等価catコマンドによるコマンド置換

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

bashでは、図Dのように、$(< ファイル名)と記述することにより、$(cat ファイル名)と等価なコマンド置換を行うことができます。単にファイルの内容で置換したい場合、外部コマンドのcatを呼び出す必要がないため効率的ですが、当然、移植性がなくなるため、使用には注意が必要です。

図D 等価catコマンドによるコマンド置換

```
$ echo "$(</etc/resolv.conf)"          bashの文法で、catを使わずにコマンド置換
nameserver 192.168.1.1                resolv.confのファイル内容が表示される
domain localdomain
$ echo "$(cat </etc/resolv.conf)"      通常のcatコマンドを使う方法
nameserver 192.168.1.1                同じく、ファイル内容が表示される
domain localdomain
```

注意事項

バッククォートとの動作の違い

$( )で囲まれた文字列中とは異なり、バッククォート(` `)で囲まれた文字列中では、\$、\`、\\の場合のみ\が特別に扱われます。したがって、次の例のような場合、$( )と` `とで動作が異なります。とくに、sedやgrepなどの正規表現中で\を多用する場合に注意が必要です。

```
$ basename $(basename '\\')  ……$( )の中でbasename '\\'を実行すると
\\  ……\\と表示される
$ basename `basename '\\'`  ……代わりに` `を用いると
\  ……\1つになってしまう
```

Memo

- Solarisのshなどの従来のシェルへの移植性を考えると、$( )よりも` `を使用するべきでしょう。

参照

expr(p.261)　単語分割(p.237)　ダブルクォート" "(p.209)　basename(p.263)

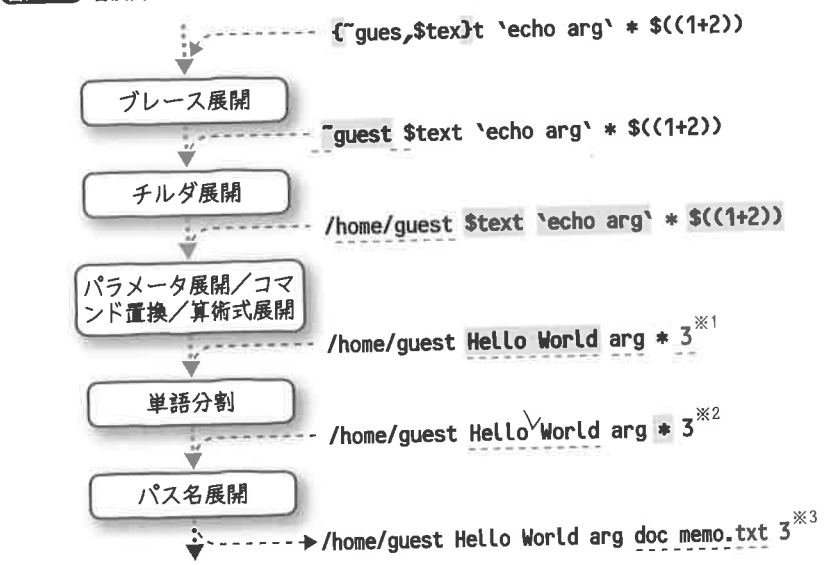
>第10章 各種展開

シェルにおける各種展開

シェル上でコマンドを実行する際に、その「コマンド名」や「引数」を直接入力または記述するばかりではなく、**パラメータ展開**によってシェル変数の値を用いたり、**コマンド置換**によってコマンドの標準出力を取り込んだり、あるいは*や?などの記号で複数のファイルを同時に指定(パス名展開[注1])したりといったことができます。このように、コマンド名やその引数を生成する方法をまとめて**展開**といいます。

シェルには、パラメータ展開(8章を参照)やコマンド置換(9.3節を参照)のほかにも、**パス名展開**、**ブレース展開**、**算術式展開**、**チルダ展開**、**単語分割**といった各種展開があり、これらの展開がすべて終了した段階で、実行するべき「コマンド」やその「引数」が決定されます。展開は以下の**図A**のような順序で行われます。本章では、これらの各種展開について解説します。

図A 各展開の「展開の順序」



※1 シェル変数textの中にはHello Worldという文字列が入っています.

※2 全体で1単語とみなされていたHello Worldが，この段階でHelloとWorldの2単語とみなされます.

※3 *が，カレントディレクトリに存在しているdocとmemo.txtというファイル名に展開されています.

注1 パス名展開は、カレントディレクトリなどのディレクトリ上に存在する複数のファイルのうち、一定規則のファイル名のファイルを一括してコマンドの引数に付けたい場合に使用します。*、?、[a-z]で指定できます。詳しくは10.2節を参照してください。

パス名展開 *

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

0文字以上の任意の文字列にマッチさせる

書式 [文字列]*[文字列]

例
```
cp *.txt /some/dir ……カレントディレクトリの、拡張子が.txtのファイルをすべて/some/dirにコピーする
```

基本事項

*は、**0文字以上の任意の文字列**にマッチし、文字列を含めて実在のファイルと照合してマッチした**パス名**に、アルファベット順にソートした上で**展開**されます。マッチするパス名が存在しない場合は、*は展開されずにそのまま残ります。*は、パス名の区切りの「/」や、先頭または/の直後にある「.」にはマッチしません。ただし、case文のパターンとして使用される場合は*はすべての文字にマッチします。

