第5章

AWK

5.1 AWK について

awk は Alfred V. Aho、Peter J. Weinberger、Brian W. Kernighan により 1977 年に設計作成され、その後、いくつかの拡張がなされて 1985 年に nawk となる。

awk は入力パターンに応じて様々なアクションを実行することによりデータ処理からレポートの作成まで広範囲の仕事を簡単に行うことのできるテキスト処理言語である。

5.2 AWK のプログラム

5.2.1 awk プログラムの構成

awk プログラムは、入力データに対する検索パターンと、パターンに適合したデータに対して実行するアクションを並べたものである。アクションは { と } で囲む。

パターンとアクションの内、どちらか一方を省略することもできる。パターンを省略した場合は、全ての入力データに対してアクションが実行される。この場合は { と } を省略できる。アクションを省略した場合は、パターンに適合したデータを標準出力に出力する。

pattern { *action* }
action
pattern

5.2.2 AWK の起動法

awk はパターンやアクションからなる awk プログラムと処理したいデータが格納されたデータファイル名を指定して起動する。データファイル名を省略すると標準入力からデータが読み込まれる。

awk プログラムの指定には二つの方法がある。

1. `awk [-F`*s*`] 'awk プログラム' [データファイル]`
2. `awk [-F`*s*`] -f awk プログラムファイル [データファイル]`

（[ ] に囲まれた部分は省力可能）

1. の形式は awk プログラムをコマンドラインから与える方法で、awk プログラムを' と' で囲む。

2. の形式は、あらかじめ awk プログラムをファイル中に作成しておく方法で、`-f` の後に awk プログラムのファイル名を指定する。awk プログラムが複数行で構成される場合や、よく使用するプログラムを保存しておきたい場合などに使用する。

オプション-F は入力データの欄区切子 (FS) を指定する。省略したときはスペースが使用される。

5.2.3 AWK プログラムの例

awk を使った簡単な例を紹介する。データファイルとして次のファイルを使用する。

```
$ ls -l > test.data
$ cat ls-l
-rw-r--r--     1  16647 Nov 14 08:51 1993 .cshrc
-rw-r--r--     1  37017 Nov 14 08:51 1993 .login
-rw-r--r--     1  12595 Nov 14 08:51 1993 .mailrc
drw-r--r--     2    512 Nov 14 08:52 1993 Mail
-rw-r--r--     1   6106 Nov 14 06:31 1994 awk.c
-rw-r--r--     1   2168 Jul 27 10:32 1994 file1
-rw-r--r--     1  10721 Jul 28 11:33 1994 file2
-rw-r--r--     1  17467 Jul 29 12:54 1994 file3
-rwxr-xr-x     1  10555 Nov 14 19:01 1994 fterm*
drw-r--r--     2   1024 Nov 14 06:27 1994 pub
-rw-r--r--     1   1248 Nov 14 06:00 1994 time.c
```

■ 5 行目の入力行を表示

```
$ awk 'NR == 5' test.data
-rw-r--r--     1   6106 Nov 14 06:31 1994 awk.c
```

システム変数 NR は、入力した行の数を返す。

■入力行の最後の欄を表示

```
$ awk '{ print $NF }' test.data
.cshrc
.login
.mailrc
Mail
awk.c
file1
file2
file3
fterm*
pub
time.c
```

システム変数 NF は入力行に含まれる欄の数を返す。この例では、欄の数は 8 である。一方$$n$ は n 番目の欄を値に持つ変数で、$NF はこの場合$8 となり 8 番目の欄（つまり最後の欄）を値に持つ変数となる。

■ファイルサイズの合計を求める

```
$ awk '{ sum += $3 } END { print sum }' test.data
116060
```

