第 2 章

AWK

AWK はプログラミング言語の一種であるが、ユニケージではパイプの途中でフィルタとして用いる。本章では、そのような用途に特化して AWK の使い方を身に付ける。

2.1 AWK とは何か

AWK（オーク）は、UNIX 系 OS には必ず付属しているツールである。公表されたのは 1977 年で、開発したのはベル研で UNIX 開発に携わったメンバー：

- Alfred Aho
- Peter Weinberger
- Brian Kernighan

である。3 人のファミリーネームの頭文字から「AWK」と名づけられた。

AWK の特徴は、最初から空白区切りのデータを処理することを想定していることである。例えば、以下のように書くと、/etc/resolv.conf の二列目（DNS サーバの IP アドレスが書いてある）が取得できる。

```
1 [hoge@lecture ~]$ cat /etc/resolv.conf | awk '{print $2}'
2                                           ----------
3                                           この部分がプログラム
```

```
8.8.8.8
8.8.4.4
```

このように、非常に短いコードでテキストを切り出すことができる。なぜ短いかというと、以下の処理が暗黙裡に行われているからである。

- テキストを一行ずつ処理する。
- 一行が分解されて、$1, $2 のような変数に格納される。

usp Tukubai もそのような設計になっているので、非常に親和性が高い。

2.2 手を動かす

【TRY】　とりあえず意味を考えずに手を動かしてみよう。

まず、vi（vim）で以下のファイル（hoge）を作る。左から順に日付、品名、売価、売れた数量という想定である。

```
20110302 バナナ 100 4
20110302 リンゴ 85 5
20110303 リンゴ 80 9
20110303 ナシ 124 3
20110304 バナナ 95 5
20110305 ナシ 120 13
```

```
#各レコードの売上を計算
[hoge@lecture ~]$ cat hoge | awk '{print $1,$2,$3*$4}'
20110302 バナナ 400
20110302 リンゴ 425
20110303 リンゴ 720
20110303 ナシ 372
20110304 バナナ 475
20110305 ナシ 1560
#以下でも同じ結果が得られることを一つずつ試す。
awk '{print $1,$2,$3*$4}' hoge
awk '{print $1,$2,$3*$4}' < hoge
[hoge@lecture ~]$ cat hoge |\
          awk '{uriage=$3*$4;print $1,$2,uriage}'
20110302 バナナ 400
20110302 リンゴ 425
```

```
20110303 リンゴ 720
20110303 ナシ 372
20110304 バナナ 475
20110305 ナシ 1560
#リンゴのレコードだけ抽出
[hoge@lecture ~]$ cat hoge | awk '$2=="リンゴ"'
20110302 リンゴ 85 5
20110303 リンゴ 80 9
#3月3日以前のレコードを抽出
[hoge@lecture ~]$ cat hoge | awk '$1<="20110303"'
20110302 バナナ 100 4
20110302 リンゴ 85 5
20110303 リンゴ 80 9
20110303 ナシ 124 3
#3月3日以前のレコードを抽出し、日付、品目、個数を表示
[hoge@lecture ~]$ cat hoge | awk '$1<="20110303"{print $1,$2,$4}'
20110302 バナナ 4
20110302 リンゴ 5
20110303 リンゴ 9
20110303 ナシ 3
```

2.3 フィールド・レコード／パターン・アクション

まずAWKを理解するために「レコード」と「フィールド」という用語について整理する。

レコードは、テキストファイルの一行のことである。上から「第一レコード」、「第二レコード」と数える。下に、先ほどのTRYで使ったテキストを再度記載する。この例の場合、20110302 バナナ 100 4が一つのレコードである。

フィールドは列を指す。区切り文字で各レコードを区切ったとき、左のデータから「第一フィールド」、「第二フィールド」と数える。下の例では半角スペースが区切り文字で、日付が第一フィールド、品目が第二フィールドに記述されている。

```
20110302 バナナ 100 4
20110302 リンゴ 85 5
20110303 リンゴ 80 9
20110303 ナシ 124 3
20110304 バナナ 95 5
```

第 3 章

sort, msort*, uniq

3.1 sort

sort（ソート）は、シェルスクリプトを書く際に非常によく用いるコマンドである。ここでは、スクリプト中で適切に sort コマンドを使いこなすために、挙動を詳しく説明していく。

3.1.1 オプションなしでの動作とソートの順番

sort コマンドは、各コマンドを辞書順に並べる。以下に例を示す。[*1]

```
[hoge@lecture ~]$ cat hoge
a
b
あ
1
~
山
c
2
.
```

[*1] TEX でテキストを作成している都合上、半角カナが出力できないため、（半角）と印を入れる。

```
ア
川
10
A
ア　（半角）
C
B
[hoge@lecture ~]$ sort hoge
.
1
10
2
A
B
C
a
b
c
~
ア　（半角）
あ
ア
山
川
```

