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

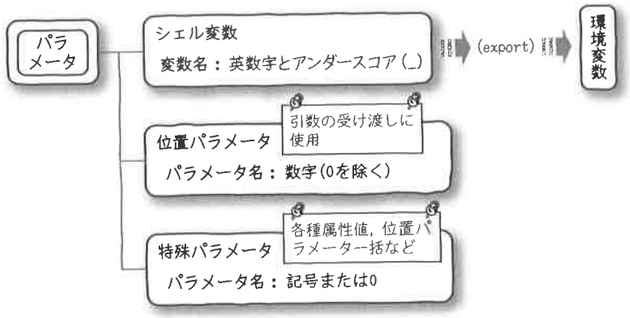
>第7章 パラメータ

シェルにおけるパラメータ

シェル上では**シェル変数**によって変数を扱えます。さらにシェル変数以外に、**位置パラメータ**や**特殊パラメータ**と呼ばれる、一種の変数が存在し、シェル変数、位置パラメータ、特殊パラメータをまとめて**パラメータ**といいます(**図A**)。

パラメータのうち、シェル変数は環境変数としてエクスポートすることが可能です。シェル変数の中にはPATHやHOMEなどのように特別な意味を持つシェル変数もあります。位置パラメータは$1、$2……などの数字のパラメータで、シェルスクリプトやシェル関数の引数の受け渡しに使用します。特殊パラメータは$?や$$などのパラメータで、シェルの属性値などが参照できるほか、位置パラメータすべてをまとめて参照できる特殊パラメータ"$@"もあります。本章では、これらのパラメータについて解説します。

図A パラメータ



シェル変数の代入と参照

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

変数や定数を使うにはシェル変数を使う

書式
- 代入：変数名=[値]
- 参照：$変数名 | ${変数名}

例

```
message='Hello World' ……シェル変数messageに値を代入
echo "$message" ……シェル変数messageの値を表示
```

基本事項

シェルスクリプト上で、変数や定数を使うにはシェル変数を使います。冒頭の代入の書式により、任意の**シェル変数**([変数名])に[値]を**代入**することができます。[値]を省略すると値が空文字列になりますが、シェル変数自体は定義されます。定義されたシェル変数の値は、冒頭の参照の書式によって**参照**できます。

[変数名]には、英数字とアンダースコア(_)が使えます。[変数名]の1文字目は数字以外である必要があります。

解説

シェルスクリプトでは、任意の変数名のシェル変数を用いて値を代入したり、参照したりして使用することができます。C言語などとは違って、使用する変数をあらかじめ宣言する必要はありません。変数の値は基本的に文字列として扱われます。

シェル変数は、exportコマンドでエクスポートすると**環境変数**になります。環境変数の代入／参照方法もシェル変数と同じです。

また、シェルの起動時にすでに設定されている環境変数については、同名のシェル変数にその値が代入されます。

シェルスクリプト中で使用する定数をシェル変数に代入して、シェル変数を定数ラベルのようにして使用することもできます。この場合、readonlyコマンドを実行してシェル変数を読み出し専用にしておくと確実です。

readonlyが設定されていないシェル変数は、unsetコマンドによって削除することができます。

シェル変数の代入と参照の例

シェル変数に代入し、その値を参照している実例を**図A❶～❺**に示します。代入時には=の前後にスペースを入れてはいけません。代入する値は、スペースや特殊記号などがシェルによって解釈されるのを防ぐため、基本的にシングルクォート(' ')で囲むようにします。

シェル変数の参照は、変数名の頭に$を付けることによって行います。ただし、この際に値に含まれるスペースや*などのパス名展開の文字が解釈されてしまうのを防ぐため、基本的に全体をダブルクォート(" ")で囲みます[注1]。

図A❸のように、シェル変数の参照時に、変数名を{ }で囲んでもかまいません。これは${パラメータ:-値}などの形式のパラメータ展開を行う場合や、変数名の直後に別の文字列が続く場合の区切りとして必要ですが、通常は省略してかまいません。通常は全体をダブルクォートで囲むため、{ }がなくても変数名の区切りは明確です。

なお、シェル変数はコマンドの引数のほか、**コマンド名自体**としても使用できます。図A❹❺では、シェル変数cmdに「echo」という値を代入して、シェル変数経由でechoコマンドを実行しています。

シェル変数からシェル変数への代入

シェル変数の参照時には"$変数名"のようにダブルクォートで囲むのが基本ですが、シェル変数の値を別のシェル変数に直接代入する場合には、ダブルクォートがなくてもこれ以上解釈は行われません。

図Bのように、連続するスペースや特殊記号を含んだ文字列をいったんシェル変数aに代入し、これをそのままシェル変数bに代入してもちゃんと値が保存されていることがわかります。ただし、最後にシェル変数の値を参照する際にダブルクォートを忘れると、文字列が解釈されてしまい、値通りの表示ではなくなってしまいます。

