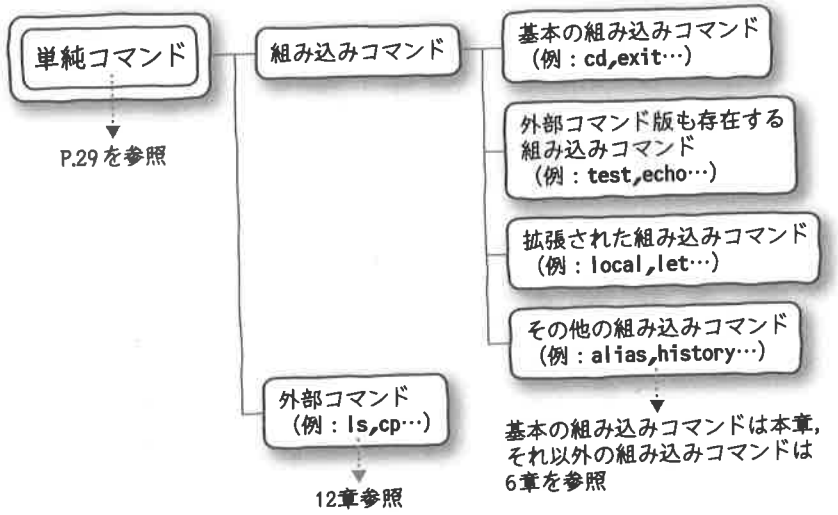
基本の組み込みコマンドについて

シェルスクリプトで使用される単純コマンドには、lsやcpコマンドのような**外部コマンド**と、cdやechoコマンドのような**組み込みコマンド**があります(**図A**)。シェルスクリプト中での使用頻度が高いtestやechoコマンドや、原理的に外部コマンドにはできないcdやexitコマンドなど、多くのコマンドがシェルの組み込みコマンドとして実装されています。

本章では、同名の外部コマンドが存在しない基本の組み込みコマンドを解説します。

図A 単純コマンドの種類



:コマンド

Linux (bash)
FreeBSD (sh)
Solaris (sh)

何もしないで単に「0」の終了ステータスを返す

書式 : [引数 ...]

例

```
while :  ……:コマンドにより終了ステータス0が返り、無限ループになる
do  ……ループの開始
  echo hello  ……メッセージを出力
done  ……ループの終了
```

基本事項

:コマンドは、何もしないヌルコマンドです。ただし、:コマンドに対するリダイレクトや、引数のパラメータ展開、コマンド置換は通常通り行われ、その結果、ファイルがオープンされたり、シェル変数が変化したり、別のコマンドが起動されたりといった動作が行われる場合があります。

終了ステータス

:コマンドの終了ステータスは「0」になります。ただし、リダイレクトやパラメータ展開でエラーが発生した場合は、終了ステータスとして「0」以外のエラーコードが返されます。

解説

:コマンドは何もしないで単に終了ステータス「0」を返すだけのコマンドです。したがってtrueコマンドと等価であり、trueコマンドの代わりに使用できます。trueコマンドは、シェルのバージョンによっては外部コマンドとして実装されている場合がありますが、:コマンドは必ず組み込みコマンドとして実装されているため、「while true」と記述するよりも、例のように「while :」と記述したほうが効率がよいでしょう。

その他、:コマンドは、パラメータ展開やリダイレクトだけを行って、コマンドは実行したくない場合や、if/for/while文のリストで何もコマンドを実行したくない場合にも使用されます。

${パラメータ?値}形式のパラメータ展開

${パラメータ?値}という形のパラメータ展開を使って、所定のパラメータがセットされているかどうかをチェックすることができます。このとき、パラメータのチェックだけを行ってコマンドは実行したくないという場合、:コマンドが使えます[注1]。

リストAのように記述すると、位置パラメータ$1(このシェルスクリプト自体の第1引数)

注1 パラメータ展開については8章を参照してください。

がセットされているかどうかがチェックされ、セットされていた場合は何も実行されず、セットされていなかった場合は「引数を指定してください」というエラーメッセージを表示してシェルスクリプトが終了します。

${パラメータ=値}形式のパラメータ展開

${パラメータ=値}という形のパラメータ展開を使って、パラメータが設定されていなかった場合のデフォルト値を設定できます。ここでも同様に:コマンドを使えば、パラメータのセットのみを行って、コマンドは実行しないようにすることができます[注2]。

リストBは、あらかじめシェル変数**CFLAGS**がセットされている場合は何もせず、**CFLAGS**がセットされていなかった場合には「'-O2 -fomit-frame-pointer'」という値を代入するという例です。

リダイレクトでサイズゼロのファイルを作成

:コマンドでも、標準入出力などのリダイレクトは行われます[注3]。そこで、**リストC**のように記述すると、:コマンドの標準出力が「flie」というファイルにリダイレクトされ、結果的にファイルサイズゼロの「file」という名前のファイルが作成されます。

この動作はtouchコマンドを使ってtouch *file*を実行した場合[注4]に似ており、touchコマンドの代わりに使用できます。ただし、touchコマンドとは違って、すでに同名のファイルが存在していた場合、そのファイルサイズがゼロになります。このことを利用して、ファイルを削除せずにファイルの中身を消去し、ファイルサイズをゼロにしたい場合にも使用されます。

なお、このようにリダイレクトを利用する場合、実は:コマンド自体を省略して**リストD**のように記述することもできます。

リストA パラメータの設定チェック

```
: ${1?'引数を指定してください'} ……$1が未設定の場合エラーメッセージを表示
```

リストB シェル変数のデフォルト設定

```
: ${CFLAGS='-O2 -fomit-frame-pointer'} ……CFLAGSが未設定の場合デフォルト値を代入
```

リストC ファイルサイズゼロのファイルを作成

```
: > file …… fileという名前のファイルサイズゼロのファイルができる
```

リストD :コマンド自体を省略し、リダイレクトのみを実行

```
> file …… fileという名前のファイルサイズゼロのファイルができる
```

注2 パラメータ展開については8章を参照してください。

注3 リダイレクトについては11章を参照してください。

注4 touchはファイルのアクセス時刻、修正時刻を変更するコマンド。touch *file*で存在しないファイルを指定すると、そのファイルをサイズゼロで新規に作成します。

構文中のヌルリストとして

if／for／while文では、構文中のリストには何らかのリストを必ず記述しなければなりません。このとき、たとえばif文でthenの直後のリストでは何も実行せず、elseでのみリストを実行したい場合があります。そのような場合に:コマンドを使えます。

構文中のヌルリストとして:コマンドを使う例は、それぞれif／for／while文の項目（p.43、p.55、p.63）を参照してください。

注意事項

パラメータ展開だけを行うには:が必要

:コマンドを省略してパラメータ展開を行おうとすると、置換されたパラメータがコマンド名と解釈され、そのコマンドが実行できないのでエラーになります。リダイレクトの場合とは違って:は省略できないので注意してください。

○正しい例

```
: ${TMPDIR=/tmp}
```

×誤った例

```
${TMPDIR=/tmp}
```

コメントアウトとは違う

すでにコマンドが記述された行の行頭に:を記述することにより、そのコマンドは実行されなくなりますが、これはコメントアウトとは違ってリダイレクトやパラメータ展開は実行されてしまいます。コメントアウトが目的の場合は#を使ってコメントアウトしてください。

参照

true（p.150） 位置パラメータ（p.162） コメントの書き方（p.23）

.コマンド

Linux (bash)
FreeBSD (sh)
Solaris (sh)

現在実行中のシェルに別のシェルスクリプトを読み込ませる

書式 . ファイル名

例
```
. "$HOME"/.profile ……ホームディレクトリにある.profileを現在のシェルに読み込む
```

基本事項

.コマンドを実行すると、引数のファイル名で指定されたファイルが現在のシェルに読み込まれます。ファイルが/を含むパス（絶対パスまたはカレントディレクトリからの相対パス）で指定されていない場合は、**PATH**を使ってファイルが検索されます。.でシェルスクリプトを読み込む場合、通常のシェルスクリプトとは違ってファイルに実行属性は必要ありません。

終了ステータス

.コマンドの終了ステータスは、読み込んだファイル中で実行された最後のリストの終了ステータスになります。ただし、ファイル中にリストが1つもない場合は、終了ステータスは「0」になります。また、ファイルが見つからないなど.コマンドの実行自体がエラーになった場合は、「0」以外の終了ステータスが返されます。

解説

複数のシェルスクリプトに共通した、**シェル変数**の定義、**シェル関数**の定義やその他の前処理を、あらかじめ別ファイルに書き出しておき、このファイルをシェルスクリプトに読み込んで実行すると便利です。このような場合に.コマンドを使います。

なお、.コマンドを使わなくても、シェルスクリプトの中で別のシェルスクリプトを実行することは可能です。しかし、別のシェルスクリプトを実行する場合は、現在実行中のシェルとは別のプロセスのシェルが起動されてしまうため、シェル変数の定義やシェル関数の定義などについては、元のシェルスクリプト上の動作環境には反映されず、無意味な動作になってしまいます。そこで.コマンドを使い、別のシェルスクリプトを現在実行中のシェルに直接読み込むようにするのです。

なお、.コマンドで読み込むシェルスクリプトには、実行属性は必要なく、1行目の#!/bin/shの行も必要ありませんが、これらがあったとしても単に無視されるだけなので問題ありません。

.コマンドと通常のシェルスクリプトとの比較

カレントディレクトリに、**リストA**のような「src_test」というファイルを作成し、これを**リストB**のように.コマンドで読み込んだ場合と、**リストC**のようにシェルスクリプトとして実行した場合で比較してみましょう。

すると、リストBでは「src_test」の中でのシェル変数**TEST_VAL**への代入が.コマンドの実行終了後も影響を及ぼし、echoで値を表示させるとたしかに値が代入されていることがわかります。

一方、リストCでは、「src_test」はシェルスクリプトとして別のシェルで実行されるため、**TEST_VAL**への値の代入は元のシェルとは関係なく、「src_test」の実行終了後に**TEST_VAL**の値を表示しても中には値が代入されていないことがわかります。

リストA src_test

```
#!/bin/sh ……直接実行もできるようにこの行も記述しておく
TEST_VAL=hello ……試しにシェル変数に値を代入
```

リストB .コマンドで読み込んだ場合

```
TEST_VAL= ……変数の値をクリア
. ./src_test ……src_testを.コマンドで読み込む
echo "$TEST_VAL" ……helloという値が表示される
```

リストC シェルスクリプトとして実行した場合

```
TEST_VAL= ……変数の値をクリア
./src_test ……src_testをシェルスクリプトとして実行
echo "$TEST_VAL" ……値は表示されない
```

引数の指定

Warning
Linux (bash) FreeBSD Solaris
この方法には制限があります。

bashでは、.コマンドで読み込むファイルに対して引数を指定することができます。指定された引数は、シェル関数の呼び出し時と同様に、ファイルを読み込んでいる間のみ、一時的に位置パラメータにセットされます。

たとえば、**リストD**のような「src_arg_test」というファイルがカレントディレクトリにある場合、**リストE**のように「Hello World」という引数を付けて.コマンドを実行すると、「src_arg_test」の中のechoコマンドに引数が渡り、「Hello World」と無事表示されます。

