誰にでも書ける #! /bin/sh 講座
第 4 回
「当たらずとも遠からず」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : シェルで作った数当てゲームを takahash くんにもらったんですけど、よくわからないところがあるんです。
root : どれどれ、見せてごらん。
yasuoka : はい。

```
% cat hb （ぽこ）
#! /bin/sh
# "hb" Version 1.0

if tty > /dev/null
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
% 
```

root : ちょっと、やってみてもいいかい？
yasuoka : ええ、どうぞ。
root : じゃ。

```
% hb （ぽこ）
? 
```

値はいくらからいくらまでなんだい？
yasuoka : 1 から 100 までです。
root : じゃ、まずは 50っと。

```
? 50 （ぽこ）
big.
? 
```

bigってのは、 50 じゃ大き過ぎるって意味かい？
yasuoka : ええ。
root : それじゃっと。

```
? 25 （ぽこ）
big.
? 12 （ぽこ）
small.
? 17 （ぽこ）
correct.
% 
```

あ、当たっちゃった。
yasuoka : やりますねー。
root : で、どこがわからないんだい？
yasuoka : 1 つは 4 行目の `if` です。
root : ああ、 `if` は `if test` しか教えてなかったんだな。じゃ、この際だから `if` の一般的な書き方を教えてあげよう。

```
if コマンド列
then コマンド列
else コマンド列
fi
```
　　if の後のコマンド列を実行した結果、返ってきたエグジットステイタスが 0 だったら then 以降を実行し、それ以外だったら else 以降を実行する。

yasuoka ： じゃ、今までの test とか [ ] とかは、何だったんですか？
root ： あれは、 testっていう名のコマンドだったんだよ。
yasuoka ： えー？

```
test 条件　　あるいは　　[ 条件 ]
```
　　条件が真であったならエグジットステイタスとして 0 を返し、偽であったなら 1 を返す。出力なし。

root ： 実は、 while や until も if と同じなんだ。

```
while コマンド列
do コマンド列
done
```
　　while の後のコマンド列を実行した結果、返ってきたエグジットステイタスが 0 だったら、 do 以降を繰り返す。
```
until コマンド列
do コマンド列
done
```
　　until の後のコマンド列を実行した結果、返ってきたエグジットステイタスが 0 以外だったら、 do 以降を繰り返す。

yasuoka ： すると 4 行目の if は、 tty > /dev/null のエグジットステイタスが 0 の時には何もせず、それ以外の時には hb: standard input is not a tty. をエラー出力して終了するんですね。この ttyってどんなコマンドなんですか？
root ： 標準入力のデバイス名を教えてくれるんだ。

```
tty
```
　　標準入力が端末（キーボード等）であったなら、そのデバイス名を出力し、エグジットステイタスとして 0 を返す。さもなくば not a tty と出力し、エグジットステイタスとして 1 を返す。出力は標準出力。
```
tty -s
```
　　標準入力が端末であったなら、エグジットステイタスとして 0 を返す。さもなくば、エグジットステイタスとして 1 を返す。出力なし。

root ： ちょっと試してみようか？

```
% sh （ぽこ）
$ tty （ぽこ）
/dev/console
$ echo $? （ぽこ）
0
$ echo | tty （ぽこ）
not a tty
$ echo $? （ぽこ）
1
$ exec true （ぽこ）
%
```

yasuoka ： その shってのは？
root ： シェルを立ち上げるコマンドだよ。何もパラメータを書かないと、標準入力が入力になる。 $ってプロンプトが出るけどね。それより見てのとおり、 tty の入力が標準入力の時には $? が 0、そうでない時には $? が 1 になってるだろ？
yasuoka ： ええ。でもこのプログラムでは、 tty の標準出力を /dev/null というファイルにリダイレクトしてますけど？
root ： うん、それは出力を捨ててるんだ。
yasuoka ： 捨てるって？
root ： /dev/nullっていうファイルへの出力は、どこへもいかずに消えてしまうんだ。
yasuoka ： へえ。
root ： でもこれは tty -s を使えば、もっと簡単に書けるね。

```
#! /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
```

yasuoka : すると、もしかしたらこうかな？

```
% echo 50 | hb（ぽこ）
hb: standard input is not a tty.
% 
```

