誰にでも使える csh 講座
第 2 回
「再び 3 回まわってワン」

安岡孝一

yasuoka : root さん、 root さん。

root : 何だい？

yasuoka : 今日は C シェルでの if とか繰り返しとかを、教えてくれるっていう約束だったじゃないですか。

root : そんな約束したっけ？ まあ、いいか。

```
% cd ~/bin（ぽこ）
% ls（ぽこ）
bow     dow     hb      users   where   who     who~
% cat bow（ぽこ）
#! /bin/sh
# "bow" Version 1.1

TIME=${1-3}
while test "$TIME" -gt 0
do echo BOWWOW
   TIME=‘expr $TIME - 1‘
done

exit 0
% ■
```

じゃ簡単なところで、これを C シェルのスクリプトに書き直してみるか。

yasuoka : はい。

```
% mv bow bow~（ぽこ）
% ■
```

```
#! /bin/csh
# "bow" Version 2.0

if ($#argv) then
  set TIME=$argv[1]
else
  set TIME=3
endif

while ($TIME > 0)
  echo BOWWOW
  @ TIME--
end

exit (0)
```

（間）

yasuoka : ええっと。

```
% chmod 755 bow（ぽこ）
% rehash（ぽこ）
% bow 7（ぽこ）
BOWWOW
BOWWOW
BOWWOW
BOWWOW
BOWWOW
BOWWOW
BOWWOW
% bow（ぽこ）
BOWWOW
BOWWOW
BOWWOW
% ■
```

よし、できた。でも、 if の書き方がシェルとは全然違うみたいですね。

root : そう。実はここがすごくややこしいんだな。シェルと C シェルの違いを、きっちりと覚えないとね。

```
if（式）then
  コマンド列
else
  コマンド列
endif
```

式を評価した結果が 0 以外ならば then のコマンド列を、さもなくば else のコマンド列を実行する。 else 節はなくてもよいが、 then の後、 else の前後、 endif の前後で必ず改行。

式は以下の通り。なお真偽判定については、真は 1、偽は 0 を値として返す。

文字列 == 文字列	2 つの文字列が等しいとき真
文字列 != 文字列	2 つの文字列が等しくないとき真
文字列 =~ 文字列	== に同じ、ただし右文字列にワイルドカード使用可
文字列 !~ 文字列	!= に同じ、ただし右文字列にワイルドカード使用可
数	数（整数のみ有効）
式 + 式	2 式の和
式 - 式	左式から右式を引いた差
式 * 式	2 式の積
式 / 式	左式を右式で割った商（小数部切り捨て）
式 % 式	左式を右式で割った余り
式 << 式	左式を右式ビット左シフト
式 >> 式	左式を右式ビット右シフト
~ 式	1 の補数
式 & 式	2 式の論理積
式 ^ 式	2 式の排他的論理和
式 | 式	2 式の論理和
式 > 式	左式が右式より大きいとき真
式 >= 式	左式が右式以上のとき真
式 < 式	左式が右式未満のとき真
式 <= 式	左式が右式以下のとき真
-e ファイル名	ファイルが存在するとき真
-z ファイル名	ファイルが存在して、そのサイズが 0 のとき真
-d ファイル名	ファイルが存在して、ディレクトリのとき真
-f ファイル名	ファイルが存在して、ディレクトリでないとき真
-r ファイル名	ファイルが存在して、 readable のとき真
-w ファイル名	ファイルが存在して、 writable のとき真
-x ファイル名	ファイルが存在して、 executable のとき真
-o ファイル名	ファイルが存在して、その持ち主であるとき真
! 式	式が 0 のとき真
式 && 式	2 式とも 0 でないとき真
式 || 式	2 式のどちらかが 0 でないとき真
{ コマンド }	エグジットステイタスが 0 のとき真
（式）	優先度を変える

yasuoka ： シェルのエグジットステイタスとは違って、足し算とかが条件式の中に使えるんですね。

root ： 各演算子の優先度も決まってるけど、ちょっと覚えきれないだろうな。

yasuoka ： 大まかなところを教えて下さい。

root ： 算術演算子では~ が最優先で、 * と / と %、 + と -、 <<と>>、 &、^、 | の順番。条件演算子では ! が最優先で、> と>= と <と <=、 == と != と =~ と !~、 &&、 || の順番。お互いの関係では、 ! が~ と * の間ってことと、 & が == より低いことに注意すればいいだろう。でもまあ、わかんない時には ( ) をどんどん使うべきだね。