リストD src_arg_test

```
echo "$1" "$2" ……渡された引数のうち、"$1"と"$2"を表示
```

リストE 引数付きで.コマンドを実行

```
. ./src_arg_test Hello World ……引数を付けて.コマンドを実行
```

sourceコマンドで記述

Warning
Linux (bash) ○ FreeBSD (sh) × Solaris (sh) ×
この方法には制限があります。

bashでは、**リストF**のように、コマンド名の.の代わりにsourceと記述することもできます。コマンド名が違うだけで、動作は.コマンドと同じです。このsourceという名前はcsh由来のものであり、たしかにsourceと書いたほうが.よりも見ためでわかりやすいかもしれませんが、従来のshとの互換性がなくなるため、使用には注意が必要です。

リストF bashではsourceと書いてもよい

```
source file
```

ファイル中でのreturnコマンド

Warning
Linux (bash) ○ FreeBSD (sh) ○ Solaris (sh) ×
この方法には制限があります。

.コマンドで読み込まれているファイルの中で、**リストG**のようにreturnコマンドを実行すると、その時点で.コマンドによるファイルの読み込みが終了します。returnに引数を付けると、その値が.コマンド自体の終了ステータスになります。

returnコマンドは、本来はシェル関数からリターンするためのものですが、このように.コマンドで読み込まれるファイル中で使用すると、ファイルの読み込みの終了という意味になります。ただし、Solarisのshでは、このようなreturnコマンドはエラーとなってしまうため、注意してください。

リストG ファイル中でのreturnコマンド

```
if [ "$i" -lt 0 ]; then ……シェル変数"$i"の値が負の場合
  return 1 ……終了ステータス1でこのファイルを終了する
fi
```

注意事項

exitすると、もとのシェルがexitしてしまう

.コマンドで読み込むファイル中でexitコマンドを実行すると、そのファイルの実行が終了するのではなく、.コマンドを実行している元のシェル自体がexitしてしまいます。これは、実行中のシェルがファイルを直接読み込んでいるという動作を考えれば当然の結果ですが、意図せずにシェルを終了してしまわないよう、注意してください。

Memo

❸ログインシェルが"$HOME"/.profileや"$HOME"/.bash_profileなどのファイルを読み込むのは、動作としては.コマンドで読み込んでいるのと同じです。

参照

シェル関数(p.76)　位置パラメータ(p.162)　return(p.118)　exit(p.103)

break

Linux (bash) ○
FreeBSD (sh) ○
Solaris (sh) ○

for文／while文のループを途中で抜ける

書式 break [数値]

例

```
found=0 ……シェル変数foundを0に初期化
for file in * ……カレントディレクトリ上のすべてのファイルについてループ
do ……ループの開始
  if cmp -s "$file" /some/dir/myfile ……もしそのファイルが/some/dir/myfileと同じならば
  then
    found=1 ……シェル変数foundに、ファイルが見つかったことを示す1を代入
    break ……for文のループを抜ける
  fi ……if文の終了
done ……ループの終了
```

基本事項

breakコマンドを、for文またはwhile文のループ中で使用すると、その時点でループを終了し、ループの外に抜けます。breakコマンドに[数値](N)の引数を付けると、for文またはwhile文のN重ループを一気に抜けることができます。[数値]を省略するとbreak 1と同じになります。

終了ステータス

breakコマンドによってループを抜けると、終了ステータスは「0」になります。

解説

for文やwhile文のループは、通常はその文自体のループ条件によってループが実行されますが、場合によってはループ中で一定の条件が成立すれば、その時点でループを終了したいことがあります。そのような場合にbreakコマンドを使います。

シェルスクリプトのbreakコマンドは、C言語のbreak文にない機能として、引数で数値を指定することができ、たとえばbreak 2で2重ループから抜けることができます。

Memo

❸bashの場合は、select文もbreakによって抜けられます。

参照

for文(p.55)　while文(p.63)　select文(p.69)

continue

Linux (bash)
FreeBSD (sh)
Solaris (sh)

for文／while文のループを次の回に進める

書式 continue [数値]

例

```
i=0 ……シェル変数iを0に初期化
while ……while文の開始
  i=`expr "$i" + 1` ……iに1を加える
  [ "$i" -le 10 ] ……iの値が10以下であればループ
do ……ループの開始
  if [ "$i" = 5 ]; then ……もしiが5であれば
    continue ……この回の実行を打ち切り、次の回のループに進む
  fi ……if文の終了
  echo "$i" ……iの値の表示
done ……ループの終了
```

基本事項

continueコマンドを、for文またはwhile文のループ中で使用すると、その時点でその回のループの実行を終了し、次の回のループに進みます。continueコマンドに数値(N)の引数を付けると、for文またはwhile文のN重ループについてcontinueの動作が行われます。数値を省略するとcontinue 1と同じになります。

終了ステータス

continueコマンドによって次のループに進むと、終了ステータスは「0」になります。

解説

一定のループ条件によってfor文やwhile文を実行中に、特定の条件が成立した場合はその回のループの残りの部分を実行せずに次の回のループに進みたいことがあります。このような場合にcontinueコマンドを使用します。

シェルスクリプトのcontinueコマンドは、C言語のcontinue文にない機能として、引数で数値を指定でき、たとえばcontinue 2で2重ループをcontinueすることができます。

Memo

- bashの場合は、select文でもcontinueが使えます。

参照

for文(p.55)　while文(p.63)　select文(p.69)

cd

Linux (bash)
FreeBSD (sh)
Solaris (sh)

別のディレクトリに移動する

書式 cd [ディレクトリ名]

例

```
cd /usr/bin ……/usr/binディレクトリに移動
```

基本事項

cdコマンドを実行すると、シェル自身のカレントディレクトリが、引数のディレクトリ名で指定されたディレクトリに変更されます。引数のディレクトリ名を省略した場合は、"$HOME"が指定されたものとみなされます。

シェル変数**CDPATH**に:で区切られたディレクトリが設定されている場合は、引数で指定されたディレクトリ名が**CDPATH**の中から検索されます。ただし、引数として/または.または..で始まるディレクトリ名が指定された場合はCDPATHは参照されません。

終了ステータス

カレントディレクトリの変更に成功した場合は、終了ステータスは「0」になります。ただし、ディレクトリが存在しないなどのエラーが発生した場合は、終了ステータスは「0」以外になります。

解説

シェルのコマンドライン上でcdコマンドでほかのディレクトリに移動するのと同じように、シェルスクリプト上でもcdコマンドでほかのディレクトリに移動することができます。引数なしでcdを実行するとホームディレクトリに戻るという動作も同じです。

ただし、UNIX系OSでは、カレントディレクトリの属性はプロセスごとに持っているため、シェルスクリプトの中でcdコマンドを実行しても、元の(コマンドラインの)シェル環境上のカレントディレクトリは一切変更されません。

ディレクトリを移動してから処理する例

シェルスクリプトの処理内容によっては、いったんディレクトリを移動したほうが処理がしやすい場合があります。たとえば、**リストA**は、**/some/dir**というディレクトリの下のすべてのファイル(.で始まるファイルを除く)を、そのファイル名の頭に「backup-」という文字列を付けてコピーする例です。ここで、もしcdコマンドを使用していなかったとすると、ファイル名の先頭に**/some/dir/**というパスが付いてしまうため、その中に「backup-」という文字列を割り込ませるのが面倒になります。

リストA ディレクトリを移動してから処理する例

```
cd /some/dir ……/some/dirに移動
for file in * ……すべてのファイルについてループ
do ……ループの開始
  cp -p "$file" backup-"$file" ……ファイル名の頭にbackup-を付けてコピー
done ……ループの終了
```

サブシェル内でディレクトリ移動

シェルスクリプト内でcdコマンドで一時的にディレクトリを移動して処理を行ったあと、再び元のディレクトリに戻って別の処理を続行したい場合があります。このような場合は、**リストB**のように、cdコマンドを含むリストをサブシェルの( )で囲みます。すると、( )の内部のみカレントディレクトリが変更されるだけで、サブシェルを抜けると元のディレクトリに戻ります。

CDPATHが設定されている場合

シェル変数**CDPATH**が設定されている場合、cdコマンドで指定されたディレクトリが、**CDPATH**に設定されたディレクトリの中から検索されるようになります。たとえば、**CDPATH**に**/usr/local**が含まれている場合、単にcd binと実行しただけで**/usr/local/bin**に移動します。

CDPATHはコマンドラインのシェル上でディレクトリ移動を楽にするために使用すると便利な場合があります。しかし、シェルスクリプト上では混乱を招くため、**CDPATH**は使わない方がいいでしょう。

リストB サブシェル内でディレクトリ移動

```
( ……サブシェルの開始
  cd /some/dir ……/some/dirに移動
  for file in * ……すべてのファイルについてループ
  do ……ループの開始
    cp -p "$file" backup-"$file" ……ファイル名の頭にbackup-を付けてコピー
  done ……ループの終了
) ……サブシェルの終了(元のディレクトリに戻る)
echo 'バックアップ完了' > log ……カレントディレクトリにログファイルを作成
```


cdコマンドの-Pオプション

Warning

Linux (bash) ○ FreeBSD (sh) ○ Solaris ×

この方法には制限があります。

bashやFreeBSDのshでは、cdコマンドで指定したディレクトリがシンボリックリンクの場合、移動先のディレクトリが、シンボリックリンクを含むパス名のまま記憶されており、その後、親ディレクトリに移動した場合に、あたかもシンボリックリンクを逆にたどるように、元のディレクトリに戻ってくることができます。

この動作を禁止し、本来の物理的なファイルシステムのディレクトリ構造通り移動するには、cdコマンドに-Pオプションを付けます。**図A**に、-Pオプションを付けた場合と付けない場合の動作の違いがわかる使用例を示します。

なお、Solarisのshでは、ディレクトリ移動は常に物理的なディレクトリ構造を元に行われるため、cdコマンドに-Pオプションはありません。

図A cdコマンドの-Pオプションの使用例

```
$ pwd ……カレントディレクトリを表示
/home/guest ……現在/home/guestにいる
$ ln -s /usr/local/bin short ……/usr/local/binへの近道のシンボリックリンクを作る
$ cd short ……シンボリックリンクをたどってディレクトリ移動
$ pwd ……カレントディレクトリを表示
/home/guest/short ……シンボリックリンクを含んだパス名が表示される
$ cd .. ……親ディレクトリに移動
$ pwd ……カレントディレクトリを表示
/home/guest ……/home/guestに戻ることができる
$ cd -P short ……今度は-Pオプションで物理的にディレクトリ移動
$ pwd ……カレントディレクトリを表示
/usr/local/bin ……/home/guest/shortではなく/usr/local/binになる
$ cd .. ……親ディレクトリに移動
$ pwd ……カレントディレクトリを表示
/usr/local ……/home/guestには戻らず、/usr/localに移動する
```

Memo

- bashまたはFreeBSDのshでcdコマンドの-Pオプションをデフォルトにするには、あらかじめset -Pコマンドを実行しておきます。この状態で一時的に-Pを無効にしてcdコマンドを実行するには、cd -Lとします。set -Pコマンドはpwdコマンドにも影響するため、同様に-Pを無効にするにはpwd -Lとします。
- bashまたはFreeBSDのshでは、cd -のようにディレクトリ名に「-」を指定すると、直前にいたディレクトリに戻れます。

参照

サブシェル(p.72) set(p.120)

eval

Linux (bash)
FreeBSD (sh)
Solaris (sh)

引数を再度解釈しコマンドを実行する

書式 **eval** [引数 ...]

