目 次

第1章　BASH-4.2

1.1　名前

bash - GNU Bourne-Again SHell

1.2　書式

bash [options] [file]

1.3　著作権

1.4　説明

bash は、標準入力やファイルから読み込んだコマンドを実行する、 sh 互換のコマンド言語インタプリタです。 bash には、Korn シェルや C シェル (ksh や csh) の便利な機能も採り入れられています。

bash は IEEE POSIX specification (IEEE Standard 1003.1) の Shell and Utilities に準拠する実装を目指しています。 bash はデフォルトで POSIX 準拠に設定することもできます。

1.5　オプション

組み込みコマンド set の説明で述べられている 1 文字のシェルオプションを、 起動時に指定できます。それに加えて、bash は以下のオプションを起動時に解釈します。

1.5.1　-c string

-c オプションが指定されると、コマンドが string から読み込まれます。 string の後に引き数があれば、これらは 位置パラメータ (positional parameter: $0 から始まるパラメータ) に代入されます。

1.5.2　-i

オプションが指定されると、 bash は 対話的 (interactive) に動作します。

1.5.3　-l

ログインシェル (後述の 起動 セクションを参照) として起動されたかのように bash を動作させます。

1.5.4　-r

-r オプションが指定されると、 bash は 制限された状態 (restricted) となります (後述の 制限付きのシェル を参照)。

1.5.5　-s

-s オプションが指定された場合と、 オプションを全て処理した後に引き数が残っていなかった場合には、コマンドは標準入力から読み込まれます。 このオプションを使うと、 対話的シェルを起動するときに 位置パラメータを設定できます。

1.5.6　-D

二重引用符によるクォート文字列 (double-quoted strings) に $ が前置されたものを、全てリストして標準出力に出力します。 これらは、カレントロケールが C または POSIX 以外のときに、 翻訳の対象となるべき文字列です。 このオプションを指定すると、 自動的に -n オプションも指定されたことになります。つまりコマンドは全く実行されません。

1.5.7　[-+]O [shopt_option]

shopt_option には、組み込みコマンド shopt (後述の シェルの組み込みコマンド を参照) に与えるのと同じシェルのオプションを指定します。 shopt_option が有効なオプションであれば、 -O でオプションが設定されます。 +O で設定解除になります。shopt_option を指定しない場合、 shopt で指定できるオプションの名前と値が標準出力に表示されます。 このとき、+O では、入力として再利用できる形で出力されます。

1.5.8　--

-- はオプションの終わりを示し、それ以降のオプション処理を行いません。 -- 以降の引き数は全て、ファイル名や引き数として扱われます。 引き数 - は -- と同じです。

bash が解釈するオプションには複数の文字からなるものもたくさんあります。このようなオプションを認識させるためには、 コマンドライン中で 1 文字のオプションよりも前に置かなければなりません。

1.5.9　--debugger

シェルの実行を開始する前に、デバッグモードを準備します。 拡張デバッグモードを有効にします (後述の組み込みコマンド shopt の extdebug オプションを参照)。

1.5.10　--dump-po-strings

-D と同じですが、 出力は GNU gettext の po (ポータブルオブジェクト) ファイル形式で行われます。

1.5.11　--dump-strings

-D と同じです。

1.5.12　--help

使用方法についてのメッセージを標準出力に表示し、正常終了します。

1.5.13　--init-file file

1.5.14　--rcfile file

対話的シェルとして起動された場合、 個人用の標準の初期化ファイル~/.bashrc の代わりに file からコマンドを実行します (後述の 起動セクションを参照)。

1.5.15　--login

-l と同じです。

1.5.16　--noediting

シェルが対話的動作の場合、コマンドラインを読み込むときに GNU readline ライブラリを使用しません。

1.5.17　--noprofile

システム全体用の起動ファイル /etc/profile および個人用の初期化ファイル ~/.bash_profile, ~/.bash_login, ~/.profile のいずれも読み込みません。デフォルトでは、 bash はログインシェルとして起動されたときにこれらのファイルを読み込みます (後述の 起動 セクションを参照)。

1.5.18　--norc

シェルが対話的動作を行う場合に、個人用初期化ファイル ~/.bashrc の読み込み・実行を行いません。シェルが sh として起動された場合には、このオプションはデフォルトで有効になります。

1.5.19　--posix

bash の動作のうち、デフォルトの振舞いが POSIX 標準と異なる部分を、POSIX 標準に準拠するように変更します (posix モード)。

1.5.20　--restricted

シェルを制限された状態にします (後述の 制限付きのシェル セクションを参照)。

1.5.21　--verbose

-v と同じです。

1.5.22　--version

実行された bash のバージョン情報を標準出力に表示し、正常終了します。

1.6　引き数

オプション処理の後に引き数が残っており、かつ -c オプションと -s オプションのいずれも指定されていない場合、 最初の引き数はファイル名とみなされ、 そのファイルにシェルコマンドが記述されているとみなされます。 このような形で bash が起動された場合、 $0 にそのファイルの名前が設定されます (残りの引き数は位置パラメータに設定されます)。 bash はこのファイルからコマンドの読み込みと実行を行い、そして終了します。 bash の終了ステータスは、 このスクリプト中で実行された最後のコマンドの終了ステータスになります。 コマンドが全く実行されなければ、終了ステータスは 0 です。 ファイルは最初にカレントディレクトリから探し、見つけられなかった場合には、PATH の中のディレクトリからスクリプトを探します。

1.7　起動

ログインシェル (login shell) とは、0 番目の引き数の最初の文字が - であるシェル、または --login オプション付きで起動されたシェルのことです。

対話的なシェルとは、 オプションでない引き数がなく、 標準入力と標準エラー出力がいずれも端末に接続されていて (これは isatty(3) で調べられます)、 -c オプションが指定されていない状態で起動されたシェル、または -i オプション付きで起動されたシェルのことです。 bash が対話的に動作している場合には、 PS1 が設定され、 $- に i が含まれます。 これを利用すると、対話的動作の状態であるかどうかを、シェルスクリプトや起動ファイルの内部で調べられます。

以下の段落では、 bash がどのように起動ファイルを実行するかを説明します。 以下のファイルのいずれかが、 「存在しているが読み込みできない」場合は、 bash はエラーを報告します。ファイル名に含まれるチルダは、 後述の展開 セクションにおける チルダ展開 の項目で述べるように展開されます。

bash が対話的なログインシェルとして起動されるか、 --login オプション付きの非対話的シェルとして起動されると、 /etc/profile ファイルが存在すれば、 bash はまずここからコマンドを読み込んで実行します。 このファイルを読んだ後、 bash は ~/.bash_profile, ~/.bash_login, ~/.profile をこの 順番で探します。 bash は、この中で最初に見つかり、かつ読み込みが可能であるファイルから コマンドを読み込んで実行し

ます。 --noprofile オプションを使ってシェルを起動すれば、 この動作を行わないようにできます。

ログインシェルが終了するときには、 ~/.bash_logout ファイルがあれば、bash はこれを読み込んで実行します。

ログインシェルでない対話的シェルとして起動されると、 ~/.bashrc ファイルがあれば、 bash はここからコマンドを読み込み、実行します。 この動作は--norc オプションで行わないようにできます。 --rcfile file オプションを使うと、 コマンドの読み込みと実行を ~/.bashrc からでなく file から行わせることができます。

(例えばシェルスクリプトを実行するために) 非対話的に起動されると、 bash は環境変数 BASH_ENV を調べ、この変数が定義されていればその値を展開し、得られた値をファイル名とみなして、 そこからコマンドの読み込みと実行を行います。 つまり bash は以下のコマンドが実行されたのと同じように動作します:

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

ただし、ファイル名を探すために PATH 環境変数の値が使われることはありません。

sh という名前で bash を起動すると、 bash は古くからある sh の起動動作をできるだけ真似しようとします。 また POSIX 標準にもできるだけ従おうとします。 対話的なログインシェルとして起動されると、あるいは --login オプション付きの非対話的シェルとして起動されると、 このシェルはまず/etc/profile と ~/.profile の順でコマンドの読み込みと実行をしようとします。 --noprofile オプションを使うと、この動作を行わないようにできます。sh という名前の対話的シェルとして起動されると、 bash は環境変数 ENV を調べ、この変数が定義されていればその値を展開し、 展開で得た値をコマンドの読み込みと実行を行うためのファイル名として使います。 sh という名前で起動されたシェルは、 ほかの起動ファイルからコマンドの読み込みと起動を行うことはないので、 --rcfile オプションは全く効果を持ちません。 sh という名前の非対話的シェルとして起動されると、 このシェルはほかの起動ファイルを何も読み込みません。 sh として起動された場合、 bash は起動ファイルの読み込みを行った後に POSIX モードに入ります。

--posix コマンドラインオプション等により bash が POSIX モードで起動されると、 bash は起動ファイルに関して POSIX 標準に従います。 このモードでは、対話的シェルは ENV 環境変数を展開し、 展開して得られた名前のファイルからコマンドの読み込みと実行を行います。 ほかの起動ファイルは全く読み込みません。

bash は、リモートシェルデーモン rshd やセキュアシェルデーモン sshd によって実行された場合など、標準入力がネットワーク接続に接続された 状態で実行されたかどうかを調べます。 この方法によって実行されていると bash が判断した場合、 ~/.bashrc が存在し、かつ読み込み可能であれば、 bash はコマンドをこのファイルから読み込んで実行します。 sh として呼び出された場合には、この動作は行いません。 --norc オプションを使えばこの動作を禁止できますし、 --rcfile オプションを使えばほかのファイルを読ませるようにもできます。 しかし一般的には rshd は これらのオプションを付けてシェルを起動しませんし、指定もできないようになっています。

シェルが実ユーザ (グループ) ID と異なる実効ユーザ (グループ) ID で起動され、かつ -p オプションが与えられていない場合は、起動ファイルは全く読み込まれず、シェル関数は環境から継承されず、SHELLOPTS, BASHOPTS, CDPATH, GLOBIGNORE が環境変数に含まれていても無視され、 実効ユーザ ID には実

ユーザ ID が設定されます。 -p オプションが起動時に与えられた場合、 起動時の動作は同じですが、 実効ユーザ ID は再設定されません。

1.8　定義

このドキュメントの残りの部分では、 以下の定義を使用します。

1.8.1　ブランク (blank)

空白文字またはタブ文字

1.8.2　単語 (word)

シェルが 1 単位とみなす文字の並び。 トークン (token) とも言われます。

1.8.3　名前 (name)

英数字とアンダースコア文字だけから構成され、 かつ最初の文字が英字かアンダースコア文字である 単語。 識別子 (identifier) とも言われます。

1.8.4　メタ文字 (metacharacter)

クォートされていない場合に、単語区切りとなる文字。 以下の文字のうちのいずれかです: | & ; ( ) < > space tab

1.8.5　制御演算子 (control operator)

制御機能を持つ トークン。 以下のシンボルのうちのいずれかです: || & && ; ;; ( ) | |& <newline>

1.9　予約語

予約語 (reserved word) とはシェルにとって特別な意味を持つ単語です。 以下の単語がクォートされておらず、 かつ単純なコマンド (simple command) の先頭の単語 (後述の シェルの文法 を参照) であるか、case コマンドや for コマンドの 3 番目の単語である場合には、予約語として認識されます:

```
! case  do done elif else esac fi for function if in select then  until
while { } time [[ ]]
```

1.10　シェルの文法

1.10.1　単純なコマンド (Simple Commands)

単純なコマンド (simple command) とは、 変数の代入を並べたもの (これは省略可能です) の後に、 ブランク区切りの単語とリダイレクションを記述し、最後に制御演算子を置いたものです。 最初の単語は実行するコマンドを指定します。 これは 0 番目の引き数となります。 残りの単語は起動されるコマンドに引き数として渡されます。

単純なコマンドの返り値はコマンドの終了コードですが、シグナル n を受けてコマンドが終了した場合には 128+n となります。

1.10.2　パイプライン (Pipeline)

パイプライン (pipeline) は、制御演算子 | または |& で区切った 1 つ以上のコマンドの並びです。 パイプラインのフォーマットを以下に示します:

```
[time [-p]] [ ! ] command [ [ | または |& ] command2 ... ]
```

command の標準出力は command2 の標準入力にパイプで接続されます。 この接続は、 コマンドで指定したどのリダイレクションよりも先に実行されます (後述の リダイレクト を参照)。 |& を使うと、command の標準エラー出力もパイプを通して command2 の標準入力に接続されます。 これは 2>&1 | の短縮形です。 この標準エラー出力の暗黙のリダイレクションは、 コマンドに指定された全てのリダイレクションの後に実行されます。

pipefail オプションが有効になっている場合を除き、 パイプラインの返却ステータスは最後のコマンドの終了ステータスになります。 pipefail が有効になっている場合には、 0 以外のステータスを返した最後の (一番右の) コマンドの値が パイプラインの返却ステータスになり、 全てのコマンドが正常終了した場合にのみ 0 になります。 パイプラインの前に、予約語である！がある場合、 そのパイプラインの終了ステータスは上記の終了ステータスを 論理否定したものになります。 値を返す前に、シェルはパイプライン中の全てのコマンドが終了するのを待ちます。

パイプラインの前に予約語 time がある場合、コマンドの実行にかかった 経過時間・ユーザ時間・システム時間がパイプラインの終了時に報告されます。 -p オプションを指定すると、 出力フォーマットが POSIX 仕様に変わります。シェルが posix モード のときには、 後に続くのが '-' で始まるトークンであれば time を予約語と認識しません。 変数 TIMEFORMAT には、経過時間情報の表示の仕方を指定するフォーマット文字列を 設定できます (後述の シェル変数 の項の TIMEFORMAT に関する説明を参照)。

シェルが posix モード のときには、time の直後が 改行でもかまいません。この場合、シェルと子プロセスがそれまでに消費した ユーザ時間とシステム時間を出力します。 このときにも、経過時間情報のフォーマットを変数 TIMEFORMAT で指定できます。

パイプライン中の各コマンドは、それぞれ別のプロセスとして (つまりサブシェル内で) 実行されます。

1.10.3　リスト

リスト (list) とは、1 つ以上のパイプラインを演算子 ;, &, &&, || のいずれかで区切って並べたものです。パイプラインの最後に ;, &, <newline> のいずれかを置くこともできます。

リスト演算子のうち、 && と || の優先順位は同じです。これらの次に、 ; と& が同じ優先順位で続きます。

リスト 中では、コマンドの区切りとして、 セミコロンの代わりに一つ以上の改行が使われることもあります。

コマンドが制御演算子 & で終わっている場合、シェルはコマンドをサブシェル内で バックグラウンド (background) で実行します。 シェルはコマンドが終了するのを待たずに、返却ステータス 0 を返します。コマンドを ; で区切った場合には、これらは順番に実行されます。 シェルはそれぞれのコマンドが終了するのを順番に待ちます。 返却ステータスは、最後に実行したコマンドの終了ステータスになります。

AND リストと OR リストは、それぞれ制御演算子 && と || で区切られたパイプラインの並びです。AND リストと OR リストは左結合で実行されます。 AND リストは

```
command1 && command2
```

という形式であり、 command1 が終了ステータス 0 を返した場合に限り command2 が実行されます。
OR リストは

```
command1 || command2
```

という形式であり、 command1 が 0 以外の終了ステータスを返した場合に限り command2 が実行されます。 AND リストと OR リストの返却ステータスは、 リスト中で最後に実行されたコマンドの終了ステータスです。

1.10.4　複合コマンド (Compound Commands)

複合コマンド (compound command) を以下に示します:

(list)

list はサブシェル内で実行されます (後述の コマンド実行環境 の項を参照)。 シェルの環境に影響を与えるような変数の代入や組み込みコマンドは、 コマンドの終了後に影響を残しません。 返却ステータスは list の終了ステータスです。

{ list; }

list が単に現在のシェル環境で実行されます。 list の最後は改行文字かセミコロンでなければなりません。これは グループコマンド (group command) と呼ばれます。返却ステータスは list の終了ステータスです。 メタ文字である (や) と違い、 { と } は 予約語であり、予約語として認識される場所に現われる必要があることに注意してください。 これらは単語分割の対象とならないため、 リスト との間が空白またはシェルのメタ文字で分かれている必要があります。

((expression))

expression が後述の 算術式評価 で説明される規則に従って評価されます。 式の値が 0 でない場合、返却ステータスは 0 になります。そうでない場合、 返されるステータスは 1 になります。これは let "expression" と全く同じものです。

[[expression]]

条件式 expression の評価値に従って 0 または 1 を返します。 式は後述の 条件式 で説明する、プライマリによって構成されます。 単語分割とパス名展開は [[と]] の間の単語に対しては行われません。チルダ展開、パラメータと変数の展開、算術式展開、コマンド置換、プロセス置換、クォート除去は実行されます。-f などの条件演算子がプライマリとして認識されるためには、 クォートされていてはいけません。

[[においては、< 演算子と > 演算子は、 現在のロケールにおける辞書順で比較します。

== 演算子と != 演算子が使われたとき、 演算子の右の文字列はパターンと解釈され、 後述のパターンマッチングで説明する規則に従ってマッチングが行われます。 シェルオプション nocasematch が有効であれば、 アルファベットの大文字と小文字を考慮せずにマッチングが行われます。 返り値は、== 演算子では文字列がマッチしたときに、 != 演算子では文字列がマッチしなかったときに 0 となり、 そうでない場合に 1 となります。 パターン中のどの部分でも、クォートすることで、 ただの文字列としてマッチングさせることができます。

そのほか、二項演算子 =~ もあります。 優先順位は == や != と同じです。 これを使うと、右辺の文字列は拡張正規表現とみなされ、 それに従って (regex(3) にあるように) マッチングが行われます。 文字列がパターンにマッチすれば返り値は 0 であり、 マッチしなければ返り値は 1 になります。 正規表現が文法的に誤っていれば、条件式の返り値は 2 になります。 シェルオプション nocasematch が有効であれば、 アルファベットの大文字と小文字を考慮せずにマッチングが行われます。 パターン中のどの部分でも、クォートすることで、 ただの文字列としてマッチングさせることができます。 正規表現中の括弧による部分式にマッチした部分文字列は、配列変数 BASH_REMATCH に保存されます。 BASH_REMATCH のインデックス 0 の要素は、文字列のうち正規表現全体にマッチした部分になります。 BASH_REMATCH のインデックス n の要素は、文字列のうち、 正規表現中の n 番目の括弧による部分式にマッチした部分になります。

式は以下の演算子を使って繋げられます。 以下に演算子を優先度の順に示します:

`( expression )`
: `expression` の値を返します。これを用いて、 演算子の通常の優先度を変更できます。

`! expression`
: `expression` が偽ならば真になります。

`expression1 && expression2`
: `expression1` と `expression2` が両方とも真であれば真になります。

`expression1 || expression2`
: `expression1` と `expression2` のどちらかが真であれば真となり

ます。

`expression1` の値だけで条件式全体の返り値が決定できれば、 `&&` 演算子と `||` 演算子は `expression2` を実行しません。

for name [[in [word ...]] ;] do list ; done

in に続く単語のリストが展開され、要素のリストが生成されます。 変数 name には、このリストの各要素が順番にセットされ、そのたびに list が実行されます。「in word」が省略された場合、 for コマンドは、設定されている位置パラメータそれぞれに対して list を一度ずつ実行します (後述の パラメータ を参照)。返却ステータスは実行された最後のコマンドの終了ステータスになります。 in に続く要素を展開した結果が空となった場合、 コマンドは全く実行されず、返却ステータス 0 が返されます。

for ((expr1 ; expr2 ; expr3)) ; do list ; done

最初に、算術式 expr1 が、後述の 算術式評価 で説明される規則に従って評価されます。 次に、算術式 expr2 がゼロになるまで繰り返し評価されます。 算術式 expr2 の評価結果がゼロでなければ、そのたびごとに list が実行され、 算術式 expr3 が評価されます。 どの算術式も、省略された場合は、 評価結果が 1 であったものとして振舞います。 返却ステータスは list 中で実行された 最後のコマンドの終了ステータスとなりますが、 算術式のいずれかが不正である場合には偽となります。

select name [in word] ; do list ; done

in に続く単語のリストが展開され、要素のリストが生成されます。 展開された単語の集合が番号付きで標準エラー出力に出力されます。「in word」が省略された場合、 位置パラメータが出力されます (後述のパラメータ を参照)。続いて PS3 プロンプトが表示され、標準入力から 1 行の読み込みが行われます。 表示された単語のいずれかに対応する数字がこの行に含まれていれば、 name の値としてその単語が設定されます。 行が空であれば、単語とプロンプトが再び表示されます。EOF を読み込むとコマンドが終了します。 これ以外の値の場合には、name には空文字列が設定されます。読み込んだ行は変数 REPLY に格納されます。 break コマンドが実行されるまで、選択を行うたびに list が実行されます。 select の終了ステータスは、 list 中で最後に実行したコマンドの終了ステータスですが、 コマンドが全く実行されなかった場合には 0 となります。

case word in [[(] pattern [| pattern] ...) list ;;] ... esac

case コマンドは最初に word を展開し、それぞれの pattern に対して順にマッチングを試みます。 マッチングの際にはパス名展開 (後述のパス名展開 を参照) と同じ規則が用いられます。 word には、チルダ展開、パラメータと変数の展開、算術式展開、 コマンド置換、プロセス置換、クォート除去が実行されます。それぞれの pattern は、チルダ展開、パラメータと変数の展開、 算術式展開、コマンド置換、プロセス置換、クォート除去のうえで比較されます。 シェルオプション nocasematch が有効であれば、 アルファベットの大文字と小文字を考慮せずにマッチングが行われます。 マッチするものが見つかると、これに対応する list が実行されます。 ;; 演算子を使うと、 最初にマッチしたパターン以降のマッチングは試みられません。 ;; の代わりに ;& を使うと、 その次のパターンに対応する list の実行に続きます。 ;; の代わりに ;;& を使うと、 次のパターンがあればそのマッチングを試み、 マッチすれば対応する list を実行します。 マッ

チするパターンが無ければ、終了ステータスは 0 になります。 マッチするものがあったら、終了ステータスは list 中で最後に実行されたコマンドの終了ステータスになります。

if list; then list; [elif list; then list;] ... [else list;] fi

最初に if list が実行されます。list の終了ステータスが 0 ならば、then list が実行されます。 そうでなければ elif list がそれぞれ順番に実行され、 list の終了ステータスが 0 ならば、対応する then list が実行され、コマンドが終了します。そうでなければ、else list が (もし存在すれば) 実行されます。 終了ステータスは最後に実行されたコマンドの終了ステータスですが、 真と評価された条件が全くない場合には 0 となります。

while list-1; do list-2; done

until list-1; do list-2; done

while コマンドは、list-1 中の最後のコマンドが終了ステータス 0 を返す間、繰り返して list-2 を実行します。until コマンドは while コマンドとほぼ同じですが、 評価の条件が逆となる点が異なります。list-2 は list-1 中の最後のコマンドが 0 以外の終了ステータスを返す限りずっと実行されます。 while コマンドと until コマンドの終了ステータスは、 do list-2 で実行された最後のコマンドの終了ステータスになりますが、 コマンドが全く実行されなかった場合には 0 になります。

1.10.5　コプロセス (Coprocesses)

コプロセス (coprocess) とは、予約語 coproc で始まるシェルのコマンドのことです。 コプロセスは、コマンドが制御演算子 & で終わっているときのように、 サブシェルで非同期に実行されます。 実行したシェルとコプロセスの間には、双方向のパイプが設けられます

コプロセスのフォーマットを以下に示します:

```
coproc [NAME] command [redirections]
```

これにより NAME という名前のコプロセスが作られます。 NAME が指定されない場合、デフォルトの名前は COPROC となります。 command が 単純なコマンド (前述) のときには、 NAME は指定できません。 指定すると、単純なコマンドの最初の単語として扱われます。 コプロセスが実行されると、実行したシェルのコンテキストに NAME という名前の配列変数 (後述の 配列 を参照) が作られます。 command の標準出力は、実行しているシェルのファイル・ディスクリプターの一つと パイプによって接続されます。このファイル・ディスクリプターは NAME[0] に代入されます。 command の標準入力は、実行しているシェルのファイル・ディスクリプターの一つと パイプによって接続されます。このファイル・ディスクリプターは NAME[1] に代入されます。 このパイプは、コマンドで指定されたほかのリダイレクトより先に設けられます (後述のリダイレクト を参照)。 ファイル・ディスクリプターは、標準的な単語展開により、シェルのコマンドの引き数やリダイレクションに指定できます。 コプロセスの実行で生成されたシェルのプロセス ID は、変数 NAME_PID の値になります。 コプロセスの終了を待つには、組み込みコマンド wait を使います。

コプロセスの返却ステータスは、command の終了ステータスです。

1.10.6　関数定義

シェル関数とは、単純なコマンドとして呼び出されて、 新しい位置パラメーターの一式を持つ複合コマンドを実行するもののことです。 シェル関数は以下の形式で宣言します:

```
name () compound-command [redirection]
function name [()] compound-command [redirection]
```

これによって name という名前の関数が定義されます。 予約語 function は省略可能です。 予約語 function が与えられた場合、括弧は省略可能です。 関数の 実体 は複合コマンド compound-command です (前述の 複合コマンド を参照)。 多くの場合、 { と } に挟まれたコマンドの リスト が用いられますが、 前述の 複合コマンド で挙げたどのコマンドを用いてもかまいません。 単純なコマンドの名前として name を指定するといつでも compound-command が実行されます。関数の定義で指定されたリダイレクション (後述の リダイレクト を参照) は、関数が実行されるときに処理されます。 関数定義の終了ステータスは、文法エラーが起きた場合や、 読み込み専用の関数が同じ名前で定義されていた場合を除き、0 です。 関数を実行したときの終了ステータスは、実体の中で最後に実行されたコマンドの終了ステータスになります (後述の 関数 を参照)。

1.11　コメント (COMMENTS)

シェルが対話的でない場合、または対話的なシェルにおいて 組み込みコマンドの shopt に対する interactive_comments オプションが有効となっている (後述の シェルの組み込みコマンド を参照すること) 場合には、 # で始まる単語があると、その単語とその行の残りの文字が全て無視されます。 対話シェルでは、 interactive_comments オプションが有効でなければコメントは使えません。対話シェルでは、 interactive_comments オプションはデフォルトで有効になっています。

1.12　クォート

クォート (quoting) を使うと、 特定の文字や単語が持つシェルに対する特別な意味をなくせます。 クォートを用いると、特殊文字の特殊な扱いを無効にしたり、 予約語が予約語として識別されることを防いだり、パラメータの展開を防いだりできます。

前述の 定義 で挙げたメタ文字 (metacharacters) にはそれぞれ特殊な意味があるので、 その文字自身を表すためにはクォートしなければなりません。

コマンドの履歴展開の機能 (後述の 履歴の展開 を参照) が使われているときには、履歴展開 文字 (通常は !) の 履歴展開を防ぐためにはクォートしなければなりません。

クォートの方法には、 エスケープ文字 (escape character), シングルクォート、ダブルクォートの 3 種類があります。

クォートされていないバックスラッシュ (\) は エスケープ文字 です。エスケープ文字は <newline> という例外を除き、 後に続く文字 1 つの文字としての値を保持させます。 \<newline> という組み合わせが現われ、 かつバックスラッシュ自身がクォートされていない場合には、 \<newline> は行を継続すること

を表します (つまり、入力ストリームから改行文字が削除され、実質的に無視されます)。

シングルクォートで文字を囲むと、 クォート内部のそれぞれの文字は文字としての値を保持します。 シングルクォートの間にシングルクォートを置くことはできません。 これはバックスラッシュを前に付けても同じです。

ダブルクォートで文字を囲むとクォート内部の全ての文字は文字としての値を保持しますが、 $, ‘, \ は例外となります。 履歴展開が有効なときには、 ! がこれに加わります。 $ と ‘ はダブルクォートの内部でも特殊な意味を失いません。 バックスラッシュの場合は、次の文字が $, ‘, ”, \, <newline> のいずれかである場合に限り特殊な意味を失いません。 前にバックスラッシュを付ければ、 ダブルクォート文字をダブルクォートによるクォートの内部でクォートできます。 履歴展開は有効であれば実行されますが、ダブルクォート中の ! がバックスラッシュでエスケープされている場合を除きます。 ! の前のバックスラッシュは削除されません。

特殊なパラメータである * と @ は、ダブルクォート内部でも特殊な意味を失いません (後述の パラメータ を参照)。

$’string’ の形式を持つ単語は特殊な扱いを受けます。 この単語は string に展開され、 それから ANSI C 標準で仕様が決められている、 バックスラッシュでエスケープされている文字に置き換えられます。 バックスラッシュエスケープシーケンスは、 (もし存在すれば) 以下のようにデコードされます:

\a　警告 (ベル)
\b　バックスペース
\e
\E　エスケープ文字
\f　フォームフィード
\n　改行
\r　復帰 (carriage return)
\t　水平タブ
\v　垂直タブ
\\　バックスラッシュ
\’　シングルクォート
\"　ダブルクォート
\nnn　8 進値が nnn である 8 ビット文字 (1 文字につき数字 1〜3 桁)
\xHH　16 進値が HH である 8 ビット文字 (16 進で 1〜2 桁)。
\uHHHH　16 進値が HHHH であるユニコード (ISO/IEC 10646) 文字 (16 進 1〜4 桁)。
\UHHHHHHHH　16 進値が HHHHHHHH であるユニコード (ISO/IEC 10646) 文字 (16 進 1〜8 桁)。
\cx　control-x の文字

展開された結果はシングルクォートされているのと同じ状態で、 ドル記号は存在しなかったかのように扱われます。

ダブルクォートされた文字列の前にドル記号があると ($"string")、 文字列は現在のロケールに従って変換されます。 現在のロケールが C または POSIX ならば、 ドル記号は無視されます。文字列が変換されたり置換されたりした場合には、 その結果はダブルクォートされているのと同じ状態になります。

1.13　パラメータ

パラメータ (parameter) は値を保持するためのものです。パラメータは 名前、 数字、後述の 特殊なパラメータ で挙げる特殊文字のいずれかで表現されます。シェルでの用法においては、 変数 (variable) とは名前 で表現されたパラメータです。 変数は値と 0 個以上の属性を持ちます。 属性は declare 組み込みコマンド (後述の シェルの組み込みコマンド の declare の項を参照) で設定されます。

パラメータに値が代入されていれば、 そのパラメータは設定 (set) されていると言われます。 空文字列も有効な値です。一度値を設定すると、組み込みコマンドの unset を使わなければ削除 (unset) できません (後述の シェルの組み込みコマンド を参照)。

変数 には、以下の構文で代入できます:

name=[value]

value が与えられなかった場合、変数には空文字列が代入されます。全ての value に対して、チルダ展開、パラメータと変数の展開、コマンド置換、 算術式展開、クォート除去が行われます (後述の 展開 を参照)。変数の 整数 属性が設定されている場合 (後述の シェルの組み込みコマンド を参照)、$((...)) の展開を使っていなくても value に対しての算術展開が行われます (後述の算術式展開 を参照)。 特殊パラメータ で後述する "$@" という例外を除いて、単語の分割は行われません。 パス名展開も実行されません。 代入文は組み込みコマンドの alias, declare, typeset, export, readonly, local の引き数でも使われます。

代入文でシェル変数や配列のインデックスに値を代入する場面では、 += 演算子を使って変数の直前の値に追加したり加算したりできます。 += を整数属性が設定された変数に対して使うと、 値は算術式展開として評価され、 変数の現在の値に加算されます。 += を配列変数への複合代入 (後述の 配列 を参照) で使うと、 変数の現在の値は (= を使ったときとは違って) 削除されません。インデックスによる配列の場合は、新しい値が最大のインデックス より一つ大きいインデックスから配列に追加されます。 連想配列の場合は、新しいキーと値の組が追加されます。 文字列の値の変数に対して使うと、値が展開されて、変数の元の値の後に追加されます。

1.13.1　位置パラメータ (Positional Parameters)

位置パラメータ (positional parameter) は、1 桁以上の数値で表されるパラメータです。ただし 0 は含みません。 位置パラメータは、シェルが起動されたときにシェルの引き数が代入されますが、 組み込みコマンドの set を使って代入し直すこともできます。 代入文を使って位置パラメータへの代入を行うことはできません。 シェル関数が実行されると、 位置パラメータは一時的に置き換えられます (後述の 関数 を参照)。

2 桁以上の数値を含む位置パラメータを展開するときには、ブレース (**{}**) で囲まなければなりません (後述の 展開 を参照)。

1.13.2　特殊パラメータ

シェルはいくつかのパラメータを特別扱いします。 このようなパラメータは参照されるだけであり、 値を代入することは許されません。

*****

(1 から始まる) 全ての位置パラメータに展開されます。 ダブルクォートの内部で展開が行われたときは、それぞれのパラメータを特別な変数である IFS の最初の文字で区切って並べた 1 つの単語に展開されます。つまり、 "$*" は "$1c$2c..." と同じです。 ここで c は変数 IFS の値の最初の文字です。 IFS が設定されていなければ、パラメータは空白で区切られます。 IFS が空文字列の場合、パラメータの間には区切り文字は入らず、 全てのパラメータは繋げられます。

@

(1 から始まる) 全ての位置パラメータに展開されます。 ダブルクォートの内部で展開が行われたときは、それぞれのパラメータは別々の単語に展開されます。 つまり "$@" は "$1" "$2" ... と同じです。 単語の中でダブルクォートの展開が行われるときには、 最初のパラメータの展開結果に元の単語のダブルクォートより前の部分が 結び付き、最後のパラメータの展開結果に元の単語のダブルクォートより 後の部分が結び付きます。 位置パラメータがない場合には、"$@" や $@ を展開しても無かったことになります (つまり取り除かれます)。

#

位置パラメータの個数を示す 10 進値に展開されます。

?

