初心者用
シェルスクリプト講座

第 3 版 1999 年 2 月 22 日

文責：斎藤輪太郎

1.はじめに

この入門書では初心者向けにシェルスクリプトの書き方を解説していきます。前提としてUNIX の基本的な知識とコマンド､コンピュータのごく基本的な知識が身についているものとします。

2.基本的な語彙

シェル.....ユーザの意志をコンピュータに伝えるためのプログラムの一種。
シェルスクリプト.....コマンドとシェル特有の制御文を並べたプログラム。シェルスクリプトを書くことによって、面倒なコマンド操作をシェルスクリプトで実行できる。略してスクリプトともいう。

例：自分がいるディレクトリとその中のファイル名を表示するには、pwd,ls と続けて入力すればいいが、pwd,ls を実行してくれるシェルスクリプトを書く事もできる。

これらのことはすぐにここで分からなくても結構です。学習していくうちに分かっていくでしょう。

3.vi エディタの使い方

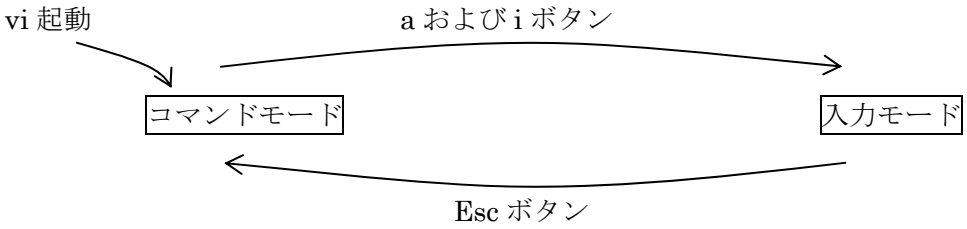
vi エディタはファイル(テキストファイル)を作るときに使います。基本的な使い方だけここでごく簡単に解説します。

まず vi <ファイル名> で<ファイル名>のファイルの作成、編集を開始できます。

□課題：vi test1 と打ち込んで test1 というファイルの編集準備をしましょう。

vi エディタには**コマンドモード**という状態と、**入力モード**という状態があります。
コマンドモードでは、カーソルの移動、文字の削除、行の削除、ファイルへの保存など編集の補助的な作業を実行できます。このモードでは原則として文字の入力はできません。
入力モードでは、実際に文字を入力することができます。

・Esc ボタンを押すことによりコマンドモードになります。
・i ボタンか a ボタンを押す事により入力モードになります。
・vi を立ち上げた直後はコマンドモードになっています。



□課題：vi test1 の後、i ボタンを押して以下のような短文を打ち込んでみましょう。

Time flies like an arrow. Two heads are better than one.

もし文字を打ち間違えてしまったら、Esc ボタンによりコマンドモードへ移り、x ボタンを押します。その後 a ボタンによりまた入力モードへ移行します。

□課題:test1 の編集が終わったらファイルをセーブしましょう。その後 vi を終わらせましょう。

ファイルのセーブは、Esc ボタンでコマンドモードに移ってから、:w ↵(リターンキー)と打ち込みます。
vi を抜けるには:q ↵と打ち込みます。ファイルをセーブしないで抜けるときには:q! ↵です。

