情報処理 II 第 7 回

情報の電子化 (2) テキスト・ファイルの処理

桂田 祐史 (かつらだ まさし)

2005 年 6 月 9 日

ホームページは http://www.math.meiji.ac.jp/~mk/syori2-2005/

1 シェルを変えてみませんか

(強制ではないです。今日 tcsh や bash を使ってみてから決めても構いません。)

Linux 環境では KDE や GNOME などのいわゆるデスクトップ環境が整備されつつあるが、やはり基本はキーボードからの文字入力が主体の CUI (character user interface) である。キーボードからのコマンド入力の効率には気を配っておく必要がある。実は Solaris で標準的なシェルに採用されていた csh (C shell) は、キーボード入力に関してはあまり工夫がされていない。Linux では標準で bash (bourne again shell[1]) というシェルが利用されているが、色々と便利な機能を持っている。また csh の機能拡張版といえる tcsh もある。tcsh あるいは bash を使うように設定しておくことを勧める。シェルを変えるためには samba00 などの Solaris 環境にログインして、以下のようなやり取りをする。

```
a308-06% telnet samba00
  (以下ユーザー名とパスワードの入力)
samba00% /usr/bin/passwd -r ldap
Enter existing login password: (パスワードの入力)
Old shell: /bin/csh
New shell: /bin/tcsh
passwd: password information changed for ee48099
samba00% logout
a308-06%
```

http://www.math.meiji.ac.jp/~mk/syori2-2005/How-to-tcsh-on-Linux/
『tcsh への誘い』

[1] もともと “Bourne shell” というものがあり、それに対して上位互換であるシェルを作り直したので、“born again” とかけて “bourne again shell” と命名された。

2 UNIX のテキスト・ファイル処理

文書は電子化されてしまえば、プログラムで様々な処理をすることが可能になる。UNIX にはテキスト・ファイルの処理のために便利な仕掛け (入出力のリダイレクション、パイプ) が用意されていて、すぐれたテキスト処理コマンド (ただし基本的に英語向け) が用意されている。その一端を体験して「どういうことが出来るのか」を知ろう。— コマンドの使い方を覚えて欲しいわけではない。

2.1 標準入出力のリダイレクション、パイプ

UNIX において標準入出力のリダイレクション、パイプは重要な機能である。

- UNIX システムでは、各プロセス[2]は、**標準入力** (standard input)、**標準出力** (standard output) と呼ばれるデータの出入口を持つ。
- 例えば C 言語の `scanf()`, `getchar()` 等は標準入力 (C では **stdin** と呼ぶ) から入力し、`printf()`, `putchar()` は標準出力 (C では **stdout** とよぶ) に出力する[3]。
- 通常は次のように結びつけられている:
  - 標準入力はキーボードからの入力に
  - 標準出力は画面への出力に

  この結合は簡単に変更できる (標準入出力の**リダイレクション** と呼ぶ)。

  標準出力をリダイレクト コマンド > ファイル名 とすると、コマンドの標準出力を (画面の代わりに) 指定したファイルに書き出す。

[2]動いているプログラムのことをプロセスと呼ぶ。

[3]C 言語のプログラムに良く出てくる `#include <stdio.h>` の “stdio” は “standard input and output” の略。

試してみよう: cal コマンドの出力を cal.out にリダイレクト

```
a308-06% cal
     6月 2005
日 月 火 水 木 金 土
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

a308-06% cal > cal.out
a308-06% cat cal.out
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

a308-06% cal 9 1752        (ちょっと不思議な結果)
```

標準入力をリダイレクト コマンド < ファイル名 とすると、(キーボードの代わりに) 指定したファイルからコマンドの標準入力に読み込む[4]。

- コマンド 1 | コマンド 2 とすると、コマンド 1 の標準出力をコマンド 2 の標準入力に渡すことが出来る。つまり

(**注意**: prog1, prog2 を適当に選ばないと試せない)

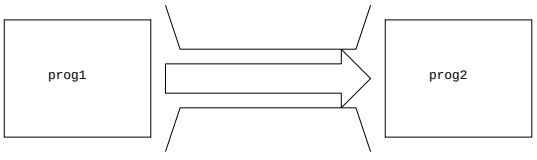
```
prog1 > datafile
prog2 < datafile
```

