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)

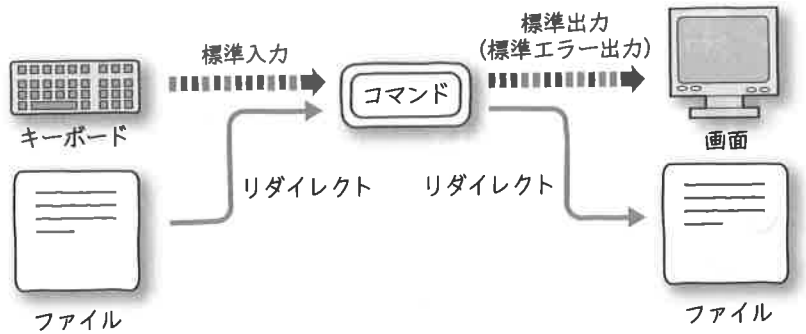
>第11章 リダイレクト

リダイレクト

多くのコマンドは、**標準入力**から入力し、**標準出力**に出力するように設計されています。標準入力と標準出力は、通常はそれぞれ「キーボード」と「画面」であるため、キーボードから入力して画面に出力することになります。標準入出力は**リダイレクト**することができ、リダイレクトすれば、キーボードから入力する代わりにファイルから読み込んだり、画面に表示する代わりにファイルに書き込んだりできます(**図A**)。

シェル上では、<、>などの記号でリダイレクトが可能です。さらに、標準出力だけでなく**標準エラー出力**をリダイレクトしたり、**ファイル記述子を使ったリダイレクト**も可能です。本章では、これらのリダイレクトについて解説します。

図A 標準入力／標準出力とリダイレクト



標準入力のリダイレクト <

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

コマンドの標準入力にファイルを入力する

書式 コマンド < ファイル名

例
```
cat < file ……fileをcatコマンドの標準入力に入力し、内容を表示する
```

基本事項

冒頭の書式のように記述すると、ファイル名で指定したファイルが**標準入力**(ファイル記述子「0」)として読み出しオープンされた状態でコマンドが実行されます。

解説

コマンド実行時の標準入力は、デフォルトではキーボードです。しかし、「コマンド < ファイル名」のように<記号を使って記述することにより、コマンドの標準入力をファイルにリダイレクトすることができます。リダイレクトされたコマンドは、標準入力としてキーボードから文字列を読み込む代わりにファイルから文字列を読み込むようになります。

なお、リダイレクトはあくまで**コマンド**に対して行われるため、単純コマンドだけでなく、構文やサブシェルなどの複合コマンドに対してもリダイレクトが可能です。

リダイレクトの場合と、コマンドの引数指定の場合との違い

冒頭の例ではcatコマンドの標準入力を「file」にリダイレクトしていますが、catコマンドでは元々コマンドの引数にファイル名を指定してファイルを読み込むこともできます。つまり、**リストA**のように記述すれば、❶も❷も、どちらも同じように動作します。

しかし、両者には次のような細かい違いがあります。❶はシェル自身が「file」をオープンし、catコマンドのほうではそのファイルをオープンする処理を行いません。もしファイルが存在しなかった場合、そのエラーメッセージはシェルから出され、catコマンドは起動すらされません。また、リダイレクトの部分はcatコマンドに対する引数ではないため、catコマンド側では、リダイレクトされたファイルのファイル名を直接知ることができません。

一方、❷では、「file」はcatコマンド自身によってオープンされます。ファイルが存在しない場合などのエラーメッセージはcatコマンド自身から出されます。

リストA リダイレクトの場合と、コマンドの引数指定の場合

```
cat < file ……❶標準入力をfileにリダイレクト
cat file ……❷fileを引数で指定し、catコマンド自身がfileをオープン
```

コマンド名の左側でのリダイレクト

リダイレクトの記号は、コマンドラインのすべての引数の右側に書くのが普通ですが、これは引数の途中に割り込んで書いてもかまいません。さらに、変わった書き方ですが、コマンド名よりも左側に書いてもかまいません。

具体的には**リストB**のようになります[注1]。❶は標準的な書き方で「file」をsort -nコマンドの標準入力にリダイレクトし、その標準出力をパイプでuniqコマンドに渡しています。

❷はリダイレクトを-nオプションよりも左に書いた例で、これでもまったく同じ意味になります。

❸はさらにsortコマンドよりも左にリダイレクトを記述した例で、これでも❶や❷とまったく同じ意味になります。

リストB リダイレクト記号の位置

```
sort -n < file | uniq ……❶標準入力のfileを数値的にソートし、重複行をカットする
sort < file -n | uniq ……❷< fileを-nよりも左側に記述しても同じ意味になる
< file sort -n | uniq ……❸< fileをsortよりも左側に記述しても同じ意味になる
```