はは一ん。けど、どうしてこんなチェックをしてるんでしょう？

root : それは、15 行目の read がミソだね。

read 変数
: 標準入力から 1 行分読み込み、変数に代入する。エンドオブファイルが入力された時には、エグジットステイタスは 1。それ以外では、エグジットステイタスは 0。

root : この read の標準入力がキーボードじゃなかったら、全然ゲームにならないからね。

yasuoka : あ、そうか。この read で、キーボードからの入力を INPUT に入れてるんですね。

root : そう。それから、16 行目から 20 行目までは、ANSWER との大小比較だ。ANSWER に入れる乱数は 10 行目で作ってるね。

yasuoka : ええ、それはわかりました。プロセス番号を 100 で割った余りに、1 を足してるんでしょ？

root : 正解。

yasuoka : じゃあ、これはわかりました。どうもありがとうございます。ついでにもう 1 つ、いいですか？

root : 他にも、もらい物があるのかい？

yasuoka : はい。

```
% cat users（ぽこ）
#! /bin/sh
# "users" Version 1.0

TEMP1=/tmp/users$$a
TEMP2=/tmp/users$$b
who > $TEMP1
cp /dev/null $TEMP2
set ‘wc $TEMP1‘
LINE=$1

while [ $LINE -gt 0 ]
do set ‘head -$LINE $TEMP1 | tail -1‘
   echo $1 >> $TEMP2
   LINE=‘expr $LINE - 1‘
done

sort $TEMP2 | uniq | pr -t -8 -w80 -l1
rm $TEMP1 $TEMP2
exit 0
% 
```

root : どういうコマンドなんだい？

yasuoka ： えっとですね。

```
% users （ぽこ）
fujii     takahash  yasuoka
% who （ぽこ）
yasuoka  console Mar  3 15:49
takahash ttyp0   Mar  3 09:12   (kinkaku:0.0)
fujii    ttyp1   Mar  3 12:06   (daikaku)
takahash ttyp3   Mar  3 16:20   (kinkaku:0.0)
% 
```

こんな風に、 login してる人をアルファベット順に出してくれるんです。

root ： そうか。で、どこがわからないの？

yasuoka ： まず 4 行目と 5 行目で TEMP1 と TEMP2って変数を設定してるんですけど、この変数の意味がわかりません。特にどうして $$ が中に入ってるのか。

root ： この変数はテンポラリファイル名を指定してるね。

yasuoka ： テンポラリファイルって何ですか？

root ： コマンドの中で臨時に使うファイルのことだよ。 Unix ではこういう臨時ファイルを、 /tmp に自由に作っていいことになってるんだ。用が済んだら、必ず消すようにしなきゃいけないけどね。

yasuoka ： でも、どうしてテンポラリファイルの名前に $$ が入ってるんですか？

root ： 例えばテンポラリファイル名を /tmp/usersa とか固定にしたとするだろ。

yasuoka ： はい。

root ： その時このプログラムを同時に複数走らせるとどうなる？

yasuoka ： あ、そうか。その複数のプログラムが、皆んな同じファイルをアクセスしてしまうんですね。

root ： しかもいちばん先に終ったやつは、テンポラリファイルを rm してしまう。

yasuoka ： それでテンポラリファイルをそれぞれ変えるために、わざわざプロセス番号をファイル名に入れてるんですね。わかりました。それから 7 行目で /dev/null を /tmp/users$$b に cp してるんですけど、これ何です？

root ： これは中身が空っぽのファイルを作ってるんだよ。

yasuoka ： え？ 中身が空っぽのファイルって touch で作れませんでした？

> touch ファイル名
> ファイルの最終書込み時刻等を変更する。ファイルが存在しない時には、大きさが 0 のファイルを作る。

root ： いや、 touch はファイルの日付を変更するんだ。やってみようか。

```
% ls -l （ぽこ）
total 5
-rwxr-xr-x  1 yasuoka       122 Mar  2 16:49 bow
-rwxr-xr-x  1 yasuoka       343 Mar  3 16:22 hb
-rwxr-xr-x  1 yasuoka       307 Mar  3 11:57 users
-rwxr-xr-x  1 yasuoka       292 Mar  2 17:04 where
-rwxr-xr-x  1 yasuoka       240 Mar  1 20:29 who
% touch who （ぽこ）
% ls -l （ぽこ）
total 5
-rwxr-xr-x  1 yasuoka       122 Mar  2 16:49 bow
-rwxr-xr-x  1 yasuoka       343 Mar  3 16:22 hb
-rwxr-xr-x  1 yasuoka       307 Mar  3 11:57 users
-rwxr-xr-x  1 yasuoka       292 Mar  2 17:04 where
-rwxr-xr-x  1 yasuoka       240 Mar  3 16:51 who
% 
```