最後に実行されたフォアグラウンドのパイプラインの 終了ステータスに展開されます。

-

現在のオプションフラグに展開されます。 これは起動のときに指定したり、組み込みコマンド set で設定したり、(-i オプション等で) シェル自身が設定したりします。

$

シェルのプロセス ID に展開されます。 () を使ったサブシェルの内部では、$ はサブシェルではなく、 現在のシェルのプロセス ID に展開されます。

!

最後に実行されたバックグラウンド (非同期) コマンドの プロセス ID に展開されます。

0

シェルまたはシェルスクリプトの名前に展開されます。 これはシェルの初期化時に設定されます。 コマンドを記述したファイルを指定して bash を起動した場合、 $0 にはそのファイルの名前が設定されます。-c オプションを付けて bash を起動した場合、実行する文字列の後に引き数があれば、その最初の値が $0 に設定されます。引き数がなければ、 bash を起動するときに使用した名前が引き数 0 として与えられ$0 に設定されます。

_

シェルの起動時には、環境または引き数リストで渡された、 実行するシェルまたはシェルスクリプトの絶対パス名が設定されます。 その後では、直前のコマンドに対する最後の引き数 (展開後のもの) に展開されます。 また、実行する各コマンドの完全なパス名が設定され、 そのコマンドの環境にエクスポートされます。 メールをチェックするときには、 このパラメータは現在チェックしているメールファイル名を保持します。

1.13.3　シェル変数

以下の変数はシェルが設定します:

BASH

現在実行している bash を起動したときに使われた、完全なファイル名に展開されます。

BASHOPTS

コロン区切りのリストで、有効になっているシェルのオプションを示します。 リスト中のそれぞれの単語は、組み込みコマンド shopt の -s オプション (後述の シェルの組み込みコマンド を参照) に対する有効な引き数になっています。 BASHOPTS に現われるオプションは、shopt コマンドで on と表示されるものです。 もし bash を起動したときに、この変数が環境変数に設定されていれば、 起動ファイルを読み込む前に、リストにある全てのシェルオプションが 有効に設定されます。 この変数は読み込み専用です。

BASHPID

現在の bash のプロセス ID に展開されます。 bash を再初期化しないサブシェルのような、いくつかの環境においては、 $$ と値が異なります。

BASH_ALIASES

エイリアスの内部的なリストに対応する連想配列変数です。 エイリアスは通常、組み込みコマンド alias で操作します。 この配列に要素を追加すると、エイリアスのリストにも追加されます。 配列から要素を削除すると、エイリアスがリストから削除されます。

BASH_ARGC

bash の現在の呼び出しスタックについて、フレームごとの引き数の数が並んだ配列変数です。現在のサブルーチン (シェル関数や、. か source で実行されたスクリプト) の引き数の数が、スタックの一番上に置かれます。サブルーチンが実行されると、渡されたパラメータの数が BASH_ARGC に追加されます。シェルが BASH_ARGC を設定するのは、拡張デバッグモードのときだけです (後述の組み込みコマンド shopt の extdebug オプションを参照)。

BASH_ARGV

bash の現在の呼び出しスタックについて、全ての引き数が並んだ配列変数です。一連の呼び出しのうち、最後に呼ばれたサブルーチンの最後の引き数が、スタックの一番上に置かれます。最初に呼ばれたサブルーチンの最初の引き数が、スタックの一番下になります。サブルーチンが実行されると、渡されたパラメータが BASH_ARGV に追加されます。シェルが BASH_ARGV を設定するのは、拡張デバッグモードのときだけです (後述の組み込みコマンド shopt の extdebug オプションを参照)。

BASH_CMDS

bash が内部に持つ、コマンドのハッシュテーブルに対応する 連想配列変数です。 ハッシュテーブルは通常、組み込みコマンド hash で操作します。 この配列に要素を追加すると、ハッシュテーブルにも追加されます。 配列から要素を削除すると、ハッシュテーブルから削除されます。

BASH_COMMAND

現在実行しているか実行しようとしているコマンドです。 ただし、トラップによってシェルが実行しているコマンドは別で、 その場合は、トラップの発動時に実行していたコマンドになります。

BASH_EXECUTION_STRING

起動オプション -c で指定されたコマンドです。

BASH_LINENO

ソースファイル中の行番号からなる配列変数で、 それぞれの要素は FUNCNAME の各要素が呼び出された位置に対応します。 つまり、`${BASH_LINENO[$i]}` はソースファイル (`${BASH_SOURCE[$i+1]}`) 中で `${FUNCNAME[$i]}` が呼び出された行番号です。 別のシェル関数から参照されると `${BASH_LINENO[$i-1]}` になります。 現在の行番号は LINENO で取得できます。

BASH_REMATCH

条件コマンド [[中の二項演算子 =~ により設定される配列変数です。インデックス 0 の要素は、正規表現全体にマッチする部分文字列です。 インデックス n の要素は、n 番目の括弧による部分式に マッチした部分文字列です。 この変数は読み込み専用です。

BASH_SOURCE

ソースファイル名からなる配列変数で、それぞれの要素は配列変数 FUNCNAME の要素のシェル関数がそれぞれ定義されたファイル名に対応します。シェル関数 `${FUNCNAME[$i]}` はファイル`${BASH_SOURCE[$i]}` で定義され、 `${BASH_SOURCE[$i+1]}` から呼ばれています。

BASH_SUBSHELL

サブシェルやサブシェル環境が作成されるたびに 1 ずつ増えます。 初期値は 0 です。

BASH_VERSINFO

読み込み専用の配列変数で、配列の各要素は現在実行されている bash のバージョン情報を持っています。 配列変数の要素に代入される内容を以下に示します: BASH_VERSINFO[0] メジャーバージョン番号 (リリース)。BASH_VERSINFO[1] マイナーバージョン番号 (バージョン)。BASH_VERSINFO[2] パッチレベル。BASH_VERSINFO[3] ビルドバージョン。BASH_VERSINFO[4] リリースステータス (beta1 など)。BASH_VERSINFO[5] MACHTYPE の値。

BASH_VERSION

現在実行している bash のバージョンを示す文字列に展開されます。

COMP_CWORD

現在カーソル位置がある単語の 配列変数 `${COMP_WORDS}` におけるインデックスです。 この変数はプログラム補完機能 (後述の プログラム補完 を参照) から呼ばれたシェル関数においてのみ有効です。

COMP_KEY

現在の補完関数を呼び出したキー (またはキーシーケンスの最後のキー) です。

COMP_LINE

現在のコマンドラインです。 この変数はプログラム補完機能 (後述のプログラム補完 を参照) から呼ばれたシェル関数や外部コマンドにおいてのみ有効です。

COMP_POINT

現在のコマンドの先頭からの相対値として与えられた カーソル位置のインデックスです。 現在のカーソル位置が現在の現在のコマンドの最後にある場合、 この変数の値は `${#COMP_LINE}` と等しくなります。この変数はプログラム補完機能 (後述の プログラム補完 を参照) から呼ばれたシェル関数や外部コマンドにおいてのみ有効です。

COMP_TYPE

補完関数を呼び出した補完のタイプに対応する整数値が設定されます。TAB は通常の補完です。 ? は連続したタブ入力による候補のリスト表示です。 ! は途中まで補完した後の候補のリスト表示です。 @ は、部分的な補完ができないときの候補のリスト表示です。 % はメニュー補完 (menu completion) です。 この変数はプログラム補完機能 (後述のプログラム補完 を参照) から呼ばれたシェル関数と外部コマンドにおいてのみ有効です。

COMP_WORDBREAKS

単語補完のときに readline ライブラリが単語分割の区切り文字として扱う文字の並びです。COMP_WORDBREAKS を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

COMP_WORDS

現在のコマンドラインの各単語からなる配列変数 (後述の 配列 参照) です。 コマンドラインは readline と同じように前述した COMP_WORDBREAKS によって単語に分割されます。 この変数はプログラム補完機能 (後述の プログラム補完 を参照) から呼ばれたシェル関数においてのみ有効です。

COPROC

無名のコプロセス (前述の コプロセス を参照) が入出力するファイル・ディスクリプターを保持する配列変数 (後述の 配列 参照) です。

DIRSTACK

現在のディレクトリスタックの内容を持つ配列変数 (後述の 配列 を参照) です。組み込みコマンド dirs を使うと、スタック中のディレクトリがスタック順に表示されます。 配列変数の要素に代入を行うと、 既にスタックに入っているディレクトリを変更できますが、 ディレクトリの追加と削除を行うためには、組み込みコマンドの pushd と popd を使わなければなりません。 この変数に代入を行っても現在の作業ディレクトリは変わりません。 DIRSTACK を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

EUID

現在のユーザの実効ユーザ ID に展開されます。 初期化はシェルの起動時に行われます。 この変数は読み込み専用です。

FUNCNAME

現在の呼び出しスタックにある全てのシェル関数名が 入った配列変数です。 インデックス 0 の要素は、実行中のシェル関数の名前です。最も下の要素 (最大のインデックスの要素) は ”main”. です。 この変数は、シェル関数を実行している間のみ存在します。 FUNCNAME への代入は効果がなく、エラーステータスを返します。 FUNCNAME を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

この変数の各要素は、BASH_LINENO や BASH_SOURCE の各要素に 対応します。これらを参照することで、呼び出しスタックの状態を確認できます。例えば、`${FUNCNAME[$i]}` はファイル `${BASH_SOURCE[$i+1]}` の行番号 `${BASH_LINENO[$i]}` から呼び出されたものです。 組み込みコマンド caller を実行すると、これらの情報から 現在の呼び出しスタックを表示します。

GROUPS

現在のユーザがメンバになっているグループのリストを含んだ配列変数です。 GROUPS への代入は効果がなく、エラーステータスを返します。GROUPS が unset された場合はこの変数の特殊な性質はなくなります。その後に再設定されたとしても元に戻りません。

HISTCMD

現在のコマンドの履歴番号 (履歴リストにおけるインデックス) です。HISTCMD を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

HOSTNAME

現在のホスト名が自動的に設定されます。

HOSTTYPE

bash を実行するマシンの種類をユニークに記述する文字列が自動的に設定されます。 デフォルト値はシステム依存です。

LINENO

この変数が参照されると、その場所ごとに、 スクリプトや関数における現在の行番号 (1 から始まります) を表す 10 進値に置き換わります。スクリプトや関数の内部でない場合には、 意味のある値が代入されることは保証されません。 LINENO を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

MACHTYPE

bash を実行するシステムの種類を完全に指定する文字列が、 GNU 標準の cpu-company-system の形式で設定されます。 デフォルト値はシステム依存です。

MAPFILE

組み込みコマンド mapfile に変数名が指定されなかったときに 読み込んだテキストを保持する配列変数 (後述の 配列 を参照) です。

OLDPWD

cd コマンドで設定された、1 つ前の作業ディレクトリ。

OPTARG

組み込みコマンド getopts で処理した最後のオプション引き数の値です (後述の シェルの組み込みコマンド を参照)。

OPTIND

組み込みコマンド getopts で次に処理されるオプション引き数のインデックスです (後述の シェルの組み込みコマンド を参照)。

OSTYPE

bash を実行するオペレーティングシステムを記述する文字列が自動的に設定されます。 デフォルト値はシステム依存です。

PIPESTATUS

フォアグラウンドで最後に実行したパイプラインの各プロセスの 終了ステータスのリストを含む配列変数です (後述の 配列 を参照)。 パイプラインには 1 つのコマンドしかなくてもかまいません。

PPID

そのシェルの親のプロセス ID。この変数は読み込み専用です。

PWD

cd コマンドで設定された現在の作業ディレクトリ。

RANDOM

このパラメータが参照されるたびに、 0 から 32767 までのランダムな整数が生成されます。 RANDOM に値を代入すると、乱数の列を初期化できます。 RANDOM を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

READLINE_LINE

readline の編集バッファの内容です。 ”bind -x” (後述の シェルの組み込みコマンド を参照) で使います。

READLINE_POINT

readline の編集バッファでの挿入ポイントの位置です。 ”bind -x” (後述の シェルの組み込みコマンド を参照) で使います。

REPLY

組み込みコマンド read に引き数が与えられなかったときに読み込まれた行が設定されます。

SECONDS

このパラメータを参照すると、シェルが起動されてからの秒数が返されます。 SECONDS に値を代入した場合、それ以降の参照において返される値は、 代入された値と代入以降の秒数を足した値になります。 SECONDS を unset すると、この変数の特殊な性質はなくなります。後で再び set しても元には戻りません。

SHELLOPTS

コロン区切りのリストで、有効になっているシェルのオプションを示します。 リスト中のそれぞれの単語は、組み込みコマンド set の -o オプション (後述の シェルの組み込みコマンド を参照) に対する有効な引き数になっています。 SHELLOPTS に入っているオプションは、set -o を実行した場合にも on であると報告されます。 この変数が bash の起動時に環境変数に入っていた場合、 どの初期化ファイルを読むよりも前にリスト中のシェルオプションが有効になります。 この変数は読み込み専用です。

SHLVL

bash が起動するときに、環境変数で渡された値から 1 増やした値が設定されます。

UID

現在のユーザのユーザ ID に展開されます。 初期化はシェルの起動時に行われます。 この変数は読み込み専用です。

以下の変数はシェルが使用します。場合によっては、 bash がデフォルト値を変数に代入します。 そのような場合についてはそれぞれ注記します。

BASH_ENV

bash がシェルスクリプトを実行するときにこの値が設定されている場合、 この値は (~/.bashrc のように) シェルを初期化するコマンドが書かれているファイル名と解釈されます。 BASH_ENV の値をファイル名として処理する前には、パラメータ展開、コマンド置換、 算術的展開が行われます。 この結果のファイルを検索する際には PATH は使用されません。

BASH_XTRACEFD

有効なファイル・ディスクリプターに対応する整数をセットすると、bash は set -x が設定されたときに、生成されたトレース出力を そのファイル・ディスクリプターに出力します。 BASH_XTRACEFD が unset されるか、新しい値が代入されると、 それまで設定されていたファイル・ディスクリプターはクローズされます。 BASH_XTRACEFD を unset するか空文字列を代入すると、 トレース出力は標準エラー出力に送られます。 BASH_XTRACEFD に 2 (標準エラー出力のファイル・ディスクリプター) を設定してしまうと、 unset したときに標準エラー出力がクローズされてしまうことに 注意してください。

CDPATH

cd コマンドの検索パスです。これは、 cd コマンドで指定した対象ディレクトリを探すディレクトリをコロンで区切って並べたリストです。 例えば、 ".:~:/usr" といった値になります。

COLUMNS

組み込みコマンド select によって、選択されたリストを表示する際の端末幅の決定に用いられます。SIGWINCH を受信すると自動的に設定されます。

COMPREPLY

bash が可能な補完候補を読み込む配列変数です。 この値はプログラム補完機能 (後述の プログラム補完を参照) によって呼び出されたシェル関数によって生成されます。

EMACS

bash が起動したときにこの環境変数が設定されて値が "t", になっていると、Emacs のシェルバッファで動作しているとみなし、 行編集を無効にします。

ENV

BASH_ENV と類似の変数で、POSIX モードで使われます。

FCEDIT

組み込みコマンド fc が使うデフォルトのエディタです。

FIGNORE

ファイル名補完 (後述の READLINE を参照) を行う際に無視するサフィックスを コロンで区切って並べたリストです。 FIGNORE のエントリのいずれかにサフィックスがマッチするファイル名は、 ファイル名補完にマッチするファイルのリストから除外されます。 例えば値として ".o:~" を設定します。

FUNCNEST

0 より大きい数値を設定すると、関数呼び出しを何重まで許すかの 最大レベルを決めます。 このレベルを超えて関数を呼び出すと、コマンドが異常終了します。

GLOBIGNORE

パス名展開で無視するファイル名の集合を定義するパターンを コロンで区切って並べたリストです。 パス名展開パターンにマッチするファイル名が GLOBIGNORE 内のパターンのどれかにもマッチする場合、マッチしたもののリストから削除されます。

HISTCONTROL

履歴リストに入れるコマンドを制御する値をコロンで区切って並べたリストです。ignorespace の値が設定されていると、空白 文字で始まる行は履歴リストに入りません。ignoredups の値が設定されていると、履歴の最後の行にマッチする行は履歴リストに入りません。ignoreboth を指定すると ignorespace と ignoredups の両方が指定されます。 erasedups の値が設定されていると、行が保存される前に、現在の行に一致する過去の行が履歴リストから削除されます。 上記以外の値を設定しても無視されます。 HISTCONTROL が設定されていない場合や、不正な値が設定されている場合には、 シェルのパーザが読み込んだ全ての行は HISTIGNORE の値が示す条件の下で履歴リストに保存されます。 複数行にまたがる複合コマンドの場合は 2 番目以降の行が調べられることはありません。よって、これらは HISTCONTROL の値に関わらず履歴に追加されます。

HISTFILE

コマンド履歴が保存されるファイルの名前 (後述の 履歴 を参照)。デフォルト値は ~/.bash_history です。設定されていない場合、 対話シェルが終了するときに履歴の保存が行われません。

HISTFILESIZE

履歴ファイルに保持する履歴の最大数です。この変数に値が代入された場合、 その行数を越えないように、必要に応じて古いエントリを削除して履歴ファイルを 切り詰めます。 デフォルト値は 500 です。対話シェルが終了するときにも、 履歴ファイルのサイズはファイル書き込みの後にこのサイズに切り詰められます。

HISTIGNORE

どのコマンド行を履歴リストに保存するかを決めるために使うパターンを コロンで区切って並べたリストです。 それぞれのパターンは行の先頭から比較され、 行全体と完全に一致しなければなりません (‘*’ が暗黙的に追加されるこ とはありません)。行に対する各パターンの評価は HISTCONTROL で指定したチェックが行われた後で実行されます。 通常のシェルのパターンマッチング文字以外に、‘&’ が履歴の前の行にマッチします。 バックスラッシュを使って ‘&’ をエスケープできます。 マッチングを試みる前にバックスラッシュは取り除かれます。 複数行にまたがる複合コマンドの場合、2 番目以降の行は調べられません。 よって、これらは HISTIGNORE の値に関わらず履歴に追加されます。

HISTSIZE

コマンド履歴に記憶するコマンドの数 (後述の HISTORY を参照)。 デフォルトは 500 です。

HISTTIMEFORMAT

この変数に空でない値が設定されると、 組み込みコマンド history で履歴エントリを表示するときに タイムスタンプを表示するための strftime(3) の書式文字列 として使われます。 この変数が設定されると、ほかのシェルのセッションでも使えるように タイムスタンプは履歴ファイルに書き込まれます。 タイムスタンプはほかの履歴行と区別するために履歴のコメントとなります。

HOME

現在のユーザのホームディレクトリです。 組み込みコマンド cd のデフォルトの引き数になります。 この変数の値は、チルダ展開を実行するときにも使われます。

HOSTFILE

/etc/hosts と同じフォーマットであり、 シェルがホスト名を補完する必要があるときに読み込むファイルの名前を示します。 シェルの実行中でも補完するホスト名のリストを変更できます。 この変数が変更された次の機会にホスト名の補完を試みるとき、 bash は新しいファイルの内容を既存のデータベースに追加します。 HOSTFILE が設定されているがその値が空文字列の場合や、 読み込めるファイルの名前ではない場合には、 bash は補完可能なホスト名のリストを取得するために/etc/hosts を使用します。 HOSTFILE が unset された場合は、ホスト名のリストはクリアされます。

IFS

内部フィールド区切り文字 (Internal Field Separator) です。展開を行った後に単語を分割する場合や、組み込みコマンドの read を使ったときに行を単語に分割する場合に使われます。 デフォルト値は “ < 空白 >< タブ >< 改行 > ” です。

IGNOREEOF

単独で入力された EOF 文字を受け取ったときの対話シェルの動作を制御します。 この変数が設定されていれば、指定されている値の数だけの EOF 文字を連続して行頭の文字として入力しなければ bash は終了しません。この変数に数値以外の値が設定されている場合や、 空の値が設定されている場合には、デフォルト値として 10 が使われます。この変数が存在しなければ、 EOF 文字はシェルへの入力の終わりを示します。

INPUTRC

readline の起動ファイルのファイル名です。これはデフォルト値の~/.inputrc (後述の READLINE を参照) を上書きします。

LANG

ロケールカテゴリが LC_ で始まる変数によって明示的に指定されて いない場合、そうしたカテゴリのロケールを決定するのに使用されます。

LC_ALL

この変数はロケールカテゴリを指定する LC_ 変数と LANG の値に優先します。

LC_COLLATE

この変数はパス名展開の結果をソートするときに使用される照合順序と、 パス名展開とパターンマッチングにおける 範囲表現、等値クラス、照合順序の動作を決定します。

LC_CTYPE

この変数は、パス名展開とパターンマッチングにおける 文字の解釈と文字クラスに含まれる文字を決めます。

LC_MESSAGES

この変数は、$ の後に続くダブルクォートされた文字列の 翻訳に使うロケールを決めます。

LC_NUMERIC

この変数は数字のフォーマットに使用するロケールカテゴリを決定します。

LINES

組み込みコマンド select によって、選択されたリストを表示する際の行数の決定に用いられます。 SIGWINCH を受信すると自動的に設定されます。

MAIL

このパラメータにファイル名またはディレクトリ名が設定されており、かつ変数 MAILPATH が設定されていなければ、 bash は指定されたファイルまたは Maildier 形式のディレクトリへの メールの到着をユーザに通知します。

MAILCHECK

bash がメールをチェックする頻度を (秒数で) 指定します。 デフォルト値は 60 秒です。 メールをチェックする時間になると、 シェルはプライマリのプロンプトを表示する前にチェックを行います。 この変数が unset された場合、 あるいはこの変数に 0 以上の数値以外が代入された場合は、 シェルはメールのチェックを行いません。

MAILPATH

メールのチェックに使うファイル名をコロンで区切って並べたリストです。 特定のファイルにメールが到着したときに出力される メッセージは、'?' を使ってファイル名をメッセージから区切ることによって指定できます。メッセージのテキスト中で使われたときは、 $_ は現在のメールファイルの名前に展開されます。 設定例: MAILPATH='/var/mail/bfox?"You have mail":~/shell-mail?"$_ has mail!"' この変数のデフォルト値は bash が与えますが、 bash が使うユーザのメールファイルの位置はシステム依存です (/var/mail/$USER 等)。

OPTERR

値として 1 が設定されている場合、 bash は組み込みコマンド getopts (後述の シェルの組み込みコマンド を参照) を使って生成したエラーメッセージを表示します。 シェルが起動されたり、シェルスクリプトが実行されたりするたびに、 OPTERR は 1 に初期化されます。

PATH

コマンドの検索パスです。 シェルがコマンドを検索するディレクトリをコロンで区切って並べたリストです (後述の コマンドの実行 を参照)。 PATH 中の長さ 0 の (空の) ディレクトリ名は、カレントディレクトリを示します。 空のディレクトリ名は、2 つのコロンを並べるか、 先頭や末尾のコロンで表します。デフォルトのパスはシステム依存で、 bash をインストールしたシステム管理者が設定します。 一般的な値は "/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin" です。

POSIXLY_CORRECT

bash が起動したときにこの環境変数が設定されていると、 起動オプション --posix を指定したときと同じように、どの初期化ファイルを読むよりも前に シェルが posix モード になります。 シェルの実行中にこの変数が設定されると、 bash は set -o posix コマンドを実行したときと同じように、posix モード が有効になります。

PROMPT_COMMAND

設定されていると、プライマリプロンプトを出す前に毎回、 この値がコマンドとして実行されます。

PROMPT_DIRTRIM

0 より大きい値が設定されると、 プロンプト文字列のエスケープシーケンス \w や \W (後述の プロンプト を参照) を展開するときに、ディレクトリがパス名の最後からこの数だけ残ります。 削られた部分は省略記号に置き換えられます。

PS1

このパラメータの値は展開されてプライマリのプロンプト文字列として使われます。 (後述の プロンプト を参照)、 デフォルト値は“ \s-\v\$ ” です。

PS2

このパラメータの値は PS1 と同じように展開され、セカンダリのプロンプト文字列として使われます。デフォルト値は “ > ” です。

PS3

このパラメータの値は select コマンド (前述の シェルの文法 を参照) のプロンプトとして使われます。

PS4

このパラメータは PS1 と同じように展開されます。この値は実行トレース中に bash が表示する各コマンド前に出力されます。 複数段の間接レベル (levels of indirection) を示すときは、 PS4 の最初の文字が必要に応じて複数回表示されます。 デフォルト値は “ + ” です。

SHELL

この環境変数にシェルのフルパス名が保存されています。 シェルを起動したときに設定されていない場合は、 bash が現在のユーザのログインシェルのフルパス名を代入します。

TIMEFORMAT

このパラメータの値は、予約語である time が先頭に付いているパイプラインに対して、 時間情報の表示の仕方を指定するフォーマット文字列として使われます。 % は、時間の値などに展開される エスケープシーケンスを示すための文字です。 エスケープシーケンスとその意味を以下に示します。 ただし、ブレース ([]) は省略可能であることを表しま す。

%%	% 文字そのもの。
%[p][l]R	経過した秒数。
%[p][l]U	ユーザモードで使われた CPU の秒数。
%[p][l]S	システムモードで使われた CPU の秒数。
%P	CPU のパーセンテージ。(%U + %S) / %R で算出されます。

p 省略可能で、精度 (precision) が何桁であるかを指定します。 つまり小数点以下の桁数を指定します。この値が 0 ならば、 小数点や小数の部分は出力されません。 また、小数点以下で指定できるのは 3 桁までです。 つまり、p の値が 3 より大きければ 3 に変更されます。 p を指定しなければ、この値は 3 となります。

l は省略可能ですが、指定すると、分を含み、 MMmSS.FF という形式の長いフォーマットになります。小数を含むかどうかは p の値によって決まります。

この変数が設定されていなければ、 bash は$'\nreal\t%3lR\nuser\t%3lU\nsys%3lS' という値が指定されているかのように動作します。 この値が空文字列ならば、時間の情報は表示されません。 フォーマット文字列の表示の際には、末尾に改行文字が追加されます。

TMOUT

0 より大きい値を設定すると、 TMOUT の値が組み込みコマンド read のデフォルトのタイムアウト値となります。 select コマンドは、端末からの入力の際に TMOUT 秒入力がないと終了します。 対話シェルではこの値は、プライマリのプロンプトを出してから 入力を待つ秒数として解釈されます この秒数待って入力が来ないと、 bash は終了します

TMPDIR

設定すると、値を bash がシェル用にテンポラリファイルを作る ディレクトリ名として使います。

auto_resume

この変数は、ユーザがジョブ制御をするときの方法を変更します。 この変数を設定した場合、1 語からなるリダイレクトなしの単純なコマンドを 入力すると、停止中のジョブの実行再開候補として扱われます。曖昧な指定は許されず、入力された文字列で始まるジョブが複数ある場合には、 最後にアクセスされたものが選ばれます。 停止中のジョブの名前 とは、コマンドを起動する際に使ったコマンドラインのことです。この変数の値に exact が設定されている場合、 与えられた文字列は停止中のジョブの名前に 正確にマッチしなければなりません。substring が設定されている場合は、 与えられた文字列は停止中のジョブの名前の部分文字列に マッチする必要があります。 substring という値は、ジョブ識別子の %? に似た機能を持っています (後述のジョブ制御 を参照)。 これ以外の値が設定されている場合、 与えられた文字列は停止中のジョブの名前の 先頭部分でなければなりません。これはジョブ識別子の %string と似た機能を持っています。

histchars

2 文字または 3 文字で、 履歴の展開とトークン分割 (後述の 履歴の展開 を参照) を制御します。 最初の文字は 履歴展開 (history expansion) 文字であり、 履歴展開の先頭を示す文字です。通常、これは '!' です。 2 番目の文字は 簡易置換 (quick substitution) 文字であり、直前に入力したコマンドの文字列を 別の文字列に置き換えて再実行するための省略表現として使います。 デフォルト値は '^' です。 3 番目の文字は省略可能です。単語の先頭でこの文字が見つかると、 行の残りの部分がコメントとなるような文字を指定します。 これは通常は '#' です。 履歴コメント文字があると、 その行の残りの単語に対する履歴置換はスキップされます。 この文字があるからといって、 必ずしもシェルのパーザが行の残りの部分をコメントとして扱うわけではありません。

1.13.4　配列

bash は 1 次元のインデックスによる配列と連想配列の変数を扱うことができます。 全ての変数は配列として使用できます。 declare 組み込みコマンドを使えば、明示的に配列を宣言できます。 配列のサイズの上限はありませんし、メンバのインデックス付けや代入を 連続的にしなければならないという条件もありません。 インデックスによる配列は整数 (算術式を含む) で参照します。インデックスは 0 から始まります。 連想配列は任意の文字列で参照します。

変数の代入の際に name[subscript]=value という記法が使われた場合、配列は自動的に生成されます。subscript は算術式として扱われますが、この式は評価すると 数値になるものでなければなりません。 subscript を評価すると 0 未満の数になる場合、最大のインデックス +1 からのオフセットとして扱われます。つまり、subscript が -1 のときは、配列の最後の要素を参照します。インデックスによる配列を明示的に宣言するには、 declare -a name を使います (後述の シェルの組み込みコマンド を参照)。 declare -a name[subscript] も許されます。このとき subscript は無視されます。

連想配列は declare -A name で作ります。

組み込みコマンドの declare と readonly を使うと、配列変数に対して属性を設定できます。 どの属性も配列のメンバ全てに対して適用されます。

配列の代入は name=(value1 ... valuen) という形式の複合代入 (compound assignment) を用いて行います。 ここでそれぞれの value の形式は [subscript]=string です。 インデックスによる配列では、ブラケット ([]) と subscript は省略できます。 ブラケットと subscript を省略しなかった場合、 そのインデックスに対して代入が行われます。 省略した場合には、代入される要素のインデックスは、 その文の中で直前に代入されたインデックスに 1 を加えたものになります。 インデックスは 0 から始まります。

連想配列への代入では、subscript は省略できません。

この記法は組み込みコマンド declare でも使えます。個別の配列要素に対する代入は、先に説明した name[subscript]=value の記法を使って行います。

配列の任意の要素は、`${name[subscript]}` を使って参照できます。 パス名展開との衝突を避けるためにブレースが必要です。 subscript が @ か * ならば、配列の参照は name の全ての要素に展開されます。こ

の 2 つの添字が異なるのは、 単語がダブルクォートの内部にある場合だけです。 単語がダブルクォートされていれば、`${name[*]}` は 1 つの単語に展開されます。 この単語は、配列の各メンバの値を特殊変数 IFS の最初の値で区切って並べたものです。一方、`${name[@]}` は、 name の各要素を別々の単語に展開します。配列のメンバが全くないときは、`${name[@]}` は空の単語に展開されます。 単語の中でダブルクォートの展開が行われるときには、 元の単語のダブルクォートより前の部分の後に 最初のパラメータの展開結果がつながり、 最後のパラメータの展開結果の後に 元の単語のダブルクォートより後の部分がつながります。これは特殊パラメータ * や @ の展開に似ています (前述の 特殊パラメータを参照)。 `${#name[subscript]}` は `${name[subscript]}` の長さに展開されます。 subscript が * または @ の場合は、 配列中の要素数に展開されます。添字なしで配列変数を参照すると、 0 番目の要素を参照したことになります。

配列変数は subscript に対して値が代入されていれば 設定されているとみなされます。 空文字列は有効な値です。

組み込みコマンドの unset は配列の削除に使われます。unset name[subscript] とすると、インデックスが subscript である配列の要素が削除されます。 パス名展開による意図しない副作用に注意してください。unset name (name は配列) または unset name[subscript] (subscript が * または@) とすると、 配列全体が削除されます。

組み込みコマンドの declare, local, readonly いずれにおいても、 -a オプションでインデックスによる配列を指定できます。 -A また、オプションで連想配列を指定できます。 両方が指定されたときには、 -A が優先します。 組み込みコマンド read では、 -a オプションを使えば標準入力から読み込んだ単語のリストを 配列に代入できます。 組み込みコマンド set と declare では、別の変数への代入に再利用できるような形で配列の値を表示します。

1.14　展開

展開はコマンドラインが単語へ分割された後に (コマンドライン上で) 行われます。行われる展開は 7 種類あります: ブレース展開 (brace expansion), チルダ展開 (tilde expansion), パラメータと変数の展開 (parameter and variable expansion), コマンド置換 (command substitution), 算術式展開 (arithmetic expansion), 単語の分割 (word splitting), パス名展開 (pathname expansion)。

展開の順序は次のようになります: ブレース展開、チルダ展開、パラメータ・変数・算術式展開、コマンド置換 (左から右へ)、単語分割、パス名展開。

これらに加えて プロセス置換 (process substitution) をサポートできるシステムもあります。

展開した部分の単語の数が変化することがあるのは、ブレース展開、 単語の分割、パス名展開だけです。ほかの展開では、 1 つの単語は 1 つの単語に展開されます。 この規則の唯一の例外は先に説明した "$@" と "`${name[@]}`" の展開 (パラメータ を参照) だけです。