このほか、a=`cmd`のようにコマンド置換の文字列を直接シェル変数に代入する場合も、ダブルクォートは必要ありません[注2]。

図A シェル変数の代入と参照の例

```
$ message='Hello World'          ❶シェル変数messageに値を代入
$ echo "$message"                ❷シェル変数messageの値を表示
Hello World                      たしかにHello Worldと表示される
$ echo "${message}"              ❸変数名を{ }で囲んでもよい
Hello World                      同じくHello Worldと表示される
$ cmd=echo                       ❹シェル変数cmdにechoというコマンド名を代入
$ "$cmd" "$message"              ❺シェル変数cmdとmessageを使ってコマンドを実行
Hello World                      これでもHello Worldと表示される
```

図B シェル変数からシェル変数への代入

```
$ a='***  Hello  World !! ***'   スペースや特殊記号を含んだ文字列を代入
$ b=$a                           ダブルクォートなしで別のシェル変数に代入
$ echo "$b"                      その値を表示すると
***  Hello  World !! ***         正しく値が代入されていることがわかる
$ echo $b                        参照時のダブルクォートを省略すると
bin doc memo.txt src Hello World !! bin doc memo.txt src
                                 カレントディレクトリのファイル名に展開されたり、
                                 連続するスペースが1個だけになったりしてしまう
```

注1　パス名展開については10.2節を参照してください。
注2　コマンド置換については9.3節を参照してください。

注意事項

=の前後にスペースを入れてはいけない

C言語その他の言語とは異なり、シェル変数の代入の場合は=の前後にスペースを入れてはいけません。次の例のようにスペースを入れてしまうと、変数名がコマンド、=と値がそのコマンドの引数とみなされてしまい、エラーになります。

×誤った例

```
$ var = 3 ……= の前後にスペースを入れる
bash: var: command not found ……varというコマンドを実行するものと
                                 見なされてエラーになる
```

サブシェル内でのシェル変数への代入はサブシェル内でのみ有効

シェル変数への代入が( )で囲まれたサブシェル内で実行されている場合、そのシェル変数はそのサブシェル内のみで有効です。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

- シェル変数の中にはPATH、HOMEなど、シェル上で特別な意味を持つシェル変数があります。
- 未定義のシェル変数を参照してもエラーにはならず、空文字列として展開されます。ただし、あらかじめset -uというコマンドを実行しておくと、未定義のシェル変数の参照はエラーになります。
- bashの場合は、declareまたはtypesetという組み込みコマンドを使って明示的にシェル変数の型を宣言することも可能です。
- bashの場合は算術式の評価を使って((var = 3))のように記述することにより、=の前後にスペースを入れることも可能です。ただし代入する値は数値に限ります。

参照

export(p.105)　readonly(p.115)　unset(p.131)　シングルクォート' '(p.207)
ダブルクォート" "(p.209)　サブシェル(p.72)　PATH(p.181)　HOME(p.184)

位置パラメータ

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

シェルスクリプトや シェル関数の引数を参照する

書式 $1 | $2 | …… | $9

例
```
echo "$1" ……第1引数を表示する
```

基本事項

シェルスクリプトやシェル関数の引数を参照するには**位置パラメータ**を用います。シェルスクリプトやシェル関数の実行時には、その引数が順に位置パラメータにセットされます。位置パラメータの値は、$1、$2、$3…のように1以上の数値の頭に$を付けることによって参照できます。

解説

シェルスクリプトでは、引数は**位置パラメータ**によって受け渡されます。位置パラメータの$1、$2…は、C言語のmain()関数におけるargv[1]、argv[2]…に相当します。シェルスクリプト内では、位置パラメータを参照することによってユーザがどういう引数を付けてシェルスクリプトを起動したかを知ることができます。さらに、位置パラメータはシェル関数呼び出し時にも引数の受け渡しのために使用されます。

なお、位置パラメータの参照時には、その値に含まれる*や?などの**パス名展開の記号**やスペースなどの**区切文字**がシェルによって解釈されないように、基本的にはダブルクォート(" ")で囲んで"$1"のようにして参照するのがよいでしょう。

位置パラメータの表示の例

リストAのようなシェルスクリプト「paramtest」をカレントディレクトリに保存し、**図A**のように適当な引数を付けてこのparamtestを実行すると、その引数の文字列が順に表示されることがわかります。

リストA paramtest

```
#!/bin/sh

echo "$1" ……1番目の位置パラメータを表示
echo "$2" ……2番目の位置パラメータを表示
echo "$3" ……3番目の位置パラメータを表示
```


setコマンドによる位置パラメータのセット

位置パラメータの値は、シェルスクリプトやシェル関数の引数によってセットされる場合のほか、**図B**のように、setコマンドを使って再設定できます。

10番目以降の位置パラメータを参照する方法

位置パラメータはいくつでもセットすることができますが、$1、$2などの形で参照できるのは$1～$9までの9個のみです。10番目以降の位置パラメータを参照するには、**図C**のように、shiftコマンドでいったん$1～$9の範囲にシフトしてから参照するようにします。

なお、shiftコマンドを実行してしまうと、元の9番目以前の位置パラメータが失われてしまいます。これを回避するには、サブシェルの( )を使って「(shift 9; echo "$1")」のように記述する方法があります。

