誰にでも書ける #! /bin/nawk -f 講座
第 3 回
「関数を定義しよう」

安岡孝一

yasuoka : root さん、root さん。

root : 何だい？

yasuoka : who の順番を login 順にするスクリプトをもらったんですけど。

```
~/bin% cat how （ぽこ）
#! /bin/nawk -f
# "how" Version 1.0
BEGIN{
  "date"|getline d;
  close("date");
  split(d,m);
  c="sort | sed 's/^[^ ]* //'";
  while(("who"|getline s)==1){
    split(s,a);
    printf("%d%02d%s %s\n",a[3]==m[2],a[4],a[5],s)|c;
  }
  close("who");
  close(c);
  exit;
}
~/bin% 
```

root : login 順って？

yasuoka : えっとですね。

```
~/bin% who （ぽこ）
yasuoka  console Apr  6 15:19
ochi     ttyh0   Apr  6 11:09
takahash ttyh1   Apr  6 13:14
~/bin% 
```

who の出力がこういう風になってる時に

```
~/bin% how （ぽこ）
ochi     ttyh0   Apr  6 11:09
takahash ttyh1   Apr  6 13:14
yasuoka  console Apr  6 15:19
~/bin% 
```

それを時刻順に並べてくれます。

root : ふーん。でもどうして howって名前なんだい？

yasuoka : さあ？ たぶん who の文字をアルファベット順に並べたんじゃないですか？

root : そうか。で、これのどこがわからないんだい？

yasuoka : 4 行目と 8 行目にある getline です。

> 文字列|getline 変数
>
> 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。コマンドの標準出力を 1 行読み込んで、変数に代入する。読み込みに成功した場合は 1、エンドオブファイルの場合は 0 を返す。
>
> getline 変数<文字列
>
> 文字列をファイル名とみなして、ファイルから 1 行読み込んで、変数に代入する。読み込みに成功した場合は 1、エンドオブファイルなら 0、ファイルが存在しない場合は −1 を返す。

root : まずは 4 行目の "date"|getline d だけど、これは Unix コマンドの date を実行して、その出力を変数 d に代入するっていう意味だ。

yasuoka : dateって、この date ですか？

```
~/bin% date （ぽこ）
Fri Apr  6 15:23:09 JST 1990
~/bin% 
```

root : そう、その date だ。getline が読み込むのは 1 行分だから、今のだと Fri Apr 6 15:23:09 JST 1990 が、変数 d に文字列として代入される。

yasuoka : わかりました。

root : それから close("date") で、date コマンドからの入力は閉じられて、次の行の split(d,m) で、m[1] に Fri が、m[2] に Apr が、m[3] に 6 が、m[4] に 15:23:09 が、m[5] に JST が、m[6] に 1990 がそれぞれ代入される。

split(文字列,配列) 　　文字列をフィールド毎に区切り、配列[1]、配列[2]、…に代入する。フィールド数を返す。 split(文字列,配列,文字列) 　　右文字列をフィールドの区切りとみなし、他は上に同じ。右文字列は長さが 1 でなければならない。

yasuoka : はい、それはわかります。

root : それから c に sort | sed 's/^[^]* //' が代入されて、8 行目の while だ。while の中の "who"|getline s だけど、最初は

```
yasuoka  console Apr  6 15:19
```

が変数 s に代入されて、値として 1 が帰ってくるので、while の条件が成立する。

yasuoka : getline は読み込みが成功した時は 1 でしたね。

root : そういうこと。次の行の split(s,a) で、a[1] に yasuoka が、a[2] に console が、a[3] に Apr が、a[4] に 6 が、a[5] に 15:19 がそれぞれ代入される。

yasuoka : はい。

root : で、10 行目の printf で

```
10615:19 yasuoka  console Apr  6 15:19
```

が、sort | sed 's/^[^]* //' に出力されるわけだ。

yasuoka : えっと、そこまではわかります。

root : それからまた while に戻って、"who"|getline s で、今度は

```
ochi     ttyh0   Apr  6 11:09
```

が s に代入される。

yasuoka : え？ どういうことですか？

root : 今、who を実行してみてごらん。

yasuoka : はい。

```
~/bin% who（ぽこ）
yasuoka  console Apr  6 15:19
ochi     ttyh0   Apr  6 11:09
takahash ttyh1   Apr  6 13:14
~/bin%
```

root : "who"|getline s を実行するたびに who が実行されるんじゃなくて、一度だけ実行された who から、1 行ずつ文字列が読み込まれるんだ。だから 2 回目の "who"|getline s では

```
ochi     ttyh0   Apr  6 11:09
```

が s に代入される。

yasuoka : パイプが付いた printf と同じですね。

root : 実行が一度だけという意味ではそうかな。読み込みの終了に close を使うしね。

yasuoka : わかりました。

root : それから split して printf で

```
10611:09 ochi     ttyh0   Apr  6 11:09
```

が、sort | sed 's/^[^]* //' に出力される。また while に戻って、今度は

```
takahash ttyh1   Apr  6 13:14
```