ほら、 who の日付が変わったろ。でも大きさが 0 になったりはしないよ。

yasuoka ： users の 7 行目では強制的に大きさ 0 のファイルを作ってるんですね。

root ： まあ、そうかな。

yasuoka ： それから 8 行目の wcってのは？

root ： それは 9 行目と合わせて、 LINE に $TEMP1 の行数を入れてるんだ。

> wc ファイル名
> ファイルの行数、単語数、文字数を標準出力に出力する。ファイル名が省略された場合は、入力は標準入力。

root ： ちょっとやってみよう。

```
% wc users （ぽこ）
      19      56     307 users
% 
```

users は 19 行、 56 単語、 307 文字ってことがわかる。

yasuoka ： ははーん。それを使って、 11 行目から 15 行目までのループの繰り返し回数を決めてるんですね。で、 $TEMP2 に who のユーザ ID の部分が逆順に入って、 17 行目っと。これ、全然わかりません。

root ： まず sort で $TEMP2 の内容を、 ASCII 順にしてる。

`sort` ファイル名
　　各行を ASCII 順に並べかえる。ファイル名は複数書いてもよい。入力はファイル、出力は標準出力。ただしファイル名が省略された場合は、入力は標準入力。

root ： さらに `uniq` で、同じ内容の行を削除する。

`uniq` ファイル名
　　連続する同じ内容の行を 1 行にまとめる。入力はファイル、出力は標準出力。ただしファイル名が省略された場合は、入力は標準入力。
`uniq -c` ファイル名
　　まとめた行数を、各行の先頭に出力する。他は上に同じ。

root ： 最後に `pr` で横に並べる。

`pr -t -`数 `-w`数 `-l`数 ファイル名
　　段組出力をおこなう。「`-`数」で段組数を、「`-w`数」で 1 行当たりの文字数を、「`-l`数」で 1 ページ当たりの行数を設定する。入力はファイル、出力は標準出力。ただしファイルが省略された場合には、入力は標準入力。

yasuoka ： 最後の `pr` がよくわかりません。
root ： そうだね。ちょっとやってみせよう。

```
% ls -1 > tmp （ぽこ）
% cat tmp （ぽこ）
bow
hb
tmp
users
where
who
% ■
```

こういう `tmp` に対して、幅 30 文字、ページ 2 行で 3 段組にすると

```
% pr -t -3 -w30 -l2 tmp （ぽこ）
bow         tmp         where
hb          users       who
% ■
```

1 段当たり 10 文字で、 2 行ごとに段組だからこんな風になる。ページ 3 行にすると

```
% pr -t -3 -w30 -l3 tmp （ぽこ）
bow         users
hb          where
tmp         who
% ■
```

で、右端に 3 行あまり。ページ 2 行で 2 段組にして、 1 ページに入り切らないようにすると

```
% pr -t -2 -w30 -l2 tmp （ぽこ）
bow             tmp
hb              users
where
who
% ■
```

最初の 4 行が、 1 ページ目の 2 行かける 2 段になる。ただし System V だと、各段組のバランスが取られるから、出力はちょっと違ってくる。

```
% rm tmp （ぽこ）
rm: remove tmp? y （ぽこ）
% ■
```

yasuoka ： `users` では 1 ページ 1 行の 8 段組で、 1 段当たり 10 文字ですね。
root ： そう。でも僕としては、この `users` には少し文句があるな。
yasuoka ： と、いいますと？
root ： 18 行目でテンポラリファイルを `rm` してるのはいいけど、もし実行中にコントロール C とかで止められちゃったら、 `/tmp` にテンポラリファイルが残ったままになっちゃう。
yasuoka ： それを防ぐ方法があるんですか？
root ： ある。でも今日はもう時間がないから、それはまた次回にしよう。
yasuoka ： はい。次回は必ず教えて下さいよ。