1.14.1　ブレース展開

ブレース展開 (brace expansion) を使うと、任意の文字列を生成できます。この仕組みは パス名展開に似ていますが、 生成されたファイル名が実在する必要はありません。 ブレース展開されるパターンは、 前置部分 (preamble: 省略可能)、 対になるブレースで囲んだコンマ区切りの文字列またはシーケンス式、 後置部分 (postscript: 省略可能) を順に並べたものです。 前置部分はブレース内にある文字列それぞれの先頭部分に追加され、 後置部分は左から右に順に展開されて得られた それぞれの文字列の末尾に追加されます。

ブレース展開は入れ子にできます。 展開して得られた文字列はソートされず、左から右への順番がそのまま残ります。 例えば `a{d,c,b}e` は 'ade ace abe' と展開されます。

シーケンス式は `{x..y[..incr]}` の形になります。x と y は整数または 1 つの文字で、 省略可能な incr は増減分の整数です。 整数が与えられると、 x と y の間の両端を含む数を全て列挙した形に展開されます。 与る整数の前に 0 を付けると、全ての項が同じ幅に揃えられます。 つまり、x か y のどちらかが 0 で始まる場合、生成される全ての項が 同じ文字数になるように、必要であれば前に 0 が付けられます。 文字が与えられると 辞書順で x と y の間の両端を含む文字を全て列挙した形に展開されます。 x と y は同じ型である必要があります。 incr が与えられると、その値が各項の間の差となります。incr のデフォルトは、1 または -1 のうち適切な方です。

ブレース展開はほかのどの展開よりも前に実行されます。 また、ほかの展開において特殊な意味を持つ文字もそのまま結果に残ります。 つまり、厳密にテキスト操作だけを行います。 bash は、展開の文脈やブレースの間のテキストに対して 文法的な解釈を適用することは一切ありません。

正しい形のブレース展開には、クォートされていない開きブレースと 閉じブレース、そしてシーケンス式か少なくとも 1 つのクォートされていないコンマが 含まれていなければなりません。正しい形でないブレース展開は全て、 変更されないでそのまま残ります。 `{` や , をバックスラッシュでクォートすれば、 ブレース展開の一部と解釈されないようにできます。 パラメータ展開との衝突を避けるため、文字列 `${` はブレース展開の対象とは解釈されません。

この仕組みは、 生成される文字列の共通先頭部分が上記の例より長い場合に、短縮表現としてよく使用されます:

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

または

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

ブレース展開の導入によって、伝統的な sh とは少し非互換になった部分があります。 sh は単語の一部として開きブレースや閉じブレースが現われても特別扱いはせず、 そのまま出力に残します。 bash はブレース展開の結果として単語からブレースを取り除きます。例えば sh に `file{1,2}` のように入力された単語はそのままの形で出力されますが、 bash ではこの同じ単語は展開されて file1 file2 のように出力されます。 厳密に sh と互換にしたければ、bash を +B オプションを付けて起動するか、 set コマンドに +B オプションを与えてブレース展開を無効にしてください (後述する シェル組み込みコマンド を参照)。

1.14.2　チルダ展開

クォートされていないチルダ (‘~’) で単語が始まった場合、クォートされていないスラッシュよりも前にある文字全て (クォートされていないスラッシュが無ければ全ての文字) はチルダプレフィックス (tilde-prefix) と解釈されます。 クォートされている文字がチルダプレフィックス中に無ければ、 チルダプレフィックス中のチルダ以降の文字は、 ログイン名 (login name) になるかもしれない文字列として扱われます。 このログイン名が空文字列ならば、チルダはシェルパラメータ HOME の値に置き換えられます。 HOME が設定されていない場合は、代わりにシェルを実行しているユーザの ホームディレクトリに置き換えられます。 ログイン名が空でなければ、 チルダプレフィックスは指定されたログイン名に対応する ホームディレクトリに置き換えられます。

チルダプレフィックスが ‘~+’ ならば、チルダプレフィックスはシェル変数 PWD の値に置き換えられます。 チルダプレフィックスが ‘~-’ ならば、シェル変数 OLDPWD の値に置き換えられます (値が設定されていれば)。 チルダプレフィックス中のチルダより後の文字が数値 N であれば (数値の前に ‘+’ や ‘-’ を置くこともできます)、 チルダプレフィックスはディレクトリスタック内の対応する要素に置換されます。 置換される要素は、チルダプレフィックスを引き数として組み込みコマンド dirs を実行したときに表示されるものです。 チルダプレフィックスにおけるチルダ以降の文字が、 先行する ‘+’ や ‘-’ のない数値である場合は、‘+’ であるとみなされます。

ログイン名が有効でない場合や、チルダ展開が失敗した場合には、 単語は置き換えられません。

全ての変数代入において、: や = の直後にクォートされていないチルダプレフィックスがないかチェックが行われます。 もし見つかった場合にはこれらもチルダ展開されます。 したがって、 PATH, MAILPATH, CDPATH への代入にチルダを含むファイル名を使えば、 シェルは展開された値を代入します。

1.14.3　パラメータの展開

‘$’ 文字があると、パラメータ展開、コマンド置換、 算術式展開が行われます。展開されるパラメータ名やシンボルは、 ブレースで括ることもできます。ブレースは省略可能ですが、 変数の直後に変数名の一部と解釈できる文字が置かれた場合に、 その文字と共に変数が展開されてしまうのを防ぐために用意されています。

ブレースを使った場合、対になるのは最初に表れる ‘}’ です。 ただしバックスラッシュでエスケープされているものや クォートされている文字列中のものは含まれませんし、ブレースの内側にある 算術式展開やコマンド置換、パラメータ展開に入っているものも含まれません。

${parameter}

parameter の値に置換されます。ブレースが必要になるのは、parameter が 2 桁以上の数字を持つ位置パラメータの場合や、parameter の直後の文字を名前の一部として解釈させたくない場合です。

parameter の最初の文字が感嘆符ならば、変数間接展開が行われます。 bash は残りの parameter からなる変数の値を変数の名前と見なします。 そしてそこで得られた名前の変数を展開した値を、置換処理の続きで使います。 これが間接展開 です。 ただし `${!prefix*}` や `${!name[@]}` の展開は例外です。 これ

は以下で説明します。 間接展開を表すには、感嘆符は左ブレースの直後に続ける必要があります。

以下に示すそれぞれの場合、word に対してチルダ展開、 パラメータ展開、コマンド置換、算術式展開が行われます。

部分文字列展開以外の場合、以下の形式で、 bash はパラメータが設定されているか、空ではないかを調べます。 コロンを省略した場合には設定されているかどうかのみを調べます。

${parameter:-word}

デフォルトの値を使います。 parameter が設定されていないか空文字列であれば、 word を展開したものに置換されます。そうでなければ、parameter の値に置換されます。

${parameter:=word}

デフォルトの値を代入します。 parameter が設定されていないか空文字列であれば、 word を展開したものが parameter に代入されます。それから parameter の値への置換が行われます。 位置パラメータや特殊パラメータへの代入をこのように行うことはできません。

${parameter:?word}

空文字列または設定されていない場合にエラーを表示します。parameter が空文字列または設定されていない場合、word を展開したもの (word がなければ パラメータが空文字列または設定されていないことを示すメッセージ) が標準エラー出力に出力されます。シェルが対話的でなければ、 シェルは終了します。パラメータに空文字列以外が設定されていれば、 parameter の値への置換が行われます。

${parameter:+word}

別の値を使用します。 parameter が空文字列または設定されていなければ、空文字列に置換されます。そうでなければ word を展開したものに置換されます。

${parameter:offset}

${parameter:offset:length}

部分文字列展開。 parameter を展開したものから最大 length 文字を取り出します。 先頭の文字は offset で指定します。length を省略すると、 offset で指定した文字を先頭にして、 parameter の残り全部が含まれる部分文字列に展開します。 length と offset は算術式です (後述の 算術式評価 を参照)。 offset を評価すると 0 未満の数になる場合、この値は parameter の値の末尾からのオフセットとして使われます。 length を評価すると 0 未満の数になる場合、 parameter が@ ではなく、配列でも連想配列でもなければ、 この値は文字数ではなく parameter の値の末尾からのオフセットとして使われ、 展開結果は 2 つのオフセットの間の部分文字列となります。 parameter が @ ならば、結果は offset から始まる length 個の位置パラメータになります。 parameter が @ または * のインデックスが付いている配列名ならば、 結果は配列の `${parameter[offset]}` を先頭とする要素 length 個となります。 負の offset は、指定された配列の最大

のインデックス + 1 からの相対値と解釈されます。 連想配列に部分文字列展開した場合の結果は決められていません。 負のオフセットを指定するときには、:- 式と混同されないよう、 1 つ以上の空白でコロンと離す必要があることに注意してください。 位置パラメータを使う場合以外は、 部分文字列のインデックスは 0 から始まります。位置パラメータの場合には、 インデックスは 1 から始まります。 位置パラメータが使われて offset が 0 の場合、 $0 の値が先頭に置かれます。

${!prefix*}

${!prefix@}

前方一致する変数名。 prefix で始まる全ての変数の名前に展開して、IFS 特殊変数の最初の文字によって区切ります。 ダブルクォートの中で @ が使われた場合、それぞれの変数の名前は 別々の単語に展開されます。

${!name[@]}

${!name[*]}

配列のキーのリスト。 name が配列変数であれば、配列 name の インデックス (キー) のリストに展開されます。 name が配列でない場合は、name が設定されていれば 0 に、 そうでなければ空に展開されます。 ダブルクォートの中で @ が使われた場合、それぞれのキーは別々の単語に展開されます。

${#parameter}

パラメータの長さ。 parameter の値に含まれる文字数に置換されます。 parameter が * または @ ならば、位置パラメータの数に置換されます。 parameter が * または @ が添字になっている配列名ならば、配列中の要素数に置換されます。

${parameter#word}

${parameter##word}

パターンに前方一致した部分を取り除く。 word が展開され、パス名展開の場合と同じようなパターンを作ります。 このパターンが parameter の値の先頭部分とマッチする場合、展開して得られる値は parameter を展開した値から最短一致パターン (“ # ”の場合) または最長一致パターン (“ ## ” の場合) を取り除いたものになります。parameter が @ または * である場合、 パターンを削除する操作は全ての位置パラメータに順番に適用され、 展開結果はリストとして得られます。 parameter が @ または * が添字になっている配列変数である場合、 パターンを削除する操作は配列の全ての要素に順番に適用され、 展開結果はリストとして得られます。

${parameter%word}

${parameter%%word}

パターンに後方一致した部分を取り除く。 word が展開され、パス名展開の場合と同じようなパターンを作ります。 このパターンが parameter を展開した値の末尾の部分とマッチする場合、展開結果は parameter を展開した値から最短一致パターン (“ % ” の場合) または最長一致パターン (“ %% ” の場合) を取り除いたものになります。parameter が @ または * である場合、 パターンを削除する操作は全ての位置パラメータに順番に適用され、 展開結果はリストとして得られます。 parameter が @ または * が添字になっている配列変数である場合、 パターンを削除する操作は配列の全ての要素に順番に適用され、 展開結果はリストとして得られます。

${parameter/pattern/string}

パターンの置換。 pattern が展開され、 パス名展開の場合と同じようなパターンを作ります。 parameter の展開が行われ、 その値のうち pattern に最長一致する部分が string に置換されます。 pattern が/ で始まる場合には、pattern に マッチした部分は全て string に置換されます。 そうでない場合には、最初にマッチした部分だけが置換されます。 pattern が # で始まる場合には、パターンは parameter を展開した値の先頭にマッチしなければなりません。 pattern が % で始まる場合には、パターンは parameter を展開した値の末尾にマッチしなければなりません。 string が空の場合には pattern にマッチした部分は削除されます。 またこの場合には、pattern の後に続く / は省略可能です。 parameter が @ または * である場合、置換操作は全ての位置パラメータに順番に適用され、 展開結果はリストとして得られます。 parameter が @ または * が添字になっている配列変数である場合、 置換操作は配列の全ての要素に順番に適用され、展開結果はリストとして得られます。

${parameter^pattern}

${parameter^^pattern}

${parameter,pattern}

${parameter,,pattern}

大文字小文字の変換。 parameter に含まれるアルファベットの大文字小文字を変換します。 pattern が展開され、 パス名展開の場合と同じようなパターンを作ります。 ^ 演算子は pattern にマッチした小文字を大文字に変換します。 , 演算子は pattern にマッチした大文字を小文字に変換します。 ^^ 演算子と ,, 演算子は、マッチした全ての文字を変換します。 ^ 演算子と , 演算子の場合は、マッチした最初の文字だけ変換します。 pattern を省略した場合、? を指定したものとして扱われ、 全ての文字にマッチします。 parameter が @ または * である場合、大文字小文字の変換は全ての位置パラメータに順番に適用され、 展開結果はリストとして得られます。 parameter が添字に @ または * の付いた配列変数の場合は、配列の要素のそれぞれに大文字小文字の変換が適用され、 結果はリストに展開されます。

1.14.4　コマンド置換

コマンド置換 (command substitution) を用いると、 コマンド名をコマンドの出力で置き換えられます。 コマンド置換には以下の 2 つの形式があります:

```
$(command)
```

または

```
‘command‘
```

bash は command を実行し、 command の標準出力でコマンド置換の部分を置き換えます。 この際、末尾の改行文字は削除されます。 文字列の途中にある改行文字は削除されませんが、 単語分割の際に削除されることがあります。 コマンド置換 $(cat file) は、同じ意味を持ち、 しかも高速な $(< file) に置き換え可能です。

バッククォートを使う古い形式の置換を用いたとき、 バックスラッシュは文字通りの意味を保ちますが、 $, ‘, \ の前にある場合は例外となります。 コマンド置換は、バックスラッシュが前置されていない バッククォートまでの部分です。 $(command) という形式を用いたときは、 括弧の間にある全ての文字がコマンドとなります。 特別扱いされる文字はありません。

コマンド置換は入れ子にできます。 バッククォート形式のときに入れ子を行うには、 内側のバッククォートをバックスラッシュでエスケープします。

置換がダブルクォート内部にある場合には、 置換の結果に対する単語分割とパス名展開は行われません。

1.14.5　算術式展開

算術式展開を使うと、算術式を評価して、その結果に置換できます。 算術式展開のフォーマットを次に示します:

$((expression))

expression はダブルクォート内部にある場合と同様に扱われますが、 括弧の内側のダブルクォートが特別扱いされることはありません。 式に含まれる全てのトークンに対して、 パラメータ展開・文字列展開・コマンド置換・クォートの削除が行われます。 算術式置換は入れ子にできます。

評価は後述の 算術式評価 で示す規則に基づいて行われます。 expression が不正であれば、 bash は評価の失敗を示すメッセージを出力し、置換を全く行いません。

1.14.6　プロセス置換

プロセス置換 (process substitution) がサポートされるのは、 名前付きパイプ (FIFO) または ファイル・ディスクリプターの /dev/fd 形式での指定 をサポートしているシステムです。これは <(list) または >(list) の形になります。 プロセス list は、その入力や出力が FIFO または /dev/fd 中の 何らかのファイルに接続された状態で実行されます。 このファイルの名前は、展開の結果として、 引き数の形で現在のコマンドに渡されます。 >(list) の形式を使った場合、 ファイルへの書き込みは list への入力となります。 <(list) の形式を使った場合、 引き数として渡されたファイルは list の出力を得るために読み込まれます。

利用可能であれば、プロセス置換 (process substitution) は、 パラメータ展開、変数展開、コマンド置換、算術式展開と同時に行われます。

1.14.7　単語の分割

パラメータ展開、コマンド置換、算術式展開が行われたのが、ダブル クォートの内側ではない場合、シェルは展開の結果をスキャンして、 単語分割 を行います。

シェルは IFS のそれぞれの文字を区切り文字として扱い、 ほかの展開の結果をこれらの文字によって単語に分割します。 IFS が設定されていないか、その値がデフォルト値の < スペース >< タブ >< 改行 > と全く同じならば、前段の展開の結果の先頭や末尾の < スペース >, < タブ >, < 改行 > の並びは無視され、 先頭と末尾以外の IFS 文字の並びで単語が区切られます。 IFS の値がデフォルト以外のときに、 スペース や タブ という空白文字の並びが単語の先頭と末尾で無視されるのは、 その空白文字が IFS の値に含まれるとき (IFS の空白文字の一つであるとき) だけです。 IFS に含まれ、 IFS 空白文字ではない文字は全て、隣接する任意の IFS 空白文字と一緒になってフィールドの区切りとなります。 IFS 空白文字の列も区切り文字として扱われます。 IFS の値が空文字列であれば、単語分割は全く行われません。

明示的に指定した空の引き数 (”” または ”) は削除されずに残ります。クォートされていない暗黙的な空の引き数が、 値を持たないパラメータを展開した結果として得られますが、 これらは削除されます。 値を持たないパラメータがダブルクォート内部で展開されると、 空である引き数となり、消されずに残ります。

展開が行われなければ単語分割も行われない点に注意してください。

1.14.8　パス名展開

-f オプションが指定されていなければ、単語分割を行った後に bash はそれぞれの単語が *, ?, [を含んでいるかどうか調べます。 これらの文字のいずれかが見つかると、その単語は パターン とみなされ、 パターンにマッチするファイル名を アルファベット順にソートしたリストに置換されます。 マッチするファイル名が見つからず、かつシェルのオプション nullglob が無効ならば、その単語は変更されずにそのまま残ります。 nullglob オプションが設定されていて、かつマッチするファイル名が見つからなければ、 その単語は削除されます。 failglob オプションが設定されていて、かつマッチするファイル名が見つからなければ、 エラーメッセージが表示されコマンドは実行されません。 シェルのオプション nocaseglob が有効ならば、マッチングにおいてアルファベットの大文字と小文字は区別さ れません。 パターンをパス名展開に使うとき、名前の先頭やスラッシュの直後の “ . ” は明示的にマッチさせなければなりません。ただしシェルのオプション dotglob が設定されている場合は例外です。 パス名のマッチングを行うとき、 スラッシュ文字は必ず明示的にマッチさせなければなりません。 これ以外の場合には、 “ . ” が特別扱いされることはありません。 シェルのオプション nocaseglob, nullglob, failglob, dotglob の詳しい説明については、後述の シェルの組み込みコマンド に書かれている shopt の説明を参照してください。

シェル変数 GLOBIGNORE を使って、 パターン にマッチするファイル名の集合を制限できます。 GLOBIGNORE が設定されていれば、マッチするファイル名のうち GLOBIGNORE 中のパターンにもマッチしたものは、 マッチしたもののリストから取り除かれます。 ファイル名 “ . ” と “ .. ” は GLOBIGNORE に空でない値が設定されていても必ず無視されます。しかし、 GLOBIGNORE に空でない値を設定するとシェルオプションの dotglob が有効になるので、 “ . ” で始まるほかのファイル名は全てマッチします。 “ . ” で始めるファイル名を無視する古い動作をさせるには、 “ .* ” を GLOBIGNORE のパターンに含めてください。GLOBIGNORE を unset すると、 dotglob オプションは無効になります。

パターンマッチング

パターンに含まれる文字のうち、以下の特殊パターン文字以外の文字は、 自分自身にマッチします。NUL 文字がパターン中に現われてはいけません。 バックスラッシュは直後の文字をエスケープします。 このバックスラッシュは、マッチングでは無視されます 特殊パターン文字をその文字そのものにマッチさせるためには、 クォートしなければなりません。

特殊パターン文字は以下の意味を持っています:

*

空文字列を含む、任意の文字列にマッチします。 シェルオプション globstar が有効で、 * がパス名展開に用いられる場面では、 連続する 2 つの * だけのパターンが使われると、 その階層以下のディレクトリとそのサブディレクトリ、 そこにある全てのファイルにマッチします。 2 つ連続した * の後に / が続く場合には、 ディレクトリとそのサブディレクトリのみにマッチします。

?

任意の 1 文字にマッチします。

[...]

括られた文字のうち任意の 1 文字にマッチします。 2 つの文字の間にハイフンを入れたものは、 範囲表現 (range expression) を表します。 ソート順で両端を含む 2 つの文字の間にある任意の 1 文字と マッチします。 ソートには現在のロケールの照合順序 (collating sequence) と文字セットが用いられます。 [の次の文字が ! または ^ ならば、括られた文字に含まれない任意の 1 文字にマッチします。 範囲表現における文字のソート順は、 現在のロケール (およびシェル変数 LC_COLLATE が指定されていればその値) によって決定されます。 - は、文字集合の最初または最後の文字として含めるとマッチングの対象にできます。] は、文字集合の最初の文字として含めるとマッチングの対象にできます。

[と] の間では、文字クラス (character classes) を指定できます。 指定には [:class:] という記法を使います。ここで class は POSIX 標準で定義されている以下のクラスのいずれかです: alnum alpha ascii blank cntrl digit graph lower print punct space upper word xdigit 文字クラスは、そのクラスの属する任意の文字にマッチします。 文字クラス word は、文字、数字、_ にマッチします。

[と] の間では、等値クラス (equivalence class) を指定できます。 指定には [=c=] という記法を使います。 これは現在のロケールにおける定義において c と同じ 照合重み (collation weight) を持つ全ての文字にマッチします。

[と] の間では、[.symbol.] という記述は照合シンボル symbol にマッチします。

組み込みコマンドの shopt を使ってシェルのオプション extglob が有効にされていると、 拡張パターンマッチング演算子がいくつか認識されるようになります。 以下の説明では、pattern-list は | で区切られ

た 1 つ以上のパターンのリストであるものとします。 以下のサブパターンを 1 つあるいは複数使うことにより、 複合パターンを作れます。

?(pattern-list)

与えられたパターンが 0 回または 1 回現われるとマッチします。

***(pattern-list)**

与えられたパターンが 0 回以上現われるとマッチします。

＋(pattern-list)

与えられたパターンが 1 回以上現われるとマッチします。

@(pattern-list)

与えられたパターンに 1 回だけマッチします。

!(pattern-list)

与えられたパターンのどれでもないものにマッチします。

1.14.9　クォートの削除

先に処理される展開の後、クォートされていない \, ', " のうち、先の展開の結果でないものは全て削除されます。

1.15　リダイレクト

シェルが解釈する特別な記法を用いると、コマンドの実行前に入出力を リダイレクト できます。またリダイレクトを使うと、 現在のシェル実行環境に対してファイルをオープンしたり クローズしたりできます。以下に示すリダイレクト演算子は、 単純なコマンド の前や途中に置くことができ、また コマンドの後に置けます。 リダイレクトは左から右へと、現われた順に処理されます。

ファイル・ディスクリプター番号で始まるリダイレクトでは、 代わりに`{varname}` という形式の単語で始めることもできます。 この場合、>&- と <&- 以外のリダイレクト演算子では、 シェルは 10 より大きいファイル・ディスクリプターを割り当て、 varname に代入します。 `{varname}` で始まる >&- や <&- の場合には、varname の値は クローズするファイル・ディスクリプターを示します。

以下の説明においては、ファイル・ディスクリプター番号が省略され、 かつリダイレクト演算子の最初の文字が < ならば、リダイレクトは標準入力 (ファイル・ディスクリプター 0) を参照します。 リダイレ

クト演算子の最初の文字が > ならば、リダイレクトは標準出力 (ファイル・ディスクリプター 1) を参照します。

以下の説明では、リダイレクト演算子の次の単語に対しては、 特に説明しない限り、 ブレース展開・チルダ展開・パラメータ展開・コマンド置換・ 算術式展開・クォート削除・パス名展開・単語分割が行われます。 その単語が複数の単語に展開された場合はエラーになります。

リダイレクトの順番には意味がある点に注意してください。 例えば、次のコマンド

ls > dirlist 2>&1

は標準出力と標準エラー出力を両方ともファイル dirlist に書き込みますが、次のコマンド

ls 2>&1 > dirlist

では標準出力だけがファイル dirlist に書き込まれます。なぜなら後者の場合には、標準エラー出力は dirlist にリダイレクトされる前の標準出力の複製となるからです。

1.15.1　/dev/...

bash は、 以下の表にあるようなファイル名がリダイレクトに使用されると、それらを特別に扱います。

/dev/fd/fd

fd が有効な整数ならばファイル・ディスクリプター fd が複製されます。

/dev/stdin

ファイル・ディスクリプター 0 が複製されます。

/dev/stdout

ファイル・ディスクリプター 1 が複製されます。

/dev/stderr

ファイル・ディスクリプター 2 が複製されます。

/dev/tcp/host/port

host が有効なホスト名またはインターネットアドレスで port が整数のポート番号ならば、 bash は対応するソケットに対して TCP 接続のオープンを試みます。

/dev/udp/host/port

host が有効なホスト名またはインターネットアドレスで port が整数のポート番号ならば、 bash は対応するソケットに対して UDP 接続のオープンを試みます。

ファイルのオープンや作成に失敗すると、リダイレクトも失敗します。

9 より大きいファイル・ディスクリプターを使ったリダイレクトには注意が 必要です。シェルが内部的に使うファイル・ディスクリプターと競合する場合があるからです。

1.15.2　入力のリダイレクト

入力をリダイレクトすると、 word を展開した結果の名前を持つファイルがオープンされ、 ファイル・ディスクリプター n で読み込めるようになります。 n が指定されていなければ、読み込みは標準入力 (ファイル・ディスクリプター 0) で行われます。

入力のリダイレクトは、一般的には以下の形式です:

[n]<word

1.15.3　出力のリダイレクト

出力をリダイレクトすると、 word の展開した結果の名前を持つファイルがオープンされ、 ファイル・ディスクリプター n で書き込めるようになります。 n が指定されていなければ、書き込みは標準出力 (ファイル・ディスクリプター 1) に行われます。 ファイルが存在しなかった場合は作成されます。ファイルが存在した場合はサイズ 0 に切り詰められます。

出力のリダイレクトは、一般的には以下の形式です:

[n]>word

リダイレクト演算子が > であり、かつ 組み込みコマンド set で noclobber オプションが有効になっている場合、 word の展開で得たファイルが存在し、かつそれが通常ファイルならば、リダイレクトは失敗します。 リダイレクト演算子が >| の場合、もしくは リダイレクト演算子が > で組み込みコマンド set で noclobber オプションが有効になっていない場合、 word という名前のファイルが存在していてもリダイレクトが試みられます。

1.15.4　リダイレクトによる追加出力

この形式を使って出力のリダイレクトを行うと、 word を展開した結果の名前を持つファイルがオープンされ、 ファイル・ディスクリプター n に対する出力がこのファイルに追加されるようになります。 n を指定しなければ、標準出力 (ファイル・ディスクリプター 1) で追加されます。 ファイルが存在しなければ、新しく作られます。

追加出力は一般的には以下の形式です:

[n]>>word

1.15.5　標準出力と標準エラー出力のリダイレクト

この構造を使うと、 標準出力 (ファイル・ディスクリプター 1) と標準エラー出力 (ファイル・ディスクリプター 2) の両方を、 word を展開した結果の名前を持つファイルにリダイレクトできます。

標準出力と標準エラー出力に対する形式は 2 つあります:

&>word と >&word

両者のうち望ましいのは前者の方です。 上記は以下と同じ意味です:

>word 2>&1

1.15.6　標準出力と標準エラー出力の追加出力

この構造を使うと、 標準出力 (ファイル・ディスクリプター 1) と標準エラー出力 (ファイル・ディスクリプター 2) の両方を、 word を展開した結果の名前を持つファイルに追加できます。

標準出力と標準エラー出力の追加出力は以下の形式です:

&>>word

これは次のものと同じ意味です:

>>word 2>&1

1.15.7　ヒアドキュメント (Here Documents)

この形式のリダイレクトを用いると、 シェルは現在のソースから入力を読み込みます。 この読み込みは word を単独で含む行 (末尾にブランク文字があってはいけません) が現われるまで続きます。 その行までに読み込んだ行は、 コマンドの標準入力として扱われます。

ヒアドキュメントの形式を以下に示します:

```
<<[-]word
        here-document
delimiter
```

word に対するパラメータ展開・コマンド置換・算術式展開・ パス名展開は全く行われません。 word が一部でもクォートされている場合は、 delimiter は word のクォートをほどいた結果 (クォート文字を削除

した結果) となり、 ヒアドキュメントに含まれる行では展開が行われなくなります。 word がクォートされていなければ、 ヒアドキュメント中の全ての行に対して パラメータ展開・コマンド置換・算術式展開が行われます。 word がクォートされていない場合には、 \<newline> という文字列は無視され、 \, $, ‘ といった文字は \ を用いてクォートしなければなりません。

リダイレクト演算子が <<- ならば、行頭にあるタブ文字は全て入力行および delimiter を含む行から取り除かれます。 これにより、シェルスクリプト中のヒアドキュメントを 自然な形でインデントさせることができます。

1.15.8　ヒアストリング (Here Strings)

ヒアドキュメントの変形で、以下の形式です:

<<<word

word は展開されてコマンドの標準入力に与えられます。

1.15.9　ファイル・ディスクリプターの複製

リダイレクト演算子

[n]<&word

を使うと入力ファイル・ディスクリプターを複製できます。 word が 1 桁以上の数値に展開された場合、n で示されるファイル・ディスクリプターが生成され、 word で指定された数値のファイル・ディスクリプターのコピーとなります。 word に含まれる数値が入力用にオープンされたファイル・ディスクリプターを指していない場合、リダイレクト・エラーが起きます。 word を評価した結果が - となった場合、ファイル・ディスクリプター n はクローズされます。 n が指定されていない場合、標準入力 (ファイル・ディスクリプター 0) が使われます。

同様に、演算子

[n]>&word

を使って出力ファイル・ディスクリプターを複製できます。 n が指定されていない場合は、標準出力 (ファイル・ディスクリプター 1) が使われます。 word に含まれる数値が、出力用にオープンされた ファイル・ディスクリプターを指していない場合、 リダイレクト・エラーが起きます。特別な場合ですが、 n が省略され、かつ word が 1 桁以上の数字には展開されなかった場合、 前に説明したように標準出力と標準エラー出力がリダイレクトされます。

1.15.10　ファイル・ディスクリプターの変更

リダイレクト演算子

[n]<&digit-

を使うと、ファイル・ディスクリプターの digit を n に変更します。 n が指定されていない場合は、標準入力 (ファイル・ディスクリプター 0) が使われます。 digit は n に複製された後にクローズされます。

同様に、演算子

[n]>&digit-

を使うと、ファイル・ディスクリプターの digit を n に変更します。 n が指定されていない場合は、標準出力 (ファイル・ディスクリプター 1) が使われます。

1.15.11　読み書きのためのファイル・ディスクリプターのオープン

リダイレクト演算子

[n]<>word

を使うと、 word を展開した結果の名前を持つファイルがファイル・ディスクリプター n での読み書きのためにオープンされます。 n が指定されていなければ、 ファイル・ディスクリプター 0 で読み書きが行われます。 ファイルが存在しなければ、新しく生成されます。

1.16　エイリアス

エイリアス (alias) を使うと、 ある単語が単純なコマンドの先頭の単語として使われた場合に、 この文字列を別の単語に置換できます。 シェルはエイリアスのリストを管理しています。 このリストは組み込みコマンドの alias と unalias を使って設定および削除できます (後述の シェルの組み込みコマンドを参照)。 各コマンドの最初の単語がクォートされていない場合、 エイリアスかどうかが確認され、 エイリアスならばその単語はそのエイリアスのテキストと置換されます。 /、$、‘、=、シェルの メタ文字 (metacharacters) やクォート文字はエイリアス名には使えません。 エイリアス名と置換されるテキストには、 シェルの入力として有効なものは何でも含めることができます。これには先に挙げた メタ文字 も含まれます。 置換されたテキストの最初の単語に対しても エイリアスかどうかの評価がされますが、 最初の単語が展開されるエイリアスと同じ場合には 展開は一度しか行われません。つまり ls が ls -F のエイリアスとなっているような場合には、 bash は置換される文字列を再帰的に展開することはありません。 エイリアスの値の最後の文字が ブランク文字 の場合、エイリアスに続く次のコマンドの単語に対しても エイリアス展開が試みられます。

エイリアスは alias コマンドで作成とリスト表示を行い、 unalias コマンドで削除します。

置換されるテキストに引き数を入れる仕組みはありません。 引き数が必要ならば、シェル関数を使わなければなりません (後述の 関数 を参照)

シェルが対話的でないときには、 shopt コマンドによって expand_aliases オプションが設定されていない場合に限り、 エイリアスの展開は行われません (後述の シェルの組み込みコマンド における shopt の説明を参照)。

エイリアスの定義や利用に関係する規則には、紛らわしい点があります。 bash は、ある行に書かれているコマンドを実行する前に必ず、 少なくとも 1 回は行全体を読み込みます。 エイリアスが展開されるのはコマンドを読み込んだときであり、 実行するときではありません。 したがって、別のコマンドと同じ行でエイリアス定義を行った場合には、 次の入力行が読み込まれるまではエイリアスの効果は現われません。同じ行にあるエイリアス定義の後のコマンドは、 新しいエイリアスの影響を受けません。 この動作は関数を実行する場合にも問題になります。 エイリアスが展開されるのは関数定義が読み込まれるときであり、関数が実行されるときではありません。 なぜなら、関数定義自身も複合コマンドだからです。 その結果として、関数内で定義されたエイリアスは、 その関数が実行されるまでは利用できません。 安全のため、エイリアス定義は独立した行で行うべきです。 複合コマンド内で alias を使ってはいけません。