解説

カレントディレクトリなどのディレクトリ上に存在する複数のファイルのうち、一定の規則のファイル名のファイルを一括してコマンドの引数に付けたい場合に**パス名展開**を使用します。*は、0文字以上の任意の文字列にマッチするため、たとえば「*.txt」は拡張子が**.txt**のファイルすべてに展開されます。また、単独の*はカレントディレクトリのすべてのファイル(ただし.で始まるファイルを除く)に展開されます。

なお、パス名展開はあくまでシェルによって行われるものであり、パス名展開を引数として起動されるコマンド(冒頭の例ではcpコマンド)には、展開済みの引数が渡されます。

いろいろなファイル名に対する展開例

いろいろなファイル名に対する展開例を**図A**に示します。ここでは、**図A❶**でls -aを実行してカレントディレクトリに存在するファイルを確認した上で、*を使ったいろいろなパス名展開をechoコマンドの引数に付けて実行し、展開されるファイル名を確認しています。**図A❷**のように、「.」で始まるファイルは「.*」と指定すれば展開できます。また、**図A❸**のように、*を2つ使用することもできます。

図A いろいろなファイル名に対する展開例

```
$ ls -a                                          ❶カレントディレクトリのファイルをすべて表示
.               .bash_profile  cat.jpg             file21.txt      memo.txt
                                                                   このようなファイルがある
..              .bashrc        dog-02.jpg          file22.txt
.bash_history   cat-01.jpg     dog.jpg             file23.txt
$ echo *                                         単独の*を展開すると

cat-01.jpg cat.jpg dog-02.jpg dog.jpg file21.txt file22.txt file23.txt memo.txt
                                                 .で始まるファイル以外のすべてのファイルに展開される
$ echo .*                                        ❷.*として展開すると
. .. .bash_history .bash_profile .bashrc         .で始まるファイル(.や..も含む)に展開される
$ echo *.txt                                     *.txtを展開すると
file21.txt file22.txt file23.txt memo.txt        すべての拡張子.txtのファイルに展開される
$ echo file*.txt                                 file*.txtと、さらに限定すると
file21.txt file22.txt file23.txt                 fileという名前で始まる拡張子.txtのファイルに展開される
$ echo *.jpg                                     *.jpgを展開すると
cat-01.jpg cat.jpg dog-02.jpg dog.jpg            すべての拡張子.jpgのファイルに展開される
$ echo *-*.jpg                                   ❸2つの*を使って*-*.jpgとすると
cat-01.jpg dog-02.jpg                            ファイル名の中に-を含み、拡張子が.jpgのファイルに展開される
```

スラッシュを含む展開例

パス名展開にスラッシュ(/)を含めて、カレントディレクトリ以外のファイルにマッチさせることもできます。**図B**のように、カレントディレクトリの1つ下のサブディレクトリである「sub1」と「sub2」の下に、拡張子**.txt**と**.jpg**のファイルが存在する場合、「*/*.txt」というパス名展開を用いると、「sub1」と「sub2」の下の拡張子**.txt**のファイルに展開されます。

図B スラッシュを含む展開例

```
$ ls -RF                                         カレントディレクトリ以下をサブディレクトリを含めて表示
sub1/  sub2/                                     2つのサブディレクトリがある

sub1:                                            sub1の下には
cat.jpg    doc01.txt  doc02.txt                  このようなファイルがある

sub2:                                            sub2の下には
doc03.txt  doc04.txt  doc05.txt  dog.jpg         このようなファイルがある
$ echo */*.txt                                   */*.txtというパス名展開を使うと
sub1/doc01.txt sub1/doc02.txt sub2/doc03.txt sub2/doc04.txt sub2/doc05.txt
                                                 このように展開される
```

Memo

- パス名展開は、grepやsedコマンドなどで使用される正規表現とは異なります。パス名展開の*は、正規表現での「.*」に相当します。
- パス名展開の*は任意の文字にマッチすることから、?とともにワイルドカード(*wild card*)とも呼ばれます。
- set -fコマンドを実行すると、パス名展開を抑制することができます。

参照

case文(p.49)

パス名展開 ?

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

任意の1文字にマッチさせる

書式 [文字列]?[文字列]

例

```
cp ???.txt /some/dir ..... カレントディレクトリにある、3文字名+.txtのファイルをすべて/some/dirにコピーする
```

基本事項

?は任意の1文字にマッチし、文字列を含めて実在のファイルと照合してマッチした**パス名**に、アルファベット順にソートした上で**展開**されます。マッチするパス名が存在しない場合は、?は展開されずにそのまま残ります。?は、パス名の区切りの「/」や、先頭または/の直後にある「.」にはマッチしません。ただし、case文のパターンとして使用される場合は?はすべての文字にマッチします。

解説

カレントディレクトリなどのディレクトリ上に存在する複数のファイルのうち、一定の規則のファイル名のファイルを一括してコマンドの引数に付けたい場合に**パス名展開**を使用します。?は、任意の1文字にマッチするため、たとえば「???.txt」は、拡張子を除いた部分のファイル名が3文字で、かつ拡張子が**.txt**のファイルすべてに展開されます。