Memo

「< ファイル」は「0< ファイル」の省略形であり、「0」はファイル記述子「0」(標準入力)を意味します。したがって「cat < file」は「cat 0< file」と書いてもかまいません。また、「3< ファイル」のように「0」以外の任意のファイル記述子を指定して、ファイルを入力としてリダイレクトすることも可能です。

Column

ファイル記述子について

ファイルをオープンすると、OSからは**ファイル記述子**という番号が割り当てられ、以降、このファイルの読み書きなどの操作は、ファイル記述子を通じて行われます。ファイル記述子のうち、**0**、**1**、**2**の3つの番号は、**表a**のようにコマンドの起動直後からすでに割り当てられています。シェル文法上でのリダイレクトでは、このファイル記述子の番号を使うこともあるため、たとえば「標準エラー出力は2番」というように、番号を覚えておくとよいでしょう。

表a コマンドの起動直後からすでに割り当てられているファイル記述子

ファイル記述子	割り当て
0	標準入力
1	標準出力
2	標準エラー出力

参照

単純コマンド(p.29)　複合コマンド(p.30)　cat(p.267)

注1　sortはテキストファイルを行単位でソートし出力するコマンドで、-nを付けて数値的にソートしています。uniqは重複行をカットして出力するコマンドです。

標準出力のリダイレクト >

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

コマンドの標準出力をファイルに出力する

書式　コマンド > ファイル名

例

```
echo 'Hello World' > file ……echoコマンドの標準出力をfileに書き込む
```

基本事項

冒頭の書式のように記述すると、(ファイル名で指定したファイルが**標準出力**(ファイル記述子「1」)として書き込みオープンされた状態でコマンドが実行されます。

解説

コマンド実行時の標準出力は、デフォルトでは画面です。しかし、「コマンド > ファイル名」のように>記号を使って記述することにより、コマンドの標準出力をファイルにリダイレクトすることができます。リダイレクトされたコマンドは、標準出力の文字列を画面に出力する代わりにファイルに書き込むようになります。

リダイレクトに指定されたファイルが存在しない場合は新たに作成されますが、すでに同名のファイルが存在する場合は、そのファイルサイズがゼロに切り詰められます。

なお、リダイレクトはあくまで**コマンド**に対して行われるため、単純コマンドだけでなく、構文やサブシェルなどの複合コマンドに対してもリダイレクトが可能です。

シェル自身がファイルをオープン

リダイレクトでは、ファイルをオープンする処理はコマンドではなくシェルによって行われます。たとえば**図A**のように、一般ユーザが書き込めない/ディレクトリ直下のファイルにリダイレクトしようとすると、エラーメッセージがシェル自身から出力されます。この場合、echoコマンドは起動されません。

また、リダイレクトの部分はコマンドに対する引数ではないため、コマンド側ではリダイレクトされたファイルのファイル名については関知しません。

図A シェル自身がファイルをオープン

```
$ echo 'Hello World' > /file ……echoの標準出力を/fileにリダイレクトしてみる
bash: /file: Permission denied ……シェル自身からエラーメッセージが出される
```

リダイレクトのみでファイルを作成

リストAのように、コマンド名を記述せずに、リダイレクトのみを実行することができます。この例では、カレントディレクトリに「timestamp」という名前の、サイズゼロのファイルが作成されます。これは、暗黙の：コマンドが省略されているものと考えてかまいません。

この動作はtouch *timestamp*を実行した場合に似ています。ただし、touchコマンド[注2]とは違って、すでに同名のファイルが存在していた場合、そのファイルサイズがゼロに切り詰められます。このことを利用して、ファイルそのものは削除せずにファイルの中身を消去したい場合に有効です。

コマンド名の左側でのリダイレクト

リダイレクトの記号は、コマンドラインのすべての引数の右側に書くのが普通ですが、これは引数の途中に割り込んで書いてもかまいません。さらに、変わった書き方ですが、コマンド名よりも左側に書いてもかまいません。

具体的には**リストB**のようになります。❶は標準的な書き方で、echoコマンドの標準出力を「file」にリダイレクトしています。

❷はリダイレクトをechoコマンドの引数(hello)よりも左に書いた例で、これでもまったく同じ意味になります。

❸はさらにechoコマンドよりも左にリダイレクトを記述した例で、これでも❶や❷とまったく同じ意味になります。

リストA リダイレクトのみでファイルを作成

```
> timestamp ……timestampというサイズゼロのファイルを作成
```

リストB リダイレクト記号の位置

```
echo hello > file ……❶echoコマンドの引数の右側にリダイレクトを記述
echo > file hello ……❷echoコマンドの引数よりも左にリダイレクトを記述
> file echo hello ……❸echoコマンドよりも左にリダイレクトを記述
```

