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

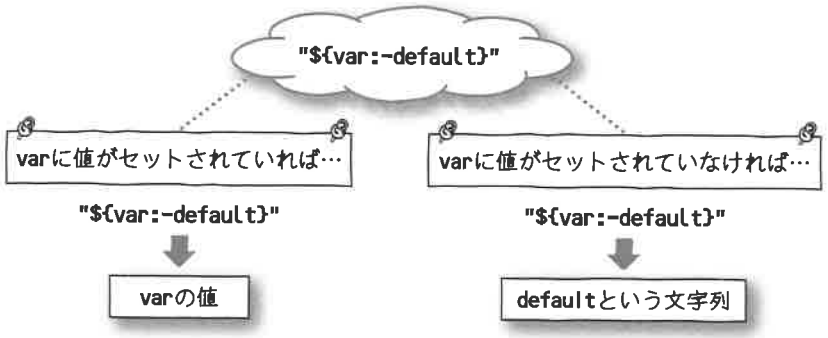
シェルが単語分割を行う際には、シェル変数**IFS**に設定されている文字を区切り文字として使用します。**IFS**の値のデフォルトは、スペース、タブ、改行の3文字です。詳しくは単語分割の項(p.237)を参照してください。

>第8章 パラメータ展開

パラメータ展開の概要

パラメータ展開とは、パラメータ(シェル変数と位置パラメータと特殊パラメータ)を、実際の値に展開することです。単なるシェル変数の参照も、パラメータ展開の一つです。しかし、パラメータ展開では、単にシェル変数などのパラメータを参照するだけでなく、パラメータの参照時に、"${var:-default}"のように何らかの条件判断を行えます。具体的にいうと、シェル変数の参照は通常はパラメータ名の頭に$を付け、さらにダブルクォートを付けて"$var"の形にしますが、ここで"${var:-default}"のような{ }を使った書き方ができ、シェル変数がセットされているかどうかによって条件判断をして違う動作をさせることができるのです(**図A**)。本章では、このような条件判断をともなうパラメータの参照など、パラメータの展開方法をまとめて「パラメータ展開」として説明します。

図A **パラメータ展開の例**



Column

パラメータ展開とダブルクォート

パラメータ展開で、${var:-default}のように記述する場合も、$varの場合と同じく全体をダブルクォートで囲んで、"${var:-default}"と記述し、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。とくに、$の後に{ }のカッコがあるため、あたかもすでにクォートされているような錯覚を受けやすいので注意してください。ダブルクォートの項(p.209)も合わせて参照してください。

${パラメータ:-値}と${パラメータ-値}

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

パラメータのデフォルト値を指定する

書式 ${パラメータ:-値}
${パラメータ-値}

例
```
cp file "${1:-/tmp}" ……fileを、引数1で指定したディレクトリにコピー、
                       引数1が未設定または空文字列の場合は/tmpにコピー
```

基本事項

${パラメータ:-値}は、パラメータが設定されていないかまたは空文字列の場合に指定した値に展開され、それ以外の場合は通常通りパラメータ自身の値に展開されます。

${パラメータ-値}は、パラメータが設定されていない場合に指定した値に展開され、それ以外の場合は通常通りパラメータ自身の値に展開されます。

解説

シェルスクリプト中でシェル変数や位置パラメータなどのパラメータを参照する際、これらのパラメータが未設定でも動作するように、そのデフォルト値を指定したい場合があります。このような時、${パラメータ:-値}や${パラメータ-値}が便利です。

${パラメータ:-値}では、パラメータが空文字列の場合もパラメータが未設定の場合と同様に扱いますが、${パラメータ-値}では、単純にパラメータ自体がセットされているかどうかだけで判断します。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

なお、これらのパラメータ展開では、展開される値は変化しても、パラメータ自身の値は変化しません。パラメータ自身の値も変化させたい場合は${パラメータ:=値}のパラメータ展開を用います[注1]。

if文での記述

冒頭の例の${パラメータ:-値}を参考までにif文で記述すると**リストA**のようになります。if文よりもパラメータ展開のほうが簡潔に記述できることがわかります。

注1 「${パラメータ:=値}と${パラメータ=値}」(p.189)を参照してください。

リストA ${パラメータ:-値}をif文で記述した例

```
if [ -n "$1" ] ……位置パラメータ"$1"が（空文字列以外に）セットされている場合
then
  cp file "$1" ……fileを"$1"にコピー
else ……位置パラメータ"$1"が未設定または空文字列の場合
  cp file /tmp ……fileを/tmpにコピー
fi
```

