プログラミング環境 第6回

久野 靖 *

1991.11.5

8 シェルとスクリプト

8.1 スクリプト

前回の練習問題で、えらく長〜い指令を作るやつを、全部端末から打ち込みながらやるのはなかなか大変でしたか?(newcsh の画面編集を使えばそうでもないか?) ところで、

```
指令 引数 ... <ファイル名
```

とやると、指令への入力が端末からでなくファイルから取られることはご存知の通り。じゃあ、この「指令」が sh や csh だったらどうなると思うか?

```
% echo 'ls -l' >ll
% cat ll
ls -l
% sh <ll
total 4
drwxr-xr-x  2 kuno          512 Oct 11 15:10 bin
-rw-r--r--  1 kuno           52 Oct 11 15:10 hello.p
-rw-r--r--  1 kuno            6 Oct 11 15:11 ll
drwxr-xr-x  2 kuno          512 Oct 11 15:10 work
%
```

つまり、「端末のかわりにファイルから指令を読みとって実行する」ことが起きる。ちょうど図1のような具合である。[1]

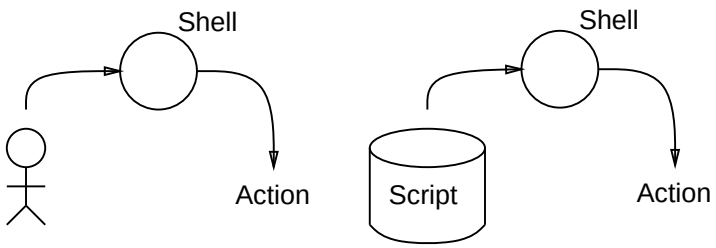

図 31: スクリプトの概念

さて、こういう機能はどういう役に立つと思うか?

- 複雑な指令を試す時、エディタで吟味しながら作れる。

*筑波大学経営システム科学専攻

[1]sh は他のフィルタ等と同様ファイル名を書くとそこから読みに行くので上のは sh ll とやっても同じ。また、sh でなく csh でもいいが、csh は実行が始まるまでがのろいし、スクリプト書きには sh の方が何かと便利なことが多い。

- そうして作った複雑な指令を再利用/反復実行できる。
- 大量の指令を生成してそれを実行させられる。

例えば最後のような使い方の例として、次のようなのも可能である。

```
ls | sed 's/^\(.*\)$/cp \1 \1.bak/' | sh
```

これは何をすると思うか? [2]

このように、「人間が 1 行ずつ打ち込む代わりに予めファイルに用意しておくもの」をスクリプト、と呼ぶ。上のは shell script だが、他にも sed script、ed script、awk script、などいろいろあり得る。

8.2 スクリプトの指令化、インタプリタ指定

実はさらに!このファイルを「実行可能」にしておくと、その名前を言うだけでこのスクリプトが実行できる。

```
% chmod +x ll
% ls -l ll
-rwxr-xr-x  1 kuno            6 Oct 11 15:11 ll
% ll
total 4
drwxr-xr-x  2 kuno          512 Oct 11 15:10 bin
-rw-r--r--  1 kuno           52 Oct 11 15:10 hello.p
-rwxr-xr-x  1 kuno            6 Oct 11 15:11 ll
drwxr-xr-x  2 kuno          512 Oct 11 15:10 work
%
```

つまり、一所懸命プログラム言語で書かなくても、指令を増やすのはわりと簡単なわけである。

ときに、ここで「質問」のある人はいませんか?「sh ll」では/bin/sh が指令を解釈してくれるのは明らかだったが、ただ「ll」ではどのシェルが解釈してくれるのだろう?実は、何も指定しないとやっぱり/bin/sh が使われるのだが、ファイルの 1 行目#!指令というのが入っていると、その指令が解釈に使われる。だから上のファイルはより厳密にはこう書いた方が曖昧さがなくてよい。

```
#!/bin/sh
ls -l
```

ところで、ここに/bin/csh などと書くことができるのはもちろんだが、次のようなのを書くとどうなると思うか?

```
#!/bin/cat
ls -l
```

さっそく見てみよう。

```
% cat ll1
#!/bin/cat
ls -l
% ll1
#!/bin/cat
ls -l
%
```

つまり!あるファイル、たとえば t の 1 行目に!#X ... というのが入っていた場合、それを実行可能にして X args... という風に名前で呼び出した時には