のようにファイルを介して prog1 の出力を prog2 に渡す代わりに

(**注意**: prog1, prog2 を適当に選ばないと試せない)

```
prog1 | prog2
```

で済ますことが出来る。この機能を**パイプ**と呼ぶ。実例は以下で (山のように) たくさん見せる。



[4]最近では cat ファイル名 | コマンド とパイプを使ってすますことが多くなって、入力のリダイレクトはあまり使われなくなったような気がする。

2.2 フィルター

まず以下の準備をしてから、その後の操作をしよう。(tcsh の設定がまだの人は、tcsh とタイプして、tcsh を起動してから実習を始めて下さい。)

実験用ファイルの準備

```
a308-06% cd                              ← ホームディレクトリィに移動する。
a308-06% tcsh                            ← tcsh を実行 (設定済みの人は不要)。
a308-06% mkdir filter                    ← filter というディレクトリィを作る。
a308-06% cd filter                       ← filter に移動する。
a308-06% ln -s ~re00018/gutenberg/* .    ← ~re00018/gutenberg にあるファイル
                                           にシンボリック・リンクを張る。
a308-06% ls                              ← どういうファイルがあるか、確認する。
                                         → alice29.txt などが見えるはず。
a308-06% cat LIST                        ← LIST の中身を見る。
```

今回配布したファイル (以下では Gutenberg テキストと呼ぶことにする) の出所については、付録 A 「古典 (文書) の電子化」を参照せよ。

UNIX システムでは、標準入出力のリダイレクション、パイプ機能を活用するため、

標準入力から入力し、標準出力に出力し、他の入出力は使わないで済む

というコマンドが多い。このようなプログラムを**フィルター**と呼ぶ。フィルターは簡単な機能しか持たないものが多いが

複数のフィルターをパイプで接続して複雑な仕事が出来る

ようになっている[5]。

以下、UNIX の代表的なフィルターをいくつか紹介する。

grep テキスト・ファイルから指定した文字列を検索して、それを含む行を表示する。

検索に用いるパターンの指定には、**正規表現** (**regular expression**[6]) と呼ばれる形式が利用できる (grep の名前の由来は global/regular-expression/print)。

- grep　パターン　検索したいファイル名 が基本的な使い方。

grep で指定したパターンを含む行の表示

```
a308-06% grep Alice alice29.txt          alice29.txt から Alice という語を含む行を探す。
a308-06% cat alice29.txt | grep Alice    こうやっても OK.
a308-06% grep 'ee[34]80' ee-list         数学科学生のパスワード・マップ ee-list から ee380, ee480 を含む行を探す。
```

[5]それから、これは機能というわけではないが、フィルターは高速に実行できるよう、相当に高度なチューニングがされているのが普通である。

[6]正規表現で記述できる言語を処理するアルゴリズムは簡単で、プログラムも自動的に作成可能である。UNIX では、文字列パターンを表すのに、正規表現が広く使えるようになっている。

- -v とすると、パターンを含まない行を表示する。

grep -v で指定したパターンを含まない行を表示

```
a308-06% grep -v tcsh ee-list   ee-list から tcsh を含まない行を探す。
```

- -i とすると、大文字・小文字を区別しないで検索する。

grep -i で大文字小文字を区別しない検索

```
a308-06% grep -i 'mark twain' *   カレント・ディレクトリィから
                                  "mark twain" を含むファイル
                                  を探す (大・小文字の違いは無視)。
```

sort テキストファイルを行単位で、順序付けて (通常はアルファベット順に) 並べ変える。

- -r　逆順に並べる。
- -n　数値としての大小で並べ変える。
- +x (x は整数)　第 x フィールドの要素の大小で並べる。

例えば、

ホームディレクトリィにあるファイルを、サイズが大きい順にリストする

```
a308-06% ls -l ~ | sort +4 -n -r
```

(ここでは ls -l ~ の出力の第5フィールドはファイルのサイズである、と仮定している。)

wc テキスト・ファイル中の行数、単語数、文字数を数える。

wc で行数、単語数、文字数を数える

```
a308-06% wc alice29.txt
```

uniq 連続する同じ内容の行を1つにまとめる。
-c とすると、同じ行が何行続いたか表示する。次の例は、今回のハイライトとも言える凝った例である。

呪文のように見えるかもしれませんが…

```
a308-06% cat alice-words | sort | uniq -c | sort -n
```

一体何をやっているのか? 一段一段確認しながらチェックしよう。
長いコマンドはシェル・スクリプト[7] にすると便利である。

[7]コマンドをシェルの文法に則って並べて作ったテキスト・ファイルを書き、chmod +x ファイル名 として作成できる実行可能なコマンドのことをシェル・スクリプト (shell script) と呼ぶ。script には台本という意味がある。例に上げたシェル・スクリプト top20 は 2 行 (実質 1 行) のテキスト・ファイルである。