Column

空文字列のパラメータをセットするには

空文字列のパラメータは、次のように、シェル変数名と=とを記述し、=の右側に何も書かないことによってセットできます。この場合、パラメータ(シェル変数)varはセットされていますが、中身は空文字列という状態になります。

```
var= ……varはセットされているが、中身は空文字列
```

参照

ダブルクォート" "(p.209)　${パラメータ:=値}と${パラメータ=値}(p.189)

${パラメータ:=値}と${パラメータ=値}

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

パラメータにデフォルト値を代入する

書式 ${パラメータ:=値}
${パラメータ=値}

例

```
cp file "${TMPDIR:=/tmp}" …… シェル変数TMPDIRが未設定または空文字列の場合は/tmpを代入し、シェル変数TMPDIRの示すディレクトリにfileをコピー
```

基本事項

${パラメータ:=値}は、パラメータが設定されていないかまたは空文字列の場合には指定した値が代入され、その値に展開されます。それ以外の場合は通常通りパラメータ自身の値に展開されます。

${パラメータ=値}は、パラメータが設定されていない場合には指定した値が代入され、その値に展開されます。それ以外の場合は通常通りパラメータ自身の値に展開されます。

解説

シェルスクリプト中でシェル変数を参照する際、シェル変数が未設定の場合にはその**デフォルト値を代入したい**場合があります。このような時、${パラメータ:=値}や${パラメータ=値}が便利です。

${パラメータ:=値}では、パラメータが空文字列の場合もパラメータが未設定の場合と同様に扱いますが、${パラメータ=値}では、単純にパラメータ自体がセットされているかどうかだけで判断します。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

なお、これらのパラメータ展開では、パラメータは値が代入可能なシェル変数である必要があり、位置パラメータや特殊パラメータは指定できません。

if文での記述

冒頭の例の${パラメータ:=値}を参考までにif文で記述すると**リストA**のようになります。if文よりもパラメータ展開のほうが簡潔に記述できることがわかります。

:コマンドの利用

パラメータ展開のみを、:コマンドを利用して先に行っておき、以降は通常通りシェル変数を参照するようにすることもできます。冒頭の例を:コマンドを利用するように書き直すと、**リストB**のようになります。

リストA ${パラメータ:-値}をif文で記述した例

```
if [ -z "$TMPDIR" ] ……シェル変数TMPDIRが未設定または空文字列の場合
then
  TMPDIR=/tmp ……シェル変数TMPDIRにデフォルト値の/tmpを代入
fi
cp file "$TMPDIR" ……fileをシェル変数TMPDIRの指すディレクトリにコピー
```

リストB :コマンドを利用した例

```
: ${TMPDIR:=/tmp} ……シェル変数TMPDIRが未設定または空文字列の場合は/tmpを代入
cp file "$TMPDIR" ……fileをシェル変数TMPDIRの指すディレクトリにコピー
```

参照

ダブルクォート" "(p.209)　　:コマンド(p.87)

${パラメータ:?値}と${パラメータ?値}

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

パラメータ未設定時にエラーメッセージを出してシェルスクリプトを終了する

書式 ${パラメータ:?値}
${パラメータ?値}

例

```
cp "${1:?ファイルが指定されていません}" /tmp ……引数1で指定されたファイルを/tmpにコピー、引数1が未設定または空文字列の場合はエラーメッセージを出して終了
```

基本事項

${パラメータ:?値}は、パラメータが設定されていないか、または空文字列の場合には、指定した値がエラーメッセージとして表示され、シェルスクリプトを終了します。それ以外の場合は通常通りパラメータ自身の値に展開されます。

${パラメータ?値}は、パラメータが設定されていない場合には指定した値がエラーメッセージとして表示され、シェルスクリプトを終了します。それ以外の場合は、通常通りパラメータ自身の値に展開されます。

