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

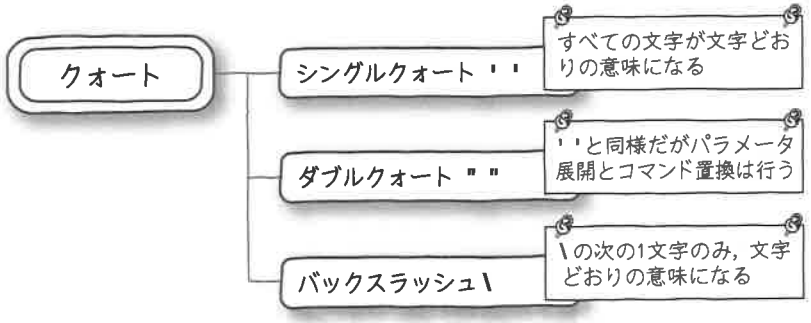
ダブルクォート " "(p.209)　　eval(p.98)

>第9章 クォートとコマンド置換

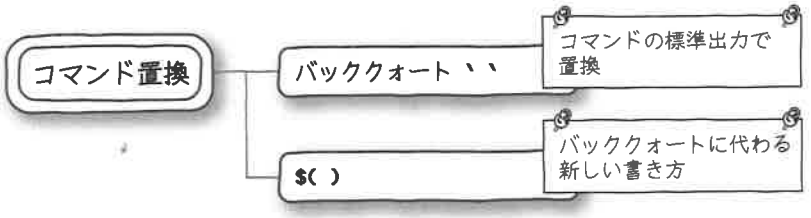
クォートとコマンド置換

シェルにはシングルクォート(' ')、ダブルクォート(" ")、バックスラッシュ(\)の3つの**クォート**があり、シェル上で特殊な意味を持つ$や*などの記号の、特殊な意味を打ち消すことができます(**図A**)。さらに、バッククォート(` `)または$( )を使った**コマンド置換**では、コマンドの標準出力を別のコマンドの引数として取り込めます(**図B**)。

図A クォート



図B コマンド置換



シングルクォート ' '

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

文字の特殊な意味を打ち消して文字列を使用する

書式 '文字列 ...'

例 echo '$5は5ドルの意味です' ..... $の特殊な意味を消して文字列をそのまま表示

基本事項

文字列を**シングルクォート**(' ')で囲むと、文字列中の各文字すべてが特殊な意味を失い、文字通りの意味として解釈されます。ただし、文字列中にシングルクォート自身を含めることはできません。

解説

シングルクォート(' ')は、クォートの中でもっとも単純明解なクォートです。シングルクォート中の文字は、スペースや改行も含めてすべて文字通りの意味になります。echoコマンドやその他のコマンドの引数で、任意の文字列をシェルの解釈を避けて使用したい場合は、基本的にシングルクォートを使うべきです。シングルクォート中ではパラメータ展開、パス名展開などの展開は一切行われません。

シングルクォートの使用テスト

図A❶のように「'$HOME'」をechoコマンドの引数に付けると、そのまま「$HOME」と表示され、文字列中の$の特殊な意味が消え、シェル変数の参照が行われていないことがわかります。一方、**図A❷**の「"$HOME"」のようにダブルクォートの場合はシェル変数が参照されてしまいます。

さらに、**図A❸❹**を見てみると、「'*'」も、そのまま「*」が表示されますが、シングルクォートがないと「*」がパス名展開により、カレントディレクトリにあるファイル名に展開されてしまいます[注1]。

注1 パス名展開については10.2節を参照してください。

図A シングルクォートの使用テスト

```
$ echo '$HOME'          ❶'$HOME'をechoコマンドの引数に指定
$HOME                   そのまま$HOMEと表示される
$ echo "$HOME"          ❷ダブルクォートを使うと
/home/guest             シェル変数の値に展開されてしまう
$ echo '*'              ❸'*'をechoコマンドの引数に指定
*                       そのまま*と表示される
$ echo *                ❹何もクォートしないと
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てしまう
```

文字列中でシングルクォートを使うには

シングルクォートで囲まれた文字列中にシングルクォート／アポストロフィ記号(')があると、そこでシングルクォートが閉じ、文字列が終了してしまいます。これは、\'のように前にバックスラッシュを付けても回避できません。

そこで、文字列中でシングルクォートを使うには、**図B**のようにいったんシングルクォートを閉じ、その直後に\'と書き、その直後から再度シングルクォートを開始します。結局「'」の代わりに「'\''」と書くことになります。