Memo

- 「> ファイル」は「1> ファイル」の省略形です。したがって「echo hello > file」は「echo hello 1> file」と書いてもかまいません。
- bashまたはFreeBSDのshで、set -Cコマンドが実行されているか、またはシェルの起動時に-Cオプションが指定されている場合、すでに存在するファイルに対して「> ファイル」を実行するとエラーになります。これは、「>| ファイル」と実行すれば強制的にファイルに書き込むことができます。

参照

単純コマンド(p.29)　複合コマンド(p.30)　：コマンド(p.87)

注2　touchについては、p.272、：コマンドの項(p.87)も参考にしてください。

標準出力のアペンドモードでのリダイレクト >>

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

コマンドの標準出力をファイルに追加出力する

書式　コマンド >> ファイル名

例

```
date > file ……dateコマンドの標準出力をfileに書き込む
ls -l >> file ……lsコマンドの標準出力をfileに追加して書き込む
```

基本事項

冒頭の書式のように記述すると、ファイル名で指定したファイルが**標準出力**(ファイル記述子「1」)として**アペンドモード**で書き込みオープンされた状態でコマンドが実行されます。

解説

コマンドの標準出力をファイルにリダイレクトする記号の>の代わりに>>を使うと、ファイルは**アペンド(追加)モード**でオープンされ、コマンドの標準出力はファイルの最後尾に追加されるようになります。

指定したファイルがすでに存在していた場合、>とは違ってファイルサイズはゼロには切り詰められません。また、ファイルが存在していなかった場合は新しいファイルが作成されます。

アペンドモードは、冒頭の例のように、複数のコマンドの標準出力を同じファイルに追加書き込みしたい場合に便利です。

標準出力のアペンドモードでのリダイレクトは、アペンドモードであることを除いて、標準出力のリダイレクトと同じです。

Memo

- 「>> ファイル名」は「1>> ファイル名」の省略形です。したがって「date >> file」は「date 1>> file」と書いてもかまいません。

参照

標準出力のリダイレクト(p.243)

標準エラー出力のリダイレクト 2>

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

コマンドの標準エラー出力をファイルに出力する

書式 コマンド 2> ファイル名

例
```
rmdir /some/dir 2> /dev/null ……エラーメッセージを出さずに/some/dirというディレクトリを削除する
```

基本事項

冒頭の書式のように記述すると、ファイル名で指定したファイルが**標準エラー出力**(ファイル記述子「2」)として書き込みオープンされた状態でコマンドが実行されます。

解説

コマンド実行時の標準エラー出力は、標準出力と同じくデフォルトでは画面に出力されます。これは「コマンド 2> ファイル名」のように>記号の前に標準エラー出力のファイル記述子である「2」を指定することにより、標準エラー出力をファイルにリダイレクトすることができます。標準エラー出力のリダイレクトは、ファイル記述子「2」に対してリダイレクトが行われることを除いて、標準出力のリダイレクトと同じです。

なお、一般に「2」以外のファイル記述子でも、たとえば「3> *file*」「4> *file*」のように記述して、それぞれファイル記述子「3」「4」をファイルにリダイレクトすることができます。

また、「2>> *file*」と記述することにより、標準エラー出力をファイルにアペンドモードでリダイレクトすることができます[注3]。

エラーメッセージを捨てる

冒頭の例のように標準エラー出力を**/dev/null**にリダイレクトすると、エラーメッセージを表示しないようにできます。冒頭では、rmdirコマンドの実行時にすでにディレクトリが削除されていて存在しない場合でも、エラーメッセージを出さないようにしているのです。コマンドによっては、cmpコマンドの-sオプションのように、エラーメッセージを非表示にできる場合がありますが、そのようなオプションがないコマンドの場合、「2> /dev/null」という書き方がよく使われます。

注3 標準出力のリダイレクトの項(p.243)、標準出力のアペンドモードでのリダイレクトの項(p.245)、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。

標準エラー出力を、標準出力とともにファイルに落とす

標準エラー出力を、標準出力と同時にリダイレクトし、同じファイルに書き込みたい場合は、**図A**のようにファイル記述子を使ったリダイレクトを用います[注4]。図Aはmakeコマンドでの例です。

図A 標準エラー出力・標準出力の同時リダイレクト
```
$ make > make-log 2>&1 ……makeの出力を、標準エラー出力を含めてmake-logに書き込む
```