なお、パス名展開はあくまでシェルによって行われるものであり、パス名展開を引数として起動されるコマンド(冒頭の例ではcpコマンド)には、展開済みの引数が渡されます。

いろいろなファイル名に対する展開例

いろいろなファイル名に対する展開例を**図A**に示します。ここでは、最初にlsコマンドを実行してカレントディレクトリに存在するファイルを確認した上で、?を使ったいろいろなパス名展開をechoコマンドの引数に付けて実行し、展開されるファイル名を確認しています。いずれも?が任意の1文字にマッチし、該当するファイル名が表示されていることがわかります。

図A いろいろなファイル名に対する展開例

```
$ ls -a                                          ❶カレントディレクトリのファイルをすべて表示
.               .bash_profile  cat.jpg             file21.txt      memo.txt
                                                                  このようなファイルがある
..              .bashrc        dog-02.jpg          file22.txt
.bash_history   cat-01.jpg     dog.jpg             file23.txt
$ echo *                                         単独の*を展開すると

cat-01.jpg cat.jpg dog-02.jpg dog.jpg file21.txt file22.txt file23.txt memo.txt
                                                 .で始まるファイル以外のすべてのファイルに展開される
$ echo .*                                        ❷.*として展開すると
. .. .bash_history .bash_profile .bashrc         .で始まるファイル(.や..も含む)に展開される
$ echo *.txt                                     *.txtを展開すると
file21.txt file22.txt file23.txt memo.txt        すべての拡張子.txtのファイルに展開される
$ echo file*.txt                                 file*.txtと、さらに限定すると
file21.txt file22.txt file23.txt                 fileという名前で始まる拡張子.txtのファイルに展開される
$ echo *.jpg                                     *.jpgを展開すると
cat-01.jpg cat.jpg dog-02.jpg dog.jpg            すべての拡張子.jpgのファイルに展開される
$ echo *-*.jpg                                   ❸2つの*を使って*-*.jpgとすると
cat-01.jpg dog-02.jpg                            ファイル名の中に-を含み、拡張子が.jpgのファイルに展開される
```

スラッシュを含む展開例

パス名展開にスラッシュ(/)を含めて、カレントディレクトリ以外のファイルにマッチさせることもできます。**図B**のように、カレントディレクトリの1つ下のサブディレクトリである「sub1」と「sub2」の下に、拡張子**.txt**と**.jpg**のファイルが存在する場合、「*/*.txt」というパス名展開を用いると、「sub1」と「sub2」の下の拡張子**.txt**のファイルに展開されます。

図B スラッシュを含む展開例

```
$ ls -RF                                         カレントディレクトリ以下をサブディレクトリを含めて表示
sub1/  sub2/                                     2つのサブディレクトリがある

sub1:                                            sub1の下には
cat.jpg    doc01.txt  doc02.txt                  このようなファイルがある

sub2:                                            sub2の下には
doc03.txt  doc04.txt  doc05.txt  dog.jpg         このようなファイルがある
$ echo */*.txt                                   */*.txtというパス名展開を使うと
sub1/doc01.txt sub1/doc02.txt sub2/doc03.txt sub2/doc04.txt sub2/doc05.txt
                                                 このように展開される
```

Memo

- パス名展開は、grepやsedコマンドなどで使用される正規表現とは異なります。パス名展開の*は、正規表現での「.*」に相当します。
- パス名展開の*は任意の文字にマッチすることから、?とともにワイルドカード(*wild card*)とも呼ばれます。
- set -fコマンドを実行すると、パス名展開を抑制することができます。

参照

case文(p.49)

パス名展開 ?

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

任意の1文字にマッチさせる

書式 [文字列]?[文字列]

例

```
cp ???.txt /some/dir ..... カレントディレクトリにある、3文字名+.txtのファイルをすべて/some/dirにコピーする
```

基本事項

?は任意の1文字にマッチし、文字列を含めて実在のファイルと照合してマッチした**パス名**に、アルファベット順にソートした上で**展開**されます。マッチするパス名が存在しない場合は、?は展開されずにそのまま残ります。?は、パス名の区切りの「/」や、先頭または/の直後にある「.」にはマッチしません。ただし、case文のパターンとして使用される場合は?はすべての文字にマッチします。

解説

カレントディレクトリなどのディレクトリ上に存在する複数のファイルのうち、一定の規則のファイル名のファイルを一括してコマンドの引数に付けたい場合に**パス名展開**を使用します。**?**は、任意の1文字にマッチするため、たとえば「???.txt」は、拡張子を除いた部分のファイル名が3文字で、かつ拡張子が**.txt**のファイルすべてに展開されます。

なお、パス名展開はあくまでシェルによって行われるものであり、パス名展開を引数として起動されるコマンド(冒頭の例ではcpコマンド)には、展開済みの引数が渡されます。

いろいろなファイル名に対する展開例

いろいろなファイル名に対する展開例を**図A**に示します。ここでは、最初にlsコマンドを実行してカレントディレクトリに存在するファイルを確認した上で、?を使ったいろいろなパス名展開をechoコマンドの引数に付けて実行し、展開されるファイル名を確認しています。いずれも?が任意の1文字にマッチし、該当するファイル名が表示されていることがわかります。