なお、ダブルクォートを使って「"Let's go !"」と記述して回避する方法もありますが、文字列中に他の特殊文字が含まれているなどで、あくまでシングルクォートを使いたい場合はやはり図Bの方法になります。

図B 文字列中でシングルクォートを使う例

```
$ echo 'Let'\''s go !'          'の代わりに'\''と書く
Let's go !                      意図通りにシングルクォート(アポストロフィ記号)が表示される
```

注意事項

シングルクォートの閉じ忘れに注意

シングルクォート中は**改行コード**もそのまま単なる文字列の一部とみなされます。したがって、シェルスクリプト中でシングルクォートの閉じ忘れがあると、次行以降に記述されたコマンドなどがすべてシングルクォート中の文字列と解釈され、シェルスクリプトが意図通りに動作しなくなります。

また、コマンドライン上でシングルクォートを閉じずに改行すると、次の例のようにセカンダリプロンプトが表示されます。意図して改行を入れた場合はこれでかまいませんが、誤って改行を入れた場合はCtrl+Cなどでいったんコマンドをキャンセルします。

```
$ echo 'hello ……シングルクォートを閉じずに改行
> ……セカンダリプロンプトが表示され、まだ' 'の中にいる
```

参照

ダブルクォート" "(p.209)　　バックスラッシュ\(p.211)

ダブルクォート " "

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

パラメータ展開とコマンド置換を除いて文字の特殊な意味を打ち消す

書式 "文字列 ..."

例 `echo "\$HOMEの値は$HOMEです"` ……「$HOMEの値は/home/guestです」のように表示される

基本事項

文字列を**ダブルクォート**(" ")で囲むと、$と`と\(ただし\の次にくる文字が$、`、"、\または改行の場合のみ)を除き、文字列中の各文字が特殊な意味を失い、文字通りの意味として解釈されます。

解説

文字列を**ダブルクォート**(" ")で囲むと、シングルクォートの場合とほぼ同様に、ほとんどの文字がその特殊な意味を失います。しかし、$と`の特殊な意味は残るため、パラメータ展開とコマンド置換は行われます。また、\の特殊な意味も一部の場合のみ残るため、ダブルクォート中で\$、\`と記述して、パラメータ展開やコマンド置換の解釈を避けることもできます。ダブルクォート(")自身や\自身はそれぞれ\"、\\と記述すれば使えます。さらに、\改行はそれ自体が取り除かれ、単なる行の継続として扱われます[注2]。

シェル変数などのパラメータ展開の際に、単に「$var」のようにして参照すると、シェル変数の値に含まれるスペースや*などの文字が解釈されてしまいます。これを避けるため、シェル変数の参照時に「"$var"」のようにダブルクォートで囲むのが基本です。コマンド置換についても同様です。

ダブルクォートの使用テスト

シェル変数を、ダブルクォート付きで参照する場合と、ダブルクォートなしで参照する場合で、その動作の違いを確認している様子を**図A**に示します。このように、ダブルクォート付きならば、シェル変数の値をそのまま得られることがわかります。

注2　パラメータ展開については10.2節、コマンド置換については9.3節を参照してください。

図A ダブルクォートの使用テスト

```
$ var='   hello   world   !!'          シェル変数に、スペースの連続を含む文字列を代入
$ echo "$var"                          ダブルクォートで囲んでシェル変数の内容を表示
   hello   world   !!                  スペースを含め、正しく表示される
$ echo $var                            ダブルクォートで囲まないと
hello world !!                         スペースが区切り文字と解釈されてしまう
$ var='*'                              シェル変数に*を代入（' 'は省略可）
$ echo "$var"                          ダブルクォートで囲んでシェル変数の内容を表示
*                                      正しく*のみが表示される
$ echo $var                            ダブルクォートで囲まないと
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てしまう
```

文字列中でダブルクォートを使うには

ダブルクォートで囲まれた文字列中では、**図B**のように頭に\を付けて\"と記述することにより、ダブルクォート自身を普通の文字として使えます。

図B 文字列中でダブルクォートを使う例

```
$ echo "He says \"Hello\""             ダブルクォートの中でダブルクォートを使う
He says "Hello"                        たしかに意図通りに表示される
```

注意事項

ダブルクォートの閉じ忘れに注意

ダブルクォート中は**改行コード**もそのまま単なる文字列の一部とみなされます。したがって、シェルスクリプト中でダブルクォートの閉じ忘れがあると、次行以降に記述されたコマンドなどがすべてダブルクォート中の文字列と解釈され、シェルスクリプトが意図通りに動作しなくなります。

