プログラミング環境 第5回

久野 靖 *

1991.10.29

7 フィルタを中心とするUnixのユティリティ

7.1 フィルタの概念

Unix ではフィルタとは「標準入力チャネルから文字を読み、加工して、標準出力チャネルに送り出す」プログラムのことである (フィルタという名前は「流れが入ってきて、何らかの加工があり、出ていく」という実世界のフィルタの類推から来ている)。

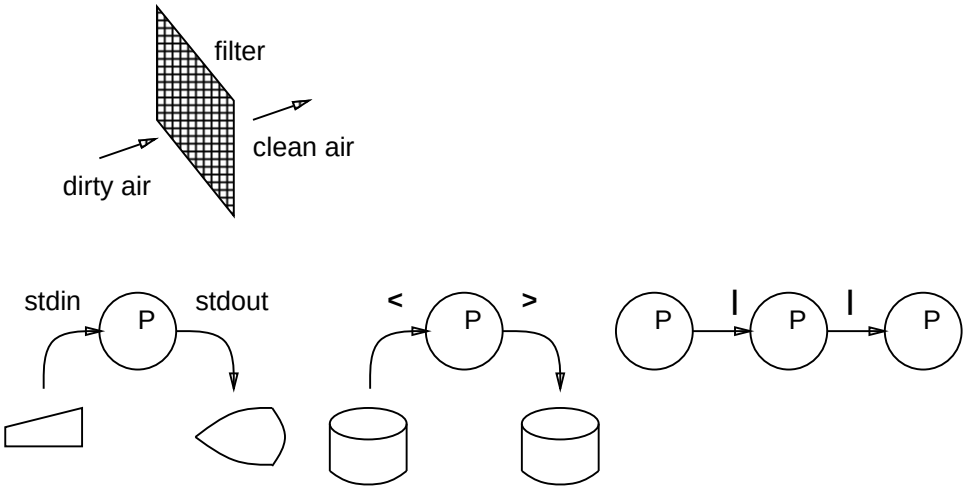


図 1: フィルタの概念

これがなぜ重要なのだろうか。まず図 1 左にあるように、特に何も指定しない状態では標準入力と標準出力はそれぞれキーボードと端末画面につながっている。これは普段指令を単独で使う状況である。次に、この入力と出力をそれぞれファイルから読む/ファイルへ書くように切替えることが独立して指定可能である。これは大量のデータを予めファイルに入れておいて処理させたり、結果を保存しておきたい場合に役に立つ。そしてさらに、あるプログラムの出力を別のプログラムの入力につなげることもできる。これを沢山直列にしたもの:

```
指令 | 指令 | ... | 指令
```

を「パイプライン」と読んでいる。これによって、一つのデータの流れをどんどん加工して処理していくようなことが簡単に指定できる。逆に言えば、一つ一つの加工は単純なものでも、それを自在に組み合わせて多様な加工が行なえる。この辺が Unix の特徴の 1 つであるとされている。

以下ではフィルタを中心に、そのような組み合わせにおいて活用される指令を紹介していく。ただし、スペース/時間の関係上、それぞれの指令についてそんなに詳しくは説明できない。細部はマニュアルページを打ち出してじっくり読んで欲しい。

*筑波大学経営システム科学専攻

7.2 cat – ファイルの連結

これまで、cat というのは何の指令だと思っていましたか?猫 (うそ)? ファイルを画面に表示?違うな。実は。

```
cat ファイル ...
```

というのは、指定された複数のファイルを連結して、標準出力に流す、という指令なのでした。だからファイルを 1 つだけ指定すると、そのファイルのみが標準出力に出るので、画面に見える。じゃあ、ただ

```
cat
```

だけだと?ファイルを指定しないと、標準入力から読んだものをそのまま標準出力に流す。ので、ただ「cat[ret]」だとキーボード入力待ちになって、一見何も起きない。これではまった人はいませんか? それはともかく、Unix のフィルタはファイル名を指定するとそこから入力するがファイル名がないと標準入力から読むのでフィルタとして使える、という風になっているのが普通である。