ファイルの編集、セーブはできましたか？cat test1 で確認しましょう。

コマンドモードで実行できるコマンドとそれに対応するキー入力をいくつか挙げておきます。

入力モードへの移行

a…カーソルの位置の直後から入力を開始
i…カーソルの位置から入力を開始

削除

x…カーソルの位置の一文字を削除
dd…カーソルの行全体を削除

カーソルの移動

k…上、j…下、h…左、l…右
$…行末、:$ ↵…ファイルの末尾

ファイルへの保存、終了

:w ┐…ファイルへの保存

:q ┐…終了

※コマンドモードでは文字の入力はできないことに注意して下さい。Escを押した後、iボタンを押すことによりほぼ確実に入力モードに移行することができます。

vi はファイルの編集を行う基本的な道具です。使い方をよく練習しておいて下さい。vi の詳しい使い方に関しては他の文献を参照して下さい。

4.シェルスクリプトの基本的な作り方

ここでは実習を交えながらシェルスクリプトの書き方を解説していきます。

自分の現在のディレクトリ(カレントディレクトリ)を表示した後、ディレクトリ内のファイルを表示するシェルを書いてみます。

vi pls で以下のようなスクリプトを書いて下さい

```
pwd
ls  -l
```

セーブして vi を抜けた後、chmod +x pls と打ち込んで下さい。

そして、./pls と打ち込んでみましょう。出力結果はもちろんひとによって全く違いますが、以下のような出力は得られましたか？

```
/home/rpts/tmp
total 44
-rw-r--r--   1 report   report     4191 Nov 17 21:27 20Osaka<->Techno.mrg
-rw-r--r--   1 report   report     2205 Jan  6 09:26 990106_1
-rw-r--r--   1 report   report     2081 Nov  6 10:02 passwd.e25c
-rw-r--r--   1 report   report     2362 Nov  6 10:00 passwd.e25i
-rw-r--r--   1 report   report     1854 Nov  6 10:02 passwd.e55
-rw-r--r--   1 report   report       23 Dec 20 20:33 tmpfile
-rw-r--r--   1 report   report     4191 Nov 17 22:24 work.mrg
```

←pwd の結果
これ以下は
ls –l の結果
↓
↓

どうでしょうか？pwd,ls といったおなじみのコマンドをただ並べるだけでシェルスクリプトになってしまうのです。

以下にシェルスクリプトを書くための基本的な流れをまとめておきます。

①vi エディタでファイルを作成。　　vi test1.sh

②chmod コマンドでスクリプトを実行できるようにする。　chmod +x test1.sh

③スクリプトの実行　./test1.sh

□課題：現在時刻とユーザ名、ホスト名を表示するスクリプトを作りましょう。

5.echo コマンド

echo コマンドは画面に文字列を表示するコマンドで、コンピュータからユーザにメッセージを伝えることができます。以下のようなスクリプトを作りましょう。

```
echo “Hello, world!”
```

これを実行してみましょう。Hello, world という出力は得られましたか？この機能を使うとコンピュータがどんな処理をしているのか、ユーザに伝えやすくなります。例えば前にやったカレントディレクトリとファイルを表示するスクリプトを以下のように書き換えてみましょう。

```
echo “You are at:”
pwd
echo “Files that you have here are:”
ls -l
```

これを実行してみましょう。どうでしょうか？少し分かりやすいスクリプトになったかと思います。

□課題：自分の名前と住所を出力するスクリプトを書いてみましょう。
□課題：前にやった現在時刻とユーザ名、ホスト名を表示するスクリプトを echo コマンドを使ってもっと親切な出力をするように改良してみましょう。

ここまではコマンドの組み合わせだけでスクリプトを書いてきました。しかし次からはいよいよ変数や条件分岐などシェルスクリプトらしいやや高度な処理を交えていきます。

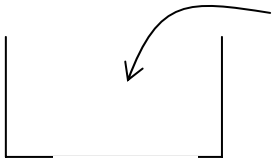
6.シェル変数

変数とはある値を(一時的に)保持しておく入れ物のことです。例えば次のスクリプトを作ってみて下さい（注意：ここから少し複雑になっていきますが、文法上、スペースを入れてはいけない箇所にスペースを入れないように注意して下さい)。

```
#!/bin/sh

var=”Hello,world!”
echo ${var}
```

実行してみましょう。Hello,world!という出力結果が得られると思います。
var=”Hello,world”というのは var という変数、つまり入れ物に”Hello,world”という文字を入れているのです。