ほとんど全ての用途において、 シェル関数でエイリアスを代用できます。

1.17　関数

シェル関数は、後で使うために一連のコマンドを保存するものです。 シェル関数の定義は既に シェルの文法 で説明しています。 シェル関数名が単純なコマンド名として使われた場合、 関数名に対応するコマンド群が実行されます。関数は現在のシェルのコンテキスト内で実行されます。 つまり、新しいプロセスを生成して関数を処理することはありません (これはシェルスクリプトと対照的な点です)。 関数の実行時には、関数に与えた引き数が位置パラメータとなります。 特殊パラメータ # は更新され、この変更が反映されます。 特殊パラメータ 0 は変わりません。 関数の実行中は FUNCNAME 変数の最初の要素に関数の名前が設定されます。

上記以外は、シェル実行環境の状態は全て、関数とその呼び出し側で同じになります。 ただし、以下の例外があります: DEBUG と RETURN のトラップ (後述の シェルの組み込みコマンド の項で、組み込みコマンド trap の説明を参照) は、 関数に trace 属性 (後述の組み込みコマンド declare の説明を参照) が与えられている場合や、-o functrace シェルオプションが 組み込みコマンド set によって有効になっている (つまり、全ての関数が DEBUG と RETURN のトラップを継承している) 場合を除いて 継承されません。 ERR トラップは、-o errtrace シェルオプションが有効になっていない限り 継承されません。

関数ローカルの変数は、組み込みコマンド local で宣言できます。 普通は、変数とその値は関数とその呼び出し側で共有されます。

FUNCNEST 変数に 0 より大きい数値をセットすると、 関数呼び出しを何重まで許すかの最大レベルを決めます。 このレベルを超えて関数を呼び出すと、コマンドが異常終了します。

組み込みコマンド return が関数中で実行された場合、関数は完了し、 関数呼び出しの次のコマンドから実行が再開されます。 再開される前に、RETURN トラップに設定されたコマンドが実行されます。 関数の完了時には、位置パラメータの値と特殊パラメータ # の値は、関数の実行前の値に戻ります。

関数の名前と定義をリスト表示するには、組み込みコマンドの declare や typeset を、オプション -f を付けて実行します。 declare や typeset をオプション -F で実行すると、関数名だけがリスト表示されます。また、extdebug シェルオプションが有効になっていると、 ソースファイルと行番号も表示されます。 関数をエクスポートして、 サブシェルでその関数が自動的に定義されている状態にできます。 これを行うには、組み込みコマンドの export に -f オプションを付けて実行します。 組み込みコマンドの unset に-f オプションを付けて実行することで、 関数定義を削除することもできます。 同じ名前のシェル関数と変数がシェルの子プロセスにエクスポートされると、 同じ環境の中に全く同じ名前のエントリが複数できてしまうことに 注意してください。これが問題を起こす場合には注意が必要です。

関数は再帰させることができます。FUNCNEST 変数を使うと、 関数の呼び出しスタックの深さを制限し、関数の呼び出し回数を制限する ことができます。デフォルトでは再帰呼び出し回数に制限は課せられていません。

1.18　算術式評価

シェルにおいては、特定の状況下で算術式を評価させることができます (組み込みコマンドの let と 算術式展開 を参照)。 評価は固定長の整数として行われ、 オーバーフローのチェックは行われません。 ただし、0 での除算はトラップされ、エラーとしてのフラグが立てられます。 演算子とその優先度、結合規則は C 言語と同じです。 以下のリストは、 同じ優先度を持つ演算子をグループとしてまとめて列挙したものです。 優先度の高いものから順に列挙しています。

`id++ id--`
: 変数を評価し、その後 increment（加算)/ decrement（減算）する。

`++id --id`
: 変数を increment（加算）/ decrement（減算）してから評価する。

`- +`
: 単項式の負と正

`! ~`
: 論理的否定とビット単位の否定

`**`
: 指数（累乗）

`* / %`
: 乗算、除算、剰余

`+ -`
: 加算と減算

`<< >>`
: 左ビットシフトと右ビットシフト

`<= >= < >`
: 比較

`== !=`
: 等値と非等値

`&`
: ビット単位の AND

`^`
: ビット単位の排他的 OR

`|`
: ビット単位の OR

`&&`
: 論理的 AND

`||`
: 論理的 OR

`expr?expr:expr`
: 条件付き実行

`= *= /= %= += -= <<= >>= &= ^= |=`
: 代入

`expr1 , expr2`

コンマ

シェル変数をオペランドにすることもできます。 パラメータ展開は式の評価より前に行われます。 式の中では、シェル変数を (パラメータ展開規則を用いずに) 変数名で参照できます。 値が空のシェル変数や宣言されていないシェル変数は、 パラメータの展開の文法を使わずに名前を参照されると、 0 として評価されます。 変数の値は、変数が参照されたときや、declare -i によって 整数 属性が設定された変数に値が代入されるときに、 算術式として評価されます。 空の値は 0 として評価されます。 式で用いるためにシェル変数の 整数属性 を有効にする必要はありません。

先頭が 0 である定数は 8 進数として解釈されます。 先頭が 0x または 0X ならば 16 進数として解釈されます。 それ以外の場合には、数値は [base#]n の形式で表します。ここで base は 2 から 64 の間の 10 進数であり、算術的な意味での基数を表します。 n はその基数における数を表します。 base# が省略されると 10 進数となります。 ある桁において 9 より大きい数字を表すには文字を使います。 文字を使う場合には、アルファベット小文字、大文字、@, _ をこの順番で使います。 base が 36 以下の場合には、大文字と小文字は区別されず、 大文字と小文字のどちらを使っても 10 から 35 までの数字を表現できます。

演算子は優先度の順に評価されます。括弧内にある部分式は最初に評価され、前述の優先規則よりも優先させることができます。

1.19　条件式

条件式は複合コマンド [[と組み込みコマンドの test および [によって使用でき、ファイルの属性を調べたり、 文字列比較や算術式比較を行ったりできます。 式は以下に示す単項のプライマリや二値のプライマリから構成されます。プライマリのいずれかの file 引き数が /dev/fd/n という形式ならば、 ファイル・ディスクリプター n が調べられます。 プライマリのいずれかの file 引き数が /dev/stdin、/dev/stdout、/dev/stderr のいずれかであれば、 対応するファイル・ディスクリプター 0、1、2 が調べられます。

ほかに指定されていなければ、ファイルに対するプライマリはシンボリックリンク を辿り、リンク自身ではなくリンク先を対象とします。

[[では < と > の演算子は現在のロケールでの辞書順で 比較します。test コマンドは ASCII 順で比較します。

1.19.1　-a file

file が存在すれば真となります。

1.19.2　-b file

file が存在し、かつブロック特殊ファイルならば真となります。

1.19.3　-c file

file が存在し、かつキャラクター特殊ファイルならば真となります。

1.19.4　-d file

file が存在し、かつディレクトリならば真となります。

1.19.5　-e file

file が存在すれば真となります。

1.19.6　-f file

file が存在し、かつ通常ファイルならば真となります。

1.19.7　-g file

file が存在し、かつ set-group-id されていれば真となります。

1.19.8　-h file

file が存在し、かつシンボリックリンクならば真となります。

1.19.9　-k file

file が存在し、かつ “ sticky ” ビットが設定されていれば真となります。

1.19.10　-p file

file が存在し、かつ名前付きパイプ (FIFO) ならば真となります。

1.19.11　-r file

file が存在し、かつ読み込み可能ならば真となります。

1.19.12　-s file

file が存在し、かつそのサイズが 0 より大きければ真となります。

1.19.13　-t fd

ファイル・ディスクリプター fd がオープンされており、かつ端末を参照していれば真となります。

1.19.14　-u file

file が存在し、 かつ set-user-id ビットが設定されていれば真となります。

1.19.15　-w file

file が存在し、かつ書き込み可能ならば真となります。

1.19.16　-x file

file が存在し、かつ実行可能ならば真となります。

1.19.17　-G file

file が存在し、かつ (実行中のシェルの) 実効グループ ID に所有されていれば真となります。

1.19.18　-L file

file が存在し、かつシンボリックリンクならば真となります。

1.19.19　-N file

file が存在し、 かつ前回読み込まれた以降に修正されていれば真となります。

1.19.20　-O file

file が存在し、かつ (実行中のシェルの) 実効ユーザ ID に所有されていれば真となります。

1.19.21　-S file

file が存在し、かつソケットならば真となります。

1.19.22　file1 -ef file2

file1 と file2 とで デバイス番号と i-ノード番号が同じならば真となります。

1.19.23　file1 -nt file2

file1 が (変更日時に関して) file2 より新しいか、 file1 が存在するが file2 が存在しなければ、真となります。

1.19.24　file1 -ot file2

file1 が file2 より古いか、 file2 が存在するのに file1 が存在しなければ、真となります。

1.19.25　-o optname

シェルオプション optname が有効ならば真となります。後述する組み込みコマンド set に対するオプションの説明中にある -o オプションを参照してください。

1.19.26　-v varname

シェル変数 varname がセットされている (値が代入されている) ならば真となります。

1.19.27　-z string

string の長さが 0 ならば真となります。

1.19.28　string

1.19.29　-n string

string の長さが 0 でなければ真となります。

1.19.30　string1 == string2

1.19.31　string1 = string2

文字列が同じならば真となります。 POSIX に準拠する形で test コマンドを使う場合には = を使う必要があります。

1.19.32　string1 != string2

2 つの文字列が異なれば真となります。

1.19.33　string1 < string2

現在のロケールにおいて、string1 が string2 よりも 辞書順で前にある場合に真となります。

1.19.34　string1 > string2

現在のロケールにおいて、string1 が string2 よりも 辞書順で後にある場合に真となります。

1.19.35　arg1 OP arg2

OP は -eq, -ne, -lt, -le, -gt, -ge. のいずれかです。 これらの算術二値演算子が真を返すのはそれぞれ、arg1 が arg2 に対して等しい場合、 等しくない場合、小さい場合、小さいか等しい場合、大きい場合、 大きいか等しい場合です。 arg1 や arg2 には、正または負の整数を使用できます。

1.20　単純なコマンドの展開

単純なコマンドを実行すると、シェルは以下に示す展開、代入、 リダイレクションを左から右の順で実行します。

1. パーザが変数代入 (コマンド名の前にあるもの) またはリダイレクションと判断した単語は、 保存されて後で処理されます。

2. 変数代入でもリダイレクションでもない単語が展開されます。 もし展開の後に残っている単語があれば、 その最初の単語がコマンド名となり、残りが引き数となります。

3. リダイレクションが前述の リダイレクト で説明したように実行されます。

4. 変数代入の = の後にあるテキストに対して、チルダ展開、 パラメータ展開、コマンド置換、算術式展開、クォート削除が行われます。 この処理は変数を代入する前に行われます。

コマンド名が残らなかった場合には、 変数を代入した結果が現在のシェル環境に効果を及ぼします。 それ以外の場合、変数は実行されるコマンドの環境に追加されるだけで、 現在のシェル環境には影響を与えません。 読み込み専用の変数に対して代入をしようとするとエラーが発生し、 そのコマンドは 0 でないステータスで終了します。

コマンド名が残らなかった場合、リダイレクションは行われますが、 現在のシェル環境は影響を受けません。 リダイレクションのエラーが起きると、 コマンドは 0 でないステータスで終了します。

展開の後にコマンド名が残っている場合、後述するように実行が進行します。そうでない場合はコマンドは終了します。 展開のいずれかがコマンド置換である場合には、 コマンドの終了ステータスは 最後に実行されたコマンド置換の終了ステータスになります。 コマンド置換が行われなかった場合には、 コマンドはステータス 0 で終了します。

1.21　コマンドの実行

コマンドが単語に分割された後に、 単純なコマンドとそれに対する引き数リスト (引き数リストは省略可能) となった場合、以下の動作が行われます。

コマンド名にスラッシュが含まれない場合、 シェルはコマンドの位置を特定しようとします。 その名前のシェル関数が存在すれば、前に 関数 で説明したようにその関数が呼び出されます。 名前が関数にマッチしない場合には、 シェルはシェルの組み込みコマンドのリストを探します。 マッチするものがあった場合、その組み込みコマンドが呼び出されます。

名前がシェル関数も組み込みコマンドでなく、 かつスラッシュを含まない場合には、 bash は PATH の各要素を検索し、その名前の実行ファイルを含むディレクトリを探します。 bash はハッシュ表を使って実行ファイルの完全なパス名を記憶します (後述の シェルの組み込みコマンド の hash の項を参照)。PATH に含まれるディレクトリの完全な探索は、 そのコマンドがハッシュ表の中にない場合にのみ行われます。 探索に失敗すると、シェルは command_not_found_handle という名前のシェル関数の定義を探します。 存

在する場合には、元のコマンドと元のコマンドの引き数を引き数として、 この関数を呼び出します。この関数の終了ステータスがシェルの 終了ステータスとなります。 この関数が定義されていない場合には、シェルはエラーメッセージを表示して終了ステータス 127 を返します。

検索に成功したか、コマンド名に 1 つ以上のスラッシュが含まれる場合には、シェルは指定されたプログラムを独立した実行環境で実行します。 引き数 0 には指定された名前が設定され、 コマンドに対する残りの引き数には (もしあれば) 指定された引き数が設定されます。

このファイルが実行可能フォーマットでないために実行が失敗し、 かつディレクトリでもない場合には、このファイルは シェルスクリプト (shell script) であるとみなされます。 シェルスクリプトとは、シェルのコマンドが書かれているファイルのことです。 シェルスクリプトを実行するためにサブシェルが呼び出されます。 このサブシェルは自分自身を再初期化し、 シェルスクリプトを処理するために 新しいシェルが起動されたかのような結果になります。ただし、 親が記憶しているコマンドの位置 (後述の シェルの組み込みコマンド における hash の項を参照) は子にも引き継がれます。

プログラムが #! で始まるファイルである場合、 最初の行の残りの部分はこのプログラムのインタプリタを指定します。 シェルは指定されたインタプリタをオペレーティングシステム上で実行します。 オペレーティングシステムは、この実行可能フォーマットを直接処理しません。 インタプリタに対する引き数は、プログラムの先頭の行のインタプリタ名の後の省略可能な引き数 1 つと、 その後のプログラム名、さらに (もしあれば) その後のコマンドへの引き数から構成されます。

1.22　コマンド実行環境

シェルは以下の要素からなる実行環境 (execution environment) を持ちます:

起動時にそのシェルが継承したオープンされているファイル。 これは組み込みコマンド exec に与えられている リダイレクション機能で変更されます。

現在の作業ディレクトリ (current working directory)。これは cd, pushd, popd で設定するか、 あるいは起動時にそのシェルが継承します。

ファイル作成モードのマスク。 これは umask で設定するか、あるいはそのシェルの親から継承します。

trap で設定された現在のトラップ。

シェルのパラメータ。これは変数の代入か set で設定されるか、 あるいはその環境内にある親シェルから継承します。

実行中に定義されるか、その環境内にある親シェルから継承したシェル関数。

起動時に有効にしたか (デフォルト値あるいはコマンドライン引き数で設定した値)、 set で有効にしたオプション。

shopt で有効にしたオプション。

alias で定義したシェルエイリアス。

各種プロセス ID。これにはバックグラウンドジョブや $$ の値、 PPID の値が含まれます。

組み込みコマンドやシェル関数以外の単純なコマンドを実行するとき、 このコマンドは独立した実行環境内で呼び出されます。 この環境は以下の要素から成り立っています。 特に断らない限り、この値はシェルから引き継がれます。

シェルのオープンしているファイルと、 コマンドに対するリダイレクションで指定した変更・追加を加えたもの。

現在の作業ディレクトリ

ファイル作成モードのマスク

エクスポートするシェル変数や関数と、 そのコマンドに対してエクスポートされた変数。 これらは環境で渡されます。

シェルに捕捉されるトラップは、 そのシェルの親から継承された値に再設定されます。 そのシェルにより無視されるトラップは無視されます。

この独立の環境内で呼び出されたコマンドが、 (親である) シェルの実行環境に影響を及ぼすことはできません。

コマンド置換や括弧でグループ化されたコマンド、非同期コマンドは、 サブシェル環境内で呼び出されます。 このサブシェル環境はシェル環境を複製したものです。 ただし、シェルにより捕捉されるトラップは、 そのシェルの起動時に親から継承した値に再設定されます。 パイプラインの一部として起動された組み込みコマンドは、 サブシェル環境で実行されます。サブシェル環境に対して行われた変更は、 元のシェル実行環境に影響を及ぼすことはできません。

コマンド置換を実行するために起動されたサブシェルは、 -e オプションの値を親シェルから継承します。posix モードでない場合、bash はそのサブシェルでは -e オプションをクリアします。

コマンドの後に & が付けられたときに、ジョブ制御が有効でなければ、 コマンドのデフォルトの標準入力は空ファイル /dev/null となります。 そうでなければ、呼び出されたコマンドは、呼び出したシェルのファイル・ディスクリプターを、リダイレクトも含めて継承します。

1.23　環境

プログラムの起動時には、 環境 (environment) と呼ばれる文字列の配列が渡されます。 これは 変数名-値 のペアからなるリストで、 変数名=値 という形になります。

シェルは、環境を操作する様々な方法を提供しています。 起動時には、シェルは自分自身の環境を調べ、 見つかった名前それぞれに対してパラメータを生成し、 それに自動的に子プロセスへの エクスポート (export) の印を付けます。実行されたコマンドは環境を継承します。 export コマンドまたは declare -x コマンドを用いると、 パラメータや関数の追加と削除を環境に対して行えます。 環境内のパラメータの値が変更されると、 新しい値は環境の一部となり、古い値と置き換わります。 実行されたコマンドが継承する環境は、 シェルの最初の環境から、 変数の値がシェル中で変更されたり、 一部のペアが unset コマンドで削除されたり、 export コマンドや declare -x コマンドで追加されたりしたものになります。

単純なコマンド や関数に対する環境は、一時的に修正できます。 これは、既に パラメータ の項で説明したように、パラメータ代入を前に置くことで行います。 このような代入が影響を与えるのは、 そのコマンドが参照する環境だけです。

-k オプションを設定 (後述の組み込みコマンド set を参照) すると、コマンド名の前に置いたものだけではなく、 全ての パラメータ代入がそのコマンドの環境に影響を与えます。

bash が外部コマンドを起動したときには、変数 _ にはコマンドの完全なファイル名が設定され、 環境変数としてそのコマンドに渡されます。

1.24　終了ステータス

実行したコマンドの終了ステータスは、waitpid システムコールまたはそれに相当する関数が返した値です。 終了ステータスは 0 から 255 の値を取りますが、後述するように、 125 より大きい値は特別にシェルによって使われることがあります。 シェルの組み込みコマンドや複合コマンドの終了ステータスも、同じ範囲に限定されています。 環境によっては、シェルは仕様で決められた失敗のモードを表す 特別の値を使います。

シェルは、終了コード 0 で終了したコマンドは正常終了したとみなします。終了コード 0 は成功を示します。 0 以外の終了コードは失敗を示します。 あるコマンドが致命的なシグナル N で終了したときには、bash は「128+N」の値を終了ステータスに使います。

コマンドが見つからなかった場合には、 そのコマンドを実行するために生成された子プロセスが ステータス 127 を返します。 コマンドが見つかったけれど実行できなかった場合には、 ステータスは 126 です。

展開やリダイレクションの際にエラーが発生し コマンドが失敗した場合には、0 より大きい終了ステータスが返されます。

シェルの組み込みコマンドは、 成功した場合にはステータス 0 (真) を返し、実行中にエラーが起こった場合には 0 でない値 (偽) を返します。 組み込みコマンドは全て、正しくない使い方であることを示すのに 終了ステータス 2 を返します。

bash 自身が返す終了ステータスは、文法エラーが起きた場合を除き、 実行した最後のコマンドの終了ステータスです。 文法エラーの場合には 0 でない値が終了ステータスとなります。 後述の組み込みコマンド

exit も参照してください。

1.25　シグナル

bash が対話的であり、トラップが全くないとき、bash は SIGTERM を無視し (したがって kill 0 では対話シェルは kill されません)、 SIGINT を捕捉し処理します (したがって組み込みコマンド wait は割り込み可能です)。 どんな場合でも、bash は SIGQUIT を無視します。 ジョブ制御が有効な状態ならば、 bash は SIGTTIN, SIGTTOU, SIGTSTP を無視します。

bash が起動した外部コマンドは、 シェルが自分の親から継承した値をシグナルハンドラに設定します。ジョブ制御が有効でないときには、非同期コマンドは、 継承したシグナルハンドラに加えて SIGINT と SIGQUIT も無視します。コマンド置換の結果として実行されたコマンドは、 キーボードで生成されたジョブ制御シグナルを無視します。 無視されるシグナルは SIGTTIN, SIGTTOU, SIGTSTP です。

デフォルトでは、シェルは SIGHUP を受け取ると終了します。 終了する前には、シェルは実行中・停止中の全てのジョブに対して SIGHUP を再送信します。停止中のジョブには SIGCONT が送られ、このジョブが SIGHUP を受け取るようにします。 特定のジョブに対してシェルからシグナルが送られないようにするためには、 組み込みコマンド disown (後述の シェルの組み込みコマンドを参照) を使って、そのジョブをジョブテーブルから削除するか、あるいは disown -h を使って、 SIGHUP を受け取らないようにマークを付けます。

シェルオプションの huponexit が shopt を使って設定されていた場合、対話的なログインシェルが終了するときに、 bash は SIGHUP を全てのジョブに送ります。

コマンドの完了を待っている間に、トラップが設定されたシグナルを bash が受け取ったとき、 そのトラップはコマンドが完了するまで実行されません。bash が組み込みコマンドの wait を使って非同期コマンドを待っているときに、 トラップが設定されているシグナルを受け取ると、 組み込みコマンド wait は即座に復帰させられます。 この際の終了ステータスは 128 より大きい値になります。 また復帰するのはトラップが実行された直後です。

1.26　ジョブ制御

ジョブ制御 (job control) とは、プロセスの実行を選択的に停止 (サスペンド/suspend) させ、 後で実行を再開させる (リジューム/resume) 機能のことです。 ユーザは通常、対話的インタフェースを通してこの機能を利用します。対話的インタフェースは、オペレーティングシステムのカーネルの端末ドライバと bash の組み合わせで提供されます。

シェルはパイプラインごとに ジョブ (job) を構成します。シェルは現在実行中のジョブのテーブルを保存しています。 このテーブルは jobs コマンドを使ってリスト表示できます。 bash がジョブを非同期的に (バックグラウンド で) 起動したときには、 bash は以下のような行を出力します:

[1] 25647

これは、このジョブのジョブ番号は 1 であり、 このジョブを構成するパイプライン中の最後のプロセスの ID が 25647 であることを示しています。 1 つのパイプラインに含まれる全てのプロセスは同じジョブのメンバです。 bash は、ジョブ制御の基礎として ジョブ という抽象化機構を使います。

ジョブ制御のためのユーザインタフェースの実装を容易にするために オペレーティングシステムは 現在の端末プロセスのグループ ID (current terminal process group ID) という情報を管理しています。 このプロセスグループのメンバ (プロセスグループ ID が現在の端末プロセスのグループ ID と等しいプロセス) は、 SIGINT のような、キーボードで生成されたシグナルを受け取ります。 このようなプロセスはフォアグラウンド (foreground) にあると言われます。 バックグラウンド (background) プロセスは、 プロセスのグループ ID が端末のグループ ID と異なるプロセスです。 このようなプロセスは、キーボードで生成したシグナルの影響を受けません。 フォアグラウンドプロセスだけが端末からの読み込みが許され、stty tostop で許可されている場合には出力も許されます。 バックグラウンドプロセスが端末からの読み込み (stty tostop が有効なときには端末への書き込みも) を行おうとすると、 このプロセスには端末ドライバから SIGTTIN (SIGTTOU) シグナルが送られます。このシグナルを捕捉しなければ、プロセスは停止します。

bash が動作しているオペレーティングシステムがジョブ制御をサポートしているならば、 ユーザは bash を使ってジョブ制御を行えます。 プロセスの動作中に サスペンド 文字 (通常は ^Z, Control-Z) を打ち込むと、そのプロセスは停止させられ、 bash に制御が戻ります。 遅延サスペンド (delayed suspend) 文字 (通常は ^Y, Control-Y) を打ち込むと、 そのプロセスは端末から入力を読み込もうとしたときに停止させられ、 制御が bash に戻ります。この時点でユーザはこのジョブの状態を操作できます。 利用できるのは、バックグラウンドで実行を継続するならば bg コマンド、フォアグラウンドで実行を継続するならば fg コマンド、プロセスを kill するなら kill コマンドです。^Z の効果は即座に現われるので、 未出力の出力や先行入力した文字が破棄されるという副作用があります。

シェル上でジョブを参照する方法はいろいろあります。 文字 % はジョブ名 (jobspec) の始まりを示します。 ジョブ番号 n は %n として参照できます。ジョブの参照には、 ジョブを起動するときに使った名前の先頭部分や コマンドライン中に現われる部分文字列を使うこともできます。 例えば、 %ce は停止中のジョブ ce を指します。 先頭の部分がマッチするジョブが複数個ある場合には、 bash はエラーを報告します。一方、 %?ce を用いると、文字列 ce をコマンドライン中に含む任意のジョブを参照できます。 部分文字列がマッチするジョブが複数個ある場合には、 bash はエラーを報告します。 シンボル%% および %+ は、シェルが覚えている カレントジョブ (current job) を指します。これは、フォアグラウンドで起動されているときに停止されたか、 バックグラウンドで起動された最後のジョブです。 前のジョブ (previous job) は%- を使って参照できます。 ジョブが一つしかない場合には、%+ と %- のどちらも使っても そのジョブを参照することができます。 ジョブに関する出力 (jobs コマンドの出力など) では、カレントジョブには必ず + というフラグが付き、前のジョブには - というフラグが付きます。 ジョブ名が付いていない単独の % もカレントジョブを示します。

ジョブの名前だけを指定すると、 そのジョブをフォアグラウンドに持ってきます。つまり、 %1 は “ fg %1 ” と同義であり、 ジョブ 1 をバックグラウンドからフォアグラウンドに持ってきます。 同様に “ %1 & ” はジョブ 1 をバックグラウンドで実行再開させます。これは “ bg %1 ” と同じ意味です。

ジョブの状態が変わると、シェルはそれを即座に知ります。 通常、 bash がジョブの状態変化を報告するのは、プロンプトを出力する直前です。 これは他の出力を妨害しないためです。 組み込みコマンド set

で -b オプションが指定されていると、 bash はジョブの状態の変化を即座に報告します。 子プロセスが終了するたびに SIGCHLD のトラップが実行されます。

ジョブを停止させたままで (checkjobs シェルオプションが 組み込みコマンド shopt により有効になっているときには実行中でも) bash を終了 (exit) させようとすると、シェルは警告メッセージを出力します。 checkjobs オプションが有効のときには、ジョブとその状態を一覧表示します。 このようなときには、 jobs コマンドを使ってジョブの状態を調べられます。 間にコマンドを挟まずに bash を再び終了させようとすると、 シェルは警告を繰り返さないで、停止中のジョブを終了させます。

1.27　プロンプト

対話的に動作している場合、 bash はコマンドを読む込み準備ができたときにプライマリプロンプト PS1 を表示し、コマンドを完成させるためにまだ入力が必要なときに セカンダリプロンプト PS2 を表示します。 bash ではこれらのプロンプト文字列をカスタマイズできます。 この際にはバックスラッシュでエスケープされた 各種特殊文字を挿入でき、 これは以下のようにデコードされます:

\a　ASCII のベル文字 (07)

\d　"曜日 月 日" という形式の日付 (例: "Tue May 26")

\D{format}

format が strftime(3) に渡され、 その結果がプロンプト文字列に挿入されます。 format が空の場合には (ブレースは必要)、 ロケールで指定された時刻表記になります。

\e　ASCII のエスケープ文字 (033)

\h　ホスト名のうち最初の '.' までの部分

\H　ホスト名

\j　シェルによって現在管理されているジョブの数

\l　シェルの端末デバイスのベース名 (basename)

\n　改行 (newline)

\r　復帰 (carriage return)

\s　シェルの名前。つまり $0 のベース名 (最後のスラッシュ以降の部分)

\t　24 時間制の HH:MM:SS 形式の現在の時刻

\T　12 時間制の HH:MM:SS 形式の現在の時刻

\@　12 時間制の am/pm 形式の現在の時刻

\A　12 時間制の HH:MM 形式の現在の時刻

\u　現在のユーザのユーザ名

\v　bash のバージョン (例: 2.00)

\V　bash のリリース。バージョンにパッチレベルを加えたもの (例 : 2.00.0)

\w　現在の作業ディレクトリ。 $HOME の部分はチルダに短縮されます。 PROMPT_DIRTRIM の値が適用されます。

\W　現在の作業ディレクトリのベース名 $HOME の部分はチルダに短縮されます。

\!　このコマンドの履歴番号

\#　このコマンドのコマンド番号
\$　実効 UID が 0 の場合に #、 それ以外の場合は $
\nnn　8 進数 nnn に対応する文字
\\　バックスラッシュ
\[　非表示文字のシーケンスの開始。 これを使うと、プロンプト中に端末の制御シーケンスを埋め込むことができます。
\]　非表示文字のシーケンスを終了します。

コマンド番号と履歴番号は異なるのが普通です: コマンドの履歴番号とは履歴リスト内での位置です。 履歴リストは履歴ファイルから読み込めます (後述の履歴 を参照)。 一方、コマンド番号は、 現在のシェルのセッション中に実行された一連のコマンドの列における位置です。 この文字列がデコードされた後、さらにパラメータ展開、コマンド置換、 算術式展開、クォート削除が適用されます。展開はシェルオプション promptvars (後述の シェルの組み込みコマンド の項にある shopt コマンドの説明を参照) の値に基づいて行われます。

1.28　READLINE ライブラリ

readline は対話シェルを使うときに入力の読み込みを処理するライブラリです。 ただし、シェルの起動時に --noediting オプションが指定された場合には使われません。 行編集は組み込みコマンド read に -e オプションを 指定したときにも使われます。 デフォルトでは、行編集に使うコマンドは emacs のコマンドに似ています。 vi 形式の行編集インタフェースも使えます。 行編集は、組み込みコマンド set (後述のシェルの組み込みコマンド を参照) に-o emacs や -o vi オプションを指定することで、いつでも有効にできます。シェルを起動した後に行編集機能を無効にするには、組み込みコマンド set に対して +o emacs オプションまたは +o vi オプションを設定してください。

1.28.1　Readline の記法

このセクションでは、emacs 形式の記法を使ってキーストロークを表します。コントロールキーは C-key で表します (例: C-n は Control-N の意味です)。同様に メタ キーは M-key で表すので、M-x は Meta-X を表すことになります。 (メタ キーがないキーボードでは、M-x は ESC x を表します。つまり、 エスケープキーを押した後に x キーを押します。これは ESC を メタプレフィックス (meta prefix) にする操作です。 M-C-x の組み合わせは、ESC-Control-x つまり、エスケープキーを押した後に、コントロールキーを押したまま x を押すことを意味します。)

Readline のコマンドには数値の 引き数 を指定できます。通常これは繰り返しの回数として作用します。ただし、場合によっては、引き数の符号が意味を持つこともあります。 順方向に作用するコマンド (例: kill-line) に負の引き数を渡すと、 コマンドは逆方向に作用します。 引き数を指定した場合の動作がこの説明と異なるコマンドについては後で説明します。

コマンドがテキストをキル (kill) すると説明されているときは、 削除されたテキストは、 後で取り出せる (ヤンク (yank) できる) ように保存されます。キルされたテキストは、キルリング (kill ring) に保存されます。 連続してキルを行うと、テキストは 1 つのまとまりとして保存されるので、 全部を一度にヤンクできます。テキストをキルしないコマンドが挟まると、 キルリング上のテキストが分離されます。

1.28.2　Readline の初期化

readline をカスタマイズするには、 コマンドを初期化ファイル (inputrc ファイル) に追加します。 このファイルの名前は、変数 INPUTRC の値から決まります。 この変数が設定されていない場合のデフォルト値は ~/.inputrc です。 readline ライブラリを使うプログラムが起動する際には、 この初期化ファイルが読み込まれ、キー割り当てと変数が設定されます。 readline の初期化ファイル中で使用できる基本的構文は、以下のように少ししかありません。 空行は無視されます。 # で始まる行はコメントです。 $ で始まる行は条件文です。 それ以外の行はキー割り当てと変数の設定です。

デフォルトのキー割り当ては inputrc ファイルで変更できます。 このライブラリを使う他のプログラムからも、 独自のコマンドとキー割り当てを追加できます。

例えば、

```
M-Control-u: universal-argument
```

または

```
C-Meta-u: universal-argument
```

を inputrc ファイルに書くと、M-C-u で readline の universal-argument コマンドが実行されるようになります。

以下に示すシンボル名を使うことができます: RUBOUT, DEL, ESC, LFD, NEWLINE, RET, RETURN, SPC, SPACE, TAB。

readline では、キーにはコマンド名だけでなく文字列を割り当てることもでき、 その場合には、そのキーが押されると割り当てられた文字列が挿入されます (マクロ)。

1.28.3　Readline のキー割り当て

inputrc ファイルにおける制御キーの割り当て方は単純です。 必要なものは、コマンド名あるいはマクロの文字列と、 これらが割り当てられるキーシーケンスだけです。 名前は 2 通りの方法で指定できます。つまり、シンボリックなキーの名前 (たいていは、Meta- や Control- プレフィックスと組み合わせて使われる) による指定と、 キーシーケンスによる指定です。

