に続く，メッセージ番号の範囲で示されるメッセージの意味を次の表に示します。

表 12-3　メッセージ番号の範囲で示されるメッセージの意味

メッセージ番号	メッセージの意味
0001～0299	ジョブの基本的な動作に関連するメッセージ
0300～0399	コマンドの引数に関連するメッセージ
0400～0699	環境ファイルに関連するメッセージ
0700～0899	ジョブ実行ログに関連するメッセージ
1600～1899	領域の割り当てに関連するメッセージ
1900～2199	ジョブステップの実行に関連するメッセージ
2200～2499	メッセージの処理に関連するメッセージ
2500～2699	データ解析に関連するメッセージ
3000～3999	デーモンに関連するメッセージ
4414～4429	スプールジョブの操作に関連するメッセージ

メッセージ番号	メッセージの意味
5300～5399	アダプタコマンドに関連するメッセージ
5400～5499	ユーザー応答コマンドに関連するメッセージ
6000～6699	バッチジョブの実行制御に関連するメッセージ
6700～6999	クロスプラットフォームに関連するメッセージ
7000～7199	開発環境に関連するメッセージ
7200～7399	アプリケーション実行エージェント機能
7400～7599	ユーザー応答機能に関連するメッセージ
7600～7799	JP1 連携機能に関連するメッセージ
7800～7999	共通機能に関連するメッセージ
9000～9999	ライセンス関連メッセージ

12.2　メッセージの出力先

JP1/Advanced Shell が出力するメッセージの出力先を次の表に示します。この表は拡張出力モードのメッセージ出力先を示しています。簡潔出力モードまたは最小出力モードのメッセージ出力先については，表の下の説明を参照してください。

表 12-4　JP1/Advanced Shell が出力するメッセージの出力先（拡張出力モードの場合）

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX0001-E	－	○	○	○	○	－
KNAX0004-I	－	－	－	○	－	－
KNAX0030-E，KNAX0031-E	－	○	－	－	○	－
KNAX0091-I，KNAX0092-I	－	－	○	○	－	－
KNAX0098-I	－	○	○	○	－	－
KNAX0101-E	－	○	○	○	○	－
KNAX0220-E	－	○	－	○	－	－
KNAX0235-E～KNAX0239-E	－	○	－	○	○	－
KNAX0240-I	－	○	－	○※1	－	－
KNAX0298-E～KNAX0299-E	－	○	○	○	○	－
KNAX0300-I	－	○	－	－	－	－
KNAX0301-E	－	○	－	－	○	－
KNAX0302-E	－	○	－	－	○	○※17
KNAX0303-E～KNAX0307-E	－	○	－	－	○	－
KNAX0308-E～KNAX0309-I	－	○	－	－	－	－
KNAX0310-E	－	○	－	－	○	－
KNAX0311-E～KNAX0336-E	－	○	－	－	○	○※17
KNAX0401-E～KNAX0702-E	－	○	－	－	○	－
KNAX0703-E	－	○	－	○	○	－
KNAX0704-E～KNAX0708-E	－	○	－	－	○	－
KNAX0719-I	－	○	－	－	－	－
KNAX0720-E～KNAX0723-E	－	○	－	－	○	－

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX0724-I	—	○	—	—	—	—
KNAX0725-E	—	○	—	—	○	—
KNAX0726-I	—	○	—	—	—	—
KNAX0727-E, KNAX0728-E	—	○	—	—	○	—
KNAX0800-E	—	○	—	—	○	—
KNAX0801-E	—	○	—	—	—	—
KNAX0802-E	—	○	—	—	○	—
KNAX0803-E	—	○	○	○	○	—
KNAX0804-E	—	○	—	—	○	—
KNAX0805-E	—	○	○	○	○	—
KNAX1600-I～KNAX1605-I	—	—	○	—	—	—
KNAX1632-E	—	—	○	—	○	—
KNAX1871-E～KNAX1880-E	—	○	—	—	—	—
KNAX1890-I	—	—	○	—	—	—
KNAX1891-E～KNAX1892-E	—	—	○	—	◎	—
KNAX1893-W	—	—	○	—	—	—
KNAX1910-E, KNAX1911-E	—	—	○	○	○	—
KNAX2201-E～KNAX2205-E	—	○	—	○	○	—
KNAX2206-E	—	○	—	—	○	—
KNAX2207-E	—	○	—	○	○	—
KNAX2208-E～KNAX2213-E	—	○	—	—	○	—
KNAX2214-E, KNAX2400-E	—	○	—	○	○	—
KNAX2499-E	—	○	—	○	—	—
KNAX2500-E	—	—	○	○	◎	—
KNAX2501-E	—	—	○	○	◎	—
KNAX3000-I	○	○※2※3	—	○	—	—
KNAX3001-I	—	○※2※3	—	○	—	—
KNAX3002-E	○	○※2※3	—	○	—	—
KNAX3003-E	—	○※2※3	—	○	—	—

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX3006-I	—	○※2	—	○	—	—
KNAX3008-W, KNAX3009-E	○	○※2	—	○	—	—
KNAX3020-E～KNAX3029-E	—	○※2	—	○	—	—
KNAX3261-I	—	—	—	○	—	—
KNAX3400-I～KNAX3542-W	—	○※2	—	○	—	—
KNAX3700-I～KNAX3799-I	○	—	—	—	—	—
KNAX3998-E, KNAX3999-E	—	○	—	—	—	—
KNAX4414-E～KNAX4429-E	—	○※4	—	—	—	—
KNAX5300-I～KNAX5372-E	—	○※5	—	○	—	—
KNAX5380-I, KNAX5381-I	—	—	—	○	—	—
KNAX5396-I～KNAX5399-E	—	○	—	—	—	—
KNAX5407-E～KNAX5499-E	—	○	—	—	—	—
KNAX6000-E～KNAX6071-E	—	—	○	—	◎	—
KNAX6072-E	—	—	○	—	◎	—
KNAX6075-E～KNAX6099-E	—	—	○	—	◎	—
KNAX6100-E	—	○	—	—	◎	—
KNAX6110-I～KNAX6129-I	—	—	○	○	—	—
KNAX6130-E	—	○	○	○	—	—
KNAX6134-E～KNAX6140-E	—	—	○	—	◎	—
KNAX6150-E	—	—	○	○	◎	—
KNAX6151-E	—	—	○	○	—	—
KNAX6152-E	—	—	○	○	○	—
KNAX6153-E	—	—	○	○	○	—
KNAX6160-I	—	—	○	○	—	—
KNAX6161-I	—	—	○	○	—	—
KNAX6190-E	—	—	○	○	◎	—
KNAX6191-E	—	—	○	○	◎	—
KNAX6192-E	—	—	○	○	◎	—
KNAX6193-E	—	—	○	○	◎	—

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX6194-E	－	－	○	○	◎	－
KNAX6200-I	－	○	○	○	○	－
KNAX6201-E	－	○	－	－	－	－
KNAX6202-E～KNAX6208-E	－	○	－	－	○	－
KNAX6209-W	－	○	－	－	－	－
KNAX6210-E～KNAX6215-E	－	○	－	○	－	－
KNAX6219-E	－	○	－	○	○	－
KNAX6220-I～KNAX6222-I	－	○	－	○	－	－
KNAX6223-E～KNAX6241-E	－	○	－	○	○	－
KNAX6242-I～KNAX6243-I	－	○	－	○	－	－
KNAX6244-E	－	○	－	○	○	－
KNAX6290-E～KNAX6298-E	－	－	－	－	○	－
KNAX6301-E～KNAX6303-E	－	○	－	－	○	－
KNAX6304-E	－	－	○	－	○	－
KNAX6305-E	－	○	－	－	○	－
KNAX6306-E	－	－	○	－	○	－
KNAX6307-W	－	－	○	－	－	－
KNAX6308-E, KNAX6309-E	－	－	○	－	○	－
KNAX6310-E～KNAX6319-E	－	－	○	－	◎	－
KNAX6320-E	－	－	○	－	○	－
KNAX6321-E	－	－	○	－	◎	－
KNAX6323-E	－	○	－	－	○	－
KNAX6324-E～KNAX6330-E	－	－	○	－	◎	－
KNAX6332-E	－	－	○	－	○	－
KNAX6333-E	－	－	○	－	◎	－
KNAX6340-E	－	－	○	○	○	－
KNAX6341-E	－	－	○	○	◎	－
KNAX6342-E	－	－	○	○	○	－
KNAX6380-I	－	○	－	－	－	－

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX6381-E	－	○	－	－	○	－
KNAX6382-I～KNAX6385-E	－	○	－	－	－	－
KNAX6399-E, KNAX6400-E	－	－	○	－	○	－
KNAX6401-E	－	－	○	－	◎	－
KNAX6403-E	－	－	○	－	◎	－
KNAX6404-E	－	－	○	－	○	－
KNAX6405-E～KNAX6407-E	－	－	○	－	◎	－
KNAX6408-E	－	－	○	－	○	－
KNAX6409-I, KNAX6410-I	－	－	○	－	－	－
KNAX6411-E～KNAX6413-E	－	－	○	－	○	－
KNAX6414-E	－	－	○	－	◎	－
KNAX6501-I	－	－	○	○	－	－
KNAX6502-I	－	－	○	○	－	－
KNAX6503-E	－	○	－	－	－	－
KNAX6504-E	－	－	○	－	○	－
KNAX6507-I～KNAX6511-I	－	－	○	○	－	－
KNAX6512-I	－	○	－	－	－	－
KNAX6521-E, KNAX6522-E	－	－	○	○	◎	－
KNAX6530-E, KNAX6531-E	－	－	○	－	◎	－
KNAX6540-I	－	－	○	○	－	－
KNAX6541-E, KNAX6542-E	－	－	○	○	○	－
KNAX6551-E～KNAX6578-I	－	－	○	○	－	－
KNAX6580-E	－	－	○	○	◎	－
KNAX6581-E	－	－	○	○	○	－
KNAX6582-E	－	－	○	○	－	－
KNAX6583-E	－	－	○	○	－	－
KNAX6584-I～KNAX6586-E	－	－	○	○	－	－
KNAX6587-E	－	○	－	○	－	－
KNAX6588-E	－	○	－	－	－	－

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX6589-W	—	—	—	○	—	—
KNAX6590-E	—	—	○	○	○	—
KNAX6591-E～KNAX6592-E	—	—	○	○	—	—
KNAX6593-E	—	○	○	○	○	—
KNAX6594-E	—	—	○	○	—	—
KNAX6595-E	—	—	○	○	○	—
KNAX6596-E	—	○	○	○	○	—
KNAX6597-I	—	○	○	○	—	—
KNAX6598-E, KNAX6599-E	—	—	○	○	○	—
KNAX6600-E～KNAX6646-E	—	○	—	○	—	—
KNAX6701-W	—	—	○	○	—	—
KNAX6710-I	—	—	○	—	—	—
KNAX6711-E, KNAX6712-E	—	—	○	—	◎	—
KNAX6713-E	—	—	○	○	○	—
KNAX6714-E, KNAX6715-E	—	—	○	—	◎	—
KNAX6718-I	—	—	○	—	—	—
KNAX6750-E～KNAX6753-E	—	○	—	○	—	—
KNAX6800-I※6, KNAX6801-I※6	—	—	—	—	—	—
KNAX6803-I～KNAX6806-I	—	—	○	—	—	—
KNAX6810-E～KNAX6812-E	—	—	○	○	○	—
KNAX6813-E	—	—	○	—	◎	—
KNAX6814-E, KNAX6815-E	—	—	○	○	○	—
KNAX6830-I～KNAX6832-I	—	—	○	○	—	—
KNAX6996-E, KNAX6997-E	—	—	○	○	○	—
KNAX6998-E	—	—	○	○	◎	—
KNAX6999-E	—	—	○	○	○	—
KNAX7000-E～KNAX7004-E	—	○	—	○	○	—
KNAX7006-W～KNAX7009-I	—	○	—	○	—	—
KNAX7010-E	—	○	—	○	○	—

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX7011-I, KNAX7012-W	—	○	—	○	—	—
KNAX7013-E, KNAX7014-E	—	○	—	○	○	—
KNAX7015-W	—	○	—	○	—	—
KNAX7016-E, KNAX7017-E	—	○	—	○	○	—
KNAX7018-I	—	○	—	○	—	—
KNAX7019-E～KNAX7022-E	—	○	—	○	○	—
KNAX7023-I	—	○	—	○	—	—
KNAX7024-E	—	○	—	○	○	—
KNAX7025-I	—	○	—	○	—	—
KNAX7026-E～KNAX7029-E	—	○	—	○	○	—
KNAX7032-I～KNAX7034-I	—	○	—	○	—	—
KNAX7035-E	—	○	—	○	○	—
KNAX7036-I, KNAX7037-I	—	○	—	○	—	—
KNAX7038-I	—	○	○	○	—	—
KNAX7039-E, KNAX7040-E	—	○	—	○	○	—
KNAX7043-I	—	○	—	○	—	—
KNAX7044-E～KNAX7046-E	—	○	—	○	○	—
KNAX7047-I, KNAX7048-I	—	○	—	○	—	—
KNAX7049-E～KNAX7052-E	—	○	—	○	○	—
KNAX7053-I	—	○	—	○	—	—
KNAX7054-E, KNAX7055-E	—	○	—	○	○	—
KNAX7056-I, KNAX7057-I	—	○	○	○	—	—
KNAX7058-I	—	○	—	—	—	—
KNAX7062-E	—	○	—	○	○	—
KNAX7063-I, KNAX7064-I	—	○	○	○	—	—
KNAX7065-I～KNAX7067-I	—	○	—	○	—	—
KNAX7068-I	—	○	○	○	—	—
KNAX7070-E	—	○	—	○	○	—
KNAX7071-E, KNAX7072-E	—	○	—	○	—	—

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX7073-I	—	○	○	○	—	—
KNAX7090-W	—	—	—	—	○	—
KNAX7091-W	—	—	—	—	○	—
KNAX7099-E, KNAX7101-E	—	○	○	○	○	—
KNAX7102-I, KNAX7103-I	—	—	—	○	—	—
KNAX7104-E～KNAX7106-E	—	○	○	○	○	—
KNAX7107-I	—	—	—	○	—	—
KNAX7108-E	—	○	○	○	○	—
KNAX7109-I	—	—	—	○	—	—
KNAX7110-E	—	○	○	○	○	—
KNAX7111-I	—	—	—	○	—	—
KNAX7112-E～KNAX7116-E	—	○	○	○	○	—
KNAX7117-I	—	—	—	○	—	—
KNAX7118-E	—	—	—	—	○	—
KNAX7119-E	—	○	○	○	○	—
KNAX7120-W	—	—	—	○	—	—
KNAX7121-E～KNAX7125-E	—	○	○	○	○	—
KNAX7126-I, KNAX7127-E	—	○	○	○	—	—
KNAX7128-E	—	○	—	○	—	—
KNAX7129-E	—	○	○	○	—	—
KNAX7200-E～KNAX7203-W	—	—	—	—	—	○※13
KNAX7204-E	—	○※15	—	—	—	○※13
KNAX7205-E	—	—	—	—	—	○※13,14
KNAX7210-E～KNAX7217-E	—	—	—	—	—	○※13
KNAX7221-E～KNAX7225-E	—	—	—	—	—	○※14

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX7250-I～KNAX7253-I	－	－	－	－	－	○※13
KNAX7254-E	－	○※15	－	－	－	○※13※16
KNAX7255-I	－	－	－	－	－	○※13
KNAX7256-E～KNAX7259-W	－	○	－	－	－	○※13
KNAX7260-W～KNAX7262-I	－	－	－	－	－	○※13
KNAX7263-E	－	－	－	－	－	○※13
KNAX7264-I～KNAX7268-I	－	○	－	－	－	○※13
KNAX7269-E	－	－	－	－	－	○※13
KNAX7270-E	－	○	－	－	－	○※13
KNAX7271-I	－	○	－	－	－	－
KNAX7400-E～KNAX7402-E	－	－	○	○	○	－
KNAX7403-E～KNAX7405-E	－	－	○	○	◎	－
KNAX7408-E	－	－	○	○	○	－
KNAX7420-E	－	○	－	－	－	－
KNAX7450-I※7	－	－	－	－	－	－
KNAX7451-I	－	－	○	○	－	－
KNAX7460-E～KNAX7465-W	－	－	○	○※8	－	－
KNAX7470-I	－	－	○	○	－	－
KNAX7500-I～KNAX7509-I※7	－	－	－	－	－	－
KNAX7550-I, KNAX7551-E※9	○	－	－	－	－	－
KNAX7552-E～KNAX7556-E※9	－	－	－	－	－	－
KNAX7560-I, KNAX7561-E※9	○	－	－	－	－	－
KNAX7600-E～KNAX7773-E	－	－	－	－	○	－

メッセージ ID の範囲	メッセージの出力先					
	stdout	stderr	JOBLOG	システム実行ログ	GUI ※12	その他
KNAX7800-I～KNAX7880-E	○	－	－	－	－	－
KNAX7892-I～KNAX7897-E	－	○	－	－	－	－
KNAX7900-I	－	－	－	－	○※10	－
KNAX7901-I	－	○	○※11	○※11	－	－
KNAX7902-I	－	－	○	○	－	－
KNAX7999-I	－	○	－	－	－	－
KNAX9000-E～KNAX9002-E	－	○	－	－	－	－

（凡例）

表の「メッセージの出力先」列の意味を次に示します。

表の項目	メッセージの出力先	特記事項
stdout	標準出力	簡潔出力モードまたは最小出力モードの場合，通常実行時のジョブと，デバッグ実行時の子孫ジョブのメッセージの出力は次のようになります。 • メッセージ種別I，W のメッセージは出力されません。 • メッセージ種別E のメッセージは該当の出力先とJOBLOG に出力されます。 デバッグ実行時のルートジョブのメッセージ種別E のメッセージはJOBLOG にも出力されます。
stderr	標準エラー出力	
JOBLOG	ジョブ実行ログ	• 拡張出力モードの場合 デバッグ実行時は，ジョブ実行中に同じメッセージが標準エラー出力にも出力されます。 子孫ジョブの場合はJOBLOG には出力されないで，ジョブ実行完了時に標準エラー出力に出力されます。 • 簡潔出力モードまたは最小出力モードの場合 メッセージ種別E のメッセージは標準エラー出力にも出力されます。
システム実行ログ	システム実行ログ	メッセージ出力時のジョブの状態によっては，この出力先にメッセージが出力されないことがあります。
GUI	メッセージ用ダイアログボックスまたはエラーウィンドウ	なし

ただし，簡潔出力モードまたは最小出力モードの場合も，次のメッセージは出力されます。

- KNAX0240-I（ADSH_JOBRC_FATAL の値を適用したことを示すメッセージ。出力のタイミングによって抑止されることもあります。）
- KNAX0300-I（usage）
- KNAX0309-I（バージョン表示）

- シグナル受信で出力するメッセージ（ただし，通常実行時の最小出力モードではシグナル受信で出力されるメッセージの一部は出力抑止される）

◎：GUI のメッセージで JP1/Advanced Shell エディタのエラーウィンドウで行番号を付けて出力します。ただし，メッセージの行番号が省略される場合については出力しません。

○：出力します。

－：出力しません。

注※1

CUI デバッグ実行時にだけ出力されます。

注※2

起動ログに出力されます。

注※3

syslog に次の内容で出力されます。

- ファシリティ：LOG_USER
- レベル：LOG_NOTICE

注※4

adshhk コマンドの引数で指定したログファイルのオープン中は，標準エラー出力ではなく，指定したログファイルに出力されます。

注※5

アダプタコマンドの出力は，JP1/IM - View の画面に出力されます。

注※6

スクリプトイメージファイルに出力します。

注※7

JP1 イベントが発行されます。

注※8

adshecho コマンドまたはadshread コマンドの実行時に出力されます。

注※9

イベントログに出力されます。

注※10

GUI からヘルプを選択した場合に起動する Web ブラウザに表示されます。

注※11

CUI デバッグ実行時，および GUI デバッグ実行時には，この出力先には出力されません。

注※12

子孫ジョブからは GUI に出力されません。

注※13

アプリケーション実行エージェント機能ログに出力します。

注※14

メッセージボックスに出力します。

注※15

adshappexec コマンドの場合に出力します。

注※16

adshappagent コマンドの起動時にはメッセージボックスに出力します。

注※17

adshappagent コマンドの場合にはアプリケーション実行エージェント機能ログに出力します。

12.2.1 メッセージに出力される行番号に関する注意事項

メッセージ KNAX6000-E～KNAX6100-E，KNAX6710-I～KNAX6712-E，KNAX6998-E に出力される，行番号に関する注意事項です。

- 複数行にまたがったコマンド置換でコマンドエラーが発生した場合，コマンド置換の最終行番号がエラー行番号としてメッセージに出力されます。

  (例) 次のような記述の場合，unset コマンドでエラーが発生しても，エラー行番号は 3 行目となります。

```
1: `unset
2:  echo pwd
3:  `
```

- 外部スクリプトの構文解析でエラーが発生した場合，エラーメッセージに出力されるジョブ定義スクリプトファイル名は外部スクリプト呼び出し元のジョブ定義スクリプト名になります。また，行番号は外部スクリプト呼び出し元の行番号となります。
- trap コマンドの action 実行中に構文エラーまたはコマンドエラーが発生すると，trap コマンドの行番号がエラー行番号としてメッセージに出力されます。

  (例 1) 複数行にまたがっているケース。エラー行番号は 1 行目となります。

```
1: trap ’pwd
2: unset
3: date’ INT
```

  (例 2) 関数を呼び出すケース。エラー行番号は 4 行目となります。

```
1: func1() {
2:  unset
3: }
4: trap func1 INT
```

12.3　メッセージの一覧

JP1/Advanced Shell が出力するメッセージと対処方法について説明します。

KNAX0001-E

メモリが不足しています。details=**保守情報**

メモリ不足が発生しました。

保守情報は，8 桁の 16 進数で表示します。**保守情報**は，システムの内部状態を示す情報です。このメッセージは，エラー通知の出力先に出力されますが，エラー発生のタイミングによって一部の出力先にだけ出力されます。

(S)

処理を終了します。

(O)

システム管理者に連絡します。システム管理者は，メモリ見積もりを見直してください。

KNAX0004-I

ジョブ識別子=**Advanced Shellのジョブ識別子**, JP1NBQSQueueName=**環境変数値**, ジョブ番号=**スケジューラのジョブ番号**

起動したバッチジョブの JP1/AJS のジョブ情報と JP1/Advanced Shell のジョブ識別子を表示します。

Advanced Shellのジョブ識別子

ジョブコントローラがバッチジョブに付与したジョブ識別子

環境変数値

バッチジョブの JP1NBQSQueueName 環境変数の値

スケジューラのジョブ番号

JP1/AJS がバッチジョブに付与した JP1 ジョブ番号

(S)

処理を続行します。

KNAX0030-E 【Windows 限定】

An error occurred while starting adshexec. function="**関数名**",error code=**エラーコード**,reason="**エラー詳細**"

ジョブコントローラの開始処理中にエラーが発生しました。**関数名**，**エラーコード**および**エラー詳細**を示します。

(S)

処理を終了します。

(O)

システム管理者に連絡します。システム管理者は，**関数名**，**エラーコード**および**エラー詳細**からエラーの原因を取り除いて，バッチジョブを再実行します。

KNAX0031-E 【Windows 限定】

An error occurred while completing adshexec process. function="**関数名**",error code=**エラーコード**,reason="**エラー詳細**"

ジョブコントローラの終了処理中にエラーが発生しました。**関数名**，**エラーコード**および**エラー詳細**を示します。

(S)

処理を終了します。

(O)

システム管理者に連絡します。システム管理者は，**関数名**，**エラーコード**および**エラー詳細**からエラーの原因を取り除いて，バッチジョブを再実行します。

KNAX0091-I

ジョブ名 ジョブが開始しました。

ジョブ名で示すバッチジョブを開始しました。

(S)

処理を続行します。

KNAX0092-I

ジョブ名.ジョブステップ名 ステップが開始しました。

ジョブ名で示すバッチジョブに定義された，**ジョブステップ名**で示すジョブステップを開始しました。

(S)

処理を続行します。

KNAX0098-I

ジョブ名 ジョブが終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

ジョブ名で示すバッチジョブが終了しました。

終了コード

バッチジョブの実行結果を示す終了コード。

終了コードの詳細は，adshexec コマンドの終了コードに関する説明を参照してください。

なお，このメッセージの出力後に adshexec コマンドの後処理でエラーが発生した場合，このメッセージで示す終了コードと adshexec コマンドの終了コードとが異なる場合があります。adshexec コマンドの最終的な終了コードは，KNAX7999-I に出力されます。

実行時間

バッチジョブの開始から終了までの実時間の合計（秒単位）。OS の API を使って取得した参考値です。

CPU時間

バッチジョブの開始から終了までの CPU 時間の合計（秒単位）。OS の API を使って取得した参考値です。

(S)

処理を続行します。

KNAX0101-E

ジョブ名 ジョブを実行中にエラーが発生しました。

ジョブ名で示すバッチジョブを実行中にエラーが発生しました。

(S)

処理を続行します。

(O)

一緒に出力されるほかのメッセージを参照してエラーの原因を取り除き，バッチジョブを再実行します。

KNAX0220-E

環境変数（"**環境変数名**"）が指定されていません。

環境変数名で示す環境変数が指定されていません。

環境変数名が「JP1_HOSTNAME」の場合，論理ホスト運用で論理ホスト名が明示指定されていないとき，JP1/AJS 以外から起動されているおそれがあります。

(S)

処理を終了します。

(O)

システム管理者に連絡して環境変数の値を見直してください。

KNAX0235-E

環境変数（"**環境変数名**"）の値が不当です。

環境変数名で示す環境変数の値が不当です。

環境変数名

環境変数名

(S)

処理を終了します。

(O)

環境変数の値を見直してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX0236-E

環境変数（"**環境変数名**"）の値が長すぎます。

または

The value specified for the environment variable "**環境変数名**" is too long.

環境変数名で示す環境変数の値が長過ぎます。

(S)

処理を終了します。

(O)

環境変数の値を見直してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX0237-E

環境変数（"**環境変数名**"）の値が空文字です。
または
The value specified for the environment variable "**環境変数名**" is an empty string.

環境変数名で示す環境変数の値が空文字です。

(S)

処理を終了します。

(O)

システム管理者に連絡して環境変数の値を見直してください。

KNAX0238-E

環境変数（"**環境変数名**"）の値に使用できない文字が含まれています。
または
The value specified for the environment variable "**環境変数名**" contains an invalid character.

環境変数名で示す環境変数の値に使用できない文字が含まれています。

(S)

処理を終了します。

(O)

環境変数の値を見直してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX0239-E

環境変数（"**環境変数名**"）の値が範囲外です。
または

The value specified for the environment variable "**環境変数名**" is out of range.

環境変数名で示す環境変数の値が範囲外です。

(S)

処理を終了します。

(O)

システム管理者に連絡して環境変数の値を見直してください。

KNAX0240-I

環境変数名 was applied. value=**環境変数値**

または

The setting specified for the environment variable **環境変数名** was applied. value=**環境変数値**

環境変数名で示す環境変数の設定が適用されました。

環境変数値

適用された環境変数の値

(S)

処理を続行します。

KNAX0298-E

スプールジョブ名の取得に失敗しました。 reason="エラー*詳細*"

エラー*詳細*で示す原因によってスプールジョブ名の取得に失敗しました。以前に取得したスプールジョブ名または JP1/Advanced Shell のジョブ名が使用されます。

(S)

処理を続行します。

(O)

シェル変数ADSH_SPOOL_JOBNAME の値を見直してください。

KNAX0299-E

内部エラーが発生しました。details=**保守情報**

メモリ確保で内部矛盾が発生しました。

このメッセージは，エラー通知の出力先に出力されますが，エラー発生のタイミングによって一部の出力先にだけ出力されます。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX0300-I

使用方法: **コマンド名 コマンド引数**

コマンド名および**コマンド引数**で示すコマンドの指定が誤っています。

(S)

処理を終了します。

(O)

コマンドを正しく指定して実行します。

KNAX0301-E

オプション（"**オプション名**"）の値が指定されていません。

オプション名の指定値が誤っています。

(S)

処理を中断します。

(O)

オプションを正しく指定します。

KNAX0302-E

オプション（"**オプション名**"）は無効なオプションです。

不正な**オプション名**を指定しています。

(S)

処理を中断します。

(O)

オプション名を正しく指定します。

KNAX0303-E

ジョブ定義スクリプトファイル名が指定されていません。

ジョブ定義スクリプトファイル名が指定されていません。

(S)

処理を中断します。

(O)

ジョブ定義スクリプトファイル名を指定して再入力します。

KNAX0305-E

コマンドの引数（"**引数**"）が不正です。

引数で示すコマンドの引数が不正です。

(S)

処理を中断します。

(O)

引数を正しく指定して再入力します。

KNAX0306-E

オプション（"**オプション名**"）の値が誤っています。

オプション名で示すオプションの値が誤っています。

(S)

処理を中断します。

(O)

オプションを正しく指定して再入力します。

KNAX0307-E

必要なオプションが指定されていません。

必要なオプションが指定されていません。

(S)

処理を終了します。

(O)

必要なオプションを指定して再入力します。

KNAX0308-E

オプション（"**オプション名1**"）とオプション（"**オプション名2**"）は同時に指定できません。

オプション名1 で示すオプションと**オプション名2** で示すオプションは，同時に指定することはできません。

(S)

処理を中断します。

(O)

オプションを正しく指定して再入力します。

KNAX0309-I

プログラム名 のバージョンは **バージョン文字列** です。

プログラム名で示すコマンドのバージョンを**バージョン文字列**に示します。

(S)

処理を終了します。

KNAX0310-E

オペランドが多すぎます。

指定したオペランドが多過ぎます。

(S)

処理を中断します。

(O)

オペランドを正しく指定します。

KNAX0311-E

コマンド名 コマンド処理に必要なオプション，パラメーターが不足しています。

コマンド名で示すコマンド処理に必要なオプション，パラメーターが不足しています。

(S)

処理を終了します。

(O)

必要なオプション，パラメーターを指定して再入力します。

KNAX0336-E

オプション（"**オプション名**"）の指定値の長さが正しくありません。

オプション名の指定値の長さが正しくありません。次の原因が考えられます。

- オプション名の指定値の長さが長過ぎます。
- オプション名の指定値の長さが 0 です。

(S)

処理を中断します。

(O)

オプションを正しく指定して再入力します。

KNAX0401-E

環境ファイルのオープンに失敗しました。reason="**エラー詳細**"

エラー詳細で示す原因によって，環境ファイルのオープンに失敗しました。

(S)

処理を終了します。

(O)

エラー詳細を基に環境ファイルが読み込めるように，権限などの問題がないかどうかを確認します。問題が解決しない場合は，システム管理者に連絡します。

KNAX0402-E

環境ファイルの読み込みに失敗しました。reason="**エラー詳細**"

エラー詳細で示す原因によって，環境ファイルの読み込みに失敗しました。

(S)

処理を終了します。

(O)

エラー詳細を基に環境ファイルが読み込めるように，権限などの問題がないかどうかを確認します。問題が解決しない場合は，システム管理者に連絡します。

KNAX0403-E

環境ファイルのファイル名が長過ぎます。

環境ファイルのファイル名が長過ぎます。

(S)

処理を終了します。

(O)

環境ファイル名の指定に問題がないかどうかを確認します。

KNAX0406-E

ホスト名の取得時にエラーが発生しました。reason="**エラー詳細**"

エラー詳細で示す原因によって，ホスト名の取得時にエラーが発生しました。UNIX の場合，ホスト名が 255 文字より多いと，このメッセージが出力されることがあります。

(S)

処理を終了します。

(O)

システム管理者に連絡して，ネットワーク上のホスト名を確認します。

KNAX0407-E

"**ファイル名**" は通常のファイルではありません。

ファイル名で示すファイルは通常のファイルではありません。

(S)

処理を終了します。

(O)

ファイルを確認します。

KNAX0410-E

環境ファイル（"**ファイル名**"）の解析でエラーが発生しました。エラーの内容はこのメッセージの前に出力されているメッセージを参照してください。

ファイル名で示す環境ファイルの解析でエラーが発生しました。エラーの内容はこのメッセージの前に出力されているメッセージを参照してください。

(S)

処理を終了します。

(O)

環境ファイルの誤りを修正してください。

KNAX0411-E

環境ファイルの行が長すぎます。line=**行番号**

行番号で示す，環境ファイルの行が長すぎます。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0431-E

環境ファイルに不当なパラメーター名が見つかりました。line=**行番号**

環境ファイルの**行番号**で示す行に不当なパラメーター名が見つかりました。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0432-E

環境ファイルのパラメーター（"**パラメーター名**"）に不当な値が見つかりました。line=**行番号**

環境ファイルの**行番号**で示すパラメーターに不当な値が見つかりました。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0433-E

環境ファイルのパラメーター（"**パラメーター名**"）に値が設定されていません。line=**行番号**

環境ファイルの**行番号**で示すパラメーターに値が設定されていません。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0434-E

環境ファイルのパラメーター（"**パラメーター名**"）が重複して定義されています。line=**行番号**

環境ファイルの**行番号**で示すパラメーターが重複して定義されています。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0435-E

環境ファイルのパラメーター（"**パラメーター名**"）を定義できる上限を超えています。line=**行番号**

環境ファイルの**行番号**で示すパラメーターが定義できる上限を超えています。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0436-E

環境ファイルのパラメーター（"**パラメーター名**"）の指定値が長すぎます。line=**行番号**

環境ファイルの**行番号**で示すパラメーターの指定値が長過ぎます。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0437-E

環境ファイルのパラメーター（"**パラメーター名**"）の指定値が範囲外です。line=**行番号**

環境ファイルの**行番号**で示すパラメーターの指定値が範囲外です。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0438-E

環境ファイルのパラメーター（"**パラメーター名**"）の指定値に不当な文字が含まれています。line=**行番号**

環境ファイルの**行番号**で示すパラメーターの指定値に不当な文字が含まれています。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0439-E

環境ファイルのパラメーター（"**パラメーター名**"）のファイルパスが絶対パスで指定されていません。line=**行番号**

環境ファイルの**行番号**の**パラメーター名**で示すファイルパスが絶対パスで指定されていません。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0441-E

環境ファイルのパラメーター（"**パラメーター名**"）で指定したディレクトリは存在しません。line=**行番号**

環境ファイルの**行番号**で示すパラメーターで指定したディレクトリはありません。

UNIX の場合，シグナルの受信が原因でこのメッセージが出力されることがあります。

(S)

処理を終了します。

(O)

環境ファイルを見直すか，または動作環境をチェックしてください。

KNAX0442-E

環境ファイルのパラメーター（"**パラメーター名**"）で指定した値はディレクトリではありません。line=**行番号**

環境ファイルの**行番号**で示すパラメーターで指定した値はディレクトリではありません。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0444-E

環境ファイルのパラメーター（"**パラメーター名**"）で指定したオペランドは多すぎます。line=**行番号**

環境ファイルの**行番号**で示すパラメーターで指定したオペランドは多過ぎます。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0445-E

環境ファイルのパラメーター（"**パラメーター名**"）で仮定されるディレクトリ（"**デフォルトディレクトリ名**"）が存在しません。

デフォルトのディレクトリがありません。

UNIX の場合，シグナルの受信が原因でこのメッセージが出力されることがあります。

(S)

処理を終了します。

(O)

動作環境を見直してください。

KNAX0446-E

環境ファイルのパラメーター（"**パラメーター名**"）で仮定したディレクトリ（"**デフォルトディレクトリ名**"）がディレクトリではありません。

デフォルトのディレクトリ名がディレクトリではありません。

(S)

処理を終了します。

(O)

動作環境を見直してください。

KNAX0449-E

必須のディレクトリ（"**ディレクトリ名**"）がありません。

必須の**ディレクトリ名**がありません。

UNIX の場合，シグナルの受信が原因でこのメッセージが出力されることがあります。

(S)

処理を終了します。

(O)

環境ファイルを見直すか，または動作環境を見直してください。

KNAX0450-E

必須のディレクトリ（"**ディレクトリ名**"）はディレクトリではありません。

必須の**ディレクトリ名**はディレクトリではありません。

(S)

処理を終了します。

(O)

環境ファイルを見直すか，または動作環境を見直してください。

KNAX0451-E 【Windows 限定】

環境ファイルのパラメーター（"**パラメーター名**"）で仮定するディレクトリ名の取得に失敗しました。

デフォルトのディレクトリ名を求める処理でエラーが発生しました。

(S)

処理を終了します。

(O)

動作環境を見直してください。

KNAX0456-E 【Windows 限定】

パラメーター（"**パラメーター名**"）で指定した値が重複して定義されています。line=**行番号**

パラメーター名で指定した値が重複して定義されています。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0458-E

パラメーターの組み合わせが正しくありません。line=**行番号**

パラメーターの組み合わせが正しくありません。次の原因が考えられます。

- phost_start パラメーターと phost_end パラメーターの組み合わせが正しくない。
- lhost_start パラメーターと lhost_end パラメーターの組み合わせが正しくない。
- パラメーターの指定順を誤っている。

- 開始のパラメーターの指定があるが，終了のパラメーターの指定がない。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0459-E

パラメーター（"**パラメーター名**"）でオペランドの指定順序に誤りがあるか，または同じオペランドを複数回指定しています。line=**行番号**

オペランドの指定順序に誤りがあるか，または同じオペランドを複数回指定しています。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0471-E

パラメーター（**パラメーター名**）の指定値が，システム環境ファイルの指定値と異なります。filename="**ファイル名**"

ファイル名の環境ファイルで指定された**パラメーター名**のパラメーターの指定値が，システム環境ファイルの指定値と異なります。

システム環境ファイルでデフォルト値を指定している場合，ジョブ環境ファイルで異なる値を指定するとエラーになります。システム環境ファイルで指定したシステム実行ログおよびトレースの設定は変更できません。

ジョブ環境ファイルで明示指定していない場合はエラーになりません。

(S)

処理を終了します。

(O)

環境ファイルを見直してください。

KNAX0472-E

環境ファイルの解析処理で，予期しないエラーが発生しました。("**関数名**", "**エラー詳細**", **保守情報**)

環境ファイルの解析処理で，予期しないエラーが発生しました。

関数名

内部の関数名

エラー詳細

障害内容を示す文字列

保守情報

保守コード

(S)

処理を終了します。

(O)

障害を取り除いて再実行してください。障害が取り除けない場合は，システム管理者に連絡してください。

KNAX0473-W

冗長なパラメーター（"**パラメーター名**"）が**環境ファイル種別**に指定されています。

環境ファイル種別で示すファイルに，冗長なパラメーターが指定されています。

生じる問題点は各パラメーターの説明を参照してください。

該当のパラメーターは**環境ファイル種別**には指定しないことを推奨します。

環境ファイル種別

ジョブ環境ファイル

(S)

処理を実行します。

(O)

生じる問題が許容できない場合は，該当するパラメーターの指定を削除してください。

KNAX0474-E

指定された値（"**パラメーター値**"）は，現在のジョブ起動方法ではパラメーター（"**パラメーター名**"）に指定できません。filename="**ファイル名**"

ファイル名で示す環境ファイルに，現在のジョブ起動方法では指定できない値を指定しています。

パラメーター値

指定できないパラメーター値

パラメーター名

エラーとなったパラメーター名

ファイル名

エラーとなったパラメーターを指定した環境ファイル

(S)

処理を終了します。

(O)

環境ファイルに設定したパラメーターの内容を見直してください。

KNAX0700-E

スプールルートディレクトリの下にバッチジョブ用のディレクトリを作成できません。reason=**エラー詳細**

スプールルートディレクトリの下にバッチジョブ用のディレクトリを作成できません。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者は環境ファイルのスプールルートディレクトリの指定，またはスプールディレクトリ自体に異常がないか見直してください。

KNAX0701-E

ファイル（"**ファイル名**"）のオープンに失敗しました。reason=**エラー詳細**

スプールジョブディレクトリ内の**ファイル名**で示されるファイルのオープンに失敗しました。または，出力用コンソールのオープンに失敗しました。

出力用コンソールの場合，**ファイル名**は CONOUT$となります。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0702-E

スプールジョブディレクトリ内のジョブ実行ログファイルの書き込みに失敗しました。reason=**エラー詳細**

スプールジョブディレクトリ内のジョブ実行ログファイルの書き込みに失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0703-E

スプールジョブディレクトリ内にファイル（"**ファイル名**"）が存在しません。

スプールジョブディレクトリ内に**ファイル名**で示されるファイルがありません。

(S)

処理を終了します。

(O)

ファイル名を見直してください。

KNAX0704-E

日付の取得に失敗しました。

日付の取得に失敗しました。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX0706-E

ファイル（"**ファイルパス**"）の作成に失敗しました。reason=**エラー詳細**

ファイルパスの作成に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0708-E

割り当てられる JOBLOG ファイル数の上限を超えました。

割り当てられる JOBLOG ファイル数の上限を超えました。次の原因が考えられます。

- スプールディレクトリ下にスプールジョブが大量に残っていて，ジョブ識別子の空きが少ない。
- ジョブ内で子孫ジョブを大量に起動している。

(S)

処理を終了します。

(O)

不要なスプールジョブを削除してジョブを再実行してください。または，不当に大量の子孫ジョブを起動していないか，ジョブ定義スクリプトを見直してください。

KNAX0719-I

STEP ステップ番号=**ステップ番号** ステップ名=**ジョブステップ名** 出力先=**出力先**

このメッセージのあとに，ジョブステップの**出力先**を出力します。

(S)

処理を続行します。

KNAX0720-E

ジョブ ID ファイルのオープンに失敗しました。reason=**エラー詳細**

ジョブ ID ファイルのオープンに失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者は環境ファイルのスプールディレクトリの指定，スプールディレクトリ自体，またはスプールディレクトリ内のファイルに異常がないか見直してください。

KNAX0721-E

ジョブ ID ファイルの入出力に失敗しました。reason=**エラー詳細**

ジョブ ID ファイルの入出力に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者は環境ファイルのスプールディレクトリの指定，スプールディレクトリ自体，またはスプールディレクトリ内のファイルに異常がないか見直してください。

KNAX0722-E

ジョブ固有のジョブ識別子を付与できません。

ジョブ固有のジョブ識別子を付与できません。スプールジョブディレクトリがこれ以上作成できない場合，このメッセージが出力されることがあります。

(S)

処理を終了します。

(O)

システム管理者に連絡します。システム管理者は前後に出力されたメッセージから原因を取り除いて再実行してください。または，不要なスプールジョブディレクトリを削除して，再実行してください。

KNAX0723-E

ジョブ ID ファイルの排他に失敗しました。reason=**エラー詳細**

ジョブ ID ファイルを排他しようとして，失敗しました。

SPOOL_DIR パラメーターに NFS 上のディレクトリを指定すると，このメッセージを出力してエラーとなることがあります。SPOOL_DIR パラメーターには，NFS 上のディレクトリを指定しないでください。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者は環境ファイルのスプールディレクトリの指定，スプールディレクトリ自体，またはスプールディレクトリ内のファイルに異常がないか見直してください。