図A いろいろなファイル名に対する展開例

```
$ ls                                    カレントディレクトリのファイルを表示
file1-a.txt  file1.txt    file11.txt   file2.txt    memo.txt
                                                    このようなファイルがある
file1-b.txt  file10.txt   file2-a.txt  file3.txt
$ echo file?.txt                        file?.txtを展開すると
file1.txt file2.txt file3.txt           file+1文字+.txtのファイルに展開される
$ echo file??.txt                       file??.txtを展開すると
file10.txt file11.txt                   file+2文字+.txtのファイルに展開される
$ echo file?-?.txt                      file?-?.txtを展開すると
file1-a.txt file1-b.txt file2-a.txt     file+1文字+-+1文字+.txtのファイルに展開される
```

スラッシュを含む展開例

パス名展開にスラッシュ(/)を含めて、カレントディレクトリ以外のファイルにマッチさせることもできます。**図B**のように、カレントディレクトリの1つ下にサブディレクトリがいくつかある場合、「sub?/doc?.txt」というパス名展開は、「sub＋1文字」サブディレクトリの下(/)の「doc＋1文字＋.txt」というファイルに展開されます。

図B スラッシュを含む展開例

```
$ ls -RF                                カレントディレクトリ以下をサブディレクトリを含めて表示
sub1/    sub2/    subdir/               3つのサブディレクトリがある

sub1:                                   sub1の下には
doc1.txt   doc10.txt  doc2.txt   memo.txt                    このようなファイルがある

sub2:                                   sub2の下には
doc15.txt  doc16.txt  doc5.txt   doc6.txt   memo.txt        このようなファイルがある

subdir:                                                     subdirの下には
doc1.txt   file2.txt  memo.txt                              このようなファイルがある
$ echo sub?/doc?.txt                    ここでsub?/doc?.txtというパス名展開を使うと
sub1/doc1.txt sub1/doc2.txt sub2/doc5.txt sub2/doc6.txt     このように展開される
```

Memo

- パス名展開は、grepやsedコマンドなどで使用される正規表現とは異なります。パス名展開の?は、正規表現での.に相当します。
- パス名展開の?は任意の文字にマッチすることから、*とともにワイルドカードとも呼ばれます。
- set -fコマンドを実行すると、パス名展開を抑制することができます。

参照

case文(p.49)

パス名展開 [a-z]

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

指定した条件の1文字にマッチさせる

書式

❶[文字列][文字1[文字2...]][文字列]
❷[文字列][文字1-文字2][文字列]
❸[文字列][!文字1[文字2...]][文字列]
❹[文字列][!文字1-文字2][文字列]

例

```
cp file[135].txt /some/dir ..... file1.txt file3.txt file5.txtを/some/dirにコピーする
cp file[a-z].txt /some/dir ..... file+英小文字1文字+.txtのファイルを/some/dirにコピーする
cp file[!135].txt /some/dir .... file+1,3,5以外の1文字+.txtのファイルを/some/dirにコピーする
cp file[!a-z].txt /some/dir .... file+英小文字以外の1文字+.txtのファイルを/some/dirにコピーする
```

基本事項

❶は、[ ]の中に書かれている任意の1文字(文字1 文字2…)にマッチします。
❷は、キャラクタコードが文字1から文字2までの範囲の任意の1文字にマッチします。
❸は、[ ]の中に書かれていない任意の1文字(文字1 文字2…)にマッチします。
❹は、キャラクタコードが文字1から文字2までの範囲以外の任意の1文字にマッチします。

いずれも、文字列を含めて実在のファイルと照合してマッチした**パス名**に、アルファベット順にソートした上で**展開**されます。マッチするパス名が存在しない場合は、パス名展開の文字列は展開されずにそのまま残ります。パス名展開の文字列は、パス名の区切りの「/」や、先頭または/の直後にある「.」にはマッチしません。ただし、case文のパターンとして使用される場合はすべての文字にマッチします。

解説

カレントディレクトリなどのディレクトリ上に存在する複数のファイルのうち、一定の規則のファイル名のファイルを一括してコマンドの引数に付けたい場合に**パス名展開**を使用します。たとえば「file[246].txt」は、「file2.txt」「file4.txt」「file6.txt」のうち存在するファイル名に展開されます。「file[1-9].txt」のように範囲を指定することも可能で、この場合は「file1.txt」「file2.txt」…「file9.txt」のうち、存在するファイル名に展開されます。

「[!246]」のように!を付けると**補集合**の意味になりこの場合は「2、4、6」以外の任意の1文字にマッチします。同様に、「[!1-9]」は1〜9までの数字以外の任意の1文字にマッチします。

bashおよびFreeBSDのshでは、補集合を表す!の記号の代わりに^を使って「[^246]」や「[^1-9]」と記述することもできます。

なお、パス名展開はあくまでシェルによって行われるものであり、パス名展開を引数として起動されるコマンド(冒頭の例ではcpコマンド)には、展開済みの引数が渡されます。

いろいろなファイル名に対する展開例

いろいろなファイル名に対する展開例を**図A**に示します。ここでは、最初にlsコマンドを実行してカレントディレクトリに存在するファイルを確認した上で、いろいろなパス名展開をechoコマンドの引数に付けて実行し、展開されるファイル名を確認しています。いずれも、パス名展開の部分が意図した1文字にマッチし、該当するファイル名が表示されていることがわかります。

