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

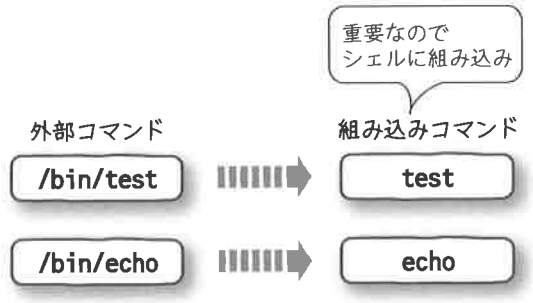
リスト（p.35）　特殊パラメータ$!（p.175）

>第6章 組み込みコマンド2

外部コマンド版も存在する組み込みコマンドと、拡張された組み込みコマンドについて

シェルスクリプトの組み込みコマンドには、testやechoコマンドのように、外部コマンド版も存在するものの、その使用頻度や重要度が高く、組み込みコマンドとして実装されているものがあります(**図A**)。本章では、testやechoのような外部コマンド版もあるコマンドに加えて、localやletなどの、あとから追加されたコマンドや、シェルスクリプト上ではなく、おもにコマンドラインで使用するコマンド(alias、history、jobsなど)についてもまとめて紹介します。

図A 外部コマンド版も存在する組み込みコマンド



echo

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

任意のメッセージを標準出力に出力する

書式 echo [-n] [-e] [文字列 ...]

Solarisのshには-nや-eのオプションはなく、常に-eオプション指定状態になる。
FreeBSDのshでは-nと-eを同時に指定できないほか、一部仕様が異なる

例
```
echo 'Hello World' ……画面にHello Worldと表示する
```

基本事項

echoコマンドは、引数で指定された文字列を標準出力に出力します。複数の引数が指定された場合は、各引数をスペースで区切って連結します。いずれも、最後に**改行コード**が付けられます。

-nオプションが指定された場合は、最後の改行が出力されません。

-eオプションが指定された場合、またはSolarisのshでは、文字列の中に**表A**に示す**エスケープシーケンス**が使用できます。

終了ステータス

終了ステータスは「0」になります。

表A echoコマンドの-eオプションで使えるエスケープシーケンス

表記	意味	16進数値
\a※1	ビープ音	0x07
\b	バックスペース	0x08
\c	改行の抑制	-
\e※1	エスケープコード	0x1b
\f	フォームフィード(FF)	0x0c
\n	改行(LF)	0x0a
\r	キャリッジリターン(CR)	0x0d
\t	タブ	0x09
\v	垂直タブ	0x0b
\\	バックスラッシュ	0x5c
\0nnn	3桁の8進数で文字を指定	-
\xnn※2	2桁の16進数で文字を指定	-

※1 Solarisのshでは\aと\eは使えない

※2 FreeBSDのshとSolarisのshでは、\xnn形式の16進数は使えない

解説

echoコマンドは、シェルスクリプト中で任意のメッセージを表示するために広く使用されます。使用頻度が高いため、echoコマンドは組み込みコマンドとして実装されています。

画面表示だけでなく、echoコマンドの標準出力をファイルにリダイレクトして、ファイルを作成することもできます[注1]。

なお、エラーなどでエラーメッセージを表示したい場合は、echoコマンドの出力を1>&2で標準エラー出力にリダイレクトするとよいでしょう[注2]。

引数は1つに

図A❶のように、echoで表示するメッセージが連続する2つ以上のスペースを含んでいる場合、これらはシェルによって単なる引数の区切りとして解釈され、echoコマンドには複数の引数として渡されます。その後、echoコマンドは複数の引数を単に1個のスペースで連結して表示するため、結果的に表示されるメッセージのスペースの数が変わってしまいます。

これを防ぐため、**図A❷**のように、メッセージは基本的にシングルクォート(' ')で囲み、echoコマンドには全体で1つの引数として渡るようにしたほうがよいでしょう。

改行の抑制

echoコマンドでは、メッセージの後に自動的に改行が行われます。この改行を行いたくない場合は、-nオプションを使います。**リストA**は、-nオプション付きのechoコマンドをwhile文のループで実行するもので、これを実行すると1秒に1個ずつ、*の記号が「****…」のように改行なしで1行で表示されます。

改行の抑制は-eオプションでもできます。-eオプションを使う場合は**リストB**のように、メッセージ(この場合は*)の後ろに\cというエスケープシーケンスを付けると、echoコマンドによって改行を行わないものと解釈されます。