「keyname:function-name」あるいは 「keyname:macro」の形式を使うときには、 keyname は英語で書き下したキーの名前となります。例を以下に示します:

```
Control-u: universal-argument
Meta-Rubout: backward-kill-word
Control-o: "> output"
```

この例では、 C-u が universal-argument に、 M-DEL が backward-kill-word に割り当てられます。 また、C-o はマクロの実行に割り当てられ、右辺値に展開されます (つまり、テキスト “> output” が編集行に挿入されます)。

後者の形式である「"keyseq":function-name」または 「"keyseq":macro」においては、 keyseq は先程の keyname とは異なり、 ダブルクォートで括ってキーシーケンス全体を示す文字列を表記しています。 以下の例で示すように、GNU Emacs 形式のキーエスケープの一部を使えます。 ただしシンボリックな文字名は認識されません。

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "Function Key 1"
```

この例でも、 C-u が universal-argument 機能に割り当てられています。 C-x C-r は re-read-init-file 機能に割り当てられ、 ESC [1 1 ~ は “Function Key 1” という文字列の挿入に割り当てられています。

GNU Emacs 形式のエスケープシーケンスを以下に全て示します:

\C-	コントロールプレフィックス
\M-	メタプレフィックス
\e	エスケープ文字
\\	バックスラッシュ
\"	" という文字
\’	’ という文字

GNU Emacs 形式のエスケープシーケンスに加えて、 別形式のバックスラッシュエスケープも使えます:

\a	警告（ベル）
\b	バックスペース
\d	削除（delete）
\f	フォームフィード
\n	改行（newline）
\r	復帰（carriage return）
\t	水平タブ
\v	垂直タブ
\nnn	8 進値が nnn である 8 ビット文字（1　文字につき数字　1〜3 桁）
\xHH	16 進値が HH である 8 ビット文字（16 進で 1〜2 桁）。

マクロのテキストを入力する際には、マクロ定義を示すために シングルクォートやダブルクォートを使わなければなりません。 クォートされていないテキストは関数名とみなされます。 マクロ本体では、前述のバックスラッシュによるエスケープは展開されます。 バックスラッシュはマクロのテキスト内の他の文字を全てエスケープします。 これには " や ’ も含まれます。

bash では、組み込みコマンドの bind を使って、readline の現在の キー割り当ての表示と変更を行えます。 また、組み込みコマンド set に対して -o オプションを使えば、編集モードを対話的利用の途中に切り替え可能です (後述の シェルの組み込みコマンド を参照)。

1.28.4　Readline の変数

Readline には、動作を細かくカスタマイズするために変数があります。 変数は inputrc ファイル中に

set variable-name value

という形式の文で設定できます。 特に断らない限り、readline の変数がとる値は On か Off のいずれかです。大文字と小文字の違いは考慮しません。 認識できない変数名は無視されます。 変数の値を読み取るとき、設定されてない場合や、空文字列、”on” (大文字と小文字は区別しない)、”1” は On とみなされます。それ以外の値は Off とみなされます。 変数とそのデフォルト値を以下に示します:

bell-style (audible)

readline が端末のベルを鳴らそうとしたときの動作を制御します。none が設定されている場合は、readline はベルを鳴らしません。visible が設定されている場合には、可能であれば可視ベル (visible bell) が用いられます。audible が設定されている場合には、readline は端末のベルを鳴らそうとします。

bind-tty-special-chars (On)

On が設定されていると、 カーネルの端末ドライバによって特別扱いされるコントロール文字の機能の代わりに readline の同等の機能を割り当てようとします。

comment-begin (“ # ”)

readline の insert-comment コマンドが実行されたときに挿入される文字列です。 このコマンドは emacs モードでは M-# に割り当てられ、vi コマンドモードでは # に割り当てられます。

completion-ignore-case (Off)

On が設定されていると、 readline がファイル名のマッチングと補完を行う際に 大文字と小文字が区別されません。

completion-prefix-display-length (0)

補完候補のリストでそのまま表示される共通先頭部分の文字数。 0 より大きい値が設定されると、補完候補を表示するときに、 この値より長い共通先頭部分は省略記号で置き換えられます

completion-query-items (100)

possible-completions コマンドが生成した補完候補の数が、 いくつを越えると表示の可否をユーザに問い合わせるのかを決めます。 この変数には 0 以上の任意の整数を設定できます。 補完の候補数がこの変数の値以上の場合には、 ユーザに対して候補を表示するかどうかの問い合わせがなされます。 そうでない場合には、単に端末に補完の候補がリスト表示されます。

convert-meta (On)

On を設定すると、readline は 8 番目のビットがセットされている文字を ASCII のキーシーケンスに変換します。 変換は、8 番目のビットを落として、 エスケープ文字を前に追加することによって行います (実際にはエスケープ文字をメタプレフィックスとして用います)。

disable-completion (Off)

On を設定すると、readline は単語の補完を行わなくなります。 補完される文字の編集行への挿入は、 その文字を self-insert に割り当てたかのように行われます。

editing-mode (emacs)

readline の起動時に emacs と vi のどちらに似たキー割り当てを使うのかを制御します。 editing-mode には emacs と vi のいずれかを指定できます。

echo-control-characters (On)

On が設定されていると、サポートしている OS では、 キーボード操作でシグナルが起こされたときに相当する文字を画面に出力します。

enable-keypad (Off)

On を設定すると、 readline は呼び出されたときにアプリケーションキーパッドを有効にしようとします。 一部のシステムでは、矢印キーを使うためにこれを有効にする必要があります。

enable-meta-key (On)

On が設定されていると、 readline は端末がサポートを要求するメタ修飾キーを有効にしようとします。多くの端末では、メタキーは 8 ビット文字を送信するのに使われます。

expand-tilde (Off)

On を設定すると、 readline が単語の補完を試みるときにチルダ展開が行われます。

history-preserve-point (Off)

On が設定されていると、 previous-history や next-history で履歴行を取り出したときに、 ポイントを行内の同じ位置に置こうとします。

history-size (0)

履歴リストに保存する履歴エントリの最大数を設定する。 0 が設定されていると、履歴リストのエントリの数は制限されません。

horizontal-scroll-mode (Off)

On が設定されていると、readline は表示の際に 1 行しか 使わないようになります。つまり、行がスクリーンの幅より長くなると、 新しい行に折り返すのではなく、1 つの入力行の中で横にスクロールします。

input-meta (Off)

On が設定されていると、readline は 8 ビットの入力が可能になります (つまり、読み込んだ文字の再上位ビットを落としません)。この動作は、端 末のサポートとは無関係に行われます。meta-flag という名前は、この変数の別名です。

isearch-terminators (“ C-[C-J ”)

インクリメンタル検索を終了させる文字からなる文字列です (終了後にその文字がコマンドとして実行されることもありません)。 この変数に値が設定されていなければ、 ESC と C-J でインクリメンタル検索が終わります。

keymap (emacs)

現在の readline のキーマップを設定します。有効なキーマップ名は、emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-command, .IR vi-insert です。vi は vi-command と等価で、emacs は emacs-standard と等価です。 デフォルト値は emacs です。editing-mode の値もデフォルトキーマップに影響を与えます。

mark-directories (On)

On が設定されていると、 補完されたディレクトリ名の末尾にスラッシュが追加されます。

mark-modified-lines (Off)

On が設定されていると、履歴行を表示する際に、 以前に変更されたものの先頭にアスタリスク (*) を付けて表示します。

mark-symlinked-directories (Off)

On が設定されていると、 ディレクトリへのシンボリックリンクが補完されたときに、 (mark-directories が有効であれば) 名前の末尾にスラッシュが追加されます。

match-hidden-files (On)

On が設定されていると、 ファイル名の補完で ‘.’ で始まる名前のファイル (隠しファイル) にもマッチします。 Off が設定されているときには、‘.’ で始まる名前のファイルを 補完するには先頭の ‘.’ をユーザが入力する必要があります。

menu-complete-display-prefix (Off)

On が設定されていると、 メニュー補完 (menu completion) で補完候補のリスト (空の場合も含む) を順番に表示する前に、共通先頭部分を表示します。

output-meta (Off)

On が設定されていると、 readline は 8 番目のビットが立っている文字を直接表示します。 メタ文字を前置したエスケープシーケンスでの表示は行いません。

page-completions (On)

On が設定されていると、 一度に一画面分の補完候補を表示するために readline は組み込みの more 風ページャを使います。

print-completions-horizontally (Off)

On が設定されていると、 readline は補完でマッチするものをアルファベット順にして、 縦方向ではなく横方向に並べて表示します。

revert-all-at-newline (Off)

On が設定されていると、 accept-line が実行されて readline から戻るときに、 履歴に加えられた全ての変更を元に戻します。 デフォルトでは、readline の各呼び出しを通じて 履歴行が変更され単一のアンドゥリストが保持されます。

show-all-if-ambiguous (Off)

これは補完機能のデフォルトの挙動を変えます。 On が設定されている場合、単語に対する補完候補が複数個あると、 ベルは鳴らされずに、マッチするものが即座にリスト表示されます。

show-all-if-unmodified (Off)

これは show-all-if-ambiguous と同様に 補完機能のデフォルトの挙動を変えます。 On が設定されている場合、単語に対する補完候補が複数個あって、 部分的な補完ができない (補完候補が共通先頭部分を持たない) と、 マッチするものが即座にリスト表示されます。 ベルは鳴らされません。

skip-completed-text (Off)

On が設定されていると、 これは補完機能がマッチしたものを行に挿入するときのデフォルトの挙動を変えます。 単語の途中で補完を実行したときにだけ意味があります。 有効になっていると、readline は、単語のカーソル以後の部分が重複しないように、 補完された単語のうちポイント以後の文字を挿入しません。

visible-stats (Off)

On が設定されていると、 stat(2) で得られるファイルの種類を表す文字が、 補完候補のリスト表示の際に追加されます。

1.28.5　Readline の条件構文

Readline には、 C 言語のプリプロセッサの条件付きコンパイル機能と 似たコンセプトに基づく機能が実装されています。 これを使うと、条件に応じてキー割り当てや変数の設定を実行できます。 パーザディレクティブは 4 つあります。

$if

$if 構文を使うと、編集モードや使用中の端末、 readline を使っているアプリケーションに応じた割り当てを行えます。 行末までの全テキストが評価対象となります。 これを分離するには文字は必要ありません。

- mode
  $if ディレクティブの mode= 形式は、 readline が emacs モードか vi モードのどちらになっているかを調べるために使います。 例えば、これを set keymap コマンドと一緒に使い、readline が emacs モードで始まったときだけ キーの割り当てを emacs-standard や emacs-ctlx キーマップに設定できます。

- term
  term= 形式を使うと、 端末固有のキー割り当てを入れることができます。 これは端末のファンクションキーが出力する キーシーケンスを割り当てる場合などに使えます。 = の右辺の単語の評価は、 端末の完全な名前および最初の - の前までの部分的な名前の両方に対して行われます。 例えば、 sun は sun と sun-cmd の両方にマッチします。

- application
  application はアプリケーション固有の設定を入れるために使います。 readline ライブラリを使っているプログラムはそれぞれ アプリケーション名 (application name) を設定するので、 初期化ファイルはそれが特定の値かどうかを調べられます。 これを使って、特定のプログラム用で便利な機能にキーシーケンスを割り当てできます。例えば、以下のコマンドは、bash の場合に、現在の単語または直前の単語をクォートするキーシーケンスを追加します。

```
$if Bash
# Quote the current or previous word
"\C-xq": "\eb\"\ef\""
$endif
```

$endif

このコマンドは $if コマンドを終了させます (上記の例の通り)。

$else

$if ディレクティブのこの分岐内に書かれたコマンドは、 評価が失敗したときに実行されます。

$include

このディレクティブはファイル名 1 つを引き数に取り、 そのファイルからコマンドとキー割り当てを読み込みます。 例えば以下のディレクティブを使うと /etc/inputrc が読み込まれます:

```
$include /etc/inputrc
```

1.28.6　検索

readline には、指定した文字列を含む行をコマンド履歴内から探すコマンドが用意されています (後述の 履歴 を参照)。 検索のモードには、 インクリメンタル (incremental) モードと 非インクリメンタル (non-incremental) モードの 2 つがあります。

インクリメンタル検索では、 ユーザが検索文字列全体の入力が終わるより前に検索が開始されます。 検索文字列に文字が追加されるたびに、 readline はそれまでに入力した文字列にマッチする履歴から 次のエントリを表示します。インクリメンタル検索では、探している履歴エントリを見つけるために 必要な数の文字を入力すれば十分です。 isearch-terminators の値に含まれる文字を入力すると、 インクリメンタル検索は終了します。 この変数に値が代入されていない場合には、エスケープ文字または Control-J でインクリメンタル検索が終了します。 Control-G はインクリメンタル検索を異常終了させ、元の行を復元します。検索が終了すると、検索文字列を含む履歴エントリが現在の行になります。

マッチした他の履歴リストのエントリを見つけるには、Control-S または Control-R を適宜入力します。これにより、 今まで入力した検索文字列にマッチする次のエントリを 履歴の前方または後方に向かって探します。 readline に割り当てられた他のキーシーケンスを入力すると検索が終わり、 入力したキーシーケンスに対応するコマンドが実行されます。 例えば、改行は検索を終了させて、その行を入力します。 これにより、履歴リスト中のコマンドが実行されます。

readline は前回のインクリメンタル検索で入力された 検索文字列を覚えています。 新しい検索文字を間に挟まずに Control-R を 2 回入力すると、 覚えている検索文字列が使われます。

インクリメンタルでない検索の場合には、検索文字列全体を読み込んでから、履歴行にマッチするものの検索を始めます。 検索文字列には、ユーザが入力したものか、現在の行の内容の一部が使えます。

1.28.7　Readline のコマンド名

以下は、 コマンドの名前およびそれらが割り当てられている デフォルトのキーシーケンスの一覧です。対応するキーシーケンスがないコマンド名には、デフォルトではキーは割り当てられていません。 以下の説明では、ポイント (point) は現在のカーソルの位置を表し、 マーク (mark) は set-mark コマンドによって保存されたカーソル位置を表します。 ポイントとマークの間のテキストは、リージョン (region) と呼びます。

1.28.8　移動コマンド

beginning-of-line (C-a)

現在の行の先頭に移動します。

end-of-line (C-e)

行の末尾に移動します。

forward-char (C-f)

1 文字進みます。

backward-char (C-b)

1 文字戻ります。

forward-word (M-f)

次の単語の最後に進みます。単語は英数字 (英字と数字) からなります。

backward-word (M-b)

現在あるいは直前の単語の先頭に戻ります。 単語は英数字 (英字と数字) からなります。

shell-forward-word

次の単語の最後に進みます。 単語はクォートされていないシェルのメタ文字で区切られます。

shell-backward-word

現在の単語あるいは前の単語の先頭に戻ります。 単語はクォートされていないシェルのメタ文字で区切られます。

clear-screen (C-l)

現在の行を画面の一番上の行に残したまま、画面をクリアします。 引き数を付けると、画面をクリアせずに現在の行を再描画します。

redraw-current-line

現在の行を再描画します。

1.28.9　履歴操作のためのコマンド

accept-line (Newline, Return)

　カーソルの位置に関わらず、行を確定します。この行が空でなければ、変数 HISTCONTROL の状態に従って、これが履歴リストに追加されます。 行が履歴リストを変更したものならば、履歴行は元の状態に戻されます。

previous-history (C-p)

　履歴リストを戻り、履歴リストから前の行を取り出します。

next-history (C-n)

　履歴リストを進み、履歴リストから次の行を取り出します。

beginning-of-history (M-<)

　履歴の最初の行に移動します。

end-of-history (M-<)

　入力履歴の最後に移動します。つまり、現在入力中の行に移動します。

reverse-search-history (C-r)

　現在の行を開始点にして後方に向かって検索を行い、 必要に応じて履歴を「遡って」いきます。これはインクリメンタル検索です。

forward-search-history (C-s)

　現在の行を開始点にして前方に向かって検索を行い、 必要に応じて履歴を「下って」いきます。これはインクリメンタル検索です。

non-incremental-reverse-search-history (M-p)

　現在の行を開始点にして、履歴リストを後方に向かって検索を行います。 ユーザが入力した文字列を非インクリメンタルモードで検索します。

non-incremental-forward-search-history (M-n)

　現在の行を開始点にして、履歴リストを前方に向かって検索を行います。 ユーザが入力した文字列を非インクリメンタルモードで検索します。

history-search-forward

履歴を前方に向かって検索し、 現在の行の先頭とポイントの間の文字列を探します。 これは非インクリメンタル検索です。

history-search-backward

履歴を後方に向かって検索し、 現在の行の先頭およびポイントの間の文字列を探します。 これは非インクリメンタル検索です。

yank-nth-arg (M-C-y)

前のコマンドに対する最初の引き数 (通常は前の行の 2 番目の単語) をポイントに挿入します。引き数 n が付いていると、前のコマンドの n 番目の単語 (前のコマンドの単語は 0 から数えます)。 引き数が負ならば、前のコマンドの最後から n 番目の単語が挿入されます。 引き数 n が一度決まると、コマンドの引き数が履歴の展開での ”!n” の指定のように展開されます。

yank-last-arg (M-., M-_)

前のコマンドの最後の引き数 (一つ前の履歴エントリの最後の単語) を挿入します。 引き数があると、yank-nth-arg と全く同じように動作します。 yank-last-arg を繰り返して呼び出すと、 履歴リストを遡って参照が行われ、 それぞれの行の最後の引き数が順番に挿入されます。繰り返して呼び出すときの引き数は、履歴の中を動く方向を決めます。負の引き数は、履歴の中を動く方向 (後方または前方) を反転します。履歴の展開での ”!$” の指定のように、 最後の引き数の展開に履歴の展開の機能が使われます。

shell-expand-line (M-C-e)

シェルが行うのと同じように行の展開を行います。 エイリアスや履歴の展開を含め、シェルの行う全ての単語展開が行われます。 履歴の展開の説明については、後述の 履歴の展開 を参照してください。

history-expand-line (M-^)

現在の行で履歴の展開を行います。 履歴の展開の説明については、後述の 履歴の展開 を参照してください。

magic-space

現在の行で履歴の展開を行い、空白を挿入します。 履歴の展開の説明については、後述の 履歴の展開 を参照してください。

alias-expand-line

現在の行でエイリアスの展開を行います。 エイリアスの展開の説明については、前述の エイリアス を参照してください。

history-and-alias-expand-line

現在の行で履歴とエイリアスの展開を行います。

insert-last-argument (M-., M-_)

yank-last-arg と同じものです。

operate-and-get-next (C-o)

現在の行を実行し、 履歴内の現在の行の次の行を編集用に取得します。 引き数は全て無視されます。

edit-and-execute-command (C-xC-e)

エディタを起動して現在のコマンドラインの内容を開き、 その結果をシェルのコマンドとして実行します。 bash はエディタとして$VISUAL, $EDITOR, emacs の順で起動を試みます。

1.28.10　テキスト編集のためのコマンド

delete-char (C-d)

ポイントの文字を削除します。 ポイントが行の先頭であり、その行に文字がなく、 さらに打ち込んだ最後の文字が delete-char に割り当てられていなければ、 EOF が返されます。

backward-delete-char (Rubout)

カーソルの前の文字を削除します。数値の引き数を与えると、 削除したテキストをキルリングに保存します。

forward-backward-delete-char

カーソルが行末になければカーソルがある位置の文字を削除します。カーソルが行末にある場合には、カーソルの前の文字を削除します。

quoted-insert (C-q, C-v)

次に打ち込んだ文字をそのまま行に追加します。 これは C-q 等の文字を挿入するために使います。

tab-insert (C-v TAB)

タブ文字を挿入します。

self-insert (a, b, A, 1, !, ...)

打ち込んだ文字を挿入します。

transpose-chars (C-t)

ポイントの前にある文字を動かし、 現在ポイントがある文字の後ろに持っていきます。 同時にポイントも前に進みます。ポイントが行の最後にある場合は、 ポイントの前の 2 文字が交換されます。 負の引き数を指定すると、何も起こりません。

transpose-words (M-t)

ポイントの前にある単語を動かし、 ポイントの後ろにある単語の後ろに持っていきます。 この際には、ポイントも始めに前にあった単語の後ろまで移動します。 ポイントが行の最後にある場合は、行の最後の 2 単語が交換されます。

upcase-word (M-u)

現在の (または後ろの) 単語を大文字にします。 負の引き数を指定すると、前の単語を大文字にしますが、 ポイントは動きません。

downcase-word (M-l)

現在の (または後ろの) 単語を小文字にします。 負の引き数を指定すると、前の単語を小文字にしますが、 ポイントは動きません。

capitalize-word (M-c)

現在の (または後ろの) 単語をキャピタライズします (単語の先頭の文字を大文字にします)。 負の引き数を指定すると前の単語をキャピタライズしますが、 ポイントは動きません。

overwrite-mode

上書きモードをトグルさせます。 正の引き数を指定すると上書きモードに切り替えます。 負の引き数を指定すると挿入モードに切り替えます。 このコマンドは emacs モードでのみ有効です。 vi モードでは違った形で上書きします。 readline() を呼び出したときは毎回、挿入モードで始まります。 上書きモードでは、self-insert が割り当てられた文字は、 ポイントの位置のテキストを右に押し出すのではなく置き換えます。backward-delete-char が割り当てられた文字は、 ポイントの前の文字をスペースで置き換えます。 このコマンドはデフォルトではキーに割り当てられていません。

1.28.11　キルとヤンク

kill-line (C-k)

ポイントから行末までのテキストをキルします。

backward-kill-line (C-x Rubout)

現在のカーソル位置から行頭までをキルします。

unix-line-discard (C-u)

ポイントから行頭までをキルします。 キルされたテキストはキルリング (kill-ring) に入ります。

kill-whole-line

現在の行の文字を全てキルします。ポイントの位置は関係ありません。

kill-word (M-d)

ポイントから現在の単語の終わりまでをキルします。 ポイントが単語と単語の間であれば、後ろの単語の終わりまでをキルします。 単語の境界は forward-word で使われているものと同じです。

backward-kill-word (M-Rubout)

ポイントの前にある単語をキルします。 単語の境界は backward-word で使われているものと同じです。

shell-kill-word (M-d)

ポイントから現在の単語の終わりまでをキルします。 ポイントが単語と単語の間であれば、後ろの単語の終わりまでをキルします。 単語の境界は shell-forward-word で使われているものと同じです。

shell-backward-kill-word (M-Rubout)

ポイントの前にある単語をキルします。 単語の境界は shell-backward-word で使われているものと同じです。

unix-word-rubout (C-w)

ポイントの前にある単語をキルします。 その際には空白を単語の境界として用います。 キルされたテキストはキルリングに入ります。

unix-filename-rubout

ポイントの前にある単語をキルします。 その際には空白とスラッシュを単語の境界として用います。 キルされたテキストはキルリングに入ります。

delete-horizontal-space (M-\)

ポイントの周りの空白とタブを全て削除します。

kill-region

現在のリージョン中のテキストをキルします。

copy-region-as-kill

リージョン中のテキストをキルバッファにコピーします。

copy-backward-word

ポイントの前の単語をキルバッファにコピーします。単語の境界は backward-word と同じです。

copy-forward-word

ポイントの後ろの単語をキルバッファにコピーします。単語の境界は forward-word と同じです。

yank (C-y)

キルリングの先頭のテキストをバッファ中のポイントにヤンクします。

yank-pop (M-y)

キルリングの順番を 1 つ移動し、新たに先頭になったテキストをヤンクします。 yank または yank-pop の後にしか使えません。

1.28.12　数値の引き数

digit-argument (M-0, M-1, ..., M--)

すでに入力された引き数にこの数字を追加するか、 またはこの数字によって新しい引き数を始めます。 M-- を使うと、負の値を指定できます。

universal-argument

引き数を指定する別の方法です。 このコマンドに続けて 1 つ以上の数字が入力された場合 (頭にマイナス記号を付けることもできます)、 これらの数字で引き数が定義されます。 このコマンドの後に数字が続いた場合、 universal-argument を再び実行すると数値の引き数を終了しますが、 そうでない場合は無視されます。 特殊なケースとして、 このコマンドの直後に数字でもマイナス記号でもない文字がある場合には、 次のコマンドに対する引き数カウントが 4 倍になります。 引き数カウントは最初は 1 なので、 この機能を一度実行すると引き数カウントは 4 になり、 もう一度実行すると引き数カウントは 16 になります。 それ以降も同様です。

1.28.13　補完

complete (TAB)

ポイントの前のテキストについて補完を試みます。 bash が補完を行う際には、 変数 (テキストが $ で始まる場合)、 ユーザ名 (テキストが~ で始まる場合)、 ホスト名 (テキストが @ で始まる場合)、 コマンド (エイリアスや関数も含みます) の順序でマッチを行います。 いずれにもマッチしない場合には、 ファイル名補完を試みます。

possible-completions (M-?)

ポイントの前のテキストの補完候補を列挙します。

insert-completions (M-*)

ポイントの前のテキストを補完して得られるものを全て挿入します。挿入されるものは、possible-completions で列挙されるものと同じです。

menu-complete

complete コマンドに似ていますが、 補完される単語を補完候補リストのうちの 1 つと置換します。 menu-complete を繰り返して実行すると、 補完候補のリストが次々と順番に挿入されます。 補完リストの最後まで来るとベルが (bell-style の設定に基づいて) 鳴らされ、元のテキストに戻ります。 引き数 n を指定すると、リスト中の位置が n 個進みます。 負の引き数を指定すると、リスト中を逆向きに戻れます。 このコマンドは TAB に割り当てることを意図して用意されたものですが、 デフォルトでは割り当ては行われていません。

menu-complete-backward

menu-complete コマンドに似ていますが、 menu-complete に負の引き数を与えたときのように、 補完候補のリストを逆向きに進みます。 このコマンドはデフォルトではキーに割り当てられていません。

delete-char-or-list

カーソルが行頭や行末に無ければ、カーソルの下の文字を削除します (delete-char と同様です)。 カーソルが行末にある場合は、possible-completions と同じ動作をします。 このコマンドはデフォルトではキーに割り当てられていません。

complete-filename (M-/)

ポイントの前のテキストについてファイル名の補完を試みます。

possible-filename-completions (C-x /)

ポイントの前のテキストについて補完候補を列挙します。 テキストはファイル名として扱われます。

complete-username (M-~)

ポイントの前のテキストについて補完を試みます。 テキストはユーザ名として扱われます。

possible-username-completions (C-x ~)

ポイントの前のテキストについて補完候補を列挙します。 テキストはユーザ名として扱われます。

complete-variable (M-$)

ポイントの前のテキストについて補完を試みます。 テキストはシェル変数として扱われます。

possible-variable-completions (C-x $)

ポイントの前のテキストについて補完候補を列挙します。 テキストはシェル変数として扱われます。

complete-hostname (M-@)

ポイントの前のテキストについて補完を試みます。 テキストはホスト名として扱います。

possible-hostname-completions (C-x @)

ポイントの前のテキストについて補完候補を列挙します。 テキストはホスト名として扱います。

complete-command (M-!)

ポイントの前のテキストについて補完を試みます。 テキストはコマンド名として扱います。 コマンド補完の際にマッチングを試みる順序は、 エイリアス、予約語、シェル関数、シェルの組み込みコマンド、実行ファイルです。

possible-command-completions (C-x !)

ポイントの前のテキストについて補完候補を列挙します。 テキストはコマンド名として扱います。

dynamic-complete-history (M-TAB)

ポイントの前のテキストについて補完を試みます。 履歴リストの各行に対してテキストの比較が行われ、マッチしたものが補完の候補となります。

dabbrev-expand

ポイントの前のテキストについてメニュー補完を試みます。 履歴リストの各行に対してテキストの比較が行われ、 マッチしたものが補完の候補となります。

complete-into-braces (M-{)

ファイル名補完を実行し、補完候補のリストを挿入します。 シェルから利用可能なリストとなるように、返される候補はブレースで括られます (前述の ブレース展開 を参照)。

1.28.14　キーボードマクロ

start-kbd-macro (C-x ()

現在のキーボードマクロに対して入力される文字列の保存を開始します。

end-kbd-macro (C-x))

現在のキーボードマクロに対して入力された文字列の保存を終了し、その定義を格納します。

call-last-kbd-macro (C-x e)

最後に定義されたキーボードマクロを再実行します。 マクロ内の各文字があたかもキーボードから入力されたかのように出力されます。

1.28.15　その他

re-read-init-file (C-x C-r)

inputrc ファイルの内容を読み込み、 このファイル中の割り当てや変数設定を取り込みます。

abort (C-g)

現在の編集行を捨て、端末のベルを鳴らします (動作は bell-style の設定に従います)。

do-uppercase-version (M-a, M-b, M-x, ...)

メタ文字と共に入力された文字 x が小文字であれば、 x に対応する大文字に割り当てられているコマンドを実行します。

prefix-meta (ESC)

次に入力される文字を、 メタ文字と共に入力されたことにします。ESC f は Meta-f と同じ意味です。

undo (C-_, C-x C-u)

インクリメンタルアンドゥを行います。 履歴は行ごとに別々に記憶されています。

revert-line (M-r)

この行に対して行った変更を全て取り消します。 このコマンドは、行が初期状態に戻るまで undo コマンドを実行するようなものです。

tilde-expand (M-&)

現在の単語についてチルダ展開を実行します。

set-mark (C-@, M-<space>)

ポイントにマークを設定します。 数字の引き数が与えられた場合には、 マークはその位置に設定されます。

exchange-point-and-mark (C-x C-x)

ポイントをマークと入れ換えます。 保存されている位置が現在のカーソル位置になり、 古いカーソル位置がマークとして保存されます。

character-search (C-])

文字を 1 つ読み込み、 その文字が次に現われる場所にポイントを移動させます。 負のカウントを与えると、その文字が前に現われた場所を探します。

character-search-backward (M-C-])

文字を 1 つ読み込み、 その文字が前に現われた場所にポイントを移動させます。 負のカウントを与えると、その文字が次に現われる場所を探します。

skip-csi-sequence

Home や End などのキーに定義されるような複数キーからなるシーケンスを 使い切るだけの文字を読み込みます。 これらのシーケンスは、通常 ESC-[で表されるコントロールシーケンス (CSI) で始まります。このシーケンスが "\[" に割り当てられているとすると、 キーが明示的に readline のコマンドに割り当てられている場合を除き、 それらのシーケンスを生み出すキーは、 はぐれた文字を編集バッファに挿入されず、何もしません。 このコマンドはデフォルトではキーに割り当てられていませんが、 多くの場合 ESC-[に割り当てられます。

insert-comment (M-#)

数字の引き数を指定しない場合、readline の comment-begin 変数の値が現在の行の先頭に挿入されます。 数字の引き数を指定すると、トグル状に動作します。 つまり、行の先頭の文字が comment-begin の値にマッチしない場合には、 その値が挿入されます。マッチする場合には、comment-begin の文字が行の先頭から削除されます。 いずれの場合も、改行が打ち込まれたのと同じように行の入力が完了します。comment-begin のデフォルト値によって、 このコマンドは現在の行をシェルのコメントとします。 数字の引き数を指定したことでコメントの文字が削除されると、 行はシェルにより実行されます。

glob-complete-word (M-g)

ポイントの前の単語がパス名展開のパターンとして扱われます。 末尾に暗黙のアスタリスクが付いているものとみなされます。 このパターンを使って補完候補のファイル名のリストが生成されます。

glob-expand-word (C-x *)

ポイントの前の単語がパス名展開のパターンとして扱われ、 この単語と置き換えられる形でマッチするファイル名のリストが挿入されます。数字の引き数を与えると、パス名展開の前に末尾にアスタリスクを追加します。

glob-list-expansions (C-x g)

glob-expand-word で生成されるのと同じ展開結果のリストが表示され、 行が再描画されます。 数字の引き数を与えると、パス名展開の前に末尾にアスタリスクを追加します。

dump-functions

全ての関数とそのキー割り当てを、 readline の出力ストリームに出力します。 数値の引き数を与えると、出力は inputrc に書き込める形に整形されます。

dump-variables

全ての設定可能な readline の変数とその値を、 readline の出力ストリームに出力します。 数値の引き数を与えると、 出力は inputrc に書き込める形に整形されます。

dump-macros

マクロに割り当てられた readline のキーシーケンスと マクロが出力する文字列を全て出力します。 数値の引き数を与えると、 出力は inputrc に書き込める形に整形されます。

display-shell-version (C-x C-v)

現在実行している bash のバージョン情報を表示します。

1.28.16　プログラム補完

complete 組み込みコマンドで 補完仕様 (compspec) が定義されているコマンドに対して 引き数の単語補完が試みられると、 プログラム補完の機能が呼び出されます (組み込みコマンド complete については、後述する シェルの組み込みコマンド を参照)。

まず、コマンド名が特定されます。 コマンド名の単語が空文字列であれば (空の行の先頭で補完しようとしたとき)、 complete の -E オプションで定義された補完仕様が使われます。 そのコマンドに対して補完仕様が定義されていれば、 その補完仕様がその単語の補完候補のリスト生成に使われます。 コマンドの単語がフルパス名であれば、 最初にフルパス名の補完仕様が検索されます。 フルパス名に対する補完仕様が見つからなかった場合は、 最後のスラッシュ以降の部分に対して該当する補完仕様を見つけようとします。補完仕様が見つからなかった場合は、 complete の -D オプションで定義された補完仕様が使われます。

補完仕様が見つかると、その補完仕様を使って マッチする単語のリストが生成されます。 補完仕様が見つからなかった場合は、 前述の 補完 の節で説明したような bash のデフォルトの補完が行われます。

まず、補完仕様で指定された動作が用いられます。 補完される単語の前置部分にマッチするものだけが返されます。 ファイル名やディレクトリ名の補完に-f や -d オプションが使用された場合は、シェル変数 FIGNORE がマッチのフィルタとして使用されます。

続いて -G オプションによって指定された ファイル名展開パターンの補完が生成されます。 パターンによって生成された単語は、 補完される単語とマッチする必要はありません。 GLOBIGNORE シェル変数はマッチのフィルタとしては使われませんが、 FIGNORE 変数は使用されます。

次に、 -W オプションで指定された引き数の文字列が考慮されます。 文字列は、最初に IFS 特殊変数の文字を区切り文字として分割されます。 シェルのクォート処理は考慮されます。 それぞれの単語は、前述の 展開 で示したように、ブレース展開、チルダ展開、パラメータと変数の展開、 コマンド置換、算術式展開、パス名展開が行われます。 結果は、前述の単語の分割で示した規則によって分割されます。 展開の結果は補完される単語の前置部分とマッチが行われ、 マッチした単語が補完候補となります。

これらのマッチが生成された後、シェル関数や -F や -C オプションで指定されたコマンドが呼び出されます。コマンドや関数が呼び出されるときは、COMP_LINE, COMP_POINT, COMP_KEY, COMP_TYPE 変数に前述のシェル変数で示すように値が設定されます。 シェル関数が呼び出される場合は、 COMP_WORDS と COMP_CWORD 変数も設定されます。 関数やコマンドが呼び出されるときは、 最初の引き数は引き数が補完されるコマンドの名前、 二番目の引き数は補完される単語、 三番目の引き数は現在のコマンドラインで補完中の単語の 前に置かれる単語となります。 補完される単語に対して生成された補完の候補はフィルタリングされません。 関数やコマンドは生成されたマッチとは無関係に補完されます。

-F で指定された関数がまず呼び出されます。 関数は、後述する compgen 組み込みコマンドを含めた、全てのシェルの機能を使ってマッチを生成します。補完候補は必ず COMPREPLY 配列変数に格納されます。

続いて -C オプションで指定されたコマンドが呼び出され、 環境変数をコマンド置換します。 このコマンドは、補完候補を 1 行にひとつずつ標準出力に出力します。 必要があれば、バックスラッシュが改行をエスケープするために使用されます。

全ての補完候補が生成された後で、 -X オプションで指定されたフィルタが補完候補に作用します。 フィルタは、パス名展開で使用されたようなパターンです。パターン中の & は補完される単語に置換されます。文字通りの & はバックスラッシュでエスケープします。 バックスラッシュはマッチを試みる前に削除されます。 パターンにマッチした補完は候補から削除されます。 先行する! はパターンを否定します。 この場合、パターンにマッチしなかった補完が削除されます。

最後に、-P と -S オプションで指定された 前置部分と後置部分が補完候補のそれぞれに加えられます。そして結果が readline 補完コードに補完候補のリストとして返されます。

直前に行われた動作が何にもマッチせず、補完仕様が定義されたときに -o dirnames オプションが complete に与えられていれば、 ディレクトリ名への補完が試みられます。

補完仕様が定義されたときに -o plusdirs オプションが complete に与えられていれば、 ディレクトリ名への補完が試みられ、マッチした候補が全て ほかの動作の結果に付け加えられます。

デフォルトでは、補完仕様が見つかった場合、 それが生成したものがなんであれ、 全ての補完候補のリストとして、補完コードに返されます。 デフォルトの bash 補完は試みられず、 readline のデフォルトのファイル名補完は無効になります。 補完仕様が定義されたときに -o bashdefault オプションが complete に

与えられていれば、 補完仕様が何にもマッチしなければ bash のデフォルトの補完が試みられます。 補完仕様が定義されたときに -o default オプションが complete に与えられていれば、 補完仕様が (試みられていれば、デフォルトの bash の補完も) 何にもマッチしなければ readline のデフォルトの補完が行われます。

補完仕様でディレクトリ名の補完が必要とされる場合、 プログラム補完の関数は readline に、 ディレクトリへのシンボリックリンクに補完された名前の最後に スラッシュを追加させます。 これは readline の変数 mark-symlinked-directories の設定に関わらず、 readline の変数 mark-directories の値に左右されます。

動的に補完を変えるための方法があります。 これは complete -D で指定されたデフォルトの補完と 組み合わせたときに非常に便利です。 補完の処理のために実行されたシェル関数は、終了ステータスとして 124 を返すことで、補完を再度試みることを指示できます。 シェル関数が 124 を返し、補完が試みられるコマンド (関数が実行されるときの第 1 引き数) に対して定義された補完仕様が 変更されていれば、プログラム補完はコマンドの新しい補完仕様を 探すために最初からやり直されます。 これにより一連の補完が、一度に読み込まれるのではなく、 補完が試みられるときに動的に組み立てられます。

例えば、補完仕様のライブラリがあり、それぞれがコマンドの名前に 合わせたファイルに保存されていると仮定すると、 以下のデフォルトの補完関数は補完を動的に読み込みます。