[2] ところで一気にパイプラインで sh に食べさせるのが恐ければもちろん、一旦できたものをファイルにとって、ゆっくりチェックしてから実行するといい。また、例えば「このディレクトリのファイルを全部消す」というスクリプトを一旦作ってから、消したくないファイルの分だけその行を取り除く、という使い方も便利なものである。

```
X ... t args...
```

という形で指令 X を呼び出したのと同じことになる。だから例えば次のようなものも可能だ。

```
% cat ll2
#!/usr/bin/sed -f
s/  */ /g
s/A/a/g
% ps | ll2
 PID TT STaT TIME COMMaND
28874 p4 S 0:01 -usr/new/csh (csh)
29097 p4 R 0:00 ps
29098 p4 S 0:00 sed -f ll2
%
```

おっと、話しが cat スクリプト (?) や sed スクリプトに脱線してしまった。シェルスクリプトの話に戻ろう。

8.3 自分用のコマンドディレクトリ

ところで、この「ll」などの指令はこのディレクトリから他へいってしまうと使えない。(なぜか?ご存じですよね?) そこで、自分用のこういう指令を入れるディレクトリを用意することにする。つまり、

- mkdir ~/bin で、自分の指令用ディレクトリを作る。
- 自分のホームディレクトリにある.cshrc を編集して、

  set path=(. ~/**bin** /usr/{bin/X11,new,ucb,bin} /bin)

  のように自分の指令用ディレクトリを path に入れる。
- source .cshrc; rehash する。

これで、自分のホームディレクトリの直下にある bin というディレクトリが path に入ったので、ここに置いてある指令 (つまり、実行可能なファイル) はどこにいても使えることになる。

```
% mv ll ~/bin
% rehash
% ll
total 3
drwxr-xr-x  2 kuno          512 Oct 11 15:10 bin
-rw-r--r--  1 kuno           52 Oct 11 15:10 hello.p
drwxr-xr-x  2 kuno          512 Oct 11 15:10 work
%
```

8.4 コマンド引数、シェル変数

ところで、「ls ファイル名」というと指定したファイルのみに表示を限定できるのだったが、上の ll もそうしたくないか?実は簡単である。シェルではシェル変数というのが使えることは既にやったが、シェルスクリプトを指令として起動した時には$1, $2, ..., $9 という特別なシェル変数が使えて、そこに 1 番目、2 番目、... の引数が入るようになっている。従って次のようにすればいい。

```
% cat ~/bin/ll
#!/bin/sh
ls -l $1
% ll hello.p
-rw-r--r--  1 kuno           52 Oct 11 15:10 hello.p
%
```

では、ファイル名を沢山書きたい時は?ls の行を

```
ls -l $1 $2 $3 $4 $5 $6 $7 $8 $9
```

にすることはすぐ考え付くが、あまりスマートでない。実は$* で「引数全部」という意味になるので、次のようにすればよい。

```
ls -l $*
```

さて、シェル変数の代入法はもうやったが、実は/bin/sh では csh と違って「set」は使わず、

```
変数名=値
```

による。ここで=の両側に空白があってはいけない。[3] 変数の値の参照はこれまで通り、$変数名による。例えば次のは 2 つ数を与えるとその和を表示する、という (暗算が弱い人には役に立つ) 例である。

```
% cat add
#!/bin/sh
sum=‘expr $1 + $2‘
echo "$1 + $2 = $sum."
% add 3 4
3 + 4 = 7.
%
```

8.5 /bin/sh の制御構造

これまでの所、シェルスクリプトについては「上から順に実行する」方法しか知らなかったわけだが、もちろん!if とか while などがちゃんとある。以下でこれらについて一通り説明しておこう。

8.5.1 for

Sh の/tt for は csh の foreach の同類で、並びのなかの値を一つずつ変数に入れながらループする、というものであるが、構文がちょっと異なる。

```
for 変数名 in 値の並び
do
  ...
  ...
done
```

ただし... の所には指令が何行でも書ける。for は引数が沢山書けるスクリプトを処理するのに便利である。例えば次のような具合である。

```
% cat fortest
#!/bin/sh
n=1
for x in $*
do
  echo "$n : $x"
  n=‘expr $n + 1‘
done
% fortest who are you
1 : who
2 : are
3 : you
%
```

[3]なぜだと思うか?ヒント:「echo = a」と「echo=a」の違いはなにか?

