誰にでも使える csh 講座
第 1 回
「.logout は #! /bin/sh じゃない」

安岡孝一

yasuoka : root さん、root さん！

root : 何だい？

yasuoka : logout できなくなっちゃったんです。

```
% logout （ぽこ）
set: Syntax error.
% 
```

root : ん？

```
% alias logout （ぽこ）
% 
```

う。どうなってるんだろ。

```
% cat ~/.logout （ぽこ）
#! /bin/sh

set ‘date | tr ":" " "‘
case $4 in
0[5-9]|10|11) WHEN=morning ;;
1[2-6]) WHEN=afternoon ;;
1[7-9]|20|21) WHEN=evening ;;
*) WHEN=night ;;
esac
echo Good $WHEN, I was ‘echo $USER | tr a-z A-Z‘.
exit 0
% 
```

あ、これだ。

yasuoka : どれですか？

root : .logout の文法が間違ってる。

yasuoka : ええ？ でも

```
% ~/.logout （ぽこ）
Good afternoon, I was YASUOKA.
% 
```

ほら、ちゃんと動きますよ。

root : じゃ、source ~/.logout としてごらん。

yasuoka : source ~/.logout ですか。

```
% source ~/.logout （ぽこ）
set: Syntax error.
% 
```

あれ？ どうしてですか？

root : source で実行されるのは C シェルのスクリプトなんだよ。

yasuoka : C シェルって、login した時にコマンドインタプリタとして立ち上がる、あの csh ですか？

root : そう、あの C シェル。

yasuoka : でもそれが.logout とどう関係あるんですか？

root : おおあり。C シェルは.logout を source で実行するんだ。

yasuoka : logout の時に？

root : そう。

yasuoka : じゃ、.logout は C シェルのスクリプトじゃないといけないんですね。

root : .cshrc や.login もね。

yasuoka : で、C シェルのスクリプトは、前にやった #! /bin/sh のシェル・スクリプトとは文法が違うんですか？

root : うん。まずはこの.logout を C シェルのスクリプトに書き直してみよう。

```
set DATE=(‘date | tr ":" " "‘)
switch ($DATE[4])
case 0[5-9]:
case 10:
case 11:
  set WHEN=morning
  breaksw
case 1[2-6]:
  set WHEN=afternoon
```

```
  breaksw
case 1[7-9]:
case 20:
case 21:
  set WHEN=evening
  breaksw
default:
  set WHEN=night
  breaksw
endsw
echo Good $WHEN, I was ‘echo $USER | tr a-z A-Z‘.
```

（間）

yasuoka ： できました。

```
% logout （ぽこ）
Good afternoon, I was YASUOKA.

login: 
```

うまくいってるみたい。

root ： source で実行する C シェルのスクリプトは executable でなくていいよ。

yasuoka ： あ、そうなんですか。じゃあ。

```
login: yasuoka （ぽこ）
Password: （ぱたぱたぱたぱたぱたぱたぽこ）
Last login: Mon Mar  5 14:09:02 on console
MoonOS 1.1 (GINKAKU) Mon Jan 1 11:11:11 GMT+0900 1990
########################################################

         'Twas brillig, and the slithy toves
           Did gyre and gimble in the wabe:

########################################################
% ls -l .logout （ぽこ）
-rwxr-xr-x  1 yasuoka       304 Mar  5 16:41 .logout
% chmod 644 .logout （ぽこ）
% ls -l .logout （ぽこ）
-rw-r--r--  1 yasuoka       304 Mar  5 16:41 .logout
% 
```