図A 引数は1つに

```
$ echo Hello   World          ❶HelloとWorldの間にスペースが3つ空けてある
Hello World                   しかし、スペース1つで表示される
$ echo 'Hello   World'        ❷メッセージ全体をシングルクォートで囲むと
Hello   World                 そのままスペース3つで表示される
```

リストA 改行の抑制例1

```
while :  ……while文による無限ループ
do  ……ループの開始
  echo -n '*'  ……改行なしで*を表示（-nオプション利用）
  sleep 1  ……1秒間待つ
done  ……ループの終了
```

注1　標準出力のリダイレクトの項(p.243)も合わせて参照してください。

注2　ファイル記述子を使ったリダイレクトの項(p.248)も合わせて参照してください。

リストB 改行の抑制例2

```
while :  ……while文による無限ループ
do  ……ループの開始
  echo -e '*\c'  ……改行なしで*を表示(-eオプションと\cを利用)
  sleep 1  ……1秒間待つ
done  ……ループの終了
```

-nや-e自体の出力

echoコマンドで「-n」や「-e」という文字列自体を表示する必要がある場合、そのままではオプションと解釈されてしまい、うまく表示できません。かといって、「echo -e -n」としても、bashでは第2引数以降もオプションと解釈されるためうまくいきません。そこで**図B**のように、-を8進数で「\055」と記述して回避すれば、意図通りに「-n」や「-e」が表示できます。

なお、FreeBSDのshでは、echoの第2引数がオプションとは解釈されないため、「echo -e -n」や「echo -e -e」としてもかまいません。また、Solarisのshでは、元々echoのオプションがないため、そのまま「echo -n」や「echo -e」で出力できます。

図B -nや-e自体の出力

```
$ echo -e '\055'n             -の8進数表示の\055を使って-nを指定
-n                            意図通りに-nと表示される
$ echo -e '\055'e             -の8進数表示の\055を使って-eを指定
-e                            意図通りに-eと表示される
```

Memo

- FreeBSDの外部コマンドのechoや、SunOS 4.xのshのechoなど、-nオプションのみが使えて-eオプションが使えないechoも存在します。
- bashでは、-eオプションの効果を取り消す-Eオプションも使えます。
- dashの場合は、Solarisのshのechoコマンドと同様に-eオプションがなく、常に-eオプション指定状態になる一方、-nオプションだけは使えます。
- -nオプションはBSD系UNIX由来、-eオプションで使えるエスケープシーケンスはSystemV系UNIX由来です。

参照

標準出力のリダイレクト(p.243)　ファイル記述子を使ったリダイレクト(p.248)
シングルクォート' '(p.207)

false

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

単に「1」の終了ステータスを返す

Solarisではfalseは外部コマンドとして実装されている

書式 false [引数 ...]

例

```
until false ..... falseコマンドにより終了ステータス1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

falseコマンドは、終了ステータス「1」を返すだけのコマンドです。引数を付けてもすべて無視されます。ただし、falseコマンドに対するリダイレクトや、引数中のパラメータ展開、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

falseコマンドの終了ステータスは「1」になります。

解説

falseコマンドは、trueコマンドとは逆に、常に終了ステータス「1」(偽)を返すコマンドです。冒頭の例のように、while文とは条件判断が逆のuntil文の条件判断に使用すれば無限ループになります。

Memo

- falseコマンドとは逆に、常に終了ステータス「0」を返すにはtrueコマンドを使います。
- サブシェルを使って(exit 1)と記述すると、falseコマンドと同じことになります。

参照

true(p.150)　while文(p.63)　サブシェル(p.72)　exit(p.103)

kill

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

プロセスにシグナルを送る

FreeBSDのkillは外部コマンドのみ

書式 kill [-シグナル名 | -シグナル番号] プロセスID [プロセスID ...]
kill -l

例