図A 位置パラメータの表示テスト

```
$ ./paramtest one two three          引数を付けてシェルスクリプトを起動
one                                  1番目の引数が表示される
two                                  2番目の引数が表示される
three                                3番目の引数が表示される
```

図B setコマンドによる位置パラメータのセット

```
$ set one two three                  setコマンドで位置パラメータをセット
$ echo "$1"                          試しに1番目の位置パラメータを表示
one                                  たしかにoneと表示される
$ echo "$2"                          試しに2番目の位置パラメータを表示
two                                  たしかにtwoと表示される
```

図C 10番目以降の位置パラメータを表示

```
$ set Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec    12個の位置パラメータをセット
$ echo "$1"                          試しに1番目の位置パラメータを表示
Jan                                  たしかにJanと表示される
$ shift 9                            位置パラメータを9回シフトする
$ echo "$1"                          元10番目の位置パラメータを表示
Oct                                  たしかにOctと表示される
$ echo "$2"                          元11番目の位置パラメータを表示
Nov                                  たしかにNovと表示される
```

bashなどで10番目以降の位置パラメータを参照する方法

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

bashまたはFreeBSDのshでは、10番目以降の位置パラメータについては、図Dのように、${10}、${11}…という書式を使って参照できます。なお、2桁以上になる数値部分は必ず{ }で囲む必要があります。ここで$10と記述してしまうと、位置パラメータ$1の後に、単純に文字「0」が並んでいるだけとみなされ、図Dの例では「Jan0」という意図しない表示になってしまうため、注意してください。

図D 10番目以降の位置パラメータを表示

```
$ set Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec …12個の位置パラメータをセット
$ echo "${10}" ……10番目の位置パラメータを表示
Oct ……たしかにOctと表示される
$ echo "${11}" ……11番目の位置パラメータを表示
Nov ……たしかにNovと表示される
```

注意事項

位置パラメータに直接代入はできない

位置パラメータは、シェル変数とは異なり、値を直接代入することはできません。位置パラメータの値を変更するにはsetコマンドを使います。

Memo

- すべての引数をまとめて受け継ぐには、位置パラメータをすべて並べるのではなく、特殊パラメータ"$@"を使います。
- 位置パラメータの総個数は、特殊パラメータ$#に保持されています。
- シェルスクリプト名がセットされる$0は、位置パラメータに似ていますが、特殊パラメータに分類されています。

参照

シェル関数(p.76)　ダブルクォート" "(p.209)　set(p.120)　shift(p.123)
サブシェル(p.72)　特殊パラメータ"$@"(p.167)　特殊パラメータ$#(p.171)
特殊パラメータ$0(p.165)

特殊パラメータ $0

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

起動されたシェルスクリプト名（第0引数）を参照する

例 `echo "$0"` ……シェルスクリプト名（第0引数）を表示する

基本事項

シェルスクリプトの起動時には、その**シェルスクリプト名**(第0引数)が特殊パラメータ$0にセットされます。

解説

シェルスクリプトの引数は位置パラメータ($1、$2…)によって受け渡されますが、その**シェルスクリプト名**自体は$0にセットされます。これは、C言語のmain()関数におけるargv[0]に相当します。シェルスクリプト内では、$0を参照することによって、自分自身のコマンド名を知ることができます。なお、位置パラメータとは違って、$0はシェル関数の呼び出し時には変化しません。

位置パラメータと同様、$0の参照の際にも、その値に*や?などのパス名展開の記号やスペースなどの区切り文字が含まれていてもかまわないように、基本的にはダブルクォート(" ")で囲んで"$0"として参照するのがよいでしょう。

特殊パラメータ$0の表示の例

リストAのようなシェルスクリプト「param0test」をカレントディレクトリに保存し、**図A**のように引数を付けずにparam0testを実行すると、$0にセットされたシェルスクリプト名を使って「Usage:」のメッセージが表示されます。なお$0には、状況により、そのシェルスクリプトの絶対パスまたは./などで始まる相対パスが含まれます。パス名を削除し、シェルスクリプト名のみを表示したい場合は、さらにbasenameコマンドを使います。

リストA param0test

```
if [ $# -lt 1 ]; then ……引数の個数が1未満ならば
  echo "Usage: $0 ファイル名" ……$0を使ってUsage:メッセージを表示
  exit 1 ……終了ステータス1で終了
fi ……if文の終了
```

図A 特殊パラメータ$0の表示テスト

```
$ ./param0test ……param0testを引数なしで実行
Usage: ./pamam0test ファイル名 ……$0が展開されてUsage:メッセージが表示される
```

シェルの-cオプションの場合の$0の値の変更

やや特殊な例ですが、シェルを-cオプション付きで起動(-cの次の引数が直接コマンドとして解釈される)した場合、コマンドの引数の後にさらに引数を付けると、これが$0の値としてセットされます。**図B**の例では、$0に「name」という値がセットされるため、echoコマンドによって「name」と表示されます。

図B シェルに-cオプションを付け、$0の値を変更した例

```
$ sh -c 'echo "$0"' name ……「-c コマンド引数」の後にnameを指定
name ……$0としてnameが表示される
```