が s に代入されるから、printf で出力されるのは

```
10613:14 takahash ttyh1   Apr  6 13:14
```

となる。

yasuoka : これで who からの入力は終りですね。

root : そういうこと。また while に戻って "who"|getline s だけど、もう読み込みができないから s への代入は行なわれなくて、値として 0 が戻ってくる。これで while の条件が成り立たないから、実行は 12 行目に移って、12 行目では who からの入力が閉じられる。

yasuoka : はい。

root : さらに 13 行目では、sort | sed 's/^[^]* //' への出力が閉じられる。ここまでに sort | sed 's/^[^]* //' に出力された文字列は

```
10615:19 yasuoka  console Apr  6 15:19
10611:09 ochi     ttyh0   Apr  6 11:09
10613:14 takahash ttyh1   Apr  6 13:14
```

だから、sort で

```
10611:09 ochi     ttyh0   Apr  6 11:09
10613:14 takahash ttyh1   Apr  6 13:14
10615:19 yasuoka  console Apr  6 15:19
```

になって、 sed 's/^[^]* //' で

```
ochi     ttyh0   Apr  6 11:09
takahash ttyh1   Apr  6 13:14
yasuoka  console Apr  6 15:19
```

になる。これが howってコマンドの出力になってるだろ？

yasuoka : えっと

```
~/bin% how（ぽこ）
ochi     ttyh0   Apr  6 11:09
takahash ttyh1   Apr  6 13:14
yasuoka  console Apr  6 15:19
~/bin% 
```

確かにそうですね。でもこういうやり方だと、 2ヶ月以上 login したままの人がいたら、結果は狂ってきますね。

root : そういうこと。ところでこの howってスクリプト、 $0 への getline を使えばもっと短くなるな。

```
#! /bin/nawk -f
# "how" Version 1.1
BEGIN{
  "date"|getline $0;
  close("date");
  m=$2;
  c="sort | sed 's/^[^ ]* //'";
  while(("who"|getline $0)==1)
    printf("%d%02d%s %s\n",$3==m,$4,$5,$0)|c;
  close("who");
  close(c);
  exit;
}
```

yasuoka : どういうことですか？

root : sub とかと同じで、 getline で $0 に代入すると、同時に $1 や NF も変更されるんだ。

yasuoka : それで split が要らなくなるんですね。わかりました。

（間）

yasuoka : root さん、 root さん。

root : 何だい？

yasuoka : さっきの how を、 2ヶ月以上 login したままの人がいても大丈夫なようにしました。

```
~/bin% cat how（ぽこ）
#! /bin/nawk -f
# "how" Version 1.2
BEGIN{
  n["Jan"]=1;
  n["Feb"]=2;
  n["Mar"]=3;
  n["Apr"]=4;
  n["May"]=5;
  n["Jun"]=6;
  n["Jul"]=7;
  n["Aug"]=8;
  n["Sep"]=9;
  n["Oct"]=10;
  n["Nov"]=11;
  n["Dec"]=12;
  "date"|getline $0;
  close("date");
  m=n[$2];
  c="sort | sed 's/^[^ ]* //'";
  while(("who"|getline $0)==1)
    printf("%02d%02d%s %s\n",(n[$3]-m+11)%12,$4,$5,$0)|c;
  close("who");
  close(c);
  exit;
}
~/bin% how（ぽこ）
ochi     ttyh0   Apr  6 11:09
takahash ttyh1   Apr  6 13:14
```

```
yasuoka  console Apr  6 15:19
~/bin% ■
```

今、そんな人いませんから、うまくいってるのかどうかわかりませんけど。

root : なかなか。nawk の配列をうまく使ったみたいだね。

yasuoka : へへー。

root : でもこれって、 n っていう関数を function で定義した方が、 nawk らしいんじゃないかな。

yasuoka : 関数？

```
#! /bin/nawk -f
# "how" Version 1.3
BEGIN{
  "date"|getline $0;
  close("date");
  m=n($2);
  c="sort | sed 's/^[^ ]* //'";
  while(("who"|getline $0)==1)
    printf("%02d%02d%s %s\n",(n($3)-m+11)%12,$4,$5,$0)|c;
  close("who");
  close(c);
  exit;
}
function n(i){
  return(index("  JanFebMarAprMayJunJulAugSepOctNovDec",i)/3);
}
```

yasuoka : どういうことですか？

root : nawk では自分で関数を定義できるんだよ。

yasuoka : 関数って？

root : sin とか cos とか substr とか index とかみたいに、値を与えたら答が帰ってくるようなやつ。この例では n っていう関数を定義して、例えば n("Apr") とすれば 4 が帰ってくるようにしたんだ。

yasuoka : それが 14 行目の function n(i){ですか？

root : 正確には 14 行目から 16 行目までが、 n っていう関数の定義だ。 nawk ではこんな風にパターンのところに関数定義が書けるんだ。ついでだから、ここで nawk のパターンをまとめておこうか。