また、コマンドライン上でダブルクォートを閉じずに改行すると、次の例のようにセカンダリプロンプトが表示されます。意図して改行を入れた場合はこれでかまいませんが、誤って改行を入れた場合はCtrl+Cなどでいったんコマンドをキャンセルします。

```
$ echo "hello ……ダブルクォートを閉じずに改行
> ……セカンダリプロンプトが表示され、まだ" "の中にいる
```

参照

シングルクォート' '(p.207)　バックスラッシュ\(p.211)

バックスラッシュ \

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

次の1文字の特殊な意味を打ち消す

書式 \文字

例 echo \$はドル記号、*はアスタリスクです ……$と*の特殊な意味を打ち消して表示

基本事項

文字の前に**バックスラッシュ**(\)を付けると\の直後の1文字が特殊な意味を失い、文字通りの意味として解釈されます。ただし、\改行(\Enter)の場合は\改行自体が取り除かれます。

解説

バックスラッシュ(\)は、1文字だけのシングルクォートに似ています。echoコマンドやその他のコマンドの引数の中に、1文字だけ特殊な文字が含まれているような場合、バックスラッシュを使うのが簡単でしょう。\$や*のようにバックスラッシュが付けば、パラメータ展開やパス名展開は行われなくなります。また、\自身は\\で表現することができます。

バックスラッシュの使用テスト

図A❶のように、「\$HOME」をechoコマンドの引数に付けると、\$の部分が単なる文字としての$と解釈され、その結果「$HOME」と表示されます。一方、バックスラッシュなしで「$HOME」と記述した場合(**図A❷**)はシェル変数が参照されてしまいます[注3]。さらに**図A❸❹**を見てみると、*も、普通の文字として*が表示されますが、バックスラッシュがないと、パス名展開により、*がカレントディレクトリにあるファイル名に展開されてしまいます[注4]。

図A バックスラッシュの使用テスト

```
$ echo \$HOME                          ❶\$HOMEをechoコマンドの引数に指定
$HOME                                  そのまま$HOMEと表示される
$ echo $HOME                           ❷バックスラッシュを付けないと
/home/guest                            シェル変数の値に展開されてしまう
$ echo \*                              ❸\*をechoコマンドの引数に指定
*                                      そのまま*と表示される
$ echo *                               ❹バックスラッシュを付けないと
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てしまう
```

注3　シェル変数の代入と参照の項(p.159)も合わせて参照してください。
注4　パス名展開については10.2節を参照してください。

バックスラッシュによる行の継続

\改行のように、バックスラッシュの直後に改行がある場合は特別で、\と改行の2文字分が、何事もなかったかのように取り除かれます。このため、\改行は任意の位置で改行を行いたい場合に利用できます。とくに、シェル文法的に区切り文字の改行を入れられない場合や、普通に改行するとリストの終端とみなされてしまう場合に便利です。

図Bは少々極端な例ですが、ls -Fというコマンドを1文字ごとに\改行を入れて5行に分けて入力したものです。このように、コマンドは正常に実行されていることがわかります。\改行は取り除かれて、スペースすら入れられずに1行につながって解釈されることに注目してください。

図B 行の継続の極端な例

```
$ l\                                    lsコマンドのlのみで行の継続
> s\                                    lsコマンドのsのみで行の継続
>  \                                    区切り文字のスペースのみで行の継続
> -\                                    -Fオプションの-のみで行の継続
> F                                     -FオプションのFを入力して最後に改行する
bin/      doc/      memo.txt  src/      正常にls -Fコマンドが実行される
```

注意事項

ダブルクォート中のバックスラッシュは動作が変わる

元々特殊な意味を持っていない文字にバックスラッシュ(\)を付けた場合、たとえば次の例の前半の「\x」は、単に「x」と解釈されます。しかし、ダブルクォートの中では、\は次に$、`、"、\または改行が来た場合のみ特殊な意味を持つため、次の例の後半の「"\x"」の「\」は特殊な意味を失い、そのまま「\x」と表示されてしまいます。

```
$ echo \x ……普通の文字にバックスラッシュを付けると
x ……普通の文字のみが表示される
$ echo "\x" ……ダブルクォートの中の\xは
\x ……そのまま\xと表示される
```

Memo

- \は、環境によっては半角の「¥」と表示されますが、UNIX系OS環境では通常「\」と表示されます。

参照

シングルクォート' '(p.207)　シェル変数の代入と参照(p.159)　ダブルクォート" "(p.209)