シェルスクリプトの利用例

```
a308-06% cat top20
#!/bin/sh
cat "$@" | sort | uniq -c | sort -r -n | head -20
a308-06% ./top20 alice-words
```

tr 文字単位の置換、削除を行う。

tr [-cds] [文字列 1 [文字列 2]]

- -d は削除することを表す。
- -s は同じ字が連続した場合、一字に置換することを表す。
- -c はパターンに含まれない文字を対象にする。

パターンには文字コードを 8 進数で指定できる (以下の例を参照) [8] 。また文字の範囲を - で指定できる。A-Z で 'A' から 'Z' までの文字 (つまり英大文字) を表す。

BSD UNIX の tr (tr)

```
a308-06% cat alice29.txt | tr a-z A-Z
a308-06% cat alice29.txt | tr A-Z a-z
a308-06% cat DOS-text.txt | tr -d "\015\032"
a308-06% cat alice29.txt | tr -cs A-Za-z '\012'
```

小文字を大文字にする。
大文字を小文字にする。
コードが 015, 032 の文字を削除する。
アルファベット以外の文字を改行に変換。
→ 一行一単語に分解する。

上記の alice-words というファイルは次のようにして作った (つまりアルファベット以外の文字の連なりを改行に変換した)。

alice-words はこうして作った

```
cat alice29.txt | tr -cs A-Za-z '\012' > alice-words
```

だから、単語の出現頻度表を得るために、次のようにすることもできる。

直接 alice29.txt から単語の出現頻度表を作る。

```
cat alice29.txt | tr -cs A-Za-z '\012' | sort | uniq -c | sort -n
```

(こんなに長いコマンドは打つ気になれない？いざとなったらシェル・スクリプト！top20 を参考にして作ってみるとよい (授業で一度だけやってみせる)。)

head 最初の数行のみ表示するコマンド。

オプションを指定しないと最初の 10 行を表示する。-x とすると最初の x 行を表示する。

[8] 015 (8 進) $= 1 \times 8 + 5 = 13$ は C-m (復帰, carriage return) の文字コード、032 (8 進) $= 3 \times 8 + 2 = 26$ は C-z (古い MS-DOS テキスト・ファイルの終り) の文字コード、012 (8 進) $= 1 \times 8 + 2 = 10$ は C-j (改行, newline (NL), linefeed (LF) とも) の文字コード。

```
a308-06% head -30 /usr/share/dict/words   ← /usr/share/dict/word の
                                            先頭から 30 行を表示する。
```

(/usr/share/dict/words は英単語を納めたファイルである。Solaris では /usr/dict/words というパス名になっている。)

tail テキストの最後の方を表示するコマンド。
オプションを何も指定しないと 10 行のみ表示する。 $-x$ とすると最後の x 行を表示する。$+x$ とすると、最初の x 行を除いた残りを表示する。

```
a308-06% cat sawyr10.txt | tail -30   ← sawyr10.txt の最後の 30 行を
                                        表示する。
```

nkf 日本語の文字コード (漢字コード) の変換をする。
(nkf については、前回のプリントに最低限のことは説明したので、ここでは略する。)

2.3 awk

awk(オーク) は強力な文字列処理機能と、C 言語に似た文法を併せ持つ言語処理系であるが、フィルターとしての性格を持ったコマンドなので[9]、ここで紹介しておく。

多分 C 言語を知っている人には、次のプログラムを理解するのは難しくないであろう (別に今理解する必要はない)。先頭に ee380 というパターンを持つ行の第 1, 5, 7 フィールドを表示し、第 7 フィールドが /bin/csh である数を数え、その割合を求めている。

awk スクリプトの例 — test.awk

```
#!/bin/gawk -f
BEGIN {
        FS = ":";
        numofstudent = 0;
        numofcsh = 0;
}

/^ee380/ {
        printf("%s %20s, %s\n", $1, $5, $7);
        numofstudent++;
        if ($7 == "/bin/csh") numofcsh++;
}

END {
        printf("%d 人中 csh を使っているのは %d 人 (%4.1f\%) です。\n",
               numofstudent, numofcsh, 100.0 * numofcsh / numofstudent);
}
```