注意事項

ファイル記述子と>との間にスペースを入れない

ファイル記述子の「2」と>との間にスペースを入れてはいけません。スペースがあると「2」という文字列が、リダイレクトではなくコマンドに対する引数であると誤って解釈されてしまいます。

○正しい例
```
rmdir /some/dir 2> /dev/null
```

×誤った例
```
rmdir /some/dir 2 > /dev/null
```

Memo

- bashまたはFreeBSDのshで、set -Cコマンドが実行されているか、またはシェルの起動時に-Cオプションが指定されている場合、すでに存在するファイルに対して「2> ファイル」を実行するとエラーになります。これは、「2>| ファイル」と実行すれば強制的にファイルに書き込むことができます。

注4 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。

ファイル記述子を使ったリダイレクト >&

オープン済みの標準出力や標準エラー出力などを複製する

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

書式 コマンド 0 | 1 | 2 | …… >&0 | 1 | 2 ……
ファイル記述子を指定する

例
```
echo 'ファイルが存在しません' 1>&2 ……標準エラー出力にエラーメッセージを出力
```

基本事項

オープン済みの標準出力や標準エラー出力またはその他のファイル記述子を操作するには、**ファイル記述子**を使ったリダイレクトを用います。冒頭の書式のように記述すると、**>&の左側**に番号で指定されたファイル記述子を、**>&の右側**に番号で指定された、出力用にオープン済みのファイル記述子にリダイレクトします。

解説

冒頭の例のように**1>&2**と記述すると、標準出力(ファイル記述子「1」)を標準エラー出力(ファイル記述子「2」)にリダイレクトできます。これは、内部的には「オープン済みのファイル記述子**2**を、ファイル記述子**1**としても使えるように複製する」という動作になっています。「ファイル記述子の複製」に着目すると、>&の右側のファイル記述子を>&の左側に複製(コピー)していることに注意してください。

ファイル記述子を使ったリダイレクトのおもな用法は「1>&2」と「2>&1」です。前者の1>&2は、echoコマンドのように、標準出力に出力する仕様のコマンドで使用して、標準エラー出力に出力させるために使います。後者の2>&1は、後述のmakeコマンドの出力のように、標準エラー出力も標準出力とまとめてリダイレクトしたい場合に使用します。

なお、「0<&3」のように、入力用にオープンされたファイル記述子をリダイレクトすることもできます。たとえば、あらかじめ「exec 3< file.txt」を実行してファイル記述子3番でfile.txtを入力用にオープンしておいてから「cat 0<&3」を実行すれば、file.txtの内容を読むことができます。

標準エラー出力と標準出力をまとめてパイプに通す

ソフトウェアをソースからコンパイルする場合に使うmakeコマンドは、通常のメッセージは標準出力に、エラーが起きた場合のメッセージは標準エラー出力に出力します。これら両方の出力をまとめてパイプ(|)に通し、teeコマンドを使ってメッセージをファイルに落としながら画面にも出力するには**図A**のようにします。teeは、標準入力をそのまま標準出力(この場合は画面)に出力するとともに、引数で指定したファイルに同じ内容を書き込むコマンドです。

ここでは、**2>&1**の記述により、標準エラー出力が標準出力にリダイレクトされ、まとめてパイプに接続されます。これは正確には、すでにパイプに接続されている標準出力が、標準エラー出力として複製されるという動作になります。

標準エラー出力と標準出力をまとめてファイルにリダイレクト

makeなどのコマンドの出力を、標準エラー出力も標準出力も両方ともファイルにリダイレクトしたい場合は、**図B**のようにします。ここでは、ファイルへのリダイレクト(> make-log)と、ファイル記述子のリダイレクト(2>&1)が同時に指定されており、これらは左から右の順に実行されます。具体的には「> make-log」によって「make-log」というファイルが標準出力(ファイル記述子「1」)としてオープンされたあと、2>&1によってそのファイル記述子「1」がファイル記述子「2」に複製されます。

ここで、感覚的にリダイレクトの順が逆に感じるかもしれませんが、「make 2>&1 > make-log」は誤りで、これでは2>&1の実行時にはまだ標準出力はデフォルトの画面のままになっているため、標準エラー出力が正しくリダイレクトできません。