変数 var

左の図を見てみましょう。
var=”Hello,world”により var に”Hello,world という文字列が一時的に入ります。これによって、var という変数の中をのぞいてみると、“Hello,world”という文字列を見ることができます。

変数の中をのぞき見るには、${変数名}とします。上の例では、${var}です。これを echo コマンドにかけることにより、実際にのぞき見たものを出力できるのです。

ちなみに、最初の行の#!/bin/sh はシェルの文法を決めるものです。この場合は Bourne Shell というシェルを指定しています（他にもＣシェルや Korn シェルなどがあり、それぞれ文法が微妙にちがいます。Ｃシェルは#!/bin/csh で指定します。)。コマンドの組み合わせ以外の文を書く時はスクリプトの先頭に書く必要があります。

□課題：自分の名前を変数に代入し、出力してみましょう。

変数の中身はいつでも変更できます。例えば、
var=”Hello”とした後に echo ${var}とすると Hello と出力されますが、その後に var=”Hi” にして echo ${var}とすると Hi と出力されます。

```
#!/bin/sh
var=”Hello”
echo ${var}
var=”Hi”
echo ${var}
```

実際に上のようなスクリプトを作って確かめてみましょう。

変数には文字列だけでなく、数値を入れることもできます。

```
#!/bin/sh
val=100
echo ${var}
```

実際に確かめてみましょう。

数値演算を行うこともできます(今回から#!/bin/sh の行は省略します。実際にスクリプトを組む時は必ず書いて下さい)。

```
var1=100
var2=200
var3=`expr ${var1} + ${var2}`
echo ${var3}
```

3行目のところで var1 + var2(つまり 100+200)の計算をしてその結果を var3 に入れています。`expr .....`は数値演算をする命令です。

□課題：シェルスクリプトを書いて 123x987 を計算してみましょう(‘*’が掛け算です)。

変数には他にコマンドの実行結果を入れることもできます。これは変数=`コマンド`という形で行います。例えば pwd の実行結果を res という変数に入れるには、res=`pwd`とします。

```
res=`pwd`
echo ${res}
```

□課題：hostname,whoami の実行結果を一度変数に入れてから表示してみましょう。

最後に変数には read コマンドを使ってユーザ入力を入れることができます。read 変数名とするとその変数にユーザ入力が入ります。例として以下のスクリプトを見てみましょう。

```
echo “What is your name? ”
read yname
echo “Oh, your name is ${yname}.”
```

実際に実行すると以下のような結果が得られるはずです。

```
What is your name?
Mike↵                    ←ここで Mike などと入力する。
Oh, your name is Mike.
```

このように read 文はユーザに入力を促し、その結果を指定した変数に入れることができます。

□課題：簡単な足し算計算プログラムを作ってみましょう。ユーザが任意の 2 つの数を入力すると、それに対する答えを返してくれるようにして下さい。出力例を以下に載せておきます。足し算の計算の仕方は、前の変数のところでやりましたね。

```
First number?
20 ↵        ← ユーザ入力
Second number?
30 ↵        ← ユーザ入力
Sum is 50.
```

7.条件分岐

今までのスクリプトでは、上から順番に命令が実行されるだけでした。ここからは、**ある条件に合致したときだけ命令を実行する手法**を学びましょう。

if 文は指定した条件のときに命令を実行する構文です。少し複雑な構造をしていますが、以下のような文法になっています。

```
if <条件>
then
    <条件>に合ったときの処理
else
    <条件に合わなかったときの処理
fi
```

例えばユーザから数の入力を促して、その数が 100 以上なら Big!,100 未満なら Small…と出力するスクリプトを書いてみましょう。

```
read num
if [   ${num} -gt 100 ]
then
    echo “Big!”
else
    echo “Small…”
fi
```

実際にこのスクリプトを何回か入力する数を変えて動かしてみましょう。うまくいきましたか？それでは少し解説しましょう。

if [${num} -gt 100] というのは num(入力された数)が 100 以上なら、という条件文です。この条件に合っていれば then のところ(echo “Big!”)が、合わなければ else のところ(echo “Small…”が実行されます。