```
_completion_loader()
{
      . "/etc/bash_completion.d/$1.sh" >/dev/null 2>&1 && return 124
}
complete -D -F _completion_loader
```

1.29　履歴 (HISTORY)

-o history オプションを組み込みコマンドの set で有効にすると、 コマンド履歴 (command history) (以前に入力したコマンドのリスト) にアクセスできるようになります。 変数 HISTSIZE の値が、履歴リストに保存するコマンドの数になります。 過去に入力したコマンドのうち、最新 HISTSIZE 個分 (デフォルトは 500 個) のテキストが保存されます。 シェルは各コマンドを、パラメータ展開や変数展開 (前述の 展開を参照) を行う前の形で履歴リストに格納します。 ただし、履歴展開は実行してから格納します。 履歴展開はシェル変数 HISTIGNORE と HISTCONTROL の値に従って実行されます。

起動時に、履歴は変数 HISTFILE (デフォルトは ~/.bash_history) が示すファイルの内容で初期化されます。 HISTFILE で指定されたファイルは、HISTFILESIZE で指定された行数を越えないように、必要に応じて切り詰められます。 履歴ファイルを読み込むときに、履歴のコメント文字で始まり直後に数字が続く行は、 直前の履歴行のタイムスタンプとして解釈されます。 こうしたタイムスタンプを表示するかどうかは、変数 HISTTIMEFORMAT の値により決まります。 対話的なシェルが終了する際には、最近の $HISTSIZE 個の行が履歴リストから $HISTFILE にコピーされます。 シェルオプション histappend (シェルの組み込みコマンド の項の shopt の説明を参照) が有効になっていると、 これらの行が履歴ファイルの末尾に追加されます。 このオプションが無効ならば、履歴ファイルは上書きされます。 HISTFILE が設定されていないか、履歴ファイルが書き込めない状態だと、 履歴は保存されません。 変数 HISTTIMEFORMAT が設定されている場合、シェルのセッションを越えて保持されるよう、 タイムスタンプが履歴のコメント文字を付けて履歴ファイルに書き込まれます。 履歴のコメント文字は、タイムスタンプをほかの履歴行と区別

するために使われます。 履歴の保存を行った後には、履歴ファイルは行数が HISTFILESIZE 行を越えないように切り詰められます。 HISTFILESIZE が設定されていなければ、切り詰めは行われません。

組み込みコマンド fc (後述の シェルの組み込みコマンド を参照) を用いると、履歴リストの一部を表示・編集して再実行できます。 組み込みコマンドの history を用いると、履歴リストを表示・編集したり、 履歴ファイルを操作したりできます。 コマンドライン編集を使っている場合には、 各編集モードでいろいろな検索コマンドが利用でき、 履歴リストへアクセスできます。

このシェルでは、どのコマンドが履歴リストに保存されるかを制御できます。HISTCONTROL 変数と HISTIGNORE 変数を設定すると、 シェルは入力されたコマンドの一部しか保存しなくなります。 シェルオプションの cmdhist を有効にすると、シェルは複数行に別れているコマンドの各行を 同じ履歴エントリに保存しようとします。 この際には、文法的な正しさを保つためにセミコロンが必要に応じて追加されます。 シェルオプションの lithist を有効にすると、このシェルは行の途中に セミコロンではなく改行文字を置く形でコマンドを保存します。 シェルオプションの設定と設定取り消しについては、後述の シェルの組み込みコマンド における説明を参照してください。

1.30　履歴の展開

このシェルは、 csh の履歴展開と同じような履歴機能をサポートしています。このセクションでは、履歴展開で使用できる記法・機能を説明します。 この機能は対話的シェルならばデフォルトで有効になっていますが、 組み込みコマンド set の +H オプション (後述の シェルの組み込みコマンド を参照) で無効にできます。 非対話的シェルの場合は、デフォルトでは履歴展開は行われません。

履歴展開は、履歴リスト中の単語を入力ストリームに入れます。 この機能を利用すると、コマンドを繰り返したり、 前のコマンドで指定したオプションを現在の入力行に挿入したり、 前のコマンドの誤りを手早く直したり、 といったことが簡単にできるようになります。

履歴展開が実行されるのは、入力行全体を読み込んだ直後であり、 シェルが行を単語に分割するよりも前です。 履歴展開の動作は 2 段階で行われます。 まず最初に、置換に使う行を履歴リストから選びます。 次に、その行のどの部分を現在の行に書き込むかを選択します。 履歴リストから選ばれた行はイベント (event) と呼ばれ、 この行のうち動作の対象となる部分を単語列 (words) と呼びます。 様々な修飾子 (modifier) が利用でき、 選択された単語列の操作が可能になっています。 行の単語への分割は入力を読み込むときと同じように行われるので、 メタ文字で区切られた複数の単語をクォートで括ったものは 1 つの単語とみなされます。 履歴展開が行われるのは、履歴展開文字が現われたときです。 履歴展開文字はデフォルトでは ! です。 履歴展開文字をクォートできるのは、 バックスラッシュ (\) とシングルクォートだけです。

履歴展開文字の直後にあって、クォートされていなくても、 履歴展開されない文字があります。 空白文字、タブ文字、改行文字、復帰文字、= です。 シェルオプションの extglob が有効になっている場合には、(も展開されません。

組み込みコマンドの shopt を用いて、何種類かあるシェルオプションを設定すると、 履歴展開の動作を調整できます。 シェルオプションの histverify が有効で (組み込みコマンド shopt の説明を参照)、かつ readline が使われている場合には、 履歴置換を行った結果はすぐにはシェルのパーザに渡されません。 展

開された行は readline の編集バッファに再び読み込まれ、さらに編集が行える状態になります。 readline を使用しており、かつシェルオプションの histreedit が有効である場合、履歴置換が失敗してもその結果は readline の編集バッファに再び読み込まれ、訂正できる状態となります。 組み込みコマンド history の -p オプションを使うと、実際に履歴展開を行う前に、 どのように展開されるのかを見ることができます。 組み込みコマンド history の-s オプションを使うと、実際にコマンドの実行をせずに、 コマンドを履歴リストの末尾に追加でき、 それ以降の呼び出しで利用できるようになります。

このシェルでは、履歴展開機構で使ういろいろな文字を制御できます (前述のシェル変数 の項目における histchars の説明を参照)。 シェルは、履歴ファイルに書き込むときに、履歴のコメント文字を使って、履歴のタイムスタンプであることが分かるようにします。

1.30.1　イベント指示子 (Event Designator)

イベント指示子は、 履歴リスト中のコマンドラインエントリを参照するものです。 絶対位置の参照でない限り、イベントは履歴リスト内の 現在の位置からの相対的な位置を示します。

!

履歴置換を開始します。ただし、 ブランク文字、 改行文字、 = 、 ((シェルオプション extglob が組み込みコマンド shopt によって有効になっている場合) のいずれかが後に続く場合は除きます。

!n

n 個目のコマンドラインを参照します。

!-n

現在から n 個前のコマンドを参照します。

!!

直前のコマンドを参照します。'!-1' と同義です。

!string

string で始まるコマンドのうち、履歴リスト中の現在位置以前で、 一番近いところで実行したものを参照します。

!?string[**?**]

string を含むコマンドのうち、履歴リスト中の現在位置以前で、 一番近いところで実行したものを参照します。 string の直後が改行文字ならば、最後の ? は省略してもかまいません。

^string1^string2^

簡易置換。 string1 を string2 に置換して直前のコマンドを繰り返します。“ !!:s/string1/string2/ ”と同義です (後述の修飾子を参照)。

!#

これまでに打ち込んだコマンドライン全体。

1.30.2　単語指示子 (Word Designators)

単語指示子 (word designator) は、 イベントから所望の欲しい単語を選ぶときに用いられます。 イベント指定と単語指示子のセパレータには : を用います。 単語指示子が ^, $, *, -, % のいずれかで始まる場合には、このセパレータは省略できます。 単語には行の先頭から順に番号が振られ、 先頭の単語が 0 になります。 単語は現在の行に、空白 1 つで区切られて挿入されます。

0 (ゼロ)

0 番目の単語。シェルにとっては、コマンドを表す単語になります。

n

n 番目の単語。

^

最初の引き数。つまり 1 番目の単語です。

$

最後の引き数。

%

‘?string?’ 検索にマッチする、一番現在に近い単語。

x-y

単語の範囲。‘0-y’ の省略形として ‘-y’ が使えます。

*****

0 番目を除く全ての単語。これは ‘1-$’ の別表現です。 イベント中に単語が 1 つしかない場合に * を使ってもエラーにはなりません。 この場合には空文字列が返されます。

x*

x-$ の省略形です。

x-

x* と同様に x-$ の省略形ですが、 ただし最後の単語は含みません。

イベント指定なしに単語指示子が与えられた場合、 直前のコマンドがイベントとして使われます。

1.30.3　修飾子 (Modifiers)

単語指示子 (省略可能) の後には、 1 個以上の以下に示す修飾子を並べて置くことができます。 それぞれの修飾子の前には ':' を付けます。

h

パス名から末尾にある部分 (ファイル名) を取り除き、 前の方 (ディレクトリ部) だけを残します。

t

パス名から前の方 (ディレクトリ部) を取り除き、 末尾にある部分 (ファイル名) だけを残します。

r

末尾にある .xxx 形式のサフィックスを取り除き、 ベース名 (basename) だけを残します。

e

末尾のサフィックスだけを残して、全ての部分を取り除きます。

p

新しいコマンドを表示しますが、実行はしません。

q

置換が行われた単語をクォートし、それ以上の置換が行われないようにします。

x

q と同じように置換後の単語をクォートしますが、 空白文字 と改行文字のところで単語に分割します。

s/old/new/

イベント行で最初に現われた old を new に置き換えます。/ の代わりに任意の区切り文字を使うこともできます。 最後の区切り文字がイベント行の最後の文字ならば、 これは省略できます。 バックスラッシュ 1 つでクォートすれば、 old と new の中で区切り文字も使えます。new に & が含まれている場合には、 & は old に置き換えられます。バックスラッシュ 1 つを前に置けば & をクォートできます。 old が空文字列ならば、最後に置換された old が設定されます。 以前に履歴置換が全く行われていない場合には、現在に一番近い !?string[?] の検索で使われた string が設定されます。

&

直前の置換を繰り返します。

g

変更をイベント行全体に適用します。 これは ‘:s’ (例:‘:gs/old/new/’) や ‘:&’ と組み合わせて使われます。 ‘:s’ と一緒に使った場合には、 / の代わりに任意の区切り文字を使えます。 また、最後の区切り文字がイベント行の最後の文字ならば、 これは省略できます。 g の別名として a を使うこともできます。

G

この修飾子に続く ‘s’ 修飾子をイベント行の各単語に 1 回ずつ適用します。

1.31　シェルの組み込みコマンド

特に断らない限り、このセクションで説明されている組み込みコマンドのうち- で始まるオプションを受け付けるものは、オプションの終わりを表す -- も受け付けます。 組み込みコマンド :, true, false, test はオプションを持たず、-- を特別扱いしません。 組み込みコマンド exit, logout, break, continue, let, shift は、- で始まる引き数として受け取るのに、 -- を必要としません。 そのほかの組み込みコマンドは、 受け取ると明記されているオプション以外を引き数として受け取り、 - で始まる引き数を不正なオプションをとて解釈します。 この解釈を防ぐには -- が必要です。

1.31.1　: [arguments]

何もしません。このコマンドは arguments を展開し、指定されたリダイレクトを実行する以外には何も行いません。 終了コード 0 を返します。

1.31.2　. filename [arguments]

1.31.3　source filename [arguments]

filename からコマンドを読み込み、現在のシェル環境の下で実行します。 filename 内の最後に実行したコマンドの終了ステータスを返します。 filename にスラッシュが含まれていない場合、 filename は PATH に含まれるディレクトリから探されます。 PATH 内で検索されるファイルは、実行可能である必要はあり

ません。 bash が posix モード 以外で動作しているときは、 PATH 中でファイルを見つけられなかった場合に、 カレントディレクトリが検索されます。 組み込みコマンド shopt に対する sourcepath オプションが無効にされている場合、 PATH の検索は行われません。 何らかの arguments が指定された場合、これらの引き数は filename を実行したときの位置パラメータとなります。 指定されなかった場合は、位置パラメータは変更されません。 返却ステータスはスクリプト内で最後に実行したコマンドのステータスです (コマンドが全く実行されなければ 0 です)。 filename が見つからない場合や読み込めない場合には偽となります。

1.31.4　alias [-p] [name[=value] ...]

alias コマンドを引き数を付けずに (あるいは -p オプションを付けて) 実行すると、エイリアスのリストが 「alias name=value」の形で標準出力に出力されます。 引き数を与えた場合には、value を与えられた name それぞれに対するエイリアスが定義されます。 value の末尾に空白があると、エイリアスが展開されたときに、 空白の次の単語についてエイリアス置換があるかどうか調べられます。 引き数リスト中に value が与えられていない name があった場合は、 それぞれに対して名前とエイリアスの値が出力されます。 エイリアスが定義されていない name が指定された場合以外は、 alias は真を返します。

1.31.5　bg [jobspec ...]

サスペンドされているジョブ jobspec を バックグラウンドで実行再開します。このジョブは、初めから & を付けて起動されていたかのように動作を続けます。 jobspec がない場合には、 シェルが記録しているカレントジョブ (current job) が使われます。 bg jobspec は通常 0 を返しますが、ジョブ制御が無効であるときに実行した場合や、 ジョブ制御が有効であっても jobspec が有効なジョブを指定していない場合や jobspec がジョブ制御なしで開始したジョブを指定している場合は異なる値を返します。

1.31.6　bind [-m keymap] [-lpsvPSV]

1.31.7　bind [-m keymap] [-q function] [-u function] [-r keyseq]

1.31.8　bind [-m keymap] -f filename

1.31.9　bind [-m keymap] -x keyseq:shell-command

1.31.10　bind [-m keymap] keyseq:function-name

1.31.11　bind readline-command

readline の現在のキー割り当てと関数割り当てを表示したり、 キーシーケンスを readline の関数やマクロに割り当てたり、 readline の変数を設定したりします。 オプション以外の引き数はすべて、.inputrc , に書くのと同じ形式のコマンドですが、 それぞれの割り当てやコマンドは独立した引き数として渡さなければなりません。 例えば ’”\C-x\C-r”: re-read-init-file’ のように指定します。 オプションを指定した場合には、以下のような意味を持ちます:

-m keymap
　　キーマップ keymap を、以降の割り当てによって変更します。 指定できる keymap 名は、 emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, vi-command, vi-insert です。 vi は vi-command と同じです。 また emacs は emacs-standard と同じです。
-l　readline 関数の名前を全てリスト表示します。
-p　readline の関数の名前と割り当てを表示します。 表示は、再び読み込みできる形式で出力されます。
-P　readline の関数の現在の名前と割り当てをリスト表示します。
-s　マクロに割り当てられた readline のキーシーケンスと、 マクロが出力する文字列を表示します。 表示は、再び読み込みできる形式で出力されます。
-S　マクロに割り当てられた readline のキーシーケンスと、 マクロが出力する文字列を表示します。
-v　readline の変数名と値を表示します。 表示は、再び読み込みできる形式で出力されます。
-V　readline の現在の変数名と値をリスト表示します。
-f filename
　　キー割り当てを filename から読み込みます。
-q function
　　指定された function を呼び出すキーを問い合わせます。
-u function
　　指定された function に割り当てられているキーの割り当てを全て取り消します。
-r keyseq
　　keyseq に対する現在の割り当てを削除します。
-x keyseq:shell-command
　　keyseq が押されるたびに、 shell-command が実行されるようにします。 shell-command を実行するとき、シェルは変数 READLINE_LINE に readline の編集バッファの内容を設定し、変数 READLINE_POINT に現在の挿入ポイントの位置を設定します。 実行したコマンドが READLINE_LINE や READLINE_POINT の値を変更した場合、新しい値が編集の状態に反映されます。

認識できないオプションが与えられた場合やエラーが起きた場合以外は、 返り値は 0 になります。

1.31.12　break [n]

for, while, until, select のループから抜けます。n が指定されていれば、n レベル分 のループを break します。 n は 1 以上でなければなりません。 n がループの深さよりも大きい場合には、全てのループから抜けます。 n が 1 未満の場合を除けば、返り値は 0 です。

1.31.13　builtin shell-builtin [arguments]

指定されたシェル組み込みコマンドを実行します。コマンドには arguments を引き数として渡し、このコマンドの終了ステータスを返します。 これはシェル組み込みコマンドと同じ名前の関数を定義するときに便利で、 その関数内で組み込みコマンドの機能を使うことができます。 組み込みコマンド cd は普通、これを使って再定義されます。shell-builtin がシェル組み込みコマンドでなければ、終了ステータスは偽となります。

1.31.14　caller [expr]

実行中のサブルーチン (シェル関数や、組み込みコマンド . か source で呼び出したスクリプト) 呼び出しのコンテキストを返します。 expr が指定されていなければ、caller は 現在のサブルーチン呼び出しの行番号とソースファイル名を表示します expr に負でない整数が与えられた場合、caller は、 現在の呼び出しスタック中で指定した位置の行番号、サブルーチン名、 ソースファイルを表示します。 スタックトレースを表示する場合などに、範囲外の情報が使われるかもしれません。現在のフレームはフレーム 0 です。 シェルがサブルーチンを呼び出していない場合や、 expr が呼び出しスタックの有効な位置に相当しない場合を除けば、 返り値は 0 です。

1.31.15　cd [-L | [-P [-e]]] [dir]

カレントディレクトリを dir に変更します。 変数 HOME の値が dir のデフォルト値です。変数 CDPATH は、 dir を含むディレクトリの検索パスを定義します。 CDPATH 内では候補ディレクトリ名はコロン (:) で区切ります。 CDPATH 中に空のディレクトリ名がある場合、これはカレントディレクトリ (つまり “ . ”) を意味します。 dir がスラッシュ(/) で始まる場合には、 CDPATH は使われません。 -P オプションは、シンボリックリンクを辿らないで物理的なディレクトリ構造を 使うように指示します (組み込みコマンド set の -P オプションも参照)。 -L オプションを指定すると、シンボリックリンクを辿るようになります。 -e オプションを -P オプションと同時に指定すると、ディレクトリの変更が成功した後に カレントディレクトリが判定できない場合、cd は失敗のステータスを返します。 引き数に - を指定するのは、 $OLDPWD を指定するのと同じ意味です。 CDPATH 内の空以外のディレクトリ名が使われたときや、 - が最初の引き数のときに、ディレクトリの変更が成功すると、 新しいディレクトリの絶対パス名が標準出力に書かれます。 ディレクトリの変更が成功した場合には返り値は真になり、 そうでない場合には偽になります。

1.31.16　command [-pVv] command [arg ...]

command に引き数 args を付けて実行します。ただし、シェル関数の通常の参照は行いません。 組み込みコマンドと PATH 内で見つかるコマンドだけが実行されます。 -p オプションが与えられると、 command の検索を行う際に PATH のデフォルト値が使われます。これにより、全ての標準ユーティリティを確実に見つけられます。 -V オプションまたは -v オプションを与えると、 command の説明が出力されます。 -v オプションでは、 command を起動するときに使われるコマンドやファイル名を示す単語が表示されます。 -V ではさらに詳しい説明が表示されます。 -V オプションや -v オプションを与えた場合、終了ステータスは command が見つかれば 0 となり、見つからなければ 1 となります。 どちらのオプションも与えなかった場合に、エラーが起きたり、command を見つけられなかったりすると、終了ステータスは 127 になります。 それ以外の場合には、組み込みコマンド command の終了ステータスは、 command の終了ステータスです。

1.31.17　compgen [option] [word]

option に従って、word にマッチする補完候補のリストを生成します。オプションには complete 組み込みコマンドと同じものが指定できますが、 -p と -r は指定できません。 マッチのリストは標準出力に出力されます。 -F や -C オプションを使用したときは、 プログラム補完機能によって設定された様々なシェル変数は、 利用可能であっても有用な値を持ちません。

マッチのリストは、同じフラグによる補完仕様で プログラム補完のコードが直接生成したかのように 生成されます。 word が指定されると、word にマッチする補完だけが表示されます。

戻り値は、 無効なオプションが指定された場合やマッチが生成されなかった場合以外は 真になります。

1.31.18　complete [-abcdefgjksuv] [-o comp-option] [-DE] [-A action] [-G glob-pat] [-W wordlist] [-F function] [-C command] [-X filterpat] [-P prefix] [-S suffix] name [name ...]

1.31.19　complete -pr [-DE] [name ...]

各 name の引き数を、どのように補完するのかを指定します。 -p オプションが指定された場合や、 何もオプションが指定されなかった場合は、 現在の補完仕様が (入力として再利用できる形で) 出力されます。 -r オプションは、それぞれの name の補完指定を削除します。name が指定されなかった場合は全ての補完指定を削除します。 -D オプションは、残りのオプションと動作を“ デフォルトの ”コマンド補完 (補完が定義されていないコマンドに対して 試みられる補完) に適用することを示します。 -E オプションは、残りのオプションと動作を“ 空の ”コマンド補完 (空行に対して試みられる補完) に適用することを示します。

単語補完が試みられたときに、補完指定が適用される流れは、 前述のプログラム補完 で説明されています。

他のオプションは、指定された場合、以下のような意味を持ちます。-G, -W, -X オプションの引き数 (必要ならば -P と -S オプションの場合も) はクォートして、 組み込みコマンド complete が呼び出される前に展開されないようにすべきです。

-o comp-option

comp-option は補完仕様の動作をいくつかの観点から制御し、単純な補完生成以外ができるようにします。 comp-option には以下のどれかひとつを指定できます。

`bashdefault`
: 補完仕様がマッチを全く生成しなかった場合に、
`bash` のデフォルトの補完を用います。

`default`
: 補完仕様がマッチを全く生成しなかった場合に、
`readline` のデフォルトの補完を用います。

`dirnames`

補完仕様がマッチを全く生成しなかった場合に、
ディレクトリ名を補完しようとします。

filenames
: 補完仕様がファイル名を生成することを readline に伝え、 readline がファイル名特有の処理（ディレクトリ名にスラッシュを加えたり、特殊文字をクォートしたり、 末尾の空白を削除したり、など） を行えるようにします。　シェル関数と共に用いることを想定しています。

nospace
: 行末で単語を補完したときに空白を付け加える動作（デフォルトの動作）を やめるように readline に伝えます。

plusdirs
: 補完仕様で定義されたマッチした候補が生成された後に、　ディレクトリ名の補完を試み、ほかの動作の結果にマッチした候補が 全て追加されます。

-A action

action は補完候補リストの生成動作で、 以下のどれかひとつを指定します。

alias　エイリアス名。-a でも指定できます。

arrayvar
: 配列変数名。

binding readline キー割り当て名。

builtin シェル組み込みコマンド名。-b でも指定できます。

command コマンド名。-c でも指定できます。

directory
: ディレクトリ名。-d でも指定できます。

disabled
: 無効にされているシェル組み込みコマンドの名前。

enabled 有効にされているシェル組み込みコマンドの名前。

export　エクスポートされたシェル変数の名前。 -e でも指定できます。

file　ファイル名。-f でも指定できます。

function
: シェル関数の名前。

group　グループ名。-g でも指定できます。

helptopic
: 組み込みコマンド help に指定できるヘルプのトピック名。

hostname
: HOSTFILE　シェル変数で指定されたファイルから得ら

れたホスト名。

`job` ジョブ名（ジョブ制御が有効な場合)。`-j` でも指定できます。

`keyword` シェルの予約語。`-k` でも指定できます。

`running` ジョブ制御が有効であれば、実行中のジョブ名。

`service` サービス名。`-s` でも指定できます。

`setopt` 組み込みコマンド `set` の `-o` オプションで有効な引き数。

`shopt` 組み込みコマンド `shopt` に指定できるシェルオプション名。

`signal` シグナル名。

`stopped` 停止しているジョブ名（ジョブ制御が有効な場合)。

`user` ユーザ名。`-u` でも指定できます。

`variable`
全てのシェル変数名。`-v` でも指定できます。

-C command

command がサブシェル環境で実行され、 その出力が補完候補として使用されます。

-F function

シェル関数 function は現在のシェル環境で実行されます。関数が終了したときに、補完候補が COMPREPLY 配列変数から取得されます。

-G globpat

パス名展開パターン globpat を展開し、 補完候補のリストを生成します。

-P prefix

ほかの全てのオプションが適用された後で、 それぞれの補完候補の先頭に prefix が付け加えられます。

-S suffix

ほかの全てのオプションが適用された後で、 それぞれの補完候補の後に suffix が付け加えられます。

-W wordlist

wordlist は IFS 特殊変数に含まれる文字を区切り文字として分割され、 分割された単語がそれぞれ展開されます。 展開結果のリストのメンバのうち、補完中の単語がマッチするものが、 補完候補となります。

-X filterpat

filterpat がパス名展開のパターンとして使用されます。 先行するオプション・引き数によって生成された補完候補のリストに適用され、 filterpat とマッチするそれぞれの補完候補がリストから削除されます。 filterpat が！で始まる場合、パターンの否定の意味になります。 つまり、filterpat にマッチしない補完対象が削除されます。

不正なオプションが指定された場合、-p と -r 以外のオプションで name が指定されなかった場合、 存在しない name の指定によって補完が削除されようとした場合、 補完の指定の追加に失敗した場合、を除いては、返り値は真になります。

1.31.20　compopt [-o option] [-DE] [+o option] [name]

name の補完オプションを option に従って変更します。 name が指定されない場合は、実行中の補完が対象になります。 options が与えられない場合は、name または現在の補完の 補完オプションを表示します。 option に指定できる値は、前述の組み込みコマンド complete で有効なものです。 -D オプションは、残りのオプションを“ デフォルトの ”コマンド補完 (補完が定義されていないコマンドに対して 試みられる補完) に適用することを示します。 -E オプションは、残りのオプションを“ 空の ”コマンド補完 (空行に対して試みられる補完) に適用することを示します。

不正なオプションが指定された場合、 補完仕様が定義されていない name のオプションを変更しようとした場合、 出力エラーが起こった場合、を除いては、返り値は真になります。

1.31.21　continue [n]

for, while, until, select ループの次の繰り返し分から実行を継続します。 n を指定すると、深さを n 個分上がったループで実行を継続します。 n は 1 以上でなければなりません。 n がループの深さよりも大きい場合、 最後のループ (「トップレベル」のループ) で実行が継続されます。 n が 1 未満の場合を除けば、返り値は 0 です。

1.31.22　declare [-aAfFgilrtux] [-p] [name[=value] ...]

1.31.23　typeset [-aAfFgilrtux] [-p] [name[=value] ...]

変数を宣言したり、変数に属性を与えたりします。 name を指定しなければ、変数の値が表示されます。 -p オプションを指定すると、各 name の属性と値が表示されます。 -p に name 引き数を指定すると、他のオプションは無視されます。 -p オプションが name 引き数なしで指定されると、 他のオプションで指定された属性を持っている 全ての変数の属性と値を表示します。 -p 以外のオプションが指定されない場合、declare は全てのシェル変数の属性と値を表示します。 -f オプションは表示をシェル関数に限定します。 -F オプションを指定すると、関数定義の表示を止めます。 関数の名前と属性だけが出力されます。 shopt によってシェルオプション extdebug が有効になっていれば、 関数が定義されているソースファイルの名前と行番号も表示されます。 -F オプションを指定すると、 -f オプションも指定したことになります。 -g オプションを指定すると、declare がシェル関数の中で実行されたときであっても、 グローバルスコープで変数の作成、変更を行います。 ほかの場合には無視されます。 以下のオプションを使うと、 指定した属性を持つ変数の出力を限定したり、 変数に属性を与えたりできます:

`-a`　各 `name` は配列変数です（前述の 配列 を参照)。
`-A`　各 `name` は連想配列変数です（前述の 配列 を参照)。
`-f`　関数名だけを使います。
`-i`　変数を整数として扱います。変数に値が代入されたときに算術式評価（ 算術式評価 を参照）が実行されます。
`-l`　変数に値が代入されると、全ての大文字は小文字に変換されます。 大文字属性は無効になります。
`-r`　`name` を読み込み専用にします。　これ以降、代入文を用いて値を代入したり `unset` したりできなくなります。
`-t`　各　`name` に `trace` 属性を与えます。 `trace` 属性を付与された関数は `DEBUG` と `RETURN` のトラップを　呼び出したシェルから受け継ぎます `trace` 属性は変数には意味を持ちません。
`-u`　変数に値が代入されると、全ての小文字は大文字に変換されます。 小文字属性は無効になります。
`-x`　`name` に印を付け、　これ以降に実行するコマンドに環境経由でエクスポートします。

‘-’ ではなく ‘+’ を使うと属性を消します。 ただし例外として、+a を使って配列変数を破棄することはできず、 +r を使って読み込み専用属性を消すことはできません。 関数内で使った場合、 -g オプションが与えられなかった場合、 local コマンドを使った場合と同様に name はローカル変数となります。 変数名に =value が続く場合、 変数の値として value が設定されます。 返り値は基本的には 0 ですが、不正なオプションに出会った場合、 “-f foo=bar”, を使って関数を定義しようとした場合、 読み込み専用の変数に代入しようとした場合、 複合代入構文を使わずに配列変数に値を代入しようとした場合 (前述の配列 を参照)、name のいずれかが正しいシェル変数名でない場合、 読み込み専用変数の読み込み専用属性を無効にしようとした場合、 配列変数の配列属性を無効にしようとした場合、 存在しない関数を -f オプションで表示しようとした場合は除きます。

1.31.24　dirs [+n] [-n] [-clpv]

オプションがないときは、 現在記憶しているディレクトリのリストが表示されます。 デフォルトでは、全てのディレクトリ名は空白で区切って 1 行で表示されます。 ディレクトリは pushd コマンドによってリストに追加されます。 popd コマンドはリストからエントリを削除します。

`+n`　オプションなしで　`dirs`　を起動したときに表示されるリストの、 左から数えて `n` 番目のエントリを表示します。 エントリは `0` から始まります。
`-n`　オプションなしで　`dirs`　を起動したときに表示されるリストの、 右から数えて `n` 番目のエントリを表示します。 エントリは `0` から始まります。
`-c`　全てのエントリを削除し、ディレクトリスタックをクリアします。
`-l`　長い形式のリスト表示を行います。　デフォルトのリスト表示フォーマットでは、 チルダを使ってホームディレクトリを表します。
`-p`　`1` 行に `1` エントリの形でディレクトリスタックを出力します。

-v　1 行に 1 エントリの形でディレクトリスタックを出力します。
各エントリの前にはスタック内での番号が表示されます。

不正なオプションが与えられた場合とインデックス n がディレクトリスタックの末尾を越えている場合を除き、 返り値は 0 となります。

1.31.25　disown [-ar] [-h] [jobspec ...]

オプションなしの場合には、それぞれの jobspec がアクティブなジョブのテーブルから削除されます。jobspec がなく、かつ -a オプションも -r オプションも 与えられていない場合には、カレントジョブが使われます。 -h オプションが与えられている場合、どの jobspec もテーブルから削除されず、 シェルが SIGHUP を受け取ってもそのジョブには SIGHUP が送られないように印が付けられます。 jobspec が与えられていない場合、 -a オプションは全てのジョブを削除するか 全てのジョブに印を付けるという意味 となります。jobspec 引き数なしで -r オプションを指定すると、 実行中のジョブだけが操作の対象となります。 jobspec が有効なジョブを指定していない場合を除き、返り値は 0 となります。

1.31.26　echo [-neE] [arg ...]

arg を空白で区切って出力し、最後に改行を出力します。 終了ステータスは常に 0 です。 -n が指定された場合、最後の改行は出力されません。 -e オプションを指定した場合、 以下に示す、バックスラッシュのエスケープ文字が解釈されるようになります。 -E オプションを指定すると、 デフォルトでこのようなエスケープ文字が解釈されるシステムであっても、 エスケープ文字が解釈されないようになります。xpg_echo シェルオプションを用いると、 echo がこれらのエスケープ文字を展開するかどうかの デフォルト動作を動的に決定できます。echo は -- をオプションの終わりと解釈しません。 echo は以下のエスケープシーケンスを解釈します:

\a　警告（ベル）
\b　バックスペース
\c　行末に改行を付けない
\e
\E　エスケープ文字
\f　フォームフィード
\n　改行
\r　復帰（carriage return）
\t　水平タブ
\v　垂直タブ
\\　バックスラッシュ
\0nnn　8 進値で nnn である 8 ビット文字（8 進数で 0〜3 桁）。
\xHH　16 進値が HH である 8 ビット文字（16 進で 1〜2 桁）。
\uHHHH 16　進値が HHHH であるユニコード（ISO/IEC 10646）文字（16 進 1〜4 桁）。
\UHHHHHHHH
16 進値が HHHHHHHH であるユニコード（ISO/IEC 10646）　文字（16 進 1〜8 桁）。

1.31.27　enable [-a] [-dnps] [-f filename] [name ...]

組み込みコマンドの有効/無効を設定します。 シェルは通常はディスクコマンドの前に組み込みコマンドを探しますが、 組み込みコマンドを無効にすると、 シェルの組み込みコマンドと同じ名前を持つディスクコマンドを、 完全なパス名を指定しなくても実行できます。 -n を用いると、それぞれの name は無効となります。 それ以外の場合には、name は有効となります。 例えば、シェル組み込みのものでなく PATH 上にある test バイナリを使うには “enable -n test”. を実行します。 -f オプションは新しい組み込みコマンド name を共有オブジェクト filename からロードするという意味です。 これは動的ロードをサポートしているシステムで使えます。 -d オプションは、以前に-f オプションでロードした組み込みコマンドを削除します。 引き数 name が与えられなかった場合や、 -p オプションが与えられた場合、シェルの組み込みコマンドのリストが表示されます。 他にオプション引き数が指定されていない場合には、 有効になっているシェル組み込みコマンド全てからなるリストが表示されます。 -n を与えると、無効にされている組み込みコマンドだけが出力されます。 -a を与えると、それぞれ有効かどうかの表示付きで全ての組み込みコマンドが出力されます。 -s を与えると、出力されるのは POSIX の特殊組み込みコマンドだけに制限されます。 name がシェル組み込みコマンドでない場合と、 共有オブジェクトからの新しい組み込みコマンドのロードに失敗した場合を除き、 返り値は 0 となります。

1.31.28　eval [arg ...]

arg を読み込み、1 つのコマンドに結合し、 このコマンドを読み込んで実行します。 その終了ステータスが eval の値として返されます。args がない場合や空の引き数しかない場合には eval は 0 を返します。

1.31.29　exec [-cl] [-a name] [command [arguments]]

command が指定されていると、シェルはこのコマンドに置き換えられます。新しいプロセスは生成されません。 arguments は command に対する引き数となります。-l オプションを与えると、シェルは command に渡す 0 番目のオプションの先頭にダッシュを設定します。 これは login(1) が行う動作です。 -c オプションを与えると、 command は空の環境で実行されます。 -a を与えると、シェルは実行するコマンドに 0 番目の引き数として name を渡します。何らかの理由で command が実行できない場合には非対話的シェルは終了します。 ただしシェルオプション execfail が設定されている場合は終了せず、この場合には偽が返されます。 ファイルが実行できない場合には、対話的シェルは偽を返します。 command が指定されていない場合、任意のリダイレクトはカレントシェルで効果を持ち、 終了ステータスは 0 となります。リダイレクトのエラーが起きた場合には、終了ステータスは 1 となります。

1.31.30　exit [n]

ステータス n でシェルを終了させます。 n を省略すると、 終了ステータスは最後に実行したコマンドの終了ステータスとなります。シェルが終了する前には、 EXIT に対するトラップが実行されます。

1.31.31　export [-fn] [name[=word]] ...

1.31.32　export -p

指定された name には印が付けられ、 これ以降に実行するコマンドの環境に 自動的にエクスポートされるようになります。 -f オプションを与えると、 name は関数を参照します。 name を与えなかった場合

や、 -p オプションを与えた場合には、 このシェル内でエクスポートされている全ての名前のリストが出力されます。 -n オプションを与えると、指定した変数からエクスポート属性が取り除かれます。 変数名に=word が続くと、変数の値に word が設定されます。 不正なオプションがあった場合、 name のいずれかが不正なシェル変数名であった場合、 関数でない名前に対して -f オプションを与えた場合を除き、export は終了ステータス 0 を返します。

1.31.33　fc [-e ename] [-lnr] [first] [last]

1.31.34　fc -s [pat=rep] [cmd]

フィックスコマンド (Fix Command)。 最初の形式では、 first から last までの範囲のコマンドが履歴リストから選択されます。 first と last には文字列か数値を指定します。 文字列はその文字列で始まる最後のコマンドを表し、 数値は履歴リスト中でのインデックスを表します (負の値は現在のコマンド番号からのオフセットとして扱われます)。 last が指定されていない場合、リスト表示の場合には 現在のコマンドが last に設定され (したがって “fc -l -10” で最近のコマンド 10 個が出力されます)、それ以外の場合には first が設定されます。 first が指定されていなければ、編集の場合には前のコマンドが first に設定され、リスト表示の場合には -16 が設定されます。

-n オプションを与えるとリスト表示でコマンド番号が付きません。 -r オプションを与えるとコマンドの順序が逆になります。 -l オプションを与えると、コマンドは標準出力にリスト表示されます。 それ以外の場合には、これらのコマンドが書かれたファイルに対し、 ename で指定したエディタが起動されます。 ename が与えられていない場合は、変数 FCEDIT の値が使われ、 FCEDIT も設定されていない場合には EDITOR の値が使われます。 どちらの変数も設定されていなければ、vi が使われます。 編集が終了すると、編集後のコマンドがエコー表示され、実行されます。

2 番目の形式では、pat の部分をそれぞれ rep で置き換えてから command が再実行されます。 これを利用している便利なエイリアスの例として “r=”fc -s””, があります。 これを用いると “r cc” と入力すれば “cc” で始まる最も新しいコマンドを実行でき、 “r” と入力すれば直前のコマンドを再実行できます。

最初の形式を用いた場合、不正なオプションがあるか、 first または last が履歴行の範囲外を指定していなければ、返り値は 0 となります。 -e オプションが与えられた場合、 返り値は最後に実行されたコマンドの返り値となります。 ただし、コマンドの一時ファイルでエラーが起きた場合には返り値は偽 (失敗) となります。 2 番目の形式を用いた場合、 終了ステータスは再実行されたコマンドの終了ステータスとなります。 ただし、 cmd が有効な履歴行を指定していない場合は、 fc は偽 (失敗) を返します。

1.31.35　fg [jobspec]

jobspec の実行をフォアグラウンドで再開し、これをカレントジョブとします。 jobspec がない場合、シェルが記録しているカレントジョブが使われます。 返り値はフォアグラウンドで再開されたコマンドの返り値ですが、 ジョブ制御が無効であるときに実行した場合や、 ジョブ制御が有効であっても jobspec が有効なジョブを指定していない場合や jobspec がジョブ制御なしで実行したジョブを指定している場合には偽となります。

1.31.36　getopts optstring name [args]

getopts は位置パラメータを解釈するシェルの処理で使います。optstring は識別の対象であるオプション文字列です。 ある文字の後にコロンがある場合、 そのオプションは引き数を一つ取ることが期待されます。引き数は空白でオプション文字と区切られていなければなりません。 コロンと疑問符はオプション文字として使えません。 呼び出されるたびに、 getopts は次に見つかったオプションをシェル変数 name に格納し (name が存在しなければ初期化を行います)、 次に処理される引き数のインデックスを変数 OPTIND に格納します。 OPTIND はシェルまたはシェルスクリプトが呼び出されるたびに 1 に初期化されます。 オプションが引き数を必要とする場合には、 getopts はその引き数を変数 OPTARG に格納します。 シェルが OPTIND を自動的に再設定することはありません。 1 つのシェルが呼び出されている間に別のパラメータの組み合わせを使う場合には、 再度 getopts を呼び出す前に手動で OPTIND の再設定を行わなければなりません。

オプションの終わりに到達すると、 getopts は 0 より大きい返り値で終了します。 OPTIND にはオプションでない最初の引き数のインデックスが設定され、 name には ? が設定されます。

getopts は通常位置パラメータを展開しますが、他の引き数が args に指定されている場合には、 getopts は位置パラメータでなくこれらの引き数を解釈対象とします。

getopts は 2 通りの方法でエラーを報告します。 optstring の最初の文字がコロンならば、 静かな (silent) エラー報告が行われます。 通常の動作では、不正なオプションがある場合や オプションの引き数が足りない場合に診断メッセージが出力されます。 変数 OPTERR に 0 が設定されている場合、 optstring の最初の文字がコロンでない場合であっても、 エラーメッセージは全く出力されません。

不正なオプションがあった場合、 getopts は ? を name に設定します。 さらに、静かなモードでない場合にはエラーメッセージが出力され、 OPTARG が削除されます。 getopts が静かなモードであれば、見つかったオプション文字は OPTARG に設定され、診断メッセージは出力されません。

必要な引き数が見つからず、かつ getopts が静かなモードでない場合には、疑問符 (?) が name に設定され、 OPTARG が削除され、診断メッセージが出力されます。 getopts が静かなモードならば、コロン (:) が name に設定され、 OPTARG には見つかったオプション文字が設定されます。

(指定の有無に関係なく) オプションが見つかった場合、 getopts は真を返します。 オプションの最後に到達した場合や、エラーが起きた場合には、 偽を返します。

1.31.37　hash [-lr] [-p filename] [-dt] [name]

hash が実行される度に、 $PATH に含まれるディレクトリの検索を行ってコマンド name の完全なパス名を調べ、 その結果を記憶します。 それまでに記憶されていたパス名は捨てられます。 -p オプションが指定されると、パス検索は実行されず、 filename がそのコマンドの完全なファイル名として使われます。 -r オプションを与えると、シェルは記憶しているパス名を全て忘れます。 -d オプションを与えると、シェルは各 name について記憶しているパス名を忘れます。 -t オプションを与えると、name に対応する完全なファイル名が表示されます。 -t に複数の name 引き数が指定された場合、 記憶されている完全なファイル名の前に name が表示されます。 -l オプションを与えると、入力として再利用できる形で出力されます。

引き数が与えられていない場合や、-l だけが与えられた場合は、 記憶しているコマンドに関する情報が出力されます。 name が見つからない場合と不正なオプションが与えられた場合を除き、 返却ステータスは真となります。

1.31.38　help [-dms] [pattern]

組み込みコマンドのヘルプ情報を表示します。 pattern が指定された場合には、 help は pattern にマッチする全てのコマンドに関する詳しいヘルプを出力します。 指定されなかった場合には、 全ての組み込みコマンドと制御構造についての説明が出力されます。

`-d` `pattern` それぞれの短い説明を表示します。
`-m` `pattern` それぞれの説明を `manpage` 風のフォーマットで表示します。
`-s` `pattern` それぞれの短い書式のみを表示します。

pattern にマッチするコマンドが全くない場合を除き、返却ステータスは 0 です。

1.31.39　history [n]

1.31.40　history -c

1.31.41　history -d offset

1.31.42　history -anrw [filename]

1.31.43　history -p arg [arg ...]

1.31.44　history -s arg [arg ...]

オプションがない場合には、行番号付きでコマンド履歴を表示します。* 付きでリスト表示されている行は変更された行です。 引き数 n を指定すると、最新の n 行だけがリスト表示されます。 シェル変数 HISTTIMEFORMAT に空でない値が設定されると、履歴エントリを表示するときに タイムスタンプを表示するための strftime(3) の書式文字列として使われます。 タイムスタンプと履歴行の間には空白は表示されません。 filename が指定されている場合、 履歴ファイルの名前として使われます。 指定されていない場合には HISTFILE の値が使われます。指定された場合、オプションは以下の意味を持ちます:

`-c` 全てのエントリを削除し、履歴リストをクリアします。
`-d offset`
`offset` 番目にある履歴エントリを削除します。
`-a` 「新しい」履歴行 (`bash` の現在のセッションの開始以降に入力された履歴行) を履歴ファイルに追加します。
`-n` まだ履歴ファイルから読み込んでいない履歴行を 現在の履歴リストに読み込みます。 読み込まれるのは、`bash` の現在のセッションの開始以降に 履歴ファイルに追加された行です。
`-r` 履歴ファイルの内容を読み込み、これらを現在の履歴として用います。
`-w` 現在の履歴を履歴ファイルに書き込みます。履歴ファイルの内

容は上書きされます。

-p　後に続く args に対して履歴置換を行い、 その結果を標準出力に表示します。　この結果は履歴リストには格納されません。 通常の履歴展開が行われないようにするため、 arg はそれぞれクォートしなければなりません。

-s　args を 1 つのエントリとして履歴リストに格納します。 履歴リストの最後のコマンドは、 args が追加される前に削除されます。

HISTTIMEFORMAT 変数が設定されていると、履歴エントリのタイムスタンプの情報は、 履歴のコメント文字を付けて履歴ファイルに書き込まれます。 履歴ファイルを読み込むときに、履歴のコメント文字で始まり直後に数字が続く行は、 直前の履歴行のタイムスタンプとして解釈されます。 不正なオプションがある場合、 履歴ファイルの読み書き時にエラーが起きた場合、 -d オプションの引き数として不正な offset の値が与えられた場合、 -p オプションの引き数として与えられた履歴展開が失敗した場合を除き、 返り値は 0 になります。

1.31.45　jobs [-lnprs] [jobspec ...]

1.31.46　jobs -x command [args ...]

最初の形式を実行すると、アクティブなジョブがリスト表示されます。オプションは以下の意味を持ちます:

-l　通常の情報に加えて、プロセス ID をリスト表示します。

-n　ユーザがステータスを最後に通知されて以降に、 ステータスの変更があったジョブに関する情報だけを表示します。

-p　そのジョブが属するプロセスグループのリーダーのプロセス ID だけを表示します。

-r　実行中のジョブだけを出力します。

-s　停止中のジョブだけを出力します。