8.5.2 while

While は Pascal などの場合と同様、条件が成り立つまでループする構文である。

```
while 条件
do
  ...
  ...
done
```

ところで、この「条件」というのは何かというと、実は任意の指令である。そしてまたまた実は、Unix では指令 (プログラム) は全て終る時に小さな数値 (完了コード) を返す。この値が 0 の場合は「成功」、0 以外の場合は「失敗」と見なされる。例えば grep 属などは探していた文字列が見つかると「成功」、見つからないと「失敗」する (普段は完了コードを気にしないので分からないが) し、-v オプションだとその反対になるので、例えば次のようなこともできる。

```
% date
Fri Nov  2 14:45:12 JST 1990
% cat whiletest
#!/bin/sh
while date | egrep -v ' 14:46' >/dev/null
do
  echo waiting...
  sleep 30
done
% whiletest
waiting...      ← 14:46 になるまで 30 秒ごとにこう打ちだし、
waiting...
%               ← 14:46 になると (30 秒の誤差はあるが) 終る。
```

だから、「いたずらしたい相手が login するまで待っていて、login してきたらすかさず悪さをする」なんていうスクリプトがこれで書ける。

8.5.3 if

If 文もごく「普通の」ものであるが、ただし Pascal などと違うのは始めから if-then-else if-then-の構造が組み込まれている点である。従って、if 文の終りは必ず if がないといけない代わり、begin と end が要らない。

```
if 条件
then
  ...
elif 条件       ← elif 部は何個あっても、なくてもよい
then
  ...
else            ← else 部も、なくてもよい
  ...
fi              ← かならず最後に fi があること
```

ここで「条件」は while の場合とまったく同じである。例えば次のような具合に使える。

```
% cat iftest
#!/bin/sh
if egrep "^$1\$" /usr/dict/words >/dev/null
then
  echo 'Oh, I know ther word.'
else
  echo "I don't know ther word $1."
fi
% iftest simplify
```

```
Oh, I know ther word.
% iftest simpl
I don't know ther word simpl.
% iftest simple
Oh, I know ther word.
%
```

8.5.4 test 指令

しかしながら、やっぱり「x ¿ y」とかそういう類の条件が書きたいこともあるし、またファイルの有無なども調べたい。そういうことのために「test」という指令が用意されている (ので、自分のプログラムを test という名前にはしない方がよい)。さらに、test には "[" という別名もあって (なんと/bin/[というファイルがある)、これを使うと

```
if [ 条件の内容... ]
```

というもっともらしい書き方ができる。条件の内容はマニュアルページ参照のこと。if といっしょに使った例を示そう。

```
% cat iftest1
#!/bin/sh
if [ $1 -lt 150 ]
then
  echo You are small.
elif [ $1 -gt 190 ]
then
  echo You are tall.
else
  echo You are quite ordinary.
fi
% iftest1 160
You are quite ordinary.
% iftest1 200
You are tall.
%
```

もちろん while といっしょに使ってもよい。

```
% cat whiletest1
#!/bin/sh
str=$1
while [ -n "$str" ]
do
  echo $str
  str=‘echo $str | sed 's/^.//'‘
done
% whiletest1 abcde
abcde
bcde
cde
de
e
%
```

8.5.5 case

ところで、sh にはファイル名のパターンを扱う機能があるのだから、パターンマッチによって条件分岐する機能があってもおかしくない。それが case である:

```
case 語
in
パターン|...|パターン) ... ;;  ← ;; までにいくつ文があってもよい
パターン|...|パターン) ... ;;
...
esac
```

ところでこのパターンは (上の由来からも分かるように)grep 属のパターンではなくてファイル名マッチングのパターンである。「それ以外なら」というのがないなあ、と思われたかもしれないが、それは要らない。というのは:

```
% cat casetest
#!/bin/sh
case $1
in
Y|y|[Yy]es|[Oo]k) echo You said yes.;;
*)                echo You didn\'t say yes.;;
esac
% casetest yes
You said yes.
% casetest oui
You didn't say yes.
%
```