例
```
eval echo \"\$$var\"  ……シェル変数varの値を変数名とするシェル変数の内容を表示
```

基本事項

evalコマンドは、その引数の文字列がシェルに入力されたものとみなして再度解釈を行い、その結果のコマンドを実行します。

終了ステータス

解釈の結果、実行されたコマンドの終了ステータスが、evalコマンドの終了ステータスになります。

解説

一般に、シェルは入力されたコマンドやその引数に対して、パラメータ展開やコマンド置換などの各種解釈を行った上で実際にコマンドを起動します。evalコマンドは、evalコマンドに渡された**引数**に対して、再度パラメータ展開やコマンド置換などの解釈を行い、その結果のコマンドを実行します。シェルがevalコマンドを実行する時点では、引数は一度解釈されているため、evalコマンドを使うと、結果的に引数の解釈が2回行われることになります。

シェル変数の間接参照

evalコマンドは、シェル変数の間接参照に利用することができます。**図A**のように、あらかじめday0～day6というシェル変数に曜日名を入れておき、「today」というシェル変数にday0～day6のいずれかとして、たとえば「day3」を代入したとすると、このシェル変数todayから曜日名に展開するには、図のようにevalコマンドを使えばいいのです。

ここで、「echo \"\$$today\"」の部分はまずシェルに1回評価されてecho "$day3"となり、この文字列のままevalコマンドに渡されます。evalコマンドはecho "$day3"を再度評価して、echo Wednesdayとなり、echoコマンドが実行されて「Wednesday」が表示されます。

シェル変数には原則的にダブルクォート(" ")を付けて、シェル変数の値の中に含まれているかもしれない*やスペースなどが再度評価されるのを防ぎますが、ここではevalによって評価される段階でダブルクォートが残るように、全体を\"と\"で囲んでいることに注意してください。ダブルクォートを問題にしない場合は、「\$$today」と書いてもかまいません。いずれにしても、2つの$のうちの左側の$は、シェルではなく、evalによって評価されるように、バックスラッシュ(\)でクォートする必要があります。

このようなevalコマンドの使い方は、配列変数を使わずに配列と同様の処理を行うのに有用です[注5]。

なお、bashの場合は、シェル変数の間接参照は、evalを使わずに「echo "${!today}"」というパラメータ展開を使っても行えます[注6]。

図A シェル変数の間接参照

```
$ day0=Sunday day1=Monday day2=Tuesday day3=Wednesday
$ day4=Thursday day5=Friday day6=Saturday      シェル変数day0～day6に曜日名を代入
$ today=day3                                   シェル変数todayに、試しにday3を代入
$ eval echo \"\$$today\"                       シェル変数todayを2回解釈して曜日名を表示
Wednesday                                      たしかにWednesdayと表示される
```

参照

ダブルクォート " "(p.209)　　バックスラッシュ\(p.211)

注5 詳しくは「bash以外のシェルで配列を使う方法」(p.279)を参照してください。
注6 「${!パラメータ}」(p.204)を参照してください。

exec

Linux (bash)
FreeBSD (sh)
Solaris (sh)

新しいプロセスを作らずにコマンドを起動する

書式 **exec** [コマンド [引数 ...]]

例 `exec myprog` ……myprogというコマンドを起動(同時にシェルは終了)

基本事項

execコマンドを実行すると、execコマンドの引数で指定されたコマンド(**外部コマンド**)が、シェル自身のプロセスIDのままで実行され、以後、起動されたコマンドに制御が移ります。新しいプロセスは作成されません。コマンドが起動できなかった場合、シェルが対話シェルでなければシェルスクリプトは終了します(bash以外の場合は対話シェルの場合でもシェルは終了します)。

引数のコマンドを省略し、リダイレクトのみを行った場合は、現在のシェル自身に対するリダイレクトと解釈され、標準入出力などが変更されます。

終了ステータス

引数で指定したコマンドが正常に起動できた場合、シェルには戻らないため、終了ステータスはありません。コマンドが起動できなかった場合は終了ステータスは「0」以外になります。

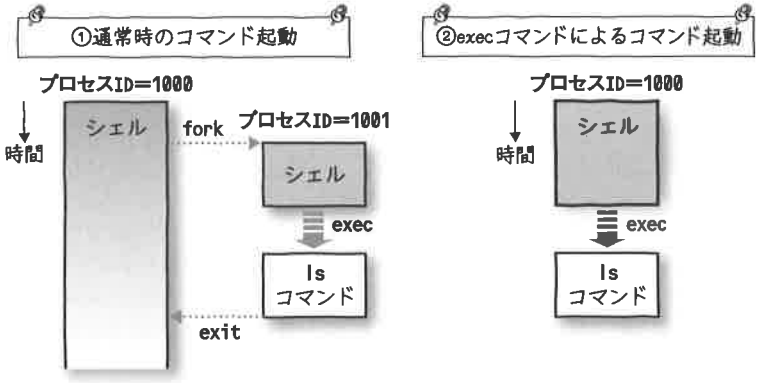
引数を指定せず、リダイレクトのみを行った場合、正常にリダイレクトが行われれば終了ステータスは「0」になります。

解説

シェルに限らず、あるプロセス(たとえばシェル)が新しいコマンドを起動する際には、まず、OSのforkシステムコールで新しいプロセスを作成し、その新しいプロセスがexecシステムコールで新しいコマンドを起動するという動作になります(**図A**)。ここで、forkシステムコールを省略し、単にexecシステムコールだけを行うと、新しいプロセスが作成されず、現在実行中のプロセス自身が新しいコマンドに置き換わるという動作になります。

シェルのexecコマンドは、このようにforkシステムコールを行わずに、指定されたコマンドを直接execシステムコールで起動するコマンドです。execコマンドを実行すると、通常はシェルには戻りません。また、execコマンドでは原理的に、実行できるのは外部コマンドのみで、シェルの組み込みコマンドは実行できません。

図A コマンドの起動



ラッパースクリプトでの利用例

環境変数の設定などの前処理を行ったあとでコマンドを起動するラッパースクリプト(*wrapper script*)では、最後のコマンドの起動時にexecを使うのが普通です。**リストA**は、環境変数**LANG**を設定する前処理のあと、execで「myprog」というコマンドを起動していますが、execを使うことによりシェルのプロセスがそのままmyprogに移行するので、余分なプロセスを消費しません。

現在のシェルの入出力をリダイレクト

execの引数でコマンドを指定せず、リダイレクトのみを行えます。

図B❶のように、exec > logを実行すると、現在のシェル自体の標準出力が「log」というファイルに変更され、以降、このシェルの標準出力はすべて「log」に書き込まれるようになります。**図B❷**のように、unameコマンドを実行しても画面には表示されず、「log」に出力されます。ただし、シェルのプロンプトの$は標準エラー出力なので、ここではリダイレクトされていません。

この状態を元に戻すには、**図B❸**のように画面である**/dev/tty**に再度リダイレクトしなおします。するとシェルの標準出力が画面に戻り、再び画面に表示されるようになります。

リストA ラッパースクリプト

```
LANG=C; export LANG ……環境変数LANGをCに設定する
exec myprog "$@" ……すべての引数を引き継いでmyprogを実行する
```

図B 現在のシェルの標準出力をリダイレクト

```
$ exec > log          ❶現在のシェルの標準出力を、logというファイルにリダイレクトする
$ uname -sm           ❷試しにunameコマンドを実行すると画面には表示されずlogファイルに出力される
$ exec > /dev/tty     ❸シェルの標準出力を画面に戻す
$ uname -sm           再度unameコマンドを実行すると
Linux i686            画面に表示される
$ cat log             logファイルの中身を表示すると
Linux i686            同じunameコマンドの出力が出力されていた
```

引数0を変更して起動

Warning
Linux (bash) ○ FreeBSD ✕ Solaris ✕
この方法には制限があります。

一般に、コマンドの起動時にはそのコマンド名が「引数0」として渡されます。bashのexecコマンドでは-lや-a *name*というオプションを使えば、「引数0」の内容を変更できます。

リストBのように-lオプションを指定すると、「引数0」の先頭に「-」が付けられ、-bashとして起動されます。シェルは、「引数0」の先頭に-があると、ログインシェルの動作となるため、bashの場合、**.bash_profile**ファイルの読み込みなどが行われます。

また、**リストC**のように、-aオプションで、「引数0」に任意の値を指定することもできます。リストCでは、「myprog」というコマンドを実行する際、「othername」という名前を「引数0」にした状態で起動しています。なお、前述のリストBは「exec -a -bash bash」と記述したのと同じことになります。

リストB ログインシェルとして起動

```
exec -l bash ……bashをログインシェルとして起動（引数0は-bash）
```

リストC 引数0を任意の名前に変更して起動

```
exec -a othername myprog
```

環境変数をすべて削除して起動

Warning
Linux (bash) ○ FreeBSD ✕ Solaris ✕
この方法には制限があります。

bashのexecコマンドには-cというオプションがあり、**リストD**のように-cを付けると、すべての環境変数が削除された状態でコマンドがexecされます。これは、-cオプションが使えないbash以外のシェルでも、envコマンドを経由して「exec env - myprog」のようにすることで、同様の動作が可能です。

リストD -cオプションを付けて起動

```
exec -c myprog ……環境変数をすべて削除してmyprogをexecする
```

注意事項

サブシェル内のexecはシェルスクリプトを終了しない

execコマンドを含むリストがサブシェルの( )で囲まれている場合、実際にexecが行われるのはそのサブシェル自身となるため、シェルスクリプト自体は終了しません。したがって、次の例は、普通にコマンドを実行したのとほぼ同じになります。

```
(exec ls -l) ……サブシェル内でlsコマンドをexec
```

参照

標準出力のリダイレクト（p.243）　サブシェル（p.72）

exit

Linux (bash) ○ FreeBSD (sh) ○ Solaris (sh) ○

シェルスクリプトを終了する

書式 exit [終了ステータス]

例

```
exit 1 ……終了ステータス1でシェルスクリプトを終了
```

基本事項

exitコマンドを実行すると、その時点でシェルスクリプトが終了します。引数の終了ステータスで終了ステータスを指定できます。

終了ステータス

exitコマンドに付けられた引数の値（終了ステータス）が、シェルスクリプトの終了ステータスになります。引数を省略した場合は、最後に実行されたリストの終了ステータスがシェルスクリプトの終了ステータスになります。

解説

exitコマンドは、おもにエラーなどでシェルスクリプトの実行を途中で中断する場合などに用いられます。なお、明示的にexitコマンドを記述しなくても、シェルスクリプトの下端まで実行が終了した場合は、そこでシェルスクリプトは終了し、その終了ステータスは最後に実行したリストの終了ステータスになります。これは、シェルスクリプトの下端に、暗黙のexit $?というコマンドが記述されているものと考えられます。