```
kill -HUP 4321 ..... プロセスID=4321のプロセスにHUP(ハングアップシグナル)を送る
```

基本事項

killコマンドは、引数のプロセスIDで指定されたプロセスに対して、**シグナル名**または**シグナル番号**で指定されたシグナルを送ります。シグナル指定が省略された場合は**SIGTERM**(15番)が送られます。シグナル名の指定では頭の「SIG」は省略します。bashなどではSIGを付けた名前や、小文字の名前で指定してもかまいません(**図A**のシグナルの一覧表示を参照)。

kill -lを実行した場合は、指定可能なシグナルの一覧が表示されます。

終了ステータス

シグナルが正常に送信できた場合、またはkill -lを実行した場合は、終了ステータスは「0」になります。

解説

killコマンドは指定のプロセスにシグナルを送ります。シグナルを受け取ったプロセスの動作は、デフォルトではプロセスの終了となるため、killコマンドは、おもに現在起動中のコマンドを終了させる目的で使用されます。なお、ジョブコントロールが有効なシェル上で**組み込みコマンド版**のkillを使う場合は、プロセスIDの代わりに、%1、%2などの「ジョブ番号」でも指定できます。

シグナル名とシグナル番号の一覧表示

図Aのように、kill -lとオプションを付けて実行すると、使用可能なシグナルの一覧が表示されます。具体的な表示内容はOSによって異なります。おもに使用されるのは、**SIGHUP**、**SIGINT**、**SIGQUIT**、**SIGKILL**、**SIGTERM**などです。

図A シグナルの一覧表示の例

```
$ kill -l                                   シグナルの一覧を表示させる-lオプションを付けて実行
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

シグナルの送信例

シグナルを実際に送信している例を**図B**に示します。このように、sleepなどの適当なコマンドをバックグラウンドで起動し、そのプロセスIDに対してkillを実行すると、シグナルが送信され、プロセスが終了します。なお、ここでのプロセスIDの指定の際に、最後に実行したバックグラウンドプロセスのIDを保持している特殊パラメータの$!を使っても構いません。

図B シグナルの送信例

```
$ sleep 100 &                試しにsleepコマンドをバックグラウンドで起動する
[1] 2614                     sleepコマンドのプロセスIDは2614番と表示される
$ kill -INT 2614             プロセスID=2614に対してINTシグナルを送る
$                            いったんシェルのプロンプトに戻るが、ここで再度Enterを押す
[1]+  Interrupt                sleep 100     INTシグナルによってsleepが終了した旨が表示される
```

Memo

- bash組み込みのkillコマンドでは、さらにいくつかの別のオプションも用意されています。

printf

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

メッセージを一定の書式に整形して標準出力に出力する

FreeBSDのprintfは外部コマンド版のみ

Solarisのprintfは外部コマンド版のみ

書式 printf フォーマット文字列 [引数 ...]

例

```
i=123 ……シェル変数iに値を代入
printf '合計%d個です\n' "$i" ……iの値を10進数として、メッセージに含めて表示
```

基本事項

printfコマンドは、C言語の標準ライブラリ関数のprintf()関数と同様に、フォーマット文字列中の%で始まる**書式指定文字列**と、それに対応する引数と、\による**エスケープシーケンス**を解釈して文字列を整形し、結果を標準出力に出力します。

C言語のprintf()にはない、%bと%qの書式指定も使えます。

%bでは、対応する引数の文字列中の\によるエスケープシーケンスを解釈します。

%qでは、対応する引数の文字列に、シェルへの入力時の解釈を避けるために必要なクォートを付加して出力します[注3]。

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

printfコマンドを使えば、数値や文字列を含むメッセージを一定の書式に整形して表示することができます。printfコマンドで使える書式指定文字列の例を**表A**に示します。このほか、詳しくはprintfのオンラインマニュアルを参照してください。

また、\によるエスケープシーケンスは、echoコマンドの-eオプションで使用できるものと基本的には同じです(p.144の表A参照)。ただし、3桁の8進数については頭に0は付けません。たとえば、「echo -e '\0123'」に相当するのは「printf '\123\n'」となります。また、FreeBSDのshとSolarisのshでは、「\xnn」形式の16進数と「\e」は使えません。

フォーマット文字列には、通常\nなどのバックスラッシュを含む文字があるため、シングルクォート(' ')で囲んで、シェルの解釈を避ける必要があります。

printfコマンドは、シェル環境によっては外部コマンド版しかない場合があります。したがって、とくに書式の整形が必要なく、単にメッセージを表示したいだけの場合は、echoコマンドを使ったほうが効率がよいでしょう。たとえば、冒頭の例についても、echoコマンドを使って「echo '合計 '"$i"' 個です '」と記述できます。

注3 FreeBSDのshとSolarisのshでは、%qは使えません。

表A 書式指定文字列の例

書式指定文字列	動作
%d	10進数値を表示
%x	16進数値を表示
%f	123.456789のような普通の表現で実数を表示
%e	1.234567e+02のような指数表現で実数を表示
%s	文字列を表示
%c	1文字を表示
%8d	8文字分のスペースを確保し、10進数値を右詰めで表示
%-8d	8文字分のスペースを確保し、10進数値を左詰めで表示
%+d	正の数の場合、+の符号を付けて10進数値を表示
%04x	4桁に満たない場合は頭に0を付けて、4桁の16進数値を表示
\0nnn	3桁の8進数で文字を指定
\xnn	2桁の16進数で文字を指定