いずれの場合も、シェルがシェルスクリプトではなく、コマンドラインを実行しているシェル(対話シェル)の場合はシェルは終了されません。

値の指定は省略でき、値を省略した場合はシェルによって既定のエラーメッセージが表示されます。

解説

シェルスクリプト中で、必要なパラメータが未設定の場合には**エラーメッセージを出して終了したい**場合があります。このような時、${パラメータ:?値}や${パラメータ?値}が便利です。このパラメータ展開では、パラメータが未設定でエラーになった場合には、コマンドの実行前にシェルスクリプトが終了します。

${パラメータ:?値}では、パラメータが空文字列の場合もパラメータが未設定の場合と同様に扱いますが、${パラメータ?値}では、単純にパラメータ自体がセットされているかどうかだけで判断します。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

if文での記述

冒頭の例の${パラメータ:?値}を参考までにif文で記述すると**リストA**のようになります。if文よりもパラメータ展開のほうが簡潔に記述できることがわかります。

:コマンドの利用

パラメータ展開のみを、:コマンドを利用して先に行っておき、以降は通常通りシェル変数を参照するようにすることもできます。冒頭の例を:コマンドを利用するように書き直すと、**リストB**のようになります。

エラーメッセージの省略

${パラメータ:?}のように、エラーメッセージの指定を省略することもできます。この場合、エラーメッセージはシェル側で用意されたメッセージになります。**図A**は、シェル変数varに空文字列以外の値がセットされているかどうかをチェックしており、その結果、エラーメッセージが表示されています。なお、ここではコマンドラインで直接実行しているため、エラー時にシェルは終了されません。

リストA ${パラメータ:-値}をif文で記述した例

```
if [ -z "$1" ] ……引数1が未設定または空文字列の場合
then
  echo 'ファイルが指定されていません' ……エラーメッセージを表示
  exit 1 ……終了ステータス1で終了
fi
cp "$1" /tmp ……引数1で指定のファイルを/tmpにコピー
```

リストB :コマンドを利用した例

```
: ${1:?ファイルが指定されていません} ……引数1が未設定または空文字列の場合はエラーメッセージを出して終了
cp "$1" /tmp ……引数1で指定のファイルを/tmpにコピー
```

図A エラーメッセージの省略

```
$ unset var                                  念のためシェル変数varをunsetする
$ : ${var:?}                                 エラーメッセージを省略してパラメータをチェック
bash: var: parameter null or not set         varが未設定という、シェルからのエラーメッセージ
```

参照

ダブルクォート" "(p.209)　　:コマンド(p.87)

${パラメータ:+値}と${パラメータ+値}

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

パラメータが設定されている場合のみ指定の値に展開する

書式
${パラメータ:+値}
${パラメータ+値}

例

```
LD_LIBRARY_PATH=/usr/local/myapp/lib${LD_LIBRARY_PATH:+:}$LD_LIBRARY_PATH
                                  :…… LD_LIBRARY_PATHの先頭にディレクトリを追加
                                      (すでにLD_LIBRARY_PATHが設定されている場合は:で区切る)
export LD_LIBRARY_PATH …… 環境変数としてエクスポートする
```

基本事項

${パラメータ:+値}は、パラメータが空文字列以外に設定されている場合に指定した値に展開され、それ以外の場合は空文字列に展開されます。

${パラメータ+値}は、パラメータが設定されている場合に指定した値に展開され、それ以外の場合は空文字列に展開されます。

解説

${パラメータ:+値}は${パラメータ:-値}とは逆に、パラメータが設定されている場合に指定の値に展開されます。したがって、すでにパラメータが設定されている場合のみ、何らかの別の値に展開したい場合に使用できます。パラメータに元々設定されていた値は展開の結果には現れません。

${パラメータ:+値}では、パラメータが空文字列の場合もパラメータが未設定の場合と同様に扱いますが、${パラメータ+値}では、単純にパラメータ自体がセットされているかどうかだけで判断します。

冒頭の例では展開結果を直接シェル変数に代入しているため、全体のダブルクォートは省略していますが、一般には、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

${パラメータ:+値}の効用

冒頭の例では、共有ライブラリのディレクトリを指定する**LD_LIBRARY_PATH**という環境変数に「/usr/local/myapp/lib」というディレクトリを追加しています。このとき、**LD_LIBRARY_PATH**にすでにほかのディレクトリが設定されている場合は、複数のディレクトリを:で区切って並べる必要があります。そこで、:が必要な時のみ:という文字に展開されるように、${LD_LIBRARY_PATH:+:}というパラメータ展開を使用しているのです。