条件のところにはいろんな文を書く事ができます。[num -eq 100]とすれば num が 100 に等しければ、という意味になりますし、[num -lt 100] なら num が 100 より小さければ、という意味になります。条件を書く時は[.....]のように括弧をつけて中に条件を書きます。

□課題:ユーザに数の入力を促して、その数が奇数なら odd number,偶数なら even number と出力するようにして下さい。ちなみに例えば 27 割 2 の余りは`expr 27 % 2`と表されます。

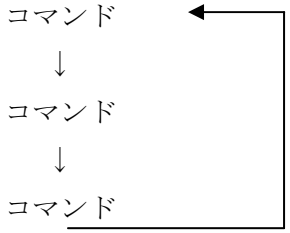
条件のところには数式の他にも様々な条件を指定することができます。以下にいくつか例を挙げておきます。

if [str1=”Gene”]　　もし str1 が”Gene”という文字列だったら～
if [-f “file1”]　　もし file1 という名前のファイルがカレントディレクトリに存在するなら～
if grep “error” /usr/adm/syslog　もし/usr/adm/syslog というファイルに”error”という文字列があるなら～

□課題 : カレントディレクトリに announce というファイルが存在するとき、そのファイルの中身を表示するようなスクリプトを作って下さい。

8.反復処理

これまでのスクリプトではプログラムの流れが必ず上から下に向かっていました。従ってこれまでの知識では同じ事を繰り返し何度もやるということができません。ここではプログラムの流れを下から上へ戻す方法、つまり**反復処理**を勉強します。



反復

プログラムの流れを下から上へ

while 文はある条件を満たす間、ずっと指定された区間の処理を繰り返す構文です。次のような書式になっています。

```
while <条件>
```

```
do
        <条件>に合致する時の処理
                        :
                        :
done
```

例えば、1 から 10 までカウントして出力スクリプトは以下のようになります。

```
count=1
while [ ${count} -le 10 ]
do
    echo ${count}
    count=`expr ${count} + 1`
done
```

だいぶプログラムらしくなってきましたね。実際にスクリプトを作り、実行してみて下さい。

スクリプトの解説をしましょう。

・まず count=1 で変数 count の値を 1 にセットしています。

・次に while 文で指定しているのは count の値が 10 以下という条件を満たしている間は do から done までの間を何回でも実行せよということです。if 文のときの条件指定と形式は同じです。

・do～done の間では、echo によって count の値が毎回出力されます。

・count=`expr ${count} + 1`は今までの count の値に 1 を足して、それを新しい count の値とせよということです。例えば count の値が今まで 3 だったなら、それがこの行で 4 に変化します。結果として、do～done の間が繰り返されるたびに count の値が 1 つずつ増え、10 までが表示されることになります。

ちょっと難しいでしょうか？でも何回も while 文を書いていくうちに感覚が身につきますから、頑張って課題をこなしましょう。

□課題：上の例にならって 200 から 10 まで 5 つずつカウントダウンするスクリプトを書きましょう(200,195,190,185,.....,20,15,10 のように数える)。

□課題：70+72+74+76+.....+198+200=?を while 文を使って計算するスクリプトを書きましょう。

□応用課題：数当てゲームをするスクリプトを書きましょう。具体的にはコンピュータが頭の中で（？）考えている数をできるだけ少ない回数で当てるゲームです。

まずコンピュータが頭の中で考える数字をどうやって決めるかですが、現在時刻の秒をそのまま使うことにしましょう。現在時刻の秒を出力するコマンドは date +%S です(man date コマンドで調べてみて下さい)。従って、

```
answer=`date +%S`
```