ところで、test 指令でも「ある文字列と別の文字列が等しいかどうか」を調べることができるのでその場に応じて適切なのを使うとよい。[4]

8.6 シェルスクリプトの位置付け

さーて、このようにシェルスクリプトでは変数も、式の計算機能も、また普通の言語にあるような制御構造もすべて備わっている。(Q. 上に挙げなかった、しかし重要な制御構造は何か?じつはその制御構造も備わっているが、お分かりか?) [5] 従ってどんなプログラムでもシェルスクリプトで書いてしまうことができるわけである。

じゃあ C などの言語はいらないのか?そこはやっぱり、まともな言語の方が実行が高速だし、大きなプログラムでもそれなりに美しく書けるような仕組みが完備している (してない言語もあるが)。

一方、シェルで書いたプログラムというのは次のような利点がある。

- すぐ実行して見られるので手軽に作り始められる
- 色々な指令を下請けに呼び出すのが簡単
- 引数、パターン、その他便利な機能がいろいろ備わっている

そこで、手軽に作って増やしていくうちにいつのまにか巨大なシェルスクリプトを書いてしまう、という人もいる。ちなみに参考書 (「Unix プログラミング環境」のことだよ) では、そうやって最初はシェルで手軽に始め、大きく遅くなってきたら C で書こう、というのを勧めている。その他シェルプログラミングについて異常に詳しく書かれているから、その部分だけでも参考書を読んで見られることをお勧めします。

[4] 参考書には「case が使えるときは if+test より速いので使うと良い」とあるが、実は最近のシェルでは test は組み込み指令なので特に遅いということはない。

[5] 実は最近のシェルでは関数 (サブルーチン) も定義できるのではありました。でもそれがなくても「名前を呼ぶと起動される」というのは既に使っているわけだ?)

A　告示、演習および課題

A.1　告示

A.2　8 節の演習

ともあれ、自分用のコマンドディレクトリを作っていないひとは作っておくこと。何かと便利ですから。

8-1. 現在位置にあるファイルを全部大文字のファイル名に直すというシェルスクリプトを自動生成し、実行してみよ。

8-2. 例にあった「2 つの数を足す」というスクリプトを拡張して、引数として指定された数をすべて加える、というのに直してみよ。[6]

8-3. 10 回'Hello.' と打ち出すスクリプトを、「hello10」という名前で作ってみよ。また、打ち出す回数、打ち出すメッセージともオプションで指定できるようにしてみよ。できればオプションがないときは標準のものが使われるようにして欲しい。[7][8]

8-5. 「`rep 指令 引数...`」とやると、指定した指令を 10 秒ごとに繰り返し実行する、という指令を作ってみよ。さらに、「`rep -5 指令 引数...`」とやると 5 秒ごと、という風に時間も指定できるようにしてみよ。もちろん、指定がなければもとの 10 秒ごと、ということにする。[9]

8-6. 「階乗」を計算するシェルコマンドを作成せよ。できれば再帰呼び出し版とループ版の両方を作り、その挙動がどう異なるか比較してみよ。[10]

8-7. 「指定した人が login するまで待ち構えていて、何かいたずらをする」スクリプトを実際に作って動かしてみよ。実験台は自分でやるか、他人の場合は了解を求めてから。どんないたずらかって?それは自分で考えるのが楽しいのでした。

A.3　6 回目の課題

本日の課題から報告を提出する場合は、上記 8-1〜8-7 のうちから最低 2 つ以上選択して下さい。

[6] ヒント:やっぱり sed を使って全部の間に「+」をはさむのでしょうねえ。

[7] if で変数がからっぽかどうか調べられますよねえ。

[8] あるいは、単に`$`変数名と書く代わりに`${` 変数名`-`値 `}` というのを書くと、指定した変数が空ならその値の代わりに「値」が使われる、という機能を利用してもいい。

[9] オプションを処理する場合には shift という指令が役に立つかも知れない。これは`$1=$2;$2=$3;$3=$4;...` をまとめて行なう、というものである。

[10] さらに、再帰呼び出し版もシェルコマンド自身を再帰的に呼び出す、というのとシェル内部の関数呼び出しを使うのとがある。よほど好きな人は両方やってみられたい。