これを、単純に**リストA**のように記述すると、元々**LD_LIBRARY_PATH**が設定されていなかった場合に、その値の右端に:が余分に付いてしまいます。

if文での記述

冒頭の例をif文で記述すると**リストB**のように、少々冗長になります。この例からも、if文よりもパラメータ展開のほうが簡潔に記述できることがわかります。

リストA 単純に設定して:が余分に付く例

```
LD_LIBRARY_PATH=/usr/local/myapp/lib:$LD_LIBRARY_PATH…LD_LIBRARY_PATHが未設定の場合、右端に:が余分に付いてしまう
export LD_LIBRARY_PATH ……環境変数としてエクスポートする
```

リストB ${パラメータ:+値}をif文で記述した例

```
if [ -n "$LD_LIBRARY_PATH" ]……LD_LIBRARY_PATHが(空文字列以外に)セットされている場合
then
  LD_LIBRARY_PATH=/usr/local/myapp/lib:$LD_LIBRARY_PATH …:をはさんでディレクトリを追加
else ……LD_LIBRARY_PATHが未設定または空文字列の場合
  LD_LIBRARY_PATH=/usr/local/myapp/lib ……LD_LIBRARY_PATHにそのままディレクトリを代入
fi
export LD_LIBRARY_PATH ……環境変数としてエクスポートする
```

参照

ダブルクォート" "(p.209)

${#パラメータ}

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

パラメータの値の文字列の長さを求める

書式 ${#パラメータ}

例

```
echo ${#var} ……シェル変数varの中身の文字列の長さを表示する
```

基本事項

${#パラメータ}は、そのパラメータの値の文字列の長さの数値に展開されます。パラメータが未設定の場合は「0」になります。

解説

シェル変数や位置パラメータなどにセットされている文字列の長さを知りたい場合、${#パラメータ}というパラメータ展開が使えます。ただし、Solarisのshなど、対応していないシェルが存在するため、使用には注意が必要です。

このパラメータ展開では、必ず何らかの数値に展開されるため、全体をダブルクォートで囲む必要はありません。

wcコマンドを使う方法

より移植性が高くなるように、このパラメータ展開を使わずにwcコマンドを使うと図A❶のようになります。ここでは、echoコマンドに-nオプションを付けて改行コードが付かないようにした上で、パイプ(|)でwcコマンドに渡し、wcコマンドでは文字数のみを表示する-cオプションを付けて長さを数えています(wcコマンドのバージョンによっては、数値の頭にいくつかのスペースが付いて表示されます)。なお、Solarisではechoコマンドで-nオプションが使えないため、図A❷のようにprintfコマンドを利用することになります(文字列の最後に\cを付けてechoする方法では、\c以外に\t、\n、\rなどの文字列が含まれているとそれらが展開されてしまうため、正しい文字列の長さが得られません)。

図A ${#パラメータ}とwcコマンドの実行例

```
$ var='Hello World'                    シェル変数varに適当な文字列を代入
$ echo ${#var}                         パラメータ展開を利用して文字列の長さを表示
11                                     11文字と表示される
$ echo -n "$var" | wc -c               wcコマンドを使って文字列の長さを計数❶
11                                     11文字と表示される
$ printf %s "$var" | wc -c             wcコマンドを使って文字列の長さを計数❷
11                                     11文字と表示される
```

注意事項

${#@}や${#*}はシェルによって違う

bashでは、${#@}や${#*}はどちらも位置パラメータの個数、つまり$#と同じになります。

しかし、FreeBSDのshでは、${#@}や${#*}は、"$*"の文字列の長さ、すなわち、すべての位置パラメータをスペースで区切って並べた文字列の長さになります。

いずれにしても移植性のため、${#@}や${#*}の使用は控えたほうがいいでしょう。

参照

echo(p.137)　wc(p.270)