ところで、それじゃcat はファイルは連結できるがフィルタとして何の役にも立たないじゃない、というとさにあらず。実はファイル名のところに-を指定するとそれは標準入力という意味になる。だから、例えば次のような使い方が可能。

```
% cat t
-----------------------------------  ←マイナスいっぱいのファイル
% ps | cat t - t                     ← ps 出力の上下に横線を。
-----------------------------------
  PID TT STAT  TIME COMMAND
23529 p1 R     0:00 ps
23530 p1 S     0:00 cat t - t
22619 p1 S     0:04 -usr/new/csh (csh)
-----------------------------------
%
```

7.3 ページャ – 止まりながら読む

長いファイルを cat などで画面に表示するとどんどん流れていってしまって読めない、というのを解消するために、1 ページずつ止まりながら表示する指令がある。これを「ページャ」などと呼ぶ。

```
more  ファイル名    -- 1 ページずつ止まりながら読む
less  ファイル名    -- 同上、ただし戻ったりもできる
```

これらはいずれも 1 画面が一杯になったところで表示が止まり、[SP] で次の画面へ進む。途中で止めたくなったら「q」で中止できる (less では最後まで来た時でも「q」を打つまでは終らない)。さらに less では「b」で 1 画面戻ることができる。

ところで、複数の指令を組み合わせて加工した結果をそのまま 1 ページずつ止まって読みたい場合にもこれらを使うことができる。つまり、

```
... | less
```

のようにパイプラインの最後に使えばよいわけである。

7.4 tee – パイプの途中

ページャをパイプラインの中ほどに入れるのは意味がないが、パイプラインの途中を流れる情報を見たいことがある。そういう時は

```
.... | tee ファイル名 | ....
```

でその場所を流れるデータを指定したファイルに取れる。なぜ tee かって?図 2 を見ればわかる。

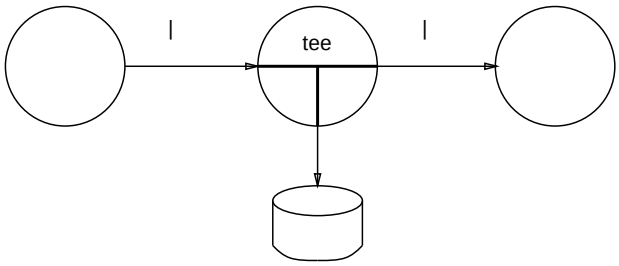


図 2: tee による途中結果の保存

7.5 head と tail

ファイルの頭の方だけ/終りの方だけ見たいことがたまにある。そういう時便利なのが head/tail である:

```
head -行数    -- 頭の n 行のみ取り出す。
tail -行数    -- 末尾の n 行のみ取り出す。
```

いずれも行数を省略すると 10 行を指定したことになる。ところで、ファイルの真ん中辺の、n 行目から m 行目までを取り出したい時は?(わかるかな?)

7.6 パターン探索

ファイルの中から特定のパターンの文字列を含む行を探す、というのは意外に役に立つ道具となるものである。Unix ではそのための指令がなぜか (歴史的事情により)3 つある。

```
grep パターン     -- 一番古くからある指令。
fgrep 文字列      -- 固定文字列のみ探索。
egrep パターン    -- 正則式一般が可能。速い。
```

fgrep 以外の 2 つは「パターン」が書けるが、これはシェルのファイル名マッチングとちょっと似ているがだいぶ違い、次のような記法を持つ。

```
.       -- 任意の 1 文字。シェルでは?だった。
[...]   -- ... の文字のどれか。シェルと同じ。
X*      -- X が 0 個以上。
^       -- 行の頭にマッチする。
$       -- 行の末尾にマッチする。
(grep のみ)
    \(...\)     -- マッチすると同時に内容を覚える。
    \1, \2,... -- 1, 2, ... 番目の覚えた内容と同じもの。
(egrep のみ)
    |, ()       -- または、かっこ。
```