エラーでexitする例

リストAは、あらかじめシェルスクリプトの「引数1」で指定されたファイルが、通常ファイルとして存在するかどうかをチェックし、存在しない場合はエラーメッセージのあと、exit 1で終了ステータス「1」を返してシェルスクリプトを中断するようにしたものです。なお、リストの最後のexit 0は必ずしも必要ありませんが、シェルスクリプトの最後で正常終了する場合にはexit 0と記述しておくとわかりやすいでしょう。

リストA エラーでexitする例

```
if [ ! -f "$1" ]; then ……引数1で指定されたファイルの存在をチェック
  echo "$1"'ファイルが存在しません' ……存在しなければエラーメッセージを出す
  exit 1 ……終了ステータス1でシェルスクリプトを中断
fi ……if文の終了
cp -p "$1" "$1".bak ……メインの処理を行う
exit 0 ……最後に終了ステータス0で終了
```

注意事項

サブシェル内のexitはサブシェルを抜けるだけ

exitコマンドを含むリストがサブシェルの( )で囲まれている場合、exitコマンドを実行するとそのサブシェルを抜けますが、シェルスクリプト自体は終了しません。したがって、次の例はそれぞれtrueコマンド、falseコマンドと等価になります。

```
(exit 0) ……trueコマンドと等価
(exit 1) ……falseコマンドと等価
```

このことを利用して、以下のように0から255までの任意の終了ステータスを返すコマンドを作ることができます。

```
(exit 12) ……終了ステータス12を返すコマンド
(exit 255) ……終了ステータス255を返すコマンド
```

パイプ中のexitは暗黙のサブシェル

特殊な例ですが、パイプ(|)を1つ以上使用したパイプラインは内部的にサブシェル扱いになるため、次の例ではシェルスクリプトは終了せず、単に終了ステータス「2」が返るのみになります。

```
true | exit 2
```

参照

コマンドの終了ステータス(p.25)　特殊パラメータ$?(p.173)　サブシェル(p.72)

export

Linux (bash)
FreeBSD (sh)
Solaris (sh)

シェル変数を環境変数としてエクスポートする

書式 export [変数名 ...]

例

```
LANG=ja_JP.eucJP ……シェル変数LANGにja_JP.eucJPを代入
export LANG ……シェル変数LANGを、環境変数LANGとしてエクスポート
```

基本事項

exportコマンドを実行すると、引数の変数名で指定されたシェル変数が**環境変数**としてエクスポートされます。exportコマンドを引数なしで実行した場合は、現在エクスポート中の環境変数の一覧が表示されます。

終了ステータス

exportコマンドの終了ステータスは「0」になります。ただし、変数名の指定が正しくないなどexportコマンド自体がエラーになった場合は、終了ステータスは「0」以外になります。

解説

シェルでは、環境変数はエクスポートされたシェル変数として扱われます。したがって、環境変数を設定するには、**シェル変数に値を代入**し、exportコマンドで**環境変数としてエクスポート**するという2段階の動作が必要です。以後、エクスポートされたシェル変数の値を更新すると、同時に環境変数の値も更新されます。

シェル変数を環境変数にエクスポート

実際にシェル変数を環境変数にエクスポートしている例を**図A**に示します。printenvコマンドを使用すると、現在設定されている環境変数を確認できます。シェル変数**TEXT**に値を代入した直後は、printenv TEXTで値が表示されず、export TEXTを実行してはじめてprintenv TEXTで値が表示されていることがわかります。

図A　シェル変数を環境変数にエクスポート

```
$ TEXT=hello ……シェル変数TEXTにhelloという文字列を代入
$ echo "$TEXT" ……試しにechoコマンドでシェル変数の値を表示
hello ……たしかにhelloと表示される
$ printenv TEXT ……しかし、環境変数TEXTは設定されていない
$ export TEXT ……シェル変数TEXTを環境変数にエクスポート
$ printenv TEXT ……再度、環境変数TEXTの値を表示してみる
hello ……たしかにhelloと表示される
```

exportは代入の前でもよい

リストAのように、変数のexportを先に行い、あとから変数に値を代入してもかまいません。exportコマンドの実行時にシェル変数が未定義であっても問題ありません。

複数の変数を同時にエクスポート

リストBのように、exportの引数に複数の変数を記述し、同時にエクスポートすることもできます。多数の環境変数を設定する場合に便利でしょう。

unsetでエクスポートの取り消し

いったんエクスポートした環境変数を取り消し、エクスポート前の状態に戻すには、**図B**のようにunsetコマンドを使います。ただし、unsetによってシェル変数自体が**未設定**の状態に戻るため、シェル変数が必要な場合は、もう一度シェル変数を設定しなおす必要があります。

リストA exportは代入の前でもよい

```
export LANG ……先にシェル変数LANGを環境変数にエクスポート
LANG=ja_JP.eucJP ……LANGに値を代入
```

リストB 複数の変数を同時にexportしてもよい

```
CFLAGS='-O2 -fomit-frame-pointer' ……シェル変数CFLAGSに値を代入
LANG=ja_JP.eucJP ……シェル変数LANGに値を代入
export CFLAGS LANG ……CFLAGSとLANGをまとめてエクスポート
```

図B unsetでexportの取り消し

```
$ TEXT=hello; export TEXT ……シェル変数TEXTに値を代入し、エクスポートする
$ printenv TEXT ……環境変数TEXTの値を表示してみる
hello ……たしかにhelloと表示される
$ unset TEXT ……変数TEXTをunsetする
$ printenv TEXT ……環境変数TEXTは未設定になる
$ echo "$TEXT" ……同時にシェル変数も未設定になる
```

変数に代入と同時にエクスポート

Warning
Linux (bash) FreeBSD (sh) ×
この方法には制限があります。

bashおよびFreeBSDのshでは、**リストC**のように、シェル変数に値を代入すると同時にエクスポートすることもできます。ただし、このように記述してしまうとSolarisのshなどの従来のshとの互換性がなくなるため、なるべく「LANG=ja_JP.eucJP; export LANG」という2段階の記述方法を用いたほうがよいでしょう。

リストC 変数に代入と同時にexport

```
export LANG=ja_JP.eucJP ……シェル変数LANGに値を代入すると同時にエクスポートする
```


export -nでエクスポートの取り消し

Warning
Linux (bash) × FreeBSD × Solaris
この方法には制限があります。

bashでは、エクスポートした環境変数を取り消すために、**図C**のようにexport -nというコマンドが使えます。

export -nは、unsetとは違ってシェル変数はセットされたまま残り、環境変数のみが未設定状態に戻ります。

図C export -nでexportの取り消し

```
$ TEXT=hello; export TEXT ……シェル変数TEXTに値を代入し、エクスポートする
$ printenv TEXT ……環境変数TEXTの値を表示してみる
hello ……たしかにhelloと表示される
$ export -n TEXT ……環境変数TEXTのエクスポートを取り消す
$ printenv TEXT ……環境変数TEXTは未設定になる
$ echo "$TEXT" ……シェル変数のほうのTEXTを表示してみる
hello ……シェル変数は値がセットされたままになっている
```

exportの-pオプション

Warning
Linux (bash) FreeBSD (sh) ×
この方法には制限があります。

エクスポートされている変数の一覧は、exportコマンドを引数なしで実行すれば表示されますが、bashやFreeBSDのshでは、ここでexport -pとオプションを付けることもできます。この-pオプションを付けると、**図D**のように、実際にエクスポートを行う際のコマンドラインを使った表示になります。ただし、bashの場合はexport -pとexportは同じで、常にコマンドライン形式の表示になり、実際にはexportに相当するdeclare -xコマンドを使った表示になります。

図D export -pの実行例(bashではexportと同じ)

```
$ export -p
declare -x HOME="/home/guest"
declare -x LANG="ja_JP.eucJP"
declare -x PATH="/home/guest/bin:/usr/local/bin:/usr/X11R6/bin:/usr/bin:/bin"
declare -x TERM="kterm"
<以下略>
```

Memo

- bashでは、export -fで、シェル関数もエクスポートできます。

参照

環境変数の設定(p.179) シェル変数の代入と参照(p.159) unset(p.131)

getopts

Linux (bash)
FreeBSD (sh)
Solaris (sh)

シェルスクリプトの引数に付けられたオプションを解析する

書式 **getopts** オプション文字列 変数名 [引数 ...]

例 `getopts cvi:o: option` ……-c、-v、-i、-oをオプションとして位置パラメータを解釈

基本事項

getoptsコマンドは位置パラメータにセットされている、-で始まる1文字オプションを解釈し、最初に見つかったオプション文字の1文字を変数名で指定したシェル変数に代入します。この時、シェル変数**OPTIND**には、次にgetoptsが実行された時に解釈するべき位置パラメータの番号が代入されます。オプション文字列には、getoptsが解釈するべき1文字オプションを並べて指定します。オプション文字の直後に:がある場合は、そのオプションが引数を必要するものと解釈され、その引数はシェル変数**OPTARG**に代入されます。

オプションではない引数まで解釈が進んだ場合は、オプションではない引数のセットされている位置パラメータの番号を**OPTIND**にセットして、「0」以外の終了ステータスで終了します。

オプションではない引数または不正なオプションが見つかった場合、指定のシェル変数には「?」という文字が代入されます。

位置パラメータ中に--という引数がセットされている場合は、それ以降の位置パラメータはオプションではないと解釈されます。

OPTINDは、初期状態では「1」がセットされていますが、新たな位置パラメータを最初から解釈させなおす場合は、**OPTIND**に再度「1」を代入する必要があります。

位置パラメータを解釈する代わりに、getoptsコマンド自体に引数を付けて、その引数を解釈させることもできます。

終了ステータス

解釈の結果、オプションが見つかった場合は終了ステータスは「0」になります。オプションではない引数まで解釈が進んだ場合は終了ステータスは「0」以外になります。

解説

シェルスクリプトの起動時にシェルスクリプトに付けられた「-v -o *file arg*」のような形式のオプションを解釈するには、getoptsコマンドが便利です。とくに、-vのような1文字オプションや、「-o *file*」のような1文字オプションとその引数の形に形式化している場合はgetoptsが使いやすいでしょう。ただし、オプション解釈にはgetoptsが必須というわけではなく、getoptsを使わないで独自に位置パラメータを解釈してもかまいません。

getoptsコマンドの記述例

getoptsコマンドを使用したシェルスクリプトの記述例を**リストA**に示します。getoptsは、オプション解釈が終了するまで繰り返し呼び出して使用するため、このようにwhile文と組み合わせて使うのが普通です。

getoptsのオプション文字列にはcvi:o:と指定しています。したがって、-c、-vのオプションと、-i *file*および-o *file*の形のオプションを扱うことになります。

getoptsの実行後、指定のシェル変数であるoptionにはオプション文字1文字が代入されているので、これをcase文を使って場合分けします。cやvの場合は単にメッセージを表示します。-iや-oの場合は、その後の引数が**OPTARG**に代入されているため、これも含めて表示します。それ以外のオプションが指定された場合はoptionには?が代入されているため、「Usage」のメッセージを表示してエラーで終了します。この時、case文のパターンとしての?は\でクォートする必要があります。

以上の処理を、位置パラメータのオプションが続くかぎり繰り返したあと、オプションでない引数に達したところでgetoptsコマンドの終了ステータスが偽になるため、while文が終了します。