なお、bashには、この目的専用の記述法があります[注5]。

ファイル記述子の退避と復帰

未使用の任意のファイル記述子を利用して、ファイル記述子の退避と復帰を行うことができます。

リストAのように、「exec > file」を実行して、以降のコマンドの標準出力を「file」にリダイレクトすることができますが、ここで標準出力を元通りに戻すことを考えてみます。標準出力が端末(画面)であると仮定してよい場合は、「exec > /dev/tty」を実行すれば標準出力が端末に戻ります。しかし、このシェルスクリプト自体の標準出力がパイプに接続されているとか、別のファイルにリダイレクトされている可能性もあります。そのような場合でも標準出力を元に戻すためには、リストAのようにあらかじめ「exec 3>&1」を実行し、標準出力(ファイル記述子1番)が示すものをファイル記述子3番にコピーしておきます。そして最後に「exec 1>&3」によって退避しておいたファイル記述子3番を標準出力にコピーし、標準出力を元通りに復帰させるのです。

なお、ここでは退避用にファイル記述子3番を利用していますが、未使用(変化させてかまわない)なら、何番のファイル記述子でも利用できます。

図A 標準エラー出力と標準出力をまとめてパイプに通す

```
$ make 2>&1 | tee make-log ……標準エラー出力を標準出力に合流させ、teeでmake-logに落としながら画面にも表示する
```

図B 標準エラー出力と標準出力をまとめてファイルにリダイレクト

```
$ make > make-log 2>&1 ……標準エラー出力と標準出力をmake-logというファイルに落とす
```

注5 標準出力と標準エラー出力の同時リダイレクトの項(p.251)も合わせて参照してください

リストA 標準出力をファイル記述子3番に退避して復帰させる例

```
exec 3>&1 ……標準出力をファイル記述子3番に複製（退避）して記憶しておく
exec > file ……標準出力を「file」にリダイレクトする
（ここで任意のコマンドを実行）……ここで実行したコマンドの標準出力は「file」に出力される
…省略…
exec 1>&3 ……退避されていたファイル記述子3番を標準出力に複製（復帰）する
```

注意事項

>&の左側にスペースを入れてはいけない

>&記号の**左側**にスペースを入れると、ファイル記述子の番号がコマンドの引数とみなされてしまい、正しく動作しません。また、>と&の間にもスペースは入れられません。一方、>&の**右側**にはスペースを入れてもかまいません。また、>&の左側の番号が「1」の場合は省略してもかまいません。しかし、1>&2や2>&1は、いずれもスペースを入れず、かつ番号も省略しない形式で統一したほうがわかりやすいでしょう。

○正しい例

```
echo 'Error message' 1>&2 ……この書き方に統一したほうがよい
echo 'Error message' 1>& 2 ……>&の右側はスペースが入ってもよい
echo 'Error message' >&2 ……>&の左側の1は省略できる
```

×誤った例

```
echo 'Error message' 1 >&2 ……>&の左側にはスペースは入れられない
echo 'Error message' 1> &2 ……>と&の間にはスペースは入れられない
```

Memo

bashでは、ファイル記述子の複製の代わりに「ファイル記述子の移動」を行うことができます。たとえば、「1>&2」の代わりに「1>&2-」と記述すると、ファイル記述子2番がファイル記述子1番に移動します。これはファイル記述子2番をファイル記述子1番に複製するとともに、移動元のファイル記述子2番をクローズするという動作になっています。

標準出力と標準エラー出力の同時リダイレクト &>

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

bashで標準出力と標準エラー出力を同時にファイルにリダイレクトするには&>が使える

書式 コマンド &> ファイル名

例

```
make &> make-log ……標準出力と標準エラー出力をmake-logというファイルに書き込む
```

基本事項

冒頭の書式のように記述すると、コマンドの標準出力と標準エラー出力の両方が、&>の右側に指定されたファイル（ファイル名）にリダイレクトされます。

解説

bashでは「&> *file*」という記述法が使え、標準出力と標準エラー出力を同時にファイルにリダイレクトすることができます。makeコマンドの出力をすべてファイルに書き込みたい場合に便利でしょう。ただし、この文法を使うのはコマンドライン上の作業だけにとどめておき、シェルスクリプト中の記述では「> *file* 2>&1」のように標準的な記述を行った方がよいでしょう[注6]。

なお、「&> *file*」の代わりに、csh系のシェル由来の「>& *file*」という記述も使えます。しかしこれは推奨されておらず、ファイル名が数字のみの場合に「>& 123」のように実行すると、ファイル記述子を使ったリダイレクトと解釈されて正しく動作しないことから、「&> *file*」を使ったほうがよいでしょう。