スラッシュを含む展開例

パス名展開にスラッシュ(/)を含めて、カレントディレクトリ以外のファイルにマッチさせることもできます。**図B**のように、カレントディレクトリの1つ下にサブディレクトリがいくつかある場合、「sub[1-9]/doc[a-z].txt」というパス名展開は、「sub＋数字1文字」のサブディレクトリの下(/)の「doc＋英小文字1文字＋.txt」というファイルに展開されます。

図A　いろいろなファイル名に対する展開例

```
$ ls                                    カレントディレクトリのファイルを表示
dog.jpg    file11.txt  file13.txt  file3.txt  fileb.txt  memo.txt
                                                          このようなファイルがある
file1.txt  file12.txt  file2.txt   filea.txt  filec.txt
$ echo file[a-z].txt                    file[a-z].txtを展開すると
filea.txt fileb.txt filec.txt           file+英小文字1文字+.txtのファイルに展開される
$ echo file[!13].txt                    file[!13].txtを展開すると
file2.txt filea.txt fileb.txt filec.txt
                                        file+1、3以外の1文字+.txtのファイルに展開される
$ echo file[0-9][0-9].txt               file[0-9][0-9].txtを展開すると
file11.txt file12.txt file13.txt        file+2桁の数字+.txtのファイルに展開される
```

図B　スラッシュを含む展開例

```
$ ls -RF                                カレントディレクトリ以下をサブディレクトリを含めて表示
sub1/  sub2/  suba/                     3つのサブディレクトリがある

sub1:                                   sub1の下には
doc1.txt  doc2.txt  doca.txt  memo.txt  このようなファイルがある

sub2:                                   sub2の下には
doc3.txt  doc4.txt  docb.txt  dog1.jpg  このようなファイルがある

suba:                                   subaの下には
doc5.txt  doc6.txt  docc.txt            このようなファイルがある
$ echo sub[1-9]/doc[a-z].txt            sub[1-9]/doc[a-z].txtというパス名展開を使うと
sub1/doca.txt sub2/docb.txt             このように展開される
```

文字クラスを使ったパス名展開

Warning

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

この方法には制限があります。

bashでは、パス名展開の[ ]の中に指定する文字として、[:alnum:]のような形式の文字クラスが使えます。実際にはパス名展開の[ ]の中に入れて使うため、**表A**のように角括弧は2重になります。たとえば[[:digit:]]は[0-9]に相当し、0から9までの数字1文字に展開されます。

文字クラスを使ったパス名展開では、環境変数LANG(またはLC_TYPEまたはLC_ALL)によって設定される言語環境によって、展開される文字の範囲が変わります(次ページの表Aの注釈)。

Memo

- パス名展開は、grepやsedコマンドなどで使用される正規表現とは異なりますが、パス名展開の[a-z][!a-z]は正規表現の[a-z][^a-z]に相当します。
- 補集合を表す記号が!なのは、古いshにおいてパイプを表す記号が|ではなく^だったため、これとの衝突を避けるために!を使うようにしたことによります。
- set -fコマンドを実行すると、パス名展開を抑制することができます。

表A 文字クラスを使ったパス名展開

文字クラスを使ったパス名展開	意味
[[:alnum:]]	英数字※1
[[:alpha:]]	英文字※1
[[:ascii:]]	7ビットASCII(半角)文字(コントロールコードも含む)
[[:blank:]]	スペース、タブ※2
[[:cntrl:]]	コントロールコード
[[:digit:]]	数字([0-9]に相当)
[[:graph:]]	表示可能文字(スペースを除く)※3
[[:lower:]]	英小文字※4
[[:print:]]	印字可能文字(スペースを含む)※5
[[:punct:]]	記号※6
[[:space:]]	スペース、タブ、改行、改ページ文字等※7
[[:upper:]]	英大文字※4
[[:word:]]	英数字とアンダースコア※8
[[:xdigit:]]	16進数([0-9A-Fa-f]に相当)

※1 日本語環境の場合は全角英数字、平仮名、片仮名、漢字等も含まれる。[[:alpha:]]であっても全角の数字は含まれるため注意。
※2 日本語環境の場合は全角スペースも含む。
※3 日本語環境の場合は全角スペースを除く全角文字、全角記号を含む。
※4 日本語環境の場合は全角の英文字/ギリシャ文字/ロシア文字の大文字または小文字を含む。
※5 日本語環境の場合は全角スペース、全角文字、全角記号を含む。
※6 日本語環境の場合は全角記号を含む。
※7 日本語環境の場合は全角スペースを含む。
※8 日本語環境の場合は全角英数字、平仮名、片仮名、漢字等を含むが、全角アンダースコアは除く。

参照

case文(p.49)

ブレース展開 {a,b,c}

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

複数の文字列の組み合わせから文字列を生成する

書式 [文字列]{文字列,[文字列]...}[文字列]

例
```
cp /some/dir/{cat,dog}.jpg .  ……/some/dirの下にあるcat.jpgとdog.jpgをカレントディレクトリにコピー
```

基本事項

ブレース展開では、{ }の中に書かれた1個以上のカンマ(,)で区切られた複数の文字列を、左から順番に使用して文字列を生成します。パス名展開とは異なり、実在のパス名との照合は行われません注2。