リストA getopts_test（getoptsの記述例）

```
while getopts cvi:o: option ……オプション解釈が続くかぎりwhile文でループする
do ……while文のループの開始
  case $option in ……case文で得られたオプション名で分岐
    c) ……cオプションだった場合
      echo '-cオプションが指定されました';; ……その旨を表示
    v) ……vオプションだった場合
      echo '-vオプションが指定されました';; ……その旨を表示
    i) ……iオプションだった場合
      echo '-iオプションで'"$OPTARG"'が指定されました';; ……
                                        オプション引数も含めその旨を表示
    o) ……oオプションだった場合
      echo '-oオプションで'"$OPTARG"'が指定されました';; ……
                                        オプション引数も含めその旨を表示
    \?) ……不正なオプションが指定された場合
      echo "Usage: $0 [-c] [-v] [-i file] [-o file] [args...]" 1>&2……
                                        Usageのエラーメッセージを表示
      exit 1;; ……エラーで終了
    esac ……case文の終了
done ……while文のループの終了
shift `expr "$OPTIND" - 1` ……OPTINDから1を引いた数だけ位置パラメータをshift
if [ $# -ge 1 ]; then ……位置パラメータが1つ以上残っている場合
  echo 'オプション以外の引数は'"$@"'です' ……残りの位置パラメータを表示
else ……位置パラメータが残っていない場合
  echo 'オプション以外の引数はありません' ……その旨を表示
fi ……if文の終了
```

真の値を返すため、キー入力が続くかぎり、whileループが繰り返し実行されることになります。

リストAの実行例はselect文の項の実行例と同じになります。実際にシェルスクリプトを起動し、メニューを見て適当な数字を入力すれば、動作が確認できるでしょう。

なお、Solarisのshではechoコマンドの-nオプションが使えないため、プロンプトを表示しているecho -nをechoに変更し、代わりにプロンプトの文字列の最後に\cを付けて改行を抑制してください。

リストA ユーザからの入力に応じてメッセージを表示

```
while ……while文の開始
  echo -n \ ……echoコマンドで選択メニューとプロンプト（改行なし）を表示
'1) up ……シングルクォートで囲まれた選択メニューとプロンプト
2) down
3) left
4) right
5) look
6) quit
コマンド? ' 1>&2 ……全体を標準エラー出力にリダイレクト
  read cmd ……シェル変数cmdに標準入力を読み込む
do ……while文のループの開始
  case $cmd in ……case文を使ってシェル変数cmdの内容で分岐
    1) ……入力が1だった場合
      echo '上に移動しました';; ……対応するメッセージを表示
    2) ……入力が2だった場合
      echo '下に移動しました';; ……対応するメッセージを表示
    3) ……入力が3だった場合
      echo '左に移動しました';; ……対応するメッセージを表示
    4) ……入力が4だった場合
      echo '右に移動しました';; ……対応するメッセージを表示
    5) ……入力が5だった場合
      echo 'アイテムが落ちています';; ……対応するメッセージを表示
    6) ……入力が6だった場合
      echo '終了します' ……終了メッセージを表示
      break;; ……while文のループを抜けて終了する
    *) ……入力がそれ以外の文字列だった場合
      echo "$cmd"'というコマンドはありません';; ……
                                    入力文字列を含めてエラーメッセージを表示
  esac ……case文の終了
  echo ……1行改行
done ……while文のループの終了
```


readの-pオプションでプロンプトを表示

bashおよびFreeBSDのshのreadコマンドには、プロンプトを指定する-pオプションがあり、echoコマンドを使わずにプロンプトを表示できます。プロンプトは標準エラー出力に出力されます。前述のリストAをread -pを使って書き直すとリストBのようになります。

リストB read -pでプロンプトを表示

```
while ……while文の開始
  read -p \ ……read -pで選択メニューとプロンプト（改行なし）を表示
'1) up ……シングルクォートで囲まれた選択メニューとプロンプト
2) down
3) left
4) right
5) look
6) quit
コマンド? ' cmd ……シェル変数cmdに標準入力を読み込む
do ……while文のループの開始
  case $cmd in ……case文を使ってシェル変数cmdの内容で分岐
    1) ……入力が1だった場合
      echo '上に移動しました';; ……対応するメッセージを表示
    2) ……入力が2だった場合
      echo '下に移動しました';; ……対応するメッセージを表示
    3) ……入力が3だった場合
      echo '左に移動しました';; ……対応するメッセージを表示
    4) ……入力が4だった場合
      echo '右に移動しました';; ……対応するメッセージを表示
    5) ……入力が5だった場合
      echo 'アイテムが落ちています';; ……対応するメッセージを表示
    6) ……入力が6だった場合
      echo '終了します' ……終了メッセージを表示
      break;; ……while文のループを抜けて終了する
    *) ……入力がそれ以外の文字列だった場合
      echo "$cmd"'というコマンドはありません';; ……
                                    入力文字列を含めてエラーメッセージを表示
  esac ……case文の終了
  echo ……1行改行
done ……while文のループの終了
```

そのほか、bashおよびFreeBSDのshのreadコマンドでは、バックスラッシュ(\)による行の継続を行わず、\を普通の文字として入力する-rオプションや、一定時間入力がなかった場合にreadコマンドを終了ステータス1で終了する-tオプションが使えます。-tオプションは待ち時間の秒数を引数で指定し、たとえば5秒待ちならばread -t 5と指定します。

readの-nオプションと-sオプション

Warning

Linux (bash) ✕ FreeBSD ✕ Solaris

この方法には制限があります。

bashのreadコマンドでは、-nオプションで最大入力文字数(バイト数)を指定することができます。たとえばread -n 5の場合、標準入力から5バイト入力したところで、行の途中であってもreadの入力動作を終了します。-sオプションを指定すると、端末からの入力の際に、入力文字をエコーバックしません。簡単なパスワードを入力するような用途などに便利でしょう。

Memo

- bashでは、readコマンドの引数の変数名を省略することができ、その場合、シェル変数REPLYに入力行が代入されます。
- bashのreadコマンドには、ほかにもオプションが存在します。

参照

while文(p.63)　　単語分割(p.237)

readonly

Linux (bash)　FreeBSD (sh)　Solaris (sh)

シェル変数を読み込み専用にする

書式 readonly [変数名 ...]

例

```
DIR=/usr/local ..... シェル変数DIRに/usr/localを代入
readonly DIR ..... シェル変数DIRを読み込み専用にする
```

基本事項

readonlyコマンドを実行すると、引数の変数名で指定されたシェル変数が読み込み専用となり、以後、値の代入もunsetもできなくなります。readonlyコマンドを引数なしで実行した場合は、読み込み専用になっている変数の一覧が表示されます。

終了ステータス

readonlyコマンドの終了ステータスは「0」になります。ただし、変数名の指定が正しくないなど、readonlyコマンド自体がエラーになった場合は終了ステータスは「0」以外になります。

解説

readonlyコマンドは、シェル変数を読み込み専用にするだけであり、とくにreadonlyを使用しなくてもシェルスクリプトは記述できます。しかし、**特定のシェル変数を定数**として使用したい場合は、readonlyを実行しておいたほうがよいでしょう。定数扱いの変数が読み込み専用に設定されていれば、誤って値を変更したりunsetするといったプログラムミスを防げます。なお、いったん読み込み専用に設定された変数を元に戻す方法はありません。

代入もunsetもできなくなる

readonlyコマンドが実行されると、**図A**のように、その変数に対して代入やunsetを実行しようとするとエラーになります。

図A　代入もunsetもできない

```
$ DIR=/usr/local                                    DIRに値を代入する
$ readonly DIR                                      DIRを読み込み専用にする
$ DIR=/tmp                                          DIRに値を代入しようとすると
bash: DIR: readonly variable                        エラーになる
$ unset DIR                                         DIRをunsetしようとすると
bash: unset: DIR: cannot unset: readonly variable   エラーになる
```

複数の変数を同時に読み込み専用に

リストAのように、readonlyの引数に複数の変数を記述し、同時に読み込み専用にすることもできます。多数の変数を読み込み専用に設定する場合に便利でしょう。

代入の前に読み込み専用にした場合

図Bのように、まだ設定していないシェル変数に対してreadonlyを実行することも、文法的には可能です。すると、以降この変数を設定することすらできなくなります。

リストA readonlyで、複数の変数を同時に指定

```
DIR=/usr/local ……シェル変数DIRに値を代入
PROG=myprog ……シェル変数PROGに値を代入
readonly DIR PROG ……DIRとPROGをまとめて読み込み専用に
```

図B 設定していないシェル変数を読み込み専用に

```
$ unset DIR ……念のためシェル変数DIRをunsetする
$ readonly DIR ……未設定のDIRを読み込み専用にする
$ DIR=/usr/local ……DIRに値を代入しようとすると
bash: DIR: readonly variable ……エラーになる
```

変数に代入と同時にreadonlyにする

Warning

Linux (bash) ○ FreeBSD (sh) × Solaris

この方法には制限があります。

bashおよびFreeBSDのshでは、**リストB**のように、シェル変数に値を代入すると同時に読み込み専用にすることもできます。ただし、このように記述してしまうとSolarisのshなどの従来のshとの互換性がなくなるため、なるべく「DIR=/usr/local; readonly DIR」という2段階の記述方法を用いたほうがよいでしょう。

リストB 変数に代入と同時にreadonlyにする

```
readonly DIR=/usr/local …… シェル変数DIRに値を代入すると同時に読み込み専用にする
```


readonlyの-pオプション

Warning

Linux (bash) ○ FreeBSD (sh) × Solaris

この方法には制限があります。

読み込み専用に設定されている変数の一覧は、readonlyコマンドを引数なしで実行すれば表示されますが、bashやFreeBSDのshでは、ここでreadonly -pとオプションを付けることもできます。この-pオプションを付けると、**図C**のように、実際にエクスポートを行う際のコマンドラインを使った表示になります。ただし、bashの場合はreadonly -pとreadonlyは同じで、常にコマンドライン形式の表示になり、実際にはreadonlyに相当するdeclare -rコマンドを使った表示になります。

図C readonly -pの実行例(bashではreadonlyと同じ)

```
$ readonly -p
declare -r BASHOPTS="cmdhist:expand_aliases:extquote:force_fignore:hostcomple
te:interactive_comments:progcomp:promptvars:sourcepath"
declare -ir BASHPID=""
declare -ar BASH_VERSINFO='([0]="4" [1]="1" [2]="7" [3]="1" [4]="release"
[5]="i686-redhat-linux-gnu")'
declare -ir EUID="1000"
declare -ir PPID="14500"
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:interactiv
e-comments:monitor"
declare -ir UID="1000"
```

注意事項

位置パラメータ、特殊パラメータはreadonlyにできない

readonlyコマンドの引数に指定できるのは、パラメータのうちの「シェル変数」のみです。位置パラメータ($1、$2など)や特殊パラメータ($@、$#など)はreadonlyの引数には指定できません。

サブシェル内でのreadonlyはサブシェル内でのみ有効

readonlyコマンドが( )で囲まれたサブシェル内で実行されている場合、読み込み専用になるのはそのサブシェルのみになります。シェル本体の変数は影響を受けないため注意してください。