KNAX0724-I

ジョブ識別子を割り当てました。Jobid=**ジョブ識別子**

ジョブ識別子を割り当てました。

ルートジョブの環境ファイルの SPOOLJOB_CHILDJOB パラメーターにオペランドとして MERGE を指定して起動された子孫ジョブの場合は出力されません。

(S)

処理を続行します。

KNAX0725-E

API でエラーが発生しました。("**API名称**", "**原因**", "**保守情報**")

API でエラーが発生しました。

(S)

処理を終了します。

(O)

原因に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。

KNAX0726-I

子孫ジョブのジョブ識別子を割り当てました。Jobid=**ジョブ識別子**

子孫ジョブのジョブ識別子を割り当てました。

(S)

処理を続行します。

KNAX0727-E

子孫ジョブ起動順序管理ファイルの排他に失敗しました。reason=**エラー詳細**

子孫ジョブ起動順序管理ファイルを排他しようとしましたが，失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。

システム管理者は，スプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないかどうかを見直してください。

KNAX0728-E

子孫ジョブ起動順序管理ファイルの形式不正を検出しました。

子孫ジョブ起動順序管理ファイルの形式不正を検出しました。

(S)

処理を終了します。

(O)

子孫ジョブ起動順序管理ファイルを不当に更新していないか見直してください。問題が解決しない場合は，システム管理者に連絡してください。

システム管理者は，スプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないかどうかを見直してください。

KNAX0800-E

スプールジョブ管理ファイルの作成に失敗しました。reason=**エラー詳細**

スプールジョブ管理ファイルを作成しようとして，失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0801-E

スプールジョブ管理ファイルの排他に失敗しました。reason=**エラー詳細**

スプールジョブ管理ファイルを排他しようとして，失敗しました。

SPOOL_DIR パラメーターに NFS 上のディレクトリを指定すると，このメッセージを出力してエラーとなることがあります。SPOOL_DIR パラメーターには，NFS 上のディレクトリを指定しないでください。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0802-E

スプールジョブ管理ファイルのオープンに失敗しました。reason=**エラー詳細**

スプールジョブ管理ファイルをオープンしようとして，失敗しました。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0803-E

スプールジョブ管理ファイルで入出力エラーが発生しました。reason=**エラー詳細**

スプールジョブ管理ファイルで入出力エラーが発生しました。

このメッセージは，エラーの通知先に出力しますが，エラー発生のタイミングによって，一部の出力先にだけ出力されることがあります。

(S)

処理を終了します。

(O)

エラー詳細に示される原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡してください。システム管理者はスプールジョブディレクトリ，またはスプールジョブディレクトリ内のファイルに異常がないか見直してください。

KNAX0804-E

スプールジョブ管理ファイルの STARTTIME に使用する現在時刻の取得に失敗しました。

スプールジョブ管理ファイルの STARTTIME に使用する現在時刻の取得に失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。

KNAX0805-E

スプールジョブ管理ファイルの ENDTIME に使用する現在時刻の取得に失敗しました。

スプールジョブ管理ファイルの ENDTIME に使用する現在時刻の取得に失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。

KNAX1600-I

ジョブ名 バッチジョブのファイル割り当てを開始しました。

ジョブ名で示すバッチジョブのファイルが割り当てられました。

(S)

処理を続行します。

KNAX1601-I

ジョブ名.ジョブステップ名 ジョブステップのファイル割り当てを開始しました。

ジョブ名と**ジョブステップ名**で示すジョブステップのファイルが割り当てられました。

(S)

処理を続行します。

KNAX1604-I

後処理指定値によってファイル（**ファイルパス**）を削除しました。

後処理指定値によって**ファイルパス**に示すファイルを削除しました。

(S)

処理を続行します。

KNAX1605-I

後処理指定値によってファイル（**ファイルパス**）を削除しようとしましたが，エラーが発生しました。reason="**エラー詳細**"

後処理指定値によって**ファイルパス**を削除しようとしましたが，**エラー詳細**に示すエラーが発生しました。

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列。

(S)

処理を続行します。

KNAX1632-E

環境変数（**環境変数名**）の値が長すぎます。

環境変数名で示す名称の環境変数を設定しようとして，値の長さが JP1/Advanced Shell で定められた上限値を超えました。

(S)

処理を終了します。

(O)

環境ファイルで指定した環境変数値を修正して，バッチジョブを再実行します。

KNAX1871-E

コマンド（**コマンド名**）で指定されたファイルパスを絶対パスに変換して正規化する処理中にエラーが発生しました。("**関数名**", "**原因**", "**保守情報**")

コマンド名で示すコマンドで，指定されたファイルパスを絶対パスに変換して正規化する処理中にエラーが発生しました。

関数名が_fullpath の場合，コマンドで指定したパス名が長過ぎる場合に発生することがあります（このとき**原因**は Invalid argument となることがあります）。

原因

表示される内容とその意味を次に示します。

原因	意味
パス名に不当なマルチバイト文字が含まれています。	【UNIX 限定】パス名に不当なマルチバイト文字が存在します。コマンドに指定したファイルパスの指定を見直してください。
パス構成要素が多過ぎます。	【UNIX 限定】パス名の構成要素が 4,096 を超えました。コマンドに指定したファイルパスを見直してください。
File name too long	【UNIX 限定】絶対パスへ変換したあとのファイルパス名長が上限を超えました。ファイルパスの指定を見直してください。
errnoから求めたエラー理由	• メモリ不足が発生しました。 • その他 OS の API 実行時にエラーが発生しました（getcwd，_fullpath でエラー）。 • 【Windows 限定】関数名が「_fullpath」，原因が「Invalid argument」の場合は，絶対パスへ変換したあとのファイルパス名長が上限を超えたことが考えられます。この場合は，コマンドに指定したファイルパスの指定を見直してください。

保守情報

内部情報

(S)

処理を終了します。

(O)

関数名と**原因**を基にエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX1872-E

コマンド（**コマンド名**）で指定されたファイルパスのチェックでエラーが発生しました。("**ファイルパス**", "**原因**", "**保守情報**")

コマンド名で示すコマンドの処理中に，指定した**ファイルパス**のチェックでエラーが発生しました。コマンドに指定したファイルパスが使用できるか見直してください。

原因

システムが通知するエラー理由が表示されます。

保守情報

内部情報

(S)

処理を終了します。

(O)

エラー要因を取り除いて，コマンドを再実行してください。

エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX1873-E

コマンド（**コマンド名**）で指定されたファイルパスは，ディレクトリです。("**ファイルパス**", "**保守情報**")

コマンド名で示すコマンドで指定した**ファイルパス**はディレクトリです。

ファイルパスには通常のファイルを指定してください。

保守情報

内部情報

(S)

処理を終了します。

(O)

エラー要因を取り除いて，コマンドを再実行してください。

KNAX1875-E

コマンド（**コマンド名**）の処理中に予期しないエラーが発生しました。("**原因**", "**保守情報**")

コマンド名で示すコマンドの処理中に予期しないエラーが発生しました。

原因

表示される内容とその意味を次に示します。

原因	意味
パス名を格納する領域が不足しました。	【UNIX 限定】パス名を格納する領域が不足しました。
_time64 関数でエラーが発生しました。	【Windows 限定】時刻を求める処理でエラーが発生しました。
clock_gettime 関数でエラーが発生しました。	【UNIX 限定】時刻を求める処理でエラーが発生しました。
nanosleep 関数でエラーが発生しました。	【UNIX 限定】待ち処理でエラーが発生しました。
Invalid argument	引数が不正です。

保守情報

内部情報

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX1877-E

コマンド（**コマンド名**）が使用する環境変数，または環境変数の値が定義されていません。("**環境変数名**", "**保守情報**")

環境変数名に示す環境変数が存在しません。または，設定値の文字列長が 0 バイト以下です。

保守情報

内部情報

(S)

処理を終了します。

(O)

該当する環境変数を変更していないかジョブ定義スクリプトを確認してください。または，コマンドをジョブコントローラ以外で起動していないか確認してください。

原因が不明の場合はシステム管理者に連絡してください。

KNAX1878-E

コマンド（**コマンド名**）が使用する割り当て管理ファイルの入出力でエラーが発生しました。("**ファイルパス**", "**原因**", "**保守情報**")

割り当て管理ファイルの入出力でエラーが発生しました。

ファイルパス

割り当て管理ファイルのファイルパス名が表示されます。

原因

表示される内容とその意味を次に示します。

原因	意味
通常ファイルではありません。	通常ファイルではありません。
ファイルが置き換わりました。	オープン処理中にファイルが置き換わりました。
api名 error : **エラー詳細**	**api名**で示す API の処理で，**エラー詳細**に示すエラーが発生しました。 **エラー詳細**は API でセットされるエラー情報です。

保守情報

内部情報

(S)

処理を終了します。

(O)

エラー要因を取り除いて再実行してください。

原因が不明の場合はシステム管理者に連絡してください。

KNAX1879-E

コマンド（**コマンド名**）が登録できるファイル数の上限を超えました。("**保守情報**")

登録できるファイル数の上限である 64 個を超えました。

保守情報

内部情報

(S)

処理を終了します。

(O)

登録するファイルの数が上限数を超えないよう，ジョブ定義スクリプトを見直してください。

KNAX1880-E

コマンド（**コマンド名**）は現在の環境設定パラメーターの設定では使用できません。parameter="**環境設定パラメーター**"

メッセージに表示されたコマンドは，現在の環境設定パラメーターの設定では使用できません。

コマンド名

コマンド名

環境設定パラメーター

要因となった環境設定パラメーターとその値

(S)

処理を終了します。

(O)

実行したコマンドは使用しないでください。このコマンドを使用したい場合は，環境設定パラメーターの指定を見直してください。

KNAX1890-I

ファイル割り当て処理指定（"**処理指定値**"）に従って解放しました。path=**ファイルパス**

ファイルパスに示すファイルを**処理指定値**に従って解放しました。

処理指定値

正常時・異常時のファイルの後処理の指定内容として，次のどちらかが表示されます。

・del：削除する

・keep：削除しない

(S)

処理を続行します。

KNAX1891-E

ファイルの後処理実行中に，割り当て管理ファイルの入出力でエラーが発生しました。("**ファイルパス**", "**原因**", "**保守情報**")

ファイルの後処理実行中に，割り当て管理ファイルの入出力でエラーが発生しました。

ファイルパス

割り当て管理ファイルのファイルパス名が表示されます。

原因

表示される内容とその意味を次に示します。

原因	意味
通常ファイルではありません。	通常ファイルではありません。
ファイルが置き換わりました。	オープン処理中にファイルが置き換わりました。
api名 error : **エラー詳細**	**api名**で示す API の処理で，**エラー詳細**に示すエラーが発生しました。 **エラー詳細**は API でセットされるエラー情報です。

保守情報

内部情報

(S)

ジョブは終了します。

(O)

システム管理者に連絡してください。

KNAX1892-E

予期しないエラーが発生しました。("**原因**", "**保守情報**")

予期しないエラーが発生しました。

原因

表示される内容とその意味を次に示します。

原因	意味
_time64 関数でエラーが発生しました。	【Windows 限定】時刻を求める処理でエラーが発生しました。
clock_gettime 関数でエラーが発生しました。	【UNIX 限定】時刻を求める処理でエラーが発生しました。
nanosleep 関数でエラーが発生しました。	待ち処理でエラーが発生しました。

保守情報

内部情報

(S)

ジョブは終了します。

(O)

システム管理者に連絡してください。

KNAX1893-W

割り当て管理ファイル中に無効なエントリが存在したため，後処理をスキップしました。("**ファイルパス**", "**保守情報**")

割り当て管理ファイル中に無効なエントリが存在したため，adshfile コマンドの後処理をスキップしました。該当するジョブの adshfile コマンドで指定したファイルの後処理が実行されていないおそれがあります。

ファイルパス

割り当て管理ファイルのファイルパス名が表示されます。

保守情報

内部情報

(S)

処理を続行します。

(O)

adshfile コマンドで削除を指定したファイルが残っている場合，必要に応じて手作業で削除してください。

KNAX1910-E

C-Time の計算結果が不当な結果となりました。

C-Time の計算結果が不当な結果となりました。

(S)

ジョブ実行ログへ出力する時刻を 0 とし，バッチジョブは続行します。

(O)

システム管理者に連絡します。

KNAX1911-E

E-Time の計算結果が不当な結果となりました。

E-Time の計算結果が不当な結果となりました。

(S)

ジョブ実行ログへ出力する時刻を 0 とし，バッチジョブは続行します。

(O)

システム管理者に連絡します。

KNAX2201-E

テキストが長いためすべてのテキストを出力できません。message number=**メッセージ番号**

メッセージ番号（KNAX に続く番号）で示すメッセージは，テキストが長いためすべてのテキストを出力できません。

(S)

処理を続行します。

(O)

バッチジョブの動作に問題がないかどうかを確認します。必要な場合，ジョブ定義スクリプトを修正します。

KNAX2202-E

ジョブ実行ログへの出力に失敗しました。message ID=**メッセージID**

メッセージID で示すメッセージは，ジョブ実行ログへの出力を失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。スプールジョブディレクトリのアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2204-E

標準出力への出力に失敗しました。message ID=**メッセージID**

メッセージID で示すメッセージは，標準出力への出力に失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。標準出力の出力先のアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2205-E

標準エラー出力への出力に失敗しました。message ID=**メッセージID**

メッセージID で示すメッセージは，標準エラー出力への出力に失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。標準エラー出力の出力先のアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2206-E

システム実行ログへの出力に失敗しました。message ID=**メッセージID**

メッセージID で示すメッセージは，システム実行ログへの出力に失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。環境ファイルのシステム実行ログに対する指定，またはシステム実行ログディレクトリのアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2207-E

トレースへの出力に失敗しました。message ID=**メッセージID**

メッセージID で示すメッセージは，トレースへの出力に失敗しました。

(S)

処理を続行します。

(O)

システム管理者に連絡します。環境ファイルのトレースに対する指定，またはトレースディレクトリのアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2208-E

システム実行ログの初期化に失敗しました。code=**保守情報**, reason=**エラー原因**

システム実行ログの初期化に失敗しました。

LOG_DIR パラメーターに NFS 上のディレクトリを指定すると，このメッセージを出力してエラーとなることがあります。LOG_DIR パラメーターには，NFS 上のディレクトリを指定しないでください。

(S)

処理を終了します。

(O)

システム管理者に連絡します。環境ファイルのシステム実行ログに対する指定，またはシステム実行ログディレクトリのアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2209-E

トレースの初期化に失敗しました。code=保守情報

トレースの初期化に失敗しました。

TRACE_DIR パラメーターに NFS 上のディレクトリを指定すると，このメッセージを出力してエラーとなることがあります。TRACE_DIR パラメーターには，NFS 上のディレクトリを指定しないでください。

(S)

処理を終了します。

(O)

システム管理者に連絡します。環境ファイルのトレースに対する指定，またはトレースディレクトリのアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2211-E

トレースの設定に誤りがあります。

トレースの設定に誤りがあります。

(S)

処理を終了します。

(O)

システム管理者に連絡します。環境ファイルのトレースに対する指定を見直してください。

KNAX2213-E

トレースの出力先の指定に誤りがあります。

トレースの出力先の指定に誤りがあります。

(S)

処理を終了します。

(O)

システム管理者に連絡します。環境ファイルのトレースに対する指定，またはトレースディレクトリのアクセス権やディスクの状態などに異常がないかどうかを見直してください。

KNAX2214-E

現在時刻の取得に失敗しました。

現在時刻の取得に失敗しました。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX2400-E

初期化されていないためメッセージを出力できません。output=**出力先**, message ID=**メッセージID**, dest=**保守情報1**, setDest=**保守情報2**

メッセージID に示すメッセージを出力しましたが，初期化されていないため出力できません。

(S)

処理を続行します。

(O)

システム管理者に連絡します。

KNAX2499-E

未定義メッセージを出力しようとしました。message number="**メッセージ番号**"

メッセージ番号で示すメッセージはマニュアルにありません。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX2500-E

コマンド名コマンドの引数の数が不当です。[filename="**ファイル名**" line=**行番号**]

コマンド名で示すコマンドの引数の数が不当です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX2501-E

データの解析でエラーが発生しました。[filename="**ファイル名**" line=**行番号**]

データの解析でエラーが発生しました。次の原因が考えられます。

CSVデータの場合

ダブルクォーテーションが閉じていません。

JSONデータの場合

- 入力データの形式が JSON の形式と一致していません。
- ダブルクォーテーションが閉じていません。
- ダブルクォーテーションで囲む必要のある値がダブルクォーテーションで囲まれていません。

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルまたはデータの内容を見直し，修正してください。

KNAX3000-I

ユーザー応答機能管理デーモンが起動しました。

ユーザー応答機能管理デーモンが起動しました。

(S)

処理を続行します。

KNAX3001-I

ユーザー応答機能管理デーモンが停止しました。

ユーザー応答機能管理デーモンが停止しました。

(S)

処理を終了します。

KNAX3002-E

ユーザー応答機能管理デーモンの起動に失敗しました。[**詳細メッセージ**]

ユーザー応答機能管理デーモンの起動に失敗しました。

このメッセージは，エラー通知として出力できる出力先に出力されますが，エラー発生のタイミングによっては，一部の出力先にだけ出力されることがあります。syslog への出力時には，[**詳細メッセージ**]にユーザー応答機能管理デーモンに関連するメッセージ（メッセージ番号 3000～3999）のメッセージテキストが付与されることがあります。

(S)

処理を終了します。

(O)

エラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行してください。

KNAX3003-E

ユーザー応答機能管理デーモンがエラー終了しました。[**詳細メッセージ**]

ユーザー応答機能管理デーモンがエラー終了しました。または，終了処理中にエラーが発生しました。syslog への出力時には，[**詳細メッセージ**]にユーザー応答機能管理デーモンに関連するメッセージ（メッセージ番号 3000～3999）のメッセージテキストが付与されることがあります。

(S)

処理を終了します。

(O)

エラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行してください。

KNAX3006-I

ユーザー応答機能管理デーモンが起動を開始しました。(pid=**PID**, uid=**UID**, gid=**GID**, username=**ユーザー名**)

ユーザー応答機能管理デーモンが起動を開始しました。

(S)

処理を終了します。

KNAX3008-W

ユーザー応答機能管理デーモンはプロセス ID=**PID** で動作していました。

前回，ユーザー応答機能管理デーモンがエラー終了したときのプロセス ID（**PID**）を表示します。

(S)

処理を続行します。

KNAX3009-E

ユーザー応答機能管理デーモンはすでに起動しています。

ユーザー応答機能管理デーモンはすでに起動しています。

このメッセージは，エラー通知として出力できる出力先に出力されますが，エラー発生のタイミングによって，一部の出力先にだけ出力されることがあります。

(S)

処理を終了します。

(O)

必要に応じて起動中のユーザー応答機能管理デーモンを終了し，再起動します。

KNAX3020-E

ユーザー応答機能管理デーモンでファイル操作エラーが発生しました。(**機能名** error **対象名** - **エラー詳細**)

ユーザー応答機能管理デーモンでファイル操作エラーが発生しました。

機能名，**対象名**，および**エラー詳細**は，エラー情報を示します。

このメッセージは，エラー通知として出力できる出力先に出力されますが，エラー発生のタイミングによって，一部の出力先にだけ出力されることがあります。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

KNAX3023-E

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。(**機能名** error **対象名** - **エラー詳細**)

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。

機能名，**対象名**，および**エラー詳細**は，エラー情報を示します。

このメッセージは，エラー通知として出力できる出力先に出力されますが，エラー発生のタイミングによっては，一部の出力先にだけ出力されることがあります。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

KNAX3024-E

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。(**機能名** error **対象名**)

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。

機能名，および**対象名**は，エラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

KNAX3025-E

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。(**機能名** error - **エラー詳細**)

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。

機能名，および**エラー詳細**は，エラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

KNAX3026-E

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。(**機能名** error)

ユーザー応答機能管理デーモンでシグナルエラーが発生しました。

機能名は，エラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

KNAX3027-E

ユーザー応答機能管理デーモンでプロセスエラーが発生しました。(**機能名** error **対象名** - **エラー詳細**)

ユーザー応答機能管理デーモンでプロセスエラーが発生しました。

機能名，**対象名**，および**エラー詳細**は，エラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

機能名はシステムの関数名を示します。システム関数のエラーメッセージからエラーの原因を調査してください。

KNAX3029-E

ユーザー応答機能管理デーモンでシステム関数エラーが発生しました。(**機能名** error - **エラー詳細**)

ユーザー応答機能管理デーモンでシステム関数エラーが発生しました。

機能名，および**エラー詳細**は，エラー情報を示します。

このメッセージは，エラー通知として出力できる出力先に出力されますが，エラー発生のタイミングによっては，一部の出力先にだけ出力されることがあります。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行します。

KNAX3261-I

ユーザー応答機能管理デーモンが終了要求のシグナル（**シグナル名**）を受信しました。

ユーザー応答機能管理デーモンが**シグナル名**で示す終了要求のシグナルを受信しました。

(S)

処理を続行します。

KNAX3400-I

環境ファイルのチェックが正常に終了しました。

環境ファイルのチェックが正常に終了しました。

(S)

処理を続行します。

KNAX3402-E

環境ファイルのチェックでエラーが発生しました。

環境ファイルのチェックでエラーが発生しました。

エラーの内容はこのメッセージの前に出力されているメッセージを参照してください。

(S)

処理を終了します。

(O)

エラーの原因を取り除きます。

KNAX3508-I

ユーザー応答機能管理デーモンが adshread コマンドによる応答要求メッセージをキャンセルしました。(**ジョブ識別子**, **行番号**, **ホスト名**)

ユーザー応答機能管理デーモンが adshread コマンドによる応答要求メッセージをキャンセルしました。

ジョブ識別子

JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

行番号

adshread コマンドを発行したジョブ定義スクリプトの行番号

ホスト名

ユーザー応答機能管理デーモンが稼働しているホストの名称

(S)

処理を続行します。

KNAX3522-E

ユーザー応答機能管理デーモンで共有メモリエラーが発生しました。(**機能名** error - **エラー詳細**)

ユーザー応答機能管理デーモンで共有メモリエラーが発生しました。

機能名，および**エラー詳細**は，エラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査してエラーの原因を取り除き，ユーザー応答機能管理デーモンを再実行してください。

機能名はシステムの関数名を示します。システム関数のエラーメッセージからエラーの原因を調査してください。

KNAX3542-W

ユーザー応答機能管理デーモンはすでに作成されている共有メモリまたはセマフォ（**対象名**）を初期化して使用します。

ユーザー応答機能管理デーモンに-f オプションが指定されたため，すでに作成されている共有メモリまたはセマフォを初期化して使用します。

対象名は，Shared memory object または Named semaphore です。

(S)

すでに作成されている共有メモリまたはセマフォを初期化して処理を続行します。

KNAX3700-I

The adshmd will now start.

ユーザー応答機能管理デーモンを起動します。

(S)

処理を続行します。

KNAX3701-I

The adshmd will now stop.

ユーザー応答機能管理デーモンを終了します。

(S)

処理を続行します。

KNAX3703-E

The adshmd is not running.

ユーザー応答機能管理デーモンが動作していません。

(S)

処理を終了します。

(O)

ユーザー応答機能管理デーモンが起動されているかどうかを確認してください。

KNAX3709-E

The adshmd could not start because another one is already running.

ユーザー応答機能管理デーモンはすでに動作しています。

(S)

処理を終了します。

(O)

必要に応じて起動中のユーザー応答機能管理デーモンを終了し，再起動してください。

KNAX3710-I

The adshmd is running.

ユーザー応答機能管理デーモンは動作しています。

(S)

処理を終了します。

KNAX3711-I

The adshmd is not running.

ユーザー応答機能管理デーモンは動作していません。

(S)

処理を終了します。

KNAX3799-I

Usage **コマンド名** [-h LogicalHostName] {start [reuse]|stop|status|conftest [EnvironFile]|help}

adshmdctl コマンドの使用方法を表示します。

(S)

処理を終了します。

KNAX3998-E

An error occurred during adshmd signal handler processing.

ユーザー応答機能管理デーモンでシグナルハンドラがエラー終了しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX3999-E

The adshmd ended abnormally because of an unexpected exception.

ユーザー応答機能管理デーモンがエラー終了しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX4414-E

対象リストファイルに記述したスプールディレクトリのパス名の長さが正しくありません。filename="**ファイル名**" line=**行番号**

指定したスプールディレクトリのパス名の長さが正しくありません。

ファイル名

対象リストファイル名

行番号

エラーが発生した対象リストファイルの行番号

(S)

後続行の指定の処理を継続します。

(O)

対象リストファイルに指定したスプールディレクトリ名に誤りがないかどうかを見直して再実行します。

KNAX4415-E

対象リストファイルの記述形式が正しくありません。filename="**ファイル名**" line=**行番号**

対象リストファイルの記述形式が正しくありません。または，コマンドの引数に日数の指定がないときに対象リストファイルに日数の指定がありません。

ファイル名

対象リストファイル名

行番号

エラーが発生した対象リストファイルの行番号

(S)

後続行の指定の処理を継続します。

(O)

対象リストファイルの指定が正しいかどうかを見直します。または，コマンドの引数に日数を指定します。

KNAX4416-E

対象リストファイルの 1 行の上限を超えています。filename="**ファイル名**" line=**行番号**

対象リストファイルの 1 行の上限を超えています。

ファイル名

対象リストファイル名

行番号

エラーが発生した対象リストファイルの行番号

(S)

処理を中断します。

(O)

対象リストファイルの指定が正しいかどうかを見直します。

KNAX4417-E

対象リストファイルの実体がオープン前とオープン後で異なります。filename="**ファイル名**"

オープン前とオープン後でファイルの実体が異なります。

ファイル名

ファイル名

(S)

該当するファイルの処理は中断します。

(O)

ファイルに異常がないかどうかを確認します。

KNAX4418-E

対象リストファイルは通常のファイルではありません。filename="**ファイル名**"

ファイルは通常のファイルでありません。

ファイル名

ファイル名

(S)

該当するファイルの処理は中断します。

(O)

ファイルに異常がないかどうかを確認します。

KNAX4419-E

ファイルの入出力処理で障害が発生しました。path="**パス名**" error="**エラー詳細**"

ファイルの入出力処理で障害が発生しました。

パス名

パス名

このメッセージが出力される理由の一つとして，スプールのディレクトリの下に，ユーザーが独自に作成したファイルやディレクトリなど，JP1/Advanced Shell が認識できないファイルが存在する可能性があります。

(S)

該当するパスに対する処理を中断します。

(O)

該当するパスに異常がないかどうかを確認します。

KNAX4420-E

予期しない障害が発生しました。errinfo="**エラー詳細**, **内部情報**"

予期しない障害が発生しました（ログファイルまたはトレースファイルのオープン処理に失敗，時刻処理で障害発生）。

(S)

スプールジョブ処理時はそのスプールジョブの処理を中断します。そのほかの場合はコマンドを終了します。

(O)

動作環境に問題ないかどうかを確認します。

KNAX4422-E

コマンドに必須のオペランド（**対象リストファイル名**｜**レポートファイル名**｜**ログファイル名**）が指定されていません。

コマンドに必須のオペランドが指定されていません。

対象リストファイル名

対象リストファイル

レポートファイル名

レポートファイル

ログファイル名

ログファイル

(S)

コマンドを終了します。

(O)

必須のオペランドを指定してコマンドを再実行します。

KNAX4423-E

コマンドに指定した日数の形式が正しくありません。

コマンドに指定した日数の形式が正しくありません。

(S)

コマンドを終了します。

(O)

正しく指定してコマンドを再実行します。

KNAX4424-E

バッチジョブの実行開始日付が求まらないため，スプールジョブは削除されていません。path="**スプールジョブディレクトリ名**"

バッチジョブの実行開始日付が求まらないため，スプールジョブは削除されていません。スプールジョブを管理するファイルが破壊されている場合があります。

スプールジョブディレクトリ名

異常があったスプールジョブディレクトリ名

(S)

該当するスプールジョブは削除しないで，処理を続行します。

(O)

必要な場合，手作業で該当するスプールジョブを削除します。

KNAX4425-E

別のプログラムで使用中のため処理できませんでした。path="**スプールディレクトリ名**"

スプールジョブディレクトリ名で示すスプールディレクトリは，別のプログラムで使用中のため処理できませんでした。

(S)

該当するスプールジョブを処理しないで，処理を続行します。

(O)

時間を置いて，該当するスプールディレクトリが別のプログラムで処理中でないことを確認し，再実行してください。

KNAX4427-W

不当なスプールジョブディレクトリが存在しましたが，そのディレクトリの処理をスキップしました。

不当なスプールジョブディレクトリが存在しましたが，そのディレクトリは処理をスキップしました。

実行が完了したジョブのスプールジョブディレクトリ名が，次の形式になっていません。

- *ジョブ識別子-スプールジョブ名*
- *ジョブ識別子-*

このメッセージは，スプールディレクトリ下に次の条件に該当する不当な名称のディレクトリまたはファイルが存在した場合に出力されます。

- 名称の長さが５バイト以下である。
- 名称の７バイト目が「-」でない。

ただし，JP1/Advanced Shell が管理するファイル（UNIX では.jobid，Windows ではadsh.jobid）は該当しません。また，ジョブ識別子だけ（名称の長さが６バイト）の場合は実行中のジョブのため，該当しません。

(S)

処理を継続します。

(O)

不当なディレクトリまたはファイルは手作業で削除してください。

KNAX4428-I

スプールジョブを削除しました。path="**パス名**"

スプールジョブを削除しました。

パス名

スプールジョブディレクトリ名

(S)

処理を継続します。

KNAX4429-E

コマンド実行中にエラーが発生しました。

コマンド実行中にエラーが発生しました。

(S)

処理を継続します。

(O)

ログファイル，レポートファイルまたは標準エラー出力に出力されたメッセージを見て，エラー内容を確認します。必要に応じて障害を取り除いてコマンドを再実行します。

KNAX5300-I

Usage: **コマンド名** [-jbspglogicalhost LogicalHostName]

アダプタコマンドの引数が誤っています。

(S)

処理を終了します。

KNAX5301-E

No value is specified for the option "**オプション名**".

アダプタコマンドのオプションの指定値が誤っています。

オプション名

アダプタコマンドのオプション名

(S)

処理を終了します。

KNAX5305-E

The specified argument "**引数**" is invalid.

アダプタコマンドに不正な**引数**を指定しました。

引数

アダプタコマンドのオプション名

(S)

処理を終了します。

KNAX5308-E

An API error occurred. (maintenance information=**保守情報**, details=**エラー詳細**)

アダプタコマンドで API エラーが発生しました。

保守情報および**エラー詳細**はエラー情報を示します。

(S)

処理を終了します。

(O)

保守情報が sem_open の場合は，次のどちらかの環境不正が考えられます。

- ユーザー応答機能管理デーモン・サービスが起動されていない。
  ユーザー応答機能管理デーモン・サービスを起動してください。
  起動できない要因が判明している場合は，その要因を取り除いて再起動してください。
  起動できない要因が不明な場合は，システム管理者に連絡してください。
- ユーザー応答機能管理サービスが登録されていない（Windows の場合）。
  サービスの登録手順に従ってユーザー応答機能管理サービスを登録し起動してください。
  サービスが登録できないか，登録はできたが起動ができない場合は，要因が判明しているときはその要因を取り除き，登録または起動してください。
  登録できない，または起動できない要因が不明な場合，システム管理者に連絡してください。

保守情報が sem_open 以外の場合は，システム管理者に連絡してください。

KNAX5309-E

An internal error occurred.

内部エラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5323-E

A signal error occurred. (function=**機能名**, target=**対象名**, details=**エラー詳細**)

アダプタコマンドでシグナルエラーが発生しました。

機能名，**対象名**，および**エラー詳細**はエラー情報を示します。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5340-E

You do not have permission to execute the command **コマンド名**.

アダプタコマンドに対する実行権限がありません。

このコマンドは，Administrators 権限のユーザーで実行する必要があります。

(S)

処理を終了します。

(O)

アダプタコマンドは，JP1/Base のプラグインサービスから起動されるプログラムです。プラグインサービスとして起動されてこのメッセージが出力された場合は，JP1/Base の設定を見直してください。

KNAX5350-E

The request header is invalid.

アダプタコマンドに渡されるリクエストヘッダが不正です。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5360-E

The request data is invalid.

アダプタコマンドに渡されるリクエストデータが不正です。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5361-E

Failed to get the identifier.

アダプタコマンドに渡されるリクエストデータ中のインジケータの取得に失敗しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5362-E

The response contains one or more non-ASCII characters.

JP1/IM - View の［応答入力］画面から応答入力した内容に ASCII 文字以外のデータが含まれています。

(S)

処理を終了します。

(O)

応答入力の内容に ASCII 文字列を指定して再度応答してください。

KNAX5371-E

The userreply function is busy.

JP1/Advanced Shell の処理がビジーです。

(S)

処理を終了します。

(O)

時間を置いて再度応答要求メッセージに応答してください。

KNAX5372-E

The message is not found.

応答要求メッセージが存在しません。次の要因が考えられます。

- ユーザー応答機能管理デーモン・サービスが起動していません。
- JP1/Base の論理ホストに関する設定に誤りがあります。

なお，adshchmsg コマンドで応答待ちメッセージに代理応答した場合，タイミングによってはこのメッセージが出力されることがあります。この場合，対処は不要です。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5380-I

The following data was received: **受信データ**

ユーザー応答機能で，JP1 からメッセージを受信しました。

(S)

処理を続行します。

KNAX5381-I

The following information was sent: **送信データ**

ユーザー応答機能で，JP1 にメッセージを送信しました。

(S)

処理を続行します。

KNAX5396-I

adshuserreply.adapter completed because signal is detected.

アダプタコマンドが終了シグナルを受信して終了しました。

(S)

処理を終了します。

KNAX5397-I

Signal handler processing completed.

アダプタコマンドがシグナルを受信しました。

(S)

処理を終了します。

(O)

core が出力されています。システム管理者に連絡してください。

KNAX5398-E

An error occurred during adshuserreply.adapter signal handler processing.

アダプタコマンドがシグナルを受信しましたが，シグナルハンドラの処理でエラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5399-E

The adshuserreply.adapter ended abnormally because of an unexpected exception.

アダプタコマンドがエラー終了しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5407-E

応答内容（-r オプションに指定した値）に ASCII 文字以外を指定しています。

adshchmsg コマンドの-r オプション（応答内容）に ASCII 文字以外を指定しています。

(S)

処理を終了します。

(O)

adshchmsg コマンドの-r オプションに ASCII 文字列を指定して，再度応答してください。

KNAX5409-E

指定された応答要求メッセージは存在しません。

adshchmsg コマンドで，-n オプションに指定した応答要求メッセージ番号に対する応答入力，または，応答要求メッセージのキャンセルを行おうとしましたが，指定された応答要求メッセージは存在しません。

(S)

処理を終了します。

(O)

次の点を確認してください。

- -n オプションに指定した応答要求メッセージ番号に誤りがないか。
- -n オプションに指定した応答要求メッセージ番号が adshlsmsg コマンドで表示されているか。

なお，指定した番号に誤りがなく，adshlsmsg コマンドを実行しても表示されない場合は，すでに応答入力が行われている可能性があります。

KNAX5410-E

API エラーが発生しました。(**保守情報** error - **エラー詳細**)

adshchmsg コマンドまたは adshlsmsg コマンドで，API エラーが発生しました。

保守情報および**エラー詳細**はエラー情報を示します。

(S)

処理を終了します。

(O)

保守情報が sem_open の場合は，次のどちらかの環境不正が考えられます。

- ユーザー応答機能管理デーモン・サービスが起動されていない。
  ユーザー応答機能管理デーモン・サービスを起動してください。

起動できない要因が判明している場合は，その要因を取り除いて再起動してください。

起動できない要因が不明な場合は，システム管理者に連絡してください。

- ユーザー応答機能管理サービスが登録されていない（Windows の場合）。

  サービスの登録手順に従ってユーザー応答機能管理サービスを登録し起動してください。

  サービスが登録できないか，登録はできたが起動ができない場合は，要因が判明しているときはその要因を取り除き，登録または起動してください。

  登録できない，または起動できない要因が不明な場合，システム管理者に連絡してください。

保守情報が sem_open 以外の場合は，システム管理者に連絡してください。

KNAX5423-E

シグナルエラーが発生しました。(**機能名** error **対象名** - **エラー詳細**)

adshchmsg コマンドまたは adshlsmsg コマンドで，シグナルエラーが発生しました。

機能名，**対象名**，および**エラー詳細**はエラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査して原因を取り除き，再度実行してください。

KNAX5424-E

シグナルエラーが発生しました。(**機能名** error **対象名**)

adshchmsg コマンドまたは adshlsmsg コマンドで，シグナルエラーが発生しました。

機能名および**対象名**はエラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査して原因を取り除き，再度実行してください。

KNAX5425-E

シグナルエラーが発生しました。(**機能名** error - **エラー詳細**)

adshchmsg コマンドまたは adshlsmsg コマンドで，シグナルエラーが発生しました。

機能名および**エラー詳細**はエラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査して原因を取り除き，再度実行してください。

KNAX5426-E

シグナルエラーが発生しました。(**機能名** error)

adshchmsg コマンドまたは adshlsmsg コマンドで，シグナルエラーが発生しました。

機能名はエラー情報を示します。

(S)

処理を終了します。

(O)

エラー情報を調査して原因を取り除き，再度実行してください。

KNAX5429-E

内部エラーが発生しました。(**保守情報**)

adshchmsg コマンドまたは adshlsmsg コマンドで，内部エラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5440-E

Administrators 権限がないためコマンド（**コマンド名**）を実行できません。

コマンド名で示すコマンドに対する実行権限がありません。

このコマンドは，Administrators 権限のユーザーで実行する必要があります。

(S)

処理を終了します。

(O)

Administrators 権限のユーザーで実行してください。

KNAX5498-E

An error occurred during **コマンド名** signal handler processing.

adshchmsg コマンドまたは adshlsmsg コマンドのシグナルハンドラの処理で，エラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX5499-E

The **コマンド名** ended abnormally because of an unexpected exception.

adshchmsg コマンドまたは adshlsmsg コマンドがエラー終了しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX6000-E

指定した組み込みコマンド（"**コマンド名**"）はサポートされていません。[filename="**ファイル名**" line=**行番号**]

JP1/Advanced Shell ではサポートしない組み込みコマンドを指定しました。

コマンド名

JP1/Advanced Shell ではサポートしない組み込みコマンド名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

エラーとなった個所を見直し，ジョブ定義スクリプトを修正します。

KNAX6001-E

指定したシェルオプション（"**シェルオプション名**"）はサポートされていません。[filename="**ファイル名**" line=**行番号**]

JP1/Advanced Shell ではサポートしないシェルオプションを指定しました。

シェルオプション名

JP1/Advanced Shell ではサポートしないシェルオプション名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

エラーとなった個所を見直し，ジョブ定義スクリプトを修正します。

KNAX6002-E

指定したシェル変数名（"**シェル変数名**"）は使用できません。[filename="**ファイル名**" line=**行番号**]

JP1/Advanced Shell では使用できないシェル変数名を指定しました。

シェル変数名

JP1/Advanced Shell では使用できないシェル変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

エラーとなったシェル変数名を別のシェル変数名に修正します。

KNAX6003-E

変数名（"**変数名**"）に不当な文字が指定されています。[filename="**ファイル名**" line=**行番号**]

変数名に不当な文字が指定されています。

変数名

不当と判断した変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンド，正規組み込みコマンド，および typeset コマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった変数名を別の変数名に修正します。

KNAX6004-E

不当な数値（"**不当な値または数値以外の値**"）です。[filename="**ファイル名**" line=**行番号**]

次のどれかの要因が考えられます。

- 整数型の変数に文字を代入しようとしました。
- 数値を指定しなければならない引数に文字を指定しました。
- 数値として不当な値を指定しました。

不当な値または数値以外の値

不当と判断した値または数値以外の値

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンド，正規組み込みコマンド，および typeset コマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった個所が代入式の場合，代入する変数の属性，または代入する値を見直し，ジョブ定義スクリプトを修正します。コマンドの場合，引数に指定した内容を見直し，ジョブ定義スクリプトを修正します。

KNAX6005-E

コマンドに指定した引数が多すぎます。[filename="**ファイル名**" line=**行番号**]

コマンドに指定した引数が多過ぎます。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった個所のコマンドの引数を見直し，ジョブ定義スクリプトを修正します。

KNAX6006-E

置換の指定が誤っています。[filename="**ファイル名**" line=**行番号**]

置換の指定が誤っています。または，cd コマンドの引数に，"カレントディレクトリパス名"に含まれない文字列を指定しています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が cd コマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった個所の変数置換，コマンド置換または引数の指定を見直し，ジョブ定義スクリプトを修正します。

KNAX6007-E

配列（"**配列名**"）の要素番号が範囲外です。[filename="**ファイル名**" line=**行番号**]

配列の要素番号が範囲外です。

配列名

指定された配列名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

配列の要素番号を 0 から 65535 までの範囲で指定するようにジョブ定義スクリプトを修正してください。

環境設定パラメーター VAR_SHELL_FUNCINFO に TYPE_A または TYPE_B を指定している場合は，関数のネストが配列の要素数の上限を超えているため，環境設定パラメーター VAR_SHELL_FUNCINFO に NONE を指定するか，ジョブ定義スクリプトを修正してください。

KNAX6008-E

読み込み専用属性の変数（"**変数名**"）に値を代入しようとしました。[filename="**ファイル名**" line=**行番号**]

読み込み専用属性の変数に値を代入しようとしました。

変数名

指定された変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が次のどれかのコマンドの場合，各コマンドの終了コードを設定し処理を継続します。

- シェル拡張コマンド
- 正規組み込みコマンド
- typeset コマンド

  配列として定義した変数を typeset コマンドで読み込み専用属性に設定した状態で，同一変数に要素番号を指定しないで値を代入するよう指定した場合，このエラーが発生し代入処理は実行されません。ただし，代入式は終了コード 0 を設定し処理を継続します。

それ以外の場合，処理を終了します。

(O)

エラーとなった個所の変数の属性または変数名を見直し，ジョブ定義スクリプトを修正してください。

KNAX6009-E

"**オプション**" は無効なオプションです。[filename="**ファイル名**" line=**行番号**]

コマンドに不当なオプションを指定しました。

オプション

コマンドに指定されたオプション

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンド，正規組み込みコマンド，および typeset コマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