コマンド置換 ``

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

コマンドの引数またはコマンド名を別のコマンドの標準出力で置換する

書式 `リスト`

リストの右端は、改行や;で終端されていなくてもかまわない

例 `basename "`pwd`"` ……パス名を除くカレントディレクトリ名を表示する

基本事項

コマンド置換では、まずバッククォート(` `)で囲まれたリストが実行され、その結果、標準出力に出力された文字列で、` `の部分が置き換えられます。標準出力の最後に**改行コード**が付いている場合は取り除かれます。

` `の中では、$、`、\の直前の\については特別に解釈されます。

解説

コマンド置換は、まず別のコマンドを実行し、その標準出力に出力された文字列を新たにコマンドの引数の一部として取り込みたい場合に使用します。

exprコマンドでの使用例(` `)

シェル変数の数値演算を行うにはexprコマンドを使いますが、ここでexprコマンドの標準出力を取り込むためにバッククォート(` `)を用います。

なお、このように` `で囲んだものを直接シェル変数に代入する場合は、置換された文字列に対してはパス名展開や単語分割が行われないため、` `の外側をダブルクォート(" ")で囲む必要はありません。

図Aでは、exprコマンドで、シェル変数"$i"の値に「1」を加える計算を行い、その結果をバッククォートで取り込み、同じシェル変数iに代入しています。値の「3」が「+1」されて「4」になっていることがわかります。

図A exprコマンドでの使用例

```
$ i=3                                   シェル変数iに3を代入
$ echo "$i"                             念のため値を表示
3                                       たしかに3と表示される
$ i=`expr "$i" + 1`                     exprコマンドで"$i"の値に1を加えて再びiに代入する
$ echo "$i"                             "$i"の値を表示
4                                       たしかに4と表示される
```

さらにダブルクォートを付ける(` `)

バッククォート(` `)で取り込んだ文字列をコマンドの引数とする場合(直接変数に代入する場合を除く)、その文字列に対してさらに**パス名展開**と**単語分割**が行われます。このため、**図B❶**のように、カレンダーを表示するcalコマンドの出力をechoコマンドの引数に取り込むと、その表示が崩れてしまいます。これは単語分割により、文字列中の複数のスペースや改行が、区切り文字とみなされてしまうことによります。

そこで、**図B❷**のように、バッククォートの外側をさらにダブルクォート(" ")で囲みます。こうすればパス名展開と単語分割が避けられ、正しくカレンダーが表示されます。

``のネスティング

バッククォート(` `)で囲まれた文字列中に別のバッククォートを用いて、コマンド置換をネスティングすることもできます。**図C**は「basename "`pwd`"」という、カレントディレクトリのパス名を除いたディレクトリ名を表示するコマンドのセット全体を、さらに` `で囲んで、シェル変数dirに代入している例です。ネスティングの構造が正しく解釈されるように、内側の「` `」は、バックスラッシュを付けて「\` \`」とする必要があります。なお、ここではシェル変数に直接代入しているため、外側の` `をさらに" "で囲む必要はありません。

図B さらにダブルクォートを付ける

```
$ echo `cal 1 2038`                          ❶calの出力をechoの引数として取り込む
1月 2038 日 月 火 水 木 金 土 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31             これでは表示が崩れてしまう
$ echo "`cal 1 2038`"                        ❷バッククォートの外側をダブルクォートで囲む
      1月 2038                               これでカレンダーが正しく表示される
日 月 火 水 木 金 土
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

図C バッククォートのネスティング

```
$ pwd                                        カレントディレクトリを表示
/usr/local/bin                               現在/usr/local/binにいる
$ basename "`pwd`"                           まずはネスティングしないバッククォート
bin                                          basenameであるbinが表示される
$ dir=`basename "\`pwd\`"`                   バッククォートをネスティングしてシェル変数に代入
$ echo "$dir"                                そのシェル変数をechoする
bin                                          正しくbinと表示される
```


等価catコマンドによるコマンド置換

Warning
Linux (bash) ○ FreeBSD (sh) × Solaris (sh) ×
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、移植性が失われるため、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`"             通常のシェル文法でcat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

バッククォートの中のバックスラッシュに注意

バッククォート(` `)で囲まれた文字列中でも、\$、\`、\\の場合は\が特別な意味を持ちます。この場合、バッククォートの中でさらにシングルクォートで囲まれていても、\の特殊な意味は消えません。したがって次の例では、シングルクォートで囲まれて'\\'となっているにもかかわらず、内側のbasenameコマンドには'\'の引数が渡され、その出力が\となり、外側のbasenameコマンドの出力も\となります。バッククォートの中でsedやgrepなどの正規表現で\を多用する場合、その\の解釈にはかなりの注意を要します。