	常に実行する
条件	条件が真のとき実行する
条件,条件	左条件が真になってから、右条件が真になるまで
BEGIN	1 行目を入力する前
END	最終行を入力した後
function 関数(変数,…)	関数の定義

条件は以下の通り。

式==式	2 式が等しいとき真
式!=式	2 式が等しくないとき真
式>式	左式が右式より大きいとき真
式>=式	左式が右式以上のとき真
式<式	左式が右式未満のとき真
式<=式	左式が右式以下のとき真
文字列~/正規表現/	文字列が正規表現にマッチするなら真
文字列~文字列	右文字列を正規表現とみなし、他は上に同じ
文字列!~/正規表現/	文字列が正規表現にマッチしないなら真
文字列!~文字列	右文字列を正規表現とみなし、他は上に同じ
!条件	条件が真でないとき真
条件&&条件	2 条件の両方が真のとき真
条件||条件	2 条件のどちらかが真のとき真
(条件)	優先度を変える

root : 本当は「式は全て条件として使用できる」と言いたいところだけど、ヴァージョンによっては、これ以外が動かない場合もあるからね。

yasuoka : そうなんですか。で、元の how に戻りますけど。

root : うん。

yasuoka : 6 行目の m=n($2) ではどういう動作が起こるんですか？

root : 14 行目の function n(i) へ実行が移るんだ。この時 i には $2 の値が代入されて、実行が始まる。今だったら Apr っていう文字列が i に代入されるわけだね。

yasuoka : はい。

root : で、 15 行目だけど、 return ってのはその中の式の値を関数の戻り値として、関数の実行を終了するっていう意味なんだ。

```
return(式);
```
式の値を関数の戻り値として、関数の実行を終了する。

yasuoka : 戻り値って？

root : ここだと `index("  JanFebMarAprMayJunJulAugSepOctNovDec",i)/3` を計算した結果を、`n(i)` の値とするってことだ。`i` が `Apr` なら 4 だね。で、4 を持って `m=n($2)` のところに戻って、`m` に 4 が代入されるわけだ。

yasuoka : それで `n[$2]` の代わりになるわけですか。9 行目も同じですよね。

root : そういうこと。

yasuoka : 関数の中には `return` しか書けないんですか？

root : いやどんな命令でも書ける。それから持っていく変数も複数書けるよ。

yasuoka : 持っていく変数って？

root : `n(i)` だけじゃなくて、`n(i,j)` みたいなのも書けるってこと。そうだな、この `how` の `m=n($2)` と `(n($3)-m+11)%12` を一度に計算するような関数を定義して、`how` を書き換えてみようか。

```
#! /bin/nawk -f
# "how" Version 1.4
BEGIN{
  t="JanFebMarAprMayJunJulAugSepOctNovDec";
  "date"|getline $0;
  close("date");
  m=$2;
  c="sort | sed 's/^[^ ]* //'";
  while(("who"|getline $0)==1)
    printf("%02d%02d%s %s\n",submonth($3,m),$4,$5,$0)|c;
  close("who");
  close(c);
  exit;
}
function submonth(i,j){
  return(((index(t,i)-index(t,j))/3+11)%12);
}
```

yasuoka : 10 行目の `submonth($3,m)` が、15 行目の `function submonth(i,j)` を呼びだすんですよね。

root : そう。

yasuoka : その時は `$3` の値が `i` に、`m` の値が `j` に入るんですよね。

root : そういうこと。

yasuoka : じゃあ、16 行目の `((index(t,i)-index(t,j))/3+11)%12` にある `t`って変数は？

root : 4 行目で代入してる `t`って変数だよ。関数の中では、関数定義に出てきた変数は関数内だけで使われるけど、それ以外の変数は普通に使われるんだ。

yasuoka : どういうことですか？

root : 例えばこの `how` の 15 行目と 16 行目の間で、`i`って変数に代入しても、それは関数の中でだけ有効で、外の変数には何ら影響を与えない。`j` についても同様だ。

yasuoka : はい。

root : でも `function submonth(i,j)` の定義に出てこない他の変数、例えば `t` とかいう変数に代入すると、それは関数の外の変数に代入される。つまり 4 行目で代入してる `t` の値が変わってしまうんだ。

yasuoka : うーん、わかったようなわからないような。

root : そうだな、これについてはもう少し細かい説明が必要なんだけど、今日はもうこんな時間だ。関数定義に出てくる変数については、また次回にしてくれるかい？

yasuoka : はい。どうもありがとうございました。