jobspec が指定された場合、 そのジョブに関する情報だけが出力されます。 不正なオプションがある場合や、不正な jobspec が与えられた場合を除き、返却ステータスは 0 です。

-x オプションが指定されると、 jobs は command や args 中の jobspec を対応するプロセスのグループ ID で置き換え、 command に args を与えて実行し、その終了ステータスを返します。

1.31.47　kill [-s sigspec | -n signum | -sigspec] [pid | jobspec] ...

1.31.48　kill -l [sigspec | exit_status]

sigspec または signum で指定されたシグナルを、 pid または jobspec で指定されたプロセスに送ります。 sigspec は、 SIGKILL のようなシグナル名 (先頭の SIG は省略可能)、またはシグナルの番号です。 signum はシグナルの番号です。 sigspec が存在しない場合には、 SIGTERM が指定されたものとします。 引き数に -l を与えるとシグナル名がリスト表示されます。 -l と同時に引き数を与えると、 引き数に対応するシグナルの名前がリスト表示され、 返却ステータスは 0 となります。 -l に対する引き数 exit_status は、

シグナル番号、またはシグナルによって終了させられた プロセスの終了ステータスを指定する数です。 少なくとも 1 つのシグナルを正常に送れた場合、 kill は真を返します。 エラーが起きた場合や不正なオプションがあった場合には、 kill は偽を返します。

1.31.49　let arg [arg ...]

各 arg は評価を行う算術式です (算術式展開 を参照)。 最後の arg を評価した結果が 0 であれば、 let は 1 を返します。 それ以外の場合には 0 が返されます。

1.31.50　local [option] [name[=value] ...]

それぞれの引き数に対して name という名前のローカル変数が生成され、 value が代入されます。 option には、 declare コマンドに使えるオプションが全て使えます。 関数内で local を使った場合、この変数 name の可視スコープは、この関数とこの関数の子に制限されます。オペランドがない場合、 local はローカル変数の一覧を標準出力に出力します。 関数の内部以外で local を使うとエラーになります。local が関数の外部で使われた場合、 不正な name が与えられた場合、 name が読み取り専用の変数であった場合以外は、local の返却ステータスは 0 となります。

logout ログインシェルを終了します。

1.31.51　mapfile [-n count] [-O origin] [-s count] [-t] [-u fd] [-C callback] [-c quantum] [array]

1.31.52　readarray [-n count] [-O origin] [-s count] [-t] [-u fd] [-C callback] [-c quantum] [array]

標準入力の各行を配列変数 array に読み込みます。 -u オプションが指定されたときは、ファイル・ディスクリプター fd から読み込みます。 変数 MAPFILE がデフォルトの array です。 オプションが指定された場合、以下の意味を持ちます:

`-n`　最大 `count` 行をコピーします。`count` が `0` であれば、全ての行をコピーします。

`-O`　`array` のインデックス `origin` から代入を始めます。デフォルトのインデックスは `0` です。

`-s`　最初の `count` 行を読み捨てます。

`-t`　読み込んだ各行の末尾にある改行文字を削除します。

`-u`　標準入力の代わりにファイル・ディスクリプター `fd` から行を読み込みます。

`-C`　`quantum` 行が読み込まれるごとに `callback` を評価します。`-c` オプションで `quantum` を指定します。

`-c`　`callback` を呼び出す間隔の行数を指定します。

-c なしで -C が指定されたとき、デフォルトの間隔は 5000 です。callback が評価されるとき、 次に代入される配列要素のインデックスと、 その要素に代入される行が、引き数として与えられます。callback は、

行が読み込まれてから 配列の要素に代入されるまでの間に評価されます。

始点が明示的に指定されない場合、mapfile は値を代入する前に array をクリアします。

mapfile は、 不正なオプションやオプション引き数が指定された場合や、 array が不正か代入できない場合や、 array がインデックスによる配列でない場合でなければ、 成功の状態を返します。

1.31.53　popd [-n] [+n] [-n]

ディレクトリスタックからエントリを削除します。 引き数がない場合には、スタック先頭のディレクトリが削除され、 新しく先頭となったディレクトリへの cd が実行されます。 引き数が指定された場合には、これは以下の意味を持ちます:

`-n`　スタックからディレクトリを削除する際に、 通常のディレクトリ変更を行いません。 したがって、スタックだけが操作されます。

`+n`　`dirs` で表示されるリストの左から数えて `n` 番目のエントリを削除します。 エントリは `0` から数えます。例えば、 ‘‘`popd +0`’’ は最初のディレクトリを削除し、 ‘‘`popd +1`’’ は `2` 番目のディレクトリを削除します。

`-n`　`dirs` で表示されるリストの右から数えて `n` 番目のエントリを削除します。 エントリは `0` から数えます。例えば、 ‘‘`popd -0`’’ は最後のディレクトリを削除し、 ‘‘`popd -1`’’ は最後の `1` つ前のディレクトリを削除します。

popd コマンドが成功した場合、 dirs も実行され、返却ステータスは 0 となります。 popd が偽を返すのは、不正なオプションがあった場合、 ディレクトリスタックが空の場合、 ディレクトリスタックの存在しないエントリが指定された場合、 ディレクトリ変更に失敗した場合です。

1.31.54　printf [-v var] format [arguments]

arguments を整形して標準出力に書き出します。 フォーマットは format で制御します。 -v オプションが与えられると、標準出力に書き出す代わりに、 変数 var に代入されます。

format は 3 つのタイプのオブジェクトを含む文字列です。 3 つのオブジェクトとは、普通の文字 (そのまま標準出力にコピーされる)、 文字エスケープシーケンス (変換されて標準出力にコピーされる)、 表示フォーマット指定 (format の後に続く各引き数 argument の表示を行う)、です。 printf(1) 標準のフォーマット指定以外に、以下の拡張フォーマットが使えます。

`%b`　`printf` は対応する `argument` 中の バックスラッシュのエスケープシーケンスを展開します。 ただし、`\c` で出力を終了し、`\'`, `\"`, `\?` のバックスラッシュは削られず、 `\0` で始まる `8` 進数のエスケープシーケンスは `4` 桁までです。

`%q`　`printf` は対応する `argument` をシェルの入力として再利用できるフォーマットで出力します。

%(datefmt)T
printf は datefmt を strftime(3) の書式文字列として使って日付と時刻の文字列を出力します。 対応する argument は、紀元 (1970 年 1 月 1 日 00:00:00 UTC) からの秒数を表す整数値です。argument には 2 つの特別な値が使えます。 -1 は現在時刻を表します。-2 はシェルが起動した時刻を表します。

文字列でないフォーマット指定に対する引き数は、C の定数として扱われます。 ただし、先頭のプラスとマイナスの記号は許されます。 また、先頭の文字がシングルクォートやダブルクォートであれば、 続く文字の ASCII コードの値が引き数の値となります。

format は必要に応じて再利用され、全ての arguments を処理します。与えられたよりも多くの arguments を format が必要とする場合、余分のフォーマット指定は、 0 と空文字列のうち、適切な方が指定されたかのように動作します。 成功した場合の返り値は 0 で、失敗した場合の返り値は 0 以外です。

1.31.55 pushd [-n] [+n] [-n]

1.31.56 pushd [-n] [dir]

ディレクトリをディレクトリのスタックに追加するか、 スタックをローテートさせます。 このとき、新しいスタックの最も上にあるものを カレントの作業ディレクトリにします。 引き数を与えなければ、一番上の 2 つのディレクトリを交換し、0 を返します。 ただし、ディレクトリスタックが空の場合を除きます。引き数を与えた場合には、以下の意味を持ちます:

-n　ディレクトリをスタックに追加したときに、 通常のディレクトリ変更を行いません。 したがって、スタックだけが操作されます。

+n　スタックをローテートさせ、n 番目のディレクトリを一番上にします。 このとき dirs が表示するリストは左から数え始め、その左端は 0 となります。

-n　スタックをローテートさせ、n 番目のディレクトリを一番上にします。 このとき dirs が表示するリストは右から数え始め、その右端は 0 となります。

dir　dir をディレクトリスタックの一番上に追加し、 そのディレクトリを新しいカレントの作業ディレクトリにします。

pushd コマンドが成功すると、 dirs コマンドも実行されます。 2 番目の形式を使った場合、 dir への cd が失敗しなければ、 pushd は 0 を返します。 最初の形式を使った場合には、 ディレクトリスタックが空の場合、 ディレクトリスタックの存在しない要素が指定された場合、 指定された新しいカレントディレクトリへの ディレクトリ変更が失敗した場合以外は、 pushd は基本的には 0 を返します。

1.31.57 pwd [-LP]

現在の作業ディレクトリの絶対パス名を出力します。 -P オプションが指定された場合や、組み込みコマンド set の -o physical オプションが有効になっている場合には、 シンボリックリンクを展開したパス名が

出力されます。 -L オプションを使うと、 出力されるパス名にはシンボリックリンクが含まれているかもしれません。 カレントディレクトリの名前を読む際にエラーが起きたり、 不正なオプションが指定されたりしなければ、返却ステータスは 0 となります。

1.31.58 read [-ers] [-a aname] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...]

標準入力、 または -u オプションの引き数として指定されたファイル・ディスクリプター fd から 1 行を読み込み、最初の単語を最初の name に代入し、2 番目の単語を 2 番目の name に代入します。以降も同様です。 余った単語とそれらの間の区切り文字は、最後の name に代入されます。 name よりも標準入力から読み込んだ単語の方が少ない場合には、 余っている name には空文字列が値として代入されます。IFS 中の文字が、行を単語に分割するために使われます。 バックスラッシュ文字 (\) を使うと、 次に読み込んだ文字の特殊な意味を消したり、行を連結したりできます。 オプションが与えられていれば、以下の意味を持ちます:

`-a aname`
: 単語を配列変数 `aname` にインデックス順に代入します。インデックスは `0` から始まります。 新しい値が代入される前に `aname` は削除されます。他の `name` 引き数は無視されます。

`-d delim`
: 改行ではなく、`delim` の最初の文字が、入力行を終了するために使われます。

`-e`
: 標準入力を端末から読み込む場合、 `readline` (前述の `READLINE` ライブラリ のセクションを参照) を使って行を取得します。 `readline` は現在の (行編集がそれまで有効になっていなければデフォルトの) 編集設定を使います。

`-i text`
: 行を取得するのに `readline` が使われるとき、入力を開始する前に編集バッファに `text` が置かれます。

`-n nchars`
: 組み込みコマンド `read` は、 入力行全体が読み込まれるのを待たず、 `nchars` 文字を読み込んだ時点で戻ります。 ただし、`nchars` 文字が読み込まれる前に区切り文字が現われたときは、 区切り文字を優先します。

`-N nchars`
: 入力行全体が読み込まれるのを待たず、 `nchars` 文字を読み込んだ時点で戻ります。 ただし、入力に `EOF` が現れたときや、`read` がタイムアウトになったときを除きます。 入力に区切り文字が現れても特別扱いはせず、 `nchars` 文字を読み込むまでは `read` が戻ることはありません。

`-p prompt`
: 入力を読み込もうとする前に標準エラー出力に `prompt` を表示します。 末尾に改行は付きません。 プロンプトが表示されるのは、入力を端末から読み込む場合だけです。

`-r`　バックスラッシュはエスケープ文字として作用しません。 バックスラッシュは行の一部とみなされます。 特に、バックスラッシュと改行の組み合わせを使って 複数の行を接続することはできません。

`-s`　静かな （silent）モード。端末に入力が行われても、文字はエコーされません。

`-t timeout`
入力行全体が `timeout` 秒以内で読み込まれない場合、 `read` をタイムアウトさせて、失敗の状態を返します。 `timeout` は小数部を持つ十進数でもかまいません。 このオプションは、 `read` が入力を端末やパイプ、その他の特殊ファイルから 読み込んでいる場合にのみ効果があります。 通常のファイルから読み込んでいる場合には影響ありません。 `timeout` が `0` のときは、`read` は指定したファイル・ディスクリプター から読み込める状態であれば成功の状態を返し、そうでなければ失敗の状態を返します。 タイムアウト時間を超えた場合、終了ステータスは `128` より大きい値になります。

`-u fd`　ファイル・ディスクリプター `fd` から入力を読み込みます。

name が全く与えられていない場合、読み込まれた行は変数 REPLY に代入されます。 ファイル末尾に到達した場合、 read がタイムアウトした場合 (この場合には終了ステータスは 128 より大きい値になります)、 -u の引き数として不正なファイル・ディスクリプターが指定した場合以外は、 終了コードは 0 です。

1.31.59　readonly [-aAf] [-p] [name[=word] ...]

指定された name に読み込み専用の印を付けます。 それ以降は、これらの name の値を変更することはできません。 -f オプションを与えた場合、name に対応する関数に同様の印が付きます。 -a オプションを与えると、配列変数だけが対象となります。 -A オプションを与えると、連想配列変数だけが対象となります。 両方のオプションを与えると、 -A が優先されます。 name 引き数が全く与えられてない場合、または -p オプションが与えられた場合、読み込み専用の名前全ての一覧が出力されます。 ほかのオプションを合わせて指定すると、出力を読み込み専用の名前の一部に限定します。 -p オプションを使うと、 入力として再利用できるようなフォーマットで出力が行われます。 変数名に =word が続くと、変数の値に word が設定されます。 返却ステータスは基本的に 0 ですが、 無効なオプションがあった場合、 name のいずれかが有効なシェル変数名でなかった場合、 -f オプションの際に関数でない name を与えた場合は除きます。

1.31.60　return [n]

指定した返り値 n で関数を終了させます。 n を省略すると、返却ステータスは 関数内で最後に実行したコマンドの返却ステータスになります。 関数の外側で使われているが、 . (source) コマンドによるスクリプトの実行中である場合、 シェルはそのスクリプトの実行を止め、 n またはスクリプト内で最後に実行されたコマンドの終了ステータスを スクリプトの終了ステータスとして返します。 関数の外側で . によるスクリプトの実行中以外に使われた場合、 返却ステータスは偽となります。 関数やスクリプトから実行が戻る前に、 RETURN トラップ に設定されたコマンドが実行されます。

1.31.61　set [--abefhkmnptuvxBCEHPT] [-o option-name] [arg ...]

1.31.62　set [+abefhkmnptuvxBCEHPT] [+o option-name] [arg ...]

オプションなしの場合は、シェル変数全ての名前と値の組が表示されます。 表示は、現在設定されている変数を設定や再設定をする入力 として再利用できるフォーマットで行われます。 読み込み専用の変数は再設定できません。 posix モード ではシェル変数だけが表示されます。出力は現在のロケールに従ってソートされます。 オプションが指定されている場合、 オプションはシェルの属性を設定または解除します。オプションが処理された後に残っている引き数があれば、 これは位置パラメータの値として扱われ、 $1, $2, ... $n の順に代入されます。オプションが指定されていれば、以下の意味を持ちます:

`-a`　値を変更したり新規に設定したりした変数および関数が、（これ以降に実行するコマンドの） 環境として 自動的にエクスポートされます。

`-b`　終了したバックグラウンドジョブのステータス報告を、 次のプライマリプロンプトの前ではなく、即座に行います。 これはジョブ制御が有効な場合に限り有効です。

`-e`　パイプライン（1 つの 単純なコマンド からなるものでもよい)、 括弧で囲まれた サブシェル のコマンド、 ブレース (前述の シェルの文法 を参照） で囲まれたコマンドのリストの一部として実行されたコマンドの 1 つ が 0 でないステータスで終了した場合、即座に終了します。 ただし、失敗したコマンドが、キーワード while または until の直後のコマンドの一部である場合、予約語 if または elif に続く条件式の一部である場合、 && または || によるコマンドのリストの一部である場合（最後の && や || の後のコマンドを除く)、 パイプラインの中の最後のコマンド以外である、 コマンドの返り値が ! で反転されている場合、のいずれかであれば、シェルは終了しません。 ERR に対するトラップが設定されていれば、シェルが終了する前に実行されます。 このオプションはシェルの環境と各サブシェルの環境に別々に適用され（前述のコマンド実行環境 を参照)、 サブシェルはサブシェル内の全てのコマンドを実行する前に終了するかもしれません。

`-f`　パス名展開を無効にします。

`-h`　実行時に参照できるようにコマンドの位置を記憶します。 これはデフォルトで有効になっています。

`-k`　コマンド名の前にある代入文だけでなく、 引数として指定された全て代入文も、 そのコマンドに対する環境変数に追加されます。

`-m`　監視モード。ジョブ制御は有効になります。ジョブ制御（前述の ジョブ制御 セクションを参照）をサポートしているシステム上の対話的シェルでは、 このオプションはデフォルトで有効です。 別のプロセスグループで実行されたバックグラウンドプロセスと、 これらの終了ステータスが書かれた行が、プロセスの終了時に表示されます。

-n　コマンドを読み込みますが実行はしません。　これを使うとシェルスクリプトの文法エラーをチェックできます。　このオプションは対話的シェルでは無視されます。

-o option-name

option-name には、以下のいずれかを指定できます:

allexport
　-a と同じです。

braceexpand
　-B と同じです。

emacs　emacs 形式のコマンド行編集インタフェースを使います。　これはシェルが対話的な場合には、デフォルトで有効です。 ただし、 --noediting オプション付きでシェルを実行した場合は除きます。　これは、read -e での編集インタフェースにも影響します。

errexit -e と同じです。

errtrace
　-E と同じです。

functrace
　-T と同じです。

hashall -h と同じです。

histexpand
　-H と同じです。

history コマンド履歴を有効にします。コマンド履歴については　履歴　セクションで説明しています。　このオプションは、対話的シェルではデフォルトで有効です。

ignoreeof
　シェルコマンドの ‘‘IGNOREEOF=10’’ を実行した場合と同じ効果を持ちます （前述の　シェル変数　を参照)。

keyword -k と同じです。

monitor -m と同じです。

noclobber
　-C と同じです。

noexec　-n と同じです。

noglob　-f と同じです。

nolog　現在は無視されます。

notify　-b と同じです。

nounset -u と同じです。

onecmd　-t と同じです。

physical
　-P と同じです。

pipefail
　設定されている場合、パイプラインの返り値は、　0 以外のステータスで終了した最後の（一番右の）コマ

ンドの値になります。　パイプラインの全てのコマンドが成功の状態で終了すると 0 になります。 このオプションは、デフォルトで無効です。

posix　bash の動作のうち、デフォルトの振舞いが POSIX 標準と異なる部分を、POSIX 標準に準拠するように変更します (posix モード)。

privileged
　-p と同じです。

verbose -v と同じです。

vi　vi　形式のコマンド行編集インタフェースを使います。 これは、read -e　での編集インタフェースにも影響します。

xtrace　-x と同じです。

option-name なしで -o オプションを与えた場合、現在のオプションが出力されます。 option-name なしで +o　オプションを与えた場合、現在のオプション設定を再生成する　set コマンドの列が標準出力に出力されます。

-p　特権 (privileged) モードを有効にします。　このモードでは $ENV と $BASH_ENV ファイルは処理されず、シェル関数は環境から継承されず、 SHELLOPTS, BASHOPTS, CDPATH, GLOBIGNORE 環境変数は定義されていても無視されます。　実ユーザ (グループ) ID と異なる実効ユーザ (グループ) ID でシェルが起動され、 かつ -p オプションが与えられていない場合、 これらの動作が行われ、実効ユーザ ID には実ユーザ ID が設定されます。　起動時に　-p オプションが与えられた場合、 実効ユーザ ID は再設定されません。　このオプションを無効にすると、　実効ユーザ　ID と実効グループ ID には、 実ユーザ ID と 実グループ ID が設定されます。

-t　コマンドを 1 つ読み込み、実行してから終了します。

-u　パラメータ展開の実行中に、特殊パラメータ "@" と "*" 以外で　設定されていない変数やパラメータをエラーとして扱います。　設定されていない変数やパラメータを展開しようとした場合、 シェルはエラーメッセージを出力します。 シェルが対話的でなければ、0 でないステータスで終了します。

-v　シェルの入力行を、読み込んだ際に表示します。

-x　単純なコマンド、for コマンド、case コマンド、 select　コマンド、算術 for コマンドをそれぞれ展開した後、 PS4 を展開した値を表示し、その後にそのコマンドと展開した引き数や、 結び付いた単語のリストを表示します。

-B　シェルはブレース展開 (前述の ブレース展開 を参照) を実行します。これはデフォルトで有効です。

-C　設定されている場合、 bash はリダイレクト演算子 >, >&, <> で既存のファイルを上書きしません。　上書きができるのは、リダイレクト演算子 >|　を　>　の代わりに使ったときで

す。

-E　設定されている場合、ERR　のトラップは、 シェル関数、コマンド置換、サブシェル環境で実行されるコマンド　に継承されます。 通常、このような場面では ERR のトラップは継承されません。

-H　! 形式の履歴置換を有効にします。　このオプションは、シェルが対話的なときにはデフォルトで有効です。

-P　設定されている場合、 cd のような現在の作業ディレクトリを変更するコマンドを実行するときに、　シェルはシンボリックリンクを辿りません。　代わりに物理的ディレクトリ構造が使われます。 デフォルトでは、 bash　がカレントディレクトリを変更するコマンドを実行する際には、　ディレクトリの論理的な接続が辿られます。

-T　設定されている場合、DEBUG と RETURN のトラップは、　シェル関数、コマンド置換、サブシェル環境で実行されるコマンドに継承されます。　通常、このような場面では　DEBUG　と RETURN のトラップは継承されません。

--　このオプションの後に引き数が続いていない場合には、　位置パラメータの設定が取り消されます。　引き数が続いている場合には、位置パラメータに　arg　が設定されます。 arg に - で始まるものが含まれていても、　オプションではなく位置パラメータとして扱われます。

-　オプションの終わりを示します。 残りの arg は全て位置パラメータに代入されます。 -x オプションと -v オプションは無効になります。　arg がない場合には、位置パラメータの内容は変化しません。

特に断らない限り、各オプションはデフォルトで無効になっています。- の代わりに + を使うと、これらのオプションは無効になります。 オプションはシェルを起動する際の引き数としても指定できます。 現在のオプションの集合は、 $- で知ることができます。 不正なオプションが無ければ、終了ステータスは必ず真となります。

1.31.63　shift [n]

n+1 ... からの位置パラメータの名前を変え、 $1 ... とします。 $# から $#-n+1 までの数字で表されるパラメータは unset されます。 n は 0 以上 $# 以下の数でなければなりません。 n が 0 ならば、どのパラメータも変更されません。 n が与えられない場合には、1 が指定されたものとみなされます。 n が $# より大きい場合、位置パラメータは変化しません。 n が $# より大きい場合や 0 より小さい場合には、 返却ステータスは 0 より大きい数になります。 それ以外の場合には 0 になります。

1.31.64　shopt [-pqsu] [-o] [optname ...]

シェルのオプション動作を制御する変数の値をトグルさせます。 オプションがない場合や、 -p オプションが指定されている場合には、 設定可能なオプション全てのリストが表示されます。 表示の際には、それ

ぞれが設定されているかどうかも示されます。 オプションの表示は、入力として再利用できるフォーマットで行われます。 その他のオプションは、以下の意味を持っています:

`-s` `optname` をそれぞれ有効にします（設定します)。
`-u` `optname` をそれぞれ無効にします（設定解除します)。
`-q` 通常の出力を止めます （静かなモード)。 返却ステータスは `optname` が設定されているかどうかを示します。 複数の `optname` 引き数と `-q` が指定されている場合には、 全ての `optnames` が有効であるときに返却ステータスが `0` となります。 それ以外のときには、`0` でない値となります。
`-o` `optname` の値を、組み込みコマンド `set` の `-o` オプションで定義されているものに制限します。

引き数 optname なしで -s オプションまたは -u オプションを使った場合、それぞれ設定されているもの、または設定されていないものだけに表示が限定されます。 特に断らない限り、shopt オプションは デフォルトで無効 (設定解除) になっています。

オプションをリスト表示したときの返却ステータスは、 全ての optnames が有効になっている場合は 0 となります。 それ以外の場合には 0 でない値となります。 設定または設定取り消しのオプションのときには、 optname が不正なシェルオプションでなければ、 返却ステータスは 0 となります。

shopt オプションのリストを以下に示します:

autocd

設定されている場合、ディレクトリの名前のコマンド名は、それが cd コマンドの引き数に指定されたものとして実行されます。 このオプションが使われるのは対話的シェルだけです。

cdable_vars

設定されている場合、組み込みコマンド cd への引き数でディレクトリでないものは変数の名前とみなされ、 その値が変更先のディレクトリとなります。

cdspell

設定されている場合、 cd コマンドのディレクトリ要素におけるスペルのちょっとした誤りは修正されます。 チェックされる誤りは、文字の入れ替わり・文字の欠け・ 1 文字余分にあることです。 訂正できた場合には、訂正後のファイル名が表示され、 コマンドは続けて実行されます。 このオプションが使われるのは対話的シェルだけです。

checkhash

設定されている場合、 bash はハッシュ表で見つけたコマンドを実行する前に 実際に存在するかどうかをチェックします。ハッシュされているコマンドが既になくなっている場合、 通常のパス検索が行われます。

checkjobs

設定されている場合、bash は対話的シェルが終了する前に、停止中のジョブや実行中のジョブの状態を一覧します。 実行中のジョブがあれば、シェルの終了は、 間にほかのコマンドを挟まずに 2 回目の終了が試みられるまで延期されます (前述の ジョブ制御 を参照)。 停止中のジョブがある場合は、シェルは常に終了を延期します。

checkwinsize

設定されている場合、bash はコマンドの実行後に毎回ウィンドウの大きさをチェックし、 必要に応じて LINES と COLUMNS の値を更新します。

cmdhist

設定されている場合、 bash は複数行に分かれているコマンドの全ての行を、 同じ履歴エントリに保存しようとします。 これを使うと、複数行に分かれているコマンドの再編集が容易になります。

compat31

設定されている場合、bash は、条件コマンド [[の ~ 演算子のクォートされた引き数に関して bash version 3.1 の動作に変更します。

compat32

設定されている場合、 bash は、条件コマンド [[の < 演算子と > 演算子によるロケール固有の文字列比較に関して、version 3.2 の動作に変更します。 bash の bash-4.1 より前のバージョンでは、ASCII コードの照合と strcmp(3) を使います。 bash-4.1 およびそれ以後のバージョンでは、 現在のロケールの照合順序と strcoll(3) を使います。

compat40

設定されている場合、 bash は、条件コマンド [[の < 演算子と > 演算子によるロケール固有の文字列比較 (前項を参照) とコマンドリストの解釈の効果に関して、 version 4.0 の動作に変更します。

compat41

設定されている場合、 bash は posix モードのときに、 ダブルクォートの中のパラメータ展開で シングルクォートを特殊文字として扱います。 シングルクォートは対応が取れている (偶数個) 必要があり、 シングルクォートの間の文字はクォートされているものとして扱われます。 これは version 4.1 の posix モードの動作です。 bash のデフォルトの動作は前のバージョンのままです。

dirspell

設定されている場合、 bash は 指定されたディレクトリ名が存在しなければ、 単語補完のときにディレクトリ名のスペルの訂正を試みます。

dotglob

設定されている場合、 bash は '.' で始まるファイル名をパス名展開の結果に含めます。

execfail

設定されている場合、 組み込みコマンド exec への引き数として指定されたファイルが実行できなくても、 対話的でないシェルが終了しません。 対話的シェルは exec に失敗しても終了しません。

expand_aliases

設定されている場合、エイリアスが前述の エイリアス セクションで説明したように展開されます。 このオプションは、対話的なシェルではデフォルトで有効です。

extdebug

設定されている場合、デバッガのための動作が有効になります。

1. 組み込みコマンド declare の -F オプションが、 引き数で指定された各関数のソースファイル名と行番号を表示します。
2. DEBUG のトラップで実行されたコマンドが 0 以外を返したとき、 次のコマンドはスキップされ実行もされません。
3. DEBUG のトラップで実行されたコマンドが 2 を返し、かつ、 シェルがサブルーチン（シェル関数や、組み込みコマンドの . や source により実行されたシェルスクリプト） を実行しているとき、return の呼び出しがシミュレートされます。
4. 上記のシェル変数の説明で述べたように、 BASH_ARGC と BASH_ARGV が更新されます。
5. 関数のトレースが有効になります。 コマンド置換、シェル関数、(command) で起動されたサブシェルは、 DEBUG と RETURN のトラップを継承します。
6. エラーのトレースが有効になります。 コマンド置換、シェル関数、(command) から起動されたサブシェルは、 ERR のトラップを継承します。

extglob

設定されている場合、拡張されたパターンマッチング機能が有効になります。 これについては、前述のパス名展開で説明しています。

extquote

設定されている場合、ダブルクォート中の `${parameter}` の展開で、$'string' と $"string" のクォートが機能します。 このオプションは、デフォルトで有効です。

failglob

設定されている場合、パス名展開でパターンがファイル名のマッチに失敗すると、 展開エラーになります。

force_fignore

設定されている場合、単語補完において、シェル変数 FIGNORE で指定されたサフィックスの単語は無視されます。 無視された単語が唯一の補完候補であったとしても無視されます。 詳しい説明については、前述の シェル変数 の FIGNORE の説明を参照してください。 このオプションは、デフォルトで有効です。

globstar

設定されている場合、** というパターンがパス名展開で使われると、 深さ 0 以上のディレクトリやサブディレクトリの全てのファイルにマッチします。 直後に / が続く場合には、ディレクトリとサブディレクトリのみにマッチします。

gnu_errfmt

設定されている場合、シェルのエラーメッセージは GNU 標準のエラーメッセージの形式で出力されます。

histappend

設定されている場合、シェルの終了時に変数 HISTFILE の値で指定しているファイルに履歴リストが追加されます。 ファイルへの上書きは行われなくなります。

histreedit

この変数が設定されており、かつ readline が使われている場合、ユーザは失敗した履歴置換を再編集できます。

histverify

この変数が設定されており、かつ readline が使われている場合、履歴置換の結果は即座にはシェルのパーザに渡されません。 その代わり、結果として得られた行は readline の編集バッファに読み込まれ、さらに修正できます。

hostcomplete

この変数が設定されており、かつ readline が使われている場合、bash は @ を含む単語を補完するときにホスト名補完を実行しようとします (前述の READLINE ライブラリ のセクションにおける 補完 を参照)。 これはデフォルトで有効になっています。

huponexit

設定されている場合、bash は対話的なログインシェルを終了するときに、 全てのジョブに SIGHUP を送ります。

interactive_comments

設定されている場合、 # で始まる単語について、その単語とその行の残りの文字を 対話的シェルに無視させることができます (前述の コメント セクションを参照)。 このオプションはデフォルトで有効になっています。

lastpipe

設定されており、かつジョブ制御が有効でなければ、 シェルはバックグラウンドでの実行ではないパイプラインの最後のコマンドを 現在のシェル環境で実行します。

lithist

設定されており、かつ cmdhist オプションが有効ならば、 複数行に分かれているコマンドは (セミコロンで区切られるのではなく) できる限り途中に改行を埋め込むことで履歴に保存されます。

login_shell

シェルがログインシェルとして起動されると (前述の INVOCATION を参照)、このオプションが設定されます。 この値を変更することはできません。

mailwarn

設定されており、かつ bash がメールのチェックに使うファイルが 前回のチェック以降にアクセスされている場合、 メッセージ “ The mail in mailfile has been read ” が表示されます。

no_empty_cmd_completion

設定されており、かつ readline が使われている場合、 空行に対してコマンド補完をさせようとしたときに、 bash は補完候補を PATH から検索しません。

nocaseglob

設定されている場合、 bash はパス名展開 (前述の パス名展開 を参照) を行うときに、 ファイル名の大文字と小文字を区別せずにマッチングを行います。

nocasematch

設定されている場合、 bash は条件コマンド case や [[実行時のパターンマッチで 大文字小文字を区別せずにパターンマッチを行います。

nullglob

設定されている場合、 bash はどのファイルにもマッチしないパターン (前述の パス名展開 を参照) を、その文字列自身ではなく、空文字列に展開します。

progcomp

設定されている場合、プログラム補完機能 (前述のプログラム補完を参照) が有効になります。 このオプションはデフォルトで有効になっています。

promptvars

設定されている場合、 プロンプト文字列に対して パラメータ展開、コマンド置換、算術式展開、クォート削除 が行われます。 この展開は前述の プロンプト セクションで説明した展開が行われた後に行われます。 このオプションはデフォルトで有効になっています。

restricted_shell

シェルが制限モードで起動された場合、 このオプションが設定されます (後述の 制限付きのシェル セクションを参照)。この値を変更することはできません。 これは起動ファイルが実行されるときにもリセットされないので、 シェルが制限付きかどうかを起動ファイル内部で知ることができます。

shift_verbose

設定されている場合、組み込みコマンド shift においてシフトの回数が位置パラメータの数を超えると、エラーメッセージが出力されます。

sourcepath

設定されている場合、組み込みコマンド source (.) は PATH の値を使って、引き数として与えられたファイルを含むディレクトリを探します。 このオプションはデフォルトで有効です。

xpg_echo

設定されている場合、 組み込みコマンド echo は デフォルトでバックスラッシュによるエスケープシーケンスを展開します。

1.31.65　suspend [-f]

SIGCONT シグナルを受け取るまで、シェルの実行をサスペンドします。ログインシェルはサスペンドできません。 -f オプションを与えると、この動作が上書きされ、ログインシェルがサスペンド できるようになります。 シェルがログインシェルかつ -f が与えられていない場合と、ジョブ制御が有効でない場合とを除いて、 返却ステータスは 0 です。

1.31.66　test expr

1.31.67　[expr]

条件式 expr を評価した結果に基づいて、ステータス 0 または 1 を返します。 演算子とオペランドそれぞれは別々の引き数でなければなりません。 式は前述の 条件式 セクションで説明したプライマリで構成されます。 test はオプションを受け取らず、 引き数 -- をオプションの終わりを示すものとして受け取りも無視もしません。

式は次に示す演算子を使って結合できます。 優先度の高い順に示します。 式の評価は後で述べるように引き数の数に依存します。 引き数が 5 つ以上のときは演算子の優先度に従います。

`! expr` expr が偽ならば真になります。

`( expr )`
　　expr　の値を返します。 これを使うと、通常の演算子の優先度を変更できます。

`expr1 -a expr2`
　　expr1 と expr2 が両方とも真ならば真になります。

`expr1 -o expr2`
　　expr1 と expr2 のいずれかが真ならば真になります。

`test` および `[` は、　引き数の数に基づいた規則の集合を用いて条件式を評価します。

引き数が 0 個
　　この式は偽です。

引き数が 1 個
　　引き数が空でない場合に限り真になります。

引き数が 2 個
　　最初の引き数が　! ならば、 2 番目の引き数が空の場合に限り真になります。 最初の引き数が、既に 条件式 セクションで説明した単項条件演算子のいずれかであれば、 単項の評価が真の場合に式は真となります。 最初の引き数が正しい単項条件演算

子でなければ、式は偽となります。

引き数が 3 個

以下の順に条件が適用されます。　2 番目の引き数が、既に 条件式　セクションで説明した二値条件演算子のいずれかであれば、　最初の引き数と 3 番目の引き数をオペランドとして使った 二値評価の結果が式の結果となります。 引き数が 3 個の場合、 -a と -o は二値演算子として扱われます。 最初の引き数が ! であれば、2 番目と 3 番目の引き数を使った、 引き数 2 つの評価の結果を否定したものが値となります。 最初の引き数が (であり、3 番目の引き数が) ならば、 2 番目の引き数を使って引き数　1 つの評価を行った値が結果となります。 これら以外の場合には、式は偽となります。

引き数が 4 個

最初の引き数が ! ならば、 残りの引き数で作った引き数 3 つの式の値を否定したものが結果となります。 それ以外の場合には、先に挙げた規則を使った優先度に従って 式が展開・評価されます。

引き数が 5 個以上

先に挙げた規則を使った優先度に従って式が展開・評価されます。

test や [では、演算子 < と > は ASCII コードに基づき辞書順にソートします

1.31.68　times

シェルとシェルから実行したプロセスのそれぞれについて、 累積のユーザ時間とシステム時間を加えたものを出力します。 返却ステータスは 0 です。

1.31.69　trap [-lp] [[arg] sigspec ...]

シェルがシグナル sigspec を受け取ると、コマンド arg が読み込まれて、実行されます。 arg が存在しない (かつ sigspec が一つ指定された) 場合か、 arg が - の場合、 指定されたシグナルは全てオリジナルの動作 (シェルの起動時に設定されていた値) にリセットされます。arg が空文字列である場合、それぞれの sigspec で指定されたシグナルは、 シェルとシェルが起動したコマンドから無視されます。 arg なしで -p オプションが与えられた場合、 各 sigspec に関連付けられた trap コマンドが表示されます。 引き数が全くないか、 -p だけが与えられた場合、 trap は各シグナルに関連付けられたコマンドのリストを出力します。 -l オプションを与えると、 シェルはシグナル名と対応する番号のリストを出力します。 それぞれの sigspec は、<signal.h> で定義されているシグナル名、またはシグナル番号です。 シグナル名は大文字小文字は区別されず、先頭の SIG は省略可能です。

sigspec が EXIT (0) であれば、シェルの終了時にコマンド arg が実行されます。 sigspec が DEBUG であれば、各々の 単純なコマンド、for コマンド、case コマンド、select コマンド、各々の算術 for コマンドの前、およびシェル関数の最初のコマンドの実行前 (前述のシェルの文法セクションを参照) に、コマンド

arg が実行されます。DEBUG のトラップの影響についての詳細は組み込みコマンド shopt の extdebug オプションの説明を参照してください。 sigspec が RETURN であれば、シェル関数の実行、または組み込みコマンドの . や source で実行されたスクリプトの実行が終わるたびにコマンド arg が実行されます。

sigspec が ERR であれば、 単純なコマンドが 0 以外の終了ステータスのときにコマンド arg が実行されます。 ただし、失敗したコマンドが、 while または until キーワード直後のコマンドリストに含まれる場合、 if 文の条件に含まれる場合、 && や || のリスト中で実行するコマンドに含まれる場合、 および、コマンドの戻り値が！で反転されている場合には、 ERR のトラップは実行されません。 これらは errexit オプションが従う条件と同じです。

シェルに入る際に無視されるシグナルは、 トラップもリセットもできません。 無視されなかったシグナルのトラップは、 サブシェルやサブシェル環境では作られたときに 元の値にリセットされます。 sigspec のいずれかが不正であれば、返却ステータスは偽になります。 それ以外の場合には、 trap は真を返します。

1.31.70　type [-aftpP] name [name ...]

オプションなしの場合には、各 name をコマンド名として使ったときに、それがどのように解釈されるかを示します。 -t オプションを使うと、 name がエイリアス、シェルの予約語、関数、 組み込みコマンド、ディスク上のファイルのいずれかの場合、 type はそれぞれ alias, keyword, function, builtin, file という文字列を出力します。 name が見つからない場合は何も出力されず、偽の終了ステータスが返されます。 -p オプションを使うと、 type は name をコマンド名として指定した場合に実行されるディスクファイルの名前を返します。ただし、 “type -t name” が file を返さない場合は、何も返しません。 -P オプションを使うと、 “type -t name” が file を返さない場合でも name を PATH から探します。 コマンドがハッシュされている場合、 -p や -P はハッシュされている値を表示します。 この場合、表示されるのは、必ずしも PATH 中で最初に現われるファイルとは限りません。 -a オプションを使うと、 type は name という名前の実行ファイルがある場所を全て出力します。 -p オプションが同時に使われていない場合に限り、 エイリアスや関数も出力されます。 -a を使うときには、ハッシュされているコマンドの表は参照されません。 -f オプションを使うと、 組み込みコマンド command と同じように、シェル関数を探しません。 type は、すべての引き数が見つかれば真を返し、 いずれかが見つからなければ偽を返します。

1.31.71　ulimit [-HSTabcdefilmnpqrstuvx [limit]]

これを使うと、シェルおよびシェルが起動するプロセスが 利用できるリソースを制御できます。 ただし、このような制御ができるシステムの場合に限ります。 -H オプションと -S オプションは、 それぞれ指定されたリソースに対する 強い制限 (hard limit) と弱い制限 (soft limit) を設定します。 強い制限は一度設定すると、root 以外のユーザが増やすことはできません。 弱い制限は強い制限の値までは増やせます。 -H と -S がどちらも指定されていない場合、 強い制限と弱い制限が両方とも設定されます。 limit の値には、数字 (単位はリソースに応じて規定)、または hard, soft, unlimited が指定できます。hard, soft, unlimited は、それぞれ現在の強い制限、現在の弱い制限、制限なしを表します。 limit を省略すると、リソースの弱い制限の現在値が表示されます。 ただし、-H が与えられている場合は除きます。 複数のリソースが指定されているときは、制限名と単位が値の前に出力されます。 他のオプションは以下のように解釈されます:

`-a`　現在の制限を全て表示する
`-b`　ソケットバッファの最大サイズ

-c　生成されるコア（core）ファイルの最大サイズ
-d　プロセスのデータセグメントの最大サイズ
-e　スケジュール優先度（"nice"）の最大値
-f　シェルとその子プロセスが生成できるファイルの最大サイズ
-i　保留中シグナルの最大数
-l　メモリにロックできる最大サイズ
-m　常駐セットサイズ（resident set size）の最大値（多くのシステムはこの制限を守りません）
-n　オープンできるファイル・ディスクリプターの最大数　（ほとんどのシステムでは、この値を設定することはできません）
-p　512 バイトブロック単位でのパイプのサイズ　（これは設定できないかもしれません）
-q　POSIX メッセージキューの最大バイト数
-r　リアルタイム・スケジューリングの優先度の最大値
-s　最大スタックサイズ
-t　CPU 時間の最大量（秒単位）
-u　1 人のユーザが使用できる最大のプロセス数
-v　そのシェルが使用できる最大の仮想メモリ量　（システムによっては、子プロセスも含まれる）
-x　ファイルロックの最大数
-T　スレッドの最大数

limit が指定された場合、その値が指定されたリソースの新しい値となります (-a は表示専用です)。 オプションが全く指定されなかった場合は、 -f が指定されたものとみなされます。 値は 1024 バイト単位で増えますが、例外として -t は秒単位、 -p 512 バイトブロック単位、 -T, -b, -n, -n, -u は単位なしの値です。 返却ステータスは基本的に 0 ですが、 不正なオプションや引き数が渡された場合、 新しい制限を設定する際にエラーが起きた場合は除きます。

1.31.72　umask [-p] [-S] [mode]

ユーザのファイル生成マスクに mode を設定します。 mode が数字で始まる場合には、これは 8 進数と解釈されます。 それ以外の場合には、chmod(1) に指定するのと同様のシンボリックなモードマスクと解釈されます。 mode が省略されると、現在のマスクの値が出力されます。-S オプションを指定すると、マスクはシンボリックな形式で表示されます。 デフォルトの出力は 8 進の数値です。 -p オプションが指定され、かつ mode が省略された場合、入力として再利用できる形式で出力が行われます。 モードが正常に変更できた場合や、 mode 引き数が全く与えられなかった場合には、 返却ステータスは 0 となります。 それ以外の場合には偽となります。

1.31.73　unalias [-a] [name ...]

定義されているエイリアスのリストから name を削除します。 -a が与えられている場合には、エイリアス定義は全て削除されます。 与えられた name が定義されているエイリアスでない場合以外は、返却ステータスは真になります。

1.31.74　unset [-fv] [name ...]

name それぞれについて、対応する変数や関数を削除します。 オプションが全く指定されていない場合や、-v オプションが指定された場合は、各 name はシェル変数を参照します。 読み込み専用の変数の設定を消すことはできません。 -f が指定されている場合、各 name はシェル関数を参照し、その関数の定義が削除されます。 設定が消された変数や関数は全て、 それ以降のコマンドに渡される環境変数からも削除されます。 COMP_WORDBREAKS, RANDOM, SECONDS, LINENO, HISTCMD, FUNCNAME, GROUPS, DIRSTACK のいずれかの設定を消した場合、これらが持つ特別な特性もなくなります。 これは後で再設定しても元に戻ることはありません。 name が読み込み専用の場合以外は、 終了ステータスは真となります。

1.31.75　wait [n ...]

指定された各プロセスを wait し、その終了ステータスを返します。各 n はプロセス ID またはジョブ指定です。 ジョブ指定を与えた場合、そのジョブのパイプラインに含まれる 全てのプロセスを wait します。 n が与えられていない場合には、現在アクティブな全ての子プロセスを wait し、 返却ステータスは 0 となります。 n が存在しないプロセスやジョブを指定している場合、 返却ステータスは 127 になります。 それ以外の場合、返却ステータスは wait していた最後のプロセスまたはジョブの終了ステータスとなります。

1.32　制限付きのシェル (RESTRICTED SHELL)

bash を rbash という名前で起動した場合や、起動時に -r オプションを指定した場合には、シェルは制限された状態になります。 制限付きのシェルは、標準のシェルよりも細かく制御された環境を用意したいときに使われます。 制限付きのシェルは bash と全く同じように動作しますが、 以下のようなことが許可されなかったり実行されなかったりします。

cd を使ってディレクトリを変更すること

SHELL, PATH, ENV, BASH_ENV の値の設定や設定取り消しを行うこと

/ を含むコマンド名を指定すること

組み込みコマンド . の引き数として / を含むファイル名を指定すること

組み込みコマンド hash のオプション -p の引き数として / を含むファイル名を指定すること

起動時にシェル環境から関数定義をインポートすること

起動時にシェル環境から SHELLOPTS の値を解釈すること

リダイレクション演算子 >, >|, <>, >&, &>, >> を使ってリダイレクトを行 なうこと

組み込みコマンド exec を用いて、シェルを別のコマンドに置き換えること

組み込みコマンド enable の -f オプションと -d オプションを使って、組み込みコマンドを追加・削除すること

組み込みコマンド enable を使って、無効になっている組み込みコマンドを有効にすること

組み込みコマンド command に -p オプションを指定すること

set +r や set +o restricted を用いて制限モードを解除すること

これらの制限は、何らかの起動ファイルを読み込んだ後に適用されます。

シェルスクリプトであると判明したファイルが実行されるとき (前述の コマンドの実行 を参照)、 rbash はスクリプト実行用に立ち上げたシェルでは制限を全て無効にします。

1.33　関連項目

