誰にでも書ける #! /bin/nawk -f 講座
第 2 回
「置き換えもできる」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : 今日は nawk で増えた命令を教えてくれるって約束だったじゃないですか。
root : ああ、そうだったね。じゃあ、前回の yasuoka を見せてくれるかい？
yasuoka : はい。

```
~/bin% cat yasuoka （ぽこ）
#! /bin/nawk -f
# "yasuoka" Version 2.1
{
  t=$0;
  while(i=index(t,"yasuoka"))
    t=substr(t,1,i-1) "YASUOKA" substr(t,i+7);
  printf("%s\n",t);
}
~/bin% ■
```

root : この yasuoka では文字列の置き換えを while で回してるけど、実は nawk にはもっと便利な置き換え命令があるんだ。

sub(/正規表現/,文字列, 変数)
　　変数中で、最初に正規表現とマッチした部分を、文字列に置き換えて変数に代入する。
sub(文字列,文字列,変数)
　　左文字列を正規表現とみなし、他は上に同じ。
gsub(/正規表現/,文字列,変数)
　　変数中で、正規表現とマッチした部分を、全て文字列に置き換えて変数に代入する。
gsub(文字列,文字列,変数)
　　左文字列を正規表現とみなし、他は上に同じ。
いずれも置き換えた数を返す。なお、文字列中の文字として \、 & を用いる際には、それぞれ \\、 \\& とすること。

yasuoka : 何だか sed の s に似てますね。
root : \(\) は使えないけどね。じゃあこの gsub を使って、 yasuoka を書き換えてごらん。
yasuoka : はい。

（間）

yasuoka : こんなもんでどうです？

```
~/bin% cat yasuoka （ぽこ）
#! /bin/nawk -f
# "yasuoka" Version 2.2
{
  t=$0;
  gsub(/yasuoka/,"YASUOKA",t);
  printf("%s\n",t);
}
~/bin% yasuoka yasuoka （ぽこ）
#! /bin/nawk -f
# "YASUOKA" Version 2.2
{
  t=$0;
  gsub(/YASUOKA/,"YASUOKA",t);
  printf("%s\n",t);
}
~/bin% ■
```

root : うん、まずは正解。でも実は、もうちょっと短くできるんだ。

```
#! /bin/nawk -f
# "yasuoka" Version 2.3
{
```

```
  gsub(/yasuoka/,"YASUOKA",$0);
  printf("%s\n",$0);
}
```

yasuoka ： $0 を置き換えることもできるんですね。

root ： そう。しかも $0 を置き換えた時は、 $1 も NF も同時に変更される。

yasuoka ： NFって何でしたっけ？

root ： 現在入力中の行のフィールド数。

$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
FNR	現在のファイルにおいて、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
FILENAME	現在入力中のファイル名
ARGV[数]	第「数」パラメータ（数は自然数のみ）
ARGC	パラメータ数+1

yasuoka ： そういえば、こういうのありましたね。けど、この ARGV と ARGCって何ですか？

root ： コマンドを実行した時のパラメータだよ。 C シェルの argv と同じだ。

yasuoka ： え、 nawk は、パラメータにファイル名以外のものが書けるんですか？

root ： うん、これが nawk でいちばん便利な機能だと思う。試しに使ってみようか。前回の sin を見せてくれるかい？

yasuoka ： はい。

```
~/bin% cat sin （ぽこ）
#! /bin/nawk -f
# "sin" Version 1.2
BEGIN{
  printf("degree? ");
}
{
  if($0=="")
    exit;
  printf("%f\ndegree? ",sin($1*0.0174532925));
}
~/bin%
```

root ： じゃこれを、パラメータから値を読み込むようにしてみよう。

```
#! /bin/nawk -f
# "sin" Version 2.0
BEGIN{
  printf("%f\n",sin(ARGV[1]*0.0174532925));
  exit;
}
```

（間）

yasuoka ： できました。どうやって使うんですか？

root ： sin 45 とでもしてごらん。

yasuoka ： サイン 45 度ですね。

```
~/bin% sin 45 （ぽこ）
0.707107
~/bin%
```

あ、ちゃんとできますね。

```
~/bin% sin 30 （ぽこ）
0.500000
~/bin% sin 60 （ぽこ）
0.866025
~/bin%
```