で、 root さん。

root ： 何だい？

yasuoka ： C シェルの文法を教えて下さい。

root ： うーん、できれば知らない方がいいんだけどな。シェルと混乱するし。でもまあ、教えようか。

yasuoka ： はい。

root ： まずは変数の代入から。

`set` 変数=文字列
: 変数への代入。

`set` 配列=(文字列 文字列 …)
: 配列への代入。最初の文字列から順に、配列[1] 配列[2] …に代入される。

`set` 配列[数]=文字列
: 配列の第「数」番目の要素への代入。第「数」番目の要素は、すでにセットされていなければならない。

`set`
: セットされている全ての変数の情報を標準出力に出力する。

`unset` 変数
: 変数をセットされていない状態に戻す。変数は配列でもよい。

yasuoka ： へえ。配列が使えるんですか。

root ： うん。このスクリプトでも DATE は配列として使ってる。 1 行目で代入してるだろ。

yasuoka ： はい。

root ： それから変数の読み出しだな。

$変数
: 変数の読み出し。

$配列
: 配列の全要素の読み出し。

$配列[数]
　　配列の「数」番目の要素の読み出し。
$配列[数-数]
　　配列の左「数」番目から右「数」番目までの読み出し。左数を省略した場合には 1 が、右数を省略した場合には配列の要素数が仮定される。
$#配列
　　配列の要素数。
$?変数
　　変数がセットされていれば 1、さもなくば 0。変数は配列でもよい。

yasuoka ： $? はエグジットステイタスじゃないんですね。でも、変数がセットされてるかどうかで 0/1 が返ってくるのは、ちょっと気持ち悪いですね。
root ： でも無いと困るよ。
yasuoka ： え、 ${変数-文字列} とかを使えば、たいていの事はできるでしょう？
root ： それはシェル。 C シェルには、そういう変数読み出しは無い。
yasuoka ： えー、そうなんですかぁ。
root ： ま、 basename とかに似たのはあるけどね。
yasuoka ： 何ですか、その basename って？

basename 文字列 文字列
　　左文字列中の最後の / 以前を削除し、末尾が右文字列と一致したならばその部分も削除して、標準出力に出力する。右文字列は省略してもよい。
dirname 文字列
　　System V のみ。文字列中の最後の / 以降を削除し、標準出力に出力する。ただし文字列が / を含んでいない場合は、. を出力する。

root ： ちょっと試してみせようか？

```
% basename /lib/libc.a （ぽこ）
libc.a
% basename /lib/libc.a .a （ぽこ）
libc
% 
```

yasuoka ： ははーん。で、これに似た C シェルでの変数の読み出し方って？

$変数:t
　　変数を、最後の / 以前を削除しながら読み出す。
$変数:r
　　変数を、最後の . 以後を削除しながら読み出す。
$変数:h
　　変数を、最後の / 以後を削除しながら読み出す。
いずれも変数は、配列や配列の要素でもよい。ただし、複数の要素に対して適用する場合には、:t、:r、:h の代わりにそれぞれ:gt、:gr、:gh を用いる。

root ： basename とは微妙に違うけどね。ちょっと試してみよう。

```
% set temp=/lib/libc.a （ぽこ）
% echo $temp （ぽこ）
/lib/libc.a
% echo $temp:t （ぽこ）
libc.a
% echo $temp:r （ぽこ）
/lib/libc
% echo $temp:h （ぽこ）
/lib
% unset temp （ぽこ）
% 
```

yasuoka ： ふーん。:t と:r は組み合わせられないんですか？
root ： 組み合わせられない。代入が 1 回必要だ。
yasuoka ： そうですか。
root ： さて、だいぶ話がそれちゃったな。元の .logout に戻ろうか。

```
% cat .logout （ぽこ）
set DATE=(‘date | tr ":" " "‘)
switch ($DATE[4])
case 0[5-9]:
case 10:
case 11:
  set WHEN=morning
  breaksw
case 1[2-6]:
  set WHEN=afternoon
```

```
    breaksw
  case 1[7-9]:
  case 20:
  case 21:
    set WHEN=evening
    breaksw
  default:
    set WHEN=night
    breaksw
  endsw
  echo Good $WHEN, I was `echo $USER | tr a-z A-Z`.
  % 
```

