HITACHI
Inspire the Next

JP1 Version 9

JP1/Advanced Shell

解説・手引・文法・操作書

3020-3-S35-30

■対象製品

P-1M12-B111 JP1/Advanced Shell 09-51（適用 OS：AIX）
P-9S12-B111 JP1/Advanced Shell 09-51-01（適用 OS：Red Hat Enterprise Linux 5（x86），Red Hat Enterprise Linux 5（AMD/Intel 64），Red Hat Enterprise Linux 5 Advanced Platform（x86），Red Hat Enterprise Linux 5 Advanced Platform（AMD/Intel 64），Red Hat Enterprise Linux Server 6（32-bit x86），Red Hat Enterprise Linux Server 6（64-bit x86_64））
P-2412-B114 JP1/Advanced Shell 09-51-01（適用 OS：Windows 7，Windows Server 2008，Windows Vista，Windows Server 2003，Windows XP）
P-2612-B214 JP1/Advanced Shell - Developer 09-51-01（適用 OS：Windows 7，Windows Server 2008，Windows Vista，Windows Server 2003，Windows XP）

■輸出時の注意

本製品を輸出される場合には，外国為替および外国貿易法ならびに米国の輸出管理関連法規などの規制をご確認の上，必要な手続きをお取りください。
なお，ご不明な場合は，弊社担当営業にお問い合わせください。

■商標類

■マイクロソフト製品の表記について

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 2008	Microsoft(R) Windows Server(R) 2008 Enterprise 日本語版
		Microsoft(R) Windows Server(R) 2008 Standard 日本語版
		Microsoft(R) Windows Server(R) 2008 R2 Datacenter 日本語版
		Microsoft(R) Windows Server(R) 2008 R2 Enterprise 日本語版
		Microsoft(R) Windows Server(R) 2008 R2 Standard 日本語版
	Windows Server 2003	Microsoft(R) Windows Server(R) 2003, Enterprise Edition 日本語版※
		Microsoft(R) Windows Server(R) 2003, Standard Edition 日本語版
		Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition 日本語版※
		Microsoft(R) Windows Server(R) 2003 R2, Standard Edition 日本語版
	Windows Server 2003 (x64)	Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition 日本語版※
		Microsoft(R) Windows Server(R) 2003, Standard x64 Edition 日本語版
		Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition 日本語版※
		Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition 日本語版
Windows	Windows 7	Microsoft(R) Windows(R) 7 Enterprise 日本語版
		Microsoft(R) Windows(R) 7 Professional 日本語版
		Microsoft(R) Windows(R) 7 Ultimate 日本語版
	Windows Vista	Microsoft(R) Windows Vista(R) Business 日本語版
		Microsoft(R) Windows Vista(R) Enterprise 日本語版
		Microsoft(R) Windows Vista(R) Ultimate 日本語版
	Windows XP	Microsoft(R) Windows(R) XP Professional Operating System

注※　これらの製品を総称して，Windows Server 2003 Enterprise Edition と表記することがあります。

■発行

2012 年 4 月　3020-3-S35-30

■著作権

変更内容

変更内容（3020-3-S35-30） JP1/Advanced Shell 09-51-01，JP1/Advanced Shell - Developer 09-51-01

追加・変更内容	変更個所
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。	2.6.3，2.6.3(1)，2.6.3(4)，2.6.3(5)，2.6.4，2.6.4(1)，5.2.2(3)，5.9.5(1)，7.2.1 の表 7-1，CHILDJOB_SHEBANG パラメーター，COMMAND_CONV_ARG パラメーター，PATH_CONV_ACCESS パラメーター
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 コマンドに戻り値を追加した。	2.6.5，2.8，3.3.1(1)，3.3.3，3.6，5 章，5.1.5(8)(b)，5.1.5(8)(d)，5.2.2(3)，7.2.1 の表 7-1，OUTPUT_STDOUT パラメーター，adshexec コマンド，adshhk コマンド，awk コマンド，find コマンド，10.2.1，付録 D
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	2.6.5(1)，2.6.14，3.3.3(1)，5.1.9，5.2.2(3)，7.2.1 の表 7-1，7.3，10.2.1，11.3
メッセージを追加した。 KNAX0308-E，KNAX6571-I，KNAX6578-I，KNAX6594-E，KNAX6805-I，KNAX6806-I，KNAX7901-I	11.2，11.3
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	11.3

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 は次の製品で構成されます。

- 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 を区別する場合は，「実行環境」，「開発環境」と表記します。

■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

■マニュアルの構成

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。
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。

第 1 章　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，処理の流れおよび機能概要について説明しています。

第 2 章　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ています。

第 3 章　バッチジョブの実行

JP1/Advanced Shell（実行環境）を使用して，バッチジョブの運用を実施します。バッチジョブの実行方法やバッチジョブでの動作について説明しています。

第 4 章　エディタの操作

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。

第 5 章　ジョブ定義スクリプトの文法

ジョブ定義スクリプトの文法について説明しています。

第 6 章　ジョブ定義スクリプトのデバッグ

UNIX 環境でコマンドを使用して，ジョブ定義スクリプトファイルをデバッグします。JP1/Advanced Shell のデバッガ機能について説明しています。

第 7 章　環境ファイルで設定するパラメーターとコマンド

環境ファイルに環境設定パラメーターを設定して，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。また，環境設定コマンドを指定して環境変数を定義します。パラメーターとコマンドの記述形式と詳細について説明しています。

第 8 章　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。

第 9 章　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シェル標準コマンド，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記述形式と詳細について説明しています。

第 10 章　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。

第 11 章　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 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　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 C　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報（関連マニュアル，マニュアルでの表記）について説明しています。

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。

■読書手順

このマニュアルは，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。

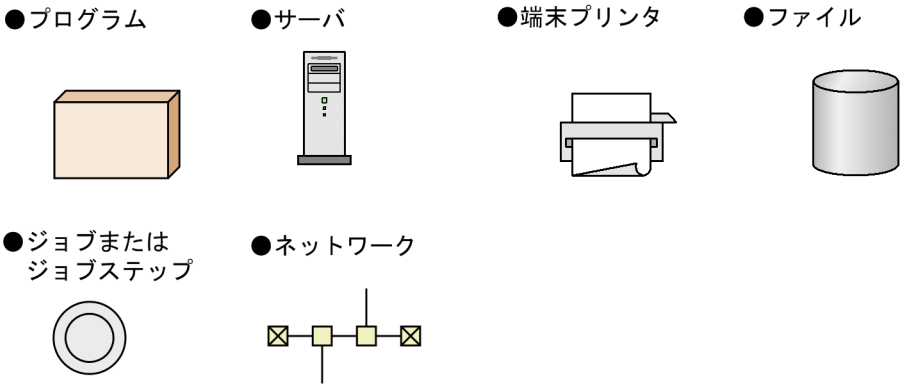
章または付録のタイトル	運用者	開発者
1.　JP1/Advanced Shell の概要	○	○
2.　JP1/Advanced Shell を利用するための準備	◎	○
3.　バッチジョブの実行	◎	△
4.　エディタの操作	－	◎
5.　ジョブ定義スクリプトの文法	－	◎
6.　ジョブ定義スクリプトのデバッグ	○	○
7.　環境ファイルで設定するパラメーターとコマンド	○	△
8.　運用時に使用するコマンド	◎	△
9.　ジョブ定義スクリプトのコマンドおよび制御文	△	◎
10.　トラブルシューティング	○	△
11.　メッセージ	△	△
付録 A　カバレージ情報を取得する対象	△	△
付録 B　各バージョンの変更内容	△	△
付録 C　このマニュアルの参考情報	△	△
付録 D　用語解説	○	○

（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。

■図中で使用する記号

このマニュアルの図中で使用する記号を，次のように定義します。



■このマニュアルで使用している記号

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。 • 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。 • ［ ］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。

記号	意味
*	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，… 「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。
△	1 個の半角のスペースを示しています。
$△_0$	0 バイト以上のスペースを指定します。スペースは省略できます。
$△_1$	1 バイト以上のスペースを必ず指定します。スペースは省略できません。

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＞	, ｜ . ｜ / ｜ ' ｜ (｜) ｜ * ｜ & ｜ + ｜ - ｜ = ｜△(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。
＜任意文字列＞	任意の文字による文字列。 1. JP1/Advanced Shell では文字種別をチェックしません。 2. 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。 3. 記号名称の範囲での利用を推奨します。
＜ 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。

目次

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。

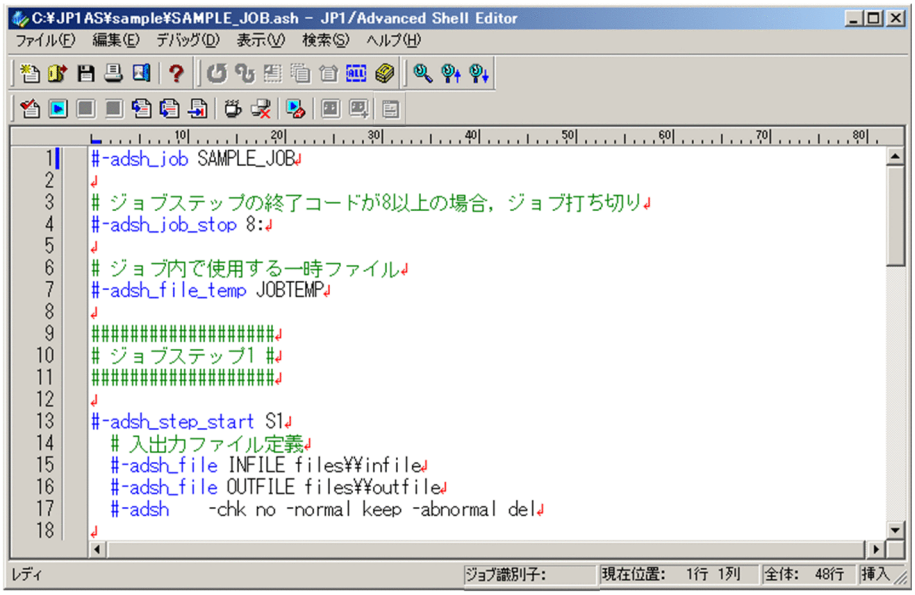
● 開発環境

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

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　バッチ業務の運用性・保守性の向上

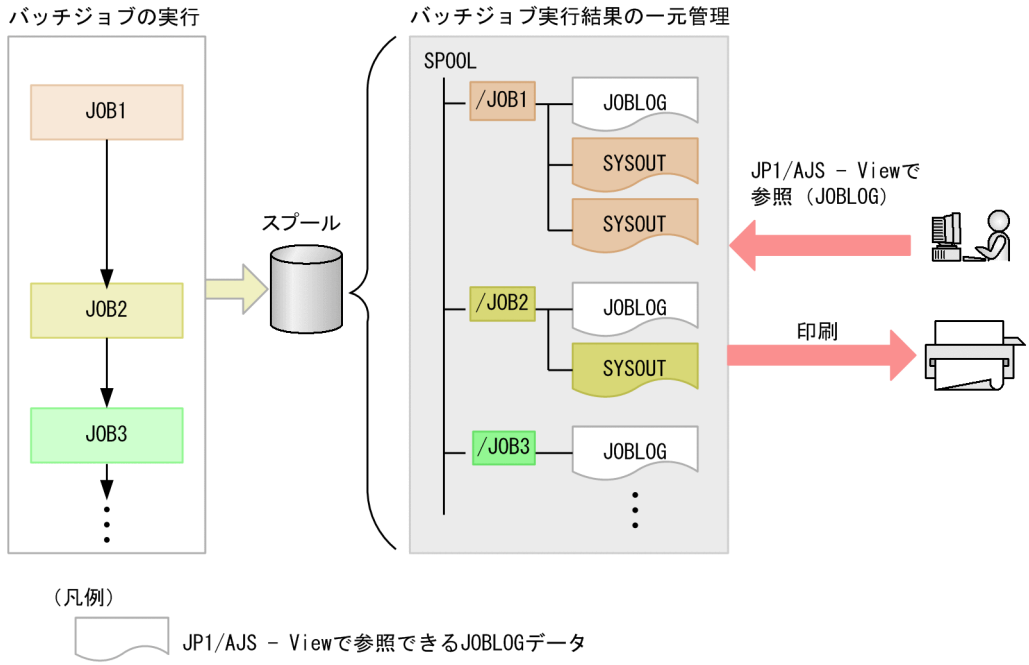
● ジョブ実行ログの運用

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。

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



ジョブ実行ログの詳細については，「10.2.1　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。

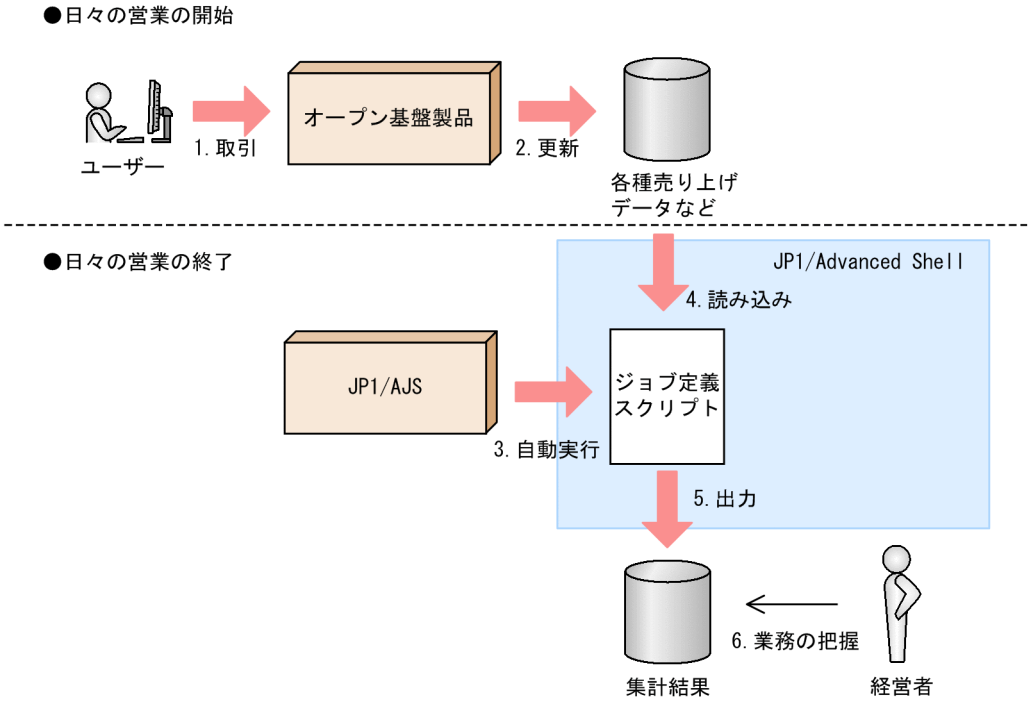
1.2　業務への応用例

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）



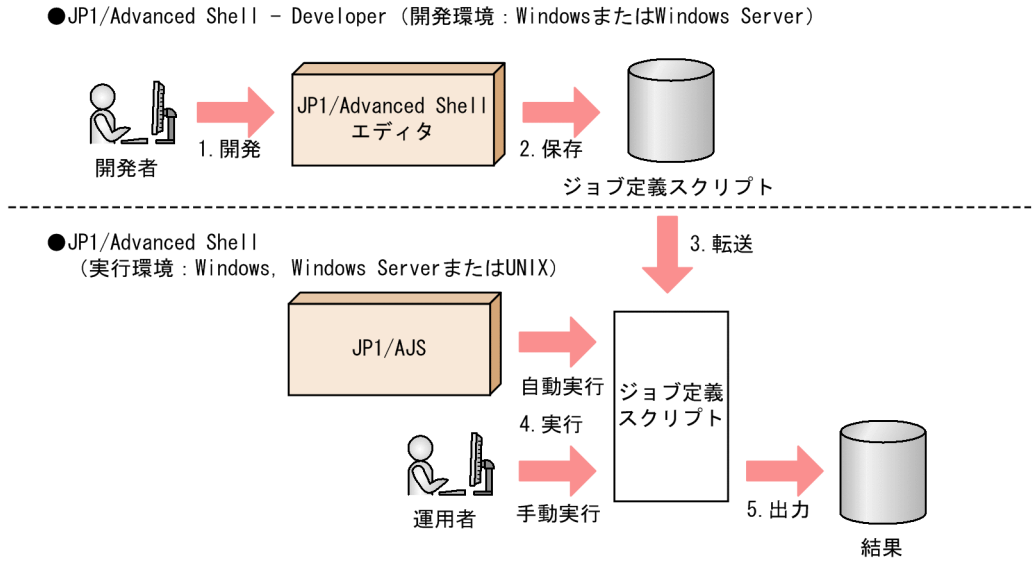
1. **日々の営業を開始し，ユーザーは商品などの取引をします。**
2. **オープン基盤製品は，各種売り上げデータなどを更新します。**
3. **日々の営業が終了して，**JP1/AJS **は指定された時間にジョブ定義スクリプトの自動実行を指示します。**
4. JP1/Advanced Shell **は各種売り上げデータを処理するためのジョブ定義スクリプトを実行します。**
5. JP1/Advanced Shell **はジョブ定義スクリプトの実行結果を出力します。**
6. **経営者は，実行結果を基に集計情報や商品の売り上げの推移などを把握できます。**

1.3　システムの全体構成

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。

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

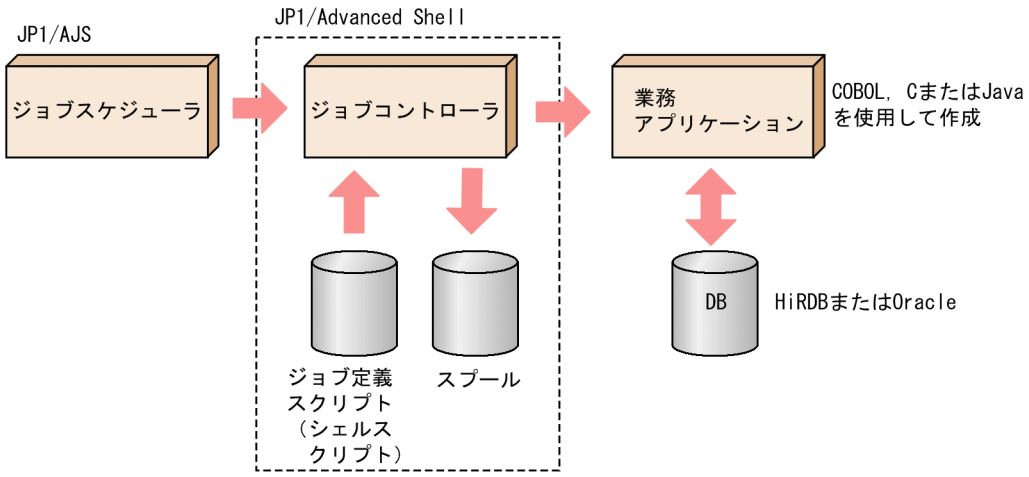


1. Windows **または** Windows Server **上の開発環境で開発者が** JP1/Advanced Shell **エディタを使ってジョブ定義スクリプトを開発します。**
2. JP1/Advanced Shell **エディタからジョブ定義スクリプトを保存します。**
3. **ジョブ定義スクリプトを** Windows，Windows Server **または** UNIX **環境に転送します。**
4. Windows，Windows Server **または** UNIX **上の実行環境で運用者が次のような方法を使ってジョブ定義スクリプトの実行を指示します。**
   - 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. **ジョブ定義スクリプトを実行した結果を出力します。**

1.4　処理の流れ

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に当たります。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

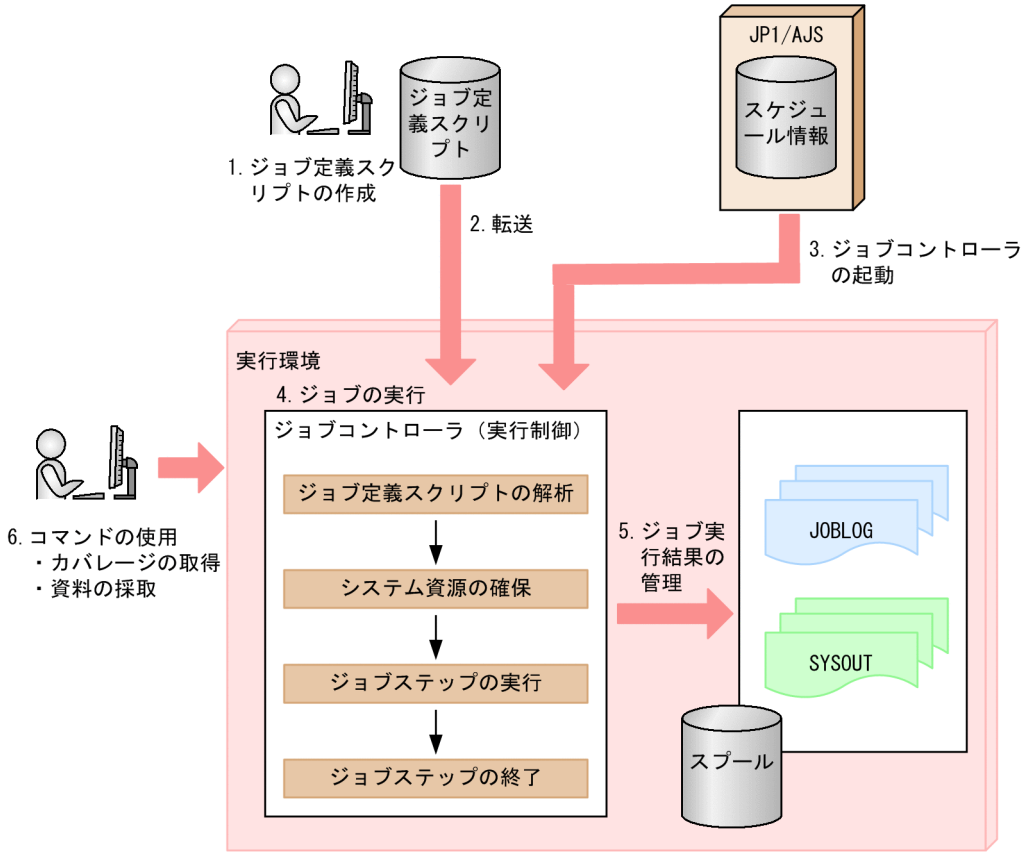


JP1/AJS と連携した場合，バッチジョブを実行するスケジュールを登録してバッチジョブを実行できます。

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 1，3，4，5 および 6 の項目が，JP1/Advanced Shell が処理する内容です。

図 1-6 JP1/Advanced Shell の運用の流れ



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（実行制御）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→システム資源の確保→ジョブステップの実行→ジョブステップの終了
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. **コマンドを使用してカバレージを取得したり，トラブルシューティングのための資料（トレースなど）を採取したりできます。**

1.5 機能概要

実行環境で利用する機能と開発環境で利用する機能とに分けて説明します。

1.5.1 実行環境で利用する機能

実行環境の機能を使用して，バッチジョブを効率良く実行したり，実行ログを取得してトラブル発生時の早期解決に役立てたりできます。バッチジョブを効率良く実行するために次の表で示す実行環境の機能を利用できます。

表 1-1 実行環境の機能

機能	関連項目	参照先
コマンドを実行する	シェル運用コマンド	8.3
	UNIX 互換コマンド	2.6.6，8.4
	シェル標準コマンド	9.3
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	5.10.1，9.4
	一時ファイル	5.10.2，9.4
	プログラム出力データファイル	5.10.3，9.4
ジョブの実行を制御する	ジョブ名を宣言する	5.9.1，9.4
	ジョブの打ち切り条件を定義する	5.9.2，9.4
	ジョブステップを開始または終了する	5.9.3，9.4
	ジョブステップで変数を使用する	5.1，6.2.20
	終了コードを定義する	2.6.11，7.3
	ジョブを強制終了したときの動作を設定する	3.9，9.3
	外部スクリプトを呼び出す	5.9.6，9.4
	子孫ジョブを起動する	2.6.5，3.3.1(1)，3.3.3，5.1.9，7.3
	カスタムジョブを登録する	2.7
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	2.6.2，2.6.3，2.6.4，2.6.14，7.3
スプールジョブを削除する	adshhk コマンド（スプールジョブを削除する）を使用する	3.7，8.3
カバレージ情報を使用する	カバレージ情報を取得する	3.8，8.3，付録 A
	カバレージ情報を表示する	3.8，8.3
	カバレージ情報をマージする	3.8，8.3
	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（カバレージ採取の一括有効化機能）	3.8，8.3
ジョブ定義スクリプトをデバッグする【UNIX 限定】	コマンドを使ってデバッグする	6.
ジョブ実行ログを採取する	バッチジョブの運用性・保守性を向上する	1.1.3
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	10.

1.5.2 開発環境で利用する機能

開発環境では，Windows の GUI 環境で次の機能を使用できます。

表 1-2 開発環境での機能

機能	関連項目	参照先
ジョブ定義スクリプトを作成する	エディタを使ってジョブ定義スクリプトを作成する	4.，5.
	シェル運用コマンドを使用する	8.3
	UNIX 互換コマンドを使用する	2.6.6，8.4
	シェル標準コマンドを使用する	9.3
	スクリプト拡張コマンドを使用する	9.4
	スクリプト制御文を使用する	9.5
	スクリプト予約語コマンドを使用する	9.6
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	5.10.1，9.4
	一時ファイル	5.10.2，9.4
	プログラム出力データファイル	5.10.3，9.4
ジョブの実行を制御する	ジョブ名を宣言する	5.9.1，9.4
	ジョブの打ち切り条件を定義する	5.9.2，9.4
	ジョブステップを開始または終了する	5.9.3，9.4
	ジョブステップで変数を使用する	5.1，6.2.20
	終了コードを定義する	2.6.11，7.3
	ジョブを強制終了したときの動作を設定する	3.9，9.3
	外部スクリプトを呼び出す	5.9.6，9.4
	子孫ジョブを起動する	2.6.5，3.3.1(1)，3.3.3，5.1.9，7.3
ジョブ定義スクリプトで変数を使用する	シェル変数	5.5
	環境変数	5.8
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	2.6.2，2.6.3，2.6.4，2.6.14，7.3
ジョブ定義スクリプトをデバッグする	エディタを使ってデバッグする（デバッグ時のカバレージ情報の採取および表示を含む）	4.4.7，6.
ジョブ実行ログを採取する	バッチジョブの運用性・保守性を向上する	1.1.3
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	10.

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。また，スプールのディレクトリ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】

Windows の場合は，任意のフォルダにインストールできます。デフォルトのフォルダは，「**システムドライブ**¥Program Files※ ¥Hitachi¥JP1AS」です。

注※

64 ビット版の Windows 7，Windows Server 2008，Windows Vista，および Windows Server 2003 の場合は，「Program Files」を「Program Files (x86)」と読み替えてください。

実行環境は，インストール先フォルダの JP1ASE フォルダにインストールされます。開発環境は，インストール先フォルダの JP1ASD フォルダにインストールされます。また，実行環境に含まれるカスタムジョブ（JP1/Advanced Shell - Custom Job）は，インストール先フォルダの JP1ASV にインストールされます。

実行環境は，サーバにインストールし，開発環境は，クライアント PC にインストールすることを推奨します。JP1/Advanced Shell - Custom Job は，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

```
├─JP1ASD                                  ：開発環境フォルダ
│      ├─readme.txt                       ：readmeファイル
│      ├─bin                              ：実行ライブラリのフォルダ
│      ├─cmd                              ：UNIX互換コマンドのフォルダ
│      ├─doc─ja─help─INDEX.HTM            ：ヘルプ（マニュアル）
│      ├─maintenance                      ：資料を採取するコマンドのフォルダ
│      └─sample                           ：サンプルデータ
│           ├sample.ase                   ：環境ファイルのテンプレート
│           └sample.ash                   ：ジョブ定義スクリプトのサンプル
├─JP1ASE                                  ：実行環境フォルダ
│      ├─readme.txt                       ：readmeファイル
│      ├─bin                              ：実行ライブラリのフォルダ
│      ├─cmd                              ：UNIX互換コマンドのフォルダ
│      ├─doc─ja─help─INDEX.HTM            ：ヘルプ（マニュアル）
│      ├─maintenance                      ：資料を採取するコマンドのフォルダ
│      ├─sample                           ：サンプルデータ
│      │     ├sample.ase                 ：環境ファイルのテンプレート
│      │     └sample.ash                 ：ジョブ定義スクリプトのサンプル
│      └─util─JP1ASVC.msi                 ：カスタムジョブのインストーラ
└─JP1ASV                                  ：カスタムジョブのフォルダ
       ├─bin                              ：実行ライブラリのフォルダ
       ├─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※
       └─maintenance                      ：資料を採取するコマンドのフォルダ
```

注※

JP1/AJS3 - View 09-50 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ディレクトリにコピーする必要があります。

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

トレース出力先フォルダは，共通アプリケーション（共通 AP）データフォルダに作成されます。

共通APデータフォルダ※

```
└─Hitachi─JP1AS─JP1ASD       ：開発環境フォルダ
                │     ├─trace ：トレースフォルダ
                │     └─uxpl  ：ログフォルダ
                ├─JP1ASE       ：実行環境フォルダ
                │     ├─trace ：トレースフォルダ
                │     └─uxpl  ：ログフォルダ
                ├─JP1ASV       ：カスタムジョブのフォルダ
                │     └─trace ：トレースフォルダ
                └─misc         ：製品共通ライブラリのフォルダ
                      ├─trace ：トレースフォルダ
                      └─uxpl  ：ログフォルダ
```

注※

Windows のバージョンによってフォルダが異なります。

- Windows Server 2003，Windows Server 2003(x64) または Windows XP の場合の例
  C:¥Documents and Settings¥All Users¥Application Data
- Windows 7，Windows Server 2008 または Windows Vista の場合の例
  C:¥ProgramData

システム実行ログ，スプールおよび一時ファイルは，全ユーザー共通文書フォルダにインストールされます。

全ユーザー共通文書フォルダ※

```
└─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
                │     ├─spool：スプールフォルダ
                │     └─temp ：一時フォルダ
                └─misc       ：製品共通ライブラリのフォルダ
                      └─log  ：ログフォルダ
```

注※

Windows のバージョンによってフォルダが異なります。

- Windows Server 2003，Windows Server 2003(x64) または Windows XP の場合の例
  C:¥Documents and Settings¥All Users¥Documents
- Windows 7，Windows Server 2008 または Windows Vista の場合の例
  C:¥Users¥Public¥Documents

2.1.2 インストール先ディレクトリ【UNIX 限定】

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

```
├─bin                          ：実行ライブラリのディレクトリ（プログラム）
├─cmd                          ：UNIX互換コマンドのディレクトリ
├─instlog                      ：インストールのログ情報
├─lib                          ：ライブラリのディレクトリ
├─log                          ：システム実行ログのディレクトリ
├─maintenance-adshcollect      ：資料を採取するコマンド
├─sample                       ：サンプルデータ
│    ├─sample.ase              ：環境ファイルのテンプレート
│    └─sample.ash              ：ジョブ定義スクリプトのサンプル
├─trace                        ：トレースのディレクトリ
└─util─JP1ASVC.msi             ：カスタムジョブのインストーラ
```

スプールと一時ディレクトリは，次のディレクトリにインストールされます。

```
/var/opt/jp1as─spool  ：スプールディレクトリ
               └─temp   ：一時ディレクトリ
```

2.2 プログラムの種類

JP1/Advanced Shell で使用する主なプログラムを Windows 環境と UNIX 環境とに分けて，次の表に示します。プログラムのファイルは，インストール先ディレクトリの bin にあります。ただし，UNIX 互換コマンドは，インストール先ディレクトリの cmd にあります。

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

プログラムの概要（アイコン）	ファイル名	説明
資料の採取	adshcollect.bat	トラブルシューティングで資料を採取するプログラムです。実行環境と開発環境で使用できます。
JP1/Advanced Shell 実行定義プログラム（  ）	adshctmj.exe	カスタムジョブで JP1/Advanced Shell の実行環境を定義するプログラムです。
PC ジョブの JP1/Advanced Shell 実行定義プログラム	adshctmjpc.bat	カスタムジョブで 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。
UNIX ジョブの JP1/Advanced Shell 実行定義プログラム	adshctmjunix.bat	カスタムジョブで 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。
カバレージ情報のマージ	adshcvmerg.exe	カバレージ情報をマージするプログラムです。実行環境と開発環境で使用できます。
コマンドからのカバレージ情報の表示	adshcvshow.exe	カバレージ情報を表示するプログラムです。実行環境と開発環境で使用できます。
エディタからのカバレージ情報の表示	adshcvview.exe	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。
JP1/Advanced Shell エディタ（  ）	adshedit.exe	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。
エディタでのデバッグ	adshesub.exe	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。
バッチジョブの実行	adshexec.exe	実行環境でバッチジョブを実行するプログラムです。シェル運用コマンドまたは JP1/AJS から起動します。
バッチジョブのジョブコントローラの起動	adshexecsub.exe	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します。このプログラムは adshexec.exe から自動的に起動されます。
スプールフォルダの自動削除	adshhk.exe	スプールフォルダを自動的に削除します。実行環境と開発環境で使用できます。
UNIX 互換コマンド	awk.exe，cat.exe，cmp.exe，cp.exe，cut.exe，date.exe，diff.exe，expr.exe，find.exe，grep.exe，head.exe，hostname.exe，ls.exe，mkdir.exe，mv.exe，rm.exe，rmdir.exe，sed.exe，sleep.exe，sort.exe，split.exe，tail.exe，uname.exe，uniq.exe，wc.exe	UNIX のコマンドを 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。

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

プログラムの概要	ファイル名	説明
資料の採取	adshcollect	実行環境でトラブルシューティングで資料を採取するプログラムです。
カバレージ情報のマージ	adshcvmerg	実行環境でカバレージ情報をマージするプログラムです。
カバレージ情報の表示	adshcvshow	実行環境でカバレージ情報を表示するプログラムです。
バッチジョブの実行	adshexec	実行環境でバッチジョブを実行するプログラムです。シェル運用コマンドまたは JP1/AJS から起動します。
スプールディレクトリの自動削除	adshhk	実行環境でスプールディレクトリを自動的に削除するプログラムです。
UNIX 互換コマンド	awk，cat，cmp，cp，cut，date，diff，expr，find，grep，head，hostname，ls，mkdir，mv，rm，rmdir，sed，sleep，sort，split，tail，uname，uniq，wc	実行環境で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。

2.3　前提条件

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について説明します。

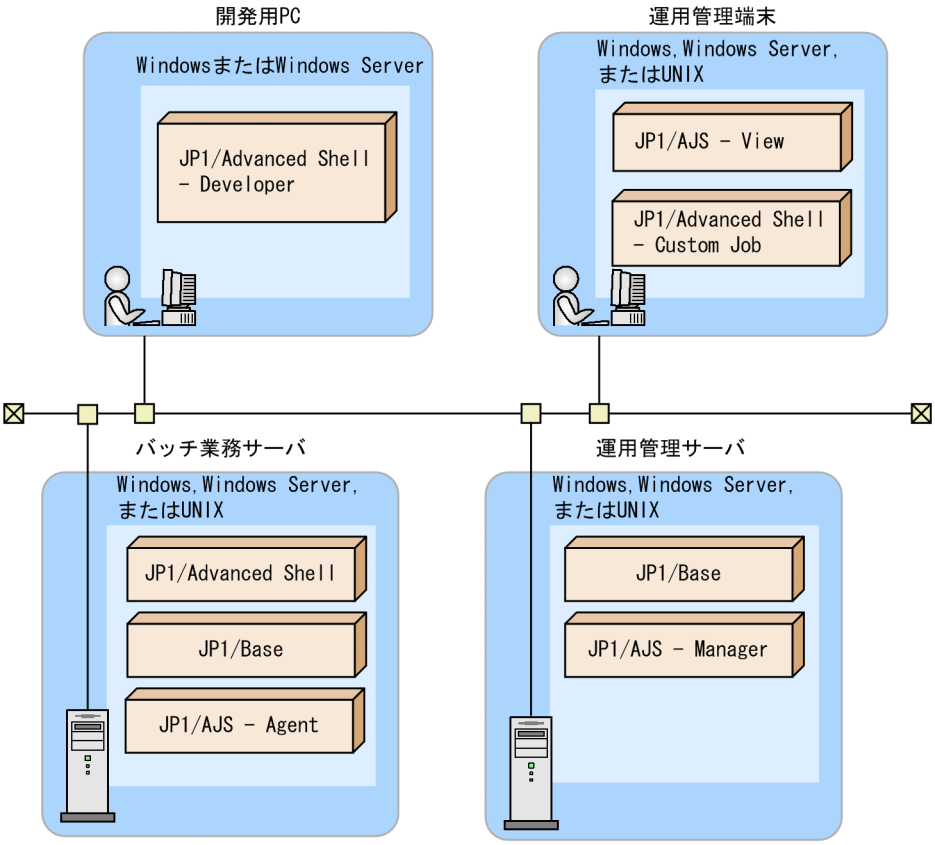
2.3.1　システム構成

JP1/AJS からバッチジョブを実行する場合と手動でバッチジョブを実行する場合とに分けて説明します。

(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

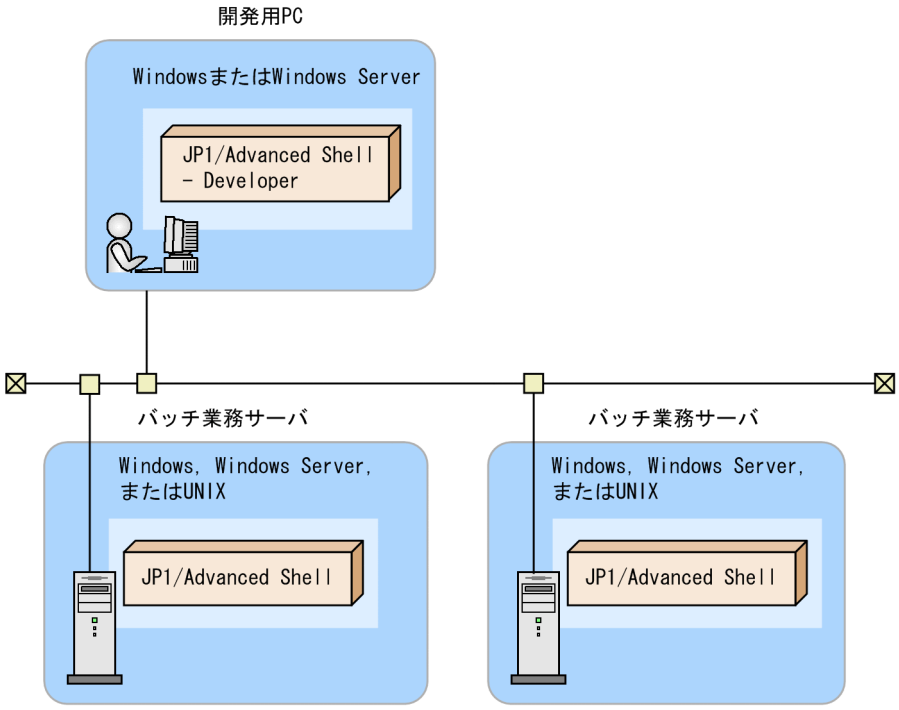


開発用 PC では，JP1/Advanced Shell - Developer を使用してジョブ定義スクリプトを作成します。バッチ業務サーバでは，ジョブ定義スクリプトを手動または自動実行します。運用管理サーバでは，実行されたジョブを管理します。運用管理端末では，JP1/AJS - View を使用してジョブの実行結果を表示したり，自動実行するジョブ定義スクリプトを定義したりします。運用管理端末で，JP1/Advanced Shell のジョブ定義を行うには JP1/Advanced Shell - Custom Job（カスタムジョブ）が必要になります。

(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



開発用 PC では，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。バッチ業務サーバでは，ジョブ定義スクリプトを手動で実行します。

2.3.2　前提プログラム

JP1/Advanced Shell の前提プログラムについて説明します。

(1)　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

プログラム	組み込み先
Windows Server 2008，Windows Server 2003，Windows 7，Windows Vista，または Windows XP	JP1/Advanced Shell と同じバッチ業務サーバ
Red Hat Enterprise Linux(R) 5， Red Hat Enterprise Linux Server 6， AIX 5L 5.3，AIX 6.1，または AIX 7.1	JP1/Advanced Shell と同じバッチ業務サーバ

(2)　開発環境の前提プログラム【Windows 限定】

開発環境の前提プログラムを次の表に示します。

表 2-4　開発環境の前提プログラム【Windows 限定】

プログラム	組み込み先
Windows Server 2008，Windows Server 2003，Windows 7，Windows Vista，または Windows XP	JP1/Advanced Shell - Developer と同じ開発用 PC

(3) カスタムジョブの前提プログラム【Windows 限定】

カスタムジョブの前提プログラムを次の表に示します。カスタムジョブは，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 2-5 カスタムジョブの前提プログラム【Windows 限定】

プログラム	組み込み先
Windows Server 2008，Windows Server 2003，Windows 7，Windows Vista，または Windows XP	JP1/Advanced Shell - Custom Job と同じ運用管理端末
JP1/AJS - View	

2.3.3 関連プログラム

JP1/Advanced Shell の関連プログラムについて説明します。

(1) 実行環境の関連プログラム

実行環境に関連するプログラムを次の表に示します。

表 2-6 実行環境の関連プログラム

実施したい処理	組み込み先	プログラム
JP1/AJS からジョブ定義スクリプトを実行する	JP1/Advanced Shell と同じバッチ業務サーバ	JP1/Base JP1/Automatic Job Management System 2 - Agent ※または JP1/Automatic Job Management System 3 - Agent ※
ジョブを管理する	運用管理サーバ	JP1/Base JP1/Automatic Job Management System 2 - Manager ※または JP1/Automatic Job Management System 3 - Manager ※
ジョブの実行結果を知りたい	運用管理端末（Windows）	JP1/Automatic Job Management System 2 - View または JP1/Automatic Job Management System 3 - View

注※

JP1/AJS - Manager には，JP1/AJS - Agent の機能が含まれているため，バッチジョブ実行システムと同一サーバ上に JP1/AJS - Manager があれば，JP1/AJS - Agent は不要です。

(2) 開発環境の関連プログラム【Windows 限定】

開発環境には，関連プログラムはありません。

(3) カスタムジョブの関連プログラム【Windows 限定】

カスタムジョブには，関連プログラムはありません。

(4) 各プログラムの役割

各プログラムの役割を次に示します。

- JP1/Base

  ユーザーの管理，バッチジョブなどのイベントの収集・管理ができます。

- JP1/Automatic Job Management System 2 - Agent **または** JP1/Automatic Job Management System 3 - Agent

  JP1/AJS - Manager から実行依頼を受けたバッチジョブなどの処理を実行できます。

● JP1/Automatic Job Management System 2 - Manager または JP1/Automatic Job Management System 3 - Manager
ジョブネットの定義情報やスケジュール情報を管理し，バッチジョブなどの処理の実行をエージェント（JP1/AJS - Agent または JP1/AJS - Manager）に依頼できます。また，JP1/AJS - Manager 自身で処理の実行もできます。

● JP1/Automatic Job Management System 2 - View または JP1/Automatic Job Management System 3 - View
ジョブネットの定義や操作，実行状況や結果の表示などができます。

2.3.4　JP1/Advanced Shell で使用するファイル

JP1/Advanced Shell で使用するファイルを次の表に示します。トレース情報の採取方法については，「10.4　資料の採取方法」を参照してください。

表 2-7　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	.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。
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	ユーザー任意※	デバッグ機能で利用する情報ファイルです。拡張子以外はジョブ定義スクリプトファイルと同じファイル名です。 ジョブ定義スクリプトファイル保存時・デバッグ中・エディタ終了時，実行環境の設定で OK を押したときに保存します。
システム実行ログ	.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。
トレース情報	.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	トラブルが発生した場合にトラブル発生の経緯を調査，および各処理の処理時間を測定したりするために採取するログ情報です。
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 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	システム内部で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_ **スクリプト名** _ **通番** .txt 通番：エディタ起動中の番号

注※
デバッグ情報ファイルは，ジョブ定義スクリプトファイルと同じディレクトリに出力します。次のような場合には，デバッグ情報ファイルの保存ができずに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

注意事項

- 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】
- 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は使用しないでください。【Windows 限定】
- ファイル名やパス名に UNC 形式の名称は使用しないでください。【Windows 限定】

2.3.5 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。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 2-8 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	UTF-8
AIX	Ja_JP ja_JP	Shift-JIS EUC

（凡例）
－：該当しません。

2.3.6 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

2.3.7 環境情報の設定での注意事項

環境情報の設定では，次の点に注意してください。

- 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。

2.4 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。なお，JP1 関連の製品を使用しない場合は，3. だけの作業が必要になります。必要に応じてインストールしてください。

1. **運用管理サーバに** JP1/Base **および** JP1/AJS - Manager **のインストールとセットアップをする。**
2. **バッチ業務サーバに** JP1/Base **および** JP1/AJS - Agent **のインストールとセットアップをする。**
3. **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。**
   Windows 環境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【Windows 限定】」を参照してください。
4. **運用管理端末に** JP1/AJS - View **のインストールとセットアップをする。**
5. **運用管理端末に** JP1/Advanced Shell - Custom Job **をインストールする。**
   カスタムジョブのインストールについては，「2.4.3　JP1/Advanced Shell - Custom Job をインストールする【Windows 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【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「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

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。

(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
  デフォルトでは「**システムドライブ**¥Program Files¥Hitachi¥JP1AS」以下の JP1ASE または JP1ASD フォルダにインストールされます。

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

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。

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。

（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. 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，［プログラムの保守］を選択する。**
5. **［プログラムの保守］で［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。

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

JP1/Advanced Shell をアンインストールする手順を次に示します。インストールと同じく JP1/NETM/DM を使ってアンインストールすることもできます。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. 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。**
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，［プログラムの保守］を選択する。**
5. **［プログラムの保守］で［削除］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。**
   アンインストールが完了します。
7. **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。**

2.4.3 JP1/Advanced Shell - Custom Job をインストールする【Windows 限定】

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** 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：**JP1/Advanced Shell のインストール先フォルダ** ¥JP1ASE¥util¥JP1ASVC.msi
   - UNIX の場合：/opt/jp1as/util/JP1ASVC.msi
3. JP1/Advanced Shell - Custom Job **のインストーラ（**JP1ASVC.msi**）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する。**
   - Windows 7，Windows Server 2008 または Windows Vista の場合：Windows の［スタート］メニューから［すべてのプログラム］－［アクセサリ］－［コマンドプロンプト］を右クリックし，［管理者として実行］を選択します。
   - Windows Server 2003，Windows XP の場合：Windows の［スタート］メニューから［すべてのプログラム］－［アクセサリ］－［コマンドプロンプト］を選択します。
5. **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。**
   msiexec /i JP1ASVC.msi
6. **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。**
   - 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。
     デフォルトでは「**システムドライブ** ¥Program Files¥Hitachi¥JP1AS」以下の JP1ASV フォルダにインストールされます。
7. **［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。
8. JP1/AJS3 - View 09-50 **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。**
   カスタムジョブアイコンのフォルダについては，「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。
   JP1/AJS - View インストール先フォルダ ¥image¥custom

(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) 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：**JP1/Advanced Shell のインストール先フォルダ** ¥JP1ASE¥util¥JP1ASVC.msi
   - UNIX の場合：/opt/jp1as/util/JP1ASVC.msi
3. JP1/Advanced Shell - Custom Job **のインストーラ（**JP1ASVC.msi**）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する。**
   - Windows 7，Windows Server 2008 または Windows Vista の場合：Windows の［スタート］メニューから［すべてのプログラム］－［アクセサリ］－［コマンドプロンプト］を右クリックし，［管理者として実行］を選択します。
   - Windows Server 2003，Windows XP の場合：Windows の［スタート］メニューから［すべてのプログラム］－［アクセサリ］－［コマンドプロンプト］を選択します。
5. **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。**
   msiexec /i JP1ASVC.msi
6. **［プログラムの保守］から［修復］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。**
   修復インストールが完了します。