コマンドに指定しているオプションの内容を見直し，ジョブ定義スクリプトを修正します。

KNAX6010-E

"**シェルオプション**" は無効なオプションです。[filename="**ファイル名**" line=**行番号**]

set コマンドで不当なシェルオプションを指定しました。

シェルオプション

指定されたシェルオプション

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

set コマンドに指定しているオプションの内容を見直し，ジョブ定義スクリプトを修正します。

KNAX6011-E

不当なシグナル（"**シグナル番号または名称**"）を指定しました。[filename="**ファイル名**" line=**行番号**]

不当な**シグナル番号または名称**を指定しました。

シグナル番号または名称

指定されたシグナル番号または名称

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

コマンドに指定しているシグナル番号またはシグナル名称を見直し，ジョブ定義スクリプトを修正します。

KNAX6012-E

不当なマスク（"**マスク**"）を指定しました。[filename="**ファイル名**" line=**行番号**]

不当なマスクを指定しました。

マスク

指定されたマスク

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

コマンドに指定しているマスクの内容を見直し，ジョブ定義スクリプトを修正します。

KNAX6013-E

上限値の変更に失敗しました。details="**エラー詳細**" [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，上限値の変更に失敗しました。

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6014-E

値が設定されていない変数（"**変数名**"）を指定しました。[filename="**ファイル名**" line=**行番号**]

nounset シェルオプションを有効にした状態で，値が設定されていない変数を指定しました。

変数名

指定された変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

nounset シェルオプションの設定が必要であるかどうかを見直します。必要な場合，変数を使用するときに値を代入するようジョブ定義スクリプトを修正します。

KNAX6015-E

引数が必要な組み込みコマンドに対して，引数を指定しないで実行しました。[filename="**ファイル名**" line=**行番号**]

引数が必要な組み込みコマンドに対して，引数を指定しないで実行しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった個所のコマンドの指定内容を見直し，ジョブ定義スクリプトを修正します。

KNAX6016-E

オプション（"**オプション**"）の値を指定しないでコマンドを実行しました。[filename="**ファイル名**" line=**行番号**]

オプションの値を指定しないでコマンドを実行しました。

オプション

指定されたオプション

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンド，正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった個所のコマンドの指定内容を見直し，ジョブ定義スクリプトを修正します。

KNAX6017-E

項目（"**項目名**"）の指定が誤っています。[filename="**ファイル名**" line=**行番号**]

制御文の指定が誤っています。

項目名

構文不正となった項目名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6018-E

制御文中の項目（"**項目名**"）の対応が誤っています。[filename="**ファイル名**" line=**行番号**]

制御文で必要な項目名の対応が誤っています。

項目名

構文不正となった項目名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6019-E

構文の解析中に不当な EOF を検出しました。[filename="**ファイル名**" line=**行番号**]

制御文の指定が誤っています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6020-E

不当なディレクトリパス（"**ディレクトリパス**"）が指定されました。[filename="**ファイル名**" line=**行番号**]

不当なディレクトリパスが指定されました。

ディレクトリパス

指定されたディレクトリパス

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ディレクトリを移動しないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6021-E

ヒアドキュメントでリダイレクトの指定が誤っています。[filename="**ファイル名**" line=**行番号**]

ヒアドキュメントでリダイレクトの指定が誤っています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6022-E

リダイレクトの指定が多過ぎます。[filename="**ファイル名**" line=**行番号**]

リダイレクトの指定が多過ぎます。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6023-W

ループ外で {break|continue} コマンドを実行しました。[filename="**ファイル名**" line=**行番号**]

ループ外で break コマンドまたは continue コマンドを実行しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6024-E

関数の定義で不当な関数名（"**関数名**"）を指定しました。[filename="**ファイル名**" line=**行番号**]

関数の定義で不当な関数名を指定しました。

関数名

指定された関数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6025-E

"**ファイル名，コマンド名または関数名**" が見つかりません。[filename="**ファイル名**" line=**行番号**]

特定できない**ファイル名，コマンド名または関数名**を指定しました。

ファイル名，コマンド名または関数名

指定されたファイル名，コマンド名または関数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が.（ドット）コマンドの場合，処理を終了します。それ以外の場合，処理を継続します。

(O)

指定したファイル名，コマンド名および関数名が正しいかどうかを見直し，ジョブ定義スクリプトを修正します。

KNAX6026-E

指定されたコマンド（"**コマンド名**"）が実行できません。reason="**エラー詳細**" [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，指定されたコマンドが実行できません。

コマンド名

指定されたコマンド名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

指定されたコマンドを実行しないで処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6027-W

{break|continue} コマンドの引数に指定した値がループのネスト数（**ループのネスト数**）を超えています。[filename="**ファイル名**" line=**行番号**]

break，continue コマンドの引数に指定した値がループのネスト数よりも多いです。

ループのネスト数

ループの処理を抜けたとき（break コマンド）またはループの処理を中断して先頭に戻ったとき（continue コマンド）のループのネスト数

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ループのネスト数分だけ break コマンドまたは continue コマンドを実行し，処理を継続します。

(O)

break コマンドまたは continue コマンドの引数を見直し，ジョブ定義スクリプトを修正します。

KNAX6028-E

builtin コマンドに組み込みコマンド以外のコマンド（"**コマンド名**"）を指定しました。
[filename="**ファイル名**" line=**行番号**]

builtin コマンドに組み込みコマンド以外のコマンドを指定しました。

コマンド名

指定されたコマンド名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

builtin コマンドに指定したコマンドの引数を見直し，ジョブ定義スクリプトを修正します。

KNAX6029-E

バックグラウンドプロセスがすでに実行されています。[filename="**ファイル名**" line=**行番号**]

バックグラウンドプロセスがすでに実行されています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6030-E

ディレクトリを移動できません。**ディレクトリパス名** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

ディレクトリを移動できません。

ディレクトリパス名

指定されたディレクトリパス名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ディレクトリを移動しないで処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6031-E

シェル変数 HOME が設定されていないため，ディレクトリを移動できません。[filename="**ファイル名**" line=**行番号**]

シェル変数 HOME が設定されていないため，ディレクトリを移動できません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ディレクトリを移動しないで処理を継続します。

(O)

シェル変数 HOME にホームディレクトリを指定し，ジョブ定義スクリプトを再実行します。

KNAX6032-E

シェル変数 OLDPWD が設定されていないため，ディレクトリを移動できません。[filename="**ファイル名**" line=**行番号**]

シェル変数 OLDPWD が設定されていないため，ディレクトリを移動できません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ディレクトリを移動しないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6033-E

カレントディレクトリを特定できないため，ディレクトリを移動できません。[filename="**ファイル名**" line=**行番号**]

カレントディレクトリを特定できないため，ディレクトリを移動できません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ディレクトリを移動しないで処理を継続します。

(O)

ジョブ定義スクリプトを再実行します。

KNAX6034-E

バックグラウンドプロセスが存在しない状態で実行しました。[filename="**ファイル名**" line=**行番号**]

バックグラウンドプロセスがない状態で実行しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6035-E

不当なファイル識別子（"**ファイル識別子**"）を指定しています。reason="**エラー詳細**" [filename="**ファイル名**" line=**行番号**]

不当なファイル識別子を指定しています。

ファイル識別子

指定されたファイル識別子

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6036-E

不当なプロセス ID（"**プロセスID**"）を指定しています。[filename="**ファイル名**" line=**行番号**]

プロセス ID に不当な値を指定しています。

プロセスID

指定されたプロセス ID

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

正規組み込みコマンドの場合，処理を継続します。特殊組み込みコマンドの場合，処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6037-E

オプションを指定しないで getopts コマンドを実行しています。[filename="**ファイル名**" line=**行番号**]

オプションを指定しないで getopts コマンドを実行しています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6038-E

name を指定しないで getopts コマンドを実行しています。[filename="**ファイル名**" line=**行番号**]

name を指定しないで getopts コマンドを実行しています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6039-E

shift コマンドの引数にコマンドライン引数の数よりも大きい値を指定しています。[filename="**ファイル名**" line=**行番号**]

コマンドラインの引数の数よりも指定された引数の方が大きい状態で shift コマンドを実行しています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

shift コマンドに指定した引数，またはコマンドライン引数の数を見直し，必要な場合，ジョブ定義スクリプトを修正します。

KNAX6040-E

]の対応が誤っています。[filename="**ファイル名**" line=**行番号**]

"]"の対応が誤っています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6041-E

指定された test コマンドまたは条件式に誤りがあります。[filename="**ファイル名**" line=**行番号**]

指定された test コマンドまたは条件式に誤りがあります。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

test コマンドの場合，処理を継続します。条件式の場合，処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6042-E

読み込み専用属性の変数を算術式に指定しています。[filename="**ファイル名**" line=**行番号**]

読み込み専用属性の変数を算術式に指定しています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

算術式は処理しないで処理を継続します。

(O)

算術式で使用している変数の属性を見直し，ジョブ定義スクリプトを修正します。

KNAX6043-W

別プロセスで実行するコマンドが指定されていません。[filename="**ファイル名**" line=**行番号**]

別プロセスで実行するコマンドが指定されていません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6044-E

シェル変数 FPATH に指定されたディレクトリに関数（"**関数名**"）の定義ファイルが見つかりません。[filename="**ファイル名**" line=**行番号**]

FPATH シェル変数にディレクトリが指定されていません。または，FPATH シェル変数に指定されたディレクトリに関数定義ファイルが見つかりません。

関数名

関数名

ファイル名

スクリプトファイル名

行番号

エラーが発生したスクリプトファイルの行番号

(S)

処理を継続します。

(O)

関数定義ファイルを格納したディレクトリが FPATH シェル変数に指定されているか確認してください。指定されている場合は，実行しようとした関数名が正しいか，FPATH シェル変数に指定されたディレクトリ内に実行しようとした関数の関数定義ファイルが存在するかを確認してください。

KNAX6045-E

関数（"**関数名**"）の定義ファイルをオープンできません。[filename="**ファイル名**" line=**行番号**]

関数定義ファイルをオープンできません。

関数名

関数名

ファイル名

スクリプトファイル名

行番号

エラーが発生したスクリプトファイルの行番号

(S)

処理を継続します。

(O)

FPATH シェル変数に指定されたディレクトリと，実行しようとした関数の関数定義ファイルの権限を確認してください。

KNAX6046-E

関数（"**関数名**"）が関数定義ファイル（"**関数定義ファイル名**"）内に定義されていません。[filename="**ファイル名**" line=**行番号**]

関数が関数定義ファイル内に定義されていません。

関数名

関数名

関数定義ファイル名

実行しようとした関数を定義する関数定義ファイル名

ファイル名

スクリプトファイル名

行番号

エラーが発生したスクリプトファイルの行番号

(S)

処理を継続します。

(O)

実行しようとした関数名と，関数定義ファイルに定義されている関数名が正しいか確認してください。

KNAX6047-E

上限値に不当な値（"**上限値**"）を指定しています。[filename="**ファイル名**" line=**行番号**]

上限値に不当な値を指定しています。

上限値

指定されたオプション

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

上限値を変更しないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6048-E

ulimit コマンドでハードリミットを変更する際にエラーが発生しました。[filename="**ファイル名**" line=**行番号**]

ulimit コマンドでリソースのハードリミットを変更する際に，エラーが発生しました。このメッセージは次のどちらかの場合に出力されます。

- ハードリミットを変更する権限があり，かつシステムで設定できない値を指定した。
- ハードリミットを変更する権限がなく，かつ設定されているハードリミットを超える値を指定した。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

上限値を変更しないで処理を継続します。ただし，変更するリソースによっては，システムで設定できない値を指定すると，実行環境や OS によって異なる値が設定される場合があります。

(O)

原因に応じて次のように対処したのち，ジョブ定義スクリプトを再実行します。

- ハードリミットを変更する権限があり，かつシステムで設定できない値を指定した場合
  ulimit コマンドの引数をシステムで設定できる値に変更してください。
- ハードリミットを変更する権限がなく，かつ設定されているハードリミットを超える値を指定した場合
  実行ユーザーに管理者権限を割り当て，ulimit コマンドの引数をシステムで設定できる値に変更してください。ハードリミットを減少させる場合は，管理者権限は必要ありません。

KNAX6049-E

ulimit コマンドに指定された資源は変更できません。[filename="**ファイル名**" line=**行番号**]

ulimit コマンドに指定された資源は変更できません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

上限値を変更しないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6050-E

未作成，または値が設定されていない変数（"**変数名**"）を変数置換に指定しています。
[filename="**ファイル名**" line=**行番号**]

未作成，または値が設定されていない変数を変数置換に指定しています。

変数名

指定された変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

変数置換に指定した変数名を見直し，必要な場合，ジョブ定義スクリプトを修正します。

KNAX6051-E

コマンド置換で指定されたリダイレクト指定内容（"**リダイレクト指定内容**"）が誤っています。
[filename="**ファイル名**" line=**行番号**]

コマンド置換で指定された**リダイレクト指定内容**が誤っています。

リダイレクト指定内容

指定されたリダイレクト指定内容

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6052-E

コマンド置換で指定された入力ファイル（"**入力ファイル名**"）をオープンできません。
[filename="**ファイル名**" line=**行番号**]

コマンド置換で指定された入力ファイルをオープンできません。

入力ファイル名

指定された入力ファイル名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6053-E

パイプの生成に失敗しました。[filename="**ファイル名**" line=**行番号**]

パイプの生成に失敗しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

メッセージに出力されたジョブ定義スクリプトファイル名の**行番号**の内容を見直し，記述に誤りがないかどうかを確認します。また，ジョブ定義スクリプト内でオープンしているファイルの数が多過ぎていないか見直します。記述に誤りがある場合は修正し，ジョブ定義スクリプトを再実行します。

再実行後も問題が解決されない場合は，システム管理者に連絡してください。

KNAX6054-E

プロセスの生成に失敗しました。[details=**保守情報**] [filename="**ファイル名**" line=**行番号**]

プロセスの生成に失敗しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

メッセージに出力されたジョブ定義スクリプトファイル名の**行番号**の内容を見直し，記述に誤りがないかどうかを確認します。記述に誤りがある場合は修正し，ジョブ定義スクリプトを再実行します。

再実行後も問題が解決されない場合は，次の要因が考えられます。

- 実行可能ファイルのパスが見つかりません。
- 実行可能ファイルが通常ファイルではありません。
- 実行可能ファイルの構成要素に検索許可がありません。
- 実行可能ファイルのパス名が長過ぎます。
- 実行可能ファイルの引数が多過ぎます。または引数の指定が無効です。
- 指定したファイルが実行可能ではありません。
- 実行可能ファイルのパス名の変換中に見つかったシンボリックリンクが多過ぎます。
- 実行中のプロセスの合計が，システムの上限値を超えています。
- 新しいプロセスを作成するためのスワッピング領域や物理メモリが十分にありません。
- オープンするファイル数が多過ぎます。

上記要因を解決した上で，ジョブ定義スクリプトを再実行します。

再実行後も問題が解決されない場合は，システム管理者に連絡してください。

KNAX6055-E

シグナルの送信に失敗しました。pid=**プロセスID** signalNo=**シグナル番号** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，指定された**プロセスID**のシグナルの送信に失敗しました。

プロセスID

指定されたプロセス ID

シグナル番号

送信に失敗したシグナル番号

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6056-W

指定されたプロセス ID（"**プロセスID**"）は不当な値のため，無視しました。[filename="**ファイル名**" line=**行番号**]

指定されたプロセス ID は不当な値のため，無視しました。

プロセスID

指定されたプロセス ID

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトファイルを修正します。

KNAX6057-E

メモリ不足が発生しました。[filename="**ファイル名**" line=**行番号**]

メモリ不足が発生しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

システム管理者に連絡します。システム管理者は，メモリ見積もりを見直してください。

KNAX6058-E

シェル変数（"**シェル変数名**"）の再帰変換可能回数を超えました。[filename="**ファイル名**" line=**行番号**]

次に示すシェル変数の再帰変換可能回数を超えたため，処理を中止しました。

- シェル変数参照時の**offset** に指定した変数の再帰変換可能回数：1,024
- シェル変数参照時の**length** に指定した変数の再帰変換可能回数：1,025

シェル変数名

循環参照または再帰的な参照となった変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

指定した変数は，循環参照指定または再帰的な参照指定となっているので，変数の指定値を見直してください。

KNAX6059-E

ジョブ定義スクリプト内でオープンしているファイル数が多過ぎます。[filename="**ファイル名**" line=**行番号**]

ジョブ定義スクリプト内でオープンしているファイル数が多過ぎます。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

メッセージに出力されたジョブ定義スクリプトファイル名の行番号の内容を見直し，記述に誤りがないかどうかを確認します。また，ジョブ定義スクリプト内でオープンしているファイルの数が多過ぎていないか見直します。記述に誤りがある場合は修正し，ジョブ定義スクリプトを再実行します。

再実行後も問題が解決されない場合は，システム管理者に連絡してください。

KNAX6061-E

ヒアドキュメントの生成に失敗しました。[filename="**ファイル名**" line=**行番号**]

ヒアドキュメントの生成に失敗しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6062-E

一時ファイルを{作成|オープン|削除}できません。 **一時ファイル名** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，一時ファイルを作成，オープン，または削除できません。

一時ファイル名

処理しようとした一時ファイル名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。ただし，ヒアドキュメントの処理でエラーが発生した場合は，処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6063-E

ファイルへの書き込みが失敗しました。**書き込み先ファイル名** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，ファイルへの書き込みが失敗しました。

書き込み先ファイル名

書き込みを実行しようとしたファイル名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6064-E

算術式（"**変数名または算術式**"）で一時的に使用する変数の作成に失敗しました。[filename="**ファイル名**" line=**行番号**]

算術式で一時的に使用する変数の作成に失敗しました。

変数名

エラーが発生した算術式に含まれる変数名

算術式

エラーが発生した算術式

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった算術演算をしないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6065-E

不当な変数を算術式（"**算術式**"）に使用しました。[filename="**ファイル名**" line=**行番号**]

不当な変数を算術式に使用しました。

算術式

エラーが発生した算術式

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった算術演算をしないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6066-E

算術式の書式が誤っています。[filename="**ファイル名**" line=**行番号**]

算術式の書式が誤っています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった算術演算をしないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6067-E

ゼロ除算を実施しました。[filename="**ファイル名**" line=**行番号**]

ゼロ除算を実施しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった算術演算をしないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6068-E

算術演算子（"**"）の指数に負の値が指定されました。[filename="**ファイル名**" line=**行番号**]

算術演算子「**」の指数に負の値が指定されました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった算術演算をしないで処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6070-E

実行完了待ちが必要なジョブがありません。[filename="**ファイル名**" line=**行番号**]

実行完了待ちが必要なジョブがありません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6071-E

カレントディレクトリを特定できません。reason="エラー詳細" [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，カレントディレクトリを特定できません。

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

ディレクトリを移動しないで処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6072-E

必要なオプションが指定されていません。[filename="**ファイル名**" line=**行番号**]

必要なオプションが指定されていません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンド，正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラーとなった個所のコマンドのオプションを見直し，ジョブ定義スクリプトを修正してください。

KNAX6075-E

リダイレクト（**リダイレクト文字**）実行時にファイル識別子の複写が失敗しました。reason="**エラー詳細**" [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，dup によってファイル識別子を複写できません。

リダイレクト文字

指定されたリダイレクト文字

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6076-E

getopts コマンドの実行中に引数の内容が変更されました。[filename="**ファイル名**" line=**行番号**]

getopts コマンドの実行中に引数の内容が変更されました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を継続します。

(O)

エラーが発生したジョブ定義スクリプトファイルの行番号の内容を見直し，記述に誤りがないことを確認します。記述が誤っていた場合はジョブ定義スクリプトを修正し，再実行します。

KNAX6077-E

オプションの指定が誤っています。[filename="**ファイル名**" line=**行番号**]

オプションの指定が誤っています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6078-E

クォーテーションの対応が誤っています。[filename="**ファイル名**" line=**行番号**]

クォーテーションの対応が誤っています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6079-E

ヒアドキュメントで指定したラベル（"**ラベル**"）が見つかりません。[filename="**ファイル名**" line=**行番号**]

ヒアドキュメントで指定したラベルが見つかりません。

ラベル

ヒアドキュメントに指定したラベル

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6080-E

ファイル識別子の複写に失敗しました。 **対象ファイル名** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，ファイル識別子の複写に失敗しました。

対象ファイル名

ファイル識別子の複写に失敗したファイル名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6081-E

ファイルのオープンに失敗しました。 **対象ファイル名** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，ファイルのオープンに失敗しました。

対象ファイル名

ファイルのオープンに失敗したファイル名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6082-E

ファイルの作成に失敗しました。 **対象ファイル名** - **エラー詳細**. [filename="**ファイル名**" line=**行番号**]

エラー詳細で示すエラーが発生したため，ファイルの作成に失敗しました。

対象ファイル名

ファイルの作成に失敗したファイル名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所がシェル拡張コマンド，正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

エラー詳細を基に原因を取り除きます。必要な場合，ジョブ定義スクリプトを修正します。

KNAX6085-E

指定されたシグナル（"**シグナル番号または名称**"）には，トラップを設定できません。
[filename="**ファイル名**" line=**行番号**]

指定されたシグナルには，トラップを設定できません。

シグナル番号または名称

指定されたシグナル番号またはシグナル名称

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

trap コマンドに指定したシグナル番号またはシグナル名称を見直し，修正します。

KNAX6097-E

指定されたシェル拡張変数（"*変数名*"）の属性は変更できません。[filename="*ファイル名*" line=*行番号*]

シェル拡張変数の変更できない属性を変更しようとしました。

変数名

属性を変更しようとしたシェル拡張変数名

ファイル名

スクリプトファイル名

行番号

エラーが発生したスクリプトファイルの行番号

(S)

処理を終了します。

(O)

次のどちらかの方法で対処してください。

- エラーとなった個所の変数名，または変更しようとした属性を見直し，ジョブ定義スクリプトを修正してください。
- エラーとなった変数が関数情報配列の場合は，環境設定パラメーターVAR_SHELL_FUNCINFO の指定を NONE に変更して再実行してください。

ジョブ定義スクリプトを再実行しても問題が解決しない場合は，システム管理者に連絡してください。

KNAX6098-E

エラーが発生しました。reason=**ソース上の行番号**, **障害解析情報** [filename="**ファイル名**" line=**行番号**]

エラーが発生しました。

ソース上の行番号

エラーが発生したソース上の行番号

障害解析情報

障害解析情報

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

エラーとなった個所が正規組み込みコマンドの場合，処理を継続します。それ以外の場合，処理を終了します。

(O)

メッセージに出力されたジョブ定義スクリプトファイル名の行番号の内容を見直し，記述に誤りがないかどうかを確認します。記述に誤りがある場合は修正し，ジョブ定義スクリプトを再実行します。再実行後も問題が解決されない場合は，システム管理者に連絡します。

KNAX6099-E

内部エラーが発生しました。reason=**ソース上の行番号**, **障害解析情報** [filename="**ファイル名**" line=**行番号**]

内部エラーが発生しました。

ソース上の行番号

エラーが発生したソース上の行番号

障害解析情報

障害解析情報

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX6100-E

コマンド（"**コマンド名**"）の実行に失敗しました。rc=**終了コード** [filename="**ファイル名**" line=**行番号**]

コマンドの実行に失敗しました。このメッセージが出力された場合，KNAX7999-I で出力された終了コードは無効となり，このメッセージで出力された**終了コード**がジョブの終了コードになります。

コマンド名

実行に失敗したコマンド名

終了コード

ジョブの終了コード

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を終了します。

(O)

このメッセージの直前に出力される KNAX6098-E に従って，エラーとなった要因を解決してからジョブ定義スクリプトを再実行してください。再実行しても問題が解決されない場合は，システム管理者に連絡してください。

KNAX6110-I

コマンド（**コマンド名**, 行番号=**行数**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6111-I

コマンド（**コマンド名**, 行番号=**行数**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6112-I

コマンド（**コマンド名**, 行番号=**行数**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6113-I

コマンド（**コマンド名**, 行番号=**行数**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6114-I

コマンド（**コマンド名**, 行番号=**行数**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6115-I

コマンド（**コマンド名**, 行番号=**行数**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6116-I

コマンド（**コマンド名**, 行番号=**行数**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6117-I

コマンド（**コマンド名**, 行番号=**行数**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているスクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6118-I

関数（**関数名**, 行番号=**行数**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

関数が正常終了しました。

関数名

実行した関数名。

行数

関数が記述されているスクリプトの行数

終了コード

関数の終了コード

実行時間

関数の実行時間（OS の API で取得された参考値）

CPU時間

関数の CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6119-I

関数（**関数名**, 行番号=**行数**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

関数が終了しました。このメッセージは，終了コードによって成功・失敗を区別しない関数が終了した場合に出力されます。

関数名

実行した関数名。

行数

関数が記述されているスクリプトの行数

終了コード

関数の終了コード

実行時間

関数の実行時間（OS の API で取得された参考値）

CPU時間

関数の CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6120-I

コマンド（**コマンド名**, **機能名**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6121-I

コマンド（**コマンド名**, **機能名**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6122-I

コマンド（**コマンド名**, **機能名**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6123-I

コマンド（**コマンド名**, **機能名**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6124-I

コマンド（**コマンド名**, **機能名**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6125-I

コマンド（**コマンド名**, **機能名**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6126-I

コマンド（**コマンド名**, **機能名**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが正常終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6127-I

コマンド（**コマンド名**, **機能名**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドが終了しました。このメッセージは，終了コードによって成功・失敗を区別しないコマンドが終了した場合に出力されます。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間（OS の API で取得された参考値）

CPU時間

コマンドの CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6128-I

関数（**関数名**，**機能名**）が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行された関数が正常終了しました。

関数名

実行した関数名。

機能名

関数を実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

関数の終了コード

実行時間

関数の実行時間（OS の API で取得された参考値）

CPU時間

関数の CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6129-I

関数（**関数名**，**機能名**）が終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行された関数が終了しました。このメッセージは，終了コードによって成功・失敗を区別しない関数が終了した場合に出力されます。

関数名

実行した関数名。

機能名

関数を実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

終了コード

関数の終了コード

実行時間

関数の実行時間（OS の API で取得された参考値）

CPU時間

関数の CPU 時間（OS の API で取得された参考値）

(S)

処理を続行します。

KNAX6130-E

イベントファイル（"**イベントファイルのパス名**"）へのジョブ定義スクリプト稼働実績情報の出力で入出力エラーが発生しました。

イベントファイルへのジョブ定義スクリプト稼働実績情報の出力で入出力エラーが発生しました。

イベントファイルのパス名

入出力エラーが発生したイベントファイルのパス名

(S)

処理を続行します。

(O)

前後のメッセージを参照して，入出力エラーの原因を調査し，解決してください。

KNAX6134-E

コマンドに指定した変数（"**変数名**"）の型が不正です。[filename="**ファイル名**" line=**行番号**]

変数名の型属性はこのコマンドでは使用できない型属性です。

変数名

変数名

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

エラーとなった変数名を見直して，ジョブ定義スクリプトを修正します。

KNAX6135-E

コマンドの引数が不足しています。[filename="**ファイル名**" line=**行番号**]

引数の数が不足しています。

adshvarconv では変数名または変換前エンコーディング，変換後エンコーディングの指定がありません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

エラーとなった変数名を見直して，ジョブ定義スクリプトを修正します。

KNAX6136-E

オプションの組み合わせが正しくない。[filename="**ファイル名**" line=**行番号**]

コマンドのオプションの組み合わせが正しくありません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

コマンドのオプション指定を見直して，ジョブ定義スクリプトを修正します。

KNAX6137-E

値の長さが範囲外です。[filename="**ファイル名**" line=**行番号**]]

コマンドに指定した値の長さが範囲外です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

コマンドの引数指定を見直して，ジョブ定義スクリプトを修正します。

KNAX6138-E

変数を複数指定することはできません。[filename="**ファイル名**" line=**行番号**]

変数を複数指定することはできません（配列で複数の要素を指定する場合を含みます）。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

コマンドの引数指定を見直して，ジョブ定義スクリプトを修正します。

KNAX6139-E【Windows 限定】

エンコーディングが無効です。[filename="**ファイル名**" line=**行番号**]

指定されたエンコーディングは無効です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

コマンドの引数指定を見直して，ジョブ定義スクリプトを修正します。

KNAX6140-E【Windows 限定】

変換できない文字が存在します。[filename="**ファイル名**" line=**行番号**]

コード変換時に，変換できない文字が存在します。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

コマンドを終了します。

(O)

変換前のデータを見直して，ジョブ定義スクリプトを修正します。

KNAX6150-E

エラー通知対象にエラーが通知されました。rc=**終了コード** line=**行番号**

ジョブまたはジョブステップにエラーが通知されました。adshjoberr コマンドによってエラーが通知された場合にこのメッセージが出力されます。

エラー通知対象

エラーを通知した対象として，次のどちらかが出力されます。

- ジョブにエラーを通知した場合：ジョブ
- ジョブステップにエラーを通知した場合：ジョブステップ

終了コード

ジョブまたはジョブステップに通知した終了コード

行番号

エラーを通知したジョブ定義スクリプトの行数

(S)

このメッセージが通知された場合，次のように動作します。

- ジョブステップ内でエラーを通知した場合，ジョブステップの onError 属性の指定に関わらず，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップがエラー終了するか，ジョブステップエラーブロックを実行します。
- ジョブステップ外でエラーを通知した場合，ジョブがエラー終了するか，run 属性が abnormal または always の後続のジョブステップを実行します。

KNAX6151-E

エラー通知対象にエラーが通知されました。rc=**終了コード，機能名**

ジョブまたはジョブステップにエラーが通知されました。adshjoberr コマンドによってエラーが通知された場合にこのメッセージが出力されます。

エラー通知対象

エラーを通知した対象として，次のどちらかが出力されます。

- ジョブにエラーを通知した場合：ジョブ
- ジョブステップにエラーを通知した場合：ジョブステップ

終了コード

ジョブまたはジョブステップに通知した終了コード

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

(S)

このメッセージが通知された場合，次のように動作します。

- ジョブステップ内でエラーを通知した場合，ジョブステップの onError 属性の指定に関わらず，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップがエラー終了するか，ジョブステップエラーブロックを実行します。
- ジョブステップ外でエラーを通知した場合，ジョブがエラー終了するか，run 属性が abnormal または always の後続のジョブステップを実行します。

KNAX6152-E

adshjoberr: コマンドラインの指定に誤りがあります。details="**詳細**"

コマンドラインの指定に誤りがあります。メッセージに表示される**詳細**の表示内容と，その意味を次に示します。

詳細の表示内容	**意味**
終了コードが指定されていません。	終了コードが指定されていません。
終了コードの指定値が範囲外です。	終了コードの指定値が範囲外です。
終了コードに数字以外を指定しています。	終了コードに数字以外を指定しています。
引数の指定が多過ぎます。	引数の指定が多過ぎます。

(S)

処理を続行します。

(O)

コマンドを正しく指定して再実行してください。

KNAX6153-E

`adshjoberr` コマンドの指定位置が不正です。

`adshjoberr` コマンドの指定位置が不正です。次の原因が考えられます。

- .env ファイル，初期設定スクリプトファイルに指定したadshjoberr コマンドが実行されました。
- ジョブステップエラーブロック内に指定したadshjoberr コマンドが実行されました。

(S)

- .env ファイル，初期設定スクリプトファイルに指定したadshjoberr コマンドが実行された場合は，処理を終了します。
- ジョブステップエラーブロック内に指定したadshjoberr コマンドが実行された場合は，処理を続行します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6160-I

関数（**関数名**, 行番号=**行数**）の実行を中断しました。

次の要因により関数の実行を中断しました。

- 関数内で exit コマンドを実行して，関数およびジョブ定義スクリプトが終了した
- 関数内で exec コマンドを実行して，関数およびジョブ定義スクリプトが終了した
- Windows で，関数実行中に trap コマンドのアクション内で return コマンドが実行されて関数が終了した

関数名

実行した関数名。

行数

関数が記述されているジョブ定義スクリプトの行数。

(S)

処理を中断します。

(O)

エラーが原因で関数の実行が中断された場合は，障害を取り除いて再実行します。

KNAX6161-I

関数（**関数名**, **機能名**）の実行を中断しました。

機能名で示す機能で実行された関数が，次の要因により実行を中断しました。

- 関数内で exit コマンドを実行して，関数およびジョブ定義スクリプトが終了した
- 関数内で exec コマンドを実行して，関数およびジョブ定義スクリプトが終了した
- Windows で，関数実行中に trap コマンドのアクション内で return コマンドが実行されて関数が終了した

関数名

実行した関数名。

機能名

関数を実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

(S)

処理を中断します。

(O)

エラーが原因で関数の実行が中断された場合は，障害を取り除いて再実行します。

KNAX6180-E

adshjava: コマンドラインの指定に誤りがあります。details="**詳細**"

コマンドラインの指定に誤りがあります。メッセージに表示される**詳細**の表示内容と，その意味を次に示します。

詳細の表示内容	**意味**
バッチサーバ名が指定されていません。	バッチサーバ名の指定がない。
バッチサーバ名が長すぎます。	バッチサーバ名が長過ぎる。
スケジュールグループ名が指定されていません。	グループ名の指定がない。
スケジュールグループ名が長すぎます。	グループ名が長過ぎる。
オプション名オプションが既に指定されています。	**オプション名**のオプションが複数回指定された。
-grp オプションと-srv オプションは同時に指定することはできません。	バッチサーバ名とグループ名は同時には指定できない。
-java オプションの指定がありません。	-java オプションの指定がない。
-java オプションの値が指定されていません。	-java オプションの後ろに何も指定がない。

(S)

処理を終了します。

(O)

コマンドを正しく指定して再実行してください。

KNAX6181-E

adshjava: コマンド処理中に継続できないエラーが発生しました。details="**詳細**"

コマンド処理中に継続できないエラーが発生しました。メッセージに表示される**詳細**の表示内容と，その意味を次に示します。

詳細の表示内容	**意味**
実行環境が正しくありません。(**内部情報**)	adshexec で実行していないなど，実行環境が正しくない。
_time64 error(**内部情報**)	時刻を求める処理でエラーが発生した。
clock_gettime error : **エラー情報**(**内部情報**)	
OpenMutex error : **エラー情報**(**内部情報**)	排他処理でエラーが発生した。
WaitForSingleObject error : **エラー情報**(**内部情報**)	事象発生を待つ処理でエラーが発生した。
CreateProcess error : **エラー情報**(**内部情報**)	cjexecjob コマンド，または cjkilljob コマンドを実行するプロセスの生成時にエラーが発生した。cjexecjob コマンド，または cjkilljob コマンドのパスの設定に誤りがある場合や，コマンドを実行する権限がないユーザーで実行している場合に，このエラーが発生することがある。
WaitForMultipleObjects error : **エラー情報**(**内部情報**)	事象発生を待つ処理でエラーが発生した。
GetExitCodeProcess error : **エラー情報**(**内部情報**)	終了コードの取得処理でエラーが発生した。
execvp error : **エラー情報**(**内部情報**)	cjexecjob コマンドの起動時にエラーが発生した。cjexecjob コマンドが見つからない場合，またはコマンドを実行する権限がないユーザーで実行している場合に，このエラーが発生することがある。
fork error : **エラー情報**(**内部情報**)	プロセスの生成時にエラーが発生した。
waitpid error : **エラー情報**(**内部情報**)	プロセスの終了を待つ処理でエラーが発生した。
SetConsoleCtrlHandler error : **エラー情報**(**内部情報**)	ハンドラの設定処理に失敗した。
setpgid error : **エラー情報**(**内部情報**)	プロセスグループの変更に失敗した。
nanosleep error : **エラー情報**(**内部情報**)	処理の遅延処理で失敗した。
SetEvent error : **エラー情報**(**内部情報**)	シグナルの通知処理に失敗した。
CreateEvent error : **エラー情報**(**内部情報**)	シグナル通知のためのイベントオブジェクトの生成に失敗した。

(S)

処理を終了します。

(O)

障害を取り除いて再実行してください。

障害を特定できない場合，または障害を取り除けない場合は，システム管理者に連絡してください。

KNAX6182-E

adshjava: A forced end process failed. "**詳細**" (**エラー番号または終了状態情報**)

強制終了処理中に継続できないエラーが発生しました。メッセージに表示される**詳細**の表示内容とその意味，および**エラー番号または終了状態情報**の表示内容を次に示します。

詳細の表示内容	意味	エラー番号または終了状態情報の表示内容
execvp error	cjkilljob コマンドの起動に失敗した。cjkilljob コマンドが見つからない場合，またはコマンドを実行する権限がないユーザーで実行している場合に，このエラーが発生することがある。	エラー番号
fork error	cjkilljob コマンドの起動に失敗した。	エラー番号
waitpid error	cjkilljob コマンドの終了待ち処理に失敗した。	エラー番号
The process of the cjkilljob command was terminated abnormally	cjkilljob コマンドのプロセスがシグナルなどでエラー終了した。	終了状態情報

エラー番号

16 進文字列表示のエラー番号を表示します。エラー番号については UNIX の errno 定義ファイル（errno.h）を参照してください。

終了状態情報

プロセス終了時に通知された終了状態情報を 16 進文字列で表示します。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。また，cjkilljob コマンドを実行して Java のバッチアプリケーションを停止させてください。

KNAX6183-E

adshjava: コマンド（**コマンド名**）が，シグナルなどでエラー終了しました。details="**終了状態情報**"（**内部情報**）

コマンド名で示すコマンドが，シグナルなどでエラー終了しました。

終了状態情報

プロセス終了時に通知された終了状態情報

(S)

処理を終了します。

(O)

システム管理者に連絡してください。Java のバッチアプリケーションを停止させる必要がある場合は，cjkilljob コマンドを実行して Java のバッチアプリケーションを停止させてください。

KNAX6189-I

adshjava: uCosminexus Application Server が使用する Java のバッチアプリケーションのジョブ ID を割り当てました。job ID="**ジョブID**"

uCosminexus Application Server が使用する Java のバッチアプリケーションのジョブ ID を割り当てました。

(S)

処理を続行します。

KNAX6190-E

コマンド名コマンドの引数の数が不当です。[filename="**ファイル名**" line=**行番号**]

コマンド名で示すコマンドの引数の数が不当です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX6191-E

コマンド名コマンドの引数に不当な文字が含まれています。[filename="**ファイル名**" line=**行番号**]

コマンド名で示すコマンドの引数に不当な文字が含まれています。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX6192-E

コマンド名コマンドの引数の長さが不当です。[filename="**ファイル名**" line=**行番号**]

コマンド名で示すコマンドの引数の長さが不当です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX6193-E

割り当てられる一時ファイルパス名の数の上限を超えました。[filename="**ファイル名**" line=**行番号**

割り当てられる一時ファイルパス名の数の上限を超えました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX6194-E

一時ファイルパス名の長さが OS の許容する最大長を超えました。[filename="**ファイル名**" line=**行番号**]

一時ファイルパス名の長さが OS の許容する最大長を超えました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX6200-I

使用方法: **コマンド名** **コマンド引数**

コマンド引数の文法を**コマンド名**とともに出力します。

(S)

処理を終了します。

KNAX6201-E

オプション（"**オプション名**"）の値が指定されていません。

オプション名のオプションの値が指定されていません。

(S)

処理を終了します。

(O)

表示されたオプションに対して，値を指定します。

KNAX6202-E

無効なオプション名（"**オプション名**"）が指定されています。

不明な**オプション名**が指定されています。

(S)

処理を終了します。

(O)

正しいオプションを指定します。

KNAX6203-E

オプションの値（"**オプション値**"）が不正です。

オプションの値が不正です。**オプション値**は，指定したオプションの値です。

(S)

処理を終了します。

(O)

オプションに対して正しい値を指定します。

KNAX6204-E

同時に指定できないオプション名（"**オプション名**"）を指定しています。

同時に指定できない**オプション名**を指定しています。

(S)

処理を終了します。

(O)

正しいオプションの組み合わせを指定します。

KNAX6206-E

コマンドの引数に asc ファイルが指定されていません。

コマンドの引数に asc ファイルが指定されていません。

(S)

処理を終了します。

(O)

コマンドの引数に asc ファイルを指定します。

KNAX6207-E

コマンドに余分な引数があります。

コマンドに余分な引数があります。

(S)

処理を終了します。

(O)

コマンドの引数を正しく指定します。

KNAX6208-E

コマンドの引数が不足しています。

コマンドの引数が不足しています。

(S)

処理を終了します。

(O)

コマンドの引数を正しく指定します。

KNAX6209-W

範囲指定の行番号がジョブ定義スクリプトの行番号の範囲外です。

範囲指定の行番号がジョブ定義スクリプトの行番号の範囲外です。

(S)

範囲指定の行番号を次のように解釈して，処理を続行します。

- 開始行の行番号がジョブ定義スクリプトの行番号の範囲外である場合，誤りのある範囲指定を無視します。
- 開始行の行番号がジョブ定義スクリプトの行番号の範囲内で，終了行の行番号がジョブ定義スクリプトの行番号の範囲外である場合，誤りのある終了行の行番号をジョブ定義スクリプトの最大行番号とします。

上記で，誤りのある範囲指定を無視した結果，有効な範囲指定がない場合，先頭部分の見出しの情報だけを出力します。ジョブ定義スクリプトとカバレージ情報は出力しません。

(O)

正しい行番号を指定します。

KNAX6210-E

asc ファイル（"**ファイル名**"）のオープンでエラーが発生しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルのオープンでエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルのオープンがエラーとなった理由を調査し，問題を解決してから再度コマンドを実行します。

Windows の場合，adshexec コマンドの asc ファイルに，パス名の最後にディレクトリ区切り文字の「¥」があるディレクトリを指定すると，エラー詳細に"No such file or directory"が表示されます。このときは，ディレクトリではなく，ファイル名を指定してください。

KNAX6211-E

asc ファイル（"**ファイル名**"）をロックできません。reason="**エラー詳細**"

ファイル名で示す asc ファイルをロックできません。

エラー詳細は，ファイルをロックできないエラーの内容を示します。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルをロックできない原因を調査し，問題を解決してから再度コマンドを実行します。asc ファイルをロックできない原因の多くは，ほかのプログラムが asc ファイルを使用しているためです。

KNAX6212-E

asc ファイル（"**ファイル名**"）の読み込みでエラーが発生しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルの読み込みでエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルの読み込みができない原因を調査し，問題を解決してから再度コマンドを実行します。

KNAX6213-E

asc ファイル（"**ファイル名**"）の形式不正を検出しました。details=**保守情報**

ファイル名で示す asc ファイルの形式不正を検出しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

ファイル名に示す asc ファイルを削除し，新規にカバレージ情報を採取します。asc ファイルを不当に更新していないかを確認します。確認事項に問題がなく，同じ現象が発生する場合は，**ファイル名**に示す asc ファイルを保存し，製品の提供元に問い合わせてください。

KNAX6214-E

asc ファイル（"**ファイル名**"）の更新でエラーが発生しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルの更新でエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルの読み込みができない原因を調査し，問題を解決してから再度コマンドを実行します。