END {*action*} は、全てのデータが読み込まれた後で 1 回だけ *action* が実行される。

■ファイルサイズが 10240 以上の入力行を表示

```
$ awk '$3 > 10240 { print }' test.data
-rw-r--r--     1  16647 Nov 14 08:51 1993 .cshrc
-rw-r--r--     1  37017 Nov 14 08:51 1993 .login
-rw-r--r--     1  12595 Nov 14 08:51 1993 .mailrc
-rw-r--r--     1  10721 Jul 28 11:33 1994 file2
-rw-r--r--     1  17467 Jul 29 12:54 1994 file3
-rwxr-xr-x     1  10555 Nov 14 19:01 1994 fterm*
```

print で引数を省略すると、$0(入力行の全ての欄) と解釈される。

■ファイル名に file という文字列を含む入力行を表示

```
$ awk '$NF ~ /file/ { print }' test.data
-rw-r--r--     1   2168 Jul 27 10:32 1994 file1
-rw-r--r--     1  10721 Jul 28 11:33 1994 file2
-rw-r--r--     1  17467 Jul 29 12:54 1994 file3
```

■欄区切子 (FS) の変更

```
$ awk '{ col = NF }{ char = $NF } END { print col, char }' test.data
8 time.c
$ awk -F: '{ col = NF }{ char = $NF } END { print col, char }' test.data
2 00 1994 time.c
```

欄区切子 (FS) の指定を省略すると、空白が欄区切子となる。従って欄の数は８である。２番目の例では、:を欄区切子に指定している。この場合は欄の数が２となる。

■ MS-DOS 風に表示

```
$ cat dir.awk
{ sub(/Jan/,"01") } # sub は置換を行う。ここでは Jan を 01 に置換えている。
{ sub(/Feb/,"02") }
{ sub(/Mar/,"03") } # #は注釈マークである。#から改行までを awk は無視する。
{ sub(/Apr/,"04") }
{ sub(/May/,"05") }
{ sub(/Jun/,"06") }
{ sub(/Jul/,"07") }
{ sub(/Aug/,"08") }
{ sub(/Sep/,"09") }
{ sub(/Oct/,"10") }
{ sub(/Nov/,"11") }
{ sub(/Dec/,"12") }
{
if ( $1 ~ /d/ )
    printf("%-13s %-8s %s-%s-%s %7s\n",$NF, "<DIR>", $7-1900, $4, $5, $6)
else
    printf("%-13s %8s %s-%s-%s %7s\n",$NF, $3, $7-1900, $4, $5, $6)
}
```

```
$ awk -f dir.awk test.data
.cshrc          16647 93-11-14  08:51
.login          37017 93-11-14  08:51
.mailrc         12595 93-11-14  08:51
Mail         <DIR>    93-11-14  08:52
awk.c            6106 94-11-14  06:31
file1            2168 94-07-27  10:32
file2           10721 94-07-28  11:33
file3           17467 94-07-29  12:54
fterm*          10555 94-11-14  19:01
pub          <DIR>    94-11-14  06:27
time.c           1248 94-11-14  06:00
```

■入力行の欄を逆順に表示

```
$ awk 'END { for(i=NF;i>0;i--) printf("%-8s", $i) }' test.data
time.c 1994 06:00 14 Nov 1248 1 -rw-r--r--
```

5.3 AWK リファレンス

5.3.1 パターン

BEGIN	入力を読み込む前
END	全ての入力を読み終えた後
式	式が真 (0 でも空列でもない) のとき
/正規表現/	正規表現に適合するような文字列を含むとき
複合パターン	&&(AND)、||(OR)、!(NOT)、括弧で結合したパターン
パターン 1, パターン 2	範囲を示すパターン。パターン 1 に適合する行からパターン 2 に適合する行まで