あと、シェルでの case にあたるのが switch だ。

```
switch (文字列)
case 文字列:
  コマンド列
  breaksw
case 文字列:
  コマンド列
  breaksw
…
default:
  コマンド列
  breaksw
endsw
```

switch の文字列と、 case の文字列のマッチングをおこない、マッチしたらその直後のコマンド列を実行する。どの case ともマッチしなかった場合は、 default のコマンド列を実行する。

root : case の後の文字列にはワイルドカードも使えるよ。それから、 2 つ以上の文字列とのマッチングには、 case を複数書けばいい。

yasuoka : C シェルのスクリプトは source でしか実行できないんですか？

root : いや、コマンドとしても実行できる。

yasuoka : え、どうするんですか？

root : シェル・スクリプトでは 1 行目に #! /bin/shって書いたろ。それと同じで、 1 行目に #! /bin/cshって書いた executable なスクリプトを、~/bin に置けばいい。

yasuoka : え、ほんとですか？

root : あんまり勧められないけどね。まあ試しに

```
% cd ~/bin （ぽこ）
% ls （ぽこ）
bow     dow     hb      users   where   who
% cat who （ぽこ）
#! /bin/sh
# "who" Version 2.3

case "$1 $2" in
"am i") HEAD="You are" ;;
"are you") HEAD="I am" ;;
"is he") HEAD="He is" ;;
"is she") HEAD="She isn't" ;;
*) /bin/who $*
   exit 0 ;;
esac

echo -n $HEAD" "
echo $USER"." | tr a-z A-Z
exit 0
% 
```

を C シェルのスクリプトに書き換えてみようか？

yasuoka : はい。

```
% mv who who~ （ぽこ）
% 
```

```
#! /bin/csh
# "who" Version 3.0
```

```
switch ("$argv")
case "am i":
  set HEAD="You are"
  breaksw
case "are you":
  set HEAD="I am"
  breaksw
case "is he":
  set HEAD="He is"
  breaksw
case "is she":
  set HEAD="She isn't"
  breaksw
default:
  /bin/who $argv
  exit (0)
  breaksw
endsw

echo -n $HEAD" "
echo $USER"." | tr a-z A-Z
exit (0)
```

（間）

yasuoka ： それからっと。

```
% chmod 755 who（ぽこ）
% rehash（ぽこ）
% 
```

これでいいんですか？

root ： OK、OK。実行してみてごらん。

yasuoka ： はい。

```
% who（ぽこ）
yasuoka  console Mar  5 16:43
hama     ttyp2   Mar  5 09:12   (ttyp1)
% who am i（ぽこ）
You are YASUOKA.
% 
```

心なしか、立ち上がりが遅いですね。

root ： うん、`#! /bin/csh` のコマンドは、最初に`.cshrc` を実行してから立ち上がるからね。`.cshrc` が長いほど、立ち上がりが遅くなる。

yasuoka ： じゃあ`.cshrc` で定義してる `alias` なんかが、スクリプトの中でも別名として使えるんですか？

root ： そうだよ。

yasuoka ： `source` で実行した時も？

root ： `source` での実行はまたちょっと違って、スクリプトをキーボードから打ち込んだのと同じような動作になる。だから `source` で実行するスクリプトの中に `alias` とか `cd` とかがあると、それは外にも影響が出る。

yasuoka ： ふーん。で、スクリプトに戻りますけど、この `$argv`ってのがコマンドのパラメータなんですか？

root ： そう。配列だから `$argv[1]` とか書くと第 1 パラメータになる。ついでだから、C シェルでの基本的な変数の読み出しを教えてあげよう。

`$0`	実行コマンド
`$$`	プロセス番号
`$<`	標準入力から 1 行入力
`$argv`	パラメータ（配列）
`$status`	直前に実行したコマンドからのエグジットステイタス
`$home`	ホームディレクトリ
`$path`	コマンドパス（配列）
`$cdpath`	cd パス（配列）
`$prompt`	コマンド待ちプロンプト
`$history`	`history` で残しておくコマンドの回数
`$savehist`	次のセッションまで残しておくコマンドの回数

root ： あとはだいたいわかるだろう？

yasuoka ： ええ。でも `exit` にはカッコがいるんですね。

root ： これはちょっと細かい話になるな。今日はもう時間がないから、それについてはまた今度にしてくれるかい？

yasuoka ： はい。どうもありがとうございました。