解説

ファイル名に規則性のない複数のファイルを一括で指定したい場合、**ブレース展開**が便利です。ブレース展開はパス名展開に似ていますが、パス名展開とは違って展開する文字列を直接指定するため、*や?などではマッチできないパス名でも利用できます。また、展開される文字列のファイルが実在する必要はなく、ファイル名以外の文字列の展開にも使用できます。

冒頭の例では「/some/dir/{cat,dog}.jpg」が「/some/dir/cat.jpg」「/some/dir/dog.jpg」という2つのパス名に展開されます。このように長いパス名で複数のファイルを指定する場合、ブレース展開を使えばキー入力を節約することができます。ただし、ブレース展開はFreeBSDやSolarisのshでは使えないため、シェルスクリプト中での使用は控えたほうがよいでしょう。

なお、ブレース展開はあくまでシェルによって行われるものであり、ブレース展開を引数として起動されるコマンド(冒頭の例ではcpコマンド)には、展開済みの引数が渡されます。

空文字列を含む展開例

ブレース展開では、展開される文字列が空文字列であってもかまいません。**図A**は、「空文字列」と「/usr」をブレース展開し、その右側に「/bin」を付けた例です。「/bin」と「/usr/bin」に展開されていることがわかります。

図A 空文字列を含む展開例

```
$ echo {,/usr}/bin                    空文字列と/usrのブレース展開
/bin /usr/bin                         /binと/usr/binに展開される
```

注2 パス名展開については10.2節を参照してください。

連番のブレース展開

bash 3以降では、{1..9}の形式の連番のブレース展開が使えます。{1..9}は{1,2,3,4,5,6,7,8,9}と同じで、1から9までの数字に展開されます。{1..100}のように2桁以上の数字を指定することも可能で、この場合1から100までの100個の数字に展開されます。{-9..15}(マイナス9からプラス15まで)のように負の数を指定することも可能です。数字の代わりに文字(1文字)を指定することもできます。{a..z}はaからzまでの英小文字に、{A..Z}はAからZまでの英大文字に展開されます。{100..1}や{z..a}のように順番を逆にすると、逆順に展開されます。

ブレース展開のネスティング

ブレース展開をネスティングすることもできます。ネスティングされたブレース展開も、それぞれ順番に展開されます。**図B**では、内部的に「/usr/{bin,sbin,{,share/}lib}」が「/usr/bin」「/usr/sbin」「/usr/{,share/}lib」と展開されたあと、「/usr/{,share/}lib」が「/usr/lib」「/usr/share/lib」と展開され、図Bのとおりの結果になっていると考えられます。

複数のブレース展開の組み合わせ

ネスティングではなく、複数のブレース展開を組み合わせることもできます。**図C**のように実行すると、4×3(=12)通りのすべての組み合わせの文字列が生成され、echoによって表示されます。

図B ブレース展開をネスティングした例

```
$ echo /usr/{bin,sbin,{,share/}lib}          ブレース展開のネスティング
/usr/bin /usr/sbin /usr/lib /usr/share/lib   期待通り展開される
```

図C 複数のブレース展開の組み合わせ

```
$ echo {a,b,c,d}{1,2,3}                      複数のブレース展開の組み合わせ
a1 a2 a3 b1 b2 b3 c1 c2 c3 d1 d2 d3          12通りの文字列が表示される
```

注意事項

case文のパターンには使えない

ブレース展開はパス名展開ではないため、case文のパターンには使えません。case文のパターンで複数の文字列にマッチさせたい場合は、case文でOR条件を表す|記号を使います。

○正しい例

```
case $var in
  cat|dog)  echo '猫または犬です';; ……catまたはdogのOR条件を|記号で表す
esac
```

×誤った例

```
case $var in
  {cat,dog})  echo '猫または犬です';; ……ここでブレース展開を使うのは誤り
esac
```


Memo

- set +Bコマンドを実行すると、ブレース展開を抑制することができます。
- ブレース展開はcsh系由来の文法です。

算術式展開 $(( ))

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

算術式を評価し その演算結果の数値に展開する

FreeBSDのshでは、一部使用できない演算子があったり、シェル変数の間接参照ができないなど、動作が異なる点がある。

書式 $((算術式))

例 `echo $((2 + 3))` ……2 + 3の足し算を行い、答を表示する

基本事項

算術式展開では算術式の評価を行い、その演算結果の**数値**に展開します。演算の規則は算術式の評価と同じです。

解説

算術式展開は算術式の評価に似ていますが、算術式を評価したあと、その真偽値を終了ステータスとして返すのではなく、演算結果の数値そのものに展開するという展開方法です。

算術式の中では、算術式の評価の項で述べられている演算子がすべて使え、**加減算**などの通常の演算だけでなく、**代入**や**インクリメント**などの、演算によってシェル変数の値が変化するような演算も行えます。

シェル変数の参照では、変数名の頭に$記号を付ける必要はありません。また、シェル変数の値が数値以外の場合は、その値の文字列をシェル変数名として該当のシェル変数が間接参照されます。この間接参照は、数値が得られるまで何度でも繰り返されます。

算術式展開を使わない方法

移植性のため、算術式展開を使わずに数値計算を行う場合は、**図A**❀のようにexprコマンドを使います。

図A 算術式展開とexprコマンドの実行例

```
$ a=2                          シェル変数aに2を代入
$ echo $((a + 3))              算術式展開を使った足し算
5                              答が表示される
$ expr "$a" + 3                ❀exprコマンドを使った足し算
5                              答が表示される
```