参照

ファイル記述子を使ったリダイレクト（p.248）

注6 ファイル記述子を使ったリダイレクトの項（p.248）を参照してください。

ファイル記述子のクローズ >&- / <&-

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

ファイル記述子をクローズするには >&-や<&-を用いる

書式 コマンド 0 | 1 | 2 | …… >&-
コマンド 0 | 1 | 2 | …… <&-
ファイル記述子を指定する

例
```
mycommand 0<&- 1>&- 2>&-
```
標準入力、標準出力、標準エラー出力をすべてクローズしてコマンドを実行

基本事項

冒頭の書式のように記述すると、>&-(出力用)または<&-(入力用)の左側に番号で指定された**ファイル記述子**が**クローズ**された状態でコマンドが実行されます。

解説

コマンドの実行の際に、標準入力、標準出力、標準エラー出力といった、通常は最初からオープンされている**ファイル記述子**や、「exec 3> file」コマンドなどで自分でオープンしたファイル記述子を、都合により**クローズ**したい場合があります。このような際にファイル記述子のクローズを用います。ファイル記述子のクローズは、**/dev/null**へのリダイレクトに似ていますが、**/dev/null**へのリダイレクトとは違ってファイル記述子自体が**未使用**の状態になります。ファイル記述子のクローズの書式は、ファイル記述子を使ったリダイレクトの書式において右側のファイル記述子の番号を「-」に変えたものと同じです。

参照

ファイル記述子を使ったリダイレクト(p.248)

読み書き両用オープン <>

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

コマンドのファイル記述子を読み書き両用でオープンしたファイルにリダイレクトする

書式 コマンド | 0 | 1 | 2 …… | <> ファイル名
ファイル記述子を指定する

例
```
mycommand <> file
```
標準入力(出力としても利用)を読み書き両用のfileにリダイレクト

基本事項

冒頭の書式のように記述すると、**<>の左側**に番号で指定されたファイル記述子が、**<>の右側**に指定されたファイルに、**読み書き両用**でオープンされた状態でリダイレクトされます。ファイル記述子の番号を省略した場合は、ファイル記述子「0」になります。

解説

UNIX系OSでは、ファイルは「読み出し専用」「書き込み専用」でのオープンのほか、「読み書き両用」でオープンすることが可能です。シェル上でも、<>の記述を使えば**読み書き両用**でファイルをオープンすることができます。<>の左にファイル記述子の番号を指定すれば、標準入力だけでなく、任意のファイル記述子がリダイレクトできます。

ヒアドキュメント

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

コマンドの標準入力に一定の文書を入力する

書式 コマンド <<[-] 終了文字列
ヒアドキュメント本体
⋮
終了文字列

例

```
md5sum -c << 'EOF' ……ヒアドキュメントでmd5sumをチェック
1cd9acec240b12af6a1f377eefd7573f  myfile1 ……md5sumの値（1行目）
d41d8cd98f00b204e9800998ecf8427e  myfile2 ……md5sumの値（2行目）
EOF ……ヒアドキュメントの終了
```

基本事項

ヒアドキュメントでは、次行以降、行頭に終了文字列が現れる直前の行までのヒアドキュメント本体の内容が、コマンドの標準入力にリダイレクトされます。

<<の右側の終了文字列がクォートされている場合は、ヒアドキュメント本体は一切の展開が行われません。終了文字列がクォートされていない場合は、ヒアドキュメント本体に対して、パラメータ展開とコマンド置換が行われます。この場合、$と`は特殊な意味を持ち、\は、$、`、\または改行の直前のみ特殊な意味を持ちます。

<<の直後に-を付けた場合はヒアドキュメント本体の行頭のタブが無視されるようになります。

解説

一定の内容の文書を即席で作成して、これをコマンドの標準入力にリダイレクトしたいことが時々あります。このような場合には**ヒアドキュメント**(*here document*)を使うと便利です。

ヒアドキュメント本体がシェルによって展開されるのを避けるため、基本的には終了文字列を**クォート**するのがいいでしょう。終了文字列のクォートは、シングルクォート('EOF')、ダブルクォート("EOF")、バックスラッシュ(\EOF)のいずれであっても動作は同じですが、その動作はシングルクォートに似ているため、シングルクォートを使うのがわかりやすいでしょう。なお、ヒアドキュメント本体のあとに書く終了文字列はクォートしてはいけません。

終了文字列は、例では「EOF」という文字列を使用していますが、これはどんな文字列でもかまいません。ヒアドキュメント本体の中にはありえない、任意のランダムな文字列を使用すればよいでしょう。

一方、終了文字列をクォートしない場合、ヒアドキュメント本体はダブルクォートで囲んだ場合と同じように解釈され、パラメータ展開などが行われます。ただし、ダブルクォートとは違って、"の直前の\は特殊な意味を持たず、\"のままになります。