辞書順であるので、1→10→2 という順序になる。数字の大小でソートするためには、オプションをつける必要がある。

【発展】~と. がアルファベットの両側に分かれるのは、ASCII コードの配置による。

sort を使う上で注意すべきことは、環境によって「辞書順」が一致していないことである。例えば、別の環境では以下のように sort の結果が異なる。

```
hoge@otherenv:~$ sort hoge
.
1
10
2
A
B
C
a
```

```
b
c
~
あ        <- 順序が前の例と異なる
ア
山
川
ア　（半角）
```

以下のように書くと、（テキストの中の文字が同じ文字コードならば）どのような環境でもソート順が一致する。

```
#LANG=C と書く
[hoge@lecture ~]$ LANG=C sort hoge
.
1
10
2
A
B
C
a
b
c
~
あ
ア
山
川
ア　（半角）
```

LANG=C をコマンドの前に書くと「言語の違いに関係なくコマンドを動かす」という意味になる。sort に LANG=C をつけた場合、ただテキストをバイナリとして扱ったときの順にソートするようになる。

usp Tukubai のコマンドのいくつかは、二つ以上のファイルを入力として受け入れるが、その中の多くのコマンドで、各々のファイルが同じルールでソートされていることを前提としている。そのため、sort の前に必ず LANG=C とつけて、ソート順を統一する必要がある。

【発展】LANG は使用する文字コードを指定するための環境変数である。

第 5 章

self*, delf*

5.1 フィールド形式とキー

ユニケージでは、フィールド形式と呼ばれるファイル形式を最も多用する。フィールド形式は、これまで扱ってきたファイルのように、空白区切りでフィールドに値を入れたものである。また、フィールド数はどのレコードも同一であるというルールがあり、大半のコマンドはフィールド数がまちまちなファイルの処理を想定していない。

さらに、フィールド形式でデータを保存する場合、ファイルの左側にキーを揃えておくと、self を介さずに join（後述）などのコマンドに入力できる場合が多い。例えば、小売チェーンのデータファイルでは、店番号やエリアコード、日付などが左側にくる。

```
#フィールド形式で保存されたデータの例
#第 1 フィールド（第 1 キー）：エリアコード（東北地区、北陸地区、等の識別番号）
#第 2 フィールド（第 2 キー）：店番号
#第 3 フィールド（第 3 キー）：年月日
#第 4 フィールド：売上
[hoge@lecture ~]$ cat AREA.TEN.DAY.URIAGE
001 0002 20110304 9438213
002 0012 20110302 23422303
002 0013 20110305 32439123
003 0042 20110306 102231004
004 0052 20110301 10430032
```

この例の類のデータでは、場合によっては日付が第 1 キーになる。

```
#第 1 フィールド（第 1 キー）： 年月日
#第 2 フィールド（第 2 キー）： エリアコード（東北地区、北陸地区、等の識別番号）
#第 3 フィールド（第 3 キー）： 店番号
#第 4 フィールド：売上
[hoge@lecture ~]$ cat DAY.AREA.TEN.URIAGE
20110301 004 0052 10430032
20110302 002 0012 23422303
20110304 001 0002 9438213
20110305 002 0013 32439123
20110306 003 0042 102231004
```

いずれの場合も、特別な事情が無い場合は第 1 キーから順にソートしておく。ソートするのは、

- 後の処理を簡略化する
- データを読みやすくする

ためである。

usp Tukubai のコマンドの多くは左側にキーがあることを前提に処理を行う。そのため、値のフィールドがキーよりも左側にあるファイルを処理する場合、処理のたびにソートしたりフィールドを入れ替えたりする必要が発生することがある。データの整理がおろそかになりかねないので、コマンドに融通を利かせるということはしない。

5.2 self によるフィールドの並び替え

5.2.1 基本的な使い方

ユニケージでは、上の例のようなキーの左寄せや順序変更、その他演算のためにフィールドの並び替えが発生する。フィールドの並び替えは awk でも可能であるが、usp Tukubai のコマンドである self を使うとより簡潔に操作を記述できる。

self の基本的な使い方は、以下の通り。self の後に、並び替えたい順にフィールドの数を入れていく。

```
#標準入力の第 1, 2, 3, 4 フィールドを第 4, 3, 2, 1 フィールドに並び替える。
[ueda@lecture ~]$ echo a b c d | self 4 3 2 1
```

```
d c b a

#ファイルをオプションの最後に指定することもできる。
[ueda@lecture ~]$ echo a b c d > hoge
[ueda@lecture ~]$ self 4 3 2 1 hoge
d c b a
```