${パラメータ#パターン}と${パラメータ##パターン}

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

パラメータの値の文字列の左側から一定のパターンを取り除く

書式 ${パラメータ#パターン}
${パラメータ##パターン}

例 `echo "${HOME##*/}"` ……"$HOME"からパス名を取り除いたディレクトリ名を表示

基本事項

${パラメータ#パターン}は、パラメータの値の文字列の**左側**からパターンに一致する最短の部分が取り除かれます。

${パラメータ##パターン}は、パラメータの値の文字列の**左側**からパターンに一致する最長の部分が取り除かれます。

いずれも、パターンにはパス名展開の特殊文字を使えます[注2]。

解説

シェル変数や位置パラメータにセットされている文字列に対して簡単な操作を行いたい場合、${パラメータ#パターン}や${パラメータ##パターン}が有用な場合があります。これらのパラメータ展開では、**左側**からパターンに一致した文字列を取り除くため、basenameコマンドに近い動作を行うことも可能です。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

basenameコマンドとの比較

図Aのようにシェル変数に「/usr/local/bin」のようなパス名が代入されている場合、${dir##*/}というパラメータ展開は、パターンに記述されている「*/」が「/usr/local/」までに一致するため、この部分が取り除かれて「bin」だけが残ります。これは結果的にbasenameコマンドの動作と同じになります。

ただし、dirに「/usr/local/bin/」のように右端に「/」を付けたパス名が代入されている場合はすべての文字列が削除されてしまい、basenameコマンドとは動作が異なってしまいます。

注2　パス名展開については10.2節を参照してください。

図A basenameコマンドとの比較

```
$ dir=/usr/local/bin                    シェル変数dirに/usr/local/binを代入
$ echo "${dir##*/}"                     パラメータ展開でbasenameコマンドに近い動作をさせる
bin                                     期待通り、ディレクトリ名のbinが表示される
$ basename "$dir"                       basenameコマンドを使うと
bin                                     当然binが表示される
```

左側からの最短一致と最長一致

${パラメータ#パターン}は最短一致、${パラメータ##パターン}は最長一致の文字列を取り除きます。これらの違いがわかる例を**図B**に示します。この例では「*.」というパターンで左から「.」までを取り除いていますが、最短一致では「backup.」までが一致して取り除かれ「tar.gz」が残るのに対し、最長一致では「backup.tar.」までが一致して「gz」のみが残ります。

図B 最短一致と最長一致の違いがわかる例

```
$ file=backup.tar.gz                    シェル変数fileにbackup.tar.gzを代入
$ echo "${file#*.}"                     最短一致で左から.までを取り除くと
tar.gz                                  左側の.までのみが取り除かれるためtar.gzが残る
$ echo "${file##*.}"                    最長一致で左から.までを取り除くと
gz                                      右側の.までが取り除かれるためgzのみが残る
```

注意事項

${@#パターン}や${*#パターン}はシェルによって違う

特殊パラメータ$@や$*に対してこれらのパラメータ展開を使用した場合、その動作はbashとFreeBSDのshでは異なるため注意が必要です。bashでは、複数の位置パラメータそれぞれから指定のパターンに一致した文字列を取り除いてから、$@や$*の展開が行われます。一方FreeBSDのshでは、$@や$*の展開を行った文字列全体から、指定のパターンに一致した文字列が取り除かれます。

参照

ダブルクォート" "(p.209)　　basename(p.263)

${パラメータ%パターン}と${パラメータ%%パターン}

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

パラメータの値の文字列の右側から一定のパターンを取り除く

書式 ${パラメータ%パターン}
${パラメータ%%パターン}

例 `echo "${HOME%/*}"` ……"$HOME"の親ディレクトリのディレクトリ名を表示

基本事項

${パラメータ%パターン}は、パラメータの値の文字列の**右側**からパターンに一致する最短の部分が取り除かれます。

${パラメータ%%パターン}は、パラメータの値の文字列の**右側**からパターンに一致する最長の部分が取り除かれます。

いずれも、パターンにはパス名展開の特殊文字を使えます[注3]。

解説

シェル変数や位置パラメータにセットされている文字列に対して簡単な操作を行いたい場合、${パラメータ%パターン}や${パラメータ%%パターン}が有用な場合があります。これらのパラメータ展開では、**右側**からパターンに一致した文字列を取り除くため、親ディレクトリ名を表示するdirnameコマンドに近い動作を行うことも可能です。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

dirnameコマンドとの比較

図Aのようにシェル変数に「/usr/local/bin」のようなパス名が代入されている場合、${dir%/*}というパラメータ展開は、パターンに記述されている「/*」が右側の「/bin」の部分に一致するため、この部分が取り除かれて「/usr/local」だけが残ります。これは結果的にdirnameコマンドの動作と同じになります。