KNAX6215-E

asc ファイル（"**ファイル名**"）のロックを解除しようとしたときにエラーが発生しました。
reason="**エラー詳細**"

ファイル名で示す asc ファイルのロックを解除しようとしたときにエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルのロックを解除しようとしたときにエラーが発生した原因を調査し，問題を解決します。

KNAX6219-E

ジョブ定義スクリプトファイル（"**ジョブ定義スクリプト名**"）の内容が，asc ファイル（"**ascファイル名**"）にカバレージ情報を採取したときの内容と異なります。

次に示すジョブ定義スクリプトが異なるため，**ascファイル名**で示す asc ファイルにカバレージ情報を継続蓄積できません。

- **ジョブ定義スクリプト名**で示すジョブ定義スクリプト
- **ascファイル名**で示す asc ファイルのカバレージ情報を採取したときのジョブ定義スクリプト

(S)

処理を終了します。

(O)

次のどちらかの対処を実施してください。

- **ascファイル名**で示す asc ファイルにカバレージ情報を継続蓄積する場合

  asc ファイル名で示す asc ファイルにカバレージ情報を採取したときのジョブ定義スクリプトファイルを使用します。

- **ascファイル名**で示す asc ファイルにカバレージ情報を継続蓄積しない場合

  ascファイル名で示す asc ファイルのカバレージ情報が不要な場合は，adshexec コマンドに-f オプションを指定します。すでに採取しているカバレージ情報を破棄し，新規にカバレージ情報を格納します（初回蓄積）。

  ascファイル名で示す asc ファイルのカバレージ情報が必要な場合は，adshexec コマンドの-o オプションで，出力先の asc ファイルを指定します。指定する asc ファイルが出力先にないことが必要です。

KNAX6220-I

カバレージ情報のマージ処理を開始します。出力先ファイル="**ファイル名**"

カバレージ情報のマージ処理を開始します。

ファイル名は，マージ結果を格納する asc ファイルのファイル名です。

(S)

処理を続行します。

KNAX6221-I

ベース asc ファイル="**ファイル名1**", マージ asc ファイル="**ファイル名2**"

ファイル名1 は，ベース asc ファイルです。

ファイル名2 は，マージ asc ファイルです。

(S)

処理を続行します。

KNAX6222-I

カバレージ情報のマージ処理が終了しました。出力先ファイル="**ascファイル名**"

カバレージ情報のマージ処理が終了しました。

ascファイル名は，マージ結果を格納した asc ファイルのファイル名です。

(S)

処理を続行します。

KNAX6223-E

asc ファイル（"**ファイル名1**"）と asc ファイル（"**ファイル名2**"）では，カバレージ情報を採取したときのジョブ定義スクリプトの内容が異なります。

ファイル名1 と**ファイル名2** の asc ファイルでは，カバレージ情報を採取した場合のジョブ定義スクリプトの内容が異なります。

ファイル名1 は，ベース asc ファイルです。

ファイル名2 は，マージ asc ファイルです。

(S)

処理を終了します。

(O)

同一のジョブ定義スクリプトファイルで採取した asc ファイルを，adshcvmerg コマンドのベース asc ファイル，マージ asc ファイルに指定します。

KNAX6225-E

処理で内部矛盾を検出しました。details=**保守情報**

処理で内部矛盾を検出しました。

(S)

処理を終了します。

(O)

保守情報とともに，製品の提供元に問い合わせます。

KNAX6226-E

スクリプト制御文の，すべてを合わせたネストが深過ぎます。

スクリプト制御文の，すべてを合わせたネストが深過ぎます。

(S)

処理を終了します。

(O)

カバレージ情報を採取する必要がある場合，スクリプト制御文のすべてを合わせたネストが深くならないように，ジョブ定義スクリプトを変更します。

KNAX6227-E

asc ファイルのバージョン番号（"**バージョン**"）が，コマンドがサポートしているバージョン番号と異なります。

asc ファイルのバージョン番号が，コマンドがサポートしているバージョン番号と異なります。

バージョンは，バージョン番号を示します。

(S)

処理を終了します。

(O)

コマンドがサポートしているバージョンの asc ファイルを指定します。

KNAX6228-E

日時の取得でエラーが発生しました。reason="**エラー詳細**"

日時の取得でエラーが発生しました。

(S)

処理を終了します。

(O)

日時の取得が失敗した原因を調査し，対策します。日時の取得には time 関数を使用しています。

KNAX6229-E

asc ファイル（"**ファイル名**"）の情報取得でエラーが発生しました。reason="**エラー詳細**"

ファイル名で示すファイルの情報取得でエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

ファイルの情報取得でエラーが発生した原因を調査し，問題を解決します。ファイルに対してアクセス権限がない場合が多いです。

KNAX6231-E

マージ asc ファイルがベース asc ファイルと同一のファイルです。

マージ asc ファイルがベース asc ファイルと同一のファイルです。

(S)

処理を終了します。

(O)

マージ asc ファイルとベース asc ファイルは，カバレージ情報のマージが必要な異なるファイルを指定します。

KNAX6232-E

コマンドを実行しているユーザーのユーザー名を取得できません。

コマンド実行しているユーザーのユーザー名を取得できません。

(S)

処理を終了します。

(O)

コマンドを実行しているユーザーのユーザー名が取得できない原因を調査し，問題を解決します。UNIX の場合，/etc/passwd にコマンドを実行しているユーザーのユーザー名が登録されていない可能性があります。

KNAX6233-E

asc ファイル（"**ファイル名**"）がすでにあります。

ファイル名で示すファイルがすでにあります。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

ファイル名で示すファイルがすでにあるため，処理を実行できません。ファイル名を変更するか，またはファイルを削除します。

KNAX6236-E

指定された asc ファイル（"**ファイル名**"）は通常のファイルではありません。

ファイル名で示すファイルは通常のファイルではありません。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

ファイル名で示すファイルの種別を確認します。

KNAX6237-E

ファイル種別のパス名がジョブ定義スクリプトファイルのパス名と重複しました。

asc ファイル（コマンドが作成する asc ファイルを含みます）のパス名がジョブ定義スクリプトファイルのパス名と重複しました。

ファイル種別

ジョブ定義スクリプトファイルのパス名と重複した asc ファイル

- 「asc ファイル」：省略，または指定された asc ファイルのパス名が重複しました。
- 「一時 asc ファイル」：一時 asc ファイルのパス名が重複しました。
- 「バックアップ asc ファイル」：バックアップ asc ファイルのパス名が重複しました。

(S)

処理を終了します。

(O)

asc ファイルのパス名を明示的に指定して，ジョブ定義スクリプトファイルのパス名とは異なるパス名にします。asc ファイルのパス名を明示的に指定している場合，明示的に指定している asc ファイルのパス名を変更して，ジョブ定義スクリプトファイルのパス名と異なるパス名にします。

KNAX6238-E

asc ファイル（"**ファイル名**"）の名称変更処理でエラーが発生しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルの名称変更処理でエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルの名称変更処理でエラーが発生した原因を調査し，問題を解決します。コマンドの実行中にファイル名に示すファイルを書き込み保護の設定や，アクセス権限の変更を行った可能性があります。

KNAX6239-E

asc ファイル（"**ファイル名**"）の削除処理でエラーが発生しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルの削除処理でエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルの削除処理でエラーが発生した原因を調査し，問題を解決します。コマンドの実行中に書き込み保護となっているか，またはアクセス権限のないファイル名に示すファイルが作成された可能性があります。

KNAX6240-E

asc ファイル（"**ファイル名**"）の位置づけ処理でエラーが発生しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルの位置づけ処理でエラーが発生しました。

ファイル名には，コマンドで指定した asc ファイルだけでなく，コマンドが一時的に作成する asc ファイルのファイル名も出力します。

(S)

処理を終了します。

(O)

asc ファイルの位置づけ処理でエラーが発生した原因を調査し，問題を解決します。

KNAX6241-E

出力 asc ファイルと入力 asc ファイルに同じファイルが指定されました。

adshcvmerg コマンドで，出力 asc ファイルと入力 asc ファイル（ベース asc ファイルまたはマージ asc ファイル）に同じファイルを指定しています。

(S)

処理を終了します。

(O)

出力 asc ファイルは，入力 asc ファイルと異なるファイルを指定します。

KNAX6242-I

asc ファイル（"**ファイル名**"）のカバレージ情報が更新されました。

adshexec コマンドで，asc ファイルのカバレージ情報が更新されました。**ファイル名**は，更新した asc ファイルのパス名です。

(S)

処理を続行します。

KNAX6243-I

asc ファイル（"**ファイル名**"）をバックアップ asc ファイルから回復しました。

adshexec コマンドで，asc ファイルをバックアップ asc ファイルから回復しました。**ファイル名**は，回復した asc ファイルのパス名です。

(S)

処理を続行します。

KNAX6244-E

asc ファイル（"**ファイル名**"）のファイルサイズの変更に失敗しました。reason="**エラー詳細**"

ファイル名で示す asc ファイルの初期化処理でエラーが発生しました。

Windows の場合，_chsize 関数で発生したエラーです。

UNIX の場合，ftruncate 関数で発生したエラーです。

(S)

処理を終了します。

(O)

asc ファイルの初期化処理でエラーが発生した原因を調査し，問題を解決します。

KNAX6290-E

カバレージ情報表示プログラムの設定が不当です。詳細コード＝**保守情報**

表示プログラムを起動するプログラムの設定が誤っています。

(S)

処理を終了します。

(O)

保守情報とともに，製品の提供元に問い合わせてください。

KNAX6291-E

カバレージ情報表示プログラムの起動に失敗しました。エラー詳細＝**エラー詳細**

カバレージ情報表示プログラムの起動に失敗しました。**エラー詳細**は，起動できない理由を示しています。

(S)

処理を終了します。

(O)

起動ができない原因を調査し，問題を解決します。

KNAX6292-E

ファイル（"**ファイル名**"）は通常のファイルではありません。

ファイル名で示すファイルは通常のファイルではありません。

(S)

処理を終了します。

(O)

ファイル名で示すファイルの種別を確認します。

KNAX6293-E

ファイル（"**ファイル名**"）の情報取得に失敗しました。reason="**エラー詳細**"

ファイル名で示すファイルの情報取得でエラーが発生しました。

(S)

処理を終了します。

(O)

ファイルの情報取得でエラーが発生した原因を調査し，問題を解決します。主な原因として，ファイルに対してアクセス権限がないことが考えられます。

KNAX6294-E

カバレージ情報表示プログラムで使用するファイル（"**ファイル名**"）のオープンに失敗しました。reason="**エラー詳細**"

カバレージ情報表示プログラムで使用するファイルのオープンでエラーが発生しました。

(S)

処理を終了します。

(O)

カバレージ情報表示プログラムで使用するファイルのオープンがエラーとなった理由を調査し，問題を解決してからカバレージ情報の表示を実行します。

KNAX6295-E

カバレージ情報表示プログラムの起動に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因によって，カバレージ表示プログラムの実行が失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行します。問題が解決しない場合は，システム管理者に連絡します。

KNAX6296-E

カバレージ情報表示プログラムで使用するファイル（"**ファイル名**"）の更新に失敗しました。reason="**エラー詳細**"

ファイル名で示すカバレージ情報表示プログラムで使用するファイルの更新でエラーが発生しました。

(S)

処理を終了します。

(O)

カバレージ情報表示プログラムで使用するファイルの書き込みができない原因を調査し，問題を解決してから再度コマンドを実行します。

KNAX6297-E

一時ディレクトリ情報の取得に失敗しました。reason="**エラー詳細**"

一時ディレクトリ情報取得でエラーが発生しました。

(S)

処理を終了します。

(O)

一時ディレクトリ情報取得でエラーが発生した原因を調査し，問題を解決します。

KNAX6298-E

カバレージ情報の表示でエラーが発生。 応答コード＝**エラー詳細**

カバレージ情報の表示でエラーが発生しました。応答コードとして表示される内容については，adshcvshowコマンドの終了コードの説明を参照してください。

(S)

処理を終了します。

(O)

カバレージ情報の表示で発生したエラーの原因を調査し，問題を解決してください。エラー発生時のメッセージがカバレージ情報蓄積ファイルに保存されている場合があります。

なお，ジョブ定義スクリプトを実行していない場合は，応答コードに6が設定されます。カバレージ情報はジョブ定義スクリプトを実行したあとに表示してください。

KNAX6301-E

カバレージ採取の一括有効化機能を使用しているときに-t オプションを指定して adshexec コマンドを実行しました。

カバレージ採取の一括有効化機能を有効にしている場合，-t オプションを指定して adshexec コマンドを実行しました。

(S)

実行を中断します。

(O)

-t オプションの指定を削除してバッチジョブを再実行します。

KNAX6302-E

カバレージ採取の一括有効化機能使用時に，使用する asc ファイル名の長さが上限を超えました。

カバレージ採取の一括有効化機能を有効にしている場合，使用する asc ファイル名の長さが上限を超えました。

asc ファイル名は環境ファイルで指定した asc ファイル名の規則で生成する場合に，生成したファイル名の長さが上限値を超えています。

(S)

実行を中断します。

(O)

環境ファイルの指定と実行するジョブ定義スクリプト名を見直してください。

KNAX6303-E

ファイル識別子の複写に失敗しました。filename="**ファイル名**" error="**エラー詳細**"

ファイル名のファイル識別子の複写に失敗しました。

ファイル名

ジョブ定義スクリプトのファイル名

エラー詳細

エラーの詳細

(S)

実行を中断します。

(O)

エラーの要因を取り除いてバッチジョブを再実行します。

KNAX6304-E

初期化処理に失敗しました。

ジョブコントローラの初期化処理に失敗しました。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルが正しいかどうかを確認します。

KNAX6305-E

ジョブ定義スクリプトファイル（"**ファイル名**"）の存在を確認できません。error="**エラー詳細**"

ジョブ定義スクリプトファイルの存在を確認できません。

ファイル名

ジョブ定義スクリプトのファイル名

エラー詳細

エラーの詳細

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルが正しいかどうかを確認します。

KNAX6306-E

ジョブ定義スクリプトファイル（"**ファイル名**"）の読み込みに失敗しました。func=**関数名** error="**エラー詳細**"

ジョブ定義スクリプトファイルの読み込みに失敗しました。

ファイル名

ジョブ定義スクリプトのファイル名

関数名

エラーが発生した関数名

エラー詳細

エラーの詳細

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルが正しいかどうかを確認します。

KNAX6307-W

表示処理で使用するジョブ定義スクリプトファイル（"**ファイル名**"）の 1 行に指定できる上限を超えています。line=**行番号**

表示処理で使用するジョブ定義スクリプトの 1 行に指定できる上限を超えています。超えた部分は切り捨てて表示します。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

超えた部分は切り捨てて表示し，処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正します。

KNAX6308-E

ジョブ定義スクリプトファイル（"**ファイル名**"）は空のファイルです。

ジョブ定義スクリプトファイルには何もありません。

ファイル名

ジョブ定義スクリプトのファイル名

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを修正します。

KNAX6309-E

スプールのジョブ定義スクリプトファイルの出力に失敗しました。reason="**エラー詳細**"

スプールのジョブ定義スクリプトファイルの出力に失敗しました。

エラー詳細

エラーの詳細

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを修正します。

KNAX6310-E

スクリプト拡張コマンドのオプション名または位置オペランド名（"**項目名**"）が正しくありません。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドに指定した**項目名**は正しくありません。

項目名

スクリプト拡張コマンドのオプション名または位置オペランド名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6311-E

スクリプト拡張コマンドに項目（"**項目名**"）が指定されていません。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドに**項目名**が指定されていません。

項目名

スクリプト拡張コマンドのオプション名または位置オペランド名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6312-E

スクリプト拡張コマンドのオプション名に指定した値（"**オプション名**"）が不正です。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドの**オプション名**に指定した値が不正です。

オプション名

スクリプト拡張コマンドのオプション名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6313-E

スクリプト拡張コマンドのオプション（"**オプション名**"）を複数指定しています。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドのオプション名を複数指定しています。

オプション名

スクリプト拡張コマンドのオプション名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6314-E

スクリプト拡張コマンドの "**文字列**" が完結していません。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドの**文字列**で示す記述は完結していません。ダブルクォーテーションまたはシングルクォーテーションの指定が閉じていないか，¥の直後にエスケープする文字がない可能性があります。

文字列

スクリプト拡張コマンドの記述

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6315-E

スクリプト拡張コマンド（"**コマンド名**"）の指定が不正です。filename="**ファイル名**" line=**行番号**

次のどれかの要因が考えられます。

- スクリプト拡張コマンドの指定位置が不正です。
- スクリプト拡張コマンドはほかのコマンドとの組み合わせが不正です。
- スクリプト拡張コマンドが先頭に記述されていません。

コマンド名

スクリプト拡張コマンド名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

構文解析時ではなく実行時にエラーが検出された場合は，行番号が 0 となることがあります。実行時にエラーが検出される例としては，コマンド置換の書式としてスクリプト拡張コマンドを使用した場合などがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6316-E

スクリプト拡張コマンドの長さが上限値を超えました。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドの長さが上限値を超えました。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6317-E

継続行の指定が不正です。filename="**ファイル名**" line=**行番号**

継続行の指定が不正です。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6318-E

不当なスクリプト拡張コマンド名（"**コマンド名**"）が指定されました。filename="**ファイル名**" line=**行番号**

不当なスクリプト拡張コマンド名が指定されました。

コマンド名

スクリプト拡張コマンド名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6319-E

指定した "**項目名**" の数が上限を超えました。filename="**ファイル名**" line=**行番号**

項目名を指定できる上限値を超えました。

項目名

スクリプト拡張コマンド，オプション名または引数

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6320-E

ジョブ定義スクリプトファイル（"**ファイル名**"）でエラーが発生しました。function=**関数名** error="**エラー詳細**"

ジョブ定義スクリプトファイルでエラーが発生しました。

ファイル名

ジョブ定義スクリプトのファイル名

関数名

エラーが発生した関数名

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6321-E

"-run {abnormal|always}" は現在の位置には指定できません。filename="**ファイル名**" line=**行番号**

ここでは指定できないオプションを指定しています。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6323-E

export パラメーターの指定が不正です。または，環境変数を設定できません。line=**行番号**

export パラメーターの指定が不正です。または，環境変数を設定できません。

行番号

環境ファイルの行番号

エラーの要因として次のどれかが考えられます。

- 環境変数名，環境変数値の長さが上限を超えています。

  PATH 環境変数にパスを追加している場合，子孫ジョブでは二重にパスが追加されるため，上限を超えていないか注意が必要です。
- 環境変数名が不正です。
- 「¥」「"」「'」の組み合わせが正しくありません。

(S)

処理を終了します。

(O)

環境ファイルを修正します。

KNAX6324-E

"**変換前ファイル名**" を絶対パスに変換できません。error="**エラー詳細**" filename="**ファイル名**"line=**行番号**

変換前ファイル名を絶対パスに変換できません。

変換前ファイル名

変換前のファイル名

エラー詳細

エラーの詳細

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6325-E

ブロック内または関数定義内のジョブステップが終了しないまま，ブロックまたは関数定義が終了しました。filename="**ファイル名**" line=**行番号**

ブロック内または関数定義内のジョブステップが終了しないまま，ブロックまたは関数定義が終了しました。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6326-E

ジョブステップの開始定義がありません。filename="**ファイル名**" line=**行番号**

ジョブステップの開始定義がありません。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6327-E

ジョブステップの定義が階層になっています。filename="**ファイル名**" line=**行番号**

ジョブステップの定義が階層になっています。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6328-E

"**コマンド名**" の定義が階層になっています。filename="**ファイル名**" line=**行番号**

コマンド名の定義が階層になっています。

コマンド名

スクリプト拡張コマンドのコマンド名

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6329-E

スクリプト拡張コマンドが途中から変更されている可能性があります。filename="**ファイル名**" line=**行番号**

スクリプト拡張コマンドが途中から変更されている可能性があります。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトが途中で変更されないことを確認して再実行します。

KNAX6330-E

ジョブ定義スクリプトを再帰的に呼び出しています。filename="**ファイル名**" line=**行番号**

ジョブ定義スクリプトを再帰的に呼び出しています。

ファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6332-E

ジョブ定義スクリプトが終了しましたが，閉じていないジョブステップがあります。

ジョブ定義スクリプトが終了しましたが，閉じていないジョブステップがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6333-E

"**ファイル名**" の存在が確認できません。error="**エラー詳細**" filename="**ジョブ定義スクリプトファイル名**" line=**行番号**

ファイル名の存在が確認できません。

エラー詳細

エラーの詳細

ジョブ定義スクリプトファイル名

ジョブ定義スクリプトのファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6380-I

ルートジョブのスプールジョブディレクトリにジョブ名を付加します。spool job directory="*スプールジョブディレクトリ*"

ルートジョブのスプールジョブディレクトリにジョブ名を付加します。このジョブ名はスプールジョブ名です。スプールジョブ名は JP1/Advanced Shell のジョブ名を使用しますが，シェル変数 ADSH_SPOOL_JOBNAME でスプールジョブ名を指定した場合は，シェル変数ADSH_SPOOL_JOBNAME で指定したスプールジョブ名を使用します。

スプールジョブディレクトリ

変更後のスプールジョブディレクトリの名称

(S)

処理を続行します。

KNAX6381-E

スプールジョブディレクトリの名称の変更処理が失敗しました。error="*エラー詳細*" jobid="*ジョブ識別子*" jobname="*ジョブ名*"

スプールジョブディレクトリの名称の変更処理が失敗しました。スプールジョブディレクトリはジョブ識別子のままです。次の原因が考えられます。

- 変更後の名称のスプールジョブディレクトリがすでに存在しています。
- Windows では，子孫プロセスを生成する外部コマンドを実行するジョブを強制終了した場合，孫以下のプロセスが同時に 256 個以上存在すると，エラー詳細が`Permission denied` でこのメッセージが出力されることがあります。
- ディレクトリ名として使用できない文字を使用しています。
- ディレクトリ名として長過ぎる名称を使用しています。

エラー詳細

エラーの詳細

ジョブ識別子

ジョブ識別子

ジョブ名

JP1/Advanced Shell のジョブ名またはシェル変数`ADSH_SPOOL_JOBNAME` で指定したスプールジョブ名

(S)

処理を続行します。

(O)

スプールジョブディレクトリを参照する場合は，同時に出力される`KNAX6382-I` で示すスプールジョブディレクトリを参照してください。

スプールジョブディレクトリを削除する場合は，`adshhk` コマンドではなく手動で削除してください。

KNAX6382-I

名称変更処理が失敗したため，変更前のスプールジョブディレクトリに保管しました。
spool job directory="**スプールジョブディレクトリ**"

スプールジョブディレクトリの名称変更処理が失敗したため，スプールジョブディレクトリに保管しました。

スプールジョブディレクトリ

変更前のスプールジョブディレクトリの名称

(S)

処理を続行します。

KNAX6383-E 【UNIX 限定】

スプールジョブのディレクトリまたはファイルのパーミッション変更処理に失敗しました。("**パス名**", "**機能名**", "**エラー詳細**")

パス名で示すスプールジョブのディレクトリまたはファイルのパーミッション変更処理に失敗しました。

機能名

OS の API 名

エラー詳細

errno で示されるエラーの詳細

(S)

機能名が chmod の場合は処理を継続します。それ以外は処理を終了します。

(O)

原因を調査して対策します。また，このスプールジョブのディレクトリまたはファイルのパーミッションは，chmod コマンドを実行して変更します。

KNAX6385-E

このスクリプト拡張コマンドは，現在の環境設定パラメーターの設定では使用できません。command name="**コマンド名**" parameter="**環境設定パラメーター**" filename="**ファイル名**" line=**行番号**

メッセージに表示された拡張スクリプトコマンドは，現在の環境設定パラメーターの設定では使用できません。

コマンド名

エラーとなった拡張スクリプトコマンドの名称

環境設定パラメーター

要因となった環境設定パラメーター名と値

ファイル名

ジョブ定義スクリプトファイル名

行番号

ジョブ定義スクリプトの行番号

継続行にエラーがある場合，その継続行の行番号ではなく，スクリプト拡張コマンドの先頭行を示すことがあります。

(S)

処理を終了します。

(O)

実行したコマンドは使用しないでください。このコマンドを使用したい場合は環境設定パラメーターの指定を見直してください。

KNAX6340-E

adshcmdrc: コマンドラインの指定に誤りがあります。details="**詳細**"

コマンドラインの指定に誤りがあります。メッセージに表示される**詳細**の表示内容と，その意味を次に示します。

詳細の表示内容	**意味**
コマンド名が指定されていません。	コマンド名が指定されていません。
コマンド名が長過ぎます。	コマンド名が長過ぎます。
コマンド名に空文字列が指定されています。	コマンド名に空文字列が指定されています。
コマンド名に不当な指定がされています。	コマンド名に不当な指定がされています。コマンドのパスが指定されている可能性があります。
しきい値が指定されていません。	しきい値が指定されていません。
しきい値の指定値が範囲外です。	しきい値の指定値が範囲外です。
しきい値に数字以外を指定しています。	しきい値に数字以外を指定しています。
引数の指定が多過ぎます。	引数の指定が多過ぎます。

(S)

処理を継続します。

(O)

コマンドを正しく指定して再実行してください。

KNAX6341-E

指定したコマンド（"**コマンド名**"）の数が上限を超えました。line=**行番号**

コマンド名で示すコマンドを指定できる上限値を超えました。

コマンド名

コマンド名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX6342-E

指定したコマンド（"**コマンド名**"）の数が上限を超えました。（**機能名**）

コマンド名

コマンド名

機能名

コマンドを実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6399-E

予期しない障害が発生しました。**関数**, **行番号**

予期しない障害が発生しました。

関数

関数名

行番号

行番号

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX6400-E

ファイル環境変数定義名（"**ファイル環境変数定義名**"）のファイルの割り当てに失敗しました。

ファイル環境変数定義名で示すファイルの割り当てに失敗しました。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6401-E

ファイル環境変数定義名（"**ファイル環境変数定義名**"）で指定したファイル（**ファイルパス**）が存在しません。

ファイル環境変数定義名で指定した**ファイルパス**のファイルがありません。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6403-E

"**一時ファイル識別名**" と同じ識別名のファイルがすでに定義されています。

一時ファイル識別名と同じ識別名のファイルがすでに定義されています。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6404-E

環境変数（"**ファイル環境変数定義名**"）の作成に失敗しました。

ファイル環境変数定義名で示す環境変数の作成に失敗しました。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6405-E

ファイル環境変数定義名（"**ファイル環境変数定義名**"）で示すファイルが存在しません。

ファイル環境変数定義名で示すファイルがありません。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6406-E

ファイル環境変数定義名（"**ファイル環境変数定義名**"）で示すファイルでエラーが発生しました。
reason="**エラー詳細**"

ファイル環境変数定義名で示すファイルを stat 関数で確認しようとした際，**エラー詳細**で示すエラーが発生しました。

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6407-E

ファイル環境変数定義名（"**ファイル環境変数定義名**"）で示すディレクトリはすでに存在します。

ファイル環境変数定義名で示すディレクトリはすでにあります。

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6408-E

ファイル環境変数定義名（"**ファイル環境変数定義名**"）で示すファイルが作成できません。reason="**エラー詳細**"

ファイル環境変数定義名で示すファイルを作成しようとした場合，**エラー詳細**で示すエラーが発生しました。

エラー詳細

エラーの詳細。errno で表されるエラー情報文字列

(S)

処理を終了します。

(O)

ジョブ定義スクリプトファイルを見直してください。

KNAX6409-I

ファイル環境変数定義名（**ファイル環境変数定義名**）へ処理指定（"**処理指定値**"）に従ってファイルを割り当てました。path=**ファイルパス**[(**ファイル存在有無表示**)]

ファイル環境変数定義名で示すファイル定義で，**処理指定値**に従って**ファイルパス**で示すファイルを割り当てました。

処理指定値

コマンドに応じて次の内容が表示されます。

#-adsh_file コマンドまたは#-adsh_file_temp コマンドの場合

-chk の指定値

#-adsh_spoolfile コマンドの場合

spoolfile

ファイル存在有無表示

通常ファイルの場合だけ出力されます。ファイルが存在する場合は「ファイルは存在します」を，ファイルが存在しない場合は「ファイルは存在しません」を出力します。

(S)

処理を続行します。

KNAX6410-I

ファイル環境変数定義名（**ファイル環境変数定義名**）で割り当てたファイルを処理指定（"**処理指定値**"）に従って解放しました。path=**ファイルパス**

ファイル環境変数定義名で示すファイル定義で，**処理指定値**に従って**ファイルパス**で示すファイルを解放しました。

処理指定値

コマンドに応じて次の内容が表示されます。

#-adsh_file コマンドまたは#-adsh_file_temp コマンドの場合

正常時・異常時のファイルの後処理の指定内容として，次のどちらかが表示されます。

・del：削除する

・keep：削除しない

#-adsh_spoolfile コマンドの場合

spoolfile

(S)

処理を続行します。

KNAX6411-E

シェル変数（"**ファイル環境変数定義名**"）の設定に失敗しました。details=**保守情報**

ファイル環境変数定義名で示すシェル変数の設定に失敗しました。

(S)

処理を終了します。

(O)

ファイル環境変数定義名で示すシェル変数が読み込み専用に設定されている可能性があるため，ジョブ定義スクリプトファイルを見直してください。見直して問題のない場合は，システム管理者に連絡してください。

KNAX6412-E

シェル変数の取得に失敗しました。details=**保守情報**

シェル変数の取得に失敗しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX6413-E

シェル変数（"**ファイル環境変数定義名**"）を元の値に回復できないか，または元の値の定義がなかった場合に現在の定義を削除できません。details=**保守情報**

ファイル環境変数定義名で示すシェル変数を元の値に回復できないか，または元の値の定義がなかった場合に現在の定義を削除できません。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX6414-E

ファイルパスの正規化処理でエラーが発生しました。name=**API名称** reason="**原因**" details=**保守情報** filename="**スクリプトファイル名**" line=**行番号**

スクリプト拡張コマンド#adsh_file コマンドで指定したファイルパスの正規化処理でエラーが発生しました。

API名称

エラーが発生した API 名称

原因

エラー内容。出力される内容と意味を次の表に示します。

項番	出力内容	意味
1	errno で表されるエラー情報文字列	UNIX または Windows のマニュアルを参照してください。
2	パス名に不当なマルチバイト文字が含まれています。	パス名に不当なマルチバイト文字が含まれています。
3	パス構成要素が多過ぎます。	パス構成要素が多過ぎます（構成要素が 4,096 個を超えました）。

スクリプトファイル名

スクリプトファイルのファイル名

行番号

スクリプトの行番号

(S)

処理を終了します。

(O)

原因を参照してエラーの原因を取り除きます。問題が解決しない場合はシステム管理者に連絡してください。

KNAX6501-I

ジョブは初期設定スクリプトファイル（"*初期設定スクリプトファイル名*"）を実行します。

ジョブは，*初期設定スクリプトファイル名*で示す初期設定スクリプトファイルを実行します。

初期設定スクリプトファイル名

実行する初期設定スクリプトファイル名

(S)

処理を続行します。

KNAX6502-I

初期設定スクリプトは終了しました。

初期設定スクリプトが終了しました。

(S)

処理を続行します。

KNAX6503-E

初期設定スクリプトファイルからルートジョブは起動できません。

初期設定スクリプトファイルからルートジョブを起動しようとしているため，ルートジョブの起動に失敗しました。

(S)

処理を終了します。

(O)

初期設定スクリプトの内容を見直し，ルートジョブを起動している個所を取り除いてください。

KNAX6504-E

初期設定スクリプト（"*初期設定スクリプトファイル名*"）は実行できません。reason="*エラー詳細*"

*エラー詳細*に示す原因で，初期設定スクリプトは実行できません。

初期設定スクリプトファイル名

実行できなかった初期設定スクリプトファイル名

(S)

処理を終了します。

(O)

*エラー詳細*の原因を取り除いて再実行してください。問題が解決しない場合はシステム管理者に連絡してください。

KNAX6507-I

ジョブ名.ジョブステップ名 run 属性で指定した条件が成立したため，ジョブステップをスキップしました。

#-adsh_step で始まるジョブステップ定義コマンドの run 属性で指定した条件が成立したため，**ジョブ名**で示すバッチジョブに定義された，**ジョブステップ名**で示すジョブステップをスキップしました。

(S)

処理を続行します。

KNAX6508-I

ジョブ名.ジョブステップ名 先行のジョブステップまたはコマンドがエラー終了したため，ジョブステップをスキップしました。

先行のジョブステップまたはコマンドがエラー終了したため，**ジョブ名**で示すバッチジョブに定義された，**ジョブステップ名**で示すジョブステップをスキップしました。

(S)

処理を続行します。

KNAX6509-I

ジョブ名.ジョブステップ名 ジョブ定義スクリプトの制御内容により，ジョブステップは実行されませんでした。

ジョブ名で示すバッチジョブに定義された**ジョブステップ名**で示すジョブステップは，if 制御文などによるジョブ定義スクリプトの制御で実行されていません。

(S)

処理を続行します。

KNAX6510-W

シェル変数（**シェル変数名**）を設定しようとしましたが，読み取り専用属性のため設定できません。
filename="**ファイル名**" line=**行番号**

ジョブコントローラが**ファイル名**で示すファイル中の**行番号**で示すコマンドの実行中に**シェル変数名**で示すシェル変数を設定しようとしましたが，読み取り専用属性のため設定できません。

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを見直し，シェル変数名で示すシェル変数の使用方法に誤りがあれば修正します。

KNAX6511-I

ジョブステップ内で有効なシェル変数 PATH は，ジョブステップの外で定義されたシェル変数 PATH から値を引き継ぎます。stepname=**ジョブステップ名**

ジョブステップ名のジョブステップに指定されたジョブステップ内で有効なシェル変数 PATH は，ジョブステップの外で定義されたシェル変数 PATH から値を引き継ぎます。

(S)

処理を続行します。

KNAX6512-I

ジョブコントローラはカスタムジョブから起動されました。

ジョブコントローラが JP1/Advanced Shell のカスタムジョブから JP1/AJS によって起動されました。

(S)

処理を続行します。

KNAX6521-E

コマンド（**コマンド名**, 行番号=**行数**）がエラー終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドがエラーで終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているジョブ定義スクリプトの行数

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間。OS の API を使って取得した参考値です。

CPU時間

コマンドの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。エラーとなったコマンドがジョブステップ内のコマンドの場合，次のように動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

ただし，コマンドの実行結果によってジョブを中断する必要がある場合は，上記の動作とならないで KNAX6584-I を出力して処理を終了します。

(O)

コマンドの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6522-E

コマンド（**コマンド名**, 行番号=**行数**）がシグナル受信によってエラー終了しました。rc=**終了コード** signalNo=**シグナル番号** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドがシグナルによってエラー終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

行数

コマンドが記述されているジョブ定義スクリプトの行数

終了コード

コマンドの終了コード

シグナル番号

コマンドが受信したシグナル番号

実行時間

コマンドの実行時間。OS の API を使って取得した参考値です。

CPU時間

コマンドの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。エラー終了したコマンドがジョブステップ内のコマンドである場合，次のとおり動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

(O)

コマンドの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6530-E

オプションの有無およびオプションの値の組み合わせに誤りがあります。filename="**ファイル名**" line=**行番号**

オプションの有無およびオプションの値の組み合わせに誤りがあります。

ファイル名

ジョブ定義スクリプトファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6531-E

"**項目名**" の指定が**スコープ**で指定できる上限値を超えました。filename="**ファイル名**" line=**行番号**

項目名が指定できる上限値を超えました。

項目名

スクリプト拡張コマンド名

スコープ

ジョブまたはステップ

ファイル名

ジョブ定義スクリプトファイルのファイル名

行番号

ジョブ定義スクリプトの行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6540-I

別プロセスで実行したコマンド群が正常終了しました。line=**行番号** rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドのグループ化などによって別プロセスで実行したジョブ定義スクリプトが正常終了しました。

行番号

別プロセスで実行したジョブ定義スクリプトが記述されているジョブ定義スクリプトの行番号

終了コード

実行したジョブ定義スクリプトの終了コード

実行時間

実行したジョブ定義スクリプトの実行時間。OS の API を使って取得した参考値です。

CPU時間

実行したジョブ定義スクリプトの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。

KNAX6541-E

別プロセスで実行したコマンド群がエラー終了しました。line=**行番号** rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドのグループ化などによって別プロセスで実行したジョブ定義スクリプトがエラー終了しました。

行番号

別プロセスで実行したジョブ定義スクリプトが記述されているジョブ定義スクリプトの行番号

終了コード

別プロセスで実行したジョブ定義スクリプトの終了コード

実行時間

別プロセスで実行したジョブ定義スクリプトの実行時間。OS の API を使って取得した参考値です。

CPU時間

別プロセスで実行したジョブ定義スクリプトの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。別プロセスで実行してエラー終了したジョブ定義スクリプトがジョブステップ内のジョブ定義スクリプトである場合，次のように動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

(O)

別プロセスで実行してエラー終了したジョブ定義スクリプトの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6542-E

別プロセスで実行したコマンド群がシグナル受信によってエラー終了しました。line=**行番号** rc=**終了コード** signalNo=**シグナル番号** E-Time=**実行時間** s C-Time=**CPU時間** s

コマンドのグループ化などによって別プロセスで実行したジョブ定義スクリプトがシグナルによってエラー終了した。

行番号

別プロセスで実行したジョブ定義スクリプトが記述されているジョブ定義スクリプトの行番号

終了コード

別プロセスで実行したジョブ定義スクリプトの終了コード

シグナル番号

別プロセスで実行したジョブ定義スクリプトが受信したシグナル番号

実行時間

別プロセスで実行したジョブ定義スクリプトの実行時間。OS の API を使って取得した参考値です。

CPU時間

別プロセスで実行したジョブ定義スクリプトの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。別プロセスで実行してエラー終了したジョブ定義スクリプトがジョブステップ内のジョブ定義スクリプトである場合，次のように動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

(O)

別プロセスで実行してエラー終了したジョブ定義スクリプトの実行結果を確認し，異常となった原因を取り除きます。

KNAX6551-E

コマンド（**コマンド名**, **機能名**）がエラー終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドがエラー終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名。次のどちらかが出力されます。

コマンド置換：コマンド置換機能

trap コマンドのアクション：trap コマンドのアクション

終了コード

コマンドの終了コード

実行時間

コマンドの実行時間。OS の API を使って取得した参考値です。

CPU時間

コマンドの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。エラーとなったコマンドがジョブステップ内のコマンドの場合，次のとおり動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

ただし，コマンドの実行結果によってバッチジョブを中断する必要がある場合は，上記の動作とならないで KNAX6584-I を出力して処理を終了します。

(O)

コマンドの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6552-E

コマンド（**コマンド名**, **機能名**）がシグナル受信によってエラー終了しました。rc=**終了コード** signalNo=**シグナル番号** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行されたコマンドがシグナルによってエラー終了しました。

コマンド名

実行したコマンド名。OS が定める最大パス長を超える場合，最大パス長で打ち切られます。

機能名

コマンドを実行した機能名。次のどちらかが出力されます。

コマンド置換：コマンド置換機能

trap コマンドのアクション：trap コマンドのアクション

終了コード

コマンドの終了コード

シグナル番号

コマンドが受信したシグナル番号

実行時間

コマンドの実行時間。OS の API を使って取得した参考値です。

CPU時間

コマンドの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。エラー終了したコマンドがジョブステップ内のコマンドの場合，次のとおり動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。

- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

(O)

コマンドの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6560-I

機能名 から別プロセスで実行したコマンド群が正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で，コマンドのグループ化などによって別プロセスで実行したジョブ定義スクリプトが正常終了しました。

機能名

コマンドを実行した機能名。次のどちらかが出力されます。

コマンド置換：コマンド置換機能

trap コマンドのアクション：trap コマンドのアクション

終了コード

実行したジョブ定義スクリプトの終了コード

実行時間

実行したジョブ定義スクリプトの実行時間。OS の API を使って取得した参考値です。

CPU時間

実行したジョブ定義スクリプトの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。

KNAX6561-E

機能名 から別プロセスで実行したコマンド群がエラー終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で，コマンドのグループ化などによって別プロセスで実行したジョブ定義スクリプトがエラー終了しました。

機能名

コマンドを実行した機能名。次のどちらかが出力されます。

コマンド置換：コマンド置換機能

trap コマンドのアクション：trap コマンドのアクション

終了コード

別プロセスで実行したジョブ定義スクリプトの終了コード

実行時間

別プロセスで実行したジョブ定義スクリプトの実行時間。OS の API を使って取得した参考値です。

CPU時間

別プロセスで実行したジョブ定義スクリプトの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。別プロセスで実行してエラー終了したジョブ定義スクリプトがジョブステップ内のジョブ定義スクリプトの場合，次のとおり動作します。

- ジョブステップの onError 属性が stop のとき，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont のとき，ジョブステップ正常ブロックの後続処理を実行します。

(O)

別プロセスで実行してエラー終了したジョブ定義スクリプトの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6562-E

機能名 から別プロセスで実行したコマンド群がシグナル受信によってエラー終了しました。rc=**終了コード** signalNo=**シグナル番号** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で，コマンドのグループ化などで別プロセスで実行したジョブ定義スクリプトがシグナルの受信によってエラー終了しました。

機能名

コマンドを実行した機能名。次のどちらかが出力されます。

コマンド置換：コマンド置換機能

trap コマンドのアクション：trap コマンドのアクション

終了コード

別プロセスで実行したジョブ定義スクリプトの終了コード

シグナル番号

別プロセスで実行したジョブ定義スクリプトが受信したシグナル番号

実行時間

別プロセスで実行したジョブ定義スクリプトの実行時間。OS の API を使って取得した参考値です。

CPU時間

別プロセスで実行したジョブ定義スクリプトの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。別プロセスで実行してエラー終了したジョブ定義スクリプトがジョブステップ内のジョブ定義スクリプトの場合，次のとおり動作します。

- ジョブステップの onError 属性が stop のとき，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont のとき，ジョブステップ正常ブロックの後続処理を実行します。

(O)

別プロセスで実行してエラー終了したジョブ定義スクリプトの実行結果を確認し，異常となった原因を取り除きます。

KNAX6571-I

子孫ジョブ名 子孫ジョブを開始しました。parent jobname=**親プロセスのジョブ名**, parent jobid=**親プロセスのジョブ識別子**

親プロセスのジョブ名，**親プロセスのジョブ識別子**で示すジョブから起動した，**子孫ジョブ名**で示す子孫ジョブを開始しました。

(S)

処理を続行します。

KNAX6572-I

子孫ジョブ（**子孫ジョブ名**）はジョブ環境ファイル（"**ジョブ環境ファイル名**"）を使用します。

子孫ジョブ名で示す子孫ジョブは，**ジョブ環境ファイル名**で示すジョブ環境ファイルを使用します。

(S)