注意事項

特殊パラメータ$0に代入はできない

$0に値を代入することはできません。それだけでなく、位置パラメータとは違ってsetコマンドやshiftコマンドの実行や、シェル関数の呼び出しの場合でも値は変化しません。

.コマンドで読み込んでも$0はセットされない

シェルスクリプトをコマンドとして実行するのではなく、.コマンドで読み込んだ場合は、$0の値は元のシェル上でセットされていた値のまま変化しません。

参照
位置パラメータ(p.162)　ダブルクォート" "(p.209)　basename(p.263)

特殊パラメータ "$@"

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

シェルスクリプトやシェル関数の引数すべてをそのまま引き継ぐ

書式 "$@"

例
```
mycommand "$@" ……引数すべてをそのまま引き継いでmycommandを起動
```

基本事項

$@は、"$@"のようにダブルクォート(" ")で囲んで記述することにより、"$1" "$2" "$3"…のようにすべての位置パラメータをそれぞれダブルクォートで囲んだ状態に展開されます。位置パラメータの個数が「0」個の場合は、何にも展開されません(空文字列にもなりません)。

解説

シェルスクリプトを起動する際に付けられた引数の内容を、それ以上解釈を加えずに**そのまま受け取りたい**ことがよくあります。そのような場合に"$@"を使います。"$@"は、実際の位置パラメータの個数に応じて、"$1" "$2" "$3"…のようにダブルクォート付きで展開されるため、位置パラメータの値としてスペース、*、?、そのほかの特殊な意味を持った文字が含まれていてもそれ以上は展開されません。なお、$@のようにダブルクォートなしで記述すると$1 $2 $3のようにダブルクォートなしで展開されますが、これはほとんど意味をなしません。$@を使う以上、常に"$@"とダブルクォート付きで用いないと意味がありません。

"$@"は、環境変数の設定などの前処理を行って、元の引数をそのまま引き継いでコマンド本体を起動するラッパースクリプトによく使用されます。

また、for文で、「in 値」の部分を省略すると、ここに「in "$@"」と指定したものと等価になります。

引数の受け渡しの例

リストAのようなシェルスクリプト「paramATtest」をカレントディレクトリに保存し、**図A**のようにいろいろな引数を付けてこのparamATtestを実行すると、その引数の文字列が順に表示され、正しく受け渡されていることがわかります。ここで、スペースを含んだ「my prog」という文字列が全体で1つの引数として認識されていることや、パス名展開の*が展開されていないことにも注目してください。

リストA paramATtest

```
#!/bin/sh

for arg in "$@" ……"$@"を使ってfor文を記述
do ……ループの開始
  echo "$arg" ……"$arg"の値の表示
done ……for文の終了
```

図A 引数の受け渡しのテスト

```
$ ./paramATtest file 'my prog' '*'  ……いろいろな引数を付けてシェルスクリプトを起動
file  ……引数1のfileが表示される
my prog  ……引数2はスペースを含めてmy progと表示される
*  ……引数3の*は、展開されずにそのまま表示される
```

注意事項

"$*"とは違う

"$@"に似た特殊パラメータとして$*がありますが、"$@"による位置パラメータの展開は$*や"$*"とは異なります。引数の受け渡しには"$@"を使用するべきです。

"$@"は読み出し専用

"$@"は読み出し専用であり、直接値を代入することはできません。ただし、位置パラメータの値が変化した場合は当然"$@"の内容も変化します。

Memo

●未設定のパラメータの参照をエラーとして扱うようにset -uを実行し、かつ位置パラメータの個数が0個の状態で"$@"を使用すると、シェルによってはエラーとして扱われてしまいます。これを回避するには${1+"$@"}または${@+"$@"}と記述します。$1または$@が設定されている場合のみ"$@"に展開するという意味です。ただし、bashの場合はset -uの状態で"$@"を使用してもエラーにならないため問題ありません。

参照

位置パラメータ(p.162)
for文(p.55)
ダブルクォート" "(p.209)
特殊パラメータ$*(p.169)
ラッパースクリプト(p.293)

特殊パラメータ $*

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

シェルスクリプトやシェル関数の引数すべてを1つに連結して参照する

例 echo "$*" ……すべての引数を1つに連結して表示する

基本事項

$*は、「$1 $2 $3 …」のようにすべての位置パラメータをスペースで区切って連結した状態に展開されます。"$*"のようにダブルクォートで囲むと、「"$1 $2 $3 …"」のように全体がダブルクォートで囲まれます。

解説

**"$*"**を使うと、シェルスクリプトやシェル関数に付けられた**引数全体**を参照することができます。ただし、"$*"は"$@"とは違ってダブルクォートが引数全体にかかり、すべての引数が1つにまとめられてしまいます。このため、"$*"は単にechoコマンドなどで引数全体を表示する程度には使えますが、各引数をそのまま引き継いでほかのコマンドに渡すような使い方ができません。実際、シェルスクリプトでは"$*"よりも"$@"のほうが多く使用されます。

"$*"の使用テスト

リストAのようなシェルスクリプト「paramASTERtest」をカレントディレクトリに保存し、**図A**のようにいろいろな引数を付けてこのparamASTERtestを実行すると、その引数の文字列が1つに連結されて表示されることがわかります。ここでは、for文は結局1回しかループしません。なお、"$*"のようにダブルクォートが付けられているので、パス名展開の*の展開は避けられています。