ただし、「dir」に「/usr/local/bin/」のように右端に「/」を付けたパス名が代入されている場合や、「/usr」のようにdirnameの結果が「/」になる場合には期待通り動作せず、dirnameコマンドとは動作が異なってしまいます。

注3　パス名展開については10.2節を参照してください。

図A dirnameコマンドとの比較

```
$ dir=/usr/local/bin          シェル変数dirに/usr/local/binを代入
$ echo "${dir%/*}"            パラメータ展開でdirnameコマンドに近い動作をさせる
/usr/local                    期待通り、/usr/localと表示される
$ dirname "$dir"              dirnameコマンドを使うと
/usr/local                    当然/usr/localと表示される
```

右側からの最短一致と最長一致

${パラメータ%パターン}は最短一致、${パラメータ%%パターン}は最長一致の文字列を取り除きます。これらの違いがわかる例を**図B**に示します。この例では「.*」というパターンで右から「.」までを取り除いていますが、最短一致では「.gz」までが一致して取り除かれ「backup.tar」が残るのに対し、最長一致では「.tar.gz」までが一致して「backup」のみが残ります。

図B 最短一致と最長一致の違いがわかる例

```
$ file=backup.tar.gz          シェル変数fileにbackup.tar.gzを代入
$ echo "${file%.*}"           最短一致で右から.までを取り除くと
backup.tar                    右側の.までのみが取り除かれるためbackup.tarが残る
$ echo "${file%%.*}"          最長一致で右から.までを取り除くと
backup                        左側の.までが取り除かれるためbackupのみが残る
```

注意事項

${@%パターン}や${*%パターン}はシェルによって違う

特殊パラメータ$@や$*に対してこれらのパラメータ展開を使用した場合、その動作はbashとFreeBSDのshでは異なるため注意が必要です。bashでは、複数の位置パラメータそれぞれから指定のパターンに一致した文字列を取り除いてから、$@や$*の展開が行われます。一方FreeBSDのshでは、$@や$*の展開を行った文字列全体から、指定のパターンに一致した文字列が取り除かれます。

参照

ダブルクォート" "(p.209)　dirname(p.265)

${パラメータ:オフセット}と${パラメータ:オフセット:長さ}

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

オフセットや長さを指定してパラメータの値の文字列を切り出す

書式 ${パラメータ:オフセット}
${パラメータ:オフセット:長さ}

例 `echo "${var:2:5}"` ……シェル変数varの左側の2文字を除いて残りの5文字分を表示

基本事項

${パラメータ:オフセット}は、パラメータの値の文字列の左側からオフセット個の文字が取り除かれた文字列に展開されます。

${パラメータ:オフセット:長さ}は、パラメータの値の文字列の左側からオフセット個の文字が取り除かれた上で、最大で長さ分の文字列に展開されます。

解説

シェル変数や位置パラメータにセットされている文字列の一部分を、オフセットや長さを指定して取り出したい場合、${パラメータ:オフセット}や${パラメータ:オフセット:長さ}を使う方法があります。ただし、FreeBSDやSolarisのshでは使えないため、注意が必要です。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

オフセットや長さを指定した実行例

オフセットや長さを指定して、実際にこのパラメータ展開を実行している様子を**図A**に示します。たしかに所定の動作を行っていることがわかります。

図A オフセットや長さを指定した実行例

```
$ message='Hello World'       シェル変数messageに適当な文字列を代入
$ echo "${message:2}"         messageの頭の2文字分を取り除いて表示
llo World                     たしかに頭の2文字が取り除かれている
$ echo "${message:2:5}"       messageの頭の2文字分を取り除き、残りの5文字分を表示
llo W                         頭の2文字分を除き、(スペースを含めて)5文字分が表示されている
```

参照

ダブルクォート" "(p.209)

${パラメータ/パターン/置換文字列}と${パラメータ//パターン/置換文字列}

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

パターンを指定して パラメータの値の文字列を置換する

書式 ${パラメータ/パターン/置換文字列}
${パラメータ//パターン/置換文字列}

例
```
echo "${var//cat/dog}" ……シェル変数varの中のcatという文字列すべてをdogに置換して表示する
```

基本事項

${パラメータ/パターン/置換文字列}は、パラメータの値の文字列の左側から見て最初にパターンに一致した文字列の部分が置換文字列に置換されます。

${パラメータ//パターン/置換文字列}は、パラメータの値の文字列の中でパターンに一致する1カ所以上の文字列の部分がすべて置換文字列に置換されます。

いずれも、パターンには、パス名展開の特殊文字を使えます[注4]。

解説

シェル変数や位置パラメータにセットされている文字列の一部分を、パターンを指定して置換したい場合、${パラメータ/パターン/置換文字列}や${パラメータ//パターン/置換文字列}を使う方法があります。ただし、FreeBSDやSolarisのshでは使えないため、注意が必要です。冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、置換後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