ヒアドキュメントによるメールの送信

ヒアドキュメントを使ってメールを送信している例を**リストA**に示します。ここでは、メール本文の文字コードを「JIS」に変換するためにnkfコマンド[注7]を使い、このnkfに対してメール本文の文字列をヒアドキュメントとして入力しています。nkfの標準出力はパイプでmailコマンド[注8]に渡しています。このように、ヒアドキュメントとパイプを同時に使うこともももちろんできます。

終了文字列のEOFはクォートしているので、メール本文中に仮に$などの文字があっても、シェルに解釈される恐れはありません。

パイプで書くこともできる

ヒアドキュメントを使わずに、同様の動作をパイプ(|)を使って行うこともできます。たとえば、冒頭のmd5sum[注9]の例をパイプを使って書くと、**リストB**のようになります。このように、基本的にはシングルクォートで囲んだ複数行の文字列をechoして、これをパイプで目的のコマンドに渡せばいいのです。

多くのシェルのヒアドキュメントの実装では、ヒアドキュメント本体を内容とするテンポラリファイルが作成され、そのファイルが標準入力にリダイレクトするようになっています。しかし、パイプを使う場合はテンポラリファイルを作成しないため、そのぶん、効率がよいと考えられます。ただし、文字列の中にシングルクォートが含まれている場合、これを別途対処しなければならないなどの注意が必要です。

リストA ヒアドキュメントによるメールの送信

```
nkf -j << 'EOF' | mail guest@example.com ……nkfでJISに変換してmailコマンドでメール送信
このメールは、 ……メール本文1行目
ヒアドキュメントによる ……メール本文2行目
メールの送信テストです。 ……メール本文3行目
EOF ……ヒアドキュメントの終了
```

リストB ヒアドキュメントの代わりにパイプを使った例

```
echo \ ……echoコマンド（わかりやすいように行の継続で改行）
'1cd9acec240b12af6a1f377eefd7573f  myfile1 ……md5sumの値（1行目）
d41d8cd98f00b204e9800998ecf8427e  myfile2' | md5sum -c ……md5sumの値（2行目）までを
                                                    パイプでmd5sumコマンドに渡す
```

Memo

「<< 終了文字列」は「0<< 終了文字列」の省略形であり、「0」はファイル記述子「0」(標準入力)を意味します。「0」以外の任意のファイル記述子に対して「3<< 終了文字列」のようにヒアドキュメントを使用することも可能です。

参照

シングルクォート' '(p.207)　ダブルクォート" "(p.209)

注7 nkfコマンドについてはAppendix「Shift_JIS➡EUC-JP一括変換」(p.306)を参考にご覧ください。
注8 mailはメールを送受信するコマンドです。
注9 md5sumを使うとMD5のチェックサム値を確認できます。

ヒアストリング

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

コマンドの標準入力に一定の文字列を入力する

書式 コマンド <<< 文字列

例
```
cat <<< 'Hello World' ……ヒアストリングでメッセージを表示
```

基本事項

ヒアストリングでは、<<<の右側に記述された1つの文字列の内容が、コマンドの標準入力にリダイレクトされます。文字列の終端には自動的に改行が付加されます。

解説

bashでは、ヒアドキュメントの代わりに**ヒアストリング**を使って一定の文字列を入力することができます。ヒアストリングで入力として扱われるデータは1つの文字列だけですが、この文字列をシングルクォートなどで囲み、クォート内に改行を入れることにより、複数行にわたる長い文字列を入力することができます。ヒアドキュメントではEOFなどの適当な終了文字列を指定した上で入力文字列の終了を示す必要がありますが、ヒアストリングでは文字列をシングルクォートなどで囲めば、そのクォートを閉じることによって文字列の終了を表せるため、より単純明解に記述することができます。

複数行をヒアストリングで入力

リストAは、4行のヒアストリングを記述し、これをcatコマンドの標準入力に入力して表示している例です。<<<の後、文字列の先頭からシングルクォートを開始し、文字列の途中ではそのまま改行します。文字列の最後の行は行末でシングルクォートを閉じます。ヒアストリングの仕様により、文字列の最後(複数行の場合は最後の行)には自動的に改行コードが付加されるため、これでちょうど4行分の入力データになります。

リストA 複数行をヒアストリングで入力
```
cat <<< 'ヒアストリング1行目 ……シングルクォートに続いて文字列を開始し、そのまま改行
ヒアストリング2行目 ……途中の行はそのまま記述（シングルクォートの途中）
ヒアストリング3行目 ……途中の行はそのまま記述（シングルクォートの途中）
ヒアストリング4行目' ……最後の行は行末でシングルクォートを閉じる
```