IPアドレスの10進／16進変換

printfコマンドを利用して、10進／16進の変換を行えます。**図A❶❷**は、IPアドレスの4バイトの数値を、10進表記から16進表記へ、およびその逆の変換を行っている例です。**図A❷**のように、16進数値をprintfコマンドの引数に付ける場合は、頭に「0x」を付けて16進数であることを明示する必要があります。

図A IPアドレスの10進／16進変換

```
$ printf '%02x %02x %02x %02x\n' 192 168 10 21 ……❶10進表記の引数を16進に変換
c0 a8 0a 15 ……16進表記に変換されて表示される
$ printf '%d.%d.%d.%d\n' 0xc0 0xa8 0x0a 0x15 ……❷16進表記の引数を10進に変換
192.168.10.21 ……10進表記に変換されて表示される
```

参照

echo(p.137)　　シングルクォート(p.207)

pwd

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

カレントディレクトリの絶対パスを表示する

書式 pwd

例 pwd ……カレントディレクトリを表示

基本事項

pwdコマンドを実行すると、カレントディレクトリの絶対パスを表示します。

終了ステータス

コマンドの実行がエラーにならないかぎり、終了ステータスは「0」になります。

解説

シェルのコマンドライン上での作業中、カレントディレクトリ名を知りたい場合にpwdコマンドを使いますが、pwdコマンドはもちろんシェルスクリプト上でも使用できます。なお、cdコマンドでの移動先ディレクトリがシンボリックリンクを含んでいる場合、その後のpwdコマンドの表示もシンボリックリンクを含む場合があります。

pwdコマンドの使用例

pwdコマンドの使用例を**図A**に示します。このように、カレントディレクトリの絶対パスをシェル変数に代入したい場合、バッククォート(` `)などによるコマンド置換を用います[注4]。

図A pwdコマンドの使用例

```
$ pwd ……カレントディレクトリの絶対パスを表示
/usr/local/bin ……/usr/local/binにいることがわかる
$ dir=`pwd` ……カレントディレクトリの絶対パスをシェル変数に代入
$ echo "$dir" ……そのシェル変数の値を表示
/usr/local/bin ……正しく/usr/local/binと表示される
```

注4 コマンド置換(` `)の項(p.213)を参照してください。

pwd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の組み込みコマンドのpwdには-Pというオプションがあり、カレントディレクトリが、元々シンボリックリンクをたどってcdコマンドで移動してきたディレクトリであっても、強制的に本来の物理的なディレクトリの絶対パスを表示させることができます。

図B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実行例を示します。

なお、外部コマンドのpwdと、Solarisのshの組み込みpwdコマンドには-Pオプションはありません。

図B pwd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
$ pwd -P                       -Pオプション付きで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
```

Memo

- bashまたはFreeBSDのshでpwd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pwdコマンドを実行するには、pwd -Lとします。set -Pコマンドはcdコマンドにも影響します。

参照

cd(p.95)　　set(p.120)

test

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

シェルスクリプトにおいて各種条件判断を行う

FreeBSDやSolarisのshのtestでは一部使用できないオプションがある

書式 test [条件式]
[[条件式]]

例