2.4.4 JP1/Advanced Shell - Custom Job をアンインストールする【Windows 限定】

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 09-50 **より古い製品がインストールされている場合，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 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。**

2.5　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について説明します。UNIX 環境では，実行環境だけを UNIX マシン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. **運用管理サーバに** JP1/Base **および** JP1/AJS - Manager **のインストールとセットアップをする。**
2. **バッチ業務サーバに** JP1/Base **および** JP1/AJS - Agent **のインストールとセットアップをする。**
3. **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，「2.5.1　JP1/Advanced Shell をインストールする【UNIX 限定】」を参照してください。
4. **運用管理端末に** JP1/AJS - View **のインストールとセットアップをする。**
   JP1/AJS - View は，Windows 環境の運用管理端末にインストールできます。

2.5.1　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「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※
   ```

   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 05-08

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

    Select Procedure ===>
```

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-9S12-B111           0950   JP1/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。

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

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。インストールと同じく JP1/NETM/DM を使ってアンインストールすることもできます。

JP1/Advanced Shell をアンインストールする場合は，ジョブコントローラを利用するプログラムをすべて終了してください。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 05-08

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

    Select Procedure ===>
```

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-9S12-B111          0950   JP1/Advanced Shell
:
:
F) Forward B) Backward J) Down K) Up Space) Select/Unselect D) Delete 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。

```
Delete PP? (y: delete, 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。

```
/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

2.5.3 バージョン情報を表示する【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。

2.6　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，必要があれば 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。また，JP1/Advanced Shell で必要なディレクトリとファイルをデフォルトのもの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。

次に，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，「10.4 資料の採取方法」を参照してください。

環境情報を設定した場合，バッチジョブを実行するときに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。

2.6.1　環境ファイルを設定する

「表 2-10　JP1/Advanced Shell で必要なディレクトリ」で使用するディレクトリに対して，環境ファイルの環境設定パラメーターを設定します。終了コードなどのほかの項目も設定できます。環境ファイルの設定の詳細については，「7.　環境ファイルで設定するパラメーターとコマンド」を参照してください。

環境ファイルを使用してバッチジョブを実行する場合，バッチジョブを実行する前に環境ファイルをシステムの環境に合わせて作成します。次に示す手順で環境ファイルを作成して設定します。

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・ファイル名にコピーする。**
   - 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. **コピーした環境ファイルに必要なパラメーターを定義する。**
   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とコマンド」に記載していますので参照して定義してください。また，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。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，「2.3.5 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

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に変換するパラメーターを定義します。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 2-9 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 または /	/
パス区切り文字	;	:
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。

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 名（例 ¥¥ マシン名 ¥）は，サポートしていません。

上記の規則によって，Windows 環境でジョブ定義スクリプトを実行させたい場合は，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は，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。
  Windows 環境で「/」と「:」を定義します。UNIX 環境で「¥¥」と「;」を定義します。
  PATH_CONV_ENABLE パラメーターでは，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。
- PATH_CONV パラメーター：絶対パスのパス名を変換する規則を定義します。
  ジョブ定義スクリプトを実行するときに先頭のパス名 1 をパス名 2 に置換します。
  パス名が変換される可能性があるのは，ジョブ定義スクリプトのダブルクォーテーションで囲まれた範囲だけです。規則に合致した文字列を含んだ，ダブルクォーテーションで囲まれた文字列の中に，パス区切り文字およびディレクトリ区切り文字が含まれる場合，それらについても変換されます。

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。

(1) 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 /home/hitachi/bin "C:¥¥Program Files"
#-adsh_conf PATH_CONV /tmp "C:¥¥temp"
```

(2) 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です。

```
#-adsh_path_var DIR,DIR2
"/home/hitachi/bin/myprog1" "/tmp/file"

DIR="/home/hitachi/bin"
"$DIR/myprog1" "/tmp/file"
DIR2=$DIR
"$DIR2/myprog2" "/tmp/file"
```

(3) 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"

DIR="C:¥¥Program Files"
"$DIR¥¥myprog1" "C:¥¥temp¥¥file"
DIR2=$DIR
"$DIR2¥¥myprog2" "C:¥¥temp¥¥file"
```

(4) 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ます。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，「5.1.5　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【Windows，Linux 限定】

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。

Windows または Linux で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。

(1) 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
```

(2) 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 < "D:¥tmp¥input.txt"
```

(3) 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"
```

(4) 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では，ファイルへの入出力が発生するタイミングで該当するファイルパスと変換規則に指定されたファイルパスが完全に一致した場合，変換後のファイルパスに変換します。JP1/Advanced Shell では，次のタイミングでファイルへの入出力が発生します。

- リダイレクト文字（<，>，<>，>>）でファイルへ入出力する※1
- .（ドット）コマンドでジョブ定義スクリプトを実行する※2
- #-adsh_script でジョブ定義スクリプトを実行する※2

注※1

次の項目については，指定したリダイレクト文字の指定に従って動作します。

- ファイルパスを変換したあとのファイルの有無
- ファイルを作成するかどうか
- ファイルに追加書き込みするかどうか
- ファイルを上書きするかどうか

注※2

. コマンドおよび #-adsh_script コマンドの引数に指定されたジョブ定義スクリプトについても，入出力が発生します。しかし，これらのジョブ定義スクリプトは，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定義規則を指定してください。

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，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

例えば，UNIX 環境を想定して作成されたジョブ定義スクリプトを，UNIX と Windows の両方で実行できます。逆に，Windows 環境を想定して作成されたジョブ定義スクリプトを，Windows と UNIX の両方で実行できます。

(5) PATH_CONV パラメーターとの組み合わせ例

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"

(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"
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。そのため，行番号 3 の定義には合致しません。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
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 の定義には合致しません。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【Windows，Linux 限定】

コマンド（組み込みコマンド，拡張コマンド，関数，エイリアス，および外部コマンド，ユーザープログラム）を実行するタイミングで変換します。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ます。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。

コマンドの実行時にコマンドの引数を変換の定義に従って変換します。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

（1） PATH_CONV パラメーターとの組み合わせ例

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"

(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
cat コマンドに指定した "/tmp/data.txt" が，"（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。そのため，行番号 3 の定義に合致しません。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
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 の定義には合致しません。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【Windows，Linux 限定】

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することで，adshexec コマンドを使用して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。

- UNIX 環境から Windows 環境にユーザーの既存資産のシェルスクリプトを移行する場合
- 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

ルートジョブと子孫ジョブの詳細については，「3.3.1(1)　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，「3.3.3　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。

(1) 子孫ジョブの起動に必要な環境設定パラメーター

ジョブ定義スクリプトファイルを直接指定して子孫ジョブを起動させる場合，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_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

注意事項

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

Windows と UNIX の両方で，UNIX 互換コマンドを使用したジョブ定義スクリプトを共用したい場合は，パス名を変換する機能を有効にして，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を定義しておく必要があります。UNIX 互換コマンドを使用する方法については，「3.2　UNIX 互換コマンドの使用方法」を参照してください。

補足

ジョブ定義スクリプトを共用する必要がない場合や既存のジョブ定義スクリプトを使用する場合は，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下記の定義を省略できます。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。

(1) Windows の場合の例

環境ファイルを次のように設定します。export コマンドは 2 行で表示されていますが，実際には 1 行です。

```
#-adsh_conf PATH_CONV_ENABLE / :
export ADSH_OSCMD_DIR="C:¥¥Program Files¥¥Hitachi¥¥JP1AS
¥¥JP1ASE¥¥cmd"
```

1. **環境ファイルでパス変換機能を有効にする。**
2. **環境ファイルで** UNIX **互換コマンドがインストールされているフォルダを環境変数（例：** ADSH_OSCMD_DIR**）に定義する。**

（2） UNIX の場合の例

環境ファイルを次のように設定します。

```
#-adsh_conf PATH_CONV_ENABLE ¥¥ ;
export ADSH_OSCMD_DIR=/opt/jp1as/cmd
```

1. **環境ファイルでパス変換機能を有効にする。**
2. **環境ファイルで** 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を環境変数（例：** ADSH_OSCMD_DIR**）に定義する。**

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。

- -h **file**：ファイルに対してシンボリックリンクを使用します。
- -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。
- -L **file**：ファイルに対してシンボリックリンクを使用します（-h の場合と同じ）。
- -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。
- **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）であることを確認します。

2.6.8 システム実行ログを出力する

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。

2.6.9 スプールを定義する

スプール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：スプールディレクトリのパス名を定義します。

2.6.10 トレースを定義する

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。
- 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。
- 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。

2.6.11 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。

2.6.12 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

2.6.13 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。

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-*
```

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。

2.6.14 UNIX のジョブ定義スクリプトから Windows のジョブ定義スクリプトに移行する

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。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。下記の指定は実際には 1 行です。

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

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ジョブ定義スクリプトごとに次のコマンドを追加します。

   ```
   #-adsh_path_var VAR
   ```

4. **変換が有効であることを確認する。**
パスを変換したい個所が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/ls" -l "/opt/jp1as/cmd/date"
```

2.6.15 シェル変数 ENV を読み込む

ジョブコントローラ起動時にシェル変数 ENV 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 の場合

2.6.16 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。

2.6.17 シェルを設定する【UNIX 限定】

(1) Linux の場合

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，ジョブ定義スクリプトファイルで指定するシェルは bash だけ使用できます。/etc/passwd，ログインスクリプトファイルなど，bash が使用できるように設定してください。

(2) AIX の場合

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，ジョブ定義スクリプトファイルで指定するシェルは ksh だけ使用できます。/etc/passwd，ログインスクリプトファイルなど，ksh が使用できるように設定してください。

2.6.18 JP1/Advanced Shell で必要なディレクトリとファイルを作成する

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 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
  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
  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
  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
  システム障害時に，トラブルシュート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。

表 2-10 JP1/Advanced Shell で必要なディレクトリ

ディレクトリ	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	TEMP_FILE_DIR	●実行環境【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥temp ●開発環境【Windows 限定】 **全ユーザー共通文書フォルダ** ¥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	SPOOL_DIR	●実行環境【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥spool ●開発環境【Windows 限定】 **全ユーザー共通文書フォルダ** ¥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	LOG_DIR LOG_FILE_CNT LOG_FILE_SIZE	●実行環境【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥log ●開発環境【Windows 限定】 **全ユーザー共通文書フォルダ** ¥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	TRACE_DIR TRACE_FILE_CNT TRACE_FILE_SIZE TRACE_LEVEL	●実行環境【Windows 限定】 **共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥trace ●開発環境【Windows 限定】 **共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥trace ●カスタムジョブ【Windows 限定】 **共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASV¥trace ●実行環境【UNIX 限定】 /opt/jp1as/trace	CRWD【Windows 限定】 1777【UNIX 限定】
環境ファイル	なし。	環境変数 ADSH_ENV に設定した環境ファイルのパス	R【Windows 限定】 0444【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 2-11 ディレクトリのファイルパーミッション

ディレクトリ	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	○	○	○	△
スプール	○	○	○	△
システム実行ログ	○	○	○	×
トレース	○	○	○	△

（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のが望ましいです。また，JP1/Advanced Shell では NFS および Hitachi Striping File System（HSFS）はサポートしません。

2.6.19 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/Advanced Shell を使用した場合は，JP1/Advanced Shell を使用しない場合に比べて，次の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。

ジョブ実行内部ログ

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

そのため，ログサイズを見積もる場合には，JP1 のマニュアルに記載されている計算式に上記の値を追加してください。

2.7　JP1/AJS - View でカスタムジョブを登録する

JP1/AJS でジョブの実行を自動化する場合，JP1/AJS - View でカスタムジョブを登録します。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」または「JP1/Automatic Job Management System 2 操作ガイド」を参照してください。

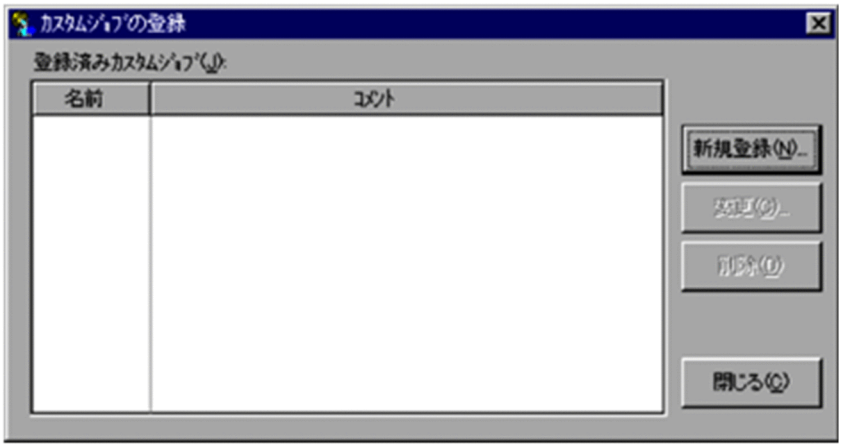
JP1/AJS - View では，運用に使用するコマンド，バッチファイルなどをジョブとして定義して，それぞれのジョブの実行順序を関連づけることで，システム運用を自動化します。ジョブを定義する場合にカスタムジョブを使用すると，直接コマンドやバッチファイルをジョブに指定するのに比べて，容易で間違いなく定義できます。

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。なお，カスタムジョブは Windows 環境で登録できますが，UNIX 環境のジョブを登録する場合は，PC ジョブとして登録して UNIX 専用に設定することで使用できます。

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］ー［**JP1_Automatic Job Management System 3 - View**］ー［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。**Windows **と** UNIX **の場合でそれぞれ次のようなイメージで登録する。**
   - 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。

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。

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，次の方法で設定してください。
マニュアル「JP1/Automatic Job Management System 3 構築ガイド 1」または「JP1/Automatic Job Management System 2 セットアップガイド」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。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。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。

2.8　ジョブネットを定義して実行する

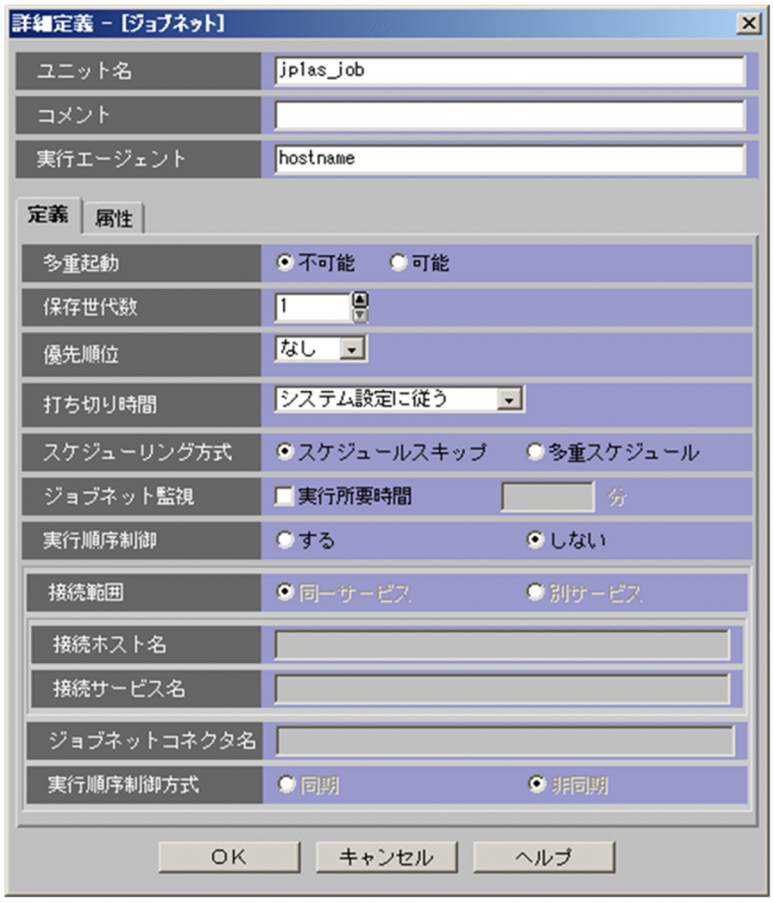
JP1/AJS でジョブの実行を自動化する場合，登録したカスタムジョブを JP1/AJS - View でジョブネットに定義し，ジョブネットを実行します。JP1/AJS - View の詳細については，マニュアル「JP1/Automatic Job Management System 3 操作ガイド」または「JP1/Automatic Job Management System 2 操作ガイド」のジョブ定義の説明を参照してください。なお，JP1/AJS で使用するアイコンの説明を次に示します。

表 2-12　JP1/AJS で使用するアイコンの説明

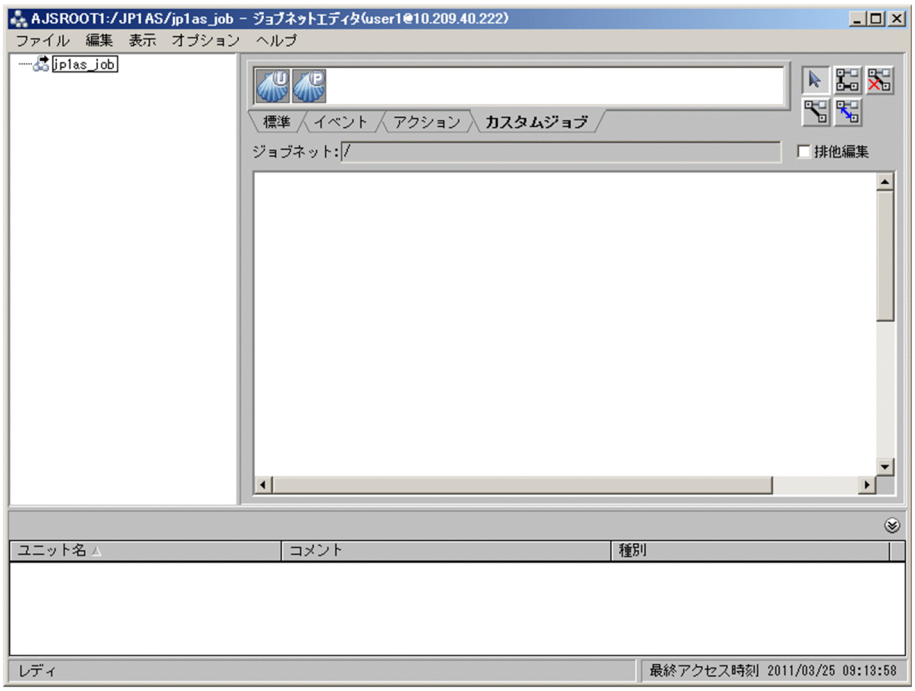
アイコン	名称	ファイル名	説明
	Windows で実行されるジョブアイコン	CUSTOM_PC_ADSHPC.gif	JP1/AJS - View のジョブネットエディタ上で使用する Windows のカスタムジョブのアイコンです。［ジョブネットエディタ］ウィンドウでの［カスタムジョブ］タブで表示されます。
	UNIX で実行されるジョブアイコン	CUSTOM_PC_ADSHUX.gif	JP1/AJS - View のジョブネットエディタ上で使用する UNIX のカスタムジョブのアイコンです。［ジョブネットエディタ］ウィンドウでの［カスタムジョブ］タブで表示されます。
	JP1/Advanced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

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。
   ジョブネットが作成され，リストエリアに表示されます。



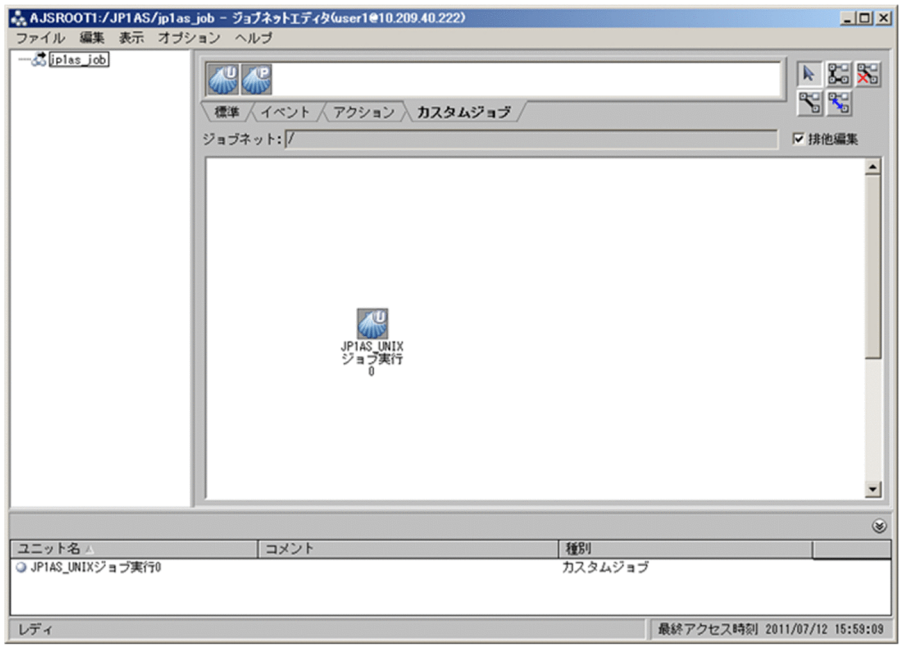
5. **作成したジョブネットをダブルクリックする。**
   ［ジョブネットエディタ］ウィンドウが表示されます。



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。**

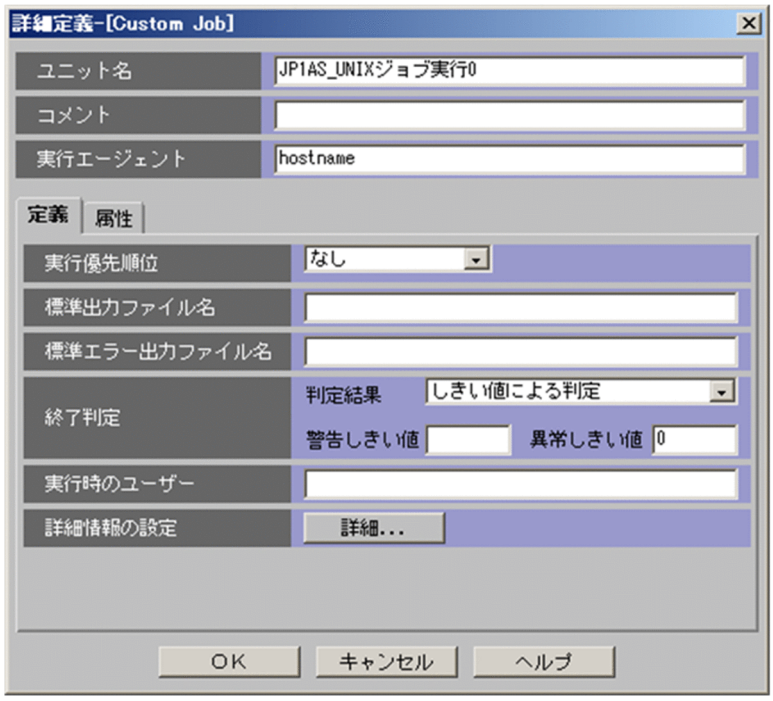


［詳細定義 -［Custom Job］］ダイアログボックスが表示されます。

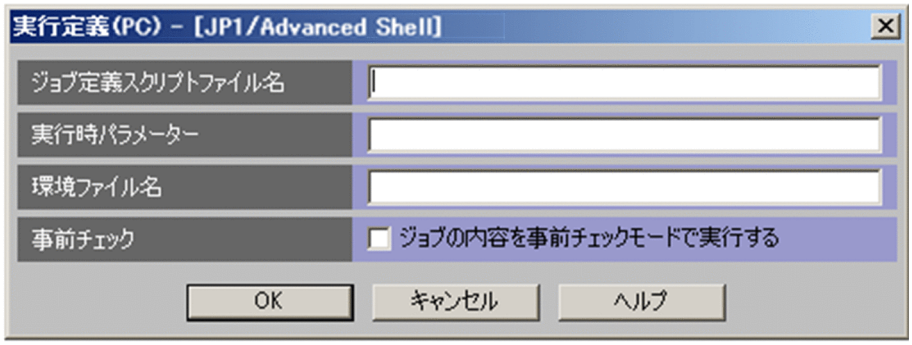
8. **［詳細定義 -［Custom Job］］ダイアログボックスでジョブに属性などを定義する。**

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。

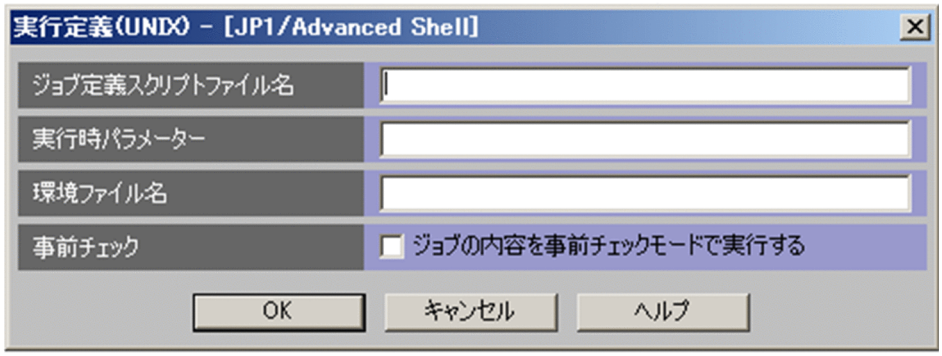
なお，Windows または Linux で，adshexec コマンドの -s オプションおよび環境ファイルの OUTPUT_STDOUT パラメーターに SPOOL を指定して，定義タブで［標準出力ファイル名］を指定したときは，空のファイルが出力されます。標準出力の内容は JP1/Advanced Shell のジョブコントローラで別ファイルに出力していて，JP1/AJS に返す標準出力には何も出力していないためです。



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 と UNIX の場合の表示を次に示します。
   - Windows の場合



   - UNIX の場合



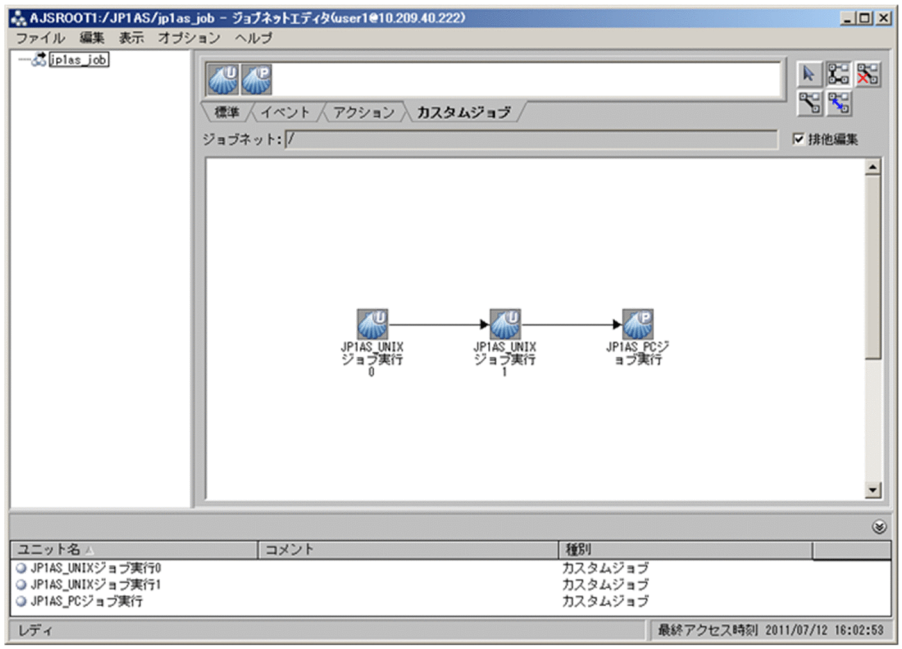
10. **［実行定義］ダイアログボックスに JP1/Advanced Shell の実行に必要な情報を設定して［OK］ボタン**

をクリックする。

- 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 ～ 1023 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 ～ 1022 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。
- 環境ファイル名～＜パス名＞ Windows の場合：((0 ～ 247 バイト))，UNIX の場合：((0 ～ 1023 バイト))
  環境ファイル名を指定します。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，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。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。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，環境ファイルの指定はないものとして実行します。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。子孫プロセスとして別のジョブコントローラを起動する場合，それらのジョブコントローラは環境変数 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，再設定した値の環境ファイルを使用します。
- 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。

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，JP1/AJS にジョブの終了コードとして返します。

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 1」または「JP1/Automatic Job Management System 2 セットアップガイド」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。

補足

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 2」または「JP1/Automatic Job Management System 2 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，「JP1/Automatic Job Management System 3 - Definition Assistant」または「JP1/Automatic Job Management System 2 - Definition Assistant」を参照してください。

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。

- 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：環境ファイル名
  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 2-13　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
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 2-14　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_GCHE	0 ～ 2 バイト	-c, 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
  次の実行時パラメーターを，ユニット定義ファイルの prm パラメーターとして定義します。
  実行時パラメーター：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

注意事項

- チェックボックスの値
  ADSH_AJS_GCHE（事前チェック）では，チェックありの場合は，"-c" を，チェックがない場合は空文字列を指定してください。
- 実行時パラメーターの値
  実行時パラメーターの情報はユニット定義の prm で指定し，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。
  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。
  また，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になっています。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。
- 定義情報の削除
  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。また，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_GCHE=-c";                                    →5.
        tho=0;
}
```

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：**環境ファイル名を指定します。**
5. ADSH_AJS_GCHE：**事前チェックを実施する場合，**-c **を指定し，実施しない場合，何も指定しません。**

ユニット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。

2.9 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** FIGURE **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

3 バッチジョブの実行

JP1/Advanced Shell（実行環境）を使用して，バッチジョブの運用を実施します。この章では，バッチジョブの実行方法やバッチジョブでの動作について説明します。

3.1 利用者の役割

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。

3.1.1 システム管理者

システム管理者は，システム運営上の責任者で，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。

3.1.2 一般ユーザー

JP1/Advanced Shell を使用する一般ユーザーは，運用者と開発者に分けられます。運用者は，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。開発者は，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，「2.8　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」または「JP1/Automatic Job Management System 2 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。

3.2 UNIX 互換コマンドの使用方法

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。

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，環境ファイルを設定しておく必要があります。UNIX 互換コマンドを使用するための環境ファイルの設定については，「2.6.6　UNIX 互換コマンドを使用する」を参照してください。

環境ファイルで 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として ADSH_OSCMD_DIR を設定した場合，次のように date コマンドを使用できます。

```
"$ADSH_OSCMD_DIR/date"
```

ジョブ定義スクリプトで cp コマンドの引数にパスを使用する場合は，#-adsh_path_var コマンドでパスの変数を定義してディレクトリ区切り文字を変換することで対応できます。

```
#-adsh_path_var ADSH_OSCMD_DIR,ADSH_OSCMD_PATH
"$ADSH_OSCMD_DIR/cp" -R "${ADSH_OSCMD_PATH}dir1/dir2" dir3
```

3.3 バッチジョブの実行方法

バッチジョブの実行方法について説明します。

3.3.1 ジョブとジョブステップ

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) ジョブ

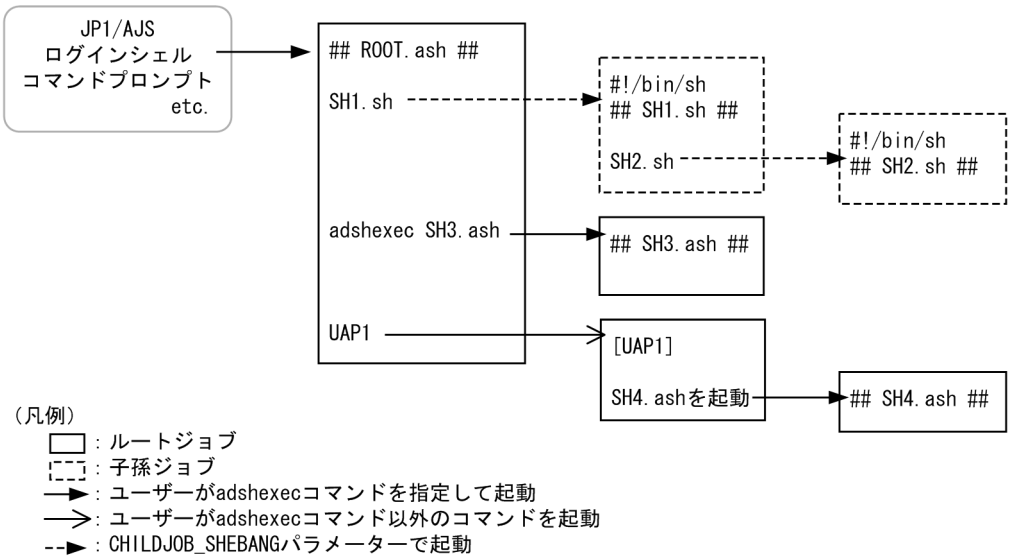
ジョブとは一般に，一般ユーザーが用意する 1 つのまとまった仕事をシステムに要求する単位です。要求する仕事は互いに独立したものと考えられます。

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，CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）によって実行されたジョブのことです。

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。



（凡例）
- □：ルートジョブ
- ┆┆：子孫ジョブ
- ——►：ユーザーがadshexecコマンドを指定して起動
- ——>：ユーザーがadshexecコマンド以外のコマンドを起動
- --►：CHILDJOB_SHEBANGパラメーターで起動

（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，ひとまとまりのコマンド群としてグループ化したものです。ジョブステップは資源を割り当てる単位になります。

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，処理のスキップもできます。

(3) ジョブとジョブステップの概念

ジョブとジョブとは互いに独立しています。したがって，同時に処理するジョブ同士，または先に実行したジョブがあとに実行する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，ジョブから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，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。

(4) 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 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。

3.3.2 バッチジョブの実行方法の種類

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のシェル変数・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てユーザープログラムから参照できるようにしたりする。**

3.3.3 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について説明します。その他のコマンドとの実行方法の比較については，「5.1.9　外部コマンドの実行」を参照してください。

(1) 子孫ジョブの動作

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，CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）によって実行されたジョブを子孫ジョブと呼びます。

子孫ジョブを起動するには，ジョブ定義スクリプトに CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）で定義したファイルをコマンド名として指定します。

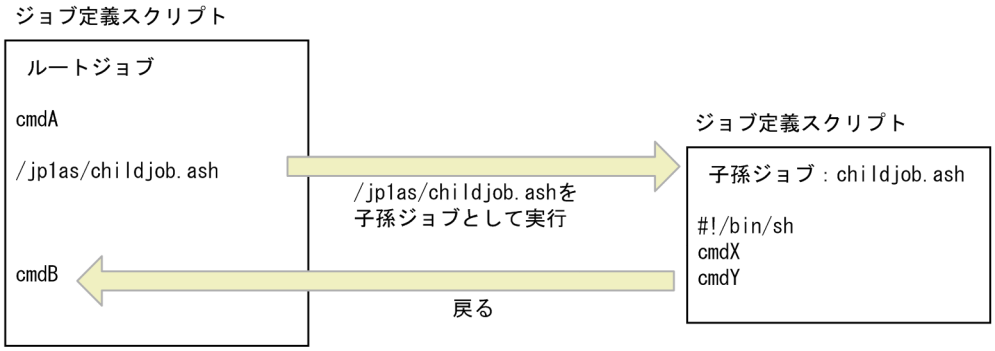
子孫ジョブを起動する場合のジョブ定義スクリプトの例を次に示します。

```
cmdA
/jp1as/childjob.ash  #コマンド名にほかのジョブ定義スクリプトを指定
cmdB
```

この例では，コマンド名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，JP1/

Advanced Shell 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。この場合の子孫ジョブを起動する動作の例を次の図に示します。

図 3-1 子孫ジョブを起動する動作の例



(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	.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。 ジョブ実行ログは JOBLOG だけを stderr に出力する	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 （KANX0091-I および KNAX0098-I）	あり （KANX6571-I および KNAX6578-I）	なし	なし
起動したコマンドを必ず wait するか	CHILDJOB_SHEBANG（Windows，Linux 限定）パラメーターの指定があれば，必ず wait する	CHILDJOB_SHEBANG（Windows，Linux 限定）パラメーターの指定があれば，必ず wait 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

機能	ジョブ		外部スクリプト	
	ルートジョブ	子孫ジョブ	. 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	しない（呼び出し元ジョブの動作に従う）	しない（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，および OUTPUT_STDOUT（Windows，Linux 限定）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) 子孫ジョブの非同期実行に関する注意事項

子孫ジョブの機能を使用（CHILDJOB_SHEBANG パラメーターを環境ファイルに 1 つ以上指定）した場合，関連するすべてのルートジョブと子孫ジョブが終了するまでは，ジョブは終了しません。そのため，& や |& による非同期実行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。

また，CHILDJOB_SHEBANG パラメーターを指定した場合と指定しない場合とで，exec コマンドの引数に外部コマンドを指定したときの adshexec コマンドの動作は次のように異なります。

● CHILDJOB_SHEBANG パラメーターを環境ファイルに一つも指定しない場合
- UNIX 版では，adshexec コマンドのプロセスを外部コマンドのプロセスで上書きして実行します。
- Windows 版では，外部コマンドを子プロセスとして実行し，完了を待たないで exec コマンド実行直前の終了コードでシェルスクリプトの実行を終了します。

● CHILDJOB_SHEBANG パラメーターを環境ファイルに 1 つ以上指定した場合
- 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。完了した外部コマンドの終了コードでシェルスクリプトの実行を終了します。

CHILDJOB_SHEBANG パラメーターを環境ファイルに 1 つ以上指定して adshexec コマンドを実行した場合，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，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。

(4) 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。

● JP1/AJS からの強制終了時

adshexec(1)	rc=143 でエラー終了

adshexec(2)	rc=143 でシグナルによるエラー終了
adshexec(3)	rc=143 でシグナルによるエラー終了
sleep	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

adshexec(1)	rc=143 でシグナルによるエラー終了
adshexec(2)	rc=143 でシグナルによるエラー終了
adshexec(3)	rc=143 でシグナルによるエラー終了
sleep	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

adshexec(1)	• 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
adshexec(2)	rc=143 でシグナルによるエラー終了
adshexec(3)	rc=143 でシグナルによるエラー終了
sleep	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(3) への SIGTERM 送信時

adshexec(1)	adshexec(2) の結果に従う
adshexec(2)	• 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
adshexec(3)	rc=143 でシグナルによるエラー終了
sleep	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(1)	adshexec(2) の結果に従う
adshexec(2)	adshexec(3) の結果に従う
adshexec(3)	• 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
sleep	rc=143 でシグナルによるエラー終了

3.4 バッチジョブを起動する

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。

3.4.1 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 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，「2.8　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。

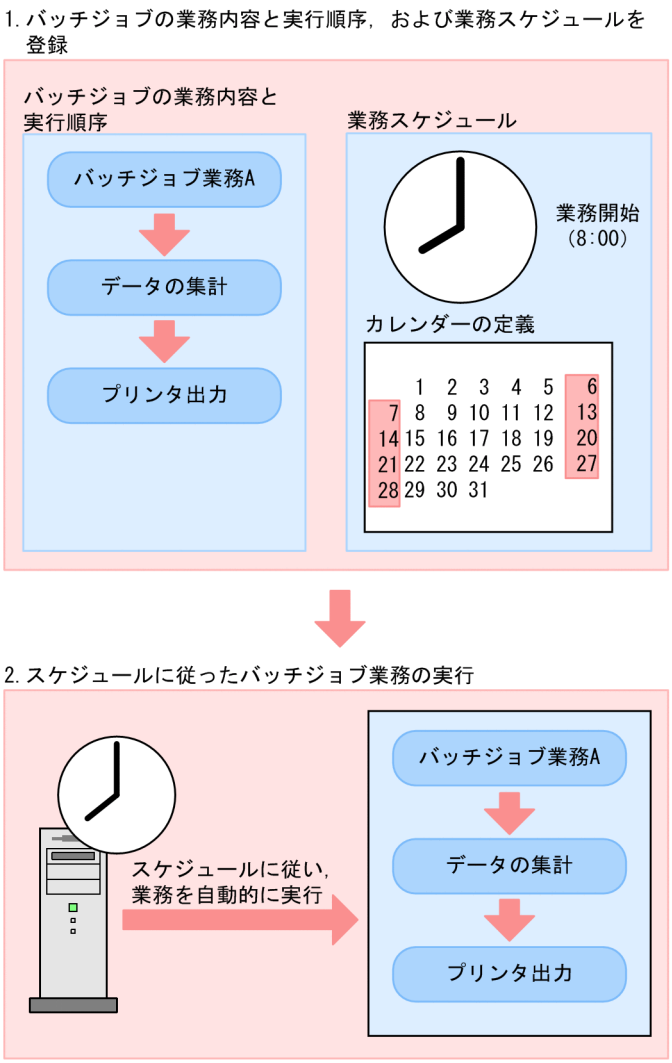
- 一時ファイル機能によって，一時的に使用するファイルを割り当てて，ジョブ終了時またはジョブステップ終了時に削除できます。
- 外部スクリプトの呼び出しによって，ジョブの定義を業務間で共用できます。
- ジョブ定義スクリプトに対する変更，追加，または削除によって，柔軟なジョブ定義ができます。

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 3-2　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。

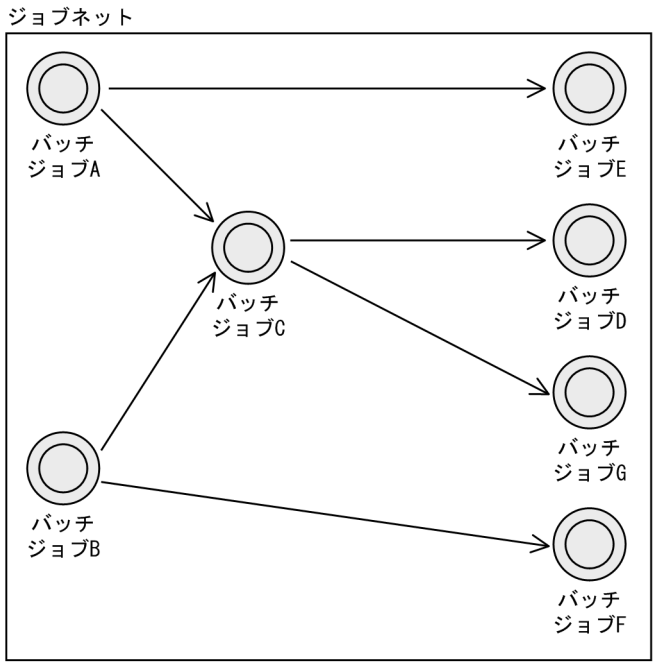
JP1/AJS だけでもコマンド，アプリケーションプログラム，またはジョブ定義スクリプトなどのそれぞれ

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 3-3　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。

3.4.2　実行環境からコマンドでバッチジョブを起動する

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 /script/batchjob1.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。

3.4.3　実行環境でバッチジョブをデバッグする【UNIX 限定】

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。

3.5 ジョブ定義スクリプトを解析して実行する

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. エディタの操作」および「5. ジョブ定義スクリプトの文法」を参照してください。

ジョブ定義スクリプトを解析して実行する機能は，次の 2 種類に分かれます。

- UNIX の標準的な機能を吸収したシェル標準コマンドの機能
- シェル運用コマンドおよびスクリプト拡張コマンドの機能

シェル標準コマンドの機能を次の表に示します。

表 3-1 シェル標準コマンドの機能

機能	説明
構文解析	ジョブ定義スクリプトに記述された構文を解析し，展開します。
条件判定	条件によって実行する処理に遷移するために必要な制御文および条件判定を行います。
算術演算	算術式を使って数値演算を実行します。
シェル変数	UNIX 標準で使用するシェル変数の中で，JP1/Advanced Shell で設定および使用できるシェル変数のことです。
シェルオプション	JP1/Advanced Shell で使用できるシェルのオプションのことです。
組み込みコマンド	JP1/Advanced Shell が提供する，ジョブ定義スクリプト内で使用できる組み込みコマンドです。
障害情報の出力	障害情報出力方法には，メッセージの出力とトレースの取得があります。 メッセージの出力とは，標準出力または標準エラー出力にメッセージを出力する機能です。トレースの取得とは，トレース情報を採取する機能です。

シェル運用コマンドおよびスクリプト拡張コマンドの機能を次の表に示します。

表 3-2 シェル運用コマンドおよびスクリプト拡張コマンドの機能

機能	詳細項目	説明
共通	拡張コマンドの文法	シェル運用コマンドおよびスクリプト拡張コマンドの文法を解析します。
	終了コード	ジョブコントローラおよびスクリプト拡張コマンドの終了コードを設定します。
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	通常ファイルのファイルパスをシェル変数および環境変数に割り当て，ジョブおよびジョブステップ終了時に後処理を実施します。
	一時ファイル	一時ファイルを作成し，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。ジョブおよびジョブステップ終了時に後処理を実施します。
	プログラム出力データファイル	出力データファイルのファイルパスを生成し，シェル変数および環境変数に割り当てます。

機能	詳細項目	説明
ジョブの実行を制御する	ジョブ定義スクリプトファイルの定義	ジョブ全体で有効な属性を定義します。
	ジョブ打ち切り条件の定義	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。
	ジョブステップの定義	ジョブ定義スクリプトの一部分を，コマンド群としてグループ化します。
	ジョブステップ内のローカルのシェル変数	ジョブステップ内で独立したシェル変数を定義します。
	終了コードを無視するコマンド	終了コードに関係なく常に正常終了とするコマンドを指定します。
	実行プログラムのパス追加	PATH 環境変数に，ディレクトリパスを追加します。
	外部スクリプトの展開	外部のジョブ定義スクリプトファイルの内容を，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。
ジョブ定義スクリプトで変数を使用する	ジョブステップ終了コードのシェル変数への設定	JP1/Advanced Shell がジョブステップの終了コードをシェル変数に格納します。
	ジョブ情報の環境変数の設定	JP1/Advanced Shell が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，ジョブステップに指定したユーザープログラムから参照できるようにします。
ジョブの実行ログを採取する	ジョブ・ジョブステップの実行状況出力（JOBLOG）	ジョブおよびジョブステップの開始・終了時にコマンド実行結果などのメッセージを出力します。
	ファイル準備・後処理結果の表示（JOBLOG）	ファイルの準備・後処理の結果をジョブ実行ログへ出力します。
	ジョブ定義スクリプト内容の表示	入力されたジョブ定義スクリプトファイルの内容をジョブ実行ログへ出力します。
	各ジョブステップの標準出力（stdout），標準エラー出力（stderr）	ユーザープログラムが出力した，標準出力や標準エラー出力のデータを，ジョブ実行ログとしてスプールに出力します。
ジョブを強制終了する	強制終了時の後処理	JP1/Advanced Shell が終了シグナルを受信（UNIX の場合）または taskkill コマンドなどによって強制終了した場合（Windows の場合），JP1/Advanced Shell は実行中の子プロセスを強制終了し，ファイルの後処理を実施します。
スクリプト拡張コマンドを使用する	バッチジョブを実行するコマンド（adshexec コマンド）	ジョブ定義スクリプトファイルを入力として読み込み，ジョブを実行するコマンドです。
環境ファイルを設定する	環境ファイルの解析	JP1/Advanced Shell の環境ファイルを解析し，文法チェックを行います。
	複数起動（複数環境）とクラスタ運用	JP1/Advanced Shell が動作する JP1 の論理ホストごとに，異なる環境ファイルを指定できます。
	環境ファイルの設定	JP1/Advanced Shell 起動時に，環境ファイルを読み込んで環境変数を一括して指定し，環境設定パラメーターを設定します。

機能	詳細項目	説明
カバレージ情報を使用する	カバレージ情報の蓄積	ジョブ定義スクリプト実行時にカバレージ情報を蓄積します。
	カバレージ情報の表示	蓄積されたカバレージ情報を表示します。
	カバレージ情報のマージ	2 つのカバレージ情報をマージします。
	カバレージ採取の一括有効化	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 または -d）を指定しなくても有効にします。

3.6 スプールのディレクトリ

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を，ジョブごとのディレクトリに出力します。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
└ジョブ識別子
    ├JOBLOG
    ├SCRIPT
    ├STDERR
    ├STDOUT
    ├ステップ番号_ステップ名_STDOUT
    ├ステップ番号_ステップ名_STDERR
    ├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
    ├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
    :
    └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
```