リストA paramASTERtest

```
#!/bin/sh

for arg in "$*" ……"$*"を使ってfor文を記述
do ……ループの開始
  echo "$arg" ……"$arg"の値の表示
done ……for文の終了
```

図A "$*"の使用テスト

```
$ ./paramASTERtest file 'my prog' '*'  ……いろいろな引数を付けてシェルスクリプトを起動
file my prog *  ……すべての引数が1つに連結されて表示される
```

注意事項

引数の受け渡しには"$@"を使う

シェルスクリプトやシェル関数の引数をそのまま受け渡すには、$*や"$*"ではなく、"$@"を使用するべきです。

$*は読み出し専用

$*は読み出し専用であり、直接値を代入することはできません。ただし、位置パラメータの値が変化した場合は当然$*の内容も変化します。

参照

位置パラメータ(p.162)　ダブルクォート" "(p.209)　for文(p.55)
特殊パラメータ"$@"(p.167)

特殊パラメータ $#

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

シェルスクリプトやシェル関数の引数の個数を参照する

例

```
if [ $# -lt 1 ]; then ……もし引数の個数が1未満なら
  exit 1 ……終了ステータス1で終了する
fi ……if文の終了
```

基本事項

シェルスクリプトやシェル関数の実行時には、その引数の個数が特殊パラメータ**$#**にセットされます。

解説

シェルスクリプトでは、**引数(位置パラメータ)の個数**は特殊パラメータ$#にセットされます。$#は、C言語のmain()関数におけるargcにほぼ相当しますが、シェルスクリプトの$#では、$0を個数に数えないため、C言語のargcより「1」だけ小さい値になります。シェル関数の呼び出し時には、そのシェル関数の引数の個数が一時的に$#にセットされます。

なお、$#には常に何らかの数値がセットされており、特殊な記号を含まないことが明らかであるため、参照時にダブルクォート(" ")で囲む必要はありません。

$#の表示テスト

実際に位置パラメータをセットし、その時の$#の値を表示してみましょう。**図A**では、setコマンドによって位置パラメータをセットした直後と、さらにshiftコマンドで位置パラメータをシフトした後の$#の値を表示しています。いずれも、たしかに位置パラメータの個数になっていることがわかります。

図A　$#の表示テスト

```
$ set one two three                    適当な位置パラメータを3つセットする
$ echo $#                              $#の値を表示
3                                      たしかに3と表示される
$ shift                                位置パラメータを1つシフトする
$ echo $#                              再び$#の値を表示
2                                      たしかに2と表示される
```

注意事項

特殊パラメータ$#は読み出し専用

特殊パラメータ$#は読み出し専用であり、直接値を代入することはできません。ただし、shiftコマンドやsetコマンドが実行されると$#の値は変化します。

Memo

●#はコメントの開始を示す記号ですが、$#の場合、単語の開始文字が#ではないため、当然ながらコメントとはみなされません。

参照

位置パラメータ(p.162)
shift(p.123)
シェル関数(p.76)
コメントの書き方(p.23)
set(p.120)

特殊パラメータ $?

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

終了ステータスを参照する

例
```
[ -f /some/dir/file ] ……/some/dir/fileというファイルが存在するかどうかチェック
echo $? ……testコマンドの終了ステータスを表示
```

基本事項

特殊パラメータ**$?**には、直前のリストの**終了ステータス**がセットされます[注3]。

解説

特殊パラメータ$?には、随時**リストの終了ステータス**がセットされます。たとえば、リストが単純コマンドの集まりである場合は、各コマンドの実行が終了するたびに$?の値が書き変わります。$?の値は、仮にその時点でシェルスクリプトをexitコマンドで終了した場合の終了ステータス、または、その時点でシェル関数をreturnコマンドで終了した場合の終了ステータスでもあります。

なお、$?には終了ステータスの数値がセットされていて、特殊な記号を含まないことが明らかであるため、参照時にダブルクォート(" ")で囲む必要はありません。

$?の参照例

実際にコマンドを実行し、$?の値をechoコマンドで表示してみましょう。図Aのようにfalseコマンドを実行すると、コマンドの終了ステータスが「1」になるため、次のechoコマンドで「1」が表示されます。ところが、この時点でechoコマンド自体の終了ステータスが「0」になるため、再度echoで$?の値を表示すると「0」になることがわかります。

図A $?の参照例

```
$ false                 falseコマンドを実行して終了ステータスを1にする
$ echo $?               $?の値を表示
1                       たしかに1と表示される
$ echo $?               再度$?の値を表示
0                       今度は直前のechoコマンド自体の終了ステータス0が表示される
```

注3 コマンドの終了ステータスの項(p.25)も合わせて参照してください。

注意事項

$?は直後に参照すること

$?の参照は、コマンド実行の直後に行う必要があります。さらに、$?の参照のためにechoコマンドなどを利用すると、その実行後に$?の値が書き変わってしまいます。したがって、終了ステータスを後で参照したい場合は、コマンド実行の直後に$?をほかのシェル変数に代入して保存する必要があります。

```
cmp -s file1 file2 ……file1とfile2の内容を比較
status=$? ……その終了ステータスをシェル変数に保存
echo 'file1とfile2を比較しました' ……メッセージを表示
exit "$status" ……保存されている終了ステータスを使ってexit
```

参照

コマンドの終了ステータス(p.25)　exit(p.103)　return(p.118)

特殊パラメータ $!