```
if [ "$i" -le 3 ] ……testコマンドで、iの値が3以下かどうかを判断
then ……thenのリストの開始
  echo 'iの値は3以下です' ……メッセージを表示
fi ……if文の終了
```

基本事項

testコマンドは、**表A**の演算子を解釈して条件式を評価し、その結果を終了ステータスとして返します。testコマンドが、コマンド名[で呼び出された場合は、コマンドの最後の引数が]である必要があります。

終了ステータス

条件式の評価結果が真ならば終了ステータスは「0」に、偽ならば終了ステータスは「1」になります。

解説

testコマンドは、if文やwhile文での条件判断に使用される重要なコマンドです。シェルスクリプトでは、**文字列の比較、数値の比較、ファイルの存在**や**属性のチェック**といった条件判断を行うには、基本的にtestコマンドを使用します。このためtestコマンドは使用頻度が高く、外部コマンドだけでなく、シェルの組み込みコマンドとしても実装されています。

testコマンドは、「test」という名前でも「[」という名前でも起動でき、[で起動した場合は最後の引数を]にするため、そのコマンドラインは[ ]という角カッコで囲んだ状態になります。testコマンドは、echoコマンドなどと同じく、あくまで単純コマンドの1つですが、コマンド名[のほうを使って、if [条件式]やwhile [条件式]の形で記述すれば、testコマンドがあたかもif文やwhile文の構文の一種であるかのように見えるでしょう。

なお、testコマンドを&&リストや||リストに使ったり、testコマンドの直後に終了ステータスを特殊パラメータ$?で参照する使い方もできます。

表A testコマンドの条件式で使用できる演算子

条件式	内容	Linux (bash)	FreeBSD (sh)	Solairs (sh)
-a [ファイル]	(-eと同じ)	○	×	×
-b [ファイル]	ファイルがブロック特殊ファイルならば真	○	○	○
-c [ファイル]	ファイルがキャラクタ特殊ファイルならば真	○	○	○
-d [ファイル]	ファイルがディレクトリならば真	○	○	○
-e [ファイル]	ファイルが存在すれば真	○	○	×
-f [ファイル]	ファイルが通常ファイルならば真	○	○	○
-g [ファイル]	ファイルがセットグループIDされていれば真	○	○	○
-h [ファイル]	ファイルがシンボリックリンクならば真	○	○	○
-k [ファイル]	ファイルのstickyビットが立っていれば真	○	○	○
-p [ファイル]	ファイルが名前付きパイプ(FIFO)ならば真	○	○	○
-r [ファイル]	ファイルが読み込み可能ならば真	○	○	○
-s [ファイル]	ファイルサイズが0よりも大きければ真	○	○	○
-t [番号]	ファイル記述子番号が端末ならば真	○	○	○
-u [ファイル]	ファイルがセットユーザIDされていれば真	○	○	○
-w [ファイル]	ファイルが書き込み可能ならば真	○	○	○
-x [ファイル]	ファイルが実行可能ならば真	○	○	○
-O [ファイル]	ファイルの所有者が実効ユーザと同じなら真	○	○	×
-G [ファイル]	ファイルのグループが実効グループと同じなら真	○	○	×
-L [ファイル]	(-hと同じ)	○	○	○
-S [ファイル]	ファイルがソケットならば真	○	○	×
-N [ファイル]	ファイルの更新時刻がアクセス時刻以降ならば真	○	×	×
[ファイル1] -nt [ファイル2]	ファイル1の更新時刻がファイル2より新しければ真	○	○	×
[ファイル1] -ot [ファイル2]	ファイル1の更新時刻がファイル2より古ければ真	○	○	×
[ファイル1] -ef [ファイル2]	ファイル1とファイル2のiノードが同じならば真	○	○	×
-o [オプション名]	指定のシェルオプションが有効になっていれば真	○	×	×
-z [文字列]	文字列が空文字列ならば真	○	○	○
-n [文字列]	文字列が空文字列でなければ真	○	○	○
[文字列]	(「-n 文字列」と同じ)	○	○	○
[文字列1] = [文字列2]	文字列1と文字列2が同じならば真	○	○	○
[文字列1] == [文字列2]	(「文字列1 = 文字列2」と同じ)	○	×	×
[文字列1] != [文字列2]	文字列1と文字列2が異なれば真	○	○	○
[文字列1] < [文字列2]	文字列1が辞書順で文字列2よりも前にあれば真	○	○	×
[文字列1] > [文字列2]	文字列1が辞書順で文字列2よりも後にあれば真	○	○	×
[数値1] -eq [数値2]	数値1と数値2が等しければ真	○	○	○
[数値1] -ne [数値2]	数値1と数値2が等しくなければれば真	○	○	○
[数値1] -lt [数値2]	数値1が数値2より小さければ真	○	○	○
[数値1] -le [数値2]	数値1が数値2より小さいか等しければ真	○	○	○
[数値1] -gt [数値2]	数値1が数値2より大きければ真	○	○	○
[数値1] -ge [数値2]	数値1が数値2より大きいか等しければ真	○	○	○
! [条件式]	条件式が偽ならば真	○	○	○
[条件式1] -a [条件式2]	条件式1と条件式2の両方が真ならば真	○	○	○
[条件式1] -o [条件式2]	条件式1と条件式2のどちらか真ならば真	○	○	○
([条件式])	条件式が真ならば真(演算優先順位のためのカッコ)	○	○	○

注意事項

各引数はスペースで区切る

testコマンドでは、引数1個につき1個の文字列、数値、演算子を与える必要があります。したがって、各引数はすべてスペースなどで区切らなければなりません。同様に[の直後や、]の直前にもスペースが必要です。

○正しい例