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で設定します。

ジョブ識別子

ジョブ通し番号と同じ名称のディレクトリで，ジョブ単位です。ジョブ終了時は「ジョブ識別子 - ジョブ名」に変更します。

JOBLOG

ジョブ実行メッセージです。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。

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。

STDERR

ジョブの標準エラー出力です。

STDOUT（Windows，Linux **限定**）

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。

ステップ番号 _ ステップ名 _STDOUT（Windows，Linux **限定**）

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。

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。

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指定した #-adsh_spoolfile に指定したファイルです。

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指定した #-adsh_spoolfile に指定したファイルです。

注意事項

- Windows の場合，出力したファイルの後ろに .sysout の拡張子が付きます。
- 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。

3.7 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磁気ディスクの空き容量を増やす必要があります。スプールジョブを削除するためには，次の adshhk コマンドを入力します。

```
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記載します。**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。**日数**で指定した日数より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，今日と昨日の 2 日間より前である，おとどい以前のスプールジョブを削除します。

この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"
"000057","JOB001","0","2011/06/13 09:04:11","2011/06/13 09:04:11","delete","","C:¥Documents
and Settings¥All Users¥Documents¥Hitachi¥jp1as¥jp1ase¥spool","15","2011/06/30 18:19:58"
"000201","ADSH000201","127","2011/06/13 09:52:25","2011/06/13 09:52:25","error","KNAX4419-E
I/O error. path=""C:¥Documents and Settings¥All
Users¥Documents¥Hitachi¥jp1as¥jp1ase¥spool¥000201-ADSH000201¥JOBLOG.sysout""
error=""Permission denied""","C:¥Documents and Settings¥All
Users¥Documents¥Hitachi¥jp1as¥jp1ase¥spool","15","2011/06/30 18:19:58"
"000203","jobname","0","2011/06/13 10:30:04","2011/06/13 10:30:05","delete","","C:¥Documents
and Settings¥All Users¥Documents¥Hitachi¥jp1as¥jp1ase¥spool","15","2011/06/30 18:19:58"
"000204","jobname","0","2011/06/13 10:30:47","2011/06/13 10:30:47","delete","","C:¥Documents
and Settings¥All Users¥Documents¥Hitachi¥jp1as¥jp1ase¥spool","15","2011/06/30 18:19:8"
```

（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。

jobid：ジョブ識別子

jobname：ジョブ名

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:mmm: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:mmm: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，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:mmm:ss の形式）

注意事項

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。

3.8 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。

3.8.1 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，コマンドを使用して，カバレージ情報を蓄積，表示またはマージできます。

1. カバレージ情報の蓄積：adshexec コマンド（-t オプション）
   ユーザーは，ジョブ定義スクリプト実行時にカバレージ情報を蓄積するオプションを指定して実行します。カバレージ情報を蓄積するオプションを指定して実行すると，カバレージ情報がカバレージ情報ファイルに蓄積されます。
2. カバレージ情報の表示：adshcvshow コマンド，JP1/Advanced Shell エディタ
   コマンドを使用して，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できます。また，エディタからもカバレージ情報を表示できます。
3. カバレージ情報のマージ：adshcvmerg コマンド
   このコマンドによって，一回の実行につき 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) Windows と UNIX でのカバレージ情報操作の相違点

次の表で示すように Windows と UNIX とでカバレージ情報の採取および表示方法に相違点があります。

表 3-3 Windows と UNIX でのカバレージ情報の採取および表示の相違点

環境	カバレージ情報の採取	カバレージ情報の表示
Windows の開発環境	エディタのデバッグ機能を使用してカバレージ情報ファイルに採取します。	• adshcvshow コマンドで表示します。 • デバッグ中にエディタから表示できます。
Windows の実行環境	• 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ファイルに採取します。 • 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。
UNIX の実行環境	• 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ファイルに採取します。 • adshexec コマンドでデバッグオプション（-d）を指定してメモリに採取します。 • adshexec コマンドでカバレージの蓄積オプション（-t）とデバッグオプション（-d）を指定してカバレージ情報ファイルおよび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。

カバレージ情報のマージについては，次の表のように Windows と UNIX で相違があります。

表 3-4 Windows と UNIX でのカバレージ情報のマージの相違点

環境	カバレージ情報のマージ
Windows の開発環境	• 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	adshcvmerg コマンドでマージします。
UNIX の実行環境	adshcvmerg コマンドでマージします。

（4） 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

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。
- Windows で採取したカバレージ情報ファイルは UNIX でのコマンドでは処理できません。
- Linux で採取したカバレージ情報ファイルは Windows および AIX でのコマンドでは処理できません。
- AIX で採取したカバレージ情報ファイルは Windows および Linux でのコマンドでは処理できません。

3.8.2 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。

以前に採取したカバレージ情報を失わないように，指定された asc ファイル（省略して標準の asc ファイル名の場合を含む）がすでに存在する場合，次のように asc ファイルを更新します。

- 指定された asc ファイル（省略して標準の asc ファイル名の場合を含む）を直接更新しない。
- 一時 asc ファイルに新しいカバレージ情報を出力する。
- 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。
- 一時 asc ファイルを，指定された asc ファイル（省略して標準の asc ファイル名の場合を含む）に変更する。
- 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，コマンドを再実行したとき，カバレージ情報を適切に回復して，コマンドの処理を続行します。

一時 asc ファイル，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，存在しません。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た場合，一時 asc ファイル，バックアップ asc ファイルが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構いません。

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（省略して標準の 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（省略して標準の 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。ファイル名は，次のようになります。ファイル名の先頭部分は，固定の文字列となります。

- adshexec コマンドの場合：adshexec_temp_ 任意

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。一時 asc ファイルを残しておいても，コマンドを再実行できます。または，一時 asc ファイルを手動で削除して再実行することができます。

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。adshexec コマンドで，残っているバックアップ asc ファイルを自動的に元の 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 3-5　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 3-6 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 3-7 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

3.8.3 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」でなくても構いません。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。

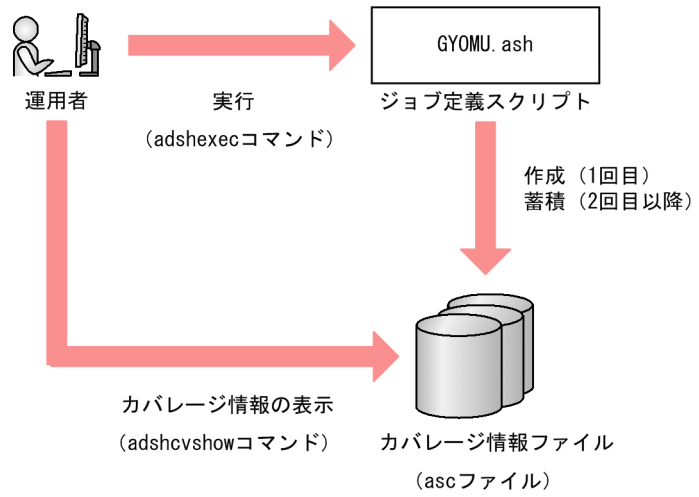
(6) 蓄積したカバレージ情報の初期化

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。

3.8.4 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 3-4　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 3-8 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（JP1/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。 • 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。

項目	説明
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。
<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。

```
                    * JP1/Advanced Shell Coverage Information *

                                                          2011-07-20 20:40:09
****    Header Information    ************************************************
Shellscript Name     : /home/test/sample
Asc version          : 1.0
Coverage Start Time  : 2011-07-20 20:40:09
Coverage End Time    : 2011-07-20 20:40:09
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
   37

****    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。

```
                    * JP1/Advanced Shell Coverage Information *

                                                                2011-07-22 18:23:31
****    Header Information    ************************************************
Shellscript Name     : /home/testlocal/all/sample1.ash
Asc version          : 1.0
Coverage Start Time  : 2011-07-22 18:23:31
Coverage End Time    : 2011-07-22 18:23:31
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
   75

****    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仕方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

次の条件に該当する #-adsh_step_start があると，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 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。

#-adsh_step_start は，次の両方の場合を C1 情報として取得します。

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。

しかし，#-adsh_step_start 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，上記の 2. のケースを実行できません。

3.8.5 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

3.8.6 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，「2.6.13　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。

3.9　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。

3.9.1　ジョブの強制終了の方法

(1) 強制終了の方法

ジョブの強制終了は，次の方法で行います。

- 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，「2.8　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孫プロセスに対して SIGTERM を送付（Windows の場合は，TerminateProcess 関数および TerminateJobObject 関数によるプロセス終了）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。このとき，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「3.9.2　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「3.9.3　強制終了時のジョブの動作【Windows 限定】」を参照してください。

なお，Windows で子孫プロセスを強制終了した場合，子孫プロセスのリターンコードは無条件に 1 になります。

注意事項

Windows の場合，adshexec コマンドの起動時に adshexecsub コマンドを合わせて起動します。adshexec コマンドを強制終了すれば，adshexecsub コマンドも終了します。
adshexecsub コマンドを強制終了しないでください。adshexecsub コマンドを強制終了した場合，実行中の子孫プロセスは終了されません。また，一時ファイルが残ったままとなる場合があります。このような場合は，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。

注意事項

Windows 環境の JP1/Advanced Shell 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。

- JP1/Advanced Shell 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。
- JP1/Advanced Shell のプロセスが，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 JP1/Advanced Shell の子プロセスが生成したプロセスは終了しません。

(2) Ctrl+C などの操作に関する注意事項【Linux 限定】

環境ファイルに CHILDJOB_SHEBANG パラメーターを指定して，ログインシェルからジョブを実行し

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※

CHILDJOB_SHEBANG パラメーターを指定している場合，adshexec コマンドのプロセスとその子プロセスとが，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。

3.9.2 シグナル受信時の動作【UNIX 限定】

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次の表に示します。

表 3-9 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	SIGTERM	【Linux の場合】 • ルートジョブが受信した場合 1 回目：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行ってから，後続コマンドを実行しないでシグナルによりエラー終了します。 2 回目：即時終了します。 • 子孫ジョブが受信した場合 受信した子孫ジョブは，受信メッセージ A を出力し，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りエラー終了します。 このとき，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 【AIX の場合】 1 回目：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行ってから，後続コマンドを実行しないでシグナルによりエラー終了します。 2 回目：即時終了します。	trap コマンドで動作を定義できません。
	SIGHUP，SIGINT，SIGXCPU，SIGXFSZ，SIGQUIT，SIGUSR1，SIGUSR2，SIGPIPE，SIGALRM，SIGVTALRM，SIGPROF	【Linux の場合】 • ルートジョブが受信した場合 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行ってから，後続命令を実行しないでシグナルによりエラー終了します。 • 子孫ジョブが受信した場合 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 【AIX の場合】 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行ってから，後続命令を実行しないでシグナルによりエラー終了します。	【Linux の場合】 • ルートジョブが受信した場合 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。 【AIX の場合】 trap コマンドで定義した動作に従います。
	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】

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
異常通知シグナル	SIGILL，SIGTRAP，SIGABRT，SIGFPE，SIGBUS，SIGSEGV，SIGSYS	【Linux の場合】 • ルートジョブが受信した場合 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 【AIX の場合】 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。	【Linux の場合】 • ルートジョブが受信した場合 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。 【AIX の場合】 trap コマンドで定義した動作に従います。
	SIGLOST，SIGIOT，SIGEMT	上記と同様です。【AIX 限定】	上記と同様です。【AIX 限定】
上記以外		【Linux の場合】 • ルートジョブが受信した場合 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 【AIX の場合】 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。	【Linux の場合】 • ルートジョブが受信した場合 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。 【AIX の場合】 trap コマンドで定義した動作に従います。※2

注※ 1

• シグナル受信時の動作の詳細については、「3.3.3(4)　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】

! 注意事項

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。

3.9.3 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。

表 3-10 強制終了時のジョブの動作

強制終了方法		ジョブの動作
制御信号	CTRL ＋ C	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）の動作 子プロセスの 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するため，制御信号を受信した adshexec.exe 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。
	CTRL ＋ BREAK	
	CTRL_CLOSE_EVENT	
	CTRL_LOGOFF_EVENT	システムの終了を優先して，後処理を実行しないで即時終了します。
	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 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。

4 エディタの操作

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］－［JP1_Advanced Shell - Developer］を選択する。**
2. **JP1_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 の終了方法を説明します。

JP1/Advanced Shell エディタウィンドウで，［ファイル］－［終了］を選択する，またはツールバーの［終了］ボタンをクリックします。

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）
- エディタの動作環境を設定する（4.4.2）
- ジョブ定義スクリプトの実行環境を設定する（4.4.3）
- 文法をチェックする（4.4.4）
- 文字列を検索および置換する（4.4.5）
- 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(1)）
- デバッグを実行・中止する（4.4.6(2)）
- ウォッチへ変数を追加する（4.4.6(3)）
- デバッグ実行時のカバレージ情報を表示する（4.4.7）
- 既存のジョブ定義スクリプトを編集する（4.5）
- 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※
- ステータスバーの表示・非表示を切り替える※
- ツールバーを整列させる※
- 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※
- ウォッチウィンドウの表示・非表示を切り替える※
- エラーウィンドウの表示・非表示を切り替える※
- ヘルプを表示する※

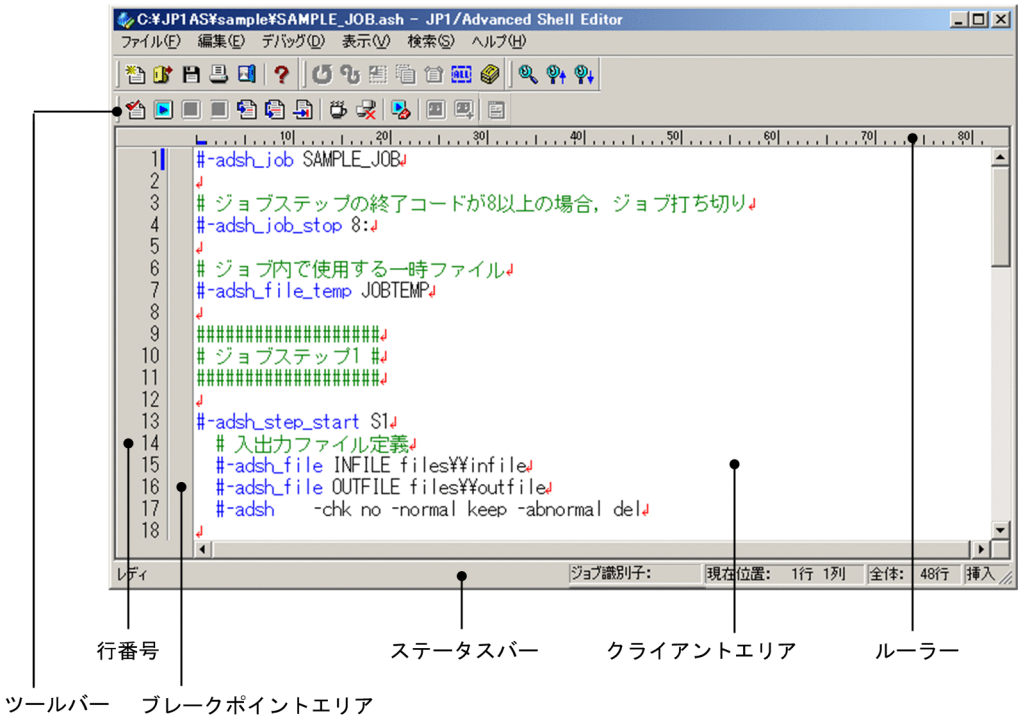
注※

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。

JP1/Advanced Shell エディタウィンドウのツールバーには，標準ツールバー，編集ツールバー，デバッグツールバーおよび検索ツールバーのボタンが表示されます。

- 標準ツールバー
  標準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。

- 編集ツールバー
  編集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
［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。

- デバッグツールバー
  デバッグ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
［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。
［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。
［ウォッチウィンドウの表示／非表示］ボタン	ウォッチウィンドウの表示・非表示を切り替えます。
［ウォッチへ追加］ボタン	指定した変数をウォッチウィンドウに追加します。
［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。

- 検索ツールバー
  検索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
［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。

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。

メニュー		機能
［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。
	［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。
	［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。
	［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］－［標準ツールバー］	標準ツールバーの表示・非表示を切り替えます。
	［ツールバー］－［編集ツールバー］	編集ツールバーの表示・非表示を切り替えます。
	［ツールバー］－［デバッグツールバー］	デバッグツールバーの表示・非表示を切り替えます。
	［ツールバー］－［検索ツールバー］	検索ツールバーの表示・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。
	［ツールバーの整列］	ツールバーを整列します。
	［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。

メニュー		機能
	［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。
	［ウォッチウィンドウを表示］	ウォッチウィンドウの表示・非表示を切り替えます。
	［エラーウィンドウを表示］	エラーウィンドウの表示・非表示を切り替えます。
［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。
	［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。
［ウォッチへ追加］	選択した変数をウォッチウィンドウへ追加します。

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+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。
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+0	×	○	指定した変数をウォッチウィンドウへ追加します。
Alt+1	×	○	ウォッチウィンドウの表示・非表示を切り替えます。
Alt+2	×	○	エラーウィンドウの表示・非表示を切り替えます。
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。

操作	編集モード	デバッグモード	機能
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

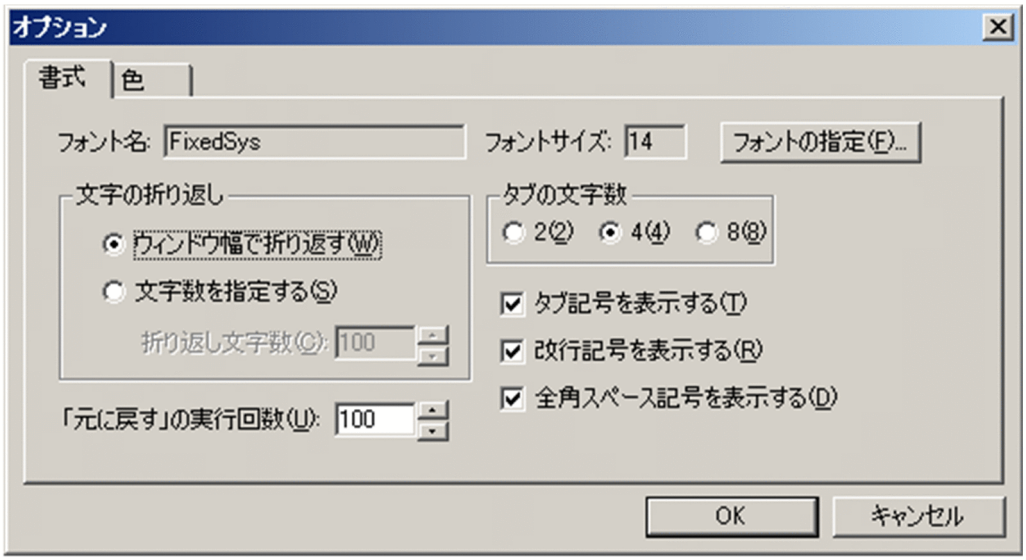
ジョブ定義スクリプトファイルを新規に作成します。

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

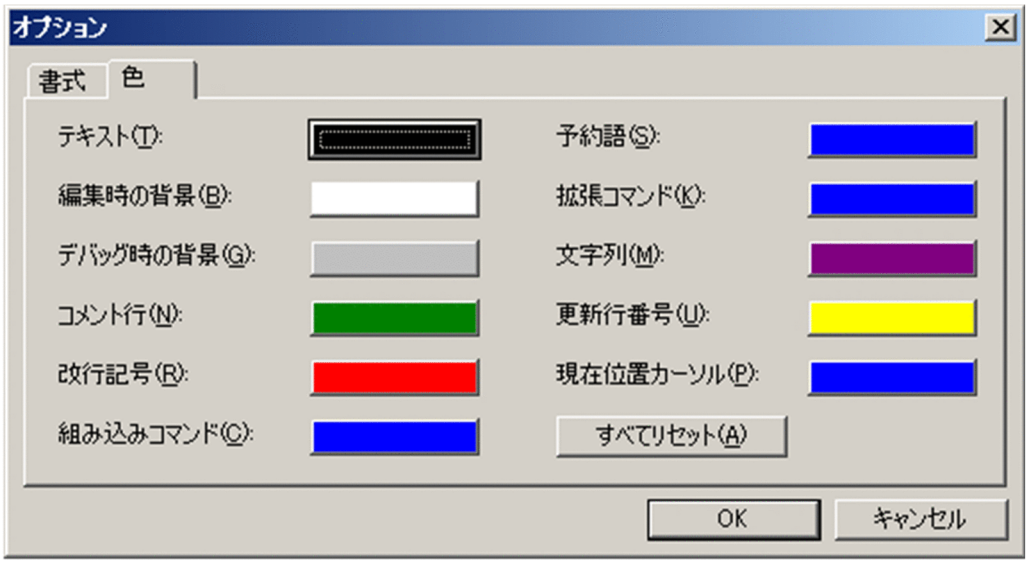
エディタの動作環境を設定します。

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。



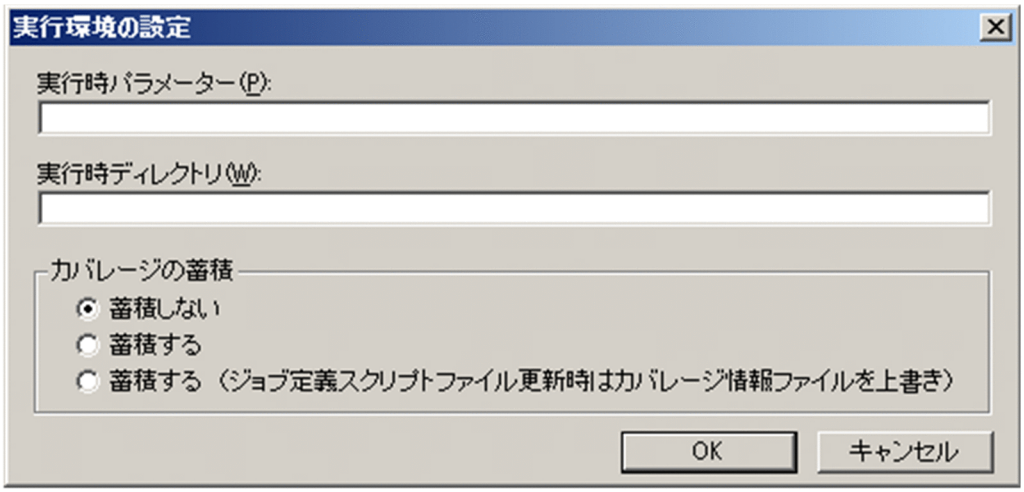
4. **表示色に関する情報を設定する。**
   すべての項目の表示色をデフォルトに戻すには［すべてリセット］ボタンをクリックしてください。
   ダイアログボックスの設定方法については，「4.7.2　オプション（色）ダイアログボックス」を参照してください。

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と実行時ディレクトリを設定できます。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。**

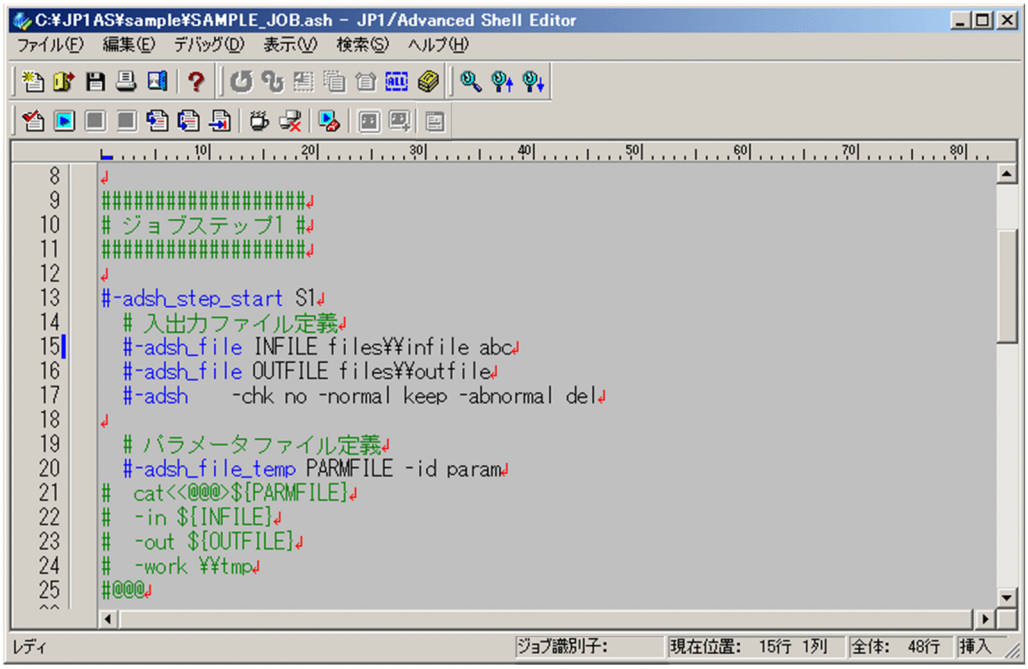
実行環境が設定され，ダイアログボックスが閉じます。

4.4.4 文法をチェックする

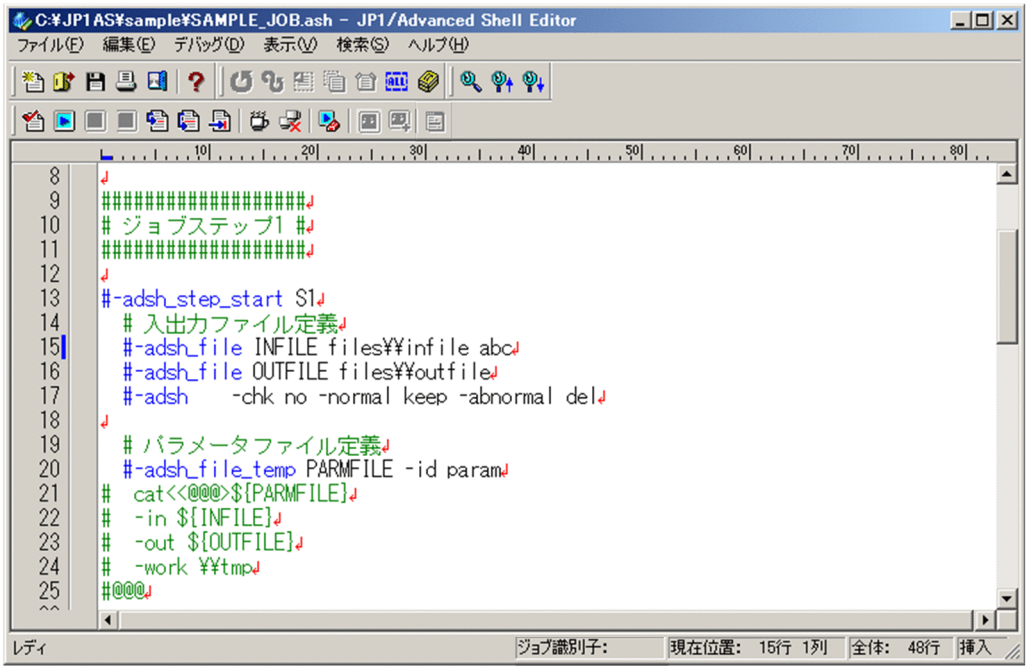
ジョブ定義スクリプトファイルの文法をチェックします。ジョブ定義スクリプトの文法が正しいかどうかのチェックだけ行い，実行しません。カバレージを蓄積するオプションを指定しても実行しないため，蓄積しません。adshexec コマンドの -c オプションの指定に相当します。

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。

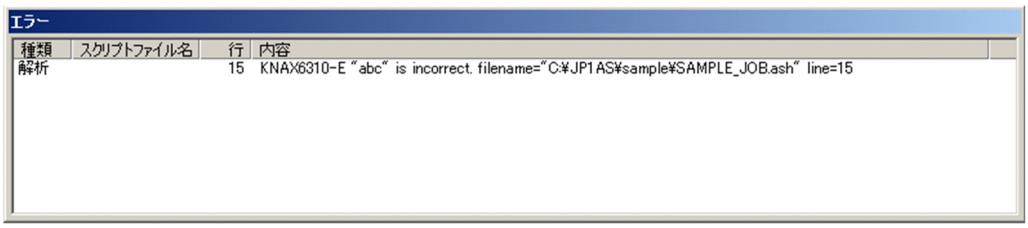
1. **［デバッグ］－［文法チェック］メニューを選択する。**
   エディタがデバッグモードになり，文法チェックが開始されます。
   文法チェック時は一瞬，画面が暗くなります。
   - **文法チェック中の表示**



   - **文法チェック終了の表示**
     文法エラーがある場合はエラーウィンドウにエラー内容が表示されます。



2. **エラーウィンドウの内容を確認する。**



エラーウィンドウについては，「4.7.4　エラーウィンドウ」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。

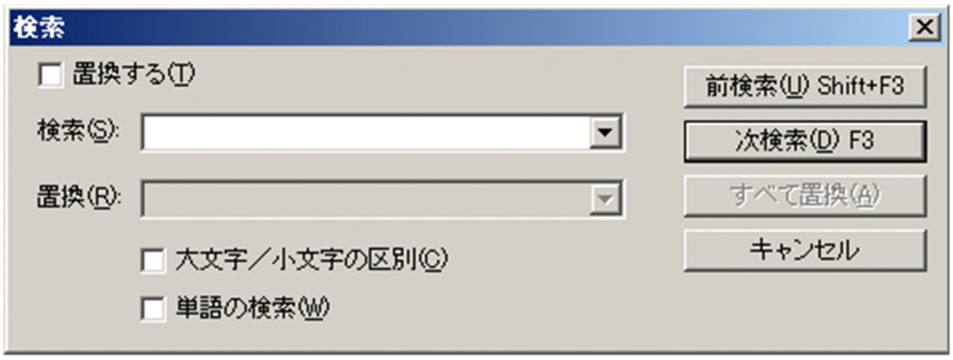
4.4.5　文字列を検索および置換する

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。

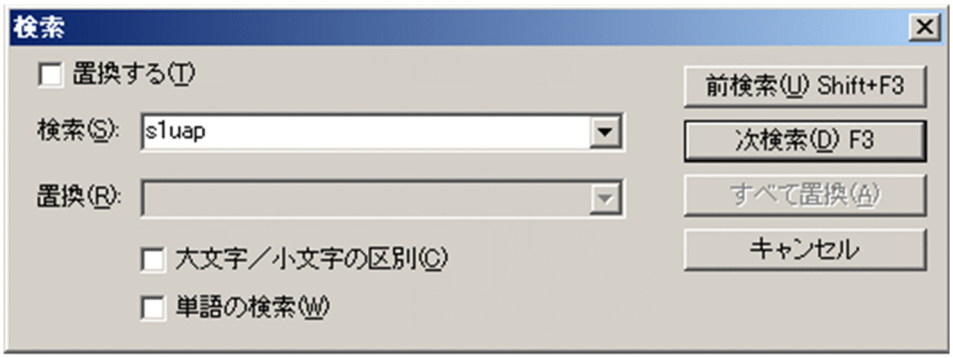
1. **［検索］－［検索］メニューを選択する。**
［検索］ダイアログボックスが表示されます。



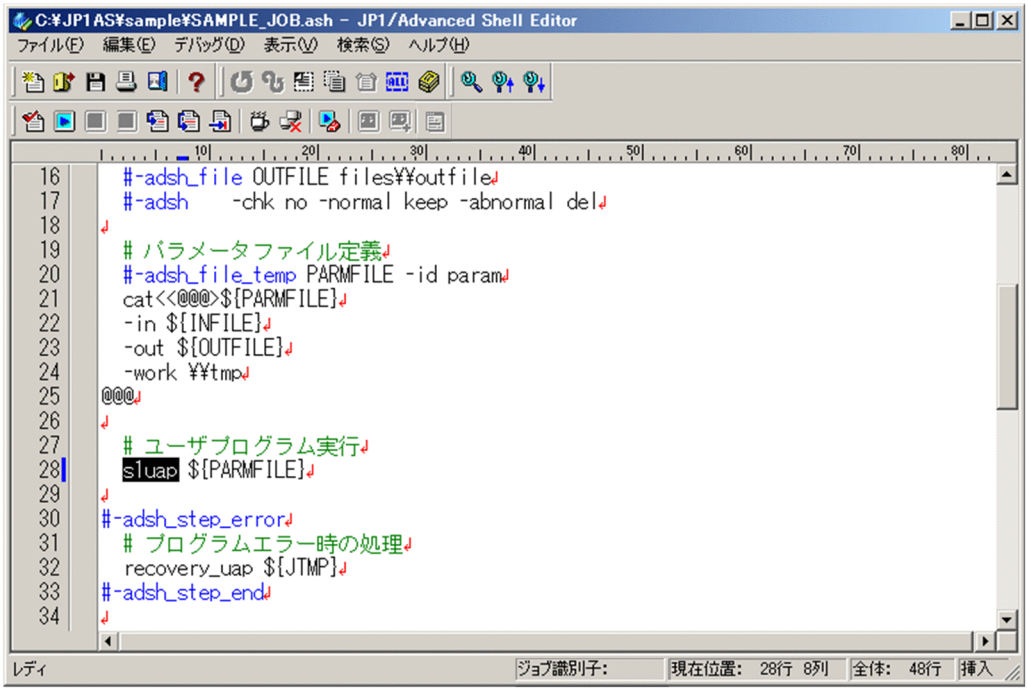
ダイアログボックスの設定方法については，「4.7.5　検索ダイアログボックス」を参照してください。

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

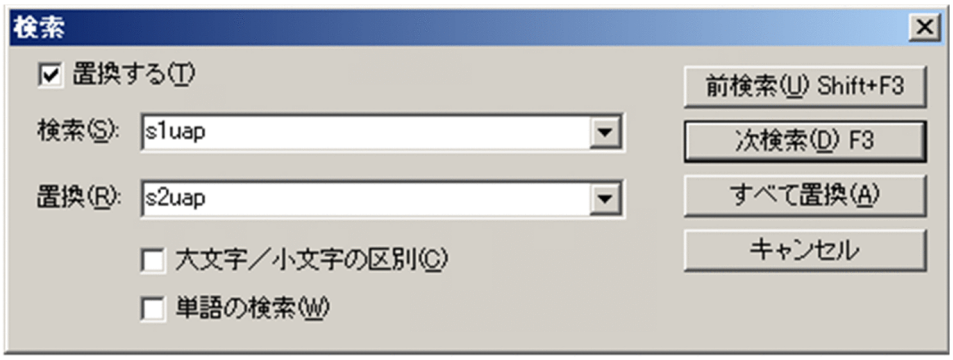
ジョブ定義スクリプトファイル中の文字列を置換します。

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，「4.7.5　検索ダイアログボックス」を参照してください。

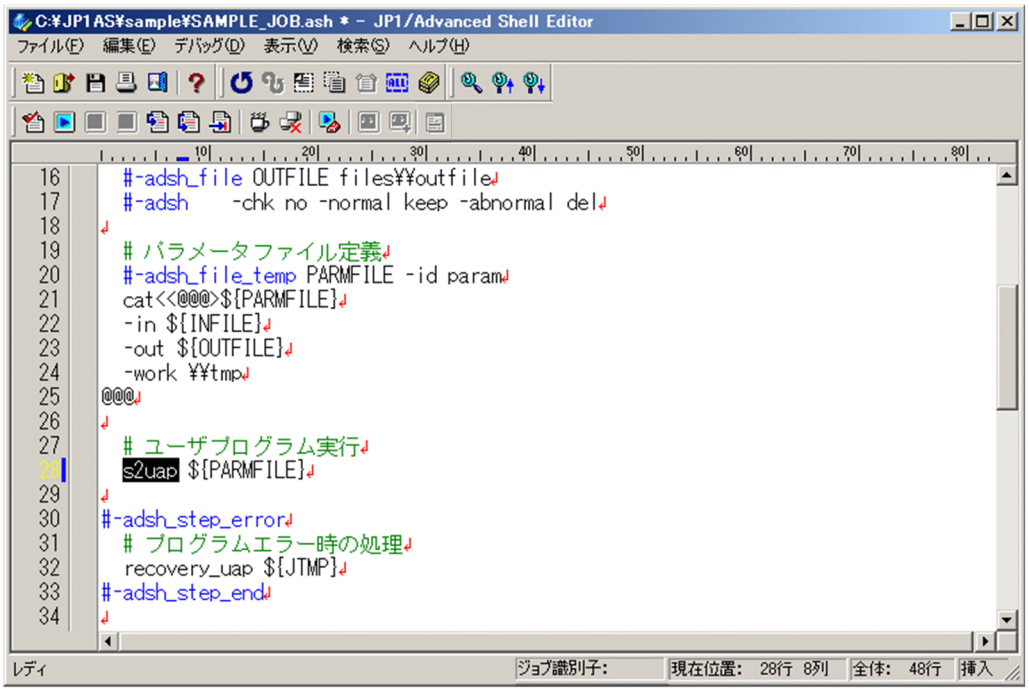
2. **［置換する］がチェックされていることを確認する。**
   ［置換する］がチェックされていない場合は，チェックしてください。

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。

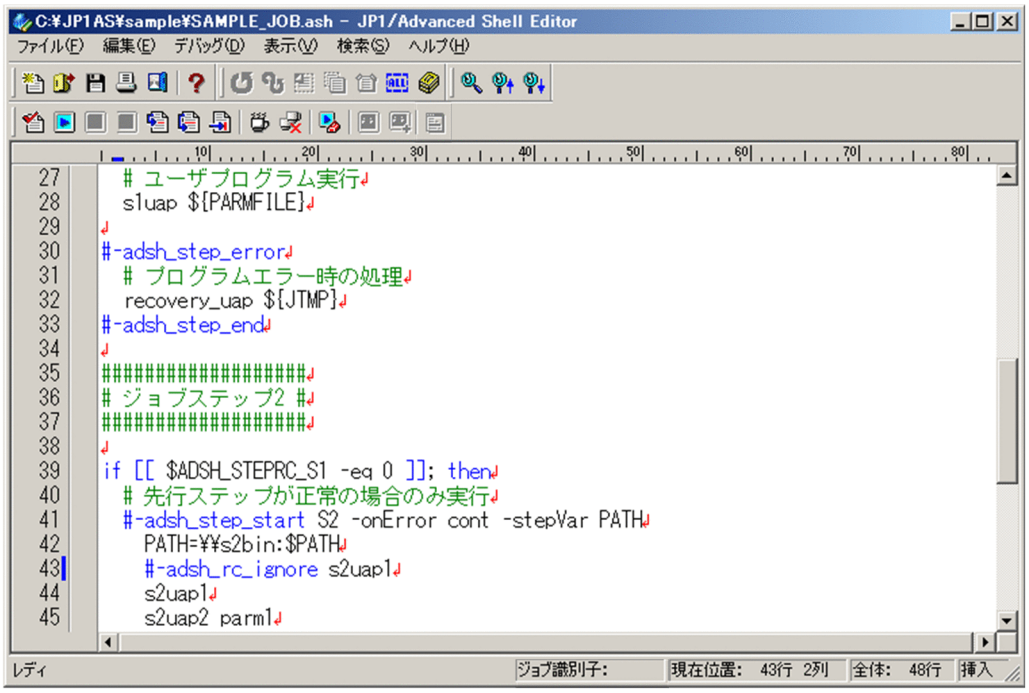
(1) デバッグ実行時のブレークポイントを設定・解除する

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。

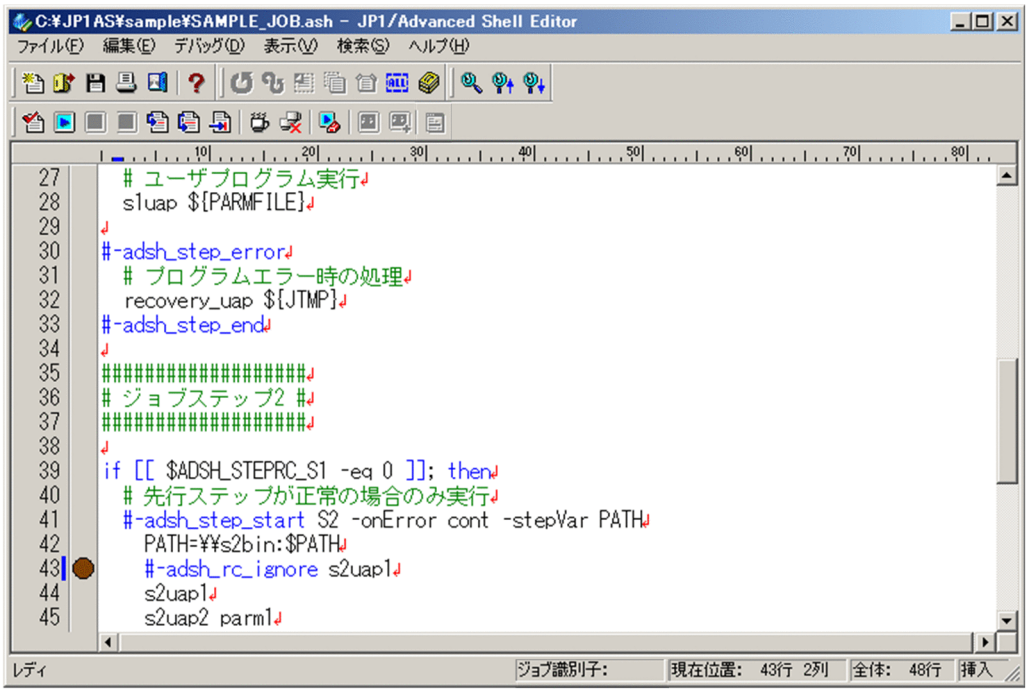
(a) ブレークポイントを設定する

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。**



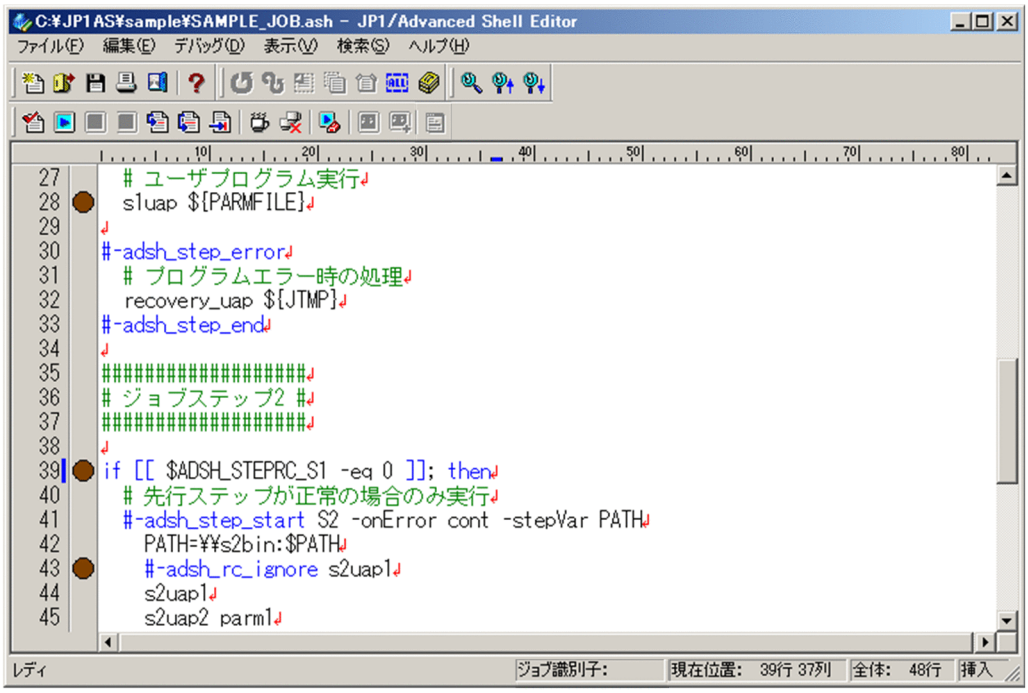
2. **［デバッグ］－［ブレークポイントの設定］メニューを選択する。**

   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。

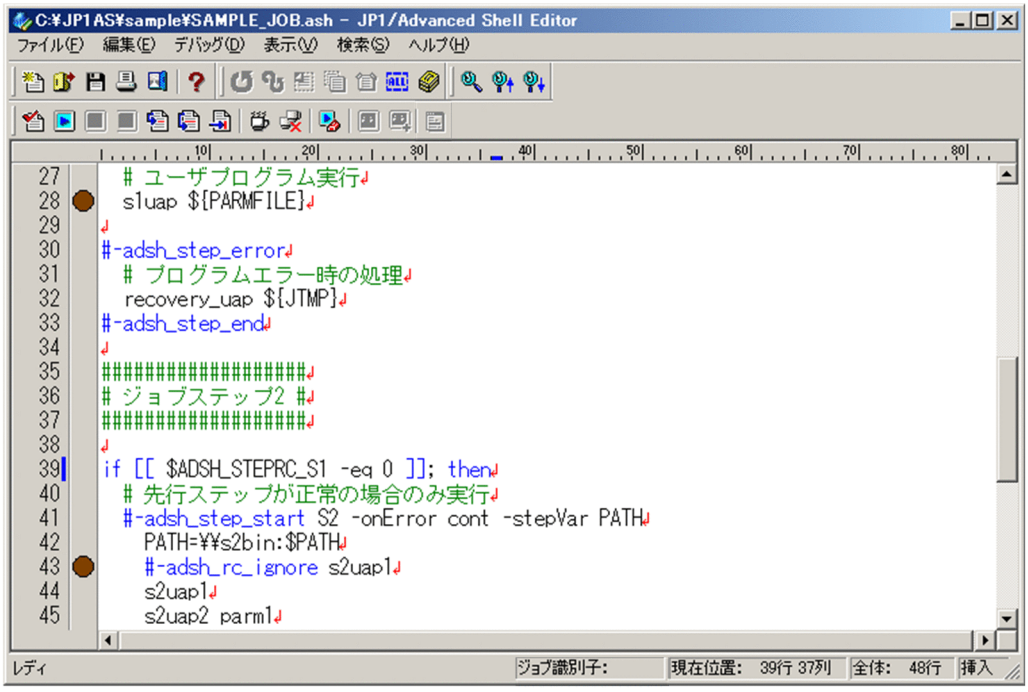


(b) 一部のブレークポイントを解除する

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。**

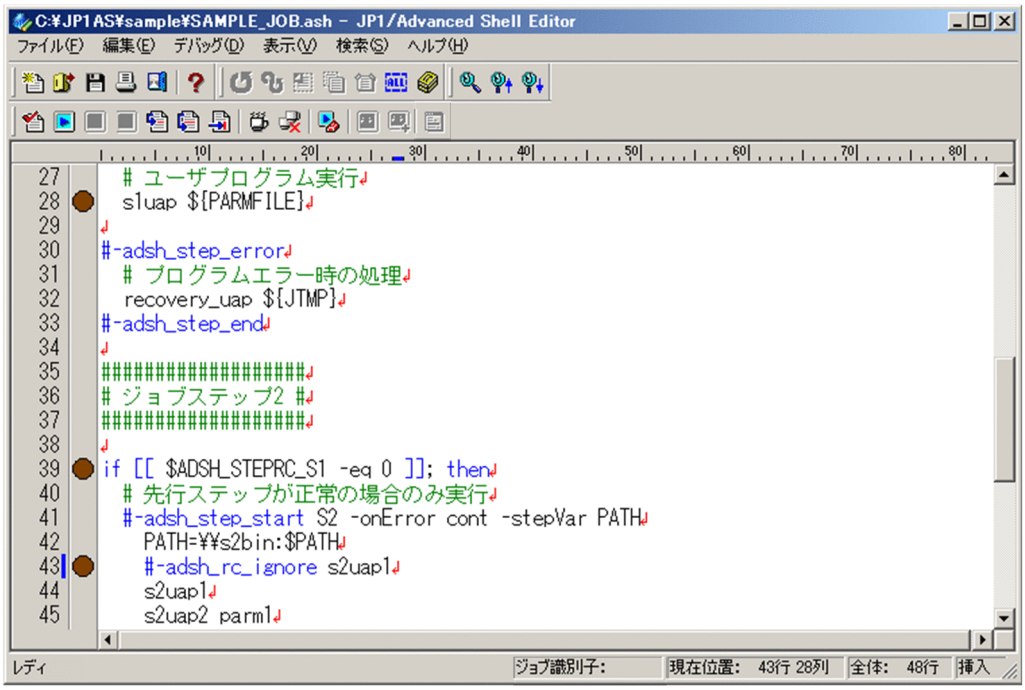


2. **［デバッグ］－［ブレークポイントの解除］メニューを選択する。**
   カーソルのある行のブレークポイントが解除されます。

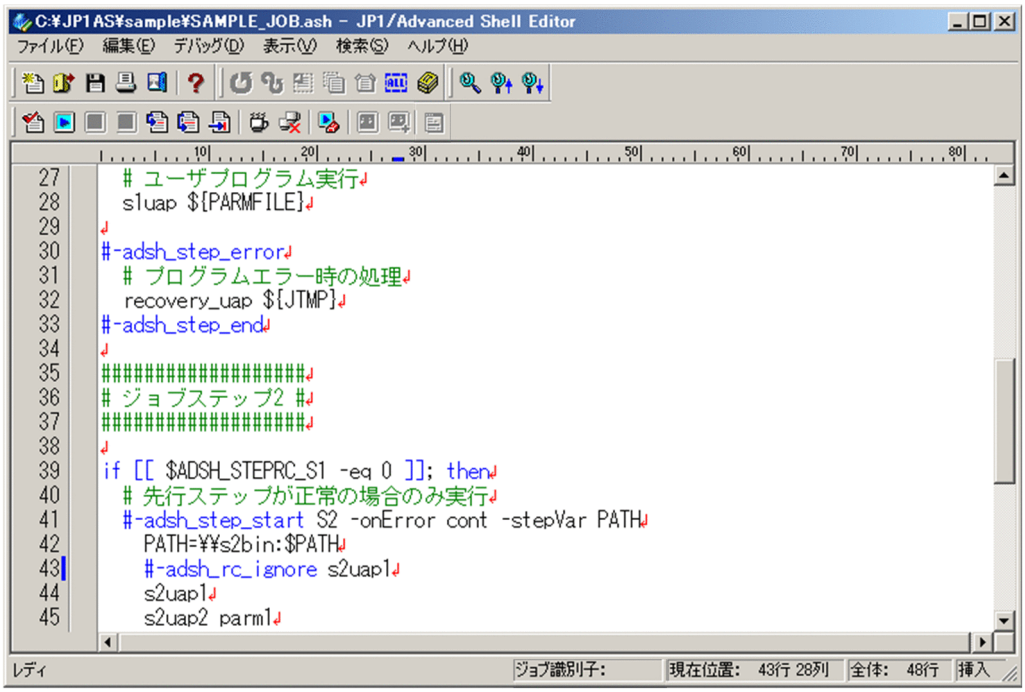


(c) すべてのブレークポイントを解除する

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

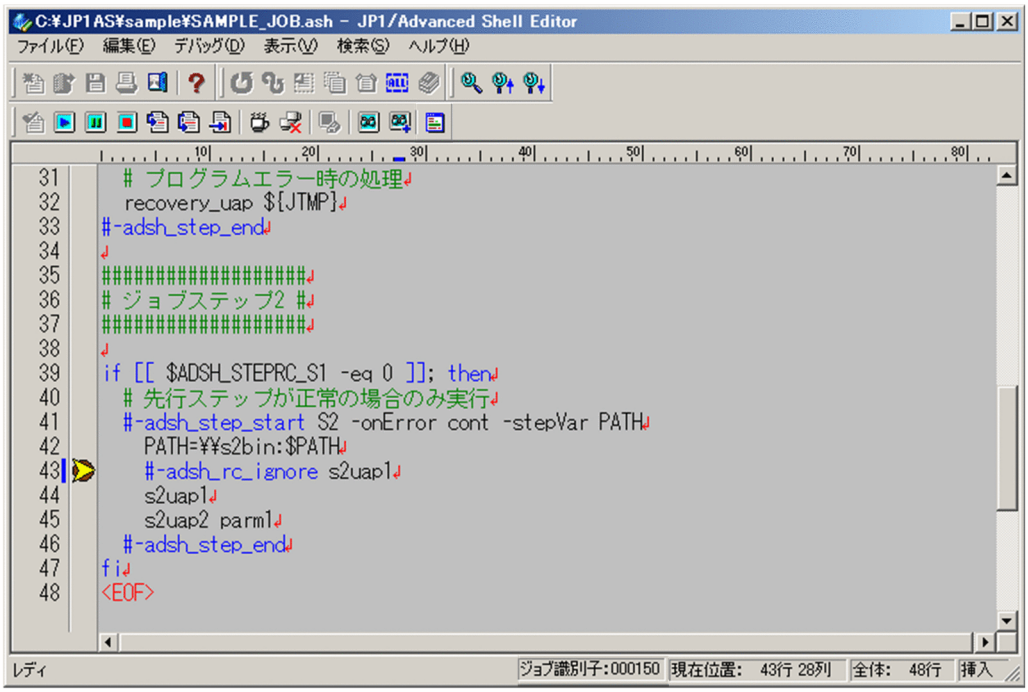
- 編集モードの場合は，任意の行にブレークポイントを設定できます。デバッグモードの場合，ブレークポイントを設定できるのは，実行対象となっているコマンド，またはステートメント単位に限られます。
- 実行できない行にブレークポイントが設定されている場合は，デバッグモード開始時に，エディタが自動的に下方向に適切な位置を検索して，そこにブレークポイントを設定します。