として考えている数を answer に格納しておけばいいでしょう。

次に一回ごとの試行でユーザが数を入力し、その数が answer より大きければ、My number is lower than that."と出力し、小さければ"My number is higher than that."と出力するようにします。一致していたら、"Exactly!!"と表示するようにします。

```
if [ guess -gt answer ]
then
    echo "My number is lower than that."
elif [ .....     .....]
then
    echo .....*
else
    echo .....
fi
```

※elif は最初の if の条件を満たさなかったとき、次に試される条件です。この条件に合った場合、*の部分が実行されます(実際に*の記号をスクリプト中に書いてはいけません！)。

そしてユーザが数を当てるまでこの処理が繰り返されるので、while 文を使ってユーザが数を当てるまで処理を繰り返すようにします。

```
guess = -1
answer=`date +%S`
while [ guess -ne answer ]
do
     echo "Guess my number:"
     read guess
     if [ .....]
          :
          :
     fi
done
```

※ -ne は等しくないを意味します。-gt はより大きいを、-lt はより小さいを意味します。

1 行目で guess の値を-1 として、最初は answer と値が絶対等しくならないようにします(guess の値と answer の値が万一等しくなると、while の条件に合わなくなってしまい、while 文の中が実行されません)。2 行目で answer の値を決めています。

あとは頑張って作ってみましょう。ここまで学んだ知識を生かせば作れるはずです。
何回の試行で数を当てたか、表示すると面白くなりそうです。出力例を以下に示します。

```
Guess my number:
20 ↵
My number is higher than that.

Guess my number:
30 ↵
My number is lower than that.

Guess my number:
25 ↵
Exactly!!
You got it in 3 tries!!
```

ちなみに、#が先頭にある行は(#!を除いて)無視されます。従って#に続けて注釈を入れるとスクリプトが理解しやすくなるでしょう。

```
read guess    # This line reads a number to guess from the user.
```

9.複数ファイルの処理

次は for 文について勉強しましょう。for 文は文字列の集合やファイルの集合に対してある決まった処理をするときに使います。書式は以下の通りです。

for 変数名 in 文字列の集合
do
　　処理
done

これで**変数名**を持つ変数に文字列の集合が順番に代入され、その後に**処理**が行われます。次の例をみてみましょう。

```
#!/bin/sh

for fruit in apple melon orange grape
do
　　echo “I like ${fruit}.”
done
```

実行結果

```
I like apple.
I like melon.
I like orange.
I like grape.
```

この場合、変数 fruit に apple,melon,orange,grape が次々と代入され、１回１回 echo によって出力されます。

文字列の集合のところにはファイルの集合をワイルドカード(*,?など)を使って記述することができます。

```
#!/bin/sh

for file_name in *.txt
do
　　echo ${file_name}
done
```

上の例ではカレントディレクトリで*.txt にあてはまるファイル名を出力します。

下の例では bacteria/*.seq にあてはまるファイル全てに対して一定の処理をしています。

```
#!/bin/sh

mkdir result
for file_name in bacteria/*.seq
do
    wc -l $file_name > result/`basename ${file_name}.res`
done
```

wc –l でファイルの行数を数え、その結果を処理したファイル名.res というファイルに格納しています。basename はコマンドの 1 つで、ディレクトリ部分の記述を省略するコマンドです。

□課題：上の例にならって、*.txt というファイルの最初の 1 行を各結果ファイルに格納するスクリプトを書きましょう。head コマンドなどが有効です。

10.コマンドライン

コマンドラインの引数について触れておきます。今まではシェルを実行するときに、
./シェル名
としていましたが、
./シェル名 引数１ 引数２ …
のように引数をつけることができます。その場合、引数 1 は$1 に、引数 2 は$2 に格納されます。

シェル名:param.sh

```
echo “Your first parameter is: $1”
```

実行のしかた:

```
./param.sh Hello
```

実行結果:

```
Your first parameter is: Hello
```

□課題：引数で指定したファイルの行数、最初の 1 行、最後の 1 行を表示するシェルを作りましょう。最後の 1 行は tail コマンドで表示できます。