誰にでも書ける #! /bin/awk -f 講座
第 1 回
「行番号のフィルタ」

安岡孝一

yasuoka ： root さん、 root さん。
root ： 何だい？
yasuoka ： takahash くんがいじめるんです。
root ： 何かいじめられるようなこと、したんだろう。
yasuoka ： root さんにも関係あるんですよ。
root ： って言うと？
yasuoka ： 前に、行番号を付ける sed のスクリプトを作ったでしょう？
root ： そんなの作ったっけ？
yasuoka ： もう忘れたんですか？

```
/home/yasuoka/bin% cat number~ （ぽこ）
#! /bin/sed -f
/ "number" Version 1.0 /{}
x
1s/^.*$/     1/
G
h
y/ 0123456789/11234567890/
G
s/^.*\([^0]0*\)\n.*\n\(.*\)[^9]9*\n.*$/\2\1/
x
s/\n/  /
/home/yasuoka/bin% number~ number~ （ぽこ）
     1  #! /bin/sed -f
     2  / "number" Version 1.0 /{}
     3  x
     4  1s/^.*$/     1/
     5  G
     6  h
     7  y/ 0123456789/11234567890/
     8  G
     9  s/^.*\([^0]0*\)\n.*\n\(.*\)[^9]9*\n.*$/\2\1/
    10  x
    11  s/\n/  /
/home/yasuoka/bin% ■
```

root ： ああ、そういえばこんなの書いたな。で、これがどうかしたのかい？
yasuoka ： takahash くんがいじめるんです。
root ： 何かいじめられるようなこと、したんだろう。
yasuoka ： このスクリプトを見せびらかしたんです。「いいだろー」って。
root ： で？
yasuoka ： そしたら「そんなの awk 使ったら、もっと簡単にできる」って

```
/home/yasuoka/bin% cat number （ぽこ）
#! /bin/awk -f
# "number" Version 2.0
BEGIN{
  i=0;
}
{
  i++;
  printf("%6d  %s\n",i,$0);
}
/home/yasuoka/bin% ■
```

これをちゃっちゃっと書いてくれたんです。
root ： ふーん。
yasuoka ： でもこれ、さっぱり読めません。

```
/home/yasuoka/bin% number number （ぽこ）
     1  #! /bin/awk -f
     2  # "number" Version 2.0
     3  BEGIN{
     4    i=0;
     5  }
     6  {
```

```
      7    i++;
      8    printf("%6d  %s\n",i,$0);
      9  }
 /home/yasuoka/bin% ■
```

何となく、C のプログラムに似てるような気はするんですけど。

root : これは awk のスクリプトだよ。

yasuoka : awkって？

root : sed なんかと同じで、フィルタを書くためのインタプリタ型言語だよ。1 行目の #! /bin/awk -fってのが、 awk のスクリプトだっていう宣言だ。

yasuoka : そうなんですか。

root : ただし、System V では 1 行目は exec awk "‘sed 1d $0‘" ${1+"$@"} と書く。

yasuoka : わかりました。

root : で、 2 行目の # から始まる行はコメント。 3 行目から最後までは、それぞれ awk の命令。 awk のスクリプトでは、 Unix のコマンドは全然使えないんだ。使えるのは awk の命令だけ。

yasuoka : sed の命令とも違うんですか？

root : うん、全然違う。

yasuoka : じゃ、その awk の命令ってのを教えて下さい。

root : そうだな。まずは awk スクリプトの基本的なルールを教えておこう。

```
/home/yasuoka/bin% cat number （ぽこ）
#! /bin/awk -f
# "number" Version 2.0
BEGIN{
  i=0;
}
{
  i++;
  printf("%6d  %s\n",i,$0);
}
/home/yasuoka/bin% ■
```

スクリプトが始動すると、まず BEGIN{から} までが実行される。この number だと 4 行目の i=0 が、まず実行されるわけだ。

yasuoka : i=0 の後に; がありますけど？

root : awk では、命令の後に必ず; を書くんだ。

yasuoka : そうなんですか。

root : で、 BEGIN{のところの実行が済んだら、次に {から} までが、入力 1 行あたりに 1 回ずつ実行される。これだと 7 行目と 8 行目が、 1 行に 1 回実行されるわけだ。

yasuoka : はい。

root : で、最後に END{から} が実行される。このスクリプトでは END{はないから、入力が終わったら、それでスクリプトは終了だ。

yasuoka : するとこの number を実行すると、まず i=0 が実行されて、それから入力 1 行毎に、 i++ と printf("%6d %s\n",i,$0) が実行されるんですね。

root : そう。

yasuoka : i=0 は、 iって変数に 0 を代入するっていう意味ですか？

root : よくわかったね。

yasuoka : で、 i++ は、 i の値を 1 増やすって意味でしょう？

root : 大正解。じゃここで、 awk での変数への代入について、教えとこう。

変数=式
　　変数に式の値を代入する。代入された値を返す。
変数+=式
　　変数に式の値を加える。加えた結果を値として返す。
変数-=式
　　変数から式の値を引く。引いた結果を値として返す。
変数*=式
　　変数を式の値倍する。結果を値として返す。
変数/=式
　　変数を式の値で割る。割った結果を値として返す。
変数%=式
　　変数を式の値で割った余りを変数に代入する。代入した値を返す。
変数++
　　変数に 1 加える。加える前の変数の値を返す。
変数--
　　変数を 1 減らす。減らす前の変数の値を返す。