- 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

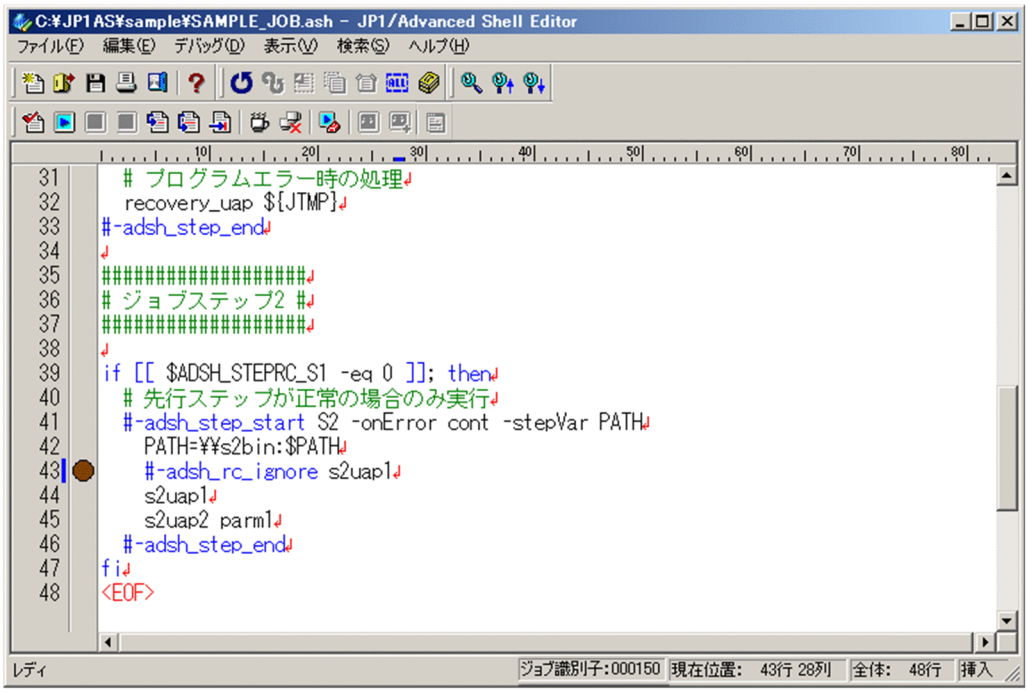
(a) ブレークポイントまで実行する場合

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，「4.4.6(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. **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。

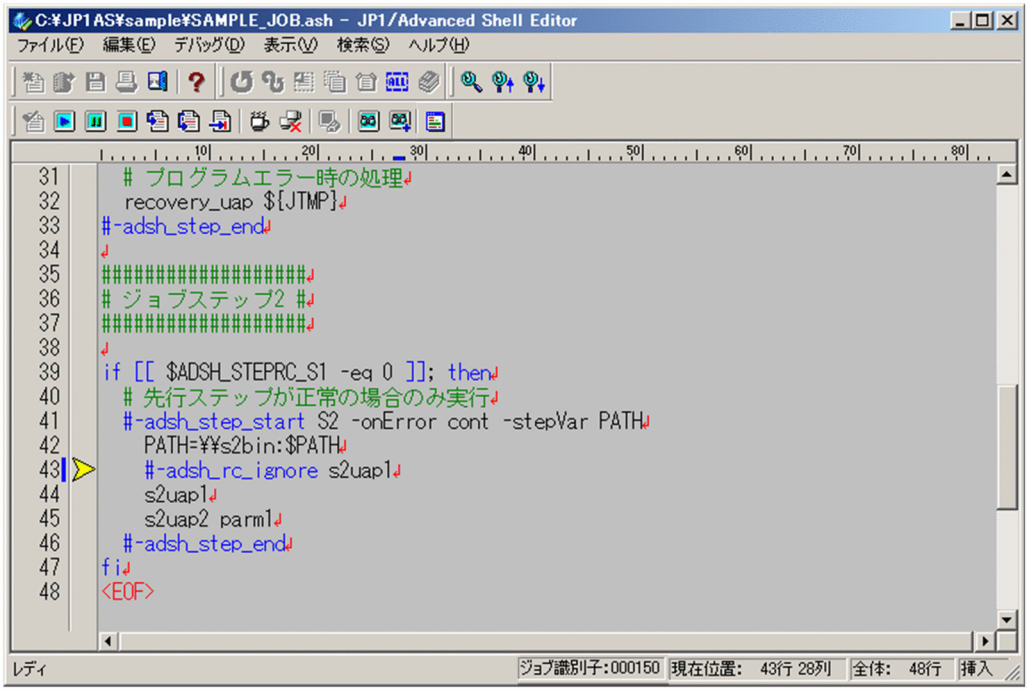


(b) 1 行ずつ実行（関数の中もステップ実行）する場合

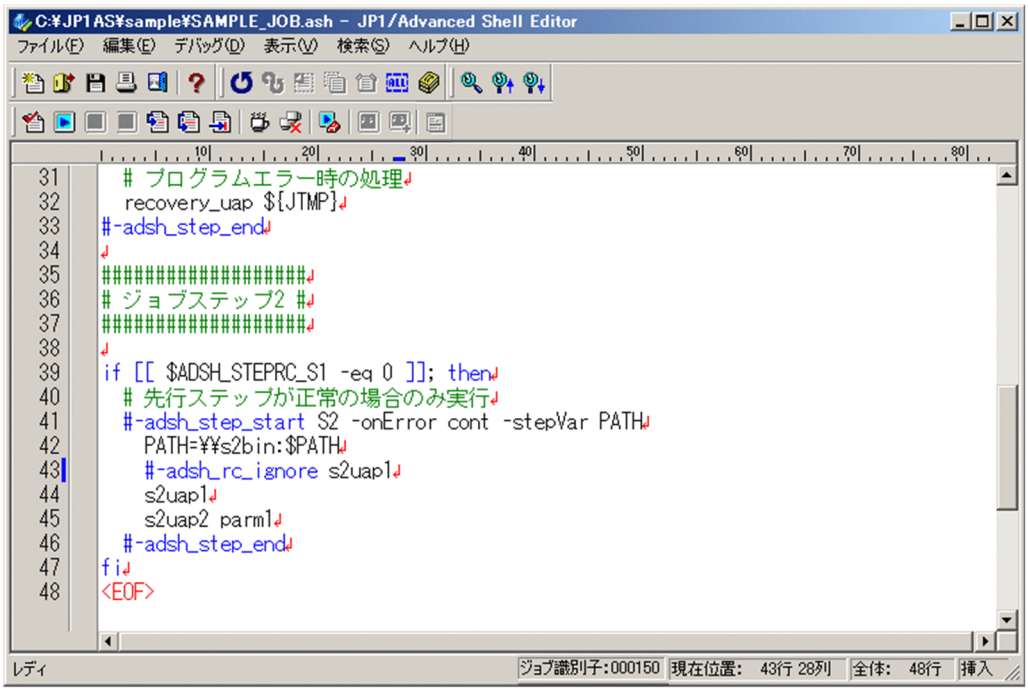
1. **［デバッグ］－［ステップイン］メニューを選択する，またはツールバーの［ステップイン］ボタンをクリックする。**

   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。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。関数の中も 1 行ずつ実行します。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。

   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。



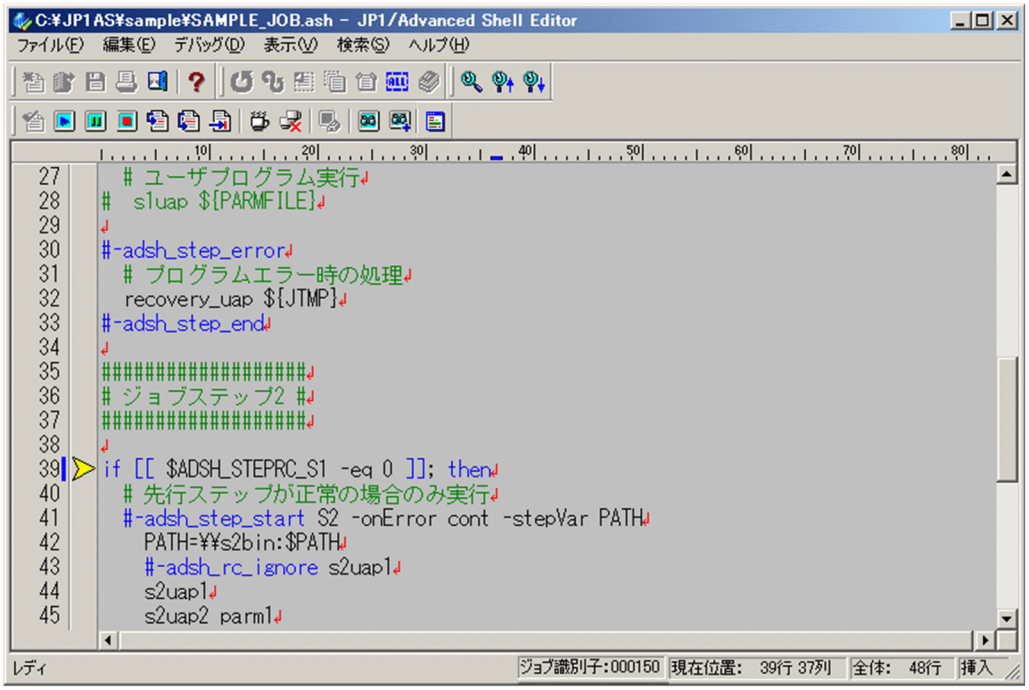
2. **デバッグを中止したい場合は，［デバッグ］－［デバッグの中止］メニューを選択する，またはツールバーの［デバッグの中止］ボタンをクリックする。**
   メニューを選択した時点で，デバッグが中止され，メッセージを出力して後処理をして終了します。なお，プロセス終了前で停止しないでデバッグを終了します。エディタは編集モードに戻ります。



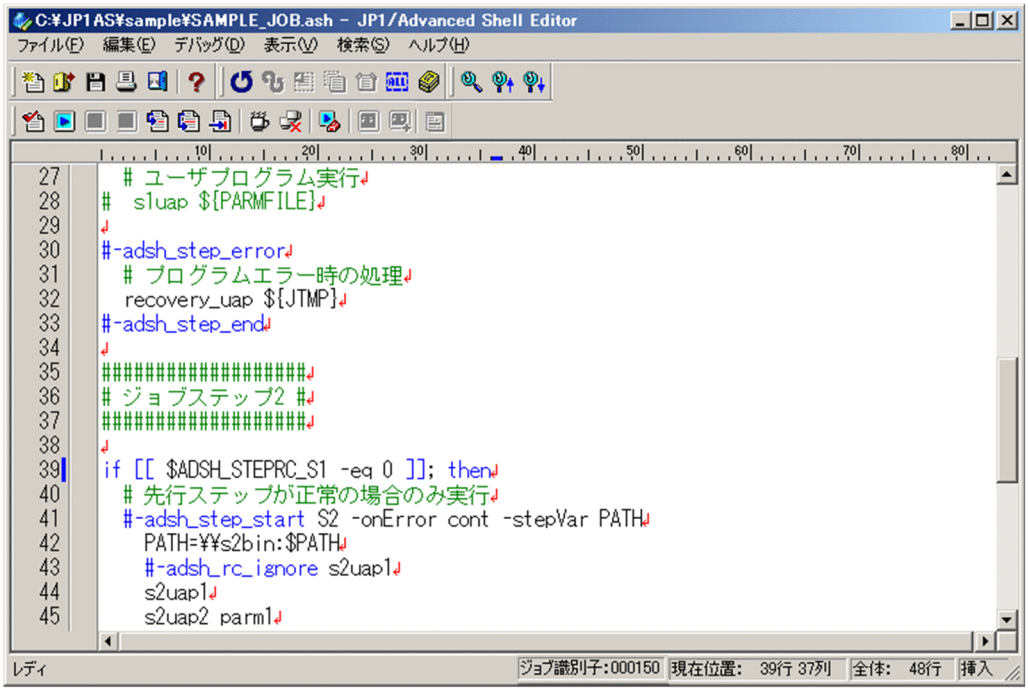
(c) 1 行ずつ実行（関数の中はステップ実行しない）する場合

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。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。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。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。

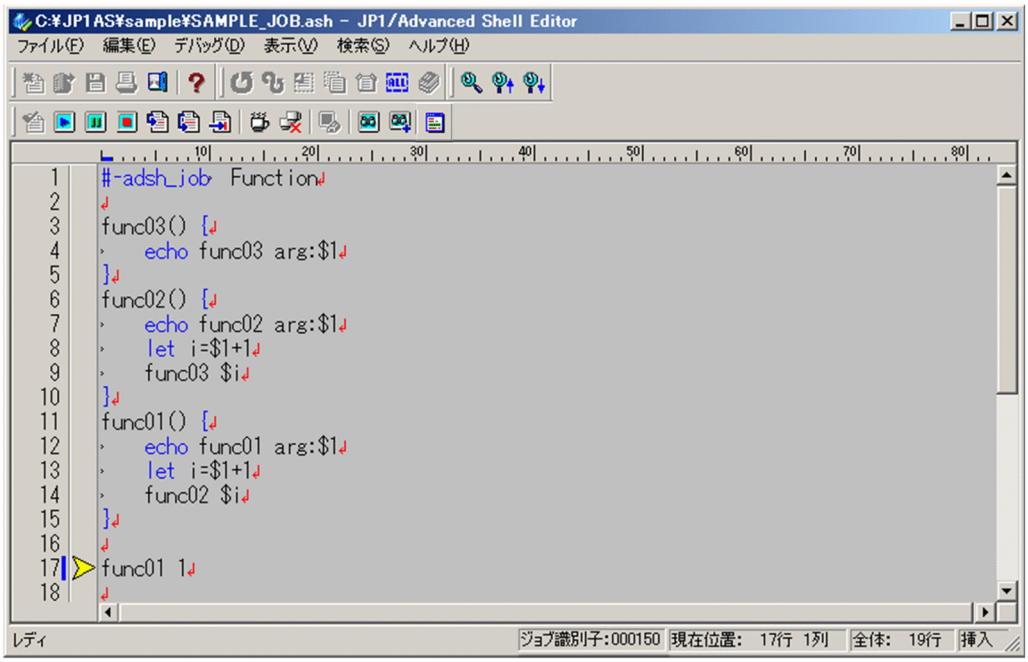


(d) 関数の終わりまで実行する場合

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。**

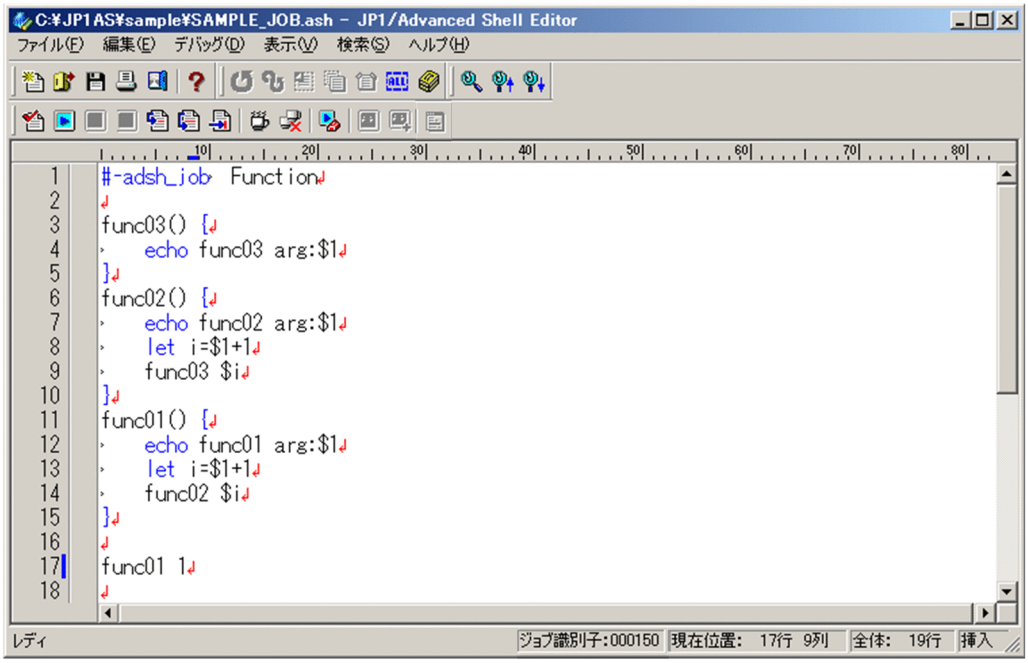
   エディタがデバッグモードになり，デバッグが始まります。次に実行される行の左側には，実行する位置を示す記号（ ）が表示されます。コメント行やスペース行は無視されます。ジョブ定義スクリプトの最後の行まで実行すると，デバッガのプロセスの終了を示す記号（ ）が表示されます。

   CUI とは異なって，関数に入っていない状態でも，ジョブ定義スクリプトの終わりまでジョブ定義スクリプトを実行します。ただし，関数をリターンするまでにブレークポイントがある場合は停止します。



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。

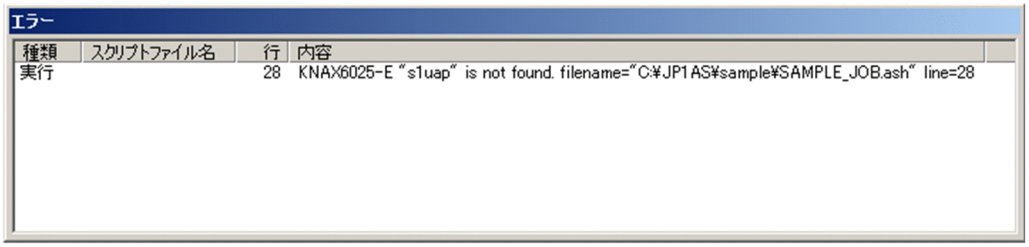
3. **デバッグを中止したい場合は，［デバッグ］－［デバッグの中止］メニューを選択する，またはツールバーの［デバッグの中止］ボタンをクリックする。**
   メニューを選択した時点で，デバッグが中止され，メッセージを出力して後処理をして終了します。なお，プロセス終了前で停止しないでデバッグを終了します。エディタは編集モードに戻ります。



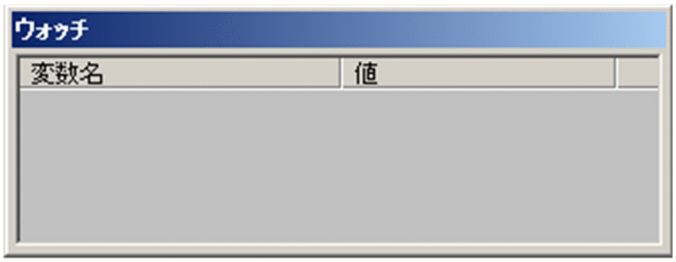
(e) デバッグ実行時のエラーウィンドウ，ウォッチウィンドウおよびコンソール

- エラーウィンドウ

デバッグで検出された解析エラーや実行エラーは，エラーウィンドウに表示されます。また，ウォッチウィンドウに追加されている変数の値は，実行停止時に更新されて，表示されます。
エラーウィンドウについては，「4.7.4　エラーウィンドウ」を参照してください。



- ウォッチウィンドウ
  ウォッチウィンドウに表示される変数の値は，ジョブ定義スクリプト停止時に更新して表示されます。ウォッチウィンドウを表示したい場合は，ジョブ定義スクリプト停止中に［表示］－［ウォッチウィンドウを表示］メニューを選択する，またはツールバーの［ウォッチウィンドウを表示］ボタンをクリックしてください。このメニューは，ジョブ定義スクリプト停止中にだけ選択できます。また，ウォッチウィンドウで変数名をダブルクリックすると［値の更新］ダイアログボックスが表示されます。［値の更新］ダイアログボックスは，変数名の値を更新できます。
  ウォッチウィンドウについては，「4.7.6　ウォッチウィンドウ」を，［値の更新］ダイアログボックスの設定方法については，「4.7.8　値の更新ダイアログボックス」を参照してください。



- 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はコンソールに表示されます。コンソールは，ジョブ定義スクリプトを実行するプロセスが動作している間は表示され，プロセスが終了すると閉じます。ジョブ定義スクリプト実行中はコンソールがアクティブになり，ジョブ定義スクリプト停止時にエディタがアクティブになります。

(3) ウォッチへ変数を追加する

ウォッチウィンドウに指定した変数を追加します。

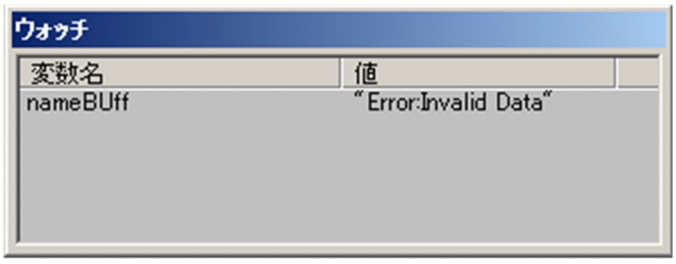
1. **デバッグモードで，［デバッグ］－［ウォッチへ変数の追加］メニューを選択する。**
   ［変数の追加］ダイアログボックスが表示されます。

   ダイアログボックスの設定方法については，「4.7.7　変数の追加ダイアログボックス」を参照してください。

2. **追加したい変数の名前を入力する。**

入力できるのは 99 バイトまでです。これを超えた変数名を入力できません。

3. **［追加］ボタンをクリックする。**
   入力した変数の名前がウォッチウィンドウに追加されます。



4. **［キャンセル］ボタンをクリックする。**
   ［変数の追加］ダイアログボックスが閉じます。

- デバッグモードで，JP1/Advanced Shell エディタウィンドウのクライアントエリアで文字列を選択してから［デバッグ］－［ウォッチへ変数の追加］メニューを選択すると，選択した文字列が「変数名」に表示されます。
- 追加した変数をウォッチウィンドウから削除する場合は，変数名を右クリックし，［変数名の削除］メニューを選択します。

4.4.7 カバレージ情報を表示する

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。カバレージ情報は一時ファイルに出力し，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］メニューが有効となるのは，次の条件を両方とも満たす場合です。1 つでも条件を満たさない場合，メニューがグレーアウトとなり，選択できません。

- ［実行環境の設定］でカバレージを［蓄積する］に設定
- デバッグ実行中のスクリプト停止中（ が終了したとき（ を表示）

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. **［デバッグ］－［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。
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。

(1) 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。**

(2) 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。

(3) 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。

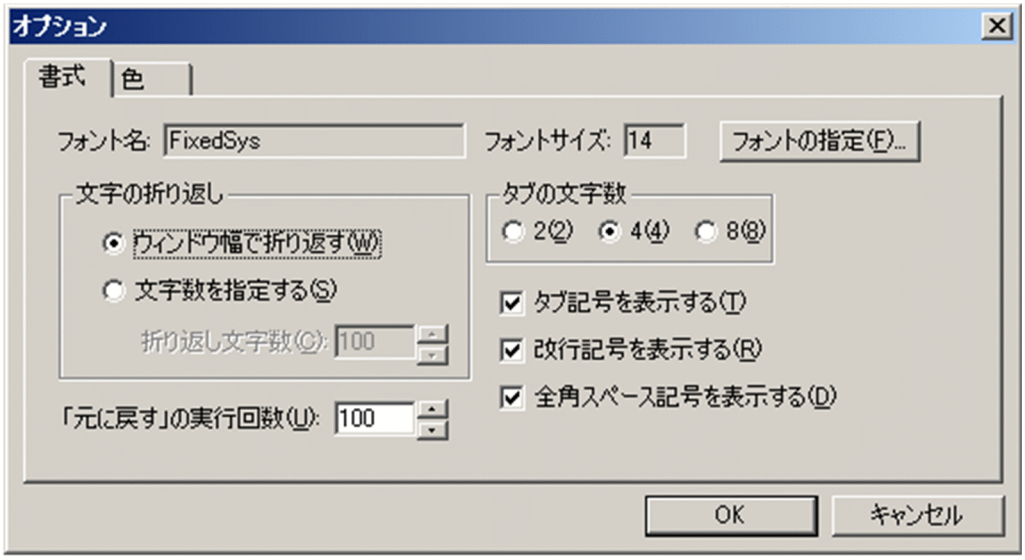
- オプション（書式）ダイアログボックス
- オプション（色）ダイアログボックス
- 実行環境の設定ダイアログボックス
- エラーウィンドウ
- 検索ダイアログボックス
- ウォッチウィンドウ
- 変数の追加ダイアログボックス
- 値の更新ダイアログボックス
- コンソール

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

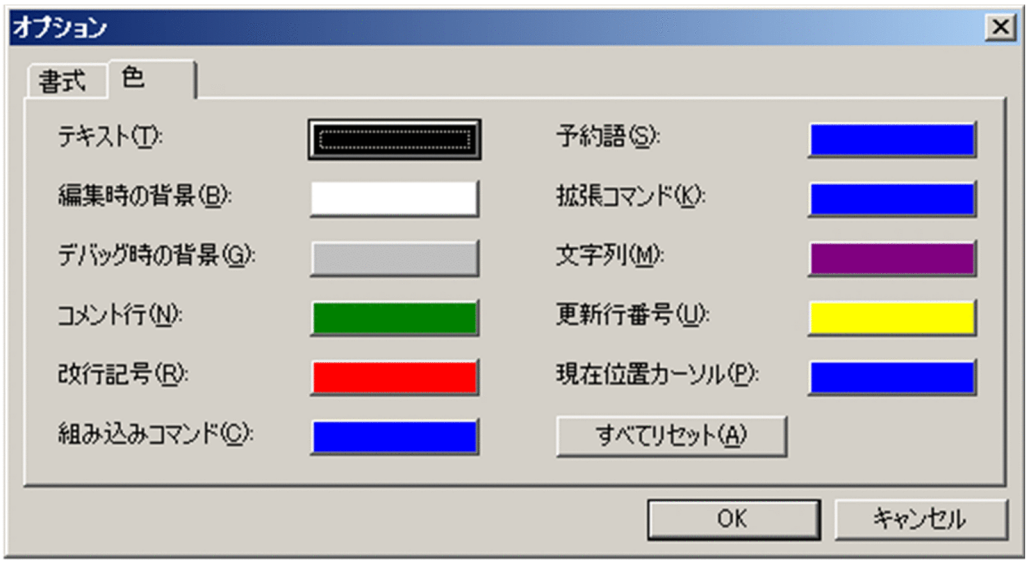
- ［OK］ボタンをクリックすると，指定した書式が設定されてダイアログボックスが閉じます。
- ［キャンセル］ボタンをクリックすると，書式を変更しないでダイアログボックスが閉じます。

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

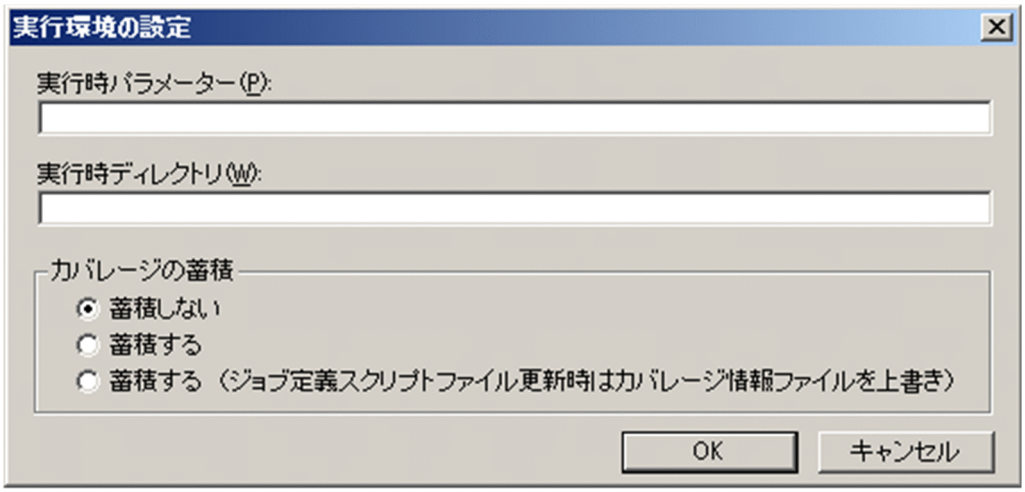
すべての項目に対する色の指定をデフォルト値に戻します。

(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:¥>
```

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。デバッグ実行中に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。デバッグ実行中にエディタからカバレージ情報を表示できます。

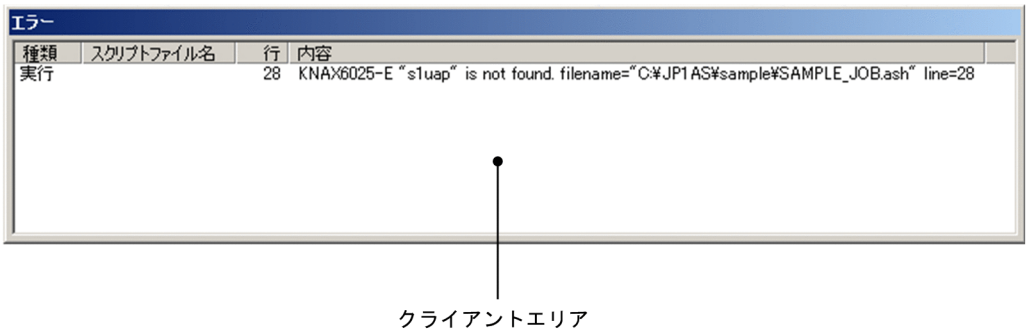
デフォルトでは「蓄積しない」が選択されています。
カバレージ情報は，カバレージ情報ファイル（asc ファイル）に保存されます。asc ファイルは，ジョブ定義スクリプトファイルがあるディレクトリに作成されます。asc ファイルがすでに存在する場合は，ジョブ定義スクリプトファイルがあるディレクトリの asc ファイルを使用します。
カバレージ情報ファイルに蓄積されたカバレージ情報を表示する場合は，adshcvshow コマンドを使用します。また，2 つのカバレージ情報ファイルのカバレージ情報をマージする場合は，adshcvmerg コマンドを使用します。カバレージ情報については，「3.8　カバレージ情報を取得する」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 エラーウィンドウ

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

種類

エラーの種類を表示します。エラーには，次の 2 つの種類があります。

- 解析：文法解析でエラーが発生しました。
- 実行：実行時エラーが発生しました。

ジョブ定義スクリプトファイル名

エラーが発生したジョブ定義スクリプトファイル名を表示します。ただし，エディタで編集中のファイルの場合はスペースになります。

行

エラー発生個所の行番号を表示します。

内容

エラーの内容を表示します。

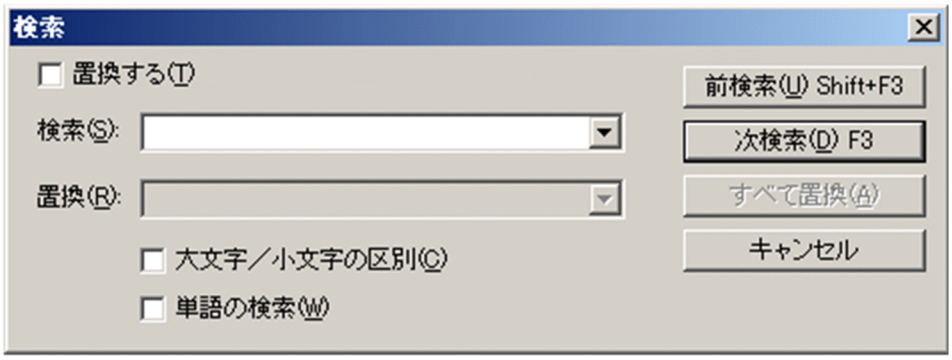
(2) エラーウィンドウの操作

- ジョブ定義スクリプトファイル名がスペースの場合，エラーの種類の表示欄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。
- ジョブ定義スクリプトファイル名がスペースの場合，エラーの種類の表示欄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。

4.7.5 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。

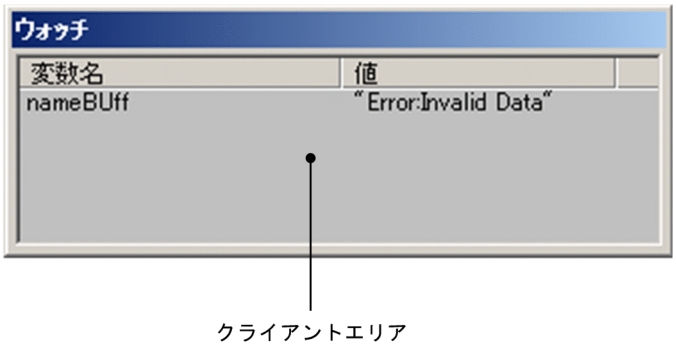
(3) 注意事項

- 「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。

4.7.6 ウォッチウィンドウ

デバッグ実行時に変数名とその変数の値を表示します。

ウォッチウィンドウへの変数の追加方法については，「4.4.6(3) ウォッチへ変数を追加する」を参照してください。



(1) クライアントエリア

変数名

変数名が表示されます。

値

変数の値が表示されます。指定した変数が不当な場合は，次に示す文字列を表示します。

- Error:Bad Number
  数値型の変数に数値以外を設定した場合に表示されます。
- Error:Internal Error
  内部エラーの場合に表示されます。
- Error:Invalid Data
  変数名が無効の場合に表示されます。また，配列変数名だけを指定したインデックス番号が有効な範囲にない場合，指定した配列変数の次元が異なる場合は値を参照できないため，表示されます。
- Error:Not Enough Core
  メモリ不足の場合に表示されます。
- Error:Read Only
  読み込み専用の変数に値を設定した場合に表示されます。
- No Value
  値が存在しない変数を指定した場合に表示されます。

読み込み専用の変数を変更しようとした場合や数値型の変数に文字列を指定した場合は，上記のエラーが表示されます。デバッグを再開すると，変更前の値が設定されています。

(2) ウォッチウィンドウの操作

- 変数名を右クリックすると，次の 2 つを選択するポップアップメニューが表示されます。
  - 値の更新
    選択した変数名の値を更新します。
  - 変数名の削除
    選択した変数名を削除します。
- 変数名をダブルクリックすると，［値の更新］ダイアログボックスが表示されます。

4.7.7 変数の追加ダイアログボックス

デバッグモードで，［デバッグ］－［ウォッチへ変数の追加］メニューを選択すると，［変数の追加］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

変数名

追加する変数名を指定します。

(2) ダイアログボックスでの操作

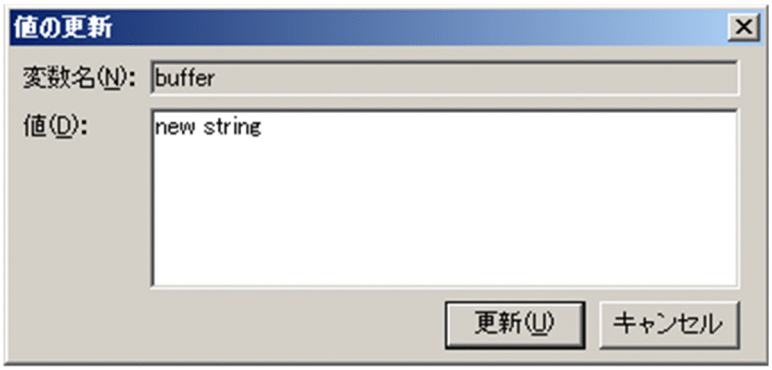
- ［追加］ボタンをクリックすると，指定した変数名がウォッチウィンドウに追加されて［変数の追加］ダイアログボックスが閉じます。
- ［キャンセル］ボタンをクリックすると，変数を追加しないでダイアログボックスが閉じます。

(3) 注意事項

- ［デバッグ］－［ウォッチへ変数の追加］メニューはデバッグモードの場合にだけ，選択できます。
- 同じ変数を複数指定しても，存在しない変数を指定してもエラーにはなりません。
- 指定できる変数名は 99 バイトまでです。これを超えた部分は切り捨てられます。
- 変数名は，99 個まで指定できます。

4.7.8 値の更新ダイアログボックス

［ウォッチウィンドウ］で変数名をダブルクリックすると，［値の更新］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

変数名

更新する対象の変数名が表示されます。

値

更新したい値を入力します。値は 1,024 バイトまで入力できます。更新する対象の変数名が未定義状態の場合，値は変更されません。
デフォルトでは現在の値が設定されています。

(2) ダイアログボックスでの操作

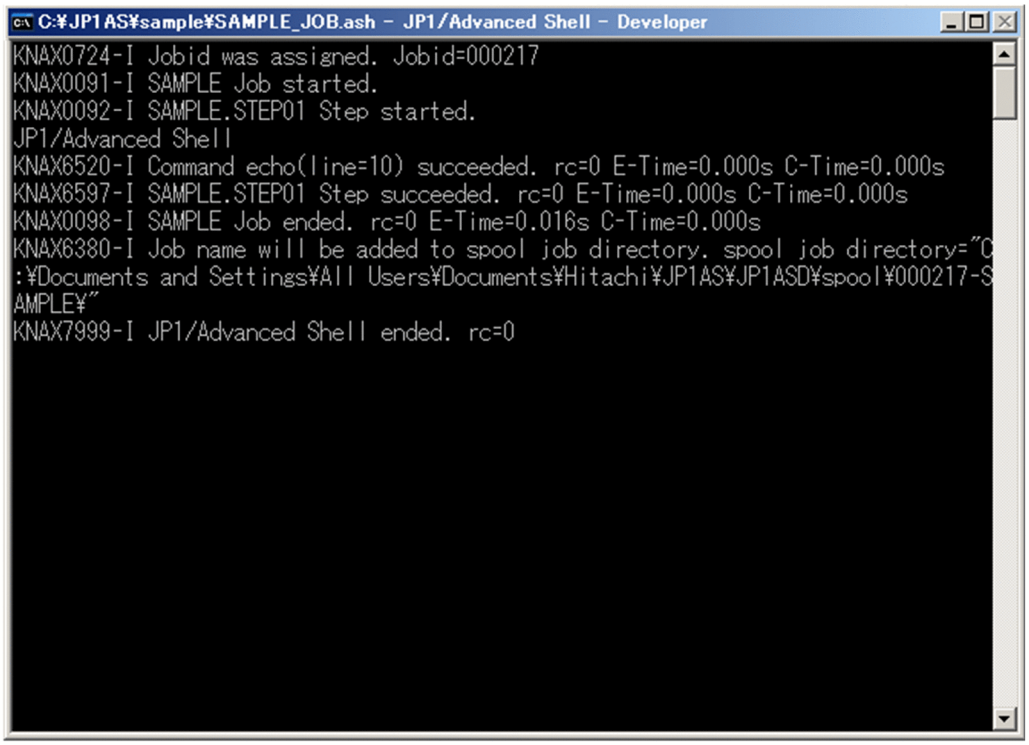
- ［更新］ボタンをクリックすると，指定した変数の値が更新されて［値の更新］ダイアログボックスが

閉じます。

- ［キャンセル］ボタンをクリックすると，変数の値を更新しないでダイアログボックスが閉じます。

4.7.9 コンソール

デバッグ実行に，標準出力，標準エラー出力およびジョブ実行ログを表示します。



(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文法

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

5.1.1　予約語

JP1/Advanced Shell では，ジョブ定義スクリプト内で制御文など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) 変数の作成

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ようとするとエラーとなるため，英大文字を使用することを推奨します。変数の命名規則を次に示します。

- 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，「5.1.11　入力行の上限」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，「6.1.3　デバッガのコマンド一覧【UNIX 限定】」を参照してください。

変数を作成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。

作成と同時に値を代入する場合は，= の右項に値を記述します。変数に代入する値は何文字でも指定できます。しかし，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 signed

long 型の範囲内でなければなりません。範囲を超えると桁あふれが発生し，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（ ' および " ）で囲んでメタキャラクタを無効化にする，または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，「5.1.5　メタキャラクタ」を参照してください。

(2) 変数の値の参照

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。
```

(3) 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，値の先頭までの空白に 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（△は空白）。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

```
********   JOB SCOPE STDOUT    ********
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

変数の 1 つとして配列の作成および参照ができます。

(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 -A 配列名 値 値 …
```

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。
```

- 1 つの要素を作成する場合

```
配列名[要素番号]=値
```

使用例

```
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。
```

set -A コマンドを実行して配列を作成すると，複数の要素を一度に作成できます。配列名 [要素番号]= 値を実行すると，1 つの要素を作成できます。この方法で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 の配列は，変数と同じになります。set -A 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) 配列の値の参照

JP1/Advanced Shell は要素番号 0 から 1,023 までの最大 1,024 個の要素を保持する 1 次元配列があります。配列の値の参照方法を次に示します。

- 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。
```

- 配列の全要素の値を参照する場合
  配列の全要素の値を参照するには，次の 4 種類の方法があります。

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。
```

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と同名の関数を定義できません。変数の命名規則については，「5.1.2(1)　変数の作成」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シェル標準コマンドの .（ドット）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。. コマンドの書式を次に示します。. 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）」を参照してください。

```
. 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 5-3　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）

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

```
********   JOB SCOPE STDOUT    ********
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
123                         ←10行目の結果。関数完了後，変数varの値は回復
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 コマンドで行います。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。

5.1.5 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。JP1/Advanced Shell のメタキャラクタを次に示します。

| & ; < > ( ) $ ` ' ¥ " ~ # * [ ] ?

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 5-4 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.
```