```
Bash Reference Manual, Brian Fox and Chet Ramey
The Gnu Readline Library, Brian Fox and Chet Ramey
The Gnu History Library, Brian Fox and Chet Ramey
Portable Operating System Interface (POSIX) Part 2: Shell and
Utilities, IEEE
sh(1), ksh(1), csh(1)
emacs(1), vi(1)
readline(3)
```

1.34　ファイル

1.34.1　/bin/bash

bash の実行ファイル。

1.34.2　/etc/profile

システム全体用の初期化ファイル。ログインシェルが実行します。

1.34.3　~/.bash_profile

個人用の初期化ファイル。ログインシェルが実行します。

1.34.4　~/.bashrc

対話シェルごとに実行される、個人用の起動ファイル。

1.34.5　~/.bash_logout

個人用のログインシェル後処理ファイル。 ログインシェルの終了時に実行されます。

1.34.6　~/.inputrc

個人用の readline 初期化ファイル。

1.35　作者

Brian Fox, Free Software Foundation
bfox@gnu.org

Chet Ramey, Case Western Reserve University
chet.ramey@case.edu

1.36　バグ報告

bash のバグを見つけたら必ず報告してください。 ただし報告の前には、それが本当にバグであることと、 バグが最新版の bash で起こることを確かめてください。 最新版は ftp://ftp.gnu.org/pub/gnu/bash/ から入手できます。

本当にバグがあると判断した場合には、 bashbug コマンドを使ってバグ報告を行います。 バグを修正してくださった場合には、 ぜひその内容も一緒にメールしてください! 提案や「哲学上の」バグ報告は、bug-bash@gnu.org にメールしたり、 ニュースグループの gnu.bash.bug に投稿したりしてくださってもかまいません。

バグ報告には必ず以下のことを書いてください:

bash のバージョン
ハードウェアとオペレーティングシステム
コンパイルに使ったコンパイラ
バグ動作の説明
バグを再現できる「レシピ」（簡単なシェルスクリプト）

bashbug コマンドは、バグ報告を送るために用意されているテンプレートに、最初の 3 項目を自動的に書き込みます。

このオンラインマニュアルに関するコメントやバグ報告は chet.ramey@case.edu 宛にお願いします。

1.37　バグ

bash は大きすぎるし、遅すぎます。

bash と昔ながらのバージョンの sh にはちょっとした違いがいくつかあります。この大部分は POSIX の仕様のせいで生じたものです。

使い方によっては、エイリアスは混乱の元になります。

シェル組み込みコマンドとシェル関数は、停止したり、実行を再開したりできません。

複合コマンドや ‘a ; b ; c’ の形式のコマンド列は、 プロセスのサスペンドを行う際に綺麗に扱うことができません。 あるプロセスを停止すると、 シェルはコマンド列の次のコマンドを即座に実行するからです。この問題はコマンド列を括弧の中に置いて サブシェルに実行させることで解決できます。 こうすれば、ひとまとまりのものとして停止できます。

配列変数は (まだ) エクスポートできません。

一度に 1 つのコプロセスしかアクティブにできません。