5.3.2 アクション

アクションはブレース（{ , }）で囲まれた 1 つ以上の文または式により構成される。以下の文や関数の中には awk では利用できず、nawk または gawk だけで利用できるものが含まれている。

atan2(*y, x*)	*y*/*x* の逆正接をラジアンで返す
break	ループ (while, for) から抜ける
close(*expr*)	ファイルやパイプのクローズ
continue	ループの次の繰返しを開始
cos(*x*)	*x* の余弦を返す
delete(*array*[*dlement*])	配列の要素を削除
do *body* while(*expr*)	*body* 中の文を実行後 *expr* を評価し、それが真ならば *body* を再実行
exit	これ以降の入力を評価せずスクリプトから抜ける
exp(*x*)	*e* の *x* 乗を返す

for(*init;test;incr*) *statement*	*init* はカウンタ変数の初期値を代入。*test* は文を実行前に毎回評価する関係式。*test* が偽になるとループが終了する。*incr* はカウンタ変数をインクリメントするのに使う
for(*item* in *array*) *statement*	連想配列読込み用。配列の各要素ごとに *statement* を実行。要素は array[*item*] を参照
getline[*var*][<*file*]	次の入力行を読む。*file* から入力を 1 行ずつ$0, NF, NR, FNR を読込み、文が実行されるたびに次の行を読込む。*var* が指定されると結果を *var* に代入し $0 を変更しない
gsub(*r, s, t*)	文字列 *t* の中から正規表現 *r* に適合するものをグローバルに *s* に置換える。*t* を指定しなければ$0 が対象となる。 置換した数を返す
if(*expr*) *statement1* [else *statement2*]	*expr* を評価し、真なら *statement1* を実行し、else があれば偽のとき *statement2* を実行
index(*substr, str*)	文字列 *str* から部分文字列 *substr* の位置を返す。存在しなければ 0 を返し、*str* が省略されたら$0 を使う
int(*x*)	*x* の小数点を切捨てた値を返す
length(*str*)	文字列 *str* の長さを返す
log(*x*)	*x* の対数を返す
match(*s, r*)	文字列 *s* の中から正規表現 *r* のパターンを探す。あれば *s* 内の位置を返し、なければ 0 を返す。RSTART と RLENGTH の値をセット
next	次の入力行を読込み、スクリプトの先頭から実行を再開
print[*output*]	*output* を標準出力に改行と共に出力する。*output* を省略すると$0 が出力
printf(*format, list*)	書式化された出力を改行なしで生成 *format* はフォーマット指定子と定数の文字列。*list* はフォーマット指定子に対応した引数リスト
rand()	0 と 1 の間の乱数の生成
return(*expr*)	ユーザ定義関数の終了した式の値を返す
sin(*x*)	*x* の正弦を返す
split(*str, array, sep*)	*sep*(FS) を区切子にして *str* を配列 *array* に分解し欄数を返す

sprintf(*format, list*)	*printf* のフォーマット指定に従って書式化した文字列を返す。データは書式化されるが出力はされない
sqrt(*x*)	*x* の平方根を返す
srand(*expr*)	*expr* を使って乱数ジェネレータ用の新しいシードを設定する。デフォルトはその時の時刻
sub(*r, s, t*)	*t* の部分文字列で *r* に最初に適合するものを *s* で置換
substr(*str, beg, len*)	文字列 *str* の先頭位置 *beg* から始る *len* に指定された文字数分の長さの部分文字列を返す
system(*command*)	指定の *command* を実行し、その終了ステータスを返す。ステータスは成功 (1)、完了 (0)、エラー (-1) を返す。コマンドの出力をスクリプトに読込むには *command* | getline を使う
tolower(*str*)	*str* 中の文字を大文字から小文字に変える (gawk のみ)
toupper(*str*)	*str* 中の文字を小文字から大文字に変える (gawk のみ)
while(*expr*) *statement*	*expr* が真である限り *statement* を実行

■ printf のフォーマット指定子

printf フォーマット指定子	
%c	ASCII 文字
%d	10 進整数
%e	浮動小数点形式 [-]d.ddddddE[+-]dd
%f	浮動小数点形式 [-]ddd.dddddd
%g	e 変換か f 変換の短い方で末尾の 0 を省略
%o	符号なし 8 進数
%s	文字列
%x	符号なし 16 進数
%%	%の出力