【TRY】フィールド形式のデータの整頓

```
#上の例の売上ファイルを適当に並び替えて以下のような整理されていないファイルを作る。
[hoge@lecture ~]$ cat URIAGE
9438213 001 20110304 0002
102231004 003 20110306 0042
23422303 002 20110302 0012
10430032 004 20110301 0052
32439123 002 20110305 0013

#エリアコード、店番号、日付順にならべ、ソートする。
[ueda@lecture ~]$ cat URIAGE | self 2 4 3 1 | LANG=C sort -k1,3
001 0002 20110304 9438213
002 0012 20110302 23422303
002 0013 20110305 32439123
003 0042 20110306 102231004
004 0052 20110301 10430032

#日付、エリアコード、店番号順にならべ、ソートする。
[ueda@lecture ~]$ cat URIAGE | self 3 2 4 1 | LANG=C sort -k1,3
20110301 004 0052 10430032
20110302 002 0012 23422303
20110304 001 0002 9438213
20110305 002 0013 32439123
20110306 003 0042 102231004
```

【発展】 上の整理されていない URIAGE ファイルは、以下のように作成された。

```
[hoge@lecture ~]$ cat DAY.AREA.TEN.URIAGE | self 4 2 1 3 |\
          awk '{print $0,rand()}' | sort -k5,5nr |\
          self 1/4 > URIAGE
```

第 6 章

sm2*, sm4*, sm5*

sm2,4,5 は、キー別に数字を足しあげるコマンドである。sm2,4,5 をまとめて sm シリーズと呼ぶ。

6.1 sm2

sm2 は、同一キーのレコードの数字を合計するコマンドである。例えば、下のファイルのように、品別、日別に売れた芋の個数が書いてあるファイルがあるとする。

```
#1:品番, 2:品名 3:日付 4:個数 5:売上
[hoge@lecture ~]$ cat URE.IMO
001 さつまいも 20110405 32 1600
001 さつまいも 20110406 24 1200
001 さつまいも 20110407 49 2450
002 じゃがいも 20110405 102 5100
002 じゃがいも 20110406 98 4900
002 じゃがいも 20110407 121 6050
```

このとき、品別でいくつ売れたか知りたいときは、以下のように打てばよい。

```
#第 1 第 2 フィールドをキーにして第 4 第 5 フィールドを合計
[hoge@lecture ~]$ cat URE.IMO | sm2 1 2 4 5
001 さつまいも 105 5250
002 じゃがいも 321 16050
```

```
#これでもよい
[hoge@lecture ~]$ sm2 1 2 4 5 URE.IMO
001 さつまいも 105 5250
002 じゃがいも 321 16050
```

数字のオプションが 4 個あって最初は分かりづらいが、以下のような意味になる。

- 最初の二個：キーとする範囲
- 次の二個：合計する対象のフィールドの範囲

キー、合計対象のフィールド共、フィールド数が一つしかないときは「1 1」のように同じ数字を並べる。特別な指定として、最初の二個のキーを 0 0 とすると、キーを無視して合計する。

```
#第 1 フィールドだけキーとして第 4 フィールドを足す。
[hoge@lecture ~]$ cat URE.IMO | sm2 1 1 4 4
001 105
002 321
#第 4 フィールドを合計する。
[hoge@lecture ~]$ cat URE.IMO | sm2 0 0 4 4
426
```

【TRY】

URE.IMO ファイルについて、第三フィールドの日付別に個数、売上を合計してみよう。

```
[hoge@lecture ~]$ cat URE.IMO | self 3/5 |\
          LANG=C sort -k1,1 | sm2 1 1 2 3
20110405 134 6700
20110406 122 6100
20110407 170 8500
```

sm2 には、+count というオプションが存在する。+count を付けると、集計したレコードの数が、キーと合計のフィールドの間に出力される。

```
[ueda@lecture ~]$ cat URE.IMO | sm2 +count 1 2 4 5
001 さつまいも 3 105 5250
002 じゃがいも 3 321 16050
```

この機能は、主に平均値を計算するために用いられる。

```
#個数と売上の日平均を計算する。
[ueda@lecture ~]$ cat URE.IMO | sm2 +count 1 2 4 5 |\
        awk '{print $1,$2,$4/$3,$5/$3}'
001 さつまいも 35 1750
002 じゃがいも 107 5350
```

6.2 sm4

以下のような小計つきの帳票を作るには、sm4 を使う。

```
1日目 A店 10
1日目 B店 20
------------
1日目 計  30
------------
2日目 A店 30
2日目 B店 40
------------
2日目 計  70
------------
```

例えば、URE.IMO ファイルについて、品目別に sm4 を適用すると以下のようになる。

```
[hoge@lecture ~]$ cat URE.IMO | sm4 1 2 3 3 4 5
001 さつまいも 20110405 32 1600
001 さつまいも 20110406 24 1200
001 さつまいも 20110407 49 2450
001 さつまいも @@@@@@@@ 105 5250
002 じゃがいも 20110405 102 5100
002 じゃがいも 20110406 98 4900
002 じゃがいも 20110407 121 6050
002 じゃがいも @@@@@@@@ 321 16050
```

オプションが 6 個あるが、左から以下のような意味となる。

- 最初の 2 個：キー（sm2 と同じ）
- 次の 2 個：無視するフィールド
- 最後の 2 個：集計対象

上の例では、第 1 第 2 フィールドをキーとして、第 3 フィールドの日付はキーでも集