Memo

❀bashでは、$((算術式))の代わりに$[算術式]と書くこともできます。

参照

算術式の評価(( ))(p.80)　　expr(p.261)

チルダ展開 ~

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

ユーザのホームディレクトリに展開する

書式 ~[ユーザ名]

例 `cat ~/.bash_profile` ……自分のホームディレクトリの下の.bash_profileを表示

基本事項

~[ユーザ名]は、そのユーザの**ホームディレクトリのパス名**に展開されます。[ユーザ名]を省略した単独の~は、自分自身(そのシェルを実行中のユーザ)のホームディレクトリのパス名に展開されます。

解説

ホームディレクトリのパス名を使用したい場合、~を使えば素早く入力できます。たとえば冒頭の例では、自分のホームディレクトリの下の**.bash_profile**を指定したことになります。ただし、チルダ展開はSolarisのshでは使えないため、シェルスクリプト中で自分のホームディレクトリを参照する場合は、~ではなく「"$HOME"」を使用したほうがよいでしょう。

なお、チルダ展開はあくまでシェルによって行われるものであり、チルダ展開を引数として起動されるコマンド(冒頭の例ではcatコマンド)には、展開済みの引数が渡されます。

チルダ展開の例

チルダ展開の例を**図A**に示します。このように、単独の「~」は自分のホームディレクトリに、「~root」はrootのホームディレクトリである/rootに展開されていることがわかります。

図A チルダ展開の例

```
$ echo ~                       ~を展開すると
/home/guest                    自分のホームディレクトリである/home/guestになる
$ echo ~root                   ~rootを展開すると
/root                          rootのホームディレクトリである/rootになる
```

Memo

❀チルダ展開はcsh系由来の文法です。

参照

HOME(p.184)

プロセス置換 <( ) / >( )

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

FIFOに接続した別プロセスを起動し、そのFIFO名に置換する

bashであっても、#!/bin/shとして起動された場合は動作しない

書式 <(リスト)
>(リスト)

例
```
diff <(sort file1) <(sort file2) ……file1とfile2をソートしてから
                                    その差分を表示
```

基本事項

<(リスト)は、FIFO(名前つきパイプ)を作成し、そのFIFOを標準出力に接続した状態でリストを別プロセスとして起動するとともに、そのFIFOのパス名に置換します[注3]。

>(リスト)は、FIFOを作成し、そのFIFOを標準入力に接続した状態でリストを別プロセスとして起動するとともに、そのFIFOのパス名に置換します[注4]。

解説

プロセス置換を使うと、標準入出力ではなく、ファイルへの入出力に対応したコマンドに対して、ファイルの代わりにFIFOを指定し、そのFIFOを介して別のプロセスからデータを読む、または書き込むといった動作ができます。同様の動作は、mkfifoコマンドを使って自分でFIFOを適当なファイル名で作成し、そのFIFOに対して読み書きを行う2つのプロセスを起動することでも可能ですが、プロセス置換を使えば、FIFOの作成をシェル側で自動的に行うことができます。

ただし、プロセス置換はFreeBSDやSolarisのshで使えないだけでなく、bashであってもset -o posixが設定された状態では動作しません。/bin/shがbashへのシンボリックリンクになっているOSで、1行目に#!/bin/shと書かれたシェルスクリプトでは、bashがコマンド名shで起動されるためset -o posixが設定され、プロセス置換は動作しません。#!/bin/shのシェルスクリプトであえてプロセス置換を使用するには、シェルスクリプト内でset +o posixを実行して、posixオプションを無効にするようにします(set -oが有効、set +oが無効の意味になるので注意)。

注3 実際には多くの場合、FIFOではなく、/dev/fd/n(nはファイル記述子番号)を使って通常のパイプで実装されています。

注4 プロセス置換は、set -o posixが設定された状態では動作しません。bashがコマンド名shで起動されている場合はset -o posixが設定されています。

プロセス置換<(リスト)の例

図A❶は、file1とfile2という2つのファイルをそれぞれソートした後、その結果をdiffコマンドを使って比較するという例です。<(sort file1)の部分では、自動的にFIFOが作成され、標準出力をFIFOに接続した状態でsort file1コマンドが起動されます。同時に、そのFIFOのパス名の文字列がdiffコマンドの第1引数として与えられます。<(sort file2)の部分も同様で、こちらのFIFOのパス名はdiffコマンドの第2引数になります。以上の結果、diffは2つのFIFOを読んでその差分を出力することになるため、2つのファイルをそれぞれソートしてから比較することができます。

なお、仮にfile2のソートは必要なく、file1のみをソートしてから比較したい場合は図A❷のように通常のパイプを使って実行することができます。diffコマンドの引数の「-」は標準入力を表し、パイプからの入力を読むことができます。

さらに、/dev/fd/3(3はファイル記述子番号)のようなデバイスが使える場合は、図A❸のように、2個のパイプとファイル記述子のリダイレクトを巧妙に使って、2つのファイルをそれぞれソートしてから比較することができます。ここでは、ソート後のfile1の内容が読めるパイプをexecでファイル記述子3に複製し、それをdiffコマンドで/dev/fd/3経由で読み、file2の方は普通にパイプに通して標準入力から読んでいます。

