誰にでも書ける #! /bin/sh 講座
第 2 回
「who are you も入れてみよう」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : 前回の who を作りなおしたんですけど、見てくれますか？
root : ああ、いいよ。
yasuoka : じゃ。

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

if [ "$1" = am -a "$2" = i ]
then echo You are YASUOKA.
else if [ "$1" = are -a "$2" = you ]
     then echo I am YASUOKA.
     else if [ "$1" = is -a "$2" = he ]
          then echo He is YASUOKA.
          else if [ "$1" = is -a "$2" = she ]
               then echo She isn\'t YASUOKA.
               else /bin/who $*
               fi
          fi
     fi
fi

exit 0
% ■
```

root : ほう、 who are you も入れたのかい？
yasuoka : ええ。

```
% who are you（ぽこ）
I am YASUOKA.
% who is he（ぽこ）
He is YASUOKA.
% who is she（ぽこ）
She isn't YASUOKA.
% ■
```

root : isn\'t のところの \ がなかなか泣かせるね。でも実は else if という形は、もっと簡単に書くことができるんだよ。
yasuoka : え、そうなんですか？

```
if A                                    if A
then B                                  then B
else if C                               elif C
     then D        は右のように簡略化できる    then D
     else E                             else E
     fi                                 fi
fi
```

root : これを使えば、君の who は次のように書き換えられるね。

```
#! /bin/sh
# "who" Version 1.2

if [ "$1" = am -a "$2" = i ]
then echo You are YASUOKA.
elif [ "$1" = are -a "$2" = you ]
then echo I am YASUOKA.
elif [ "$1" = is -a "$2" = he ]
then echo He is YASUOKA.
elif [ "$1" = is -a "$2" = she ]
then echo She isn\'t YASUOKA.
else /bin/who $*
fi

exit 0
```

root ： でもこれ、できれば case を使った方が、速いし見やすいんじゃないかな。
yasuoka ： え？ caseって？

```
case 文字列 in
文字列) コマンド列 ;;
文字列) コマンド列 ;;
  …
esac
```

case と in の間の文字列と、) の前の文字列のマッチングをおこない、マッチしたら、) の後のコマンド列を実行する。

root ： 2 つの文字列のどちらかとのマッチングの時には

```
  文字列|文字列) コマンド列 ;;
```

ってとこかな。
yasuoka ： その case を使ってさっきの who を書き換えると、どうなります？
root ： うーん、そうだな。

```
#! /bin/sh
# "who" Version 2.0

case "$1 $2" in
"am i") echo You are YASUOKA. ;;
"are you") echo I am YASUOKA. ;;
"is he") echo He is YASUOKA. ;;
"is she") echo She isn\'t YASUOKA. ;;
*) /bin/who $* ;;
esac

exit 0
```

root ： ま、こんなもんかな？
yasuoka ： 最後の *)ってのは else ですか？
root ： うーん、ようするに単なる * はどんな文字列ともマッチするんだよ。ただ case では評価は前から順番におこなわれるから、 "am i"でも "are you"でも "is he"でも "is she"でもなかったものだけが、 * にマッチすることになるんだ。ついでだから、他のワイルドカードも教えてあげよう。

*	ヌルを含めていかなる文字列ともマッチ
?	いかなる 1 文字ともマッチ
[複数の文字]	[ ] の中のいかなる 1 文字ともマッチ
[文字-文字]	2 つの文字の間のいかなる 1 文字ともマッチ

root ： でも僕としては、この who にはまだまだ文句があるな。
yasuoka ： と、いいますと？
root ： この who は君専用になってるだろ？
yasuoka ： ええ。
root ： つまり他の人が、これを使いたいな、と思った時には、あらかじめプログラム中の YASUOKA を、自分の名前に書き換えなきゃいけない。
yasuoka ： はあ。
root ： それよりは $USER かなんかを使って、他の誰でもが書き換えなしに使えるようにした方が、カッコいいと思うんだけど。
yasuoka ： そうですかぁ？

```
#! /bin/sh
# "who" Version 2.1

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 $HEAD $USER"."
exit 0
```

yasuoka ： この HEAD="You are"ってのは？
root ： HEADっていう変数に You areっていう文字列を代入するっていう意味だよ。シェルでは、変数は宣言なしに使うことができるんだ。

変数=文字列
　　変数への代入。= の前後に空白をあけないよう注意すること。
$変数
　　変数の読み出し。

（間）

yasuoka ： できました。でもこれ、ちょっと変です。
root ： 変って？
yasuoka ： 小文字になっちゃうんです。

```
% who am i （ぽこ）
You are yasuoka.
% ■
```

root ： ユーザ ID は普通、小文字だからな。やっぱり大文字の方がいいかい？
yasuoka ： もちろん！
root ： そうか。じゃ、これでどうかな？

```
#! /bin/sh
# "who" Version 2.2

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 $HEAD" " | tr -d '\012'
echo $USER"." | tr a-z A-Z
exit 0
```

（間）

yasuoka ： できました。
root ： どれどれ。