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

最も新しくバックグラウンドで起動したコマンドのプロセスIDを参照する

例

```
sleep 10 & ……適当なコマンドをバックグラウンドで起動
echo $! ……そのコマンドのプロセスIDを表示する
```

基本事項

特殊パラメータ**$!**には、最も新しく**バックグラウンド**で起動した**コマンドのプロセスID**がセットされます。コマンドを一度もバックグラウンドで起動していない場合は、空文字列になります。

解説

リストの区切り文字や終端に&を付け、コマンドを**バックグラウンド**で起動すると、その**コマンドのプロセスID**が**$!**にセットされます。したがって、$!をwaitコマンドやkillコマンドでプロセスIDを参照する場合などに利用できます。なお、$!の値はバックグラウンドでコマンドを起動するたびに上書きされるため、2個以上のコマンドをバックグラウンドで起動し、そのプロセスIDを知りたい場合は、適宜シェル変数に$!の値を代入して保存する必要があります。

参照

wait(p.133)　kill(p.141)

特殊パラメータ $$

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

シェル自身のプロセスIDを参照する

例 `touch /tmp/tempfile$$` ……/tmp/tempfileXXXXという形式のファイルを作成する

基本事項

シェルの起動時には**シェル自身のプロセスID**の数値が特殊パラメータ$$にセットされます。

解説

シェル自身のプロセスIDがセットされている$$は、おもにテンポラリファイルのファイル名を生成するために利用されます。シェルスクリプト中でテンポラリファイルを使用する場合、ファイル名として固定の文字列を使用すると、同じシェルスクリプトが同時に複数起動された場合に、テンポラリファイル名が競合して正常に動作しません。そこで、テンポラリファイルのファイル名に、シェルのプロセスIDというユニークな(一意の)値を埋め込むという方法が取られるのです。

なお、$$にはプロセスIDという数値がセットされていて、特殊な記号を含まないことが明らかであるため、参照時にダブルクォート(" ")で囲む必要はありません。

$$を利用してテンポラリファイルを作る

実際に$$を利用してテンポラリファイルを作成している例を**リストA**に示します。ここでは、$$を含めたテンポラリファイル名を、いったんシェル変数**TEMPFILE**に代入して使用していますが、そのファイル名は、シェルスクリプトの起動タイミングによって、「/tmp/tempfile1234」になったり、「/tmp/tempfile5678」になったりするはずです。

リストA $$を利用してテンポラリファイルを作る

```
TEMPFILE=/tmp/tempfile$$ ……$$を含めてテンポラリファイルのファイル名を決定
nkf -Se "$file" > "$TEMPFILE" ……nkfコマンドで EUC に変換してテンポラリファイルに出力
mv -f "$TEMPFILE" "$file" ……そのテンポラリファイルを元のファイルに上書き
```

注意事項

サブシェル内の$$は元のシェルの値と同じ

サブシェルの( )の中に$$を記述しても、$$の値はサブシェルのプロセスIDではなく、元のシェルのプロセスIDに展開されます。

特殊パラメータ$$は読み出し専用

特殊パラメータ$$は読み出し専用であり、直接値を代入することはできません。

特殊パラメータ $-

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

現在のシェルに設定されている オプションフラグを参照する

例 `echo $-` ……現在のシェルのオプションフラグを表示する

基本事項

特殊パラメータ**$-**は、現在の**シェルのオプションフラグ**に展開されます。

解説

シェルには、**シェルの起動時**または**setコマンド**によって設定可能な、-aや-eそのほかの**オプションフラグ**があります。このオプションフラグは、特殊パラメータ**$-**を使って参照することができます。

$-を使った実行例

$-を使った実行例を**図A**に示します。とくにオプションを付けずにシェルを起動した場合でも、いくつかのオプションフラグは最初から設定されています。図のようにsetコマンドを使ってオプションフラグを操作すると、特殊パラメータ$-の値が変化していることがわかります。

図A $-を使った実行例

```
$ echo $-                    現在のオプションフラグを表示
himBH                        -h、-iフラグほか、このようにセットされている
$ set -a                     setコマンドでaフラグをセット
$ echo $-                    現在のオプションフラグを表示
ahimBH                       たしかにaフラグが追加された
$ set +a                     setコマンドでaフラグをリセット
$ echo $-                    現在のオプションフラグを表示
himBH                        たしかにaフラグが削除された
```

参照

set(p.120)

特殊パラメータ $_

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

直前に実行したコマンドの最後の引数を参照する

例
```
ls -l /some/dir ……/some/dirディレクトリに対してls -lコマンドを実行
cd $_ ……/some/dirディレクトリに移動
```