これを使って、英単語が沢山入っているファイル/usr/dict/words からパターンにはまる単語のみを取り出してみる。

```
% egrep 'tion' /usr/dict/words | head -5
ablution        ← tion で終るのはいっぱいあるだろうな
abolition
abovementioned  ← あれ、途中にもある。
absolution
absorption
Broken pipe     ← 5 個あると head が終ってパイプラインが壊れるが無害
% egrep 'tion$' /usr/dict/words | head -5
ablution        ← 今度こそ末尾。
abolition
```

```
absolution
absorption
abstention
% egrep '^ax' /usr/dict/words | head -5
ax              ← 今度は先頭が ax
axe
axial
axiology
axiom
% egrep '^.....$' /usr/dict/words | head -5
Aaron           ← ちょうど 5 文字。
Ababa
aback
abase
abash
Broken pipe
% egrep '^[aeiou][aeiou]*$' /usr/dict/words | head -5
a               ← 母音ばっかりのやつ。
e
i
ii
iii
% egrep '(tion|tive)$' /usr/dict/words | head -5
ablution        ← tion または tive で終る。
abolition
absolution
absorption
absorptive
Broken pipe
% grep '\(..\)\1' /usr/dict/words | head -5
Ababa           ← 2 文字の繰り返し。
adventitious
agglutinin
allele
arginine
% grep '\(.\)\(.\).\2\1' /usr/dict/words | head -5
acetate         ← 5 文字の回文 (?)。
acidic
adenine
Ahmedabad
anastomotic
%
```

7.7 文字置換

tr は次の形式により、文字列置換を実行する。

```
tr [-cds] S1 S2
```

これは何のことかというと、特に変わったオプションを指定しなければ、図 3 にあるように入力ファイルのなかにある文字 X_1 はすべて文字 Y_1 に、文字 X_2 は Y_2 に、というように対応した文字に置換えを行なう、という意味である。なお S2 の方が短い場合は S2 の最後の文字を複数個重複させて S1 と同じ長さにしてから用いる。また、S1、S2 とも「どこから-どこまで」という記法が使える (下の例参照)。ではオプションであるが。

```
-c  -- S1 に入っている文字「以外」を S1 として指定したと見なす。
-d  -- S1 に入っている文字を置き換えるのでなく、消す。
-s  -- 置き換える時、連続した置き換えは 1 個だけ出力 (sqeeze)。
```

では例を見ていただく。

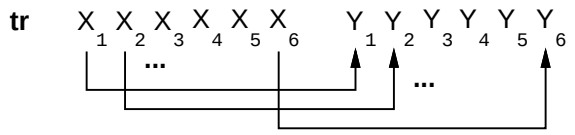


図 3: tr による文字列置換

```
% cat t                ← これがサンプル。
this is a pen.
that isn't a dog.
% tr a-z A-Z <t        ← まずはまっとうなのから。
THIS IS A PEN.           A-Z と書いたら ABCD...XYZ と同じ。
THAT ISN'T A DOG.
% tr aeiou % <t        ← 母音をすべて%に。
th%s %s % p%n.
th%t %sn't % d%g.
% tr -c a-z # <t       ← 英字以外を#に。[ret] も#になる!!
this#is#a#pen##that#isn#t#a#dog##% tr -cd a-z <t  ←英字以外消去
thisisapenthatisntadog% tr -cs a-z '\012' <t  ←英字以外 [ret] に
this                                             ただし squeeze.
is
a
pen
that
isn
t
a
dog
%
```

7.8 ストリームエディタ

sed は nemacs などと同様文書を編集する機能を提供するが、ただしそれは対話的に行なうのではなく、予め「編集指令」を与えておいてそれに応じた編集を一気に行なう形を取る。(Q. そういうエディタは、nemacs のような**対話型**エディタと比べてどういう利点を持つか?)

sed では一つまたは複数の編集指令を与えて次のような形で呼び出す。

```
sed [-n] '指令'-- 指令が 1 個の場合
sed [-n] ( -e '指令' )...   -- 指令が複数の場合
sed [-n] -f 指令ファイル     -- 指令を別のファイルに 1 行 1 個ずつ用意
```

そして、図 4 にあるように、入力から行が 1 つ読まれては、それが各指令によって順に加工され、最後に出力に書かれる (ただし-n オプションが指定された場合は書かれない)、という動作が入力が尽きるまで実行される。