実行ジョブの STDOUT ファイルの内容

```
********   JOB SCOPE STDOUT    ********
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 ¥をエスケープ扱いで¥nを文字として出力
```

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 5-5 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。

位置パラメーターおよび特殊文字		意味
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) 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。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。

(4) 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方法を次に示します。

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。

(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 5-6 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。
*	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「表 5-7 ワイルドカード [ ] の記述例」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 5-7 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「5.7 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 5-8　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
${**variable**:-**word**}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。variable は定義されていますが NULL または未定義の場合，word の展開結果を返します。variable の値は変更しません。	cnt=${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
${**variable**:?[**word**]}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。variable は定義されていて NULL または未定義の場合，word の展開結果（word 省略時は未定義を表すメッセージ）を標準エラー出力に出力し，ジョブ定義スクリプトは終了します。variable の値は変更しません。	cnt=${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}	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
${**variable**?[**word**]}	variable を変数として定義しかつ値を代入している場合，variable の値を返します。variable は定義されていて NULL の場合，variable の値 (NULL) を返します。variable が未定義の場合，word の展開結果（word 省略時は未定義を表すメッセージ）を標準エラー出力に出力し，ジョブ定義スクリプトは終了します。variable の値は変更しません。	cnt=${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

書式	説明	使用例	結果
${**variable**:+**word**}	variable を変数として定義しかつ値を代入している場合，word の展開結果を返します。それ以外の場合は，NULL を返します。variable の値は変更しません。	cnt=${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変数が配列の場合の，要素数への変数置換を行う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 5-9　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，variable に格納されている値の長さに置換されます。
${#**array**[*]}	array で指定された配列の要素数に置換されます。
${#**array**[@]}	

- 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 5-10　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。
	${**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。

実行例を次に示します。

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
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

```
********   JOB SCOPE STDOUT    ********
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
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

```
********   JOB SCOPE STDOUT    ********
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
```

(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，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。

表 5-11 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。

$( ) 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，動作が異なります。逆引用符形式で command の文字列に「¥」が含まれていると，「¥」はメタキャラクタとして扱われます。そのため，実行結果に違いがあります。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

```
echo $(echo '¥$x')
```

  実行結果

```
¥$x
```

- 逆引用符形式の場合

  実行例

```
echo `echo '¥$x'`
```

  実行結果

```
$x
```

(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 5-12 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                      # 1.カレントディレクトリに存在するファイルを表示
ls ?(*.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 *(*.sh|*.exe)        # 3.拡張子がsh,exeのどちらかのファイル名を表示
ls +(*.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|*.jhs)         # 5.拡張子がcまたはjhsのファイル名を表示
ls !(*.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

```
********   JOB SCOPE STDOUT    ********
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

```
********   JOB SCOPE STDOUT    ********
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

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 5-13 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。
< file	file を標準入力として使用します。
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。
>|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。
<>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。
<<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。
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。
>&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。
|& ※1	UNIX の場合，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 ※2	バックグラウンドプロセスの出力先を標準出力とします。
<&p ※2	バックグラウンドプロセスの入力は標準入力からとなります。

注※ 1

「|&」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※ 2

Windows の実行環境の場合，バックグラウンドプロセスを生成できないため使用できません。

(b) ファイルディスクリプタ

JP1/Advanced Shell 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 5-14 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	―

入出力種別	ファイルディスクリプタ	備考
標準出力	1	Windows または Linux で，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，JP1/Advanced Shell がシェルからスプールの 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	JP1/Advanced Shell がシェルからスプールの 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。

表 5-15 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。
<< ¥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。

Windows の場合，パイプラインのコマンドは逐次的に実行されます。組み込みコマンド，関数，外部コマンドのうち，外部コマンドだけは別プロセスで実行されます。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。

UNIX の場合，パイプラインは左から右へ処理が行われ，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。ただし，最後の命令に直接，またはエイリアス定義によって read コマンドを指定したときは，標準出力の内容を変数に代入するため，カレントプロセスで実行されます。使用例を次に示します。

● 例 1

```
ls | grep xxx | read STR
```

変数 STR には read コマンドで読み込んだ内容が格納されます。

● 例 2

```
alias READ="read STR"
ls | grep xxx | READ
```

変数 STR には read コマンドで読み込んだ内容が格納されます。

● 例 3

```
ls | grep xxx | { read STR; }
```

{ read STR; } は別プロセスで実行されます。

パイプラインは次の条件のどれかを満たした場合，構文不正でエラー終了します。

1. パイプラインの先頭のコマンドを指定していない。
   （例）| wc -l | bc
2. パイプとパイプの間のコマンドを指定していない。
   （例）ls | | bc
3. パイプラインの最後のコマンドを指定しないでファイルの終端に達した。
   （例）ls | wc -l |
   <EOF>

ジョブ定義スクリプト実行時にパイプラインで実行するコマンドが指定されていないことを検出した場合，その内容を示すメッセージを出力します。

ジョブ定義スクリプトの内容

```
alias ls="ls -lt"
ls | >&2
```

実行ジョブの JOBLOG ファイルの内容

```
<省略>
********  JP1/Advanced Shell MESSAGE  ********
13:12:08 034767 KNAX0091-I ADSH034767 Job started.
13:12:08 034767 KNAX6520-I Command alias(line=1) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
13:12:08 034767 KNAX6043-W Command to execute in another process is not specified. filename="/
home/jp1as/script/pipe.ash" line=2  ※
13:12:08 034767 KNAX6522-E Command ls(line=2) ended abnormally because the command received a
signal. rc=141 signalNo=13 E-Time=0.013s C-Time=0.010s
13:12:08 034767 KNAX6540-I Another process script(line=2) succeeded. rc=0 E-Time=0.003s
C-Time=0.000s
13:12:08 034767 KNAX0101-E ADSH034767 An error occurred during execution of job.
13:12:08 034767 KNAX0098-I ADSH034767 Job ended. rc=0 E-Time=0.016s C-Time=0.010s
```

注※

KNAX6043-W メッセージが出力されます。

(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 5-16 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 5-17 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**; ...)	サブシェル（子プロセス）※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**; ...;}	カレントシェル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） }	

注※

Windows の実行環境の場合，サブシェル（子プロセス）でグループ化されたコマンドの実行はサポートしていません。

(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。

メタキャラクタ	意味
~	シェル変数の HOME に置き換えられます。
~ 任意の文字列	【UNIX 限定】 / 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。
~+	シェル変数の PWD に置き換えられます。
~-	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

$HOME 変数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。

注 3

「&」は Windows の実行環境ではサポートしていません。

5.1.6 別プロセスでの実行【UNIX 限定】

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。別プロセスで実行する使用例を次に示します。

使用例

パイプ (|) による別プロセスの実行

```
ls -lt | grep ash
```

コマンド置換 ($(), ``) による別プロセスの実行

```
$(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
`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
```

|& によるコプロセスの実行

```
echo abc |&       # 文字列abcをコプロセス実行で出力
sleep 1
read -p STR       # コ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

```
(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する
```

& によるバックグラウンド実行

```
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，JOBLOG 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a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 JOBLOG ファイルの内容

```
<省略>
********  JP1/Advanced Shell MESSAGE  ********
18:35:04 003001 KNAX0091-I ADSH003001 Job started.
18:35:04 003001 KNAX6520-I Command ls=ls -lt(line=1) succeeded. rc=0 E-Time=0.000s
C-Time=0.000s
18:35:04 003001 KNAX6520-I Command alias(line=2) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
18:35:04 003001 KNAX6520-I Command $ls(line=3) succeeded. rc=0 E-Time=0.074s C-Time=0.010s
18:35:04 003001 KNAX6520-I Command grep(line=3) succeeded. rc=0 E-Time=0.072s C-Time=0.010s
18:35:04 003001 KNAX0098-I ADSH003001 Job ended. rc=0 E-Time=0.078s C-Time=0.020s
```

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。

注意事項

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，戻り値 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，戻り値 1 で正常終了します。
- 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，戻り値 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。

5.1.7 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
```

パターンの書式が不正と判定される場合を次の表に示します。

表 5-18　パターンの書式が不正と判定される場合

パターンの書式が不正と判定される場合	例
「[ ]」に囲まれた文字列で，「(」に対応する「)」がありません。	[a*(b])
「( )」に囲まれた文字列で，「[」に対応する「]」がありません。	*(a[b)c]
「[ ]」に囲まれた文字列が，「( )」に囲まれていない「|」を含みます。	*(a[foo|bar]) a[foo|bar]
「[」に対応する「]」がありません。	[a

注 1

「[ ]」に囲まれた文字列では，「[」は文字として扱います。例えば，「[[a]」はパターンの書式として正当となります。この場合，最初の「[」は「]」に対応し，2 つ目の「[」は文字として扱います。

注 2

「[」の直後および「[!」の直後の「]」は，文字として扱うため不正と見なされません。

5.1.8　スクリプト拡張コマンドの実行

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。

5.1.9　外部コマンドの実行

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，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いいます。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

Windows の場合，「.com」，「.exe」，「.cmd」，「.bat」の実行ファイルのときは，ジョブ定義スクリプトから実行できます。ただし，環境変数 PATHEXT に上記の拡張子を登録しておくと，拡張子を指定しなくてもジョブ定義スクリプトから実行できます。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，シェル変数 PATHEXT に登録されている拡張子の順番で実行されます。例えば，シェル変数 PATHEXT の値が「.COM;.EXE;」の場合に「ls.com」と「ls.exe」がシェル変数 PATH の示す場所にあるときは，「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。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。
```

Windows の場合，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を行います。

- "（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

(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比較

ジョブコントローラが，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順序は，次のようになります。

【Windows 限定】

1. 環境ファイルの CHILDJOB_SHEBANG パラメーターで定義した条件を満たすファイルの場合，子孫ジョブとして実行します。
2. 順序 1 でない場合，拡張子が exe，bat，cmd，または com のファイルのときは，コマンドとして実行します。
3. 順序 1 ～ 2 のどちらでもない場合，コマンドの起動に失敗して続行可能なエラーになります。

【UNIX 限定】

1. 指定されたファイルが実行権限のあるバイナリファイルの場合，コマンドとして実行します。
2. Linux でかつ順序 1 でない場合，環境ファイルの CHILDJOB_SHEBANG パラメーターで定義した条件を満たす実行権限のあるファイルのときは，子孫ジョブとして実行します。
3. 順序 1，2 のどちらでもない場合，実行権限のあるテキストファイルのときは，#! に指定した実行プログラムで実行します。#! の指定がないときは「/bin/sh」で実行されます。
4. 順序 1 ～ 3 のどれでもない場合，コマンドの起動に失敗して続行可能なエラーになります。

コマンドの実行方法の比較を，Windows と UNIX に分けて次の表に示します。表内の「順序」の欄が，上記の順序の番号に対応します。

表 5-19 コマンドの実行方法の比較【Windows 限定】

順序	ファイルの種類	動作
1.	環境ファイルの CHILDJOB_SHEBANG パラメーターで指定された条件を満たすファイル	子孫ジョブとして実行します。
2.	拡張子が exe，bat，cmd，または com のファイル	コマンドとして実行します。
3.	順序 1 ～ 2 以外	続行可能エラーとなります。

表 5-20 コマンドの実行方法の比較【UNIX 限定】

順序	ファイルの種類	動作
1.	実行権限のあるバイナリファイル	コマンドとして実行します。
2.	Linux でかつ環境ファイルの CHILDJOB_SHEBANG パラメーターで指定された条件を満たす実行権限のあるファイル	子孫ジョブとして実行します。

順序	ファイルの種類	動作
3.	実行権限のあるテキストファイル	#! で指定したプログラムで起動します。 #! がないときは，「/bin/sh」で起動します。
4.	順序 1 ～ 3 以外	続行可能エラーとなります。

5.1.10 ジョブ定義スクリプトの実行

(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
```

UNIX の場合，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。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パス
```

次の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行ってから実行します。ただし，ジョブ定義スクリプトファイル内で . コマンドで読み込んでいる外部ファイルについては，書式チェックを行いません。

5.1.11 入力行の上限

ジョブ定義スクリプトの 1 行は 8191 バイト以下にしてください。1 行が 8192 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，「5.1.5(4) 行継続」を参照してください。

また，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，8191 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，「9.1.2(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。

5.2 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，「9.5 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 `cmd true`   ←cmd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 cmd 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 `cmd true`       ←cmd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 cmd 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，「9.4　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 [[]] 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。test コマンドの代わりに [[]] 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。

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。

[[]] を使用して条件判定するときの使用例を次に示します。

[[]] で条件判定するときの使用例

```
if [[ $arg1 -eq $args ]]; then
    echo TRUE
fi
```

(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。

表 5-21 数値の比較に使用する演算子

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
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。	○	×
数値 1 == **数値 2**	数値 1 と数値 2 が等しい場合は真。	○※ 1	○
数値 1 != **数値 2**	数値 1 と数値 2 が等しくない場合は真。	○※ 1	○
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。	○※ 1※ 2	○
数値 1 < **数値 2**	数値 1 が数値 2 より小さい場合は真。	○※ 1※ 2	○
数値 1 <= **数値 2**	数値 1 が数値 2 以下の場合は真。	×	○

（凡例）

○：使用できます。

×：使用できません。

注※ 1

数値ではなく，文字列として比較します。文字列比較については，「5.2.2(2) 文字列比較」を参照してください。

注※ 2

[[ ]] で使用できます。test コマンドでは使用できません。

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。

表 5-22 文字列の比較に使用する演算子

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
-n **文字列**	文字列の長さが 1 文字以上の場合は真。	○	×

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
-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,「表 5-34　set コマンドで設定するシェルオプション」の「名称」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。	○※	×

（凡例）
　○：使用できます。
　×：使用できません。

注※
　[[ ]] で使用できます。test コマンド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，test コマンド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,「5.1.5(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。

表 5-23 ファイルの形式や権限などの属性を評価する演算子

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
-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。※1	○	×
-c **file**	file が存在し，キャラクタ型デバイスの場合は真。※1	○	×
-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，レギュラー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。※1	○	×
-h **file**	file が存在し，シンボリックリンクの場合は真。※2	○	×
-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。※1	○	×
-p **file**	file が存在し，パイプファイルの場合は真。※1	○	×
-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**	file が存在し，ファイルサイズが 1 以上の場合は真。	○	×
-t **fd**	端末をオープンしている fd の場合は真。※3	○	×
-u **file**	file が存在し，setuid ビットがセットされている場合は真。※1	○	×
-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。 • 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_SHEBANG パラメーター（Windows，Linux 限定）で設定した条件に一致するファイルである※4 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。※2	○	×
-L **file**	file が存在し，シンボリックリンクの場合は真。※2	○	×
-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。※2	○	×
-S **file**	file が存在し，ソケットの場合は真。※1	○	×

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
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。※2	○	×
file1 -nt **file2**	file1 と file2 が存在し，file1 の更新日付が file2 よりも新しい場合は真。	○	×
file1 -ot **file2**	file1 と file2 が存在し，file1 の更新日付が file2 よりも古い場合は真。	○	×
-H **file**	常に偽。	○	×

（凡例）

○：使用できます。

×：使用できません。

注※ 1

Windows 環境の場合，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。

注※ 2

Windows 環境で test コマンドを実行する場合，判定時の処理がサポートされていないため，エラーとなります。ただし，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「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。

注※ 3

fd に 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。

注※ 4

CHILDJOB_SHEBANG 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「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）【Windows，Linux 限定】」を参照してください。

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 5-24　論理演算で評価する場合に使用する演算子

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) 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。	○※	×

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
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※

test コマンドで使用できます。[[ ]] 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。

表 5-25 JP1/Advanced Shell で使用できる三項演算子

演算子を用いた条件式	判定	test コマンドまたは [[]] での使用可否	let コマンドまたは (()) での使用可否
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。	×	○

（凡例）
○：使用できます。
×：使用できません。

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。

- 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，クォーテーションで算術式を囲む必要があります。
- 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 5-26 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 をか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。

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 5-27 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 5-28 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 5-29 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 をかけた結果を num1 に代入します。
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 5-30 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	?:（三項演算子）
13	=，+=，-=，*=，/=，%=，&=，^=，|=，<<=，>>=

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の名前はすべて大文字に変換されてシェル変数として取り込まれます。

5.5.1 設定および使用できるシェル変数

JP1/Advanced Shell が設定するシェル変数を次の表に示します。

表 5-31 JP1/Advanced Shell が設定するシェル変数

名称	意味
#	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です。
-	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です。
$	シェルのプロセス ID です。
_	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。 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 です。
LINENO	実行中のジョブ定義スクリプトの現在行の行番号です。
OLDPWD	cd コマンドで設定された直前の作業ディレクトリです。
OPTARG	getopts コマンドで処理された最後のオプション引数の値です。
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です。
PPID	Windows の場合は 0 です。 UNIX の場合はシェルの親のプロセス番号です。
PWD	現在の作業ディレクトリです。
RANDOM	0 から 32767(=0x7FFF) までの整数の乱数です。
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を格納する変数です。
SECONDS	シェルが起動してからの経過秒数です。

JP1/Advanced Shell で使用できるシェル変数を次の表に示します。

表 5-32 JP1/Advanced Shell で使用できるシェル変数

名称	意味
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。

名称	意味
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 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。
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	シェル標準コマンドで一時ファイルを /tmp ※以外に作成する場合，一時ファイルを作成するディレクトリパスを指定します。このシェル変数を変更しても，スクリプト拡張コマンドの一時ファイルには影響しません。

注※

Windows の場合，TMPDIR を定義しないでシェル標準コマンドを実行すると，一時ファイルの作成先は次のようになります。

・実行環境で実行する場合：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASE

・開発環境で実行する場合：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASD

#-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。機能の詳細については，「5.9.5　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。

5.5.2　使用できないシェル変数

JP1/Advanced Shell では，変数名として使用できないシェル変数があります。このシェル変数を使用すると，エラーメッセージが表示されます。使用できないシェル変数を次に示します。

COLUMNS，EDITOR，EXECSHELL，FCEDIT，FPATH，HISTFILE，HISTSIZE，KSH_VERSION，MAIL，MAILCHECK，MAILPATH，POSIXLY_CORRECT，SH_VERSION，TMOUT，VISUAL

また，次のシェル変数は JP1/Advanced Shell が内部的に使用するシェル変数であるため，使用しないでください。使用してもエラーメッセージは出力しません。

PS1，PS2，PS3

5.6 ジョブステップ終了コードにシェル変数を使用する

ジョブステップの終了コードを自動的にシェル変数に格納します。

ジョブステップの終了コードを格納するシェル変数の一覧を次の表に示します。

表 5-33 ジョブステップの終了コードを格納するシェル変数の一覧

シェル変数名	値
ADSH_STEPRC_ ジョブステップ名	ジョブステップ名で示すジョブステップの終了コードです。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。
ADSH_RC_STEPMAX	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です。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	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です。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_STEPLAST	過去に実行した最終ジョブステップの終了コードです。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。

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。
```

注意事項

シェル変数は更新しないでください。

5.7 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。

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，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。

5.7.1 set コマンドで設定する場合

set コマンドで設定するシェルオプションを次の表に示します。

表 5-34　set コマンドで設定す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	-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 ※	-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，「5.1.5(6)　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，「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。	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，エラーになりシェルが終了します。	無効
verbose	-v -o verbose	シェル入力行を読み込みかつ同じ内容を標準エラー出力に出力します。入力行を解析および実行する前に出力します。	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。	無効

注※

Windows の実行環境ではサポートしていません。

注意事項

- set コマンドで 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，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，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
```

- 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行う場合，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出力を行っ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 Breakpoint "1": filename="test.ash" line=1
KNAX7032-I Stop in the script "test.ash".
1: set -o verbose
Current: 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
KNAX6520-I Command set(line=1) succeeded. 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 Stop in the script "test.ash".
2: echo "Line 002"
Current: echo
(adshdb)
Line 002
     :
```

5.7.2 adshexec コマンドで設定する場合

adshexec コマンドで設定するシェルオプションを次の表に示します。

表 5-35 adshexec コマンドで設定す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，実行はしません。

5.8 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。

ジョブ定義スクリプトファイルやユーザープログラムが参照できる環境変数を次の表に示します。

表 5-36 ユーザープログラムが受け取ることのできる環境変数

環境変数名	設定する内容	設定されるタイミング
ADSH_JOB_NAME	ジョブ名。	ジョブ開始時
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
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

5.9　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，「9.4　スクリプト拡張コマンド」を参照してください。

5.9.1　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

5.9.2　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
```

5.9.3 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，「9.4　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，JP1/Advanced Shell 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，JP1/Advanced Shell 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，PATH シェル変数へのパス追加をジョブステップローカルに行えます。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。

ジョブステップローカルな PATH シェル変数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 PATH シェル変数の値を，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.
```

ジョブステップローカルな PATH シェル変数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。

#-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，「5.9.5　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
```

5.9.4 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

(1) 有効範囲

この定義は，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
```

5.9.5 パス名を扱うシェル変数を定義する

#-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用いると，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。

JP1/Advanced Shell 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。

- "（ダブルクォーテーション）で囲まれた文字列
- 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

(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 /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
```

●ジョブ定義スクリプトの指定

```
#-adsh_path_var PATH,DIR,DIR3    ←シェル変数の定義1

DIR="/home/hitachi/bin"    ←「DIR="C:¥¥hitachi¥¥bin"」にパス文字列の置換1で変換

"$DIR/myprog"    ←「"$DIR¥¥myprog"」にシェル変数の定義1で変換

"${DIR}/myprog"    ←「"${DIR}¥¥myprog"」にシェル変数の定義1で変換

DIR2=$DIR
"$DIR2/myprog"    ←「$DIR2」はシェル変数の定義1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"    ←「"D:¥¥jp1as_tmp¥¥file"」にパス文字列の置換2で変換

DIR3=""
ls "$DIR3../bin"    ←「"..¥¥bin"」にシェル変数の定義1で変換。相対パスも変換

DIR4="/home/hitachi/sbin:$DIR2"    ←「C:¥¥hitachi¥¥sbin;$DIR2」にパス文字列の
                                     置換1で変換。パス区切り文字も変換

PATH="../bin/:$DIR"    ←「..¥¥bin¥¥;$DIR」にシェル変数の定義1で変換
                         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（Windows，Linux
限定）
```

●ジョブ定義スクリプトの指定

```
#-adsh_path_var PATH,DIR,DIR3    ←シェル変数の定義1

DIR="C:¥¥hitachi¥¥bin"    ←「DIR="/home/hitachi/bin"」にパス文字列の置換1で変換
"$DIR¥¥myprog"    ←「"$DIR/myprog"」にシェル変数の定義1で変換

"${DIR}¥¥myprog"    ←「"${DIR}/myprog"」にシェル変数の定義1で変換

DIR2=$DIR
"$DIR2¥¥myprog"    ←「$DIR2」はシェル変数の定義1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"    ←「"/tmp/jp1as/file"」にパス文字列の置換2で変換

DIR3=""
ls "$DIR3..¥¥bin"    ←「"../bin"」にシェル変数の定義1で変換。相対パスも変換

DIR4="C:¥¥hitachi¥¥sbin;$DIR2"    ←「/home/hitachi/sbin:$DIR2」にパス文字列の
                                    置換1で変換。パス区切り文字も変換

PATH="..¥¥bin¥¥;$DIR"    ←「../bin/:$DIR」にシェル変数の定義1で変換
                           パス区切り文字も変換

"$DIR2¥¥myprog" > nul    ←「/dev/null」にファイル入出力時のファイルパスの変換で変換
```

(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。

```
********  JP1/Advanced Shell MESSAGE  ********
（省略）
21:58:28 000903 KNAX6803-I Access path convert rule matched. filename="D:¥home¥path_conv.ash"
line=4 rule_str="./local.log":"./myprog.log"
（省略）

********   Script IMAGE    ********

***** D:¥home¥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

***** converted lines in D:¥home¥path_conv.ash *****
0003 : DIR="C:¥¥Program Files"; DIR2="C:¥¥temp¥¥tmpfile"
0004 : "$DIR¥¥myprog"

***** CONVERSION INFORMATION *****
21:58:28 000903 KNAX6800-I Path convert rule matched. filename="D:¥home¥path_conv.ash" line=3
rule_str="/home/hitachi/bin":"C:¥¥Program Files"
21:58:28 000903 KNAX6800-I Path convert rule matched. filename="D:¥home¥path_conv.ash" line=3
rule_str="/tmp":"C:¥¥temp"
21:58:28 000903 KNAX6801-I Path convert rule matched. filename="D:¥home¥path_conv.ash" line=4
rule_var="DIR"
```

5.9.6 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 JP1/Advanced Shell 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 . コマンドとは異なって，JP1/Advanced Shell 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

#-adsh_step_start exS2
  exUap01
#-adsh_step_end
```

script.ash

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

uap01

#-adsh_script /scripts/exScript.ash

uap2
```

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6行がexScript.ashが展開された部分
  exUap01
#-adsh_step_end

#-adsh_step_start exS2
  exUap01
#-adsh_step_end

uap2
```

(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
```

5.9.7 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 5-37 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	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
```

5.9.8 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。

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。

なお，ジョブステップ正常ブロック内で，引数に外部コマンドを指定した exec コマンドを実行した場合は，ジョブステップの終了コードが exec コマンド実行直前の終了コードとなります。

(3) コマンドの終了コード

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。

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，JP1/Advanced Shell では，次に示す値をコマンドの終了コードとして扱います。

【UNIX 限定】

コマンドがリターンする値の下位 8 ビット

【Windows 限定】

- 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
- コマンドがリターンする値が 0 未満の場合，255
- 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 5-38 例外として扱う例外コードと意味

項番	例外コード	意味
1	0xC0000005	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	ブレークポイントに到達しました。
3	0x80000002	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	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	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	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	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	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。
9	0xC0000030	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	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	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	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	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	整数演算の結果，オーバーフローが発生しました。
15	0xC0000096	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	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。

コマンドの実行結果が，次の表のどれかに該当する場合，そのコマンドがエラー終了したと見なします。

コマンド実行結果	終了コード
コマンドの終了コードが 0 ではない場合（successRC 属性で値を変更可能）	コマンドの終了コード
コマンドがシグナル終了した場合	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
指定されたコマンドが存在しないで，実行できなかった場合	127

なお，次の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。

- #-adsh_rc_ignore コマンドで指定したコマンド
- 次のシェル標準コマンド
  true　false

（4）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，正常かエラーかの判定が正しく行われるように，次の方法でジョブ定義スクリプトを記述してください。

- 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。

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。

5.9.9　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
- 関数の外に指定した return コマンド

（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。

- unset コマンドを引数を指定しないで実行する
- return コマンドの引数に文字列を指定する

5.9.10　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 5-39 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。
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 5-40 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 5-41 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 5-42 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`	○	○	○	△
	－	－	－	－
`#-adsh_step_start NO -run normal`	△	△	○	△
`  echo "run normal step"`	△	△	○	△
`  cmdNormal`	△	△	○	△
`#-adsh_step_end`	△	△	○	△
	－	－	－	－
`#-adsh_step_start AB -run abnormal`	○	○	△	△
`  echo "run abnormal step"`	○	○	△	△

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
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 5-43　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。 • 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 5-44　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	—	—	—

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_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	○	○	△
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。

5.9.11 コマンド実行結果の出力に関する注意事項

JOBLOG 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。

- ジョブ実行中にコマンドが終了した場合に限り，ジョブログに終了メッセージを出力します。ジョブ実行完了後にコマンドが終了した場合は，ジョブログにコマンドの終了メッセージを出力しません。なお，環境ファイルに CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）を 1 つ以上指定している場合は，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。

(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。
- exec コマンドの引数にコマンドを指定した場合
  exec コマンドを実行した時点でジョブが終了するため，exec コマンド自身の実行結果および引数のコマンドの実行結果のどちらも出力しません。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。

5.10　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ついて，ファイルの割り当ておよび後処理ができます。

スクリプト拡張コマンドについては，「9.4　スクリプト拡張コマンド」を参照してください。

5.10.1　通常ファイルの割り当ておよび後処理をする

#-adsh_file コマンドを使用して，ジョブまたはジョブステップ，これらから起動するコマンドで使用する通常ファイルの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。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。

(1) 通常ファイルの割り当て

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。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，-chk 属性に 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，-chk 属性に 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。

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。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 属性または -abnormal 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 属性の指定に従って後処理をします。-normal 属性の指定および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，-abnormal 属性の指定に従って後処理をします。-abnormal 属性の指定および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。

ただし，ファイル割り当て処理中にエラーが発生した場合，エラーが発生したジョブまたはジョブステップで割り当てた通常ファイルについては，-normal 属性および -abnormal 属性の指定に関係なく keep を仮定し，ファイルを削除しません。

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。

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。

5.10.2 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，ファイルのパーミッションは 0600 で固定です。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とコマンド」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。

5.10.3 プログラム出力データファイルの割り当てをする

#-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とコマンド」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
```

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。

(3) プログラム出力データファイルの使用例

```
#-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
```

ジョブ定義スクリプトを実行した場合，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

5.11　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の図に示します。

● ジョブ定義スクリプトファイルの例

```
#!/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
```

ジョブ定義スクリプトファイルの例の番号は，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

UNIX 環境でコマンドを使用して，ジョブ定義スクリプトファイルをデバッグできます。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。

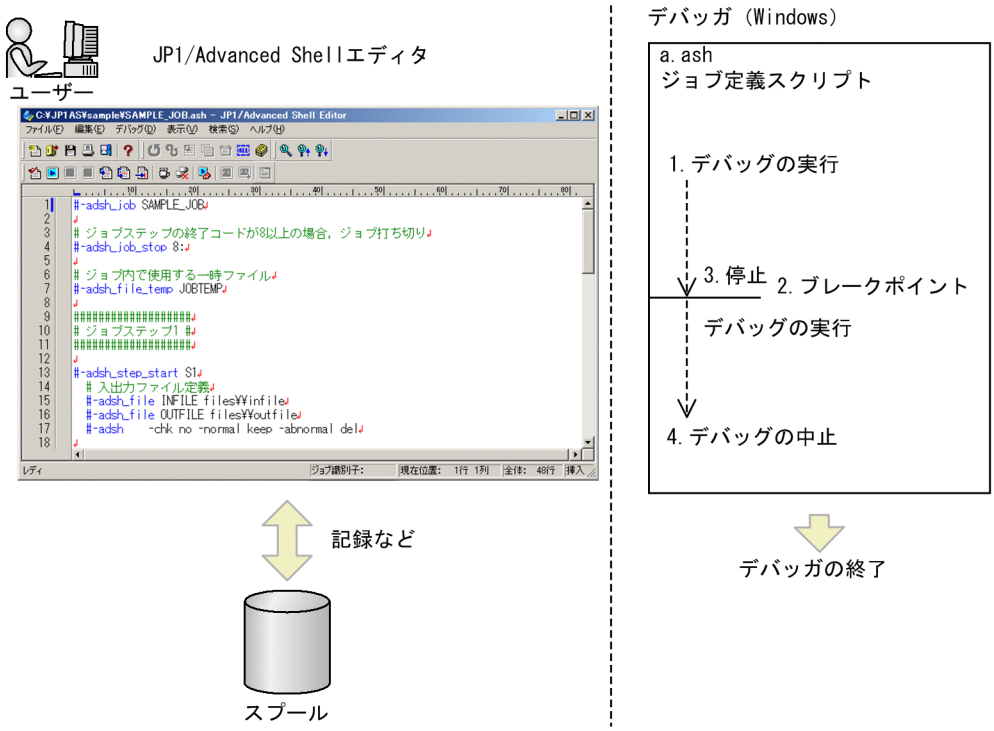
デバッガを利用すると次のことができます。

- 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※
- ヘルプの表示

注※
CUI でのデバッガの場合に利用できます。

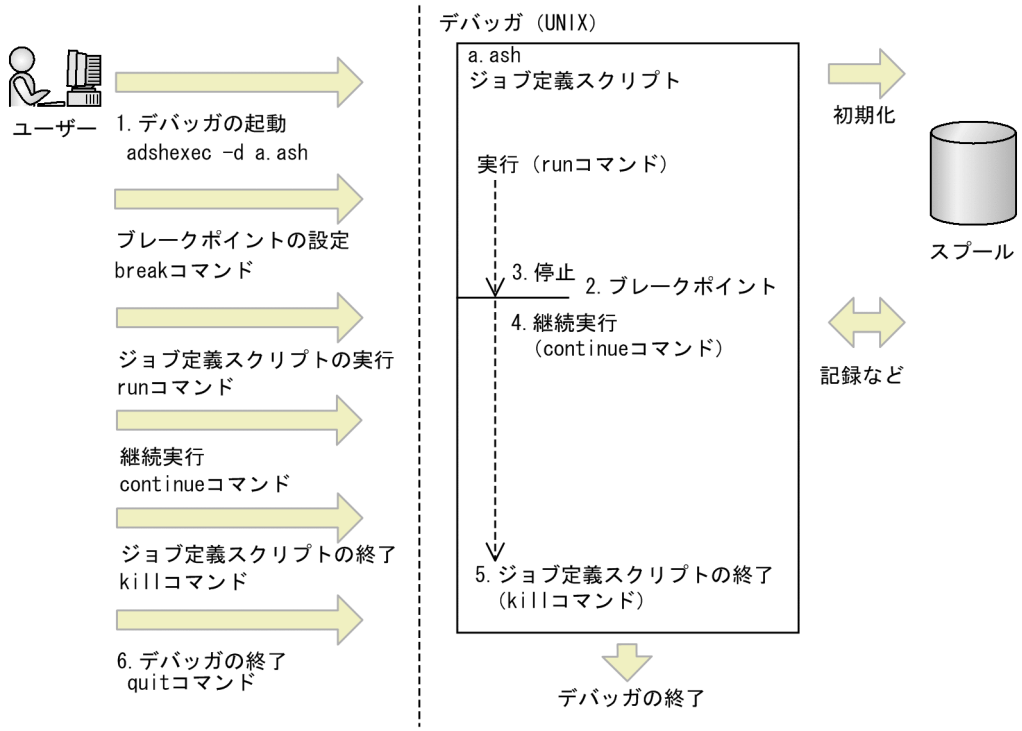
デバッグの実行の概要について次の 2 つの図に示します。

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。

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。

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。

（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，「4.7.4 エラーウィンドウ」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：JP1/Advanced Shell 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 CUI 操作でデバッグできます。

(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，前の実行で設定した次の情報が初期化されます。

- シェル変数
- 環境変数

また，次の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：JP1/Advanced Shell 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：JP1/Advanced Shell 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 デバッガのコマンド一覧【UNIX 限定】

デバッガのコマンドを次の表に示します。各コマンドは短縮形で実行できます。

表 6-1 デバッガのコマンド

コマンド名	機能	コマンドの短縮形	マニュアルの参照先
break	ブレークポイントを設定します。	b	6.2.4
cd	ディレクトリを移動します。	cd	6.2.24
continue	継続実行をします。	c	6.2.9
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
exec	ログインシェルを起動します。	ex	6.2.25
finish	関数を実行します。	f	6.2.10
help	ヘルプを表示します。	h	6.2.26
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	6.2.14
info coverage	デバッグ途中のカバレージ情報を表示します。	i c	6.2.15
info functions	関数情報を表示します。	i f	6.2.16
info jobsteps	ジョブステップ情報を表示します。	i j	6.2.17
info signals	シグナル情報を表示します。	i si	6.2.18
info variables	変数情報を表示します。	i v	6.2.19
kill	ジョブ定義スクリプトを終了します。	k	6.2.3
list	ソースファイルを表示します。	l	6.2.23
next	関数内で停止しないで逐次実行をします。	n	6.2.8
print	変数の値を表示します。	p	6.2.21
quit	デバッガを終了します。	q	6.2.1

コマンド名	機能	コマンドの短縮形	マニュアルの参照先
return	関数を終了します。	ret	6.2.11
run	ジョブ定義スクリプトを実行します。	r	6.2.2
set	変数の値を設定します。	set	6.2.20
signal	シグナルを送信します。	si	6.2.12
step	関数内も含んで逐次実行をします。	s	6.2.8
watch	ウォッチポイントを設定します。	wa	6.2.5
where	バックトレースを表示します。	whe	6.2.22

デバッガの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。

6.1.4 GUI デバッガの機能一覧【Windows 限定】

GUI デバッガの機能について次の表に示します。

表 6-2 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
デバッグを中止する	4.4.6(2)
ジョブ定義スクリプトを停止する	4.4.6(2)(d)
ブレークポイントを設定する	4.4.6(1)
ブレークポイントを解除する	4.4.6(1)
逐次実行する	4.4.6(2)(b)，4.4.6(2)(c)
継続実行する	4.4.6(2)(a)
関数を実行する	4.4.6(2)(d)
変数の値を設定または表示する	4.4.6(3)，4.7.6

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。	○

ジョブ定義スクリプトで使用する要素	内容	停止の可否
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。	×
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。	○
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，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。

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について説明します。

コマンドの記述形式を次に示します。

```
△0コマンド名［△1-オプション名［△1値］］…［△1-オプション名［△1値］］
［△1任意名］
```

- 最初にオプションを指定し，次に任意名を指定します。オプションの前に任意名を指定した場合は，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）。
- オプションを連続して指定する場合，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。
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
```

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
```

```
(adshdb) break 8
KNAX7018-I Breakpoint "1": filename="test.ash" line=8
(adshdb) run
…
KNAX7018-I Breakpoint "1": filename="test.ash" line=8
KNAX7032-I Stop in the script "test.ash".
8: funcA
Current: 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。

```
Old Value = 更新前の値
New Value = 更新後の値
Line = 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]
- 変数とその配列の 0 番目（例：aaa と aaa[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

「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"
```

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 Watchpoint "1": variable "b"
Old Value = <No value>
New Value = 5
Line = 3
KNAX7032-I Stop in the script "test.ash".
4: c=10
Current: 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
Current：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**

```
Line：行番号
Current：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。

  ```
  Line：<Trap action>
  Current：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。

  ```
  Current：<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"」
Current: 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**

```
Line: 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
Current: 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
Current: val=1
(adshdb) step
…
KNAX7032-I Stop in the script "test.ash".
7: num=2
Current: 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
Current: funcA
(adshdb) step
KNAX7032-I Stop in the script "test.ash".
2:   echo "funcA"
Current: 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
Current: val=1
(adshdb) next
…
KNAX7032-I Stop in the script "test.ash".
7: num=2
Current: 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
Current: funcA
(adshdb) next
…
KNAX7032-I Stop in the script "test.ash".
9: echo $num
Current: 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"
Current: echo
(adshdb) finish
KNAX7036-I Run till exit of current function.
…
Num  Function  File:Line
0    <main>    test.ash:9
KNAX7032-I Stop in the script "test.ash".
9: echo $num
Current: 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"
Current: echo
(adshdb) return
KNAX7037-I Exit the current function? (y or n)
y
KNAX7068-I Commands are skipped until end of function.
…
Num  Function  File:Line
0    <main>    test.ash:9
KNAX7032-I Stop in the script "test.ash".
9: echo $num
Current: 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，「3.9.2　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。

6.2.13 ジョブ定義スクリプトの情報を表示する（info コマンド）

デバッグ中のジョブ定義スクリプトの情報を表示するコマンドは，info コマンドです。info の次に表示対象を指定すると，さまざまな情報を表示できます。info コマンドの短縮形は "i" です。指定できる info コマンドと表示対象の組み合わせを次に示します。

- 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。
- info coverage
  デバッグ途中のカバレージ情報を表示します。
- info functions
  関数情報を表示します。
- info jobsteps
  ジョブステップ情報を表示します。
- info signals
  シグナル情報を表示します。
- info variables
  変数情報を表示します。

注意事項

info コマンドに表示対象を指定しなかった場合，または指定できる表示対象以外を指定した場合は，エラーメッセージが出力されます。

6.2.14 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
```

6.2.15 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，「3.8　カバレージ情報を取得する」を参照してください。

6.2.16 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
```

6.2.17 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
```

6.2.18 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，「3.9.2　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　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 [(Step local)]
…
```

- **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（Step local）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[(Step local)]
```

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 と aaa[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。

```
XXX = xxx(Step local)
YYY = /yyy/yyy(Step local)
AAA = aaa
BBB = /bbb/bbb
```

6.2.20 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。
- 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
```

- 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。

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 属性を指定し，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
```

6.2.21 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]
```

6.2.22 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
```

6.2.23 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。停止してから二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。
二回目以降に 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"
```

6.2.24 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 Working directory: /home/xxx/work/
(adshdb) exec cat test.txt
aaa bbb ccc
```

6.2.25 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
```

6.2.26 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。

```
Available commands:
break      cd         continue   delete     exec
finish     help       info       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とコマンド

環境ファイルに環境設定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などについて定義できます。また，環境設定コマンドを指定して環境変数を定義します。
この章では，パラメーターとコマンドの記述形式と詳細について説明します。

7.1 パラメーターとコマンドの記述形式

環境ファイルで設定するパラメーターとコマンドの記述形式について説明します。

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。

7.1.1 パラメーターの記述形式

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。
- 環境設定パラメーターでは，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。

7.1.2 コマンドの記述形式

コマンドの記述形式を次に示します。

△$_0$**コマンド名**△$_1$**環境変数名=環境変数値**

- コマンドは，#-adsh_conf とは別の行に記述します。
- コマンドには，1 行で 1 個の環境変数を記述します。
- 既存の環境変数を参照して設定することはできません。

7.2　パラメーターとコマンドの一覧

環境ファイルに設定するパラメーターとコマンドについて説明します。

7.2.1　パラメーターの一覧

JP1/Advanced Shell の環境ファイルに設定するパラメーターを次の表に示します。

表 7-1　JP1/Advanced Shell の環境ファイルに設定するパラメーター

パラメーター名	定義内容	パラメーター指定	指定できる個数
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。	任意	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。		
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。		
BATCH_CVR	カバレージ採取の一括有効化機能を有効にします。		
CHILDJOB_SHEBANG【Windows，Linux 限定】	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
COMMAND_CONV_ARG【Windows，Linux 限定】	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。		
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※1	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※2	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※2	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。		
OUTPUT_STDOUT【Windows，Linux 限定】	ルートジョブの標準出力の出力先を定義します。		
PATH_CONV	絶対パスのパス名を変換する規則を定義します。		255
PATH_CONV_ACCESS【Windows，Linux 限定】	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規則を定義します。		
PATH_CONV_ENABLE	パス変換機能を有効にします。		1
SPOOL_DIR※1，※3	スプールディレクトリのパス名を定義します。		
TEMP_FILE_DIR※1	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。		
TRACE_DIR※1	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。		
TRACE_FILE_CNT※4	トレースを出力する面数を定義します。		
TRACE_FILE_SIZE※4	トレースを出力するファイルサイズを定義します。		
TRACE_LEVEL	トレースを出力するレベルを定義します。		
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

注※ 1

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※ 2

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※ 3

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※ 4

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 コマンドの一覧

JP1/Advanced Shell の環境ファイルに設定するコマンドを次の表に示します。

表 7-2 JP1/Advanced Shell の環境ファイルに設定するコマンド

コマンド名	定義内容	コマンド指定	指定できる個数
export	adshexec コマンドの起動時に有効としたい環境変数を定義します。	任意	無制限

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。

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　～<符合なし整数> ((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。

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　～<符合なし整数> ((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。

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 ～ 1023 **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

- 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。

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

- 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。

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）【Windows，Linux 限定】

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した「#!」で開始する実行プログラムパスを定義します。

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，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。「#!」の直後から行末までをパス名の比較対象とします。

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

- 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。
- JP1/Advanced Shell - Developer からファイルを直接指定した場合，この機能を使用しても子孫ジョブとして実行できません。実行しない旨のメッセージを出力し，戻り値 0 で処理を続行します。
- 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。また，JP1/Advanced Shell - Developer では子孫ジョブとして実行できませんが，同様にファイルに対する判定は真となります。
- 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。
- 【UNIX 限定】このパラメーターを指定した場合，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場合があります。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場合は，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。

使用例

● 先頭に「#!/bin/sh」または「#!/bin/ksh」が記述された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_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コマンド名として直接指定した場合に，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_SHEBANG  "/bin/ksh -x"
```

● 先頭に「#!/usr/bin/env ksh」が記述されたファイルをコマンド名として直接指定した場合に，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_SHEBANG  "/usr/bin/env ksh"
```

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）【Windows，Linux 限定】

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，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

- 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

  （例）
  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

● 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"
```

● 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
```

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 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。

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))《全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASE¥log》

Windows の開発環境の場合　～＜パス名＞ ((1 ～ 128 バイト))《全ユーザー共通文書フォルダ¥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。

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。

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。

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。

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。

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。

OUTPUT_STDOUT パラメーター（ルートジョブの出力先を定義する）【Windows，Linux 限定】

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。

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。

PATH_CONV パラメーター（絶対パスのパス名を変換する規則を定義する）

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絶対パスのパス名を変換する規則を定義します。絶対パスとは，UNIX では「/」から始まるパスを，Windows では「" ドライブレター ":¥」から始まるパスのことです。ただし，JP1/Advanced Shell では「¥」をエスケープ文字として扱うため，Windows では「" ドライブレター ":¥¥」と指定してください。

PATH_CONV_ENABLE パラメーターで定義された変換前のパス区切り文字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ジョブ定義スクリプトの実行時にパス名 2 に置換します。また，規則に合致した文字列を含んだ，"（ダブルクォーテーション）で囲まれた文字列の中に，パス区切り文字およびディレクトリ区切り文字が含まれる場合，それらについても変換されます。

オペランド

パス名 1　～<パス名> ((1 ～ 247 **バイト**))

変換前の絶対パスを指定します。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 の絶対パス，UNIX の場合は Windows の絶対パスを指定できます。ディレクトリ区切り文字として Windows の場合は，「/」を指定します。UNIX の場合は，「¥¥」を使用します。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 2　～<パス名> ((1 ～ 247 **バイト**))

変換後の絶対パスを指定します。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。

ディレクトリ区切り文字として Windows で実行する場合は，「¥¥」を指定します。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。UNIX で実行する場合は，「/」を使用します。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） $

注意事項

- 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。
- パス名が変換されるのは，ジョブ定義スクリプトの中の "（ダブルクォーテーション）で囲まれた範囲内だけです。

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
```

PATH_CONV_ACCESS パラメーター（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規則を定義する）【Windows，Linux 限定】

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規則を定義します。

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
- 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
```

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。

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
```

SPOOL_DIR パラメーター（スプール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))《全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASE¥spool》

Windows の開発環境の場合　～＜パス名＞ ((1 ～ 128 バイト))《全ユーザー共通文書フォルダ¥Hitachi¥JP1AS¥JP1ASD¥spool》

UNIX の場合　～＜パス名＞ ((1 ～ 128 バイト))《/var/opt/jp1as/spool》

スプールディレクトリのパス名を指定します。

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。

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**))《**全ユーザー共通文書フォルダ**¥Hitachi¥JP1AS¥JP1ASE¥temp》

Windows **の開発環境の場合　～<パス名>** ((1 ～ 128 **バイト**))《**全ユーザー共通文書フォルダ**¥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。

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。

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 AP データフォルダ**¥Hitachi¥JP1AS¥JP1ASE¥trace》

Windows **の開発環境の場合　～<パス名>** ((1 ～ 128 **バイト**))《**共通 AP データフォルダ**¥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。

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。

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。

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。

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。

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。

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。

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「-h」を使った条件式
- 演算子「-G」を使った条件式
- 演算子「-L」を使った条件式
- 演算子「-O」を使った条件式
- 演算子「-ef」を使った条件式

なお，JP1/Advanced Shell でサポートしていない条件式を判定する場合にこのパラメーターで動作を指定していないときは，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}

Windows 環境の JP1/Advanced Shell でサポートしていない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}