++変数
　　変数に 1 加える。加えた後の変数の値を返す。
--変数
　　変数を 1 減らす。減らした後の変数の値を返す。

yasuoka ： 四則演算は全部あるんですね。でも、代入した値を返す、って何ですか？
root ： 代入式は、それ自身も式として使えるんだよ。例えば、 j=i++っていう代入式は、 i を 1 増やすと同時に、増やす前の i の値を j に代入する。
yasuoka ： へーえ。すると i=j=k=0 は、 k と j と i の全部に 0 を代入するんですか？
root ： そのとおり。実は i=j=k="0"でも同じだ。 awk では、文字列と数の区別はないからね。
yasuoka ： 文字列は、ダブルクウォートで囲むんですか？
root ： そう。
yasuoka ： すると、 i="j"でも i=j でも、 i には jって文字列が入るんですね。
root ： いや、違う。 i="j"なら、 i には確かに jって文字列が入るけど、 i=j だと、 jっていう変数の値を読み出して、それを i に代入する。
yasuoka ： あ、単に jって書くと、 jって変数の読み出しになるんですか。
root ： そう。
yasuoka ： $ は要らないんですか？
root ： $j と書くと、今入力中の行の第 j フィールドを読み出すことになる。

$0	現在入力中の行全体
$数	現在入力中の行の第「数」フィールド（数は自然数のみ）
NR	現在入力中の行が何行目か
NF	現在入力中の行のフィールド数

root ： 1 行毎に同じ命令を繰り返すわけだから、行の内容を読み出す方法がないと、困るだろ。
yasuoka ： 8 行目の printf("%6d　%s\n",i,$0) の $0 もそれですか？
root ： うん、そう。今入力中の行全体を読み出す。
yasuoka ： で、この printf("%6d　%s\n",i,$0)って、どういう命令なんですか？
root ： 文字列や式の値を出力する命令だよ。これだと、変数 i の値を右詰め 6 桁で、その後に空白を 2 文字、さらにその後に現在入力中の行全体、そして最後に改行コードを出力する。

printf(フォーマット);
　　フォーマットにしたがって、文字列を標準出力に出力する。フォーマットは以下の形である。
　　　　文字列,式,式…
　　式の値によって、文字列中のフォーマット制御文字列が置き換えられる。式の個数は、文字列中のフォーマット制御文字列の個数と、一致していなければならない。
フォーマット制御文字列は、以下の通り。

%c	数に対応する ASCII 文字
%d	10 進整数
%f	小数部 6 桁の 10 進数
%.数f	小数部「数」桁の 10 進数（数は自然数のみ）
%o	8 進整数
%s	文字列
%.数s	「数」文字を越えない文字列（数は自然数のみ）
%x	16 進整数

いずれも、 % の後に整数を書くことにより、桁数を指定できる。例えば、 %12d と指定すると、これに対応する値が 12 桁未満の時には、値の前に空白が詰まって 12 文字分となる。 %-12d と指定すると、対応する値が 12 桁未満の時には、値の後に空白が詰まって 12 文字分となる。 %012d では、値の前に 0 が詰まって 12 文字分となる。
特殊文字のための制御文字列は、以下の通り。

%%	%
\n	改行コード
\t	タブ
\\	\

\n、 \t、 \\ は通常の文字列中にも使用できる。

root ： ようするに、 "%6d　%s\n"の %6d のところが i の値で、 %s のところが $0 の値で、それぞれ置き換えられて、出力されるんだよ。
yasuoka ： なかなか難しいですね。

```
/home/yasuoka/bin% who （ぽこ）
yasuoka  console Mar 15 12:51
hama     ttyp0   Mar 15 08:17   (daikaku)
/home/yasuoka/bin% who | number （ぽこ）
```

```
     1  yasuoka  console Mar 15 12:51
     2  hama     ttyp0   Mar 15 08:17   (daikaku)
/home/yasuoka/bin% 
```