%と制御文字の間に以下のパラメータをつけることができる。

-	欄の中で値を左に揃える
n	欄の幅を *n* 文字にする
.*m*	文字列の最大幅、もしくは小数点以下の桁数

5.3.3 正規表現

c	メタキャラではない任意のリテラル文字 *c* に適合
¥	¥自身と ¥に続くメタキャラに適合
^	これに続く正規表現と行の先頭に適合
$	これに続く正規表現と行の末尾に適合
.	改行を除く任意の 1 文字に適合
[ ]	[ ] に囲まれた文字のいずれかに適合。文字の先頭に^があると反対の意味になる。文字の範囲は-で表す
r1|*r2*	*r1* か *r2* のどちらかに適合すれば適合 (選択)
(*r1*)(*r2*)	*r1* が *x* に、*r2* が *y* に適合するとき *xy* に適合 (連接)
r∗	*r* に適合する 0 個以上の文字列に適合
r+	*r* に適合する 1 個以上の文字列に適合
r?	*r* に 0 個か 1 個の文字列に適合
(*r*)	*r* に適合する文字列に適合 (グループ化)

■正規表現の例

¥$	文字$
^C	先頭に文字 C を含む文字列
C$	末尾に文字 C を含む文字列
^C$	文字 C 1 文字からなる文字列
^.$	任意の 1 文字からなる文字列
^...$	任意の 3 文字からなる文字列
...	任意の連続した 3 文字以上の文字列
^[ABC]	文字列の先頭に文字 A,B,C いずれかを含む文字列
^[^ABC]	文字列の先頭に文字 A,B,C いずれも含まない文字列
[^ABC]	文字 A,B,C いずれも含まない文字列
^[^a-z]$	英小文字以外の 1 文字からなる文字列
(A|B)(C|D)	AC,AD,BC,BD を含む文字列
B∗	空列、B、BB など
AB∗C	AC、ABC、ABBC など
AB+C	ABC、ABBC、ABBBC など
ABB*C	ABC、ABBC、ABBBC など
AB?C	AC または ABC
[A-Z]+	英大文字からなる 1 文字以上の文字列
(AB)+C	ABC、ABABC、ABABABC など
^[0-9]+$	数字だけからなる文字列

5.3.4 演算子

= += -= *= /= %= ^=	代入
?:	条件式
||	論理和 (OR)
&&	論理積 (AND)
in	配列要素
~ !~	正規表現適合、非適合
< <= > >= != ==	比較
+ -	和、差
* / %	積、商、剰余
+ - !	単項和、単項差、論理否定 (NOT)
^	累乗
++ --	インクリメント、デクリメント (前置、後置可能)
$	欄参照

5.3.5 システム変数

- **awk**

FILENAME	現在の入力ファイル名
FS	入力欄区切子 (デフォルトは空白)
NF	カレントレコードの欄数
NR	カレントレコード番号
OFS	出力欄区切子 (デフォルトは空白)
ORS	出力レコード区切子 (デフォルトは改行)
RS	入力レコード区切子 (デフォルトは改行)

- **nawk**

ARGC	コマンドライン引数の数
ARGV	コマンドライン引数の入った配列
FNR	カレントレコードの相対番号
OFMT	数の出力書式 (デフォルトは%.6g)
RSTART	match 関数で適合した文字列中の最初の位置
RLEGTH	match 関数で適合した文字列の長さ
SUBSEP	配列添字の区切り文字 (デフォルトは ¥034)

- **gawk**

ENVIRON	環境変数の連想配列
IGNORECASE	パターンマッチで文字の大小を区別しない

5.3.6 エスケープシーケンス

¥a	警告文字。通常はビープ音	¥t	水平タブ
¥b	バックスペース	¥v	垂直タブ
¥f	フォームフィード	¥*ddd*	1 から 3 桁までの 8 進数
¥n	改行	¥x*hex*	16 進数
¥r	復帰		

5.3.7 AWK の制限

欄の数	100 以内
レコードあたりの文字数	3000 文字以下
欄あたりの文字数	1024 文字以下
printf 文字あたり 3000 文字	3000 文字以下
リテラル文字の文字数	400 文字以下
文字クラスの文字数	400 文字以下
オープンできるファイルの数	15 以内
パイプの数	1
倍精度浮動小数点	可
全角文字	不可