```
[ "$var1" = "$var2" ] ……シェル変数var1とvar2の内容が等しければ真
```

×誤った例

```
[ "$var1"="$var2" ] ……"$var1"="$var2"という文字列全体が空文字列か
                       どうかという判断が行われてしまう
```

文字列の比較は==ではなく=で

本来testコマンドでは、2つの文字列が同じならば真になる演算子は=です。bashでは、=の代わりにC言語風に==も使えるようになっていますが、移植性を考えて=で記述したほうがよいでしょう。

なお、=はあくまで文字列の比較であり、数値を比較したい場合は-eqを使います。

一般的な記述例

```
[ "$str" = hello ] ……シェル変数strの内容がhelloならば真
```

bashでの記述例

```
[ "$str" == hello ] ……シェル変数strの内容がhelloならば真(bash以外ではエラー)
```

演算子の< >や( )はクォートが必要

testコマンドの演算子にある「< >」や「( )」を使用する際には、これらの記号がシェルに解釈されないように、頭に\を付けて、それぞれ「\< \>」「\(\)」とする必要があります。とくに、>のクォートを忘れるとファイルへのリダイレクトとみなされ、意図せずにファイルが作成されてしまうため、注意してください。

なお、\<や\>はあくまで文字列の比較であり、数値を比較したい場合はそれぞれ-ltや-gtを使います。

○正しい例

```
[ "$str" \> paaa ] ……シェル変数strの内容が辞書順でpaaaより後ならば真
```

×誤った例

```
[ "$str" > paaa ] ……paaaというファイルが作成され、"$str"のみがtestコマンドの引数になる
```

Memo

- bashでは、[ ]の代わりに複合コマンドの[[ ]]を使うこともできます。[ ]と[[ ]]とでは一部文法が異なります。

参照

if文(p.43) while文(p.63) &&リスト(p.37) ||リスト(p.39)
特殊パラメータ$?(p.173) 条件式の評価[[ ]](p.83)

true

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

単に「0」の終了ステータスを返す

Solarisではtrueは外部コマンドとして実装されている

書式 **true** [引数 ...]

例
```
while true ……true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

trueコマンドは、終了ステータス「0」を返すだけのコマンドです。引数を付けてもすべて無視されますが、trueコマンドに対するリダイレクトや、引数中のパラメータ展開、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

true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

trueコマンドは、冒頭の例のように、while文の条件判断などで固定的に真の値を得たい場合に使用されます。trueコマンドの動作は:コマンドと同じです。Solarisのshなどのように、trueが組み込みコマンドではなく、外部コマンドのみで実装されている環境も存在するため、trueコマンドの代わりに組み込みコマンドの:コマンドを使ったほうが効率がよいでしょう。

Memo

- trueコマンドとは逆に、常に終了ステータス「1」を返すにはfalseコマンドを使います。
- サブシェルを使って(exit 0)と記述すると、trueコマンドと同じことになります。

参照

while文(p.63)　:コマンド(p.87)　false(p.140)
サブシェル(p.72)　exit(p.103)

builtin

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

シェル関数と同名の組み込みコマンドを優先的に実行する

書式 **builtin** コマンド名 [引数 ...]

例
```
echo() ……echoという名前のシェル関数の定義の開始
{
  builtin echo 'echo output: ' "$@" ……引数を追加し、組み込みコマンドのechoを呼び出す
} ……シェル関数の定義の終了
```

基本事項

builtinコマンドはコマンド名で指定されたコマンドがたとえシェル関数として定義されていても、シェル関数ではなく、**組み込みコマンドを実行**します。builtinコマンドに付けられた引数は、そのまま組み込みコマンドに渡されます。

終了ステータス

実行された組み込みコマンドの終了ステータスが、builtinコマンドの終了ステータスになります。

解説

シェルスクリプトでシェル関数を定義する際に、echo、cdなどの組み込みコマンドと同名のシェル関数を定義することができます。シェル上でのコマンド実行時には、組み込みコマンドよりもシェル関数が優先されて実行されるため、実質的に組み込みコマンドをシェル関数で再定義するようなことが可能です。

ここで、シェル関数内から元の組み込みコマンドを呼び出す際に、そのままコマンド名を記述すると、それは自分自身のシェル関数の呼び出しになり、**無限に再帰呼び出し**が発生し、シェルスクリプトが正常に動作しません。そこで、builtinコマンドを使い、組み込みコマンドを優先的に指定して実行するようにするのです。

冒頭の例では、組み込みコマンドのechoをシェル関数で再定義し、echoが実行された時にメッセージの先頭に「echo output: 」という文字列が付くようにしています。なお、この例ではecho -eやecho -nのオプションは考慮していません。

参照

シェル関数(p.76)

command

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

シェル関数と同名の組み込みコマンドまたは外部コマンドを優先的に実行する

FreeBSDのshでは、組み込みコマンドではなく外部コマンドのみが実行される

書式 **command** コマンド名 [引数 ...]