```
% who am i （ぽこ）
You are YASUOKA.
% ■
```

よしよし、予定どおりだな。
yasuoka ： どんな仕掛けなんですか？
root ： trっていう置き換えコマンドが、まあすべてだね。

tr 文字列 文字列
　　前の文字列中の各文字を、後の文字列中の同じ順番のところにある文字に置き換える。なお、2 つの文字列は同じ長さでなければならない。
tr -d 文字列
　　文字列中の各文字を削除する。
いずれも、入力は標準入力、出力は標準出力である。なお略記法として、例えば cdef に対しては、BSD では c-f が、System V では'[c-f]' が許されている。

yasuoka ： この標準入力とか、標準出力とかって、何ですか？
root ： まあ普通、標準入力はキーボード、標準出力はディスプレイだな。あと、エラー出力っていうのもあって、これもディスプレイになっている。
yasuoka ： あれ、すると下から 2 行目と 3 行目の tr の入力はキーボードなんですか？
root ： いやいや、その直前にそれぞれ | があるだろ。|ってのは、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んだ。パイプとかリダイレクトとかは知ってるね？
yasuoka ： まあ、一応は。
root ： 何だ、自信のなさそうな返事だな。じゃあこの際、シェルでのパイプとリダイレクトを教えておこうか。C シェルとは、少しばかり違うしね。

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 > 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。
コマンド 2> ファイル名
　　コマンドのエラー出力をファイルに書き出す。
コマンド 2>> ファイル名
　　コマンドのエラー出力をファイルに書き加える。
コマンド >&2
　　コマンドの標準出力をエラー出力に出す。
コマンド 2>&1
　　コマンドのエラー出力を標準出力に出す。
なお、標準出力とエラー出力の両方をリダイレクトする際、
　　コマンド > ファイル名 2>&1
は、標準出力とエラー出力の両方がファイルに書き出されるが、
　　コマンド 2>&1 > ファイル名 | コマンド
は、標準出力のみがファイルに書き出され、エラー出力は後のコマンドの標準入力に繋がれることになるので、注意が必要である。

yasuoka ： 思ったよりたくさんあるんですね。
root ： まあね。 C シェルのリダイレクトとはエラー出力関係が違うから、注意した方がいいよ。
yasuoka ： それからあの、下から 3 行目の tr -d ’\012’って、何を削除してるんですか？
root ： 改行コードだよ。 tr では \ の後に 3 桁の 8 進数をおくことによって、文字を ASCII コードで表せるんだ。
yasuoka ： あれっ？ 確か echo のオプションに、改行しない、ってのがありませんでしたか？
root ： よく知ってるね。ただそれは、 BSD と System V とでやり方に違いがあるんだよ。

echo 文字列
　　文字列を標準出力に出力する。文字列は複数書いてもよい。最後に改行コードを出力する。
echo -n 文字列
　　BSD のみ。上と同じで、改行コードを出力しない。
echo 文字列’\c’
　　System V のみ。上と同じで、改行コードを出力しない。

yasuoka ： そうなんですか。
root ： すぉーなんですよ。ま、これを使ってさっきのプログラムを書きなおすと、 BSD ではこうなる。

```
#! /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
```

root ： System V ではこうだ。

```
:
# "who" Version 2.3 for System V

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 $HEAD' \c'
echo ${USER:-$LOGNAME}"." | tr '[a-z]' '[A-Z]'
exit 0
```

yasuoka : すみません。下から 2 行目の ${USER:-$LOGNAME} ってのは、何ですか？

root : ああ、これは変数の読み出しの特殊なやり方だよ。USER がヌルかセットされていなかったら、代わりに LOGNAME を読み出してるんだ。System V では、ログイン名が LOGNAME に入ってるマシンもあるから、こういう風にしてるんだ。

${変数}
　「$変数」に同じ。変数の読み出し。
${変数-文字列}
　変数を読み出す。ただし、変数がセットされていなかったら、代わりに文字列を返す。
${変数=文字列}
　変数を読み出す。ただし、変数がセットされていなかったら、代わりに文字列を返し、さらに変数に文字列を代入する。
${変数?文字列}
　変数を読み出す。ただし、変数がセットされていなかったら、文字列をエラー出力してエグジットステイタス 1 で強制終了。
${変数+文字列}
　変数がセットされていたら文字列を返し、さもなくばヌルを返す。
${変数:-文字列}
　System V のみ。変数を読み出す。ただし、変数がヌルもしくはセットされていなかったら、代わりに文字列を返す。
${変数:=文字列}
　System V のみ。変数を読み出す。ただし、変数がヌルもしくはセットされていなかったら、代わりに文字列を返し、さらに変数に文字列を代入する。
${変数:?文字列}
　System V のみ。変数を読み出す。ただし、変数がヌルもしくはセットされていなかったら、文字列をエラー出力してエグジットステイタス 1 で強制終了。
${変数:+文字列}
　System V のみ。変数がヌルでなかったなら文字列を返し、さもなくばヌルを返す。
$1 などに対してこのような読み出しをおこなう場合には、例えば ${1:-文字列} という風に書く。ただし、$1 や $# などに対しては、= や := は使用できない。

yasuoka : そういえば、$1 なんかには代入はできないんですか？

root : いいや、set を使えばなんとかなるよ。

set 文字列 文字列 …
　文字列をパラメータとみなして、$1 $2 …および $*、$@、$# をセットしなおす。ただし、$0 や $$ は変わらない。
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。

yasuoka : set の後の文字列が、コマンドを起動した時のパラメータと同じように扱われるんですね。

root : ま、そうかな。あっと、もうこんな時間だ。まだまだ話したいことはあるけど、続きはまた今度にしてくれるかい？

yasuoka : え、もう終わりなんですかぁ？ 繰り返しなんかも聞きたかったのに。

root : すまないね。じゃ、また次回。