ヒアストリングの中でパラメータ展開とコマンド置換

ヒアストリングの文字列の中でパラメータ展開やコマンド置換を行いたい場合は、**リストB**のように、展開を行わせたい部分の直前でシングルクォートをいったん閉じ、展開させる部分のみ**ダブルクォートに切り替え**ます。ヒアストリングは全体で1つの文字列になる必要があるため、シングルクォートとダブルクォートとの切り替え部分はスペースを入れずに連続して記述します。

ヒアストリングによるメールの送信

リストCは、ヒアストリングを使ったメールの送信です。メール本文の文字コードをJISコードに変換するためのnkf -jコマンドに対して、ヒアストリングでメール本文を入力します。最後に、この出力をパイプ経由でmailコマンドに渡します。メール本文中にはシングルクォート以外のすべての文字がそのまま記述できます。本文中にシングルクォートを記述したい場合は、ヒアストリングのシングルクォートをいったん閉じ、直後にバックスラッシュを使って「\'」と記述した後、ヒアストリングのシングルクォートを再開します。形式的には「'\''」と記述することになります。

リストB ヒアストリングの中でパラメータ展開とコマンド置換
```
cat <<< 'ヒアストリングのテスト ……シングルクォートに続いて文字列を開始し、そのまま改行
$HOMEの値は'"$HOME"'です。 ……パラメータ展開を行う部分のみダブルクォートにする
`pwd`は'"`pwd`"'に置換されます。' ……コマンド置換を行う部分のみダブルクォートにする
```

リストC ヒアストリングによるメールの送信
```
nkf -j <<< 'このメールは、 ……メール本文1行目
ヒアストリングによるメールの送信テストです。 ……メール本文2行目
シングルクォートは'\''として記述します。 ……シングルクォートの記述方法
以上' | mail guest@example.com ……ヒアストリングを終了し、mailコマンドでメールを送信
```

Memo

●ヒアストリングの「<<< 文字列」は「0<<< 文字列」の省略形です。「0」以外の任意のファイル記述子に対して「3<<< 文字列」のように記述してヒアストリングを使用することも可能です。

参照

ヒアドキュメント(p.254) シングルクォート' '(p.207) ダブルクォート" "(p.209)

Column

ヒアドキュメントとヒアストリングの実際

ヒアドキュメントやヒアストリングがシェル内部でどのように扱われているかはシェルの種類によって異なりますが、bashの場合はシェル内部で一時ファイルを作成することによって実現されています。

具体的には、ヒアドキュメントなどに書かれた文字列を内容とする一時ファイルが**/tmp**ディレクトリ以下に作成され、そのファイルを所定のファイル記述子(通常は標準入力の0番)でオープンした状態でコマンドが実行されます。この一時ファイルはオープン直後に削除されるため、**/tmp**ディレクトリを見ても一時ファイルを見ることはできません。UNIX系OSではファイルを削除しても、そのファイルをオープン中のプロセスが終了するまではファイル実体が存在し続けるため、実行されたコマンドからはファイル記述子を通してヒアドキュメントなどの内容を読むことができます。

Linuxの場合は、**図a**のようにヒアドキュメント等を使用しコマンド(ls)自身で**/dev/fd/0**(または**/proc/self/fd/0**)のシンボリックリンクのリンク先を見ることにより、ヒアドキュメント等に使用されている一時ファイルの正体を確認できます。「(deleted)」の表示から、一時ファイルは削除済みであることがわかります。

なお、一時ファイルが作成されるディレクトリ(**/tmp**)は、シェル変数**TMPDIR**を設定することにより変更できます(bashのバージョンによっては変更できません)。また、ディスクレスマシンなどで**/tmp**ディレクトリなどがNFS(*Network File System*)マウントされている場合、NFSの仕様により、削除された一時ファイルは「/tmp/.nfs005f80f300002852」のような名前にリネームされ、この場合はファイルとしても見えてしまいます。

図a ヒアドキュメントとヒアストリングの一時ファイルの正体を調べる

```
$ ls -l /dev/fd/0 << EOF                          ヒアドキュメントを実行
Hello World
EOF

lr-x------ 1 guest guest 64 Mar  5 11:38 /dev/fd/0 -> /tmp/sh-th
d-1299306069 (deleted)                            一時ファイルへのリンクと、
                                                  (deleted)の表示（実際は1行）

$ ls -l /dev/fd/0 <<< 'Hello World'               ヒアストリングを実行
lr-x------ 1 guest guest 64 Mar  5 11:40 /dev/fd/0 -> /tmp/sh-th
d-1299303718 (deleted)                            一時ファイルへのリンクと、
                                                  (deleted)の表示（実際は1行）
```


>第12章 よく使う外部コマンド