処理を続行します。

KNAX6578-I

子孫ジョブ名 子孫ジョブが終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

子孫ジョブ名で示す子孫ジョブが終了しました。

終了コード

子孫ジョブの実行結果を示す終了コード。

終了コードの詳細は，adshexec コマンドの終了コードに関する説明を参照してください。

なお，このメッセージの出力後に adshexec コマンドの後処理でエラーが発生した場合，このメッセージで示す終了コードと adshexec コマンドの終了コードとが異なるときがあります。adshexec コマンドの最終的な終了コードは，親のジョブの JOBLOG に出力されます。

実行時間

子孫ジョブの開始から終了までの実時間の合計（秒単位）。OS の API を使って取得した参考値です。

CPU時間

子孫ジョブの開始から終了までの CPU 時間の合計（秒単位）。OS の API を使って取得した参考値です。

(S)

処理を続行します。

KNAX6580-E

関数（**関数名**, 行番号=**行数**）がエラー終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

関数がエラーで終了しました。

関数名

実行した関数名。

行数

関数が記述されているジョブ定義スクリプトの行数

終了コード

関数の終了コード

実行時間

関数の実行時間。OS の API を使って取得した参考値です。

CPU時間

関数の CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。エラーとなった関数がジョブステップ内の関数の場合，次のように動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。

- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

(O)

関数の実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6581-E

関数（**関数名**, **機能名**）がエラー終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

機能名で示す機能で実行された関数がエラー終了しました。

関数名

実行した関数名。

機能名

関数を実行した機能名。次のどちらかが出力されます。

- コマンド置換：コマンド置換機能
- trap コマンドのアクション：trap コマンドのアクション

終了コード

関数の終了コード

実行時間

関数の実行時間。OS の API を使って取得した参考値です。

CPU時間

関数の CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。エラーとなった関数がジョブステップ内の関数の場合，次のように動作します。

- ジョブステップの onError 属性が stop の場合，ジョブステップ正常ブロックの後続処理は実行しないでジョブステップエラーブロックを実行します。
- ジョブステップの onError 属性が cont の場合，ジョブステップ正常ブロックの後続処理を実行します。

(O)

関数の実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6582-E

関数（**関数名**, 行番号=**行数**）の実行を中断しました。

次の要因により関数の実行を中断しました。

- 関数実行中にジョブコントローラがシグナルを受信してエラー終了した
- 関数実行中にジョブコントローラが TerminateProcess などによるプロセス即時終了によりエラー終了した
- 関数実行中にジョブコントローラが制御信号を受信してエラー終了した
- 特殊組み込みコマンドのエラー，およびジョブコントローラが処理を続行できないと判断されるエラーが関数内で発生した（ただし，typeset コマンドのエラー，関数内または外部スクリプト内で実行した return コマンドのエラーを除く）
- 関数実行中に CUI デバッガの kill コマンドによりジョブ定義スクリプトを終了した
- 関数実行中に CUI デバッガの quit コマンドによりデバッガを終了した
- 関数実行中に GUI デバッガによりデバッグを中止した

関数名

実行した関数名。

行数

関数が記述されているジョブ定義スクリプトの行数。

(S)

処理を中断します。

(O)

エラーが原因で関数の実行が中断された場合は，障害を取り除いて再実行します。

KNAX6583-E

関数（**関数名**, **機能名**）の実行を中断しました。

機能名で示す機能で実行された関数が，次の要因により実行を中断しました。

- 関数実行中にジョブコントローラがシグナルを受信してエラー終了した
- 関数実行中にジョブコントローラが TerminateProcess などによるプロセス即時終了によりエラー終了した
- 関数実行中にジョブコントローラが制御信号を受信してエラー終了した

- 特殊組み込みコマンドのエラー，およびジョブコントローラが処理を続行できないと判断されるエラーが関数内で発生した（ただし，typeset コマンドのエラー，関数内または外部スクリプト内で実行した return コマンドのエラーを除く）
- 関数実行中に CUI デバッガの kill コマンドによりジョブ定義スクリプトを終了した
- 関数実行中に CUI デバッガの quit コマンドによりデバッガを終了した
- 関数実行中に GUI デバッガによりデバッグを中止した

関数名

実行した関数名。

機能名

関数を実行した機能名として，次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

(S)

処理を中断します。

(O)

エラーが原因で関数の実行が中断された場合は，障害を取り除いて再実行します。

KNAX6584-I

ジョブ定義スクリプトの実行を停止するコマンドが実行されたため，バッチジョブの実行を中断しました。

ジョブ定義スクリプトの実行を停止するコマンドが実行されたため，バッチジョブの実行を中断しました。後続ジョブステップ・後続のジョブ定義スクリプトは実行しません。run 属性が abnormal や always のジョブステップについても実行しません。

また，この場合，実行したコマンドに対しては，#-adsh_rc_ignore コマンドの指定，および#-adsh_step_start コマンドの successRC 属性の指定は有効になりません。

このメッセージは，次のどれかの場合に出力されます。

- exit コマンドを実行して，ジョブ定義スクリプトが即時終了した場合
- 関数外かつ外部スクリプトでない位置での return コマンドを実行して，ジョブ定義スクリプトが即時終了した場合
- exec コマンドの引数に実行可能なコマンドを指定して実行し，スクリプトが終了した場合

- 特殊組み込みコマンドのエラー，およびジョブコントローラが処理を続行できないと判断されるエラーが発生した場合（ただし，typeset コマンドのエラー，関数内または外部スクリプト内で実行した return コマンドのエラーを除く）

(S)

バッチジョブの実行は中断します。

(O)

エラーが原因でバッチジョブの実行が中断された場合は，障害を取り除いてバッチジョブを再実行します。

KNAX6585-I

ジョブステップの終了コードが#-adsh_job_stop コマンドで指定した条件を満たしているため，バッチジョブの実行を中断しました。

ジョブステップの終了コードが#-adsh_job_stop コマンドで指定した条件を満たしているため，バッチジョブの実行を中断しました。

(S)

処理を終了します。

(O)

問題が発生している場合，ジョブステップの実行結果を参照して原因を取り除き，バッチジョブを再実行します。

KNAX6586-E

ジョブ定義スクリプトの実行を継続できないエラーが発生したため，バッチジョブの実行を中断しました。

ジョブ定義スクリプトの実行を継続できないエラーが発生したため，バッチジョブの実行を中断しました。後続ジョブステップ・後続のジョブ定義スクリプトは実行しません。run 属性が abnormal や always のジョブステップについても実行しません。

(S)

ジョブの実行を中断します。

(O)

エラーの原因を取り除いてジョブを再実行します。

KNAX6587-E

起動された子孫ジョブの数が上限値を超えました。

起動された子孫ジョブの数が上限値を超えました。

(S)

処理を終了します。

(O)

1 つのルートジョブの中で起動する子孫ジョブ（子孫ジョブから起動する子孫ジョブを含む）の数が 9,999,999 個以下になるように，ジョブ定義スクリプトファイルを見直してください。

KNAX6588-E

API error occurred. name=**API名称**, reason=**エラー詳細**

API名称で示す API の呼び出しで，**エラー詳細**に示すエラーが発生しました。

(S)

処理を終了します。

(O)

原因および一緒に出力されるほかのメッセージを参照してエラーの原因を取り除きます。問題が解決しない場合，システム管理者に連絡します。

KNAX6589-W

エラーが発生しましたが，ジョブを続行します。name=**API名称**, reason=**原因**, details=**保守情報**

API名称で示す API の呼び出しで，**原因**に示すエラーが発生しましたが，ジョブを続行します。次の原因が考えられます。

- **API名称**が CreateFile の場合，標準入力，標準出力，および標準エラー出力が使用できない環境で adshexec コマンドが実行された可能性があります。
- **API名称**が isatty の場合，標準入力が使用できない環境で adshexec コマンドが実行された可能性があります。

(S)

処理を続行します。

(O)

ジョブの動作に問題が発生していない場合，対処は不要です。問題が発生している場合，エラーの原因を取り除き，バッチジョブを再実行します。問題が解決しない場合は，システム管理者に連絡します。

KNAX6590-E

機能名 でエラーが発生しました。details=**保守情報**

機能名で示す機能が失敗しました。

機能名

エラーが発生した関数の機能名。次のどれかが出力されます。

シェル変数の取得に関する機能の場合：get variable

シェル変数の設定に関する機能の場合：set variable

シェル変数の設定削除に関する機能の場合：unset variable

外部スクリプトの関する機能の場合：run external script

ジョブ定義スクリプトの終了コード関する機能の場合：set rc

ファイルオープンに関する機能の場合：open file

(S)

エラーが発生した機能に応じて，処理を続行，または終了します。

(O)

システム管理者に連絡します。

KNAX6591-E

終了要求シグナルを受信したため，ジョブコントローラが強制終了しました。signalNo=**シグナル番号**

ジョブ実行中に**シグナル番号**で示す終了要求シグナルを受信したため，ジョブコントローラが強制終了しました。

(S)

ジョブコントローラは起動中の子孫プロセスを強制終了し，ファイルの削除などの必要な後処理を実行してから終了します。

(O)

ジョブが強制終了されていることを確認します。

KNAX6592-E

制御信号を受信したため，ジョブコントローラが強制終了しました。ctrlType=**制御種別**

ジョブ実行中に**制御種別**で示す制御信号を受信したため，ジョブコントローラが強制終了しました。**制御種別**には，次のどれかの文字列が出力されます。

- CTRL + C 信号を受け取った場合：CTRL + C
- CTRL + BREAK 信号を受け取った場合：CTRL + BREAK
- ユーザーがコンソールを閉じた場合（CTRL + C の受信時）：CLOSE EVENT

(S)

ジョブコントローラは起動中の子孫プロセスを強制終了し，ファイルの削除などの必要な後処理を実行してから終了します。

(O)

ジョブが強制終了されていることを確認します。

KNAX6593-E

ジョブコントローラが強制終了の要求を受け取りました。

ジョブコントローラが強制終了の要求を受け取りました。

(S)

ジョブコントローラは起動中の子孫プロセスを強制終了し，ファイルの削除などの必要な後処理を実行してから終了します。

(O)

バッチジョブが強制終了されていることを確認します。

KNAX6594-E

TerminateProcess などのプロセス終了の操作によって，ジョブコントローラが強制終了しました。

TerminateProcess などのプロセス終了の操作によって，ジョブコントローラが強制終了しました。

(S)

ジョブコントローラは起動中の子孫プロセスを強制終了し，ファイルの削除などの必要な後処理を実行してから終了します。

(O)

バッチジョブが強制終了されていることを確認します。

KNAX6595-E

エラーが発生しました。reason=**原因**,details=**保守情報**

次のどちらかの要因で，**原因**に示すエラーが発生しました。

- スクリプト拡張コマンドの指定位置が不正です。
- 内部エラーが発生しました。

(S)

処理を終了します。

(O)

スクリプト拡張コマンドの指定位置を確認し，ジョブ定義スクリプトを修正します。スクリプト拡張コマンドの指定位置に問題がない場合，システム管理者に連絡してください。

KNAX6596-E

ジョブ名.ジョブステップ名 ジョブステップがエラー終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

ジョブステップがエラーで終了しました。

ジョブ名

ジョブ名

ジョブステップ名

ジョブステップ名

終了コード

ジョブステップの終了コード

実行時間

ジョブステップの実行時間。OS の API を使って取得した参考値です。

CPU時間

ジョブステップの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行し，後続ジョブステップの中で run 属性が abnormal または always のジョブステップだけを実行します。ただし，次の場合はシェルを終了します。

- ジョブ定義スクリプトの実行を停止するコマンドが実行されたため，KNAX6584-I を出力した場合
- 上記以外でジョブ定義スクリプトの実行を停止するエラーが発生した場合

(O)

ジョブステップ内の各コマンドの実行結果を確認し，エラーとなった原因を取り除きます。

KNAX6597-I

ジョブ名.ジョブステップ名 ジョブステップが正常終了しました。rc=**終了コード** E-Time=**実行時間** s C-Time=**CPU時間** s

ジョブステップが正常に終了しました。

ジョブ名

ジョブ名

ジョブステップ名

ジョブステップ名

終了コード

ジョブステップの終了コード

実行時間

ジョブステップの実行時間。OS の API を使って取得した参考値です。

CPU時間

ジョブステップの CPU 時間。OS の API を使って取得した参考値です。

(S)

処理を続行します。

KNAX6598-E

API エラーが発生しました。name=**API名称**, reason=**原因**, details=**保守情報**

API名称で示す API 呼び出しで，**原因**に示すエラーが発生しました。

(S)

処理を終了します。

(O)

原因および一緒に出力されるほかのメッセージを参照してエラーの原因を取り除きます。問題が解決しない場合，システム管理者に連絡してください。

KNAX6599-E

内部エラーが発生しました。reason=**原因**, details=**保守情報**

原因に示す内部エラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX6600-E

エラーが発生しました。**関数名**([**引数**,…]) rc=**リターンコード**, error number=**エラー番号**, internal info=(**保守情報**)

関数名で示す関数でエラーが発生しました。

関数名

エラーが発生した関数（プラットフォームが提供している関数）

引数

関数の引数。出力されない場合もあります。

リターンコード

関数の戻り値

エラー番号

エラー内容を示すグローバル変数 errno の値。10 進数です。

(S)

処理を続行するかどうかは状況に依存します。このメッセージに続いて出力されるメッセージを参照してください。

(O)

メッセージ内容を基に原因を調査し，解決してください。

KNAX6601-E

処理中に内部矛盾を検出しました。details=**保守情報**

処理中に内部矛盾を検出しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX6602-E

標準出力への出力時に入出力エラーが発生しました。

標準出力 stdout への出力時に入出力エラーが発生しました。

(S)

処理を終了します。

(O)

このメッセージの前後に出力されたメッセージの内容を基に，標準出力の入出力エラーの原因を調査し，問題を解決してください。

KNAX6603-E

現在の日付を取得できませんでした。

現在の日付を取得できませんでした。

(S)

処理を終了します。

(O)

このメッセージの前後に出力されたメッセージの内容を基に，エラーの原因を調査し，問題を解決してください。

KNAX6604-E 【Windows 限定】

エラーが発生しました。**関数名**(**引数**) rc=**リターンコード**, last error code=**システムエラーコード**, internal info=(**保守情報**)

関数名で示す関数でエラーが発生しました。

関数名

エラーが発生した関数名（プラットフォームが提供している関数）

引数

関数の引数で障害調査用の情報

必ず出力されるとは限りません。また，すべての引数が出力されるとは限りません。

リターンコード

関数の戻り値

システムエラーコード

エラー内容を示す LastError()の戻り値

(S)

処理はこのメッセージの前後に出力されたメッセージに依存します。

(O)

このメッセージの前後に出力されたメッセージの内容を基に，エラーの原因を調査し，問題を解決してください。

KNAX6605-E

日時を内部表現に変換するときにエラーが発生しました。time=**日時**

日時を内部表現に変換するときにエラーが発生しました。

日時

内部表現に変換できなかった日時

(S)

処理を終了します。

(O)

コマンドが解釈した結果の日時が，次に示す範囲に収まる日時を指定します。

この日時の範囲は UTC で示した日時です。

1970 年 1 月 1 日 00:00:00～2038 年 1 月 19 日 03:14:07

ただし，プラットフォームによっては，この範囲であってもエラーになることがあります。

KNAX6610-E

無効なオプション名（"**オプション名**"）が指定されています。

不明なオプションが指定されています。

オプション名

不明なオプション名

(S)

処理を終了します。

(O)

正しいオプションを指定してください。

KNAX6611-E

オプション（"**オプション名**"）の値が指定されていません。

オプション名で示すオプションの値が指定されていません。

(S)

処理を終了します。

(O)

出力されたオプションに対して，値を指定してください。

KNAX6612-E

オプションの値（"**オプション値**"）が正しくありません。

オプションの値が正しくありません。

オプション値

指定されたオプションの値

(S)

処理を終了します。

(O)

オプションに対して正しい値を指定してください。

KNAX6613-I

ジョブの実行開始日時の下限: YYYY-MM-DD hh:mm:ss+hhmm
　または
ジョブの実行開始日時の下限: 指定されていません。

「YYYY-MM-DD hh:mm:ss+hhmm」に，コマンドが解釈した「ジョブの実行開始日時の下限」を示します。
「+hhmm」はローカルタイム - UTC の時差の時分による表現です。

「指定されていません。」は，-s オプションが指定されていないことを示します。

(S)

処理を続行します。

KNAX6614-I

ジョブの実行開始日時の上限: YYYY-MM-DD hh:mm:ss+hhmm
　または
ジョブの実行開始日時の上限: 指定されていません。

「YYYY-MM-DD hh:mm:ss+hhmm」に，コマンドが解釈した「ジョブの実行開始日時の上限」を示します。
「+hhmm」はローカルタイム - UTC の時差の時分による表現です。

「指定されていません。」は，-e オプションが指定されていないことを示します。

(S)

処理を続行します。

KNAX6615-E

ジョブの実行開始日時の下限と上限が逆転しています。

ジョブの実行開始日時の下限と上限が逆転しています。

(S)

処理を終了します。

(O)

ジョブの実行開始日時の上限（adshevtout コマンドの-e オプションで指定）が，ジョブの実行開始日時の下限（adshevtout コマンドの-s オプションで指定）よりあとになるよう指定してください。

KNAX6616-E

コマンドに余分な引数があります。

コマンドに余分な引数があります

(S)

処理を終了します。

(O)

コマンドの引数を正しく指定してください。

KNAX6632-E

スプールディレクトリが見つかりません。directory name="**スプールディレクトリ名**"

スプールディレクトリが見つかりません。

スプールディレクトリ名

参照しようとしたディレクトリ

(S)

処理を終了します。

(O)

環境ファイルで指定したスプールディレクトリが正しいか確認してください。

KNAX6633-E

スプールディレクトリを参照できません。directory name="**スプールディレクトリ名**"

スプールディレクトリを参照できません。

スプールディレクトリ名

参照しようとしたディレクトリ

(S)

処理を終了します。

(O)

コマンドを実行するユーザーにスプールディレクトリを参照する権限があるか，確認してください。

KNAX6634-E

初期化処理でエラーが発生しました。

adshevtout コマンドの初期化処理でエラーが発生しました。

(S)

処理を終了します。

(O)

このメッセージより先に出力されたメッセージを参照して問題を解決してください。

KNAX6635-E

スプールを参照するための排他制御でエラーが発生しました。directory name="**スプールディレクトリ名**"

スプールを参照するための排他制御でエラーが発生しました。

スプールディレクトリ名

参照しようとしたディレクトリ

(S)

処理を終了します。

(O)

次の内容を確認してください。

- 環境ファイルで指定したスプールディレクトリが正しいか。
- スプールディレクトリを参照または更新できるアクセス権限があるか。
- ロックファイルを作成，参照，または更新できるか。

KNAX6636-E

ほかのコマンドがスプールにアクセスしているため，スプールを参照できませんでした。directory name="**スプールディレクトリ名**"

ほかのコマンドがスプールにアクセスしているため，スプールを参照できませんでした。

スプールディレクトリ名

参照しようとしたディレクトリ

(S)

処理を終了します。

(O)

adshhk コマンドを実行していない状態で，コマンドを再実行してください。

KNAX6640-I

イベントファイル（"**イベントファイルのパス名**"）への参照権限がないため，イベントファイルの参照をスキップしました。

コマンドを実行したユーザーにはイベントファイルへの参照権限がないため，イベントファイルの参照をスキップしました。

イベントファイルのパス名

スキップしたイベントファイルのパス名

(S)

処理を続行します。

KNAX6644-E

イベントファイル（"**イベントファイルのパス名**"）の読み込み時に入出力エラーが発生しました。

イベントファイルの読み込み時に入出力エラーが発生しました。

イベントファイルのパス名

入出力エラーが発生したイベントファイルのパス名

(S)

入出力エラーが発生したイベントファイルの処理を中止し，別のイベントファイルの処理を続行します。

(O)

このメッセージの前後に出力されたメッセージの内容を基に，イベントファイルの入出力エラーの原因を調査し，問題を解決してください。

KNAX6645-W

イベントファイル（"**イベントファイルのパス名**"）は，このコマンドでサポートしていないバージョン（**バージョン番号**）のイベントファイルです。

イベントファイルは，このコマンドでサポートしていないバージョンのイベントファイルです。

イベントファイルのパス名

イベントファイルのパス名

バージョン番号

イベントファイルのバージョン番号

(S)

出力したイベントファイルを参照しないで，別のイベントファイルの処理を続行します。

(O)

イベントファイルを作成した JP1/Advanced Shell のバージョンに対応するコマンドを実行してください。

KNAX6646-E

イベントファイル（"**イベントファイルのパス名**"）が壊れています。

イベントファイルが壊れています。

イベントファイルのパス名

壊れていると判断したイベントファイルのパス名

(S)

壊れていると判断したイベントファイルの処理を中止し，別のイベントファイルの処理を続行します。

(O)

イベントファイルを不当に更新していないか確認してください。原因が確認できない場合は，システム管理者に連絡してください。

KNAX6701-W

ジョブオブジェクトを使用できません。

ジョブオブジェクトを使用できません。バッチジョブを強制終了したとき，子プロセスが作成したプロセスについては TerminateProcess 関数によるプロセス終了を行いません。

(S)

処理を継続します。

KNAX6710-I

組み込みコマンド "**コマンド名**"[に指定されているオプション "**オプション名**"] は現在のプラットフォームではサポートしていません。組み込みコマンドの終了コードは "**終了コード**" です。[filename="**ファイル名**" line=**行番号**]

現在のプラットフォームではサポートされていない組み込みコマンド（または組み込みコマンドのオプション）を指定して実行しようとしました。または，環境設定パラメーター TRAP_ACTION_SIGTERM に DISABLE を指定した環境で trap コマンドを実行しようとしました。

コマンド名

組み込みコマンド名

オプション名

組み込みコマンドに指定されたオプション名

終了コード

組み込みコマンドの終了コード

ファイル名

ジョブ定義スクリプトファイル名

行番号

組み込みコマンドを実行しようとした行の行番号

(S)

終了コードに示す値を組み込みコマンドの終了コードとして処理を継続します。

KNAX6711-E

組み込みコマンド "**コマンド名**"[に指定されているオプション "**オプション名**"] は現在のプラットフォームではサポートしていません。[filename="**ファイル名**" line=**行番号**]

現在のプラットフォームではサポートされていない組み込みコマンド（または組み込みコマンドのオプション）を指定して実行しようとしました。

コマンド名

組み込みコマンド名

オプション名

組み込みコマンドに指定されたオプション名

ファイル名

ジョブ定義スクリプトファイル名

行番号

組み込みコマンドを実行しようとした行の行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6712-E

現在のプラットフォームでは，名前がすべて大文字でない変数（"**シェル変数名**"）はエクスポートできません。[filename="**ファイル名**" line=**行番号**]

現在のプラットフォームでは，名前がすべて大文字でない変数はエクスポートできません。

シェル変数名

エクスポートしようとしたシェル変数の名前

ファイル名

ジョブ定義スクリプトファイル名

行番号

export コマンドを実行しようとした行の行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6713-E

必須のディレクトリ（"**ディレクトリ名**"）が存在しません。

現在のプラットフォームで，JP1/Advanced Shell に必須のディレクトリが存在しません。

ディレクトリ名

存在しないディレクトリのディレクトリ名

(S)

処理を終了します。

(O)

セットアップの手順を見直してください。

KNAX6714-E

現在のプラットフォームでは，バックグラウンド実行できません。[filename="**ファイル名**" line=**行番号**]

現在のプラットフォームでは，バックグラウンドでジョブ定義スクリプトを実行できません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

バックグラウンドで実行しようとした行の行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6715-E

現在のプラットフォームでは，サブシェルの実行はできません。[filename="**ファイル名**" line=**行番号**]

現在のプラットフォームでは，サブシェルでジョブ定義スクリプトを実行できません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

サブシェルで実行しようとした行の行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを修正します。

KNAX6718-I

The trap action was not set because an unsupported termination method was specified in the trap command.[filename="**ファイル名**" line=**行番号**]

trap コマンドに未サポートの強制終了方法を指定したため，アクションが設定されません。

ファイル名

ジョブ定義スクリプトファイル名

行番号

組み込みコマンドを実行しようとした行の行番号

(S)

処理を継続します。

KNAX6750-E

ファイルまたはディレクトリ（"**パス名**"）が存在しません。

パス名で示すファイルまたはディレクトリがありません。

(S)

処理を継続します。

(O)

パス名に指定したファイルまたはディレクトリが正しいか見直してください。

KNAX6751-E

ACL 情報の取得に失敗しました。path="**パス名**", reason=**原因**, details=**保守情報**

パス名で示すファイルまたはディレクトリの ACL の情報の取得に失敗しました。

(S)

処理を継続します。

(O)

adshscripttool コマンドの実行ユーザーが，パス名に指定したファイルまたはディレクトリの所有者であるか確認してください。所有者でない場合，所有者で実行してください。

KNAX6752-E

モードの書式に誤りがあります。

adshscripttool コマンドの-fmode オプションに指定したモードの書式に誤りがあります。

(S)

処理を継続します。

(O)

モードの指定を見直してください。

KNAX6753-E

adshscripttool: コマンドラインの指定に誤りがあります。details="詳細"

コマンドラインの指定に誤りがあります。メッセージに表示される詳細の表示内容と，その意味を次に示します。

詳細の表示内容	意味
必要なオプションが指定されていません。	必要なオプションが指定されていません。
パス名が指定されていません。	パス名が指定されていません。
モードが指定されていません。	モードが指定されていません。
-s オプションの値が指定されていません。	-s オプションの値が指定されていません。
-s オプションを先頭に指定することはできません。	-s オプションを先頭に指定することはできません。
-s オプションの値が間違っています。	-s オプションの値が間違っています。
パス名を複数指定することはできません。	パス名を複数指定することはできません。
モードを複数指定することはできません。	モードを複数指定することはできません。
パス名に空文字列が指定されました。	パス名に空文字列が指定されました。
モードに空文字列が指定されました。	モードに空文字列が指定されました。
-s オプションの値に空文字列が指定されました。	-s オプションの値に空文字列が指定されました。

(S)

処理を継続します。

(O)

コマンドを正しく指定して再実行してください。

KNAX6800-I

パスの変換規則に合致しました。[filename="**ファイル名**" line=**行番号**] rule_str="**パス名1**":"**パス名2**"

パス名1 から**パス名2** に置換する変換規則に合致しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

パスの変換規則に合致した行の行番号

(S)

処理を継続します。

KNAX6801-I

パスを扱うシェル変数による変換規則に合致しました。[filename="**ファイル名**" line=**行番号**] rule_var="**シェル変数名**"

パスを扱うシェル変数による変換規則に合致しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

パスの変換規則に合致した行の行番号

シェル変数名

パス名を扱うシェル変数の名前

(S)

処理を継続します。

KNAX6803-I

PATH_CONV_ACCESS パラメーターの変換規則に合致しました。[filename="**ファイル名**" line=**行番号**] rule_str="**パス名1**":"**パス名2**"

パス名1 から**パス名2** に置換する変換規則に合致しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

PATH_CONV_ACCESS パラメーターの変換規則に合致した行の行番号

(S)

処理を継続します。

KNAX6804-I

COMMAND_CONV_ARG パラメーターの変換規則に合致しました。[filename="**ファイル名**" line=**行番号**] rule_str="**引数1**":"**引数2**"

引数1 から**引数2** に置換する変換規則に合致しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

COMMAND_CONV_ARG パラメーターの変換規則に合致した行の行番号

(S)

処理を継続します。

KNAX6805-I

機能名の中で PATH_CONV_ACCESS パラメーターの変換規則に合致しました。rule_str="**パス名1**":"**パス名2**"

機能名で示す機能を実行している時に，**パス名1** から**パス名2** に置換する変換規則に合致しました。

機能名

変換規則に合致した機能名。次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

(S)

処理を継続します。

KNAX6806-I

機能名の中で COMMAND_CONV_ARG パラメーターの変換規則に合致しました。rule_str="**引数1**":"**引数2**"

機能名で示す機能を実行している時に，**引数1** から**引数2** に置換する変換規則に合致しました。

機能名

変換規則に合致した機能名。次のどちらかが出力されます。

- コマンド置換機能の場合：コマンド置換
- trap コマンドのアクションの場合：trap コマンドのアクション

(S)

処理を継続します。

KNAX6810-E

PATH_CONV_ENABLE パラメーターで指定したディレクトリ区切り文字が不正です。

PATH_CONV_ENABLE パラメーターで指定したディレクトリ区切り文字が不正です。

(S)

処理を終了します。

(O)

環境ファイルを修正します。

KNAX6811-E

PATH_CONV_ENABLE パラメーターで指定したパス区切り文字が不正です。

PATH_CONV_ENABLE パラメーターで指定したパス区切り文字が不正です。

(S)

処理を終了します。

(O)

環境ファイルを修正します。

KNAX6812-E

PATH_CONV パラメーターで指定したパス名（"**パス名**"）が不正です。

PATH_CONV パラメーターで指定したパスが不正です。

パス名

PATH_CONV パラメーターで指定したパス名

(S)

処理を終了します。

(O)

環境ファイルを修正します。

KNAX6813-E

ジョブ定義スクリプトの 1 行の上限を超えました。[filename="**ファイル名**" line=**行番号**]

ジョブ定義スクリプトの 1 行の上限を超えました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

上限を超えた行の行番号

(S)

処理を終了します。

(O)

ジョブ定義スクリプトを見直します。

KNAX6814-E

"**引数**" は不正な引数です。

COMMAND_CONV_ARG パラメーター，または PATH_CONV_ACCESS パラメーターで指定した引数が不正です。

引数

COMMAND_CONV_ARG パラメーター，または PATH_CONV_ACCESS パラメーターで指定した引数

(S)

処理を終了します。

(O)

環境ファイルを修正します。

KNAX6815-E

引数が指定されていません。

COMMAND_CONV_ARG パラメーター，または PATH_CONV_ACCESS パラメーターの引数が指定されていません。

(S)

処理を終了します。

(O)

環境ファイルを修正します。

KNAX6830-I

CHILDJOB_PGM パラメーターによる子孫ジョブ実行の読み替え規則に合致しました。cmdline="**コマンドライン**"

CHILDJOB_PGM パラメーターによる子孫ジョブ実行の読み替え規則に合致しました。または，adshscripttool -exec コマンドを実行して子孫ジョブを実行しました。

コマンドライン

読み替えるコマンドのコマンドライン

(S)

処理を継続します。

KNAX6831-I

CHILDJOB_SHEBANG パラメーターによる子孫ジョブ実行の読み替え規則に合致しました。script="**スクリプト名**" shebang="**シェバン**"

CHILDJOB_SHEBANG パラメーターによる子孫ジョブ実行の読み替え規則に合致しました。

スクリプト名

子孫ジョブとして実行するスクリプトファイルのファイル名

シェバン

子孫ジョブとして実行するスクリプトファイルの 1 行目に記述した，#!で始まる行

(S)

処理を継続します。

KNAX6832-I

CHILDJOB_EXT パラメーターによる子孫ジョブ実行の読み替え規則に合致しました。script="**スクリプト名**"

CHILDJOB_EXT パラメーターによる子孫ジョブ実行の読み替え規則に合致しました。

スクリプト名

子孫ジョブとして実行するスクリプトファイルのファイル名

(S)

処理を継続します。

KNAX6996-I

API エラーが発生しました。name=**API名称**, reason=**原因**

API名称で示す API 呼び出しで，**原因**で示すエラーが発生しました。

(S)

処理を終了します。

(O)

原因を参照してエラーの原因を取り除きます。問題が解決しない場合，システム管理者に連絡してください。

KNAX6997-E

API エラーが発生しました。name=**API名称** reason=**原因**

API名称で示す API 呼び出しで，**原因**で示すエラーが発生しました。

API名称が「AssignProcessToJobObject」であり，**原因**が「Access is denied.」の場合は，ジョブ定義スクリプト中から子孫ジョブではなくルートジョブとしてジョブコントローラを呼び出している，またはジョブオブジェクトを使用するJP1/Advanced Shell以外のプログラムからジョブコントローラを呼び出している可能性があります。

(S)

処理を終了します。ただし，API名称が「AssignProcessToJobObject」の場合は，処理を継続します。

(O)

- ジョブ定義スクリプト中から子孫ジョブではなくルートジョブとしてジョブコントローラを呼び出している場合

  子孫ジョブとしてジョブコントローラを呼び出してください。

- ジョブオブジェクトを使用するJP1/Advanced Shell以外のプログラムからジョブコントローラを呼び出している場合

  ジョブコントローラはジョブの強制終了時に，孫以下のプロセスを強制終了しないで終了します。ジョブコントローラを呼び出しているプログラムの仕様も確認し，強制終了時の動作に問題がなければエラーに対する対処は不要です。問題があるときは，システム管理者に連絡してください。

- 上記以外の場合

  原因を参照してエラーの原因を取り除きます。問題が解決しないときは，システム管理者に連絡してください。

KNAX6998-E

メモリ不足が発生しました。[filename="**ファイル名**" line=**行番号**]

メモリ不足が発生しました。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生した行の行番号

(S)

処理を終了します。

(O)

システム管理者に連絡します。システム管理者は，メモリ見積もりを見直してください。

KNAX6999-E

内部エラーが発生しました。reason="**原因**" details="**保守情報**"

内部エラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX7000-E

入力文字列が長すぎます。

入力文字列が長過ぎます。

(S)

処理を続行します。

(O)

入力文字列長を見直して再度入力します。

KNAX7001-E

指定したコマンド（"**コマンド名**"）は存在しません。

コマンド名で示すデバッガのコマンドがありません。

(S)

処理を続行します。

(O)

正しいコマンド名を再度入力します。

KNAX7002-E

引数の指定が多すぎます。

引数の指定が多過ぎます。

(S)

処理を続行します。

(O)

正しい引数を指定してコマンドを再実行します。

KNAX7003-E

デバッガコマンド（"**コマンド名**"）は引数を取りません。

コマンド名で示すデバッガコマンドは引数を取りません。

(S)

処理を続行します。

(O)

正しい引数を指定してコマンドを再実行します。

KNAX7004-E

デバッガコマンド（"**コマンド名**"）に必要な引数（**引数詳細**）が指定されていません。

コマンド名で示すデバッガのコマンドは**引数詳細**で示す引数を取ります。

(S)

処理を続行します。

(O)

正しい引数を指定してコマンドを再実行します。

KNAX7006-W

ジョブ定義スクリプトを最初から再実行しますか？(y または n)

ジョブ定義スクリプトを最初から再実行するかどうかを確認します。

(S)

処理を続行します。

(O)

再実行する場合，y を入力します。再実行しない場合，n を入力します。

KNAX7007-I

ジョブ定義スクリプトの実行を開始します。：**ジョブ定義スクリプトのパス名 引数**

ジョブ定義スクリプトの実行を開始します。

(S)

処理を続行します。

KNAX7008-I

ジョブ定義スクリプトを再実行しませんでした。

ジョブ定義スクリプトを再実行しませんでした。

(S)

処理を続行します。

KNAX7009-I

デバッグ中のジョブ定義スクリプトを終了しますか？(y または n)

デバッグ中のジョブ定義スクリプトを終了するかどうかを確認します。

(S)

処理を続行します。

(O)

終了する場合，y を入力します。終了しない場合，n を入力します。

KNAX7010-E

ジョブ定義スクリプトが実行されていません。

ジョブ定義スクリプトが実行していません。

(S)

処理を続行します。

(O)

ジョブ定義スクリプトを実行中にコマンドを再実行します。

KNAX7011-I

デバッガコマンド（"**コマンド名**"）は実行されませんでした。

コマンド名で示すデバッガのコマンドは実行されませんでした。

(S)

処理を続行します。

KNAX7012-W

ジョブ定義スクリプト実行中にデバッガを終了しますか？(y または n)

ジョブ定義スクリプト実行中にデバッガを終了するかどうかを確認します。

(S)

処理を続行します。

(O)

終了する場合，y を入力します。終了しない場合，n を入力します。

KNAX7013-E

ジョブ定義スクリプトファイル（"**ファイル名**"）は解析されていません。

ファイル名で示すジョブ定義スクリプトファイルは解析されていません。

(S)

処理を続行します。

(O)

正しいファイル名を指定してコマンドを再実行します。または，ジョブ定義スクリプトファイル内の外部スクリプト呼び出しに#-adsh_script コマンドを使用します。

KNAX7014-E

ブレークポイントとウォッチポイントの番号が上限を超えました。

ブレークポイントとウォッチポイントの番号が上限を超えました。

(S)

処理を続行します。

(O)

デバッガを再起動してからコマンドを再実行します。

KNAX7015-W

指定した行番号（"**行番号**"）にブレークポイントを設定できません。次に設定できる行にブレークポイントを設定します。

行番号に示す行にはブレークポイントを設定できません。次に設定できる行にブレークポイントを設定します。

(S)

処理を続行します。

KNAX7016-E

指定した行番号（"**行番号**"）にブレークポイントを設定できません。次に設定できる行もありません。

行番号に示す行にはブレークポイントを設定できません。次に設定できる行がありません。

(S)

処理を続行します。

(O)

正しい行番号を指定してコマンドを再実行します。

KNAX7017-E

ファイル（"**ファイル名**"）が存在しません。

ファイル名で示すファイルはありません。

(S)

処理を続行します。

(O)

正しいファイル名を指定してコマンドを再実行します。

KNAX7018-I

ブレークポイント "**ブレークポイント番号**": ファイル名="**ファイル名**" 行番号=**行番号**

ブレークポイントの情報を表示します。**ファイル名**はベース名で表示します。

(S)

処理を続行します。

KNAX7019-E

行番号（"**行番号**"）は存在しません。

行番号で示す行はありません。

(S)

処理を続行します。

(O)

正しい行番号を指定してコマンドを再実行します。または，行を移動してからコマンドを再実行します。

KNAX7020-E

関数名（"**関数名**"）が定義されていません。

関数名で示す関数は定義されていません。

(S)

処理を続行します。

(O)

正しい関数名を指定してコマンドを再実行します。

KNAX7021-E

ジョブステップ名（"**ジョブステップ名**"）が定義されていません。

ジョブステップ名で示すジョブステップは定義されていません。

(S)

処理を続行します。

(O)

正しいジョブステップ名を指定してコマンドを再実行します。

KNAX7022-E

変数名（"**変数名**"）が不正です。

変数名で示す変数名が不正です。

(S)

処理を続行します。

(O)

正しい変数名を指定してコマンドを再実行します。

KNAX7023-I

ウォッチポイント "**ウォッチポイント番号**": 変数名 "**変数名**"

ウォッチポイントの情報を表示します。

(S)

処理を続行します。

KNAX7024-E

ブレークポイント・ウォッチポイント番号の範囲（"**番号範囲**"）が不正です。

番号範囲で示すブレークポイント・ウォッチポイント番号の範囲が不正です。

(S)

処理を続行します。

(O)

正しい番号範囲を指定してコマンドを再実行します。

KNAX7025-I

すべてのブレークポイントとウォッチポイントを削除しますか？(y または n)

すべてのブレークポイントとウォッチポイントを削除するかどうかを確認します。

(S)

処理を続行します。

(O)

削除する場合，y を入力します。削除しない場合，n を入力します。

KNAX7026-E

ブレークポイントとウォッチポイントがどちらも設定されていないため，削除できません。

ブレークポイントとウォッチポイントがどちらも設定されていないため，ブレークポイントとウォッチポイントを削除できません。

(S)

処理を続行します。

(O)

必要な場合，ブレークポイントまたはウォッチポイントを設定した状態でコマンドを再実行します。

KNAX7027-E

"**番号**" 番のブレークポイントまたはウォッチポイントがありません。

番号で示すブレークポイントまたはウォッチポイントが設定されていません。

(S)

処理を続行します。

(O)

正しい番号を指定してコマンドを再実行します。

KNAX7028-E

"**番号範囲**" 番にブレークポイントとウォッチポイントがありません。

番号範囲で示す範囲にブレークポイントとウォッチポイントがありません。

(S)

処理を続行します。

(O)

正しい番号範囲を指定してコマンドを再実行します。

KNAX7029-E

コマンドの引数（"**引数**"）が不正です。

引数で示すコマンドの引数が不正です。

(S)

処理を続行します。

(O)

引数の指定方法を見直してコマンドを再実行します。

KNAX7032-I

ジョブ定義スクリプトファイル（"**ジョブ定義スクリプト名**"）の中で実行を停止しました。

ジョブ定義スクリプト名に示すジョブ定義スクリプトの中で実行を停止しました。

(S)

処理を続行します。

KNAX7033-I

外部スクリプトファイル（"**ジョブ定義スクリプト名**"）の中で実行を停止しました。

ジョブ定義スクリプト名に示す外部スクリプトの中で実行を停止しました。

(S)

処理を続行します。

KNAX7034-I

ジョブ定義スクリプトの実行を継続します。

ジョブ定義スクリプトを継続して実行します。

(S)

処理を続行します。

KNAX7035-E

デバッガコマンド（"**コマンド名**"）は最も外側では実行できません。

コマンド名で示すデバッガコマンドは最も外側では実行できません。

(S)

処理を続行します。

(O)

関数内で停止している状態でコマンドを再実行します。

KNAX7036-I

現在の関数の終わりまで実行します。

現在の関数の終わりまで実行します。

(S)

処理を続行します。

KNAX7037-I

現在の関数を終了しますか？(y または n)

現在の関数を終了するかどうかを確認します。

(S)

処理を続行します。

(O)

終了する場合，y を入力します。終了しない場合，n を入力します。

KNAX7038-I

シグナル（"**シグナル名**"）をジョブ定義スクリプトに送信します。

シグナル名で示すシグナルをジョブ定義スクリプトに送信します。

(S)

処理を続行します。

KNAX7039-E

シグナル番号（"**シグナル番号**"）はありません。

シグナル番号で示すシグナルはありません。

(S)

処理を続行します。

(O)

正しいシグナル番号を指定してコマンドを再実行します。

KNAX7040-E

シグナル名（"**シグナル名**"）はありません。

シグナル名で示すシグナルはありません。

(S)

処理を続行します。

(O)

正しいシグナル名を指定してコマンドを再実行します。

KNAX7043-I

シグナル（"**シグナル名**"）（Stop=Yes）を受信したため，ジョブ定義スクリプトが停止しました。

受信時にジョブ定義スクリプトを停止する**シグナル名**で示すシグナルを受信したため，ジョブ定義スクリプトが停止しました。

(S)

処理を続行します。

KNAX7044-E

デバッガコマンド（"**コマンド名**"）はサブコマンドを必要とします。

コマンド名で示すデバッガのコマンドはサブコマンドを必要とします。

(S)

処理を続行します。

(O)

サブコマンドを指定してコマンドを再実行します。

KNAX7045-E

デバッガコマンド（"**コマンド名**"）のサブコマンド名（"**サブコマンド名**"）が不正です。

コマンド名で示すデバッガのコマンドの**サブコマンド名**が不正です。

(S)

処理を続行します。

(O)

正しいサブコマンド名を指定してコマンドを再実行します。

KNAX7046-E

変数（"**変数名**"）が定義されていません。

変数名で示す変数が定義されていません。

(S)

処理を続行します。

(O)

正しい変数名を指定してコマンドを再実行します。

KNAX7047-I

ブレークポイントとウォッチポイントがどちらも設定されていません。

ブレークポイントとウォッチポイントがどちらも設定されていません。

(S)

処理を続行します。

(O)

必要な場合，ブレークポイントまたはウォッチポイントを設定した状態でコマンドを再実行します。

KNAX7048-I

ディレクトリパス（**ディレクトリパス名**）に作業ディレクトリを移動しました。

ディレクトリパス名で示すディレクトリパスに作業ディレクトリを移動しました。

(S)

処理を続行します。

KNAX7049-E

作業ディレクトリを移動できません。(reason=**エラー詳細**)

エラー詳細で示す原因によって，作業ディレクトリを移動できません。

(S)

処理を続行します。

(O)

エラー詳細の原因を取り除いてコマンドを再実行します。

KNAX7050-E

デバッガコマンド（"**コマンド名**"）の引数に "&" を指定できません。

コマンド名で示すデバッガのコマンドの引数に&を指定できません。

(S)

処理を続行します。

(O)

バックグラウンド実行以外の目的で&を使用する場合，¥&と指定してください。

KNAX7052-E

変数の型と代入する値の型とが異なっています。

変数の型と代入する値の型とが異なっています。

(S)

処理を続行します。

(O)

変数の型と代入する値の型を見直してコマンドを再実行します。

KNAX7053-I

使用方法: **コマンド名 引数**

コマンド名で示すデバッガコマンドの使用方法を表示します。

(S)

処理を続行します。

KNAX7054-E

変数（"**変数名**"）は読み込み専用です。

変数名で示す変数は読み込み専用です。

(S)

処理を続行します。

(O)

変数の属性を見直してからコマンドを再実行します。

KNAX7055-E

"**行番号**" 行にはブレークポイントがすでに設定されています。

行番号に示す行にはブレークポイントがすでに設定されています。

(S)

処理を続行します。

(O)

必要があれば，ブレークポイントを削除したあと，コマンドを再実行します。または，異なる行番号を指定してコマンドを再実行します。

KNAX7056-I

変数（"**変数名**"）の値が **数値** に変更されました。

変数名に示す変数の値が**数値**に変更されました。

数値

set コマンドの代入式の右辺に記述した数値，または右辺に記述した変数の値（数値）を示します。

(S)

処理を続行します。

KNAX7057-I

変数（"**変数名**"）の値が "**文字列**" に変更されました。

変数名に示す変数の値が**文字列**に変更されました。

文字列

set コマンドの代入式の右辺に記述した文字列，または右辺に記述した変数の値（文字列）を示します。

(S)

処理を続行します。

KNAX7058-I

Debugger received signal "**シグナル名**".