例
```
ls() ..... lsという名前のシェル関数の定義の開始
{
  command ls -F "$@" ..... -Fオプションを追加し、外部コマンドのlsを呼び出す
} ..... シェル関数の定義の終了
```

基本事項

commandコマンドは、コマンド名で指定されたコマンドがたとえシェル関数として定義されていても、シェル関数ではなく、**組み込みコマンドまたは外部コマンドを実行**します。commandコマンドに付けられた引数は、そのまま実行されるコマンドに渡されます。

終了ステータス

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

解説

シェルスクリプトでシェル関数を定義する際に、ls、cpなどの外部コマンドと同名のシェル関数を定義することができます。シェル上でのコマンド実行時には、外部コマンドよりもシェル関数が優先されて実行されるため、実質的に外部コマンドをシェル関数で再定義するようなことが可能です。

ここで、シェル関数内から元の外部コマンドを呼び出す際に、そのままコマンド名を記述すると、それは自分自身のシェル関数の呼び出しになり、**無限に再帰呼び出し**が発生し、シェルスクリプトが正常に動作しません。そこで、commandコマンドを使い、外部コマンドを優先的に指定して実行するようにするのです。冒頭の例では、外部コマンドのlsをシェル関数で再定義し、lsが実行された時に常に-Fオプションが付くようにしています。

なお、bashのcommandコマンドでは、同名の組み込みコマンドがある場合、外部コマンドよりも組み込みコマンドが優先されて実行されますが、FreeBSDのshでは、外部コマンドのみが実行されます。

Memo

bashのcommandコマンドには、-pオプションほかいくつかのオプションも存在します。

参照

シェル関数(p.76)

let

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

算術式を評価するのにletコマンドを使う方法もある

FreeBSDのshにもletコマンドはあるが、演算結果の数値が標準出力に出力されたり、算術式の引数の扱い方が違うなど仕様が異なるため、FreeBSDでは非対応であるとみなす

書式 **let** 算術式 [算術式 ...]

例
```
let 'i >= 3' ..... シェル変数iの値が3以上ならば真(終了ステータス0)になる
```

基本事項

letコマンドは、引数で指定された算術式を評価し、その結果を終了ステータスとして返します。算術式の評価方法は、(( ))を使った算術式の評価と同じです。

引数に算術式を複数指定した場合は、各算術式が評価され、最後の算術式の評価結果が終了ステータスになります。

終了ステータス

最後の算術式の評価結果が真(「0」以外)なら、算術式の評価の終了ステータスは真(0)に、最後の算術式の評価結果が偽(0)なら、算術式の評価の終了ステータスは偽(1)になります。

解説

letコマンドは、複数の算術式を引数にすることもできる点を除いて(( ))を使った算術式の評価と同じです。letコマンドの終了ステータスをif文やwhile文の条件判断に用いたり、あるいは代入などのシェル変数の変化をともなう演算を行うために使用することができます。

注意事項

算術式はシングルクォートで囲む

letコマンドでは、1つの算術式を1つの引数として与える必要があります。したがって、算術式は通常、シングルクォートで囲むことになります。シングルクォートで囲まないと、算術式中のスペースで分割され、別の引数とみなされてしまうため、正しく動作しません。

○正しい例
```
let 'b = a + 3' ..... 算術式全体をシングルクォートで囲む
```
×誤った例
```
let b = a + 3 ..... シングルクォートで囲まないと別引数とみなされ、エラーになる
```

参照

算術式の評価(( ))(p.80)　　シングルクォート' '(p.207)

local

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

シェル関数内でローカル変数を使う

書式 **local** 変数名 [=値] ...