Memo

- bashのreadonlyには、配列やシェル関数に関連した-aや-fのオプションもあります。

参照

シェル変数の代入と参照(p.159) unset(p.131) サブシェル(p.72)

return

Linux (bash)
FreeBSD (sh)
Solaris (sh)

シェル関数を終了する

書式 **return** [終了ステータス]

例

```
func() ……シェル関数の定義開始
{
  return 1 ……終了ステータス1でシェル関数を終了
} ……シェル関数の定義終了
```

基本事項

returnコマンドを実行すると、実行中のシェル関数を終了し、シェル関数の呼び出し元に戻ります。引数の終了ステータスで終了ステータスを指定できます。

終了ステータス

returnコマンドに付けられた引数の値が、シェル関数の終了ステータスになります。引数を省略した場合は、シェル関数内で最後に実行されたリストの終了ステータスがシェル関数の終了ステータスになります。

解説

returnコマンドは、おもにエラーなどでシェル関数の実行を途中で中断する場合に用いられます。なお、明示的にreturnコマンドを記述しなくても、シェル関数の最後まで実行が終了した場合は、そこでシェル関数は終了し、その終了ステータスは最後に実行したリストの終了ステータスになります。これは、シェル関数の最後に、暗黙のreturn $?というコマンドが記述されているものと考えられます[注8]。

エラーでreturnする例

リストAは、あらかじめシェル関数の「引数1」で指定されたファイルが、通常ファイルとして存在するかどうかをチェックし、存在しない場合はエラーメッセージのあと、return 1で終了ステータス「1」を返してシェル関数を中断するようにしたものです。なお、リストの最後のreturn 0は必ずしも必要ありません。

注8 「コマンドの終了ステータス」(p.25)と「特殊パラメータ$?」(p.173)も合わせて参照してください。

リストA エラーでreturnする例

```
func() ……シェル関数funcの定義開始
{
  if [ ! -f "$1" ]; then ……引数1で指定されたファイルの存在をチェック
    echo "$1"'ファイルが存在しません' ……存在しなければエラーメッセージを出す
    return 1 ……終了ステータス1でシェル関数を中断
  fi ……if文の終了
  cp -p "$1" "$1".bak ……メインの処理を行う
  return 0 ……最後に終了ステータス0で終了
} ……シェル関数funcの定義終了
```

.コマンドで読み込んだファイル中でのreturn

bashまたはFreeBSDのshでは、.コマンドで読み込んだファイル中でreturnコマンドを実行すると、読み込んだファイルの実行を終了することができます。

Warning
Linux (bash) FreeBSD (sh) Solaris (sh)
この方法には制限があります。

注意事項

サブシェル内のreturnはサブシェルを抜けるだけ

シェル関数内でreturnコマンドを含むリストがサブシェルの( )で囲まれている場合、returnコマンドを実行してもそのサブシェルを抜けるだけになり、シェル関数は終了しません。したがって、次の例ではreturnの後のechoコマンドまで実行されてしまい、終了ステータスもechoによって「0」が返されます。

```
func() { (return 3); echo message; }
```

パイプ中のreturnは暗黙のサブシェル

特殊な例ですが、パイプ(|)を1つ以上使用したパイプラインは内部的にサブシェル扱いになるため、次の例ではシェル関数は終了せず、returnの後のechoコマンドまでが実行されます。

```
func() { true | return 3; echo message; }
```

参照

シェル関数(p.76)
.コマンド(p.90)
コマンドの終了ステータス(p.25)
サブシェル(p.72)
特殊パラメータ$?(p.173)

set

シェルにオプションフラグをセットする、または位置パラメータをセットする

○ Linux (bash)
○ FreeBSD (sh)
○ Solaris (sh)

書式 set [-オプションフラグ | +オプションフラグ] [引数 ...]

例

```
set -a ..... シェルに-aのオプションフラグをセット
set one two ..... 位置パラメータ$1、$2に、それぞれone twoをセット
```

表 setコマンドでセットできるオプションフラグ

フラグ	意味	Linux (bash)	FreeBSD (sh)	Solairs (sh)
-a	変数に代入すると自動的にエクスポートされる	○	○	○
-b	バックグラウンドジョブが終了したらすぐに報告	○	×	×
-e	コマンドが偽の終了ステータスで終了するとシェルを終了	○	○	○
-f	パス名展開を行わない	○	○	○
-h	コマンド実行時にハッシュテーブルを使う	○	×	○※3
-k	コマンド名の右側でも環境変数に代入できる	○	×	○
-m	ジョブコントロールを有効にする	○	○	○
-n	コマンドを読み込むのみで、実際には実行しない	○	○	○
-o	各種拡張オプションをセットする	○	○	×
-p	特権モードを有効にする	○	○	×
-t	コマンドを1つだけ実行してシェルを終了する	○	×	○
-u	未設定のパラメータの参照をエラーとして扱う	○	○	○
-v※1	コマンド入力時に、コマンド入力行をそのまま表示	○	○	○
-x※1	コマンド実行時に、展開後のコマンド行を表示	○	○	○
-B	ブレース展開を有効にする	○	×	×
-C	リダイレクト時に、存在するファイルを上書きしない	○	○	×
-H	!によるヒストリ置換を有効にする	○	×	×
-P	cdやpwdコマンドで、常に-Pオプションを使用する	○	○	×
-	-vと-xフラグを削除し、残りの引数を位置パラメータに代入	○	○	○
--	残りの引数を位置パラメータに代入※2	○	○	○

※1 作成したシェルスクリプトの記述の確認や動作の確認をしたい場合、-vや-xオプションを利用できる。シェルスクリプト内にset -xのように記述する。また別の確認方法として、コマンドラインでsh -x [スクリプト名]のようにシェルを実行して確認することもできる

※2 bashやFreeBSDのshでは、set --で、残りの引数がない場合は位置パラメータは削除される。Solarisのshではset --で残りの引数がなくても位置パラメータは削除されない。いずれも、set -で残りの引数がない場合は位置パラメータはそのままになる

※3 Solarisのshの-hは少し意味が異なり、-hを設定すると、シェル関数の中で使われている外部コマンドを、シェル関数の定義の時点でハッシュテーブルに登録するようになる

基本事項

setコマンドを[-オプションフラグ]の形式で実行すると、該当のオプションフラグがセットされます。setコマンドを[+オプションフラグ]の形式で実行すると、該当のオプションフラグはリセットされます。オプションフラグとしては**表A**のものが使用できます。

setコマンドに、-や+で始まらない[引数]を付けて実行した場合、あるいは、-または--のオプションフラグに続いて[引数]を指定した場合は、それらの[引数]が順に位置パラメータにセットされます。

setコマンドを引数なしで実行した場合はすべてのシェル変数とその値が一覧表示されます。

終了ステータス

エラーが発生しないかぎり、終了ステータスは「0」になります。

解説

setコマンドには「オプションフラグのセット／リセット」「位置パラメータのセット」「シェル変数の一覧表示」という、異なる3種類の使用法があります。これらを順に説明します。

オプションフラグのセット

オプションフラグとしては**表A**の各オプションフラグが使用できます。シェルによって使用できるフラグに若干の違いがあるため、移植性には注意してください。オプションフラグをセットすると、そのフラグによってシェルの動作が一部変更されます。

図Aは、-fをセットして、パス名展開を禁止する実行例です。なお、現在シェルに設定されているオプションフラグの状態は、特殊パラメータ$-で参照できます。また、setで設定できるオプションフラグは、シェル自身の起動時のオプションとしても指定できます。たとえばシェルスクリプトの冒頭でset -fを実行する代わりに、1行目に#!/bin/sh -fと記述しておくことができます。

位置パラメータのセット

setコマンドを使って、位置パラメータをセットしている例を**図B**に示します。このように、setコマンドの引数として、-や+で始まらない「one two three」のような文字列を直接引数にする場合は、これらがオプションフラグと誤認されることはないため、そのまま位置パラメータに代入されます。

一方、-aなどのように、先頭に-が付いている文字列を位置パラメータにセットしたい場合は、set -- -aとして、-aがオプションフラグとはみなされないようにする必要があります。とくに、図Bのように、シェル変数の値を位置パラメータに代入する場合には、setではなくset --を使うように注意してください。

シェル変数の一覧表示

setコマンドを引数なしで実行し、シェル変数の一覧を表示している例を**図C**に示します。このように、すべてのシェル変数がその値とともに表示されることがわかります。

図A オプションフラグのセットの例

```
$ echo *                          *をechoしてみる
bin memo.txt src tmp              カレントディレクトリのファイル名に展開される
$ set -f                          -fオプションフラグで、パス名展開を禁止する
$ echo *                          再び*をechoする
*                                 *のまま展開されないことがわかる
$ set +f                          +fで、-fオプションフラグをリセットし、パス名展開を有効にする
$ echo *                          再び*をechoする
bin memo.txt src tmp              元通りパス名展開される
```

図B 位置パラメータのセットの例

```
$ set one two three               位置パラメータに"$1"から順にone two threeをセット
$ echo "$2"                       試しに"$2"を表示
two                               たしかにtwoと表示される
$ opt=-a                          シェル変数optに-aという文字列を代入
$ set -- "$opt"                   "$opt"を位置パラメータにセット(set --を使う)
$ echo "$1"                       "$1"の内容を表示
-a                                無事-aという文字列が表示される
```

図C シェル変数の一覧表示の例

```
$ set                             setコマンドを引数なしで実行
BASH=/bin/bash                    以下、シェル変数名とその値が一覧表示される
BASHOPTS=cmdhist:expand_aliases:extquote:force_fignore:hostcomplete:interactive_
comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="1" [2]="7" [3]="1" [4]="release" [5]="i686-redhat-
linux-gnu")
     ⋮
<以下略>
```

参照

位置パラメータ(p.162)　　特殊パラメータ$-(p.177)

shift

Linux (bash)
FreeBSD (sh)
Solaris (sh)

位置パラメータをシフトする

書式 shift [シフト回数]

例

```
while [ $# -gt 0 ] ……残りの引数があるかぎりループ
do ……ループの開始
  echo "$1" ……引数1を表示する
  shift ……引数をシフトする
done ……ループの終了
```

基本事項

shiftコマンドを引数なしで実行すると、位置パラメータ"$2"が新たに"$1"に、"$3"が新たに"$2"に、というように順にシフトされます。引数でシフト回数を指定した場合は、その回数分だけシフトされます。シフト回数が「0」の場合は位置パラメータは変化しません。

終了ステータス

shiftコマンドの終了ステータスは「0」になります。ただし、シフト回数が引数の個数を超えていてシフトできない場合は終了ステータスは「0」以外になります。

解説

shiftコマンドは、おもにwhile文で、引数(位置パラメータ)を順に解釈しながらループする場合に使用されます。"$1"に対する解釈が終わった時点でshiftを実行すれば、次の引数が"$1"のところにシフトしてくるので、効率よく処理が行えます。

引数シフトの例

setコマンドを使って適当な位置パラメータをセットし、それをシフトしながら位置パラメータの様子を表示させた例を**図A**に示します。

すべての引数をシフトするには

特殊パラメータ$#には、位置パラメータの個数が入っています。したがって、すべての引数をシフトして、引数が何もセットされていない状態にするには、**図B**のようにshift $#と記述すればよいのです。シフトの結果、$#の値は「0」になります[注9]。