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。

- 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。

7.4　環境設定コマンド

JP1/Advanced Shell の環境ファイルに設定するコマンドについて説明します。コマンドを設定する場合，次の点に注意してください。

- コマンドは，#-adsh_conf とは別の行に記述します。
- コマンドには，1 行で 1 個の環境変数を記述します。
- 既存の環境変数を参照して設定することはできません。

export コマンド（環境変数を定義する）

形式

export **環境変数名=環境変数値**

機能

adshexec コマンドの起動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。

環境変数値～＜任意文字列＞ ((0 ～ 1023 **バイト**))

環境変数に設定する値を指定します。
区切り文字と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（'），およびエスケープ文字（¥）だけが使用できます。
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。

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
```

8 運用時に使用するコマンド

Windows または UNIX の実行環境で使用できます。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

コマンドの記述形式を次に示します。

△$_0$コマンド名［△$_1$-オプション名［△$_1$値］］…［△$_1$-オプション名［△$_1$値］］
［△$_1$任意名］

- 最初にオプションを指定し，次に任意名を指定します。オプションの前に任意名を指定した場合は，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）。
- オプションを連続して指定する場合，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。
- オプション名にマルチバイト文字は使用できません。

8.1.1 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

JP1/Advanced Shell のシェル運用コマンドを次の表に示します。UNIX 互換コマン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。

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
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。	
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。	
adshhk	スプール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。	

注※

Windows の実行環境および UNIX で使用できます。

8.3　シェル運用コマンド

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 ～ 1005 **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 ～ 1005 **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 ～ 1005 **バイト**))
マージするファイルのパス名を指定します。

戻り値

戻り値	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。
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

戻り値	意味
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
```

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 ～ 1005 **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。

戻り値

戻り値	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
```

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］［-t［-f］［-o **ascファイルのパス名**］］［-s｛SPOOL｜PARENT｝］**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］［-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**

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。このコマンドのオプション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。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。

-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。
ジョブ定義スクリプトファイルの文法チェックだけを行い，バッチジョブは実行しません。

-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 オプションは，-t オプションの指定が必要です。

オプションを指定した場合

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は，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。

オプションを指定しない場合

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がバックアップ情報と異なるとき，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 ～ 1005 **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

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
-o オプションを指定しない場合の asc ファイル名：/home/user1/test/script1_user1.asc

-s {SPOOL ｜ PARENT}【Windows，Linux 限定】

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（Windows，Linux 限定）の指定に従って動作します。

- 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。

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 ～ 1023 **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 ～ 1022))

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。

戻り値

契機	戻り値	
	-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	－
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	1
設定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	1
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
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	－

（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
```

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。

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，削除するスプールディレクトリ名と日数を指定しておきます。指定した日数より前に実行したスプールジョブが，指定したスプール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，4095 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。

"**スプールディレクトリ名**"［,"**日数**"］

スプールディレクトリ名　～<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　～<数字> ((1 ～ 999))

指定した日数より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。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

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　～<数字> ((1 ～ 999))

指定した日数より前に実行したバッチジョブを削除します。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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

戻り値	意味
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 - 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のため，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。

使用例

● /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および /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を削除します。レポートは /home/kanrisya/hk/result.csv ファイルに保管します。

- 対象リストファイル名：/home/kanrisya/hk/target
- 対象リストファイルの記述内容："/home/user001/jp1as/spool","7"
  "/home/user999/jp1as/spool","30"

```
adshhk /home/kanrisya/hk/target  /home/kanrisya/hk/result.csv  /home/kanrisya/
hk/result.log
```

8.4 UNIX 互換コマンド

JP1/Advanced Shell では Windows および UNIX で共通のコマンドが使用できます。また，ジョブ定義スクリプトファイルにも使用できます。共通に使用できる UNIX 互換コマンドを次の表に示します。

表 8-2 UNIX 互換コマンド

コマンド名	機能概要	コマンドの格納場所
awk	テキストの加工やパターン処理をします。	【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
cat	ファイルの内容を標準出力に出力します。	
cmp	バイナリファイルの内容を比較します。	
cp	ファイルまたはディレクトリをコピーします。	
cut	各行の選択範囲を標準出力に表示します。	
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。	
diff	2 つのファイルを比較します。	
expr	式を評価します。	
find	ディレクトリ内のファイルを検索します。	
grep	ファイル内の文字を検索します。	
head	ファイルの最初の部分を表示します。	
hostname	ホスト名を表示します。設定はできません。	
ls	ファイルまたはディレクトリの内容を表示します。	
mkdir	ディレクトリを作成します。	
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。	
rm	ファイルまたはディレクトリを削除します。	
rmdir	空のディレクトリを削除します。	
sed	テキスト中の文字列を置換します。	
sleep	指定された時間だけ停止します。	
sort	テキストファイルをソートします。	
split	ファイルを分割します。	
tail	ファイルの最後の部分を表示します。	
uname	OS またはハードウェアの情報を表示します。	
uniq	ソートされたファイルから重複した行を削除します。	
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。	

UNIX 互換コマンドには，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，アクセス権限およびシンボリックリンクなどに制限事項があります。

UNIX 互換コマンドの制限事項について次の表に示します。

表 8-3 UNIX 互換コマンドの制限事項

コマンド名	制限事項
共通	• 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。 • Windows の場合，ファイルに関して次に示す制限があります。 ・ファイル名は最大 246 バイトです。 ・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。 ・ファイル名に NTFS のストリームは使用しないでください。 ・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は使用しないでください。 • 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。 • プログラムを実行する機能（awk コマンドの system 関数および find コマンドの -exec プライマリ，-ok プライマリなど）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。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。
awk	• 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。
cat	制限事項はありません。
cmp	制限事項はありません。
cp	• Windows の場合，-H オプション，-L オプションまたは -P オプションを指定して，引数にシンボリックリンクを設定したり，ディレクトリ中にシンボリックリンクがあったりしても，シンボリックリンクをサポートしていないため，リンク先を表示できません。また，-r と -R オプションは同じ動作になります。 • 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	制限事項はありません。
date	-a オプション（時刻設定）は使用できません。
diff	制限事項はありません。
expr	制限事項はありません。
find	制限事項はありません。
grep	Windows の場合，シンボリックリンクのリンク先を表示できません。
head	制限事項はありません。
hostname	制限事項はありません。
ls	• Windows の場合，シンボリックリンクのリンク先を表示できません。 • Windows の場合，-l オプションでグループ，リンク数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。「日付と時刻のプロパティ」で設定したタイムゾーンが使用されます。
mkdir	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	• Windows の場合，オーバ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，シンボリックリンクのリンク先を表示できません。 • 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 /suid/ モードは保持されません。
rm	• Windows の場合，シンボリックリンクのリンク先を表示できません。 • Windows の場合，オーバ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	制限事項はありません。

コマンド名	制限事項
sed	制限事項はありません。
sleep	制限事項はありません。
sort	制限事項はありません。
split	制限事項はありません。
tail	制限事項はありません。
uname	制限事項はありません。
uniq	制限事項はありません。
wc	制限事項はありません。

以降で説明する各 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。

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
- 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，正規表現にはメタキャラクタを使用できます。使用できるメタキャラクタを次の表に示します。

表 8-4　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	○	×
{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¥}	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。

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。
	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戻り値として返します。式 expr を指定しない場合は，0 がコマンドの戻り値となります。 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戻り値となります。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戻り値となります。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戻り値は式 expr で指定した値とは異なります。JP1/Advanced Shell でのコマンドの戻り値の扱いについては「5.9.8 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。

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。返り値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。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。返り値として正規表現 r があった場合は 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。

関数名	内容
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.
```

1. 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。
2. 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。
3. パイプが閉じられます。
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。

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。

変数名	内容
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。
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。
%	% の文字です。

エスケープ文字

-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）で囲まれた文字列にはエスケープ文字を使用できます。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（/）で囲んだパターン及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可能な値を 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

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。

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

戻り値

戻り値	意味
0	正常終了

戻り値	意味
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戻り値

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，動作を保障しません。
- 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，関連付けられているファイルのパス名を指定した場合，関連付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。

- exit 文でコマンドの戻り値を返す場合，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戻り値と 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戻り値の詳細については，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

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

```
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

```
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi grp0000 Fukuoka
hitachi group04 Hokkaido
```

```
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

```
hitachi#group01#Nagoya
HITACHI#group02#Hiroshima
hitachi#group03#Ooita
```

```
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

```
# 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

```
group01 Tokyo
group02 Yokohama
group03 Fukuoka
```

```
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

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

- file02.txt

```
hitachi#group01#Nagoya
HITACHI#group02#Hiroshima
hitachi#group03#Ooita
```

```
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

```
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

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

- file05.txt

```
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
```

```
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

```
{
    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。

```
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，コマンドの戻り値として，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

```
{
    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 の値を戻り値としてコマンドを終了します。

• file04.txt

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

• file07.txt

```
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
hitachi group03 Okinawa
```

```
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

```
{
    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

```
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
```

```
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

```
{
    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

```
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
```

```
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

```
BEGIN{
    str = "日立:hitachi"
    print "Column = " index(str, "hitachi")
}
```

```
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

```
BEGIN{
    str = "日立:hitachi"
    print "Length = " length(str)
}
```

```
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

```
BEGIN{
    str = "hitachi:日立製作所"
    print "Column  = " match(str, /製.所/)
    print "RSTART  = " RSTART
    print "RLENGTH = " RLENGTH
}
```

```
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

```
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]
    }
}
```

```
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

```
BEGIN{
    str = "hitachi:日立製作所"
    rtnstr = substr(str, 11, 2)
    print "SUBSTR = " rtnstr
}
```

```
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

```
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

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

```
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

```
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

```
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
```

```
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

```
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])
    }
}
```

```
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

```
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

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

```
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

```
BEGIN{
a = 3
b = 4
result = func01(a, b, c)
print "func01 = " result
}
function func01(x,y){
x *= x
y *= y
return x + y
}
```

```
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

```
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。

```
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

```
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。

```
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
```

● JP1/Advanced Shell 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

```
#-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

```
{
    print FILENAME,":",$0 > ENVIRON["TEMP"]
}
END{
while(getline var < ENVIRON["TEMP"])
    print var
}
```

• file12.txt

```
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。

- Windows の例

```
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 ...]] ...
```

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。

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。

-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。

戻り値

戻り値	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。

使用例

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。「△」はスペース，「→」はタブを表します。

- abc.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
ccccccccccccccc

    →       →       →

△△△△△△△△△△△△
ddddddddddddd
```

- abcdex.txt

```
aaaaaaaaaaa
△△△△△△△
△△△△△△△
△△△△△△△
△△△△△△△
△△△△△△△
bbbbbbbb




ccccccccccccc



ddddddddddddd







eeeeeeeeeeeeeeeee
    →       →
    →       →
```

● -b オプションを指定し，空行以外に行番号を付けます。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat -b abc.txt
     1→aaaaaaaaaaa

     2→bbbbbbbb

     3→△△△△△△△
     4→ccccccccccccccc

     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。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat -n abc.txt
     1→aaaaaaaaaaa
```

```
 2→
 3→bbbbbbbb
 4→
 5→
 6→ccccccccccccccc
 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。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥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。

• Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cat -w
cat: illegal option -- w
usage: cat [-bnsu] [file ...]
```

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。

● 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
```

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。

戻り値

戻り値	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

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。「△」はスペース，「→」はタブを示します。

- abc.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
ccccccccccccccc

    →       →       →

△△△△△△△△△△△
ddddddddddd
```

- abcd.txt

```
aaaaaaaaaaa

bbbbbbbb

△△△△△△△
ccccccccccccccc
△△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△
ddddddddddddd
eeeeeeeeeeeeeeeee
   →       →
   →       →
```

● -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戻り値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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cmp -w
cmp: illegal option -- w
usage: cmp [-l | -s] file1 file2 [skip1 [skip2]]
```

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。

● 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
```

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] [-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] [-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

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

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，戻り値 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

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更新時刻およびファイルアクセス時刻を保持します。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，アクセス権およびアクセス時刻を保持します。

-R | -r

ディレクトリを再帰的にコピーします。

-H

UNIX の場合，-R オプションまたは -r オプションとともに指定すると，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。
-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。
Windows の場合，指定は無視されます。

-L

UNIX の場合，-R オプションまたは -r オプションとともに指定すると，遭遇したすべてのシンボリックリンクをたどります。
-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。
Windows の場合，指定は無視されます。

-P

UNIX の場合，-R オプションまたは -r オプションとともに指定すると，すべてのシンボリックリン

クをたどりません。
-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。
Windows の場合，指定は無視されます。

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。

戻り値

戻り値	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。
- 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。

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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥cp -w
cp: illegal option -- w
usage: cp [-fip] [-Rr [-H | -L | -P]] source target
       cp [-fip] [-Rr [-H | -L | -P]] source ... directory
```

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。

● 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
```

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］［パス名 ...］
cut -c リスト［パス名 ...］
cut -f リスト［-s］［-d デリミタ］［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

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。

-c

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。

-f

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。

リスト

カラム位置または区切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。パス名を省略すると標準入力から入力します。

-s

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

デリミタで指定された先頭 1 バイトをフィールド区切文字にします。-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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

使用例

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。

- test.txt

```
123:5678:abcdef:hijkl
field1:field2:field3:filed4
sssssssssssssssssssss
```

● 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
sssssssssssssssssssss
```

● 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%¥cut -z
cut: illegal option -- z
usage: cut -b list [-n] [file ...]
       cut -c list [file ...]
       cut -f list [-s] [-d delim] [file ...]
```

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。

date コマンド（システムの日付と時刻を表示する）

形式

date［-u］［-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（世界協定時）の日付を表示します。

-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 ～ 2147483647 です。範囲外の値を指定した場合，出力する内容は保障できません。

+ 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 strftime 関数の書式指定コードが指定できます。

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。

戻り値

戻り値	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。
- 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。

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
```

● 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%¥date -a
date: illegal option -- a
usage: date [-u] [-r seconds] [+format]
```

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。

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**］［-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**］［-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

ファイルをテキストと見なして比較します。

-b

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

英大文字と英小文字を区別しません。

-s

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

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**

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。-c に続けてスペースを空けないで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。行数を指定しない場合，3 行ずつ出力します。

-C **行数**

差異の前後に行数で指定した行数ずつ出力します。

-q

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

比較するパス名を標準出力に出力します。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。-u に続けてスペースを空けないで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。行数を指定しない場合，3 行ずつ出力します。

-U 行数

比較するパス名を標準出力に出力します。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。差異の前後に行数で指定した行数ずつ出力します。

-L 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。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。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。3 つ以上指定した場合，3 つ目以降は無視します。

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，/tmp に出力します。
- Windows の場合
  共通 AP データフォルダ ¥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，/tmp に出力します。
- Windows の場合
  共通 AP データフォルダ ¥HITACHI¥JP1AS¥misc に出力します。

-r

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 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 • ! 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。

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更新日時を表示します。
2. file2 のファイル情報として，ファイル名とファイル更新日時を表示します。
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更新日時を示します。
2. file2 のファイル情報として，ファイル名とファイル更新日時を示します。
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。

戻り値

戻り値	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 オプションおよび -U 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，端末からの標準入力の入力中や比較処理の実行中に diff コマンドの実行を中断すると，一時ファイル「diff.**xxxxxxxx**（**xxxxxxxx** は任意の 8 文字の文字列）」が残るときがあります。このときは，手動で一時ファイルを削除してください。

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

△△△△△△△
ccccccccccccccc

   →       →       →

△△△△△△△△△△△△
ddddddddddddd
△△△eeeeeeeeeeee
```

- abcd.txt

```
aaAAAAAaaaa

bbBBBbbb

△△△△△△△
ccccccccccccccc
△△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△
△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△
ddddddddddddd
eeeeeeeeeeee
```

- wxy.txt

```
aaaaaaaaaaa

bbbbbbbb
xxxxxxxxxxxxxx

ccccccccccccccc
ddddddddddddd
eeeeeeeeeeee
fffffffffffffff
ggggggggg
```

- wxyz.txt

```
aaaaaaaaaaa

bbbBBBbb
xxxxxxxxxxxxxx

ccccccccccccccc
ddddddddddddd
fffffffffffffff
ggggggggg
hhhhhhhhhhhhhhhhh
```

● 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
< △△△eeeeeeeeeee
---
> eeeeeeeeeee
```

● -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
< aaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
12c12
< △△△eeeeeeeeeee
---
> eeeeeeeeeee
```

● -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
< △△△eeeeeeeeeee
---
> eeeeeeeeeee
```

● -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
< aaaaaaaaaa
---
> 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
  xxxxxxxxxxxxx

  ccccccccccccccc
  ddddddddddd
- eeeeeeeeeee
  fffffffffffffff
  ggggggggg
--- 1,10 ----
  aaaaaaaaaaa

! bbbBBBbb
  xxxxxxxxxxxxx

  ccccccccccccccc
  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
 xxxxxxxxxxxxx

 ccccccccccccccc
 ddddddddddd
-eeeeeeeeeee
 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
  xxxxxxxxxxxxx
--- 2,4 ----

! bbbBBBbb
  xxxxxxxxxxxxx
***************
*** 7,10 ****
  ddddddddddd
```

```
- eeeeeeeeeeee
  fffffffffffff
  ggggggggg
--- 7,10 ----
  ddddddddddd
  fffffffffffff
  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
 xxxxxxxxxxxxx
@@ -7,4 +7,4 @@
 ddddddddddd
-eeeeeeeeeee
 fffffffffffff
 ggggggggg
+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

  △△△△△△△
  ccccccccccccccc
! △△△△△△△△△△△△△△△△△△△
! △△△△△△△△△△△△△△△△△△△
! △△△△△△△△△△△△△△△△△
! △△△△△△△△△△△△△△△△△△△△△△△△△△△△△△△
  ddddddddddd
! eeeeeeeeeee
```

● -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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥diff -z
diff: illegal option -- z
usage: diff [-abisw] [-c[number] | -C number | -q | -u[number] | -U number]
            [-L label] file1 file2
       diff [-abirsw] [-c[number] | -C number | -q | -u[number] | -U number]
            [-L label] dir1 dir2
```

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。

● 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
```

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・式及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

優先順位の低い順に演算子を示します。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。expr1 と expr2 には式を指定します。

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返り値 2 を返します。

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。

戻り値

戻り値	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。例えば，4294967295 ＋ 1 は 0 となります。また，値として 42949672950 を指定した場合は，42949672950 という文字列で保存されます。文字列として取り出す場合（expr 42949672950 という指定をした場合），文字列としての 42949672950 が取り出され表示されます。数値として取り出す場合（expr 42949672950 + 1 という指定をした場合），42949672950 = 10011111111111111111111111111110110 であり，上位 4bit があふれているため，無視されて 11111111111111111111111111110110 となり，それに 1 を加えて 11111111111111111111111111110111 となります。これは，16 進数で fffffff7 であり 10 進数で -9 です。

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。

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
$
```

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。

オプションはダッシュ（-）と共に一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

UNIX の場合，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。
Windows の場合，-H オプションを指定しても無視されます。

-h

UNIX の場合，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。
Windows の場合，-h オプションを指定しても無視されます。

-L

UNIX の場合，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。
Windows の場合，-L オプションを指定しても無視されます。

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。-f オプションの引数のパス名に検索を開始するパス名を指定して，かつ find コマンドの引数のパス名を省略した場合，検索式の始まりを明示するために，2 つ続けたダッシュ（--）を指定する必要があります。

- 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時刻と，find が実行を開始した時刻の差が時間差で指定された分のときは真です。時刻の差は，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。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時刻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時刻と，find が実行を開始した時刻の差が時間差で指定された日のときは真です。時刻の差は，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。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，ファイルおよびディレクトリの状態（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）が最後に変更された時刻と，find が実行を開始した時刻の差が時間差で指定された分のときは真です。時刻の差は，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。n 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，ファイルおよびディレクトリの情報が最後に変更された時刻が，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，ファイルおよびディレクトリの状態（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時刻と，find が実行を開始した時刻の差が時間差で指定された日のときは真です。時刻の差は，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。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戻り値 0 で終了した場合，真です。

-follow

常に真です。
UNIX の場合，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。番号の範囲は 2147483647(0x7fffffff) までで，それ以上を指定しても 2147483647 となります。
- 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**

Windows の場合，常に偽となります。
UNIX の場合，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。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

Windows の場合は，ファイルパーミッション，所有者名，サイズ（バイト単位），最終修正時刻およびパス名を標準出力に出力します。常に真です。
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），最終修正時刻およびパス名を標準出力に出力します。常に真です。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時刻と，find が実行を開始した時刻の差が時間差で指定された分のときは真です。時刻の差は，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。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時刻と，find が実行を開始した時刻の差が時間差で指定された日のときは真です。時刻の差は，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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。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。

-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，パス名の最終修正時刻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戻り値 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，パーミッションをダッシュ（-）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。ダッシュ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
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， 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。

-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，この引数を指定できません。引数を指定した場合，エラー（find: -perm: unknown option）になります。

-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。

- サイズに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します。符号を指定しない場合は指定値を，+ を前置した場合は指定値より大きい，- を前置した場合は指定値より小さいとして扱います。n 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。
- 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（Windows では指定できません）
- 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。

戻り値

戻り値	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。

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ローカルファイルシステムに存在する core ファイルを表示します。

```
$ find . -fstype local -name '*core*'
./core
$
```

● 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。

• Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥find -w
find: illegal option -- w
usage: find [-dHhL] path ... [expression]
```

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。

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] [-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表示します。

-c
選択された行数だけ標準出力に表示します。

-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。

-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表示します。-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表示します。-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。

-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，表示はしません。

-v

パターンに一致しなかった行を表示します。

-w

指定文字列が単語として含まれている行を表示します。
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表示します。

-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表示します。

-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表示します。

-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。

-e パターン

「-」で始まるパターンを指定するときに使用します。

-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。

［パターン］

検索するパターンを指定します。

［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。

戻り値

戻り値	意味
0	正常終了。 • パターンを含む行が存在します。 • -v オプション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 オプション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

- Windows の場合，シンボリックリンクのリンク先を表示しません。
- ファイルの先頭から 8192 バイト内に表示可能な 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]ccccccccc
555555555:ABCD:111111111
ABCD
ABCD_XYZ
0000<ABCD>0000
/* ABCD */
```

● 複数ファイル指定して，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
```

● -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]ccccccccc
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

```
ABABAB
ACACACAC
ABABAB
```

- 一致しないため，何も表示されません。

```
grep -x ABA file.txt
```

- 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
012345678901234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
-XYZ
--
JJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKK
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
012345678901234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
-XYZ
555555555:ABCD:111111111
abababababababababababab
--
KKKKKKKKKKKKKKK
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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥grep -d
grep: illegal option -- d
usage: grep [-abcEGIiLlnqRrsUvwx] [-A num] [-B num] [-C[num]]
        [-e pattern] [-f file] [pattern] [file ...]
```

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。

● 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
```

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，すべてのファイルに対して処理を行ってから，オープンに失敗したファイルが 1 つでもあると戻り値 1 で終了します。

戻り値

戻り値	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
```

● 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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥head -d
head: illegal option -- d
usage: head [-count | -n count] [file ...]
```

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。

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。

戻り値

戻り値	意味
0	正常終了
1 以上	エラー終了

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
```

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] [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。

引数

-1

1 行に 1 エントリ（1 列）で出力します。

-A

「.」および「..」を除いてすべてのエントリを出力します。

-a

「.」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

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

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）のとき，修正時刻ではなく状態変更時刻を使います。

-d

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。

-F

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。

-f

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。

-h

ロングフォーマット使用時，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

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 オプションおよび -s 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

UNIX の場合，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。
Windows の場合，常に参照しているファイルの情報を出力します。

-l

次の項目をロングフォーマットで出力します。

- UNIX の場合
  アクセス権，ブロック数，所有者名，グループ名，サイズ，修正時刻，ファイル名またはディレクトリ名
- Windows の場合
  ファイル所有者のアクセス権，所有者名，サイズ（ディレクトリの場合は表示しません），修正時刻，ファイル名またはディレクトリ名

-m

ファイル名をコンマ（,）で区切って出力します。

-n

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。

-p

ディレクトリ名の直後に「/」を出力します。

-q

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

サブディレクトリを再帰的に出力します。

-r

逆順にソートして出力します。

-S

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

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，年を含む時間情報を出力します。-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

修正時刻でソートします。最新の修正が先頭になります。

-u

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）の場合，修正時刻ではなく最終アクセス時刻を使用します。

-x

横方向にソートして，複数列で出力します。

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。複数指定ができます。

戻り値

戻り値	意味
0	正常終了
1 以上	エラー終了

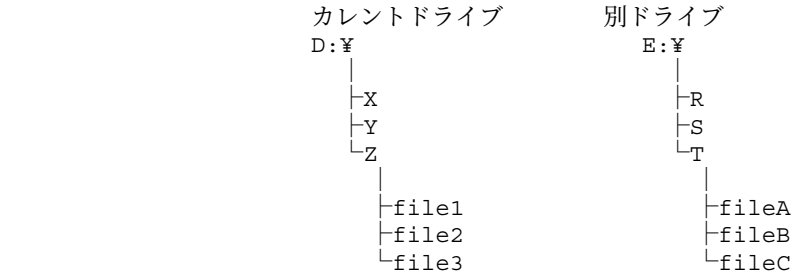
注意事項

- -1 オプション，-C オプション，-l オプション，-m オプションおよび -x 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。
- -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。
- ファイルの時刻がコマンド実行時刻より 182 日（約半年）以上昔の場合や，182 日以上未来の場合は時刻の代わりに年を表示します。
- Windows の場合，-F オプション，-c オプションおよび -u オプションは無視されます。
- Windows の場合，ユーザー名が取得できない場合は「...」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。
  - 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。なお，JP1/Advanced Shell のジョブ定義スクリプト内には，定義できません。
  - 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。512 の倍数で指定します。
  - 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，TZ 環境変数の値は日付時刻の表示に影響しません。「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。
- 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。

  例

  カレントドライブ（D:）を指定して，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。

構成




```
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

D:¥Z>
```

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%¥ls
cmp.exe      grep.exe     mv.exe       sleep.exe    cp.exe
hostname.exe spool        Adshuxpl.dll date.exe     ls.exe
rm.exe       tmp          cat.exe      mkdir.exe    rmdir.exe    uname.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
Adshuxpl.dll
cat.exe
cmp.exe
cp.exe
date.exe
grep.exe
hostname.exe
ls.exe
mkdir.exe
mv.exe
rm.exe
rmdir.exe
sleep.exe
spool
tmp
uname.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
abcde.txt    date.exe     hostname.exe rm.exe       uname.exe
abcdex.txt   ls.exe       rmdir.exe    Adshuxpl.dll cat.exe
file1.txt    mkdir.exe    sleep.exe    abc.txt      cmp.exe
mv.exe       spool        abcd.txt     cp.exe       grep.exe
tmp
```

● -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
.            cat.exe      mv.exe       spool
..           cmp.exe      grep.exe     tmp
cp.exe       hostname.exe rm.exe       uname.exe
date.exe     ls.exe       rmdir.exe    Adshuxpl.dll
mkdir.exe    sleep.exe
```

● -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
cmp.exe      mkdir.exe    rmdir.exe    uname.exe    cp.exe
grep.exe     mv.exe       sleep.exe    Adshuxpl.dll date.exe
hostname.exe spool        cat.exe      ls.exe       rm.exe
tmp
```

● -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
Adshuxpl.dll cmp.exe      mkdir.exe    rmdir.exe    uname.exe
cp.exe       grep.exe     mv.exe       sleep.exe    date.exe
hostname.exe spool        cat.exe      ls.exe       rm.exe
tmp
```

● -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 1069363
-rw------- 439808 May 09 11:31 Adshuxpl.dll
-rwx------  10752 May 09 11:34 cat.exe
-rwx------  10240 May 09 11:34 cmp.exe
-rwx------  18432 May 09 11:34 cp.exe
-rwx------  10240 May 09 11:34 date.exe
-rwx------  43008 May 09 11:33 grep.exe
-rwx------   7680 May 09 11:33 hostname.exe
-rwx------  22528 May 09 16:27 ls.exe
-rwx------   8192 May 09 11:33 mkdir.exe
-rwx------  12288 May 09 11:33 mv.exe
-rwx------  16384 May 09 11:33 rm.exe
-rwx------   8192 May 09 11:32 rmdir.exe
-rwx------   8192 May 09 11:32 sleep.exe
drwx------        May 10 08:50 spool
drwx------        May 10 08:50 tmp
-rwx------   9216 May 09 11:32 uname.exe
```

● -h オプションをロングフォーマットとともに指定して，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 1069363
-rw------- 10379780    430K May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10.5K May 09 11:34 cat.exe
-rwx------ 10379780   10.0K May 09 11:34 cmp.exe
-rwx------ 10379780   18.0K May 09 11:34 cp.exe
-rwx------ 10379780   10.0K May 09 11:34 date.exe
-rwx------ 10379780   42.0K May 09 11:33 grep.exe
-rwx------ 10379780    7.5K May 09 11:33 hostname.exe
-rwx------ 10379780   22.0K May 09 16:27 ls.exe
-rwx------ 10379780    8.0K May 09 11:33 mkdir.exe
-rwx------ 10379780   12.0K May 09 11:33 mv.exe
-rwx------ 10379780   16.0K May 09 11:33 rm.exe
-rwx------ 10379780    8.0K May 09 11:32 rmdir.exe
-rwx------ 10379780    8.0K May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9.0K May 09 11:32 uname.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 cp.exe         0 hostname.exe  0 rm.exe         0 uname.exe
0 date.exe       0 ls.exe        0 rmdir.exe      0 Adshuxpl.dll
0 mkdir.exe      0 sleep.exe     0 cat.exe        0 mv.exe
0 spool          0 cmp.exe       0 grep.exe       0 tmp
```

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -il
total 1069363
0 -rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
0 -rwx------ 10379780   10752 May 09 11:34 cat.exe
0 -rwx------ 10379780   10240 May 09 11:34 cmp.exe
0 -rwx------ 10379780   18432 May 09 11:34 cp.exe
0 -rwx------ 10379780   10240 May 09 11:34 date.exe
0 -rwx------ 10379780   43008 May 09 11:33 grep.exe
0 -rwx------ 10379780    7680 May 09 11:33 hostname.exe
0 -rwx------ 10379780   22528 May 09 16:27 ls.exe
0 -rwx------ 10379780    8192 May 09 11:33 mkdir.exe
0 -rwx------ 10379780   12288 May 09 11:33 mv.exe
0 -rwx------ 10379780   16384 May 09 11:33 rm.exe
0 -rwx------ 10379780    8192 May 09 11:32 rmdir.exe
0 -rwx------ 10379780    8192 May 09 11:32 sleep.exe
0 drwx------ 10379780         May 10 08:50 spool
0 drwx------ 10379780         May 10 08:50 tmp
0 -rwx------ 10379780    9216 May 09 11:32 uname.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，リンク数およびディレクトリサイズは表示しません。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -l
total 1069359
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9216 May 09 11:32 uname.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修正時刻ではなく状態変更時刻を表示します。Windows の場合は，-c オプションの指定を無視して修正時刻を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lc
total 1069363
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9216 May 09 11:32 uname.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修正時刻ではなく最終アクセス時刻を表示します。Windows の場合は，-u オプションの指定を無視して修正時刻を表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lu
total 1069363
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9216 May 09 11:32 uname.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
Adshuxpl.dll, cat.exe, cmp.exe, cp.exe, date.exe,
grep.exe, hostname.exe, ls.exe, mkdir.exe, mv.exe,
rm.exe, rmdir.exe, sleep.exe, spool, tmp, uname.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 1069363
-rw------- 0  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 0   10752 May 09 11:34 cat.exe
-rwx------ 0   10240 May 09 11:34 cmp.exe
-rwx------ 0   18432 May 09 11:34 cp.exe
-rwx------ 0   10240 May 09 11:34 date.exe
-rwx------ 0   43008 May 09 11:33 grep.exe
-rwx------ 0    7680 May 09 11:33 hostname.exe
-rwx------ 0   22528 May 09 16:27 ls.exe
-rwx------ 0    8192 May 09 11:33 mkdir.exe
-rwx------ 0   12288 May 09 11:33 mv.exe
-rwx------ 0   16384 May 09 11:33 rm.exe
-rwx------ 0    8192 May 09 11:32 rmdir.exe
-rwx------ 0    8192 May 09 11:32 sleep.exe
drwx------ 0         May 10 08:50 spool
drwx------ 0         May 10 08:50 tmp
-rwx------ 0    9216 May 09 11:32 uname.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
cp.exe          hostname.exe    rm.exe          uname.exe
date.exe        ls.exe          rmdir.exe       mkdir.exe
sleep.exe       cat.exe         mv.exe          spool/
cmp.exe         grep.exe        tmp/

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -alp
total 1069363
drwx------ 10379780         May 10 09:45 ./
drwx------ 10379780         May 10 10:02 ../
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
```

```
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool/
drwx------ 10379780         May 10 08:50 tmp/
-rwx------ 10379780    9216 May 09 11:32 uname.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
spool         hostname.exe date.exe     Adshuxpl.dll
sleep.exe     mv.exe       grep.exe     cp.exe
uname.exe     rmdir.exe    mkdir.exe    cmp.exe
tmp           rm.exe       ls.exe       cat.exe
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -rl
total 1069363
-rwx------ 10379780    9216 May 09 11:32 uname.exe
drwx------ 10379780         May 10 08:50 tmp
drwx------ 10379780         May 10 08:50 spool
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
```

● -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
Adshuxpl.dll rm.exe       cmp.exe      mkdir.exe
ls.exe       mv.exe       date.exe     rmdir.exe    spool
cat.exe      sleep.exe    tmp          grep.exe     cp.exe
uname.exe    hostname.exe
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lS
total 1069363
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
```

```
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780    9216 May 09 11:32 uname.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
```

● -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 1070036
0 -rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
0 -rw------- 10379780      90 May 10 10:46 abc.txt
0 -rw------- 10379780     152 May 10 15:25 abcd.txt
0 -rw------- 10379780     179 May 10 11:01 abcde.txt
0 -rw------- 10379780     146 May 10 15:22 abcdex.txt
0 -rwx------ 10379780   10752 May 09 11:34 cat.exe
0 -rwx------ 10379780   10240 May 09 11:34 cmp.exe
0 -rwx------ 10379780   18432 May 09 11:34 cp.exe
0 -rwx------ 10379780   10240 May 09 11:34 date.exe
0 -rw------- 10379780     106 May 10 13:58 file1.txt
0 -rwx------ 10379780   43008 May 09 11:33 grep.exe
0 -rwx------ 10379780    7680 May 09 11:33 hostname.exe
0 -rwx------ 10379780   22528 May 09 16:27 ls.exe
0 -rwx------ 10379780    8192 May 09 11:33 mkdir.exe
0 -rwx------ 10379780   12288 May 09 11:33 mv.exe
0 -rwx------ 10379780   16384 May 09 11:33 rm.exe
0 -rwx------ 10379780    8192 May 09 11:32 rmdir.exe
0 -rwx------ 10379780    8192 May 09 11:32 sleep.exe
0 drwx------ 10379780         May 10 08:50 spool
0 drwx------ 10379780         May 10 08:50 tmp
0 -rwx------ 10379780    9216 May 09 11:32 uname.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 1069363
-rw------- 10379780  439808 May 09 11:31:40 2011 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34:28 2011 cat.exe
-rwx------ 10379780   10240 May 09 11:34:20 2011 cmp.exe
-rwx------ 10379780   18432 May 09 11:34:35 2011 cp.exe
-rwx------ 10379780   10240 May 09 11:34:13 2011 date.exe
-rwx------ 10379780   43008 May 09 11:33:44 2011 grep.exe
-rwx------ 10379780    7680 May 09 11:33:34 2011 hostname.exe
-rwx------ 10379780   22528 May 09 16:27:40 2011 ls.exe
-rwx------ 10379780    8192 May 09 11:33:15 2011 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33:53 2011 mv.exe
-rwx------ 10379780   16384 May 09 11:33:01 2011 rm.exe
-rwx------ 10379780    8192 May 09 11:32:54 2011 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32:49 2011 sleep.exe
drwx------ 10379780         May 10 08:50:19 2011 spool
drwx------ 10379780         May 10 08:50:19 2011 tmp
-rwx------ 10379780    9216 May 09 11:32:44 2011 uname.exe
```

● -t オプションを指定して，修正時刻でソートします。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -t
date.exe     hostname.exe rmdir.exe
spool        cp.exe       mkdir.exe    sleep.exe
```

```
tmp           cat.exe      mv.exe       uname.exe    Adshuxpl.dll
ls.exe        cmp.exe      grep.exe     rm.exe

C:¥TEMP>%ADSH_OSCMD_DIR%¥ls -lt
total 1069363
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
-rwx------ 10379780    9216 May 09 11:32 uname.exe
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
```

● -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
Adshuxpl.dll cat.exe      cmp.exe      cp.exe
date.exe       grep.exe     hostname.exe ls.exe       mkdir.exe
mv.exe         rm.exe       rmdir.exe    sleep.exe    spool
tmp          uname.exe
```

● カレントドライブ（D:¥）を指定して，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

D:¥Z>
```

● 別ドライブ（E:）を指定して，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

D:¥Z>
```

● 別ドライブ（E:¥）を指定して，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

D:¥Z>
```

● 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%¥ls -o
ls: illegal option -- o
usage: ls [-1AaCcdFfghikLlmnpqRrSsTtux] [file ...]
```

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。

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
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。

-perm の指定値	同等の数値指定	説明
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。
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。

戻り値

戻り値	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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mkdir -w
mkdir: illegal option -- w
usage: mkdir [-p] [-m mode] directory ...
```

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。

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] 移動元 移動先
mv [-f] [-i] 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

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

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。

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。

戻り値

戻り値	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，オーバライド時にオーナーのアクセス権以外は表示しません。
- Windows の場合，シンボリックリンクはサポートしていません。
- 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。
- 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

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。
- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥mv -w
mv: illegal option -- w
usage: mv [-fi] source target
       mv [-fi] source ... directory
```

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。

● 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
```

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。

戻り値

戻り値	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。
- -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。

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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥rm -w
rm: illegal option -- w
usage: rm [-dfiRr] file ...
```

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。

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。

戻り値

戻り値	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

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 ...
```

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。
なお，環境変数 COLUMNS は JP1/Advanced Shell のジョブ定義スクリプト内に定義できません。

[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

[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）

エスケープ文字	意味
¥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

戻り値

戻り値	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

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
HITACHI group05 Ooita
HITACHI group06 Hiroshima
```

```
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

```
/file/{
i¥
<FILE-LINE>¥
 [FILE-BEGIN]
a¥
 [FILE-END]
}
/file/!{
c¥
<DIR-LINE>
}
```

• file02.txt

```
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.
```

```
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

```
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
```

```
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

```
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
```

```
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

```
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
```

```
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

```
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
```

```
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

```
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
```

• -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

```
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
```

```
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

```
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
```

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -n "/東京/,/沖縄/!w dir¥¥out.txt" file09.txt
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
C:¥DIR>%ADSH_OSCMD_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

```
あ い う え お あ い う
お え う い あ
```

```
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

```
#n
/ print/{
=
p
}
```

• prog01.awk

```
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。

```
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

```
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。

```
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 copyright.txt です。

• scpt03.sed

```
/^<Copyright>/{
r copyright.txt
d
}
```

• prog02.awk

```
########################################################################
<Copyright>
########################################################################
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]
    }
}
```

• copyright.txt

```
# Copyright: All Rights Reserved. Copyright (C) 2011, Hitachi, Ltd.
# Product: JP1/Advanced Shell
# License: Licensed Material of Hitachi, Ltd.
```

```
C:¥DIR>%ADSH_OSCMD_DIR%¥sed -f scpt03.sed prog02.awk
########################################################################
# Copyright: All Rights Reserved. Copyright (C) 2011, Hitachi, Ltd.
# Product: JP1/Advanced Shell
# License: Licensed Material of Hitachi, Ltd.
########################################################################
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

```
/^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

```
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.
```

```
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

```
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.
```

```
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。

- Windows の場合

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥sed -x
sed: illegal option -- x
usage: sed [-aEnru] command [file ...]
       sed [-aEnru] [-e command] ... [-f command_file] ... [file ...]
```

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。

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。

戻り値

戻り値	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
```

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，戻り値 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，戻り値 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 AP データフォルダ** ¥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，前の空白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

ソートキーに対するオプションは，2 種類あります。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。

b
: グローバルオプションは，-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，グローバルの指定を置き換えます。

グローバルな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。

戻り値

戻り値	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。

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
```

● 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 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
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
```

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［プレフィックス］］
```

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 255 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。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。

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。

プレフィックス

分割後にファイルの名前に付けるプレ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。
プレフィックス＋サフィックス
プレ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。

戻り値

戻り値	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レフィッ

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。
- 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。

• Windows の例

```
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]]
```

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。

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 オプションとともに指定した場合，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 オプションとともに指定した場合，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の指定とともに指定した場合，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 つでもあると戻り値 1 で終了します。

戻り値

戻り値	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)78901234567890123456789012345678T
00001xxx00:1234567890123(中略)78901234567890123456789012345678T
00002xxx00:1234567890123(中略)78901234567890123456789012345678T
    (中略)
00098xxx00:1234567890123(中略)78901234567890123456789012345678T
00099xxx00:1234567890123(中略)78901234567890123456789012345678T
00100xxx00:1234567890123(中略)78901234567890123456789012345678T
$ tail -b 1 block.txt
45678T
00096xxx00:1234567890123(中略)78901234567890123456789012345678T
00097xxx00:1234567890123(中略)78901234567890123456789012345678T
00098xxx00:1234567890123(中略)78901234567890123456789012345678T
00099xxx00:1234567890123(中略)78901234567890123456789012345678T
00100xxx00:1234567890123(中略)78901234567890123456789012345678T
$ tail -rb 1 block.txt
00100xxx00:1234567890123(中略)78901234567890123456789012345678T
00099xxx00:1234567890123(中略)78901234567890123456789012345678T
00098xxx00:1234567890123(中略)78901234567890123456789012345678T
00097xxx00:1234567890123(中略)78901234567890123456789012345678T
00096xxx00:1234567890123(中略)78901234567890123456789012345678T
45678T
$ tail -c 110 block.txt
2345678T
00100xxx00:1234567890123(中略)78901234567890123456789012345678T
$ tail -rc 110 block.txt
00100xxx00:1234567890123(中略)78901234567890123456789012345678T
2345678T
$
```

● 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%¥tail -z
tail: illegal option -- z
usage: tail [-r] [-b number | -c number | -n number | -number] [file ...]
```

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。

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

```
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

-a

マシン（ハードウェア）のタイプ，ノード名，OS のリリース，OS 名および OS のバージョンを表示します。

-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。

-s

OS 名を表示します。

-v

OS のバージョンを表示します。

戻り値

戻り値	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，Administrator 権限を利用して情報を取得しているため，Administrator 権限を持つユーザで使用してください。Administrator 権限を持たないユーザーが uname コマンドを実行した場合はエラーになります。
- 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
Microsoft Windows XP Professional|C:¥WINDOWS|¥Device¥Harddisk0¥Partition1
```

• 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，-m オプション，-n オプション，-r オプション，-s オプションおよび -v オプションをすべて実行した場合と同様に表示します。

• Windows の場合

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -a
Microsoft Windows XP Professional|C:¥WINDOWS|¥Device¥Harddisk0¥Partition1
 IO11418
 unknown
 5.1.2600
 X86-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
X86-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
IO11418
```

● -r オプションを指定して，OS のリリースを表示します。

```
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。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -s
Microsoft Windows XP Professional|C:¥WINDOWS|¥Device¥Harddisk0¥Partition1
```

● -v オプションを指定して，OS のバージョンを表示します。

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥uname -v
5.1.2600
```

● 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%¥uname -p
uname: illegal option -- p
usage: uname [-amnrsv]
```

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。

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［-cdu］［**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。

戻り値

戻り値	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，動作を保障しません。

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
ddddddddddddddddddddddd
ddddddddddddddddddddddddd
ddddddddddddddddddddddddeee    →重複している文字列
ddddddddddddddddddddddddeee    →重複している文字列
```

● 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:¥USR¥JP1¥oscmd¥bin>uniq file1.txt
aaaa
aaaaaaa
bbbbbbb
bbbbbbbbbbb
bcbcbcbcbcb
ddddddddddddddddddd
ddddddddddddddddddddddd
ddddddddddddddddddddddddd
ddddddddddddddddddddddddddeee

C:¥USR¥JP1¥oscmd¥bin>
```

● -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:¥USR¥JP1¥oscmd¥bin>uniq -c file1.txt
   1 aaaa
   2 aaaaaaa
   1 bbbbbbb
   4 bbbbbbbbbbb
   1 bcbcbcbcbcb
   1 ddddddddddddddddddd
   1 ddddddddddddddddddddddd
   1 ddddddddddddddddddddddddd
   2 ddddddddddddddddddddddddddeee

C:¥USR¥JP1¥oscmd¥bin>
```

● -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:¥USR¥JP1¥oscmd¥bin>uniq -d file1.txt
aaaaaaa
bbbbbbbbbbb
ddddddddddddddddddddddddddeee

C:¥USR¥JP1¥oscmd¥bin>
```

● -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:¥USR¥JP1¥oscmd¥bin>uniq -u file1.txt
aaaa
bbbbbbb
bcbcbcbcbcb
ddddddddddddddddddd
ddddddddddddddddddddddd
ddddddddddddddddddddddddd

C:¥USR¥JP1¥oscmd¥bin>
```

● 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%¥uniq -w
uniq: illegal option -- w
usage: uniq [-cdu] [input_file [output_file]]
```

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。

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。

戻り値

戻り値	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。

- Windows の例

```
C:¥TEMP>%ADSH_OSCMD_DIR%¥wc -z
wc: illegal option -- z
usage: wc [-clmw] [file ...]
```

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。

● 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:¥USR¥JP1¥oscmd¥bin>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

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シェル標準コマンド，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記述形式と詳細について説明します。

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
- 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では，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。

カバレージ情報を採取する場合は，次の点に注意してください。

- 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。また，次に示すキーワードを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
```

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［△$_1$-オプション名［△$_1$値］］…［△$_1$-オプション名［△$_1$値］］
［△$_1$任意名］

- 最初にオプションを指定し，次に任意名を指定します。オプションの前に任意名を指定した場合は，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）。
- オプションを連続して指定する場合，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 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。
- 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「9.1.2(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
＜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，「5.1.5　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

9.1.3 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。

9.1.4 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-オプション名［△1値］］…［△1-オプション名［△1値］］
［任意名］
```

- 最初にオプションを指定し，次に任意名を指定します。オプションの前に任意名を指定した場合は，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）。
- オプションを連続して指定する場合，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，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【UNIX 限定】	シグナルを受け取っ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。
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。

コマンド名	機能概要
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 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 9-4 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。	4095
#-adsh_file_temp	一時ファイルの割り当ておよび後処理をします。	4095
#-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。	1023
#-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。	1023
#-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。	4095
#-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：4095 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。	4095

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。

9.2.3 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 9-5 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。

制御文	機能概要
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。

9.2.4 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 9-6 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とは，組み込みコマンドのことです。

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できます。JP1/Advanced Shell は，組み込みコマンドとして次の 2 つの種類があります。

- 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。

. 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，. 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，「9.4　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。 • 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。

戻り値

戻り値	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

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
```

: コマンド（引数を展開する）

形式

:

機能

引数を展開します。

戻り値

戻り値	意味
0	正常終了
1	エラー終了 • 引数に ${variable:?word} または ${variable?word} の書式による変数置換を指定し，かつ variable に値が格納されていません。

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
```

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。

戻り値

戻り値	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

```
********   JOB SCOPE STDOUT    ********
functions='typeset -f'
```

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。

戻り値

戻り値	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。

注意事項

- n に signed int 型の範囲を超える値を指定すると桁あふれが発生し，あふれた値で動作します。signed int 型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
```

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。

戻り値

戻り値	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