例

```
func() ……シェル関数funcの宣言の開始
{
  local i ……シェル変数iをローカル変数として宣言
  i=3 ……ローカル変数になったシェル変数iに値を代入
  echo "$i" ……試しにシェル変数iの値を表示
  :
} ……シェル関数の宣言の終了
```

基本事項

シェル関数内で**localコマンド**を使用すると、以降、引数の変数名で指定したシェル変数が**ローカル変数**として扱われ、値を変更してもシェル関数の呼び出し元のシェル変数には影響を与えないようになります。localコマンドでシェル変数に同時に値を代入したり、複数のシェル変数を同時に指定することもできます。

シェル関数内以外でlocalコマンドを実行するとエラーになります[注5]。

終了ステータス

コマンド文法にエラーがないかぎり、終了ステータスは「0」になります。

解説

シェル関数では、位置パラメータを除き、シェル変数は**基本的にはグローバル変数**として扱われます。シェル関数内でシェル変数に値を代入すると、シェル関数からリターンしたあとも、そのシェル変数には値が代入されたままになります。

そこで、シェル関数内だけで有効な**ローカル変数**がほしい場合、localコマンドを使用してシェル変数をローカル変数として宣言します。localを使えば、シェル変数内でのローカル変数への代入は、シェル関数の呼び出し元の同名のシェル変数に影響しません。

ただし、Solarisのshなど、localコマンドが使えないシェルも存在するため、移植性のためには、シェル関数内で**サブシェル**を使う方法でローカル変数を実現したほうがよいでしょう。

注5　シェル関数の項(p.76)も合わせて参照してください。

注意事項

local宣言は先に

シェル関数内でlocalコマンドを実行する前にシェル変数に値を代入すると、その代入については呼び出し元に影響を与えてしまいます。通常、localコマンドはシェル関数の宣言直後に、少なくともシェル変数への代入よりも前に行う必要があります。

○正しい例

```
func() ……シェル関数funcの宣言の開始
{
  local i ……シェル変数iをローカル変数として宣言
  i=3 ……ローカル変数iに3を代入
  i=4 ……ローカル変数iに4を代入
} ……シェル関数の宣言の終了
```

×誤った例

```
func() ……シェル関数funcの宣言の開始
{
  i=3 ……ローカル変数になっていないので、
        呼び出し元のシェル変数iも3に書き換わる
  local i ……後でローカル宣言
  i=4 ……これはローカル変数への代入として扱われる
} ……シェル関数の宣言の終了
```

参照

サブシェル(p.72)

その他の組み込みコマンド

解説

シェルには、前述のコマンド以外にも、**表A**のようにたくさんの組み込みコマンドが存在します。しかし、これらは、エイリアス関連、ディレクトリスタック関連、ヒストリ／行編集／補完関連、ジョブコントロール関連のコマンドや、そのほかの情報などの表示設定コマンドなどであり、おもに**コマンドライン上**や、**bash限定環境**で使用されるコマンドです。これらのコマンドは通常はシェルスクリプト中では使用しないため、ここではコマンドの紹介にとどめます。詳しくは、各シェルのオンラインマニュアルを参照してください。

表A　その他の組み込みコマンド※

コマンド	分類	概説	Linux (bash)	FreeBSD (sh)	Solairs (sh)
alias	A	エイリアスの設定と参照	○	○	×
bg	J	ジョブをバックグラウンドで実行	○	○	○
bind	H	行編集キーの割り当て	○	○	×
compgen	H	補完リストの生成	○	×	×
complete	H	補完機能の設定	○	×	×
declare		シェル変数の宣言と属性の設定	○	×	×
typeset		シェル変数の宣言と属性の設定	○	×	×
dirs	D	ディレクトリスタックを表示	○	×	×
disown	J	ジョブテーブルからの削除	○	×	×
enable		組み込みコマンドの有効無効設定	○	×	×
fc	H	ヒストリの編集	○	○	×
fg	J	ジョブをフォアグラウンドで実行	○	○	○
hash		ハッシュテーブルの表示とクリア	○	○	○
help		組み込みコマンドのhelpの表示	○	×	×
history	H	ヒストリの表示	○	×	×
jobs	J	ジョブの表示	○	○	○
logout		ログインシェルをexitする	○	×	×
popd	D	ディレクトリスタックからPOPする	○	×	×
pushd	D	ディレクトリスタックにPUSHする	○	×	×
shopt		シェルのオプションの表示と設定	○	×	×
suspend	J	シェルをサスペンドする	○	×	○
times		合計プロセス時間の表示	○	○	○
ulimit		リソース制限の表示と設定	○	○	○
unalias	A	エイリアスの解除	○	○	×

※　分類欄の記号について：
A：エイリアス　　H：ヒストリ／行編集／補完
D：ディレクトリスタック　　J：ジョブコントロール

>第7章 パラメータ