あ、そうか。初めて printf に来た時には、i が 1 で $0 が入力の 1 行目だから、1 の後に入力の 1 行目がくっつくんだ。で、次に printf に来るのは i++ を実行した後だから、i が 2 で $0 が入力の 2 行目になってると。なーんだ。

```
/home/yasuoka/bin% cat number （ぽこ）
#! /bin/awk -f
# "number" Version 2.0
BEGIN{
  i=0;
}
{
  i++;
  printf("%6d  %s\n",i,$0);
}
/home/yasuoka/bin% 
```

root ： 実は 4 行目の i=0 は必要ないんだけどね。

yasuoka ： え、なぜですか？

root ： awk では、まだ代入されてない変数は、全部ヌルになってるんだよ。で、ヌルに 1 を加えると、数字以外は 0 だとみなされるから、結果は 1 になる。だから i=0 がなくても、最初の i++ で i はちゃんと 1 になるはずだ。

```
#! /bin/awk -f
# "number" Version 2.1
{
  i++;
  printf("%6d  %s\n",i,$0);
}
```

（間）

yasuoka ： こんなに短くなるんですか。

```
/home/yasuoka/bin% number number （ぽこ）
     1  #! /bin/awk -f
     2  # "number" Version 2.1
     3  {
     4    i++;
     5    printf("%6d  %s\n",i,$0);
     6  }
/home/yasuoka/bin% 
```

あ、でもこれ、4 行目の i++ と、次の行の printf("%6d %s\n",i,$0) をまとめて、printf("%6d %s\n",++i,$0)ってしたら、もっと短くなるんじゃないですか？

root ： うーん、printf のフォーマットの中の式には、代入式は使えないんだよ。最近の awk では使えるのもあるみたいだけど…。とにかく、printf の中で使えるのは、普通の式だけだと考えておいた方がいい。

yasuoka ： 普通の式って？

文字列	文字列（複数並べてもよい）
数	10 進数（小数も可）
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
式%式	左式を右式で割った余り
int(式)	式の整数部
log(式)	式の自然対数
exp(式)	e の「式」乗
sqrt(式)	式の平方根
length(文字列)	文字列の長さ
substr(文字列,式)	文字列の「式」文字目以降
substr(文字列,式,式)	文字列の左「式」文字目から右「式」文字
index(文字列,文字列)	左文字列の何文字目に右文字列を含むか（ない時 0）
sprintf(フォーマット)	フォーマットにしたがった文字列
(式)	優先度を変える

root ： ま、こんなもんかな。

yasuoka ： 平方根とか、色々使えるんですね。

root ： うん。ところでさっきの number だけど、NR を使えばもっと短くなるな。

```
#! /bin/awk -f
# "number" Version 2.2
{
  printf("%6d  %s\n",NR,$0);
}
```

yasuoka : え？ i はどこへ行ったんですか？

root : わざわざ 1 行毎に i をインクリメントしなくても、 NR に今の行が何行目かが、入ってるからね。これを使えば一発だよ。

yasuoka : あ、そうか。

```
/home/yasuoka/bin% number number （ぽこ）
     1  #! /bin/awk -f
     2  # "number" Version 2.2
     3  {
     4    printf("%6d  %s\n",NR,$0);
     5  }
/home/yasuoka/bin% 
```

エグジットステイタスは、どうなるんだろ。

```
/home/yasuoka/bin% !! ; echo $status （ぽこ）
number number ; echo $status
     1  #! /bin/awk -f
     2  # "number" Version 2.2
     3  {
     4    printf("%6d  %s\n",NR,$0);
     5  }
0
/home/yasuoka/bin% 
```

0 か。エグジットステイタスは変えられないんですか？

root : exit を使えば、変えられるよ。

```
exit(式);
      式の値をエグジットステイタスとして、実行を終了する。
exit;
      実行を終了する。それ以前に exit(式); が実行されていない場合、エグジットステイタスは 0。
BEGIN{および END{以外での exit は、 END{を実行してから終了する。
```

root : 例えばこのスクリプトを、エグジットステイタス 3 で終わらせたいのなら

```
#! /bin/awk -f
# "number" Version 2.2
{
  printf("%6d  %s\n",NR,$0);
}
END{
  exit(3);
}
```

ってなとこかな。

yasuoka : 普通に実行が終了した時のエグジットステイタスは 0 なんですよね。

root : そうだけど。

yasuoka : じゃあ、単なる exit;って、必要ないんじゃないですか？

root : どうして？

yasuoka : だって、最終行まで行ってもエグジットステイタスは 0 なんだし、 exit; で終了してもエグジットステイタスは 0 なんでしょ。わざわざ exit; で終了する必要がないじゃないですか。

root : END{の中なら、そうかな。でも、途中の行で実行を終了させたいのなら、必要だろ？

yasuoka : え？

root : 例えば、入力の最初の空行までを出力する、みたいなスクリプトだと、 $0 がヌルになった時に exit する、って書き方が一般的だろ？

yasuoka : あ、そうですね。でも、 $0 がヌルなら exit、ってどう書くんですか？

root : 普通は

```
if($0=="")
  exit;
```

って書くんだけど、今日はもう時間がないから、細かい説明はまた次回にしよう。次回は awk での if の書き方だ。

yasuoka : はい。どうもありがとうございました。