基本事項

特殊パラメータ$_には、直前に実行したコマンドの最後の引数(引数がない場合はコマンド名)がセットされます。シェルを起動した直後には、シェル自身のコマンド名がセットされます。

解説

直前に実行したコマンドの最後の引数を再利用して別のコマンドを実行したいような場合、$_を利用すると便利でしょう。ただし、$_は、おもにコマンドラインでの入力の手間を省くためのものであり、シェルスクリプト上では通常は使用されません。

$_を使った実行例

$_を使った実行例を図Aに示します。直前のコマンドの最後の引数がセットされていることがわかります。多くのコマンドでは最後の引数は何らかのファイル名またはディレクトリ名となるため、$_を利用して次のコマンドの入力を簡略化できるでしょう。

図A $_を使った実行例

```
$ ls -F work                    workディレクトリに対してls -Fコマンドを実行
bin/  memo.txt  src/  tmp/      存在するファイルが表示される
$ echo $_                       ここで$_の値を参照すると
work                            最後の引数のworkがセットされていることがわかる
```


環境変数の設定

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

シェル変数をexportして環境変数を設定する

書式 変数名=[値]; export 変数名

例
```
LANG=C; export LANG ……環境変数LANGをCという値にセットする
```

解説

シェルでは、環境変数はexportされたシェル変数として扱われるため、**環境変数を設定**するには、同名のシェル変数([変数名])に[値]を代入するなどして設定するとともに、exportコマンドを使ってシェル変数([変数名])をexportする必要があります。

なお、1つのコマンドのみ環境変数を一時的に変更したい場合は、次項のように、exportコマンドを使わずに、シェルの文法上で環境変数の一時変更を行えます。

環境変数を未設定状態にするにはunsetコマンドを使います。ただし、unsetによって環境変数だけでなく、シェル変数自体が未設定になります。

参照

export(p.105)　環境変数の一時変更(p.180)

環境変数の一時変更

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

単純コマンドの左側に環境変数の代入文を記述する

書式 [変数名=[値] …] 単純コマンド [引数 …]

例
```
LANG=C date ……環境変数LANGをCに一時的に変更してdateを実行
```

基本事項

環境変数を継続的に設定(または変更)するのではなく、**特定の単純コマンドのみ**に対して**一時的に環境変数を設定**することができます。冒頭の書式のように単純コマンドのコマンド名よりも左側に変数名と値を用いた代入文を記述すると、これは指定の単純コマンドの実行時のみ有効な環境変数と解釈されます。

この方法では、exportコマンドを使わなくても環境変数を設定できますが、環境変数が設定されるのは指定の**単純コマンド**のみであり[注4]、シェル自体の環境変数は一切変更されません。

解説

図Aは、環境変数**LANG**を一時変更してdateコマンドを実行している例です。日本語環境では**LANG**には「ja_JP.eucJP」がセットされており、dateは日本語で日付を表示しますが、**図A❶**のように**LANG**を「C」に一時変更するとdateの表示は英語に切り替わります。しかし、環境変数の変更はこのdateの実行1回限りであり、元のシェルの**LANG**の値は変わりません。また**図A❷**のように、**LANG**と**TZ**の2つの環境変数を同時に一時変更することもできます。ここでは「TZ=UTC」を追加したことにより、時刻が「UTC」で表示されています。

図A 環境変数の一時変更例

```
$ printenv LANG                    現在の環境変数LANGの値を表示
ja_JP.eucJP                        日本語EUCのlocaleに設定されている
$ date                             dateコマンドで日付を表示させる
2038年  1月19日 火曜日12:14:05 JST  日本語で日付が表示される
$ LANG=C date                      ❶環境変数LANGをCに一時変更し、dateを実行する
Tue Jan 19 12:14:06 JST 2038       英語で日付が表示される
$ printenv LANG                    printenvコマンドで環境変数LANGの値を確認
ja_JP.eucJP                        日本語EUCのlocaleのまま変わっていない
$ LANG=C TZ=UTC date               ❷LANGをCにして、さらにタイムゾーンをUTCに変更してdateを実行
Tue Jan 19 03:14:07 UTC 2038       英語で、UTCのタイムゾーンで日付が表示される
```

注4 **環境変数の一時変更ができるのは単純コマンドのみ**です。if文などの構文やサブシェルの()の前に代入文を書くと文法エラーになります。

PATH

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

外部コマンドの検索パスを設定するシェル変数

解説

シェルが外部コマンドを実行する際、コマンド名が/を含んでいない場合[注5]、シェル変数**PATH**に設定されている:で区切られた複数のディレクトリを左から順に検索し、実行するべき外部コマンドを探します。

PATHの値は、ユーザのログイン時に、**"$HOME"/.profile**などのファイルによって設定されるのが普通です。また、**PATH**は環境変数としてexportし、シェルから起動される子プロセスにも**PATH**の設定が反映されるようにします。

PATHの追加の例

図Aは、現在の**PATH**の先頭に、**"$HOME"/bin**を追加する例です。**PATH**の追加では、それまでに設定されていた**PATH**もそのまま有効になるように、「PATH=$HOME/bin:$PATH」のように、以前の**PATH**を参照しつつ、新たなディレクトリを追加し、:で区切って代入する必要があります。シェル変数への代入時にはダブルクォートはいりません。なお、**PATH**はすでにexportされているのが普通ですが、念のため再度exportしておくとよいでしょう。