注9 bashやFreeBSDのshでは、set --でも位置パラメータを未設定状態にできます。

図A 引数シフトの例

```
$ set one two three four five          setコマンドで適当な引数を5個セットする
$ echo "$@"                             現在の位置パラメータを表示
one two three four five                たしかに5個そのまま表示される
$ shift                                1回シフト
$ echo "$@"                             現在の位置パラメータを表示
two three four five                    "$1"がなくなり、"$2"から先がシフトしている
$ shift 2                              さらに2回シフト
$ echo "$@"                             現在の位置パラメータを表示
four five                              たしかに2回シフトしている
```

図B すべての引数をシフトする例

```
$ set 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5    setコマンドで適当な引数を15個セットする
$ echo $#                              念のため$#の値を表示
15                                     たしかに15と表示される
$ shift $#                             すべての引数をシフトする
$ echo $#                              再度$#の値を表示
0                                      引数がなくなり、$#の値は0となる
```

引数の個数以上にシフトしようとした場合

シフト回数が実際の引数の個数を超えていてシフトできない場合の動作は、シェルによって違いがあります。bashやFreeBSDのshでは、位置パラメータの状態は変化しません。一方、Solarisのshでは、すべての位置パラメータがシフトし、位置パラメータ未設定の状態になります。

いずれにしても、引数の個数を超えたシフトはshiftコマンドにとってはエラーの動作であり、このような状態にならないようにシェルスクリプトを記述する必要があります。

注意事項

サブシェル内やシェル関数内でのshiftは、それらの中でのみ有効

shiftコマンドが( )で囲まれたサブシェル内やシェル関数内で実行された場合、位置パラメータがシフトするのはそのサブシェルやシェル関数内のみになります。シェル本体の位置パラメータはシフトされないため、注意してください。

Memo

- shift 0は結果的に:コマンドと同じですが、:コマンドとは違ってshift 0の後に余分な引数は付けられません。

参照

位置パラメータ(p.162)　while文(p.63)　set(p.120)
特殊パラメータ$#(p.171)　サブシェル(p.72)　シェル関数(p.76)

trap

Linux (bash)
FreeBSD (sh)
Solaris (sh)

シグナルを受け取った時に指定のコマンドを実行させる

書式 **trap** [コマンド] [シグナル番号 ...]

例 `trap 'echo 割り込みシグナル受信' 2` ……SIGINT受信時にメッセージを表示

基本事項

trapコマンドを実行すると、シェルが引数のシグナル番号で指定したシグナルを受信した際に、指定のコマンドを実行するように設定されます。引数のコマンドとして空文字列を指定した場合は、指定のシグナルがシェルによって無視されるようになります。引数のコマンドを省略した場合は、指定のシグナルの設定が解除されます。trapコマンドを引数なしで実行した場合は、現在のtrapの設定が表示されます。

シグナル番号として「0」を指定すると、シェルの終了時に指定のコマンドが実行されます。

trapの引数のコマンドは、1つの引数として与える必要があり、trapコマンドの実行時に一度解釈されたあと、シグナルの受信後のコマンドの実行時に再度解釈されます。

終了ステータス

エラーが発生しないかぎり、終了ステータスは「0」になります。

解説

シェルスクリプトを起動後、通常は、キーボードからCtrl+Cを入力して2番のシグナル(SIGINT)を発生させたり、ほかの端末エミュレータなどからkillコマンドでシグナルを送ることによってシェルスクリプトを中断させたりできます。trapコマンドは、この、シグナル受信時の動作を変更し、シグナル受信時に何らかのコマンドを実行させたり、あるいはシグナルを無視させたりすることが可能です。たとえば、あらかじめtrap '' 2というコマンドを実行しているシェルスクリプトは、Ctrl+Cを入力しても終了しません。ただし、9番のSIGKILLのように、無視することができないシグナルもあります。なお、シグナル一覧はkill -lを実行して表示できます[注10]。

シグナル受信の例

図Aは、コマンドライン上でtrapを設定したあと、自分自身のシェルにkillコマンドでシグナルを送っている例です。ここでは、シグナル番号「2」(SIGINT)に対し、シェル変数messageの内容を表示するというtrapを設定した上で、killコマンドで、シェル自身のプロセスID番号の入った特殊パラメータ$$を参照してシグナルを送っています。

注10 kill -lの実行例は、「kill」の項目(p.141)を参照してください。

すると、図A❶❷のように、シグナル受信時にメッセージが表示されます。また、シェル変数messageの内容は、実際にシグナルを受信したあとに参照されていることもわかります。なお、図A❸で最後にtrap 2を実行してtrapを解除し、元の状態に戻しています。

シグナル受信時にテンポラリファイルを削除

テンポラリファイルを使用するシェルスクリプトの場合、シグナルを受信してシェルスクリプトを中断する際に、そのテンポラリファイルを削除する必要があります。そのような場合、**リストA**のようなtrapコマンドを、シェルスクリプトのはじめのほうに書いておけば、シグナルの受信時にテンポラリファイルが削除できます。ここでは、念のため終了時にメッセージも表示するようにしています。

リストでは、trapコマンドの第1引数としてrm、echo、exitの3つのコマンドを、全体をシングルクォート(' ')で囲んで1つの引数として与え、さらに、わかりやすいようにシングルクォート中で改行していることに注意してください。なお、シェル変数**TMPFILE**には、シェルスクリプト中で適切なテンポラリファイル名が代入されているものとします。

また、trapを設定するシグナル番号は、1(**SIGHUP**)、2(**SIGINT**)、3(**SIGQUIT**)、15(**SIGTERM**)としていますが、大抵の場合これで十分でしょう。

図A シグナル受信の例

```
$ trap 'echo "$message"' 2          2番のシグナル（SIGINT）にtrapを設定
$ trap                              現在のtrapの一覧を表示
trap -- 'echo "$message"' SIGINT    たしかにSIGINTが設定されている
$ message='trap test'               シェル変数messageに、適当なメッセージを代入
$ kill -2 $$                        シェル自身のプロセス（$$）に2番のシグナルを送る
trap test                           ❶シェル変数messageの内容が表示される
$ message='hello   world'           シェル変数messageの内容を変更
$ kill -2 $$                        再び2番のシグナルを送る
hello   world                       ❷変更されたメッセージが、スペースも保存されて表示される
$ trap 2                            ❸2番のシグナルのtrapを解除
$ trap                              trapコマンドで一覧を表示させても何も表示されない
```

リストA シグナル受信時にテンポラリファイルを削除

```
trap '                              ……trapコマンドの開始（シングルクォートの途中で改行）
  rm -f "$TMPFILE"                  ……テンポラリファイルを削除
  echoシグナルにより終了します      ……メッセージを表示
  exit 1                            ……終了ステータス1で終了
' 1 2 3 15                          ……以上のコマンドを、シグナル番号1 2 3 15に対して設定
```

Memo

- シグナル番号を指定する代わりに、HUPやINTなどのシグナル名でも指定できます。
- bashのtrapコマンドには-l、-pのオプションも存在します。

参照

kill(p.141)

type

Linux (bash) / FreeBSD (sh) / Solaris (sh)

外部コマンドのフルパスを調べたり、組み込みコマンドかどうかをチェックしたりする

書式 type [コマンド名 ...]

基本事項

typeコマンドは、引数のコマンド名で指定されたコマンドが外部コマンドか、シェルの組み込みコマンドか、あるいはシェル関数かを判断し、その旨を表示します。外部コマンドの場合は**PATH**から検索され、そのコマンドのフルパスが表示されます[注11]。

終了ステータス

引数で指定されたコマンドが存在する場合は「0」、存在しない場合は「0」以外になります[注12]。

解説

typeコマンドは、与えられたシェルやOSの環境において、各種コマンドが実装されているかどうか、また、外部コマンドの場合は**PATH**上のどのディレクトリのものが使用されているかをチェックするために使います。実際には、シェルスクリプト中ではなく、おもにコマンドライン上で使用されます。コマンドの状況を調べている様子を**図A**に示します。

図A 各種コマンドの状況を調べる

```
$ type cp                           cpコマンドについて調べる
cp is /bin/cp                       cpは/bin/cpに存在する外部コマンド
$ type echo                         echoコマンドについて調べる
echo is a shell builtin             echoはシェル組み込みコマンド
$ type func                         funcというコマンドについて調べる
bash: type: func: not found         funcというコマンドはない
$ func() { echo Hello;}             funcというシェル関数を定義
$ type func                         再びfuncコマンドについて調べる
func is a function                  funcはシェル関数
func ()                             シェル関数の中身が表示される
{                                   ┊
    echo Hello                      FreeBSDのshでは、シェル関数の中身は表示されない
}
```

Memo

- bashでは、typeコマンドに-a、-f、-t、-p、-Pというオプションも存在します。
- typeコマンドは、csh系でのwhichコマンドに相当します。

注11 bashやFreeBSDのshでは、aliasやshell keywordといった表示になる場合もあります。
注12 SunOS 4.xのshなどでは、終了ステータスは常に「0」になります。

umask

Linux (bash) / FreeBSD (sh) / Solaris (sh)

シェル自体のumask値を設定／表示する

書式 umask [マスク値(8進数)]

例
```
umask 027 ……umask値を027に設定する
```

基本事項

umaskコマンドを実行すると、引数で指定された マスク値(8進数) がシェル自身の**umask値**として設定されます。umaskコマンドを引数なしで実行すると、現在のumask値を表示します。

終了ステータス

umaskコマンドの終了ステータスは「0」になります。ただし、umask値の指定が正しくないなど、umaskコマンド自体がエラーになった場合は終了ステータスは「0」以外になります。

解説

UNIX系OSでは、各プロセスが**umask値**と呼ばれる属性値を持っており、新規にファイルやディレクトリを作成する場合、そのパーミッションはumask値の影響を受けます。たとえば、新たなファイルが8進数表記の「777」(rwxrwxrwx)のパーミッションで作成されようとした時、umask値が「027」(----w-rwx)なら、これらのビットがマスクされ、その結果、作成されるファイルのパーミッションは「750」(rwxr-x---)になります。

umaskコマンドは、シェル自体のumask値を設定するためのもので、設定したumask値はシェル自身からその子プロセスにも受け継がれます。シェルスクリプトでは、たとえば**/tmp**ディレクトリ以下などにテンポラリファイルを作成する際に、そのファイルが他人から読めてしまうなどの危険を回避するため、あらかじめumask 027またはumask 077を実行しておくとよいでしょう。

umaskの使用例

umaskの使用例を**図A**に示します。このように、umask値「022」では、作成されるディレクトリのパーミッションがrwxr-xr-xになりますが、umask値「027」ではrwxr-x---に、umask値「077」ではrwx------になります[注13]。