```
********   JOB SCOPE STDOUT    ********
/tmp
```

cd コマンド（カレントディレクトリを移動する）

形式

```
cd ［ディレクトリパス］
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。

戻り値

戻り値	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 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。

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

```
********   JOB SCOPE STDOUT    ********
/var/log
/var/lib
/var/lib
```

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，whence コマンド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，組み込みコマンドおよび関数かどうかを出力します。出力形式は whence コマンドを参照してください。

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。

戻り値

-v，または -V を指定しなかった場合

戻り値	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

戻り値	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

- 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 JOBLOG 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

```
********    JOB SCOPE STDOUT     ********
/tmp
```

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。

戻り値

戻り値	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。

注意事項

- n に signed int 型の範囲を超える値を指定すると桁あふれが発生し，あふれた値で動作します。signed int 型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
```

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。-e，-E オプションをどちらも指定しなかった場合は，-e オプションが指定された場合と同じ動作をします。-e，-E 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動作します。

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。

-e

エスケープ文字を置き換えて出力する場合は，¥ 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
¥0**nnn**	アスキーコードが nnn（8 進数）の文字
¥¥	1 つのバックスラッシュ文字

エスケープ文字の置き換えは，"（ダブルクォーテーション）で囲んでください。

-E

エスケープ文字を置き換えないで出力の最後で改行したあと，標準出力に出力します。

args

引数（出力する内容）を指定します。

戻り値

戻り値	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。

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

```
********    JOB SCOPE STDOUT     ********
ja_JP.eucJP
```

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。

戻り値

戻り値	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。

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
```

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。

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，「5.1.5(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。

戻り値

戻り値	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

- Windows の場合，このコマンドで外部コマンドを実行するとプロセス ID が変わります。
- このコマンドの実行結果および呼び出した外部コマンドの実行結果は，JOBLOG 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。
- 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。
- exec コマンドは，引数に指定した外部コマンドで adshexec コマンドのプロセスを置き換えて実行します。したがって，終了コードには次に示す値が設定されます。
  - KNAX0098-I および KNAX7999-I に表示される終了コードは，exec コマンドの実行直前のバッチジョブの終了コードになります。
  - プロセスの終了コードは，引数に指定した外部コマンドの終了コードとなります。
  - JP1/AJS から adshexec コマンドを起動していた場合は，JP1/AJS のジョブの終了コードも，引数に指定した外部コマンドの終了コードとなります。
- exec コマンドの引数に外部コマンド名を指定して，ジョブステップエラーブロック内で実行すると，プロセスの終了コードはエラーとなったジョブステップの終了コードではなく，exec コマンドの引数に指定した外部コマンドの終了コードとなります。
  エラーとなったジョブステップの終了コードをプロセスの終了コードとしたい場合は，ジョブステップエラーブロック内で，exec コマンドの引数に外部コマンドを指定して実行しないでください。

使用例

● pwd コマンドを実行します。

```
exec pwd
```

● 標準出力先を file01 に切り替えます。

```
exec > file01
```

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戻り値と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 で割った余りを戻り値として，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 の補数を戻り値として，正常終了します。

戻り値

戻り値	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 signed int 型の範囲を超える値を指定すると桁あふれが発生し，あふれた値で動作します。signed int 型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。
- 「5.1.6　別プロセスでの実行【UNIX 限定】」に示す書式で，n に数字以外を指定してこのコマンドを実行した場合，戻り値 1 で正常終了します。

使用例

● 終了コード 2 でシェルを終了します。

```
exit 2
```

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 の場合，変数の名称は英大文字である必要があります。Windows では，名前がすべて大文字のシェル変数だけエクスポートできます。名前に小文字が含まれるシェル変数をエクスポートしようとしたときは，エラーメッセージを出力し，バッチジョブを終了します。
name にはエクスポートする変数名，配列名および関数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。

戻り値

戻り値	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 TMOUT に 30 を代入してエクスポートします。

```
export TMOUT=30
```

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。

戻り値

戻り値	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 KNAX6523-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

```
********   JOB SCOPE STDOUT    ********
1
```

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。

戻り値

戻り値	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。

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

```
********   JOB SCOPE STDOUT    ********
b
```

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。

戻り値

戻り値	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強制終了します。
- 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。

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 を強制終了します。

```
kill -KILL 4725
```

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。

戻り値

戻り値	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

  ```
  ********    JOB SCOPE STDOUT     ********
  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，戻り値は 0 になります。

  ジョブ定義スクリプトの内容

  ```
  a=3
  let -a
  ```

  実行ジョブの STDOUT ファイルの内容

  ```
  ********    JOB SCOPE STDOUT     ********
  0
  ```

- 「5.1.6　別プロセスでの実行【UNIX 限定】」に示す書式で，算術式を指定しないで let コマンドを実行した場合，戻り値 1 で正常終了します。

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

```
********    JOB SCOPE STDOUT     ********
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

```
********   JOB SCOPE STDOUT    ********
3
```

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
¥0**nnn**	アスキーコードが nnn（8 進数）の文字
¥¥	1 つのバックスラッシュ文字

-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。

戻り値

戻り値	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。

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

```
********   JOB SCOPE STDOUT    ********
abc
```

● コプロセス（coproc.sh）の標準入力に文字列 abc を出力します。

```
coproc.sh |&
print -p abc
```

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のパスを出力します。-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

シンボリックリンクを含まないパスを出力します。-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。
Windows の場合，-P オプションを指定しても無視されます。

戻り値

戻り値	意味
0	正常終了

戻り値	意味
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

```
********   JOB SCOPE STDOUT    ********
/tmp
```

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，シェルが実行したプロセスの出力から読み込みます。

-r

行末に ¥ があっても，次の行を継続行として読み込みません。

-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。

戻り値

戻り値	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

- 標準入力で 1 行に入力できる文字列は 4095 バイトまでです。4096 バイト目以降の文字列は受け付けません。
- 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。
- 「5.1.6　別プロセスでの実行【UNIX 限定】」に示す書式でこのコマンドを実行した場合，ファイルの終了（EOF）を検出すると，戻り値 1 でエラー終了します。#-adsh_rc_ignore コマンドや，#-adsh_step_start コマンドの successRC 属性を使用して，エラー終了とならないようにしてください。

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

```
********   JOB SCOPE STDOUT    ********
HITACHI
JP1
Advanced Shell
```

● コプロセス（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
```

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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

注意事項

- 変数または配列を指定できますが，関数には影響を与えません。
- 定義されていない変数を読み込み専用属性にできます。
- 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。

使用例

● 変数 test の属性を読み込み専用に変更します。

```
readonly test
```

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))

復帰時の戻り値を指定します。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戻り値として復帰します。n に 256 以上の値を指定した場合，n の値を 256 で割った余りを戻り値として，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 の補数を戻り値として，正常終了します。

戻り値

戻り値	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 signed int 型の範囲を超える値を指定すると桁あふれが発生し，あふれた値で動作します。signed int 型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。
- 「5.1.6　別プロセスでの実行【UNIX 限定】」に示す書式で，n に数字以外を指定してこのコマンドを実行した場合，戻り値 1 で正常終了します。

使用例

● 関数または外部スクリプトから戻り値 2 で復帰し，呼び出し元の処理を継続します。

```
return 2
```

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 ...]
```

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，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，「5.7　シェルオプション」を参照してください。

-A ｜ +A

配列に値を代入する場合に指定します。
-A または +A オプションを指定し実行すると，name に指定した配列に，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。作成できる配列の要素数は 2 から 1024 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。-A オプションは，+A と指定しても -A が指定された場合と同じ動作をします。

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

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。

戻り値

戻り値	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

● 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
```

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。

戻り値

戻り値	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
```

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。

戻り値

戻り値	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。
- 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
```

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。

戻り値

戻り値	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

```
********   JOB SCOPE STDOUT    ********
Shell:      0.00s user     0.01s system
Kids:       0.01s user     0.10s system
```

trap コマンド（シグナルを受け取ったときの動作を設定する）【UNIX 限定】

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	設定できません
			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

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。

拡張機能で **action** を登録したシグナルに関しては，シグナルに設定されている動作を表示しません。

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 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。

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。ただし，SIGTERM は指定できません。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。"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 typeset コマンドおよび特殊組み込みコマンド以外のコマンドが 0 以外の戻り値で完了すると，action に指定した動作を実行します。
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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

注意事項

- signal に指定するシグナル番号に signed int 型の範囲を超える値を指定すると桁あふれが発生し，あふれた値で動作します。signed int 型の範囲内の値を指定してください。
- Windows 環境でこのコマンドを実行した場合は，メッセージを出力して，戻り値 0 で正常終了します。シグナルに対して処理はしません。
- 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

● INT シグナルを受け取った場合，'trapped.' のメッセージを出力します。

```
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
trap -- 'echo Hangup.' HUP
trap -- 'echo trapped.' INT
```

実行ジョブの STDOUT ファイルの内容

```
********   JOB SCOPE STDOUT    ********
trap -- 'echo Hangup.' HUP
trap -- 'echo trapped.' INT
```

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。

戻り値

戻り値	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 KNAX6523-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

```
********   JOB SCOPE STDOUT    ********
0
```

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] ]
         [-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として定義します。関数内で有効な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，value 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。-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，value 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，value 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。
  +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。
  +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。

{-f ｜ +f} [-t ｜ +t]

- -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。

-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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

注意事項

- 変数または配列を指定できますが，関数には影響を与えません。
- 定義されていない変数を読み込み専用属性にできます。
- n に signed int 型の範囲を超える値を指定すると桁あふれが発生し，あふれた値で動作します。signed int 型の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。

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
```

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
stack(kbytes)	上限値	スタック領域サイズの上限

出力形式		内容
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 -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。

-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

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

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

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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

注意事項

- Windows の場合，メッセージを出力し，常に正常終了します。
- 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。

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

```
********   JOB SCOPE STDOUT    ********
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
```

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。

戻り値

戻り値	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。
- Windows の場合にこのコマンドを実行すると，サポートしていないことをメッセージに出力し，戻り値 0 を返して正常終了します。ファイルモード作成マスクは設定しません。

使用例

● ファイル作成時にユーザーの write 権限だけを設定します。

```
umask u=w
```

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，戻り値 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。

戻り値

戻り値	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
```

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］」を name に指定します。
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。

戻り値

戻り値	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
```

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。

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。
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。

戻り値

戻り値	意味
0	正常終了

戻り値	意味
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
```

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，**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。エイリアスのときはエイリアスの値を出力します。予約語や組み込みコマンドおよび関数のときは **name** を出力します。どれにも該当しない場合は，何も出力しないで戻り値 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，戻り値 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，組み込みコマンドおよび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。

出力内容	意味
name is a reserved word	**name** は予約語です。
name is a function	**name** は関数です。
name is a traced function	**name** はトレースされた関数です。
name is a shell builtin	**name** は組み込みコマンドです。
name is a special shell builtin	**name** は特殊組み込みコマンドです。
name is a shell builtin not supported	**name** は JP1/Advanced Shell で提供しないコマンドです。
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。
name is an alias for ’**エイリアスの値**’	**name** はエイリアスです。

出力内容	意味
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。
name not found	**name** はコマンド，予約語，エイリアス，組み込み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，戻り値 1 でエラー終了します。

戻り値

戻り値	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

- 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

```
********   JOB SCOPE STDOUT    ********
/bin/pwd
```

9.4 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戻り値は，環境設定パラメーターの「ADSHCMD_RC_ERROR」および「ADSHCMD_RC_SUCCESS」で変更できます。ただし，次の場合の戻り値は変更できません。

- #-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とコマンド」を参照してください。

#-adsh_file コマンド（通常ファイルの割り当ておよび後処理をする）

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，4095 個まで指定できます。

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，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 ～ 1023 **バイト**))

割り当てる通常ファイルのパスを指定します。

-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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

#-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，4095 個まで指定できます。

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，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。JP1/Advanced Shell 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。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

#-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，JP1/Advanced Shell が作成するファイル名の一部にも使用されます。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

#-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，1023 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。

戻り値

戻り値	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。

#-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

「**シェル変数名**」で示すシェル変数を，パス名を扱うシェル変数として定義します。#-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：継続行は指定できます。

JP1/Advanced Shell 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。

- "（ダブルクォーテーション）で囲まれた文字列
- 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先頭で，パスを扱う変数が使われているものを含む。

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。

戻り値

戻り値	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中のシェル変数に対してパス名を設定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。
- パス名が変換されるのは，ジョブ定義スクリプトの中の "（ダブルクォーテーション）で囲まれた範囲内だけです。

#-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

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。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になります。

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。

戻り値

戻り値	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「5.1.6　別プロセスでの実行【UNIX 限定】」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。

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
```

#-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 JP1/Advanced Shell 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。

戻り値

戻り値	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戻り値は，環境設定パラメーターで変更できません。

注意事項

- シェル標準コマンドの . コマンドと次の個所が異なります。
  - . 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。
  - . 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。
- 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。

#-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。

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，小文字は指定できません。

戻り値

戻り値	意味
0	正常終了
1	エラー終了

#-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。

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。

終了コード定義

～＜符合なし整数＞ ((0 ～ 255))

省略した場合，0 を仮定します。終了コード定義は 8 個まで指定できます。

- **終了コード**
  終了コードの場合，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。

-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。

戻り値

#-adsh_step_start，#-adsh_step_error の場合

戻り値	意味
0	正常終了
1	エラー終了

#-adsh_step_end の場合

戻り値	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
exec コマンド実行直前の終了コード	正常終了 • ステップ正常ブロック内で引数に外部コマンドを指定した exec 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戻り値は，環境設定パラメーターで変更できません。

注意事項

- ジョブステップ正常ブロックまたはジョブステップエラーブロックと，制御文（if，for，while，until，case）のブロックが交差しないようにしてください。交差した場合，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関数呼び出しや外部スクリプト展開がある場合，関数や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

```
  KNAX6523-I Command cmdA(line=4) was executed. rc=4 E-Time=0.001s
C-Time=0.000s
  KNAX6523-I Command cmdA(line=5) was executed. rc=4 E-Time=0.026s
C-Time=0.000s
  KNAX6597-I ADSH012712.S1 Step succeeded. rc=4 E-Time=0.030s 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。

使用例

● if 制御文のブロックと，ジョブステップ正常ブロックが交差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
```

9.5 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。

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
```

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
```

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
```

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
```

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

● 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
```

9.6　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。

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。

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

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 に引数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。

戻り値

戻り値	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 JOBLOG 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

```
********   JOB SCOPE STDOUT    ********
Thu Jul  7 11:06:38 JST 2011
```

実行ジョブの STDERR ファイルの内容

```
********   JOB SCOPE STDERR    ********
    0.01s real     0.00s user     0.00s system
```

10 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。

10.1 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，次に示す手順で対処します。

1. **現象の確認**
   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。各メッセージの要因と対処方法については，「11. メッセージ」を参照してください。

2. **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。
   - 問題の修正と確認
     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。

3. **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。「10.3 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。

10.2 ログ情報の種類

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 10-1 JP1/Advanced Shell 運用時に出力されるログ情報と格納先

ログ情報	格納先
ジョブ実行ログ（バッチジョブのログ）	スプールディレクトリの配下
システム実行ログ（JP1/Advanced Shell の実行ログ）	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 LOG_DIR パラメーターの指定がない場合，デフォルト値になります。
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 パラメーターで指定したディレクトリ TRACE_DIR パラメーターの指定がない場合，デフォルト値になります。

各ログ情報の詳細を次に示します。

10.2.1 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，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）【Windows，Linux 限定】※
- ユーザープログラムの標準エラー出力（stderr）
- カバレージ取得に関するメッセージ

注※

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 を指定した場合だけ出力されます。

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 ファイルを参照してください。

(1) 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，実行したジョブによって次のように異なります。

● ルートジョブ

- 標準出力は，adshexec コマンドの -s オプションおよび環境ファイルの OUTPUT_STDOUT パラメーターの指定に従って出力されます。
- 標準エラー出力は，スプール内のファイルに出力されます。
- 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，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。

● 子孫ジョブ

- 標準出力および標準エラー出力は，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。
- ジョブ実行後，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内に割り当てたプログラム出力データファイルが削除されます。
- 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。その他のジョブ実行ログの情報は出力されません。また，次に示す情報は出力されません。
  ・KNAX6380-I および KNAX7999-I メッセージ
  ・JOBLOG の次に示すヘッダ行

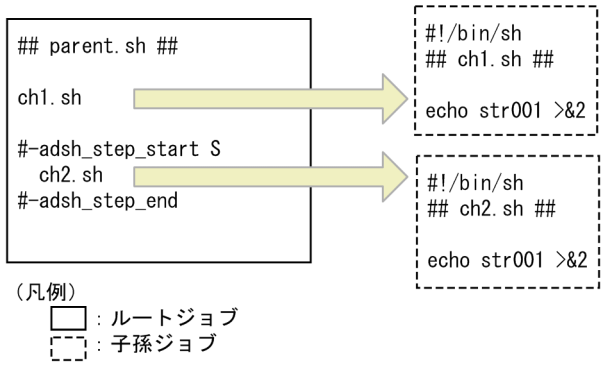
```
------------------------------------------------------------------
 JP1/Advanced Shell バージョン番号

 [Information]
   Jobid           : ジョブ識別子
   Spool Directory : スプールジョブディレクトリパス
   Date            : 実行日付
   Configure File  : 環境ファイルパス
   HostName        : ホスト名
 [JP1 Parameter]
   JP1/AJSから渡された環境変数
------------------------------------------------------------------
********  JP1/Advanced Shell MESSAGE  ********
```

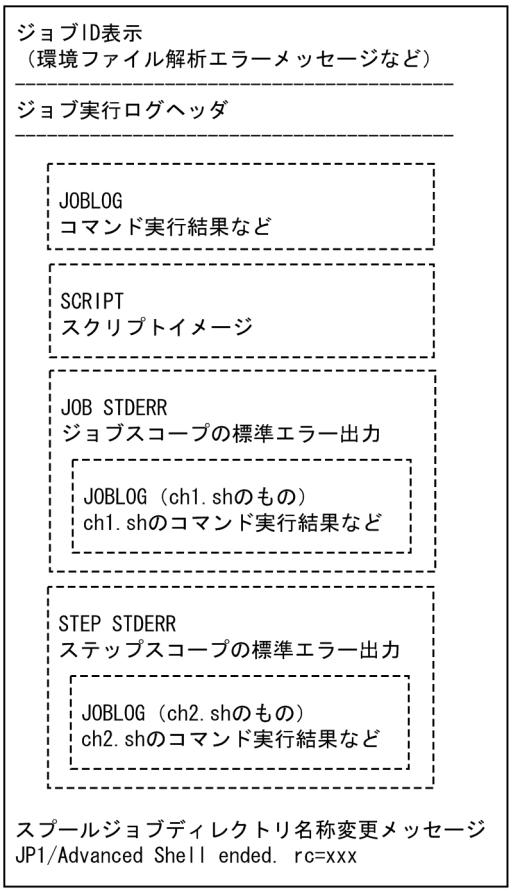
(2) ジョブ実行ログの出力例

ジョブの定義例と，そのジョブの実行後に標準エラー出力へ出力されるジョブ実行ログの例を次に示します。

● 例 1（ch1.sh と ch2.sh を定義）



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の内容，およびその中で子孫ジョブの実行結果が出力される個所を次に示します。



この場合に標準エラー出力へ出力されるジョブ実行ログの内容の具体例を次に示します。

```
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=002166
----------------------------------------------------------------
 JP1/Advanced Shell 09-51-01

 [Information]
   Jobid           : 002166
   Spool Directory : /home/user/jp1as/spool/002166/
   Date            : 2012/04/09
   Configure File  : /home/user/jp1as/adsh.ase
   HostName        : host
 [JP1 Parameter]
----------------------------------------------------------------
********  JP1/Advanced Shell MESSAGE  ********
15:42:11 002166 KNAX0091-I ADSH002166 Job started.
15:42:11 002166 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
15:42:11 002166 KNAX6520-I Command ./ch1.sh(line=2) succeeded. rc=0 E-Time=0.004s C-
Time=0.000s
15:42:11 002166 KNAX0092-I ADSH002166.S Step started.
15:42:11 002166 KNAX6520-I Command ./ch2.sh(line=5) succeeded. rc=0 E-Time=0.004s C-
Time=0.000s
15:42:11 002166 KNAX6597-I ADSH002166.S Step succeeded. rc=0 E-Time=0.005s C-Time=0.000s
15:42:11 002166 KNAX0098-I ADSH002166 Job ended. rc=0 E-Time=0.111s C-Time=0.000s

********   Script IMAGE    ********

***** /home/user/jp1as/parent.sh *****
0001 : ## parent.sh ##
0002 : ch1.sh
0003 :
0004 : #-adsh_step_start S
0005 :   ch2.sh
0006 : #-adsh_step_end
0007 :

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=002167
str001
15:42:11 002167 KNAX6571-I ADSH002167 Child job started. parent jobname=ADSH002166, parent
jobid=002166
15:42:11 002167 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
15:42:11 002167 KNAX6520-I Command echo(line=4) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
15:42:11 002167 KNAX6578-I ADSH002167 Child job ended. rc=0 E-Time=0.000s C-Time=0.000s
KNAX6597-I ADSH002166.S Step succeeded. rc=0 E-Time=0.005s C-Time=0.000s
KNAX0098-I ADSH002166 Job ended. rc=0 E-Time=0.111s C-Time=0.000s
```

ch1.sh出力

（次の図に続く）

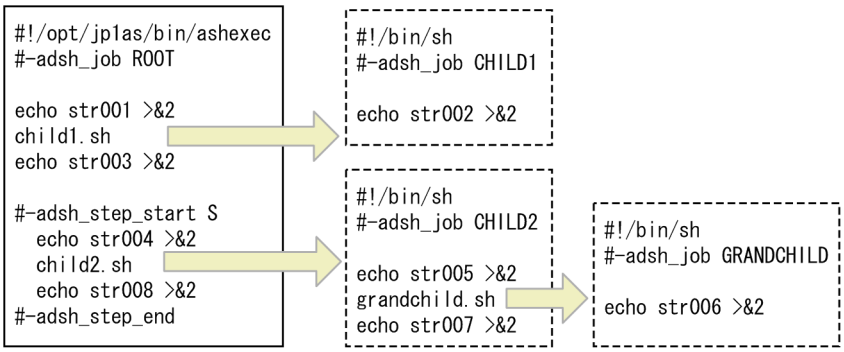
（前の図の続き）

```
******** JOBSTEP OUTPUT ********
KNAX0719-I STEP 0001,S,STDERR
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=002168
str002
15:42:11 002168 KNAX6571-I ADSH002168 Child job started. parent jobname=ADSH002166, parent
jobid=002166
15:42:11 002168 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
15:42:11 002168 KNAX6520-I Command echo(line=4) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
15:42:11 002168 KNAX6578-I ADSH002168 Child job ended. rc=0 E-Time=0.000s C-Time=0.000s

KNAX6380-I Job name will be added to spool job directory. spool job directory=
"/home/user/jp1as/spool/002166-ADSH002166/"
KNAX7999-I JP1/Advanced Shell ended. rc=0
```

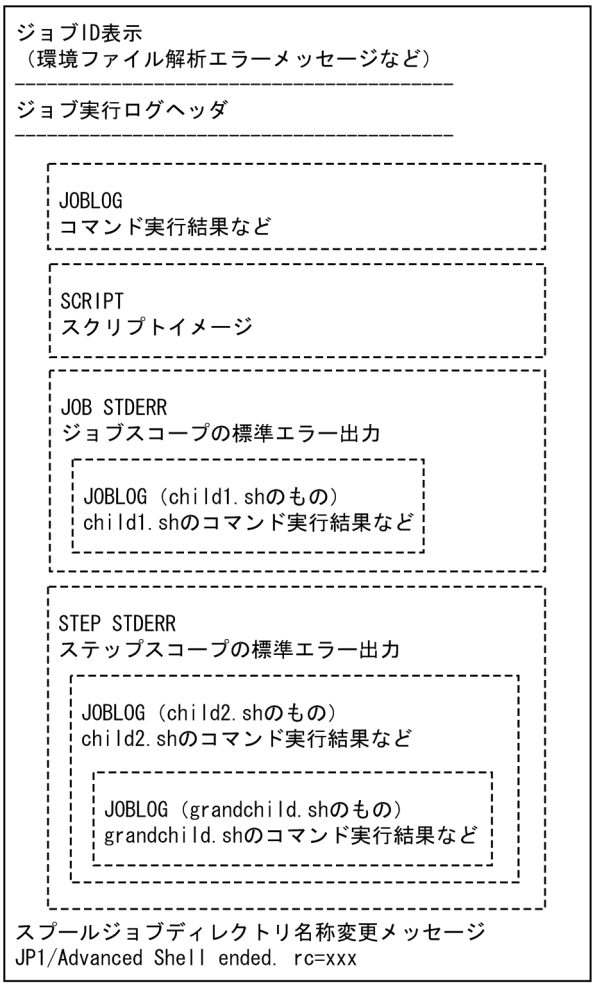
ch2.sh出力

● 例 2（child1.sh，child2.sh，grandchild.sh を定義）



（凡例）
：ルートジョブ
：子孫ジョブ

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の内容，およびその中で子孫ジョブの実行結果が出力される個所を次に示します。



この場合に標準エラー出力へ出力されるジョブ実行ログの内容の具体例を次に示します。

```
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=000632
----------------------------------------------------------------
 JP1/Advanced Shell 09-51-01

 [Information]
   Jobid           : 000632
   Spool Directory : /home/user/jp1as/spool/000632/
   Date            : 2012/03/30
   Configure File  : /home/user/jp1as/adsh.ase
   HostName        : host
 [JP1 Parameter]
----------------------------------------------------------------
********  JP1/Advanced Shell MESSAGE  ********
11:07:13 000632 KNAX0091-I ROOT Job started.
11:07:13 000632 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
11:07:13 000632 KNAX6520-I Command echo(line=4) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000632 KNAX6520-I Command ./child1.sh(line=5) succeeded. rc=0 E-Time=0.036s C-
Time=0.000s
11:07:13 000632 KNAX6520-I Command echo(line=6) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000632 KNAX0092-I ROOT.S Step started.
11:07:13 000632 KNAX6520-I Command echo(line=9) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000632 KNAX6520-I Command ./child2.sh(line=10) succeeded. rc=0 E-Time=0.083s C-
Time=0.020s
11:07:13 000632 KNAX6520-I Command echo(line=11) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000632 KNAX6597-I ROOT.S Step succeeded. rc=0 E-Time=0.084s C-Time=0.020s
11:07:13 000632 KNAX0098-I ROOT Job ended. rc=0 E-Time=0.122s C-Time=0.020s

********   Script IMAGE    ********

***** /home/user/jp1as/root.sh *****
0001 : #!/home/user/jp1as/BJEX-BASE/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
0013 :

***** CONVERSION INFORMATION *****
```

（次の図に続く）

（前の図の続き）

```
********   JOB SCOPE STDERR    ********                          CHILD1出力
str001
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=000633
str002
11:07:13 000633 KNAX6571-I CHILD1 Child job started. parent jobname=ROOT, parent jobid=000632
11:07:13 000633 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
11:07:13 000633 KNAX6520-I Command echo(line=4) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000633 KNAX6578-I CHILD1 Child job ended. rc=0 E-Time=0.000s C-Time=0.000s
str003
KNAX6597-I ROOT.S Step succeeded. rc=0 E-Time=0.084s C-Time=0.020s
KNAX0098-I ROOT Job ended. rc=0 E-Time=0.122s C-Time=0.020s

******** JOBSTEP OUTPUT ********
KNAX0719-I STEP 0001,S,STDERR                                    CHILD2出力
str004
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=000634
str005
KNAX7901-I adshexec waits for all asynchronous processes at the end of the job.
KNAX0724-I Jobid was assigned. Jobid=000635                  GRANDCHILD出力
str006
11:07:13 000635 KNAX6571-I GRANDCHILD Child job started. parent jobname=CHILD2, parent
jobid=000634
11:07:13 000635 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
11:07:13 000635 KNAX6520-I Command echo(line=4) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000635 KNAX6578-I GRANDCHILD Child job ended. rc=0 E-Time=0.000s C-Time=0.000s
str007
11:07:13 000634 KNAX6571-I CHILD2 Child job started. parent jobname=ROOT, parent jobid=000632
11:07:13 000634 KNAX7901-I adshexec waits for all asynchronous processes at the end of the
job.
11:07:13 000634 KNAX6520-I Command echo(line=4) succeeded. rc=0 E-Time=0.001s C-Time=0.000s
11:07:13 000634 KNAX6520-I Command ./grandchild.sh(line=5) succeeded. rc=0 E-Time=0.062s C-
Time=0.010s
11:07:13 000634 KNAX6520-I Command echo(line=6) succeeded. rc=0 E-Time=0.000s C-Time=0.000s
11:07:13 000634 KNAX6578-I CHILD2 Child job ended. rc=0 E-Time=0.065s C-Time=0.010s
str008                                                           CHILD2出力

KNAX6380-I Job name will be added to spool job directory. spool job directory=
/home/user/jp1as/spool/000632-ROOT/
KNAX7999-I JP1/Advanced Shell ended. rc=0
```

(3) 子孫ジョブが異常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。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。

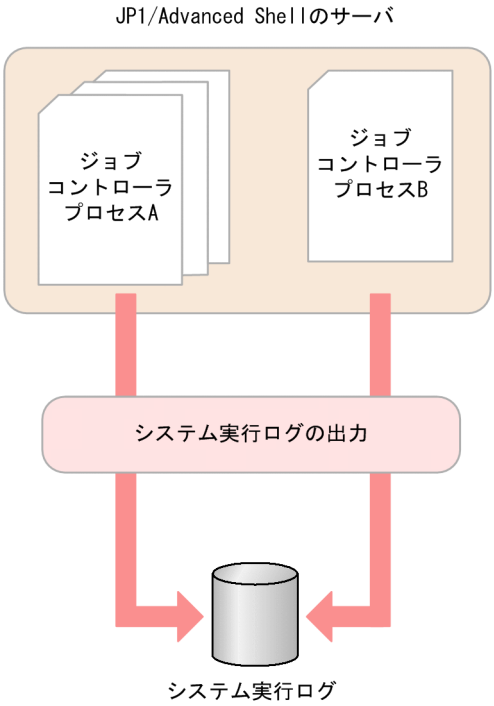
10.2.2 システム実行ログ

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。

(1) 機能

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 10-1　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，「11.2　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。

(2) 形式

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=2011-02-19T17:51:59.324+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0004-I, msg="JobID=000018, JP1NBQSQueueName=, JP1JobID="
seqnum=2, date=2011-02-19T17:51:59.325+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0900-I, msg="Script file ""test.ash"" is parsing."
seqnum=3, date=2011-02-19T17:51:59.395+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0909-I, msg="Script file ""test.ash"" was parsed. code=0"
seqnum=4, date=2011-02-19T17:51:59.397+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0010-E, msg="Message queue open failed. reason=No such file or directory"
seqnum=5, date=2011-02-19T17:51:59.398+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0091-I, msg="JOB1 Job start."
seqnum=6, date=2011-02-19T17:51:59.591+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX1600-I, msg="JOB1 Job allocated."
seqnum=7, date=2011-02-19T17:51:59.594+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0092-I, msg="JOB1.STEP1 Step start."
seqnum=8, date=2011-02-19T17:51:59.789+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX1601-I, msg="JOB1.STEP1 Step allocated."
seqnum=9, date=2011-02-19T17:51:59.806+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0097-I, msg="JOB1.STEP1 Step ended. rc=0 E-Time=0.212s C-Time=0.010s"
seqnum=10, date=2011-02-19T17:51:59.991+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0092-I, msg="JOB1.STEP2 Step start."
seqnum=11, date=2011-02-19T17:52:00.189+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX1601-I, msg="JOB1.STEP2 Step allocated."
seqnum=12, date=2011-02-19T17:52:00.237+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0097-I, msg="JOB1.STEP2 Step ended. rc=0 E-Time=0.246s C-Time=0.020s"
seqnum=13, date=2011-02-19T17:52:00.391+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0092-I, msg="JOB1.STEP3 Step start."
seqnum=14, date=2011-02-19T17:52:00.589+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX1601-I, msg="JOB1.STEP3 Step allocated."
seqnum=15, date=2011-02-19T17:52:00.602+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0097-I, msg="JOB1.STEP3 Step ended. rc=0 E-Time=0.210s C-Time=0.000s"
seqnum=16, date=2011-02-19T17:52:00.790+09:00, pgmid=Exec_c, jobid=18, pid=96992,
msgid=KNAX0098-I, msg="JOB1 Job ended. rc=0 E-Time=0.668s C-Time=0.030s"
```

システム実行ログに出力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 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

(3) 注意事項

システム実行ログについては，次の点に注意してください。

- 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。
- 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。

10.2.3 トレースログ

トレースログは，トラブルが発生した場合にトラブル発生の経緯を調査したり，各処理の処理時間を測定したりするために採取するログ情報です。

トレースログは，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下の AdshTrace_n.log（n は面数）に出力されます。ファイル面数は TRACE_FILE_CNT パラメーター，ファイルサイズは TRACE_FILE_SIZE パラメーターで指定します。

トレースログは，次の 5 種類がありますが，環境設定パラメーターで指定して変更できるのは，先頭の 2 種類です。記載しているトレースログの出力先は，デフォルト値を示します。

- JP1/Advanced Shell の実行環境のトレースログ（Windows および UNIX）
  次の出力先，面数，およびファイルサイズを環境設定パラメーターで変更できます。
  - Windows での出力先：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥trace
  - UNIX での出力先：/opt/jp1as/trace
  - 面数：≪ 4 ≫ ((1 ～ 64))
  - ファイルサイズ：≪ 2 ≫ ((1 ～ 16))
- JP1/Advanced Shell - Developer のエディタ以外
  次の出力先，面数，およびファイルサイズを環境設定パラメーターで変更できます。
  - 出力先：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥trace
  - 面数：≪ 4 ≫ ((1 ～ 64))
  - ファイルサイズ：≪ 2 ≫ ((1 ～ 16))
- JP1/Advanced Shell のカスタムジョブのトレースログ
  - 出力先：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASV¥trace
  - 面数：1
  - ファイルサイズ：1
- JP1/Advanced Shell - Developer のエディタのトレースログ
  - 出力先：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥adshedit¥trace
  - 面数：1
  - ファイルサイズ：1
- JP1/Advanced Shell，JP1/Advanced Shell - Developer の共通コマンドのトレースログ
  - 出力先：**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥misc¥trace
  - 面数：4
  - ファイルサイズ：2

注意事項

- トレースログファイルは，TRACE_FILE_CNT パラメーターで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。
- 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バッチジョブがないことを確認してから行ってください。

10.3 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 10-2 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
稼働情報	JP1/Advanced Shell が出力する稼働情報	システム実行ログ，トレースログ
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，エラーログ

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，「10.4 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。

10.3.1 稼働情報

採取が必要な稼働情報を次の表に示します。

表 10-3 採取が必要な稼働情報

種類	内容	出力先
システム実行ログ	JP1/Advanced Shell の内部情報	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。デフォルト値については，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。
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。デフォルト値については，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。 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。

10.3.2 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，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 10-4 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。 • システムドライブ :¥Documents and Settings¥All Users¥Application Data¥Microsoft¥Dr Watson¥drwtsn32.log

種類	内容	出力先
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，システムによって次のディレクトリに出力されます。 • adshexec 起動時のカレントディレクトリ

10.3.3 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 10-5 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 -e オプションで指定した環境ファイル
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。

10.4 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，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 コマンドの形式，機能，引数，使用方法，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 の場合と UNIX の場合とで異なることがあります。

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］
```

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。

このコマンドは，障害発生時の実行ユーザーの権限で実行する必要があります。

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，ADSHyyyymmddhhmmss です。yyyymmdd は adshcollect コマンドを起動した日付，hhmmss は 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となります。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，ADSHyyyymmddhhmmss.tar です。yyyymmdd は adshcollect コマンドを起動した日付，hhmmss は 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となります。

● -f 定義ファイル名

任意の定義ファイル名を指定します。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。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。指定がない場合は，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。環境変数 ADSH_ENV の指定もない場合は，SPOOL_DIR，LOG_DIR および TRACE_DIR はデフォルト値になります。

使用方法

JP1/Advanced Shell を標準的な構成でインストールした場合のコマンドの使用手順を次に示します。

1. 環境ファイルには障害が発生したときの環境ファイルを指定してください。また，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。詳細については，「2.6　JP1/Advanced Shell の環境情報を設定する」を参照してください。
2. 定義ファイルは必要に応じて，任意の場所に作成してください。core または dump を採取する必要がない場合は作成および定義は不要です。
3. JP1/Advanced Shell の運用で使用しないディレクトリを保守情報出力先ディレクトリに指定して adshcollect コマンドを実行します。
   保守情報のファイルが作成されます。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が必要です。JP1/Advanced Shell で使用しないディレクトリである必要があります。
   /tmp ディレクトリを使用した例を次に示します。

```
/opt/jp1as/maintenance/adshcollect /tmp
```

定義ファイルと環境ファイルの設定

定義ファイルと環境ファイルで adshcollect コマンドの動作を定義します。定義ファイルは，キーワードと値をスペースで区切って記述します。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指定の関係について次の表に示します。定義ファイルには CORE および DUMP 以外を指定できません。例えば，コメントなどは記入できません。

表 10-6　定義ファイルのキーワードと指定の関係

キーワード	内容	指定	複数指定	ワイルドカード
DUMP【Windows 限定】	ワトソンログなど，Windows で採取したいダンプ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ってください。	任意	○※	×
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。	任意	○	×

（凡例）
　○：指定できます。
　×：指定できません。

注※
　DUMP キーワードは，16 個まで指定できます。

環境ファイルのキーワードと指定の関係について次の表に示します。なお，環境ファイルに指定がない場合，パス名のデフォルト値の各情報を採取します。

表 10-7 環境ファイルのキーワードと指定の関係

キーワード	内容	指定	複数指定	ワイルドカード
SPOOL_DIR	スプールディレクトリのパス名 パス名のデフォルト値は次のとおりです。 • 実行環境の場合【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥spool • 開発環境の場合【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥spool • 実行環境の場合【UNIX 限定】 /var/opt/jp1as/spool Windows の場合で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ってください。	任意	×	×
LOG_DIR	システム実行ログ出力先ディレクトリのパス名 パス名のデフォルト値は次のとおりです。 • 実行環境の場合【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥log • 開発環境の場合【Windows 限定】 **全ユーザー共通文書フォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥log • 実行環境の場合【UNIX 限定】 /opt/jp1as/log Windows の場合で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ってください。	任意	×	×
TRACE_DIR	トレースログ出力先ディレクトリのパス名 パス名のデフォルト値は次のとおりです。 • 実行環境の場合【Windows 限定】 **共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥trace • 開発環境の場合【Windows 限定】 **共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥trace • 実行環境の場合【UNIX 限定】 /opt/jp1as/trace Windows の場合で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ってください。	任意	×	×

（凡例）

×：指定できません。

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"
```

定義ファイルと環境ファイルの指定例【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
```

ディスク使用量

● 保守情報を出力した圧縮ファイル

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（Windows 環境の場合 DUMP ファイル，UNIX 環境の場合 CORE ファイルになります）

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 10-8　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	［環境ファイルの SPOOL_DIR］¥adsh.jobid 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，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	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） 環境ファイルで変更できます。パス名のデフォルト値については，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	［環境ファイルの LOG_FILE_SIZE］×（n ＋ 1）MB	○
	［環境ファイルの LOG_DIR］¥AdshLog.conf 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，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	1KB 程度	○
UXPL の実行ログ	**共通 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASE¥uxpl¥spool¥uxpllog［n］.txt（n は面数：最大 2 面）	5MB	○
	共通 AP データフォルダ ¥Hitachi¥JP1AS¥JP1ASD¥uxpl¥spool¥uxpllog［n］.txt（n は面数：最大 2 面）	5MB	○

ファイルの種類	ファイル名	最大サイズ	採取
	共通 AP データフォルダ ¥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） 環境ファイルで変更できます。パス名のデフォルト値については，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	［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 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASV¥trace¥AdshTrace_1.log	1MB	○
トレースログ（エディタ）	**共通 AP データフォルダ** ¥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 AP データフォルダ** ¥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 AP データフォルダ** ¥Hitachi¥JP1AS¥JP1ASD¥adshedit¥trace¥adshedit.txt	ユーザー環境の設定による	○
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	△
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 は adshcollect コマンドを起動した日付，hhmmss は adshcollect コマンドを起動した時刻）	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。

表 10-9　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 デフォルト値あり。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，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	1KB 程度	○

ファイルの種類	ファイル名	最大サイズ	採取
システム実行ログ	［環境ファイルの LOG_DIR］/AdshLog.log デフォルト値あり。環境ファイルで変更できます。 ［環境ファイルの LOG_DIR］/AdshLog_[n].log （n は面数） パス名のデフォルト値については，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	［環境ファイルの LOG_FILE_SIZE］×（n ＋ 1）MB	○
	［環境ファイルの LOG_DIR］/AdshLog.conf デフォルト値あり。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，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	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） デフォルト値あり。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，「表 10-7 環境ファイルのキーワードと指定の関係」を参照してください。	［環境ファイルの TRACE_FILE_SIZE］× nMB	○
core ファイル	定義ファイルの CORE の core 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	△
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	/etc/environment	ユーザー環境の設定による	○
環境情報ファイル	ADSHTMPyyyymmddhhmmss.txt （yyyymmdd は adshcollect コマンドを起動した日付，hhmmss は adshcollect コマンドを起動した時刻）	ユーザー環境の設定による	○
tar のログ	ADSHTARyyyymmddhhmmss.txt （yyyymmdd は adshcollect コマンドを起動した日付，hhmmss は adshcollect コマンドを起動した時刻）	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おく必要があります。
- 保守情報出力先ディレクトリに，出力ファイル，および一時ファイルを作成します。このため，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。

11 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。

11.1 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。

11.1.1 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
  タイプコードです。メッセージに対する処置の指標を示します。タイプコードには，次の表に示す種類があります。

表 11-1 タイプコード

タイプコード	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。

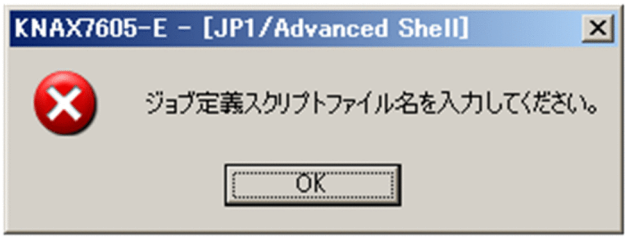
なお，ジョブ実行ログに出力するメッセージの場合，次のように時刻およびジョブ識別子が付加されます。

```
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。

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。

図 11-1　メッセージ用ダイアログボックス



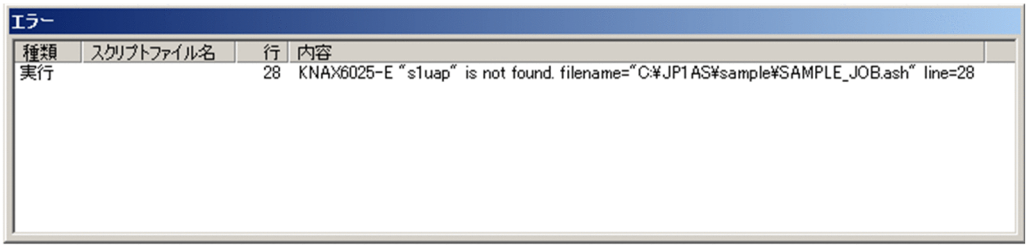
メッセージ用ダイアログボックスでは，メッセージの内容によってメッセージの種類がアイコンで表示されます。メッセージ用ダイアログボックスでのアイコンの意味を次の表に示します。

表 11-2　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。

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 11-2　エラーウィンドウ



11.1.2　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，各々が 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**エラー詳細**と記載している，エラーの詳細については，「11.4　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。

11.1.3 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 11-3 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	基本稼働情報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
4414 ～ 4429	スプールジョブの操作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 ～ 7599	開発環境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

11.2 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。

表 11-4 JP1/Advanced Shell が出力するメッセージの出力先

メッセージ ID の範囲	stdout	stderr	JOBLOG	システム実行ログ	GUI
KNAX0001-E	−	○	○	○	○
KNAX0004-I	−	−	−	○	−
KNAX0020-E	−	○	−	○	○
KNAX0030-E，KNAX0031-E	−	○	−	−	○
KNAX0091-I，KNAX0092-I	−	−	○	○	−
KNAX0098-I	−	○	○	○	−
KNAX0101-E	−	○	○	○	○
KNAX0236-E，KNAX0238-E	−	○	−	○	−
KNAX0299-E	−	○	○	○	○
KNAX0300-I	−	○	−	−	−
KNAX0301-E ～ KNAX0307-E	−	○	−	−	○
KNAX0308-E ～ KNAX0309-I	−	○	−	−	−
KNAX0310-E ～ KNAX0336-E	−	○	−	−	○
KNAX0338-E	−	○	−	−	−
KNAX0401-E ～ KNAX0702-E	−	○	−	−	○
KNAX0703-E	−	○	−	○	○
KNAX0704-E ～ KNAX0706-E	−	○	−	−	○
KNAX0719-I	−	○	−	−	−
KNAX0720-E ～ KNAX0723-E	−	○	−	−	○
KNAX0724-I	−	○	−	−	−
KNAX0800-E	−	○	−	−	○
KNAX0801-E	−	○	−	−	−
KNAX0802-E	−	○	−	−	○
KNAX0803-E	−	○	○	○	○
KNAX0804-E	−	○	−	−	○
KNAX0805-E	−	○	○	○	○
KNAX1600-I ～ KNAX1605-I	−	−	○	−	−
KNAX1632-E	−	−	○	−	○
KNAX1910-E，KNAX1911-E	−	−	○	○	○
KNAX2201-E ～ KNAX2205-E	−	○	−	○	○
KNAX2206-E	−	○	−	−	○
KNAX2207-E	−	○	−	○	○
KNAX2208-E ～ KNAX2213-E	−	○	−	−	○
KNAX2214-E，KNAX2400-E	−	○	−	○	○

メッセージ ID の範囲	stdout	stderr	JOBLOG	システム実行ログ	GUI
KNAX2499-E	―	○	―	○	―
KNAX4414-E ～ KNAX4429-E	―	○※1	―	―	―
KNAX6000-E ～ KNAX6099-E	―	―	○	―	◎
KNAX6100-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	―	○	―	○	○
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	―	―	○	―	◎
KNAX6380-I	―	○	―	―	―
KNAX6381-E	―	○	―	―	○
KNAX6382-I	―	○	―	―	―
KNAX6399-E，KNAX6400-E	―	―	○	―	○
KNAX6401-E	―	―	○	―	◎
KNAX6402-E	―	―	○	―	○
KNAX6403-E	―	―	○	―	◎
KNAX6404-E	―	―	○	―	○
KNAX6405-E ～ KNAX6407-E	―	―	○	―	◎
KNAX6408-E	―	―	○	―	○
KNAX6409-I，KNAX6410-I	―	―	○	―	―
KNAX6411-E ～ KNAX6413-E	―	―	○	―	○