パターンを指定して、実際にこのパラメータ展開を実行している様子を**図A**に示します。

図A パターンで置換する実行例

```
$ message='Hello World'          シェル変数messageに適当な文字列を代入
$ echo "${message/l/X}"          最初のlをXに置換して表示
HeXlo World                      たしかに、最も左にあるlのみがXに置換されている
$ echo "${message//l/X}"         すべてのlをXに置換して表示
HeXXo WorXd                      たしかに、すべてのlがXに置換されている
```

参照

ダブルクォート" "(p.209)

注4 パス名展開については10.2節を参照してください。

${!変数名@}または${!変数名*}

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

指定した文字列で始まる変数名を一覧表示する

書式 ${!変数名@}
${!変数名*}

例
```
echo ${!P*} ……Pで始まるシェル変数名をすべてリストする
```

基本事項

${!変数名@}**または**${!変数名*}は、現在セットされているシェル変数のうち、指定した変数名で始まるすべてのシェル変数名をスペース(シェル変数IFSの最初の文字)で区切ったものに展開されます。

解説

現在セットされているシェル変数のうち、**特定の文字**や**文字列**で始まるシェル変数名のみを表示したい場合に${!変数名@}または${!変数名*}が使えます。ただし、FreeBSDやSolarisのshでは使えないため、注意が必要です。

${!変数名@}の全体をダブルクォートで囲むと、特殊パラメータの"$@"の展開と同様に個々の文字列にダブルクォートが付いたものとして解釈されます。${!変数名*}をダブルクォートで囲んだ場合は、特殊パラメータの"$*"の展開と同様に、文字列全体にダブルクォートが付いたものと解釈されます。

なお、このパラメータ展開でパラメータとして指定できるのは**シェル変数のみ**であり、位置パラメータや特殊パラメータは指定できません。

シェル変数名のリストの実行例

「P」で始まるシェル変数名をすべて表示している例を**図A**に示します。**PAGER**、**PATH**その他のシェル変数がセットされていることがわかります。

図A シェル変数名のリストの実行例

```
$ echo ${!P*}                                  Pで始まるシェル変数名をすべてリストする
PAGER PATH PIPESTATUS PPID PS1 PS2 PS4 PWD     Pで始まるシェル変数名が表示される
```

間接参照 ${!パラメータ}

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

パラメータの値をパラメータ名とみなし、さらにその値を参照する

書式 ${! パラメータ }

例

```
echo "${!var}"  ……シェル変数varの値をパラメータ名とみなし、そのパラメータの値を表示する
```

基本事項

${! パラメータ } は、指定の パラメータ の値が新たにパラメータ名とみなされ、そのパラメータの値に展開されます。

解説

シェル変数や位置パラメータの中に別のシェル変数名などのパラメータ名をセットして、間接的に値を参照したい場合に${! パラメータ }が使えます。ただし、FreeBSDやSolarisのshでは使えないため、注意が必要です。

冒頭の例のように、このパラメータ展開の場合も、通常のパラメータの参照の場合と同じく、全体をダブルクォートで囲んで、展開後の値がさらに余分に解釈されてしまうのを防いだほうがよいでしょう。

パラメータの間接参照の実行例

パラメータを間接参照している例を**図A**に示します。図のように、シェル変数varがmessageに展開され、さらにその値の「hello」に展開されていることがわかります。

図A シェル変数名のリストの実行例

```
$ message=hello          シェル変数messageに適当な文字列を代入
$ var=message            シェル変数名messageを、シェル変数varに代入
$ echo "${!var}"         varをパラメータ展開する
hello                    たしかにシェル変数messageが参照されてhelloと表示される
```

Memo

●パラメータの間接参照は、evalを使って行ったほうが移植性が高まります。

参照

ダブルクォート " "(p.209)　　eval(p.98)

>第9章 クォートとコマンド置換