うん、完璧。

```
~/bin% sin -30 （ぽこ）
/bin/nawk: unknown option: -30.
~/bin%
```

あれ、変なのが出た。

root ： うーん、確かにこれは問題だな。

yasuoka ： どういうことですか？

root ： nawk のスクリプトでは、 - から始まるパラメータは ARGV に渡されずに、nawk 自身が解釈しちゃうんだよ。それでこういうエラーになる。

yasuoka ： それは悲しい。何とかならないんですか？

root ： 第 1 パラメータじゃなければいいんだけど…。

```
#! /bin/nawk -f
# "sin" Version 2.1
BEGIN{
  for(i=1;i<ARGC;i++)
    printf("sin(%f)=%f\n",ARGV[i],sin(ARGV[i]*0.0174532925));
  exit;
}
```

yasuoka : これでどうなったんですか？

root : 第 2 パラメータ以降も、サインを計算するようにしたんだよ。もうできたかい？

yasuoka : はい。

```
~/bin% sin -30 （ぽこ）
/bin/nawk: unknown option: -30.
~/bin% ■
```

やっぱり変ですよ。

root : sin 0 -30 とでもしてみてくれるかい？

yasuoka : sin 0 -30 ですね。

```
~/bin% sin 0 -30 （ぽこ）
sin(0.000000)=0.000000
sin(-30.000000)=-0.500000
~/bin% ■
```

あ、こういうことですか。でもあんまりきれいな方法じゃないですね。

root : まあね。でも nawk では、 - から始まる第 1 パラメータは使えないから、どうにも仕方ない。

yasuoka : そうなんですか。じゃあ、 root さん。

root : 何だい？

yasuoka : nawk の他の新しい機能を教えて下さい。

root : そうだな。前に階乗を計算するスクリプトを作らなかったっけ？

yasuoka : これですか？

```
~/bin% factorial （ぽこ）
number? 10 （ぽこ）
10!=3628800
number? q （ぽこ）
~/bin% cat factorial （ぽこ）
[ [ ] ; exec dc $0 #] c
[ "factorial" Version 3.2 ] s.
[0] 0 :p [1] 1 :p [2] 2 :p [3] 3 :p [4] 4 :p
[5] 5 :p [6] 6 :p [7] 7 :p [8] 8 :p [9] 9 :p
[d SA 10 / d 0 <P LA 10 % ;p P] sP
[la * la 1 - d sa 1 <!] s!
[[number? ] P ? d sa lP x s. [!=] P
1 la 1 <! p s. d x] d x q
~/bin% ■
```

root : そうそう。これの入力をパラメータから取ってこれるようにしようか。

```
#! /bin/nawk -f
# "factorial" Version 4.0
BEGIN{
  printf("1 ")|"dc";
  for(i=ARGV[1];i>1;i--)
    printf("%d * ",i)|"dc";
  printf("p q\n")|"dc";
  close("dc");
  exit;
}
```

（間）

yasuoka : できました。

```
~/bin% factorial （ぽこ）
1
~/bin% ■
```

あれ？

root : パラメータを取るようにしたんだよ。

yasuoka : あ、そうでしたね。

```
~/bin% factorial 10 （ぽこ）
3628800
~/bin% ■
```

うまくいきました。でもこれ、どういう仕掛けなんですか？ printf の後にパイプが繋がってるみたいですけど。

root ： その通り。これで printf の出力が後の dc の入力になるんだ。

printf(フォーマット)|文字列; 　文字列を Unix コマンドとみなして、シェルを通じて実行する。printf の出力が標準入力に繋がれる。 printf(フォーマット)>文字列; 　文字列をファイル名とみなして、printf の出力をファイルに書き出す。 printf(フォーマット)>>文字列; 　文字列をファイル名とみなして、printf の出力をファイルに書き加える。

root ： もし |"dc"がなかったとして、このスクリプトを factorial 5 で実行したらどうなる？

yasuoka ： えっと、１５＊４＊３＊２＊ｐｑが表示されます。close("dc") がよく分かりませんけど。

root ： うん、まあそういうことだ。で、全部の printf に |"dc"が付いてるから、１５＊４＊３＊２＊ｐｑが dc コマンドの標準入力に入力される。

yasuoka ： それで 5 の階乗が計算できるんですね。

```
~/bin% cat factorial （ぽこ）
#! /bin/nawk -f
# "factorial" Version 4.0
BEGIN{
  printf("1 ")|"dc";
  for(i=ARGV[1];i>1;i--)
    printf("%d * ",i)|"dc";
  printf("p q\n")|"dc";
  close("dc");
  exit;
}
~/bin% ■
```