デバッガが**シグナル名**に示すシグナルを受信しました。

(S)

処理を続行します。

KNAX7062-E

変数（"**変数名**"）が値を持っていません。

変数名に示す変数が値を持っていません。

(S)

処理を続行します。

(O)

値を持つ変数名を指定してコマンドを再実行します。

KNAX7063-I

ジョブ定義スクリプト実行中にコマンド（"**コマンド名**"）の要求を受け付けました。

ジョブ定義スクリプト実行中に**コマンド名**に示すコマンドの要求を受け付けました。

(S)

バッチジョブをキャンセルします。

(O)

必要があれば，再実行します。

KNAX7064-I

エディタからのキャンセル要求を受け付けました。

エディタからのキャンセル要求を受け付けました。

(S)

バッチジョブをキャンセルします。

(O)

必要があれば，再実行します。

KNAX7065-I

ジョブステップが定義されていません。

ジョブステップが定義されていません。

(S)

処理を続行します。

(O)

必要があれば，ジョブ定義スクリプトにジョブステップを定義したあと，コマンドを再実行します。

KNAX7066-I

関数が定義されていません。

関数が定義されていません。

(S)

処理を続行します。

(O)

必要があれば，ジョブ定義スクリプトに関数を定義したあと，コマンドを再実行します。または，関数の定義が有効になってからコマンドを再実行します。

KNAX7067-I

変数が定義されていません。

変数が定義されていません。

(S)

処理を続行します。

(O)

必要があれば，ジョブ定義スクリプトに変数を定義したあと，コマンドを再実行します。または，変数の定義が有効になってからコマンドを再実行します。

KNAX7068-I

関数の終わりまでコマンドをスキップします。

関数の終わりまでコマンドをスキップします。

(S)

処理を続行します。

KNAX7070-E

変数（"**変数名**"）にはウォッチポイントがすでに設定されているため，新たなウォッチポイントは設定されません。

変数名に示す変数にはウォッチポイントがすでに設定されているため，新たなウォッチポイントは設定されません。

(S)

処理を続行します。

(O)

必要があれば，ウォッチポイントを削除したあと，コマンドを再実行します。または，異なる変数名を指定してコマンドを再実行します。

KNAX7071-E

デバッガが**エラー要因**で実行を停止しているため，ブレークポイントを設定できません。

エラー要因に示す要因でデバッガがジョブ定義スクリプトの実行を停止しているため，ブレークポイントを設定できません。

エラー要因

次のどちらかが出力されます。

- trap コマンドのアクション：trap のアクションを実行中に停止している
- EOF：EOF で停止している

(S)

処理を続行します。

(O)

エラー要因を解決した上でコマンドを再実行します。または，適切な引数を指定してコマンドを再実行します。

KNAX7072-E

デバッガが**エラー要因**で実行を停止しているため，ソースファイル行の内容を表示できません。

エラー要因に示す要因でデバッガがジョブ定義スクリプトの実行を停止しているため，ソースファイル行の内容を表示できません。

エラー要因

次のどちらかが出力されます。

- trap コマンドのアクション：trap のアクションを実行中に停止している
- EOF：EOF で停止している

(S)

処理を続行します。

(O)

エラー要因を解決した上でコマンドを再実行します。または，適切な引数を指定してコマンドを再実行します。

KNAX7073-I

エディタからトラップアクションの実行要求を受け付けました。parameter value="**パラメーター値**"

エディタからトラップアクションの実行要求を受け付けました。

パラメーター値

環境設定パラメーター TRAP_ACTION_SIGTERM に指定した値です。

環境設定パラメーター TRAP_ACTION_SIGTERM に DISABLE を指定している場合，または trap コマンドによるアクションの定義がない場合，このメッセージが出力されても trap コマンドのアクションは実行されません。

(S)

処理を続行します。

KNAX7090-W

表示行数が超えたため表示できないエラー情報があります。

上限を超えて表示できないエラー情報が 1 つ以上あります。

(S)

処理を続行します。

(O)

値を持つ変数名を指定してコマンドを再実行します。

KNAX7091-W

変数管理領域のサイズが上限を超えたため，表示できない変数情報があります。

上限を超えて表示できないエラー情報が 1 つ以上あります。

(S)

処理を続行します。

KNAX7099-E

デバッガがエラー終了しました。

デバッガがエラー終了しました。

(S)

処理を終了します。

(O)

このメッセージとともに出力されるほかのメッセージを参照して原因を取り除き，再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7101-E

デバッガの子プロセスの生成が失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因で，デバッガの子プロセスの生成が失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7102-I

デバッガが子プロセスを生成しました。(pid=**プロセスID**)

デバッガの子プロセスを生成しました。

(S)

処理を続行します。

KNAX7103-I

デバッガの子プロセスが終了しました。(pid=**プロセスID**)

デバッガの子プロセスが終了しました。

(S)

処理を続行します。

KNAX7104-E

デバッガでジョブ定義スクリプトを実行したときに，ログファイルのオープン処理に失敗しました。

デバッガでジョブ定義スクリプトを実行したときに，ログファイルのオープン処理に失敗しました。

(S)

処理を終了します。

(O)

エラーの原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7105-E

デバッガの作業ディレクトリへの移動に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因で，デバッガの作業ディレクトリへの移動に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行します。

KNAX7106-E

ジョブ定義スクリプトの作業ディレクトリへの移動に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因で，ジョブ定義スクリプトの作業ディレクトリへの移動に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行します。

KNAX7107-I

デバッガコマンドの入力中にシグナルを受信しました。

デバッガのコマンド入力中にシグナルを受信しました。

(S)

処理を終了するシグナルのときは終了し，処理を継続するシグナルのときは継続します。

(O)

必要があれば，再実行します。

KNAX7108-E

デバッガコマンドの入力に失敗しました。

デバッガのコマンドの入力に失敗しました。

(S)

処理を終了します。

(O)

一緒に出力されるほかのメッセージを参照してエラーの原因を取り除いて再実行します。

KNAX7109-I

デバッガコマンドで EOF が入力されました。

デバッガのコマンドで EOF が入力されました。

(S)

処理を終了します。

KNAX7110-E

DBG ファイル（"**ファイルパス**"）のオープン処理に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因で，ファイルパスに示すファイルのオープン処理に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行します。

KNAX7111-I

DBG ファイル（"**ファイルパス**"）の解析を開始します。

ファイルパスで示すファイルの解析を開始します。

(S)

処理を継続します。

KNAX7112-E

DBG ファイル（"**ファイルパス**"）のフォーマットが不正です。details=**保守情報**

ファイルパスで示すファイルのフォーマットが不正でした。**保守情報**は不正があった内部データの位置や不正の要因を示す情報です。

(S)

処理を終了します。

(O)

再実行します。必要があれば，システム管理者に連絡します。

KNAX7113-E

API（"**API名**"）でエラーが発生しました。(reason=**エラー詳細**)

エラー詳細に示す原因で，**API名**の処理に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7114-E

デバッガの子プロセスのウェイト処理に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因で，デバッガの子プロセスのウェイト処理に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7115-E

exec コマンドのプロセスの実行が失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因によって，exec コマンドプロセスの実行が失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7116-E

ディスクの空き容量が不足しています。

ディスクの空き容量が不足しています。

(S)

処理を終了します。

(O)

ディスクの空き容量を増やし，再実行します。

KNAX7117-I

DBG ファイル（"**ファイルパス**"）の解析が終了しました。

ファイルパスに示す DBG ファイルの解析が終了しました。

(S)

処理を続行します。

KNAX7118-E

コンソールの作成に失敗しました。

コンソールの作成に失敗しました。

(S)

処理を終了します。

(O)

再実行します。再実行しても発生する場合，システム管理者に連絡します。

KNAX7119-E

ファイル識別子の複製に失敗しました。

ファイルディスクリプタの複製に失敗しました。

(S)

処理を終了します。

(O)

再実行します。再実行しても発生する場合，システム管理者に連絡します。

KNAX7120-W

親プロセスは SIGCHLD を受信しましたが，子プロセスは終了していません。

親プロセスは SIGCHLD を受信しましたが，子プロセスの状態は変化しません。

(S)

処理を続行します。

KNAX7121-E

exec コマンドのプロセス生成に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因によって，exec コマンドのプロセス生成に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7122-E

exec コマンドのプロセスのウェイト処理に失敗しました。(reason=**エラー詳細**)

エラー詳細に示す原因によって，exec コマンドのプロセスのウェイト処理に失敗しました。

(S)

処理を終了します。

(O)

エラー詳細の原因を取り除いて再実行してください。問題が解決しない場合は，システム管理者に連絡してください。

KNAX7123-E

変数（"**変数名**"）の値の取得に失敗しました。details=**保守情報**

変数名に示す変数の値の取得に失敗しました。すべての変数を取得しようとした場合，**変数名**に<All variables>と示します。

システムのメモリが不足している場合，このメッセージが出力されることがあります。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX7124-E

変数（"**変数名**"）の値の設定に失敗しました。details=**保守情報**

変数名に示す変数の値の設定に失敗しました。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX7125-E

関数（"**関数名**"）の情報の取得に失敗しました。details=**保守情報**

関数名に示す関数の情報の取得に失敗しました。すべての関数を取得しようとした場合，関数名に<All functions>と表示します。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX7126-I

エラー注入モードを {"on" | "off"} に設定しました。

エラー注入モードを設定しました。

- on
  エラー注入モードを有効にしました。
- off
  エラー注入モードを無効にしました。

(S)

処理を終了します。

KNAX7127-E

エラー注入モードの再設定に失敗しました。

エラー注入モードを一度有効にしてジョブ定義スクリプトを再開しているため，エラー注入モードの再設定に失敗しました。

(S)

処理を続行します。

(O)

ジョブ定義スクリプトを最後まで実行するか，または再実行する必要があります。

KNAX7128-E

デバッガの DBG ファイルを削除できません。(filename=**ファイルパス**, reason=**エラー詳細**)

エラー詳細に示す原因によって，デバッガの DBG ファイルを削除できません。

ファイルパス

DBG ファイルのファイルパス

エラー詳細

エラー詳細

(S)

処理を続行します。

(O)

不要なファイルを rm コマンドなどで削除してください。

KNAX7129-E

エラー注入モードを有効にできません。

CMDRC_CMDGRP_CHECK パラメーターに FUNCTION を指定した場合，関数内で停止中にエラー注入モードを有効にできません。

(S)

処理を続行します。

(O)

必要な場合，関数外で停止してからエラー注入モードを有効にしてください。

KNAX7200-I

アプリケーション実行エージェントプログラムを起動します。(user name=**ユーザー名**, domain name=**ドメイン名**)

アプリケーション実行エージェントプログラムを起動します。

ユーザー名

adshappagent コマンドを起動したユーザー名

ドメイン名

adshappagent コマンドを起動したドメイン名

(S)

処理を続行します。

KNAX7201-I

アプリケーション実行エージェントプログラムが終了しました。(user name=**ユーザー名**, domain name=**ドメイン名**

アプリケーション実行エージェントプログラムが終了しました。

ユーザー名

adshappagent コマンドを起動したユーザー名

ドメイン名

adshappagent コマンドを起動したドメイン名

(S)

処理を続行します。

KNAX7203-W

レジストリ(“**レジストリ値**”)の指定値(“**指定値**”)が不正です。デフォルト値を仮定して処理を行います。

レジストリに指定した値が不正です。

レジストリ値

エラーの発生したレジストリ値

指定値

レジストリの指定値

(S)

デフォルト値を仮定して処理を続行します。

(O)

レジストリの指定値を見直します。

KNAX7204-E

コマンド名コマンドを実行中にエラーが発生しました。(API=**関数名**, error code=**エラーコード**)

コマンド名で示すコマンドを実行中にエラーが発生しました。

「グローバルオブジェクトの作成」権限が有効でない場合にこのメッセージが出力される場合があります。

コマンド名

エラーの発生したコマンド名

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

(S)

処理を中断します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。

「グローバルオブジェクトの作成」権限が有効でない場合は，「グローバルオブジェクトの作成」権限を有効にして再度実行してください。

セキュリティ上の問題などから「グローバルオブジェクトの作成」権限を有効にできない場合は，「グローバルオブジェクトの作成」権限を有効にできるユーザーでアプリケーション実行エージェント機能を使用してください。

エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7205-E

アプリケーション実行エージェントプログラムが既に起動されています。

アプリケーション実行エージェントプログラムが既に起動されています。

(S)

処理を中断します。

KNAX7210-E

実行アプリケーションの起動に失敗しました。viewname=**表示名**, error code=**エラーコード**)

実行アプリケーション実行要求処理実行に失敗しました。

表示名

実行が完了した実行アプリケーションの表示名

表示名の指定がなかった場合は，実行アプリケーション名を出力します。

エラー詳細

エラー詳細

(S)

処理を続行します。

(O)

エラーコードからエラー要因を取り除いて，コマンドを再実行してください。

存在しない実行アプリケーションを指定した場合，および実行アプリケーションを実行するのに必要な権限が不足している場合に発生する可能性があります。実行アプリケーションの格納フォルダなども含めて確認してください。

エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7211-I

アプリケーション実行エージェントプログラムのプロセスを開始します。

アプリケーション実行エージェントプログラムの処理を開始します。

(S)

処理を続行します。

KNAX7212-I

アプリケーション実行エージェントプログラムのプロセスを終了します。Detail=**保守情報**)

アプリケーション実行エージェントプログラムのプロセスを終了します。

保守情報

プロセスの識別情報

(S)

処理を続行します。

KNAX7213-I

実行アプリケーションの実行が完了しました。viewname=**表示名**, code=**終了コード**)

実行アプリケーションの実行が完了しました。

表示名

実行が完了した実行アプリケーションの表示名

表示名の指定がなかった場合は，実行アプリケーション名を出力します。

終了コード

実行が完了した実行アプリケーションの終了コード

(S)

処理を続行します。

KNAX7215-E

スタートアップ登録に失敗しました。(API=**関数名**, error code=**エラーコード**, user name=**ユーザー名**, domain name=**ドメイン名**)

スタートアップ登録に失敗しました。

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

ユーザー名

adshappagent コマンドを起動したユーザー名

ドメイン名

adshappagent コマンドを起動したドメイン名

(S)

処理を中断します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7216-E

スタートアップ解除に失敗しました。(API=**関数名**, error code=**エラーコード**, user name=**ユーザー名**, domain name=**ドメイン名**)

スタートアップ解除に失敗しました。

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

ユーザー名

adshappagent コマンドを起動したユーザー名

ドメイン名

adshappagent コマンドを起動したドメイン名

(S)

処理を中断します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7217-E

終了処理に失敗しました。(API=**関数名**, error code=**エラーコード**, user name=**ユーザー名**, domain name=**ドメイン名**)

アプリケーション実行エージェントプログラムの終了処理に失敗しました。

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

ユーザー名

adshappagent コマンドを起動したユーザー名

ドメイン名

adshappagent コマンドを起動したドメイン名

(S)

処理を続行します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7221-E

スタートアップ登録に失敗しました。

スタートアップ登録に失敗しました。

(S)

処理を中断します。

(O)

アプリケーション実行エージェント機能ログに出力されているメッセージよりエラー要因を取り除いて，再実行してください。

KNAX7222-E

スタートアップ解除に失敗しました。

スタートアップ解除に失敗しました。

(S)

処理を中断します。

(O)

アプリケーション実行エージェント機能ログに出力されているメッセージよりエラー要因を取り除いて，再実行してください。

KNAX7223-E

終了処理に失敗しました。

アプリケーション実行エージェントプログラムの終了処理に失敗しました。

(S)

処理を続行します。

(O)

アプリケーション実行エージェント機能ログに出力されているメッセージよりエラー要因を取り除いて，再実行してください。

KNAX7225-E

コマンド名コマンドを実行中にエラーが発生しました。

コマンド名で示すコマンドを実行中にエラーが発生しました。

(S)

処理を中断します。

(O)

アプリケーション実行エージェント機能ログに出力されているメッセージよりエラー要因を取り除いて，再実行してください。

KNAX7250-I

GUI アプリケーション実行コマンドを開始します。(user name=**ユーザー名**, domain name=**ドメイン名**)

GUI アプリケーション実行コマンドを開始します。

ユーザー名

adshappexec コマンドを起動したユーザー名

ドメイン名

adshappexec コマンドを起動したドメイン名

(S)

処理を続行します。

KNAX7251-I

GUI アプリケーション実行コマンドが終了しました。(user name=**ユーザー名**, domain name=**ドメイン名**)

GUI アプリケーション実行コマンドが終了しました。

ユーザー名

adshappexec コマンドを起動したユーザー名

ドメイン名

adshappexec コマンドを起動したドメイン名

(S)

処理を続行します。

KNAX7254-E

実行ユーザー名の取得に失敗しました。(API=reason=**関数名**, error code=**エラーコード**)

実行ユーザー名の取得に失敗しました。

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

(S)

処理を中断します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7255-I

アプリケーション実行エージェントプログラムが起動されていませんでした。再度アプリケーション実行エージェントプログラムの起動確認を行います。

アプリケーション実行エージェントプログラムが起動されていませんでした。再度アプリケーション実行エージェントプログラムの起動確認を行います。

(S)

処理を続行します。

KNAX7256-E

アプリケーション実行エージェントプログラムが起動されていませんでした。

アプリケーション実行エージェントプログラムが起動されていませんでした。

(S)

処理を中断します。

(O)

アプリケーション実行エージェントプログラムを起動した後に，コマンドを再実行してください。

KNAX7257-E

adshappexec コマンドをキャンセルしました。

ジョブコントローラが強制終了を受け付けたため，adshappexec コマンドをキャンセルしました。

(S)

処理を終了します。

KNAX7258-E

アプリケーション実行エージェントプログラムが終了しました。

アプリケーション実行エージェントプログラムが終了しました。

(S)

処理を中断します。

(O)

アプリケーション実行エージェントプログラムを起動した後に，コマンドを再実行してください。

KNAX7259-W

GUI アプリケーション実行コマンドは，実行アプリケーションの終了待ちを行いません。viewname=**表示名**

GUI アプリケーション実行コマンドは，実行アプリケーションのプロセス起動に関する仕様によって，実行アプリケーションの終了を待たないで終了します。

表示名

実行が完了した実行アプリケーションの表示名。表示名の指定がなかった場合は，実行アプリケーション名を出力します。

(S)

処理を中断します。

(O)

実行アプリケーションのプロセス起動に関する仕様を確認してください。

Excel ファイルを実行アプリケーションとして実行する前から，すでに Excel 自体が動作していたときに本メッセージが出力されることがあります。この場合，Excel が起動していない状態で運用するか，-w オプションを使用しない運用としてください。

KNAX7260-W

JP1/AJS のジョブの実行 ID 取得に失敗しました。

JP1/AJS のジョブの実行 ID 取得に失敗しました。

(S)

「-----」を設定し処理を続行します。

KNAX7261-I

アプリケーション実行エージェントプログラムは，実行アプリケーションの終了待ちを行います。
Detail=**保守情報**

アプリケーション実行エージェントプログラムは，実行アプリケーションの終了待ちを行います。

保守情報

プロセスの識別情報

(S)

処理を続行します。

KNAX7262-I

アプリケーション実行エージェントプログラムは，実行アプリケーションの終了待ちを行いません。
Detail=**保守情報**

アプリケーション実行エージェントプログラムは，実行アプリケーションの終了待ちを行いません。

保守情報

プロセスの識別情報

(S)

処理を続行します。

KNAX7263-E

アプリケーション実行エージェントプログラムの終了に失敗しました。(API=**関数名**, error code=**エラーコード**)

アプリケーション実行エージェントプログラムの終了に失敗しました。

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

(S)

処理を続行します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7264-I

実行アプリケーションが終了しました。viewname=**表示名**, code=**終了コード**, pid=**プロセスID**)

実行アプリケーションが終了しました。

表示名

実行が完了した実行アプリケーションの表示名

表示名の指定がなかった場合は，実行アプリケーション名を出力します。

終了コード

実行が完了した実行アプリケーションの終了コード

プロセスID

アプリケーション実行エージェントプログラムのプロセス ID

(S)

処理を続行します。

KNAX7268-I

実行アプリケーションを起動しました。viewname=**表示名**, pid=**プロセスID**)

実行アプリケーションを起動しました。

表示名

実行が完了した実行アプリケーションの表示名

表示名の指定がなかった場合は，実行アプリケーション名を出力します。

プロセスID

アプリケーション実行エージェントプログラムのプロセス ID

(S)

処理を続行します。

KNAX7269-E

アプリケーション実行エージェント機能用共有メモリの内容出力に失敗しました。(API=**関数名**, error code=**エラーコード**)

アプリケーション実行エージェント機能用共有メモリの内容出力に失敗しました。

関数名

エラーの発生した OS の関数名

エラーコード

エラー情報

(S)

処理を続行します。

(O)

関数名とエラーコードからエラー要因を取り除いて，コマンドを再実行してください。エラー要因を取り除けない場合はシステム管理者に連絡してください。

KNAX7270-E

アプリケーション実行エージェントプログラムは起動されていませんでした。

アプリケーション実行エージェントプログラムは起動されていませんでした。

(S)

処理を中断します。

KNAX7271-I

使用方法: **コマンド名 コマンド引数**

コマンド名および**コマンド引数**で示すコマンドの指定が誤っています。

出力する usage の内容の例

KNAX7271-I 使用方法: adshappexec [-m] [-d work-folder] [-v view-name] {-w application-path-name | -n application-path-name} [-- application-parameter...]

(S)

処理を中断します。

(O)

コマンドを正しく指定して実行してください。

KNAX7400-E

応答要求メッセージの出力数が上限を超えました。

adshread コマンドで応答要求メッセージの出力数が上限を超えました。

(S)

処理を繰り返します。

(O)

USERREPLY_WAIT_MAXCOUNT パラメーターの設定を見直してください。

KNAX7402-E

応答要求メッセージの処理中に，共有メモリの操作でエラーが発生しました。(errinfo=**エラー情報**, function=**保守情報**)

adshread コマンドで応答要求メッセージの処理中に，共有メモリの操作でエラーが発生しました。

エラー情報

エラーが発生した API が出力する状態コード

保守情報

エラーが発生した API 名

(S)

処理を続行します。

(O)

エラー情報を確認して対処してください。**エラー情報**については，「12.4.4　ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照してください。

KNAX7403-E

引数の数が不当です。filename="**ファイル名**" line=**行番号**

adshecho コマンドまたは adshread コマンドの引数の数が不当です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX7404-E

adshread コマンドに指定した変数（"**変数名**"）の型が不正です。filename="**ファイル名**" line=**行番号**

adshread コマンドに指定した変数が数値型です。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX7405-E

adshecho コマンドに指定した事象通知メッセージ，または adshread コマンドに指定した応答要求メッセージが長すぎます。filename="**ファイル名**" line=**行番号**

adshecho コマンドに指定した事象通知メッセージ，または adshread コマンドに指定した応答要求メッセージが長過ぎます。

ファイル名

ジョブ定義スクリプトファイル名

行番号

エラーが発生したジョブ定義スクリプトファイルの行番号

(S)

処理を続行します。

(O)

ジョブ定義スクリプトファイルを修正してください。

KNAX7408-E

内部エラーが発生しました。(details=**保守情報**)

adshecho コマンドまたは adshread コマンド処理で内部エラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX7420-E

標準入力から応答入力した内容に ASCII 文字以外が指定されました。再度入力してください。

標準入力から応答入力した内容に ASCII 文字以外が指定されました。

(S)

処理を続行します。

(O)

応答入力の内容に ASCII 文字を指定して再度応答してください。

KNAX7450-I 【Windows 限定】

応答要求メッセージをキャンセルしました。(**ジョブ識別子**, **行番号**, **ホスト名**)

ジョブコントローラが強制終了を受け付けたため，adshread コマンドによる応答要求メッセージをキャンセルしました。

ジョブ識別子

JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

行番号

adshread コマンドを発行したジョブ定義スクリプトの行番号

ホスト名

サービスまたはデーモンが稼働しているホストの名称

(S)

処理を終了します。

KNAX7450-I 【UNIX 限定】

adshexec が応答要求をキャンセルしました。(**ジョブ識別子**, **行番号**, **ホスト名**)

ジョブコントローラが強制終了を受け付けたため，adshread コマンドによる応答要求メッセージをキャンセルしました。

ジョブ識別子

JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

行番号

adshread コマンドを発行したジョブ定義スクリプトの行番号

ホスト名

サービスまたはデーモンが稼働しているホストの名称

(S)

処理を終了します。

KNAX7451-I

応答要求メッセージをキャンセルしました。(**ジョブ識別子**, **行番号**, **ホスト名**)

ジョブコントローラが終了要求を受け付けたため，adshread コマンドによる応答要求メッセージをキャンセルしました。

ジョブ識別子

JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

行番号

adshread コマンドを発行したジョブ定義スクリプトの行番号

ホスト名

サービスまたはデーモンが稼働しているホストの名称

(S)

処理を終了します。

KNAX7460-E

JP1 イベントの発行に失敗しました。(errinfo=**エラー情報**, function=**保守情報**)

JP1 イベントの発行に失敗しました。

エラー情報

エラーが発生した API が出力する状態コード

保守情報

エラーが発生した API 名

(S)

処理を続行します。

(O)

エラー情報を確認して対処してください。**エラー情報**については，「12.4.4　ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照してください。

KNAX7461-E

JP1 イベントの発行に失敗しました。(errinfo=**エラー情報**, function=open_sender, host=**ホスト名**)

JP1 イベントの発行に失敗しました。

エラー情報

エラーが発生した API が出力する状態コード

ホスト名

JP1/Advanced Shell が稼働するバッチ業務サーバのホスト名

(S)

処理を続行します。

(O)

エラー情報を確認して対処してください。**エラー情報**については，「12.4.4　ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照してください。

KNAX7462-E

JP1 イベントの発行に失敗しました。(errinfo=**エラー情報**, function=**保守情報**, dest=**JP1/IM - Managerが稼働している運用管理サーバのホスト名**, seqno=**発行元イベントDB内通し番号**)

JP1 イベントの発行に失敗しました。

エラー情報

エラーが発生した API が出力する状態コード

保守情報

event_send（イベント送信）または check_arrival（イベント到達確認）

JP1/IM - Managerが稼働している運用管理サーバのホスト名

システム環境ファイルの HOSTNAME_JP1IM_MANAGER パラメーターで指定したホスト名。HOSTNAME_JP1IM_MANAGER パラメーターの指定を省略した場合は，JP1/Advanced Shell が稼働するバッチ業務サーバの物理ホスト名。

発行元イベントDB内通し番号

JP1/Base の発行元イベント DB 内通し番号

(S)

処理を続行します。

(O)

エラー情報を確認して対処してください。**エラー情報**については，「12.4.4　ユーザー応答機能で表示されるエラー情報の意味および対処方法」を参照してください。

ただし，このメッセージが表示された場合でも，JP1/IM - View にイベントが到達している可能性があります。到達したイベントが応答待ちイベントの場合は，手動で応答待ちイベントの滞留を解除してください。

KNAX7464-E

JP1 イベントの転送に失敗しました。(function=check_arrival, dest=**転送先ホスト名**, seqno=**発行元イベントDB内通し番号**)

自ホストの JP1/Base から JP1/IM - Manager が稼働しているホストの JP1/Base に対して JP1 イベントの転送に失敗しました。

転送先ホスト名

システム環境ファイルの HOSTNAME_JP1IM_MANAGER パラメーターで指定したホスト名

発行元イベントDB内通し番号

JP1/Base の発行元イベント DB 内通し番号

(S)

処理を続行します。

(O)

次の点を確認してください。

- JP1/IM - Manager が稼働しているホストに JP1/Base がインストールされているか。
- JP1/IM - Manager が稼働しているホストで JP1/Base のイベントサーバが起動しているか。
- 自ホストと JP1/IM - Manager が稼働しているホストの間で，JP1/Base のコネクションが確立されているか。

ただし，このメッセージが表示された場合でも，JP1/IM - View にイベントが到達している可能性があります。到達したイベントが応答待ちイベントの場合は，手動で応答待ちイベントの滞留を解除してください。

KNAX7465-W

HOSTNAME_JP1IM_MANAGER で指定されたホストに JP1 イベントを転送中です。
(function=check_arrival, dest=**転送先ホスト名**, seqno=**発行元イベントDB内通し番号**)

JP1/Base が，HOSTNAME_JP1IM_MANAGER で指定されたホストに JP1 イベントを転送中です。

転送先ホスト名

システム環境ファイルの HOSTNAME_JP1IM_MANAGER パラメーターで指定したホスト名

発行元イベントDB内通し番号

JP1/Base の発行元イベント DB 内通し番号

(S)

処理を続行します。

KNAX7470-I

JP1 イベント発行のための流量制御をします。(wait=**待ち時間**)

JP1 イベント発行のための流量制御をします。

待ち時間

流量制御のために JP1 イベントの発行を遅らせた時間（ミリ秒）

(S)

処理を続行します。

KNAX7500-I

adshmd が起動しました。

ユーザー応答機能管理デーモンが起動しました。

(S)

処理を続行します。

KNAX7501-I

adshmd が終了しました。

ユーザー応答機能管理デーモンが終了しました。

(S)

処理を続行します。

KNAX7502-E

adshmd が起動できません。

ユーザー応答機能管理デーモンの起動に失敗しました。

(S)

処理を終了します。

(O)

原因を取り除いて，再起動してください。

KNAX7503-E

adshmd でエラーが発生しました。

ユーザー応答機能管理デーモンでエラーが発生しました。

(S)

処理を終了します。

(O)

原因を取り除いて，再起動してください。

KNAX7508-I

プログラム名が応答要求をキャンセルしました。 (**ジョブ識別子**, **行番号**, **ホスト名**)

adshread コマンドによる応答要求メッセージをキャンセルしました。

プログラム名

応答要求メッセージをキャンセルしたサービス名またはデーモン名

ジョブ識別子

JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

行番号

adshread コマンドを発行したジョブ定義スクリプトの行番号

ホスト名

サービスまたはデーモンが稼働しているホストの名称

(S)

処理を続行します。

KNAX7509-I

adshchmsg が応答要求をキャンセルしました。 (**ジョブ識別子**, **行番号**, **ホスト名**)

adshchmsg コマンドの-n オプションで指定した応答要求メッセージ番号の応答要求メッセージをキャンセルしました。

ジョブ識別子

JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

行番号

adshread コマンドを発行したジョブ定義スクリプトの行番号

ホスト名

サービスまたはデーモンが稼働しているホストの名称

(S)

処理を続行します。

KNAX7550-I

サービスの登録に成功しました。

AdshmSvcD サービスまたは AdshmSvcE サービスの登録に成功しました。

(S)

処理を続行します。

KNAX7551-E

サービスの登録が失敗しました。(API=**保守情報**, error code=**エラーコード**)

AdshmSvcD サービスまたは AdshmSvcE サービスの登録に失敗しました。

(S)

処理を中止します。

(O)

すでに AdshmSvcD サービスまたは AdshmSvcE サービスが登録されていることが原因である場合は，このメッセージは無視してかまいません。

adshmsvcd コマンドまたは adshmsvce コマンドを実行してこのメッセージが出力された場合は，不当なオプションが指定されたおそれがあるため，オプションの指定内容を見直してください。

それ以外の場合は，システム管理者に連絡してください。

KNAX7552-E

{AdshmSvcE | AdshmSvcD} サービスの初期化に失敗しました。(API=**保守情報**, error code=**エラーコード**)

AdshmSvcD サービスまたは AdshmSvcE サービスの初期化処理で，エラーが発生しました。

(S)

処理を中止します。

(O)

保守情報が jhs_env_conf_readConfig の場合，システム環境ファイルの内容が不正な場合があるため，内容を見直してください。

保守情報が jhs_env_conf_readConfig 以外の場合およびシステム環境ファイルの内容が正しい場合は，システム管理者に連絡してください。

KNAX7553-E

API "**API名**" でエラーが発生しました。(error code=**エラーコード**)

AdshmSvcD サービスまたは AdshmSvcE サービスの処理中に，Mutex の生成またはオープンに失敗しました。

(S)

処理を中止します。

(O)

AdshmSvcD サービスまたは AdshmSvcE サービスを起動するアカウントに対し，サーバの管理者権限を持つユーザーが指定されていないと，このエラーが発生する場合があります。

指定誤りの場合は修正して再実行してください。誤りでない場合はシステム管理者に連絡してください。

KNAX7556-E

サービスの登録で指定された論理ホスト名が長過ぎます。

サービスの登録で指定された論理ホスト名が長過ぎます。

(S)

サービスの登録処理を中止します。

(O)

適切な論理ホスト名を指定してコマンドを再度実行してください。

KNAX7560-I

サービスの設定処理が成功しました。

AdshmSvcD サービスまたは AdshmSvcE サービスの設定処理に成功しました。

(S)

処理を続行します。

KNAX7561-E

サービスの設定処理が失敗しました。(API=**保守情報**, error code=**エラーコード**)

AdshmSvcD サービスまたは AdshmSvcE サービスの設定処理が失敗しました。ユーザー応答機能の設定が誤っている場合，このメッセージが出力されることがあります。

(S)

処理を中止します。

(O)

システム管理者に連絡してください。システム管理者はユーザー応答機能の設定を見直してください。

KNAX7600-E

カスタムジョブ定義プログラムの起動方法が不正です。

JP1/AJS - View が起動したカスタムジョブ定義プログラムの起動情報が不正です。

(S)

処理を終了します。

(O)

次に示すどちらかの対策をします。

- カスタムジョブ定義情報が不正な場合があるため，カスタムジョブを再定義します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7601-E

起動情報の形式が不正です。

JP1/AJS - View が起動したカスタムジョブ定義プログラムの起動情報が不正です。

(S)

処理を終了します。

(O)

次に示すどちらかの対策をします。

- カスタムジョブ定義情報が不正な場合があるため，カスタムジョブを再定義します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7602-E

Advanced Shell のカスタムジョブ登録を正しく行ってください。

JP1/AJS - View のカスタムジョブ登録で設定した内容が不正です。

(S)

処理を終了します。

(O)

次に示すどちらかの対策をします。

- カスタムジョブ登録情報を確認します。

- JP1/Advanced Shell - Custom Job を再インストールして，定義プログラムを新しくします。

KNAX7603-E

Advanced Shell の定義プログラムの内容が不正です。

JP1/AJS - View のカスタムジョブ登録で設定した定義プログラムの内容が不正です。

(S)

処理を終了します。

(O)

次に示すどちらかの対策をします。

- JP1/AJS - View のカスタムジョブ登録で設定した定義プログラムの内容を確認します。
- JP1/Advanced Shell - Custom Job を再インストールして，定義プログラムを新しくします。

KNAX7604-E

定義項目名に不正な文字が入力されています。

ジョブ定義画面の**定義項目名**で示すフィールドに不正な文字が入力されました。

(S)

処理を中断し，入力画面に戻ります。

(O)

定義項目名で示すフィールドに入力されている不正な文字を削除します。

KNAX7605-E

定義項目名を入力してください。

ジョブ定義画面の**定義項目名**で示す，入力が必須のフィールドの指定が省略されました。

(S)

処理を中断し，入力画面に戻ります。

(O)

定義項目名で示すフィールドに値を入力してください。

KNAX7606-E

登録済みの定義情報が不正です。

登録済みの定義情報が不正です。

(S)

処理を中断します。ジョブの定義は変更されていません。

(O)

次に示すどれかの対策をします。

- JP1/AJS - View のジョブネットエディタ上で作成したジョブを作り直し，再実行します。
- ajsdefine コマンドおよび JP1/AJS - Definition Assistant でジョブを定義した場合は，指定できる文字の種別や文字列の長さを見直してジョブを作り直し，再実行します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7607-E

Advanced Shell の定義情報を登録できません。(error code = **エラーコード**)

JP1/AJS にジョブの定義を登録できません。

(S)

処理を中断します。ジョブの定義は変更されていません。

(O)

次に示すどちらかの対策をします。

- JP1/AJS - View のジョブネットエディタ上で作成したジョブを作り直し，再実行します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7608-E

Advanced Shell の定義情報を登録できません。(error code = **エラーコード**) / (reason = **エラー詳細**)

JP1/AJS にジョブ定義を登録できません。

(S)

処理を中断します。ジョブの定義は変更されていません。

(O)

次に示すどちらかの対策をします。

- JP1/AJS - View のジョブネットエディタ上で作成したジョブを作り直し，再実行します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7609-E

前回登録した Advanced Shell の定義情報を取得できません。(error code = **エラーコード**)

前回登録した JP1/Advanced Shell の定義情報を取得できません。

(S)

処理を中断します。ジョブ定義は変更されていません。

(O)

次に示すどちらかの対策をします。

- JP1/AJS - View のジョブネットエディタ上で作成したジョブを作り直し，再実行します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7610-E

入力情報の破棄に失敗しました。(reason = **エラー詳細**)

入力した情報の破棄が失敗しました。

(S)

処理を中断します。ジョブの定義は破棄されていません。

(O)

次に示すどちらかの対策をします。

- JP1/AJS - View のジョブネットエディタ上で作成したジョブを作り直し，再実行します。
- JP1/AJS - View のバージョンが，前提バージョンであることを確認します。

KNAX7611-E

論理エラーが発生しました。(**関数ID**)

JP1/Advanced Shell の定義で，内部的な論理エラーが発生しました。

(S)

処理を中断します。

(O)

システム管理者に連絡します。

KNAX7750-E

論理エラーが発生しました。(**関数ID**) / (reason = **エラー詳細**)

メッセージ出力処理で，内部的な論理エラーが発生しました。

(S)

処理を中断します。

(O)

システム管理者に連絡します。

KNAX7770-E

ヘルプファイルの起動に失敗しました。(error code = **エラーコード**)

ヘルプファイルの起動に失敗しました。

(S)

元の画面に戻ります。

(O)

エラーコードに示される ShellExecute 関数の終了コードを調べ，適切に処置します。

KNAX7771-E

ヘルプファイルが見つかりません。(file name = **ファイル名**)

ヘルプファイルが見つかりません。

(S)

元の画面に戻ります。

(O)

JP1/Advanced Shell - Custom Job の修復インストールを試みます。

KNAX7772-E

ヘルプファイルの起動に失敗しました。(reason = **エラー詳細**)

ヘルプファイルの起動に失敗しました。

(S)

元の画面に戻ります。

(O)

JP1/Advanced Shell - Custom Job の修復インストールを試みます。

KNAX7773-E

ヘルプファイルの起動に失敗しました。

ヘルプファイルの起動に失敗しました。

(S)

元の画面に戻ります。

(O)

JP1/Advanced Shell - Custom Job の修復インストールを試みます。

KNAX7800-I

adshcollect:RAS completed collection of **ファイル名**

採取したファイル名

採取したファイル名

…

採取したファイル名をまとめて tar ファイル（**ファイル名**）を作成しました。

(S)

処理を終了します。

(O)

作成されたファイルをシステム管理者に渡してください。

KNAX7801-I

```
adshcollect:RAS completed collection of ファイル名
  採取したファイル名
  採取したファイル名
  …
```

採取したファイル名の内容をまとめて**ファイル名**のファイルを作成しました。

(S)

処理を終了します。

(O)

作成されたファイルをユーザーの圧縮ツールを使用して圧縮し，システム管理者に渡してください。

KNAX7802-E

```
Usage: adshcollect Directory [-f FileName] [-e FileName] [-h LogicalHostName]
        Directory : Specify output directory
        -f FileName : Specify a config file
        -e FileName : Specify an environment file
        -h LogicalHostName : Specify a logical host
```

オプションの設定が誤っています。

(S)

処理を終了します。

(O)

オプションを正しく設定し，再実行します。

KNAX7803-E

```
adshcollect:RAS error:出力先ディレクトリ(Permission denied).
```

出力先ディレクトリにアクセス権限がありません。

(S)

処理を終了します。

(O)

出力先ディレクトリにアクセス権を付与するか，別のディレクトリを指定して再実行します。

KNAX7804-E

adshcollect:RAS error:**出力先ディレクトリ**(not found or not a directory).