メッセージ ID の範囲	stdout	stderr	JOBLOG	システム実行ログ	GUI
KNAX6507-I ～ KNAX6511-I	－	－	○	○	－
KNAX6512-I	－	○	－	－	－
KNAX6513-W	－	－	○	○	－
KNAX6514-W	－	○	－	○	－
KNAX6520-I	－	－	○	○	－
KNAX6521-E ～ KNAX6522-E	－	－	○	○	◎
KNAX6523-I	－	－	○	○	－
KNAX6530-E，KNAX6531-E	－	－	○	－	◎
KNAX6540-I	－	－	○	○	－
KNAX6541-E，KNAX6542-E	－	－	○	○	○
KNAX6550-I ～ KNAX6553-I	－	－	○	○	－
KNAX6560-I ～ KNAX6586-E	－	－	○	○	－
KNAX6588-E	－	○	－	－	－
KNAX6590-E ～ KNAX6592-E	－	－	○	○	－
KNAX6593-E	－	○	○	○	○
KNAX6594-E	－	－	○	○	－
KNAX6596-E	－	○	○	○	○
KNAX6597-I	－	○	○	○	－
KNAX6598-E，KNAX6599-E	－	－	○	○	○
KNAX6701-W	－	－	○	○	－
KNAX6710-I	－	－	○	－	－
KNAX6711-E，KNAX6712-E	－	－	○	－	◎
KNAX6713-E	－	－	○	○	○
KNAX6714-E，KNAX6715-E	－	－	○	－	◎
KNAX6716-W	－	－	○	－	－
KNAX6800-I※2，KNAX6801-I※2	－	－	－	－	－
KNAX6803-I ～ KNAX6806-I	－	－	○	－	－
KNAX6810-E ～ KNAX6812-E	－	－	○	○	○
KNAX6813-E	－	－	○	－	◎
KNAX6814-E，KNAX6815-E	－	－	○	○	○
KNAX6997-E	－	－	○	○	○
KNAX6998-E	－	－	○	○	◎
KNAX6999-E	－	－	○	○	○
KNAX7000-E ～ KNAX7005-E	－	○	－	○	○
KNAX7006-W ～ KNAX7009-I	－	○	－	○	－
KNAX7010-E	－	○	－	○	○
KNAX7011-I，KNAX7012-W	－	○	－	○	－
KNAX7013-E，KNAX7014-E	－	○	－	○	○

メッセージ ID の範囲	stdout	stderr	JOBLOG	システム実行ログ	GUI
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	−	○	−	○	−
KNAX7090-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	−	−	−	−	○

メッセージ ID の範囲	stdout	stderr	JOBLOG	システム実行ログ	GUI
KNAX7119-E	―	○	○	○	○
KNAX7120-W	―	―	―	○	―
KNAX7121-E ～ KNAX7125-E	―	○	○	○	○
KNAX7600-E ～ KNAX7773-E	―	―	―	―	○
KNAX7800-I ～ KNAX7880-E	○	―	―	―	―
KNAX7892-I ～ KNAX7897-E	―	○	―	―	―
KNAX7900-I	―	―	―	―	○※3
KNAX7901-I	―	○	○	○	―
KNAX7999-I	―	○	―	―	―

（凡例）

stdout：標準出力を示します。

stderr：標準エラー出力を示します。

JOBLOG：ジョブ実行ログを示します。

システム実行ログ：システム実行ログを示します。

GUI：GUI で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します。

◎：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

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※ 2

スクリプトイメージファイルに出力します。

注※ 3

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。

注意事項

KNAX6000-E から KNAX6100-E，KNAX6710-I から KNAX6713-E，KNAX6998-E のメッセージに関する注意事項です。

1. 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 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: `

2. 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。
3. 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
```

11.3 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

Memory shortage. DETAIL= **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しか出力されません。

(S) 処理を終了します。

(O) システム管理者に連絡します。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

Jobid=**JP1/Advanced Shell のジョブ識別子** , JP1NBQSQueueName= **環境変数値** , JP1JobID=**JP1 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。

JP1/Advanced Shell のジョブ識別子：JP1/Advanced Shell がバッチジョブに付与したジョブ識別子

環境変数値：バッチジョブの JP1NBQSQueueName 環境変数の値

JP1 ジョブ番号：バッチジョブの JP1/Advanced Shell が与えた JP1 ジョブ番号

(S) 処理を続行します。

KNAX0020-E 【Windows 限定】

Initialization of adshexecsub process failed. (function= **関数名** , detail= **詳細情報**)

adshexecsub プロセスの初期化に失敗しました。**関数名**と**詳細情報**を示します。要因として，次のことが考えられます。

- adshexecsub.exe を直接実行しています。

(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**と**詳細情報**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。

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 Job started.

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** Step started.

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 Job ended. 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。次の両方の条件に合致する場合，**終了コード**には exec コマンドの実行直前のバッチジョブの終了コードが出力されます。

- 環境ファイルに CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）を 1 つも指定していない
- exec コマンドでバッチジョブの実行を終了する

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 An error occurred during execution of job.

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。

KNAX0236-E

String size of environment variable " **環境変数名** " value is invalid.

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) 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

Invalid char is included in environment variable " **環境変数名** " value.

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) 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。

KNAX0299-E

Internal error occurred. Detail= **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しか出力されません。

(S) 処理を終了します。

(O) システム管理者に連絡します。

KNAX0300-I

Usage: **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

Option " **オプション名** " value is not specified.

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

Option " **オプション名** " was unknown.

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

SCRIPT file name is not specified.

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。

KNAX0304-E

Option **オプション名** is specified with other exclusive option.

オプション名で示すオプションに不当な値があります。

(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。

KNAX0305-E

Option **オプション名** is invalid.

オプション名で示す不正なオプションを指定しています。

(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。

KNAX0306-E

Option " **オプション名** " value is invalid.

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

The option is not specified.

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

Option " **オプション名 1**" and " **オプション名 2**" cannot be specified at the same time.

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 version is **バージョン文字列** .

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

Too many operands.

指定したオプションが多過ぎます。

(S) 処理を中断します。

(O) コマンドとオプションを正しく指定します。

KNAX0311-E

One or more necessary options for **コマンド名** are missing.

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

String size of option " **オプション名** " value is invalid.

オプション名の指定値が長過ぎます。

(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。

KNAX0338-E

Invalid char is included in option " **オプション名** " value.

オプション名で示すオプションの指定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) オプション名を正しく指定して再入力します。

KNAX0401-E

Config file open failed. 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

Config file read error. 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

Config file name too long.

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

Failed to get host name. 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。

(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** " is not an ordinary file.

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。

KNAX0411-E

Line size exceeds limits. line= **行番号**

環境ファイルの**行番号**で示す行の解析で改行コードが見つかりません。

(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。

KNAX0431-E

Parameter name is invalid. 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

Parameter value of " **パラメーター名** " is invalid. 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

Parameter value of " **パラメーター名** " is missing. line= **行番号**

環境ファイルの**行番号**で示すパラメーターにオペランドがありません。

(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。

KNAX0434-E

Parameter " **パラメーター名** " is specified multiple. 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

Number of parameter " **パラメーター名** " exceed limit. 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

String size of parameter " **パラメーター名** " value is invalid. 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

Value scope of parameter " **パラメーター名** " value is invalid. 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

Wrong char is included in parameter " **パラメーター名** " value. 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

Directory " **パラメーター名** " must be specified in absolute path. 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

Directory of " **パラメーター名** " is not exist. 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。

(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

Parameter value of " **パラメーター名** " is not directory. 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

There are too many value for " **パラメーター名** ". 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

Default directory of " **パラメーター名** " does not exist: " **ディレクトリ名** "

デフォルトのディレクトリがありません。

(S) 処理を終了します。

(O) 動作環境を見直してください。

KNAX0446-E

Default directory of " **パラメーター名** " is not valid directory: " **ディレクトリ名** "

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

The required directory " **ディレクトリ名** " does not exist.

必須の**ディレクトリ名**がありません。

(S) 処理を終了します。

(O) 環境ファイルを見直すか，または動作環境をチェックしてください。

KNAX0450-E

The required directory " **ディレクトリ名** " is not valid directory.

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】

An error occurred while getting a default directory: " **パラメーター名** "

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】

Parameter value of " **パラメーター名** " is specified multiple. 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。

KNAX0700-E

Could not create the job directory. reason= **エラー詳細**

スプール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) 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。

KNAX0701-E

" **ファイル名** " open failed. 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。

(S) 処理を終了します。

(O) 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。

KNAX0702-E

Joblog file I/O error. 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) 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。

KNAX0703-E

" **ファイル名** " does not exist.

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

Failed to get the current date.

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。

KNAX0705-E

Failed to get host name.

ホスト名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

Failed to create " **ファイルパス** ". 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) 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。

KNAX0719-I

STEP **ステップ番号** , **ジョブステップ名** , **出力先の内容**

このメッセージのあとに，ジョブステップの**出力先の内容**を出力します。

(S) 処理を続行します。

KNAX0720-E

Failed to open jobid file. 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) 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。

KNAX0721-E

Jobid file I/O error. 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) 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。

KNAX0722-E

Failed to allocate Jobid.

ジョブ固有のジョブ識別子を付与できません。

(S) 処理を終了します。

(O) システム管理者に連絡します。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ます。

KNAX0723-E

Failed to lock jobid file. 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。

(S) 処理を終了します。

(O) 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。

KNAX0724-I

Jobid was assigned. Jobid= **ジョブ識別子**

ジョブ識別子を割り当てました。

(S) 処理を続行します。

KNAX0800-E

Failed to create the {.sysout | sysout.ini} . 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) 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。

KNAX0801-E

The .sysout lock error. 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。

(S) 処理を終了します。

(O) 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。

KNAX0802-E

The {.sysout | sysout.ini} open failed. 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) 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。

KNAX0803-E

The {.sysout | sysout.ini} I/O error. 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，一部の出力先にしか出力されないことがあります。

(S) 処理を終了します。

(O) システム管理者に連絡します。**エラー詳細**の原因を取り除いて再実行します。

KNAX0804-E

Failed to get the current time used for STARTTIME in the {.sysout | sysout.ini} .

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

Failed to get the current time used for ENDTIME in the {.sysout | sysout.ini} .

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 Job allocated.

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** Step allocated.

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

ファイルパス is deleted.

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

ファイルパス is not deleted due to " **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

Length of environment variable value exceeded the limit. (environment variable name= **環境変数名**)

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。

KNAX1910-E

Invalid C-Time.

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

Invalid E-Time.

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

The message is too long. msgid= **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

Output to JOBLOG failed. msgid= **メッセージ番号**

メッセージ番号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に異常がないかどうかを見直してください。

KNAX2204-E

Output to stdout failed. msgid= **メッセージ番号**

メッセージ番号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。スプールジョブディレクトリに異常がないかどうかを見直してください。

KNAX2205-E

Output to stderr failed. msgid= **メッセージ番号**

メッセージ番号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。スプールジョブディレクトリに異常がないかどうかを見直してください。

KNAX2206-E

Output to log failed. msgid= **メッセージ番号**

メッセージ番号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に異常がないかどうかを見直してください。

KNAX2207-E

Output to trace failed. msgid= **メッセージ番号**

メッセージ番号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に異常がないかどうかを見直してください。

KNAX2208-E

Initialization of log failed. 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。

(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に異常がないかどうかを見直してください。

KNAX2209-E

Initialization of trace failed. code= **保守情報**

トレースの初期化に失敗しました。

(S) 処理を終了します。

(O) システム管理者に連絡します。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に異常がないかどうかを見直してください。

KNAX2210-E

The file size or the number of files used by log is invalid.

システム実行ログ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。環境ファイルのシステム実行ログに対する指定を見直してください。

KNAX2211-E

The file size or the number of files used by trace is invalid.

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。

KNAX2212-E

The file path used by log is invalid.

システム実行ログ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。環境ファイルのシステム実行ログに対する指定を見直してください。

KNAX2213-E

The file path used by trace is invalid.

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を見直してください。

KNAX2214-E

Failed to get the current time.

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

Initialization is not completed. output= **出力先** , msgid= **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

Message id " **メッセージ番号** " is not defined.

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。

KNAX4414-E

Size of the spool directory path is invalid. 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

Data format error. 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

Over max line size. 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

File was swapped. 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

Not file. 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

I/O error. 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

Fatal error. error=" **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

Not specify the {target-list-file | report-file | log-file} .

コマンドに必須のオペランドが指定されていません。

target-list-file：対象リストファイル

report-file：レポートファイル

log-file：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

Invalid days.

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

Unable to get the date for the start of the job execution. Spool job was not deleted. 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。

KNAX4427-W

The invalid spool job directory was skipped.

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 - 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。
- 名称のジョブ名部分が 32 バイト以上である。

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

Spool job was removed. 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

The error occurred.

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。

KNAX6000-E

Builtin command " **コマンド名** " is not supported.［ 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

Shell option " **シェルオプション名** " is not supported.［ 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

Specified variable " **シェル変数名** " is unusable.［ 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** " is not an identifier.［ filename=" **ファイル名** " line= **行番号**］

不当な変数をジョブ定義スクリプトに指定しました。

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

" **不当な値または数値以外の値** " is bad number.［ 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

Too many arguments.［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

Bad substitution.［filename=" **ファイル名** " line= **行番号**］

置換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) エラーとなった個所の変数置換またはコマンド置換の指定を見直し，ジョブ定義スクリプトを修正します。

KNAX6007-E

" **配列名** " subscript out of range.［filename=" **ファイル名** " line= **行番号**］

配列の要素番号が上限を超えています。

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) 処理を終了します。

(O) 配列の要素番号を 0 から 1023 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ます。

KNAX6008-E

" **変数名** " is read only.［ 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ます。

KNAX6009-E

" **オプション** " is unknown option.［ 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

" **オプション** " is bad option.［ 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。

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

" **シグナル番号または名称** " is bad signal.［ 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

" **マスク** " is bad mask.［ 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

Bad limit - **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

Specified variable " **変数名** " is not set.［ 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

Argument is not specified.［ 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** " requires argument.［ 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

" **単語** " unexpected.［ filename=" **ファイル名** " line= **行番号**］

制御文の指定が誤っています。

単語：構文不正となった単語

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

" **単語** " unmatched.［ filename=" **ファイル名** " line= **行番号**］

制御文で必要な単語の対応が誤っています。

単語：構文不正となった単語

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

Unexpected EOF.［ 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

" **ディレクトリパス** " is bad directory.［ 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

Too many <<'s.［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

Too many redirections.［filename=" **ファイル名** " line= **行番号**］

リダイレクトで使用しているファイル識別子が多過ぎます。

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

Cannot ｛break ｜ continue｝.［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

" **関数名** " is invalid function name.［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** " is not found.［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

" **コマンド名** " cannot execute - **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

Specified value in the ｛break ｜ continue｝command is more than number of loop nests **ループのネスト数** .［ 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

" **コマンド名** " is not a builtin.［ 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

Coprocess already exists.［ 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

Cannot change directory. **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

Shell variable HOME is not set. [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

Shell variable OLDPWD is not set. [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

Don't know current directory. [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

No coprocess.［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

" **ファイル識別子** " is invalid file descriptor - **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

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

Arguments must be process ids " **プロセス ID**".［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

Missing options argument.［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

Missing name argument.［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

Nothing to shift.［ 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

Missing].［ 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

Specified conditional expression is invalid.［ 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

Readonly variable is specified in the expression.［ 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

Command to execute in another process is not specified.［ 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。

KNAX6047-E

Specified limit " **上限値** " is invalid.［ 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

Exceeds allowable limit.［ filename=" **ファイル名** " line= **行番号**］

上限値を変更する権限がありません。

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) 上限値を変更しないで処理を継続します。

(O) 実行ユーザーに上限値を変更する権限を割り当て，ジョブ定義スクリプトを再実行します。または，ジョブ定義スクリプトを修正します。

KNAX6049-E

Cannot change limit.［ 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

" **変数名** " is null or not set.［ 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

Specified " **リダイレクト文字** " is invalid.［ filename=" **ファイル名** " line= **行番号**］

コマンド置換で指定された**リダイレクト文字**が誤っています。

リダイレクト文字：指定されたリダイレクト文字

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

Cannot open input file " **入力ファイル名** ".［ 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

Cannot create pipe.［ 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

Failed to create process.［ 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

Failed to send the signal. 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。

KNAX6057-E

Cannot allocate memory.［ 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。

KNAX6059-E

Too many files open in shell.［ 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。

KNAX6060-E

Timed out waiting for input.［filename=" **ファイル名** " line= **行番号**］

シェル変数 TMOUT に指定した時間に達したため，タイムアウトが発生しました。

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) ジョブ定義スクリプトを修正します。

KNAX6061-E

Here document missing.［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

Cannot ｛create ｜ open｝temporary file **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またはオープンできません。

一時ファイル名：作成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

Error writing **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

Failed to generate the temporary variable " **変数名または算術式** " in the calculating formula.［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

Invalid " **変数名または算術式** " is used in the calculating formula.［ 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。

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。

KNAX6066-E

Format of expression is invalid.［ 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

Expression is zero divisor.［ 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。

KNAX6070-E

No last job.［ 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

Cannot get current directory - **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。

KNAX6075-E

Could not finish (dup) redirection **リダイレクト文字** : **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

Arguments changed since last call. [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

Invalid option specified. [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

No closing quote. [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

Here document " **ラベル** " unclosed.［ 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

Cannot dup **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

Cannot open **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

Cannot create **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

Cannot be set to the specified signal " **シグナル番号または名称** ". [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。

KNAX6098-E

Error occurred. 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

Internal error occurred. 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

Failed to execute the command " **コマンド名** ". 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。KNAX7999-I で出力された**終了コード**は無効となり，このメッセージで示す**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の名前

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) 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ます。

KNAX6200-I

Usage: **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

Option **オプション名** value is not specified.

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

Option **オプション名** was unknown.

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

Option " **オプション値** " value format is invalid.

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

Option " **オプション名** " is specified with other exclusive option.

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

Asc file name is not specified.

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

Too many operands.

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

One or more operands are missing.

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

Line number is out of script line number range.

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 file " **ファイル名** " open failed. 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 file " **ファイル名** " lock failed. 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 file " **ファイル名** " read failed. 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 file " **ファイル名** " format is invalid. detail= **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 file " **ファイル名** " update failed. 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 file " **ファイル名** " unlock failed. 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

Script " **ジョブ定義スクリプト名** " and asc file "**asc ファイル名** " have different script data.

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) 次のどれかの対処を実施してください。

1. **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。
2. **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

Merging two asc files is started. Output file " **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

Main file=" **ファイル名 1**" sub file=" **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

Merging two asc files ended. Output file "**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

Two asc files " **ファイル名 1**" and " **ファイル名 2**" have different script data.

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

Internal error occurred. detail= **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

Nest of control statement exceeded limit.

if 文，case 文，for 文，while 文，until 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，if 文，case 文，for 文，while 文，until 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

The version of " **バージョン** " do not match.

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

Failed to get the time information. 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

Failed to get the information of " **ファイル名** ". 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。

KNAX6230-E

Asc file " **ファイル名** " backup failed. reason=" **エラー詳細** "

ファイル名で示す asc ファイルのバックアップファイルの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 ファイルのバックアップファイルの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。
ファイル名に示すファイルがすでにある可能性があります。

KNAX6231-E

Merge asc file is the same as base asc file.

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

Failed to get the user name.

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。
Linux 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

" **ファイル名** " already exists.

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

" **ファイル名** " is not regular file.

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

Script file is the same as **ファイル種別** .

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 file"：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。
- "temporary asc file"：一時 asc ファイルのパス名が重複しました。
- "backup asc file"：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 file " **ファイル名** " rename failed. 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 file " **ファイル名** " remove failed. 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 file " **ファイル名** " seek failed. 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

Input asc file is the same as output asc file.

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）が同一パスの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 ファイルのパス名は，入力 asc ファイルと異なるパス名を指定します。

KNAX6242-I

Asc file " **ファイル名** " has been updated.

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 file " **ファイル名** " is restored from backup asc file.

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 file " **ファイル名** " truncate failed. 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。

KNAX6301-E

Coverage option is specified in batch coverage mode.

一括してカバレージ情報を採取してバッチジョブを実行している場合，-t オプションを指定した 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

Asc file name size is exceeded limits for batch coverage function.

一括してカバレージ情報を採取する場合，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

Failed to duplicate file descriptors. 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

Initialization failed.

JP1/Advanced Shell 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

Can not confirm the existence of a script file. filename=" **ファイル名** " 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

Can not read script file. filename=" **ファイル名** " 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

Line is truncated because line size limit is exceeded. filename=" **ファイル名** " 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

Script file is empty. filename=" **ファイル名** "

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

SCRIPT file in spool I/O error. 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

" **項目名** " is incorrect. 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

" **項目名** " is not specified. 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

" **オプション名** " value is incorrect. 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

" **オプション名** " given multiple. 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

" **文字列** " is not complete descriptions. 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

" **コマンド名** " order in the commands is incorrect. 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

The instruction length limit is exceeded. 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

Incorrect specification of continuation lines. 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

Instruction given is incorrect. " **コマンド名** " 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

" **項目名** " number limit is exceeded. 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

Script file error. filename=" **ファイル名** " 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 で示されるエラー情報

(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

In this position, "-run {abnormal | always}" can not be specified. 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 command failed. line= **行番号**

export コマンドの指定が不正です。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

(S) 処理を終了します。

(O) 環境ファイルを修正します。

KNAX6324-E

Can not convert to realpath. before filename=" **変換前ファイル名** " 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

Step across the block. 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

Not defined the start of the step. 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

Step nested definitions. 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** " nested definitions. 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

The instruction has been changed from the first script. 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

The script called recursively. 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

The step has not closed, but the script finished.

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

Can not confirm the existence of " **ファイル名** ". 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

Job name will be added to spool job directory. 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。

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

Failed to change the name of the spool job directory. 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。変更後のスプールジョブディレクトリがすでにある場合などに発生します。

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：ジョブ名

(S) 処理を続行します。

KNAX6382-I

Spool job directory is stored unchanged. 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：変換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。

KNAX6399-E

Fatal error. **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

File_env " **ファイル環境変数定義名** " failed to allocate.

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

File_env " **ファイル環境変数定義名** " does not exist. file path= **ファイルパス**

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。

KNAX6402-E

File_env " **ファイル環境変数定義名** " does not have read permission. file path= **ファイルパス**

ファイル環境変数定義名で指定した**ファイルパス**のファイルに読み込み権限がありません。

(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

Temp ID " **一時ファイル識別名** " is already defined.

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

File_env " **ファイル環境変数定義名** " putenv failed.

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

File_env " **ファイル環境変数定義名** " file does not exist.

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

File_env " **ファイル環境変数定義名** " stat error. 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。

KNAX6407-E

File name of File_env " **ファイル環境変数定義名** " is already allocated directory.

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

File_env " **ファイル環境変数定義名** " create error. 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

File **ファイル環境変数定義名** is allocated as " **処理指定値** ". 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。

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 exist を，ファイルが存在しない場合は not exist を出力します。

(S) 処理を続行します。

KNAX6410-I

File **ファイル環境変数定義名** is deallocated as " **処理指定値** ". 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。

(S) 処理を続行します。

KNAX6411-E

File env " **ファイル環境変数定義名** " cannot set shell variable. detail= **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

Cannot get shell variable. detail= **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

File_env " **ファイル環境変数定義名** " cannot be restored to former shell variable. detail= **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。

KNAX6507-I

ジョブ名 . **ジョブステップ名** Step was skipped because of run attribute.

#-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** Step was skipped because previous step or command ended abnormally.

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** Step was not executed because of script context.

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

adshexec cannot set shell variable **シェル変数名** because this variable is readonly. 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

Value of step local variable PATH is passed from outside the step. 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

adshexec start by custom job.

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。

KNAX6513-W

Stderr file for child job cannot be created because spool job directory of root job does not exist. jobid= **子孫ジョブ識別子**

ルートジョブのスプールジョブディレクトリが存在しないため，ジョブ識別子が**子孫ジョブ識別子**である子孫ジョブの標準エラー出力を出力するファイルを作成できません。ルートジョブのスプールジョブディレクトリは削除されたか，または不正な名称に変更されています。
この子孫ジョブの標準エラー出力は，親プロセスのジョブの標準エラー出力に出力されます。

(S) 処理を続行します。

(O) この子孫ジョブの標準エラー出力の内容を確認する場合は，親プロセスのジョブの標準エラー出力を確認します。

KNAX6514-W

Stderr file for child job cannot be renamed because spool job directory of root job does not exist. jobid= **子孫ジョブ識別子**

ルートジョブのスプールジョブディレクトリが存在しないため，ジョブ識別子が**子孫ジョブ識別子**である子孫ジョブの標準エラー出力を出力するファイルの名称を変更できません。ルートジョブのスプールジョブディレクトリは削除されたか，または不正な名称に変更されています。
ルートジョブのスプールジョブディレクトリが不正な名称に変更されている場合，この子孫ジョブの標準エラー出力を出力したファイルの名称は「TMP_CHILDJOB_ プロセス ID」という名称になります。

(S) 処理を続行します。

KNAX6520-I

Command **コマンド名** (line= **行数**) succeeded. 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。

KNAX6521-E

Command **コマンド名** (line= **行数**) failed. 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

Command **コマンド名** (line= **行数**) ended abnormally because the command received a signal. 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。

KNAX6523-I

Command **コマンド名** (line= **行数**) was executed. rc= **終了コード** E-Time= **実行時間** s C-Time=**CPU 時間** s

コマンド名で示す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ジョブ定義スクリプトの行数

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：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。

KNAX6530-E

It is a combination of the option that cannot be specified. 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

A specified upper bound of " **項目名** " that can be specified in the **スコープ** is exceeded. 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：job または step

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

Another process script(line= **行数**) succeeded. 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。

行数：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数

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

Another process script(line= **行数**) failed. 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。

行数：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スクリプトの行数

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

Another process script(line= **行数**) ended abnormally because the script received a signal. 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。

行数：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数

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。

KNAX6550-I

Command **コマンド名** for **機能名** succeeded. 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。次のどちらかが出力されます。

command replace：コマンド置換機能

trap action：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。OS の API を使って取得した参考値です。

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

(S) 処理を続行します。

KNAX6551-E

Command **コマンド名** for **機能名** failed. 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。

command replace：コマンド置換機能

trap action：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

Command **コマンド名** for **機能名** ended abnormally because the command received a signal. 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。

command replace：コマンド置換機能

trap action：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。

KNAX6553-I

Command **コマンド名** for **機能名** was executed. 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。次のどちらかが出力されます。

command replace：コマンド置換機能
trap action：trap コマンドのアクション

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。

KNAX6560-I

Another process script for **機能名** succeeded. 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。

command replace：コマンド置換機能
trap action：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

Another process script for **機能名** failed. 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。

command replace：コマンド置換機能
trap action：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

Another process script for **機能名** ended abnormally because the script received a signal. 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。

command replace：コマンド置換機能

trap action：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 Child job started. 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。

KNAX6578-I

子孫ジョブ名 Child job ended. 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。次の両方の条件に合致する場合，**終了コード**には exec コマンドの実行直前の子孫ジョブの終了コードが出力されます。

- 環境ファイルに CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）を 1 つも指定していない
- exec コマンドでバッチジョブの実行を終了する

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。

KNAX6584-I

Job is stopped because the command that stops script was executed.

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，次のどちらかの場合に出力されます。

- exit コマンド，および関数外での 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
- 特殊組み込みコマンドのエラー，および JP1/Advanced Shell で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

(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

Job is stopped because #-adsh_job_stop command applied.

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

Job is stopped because an error which cannot continue the script occurred.

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。

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。

KNAX6590-E

An error occurred **機能名** . detail= **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

adshexec ended abnormally because of termination request signal. 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

adshexec ended abnormally because of termination request control signal. 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

adshexec received terminate request.

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

adshexec ended abnormally because of operation for terminating process.

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。

KNAX6596-E

ジョブ名 . **ジョブステップ名** Step failed. 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** Step succeeded. 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 error occurred. name=**API 名称** , reason= **原因** , detail= **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

Internal error occurred. reason= **原因** , detail= **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。

KNAX6701-W

Cannot use job object.

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

Builtin command " **コマンド名** " ［with option " **オプション名** "］is not supported on the current platform. It returns " **戻り値** ".［ filename=" **ファイル名** " line=" **行番号** "］

現在のプラットフォームでは，JP1/Advanced Shell でサポートしない組み込みコマンドを実行しようとしました。または，現在のプラットフォームでは，JP1/Advanced Shell でサポートしないオプションを指定して組み込み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

戻り値：組み込みコマンドの戻り値

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) 戻り値に示す値を組み込みコマンドの戻り値として処理を継続します。

KNAX6711-E

Builtin command " **コマンド名** " ［with option " **オプション名** "］is not supported on the current platform. Job failed.［ filename=" **ファイル名** " line= **行番号**］

現在のプラットフォームでは，JP1/Advanced Shell でサポートしない組み込みコマンドを実行しようとしました。または，現在のプラットフォームでは，JP1/Advanced Shell でサポートしないオプションを指定して組み込み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

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

Specified variable " **シェル変数名** " cannot be exported because the name is not in all capital letters on the current platform.［ 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

The required directory " **ディレクトリ名** " does not exist.

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

Background execution is not supported.［ 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

Subshell execution is not supported.［ 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。

KNAX6716-W

Command " **コマンド名** " was ignored because it was executed in editor. It returns " **戻り値** ".［ filename=" **ファイル名** " line= **行番号**］

エディタから実行したため，コマンド名に示すコマンドは無視されました。

コマンド名：実行しようとしたコマンド名。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。

戻り値：無視されたコマンドの戻り値

ファイル名：ジョブ定義スクリプトファイル名

行番号：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。

(O) コマンド名に示すコマンドを実行する場合，実行するプログラムのパスを指定しておきます。また，コマンド名を引数として渡すようにジョブ定義スクリプトを修正します。

KNAX6800-I

Path convert rule matched.［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

Path convert rule matched.［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。

シェル変数名：パス名を扱うシェル変数の名前

ファイル名：ジョブ定義スクリプトファイル名

行番号：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。

KNAX6803-I

Access path convert rule matched.［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 属性（Windows，Linux 限定）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 arg convert rule matched.［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 属性（Windows，Linux 限定）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

Access path in **機能名** convert rule matched. 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：command replace
- trap コマンドのアクションの場合：trap action

(S) 処理を継続します。

KNAX6806-I

Command arg in **機能名** convert rule matched. 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：command replace
- trap コマンドのアクションの場合：trap action

(S) 処理を継続します。

KNAX6810-E

Directory delimiter is invalid.

#-adsh_conf コマンドの 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 delimiter is invalid.

#-adsh_conf コマンドの 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

Invalid path is specified. path=" **パス名** "

#-adsh_conf コマンドの PATH_CONV パラメーターで指定したパスが不正です。

パス名：#-adsh_conf コマンドの 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

Line size limit is exceeded.［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

Invalid argument is specified. arg=" **引数** "

#-adsh_conf コマンドの COMMAND_CONV_ARG 属性（Windows，Linux 限定），または PATH_CONV_ACCESS 属性（Windows，Linux 限定）で指定した引数が不正です。

引数：#-adsh_conf コマンドの COMMAND_CONV_ARG 属性（Windows，Linux 限定），または PATH_CONV_ACCESS 属性（Windows，Linux 限定）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

Argument is not specified.

#-adsh_conf コマンドの COMMAND_CONV_ARG 属性（Windows，Linux 限定），または PATH_CONV_ACCESS 属性（Windows，Linux 限定）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。

KNAX6997-E

API error occurred. 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 JP1/Advanced Shell を呼び出している可能性があります。
この場合，ジョブオブジェクトに関する処理は親プロセスですでに実行されているため，このメッセージが出力されても (O) に示す対処は必要ありません。ただし，ジョブ定義スクリプトから外部の JP1/Advanced Shell を呼び出す場合は，#-adsh_script コマンドの利用を検討してください。

(S) 処理を終了します。ただし，ジョブオブジェクトの作成，またはジョブオブジェクトへのプロセスの関連づけでエラーが発生した場合は，処理を継続します。

(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ます。

KNAX6998-E

cannot allocate memory.［ 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

Internal error occurred. reason=" **原因** " detail=" **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

Input is too long.

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

Specified command " **コマンド名** " does not exist.

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

Too many arguments are specified.

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

Command " **コマンド名** " does not take any arguments.

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

Command " **コマンド名** " requires argument (**引数詳細**).

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。

KNAX7005-E

Command " **コマンド名** " requires arguments (**引数詳細**).

コマンド名で示すデバッガのコマンドは**引数詳細**で示す複数の引数を取ります。

(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

Restart the script from the beginning? (y or 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

Starting the script: **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

The script was not restarted.

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

Kill the script being debugged? (y or 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

The script is not running.

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

Command " **コマンド名** " was not executed.

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

The script is running. Exit the debugger? (y or 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

File " **ファイル名** " is not parsed.

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

Number of breakpoints and watchpoints exceeds the limit.

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

Breakpoint cannot be set at line " **行番号** ". Setting breakpoint at next available line.

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

Breakpoint cannot be set at line " **行番号** ". Next available line does not exist.

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

File " **ファイル名** " does not exist.

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

Breakpoint " **ブレークポイント番号** ": filename=" **ファイル名** " line= **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

Line " **行番号** " does not exist.

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

Specified function " **関数名** " is not defined.

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

Specified jobstep " **ジョブステップ名** " is not defined.

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

Specified variable " **変数名** " is invalid.

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

Watchpoint " **ウォッチポイント番号** ": variable " **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

Specified range " **番号範囲** " is invalid.

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

Delete all breakpoints and watchpoints? (y or 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

Breakpoints and watchpoints are not defined.

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

Breakpoint or watchpoint " **番号** " is not defined.

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

Breakpoints and watchpoints do not exist in the range " **番号範囲** ".

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

Specified argument " **引数** " is invalid.

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

Stop in the script " **ジョブ定義スクリプト名** ".

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

Stop in the external script " **ジョブ定義スクリプト名** ".

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

Continuing the script.

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

Command " **コマンド名** " cannot be executed in the outermost place.

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

Run till exit of current function.

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

Exit the current function? (y or 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

Sending signal " **シグナル名** " to the script.

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

Specified signal number " **シグナル番号** " does not exist.

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

Specified signal " **シグナル名** " does not exist.

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

Script stopped because signal " **シグナル名** "(Stop=Yes) received before.

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

Command " **コマンド名** " requires subcommand.

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

Specified " **コマンド名** " command " **サブコマンド名** " is invalid.

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

Specified variable " **変数名** " is not defined.

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

Breakpoints and watchpoints are not defined.

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

Working directory: **ディレクトリパス名**

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

Working directory cannot be changed. (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

Symbol "&" cannot be specified in arguments of command " **コマンド名** ".

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

Type of assignment value is invalid.

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

Usage: **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

Specified variable " **変数名** " is read-only.

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

Breakpoint has already been set at line " **行番号** ".

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

Value of variable " **変数名** " was changed to **数値** .

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

Value of variable " **変数名** " was changed to " **文字列** ".

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

Specified variable " **変数名** " has no value.

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

Command " **コマンド名** " request was received in running.

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

Cancel request from editor was received.

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

Jobsteps are not defined.

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

Functions are not defined.

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

Variables are not defined.

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

Commands are skipped until end of function.

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

Watchpoint has already been set for variable " **変数名** ".

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

Breakpoint cannot be set because of **エラー要因** .

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 action：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

Line cannot be listed because of **エラー要因** .

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 action：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。

KNAX7090-W

One or more error information exist.

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。

KNAX7099-E

Debugger ended abnormally.

デバッガ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ます。

KNAX7101-E

Failed to create debugger child process. (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) 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。

KNAX7102-I

The debugger child process was created. (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

The debugger child process ended. (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

Failed to open log file in debugger.

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) 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。

KNAX7105-E

Failed to move to debugger work directory. (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

Failed to move to script work directory. (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

Received signal in input debugger command.

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

Failed to input debugger command.

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

EOF was input with debugger command.

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

Failed to open DBG file " **ファイルパス** ". (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 file " **ファイルパス** " parsing start.

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 file " **ファイルパス** " format is invalid. DETAIL= **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

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 名**の処理に失敗しました。

(S) 処理を終了します。

(O) システム管理者に連絡します。エラー詳細の原因を取り除いて再実行します。

KNAX7114-E

Failed to wait the debugger child process. (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) 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。

KNAX7115-E

Failed to execute the exec command process. (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) 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。

KNAX7116-E

Disk free space is low.

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 file " **ファイルパス** " parsing ended normally.

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

Failed to create console.

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

Failed to duplicate file descriptor.

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

Parent process received SIGCHLD, but child process had no change.

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

Failed to create the exec command process. (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) 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。

KNAX7122-E

Failed to wait the exec command process. (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) 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。

KNAX7123-E

Failed to get value of variable " **変数名** ". detail= **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。

(S) 処理を終了します。

(O) 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。

KNAX7124-E

Failed to set value of variable " **変数名** ". detail= **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

Failed to get information of function " **関数名** ". detail= **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。

KNAX7600-E

JP1/AJS - View **から起動してください。**

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

JP1/AJS - View **から取得した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

JP1/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

JP1/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

JP1/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

JP1/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 JP1/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

入力情報の取り消し処理に失敗しました。(reason = **エラー詳細**)

入力した情報の取り消し処理が失敗しました。

(S) 処理を中断します。ジョブの定義は取り消されていません。

(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 関数の戻り値を調べ，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］

directory : Specify output directory

-f filename : Specify a config file

-e filename : Specify an environment file
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。

(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 CD-ROM からインストール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 CD-ROM からインストールディレクトリへマニュアルをコピーします。

KNAX7901-I

adshexec waits for all asynchronous processes at the end of the job.

adshexec コマンドは，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。このメッセージは次の条件を満たした場合に出力されます。

- 環境ファイルパラメーターに CHILDJOB_SHEBANG（Windows，Linux 限定）パラメーターを 1 つ以上指定したとき。

(S) 処理を続行します。

KNAX7999-I

JP1/Advanced Shell ended. 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。
次の両方の条件に合致する場合，**終了コード**には exec コマンドの実行直前のバッチジョブの終了コードが出力されます。

- 環境ファイルに CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）を 1 つも指定していない
- exec コマンドでバッチジョブの実行を終了する

(S) 処理を続行します。

11.4 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 固有の場合とに分けて説明します。

11.4.1 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 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 11-5 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）

エラー番号（errno）	エラーの詳細	ニモニック	原因	対策
2	No such file or directory	ENOENT	ファイルまたはディレクトリが見つかりません。	ファイルの存在を確認してください。
5	Input/Output error	EIO	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。
6	No such device or address	ENXIO	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。
7	Arg list too long	E2BIG	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 コマンド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。
11	Resource temporarily unavailable	EAGAIN	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。
12	Not enough space	ENOMEM	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。

エラー番号（errno）	エラーの詳細	ニモニック	原因	対策
13	Permission denied	EACCES	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。	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。
14	Bad address	EFAULT	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。
17	File exists	EEXIST	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。
22	Invalid argument	EINVAL	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。
23	Too many open files in system	ENFILE	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。
24	Too many open files	EMFILE	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。
27	File too large	EFBIG	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。
28	No space left on device	ENOSPC	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。

11.4.2 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 11-6 エラーの詳細に対する原因と対策（UNIX の場合）

エラー番号（errno）	エラーの詳細	ニモニック	原因	対策
2	No such file or directory	ENOENT	ファイルまたはディレクトリが見つかりません。	ファイルの存在を確認してください。
5	I/O error	EIO	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。

エラー番号 (errno)	エラーの詳細	ニモニック	原因	対策
6	No such device or address	ENXIO	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 UNIX のマニュアルを参照してください。
7	Arg list too long	E2BIG	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 コマンド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。
11	Resource temporarily unavailable	EAGAIN	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。
12	Not enough space	ENOMEM	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。
13	Permission denied	EACCES	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。
14	Bad address	EFAULT	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。
17	File exists	EEXIST	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。
22	Invalid argument	EINVAL	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。
23	File table overflow	ENFILE	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。
24	Too many open files	EMFILE	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。
27	File too large	EFBIG	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。

エラー番号 (errno)	エラーの詳細	ニモニック	原因	対策
28	No space left on device	ENOSPC	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。
86	File name too long	ENAMETOOLONG	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。

11.4.3 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 11-7 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	Over max line size	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	Can not get application data folder name	アプリケーションデータ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。
	Can not get common documents folder name	共通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	illegal file descriptor name	指定されたファイルディスクリプタが 1 桁の数字でありません。	ファイルディスクリプタの指定を見直します。
	bad file descriptor	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ディスクリプタを指定しました。	ファイルディスクリプタ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。
	fd not open for writing	書き込みでオープンされていないファイルディスクリプタへの書き込みを指定しました。	ファイルディスクリプタの指定を見直します。
	fd not open for reading	読み込みでオープンされていないファイルディスクリプタに読み込みを指定しました。	ファイルディスクリプタの指定を見直します。
	no coprocess	バックグラウンドプロセスへのファイルディスクリプタを指定しましたが，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	Is not a regular file	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。
KNAX6333-E	Is not a regular file	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。

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。

- 特殊組み込みコマンド
- 正規組み込みコマンド
- スクリプト拡張コマンド
- その他のコマンド

(1) 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
. コマンド	○	×
: 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	○	×

（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) 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	○	×
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。

(3) スクリプト拡張コマンド

表 A-3　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	○	×
#-adsh_script コマンド	○	×
#-adsh_spoolfile コマンド	○	×
#-adsh_step_start コマンド	○	○※1

項目	C0	C1
#-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，「3.8.4(3)(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，「3.8.4(3)(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-4　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-5　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	×	×
elif	×	○
elif の条件	○	×
else	×	○
fi	×	△
for	×	○
変数	×	×
in	×	×
ワードリスト	×	×
do	×	×
done	×	○
while	×	○
while の条件	○	×

項目	C0	C1
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-6　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-7　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-8　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　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別に説明します。

付録 B.1　3020-3-S35-20 での変更内容

JP1/Advanced Shell 09-51，および JP1/Advanced Shell - Developer 09-51 に対応したマニュアルの変更内容です。

- 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。
  . コマンド，：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 B.2　3020-3-S35-10 での変更内容

JP1/Advanced Shell 09-50-01，および JP1/Advanced Shell - Developer 09-50-01 に対応したマニュアルの変更内容です。

- 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 AP データ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戻り値の説明を変更した。
- 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。
  . コマンド，：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 C　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 C.1　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/AJS 関連

- JP1 Version 9 JP1/Automatic Job Management System 3 導入ガイド（3020-3-S02）
- JP1 Version 9 JP1/Automatic Job Management System 3 設計ガイド（システム構築編）（3020-3-S03）
- JP1 Version 9 JP1/Automatic Job Management System 3 設計ガイド（業務設計編）（3020-3-S04）
- JP1 Version 9 JP1/Automatic Job Management System 3 構築ガイド 1（3020-3-S05）
- JP1 Version 9 JP1/Automatic Job Management System 3 構築ガイド 2（3020-3-S06）
- JP1 Version 9 JP1/Automatic Job Management System 3 運用ガイド（3020-3-S07）
- JP1 Version 9 JP1/Automatic Job Management System 3 トラブルシューティング（3020-3-S08）
- JP1 Version 9 JP1/Automatic Job Management System 3 操作ガイド（3020-3-S09）
- JP1 Version 9 JP1/Automatic Job Management System 3 コマンドリファレンス 1（3020-3-S10）
- JP1 Version 9 JP1/Automatic Job Management System 3 コマンドリファレンス 2（3020-3-S11）
- JP1 Version 9 JP1/Automatic Job Management System 3 連携ガイド（3020-3-S12）
- JP1 Version 9 JP1/Automatic Job Management System 3 メッセージ 1（3020-3-S13）
- JP1 Version 9 JP1/Automatic Job Management System 3 メッセージ 2（3020-3-S14）
- JP1 Version 9 JP1/Automatic Job Management System 3 - Definition Assistant（3020-3-S17）
- JP1 Version 8 JP1/Automatic Job Management System 2 解説（3020-3-K21）
- JP1 Version 8 JP1/Automatic Job Management System 2 設計・運用ガイド（3020-3-K22）
- JP1 Version 8 JP1/Automatic Job Management System 2 セットアップガイド（3020-3-K23）
- JP1 Version 8 JP1/Automatic Job Management System 2 操作ガイド（3020-3-K24）
- JP1 Version 8 JP1/Automatic Job Management System 2 コマンドリファレンス（3020-3-K25）
- JP1 Version 8 JP1/Automatic Job Management System 2 連携ガイド（3020-3-K27）
- JP1 Version 8 JP1/Automatic Job Management System 2 メッセージ（3020-3-K28）
- JP1 Version 8 JP1/Automatic Job Management System 2 - Definition Assistant（3020-3-K37）

JP1/NETM/DM 関連

- JP1 Version 9 JP1/NETM/DM 導入・設計ガイド (Windows(R) 用)（3020-3-S79）
- JP1 Version 9 JP1/NETM/DM 運用ガイド 1(Windows(R) 用)（3020-3-S81）
- JP1 Version 6 JP1/NETM/DM Manager（3000-3-841）
- JP1 Version 8 JP1/NETM/DM SubManager(UNIX(R) 用)（3020-3-L42）

JP1/Base 関連

- JP1 Version 9 JP1/Base 運用ガイド（3020-3-R71）

付録 C.2　このマニュアルでの表記

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
UNIX	Linux	Red Hat Enterprise Linux 5（AMD/Intel 64）
		Red Hat Enterprise Linux 5（x86）
		Red Hat Enterprise Linux 5 Advanced Platform（AMD/Intel 64）
		Red Hat Enterprise Linux 5 Advanced Platform（x86）
		Red Hat Enterprise Linux Server 6（64-bit x86_64）
		Red Hat Enterprise Linux Server 6（32-bit x86）
	AIX	AIX 5L 5.3，AIX 6.1，または AIX 7.1
JP1/AJS	JP1/AJS2	JP1/Automatic Job Management System 2 - Agent
		JP1/Automatic Job Management System 2 - Manager
		JP1/Automatic Job Management System 2 - View
	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/AJS2 - Agent	JP1/Automatic Job Management System 2 - Agent
	JP1/AJS3 - Agent	JP1/Automatic Job Management System 3 - Agent
JP1/AJS - Manager	JP1/AJS2 - Manager	JP1/Automatic Job Management System 2 - Manager
	JP1/AJS3 - Manager	JP1/Automatic Job Management System 3 - Manager
JP1/AJS - View	JP1/AJS2 - View	JP1/Automatic Job Management System 2 - View
	JP1/AJS3 - View	JP1/Automatic Job Management System 3 - View

付録 C.3　ディレクトリの表記について

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。

付録 C.4　KB（キロバイト）などの単位表記について

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。

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。AIX では環境変数が設定されていても，.env ファイルを読み込みません。

（英字）

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/AJS

JP1/AJS2 と JP1/AJS3 の両方を示す場合に使用します。

JP1/AJS2

JP1/Automatic Job Management System 2 の略で，JP1 関連製品の 1 つです。JP1/Advanced Shell は，JP1/AJS2 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。

JP1/AJS3

JP1/Automatic Job Management System 3 の略で，JP1/AJS2 の後継製品です。

JP1/Script

ジョブを制御する JP1 スクリプトを作成して実行するためのプログラムです。Windows または UNIX のどちらかでジョブを制御できます。

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）

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。

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。コマンドにもオプションがあります。JP1/Advanced Shell では，ハイフン（-）で始まるコマンドの引数を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。

（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，環境変数や環境設定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
JP1/Advanced Shell が環境ファイルに設定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。ジョブ定義スクリプトで使用でき，シェルまたはコマンドプロンプトで実行できるコマンドです。JP1/Advanced Shell では正規組み込みコマンドと特殊組み込みコマンドがあります。

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）

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
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。

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，CHILDJOB_SHEBANG パラメーター（Windows，Linux 限定）によって実行されたジョブのことです。

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。

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。

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。

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。

（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。

トレース情報
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
JP1/Advanced Shell 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。JP1/Advanced Shell 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。

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やレジスタの値を確認できます。

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システム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。

ルートジョブ

Windows または Linux で，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。

（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

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

に

は

ひ

ふ

へ

ほ

ま

め

も

よ

り

る

れ

ろ

わ