注13 ディレクトリではなく、ファイルを作成した場合は実行属性がつかないため、たとえば、umask値「027」でrw-r-----になります。

図A umaskの使用例

```
$ umask                                                  現在のumask値を表示
0022                                                     umask値は022
$ mkdir work                                             新規ディレクトリを作成
$ ls -ld work                                            パーミッションを表示してみる
drwxr-xr-x   2 guest guest    4096 Jun 21 14:23 work     rwxr-xr-xになった
$ rmdir work                                             いったんディレクトリを削除
$ umask 027                                              umask値を027に変更
$ mkdir work                                             再度ディレクトリを作成
$ ls -ld work                                            パーミッションを表示してみる
drwxr-x---   2 guest guest    4096 Jun 21 14:23 work     今度はrwxr-x---になった
$ rmdir work                                             いったんディレクトリを削除
$ umask 077                                              umask値を077に変更
$ mkdir work                                             再度ディレクトリを作成
$ ls -ld work                                            パーミッションを表示してみる
drwx------   2 guest guest    4096 Jun 21 14:23 work     今度はrwx------になった
```

chmodコマンド風のumask値の指定

bashとFreeBSDのshでは、umask値として8進数の値を使用する代わりに、**図B**のようにchmodコマンド風のo-rxのような表記を用いることもできます。

Warning
Linux (bash) / FreeBSD (sh) / ✕ Solaris (sh)
この方法には制限があります。

図B chmod風にumask値を指定した実行例

```
$ umask                                                  現在のumask値を表示
0022                                                     umask値は022
$ umask o-rx                                             otherのrとxのパーミッションを落とす
$ umask                                                  再びumask値を表示
0027                                                     たしかに027に変わっている
$ umask g-rx                                             さらにgroupのrとxのパーミッションを落とす
$ umask                                                  再びumask値を表示
0077                                                     たしかに077に変わっている
```

umaskの-Sオプション

bashとFreeBSDのshでは、**図C**のように、umaskコマンドに-Sオプションを付け、umask値をchmodコマンド風に表示することもできます。

Warning
Linux (bash) / FreeBSD (sh) / ✕ Solaris (sh)
この方法には制限があります。

図C umask -Sの実行例

```
$ umask 027                                              umask値を027に設定
$ umask -S                                               umaskを-Sオプション付きで実行
u=rwx,g=rx,o=                                            umask値がchmodコマンド風に表示される
```

umaskの-pオプション

Warning
Linux (bash) ○ FreeBSD (sh) × Solaris (sh) ×
この方法には制限があります。

umask値は、umaskコマンドを引数なしで実行すれば表示されますが、bashでは、ここでumask -pとオプションを付けることもできます。この-pオプションを付けると図Dのように、実際にumask値を設定する際のコマンドラインを使った表示になります。

図D umask -pの実行例

```
$ umask 027                    umask値を027に設定
$ umask                        umask値を表示
0027                           umask値027が表示される
$ umask -p                     umask -pでumask値を表示
umask 0027                     コマンドライン形式でumask値が表示される
```

注意事項

サブシェル内でのumaskはサブシェル内でのみ有効

umaskコマンドが( )で囲まれたサブシェル内で実行されている場合、umask値はそのサブシェルのみに設定されます。シェル本体のumask値は変更されないので注意してください。

参照

サブシェル(p.72)

unset

Linux (bash) ○ FreeBSD (sh) ○ Solaris (sh) ○

シェル変数またはシェル関数を削除する

書式 unset [変数名 | 関数名 ...]

例 unset TEXT ……シェル変数TEXTを削除する

基本事項

unsetコマンドを実行すると、引数で指定されたシェル変数(変数名)またはシェル関数(関数名)が削除されます。シェル変数がexportされていた場合はその環境変数も削除されます。なお、readonlyが実行されているシェル変数／シェル関数は削除できません。

終了ステータス

unsetコマンドの終了ステータスは「0」です。ただし、変数名／関数名の指定が正しくないなど、unsetコマンド自体がエラーになった場合は終了ステータスは「0」以外になります。

解説

unsetコマンドは、すでに設定されているシェル変数／環境変数やシェル関数を取り消したい場合に使用します。シェル変数のexportを取り消すためにもunsetを使用しますが、readonlyについてはunsetでも取り消せません。

シェル変数／環境変数の削除

シェル変数に値を代入し、さらに環境変数にexportしたものをunsetしている様子を**図A**に示します。このように、unsetを実行するとシェル変数と環境変数が削除されます。

unsetコマンドの引数に複数のシェル変数またはシェル関数を指定して、これらを同時に削除することもできます。**図B**はシェル変数とシェル関数を同時に削除している例です。

図A シェル変数と環境変数の削除

```
$ DIR=/usr/local                シェル変数DIRに値を代入する
$ echo "$DIR"                   シェル変数DIRの値を表示してみる
/usr/local                      たしかに表示される
$ export DIR                    シェル変数DIRを環境変数にエクスポート
$ printenv DIR                  環境変数DIRの値を表示してみる
/usr/local                      たしかに表示される
$ uset DIR                      シェル変数／環境変数DIRをunsetする
$ echo "$DIR"                   シェル変数DIRの値を表示してみる
                                改行以外なにも表示されない
$ printenv DIR                  環境変数DIRの値も表示されない
```

図B 複数のシェル変数／シェル関数を同時に削除

```
$ TEXT=Hello                          シェル変数TEXTに値を代入
$ func() { echo World;}               echoを実行するシェル関数funcを定義
$ echo "$TEXT"                        シェル変数TEXTの値を表示
Hello                                 たしかに表示される
$ func                                シェル関数funcを実行
World                                 たしかに実行される
$ unset TEXT func                     TEXTとfuncを同時にunsetする
$ echo "$TEXT"                        シェル変数TEXTの値を表示してみる
                                      改行以外なにも表示されない
$ func                                シェル関数funcを実行してみる
bash: func: command not found         コマンドが見つからないというエラーになる
```

unsetの-vオプション／-fオプション

Warning

Linux (bash) FreeBSD (sh) Solaris

この方法には制限があります。

bashやFreeBSDのshには、unsetコマンドに-vや-fのオプションがあり、unsetの対象として、それぞれシェル変数またはシェル関数を指定できます。

bashでは、シェル変数と同名のシェル関数が同時に設定されている場合、unsetコマンドを-vや-fのオプションなしで実行すると、**シェル変数➡シェル関数**の順で先に見つかった一方のみがunsetされます。ここで、-vオプションでシェル変数を、-fオプションでシェル関数をunsetすることを明示的に指定することもできます。

FreeBSDのshでは、オプションなしのunsetはunset -vと同じであり、シェル変数のみがunsetされます。シェル関数をunsetするには、unset -fとする必要があります。

Solarisのshでは、シェル変数と同名のシェル関数を定義しようとすると、あとから定義したほうで上書きされ、同じ名前のシェル変数とシェル関数は同時には存在できません。したがってunsetコマンドを実行すると、その時点で定義されているシェル変数またはシェル関数がunsetされるだけであり、-vや-fのオプションは存在しません。

なお、そもそもシェル変数と同名のシェル関数を定義することは混乱の元となるため、避けたほうが賢明でしょう。

注意事項

位置パラメータ、特殊パラメータはunsetできない

unsetコマンドの引数に、位置パラメータ($1、$2など)や特殊パラメータ($@、$#など)を指定することはできません。

サブシェル内でのunsetはサブシェル内でのみ有効

unsetコマンドが( )で囲まれたサブシェル内で実行されている場合、シェル変数やシェル関数が削除されるのはそのサブシェルのみになります。シェル本体のほうは影響を受けないため注意してください。

参照

シェル変数の代入と参照(p.159) シェル関数(p.76) export(p.105)
環境変数の設定(p.179) readonly(p.115) サブシェル(p.72)

wait

Linux (bash) FreeBSD (sh) Solaris (sh)

バックグラウンドで起動したコマンドの終了を待つ

書式 **wait** [プロセスID]

例 `wait 4321` ……プロセスID=4321番のプロセスの終了を待つ

基本事項

waitコマンドは、現在のシェルからバックグラウンドで実行されたコマンドのうち、引数のプロセスIDで指定されたプロセスの終了を待ちます。引数のプロセスIDが省略された場合は、現在バックグラウンドで実行中のすべてのプロセスの終了を待ちます。

終了ステータス

引数で指定したプロセスの終了ステータスがwaitコマンドの終了ステータスになります。ただし、引数を指定しなかった場合は終了ステータスは「0」になります。

解説

シェルがコマンドを起動する際には、通常はリストの区切り文字や終端に&を付けないため、コマンドは**フォアグラウンド**で起動されます。シェルは、フォアグラウンドで起動されたコマンドの終了を待ち、1つのコマンドが終了してから次のコマンドの実行に移ります。一方、リストの区切り文字や終端に&が付けられた場合はコマンドは**バックグラウンド**で起動され、シェルはバックグラウンドで起動されたコマンドの終了を待たずに次のコマンドの実行に移ります。

ここで、いったんバックグラウンドで起動されたコマンドの終了を待つためにwaitコマンドを使用します。waitコマンドを実行すると、バックグラウンドで起動中のコマンドが終了するまでwaitコマンドは終了しません。waitコマンドにプロセスIDの引数を指定した場合は、そのバックグラウンドで起動中のコマンドの終了ステータスを取得できます。

なお、waitコマンドの引数には、プロセスIDのほか、%1、%2などの「ジョブ番号」を使ってもかまいません。

waitコマンドの使用例

waitコマンドの使用例を**図A**に示します。ここでは、sleepコマンドで10秒待ってから終了ステータス「25」を返すという動作を行うサブシェルを、バックグラウンドで起動しています。バックグラウンドのプロセスIDは、シェルによって「2687」と表示されています。特殊パラメータ$!には、直近にバックグラウンドで起動されたコマンドのプロセスID(この場合は「2687」)がセットされているので、これを引数に利用してwait $!としてサブシェルの終了を待ちます。

すると10秒後、シェルのメッセージとともにwaitコマンドが終了し、シェルのプロンプトに戻ります。ここで、waitコマンドの終了ステータスにはサブシェルの終了ステータスがセットされているはずなので、試しに特殊パラメータ$?の値を表示してみると、たしかに、はじめにセットした「25」になっていることがわかります。

図A waitコマンドの使用例

```
$ (sleep 10; exit 25) &          10秒間待って終了ステータス25を返す
                                 サブシェルをバックグラウンドで起動
[1] 2687                         バックグラウンドのプロセスIDが表示される
$ wait $!                        バックグラウンドのプロセスを指定してwaitする
[1]+  Exit 25   ( sleep 10; exit 25 )   10秒後、シェルのメッセージとともにwaitが終了する
$ echo $?                        試しにwaitコマンドの終了プロセスを表示
25                               たしかに、バックグラウンドのプロセスの
                                 終了ステータスになっている
```

注意事項

終了ステータスを取得したい場合は引数を指定

たとえバックグラウンドで実行しているコマンドが1つしかない場合でも、waitコマンドで終了ステータスを取得したい場合は、プロセスIDを明示的に指定する必要があります。これには、特殊パラメータ$!を利用するとよいでしょう。

```
wait $! ……終了ステータスを取得できる
wait ……終了ステータスは常に0になる
```

参照

リスト（p.35）　特殊パラメータ$!（p.175）

>第6章 組み込みコマンド2

6.1 概要 …… 136
6.2 組み込みコマンド（外部コマンド版もあり） …… 137
6.3 組み込みコマンド（拡張） …… 151
6.4 組み込みコマンド（その他） …… 156