出力先ディレクトリがないかまたはディレクトリではありません。

(S)

処理を終了します。

(O)

正しい出力先を指定して再実行します。

KNAX7805-E

adshcollect:RAS error:**定義ファイル名**(not found or not a file).

定義ファイル名がないかまたはファイルではありません。

(S)

処理を終了します。

(O)

正しい定義ファイルを指定して再実行します。

KNAX7806-E

adshcollect:RAS error:**定義ファイル名**(Permission denied).

定義ファイル名にアクセス権限がありません。

(S)

処理を終了します。

(O)

定義ファイルへのアクセス権限を設定して再実行します。

KNAX7807-E

adshcollect:RAS error:**環境ファイル名**(not found or not a file).

環境ファイル名がないか，またはファイルではありません。

(S)

処理を終了します。

(O)

正しい環境ファイルを指定して再実行します。

KNAX7808-E

adshcollect:RAS error:**環境ファイル名**(Permission denied).

環境ファイル名にアクセス権限がありません。

(S)

処理を終了します。

(O)

環境ファイルへのアクセス権限を設定して再実行します。

KNAX7809-E

adshcollect:RAS error:**定義ファイル名**("**キーワード**" Syntax Error).

定義ファイル中に不正なキーワードが指定されました。

(S)

処理を終了します。

(O)

定義ファイルを正しく設定して再実行します。

KNAX7810-E

adshcollect:RAS error:**指定値**(not found or not a file).

定義ファイルのキーワードの**指定値**がないか，またはファイルではありません。

(S)

処理を終了します。

(O)

定義ファイルを正しく設定して再実行します。

KNAX7811-W

adshcollect:RAS error:**指定値**(Permission denied).

定義ファイルのキーワードの**指定値**にアクセス権限がありません。

(S)

処理を継続します。

(O)

指定値にアクセス権を付与するか，または別の値を指定して再実行します。

KNAX7812-E

adshcollect:RAS error:**指定値**(not found or not a directory).

定義ファイルの**指定値**が存在しないか，またはディレクトリではありません。

(S)

処理を終了します。

(O)

定義ファイルを正しく設定して再実行します。

KNAX7813-E

adshcollect:RAS error:**指定値**(not found or not a directory).

環境ファイルの必須キーワードの**指定値**がないか，またはディレクトリではありません。

(S)

処理を終了します。

(O)

環境ファイルを正しく設定して再実行します。

KNAX7814-E

adshcollect:RAS error:**指定値**(Permission denied).

環境ファイルのキーワードの**指定値**にアクセス権限がありません。

(S)

処理を終了します。

(O)

指定値にアクセス権を付与するか，または別の値を指定して再実行します。

KNAX7880-E

Failed to "**OSのAPI名**". (reason=**エラー詳細**)

エラー詳細に示す原因で，OS の API の処理に失敗しました。

(S)

OSのAPI名が"dladdr"以外の場合，処理を終了します。"dladdr"の場合は処理を続行します。

(O)

OSのAPI名が"dladdr"以外の場合，システム管理者に連絡します。エラー詳細の原因を取り除いて再実行します。

KNAX7892-I

adshexec received abnormal signal.

ジョブコントローラに対するプログラム異常通知シグナルを受け取りました。

(S)

処理を続行します。

KNAX7893-I

adshexec received signal "**シグナル名**".

ジョブコントローラに対する**シグナル名**に示すシグナルを受け取りました。このメッセージは，ジョブコントローラが実行の制御に必要なシグナルを受信した場合に出力されます。

(S)

処理を続行します。

KNAX7894-E

adshexec is ended because of terminate request of second times.

ジョブコントローラに対する 2 回目の SIGTERM のシグナルを受け取りました。

(S)

ジョブコントローラは，即時に終了します。一時ファイルの削除やファイルの後始末などの後処理を行いません。

(O)

必要に応じてバッチジョブで作成された資源の後処理をしてください。

KNAX7895-E

adshexec ended abnormally.

ジョブコントローラがエラー終了しました。

(S)

ジョブコントローラは，即時に終了します。一時ファイルの削除やファイルの後始末などの後処理を行いません。

(O)

必要に応じてバッチジョブで作成された資源の後処理をしてください。

KNAX7896-I

adshexec received terminate request.

ジョブコントローラに対する終了要求を受け取りました。

(S)

後処理をして，終了します。デバッグ実行で停止中の場合，ジョブ定義スクリプトを再実行してから後処理をして，終了します。

KNAX7897-E

Fatal error occurred in 保守情報.

adshexec コマンドの重大なエラーが発生しました。

(S)

処理を終了します。

(O)

システム管理者に連絡します。

KNAX7900-I

マニュアルは組み込まれていません。
マニュアルのインストールメディアから HTML ファイルと画像ファイルをコピーしてください。

マニュアルのインストールメディアからインストール先ディレクトリへマニュアルをコピーしていません。

(S)

［閉じる］ボタンを押すまで Web ブラウザを開いた状態となります。

(O)

［閉じる］ボタンを押して Web ブラウザを閉じ，マニュアルに記載された手順に従ってマニュアルのインストールメディアからインストール先ディレクトリへマニュアルをコピーしてください。

KNAX7901-I

ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスの完了を待ちます。

ジョブコントローラは，ジョブ終了時にすべての非同期実行プロセスを wait します。

このメッセージは，ルートジョブの環境ファイルの SPOOLJOB_CHILDJOB パラメーターに MERGE を指定して起動された子孫ジョブの場合は出力されません。

(S)

処理を続行します。

KNAX7902-I

ジョブコントローラは，"**入力モード**"で動作します。

ジョブコントローラは，**入力モード**で示すモードで動作します。

入力モード

ジョブコントローラの入力モードとして次のどちらかが出力されます。入力モードについての詳細は，「(2)　ジョブの入力モード」を参照してください。

出力内容	意味
端末入力モード	端末入力モード。標準入力が端末に関連づけられています。
非端末入力モード	非端末入力モード。標準入力が端末に関連づけられていません。

(S)

処理を続行します。

KNAX7999-I

ルートジョブのジョブコントローラがバッチジョブを終了しました。rc=**終了コード**

ルートジョブのジョブコントローラが**終了コード**で示す終了コードでバッチジョブを終了しました。

(S)

処理を続行します。

KNAX9000-E

The validity period for a product expired. program=**コマンド名**

ライセンスの有効期限が過ぎました。

コマンド名

エラーが発生したコマンド名

(S)

処理を終了します。

(O)

引き続き使用する場合は，製品版をインストールしてください。

KNAX9001-E

Failed to authenticate the product. (**コマンド名**, **内部情報**)

ライセンスの認証処理に失敗しました。

コマンド名

エラーが発生したコマンド名

内部情報

エラーの内容を示す内部情報

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

KNAX9002-E

An error occurred. detail=**コマンド名**, adshhlicauth error, rc=**終了コード**

ライセンスの認証処理で予期しないエラーが発生しました。

コマンド名

エラーが発生したコマンド名

終了コード

エラーの内容を示す内部情報

(S)

処理を終了します。

(O)

システム管理者に連絡してください。

12.4　エラーの詳細

メッセージテキストに表示される**エラー詳細**の内容について，Windows の場合，UNIX の場合，JP1/Advanced Shell 固有の場合に分けて説明します。

12.4.1　エラーの詳細（Windows の場合）

JP1/Advanced Shell が出力するメッセージは，C ランタイムの関数および Win32(R) API のエラー情報を含む場合があります。

JP1/Advanced Shell の環境で発生しやすい，代表的な C ランタイム関数のエラー情報に対する原因と対策（Windows の場合）を次の表に示します。表にないエラーおよび Win32 API のエラー情報については，使用している Windows のマニュアルを参照してください。

表 12-5　C ランタイム関数のエラー情報に対する原因と対策（Windows の場合）

ニモニック	エラーの詳細	原因	対策
ENOENT	No such file or directory	次の原因が考えられます。 • ファイル，またはディレクトリが見つかりません。 • シンボリックリンクのネストが深過ぎます。	次の対策を実施します。 • ファイルの存在を確認してください。 • シンボリックリンクのネスト数が OS の上限を超えないように変更してください。
EIO	Input/Output error	入出力エラーが発生しました。	Windows またはハードウェアの情報に従ってください。
ENXIO	No such device or address	ファイルに対するアクセス権がありません。	デバイスがあるか，またはデバイスを有効にしているかを確認してください。デバイスを有効にしていない場合は，有効にしてください。それ以外の原因の場合は，使用している Windows のマニュアルを参照してください。
E2BIG	Arg list too long	処理プログラムの引数または環境変数用の領域が不足しています。	処理プログラムの引数を確認します。 export パラメーターなどによる環境変数の設定やファイル管理機能のスクリプト拡張コマンドの使用方法を見直し，不要な環境変数の設定を削除します。
EAGAIN	Resource temporarily unavailable	プロセスの数が多過ぎるか，または一時的なメモリ不足が発生しています。	再実行してもエラーが発生する場合は，不要なプロセスを停止させてください。
ENOMEM	Not enough space	次の原因が考えられます。 • スワップ領域または仮想メモリの不足のため，プロセスを新しく生成できません。	次の対策を実施します。 • スワップ領域または仮想メモリが足りない場合は，拡張してください。拡張できない場合は，不要なプロセスを停止させてください。

ニモニック	エラーの詳細	原因	対策
ENOMEM	Not enough space	• プロセスの数が多過ぎるか，または一部のプロセスが大量のメモリを消費しています。	• 一部のプロセスが大量のメモリを消費している場合は，該当するプロセスをいったん停止できないかどうかを検討してください。
EACCES	Permission denied	次の原因が考えられます。 • アクセス権限が不正です。 • JP1/Advanced Shell のコマンドの引数として，ファイルを指定する場所にディレクトリを指定しました。 • コマンドとして実行しようとしたファイルの拡張子が exe，bat，cmd，または com ではありません。 • ファイルへのシンボリックリンクのリンク先がディレクトリ，またはディレクトリへのシンボリックリンクのリンク先が通常ファイルのシンボリックリンクにアクセスしました。	次の対策を実施します。 • ファイルに対するアクセス権限が正しいかどうかを確認してください。 • JP1/Advanced Shell のコマンドの引数を見直し，ファイルを指定する場所にディレクトリを指定していないかどうかを確認してください。 • CHILDJOB_SHEBANG パラメーターを指定している場合，パラメーターの指定値，および，実行しようとしたファイルの先頭の「#! **実行プログラムパス**」を確認してください。 • CHILDJOB_EXT パラメーターを指定している場合は，パラメーターの指定値，および，実行しようとしたファイルの拡張子を確認してください。 • シンボリックリンクの参照先が正しいか確認してください。
EFAULT	Bad address	アクセスできない領域に書き込みをしようとしました。書き込みをしようとしたディスクが切り離された場合があります。	系切り替えに伴うディスクの切り替え中の場合は，問題ないので無視してください。 誤ってディスクを切り離してしまった場合は，該当するファイルをバックアップから回復するか，または初期化してから使用してください。 上記以外の場合は，システム管理者に連絡してください。
EEXIST	File exists	作成しようとしたファイルはすでにあります。	ファイル名を変更して再実行します。既存のファイルが不要の場合，削除してから再実行してください。
EINVAL	Invalid argument	次の原因が考えられます。 • メモリ管理情報の不正を検知しました。 • シンボリックリンクのネストが深すぎます。 • シンボリックリンク作成権限を持たないユーザでシンボリックリンクの作成，コピー，または移動を行おうとしました。	次の対策を実施します。 • システム管理者に連絡してください。 • シンボリックリンクのネスト数が OS の上限を超えないように変更してください。 • シンボリックリンク作成権限を持つユーザで実行してください。 • シンボリックリンクを使用する場合は，NTFS を使用してください。

ニモニック	エラーの詳細	原因	対策
EINVAL	Invalid argument	• NTFS 以外のファイルシステム上にシンボリックリンクの作成，コピー，または移動を行おうとしました。	次の対策を実施します。 • システム管理者に連絡してください。 • シンボリックリンクのネスト数が OS の上限を超えないように変更してください。 • シンボリックリンク作成権限を持つユーザで実行してください。 • シンボリックリンクを使用する場合は，NTFS を使用してください。
ENFILE	Too many open files in system	ファイルのオープン数がシステムの上限を超えました。	システム全体で使用中のファイルの数を確認し，不要なファイルを閉じてください。
EMFILE	Too many open files	該当するプロセスでオープンしているファイル数が多過ぎます。	システム管理者に連絡してください。
EFBIG	File too large	ファイルの大きさがシステム制限値を超えました。	使用するファイルサイズを見直してください。
ENOSPC	No space left on device	ファイルシステムに十分な空き領域がありません。	空き領域を確保してください。

12.4.2　エラーの詳細（UNIX の場合）

JP1/Advanced Shell の環境で発生しやすいエラーの詳細に対する原因と対策を次の表に示します。表にないエラーについては，使用している UNIX のマニュアルを参照してください。

JP1/Advanced Shell の環境で発生しやすいエラーの内容だけを記載しています。記載されていないエラーの詳細については，メッセージで表示されたエラー番号（errno）に該当するニモニックを使用している UNIX の errno 定義ファイル（errno.h）を調べてください。

表 12-6　エラーの詳細に対する原因と対策（UNIX の場合）

ニモニック	エラーの詳細	原因	対策
ENOENT	No such file or directory	次のどちらかの原因が考えられます。 • ファイル，またはディレクトリが見つかりません。 • シンボリックリンクのネストが深すぎます。	次の対策を実施します。 • ファイルの存在を確認してください。 • シンボリックリンクのネスト数が OS の上限を超えないように変更してください。
EIO	I/O error	入出力エラーが発生しました。	UNIX またはハードウェアの情報に従ってください。
ENXIO	No such device or address	ファイルに対するアクセス権がありません。	デバイスがあるか，またはデバイスを有効にしているかを確認してください。デバイスを有効にしていない場合は，有効

ニモニック	エラーの詳細	原因	対策
ENXIO	No such device or address	ファイルに対するアクセス権がありません。	にしてください。それ以外の原因の場合は，使用している UNIX のマニュアルを参照してください。
E2BIG	Arg list too long	処理プログラムの引数または環境変数用の領域が不足しています。	処理プログラムの引数を確認します。 export パラメーターなどによる環境変数の設定やファイル管理機能のスクリプト拡張コマンドの使用方法を見直し，不要な環境変数の設定を削除します。
EAGAIN	Resource temporarily unavailable	プロセスの数が多過ぎるか，または一時的なメモリ不足が発生しています。	再実行してもエラーが発生する場合は，不要なプロセスを停止させてください。
ENOMEM	Not enough space	次の原因が考えられます。 • スワップ領域または仮想メモリの不足のため，プロセスを新しく生成できません。 • プロセスの数が多過ぎるか，または一部のプロセスが大量のメモリを消費しています。	次の対策を実施します。 • スワップ領域または仮想メモリが足りない場合は，拡張してください。拡張できない場合は，不要なプロセスを停止させてください。 • 一部のプロセスが大量のメモリを消費している場合は，該当するプロセスをいったん停止できないかどうかを検討してください。
EACCES	Permission denied	アクセス権限が不正です。	ファイルに対するアクセス権限が正しいかどうかを確認してください。
EFAULT	Bad address	アクセスできない領域に書き込みをしようとしました。書き込みをしようとしたディスクが切り離された場合があります。	系切り替えに伴うディスクの切り替え中の場合は，問題ないので無視してください。 誤ってディスクを切り離してしまった場合は，該当するファイルをバックアップから回復するか，または初期化してから使用してください。 上記以外の場合は，システム管理者に連絡してください。
EEXIST	File exists	作成しようとしたファイルはすでにあります。	ファイル名を変更して再実行します。既存のファイルが不要の場合，削除してから再実行してください。
EINVAL	Invalid argument	メモリ管理情報の不正を検知しました。	システム管理者に連絡してください。
ENFILE	File table overflow	ファイルのオープン数がシステムの上限を超えました。	UNIX のカーネルパラメーターの，システムでオープンできるファイル最大数（maxuproc×nofiles）の指定値を大きくしてください。
EMFILE	Too many open files	該当するプロセスでオープンしているファイル数が多過ぎます。	UNIX のカーネルパラメーターの，プロセスでオープンできるファイル数の最大値（nofiles）を大きくしてください。

ニモニック	エラーの詳細	原因	対策
EFBIG	File too large	ファイルの大きさがシステム制限値を超えました。	使用するファイルサイズを見直してください。
ENOSPC	No space left on device	ファイルシステムに十分な空き領域がありません。	空き領域を確保してください。
ENAMETOOLONG	File name too long	ファイル名の長さが長過ぎます。	ファイル名の長さを見直してください。

12.4.3　エラーの詳細（JP1/Advanced Shell 固有の場合）

JP1/Advanced Shell が固有に出力するエラーの詳細に対する原因と対策を次の表に示します。

表 12-7　エラーの詳細に対する原因と対策（JP1/Advanced Shell 固有の場合）

メッセージ ID	エラーの詳細	原因	対処
KNAX4419-E	1 行のサイズが上限を超えています。	1 行のサイズが上限を超えています。	行のサイズを見直し，上限以内で記述します。
KNAX4420-E	共通アプリケーションフォルダが見つかりません。	共通アプリケーションフォルダが見つかりません。	実行環境に問題がないか確認します。
	共有ドキュメントフォルダが見つかりません。	共有ドキュメントフォルダが見つかりません。	実行環境に問題がないか確認します。
KNAX6035-E	ファイル識別子の指定が正しくありません。	指定されたファイル識別子が 1 桁の数字でありません。	ファイル識別子の指定を見直します。
	使用可能なファイル識別子ではありません。	オープンしていないファイルや，ほかのプロセスによって操作が禁止されているファイルに対するファイル識別子を指定しました。	ファイル識別子をオープンしているかを確認します。オープンしている場合は，ほかのプロセスによるロックなどで操作が禁止されていないかを確認します。
	書き込みでオープンされたファイル識別子ではありません。	書き込みでオープンされていないファイル識別子への書き込みを指定しました。	ファイル識別子の指定を見直します。
	読み込みでオープンされたファイル識別子ではありません。	読み込みでオープンされていないファイル識別子に読み込みを指定しました。	ファイル識別子の指定を見直します。
	バックグラウンドプロセスが存在しません。	バックグラウンドプロセスへのファイル識別子を指定しましたが，バックグラウンドプロセスが存在しませんでした。	バックグラウンドプロセスを起動しているか，またはすでに終了していないかを見直します。
KNAX6305-E	通常ファイルではありません。	指定のファイルは通常ファイルではありません。	ファイルの指定を見直します。

メッセージ ID	エラーの詳細	原因	対処
KNAX6333-E	通常ファイルではありません。	指定のファイルは通常ファイルではありません。	ファイルの指定を見直します。
KNAX6588-E	Error in signal handler	シグナルハンドラの処理でエラーが発生しました。	実行環境に異常がないか見直します。

12.4.4　ユーザー応答機能で表示されるエラー情報の意味および対処方法

ユーザー応答機能の使用時に，次のメッセージに出力されるエラー情報の意味と，その際の対処を説明します。なお，ユーザー応答機能の情報を adshcollect コマンドで採取する場合は，管理者権限で実行する必要があります。

- KNAX7402-E
- KNAX7460-E
- KNAX7461-E
- KNAX7462-E

(1)　KNAX7402-E メッセージの詳細情報に表示されるエラー情報の意味および対処方法

表 12-8　KNAX7402-E メッセージの詳細情報に表示されるエラー情報の意味および対処方法

エラー番号	意味	対処
1	未実装の API が呼び出されました。	システム管理者に連絡してください。 システム管理者は，「11.　トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
2	共有メモリの参照に失敗しました。 ユーザー応答機能管理デーモン・サービスが起動していないおそれがあります。	【運用者】 ジョブ環境ファイルの設定が正しいか確認してください。SPOOL_DIR パラメーターはジョブ環境ファイルには指定できません。 また，マシンの管理者権限を持つユーザーに連絡し，次の点を確認してください。 • ユーザー応答機能管理デーモン・サービスが起動しているか • 環境ファイルの設定が正しいか 【マシンの管理者権限を持つユーザー】 次の点を確認してください。 • ユーザー応答機能管理デーモン・サービスが起動しているか • 環境ファイルの設定が正しいか SPOOL_DIR パラメーターはシステム環境ファイルだけに指定し，ジョブ環境ファイルには指定できません。 • システム環境ファイルの変更後にユーザー応答機能管理デーモン・サービスを再起動しているか

エラー番号	意味	対処
2	共有メモリの参照に失敗しました。 ユーザー応答機能管理デーモン・サービスが起動していないおそれがあります。	問題が解決しない場合は，システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
3	空きメモリ領域が不足しています。	システム管理者に連絡してください。 システム管理者はメモリを見積もり直す必要があります。
4	不正な引数が渡されました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
5	データの不整合を検出しました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
6	サポートしていない文字エンコーディングが指定されました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
7	アンダーフローが発生しました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
8	オーバーフローが発生しました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
9	API の呼び出し順序が間違っています。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
10	オブジェクトなどの内部状態に不整合が発生しました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
11	リソースへのアクセスが拒否されました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
12	指定されたファイルが存在しません。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
13	ファイルをオープンできません。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
14	ファイルに対するメモリマッピングを作成できません。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。

エラー番号	意味	対処
15	ファイルからの読み込み処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，読み込み権限があることを確認してください。
16	ファイルへの書き込み処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
17	ファイルのシーク処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
18	ファイルへのフラッシュ処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
19	ファイルのリネーム処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
20	ファイルのコピー処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
21	ファイルの削除処理でエラーが発生しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
22	ディレクトリの作成に失敗しました。	環境ファイルのログディレクトリで指定したディレクトリおよびその配下のログファイルに，書き込み権限があることを確認してください。
23	プロセス間のロックに失敗しました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
24	ユーザープログラムによって無効にされた機能を利用しようとしました。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
25	ユーザープログラムによって禁止されているため，上書きできません。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
400	パラメーターが不正です。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
404	adshchmsg コマンドによって応答要求メッセージが削除されたか，または共有メモリを参照できません。	運用者はマシンの管理者権限を持つユーザーに連絡して，次の点を確認してください。 • adshchmsg コマンドの-d オプションで応答要求メッセージを削除したか • ユーザー応答機能管理デーモン・サービスが起動しているか

エラー番号	意味	対処
404	adshchmsg コマンドによって応答要求メッセージが削除されたか，または共有メモリを参照できません。	マシンの管理者権限を持つユーザーは，ユーザー応答機能管理デーモン・サービスが起動していることを確認してください。問題が解決しない場合は，システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
409	同一 PID から応答要求メッセージが出力されました。	マルチスレッドで同時に応答要求メッセージが出力されたことが考えられます。 アプリケーションを見直してください。
503	応答待ちイベントの出力数が上限を超えました。	USERREPLY_WAIT_MAXCOUNT パラメーターの値を見直してください。

(2) KNAX7460-E，KNAX7461-E，KNAX7462-E メッセージの詳細情報に表示されるエラー情報の意味および対処方法

表 12-9 KNAX7460-E，KNAX7461-E，KNAX7462-E メッセージの詳細情報に表示されるエラー情報の意味および対処方法

エラー番号	意味	対処
10	パラメーターが不正です。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
11	関数発行の順序が不正です。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
30	指定属性は登録済みです。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
32	登録できる拡張属性数を超えています。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
33	登録できる拡張属性の合計サイズを超えています。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
40	イベントサービスに接続できません。	自ホストの JP1/Base のイベントサービスの起動を確認してください。
43	入出力エラーです。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。

エラー番号	意味	対処
50	JP1/Base のライブラリが見つかりません。	JP1/Base をインストールして，ジョブを再実行してください。
51	メモリが不足しています。	システム管理者に連絡してください。 システム管理者はメモリを見積もり直す必要があります。
52	ファイルオープン数が限界です。	システム管理者に連絡してください。 システム管理者はオープンできる fd 数を見積もり直す必要があります。
60	JP1 イベントが初期化処理されていません。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。
70	システムエラーです。	システム管理者に連絡してください。 システム管理者は，「11. トラブルシューティング」に従って資料を採取し，トラブルの回避および対処を実施してください。

付録

付録 A　カバレージ情報を取得する対象

カバレージ情報には C0 情報および C1 情報があります。カバレージ情報を取得する対象について説明します。

なお，次の項目については，カバレージ情報を取得しません。

- 条件式
- 算術演算
- 変数

付録 A.1　カバレージ情報を取得するコマンド

次のコマンドを使用した場合，カバレージ情報を取得するかどうかについて説明します。

- シェル標準コマンド
- シェル拡張コマンド
- スクリプト拡張コマンド
- その他のコマンド

(1)　シェル標準コマンド

(a)　特殊組み込みコマンド

表 A-1　カバレージ情報を取得する特殊組み込みコマンド

項目	C0	C1
.（ドット）コマンド	○	×
:（コロン）コマンド	○	×
break コマンド	○	×
continue コマンド	○	×
eval コマンド	○	×
exec コマンド	○	×
exit コマンド	○	×
export コマンド	○	×
readonly コマンド	○	×
return コマンド	○	×

項目	C0	C1
set コマンド	○	×
shift コマンド	○	×
trap コマンド	○	×
typeset コマンド	○	×
unset コマンド	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

(b)　正規組み込みコマンド

表 A-2　カバレージ情報を取得する正規組み込みコマンド

項目	C0	C1
alias コマンド	○	×
builtin コマンド	○	×
cd コマンド	○	×
command コマンド	○	×
echo コマンド	○	×
false コマンド	○	×
getopts コマンド	○	×
kill コマンド	○	×
let コマンド	○	×
print コマンド	○	×
pwd コマンド	○	×
read コマンド	○	×
test コマンド	○	×
times コマンド	○	×
true コマンド	○	×
ulimit コマンド	○	×
umask コマンド	○	×
unalias コマンド	○	×
wait コマンド	○	×

項目	C0	C1
whence コマンド	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

(2)　シェル拡張コマンド

表 A-3　カバレージ情報を取得するシェル拡張コマンド

項目	C0	C1
adshecho コマンド	○	×
adshread コマンド	○	×
adshscripttool コマンド	○	×
adshcmdrc コマンド	○	×
adshjoberr コマンド	○	×
adshvarconv コマンド	○	×
adshmktemp コマンド	○	×
adshparsecsv コマンド	○	×
adshparsejson コマンド	○	×
adshappexec コマンド	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

(3)　スクリプト拡張コマンド

表 A-4　カバレージ情報を取得するスクリプト拡張コマンド

項目	C0	C1
#-adsh_file コマンド	○	×
#-adsh_file_temp コマンド	○	×
#-adsh_job コマンド	○	×
#-adsh_job_stop コマンド	○	×
#-adsh_path_var コマンド	○	×
#-adsh_rc_ignore コマンド	○	×

項目	C0	C1
#-adsh_script コマンド	○	×
#-adsh_spoolfile コマンド	○	×
#-adsh_step_start コマンド	○	○※1
#-adsh_step_error コマンド	○	○※2
#-adsh_step_end コマンド	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

注※1

C1 に表示する情報の詳細については，「(e)　#-adsh_step_start コマンドの場合」を参照してください。

注※2

C1 に表示する情報の詳細については，「(f)　#-adsh_step_error コマンドの場合」を参照してください。

(4)　その他のコマンド

JP1/Advanced Shell 以外のコマンド（OS のコマンド，ユーザーが作成したコマンドなど）を使用した場合に，カバレージ情報を取得するかどうかを次の表に示します。

表 A-5　カバレージ情報を取得するその他のコマンド

項目	C0	C1
その他のコマンド	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

付録 A.2　カバレージ情報を取得する制御文

制御文を使用した場合に，カバレージ情報を取得するかどうかを次の表に示します。

表 A-6　カバレージ情報を取得する制御文

項目	C0	C1
if	×	○
if の条件	○	×
then	×	×

項目	C0	C1
elif	×	○
elif の条件	○	×
else	×	○
fi	×	△
for	×	○
変数	×	×
in	×	×
ワードリスト	×	×
do	×	×
done	×	○
while	×	○
while の条件	○	×
until	×	○
until の条件	○	×
case	×	×
式	×	×
パターン)	×	○
*)	×	○
;;	×	×
esac	×	△

（凡例）

○：カバレージ情報を取得して表示します。

△：次の場合にカバレージ情報を取得して表示します。

fi：else 節を指定していない場合

esac：*パターンを指定していない場合。*パターンとは，case 文でどのパターンにも一致しなかった場合のパターンです。

×：カバレージ情報を取得しません。

付録 A.3　カバレージ情報を取得する関数

関数を呼び出す場合に，カバレージ情報を取得するかどうかを次の表に示します。関数を定義する場合は，カバレージ情報を取得しません。

表 A-7　カバレージ情報を取得する関数の呼び出し

項目	C0	C1
関数名の呼び出し	○	×
function の実行	×	×
関数名の実行	×	×
( )の部分の実行	×	×
{で始まる処理の実行	×	×
コマンドおよび制御文の実行	○	△
}で終わる処理の実行	×	×

（凡例）

○：カバレージ情報を取得して表示します。

△：実行する制御文に C1 情報がある場合は，カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

付録 A.4　カバレージ情報を取得するメタキャラクタ

メタキャラクタの中では，コマンドセパレータの場合だけカバレージ情報を取得します。カバレージ情報を取得するコマンドセパレータを次の表に示します。

表 A-8　カバレージ情報を取得するコマンドセパレータ

項目	C0	C1
cmd_1;cmd_2	○	×
cmd_1&&cmd_2	○	×
cmd_1||cmd_2	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

なお，メタキャラクタを使用した次の機能では，カバレージ情報を取得しません。

- コメント
- 行継続
- 変数置換
- コマンド置換

- ファイル名置換
- リダイレクト
- ヒアドキュメント
- コマンドのグループ化
- その他のメタキャラクタ

付録 A.5　カバレージ情報を取得するシェル変数の動作

次の表で示すように，シェル変数に値を代入する場合，カバレージ情報を取得します。

表 A-9　カバレージ情報を取得するシェル変数の動作

項目	C0	C1
シェル変数=値	○	×

（凡例）

○：カバレージ情報を取得して表示します。

×：カバレージ情報を取得しません。

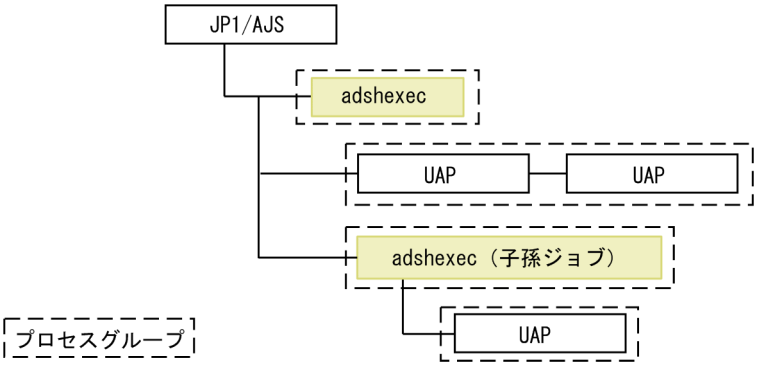
付録 B　JP1/AJS 以外のジョブスケジューラから起動する場合【UNIX 限定】

実行環境で JP1/AJS 以外のジョブスケジューラを使用して JP1/Advanced Shell のバッチジョブを起動する方法について説明します。

なお，JP1/AJS 以外のジョブスケジューラでのバッチジョブ業務の自動化の詳細についてはジョブスケジューラのマニュアルを参照してください。

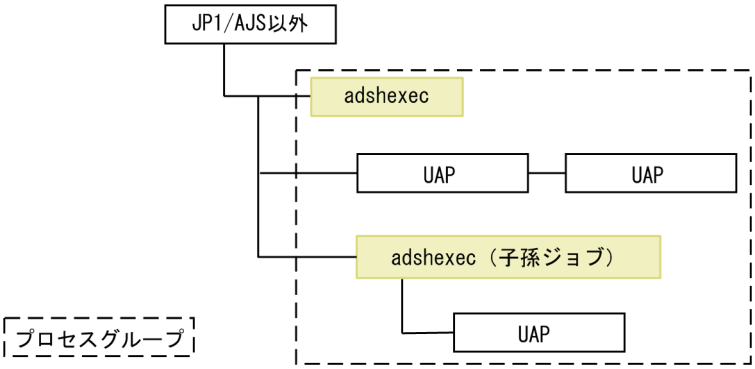
JP1/Advanced Shell は JP1/AJS からバッチジョブを起動すると，adshexec コマンドのプロセス，および子孫プロセスは，異なるプロセスグループで実行します。これは JP1/Advanced Shell の多くの機能が JP1/AJS と連携しているためです。

図 B-1　JP1/AJS から起動した時のプロセスグループ



一方，JP1/AJS 以外のジョブスケジューラから起動した場合は，adshexec コマンドのプロセス，および子孫プロセスは，同じプロセスグループで実行します。

図 B-2　JP1/AJS 以外のジョブスケジューラから起動した時のプロセスグループ



これによって，JP1/AJS 以外のジョブスケジューラから起動した場合も強制終了でジョブを即時終了できます。ただし，JP1/AJS と連携が必要な JP1/Advanced Shell の一部の機能については使用できません。詳細は「付録 B.3　JP1/AJS 以外のジョブスケジューラから起動する際の注意事項」を参照してください。

付録 B.1　JP1/AJS 以外のジョブスケジューラから起動するための準備

JP1/AJS 以外のジョブスケジューラを使用して JP1/Advanced Shell のバッチジョブを起動する場合，次のことを実施してください。

1. 使用するジョブスケジューラが JP1/AJS 以外であることを定義するため，環境ファイルに環境設定パラメーター SCHEDULER_SELECT を「OTHER」で定義してください。

```
#-adsh_conf SCHEDULER_SELECT OTHER
```

2. 環境変数 AJS_BJEX_STOP は，JP1/AJS と JP1/Advanced Shell が連携するために使用する環境変数です。そのため，環境変数 AJS_BJEX_STOP を定義している場合は定義を無効にしてください。

付録 B.2　SCHEDULER_SELECT パラメーター（使用するジョブスケジューラを選択する）

形式

```
#-adsh_conf SCHEDULER_SELECT {AJS | OTHER}
```

機能

使用するジョブスケジューラを選択します。

ジョブコントローラは，JP1/AJS 以外のジョブスケジューラからもジョブを実行できます。ただし，JP1/Advanced Shell が提供する一部の機能を使用できません。使用できない機能，および注意事項については，「付録 B.3　JP1/AJS 以外のジョブスケジューラから起動する際の注意事項」を参照してください。

JP1/AJS 以外のジョブスケジューラを使用してジョブを実行する場合は，オペランドに OTHER を指定してください。JP1/AJS をジョブスケジューラとして使用する場合，またはジョブスケジューラを使用しない場合はオペランドに AJS を指定してください。

なお，このパラメーターの指定を省略すると，ジョブコントローラは AJS が指定されたと解釈して動作します。

オペランド

AJS

次のどれかの場合は，AJS を指定してください。

- ジョブスケジューラとして JP1/AJS を使用する。
- ジョブスケジューラを使用しない。

OTHER

JP1/AJS 以外のジョブスケジューラを使用する場合は，OTHER を指定してください。

注意事項

- オペランドに OTHER を指定しても，環境変数 AJS_BJEX_STOP の値が TERM の場合，JP1/AJS を使用すると判断します。オペランドに OTHER を指定する場合は，環境変数 AJS_BJEX_STOP の定義を無効にしてください。
- ルートジョブと子孫ジョブで同じオペランドを指定してください。
- システム環境ファイルとジョブ環境ファイルの両方にこのパラメーターが定義されていた場合，ジョブ環境ファイルでの定義が有効になります。
- このパラメーターを同一の環境ファイルで同一のホストに対して複数定義した場合，パラメーターエラーとなります。

付録 B.3　JP1/AJS 以外のジョブスケジューラから起動する際の注意事項

JP1/AJS 以外のジョブスケジューラを使用して JP1/Advanced Shell のバッチジョブを起動する場合，次の点に注意してください。

なお，JP1/AJS 以外のジョブスケジューラからジョブを強制終了すると，SIGKILL シグナルが送信されると想定しています。強制終了時に送信されるシグナルについてはジョブスケジューラのマニュアルを参照してください。

- 環境設定パラメーター TRAP_ACTION_SIGTERM を定義しても，強制終了時のユーザー固有の後処理は実行できません。
- JP1/Advanced Shell はジョブ実行のために一時ファイルを作成する場合があります。一時ファイルを格納するディレクトリパスは環境設定パラメーター TEMP_FILE_DIR に指定します。通常，一時ファイルは削除されますが，実行中のジョブを強制終了すると削除されず残ることがあります。このような場合は手動で削除してください。
- 実行中のジョブを強制終了すると，#-adsh_file コマンド，#-adsh_file_temp コマンド，#-adsh_spoolfile コマンド，adshfile コマンドで作成したファイルが残る場合があります。このような場合は手動で削除してください。
- adshread コマンドが応答要求メッセージの応答待ちの状態で，JP1/AJS 以外のジョブスケジューラから強制終了した場合，共有メモリ上に応答要求メッセージの情報が残り，JP1/IM - View に応答待ちイベントが滞留したままになることがあります。その場合，adshchmsg コマンドの-d オプションで応答要求メッセージの応答待ち状態をキャンセルするか，ユーザー応答機能管理デーモン・サービスを再起動してください。
- adshjava コマンドで Java バッチアプリケーションを実行した状態で，JP1/AJS 以外のジョブスケジューラから強制終了した場合，Java バッチアプリケーションは強制終了されないで実行し続ける場合があります。

- 次の表に示す稼働実績情報の項目は採取されません。また，adshevtout コマンドに次の表に示すオプションを指定しても稼働実績情報を出力できません。

表 B-1　採取されない稼働実績情報の項目と対応する adshevtout コマンドのオプション

項目名	対応する adshevtout のオプション	内容
Jp1ajsService	-c	JP1/AJS のスケジューラーサービス名
Jp1ajsJobName	-g	JP1/AJS のジョブ名
Jp1ajsExecId	-k	JP1/AJS のジョブの実行 ID
Jp1ajsJobId	-n	JP1/AJS のジョブ番号
Jp1ajsRootJobnet	-r	JP1/AJS のルートジョブネット名

付録 C　各バージョンの変更内容

各バージョンの変更内容をマニュアルの版ごとに説明します。

付録 C.1　11-01 での変更内容

- 次に示す UNIX 互換コマンドを使用できるようにした。
  - gunzip
  - gzip
  - printf
- JOBLOG_SUPPRESS_MSG パラメーターでジョブ実行ログへの出力を抑止できるメッセージ ID を追加した。
- UNIX 互換コマンドのtar コマンドに-z オプションを追加した。
- メッセージKNAX7091-W を追加した。

付録 C.2　11-00 での変更内容

- tar コマンドを追加した。
- cp コマンドおよび mv コマンドにロングオプションの指定を追加した。
- date コマンドに書式指定コード%を追加した。
- パス変換機能に関する説明を追加した。
- JP1/AJS 以外のジョブスケジューラを使用して JP1/Advanced Shell のバッチジョブを起動する方法を追加した。
- 別プロセス化を伴う構文の説明を追加した。
- 二次元配列について説明を追加した。
- 外部コマンドの起動方法の定義を追加した。
- export パラメーターの環境変数値で指定できる値の上限を変更した。
- Windows シンボリックリンクの説明を追加した。
- UNIX ジョブのコマンド文でジョブ定義できるようにした。
- シェルスクリプト開発部品を提供した。
- JP1/AJS からの GUI プログラム実行機能について説明した。
- ジョブステップのエラー判定について説明した。
- インタフェースを変更した。
- シェルスクリプト入力支援を追加した。

- メッセージの出力先を追加した。
- メッセージ本文を変更した。

  KNAX7073-I
- メッセージを追加した。

  KNAX2500-E，KNAX2501-E，KNAX6118-I，KNAX6119-I，KNAX6128-I，KNAX6129-I，KNAX6134-E，KNAX6135-E，KNAX6136-E，KNAX6137-E，KNAX6138-E，KNAX6139-E，KNAX6140-E，KNAX6150-E，KNAX6151-E，KNAX6152-E，KNAX6153-E，KNAX6190-E，KNAX6191-E，KNAX6192-E，KNAX6193-E，KNAX6194-E，KNAX6340-E，KNAX6341-E，KNAX6342-E，KNAX6580-E，KNAX6581-E，KNAX6595-E，KNAX6996-I KNAX7200-I，KNAX7201-I，KNAX7203-W，KNAX7204-E，KNAX7205-E，KNAX7210-E，KNAX7211-I，KNAX7212-I，KNAX7213-I，KNAX7215-E，KNAX7216-E，KNAX7217-E，KNAX7221-E，KNAX7222-E，KNAX7223-E，KNAX7225-E，KNAX7250-I，KNAX7251-I，KNAX7254-E，KNAX7255-I，KNAX7256-E，KNAX7258-E，KNAX7259-W，KNAX7260-W，KNAX7261-I，KNAX7262-I，KNAX7263-E，KNAX7264-I，KNAX7268-I，KNAX7269-E，KNAX7270-E，KNAX7271-I
- メッセージの説明を追加した。

  KNAX6004-E，KNAX6016-E，KNAX6054-E，KNAX6072-E

付録 C.3　10-51 での変更内容

- ジョブ強制終了時にユーザー固有の後処理を実行できるようにした。
- UNIX 互換コマンドで who コマンド（スクリプト形式）を使用できるようにした。また，chmod コマンドと su コマンドのサンプルスクリプトファイルのファイル名を変更した【Windows 限定】。
- 次に示す UNIX 互換コマンドを使用できるようにした。
  - dirname
  - expand
  - getopt
  - stat

  また，次に示す UNIX 互換コマンドにオプションを追加した。
  - cut
  - date
  - diff
  - expr
  - ls
- ジョブ実行時にスプールジョブディレクトリが作成されない設定ができるようにした（スプールジョブ作成抑止機能）。これに伴って次の環境設定パラメーターを追加した。

- SPOOLJOB_CREATE パラメーター
- AIX および HP-UX で，文字コード UTF-8 を使用できるようにした。
- 次に示す UNIX 互換コマンドに，ロングオプション形式のオプションを使用できるようにした。
  - cut
  - date
  - diff
  - ls
- CHILDJOB_SHEBANG パラメーターのデフォルト定義に合致するジョブ定義スクリプトを子孫ジョブとして実行できるようにした。これに関連して次の機能を追加した。
  - JOBLOG_SUPPRESS_MSG パラメーターの抑止対象メッセージに，子孫ジョブ関連のメッセージを追加した。
  - サンプルスクリプトファイルの実行方法を関数形式から子孫ジョブ形式へ変更した。
- 出力するメッセージの種類を最小限に抑止できる出力モード（最小出力モード）を追加した。これに伴って次の環境設定パラメーターにオペランドを追加した。
  - OUTPUT_MODE_CHILD パラメーター
  - OUTPUT_MODE_ROOT パラメーター