```
$ basename `basename '\\'`  ……` `の中の'\\'に対してbasenameを実行
\  ……\1つになってしまう
```

Memo

- bashやFreeBSDのshでは、` リスト `の代わりに$(リスト)と書く新しい書式も使えます。詳しくはコマンド置換($( ))の項(p.216)を参照してください。

参照

expr(p.261)　単語分割(p.237)　ダブルクォート" "(p.209)　basename(p.263)
シングルクォート' '(p.207)

コマンド置換 $( )

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

コマンドの引数またはコマンド名を別のコマンドの標準出力で置換する

書式 $(リスト)

リストの右端は、改行や；で終端されていなくてもかまわない

例 `basename "$(pwd)"` ……パス名を除くカレントディレクトリ名を表示する

基本事項

コマンド置換では、まず$( )で囲まれたリストが実行され、その結果、標準出力に出力された文字列で$( )の部分が置き換えられます。標準出力の最後に**改行コード**が付いている場合は取り除かれます。

` `によるコマンド置換とは異なり、$( )の中では\$、\`、\\が特別に解釈されることはありません。

解説

$( )は、` `に代わるコマンド置換の新しい記述形式です。動作は` `を使った場合と基本的に同じですが、リスト中の\の解釈について一部違いがあります。$( )によるコマンド置換は、括弧の対応により、その開始と終了がわかりやすく、$( )をネスティングして記述することも容易です。ただし、Solarisのshでは使えないので注意が必要です。

exprコマンドでの使用例($( ))

シェル変数の数値演算を行うにはexprコマンドを使いますが、ここでexprコマンドの標準出力を取り込むために$( )を用います。

なお、このように$( )で囲んだものを直接シェル変数に代入する場合は、置換された文字列に対してはパス名展開や単語分割が行われないため、$( )の外側をダブルクォートで囲む必要はありません。

図Aでは、exprコマンドで、シェル変数"$i"の値に「1」を加える計算を行い、その結果を$( )で取り込み、同じシェル変数iに代入しています。値の「3」が「＋1」されて「4」になっていることがわかります。

図A exprコマンドでの使用例

```
$ i=3                              シェル変数iに3を代入
$ echo "$i"                        念のため値を表示
3                                  たしかに3と表示される
$ i=$(expr "$i" + 1)               exprコマンドで"$i"の値に1を加えて再びiに代入する
$ echo "$i"                        "$i"の値を表示
4                                  たしかに4と表示される
```


さらにダブルクォートを付ける($( ))

$( )で取り込んだ文字列をコマンドの引数とする場合(直接変数に代入する場合を除く)、その文字列に対してさらにパス名展開と単語分割が行われます。このため、**図B❶**のように、カレンダーを表示するcalコマンドの出力をechoコマンドの引数に取り込むと、その表示が崩れてしまいます。これは単語分割により、文字列中の複数のスペースや改行が、区切り文字とみなされてしまうことによります。

そこで、**図B❷**のように、$( )の外側をさらにダブルクォートで囲みます。こうすればパス名展開と単語分割が避けられ、正しくカレンダーが表示されます[注5]。

$( )のネスティング

$( )で囲まれた文字列中に別の$( )を記述して、コマンド置換をネスティングすることもできます。$( )は、カッコの対応がわかりやすいため、バッククォート(` `)を使うよりもネスティングの記述が容易になります。**図C**は、basename "$(pwd)"という、カレントディレクトリのパス名を除いたディレクトリ名を表示するコマンドのセット全体を、さらに$( )で囲んで、シェル変数dirに代入している例です。なお、ここではシェル変数に直接代入しているため、外側の$( )をさらに" "で囲む必要はありません。

図B さらにダブルクォートを付ける

```
$ echo $(cal 1 2038)                    ❶calの出力をechoの引数として取り込む
1月 2038 日 月 火 水 木 金 土 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31        これでは表示が崩れてしまう
$ echo "$(cal 1 2038)"                  ❷$( )の外側をダブルクォートで囲む
      1月 2038                          これでカレンダーが正しく表示される
日 月 火 水 木 金 土
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

図C $( )のネスティング

```
$ pwd                                   カレントディレクトリを表示
/usr/local/bin                          現在/usr/local/binにいる
$ basename "$(pwd)"                     まずはネスティングしないコマンド置換
bin                                     basenameであるbinが表示される
$ dir=$(basename "$(pwd)")              $( )をネスティングしてシェル変数に代入
$ echo "$dir"                           そのシェル変数をechoする
bin                                     正しくbinと表示される
```

注5 パス名展開については10.2節を参照してください。

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