プロセス置換>(リスト)の例

図B❶はcpコマンドのコピー先のファイルとしてFIFOを指定する例です。cpコマンドは標準出力への出力はできませんが、出力先にFIFOを使うことにより、cpの出力をあたかもパイプに接続したような動作が可能です。ここでは>(cat -n)というプロセス置換が用いられているため、FIFOを標準入力に接続した状態でcat -nコマンドが起動され、ファイルの内容に行番号を付けて表示されます。ただし、この例の場合は図B❷のように直接cat -nコマンドを実行したのと同じになります。

なお、FIFOとして具体的にどのようなパス名が使用されているかは、図B❸❹のように、ヌルコマンド「:」を使ったプロセス置換を行い、その置換結果の文字列をechoコマンドで表示してみれば確認できます。具体的なパス名はシェルやOSのバージョンによって異なります。

/dev/fd/n(nはファイル記述子番号)や/dev/stdoutが使える環境では、図B❺❻のように、プロセス置換を使わずに実行することもできます。

図A プロセス置換<(リスト)の例

```
$ set +o posix                                念のためposixオプションを無効にする
$ diff <(sort file1) <(sort file2)            file1とfile2をソートしてから比較❶
…省略…                                       比較結果（差分）が表示される
$ sort file1 | diff - file2                   file1のみソートする場合は通常のパイプでOK❷
…省略…                                       比較結果（差分）が表示される
$ sort file1 | (exec 3<&0; sort file2 | diff /dev/fd/3 -)
                                              /dev/fd/3等が使える場合は通常のパイプでOK❸
…省略…                                       比較結果（差分）が表示される
```

図B プロセス置換 >(リスト)の例

```
$ set +o posix          念のためposixオプションを無効にする
$ cp file >(cat -n)     fileをFIFOにコピーし、FIFOに対してcat -nを実行❶
…省略…                 行番号つきのfileの内容が表示される
$ cat -n file           fileに対して直接cat -nを実行したのと同じ❷
…省略…                 同じく行番号つきのfileの内容が表示される
$ echo >(:)             ヌルコマンド「:」を使ってプロセス置換を実行し、echoで表示❸
/dev/fd/63              このようなパス名が使用されている
$ echo <(:)             ヌルコマンド「:」を使ってプロセス置換を実行し、echoで表示❹
/dev/fd/63              このようなパス名が使用されている
$ cp file /dev/fd/3 3>&1 | cat -n    自分で/dev/fd/3等を用いてもよい❺
…省略…                 同じく行番号つきのfileの内容が表示される
$ cp file /dev/stdout | cat -n       /dev/stdoutを用いてもよい❻
…省略…                 同じく行番号つきのfileの内容が表示される
```


単語分割(IFS)

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

パラメータ展開などの結果をコマンド名や引数に分割する

解説

パラメータ展開、コマンド置換の結果は、シェル変数**IFS**にセットされている文字を区切り文字として**単語分割**が行われ、(その後のパス名展開を経て)最終的な**コマンド名**と**引数**が決定されます。**IFS**にはデフォルトで**スペース、タブ、改行**の3文字がセットされています。パラメータ展開やコマンド置換の結果の文字列をそのまま使用したい場合は、"$var"や"`pwd`"のように全体をダブルクォート(" ")で囲んで単語分割を避けるようにします。

IFSの値は変更することもでき、**IFS**の値を一時的に変更してからsetやreadを実行すれば、任意の文字を区切り文字として位置パラメータやシェル変数にセットすることができます。

IFSを変更する例

IFSを変更する例を**図A**に示します。シェル変数**PATH**に設定されている、:で区切られたディレクトリ名を分割するために、**IFS**の値を一時的に:に変更しています。なお、**IFS**の値の変更はシェルの動作に影響を与えるため注意して行う必要があり、ここでは**IFS_SAVE**という別のシェル変数に値を退避させ、目的の単語分割が終わったらすぐに**IFS**の値を元に戻しています。

PATHの単語分割の結果は、setコマンドによって位置パラメータに取り込んでいます。ここでは通常とは異なり、単語分割を行わせるために$PATHをダブルクォートでは囲みません。

無事位置パラメータにセットされると、"$1"や"$2"などで、**PATH**に設定されていたディレクトリが順に参照できることがわかります。

図A IFSを変更する例

```
$ echo "$PATH"                現在のPATHの設定を表示
/home/guest/bin:/usr/local/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
$ IFS_SAVE=$IFS               IFSの値をIFS_SAVEに退避
$ IFS=:                       IFSの値を:に変更
$ set $PATH                   この状態でPATHを単語分割し、位置パラメータにセット
$ IFS=$IFS_SAVE               IFSの値を元に戻す
$ echo "$1"                   位置パラメータ"$1"の値を表示
/home/guest/bin               1番目のPATHが表示される
$ echo "$2"                   位置パラメータ"$2"の値を表示
/usr/local/bin                2番目のPATHが表示される
```

Memo

- Solarisのshでは、パラメータ展開などの展開が行われなくても、IFSによる単語分割が行われます。

参照

IFS(p.184)　ダブルクォート" "(p.209)　set(p.120)　read(p.111)

>第11章 リダイレクト