  また，次のコマンドにオプションを追加した。
  - adshexec コマンド
  - adshscripttool コマンド
- 配列の添え字の上限数を 1,023 から 65,535 へ変更した。
- ジョブを実行する adshexec コマンドに，シェルを直接記述できるようにした。
- 【Windows 限定】環境変数名に小文字を使用できるようにした。これに伴って次の環境設定パラメーターを追加した。
  - VAR_ENV_NAME_LOWERCASE パラメーター
- 変数置換で部分文字列展開できるようにした。
- 複数の配列要素を一度に作成する指定方法を追加した。
- ジョブが実行中の関数の情報を配列（関数情報配列）へ格納できるようにした。これに伴って次の環境設定パラメーターを追加した。
  - VAR_SHELL_FUNCINFO パラメーター
- 変数の値の文字列長および配列の要素数への変数置換の書式で，置換される変数値の長さの単位を指定できるようにした。これに伴って次の環境設定パラメーターを追加した。
  - VAR_SHELL_GETLENGTH パラメーター
- adshscripttool コマンドの-r オプションにジョブ内容を直接指定した場合に，ジョブ定義スクリプトファイル名として出力される内容を変更した。

- メッセージを追加した。

  KNAX0235-E，KNAX0474-E，KNAX1880-E，KNAX6058-E，KNAX6072-E，KNAX6097-E，KNAX6385-E，KNAX6718-I，KNAX7073-I，KNAX7128-E

- メッセージの説明を変更した。

  KNAX0411-E，KNAX0441-E，KNAX0445-E，KNAX0449-E，KNAX1873-E，KNAX5407-E，KNAX6007-E，KNAX6008-E，KNAX6022-E，KNAX6226-E，KNAX6241-E，KNAX6382-I，KNAX6710-I，KNAX6997-E，KNAX7450-I，KNAX7451-I，KNAX7901-I，KNAX7902-I

- メッセージテキストを変更した。

  KNAX9000-E，KNAX9001-E

- 用語解説に「トラップアクション」の説明を追加した。

付録 C.4　10-50 での変更内容

- 適用 OS として Solaris 11 を追加した。
- 次に示す UNIX 互換コマンドを使用できるようにした。
  - chmod
  - su

  これに伴って，ジョブ定義スクリプトの作成を支援する次のシェル拡張コマンドを追加した【Windows 限定】。

  - adshscripttool
- ファイルの動的管理をサポートした。

  これに伴って次のコマンドを追加した。

  - adshfile
- 次に示す UNIX 互換コマンドを使用できるようにした。
  - basename
  - egrep
  - paste
  - touch
  - which

  また，egrep コマンドの追加に伴って，grep コマンドに-h オプションを追加した。

- 2GB を超えるファイルを一部使用できるようにした。
- Windows 版でファイル名やパス名を UNC 形式で指定できるようにした（一部のコマンドおよびファイルを除く）。
- 環境変数一覧の記載を追加した。

- ジョブ定義スクリプトのパス変換に次の機能を追加した。
  - 「''」で囲まれていない個所全体の変換を選択できるようにした【Windows 限定】。
  - 相対パスの変換ができるようにした。

  これに伴って次の環境設定パラメーターを追加した【Windows 限定】。
  - PATH_CONV_RULE
- 次のシェル変数を追加し，コマンドの格納フォルダを変数指定できるようにした。
  - ADSH_DIR_BIN
  - ADSH_DIR_CMD
- スプールジョブディレクトリへの標準出力と標準エラー出力への出力を抑止できるようにした（簡潔出力モード）。

  これに伴って次の環境設定パラメーターを追加した。
  - OUTPUT_MODE_CHILD
  - OUTPUT_MODE_ROOT

  また，adshexec コマンドにオプションを追加した。
- 子孫ジョブのジョブ実行ログをルートジョブのジョブ実行ログへマージできるようにした。

  これに伴って次の環境設定パラメーターを追加した。
  - SPOOLJOB_CHILDJOB
- UNIX 版でパイプ「|」で接続したコマンドのうち，最後のコマンドの実行を別プロセスとカレントプロセスのどちらにするか選択できるようにした。

  これに伴って次の環境設定パラメーターを追加した。
  - PIPE_CMD_LAST
- シェルオプション xtrace の有効化を adshexec コマンドと JP1/Advanced Shell エディタでも設定できるようにした。

  これによって，ジョブ定義スクリプトを修正しなくても実行コマンドおよびその引数を標準エラー出力へ出力できるようにした。
- C1 実行率が 100%にならない場合の対処として，次の方法でデバッグ時にエラーをシミュレートできるようにした。
  - Windows 版の JP1/Advanced Shell エディタにメニュー［エラー注入モード］を追加した。
  - UNIX 版に joberrmode コマンドと info status コマンドを追加した。
- エスケープ文字として ASCII コードの 16 進数表記を使用できるようにした。

  また，echo コマンドでエスケープ文字の解釈を指定しなかった場合の動作を環境設定パラメーターで設定できるようにした。

  これに伴って次の環境設定パラメーターを追加した。
  - ESCAPE_SEQ_ECHO_DEFAULT

- ESCAPE_SEQ_ECHO_HEX
- 算術演算子に累乗演算子「**」を追加した。
- コマンドの終了コードに対し，正常終了となるしきい値を設定できるようにした。
  これに伴って次の環境設定パラメーターを追加した。
  - CMDRC_THRESHOLD_DEFINE
  - CMDRC_THRESHOLD_USE_PRESET
- uname コマンドで，Administrators 権限がなくても Windows OS と判断できるようにした。これに伴って，-w オプションを追加した【Windows 限定】。
- メッセージの出力先を変更した。
  KNAX6590-E，KNAX7400-E，KNAX7402-E，KNAX7403-E，KNAX7404-E，KNAX7405-E，KNAX7408-E
- メッセージを追加した。
  KNAX0708-E，KNAX0725-E～KNAX0728-E，KNAX1871-E～KNAX1873-E，KNAX1875-E，KNAX1877-E～KNAX1879-E，KNAX1890-I～KNAX1893-W，KNAX6056-W，KNAX6068-E，KNAX6384-E，KNAX6572-I，KNAX6587-E，KNAX6750-E～KNAX6753-E，KNAX6759-E，KNAX7126-I，KNAX7127-E
- メッセージの説明を変更した。
  KNAX0336-E，KNAX0724-I，KNAX6048-E，KNAX6409-I，KNAX6410-I，KNAX6584-I，KNAX6830-I，KNAX7901-I

JP1/Advanced Shell 10-01，および JP1/Advanced Shell - Developer 10-01 に対応したマニュアルの変更内容です。

- ジョブ続行不可でエラー終了するケースに対し，終了コードを定義できるようにした。
  これに伴って次の環境変数を追加した。
  - ADSH_JOBRC_FATAL
- UNIX でスプールジョブのディレクトリまたはファイルのパーミッションを変更できるようにした。
  これに伴って次の環境設定パラメーターを追加した。
  - PERMISSION_SPOOLJOB_DIR
  - PERMISSION_SPOOLJOB_FILE
- ジョブ実行ログの標準エラー出力への出力内容を限定できる機能を追加した。
  これに伴って次の環境設定パラメーターを追加した。
  - JOBEXECLOG_PRINT
- メッセージの出力先を変更した。
  KNAX0236-E，KNAX0238-E
- メッセージを追加した。

KNAX0237-E，KNAX0239-E，KNAX0240-I，KNAX0459-E，KNAX6383-E

付録 C.5　10-00-01 での変更内容

- 適用 OS として Windows Server 2012 および Windows 8 を追加した。
- クラスタシステムでの運用に関する記述を追加した。

付録 C.6　10-00 での変更内容

- 適用 OS の追加に伴い，HP-UX 環境，Solaris 環境の説明を追加した。
- 適用 OS の変更に伴い，AIX 環境の説明を変更した。
- JP1/IM と連携して，事象通知メッセージまたは応答要求メッセージを JP1 イベントとして発行し，その応答を受け取る機能を追加した（ユーザー応答機能）。これに伴い，論理ホストに関する記述を追加した。

  また，次のダイアログボックスに設定項目を追加した。

  - ［実行定義］ダイアログボックス
  - ［実行環境の設定］ダイアログボックス

  また，次のパラメーターおよびコマンドを追加した。

  - HOSTNAME_JP1IM_MANAGER パラメーター
  - JOBLOG_SUPPRESS_MSG パラメーター
  - USERREPLY_DEBUG_DESTINATION パラメーター
  - USERREPLY_JP1EVENT_INTERVAL パラメーター
  - USERREPLY_WAIT_MAXCOUNT パラメーター
  - adshchmsg コマンド
  - adshlsmsg コマンド
  - adshmdctl コマンド
  - adshmsvce コマンド
  - adshmsvcd コマンド
  - adshecho コマンド
  - adshread コマンド
  - lhost_start パラメーター
  - lhost_end パラメーター
  - phost_start パラメーター

- phost_end パラメーター

- システム環境ファイルを追加した。これに伴って，既存の「環境ファイル」の表記を「ジョブ環境ファイル」へ変更した。

  また，「環境ファイル」はシステム環境ファイルとジョブ環境ファイルの総称とした。

- 環境ファイルに設定する export コマンドの表記を「export パラメーター」へ変更した。

- ジョブ定義スクリプトファイルに，子孫ジョブとして実行する拡張子や，パスの読み替えを定義する機能を追加した。これに伴って次のパラメーターを追加した。

  - CHILDJOB_EXT パラメーター
  - CHILDJOB_PGM パラメーター

- ジョブ定義スクリプト稼働実績情報を出力できるようにした（JP1/Advanced Shell - Developer は対象外）。これに伴って次のパラメーターおよびコマンドを追加した。

  - EVENT_COLLECT パラメーター
  - adshevtout コマンド

  また，この機能の追加に伴い，adshhk コマンドのスプールジョブの記述を変更した。

- メッセージを追加した。

  KNAX0220-E，KNAX0410-E，KNAX0458-E，KNAX0471-E，KNAX0472-E，KNAX0473-W，KNAX3000-I，KNAX3001-I，KNAX3002-E，KNAX3003-E，KNAX3006-I，KNAX3008-W，KNAX3009-E，KNAX3020-E，KNAX3023-E，KNAX3024-E，KNAX3025-E，KNAX3026-E，KNAX3027-E，KNAX3029-E，KNAX3261-I，KNAX3400-I，KNAX3402-E，KNAX3508-I，KNAX3522-E，KNAX3542-W，KNAX3700-I，KNAX3701-I，KNAX3703-E，KNAX3709-E，KNAX3710-I，KNAX3711-I，KNAX3799-I，KNAX3998-E，KNAX3999-E，KNAX4425-E，KNAX5300-I，KNAX5301-E，KNAX5305-E，KNAX5308-E，KNAX5309-E，KNAX5323-E，KNAX5340-E，KNAX5350-E，KNAX5360-E，KNAX5361-E，KNAX5362-E，KNAX5371-E，KNAX5372-E，KNAX5380-I，KNAX5381-I，KNAX5396-I，KNAX5397-I，KNAX5398-E，KNAX5399-E，KNAX5407-E，KNAX5409-E，KNAX5410-E，KNAX5423-E，KNAX5424-E，KNAX5425-E，KNAX5426-E，KNAX5429-E，KNAX5440-E，KNAX5498-E，KNAX5499-E，KNAX6045-E，KNAX6046-E，KNAX6100-E，KNAX6110-I，KNAX6111-I，KNAX6112-I，KNAX6113-I，KNAX6114-I，KNAX6115-I，KNAX6116-I，KNAX6117-I，KNAX6120-I，KNAX6121-I，KNAX6122-I，KNAX6123-I，KNAX6124-I，KNAX6125-I，KNAX6126-I，KNAX6127-I，KNAX6130-E，KNAX6180-E，KNAX6181-E，KNAX6182-E，KNAX6183-E，KNAX6189-I，KNAX6290-E，KNAX6291-E，KNAX6292-E，KNAX6293-E，KNAX6294-E，KNAX6295-E，KNAX6296-E，KNAX6297-E，KNAX6298-E，KNAX6600-E，KNAX6601-E，KNAX6602-E，KNAX6603-E，KNAX6604-E，KNAX6605-E，KNAX6610-E，KNAX6611-E，KNAX6612-E，KNAX6613-I，KNAX6614-I，KNAX6615-E，KNAX6616-E，KNAX6632-E，KNAX6633-E，KNAX6634-E，KNAX6635-E，KNAX6636-E，KNAX6640-I，KNAX6644-E，KNAX6645-W，KNAX6646-E，KNAX6830-I，KNAX6831-I，KNAX6832-I，KNAX7400-E，KNAX7402-E，KNAX7403-E，KNAX7404-E，KNAX7405-E，KNAX7408-E，KNAX7420-E，KNAX7450-I，KNAX7451-I，KNAX7460-E，KNAX7461-E，KNAX7462-E，KNAX7464-E，KNAX7465-W，KNAX7470-I，KNAX7500-I，KNAX7501-I，KNAX7502-E，KNAX7503-E，

KNAX7508-I，KNAX7509-I，KNAX7550-I，KNAX7551-E，KNAX7552-E，KNAX7553-E，KNAX7554-E，KNAX7555-E，KNAX7556-E，KNAX7560-I，KNAX7561-E，KNAX7582-E，KNAX7902-I

- メッセージを削除した。

  KNAX6060-E，KNAX6520-I，KNAX6523-I，KNAX6550-I，KNAX6553-I，KNAX6716-W，KNAX6717-I

- メッセージの説明を変更した。

  KNAX0020-E，KNAX0098-I，KNAX0406-E，KNAX0433-E，KNAX0700-E，KNAX0701-E，KNAX0702-E，KNAX0706-E，KNAX0720-E，KNAX0721-E，KNAX0722-E，KNAX0723-E，KNAX0800-E，KNAX0801-E，KNAX0802-E，KNAX0803-E，KNAX2202-E，KNAX2204-E，KNAX2205-E，KNAX2206-E，KNAX2207-E，KNAX2208-E，KNAX2209-E，KNAX2213-E，KNAX6003-E，KNAX6007-E，KNAX6008-E，KNAX6009-E，KNAX6035-E，KNAX6044-E，KNAX6054-E，KNAX6062-E，KNAX6209-W，KNAX6232-E，KNAX6323-E，KNAX6578-I，KNAX6803-I，KNAX6804-I，KNAX6810-E，KNAX6811-E，KNAX6812-E，KNAX6814-E，KNAX6815-E，KNAX7099-E，KNAX7101-E，KNAX7104-E，KNAX7113-E，KNAX7114-E，KNAX7115-E，KNAX7121-E，KNAX7122-E，KNAX7123-E，KNAX7802-E，KNAX7901-I，KNAX7999-I

- マニュアルの章を記載内容で分類し，編タイトルを付加した。

  追加した編タイトルと，その編に含まれる章は次のとおり。

編タイトル	編に含まれる章
第 1 編　概要編	1 章
第 2 編　構築編	2 章
第 3 編　運用編	3 章～6 章
第 4 編　リファレンス編	7 章～9 章
第 5 編　トラブルシューティング編	10 章～11 章

また，マニュアルの章タイトルを次のように変更した。

旧版（3020-3-S35-30）の章タイトル	新版（3021-3-133）の章タイトル
4.　エディタの操作	4.　ジョブ定義スクリプトの作成（エディタを使用する場合）【Windows 限定】
5.　ジョブ定義スクリプトの文法	5.　ジョブ定義スクリプトの作成（シェルで記述する場合）
7.　環境ファイルで設定するパラメーターとコマンド	7.　環境ファイルで設定するパラメーター

付録 C.7　09-51-01 での変更内容

- Windows および Linux で，UNIX 環境のファイルパス「/dev/null」を Windows 環境の「nul」へ変換する機能の追加に伴い，次に示す説明を追加・変更した。

- 環境設定パラメーター PATH_CONV_ACCESS を追加した（前版で記載していたパラメーター ACCESS_PATH_CONV の名称を変更）。
- 環境設定パラメーター COMMAND_CONV_ARG を追加した（前版で記載していたパラメーター COMMAND_ARG_CONV の名称を変更）。
- Windows 環境および Linux 環境で環境設定パラメーター CHILDJOB_SHEBANG を使用できるようにした。

- Windows および Linux で，ジョブ定義スクリプトの標準入力と標準出力ができるようにした。これに伴って次の説明を追加・変更した。
  - 子孫ジョブの用語定義および動作に関する記述を修正した。
  - OUTPUT_STDOUT パラメーターを追加した。
  - adshexec コマンドに-s オプションを追加し，ルートジョブの標準出力の出力先を指定できるようにした。
  - adshhk コマンドに終了コードを追加した。
- 次に示す機能は提供しないため，記述を削除した。
  - CHILDJOB_EXT パラメーター
  - EXEC_FORMAT_EXT パラメーター
- メッセージを追加した。

  KNAX0308-E，KNAX6571-I，KNAX6578-I，KNAX6594-E，KNAX6805-I，KNAX6806-I，KNAX7901-I
- メッセージの説明を変更した。

  KNAX0098-I，KNAX4419-E，KNAX4427-W，KNAX6053-E，KNAX6054-E，KNAX6059-E，KNAX6380-I，KNAX6803-I，KNAX6804-I，KNAX6814-E，KNAX6815-E，KNAX7999-I

付録 C.8　09-51 での変更内容

- 業務への応用例の説明を追加した。
- ジョブ定義スクリプトを Windows と UNIX で使用できるようにするための次の機能の説明を追加，変更した。
  - ファイルの入出力時にファイルパスを変換する
  - コマンド実行時に引数を変換する

  また，上記の機能の追加に伴って次の環境設定パラメーターを追加した。
  - ACCESS_PATH_CONV パラメーター
  - COMMAND_ARG_CONV パラメーター
  - EXEC_FORMAT_EXT パラメーター

- 子孫ジョブを起動できるようにするための説明を追加した。
  また，子孫ジョブのサポートに伴って次の環境設定パラメーターを追加した。
  - CHILDJOB_EXT パラメーター
  - CHILDJOB_SHEBANG パラメーター
- デバッグ中にエディタからカバレージ情報を表示できるようにした。
- 次の UNIX 互換コマンドを追加した。
  awk，cut，diff，expr，find，head，sed，sort，split，tail，uniq，wc
- AIX 環境で JP1/Advanced Shell が動作するようになり，次の説明を追加，変更した。
  - 実行環境の前提プログラムの説明
  - LANG 環境変数の説明
  - CD-ROM 媒体を使ったインストール
  - シェル変数 ENV の説明
  - シェルの設定の説明
  - 異なるプラットフォーム間でのカバレージ情報の相互運用
  - シグナル受信時の動作
  - signal コマンドの注意事項
  - trap コマンドの signal の説明
  - 用語解説の.env ファイルの説明
- 前提条件に次の説明を追加した。
  - JP1/Advanced Shell を使用するときのエンコーディング
  - ローカルタイムの設定
- ジョブコントローラ起動時にシェル変数 ENV を読み込む機能を追加した。また，このサポートに伴って次の環境設定パラメーターを追加した。
  - KSH_ENV_READ パラメーター
- ジョブ定義スクリプトの文法で次の説明および使用例を追加，変更した。
  - 変数の値の参照，配列，関数，メタキャラクタ，変数置換，ファイル名置換，算術展開，リダイレクト，パイプ，別プロセスでの実行，パターンマッチング
- コマンドの終了コードについての説明を追加した。
- ジョブ実行中にエラーが発生した場合の動作についての説明を追加した。
- ジョブ定義スクリプトでの注意事項を追加した。
- デバッグ時のコマンドで使用例を追加，変更した。
  set，cd

- 次の特殊組み込みコマンドおよびスクリプト予約語コマンドで注意事項，および使用例を追加，変更した。

  .（ドット）コマンド，:（コロン）コマンド，break コマンド，continue コマンド，eval コマンド，exec コマンド，exit コマンド，export コマンド，readonly コマンド，return コマンド，set コマンド，shift コマンド，trap コマンド，typeset コマンド，unset コマンド，time コマンド
- 次の正規組み込みコマンドで注意事項，および使用例を追加，変更した。

  alias コマンド，builtin コマンド，cd コマンド，command コマンド，echo コマンド，false コマンド，getopts コマンド，kill コマンド，let コマンド，print コマンド，pwd コマンド，read コマンド，test コマンド，times コマンド，true コマンド，ulimit コマンド，umask コマンド，unalias コマンド，wait コマンド，whence コマンド
- 標準エラー出力に出力するジョブ実行ログの出力形式の説明を追加，変更した。
- メッセージを追加した。

  KNAX4427-W，KNAX6043-W，KNAX6290-E～KNAX6297-E，KNAX6513-W，KNAX6514-W，KNAX6586-E，KNAX6591-E，KNAX6592-E，KNAX6716-W，KNAX6717-I，KNAX6803-I，KNAX6804-I，KNAX6814-E，KNAX6815-E，KNAX7897-E
- JP1/Advanced Shell が固有に出力するエラーの詳細を追加した。
- マニュアルの構成を次のように変更した。

変更前の章番号（3020-3-S35-10）	変更後の章番号（3020-3-S35-20）
4.　ジョブ定義スクリプトの作成	4.　エディタの操作
	5.　ジョブ定義スクリプトの文法

付録 C.9　9-50-01 での変更内容

- カバレージ情報を採取する機能の説明を追加，変更した。
- 共通アプリケーションフォルダにログフォルダを追加した。
- ハードリンク，シンボリックリンクおよびジャンクションについての注意事項を追加した。
- メタキャラクタについての注釈および注意事項を追加した。
- 次の説明を追加した。
  - 環境ファイルで TRACE_FILE_CNT と TRACE_FILE_SIZE を変更した場合
  - トレースファイルの面数およびファイルサイズを小さくする場合
- SCRIPT の説明に，出力される内容を追加した。
- 「デバッグの停止」を「スクリプトの停止」に変更した。
- 予約語の説明を変更した。
- 入力行の上限および入力文字数の上限について追加した。

- 関数の名称がほかの関数と重複していた場合について追加した。
- 入力行の上限についての説明を追加した。
- test コマンドではワイルドカードを使用できないことを追記した。
- fd の指定についての注釈を変更した。
- ジョブステップ終了コードについての説明を追加した。
- スクリプト拡張コマンドの終了コードおよび終了コードの説明を変更した。
- コマンド実行結果の出力に関する注意事項を変更した。
- E-Time についての注意事項を追加した。
- ブレークポイントとウォッチポイントの上限についての注意事項を変更した。
- ファイルのパス名の記述を追加した。
- コマンドの説明を変更した。

  .（ドット）コマンド，:（コロン）コマンド，builtin コマンド，command コマンド，eval コマンド，exec コマンド，exit コマンド，false コマンド，kill コマンド，let コマンド，read コマンド，return コマンド，test コマンド，true コマンド，unset コマンド，wait コマンド

  #-adsh_path_var コマンド，#-adsh_script コマンド，#-adsh_step_start コマンド，#-adsh_step_error コマンド，#-adsh_step_end コマンド，time コマンド
- カバレージ情報のメッセージを追加，変更した。

  KNAX6200-I，KNAX6201-E，KNAX6202-E～KNAX6208-E，KNAX6209-W，KNAX6210-E～KNAX6215-E，KNAX6219-E，KNAX6220-I～KNAX6222-I，KNAX6223-E～KNAX6241-E，KNAX6242-I～KNAX6243-I
- メッセージを変更した。

  KNAX2201-E，KNAX6508-I，KNAX6523-I，KNAX6553-I，KNAX6584-I

付録 D　用語解説

このマニュアルで使用する用語について解説します。

（記号）

.env ファイル

環境変数ENV にファイルパスを設定し，シェル起動時に読み込むファイルです。ファイルを読み込むかどうかは，環境設定パラメーターKSH_ENV_READ で指定することができます。

（英字）

export パラメーター

環境ファイルに設定するパラメーターのうち，コマンド起動時に環境変数を設定するために指定するパラメーターです。

GUI アプリケーション実行プログラム

アプリケーション実行エージェント機能を使用する場合に，アプリケーション実行エージェントプログラムに連絡をするプログラムです。JP1/AJS から実行アプリケーションを実行する場合には PC ジョブの実行ファイル名，またはカスタムジョブの実行プログラムに定義します。

JP1/Advanced Shell

バッチジョブのためのジョブ定義スクリプトを作成・実行するための製品です。JP1/Advanced Shell は，JP1/Advanced Shell と JP1/Advanced Shell - Developer とに分けられます。JP1/Advanced Shell では，バッチジョブのためのジョブ定義スクリプトを実行でき，狭義には JP1/Advanced Shell を実行環境と呼びます。同じジョブ定義スクリプトのバッチジョブを Windows および UNIX の両方で実行できます。

JP1/Advanced Shell - Custom Job

運用管理端末で JP1/Advanced Shell の定義を行うためのプログラムのことです。

JP1/Advanced Shell - Developer

バッチジョブのためのジョブ定義スクリプトを開発するための製品です。ジョブ定義スクリプトを開発するため，開発環境と呼ぶこともあります。

JP1/AJS3

JP1/Automatic Job Management System 3 の略で，JP1/AJS2 の後継製品です。JP1/Advanced Shell は，JP1/AJS3 と連携することで，複数の PC 間での分散処理が実現できます。

UNIX 互換コマンド

JP1/Advanced Shell では，UNIX でよく使用されるコマンドの一部を使用できます。Windows 環境でも使用でき，UNIX から Windows への移行性を向上できます。ls コマンドなどがあります。

（ア行）

アプリケーション実行エージェント機能

ユーザー指定のアプリケーションをユーザーのログオン空間で実行する機能です。

アプリケーション実行エージェントプログラム

ユーザーごとに動作し，実行アプリケーションを実行する目的で常駐させておくプログラムです。アプリケーション実行エージェント機能を使用する場合には，スタートアップに登録しておくことを推奨します。

一時ファイル

ジョブ実行時に一時的に使用するファイルです。ジョブまたはジョブステップによって作成され，ジョブ終了時には自動的に削除されます。#-adsh_file_temp コマンドで定義できます。

ウォッチポイント

ある変数や式の値が変化したときにジョブ定義スクリプトを停止させる，特別なブレークポイントです。ウォッチポイントは，ほかのブレークポイントと同じように管理できます。

エディタ

開発環境に付属するさまざまな機能を利用して，効率良くジョブ定義スクリプトを作成できます。

エラー注入モード

デバッグ実行時にエラーの発生をシミュレーションするモードです。

エラー注入モードの有効/無効を切り替えるには，UNIX ではjoberrmode コマンドを実行します。Windows では JP1/Advanced Shell エディタでメニュー［エラー注入モード］を選択します。

応答待ちイベント

応答要求メッセージを通知する JP1 イベントのことです。

応答要求メッセージ

運用者からの応答を求めるメッセージのことです。

オプション

コンピュータの入力装置から入力する指示に対して，選択的な機能を付け加えます。このことをオプションといいます。

JP1/Advanced Shell では，1 個のハイフン（-）に続く 1 文字のコマンド引数をショートオプション，連続する 2 個のハイフン（--）に続くコマンド引数をロングオプションと呼んでいます。

オプションの右側に指定する引数のことをオプションの値と呼びます。

オペランド

コマンドラインに指定するコマンド引数のうち，オプション名とオプション値のほかに指定する，規定のコマンド引数のことです。また，パラメーターの値もオペランドと呼びます。

（カ行）

開発環境

JP1/Advanced Shell - Developer が提供する，バッチ処理のためのジョブ定義スクリプトを開発するための環境です。

外部コマンド

シェルの組み込みコマンドではない，UNIX 互換コマンド，OS が提供するコマンドおよびユーザーによって作成される実行ファイルやプログラムのことを指します。

カスタムジョブ

ある特定の機能を持つジョブをJP1/AJSで実行できるように定義したジョブです。JP1/Advanced ShellでJP1/AJSのカスタムジョブ機能を利用するには，JP1/Advanced Shell用のカスタムジョブコンポーネントが必要です。

カバレージ情報

プログラムのテストがどれだけ網羅されているかを示す指標です。C0（ステートメントカバレージ情報）とC1（ブランチカバレージ情報）とがあります。

C0は，ジョブ定義スクリプトのコマンドをどれだけ実行したかの指標（%）です。

C1は，ジョブ定義スクリプトの分岐をどれだけ実行したかの指標（%）です。

環境情報

JP1/Advanced Shellを起動する前に設定が必要な，環境変数や環境ファイルのパラメーターなどの情報のことです。

環境設定パラメーター

環境ファイルに設定するパラメーターのうち，「`#-adsh_conf` *パラメーター 値*」という形式で指定し，JP1/Advanced Shellの実行環境を定義するパラメーターのことです。

環境ファイル

環境情報を格納したファイルのことです。

環境変数

ユーザーが設定できるシステムの各種の設定を格納した変数のことです。

クォーテーション

シングルクォーテーション（'）とダブルクォーテーション（"）があります。

組み込みコマンド

シェル本体に組み込まれたコマンドであり，シェル自身によって実行できます。JP1/Advanced Shellではシェル標準コマンド（特殊組み込みコマンドおよび正規組み込みコマンド），シェル拡張コマンドを提供します。特殊組み込みコマンドは，コマンドの構文を誤るとコマンドを実行しているシェルが終了する特徴を持ちます。正規組み込みコマンドは，コマンドの構文を誤ってもコマンドを実行しているシェルは終了しないで継続します。

コアダンプ

トレースプログラムが取得する保守情報の1つで，`core`ファイルと`dump`ファイルを指します。何らかのトラブルが発生した場合，メモリ上の情報をファイルに保存しておき，トラブルシューティングに活用します。

コマンド

シェル，コマンドプロンプトまたはジョブ定義スクリプトから実行するJP1/Advanced Shellで使用できるコマンドの総称のことです。

コマンドセパレータ

JP1/Advanced Shellでジョブ定義スクリプトの1行に複数のコマンドを記述できるようにするための機能です。

コマンドのグループ化

JP1/Advanced Shell で複数のコマンドをまとめて実行する機能のことをいいます。

コマンドプロンプト

Windows 環境でコマンドの入力を促すものです。

コマンドライン

ユーザーがコマンドを入力するための行です。Windows では，コマンドプロンプトにあり，行は>の次から入力します。UNIX では，シェルにあり，行は%の次から入力します。

コンソール

端末画面のことです。

（サ行）

サブシェル

UNIX 環境で，ジョブ定義スクリプトで外部コマンドや特定の構文を実行した際に自動的かつ一時的に生成される，ルートジョブでも子孫ジョブでもない，ジョブコントローラと同一名称の子プロセスです。

算術演算

ジョブ定義スクリプトで演算子を使用して変数に代入されている値を数値として扱って，計算を実施することをいいます。

シェル

コンピュータの入力装置から入力された指示を解釈して，OS に伝えるプログラムのことです。

シェル運用コマンド

実行形式のバイナリファイルやシェルスクリプトとして提供しているコマンドであり，ジョブ定義スクリプトだけで使用できるコマンドと，ジョブ定義スクリプトだけでなく OS のシェルやコマンドプロンプトなどからも使用できるコマンドがあります。シェル運用コマンドには，adshexec コマンド（バッチジョブを実行するコマンド）などがあります。

シェルオプション

シェルでコンピュータの入力装置から入力する指示に対して，選択的な機能を付け加えます。このことをシェルオプションといいます。

シェル拡張コマンド

シェル本体に組み込まれたコマンドであり，シェル自身のプロセスで実行されます。ジョブ定義スクリプトで使用できるコマンドです。

シェル拡張変数

JP1/Advanced Shell が提供する特別な意味を持つシェル変数です。

シェルコマンド

シェルまたはコマンドプロンプトから実行する JP1/Advanced Shell で使用できるコマンドの総称のことです。

シェルスクリプト

テキストファイルにコマンドを並べて記載しておき，シェルからそのコマンドを続けて実行できるようにしたテキストファイルを，シェルスクリプトといいます。JP1/Advanced Shell のシェルスクリプトは，Windows 環境と UNIX 環境で実行でき，ジョブ定義スクリプトといいます。

シェル標準コマンド

シェル本体に組み込まれたコマンドであり，シェル自身のプロセスで実行されます。ジョブ定義スクリプトで使用できるコマンドです。

シェル変数

ジョブ定義スクリプト内で値を代入する領域のことです。変数の作成変数の値を参照できます。

シグナル

UNIX の場合にプロセス間で非同期イベントの発生を伝える機構です。JP1/Advanced Shell では，ジョブの強制終了などに使用します。

システム実行ログ

システム管理者が JP1/Advanced Shell によるジョブ実行状況を統合管理するため，ジョブコントローラから出力されるログのことです。複数のジョブコントローラが出力するログを，1 つのログにまとめて出力できます。

子孫ジョブ

ルートジョブの子孫プロセスとして実行されるジョブ定義スクリプトのうち，次のどれかのパラメーターの指定，またはパラメーターのデフォルト定義によって実行されたジョブのことです。

- CHILDJOB_EXT パラメーター
- CHILDJOB_PGM パラメーター
- CHILDJOB_SHEBANG パラメーター

子孫ジョブ実行ログ出力ファイル

子孫ジョブが作成してルートジョブのスプールジョブディレクトリ内に出力する，子孫ジョブのジョブ実行ログ出力ファイルのことです。

実行アプリケーション

アプリケーション実行エージェント機能で実行するアプリケーション（プログラム）のことです。

実行環境

JP1/Advanced Shell が提供する，バッチ業務を実行するための環境です。狭義には，JP1/Advanced Shell のことです。

終了コード

ジョブ定義スクリプトまたはコマンドを実行した場合に返信されるコードのことです。

条件式

ジョブ定義スクリプトで使用する，数値比較，文字列比較，ファイル属性，論理結合の演算子および三項演算子を使って表す計算式のことです。

条件パラメーター

環境ファイルに設定するパラメーターのうち，物理ホストまたは特定の論理ホストだけで有効とする環境設定パラメーターおよびexport パラメーターを設定するために指定するパラメーターです。

条件判定

ジョブ定義スクリプトで，制御文に記述した条件式の結果を基に実行する処理を制御することです。

ショートオプション

コマンドの引数に指定するオプションのうち，先頭がハイフン（-）で始まり，そのあとに文字が 1 つ続く形式のオプションのことです。

初期設定スクリプトファイル

ジョブコントローラがジョブ定義スクリプトを実行する前に実行する，初期化用のスクリプトファイルのことです。

ジョブコントローラ

ジョブ実行時にジョブをコントロールするためのプログラムです。adshexec コマンドがジョブコントローラに該当します。

ジョブ識別子

ジョブ実行時に JP1/Advanced Shell が与える000001 から999999 の識別番号です。各ジョブには別々の識別子が与えられ，ジョブ識別子によって一意にジョブを特定できます。ジョブ識別子を999999 まで使用すると，ラップアラウンドして000001 以降の未使用のジョブ識別子を使用します。

ジョブ実行ログ

ジョブやジョブステップの開始・終了メッセージなどの，ジョブが出力したメッセージの集まりです。ジョブ実行ログの内容は，ジョブ終了時にジョブコントローラの標準エラー出力に出力します。

ジョブ情報

ジョブに付随した情報のことです。ジョブ名，ジョブ識別子およびジョブステップ名などがあります。

ジョブスケジューラ

ジョブのスケジュールを行う製品であり，JP1/Advanced Shell では関連製品として JP1/AJS と連携できます。

ジョブステップ

ある業務（仕事）を行うための最小単位で，JP1/Advanced Shell ではジョブ定義スクリプトで記載されたジョブ内で，ある処理の単位で区切った範囲をいいます。ジョブステップの集まりがジョブになります。#-adsh_step_start コマンド，#-adsh_step_error コマンド（省略できます），および#-adsh_step_end コマンドを記述して定義できます。

ジョブ定義スクリプトファイル

ジョブ定義スクリプトで作成した，ジョブを定義したプログラムのファイルのことです。

ジョブネット

実行順序を関連づけたジョブの集まりです。ジョブネット内のジョブは，あらかじめ定義した実行順序に従って自動的に実行されます。ジョブネットは，JP1/AJS の機能です。

シンボリックリンク

実際のファイルパスを格納したファイルを使ってリンクすることです。

スクリプト

テキストファイルにコマンドを並べて記載しておき，シェルからそのコマンドを続けて実行できるようにしたテキストファイルを，スクリプトといいます。JP1/Advanced Shell のスクリプトは，Windows 環境と UNIX 環境で実行でき，ジョブ定義スクリプトともいいます。

スクリプト開発部品

JP1/Advanced Shell が提供する，関数形式のジョブ定義スクリプトです。空白を削除した文字列の取得や，日付の経過日数の取得，ファイルサイズの取得など，汎用的な処理を関数として呼び出すことができます。

スクリプト拡張コマンド

ジョブ定義スクリプトで実行するコマンドです。通常のシェルスクリプトのコマンドに対してバッチジョブの実行を制御するための機能を付け加えたコマンドです。ジョブ実行制御コマンドともいいます。JP1/Advanced Shell では，`#-adsh` で始まるコマンドがあります。

スクリプト制御文

ジョブ定義スクリプトでコマンドを制御する文のことです。`if` 文，`for` 文，`while` 文，`until` 文および `case` 文があります。

スクリプトファイル

作成したスクリプトを保存したファイルです。

スクリプト予約語コマンド

ジョブ定義スクリプトで予約語として使用できるコマンドのことです。`time` コマンドがあります。

スプール

JP1/Advanced Shell でジョブの実行結果やジョブ実行ログを格納する場所です。

スプールジョブ

スプールディレクトリに作成されたジョブごとの実行結果のことです。

スプールジョブ名

ジョブ終了時にスプールジョブディレクトリ名に付加するジョブ名です。他のジョブ名と誤解を生じない場合には，単にジョブ名と表現する場合があります。

正規組み込みコマンド

シェル標準コマンドの組み込みコマンドの一種です。コマンドの構文を誤ってもコマンドを実行しているシェルが終了しないコマンドです。

制御文

スクリプト制御文と同じ意味です。

総称変数名

変数名の一部が同一である変数名を総称して表す変数名の表記方法です。

次の形式で表現します。

[*][*文字列*][*]

*は 0 文字以上の文字を示します。

（タ行）

ダイアログボックス

ユーザーに応答を促すウィンドウのことです。

通常ファイル

ジョブ定義スクリプトの入力および出力に使用するファイルです。ジョブ終了後にジョブ結果として残すファイルですが，ジョブの実行中に削除することもできます。#-adsh_file コマンドまたはadshfile コマンドで定義できます。

定義ファイル

トラブルシューティングのための資料を採取するディレクトリを定義しておくファイルです。

デバッガ

開発環境で作成したジョブ定義スクリプトをテストして不具合を調査するプログラムです。Windows 環境では，JP1/Advanced Shell エディタのデバッグ機能を使います。UNIX 環境では，adshexec コマンドに-d オプションを指定してデバッガを起動します。

デバッグ

開発環境で作成したジョブ定義スクリプトをテストして不具合を調査することです。デバッガを起動して調査します。

特殊組み込みコマンド

シェル標準コマンドの組み込みコマンドの一種です。コマンドの構文を誤るとコマンドを実行しているシェルが終了するコマンドです。

トラップアクション

trap コマンドの引数「action」に設定する動作のことです。

トレースログ

JP1/Advanced Shell でトラブルが発生した場合に，問題点を解明するために採取する情報のことです。

（ハ行）

パイプ

前のコマンドの標準出力を次のコマンドの標準入力へ連結する機能のことです。

バッチ業務サーバ

JP1/Advanced Shell をインストールしてバッチジョブを実行するサーバのことです。JP1/AJS を使用する場合，JP1/AJS - Agent または JP1/AJS - Manager をインストールします。

バッチジョブ

バッチ処理で実行するジョブのことです。

バッチ処理

収集したデータやトランザクションを 1 日分，1 週間分，1 か月分などにまとめて一括処理することです。

ヒアドキュメント

ジョブ定義スクリプト内で使用されるリダイレクト機能のことです。標準入力をジョブ定義スクリプト内で生成します。

引数

コマンドラインやジョブ定義スクリプトにコマンドを実行する記述をする場合，コマンド名の後ろに区切り文字で区切って指定する項目の総称を引数といいます。

標準エラー出力（stderr）

プログラムがエラーなどのメッセージを出力するストリームです。

標準出力（stdout）

プログラムがデータを出力するストリームです。

標準入力（stdin）

プログラムへデータを入力するストリームです。

ファイルディスクリプタ

ジョブコントローラでの入出力種別ごとに，番号を付けて区別するようにしたものです。ジョブコントローラでは，標準出力に1，標準エラー出力に2，およびそれ以外に3～9 を割り当てて使用できます。

ファイルの割り当て

JP1/Advanced Shell では，ファイルの後処理を登録することを含めてファイルの割り当てといいます。

ブレークポイント

ジョブ定義スクリプトの開発時にジョブ定義スクリプトの動作状態を確認するために，ジョブ定義スクリプト中に挿入される強制実行停止コードのことです。ブレークポイントではデバッガが処理を停止するため，開発者は停止直前の変数を確認できます。

プログラム出力データファイル

ユーザープログラムの出力結果をジョブ実行ログと同様に一元管理するために，JP1/Advanced Shell が自動的にファイル名を作成して，ユーザープログラムが実行結果を出力するためのファイルです。

ベース名

ファイル名から「*.拡張子*」を除いた部分の名称です。バッチジョブを実行するコマンドのプログラム（adshexec.exe）のベース名は，adshexec となります。

変数

ジョブ定義スクリプト内で値を扱うために使用する領域および配列のことです。変数にはシェル変数および環境変数も含まれます。

（マ行）

メタキャラクタ

ジョブ定義スクリプト内でそれぞれに特別な意味を持つキャラクタ（文字列）のことです。

（ラ行）

リダイレクト

ジョブ定義スクリプトでは，コマンド実行前に実行結果の出力先の変更やコマンド実行に必要な情報の入力先を変更できます。これをリダイレクトといいます。通常，標準入力はキーボードに，標準出力は画面に割り当てられていますが，リダイレクトではこれらの割り当てを変更します。

流量制御

adshecho コマンドやadshread コマンドの実行時に発行される JP1 イベントに対して，発行間隔を制御する機能です。

ルートジョブ

JP1/AJS やログインシェルなどから実行するジョブのうち，子孫ジョブ以外のジョブのことです。

ログ

コンピュータが出力する記録情報のことです。ログには記録した時間やメッセージなどが出力されます。

ロングオプション

コマンドの引数に指定するオプションのうち，先頭が連続する 2 個のハイフン（-）で始まり，そのあとに文字列が続く形式のオプションのことです。

（ワ行）

ワイルドカード

ワイルドカードは*と?で記述します。*は任意の文字列を，?は任意の 1 文字を表します。

また，[ ]で囲まれた文字列の 1 文字に合致させたり，「-」で範囲を指定したり，「!」でその文字列以外を指定したり，コンマで区切られた文字列のどれかを選択させたりできます。

索引

記号

数字

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

は

ひ

ふ

へ

ほ

ま

め

も

ゆ

よ

ら

り

る

れ

ろ

わ

株式会社 日立製作所
〒 100-8280　東京都千代田区丸の内一丁目 6 番 6 号