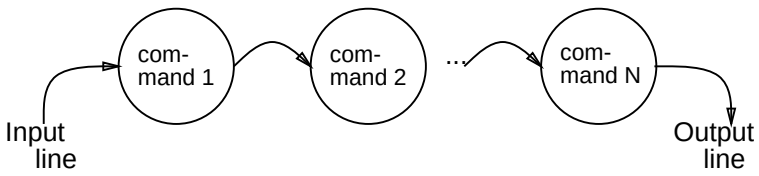


図 4: sed による編集の概念

指令の形式は

指令=[アドレス [, アドレス]] 指令名 [引数]

のようなものである。アドレスというのは「どの行」を意味し、各指令は「この行をどうこうせよ」または「この行からこの行までをどうこうせよ」という処理を行なう。アドレスとしては次の 3 種類がある。

```
数           → n 行目、をあらわす。
$           → 最後の行、をあらわす。
/パターン/  → そのパターンが見つかった行、をあらわす。
```

指令の代表的なものとしては次のものがある。

```
s/パターン/文字列/[g]    -- 行内にパターンが見つかればそれを文字列に置き換え
                            (ただし g が指定してなければ各行最初の 1 個のみ)
d                        -- その行を消す
p                        -- その行を打ち出す (-n 指定の時有用)
```

さて、では実例であるが。

```
% ls -l                      ← もとの ls -l だと。
total 109
-rw-r--r--  1 kuno        27649 Oct 29 22:17 r1.tex
-rw-r--r--  1 kuno        21969 Oct 29 22:17 r2.tex
-rw-r--r--  1 kuno        18178 Oct 29 22:17 r3.tex
% ls -l | sed '1d'          ← 1 行目を消す。
-rw-r--r--  1 kuno        27649 Oct 29 22:17 r1.tex
-rw-r--r--  1 kuno        21969 Oct 29 22:17 r2.tex
-rw-r--r--  1 kuno        18178 Oct 29 22:17 r3.tex
% ls -l | sed -e '1d' -e 's/^.*    //'
27649 Oct 29 22:17 r1.tex  ← 頭の方を消して、
21969 Oct 29 22:17 r2.tex
18178 Oct 29 22:17 r3.tex
% ls -l | sed -e '1d' -e 's/^.*    //' -e 's/ .* / /'
27649 r1.tex                ← 真ん中の日付も消して、
21969 r2.tex
18178 r3.tex
% ls -l | sed -e '1d' -e 's/^.*    //' -e 's/ .* / /' -e 's/\(.*\) \(.*\)/\2 \1/'
r1.tex 27649               ← ひっくり返した!
r2.tex 21969
r3.tex 18178
%
```

7.9 整列

行を何らかの順 (例えばどの部分かのアルファベット順とか数字の順) に並べ替えるというのもよく使われる操作である。Unix ではこれは sort 指令によって行なうが、「どの順で」を指定するので結構むずかしい (何も指定しなければ行全体のアルファベット順だが、これでも結構役に立つ)。

```
sort [f][n][r]
```

ただし各オプションの意味は次の通り。

```
f  -- 大文字小文字の区別をしない (普通は区別する)
n  -- 数値比較する (普通は文字列比較する)
r  -- 降順に並べる (普通は昇順)
```

これらの必要性は次を見れば分かる。

```
% cat t
These
are
dogs
```

```
% sort t      ← そのまま整列すると...
These            同じ!大文字のコードは小文字より若い!
are
dogs
% sort -f t  ← 大文字小文字を区別しなければ辞書順になる。
are
dogs
These
% cat t1      ← 数値の並び。
1
10
4
% sort t1     ← 整列すると...
1                変わらない!辞書順ではこうなる。
10
4
% sort -n t1  ← 数値順を指定すれば OK
1
4
10
%
```

ところで、行全体をまとめて整列するのでなく、空白で区切られた欄を見てどの欄を使って整列、という使い方もできる。その場合は

```
sort (+n[f][n][r]) ...
```

という指定になる。ちなみに n が「何番目の欄」(0 から数える) 意味する。複数指定すると最初のものが第 1 のキー、次のものが第 2 のキー、... となる。例えば。

```
% cat t2          ← 単語と数字のペア。
These 1
are 10
dogs 4
% sort +1nr t2   ← 数字部分の、大きい順に並べる。
are 10
dogs 4
These 1
%
```