これで**"$HOME"/bin**が**PATH**に追加されたことにより、以降**"$HOME"/bin**の下にある外部コマンドが絶対パスなどの指定なしで実行できるようになります。

図A PATHの追加の例

```
$ echo "$PATH"                      現在のPATHの値を表示
/usr/local/bin:/usr/X11R6/bin:/usr/bin:/bin   このように4つのディレクトリが設定されている
$ PATH=$HOME/bin:$PATH             PATHの先頭に$HOME/binを追加
$ echo "$PATH"                     再度PATHの値を表示
/home/guest/bin:/usr/local/bin:/usr/X11R6/bin:/usr/bin:/bin   たしかに追加された
$ export PATH                      念のため、再度PATHを環境変数にexport
```

参照

export(p.105)　HOME(p.184)

注5 絶対パスや、カレントディレクトリからの相対パスではない場合を指します。

PS1 / PS2

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

シェルのプロンプトを設定するシェル変数

解説

シェルがシェルスクリプトを実行中(非対話シェル)ではなく、コマンドラインを実行中(対話シェル)の場合は、コマンド1行ごとに標準エラー出力(通常は画面)にプロンプトを表示し、ユーザにコマンド入力を促します。

このプロンプトの文字列は、プライマリプロンプトであるシェル変数**PS1**に設定されており、ユーザの好みにより、変更することも可能です。**PS1**のデフォルトは、一般ユーザの場合は「$ 」、root(特権ユーザ)の場合は「# 」です。

一方、セカンダリプロンプトの**PS2**にはデフォルトで「> 」という文字列がセットされており、これは、if文/for文などの構文の入力中や、クォートの途中での改行時などに表示され、まだコマンドが完結していない状態であることを示します(**表A**)。

プロンプトの表示例

図Aは、echoコマンドの引数として、改行を含むメッセージを与えている例です。ここでは、シングルクォートの途中で改行したところで、シェルのプロンプトが**PS2**の「> 」に変わっていることがわかります。その後、2行目のメッセージを入力し、シングルクォートを閉じて改行するとechoコマンドが完結するため、実際にechoが実行され、改行を含む2行のメッセージが表示されます。最後には元通り**PS1**の「$ 」のプロンプトに戻ります。

表A シェルのプロンプト

シェル変数	意味	デフォルト値
PS1	プライマリプロンプト	「$ 」
PS2	セカンダリプロンプト	「> 」

図A プロンプトの表示例

```
$ echo 'hello                echoコマンドの引数のメッセージのシングルクォートの途中で改行
> world'                     PS2が表示されるので、メッセージの続きを入力しシングルクォートを閉じる
hello                        echoコマンドが実行され、メッセージが2行分表示される(1行目)
world                        (2行目)
$                            再びPS1が表示される
```


PS1に設定できる特殊文字列

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

bashでは、**PS1**などのプロンプトの値として、\uなどの\で始まる特殊文字列を使用することができ、たとえば、\uはプロンプトの表示時に「ユーザ名」に展開されます。実際に**PS1**に特殊文字列を設定している様子を**図B**に示します。**PS1**への代入時には、\やスペースやそのほかの記号が解釈されないように、文字列全体をシングルクォートで囲みます。この設定により、シェルのプロンプトが「[guest@myhost doc]$ 」のようなスタイルに変わります。なお、プロンプトで使用できる特殊文字列についての詳細は、bashのオンラインマニュアルを参照してください。

図B 特殊文字列を使用したプロンプト

```
$ PS1='[\u@\h \W]\$ '        PS1に、\uなどの特殊文字列を代入※
[guest@myhost doc]$          するとユーザ名などを含むプロンプトに変わる
```

※プロンプトに使用している特殊文字列

- \u:ユーザ名
- \h:ホスト名(ドメイン部分を除く)
- \W:カレントディレクトリ名(パス部分を除く)

Memo

- bashでは、**PS1**、**PS2**のほかに、select文のプロンプトとして使用される**PS3**や、オプションフラグ-x設定時の表示に使用される**PS4**も存在します。

HOME

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

自分自身のホームディレクトリが設定されているシェル変数

解説

シェルを実行中のユーザのホームディレクトリの絶対パスは、シェル変数HOMEに設定されています。これは、ユーザのログイン時に設定される環境変数HOMEの値を受け継いだものです。

HOMEの値は、cdコマンドを引数なしで実行してホームディレクトリに移動する場合や、チルダ展開の際に参照されます。詳しくはcdコマンドの項(p.95)を参照してください。

参照

cd(p.95)　　チルダ展開~(p.233)

IFS

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

単語分割に用いられる区切り文字が設定されているシェル変数

解説

シェルが単語分割を行う際には、シェル変数IFSに設定されている文字を区切り文字として使用します。IFSの値のデフォルトは、スペース、タブ、改行の3文字です。詳しくは単語分割の項(p.237)を参照してください。