yasuoka ： はい。で、この 4 行目の if（$#argv）then ってのは、どういうときに真になるんですか？

root ： argv の要素数が 0 でないとき、つまりコマンドにパラメータが少なくとも 1 個与えられたときだ。

yasuoka ： ああ、それでパラメータがなかったときには、 else の set TIME=3 を実行するわけですね。それから 10 行目ですけど、これも while のカッコの中は式ですか？

root ： よくわかったね。これもシェルとは違うから、気を付けないとね。

```
while（式）
  コマンド列
end
```

式を評価した結果が 0 以外である間、コマンド列を繰り返し実行する。 while（式）の後、 end の前後で必ず改行。式については if に同じ。

yasuoka ： で、 11 行目で BOWWOW を表示するっと。この 12 行目の @ TIME--って何ですか？

root ： TIME をデクリメントしてるんだよ。

yasuoka ： デクリメントって？
root ： 1 減らすこと。逆に 1 増やすのはインクリメントだ。

```
@ 変数=(式)
        変数に式の値を代入する。
@ 変数+=(式)
        変数に式の値を加える。
@ 変数-=(式)
        変数から式の値を引く。
@ 変数*=(式)
        変数を式の値倍する。
@ 変数/=(式)
        変数を式の値で割る。
@ 変数%=(式)
        変数を式の値で割った余りを変数に代入する。
@ 変数++
        変数に 1 加える。
@ 変数--
        変数を 1 減らす。
いずれも変数は配列の 1 要素でもよい。式については if に同じ。
```

yasuoka ： 演算に expr を使わなくってもいいんですね。
root ： まあ、そうかな。ここらへんは C シェルならでは、だね。
yasuoka ： で、 TIME の回数だけ繰り返して、最後に exit (0) で終了。 C シェルの exit にはカッコがいるんですか？
root ： うん。しかも中には式が書ける。

```
exit (式)
        式の値をエグジットステイタスとして、プログラムを終了する。式については if に同じ。
exit
        $status をエグジットステイタスとして、プログラムを終了する。
```

yasuoka ： じゃあ、 root さん、 root さん。
root ： 何だい？
yasuoka ： この数当てゲームのシェル・スクリプトも

```
% cat hb (ぽこ)
#! /bin/sh
# "hb" Version 1.1

if tty -s
then :
else echo hb: standard input is not a tty. >&2
     exit 1
fi

ANSWER=`expr $$ % 100 + 1`
INPUT=0

until [ "$INPUT" -eq $ANSWER ]
do echo -n '? '
   read INPUT
   if [ "$INPUT" -gt $ANSWER ]
   then echo big.
   elif [ "$INPUT" -lt $ANSWER ]
   then echo small.
   fi
done

echo correct.
exit 0
% 
```

C シェルに直して下さいよ。
root ： うーん。

```
#! /bin/csh
# "hb" Version 2.0

if (! { tty -s }) then
```

```
  echo hb: standard input is not a tty.
  exit (1)
endif

@ ANSWER=($$ % 100 + 1)
set INPUT=0

while ($INPUT != $ANSWER)
  echo -n '? '
  set INPUT=$<
  if ($INPUT > $ANSWER) then
    echo big.
  else if ($INPUT < $ANSWER) then
    echo small.
  endif
end

echo correct.
exit (0)
```

root : 完全な置き換えは無理だなぁ。何せ、シェルよりも C シェルの方が能力が低いからね。

yasuoka : え、どこがですか？

root : `#! /bin/sh` の方では `hb: standard input is not a tty` がエラー出力に出てたろ。でもこれでは、そうはなってない。

yasuoka : エラー出力への切替えは出来ないんですか？

root : 出来ない。一応、C シェルでのパイプとリダイレクトを教えとこうか。

コマンド `|` コマンド
　　前のコマンドの標準出力を、後のコマンドの標準入力に繋ぐ。
コマンド `|&` コマンド
　　前のコマンドの標準出力とエラー出力を、後のコマンドの標準入力に繋ぐ。
コマンド `>` ファイル名
　　コマンドの標準出力をファイルに書き出す。
コマンド `>&` ファイル名
　　コマンドの標準出力とエラー出力をファイルに書き出す。
コマンド `>>` ファイル名
　　コマンドの標準出力をファイルに書き加える。
コマンド `>>&` ファイル名
　　コマンドの標準出力とエラー出力をファイルに書き加える。
コマンド `<` ファイル名
　　ファイルを読み出してコマンドの標準入力とする。
コマンド `<<` '区切りの文字列'
文字列
文字列
…
'区切りの文字列'
　　区切りの文字列にはさまれた文字列を、コマンドの標準入力とする。区切りの文字列には何を用いてもかまわない。