7.10 重複の除去

隣接する行が同じものだったらその重複を除いて 1 つにする、というフィルターがある:

```
uniq [-c]
```

ちなみに-c を指定すると重複の個数を数えて一緒に出力してくれる。これを利用したとても有名な例を挙げておく (何をやると思うか?)

```
tr -cs A-Za-z '\012' | sort | uniq -c
```

まあ答えは簡単だから。

```
% cat t                          ← もとファイル
this is a pen.
that is a dog.
% tr -cs A-Za-z '\012' <t   ← さっきやった。
this
is
a
pen
that
is
a
dog
% tr -cs A-Za-z '\012' <t | sort  ← さらに並べて。
a
a
dog
is
is
pen
that
this
% tr -cs A-Za-z '\012' <t | sort | uniq -c
   2 a                           ← 数えると、単語頻度表になるのだ。
   1 dog
   2 is
   1 pen
   1 that
   1 this
%
```

7.11 フィルタのまとめ

Unix のフィルタは一つ一つは割合単純な機能のものばかりだが、それを組み合わせると色々面白いことができる、ということになっている。その面白さを味わっていただければ幸いである。練習問題もそういう意味ではパズルっぽくできているので、頭をひねってみて欲しい。あと、面白いだけでなく、「複雑な小数のツール」と「単純な多数のツール」という思想の比較もしてみて欲しい。

A　演習および課題

A.1　7 節の演習

7-1. n 個のプロセスをパイプラインに並べて実行している時、プロセスは何個できていると思うか。またそれらの親個関係はどうか。ともに予想し実地に調べてみよ。

7-2. ファイル/usr/dict/words には主要な英単語が並んでいる (見てみよ)。この中から、母音が一番多く含まれる語、一番長く連続して現れる語のベスト 5 を探してみよ。

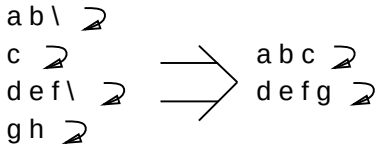
7-3. あやふやにしか覚えていない単語の正しいスペルをみつける方法を考案し、実際に試してみよ。[1]

7-4. 「ls -l」ではファイルのバイト数が分かるが、その他の情報もついているので見にくい。これをファイル名とバイト数のみから成るファイルに変換し (これは例題にある)、さらにぎざぎざがなく、ファイル名の左端、およびバイト数の右端が縦にきれいに揃っているようにしてみよ。(例題では数字の桁数がたまたま揃っていたのだった)

7-5. 数字のみの行がずらっとならんだファイルについて、その数の合計をとる方法を考案せよ。これをもとに、ディレクトリにあるファイルの容量合計を計算してみよ。[2]

7-6. Script で作ったファイルというのは [DEL] キーを押した所が「なんとか-BS-SP-BS」という形になっていて打ち出すと美しくない。あと余分な [CR] も入っている。これらを自動的に取り除いてきれいにする方法を探求せよ。(ヒント:制御コードは tr を使えば「見える字」に直せる。その後で sed を使う。なんなら sed のループも使うと良い。)

7-7. シェルの継続行のようなものをくっつける処理をしてみよ。すなわち、次の図のような変換をしてみよ。[3]



A.2　5 回目の課題

本日の課題から報告を提出する場合は、上記 7-1〜7-7 のうちから最低 2 つ以上選択して下さい。

[1] ヒント:何かうまい「正規形」を考え、それに変換して同じ形になるものを探すというのはどうか。例えば「母音は全部消去し、L と R はすべて R に直す」という正規形を使うとする。「アルバイトは arbaite かな?」と思ってその正規形を作ると「RBT」。一方正しいつづりも「albeit」だから「RBT」になるので、うまくやればちゃんと探せる。

[2] ヒント:expr という指令を使うと計算ができる。例えば「expr 1 + 2」は 3 を表示する。だから、ファイルを「... + ... + ...」という形に変形してから `expr ‘cat そのファイル‘` とすればいい。

[3] ヒント:sed のマニュアルをよーく読む!あるいは、tr で行の切れ目を別の字にかえてくっつけてから処理し、最後にまた tr で戻す。sed は改行が 1 個はないといけないので、「`echo ‘改行をなくしたファイル‘ | sed ...`」がいいかも知れない。