でもこれ変ですよ。|"dc"が 3 箇所もあるじゃないですか。

root ： そこがシェルのパイプと nawk のパイプの違いなんだ。nawk では同じ文字列への printf は、最終的に全部まとめておこなわれる。つまりこのスクリプトでも、実行される dc はただ一つで、そこへ全部の printf の出力がまとめて入力される。で、出力の終了を宣言するのが 8 行目の close だ。

close(文字列); 　文字列で示される Unix コマンドあるいはファイルとの入出力を終了する。

yasuoka ： ふーん、そういうことなんですか。

root ： まあ、実行終了時に全ての入出力は閉じられるから、この close はなくてもいいんだけどね。

（間）

yasuoka ： root さん、root さん。

root ： 何だい？

yasuoka ： さっきの factorial、ちょっと改造したんですけど、結果が変なんです。

root ： 変って？

yasuoka ： えっと

```
~/bin% cat factorial （ぽこ）
#! /bin/nawk -f
# "factorial" Version 4.1
BEGIN{
  printf("%d!=",ARGV[1]);
  printf("1 ")|"dc";
  for(i=ARGV[1];i>1;i--)
    printf("%d * ",i)|"dc";
  printf("p q\n")|"dc";
  close("dc");
  exit;
}
~/bin% ■
```

例えば factorial 10 を実行した時には、10!=3628800 が出力されるようにしたんですけど

```
~/bin% factorial 10 （ぽこ）
3628800
10!=~/bin% ■
```

逆の順番に出ちゃうんです。どうしてなんでしょう？

root ： printf からの標準出力と dc の標準出力は、それぞれ別々の経路で表示されるからね。どっちが先になるかは、その時その時によるんだよ。

yasuoka : えー、何とかなりませんか？
root : printf の標準出力を強制的に表示させるしかないな。

system(文字列)
文字列を Unix コマンドとみなして、シェルを通じて実行する。実行前に printf(フォーマット); の出力は、全て強制的に出力される。エグジットステイタスを返す。

yasuoka : m4 の syscmd に似てますね。でもこれをどうやって使えばいいんですか？
root : system() を使う。

```
#! /bin/nawk -f
# "factorial" Version 4.2
BEGIN{
  printf("%d!=",ARGV[1]);
  system();
  printf("1 ")|"dc";
  for(i=ARGV[1];i>1;i--)
    printf("%d * ",i)|"dc";
  printf("p q\n")|"dc";
  close("dc");
  exit;
}
```

（間）

root : うまくいったかい？
yasuoka : はい。

```
~/bin% factorial 10 （ぽこ）
10!=3628800
~/bin% 
```

でも system の中に何も書いてないのに、どうしてうまくいくんですか？
root : ヌルが system に渡ると、シェルは何も実行せずに帰ってくるからね。でも system の実行前には、 printf の出力は強制的に表示されるから、この例ではうまくいくんだ。まあ、こんな難しいことしなくても、出力は全部 dc にまかせるって手もあるけどね。

```
#! /bin/nawk -f
# "factorial" Version 4.3
BEGIN{
  printf("[%d!=] P 1 ",ARGV[1])|"dc";
  for(i=ARGV[1];i>1;i--)
    printf("%d * ",i)|"dc";
  printf("p q\n")|"dc";
  close("dc");
  exit;
}
```

yasuoka : そういえば、そうですね。
root : さて、 nawk の色んな新しい機能を話したけど、もうこんな時間だ。今日はこのくらいでいいかい？
yasuoka : はい、色々と勉強になりました。どうもありがとうございました。