yasuoka : 標準出力とエラー出力を混ぜることは出来るんですね。

root : でも、切替えは出来ない。

yasuoka : うーん。

root : ま、それはさておいて。

yasuoka : はい。

root : `#! /bin/sh` での `elif` だけど、C シェルではそういう特殊なものはないんだ。改行が `else` の後に入るかどうかで、`endif` の数が変わる。

```
if (A) then
  B
else
  if (C) then
    D
  else
    E
  endif
endif
```

は右のように簡略化できる

```
if (A) then
  B
else if (C) then
  D
else
  E
endif
```

yasuoka : それが 17 行目ですね。

root : そう。それから 1 行入力には `$<`を使う。

yasuoka ： 14 行目ですね。エンドオブファイルが入力された時には、どうなるんですか？

root ： どうもならないよ。空行が入力されたのと同じ。

yasuoka ： じゃ、シェルの `while read` みたいなのは？

root ： 使えない。

yasuoka ： 残念。そこも能力が低いんですね。

root ： それからシェルの `until` は C シェルにはないから、条件を逆にして `while` を使うことになる。

yasuoka ： シェルの `for` にあたるのはないんですか？

root ： あるよ。 `foreach` だ。

> `foreach` 変数（文字列 文字列 …）
> 　コマンド列
> `end`
> 　　変数に文字列を順々に代入し、文字列の個数だけコマンド列を実行する。
> 　　`foreach` 文の後、 `end` の前後で必ず改行。

yasuoka ： すると条件分岐もループもだいたいシェルと同じように使えるんですね。

root ： まあね。

yasuoka ： あ、そうだ。ループからの飛び出しとかは？

root ： `continue` とか `break` とかかい？

yasuoka ： はい。

root ： あるにはあるんだけどね。

> `continue`
> 　　ループの `end` にジャンプする。ループを繰り返す条件が成立していれば、再度ループを繰り返す。
> `break`
> 　　ループから飛び出す。

yasuoka ： あれ、二重ループは飛び出せないんですか？

root ： いや、 `break ; break` と 1 行に書けば飛び出せるよ。

yasuoka ： え？ じゃ、シェルでの `continue 2` は `continue ; continue` ですか？

root ： いいや、 `break ; continue` だよ。ループを 1 つ飛び出した後 `continue` だからね。 C シェルでは `break` と同じ行にコマンドを書いておくと、それを実行しながら `break` するんだ。

yasuoka ： わー、気持ち悪ーい。

root ： 気持ち悪いだろう。だから `break ; break ; break` が三重ループの飛び出しになるんだ。

yasuoka ： `break ; echo OUT` とかすると、 `break` しながら `OUT` を出力するんですか？

root ： そういうことになる。

yasuoka ： どうしてそんなに気持ち悪いんでしょう？

root ： 腐ってるからかな。ヴァージョンによって、非互換なところも多いしね。

yasuoka ： どういうところですか？

root ： 例えば `set` とか `@` とかの代入で、 `=` の前後に空白を入れなければいけない C シェルもあるし、そうかと思うと入れてはいけない C シェルもある。あるいは、変数の名前に数字を使えない C シェルとかね。

yasuoka ： そうなんですか。

root ： これでわかったろ、 C シェルのスクリプトなんか書くもんじゃないって。

yasuoka ： じゃあ、どうして皆んな C シェルを使ってるんですか？ シェルの方がよっぽどきれいなのに。

root ： そりゃ `history` と `alias` だな。シェルにはどっちもないからね。そういう対話的な部分については、 C シェルの方が遥かに優ってる。

yasuoka ： `history`ってそんなに便利ですか？

root ： 便利、便利。使い込めば込むほどね。

yasuoka ： そうかなぁ。

root ： よし、次は `history` についてみっちりやろう。 C シェルの名誉挽回だ。

yasuoka ： えー。

root ： もう決めたからね。じゃあまた今度。