[9]似たような性格を持っているコマンドに **Perl**(パール) がある。Perl は WWW の CGI のプログラムを記述する言語として普及している。

という awk のプログラム[10] test.awk を使って、

```
a308-06% ./test.awk ee-list
```

とする。
(ちなみに、私はこの gawk を用いて、レポートや試験の採点処理をしている。)

3 レポート課題 4

〆切は 6 月 29 日 (水曜)。Subject は「情報処理 II 課題 4」とすること。

課題 4 英文中のアルファベットの出現頻度は 'e' が一番高く、その次は… などと言われ、古典的な推理小説[11]の暗号[13]の話の種になったりしている[14]。Gutenberg Project の中のテキストで、そのことを確かめて見よ (各テキスト毎に使用頻度の上位がどうなっているか記録をとって、それを自分のことばでまとめなさい、ということ)。手作業ではなく、なるべくコンピューターにやらせること。テキストごとに大きな違いがあるか？文字が別の記号に置き換えられた場合、出現頻度情報から解読することの可能性について論ぜよ (要するに他の文字の出現頻度はどの程度まで一定しているのか調べる — 実際に試してみると良いのだけど)。なお、文字の頻度を調べる hindo.c というプログラムを用意した[15]。(このプログラムは文字の出現頻度順には表示しないが、sort を使えば簡単に頻度順に並べられる。どうすればいいか？今回説明した話の簡単な応用である。)

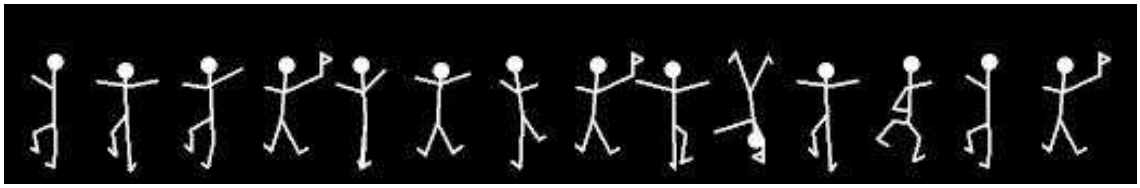
hindo.c のコンパイルと使用例

```
a308-06% gcc -o hindo hindo.c
a308-06% cat hindo.c | ./hindo
```

テキスト、あるいは作家ごとに単語の使用頻度の癖のようなものがあると思われるが、そのことを Gutenberg テキストで実際に調べてみよ。例えばルイス・キャロルとマークトウェインの書いたものにどの程度の差があるか？

また、できれば ~re00018/gutenberg/ にあるテキスト・ファイル以外のテキストを探

[10] これも (シェル・スクリプトと同様のニュアンスで) awk スクリプトと呼ばれることが多い。
[11] 例えば、エドガー・アラン・ポー「黄金虫」[12] やアーサー・コナン・ドイル「踊る人形」 (いわゆるシャーロック・ホームズもの)。



これは http://hp.vector.co.jp/authors/VA023018/danceed.htm にあったサンプルを拝借。それにしても、TEX 用の METAFONT で踊る人形のフォントを作った人 (Alan M. Stanier) がいるなんて…
[13] 比較的最近話題になった楽しい読み物として、サイモン・シン「暗号解読 — ロゼッタストーンから量子暗号まで」新潮社 (2001) をあげておく。
[14] もっとも、世の中には奇特な人もいて、'e' を一度も使わずに書いた小説というものもあるそうだ。
[15] hindo.c は filter ディレクトリィに入っている。

して入手し (その方法も説明せよ)、同じような解析を行なえ。

4 研究課題 (換字暗号解読)

「可能性について考えよ」だけではつまらないので、余裕と興味のある人は試しに解読に挑戦して下さい。filter ディレクトリィにある "angou.txt" は[16]は

こうやって作った

```
cat もとのテキスト | tr A-Z XXXXXXXXXXXXXXXXXXXXXXXXXX | tr a-z
xxxxxxxxxxxxxxxxxxxxxxxxxx > angou.txt
```

のようにして作った ("$XXX\cdots XXX$", "$xxx\cdots xxx$" はそれぞれ、"ABCDEFGHIJKLMNOPQRSTUVWXYZ", "abcdefghijklmnopqrstuvwxyz" を並べ替えて作った文字列である)。元に戻すには、

こうやれば解読できる

```
cat angou.txt | tr XXXXXXXXXXXXXXXXXXXXXXXXXX A-Z | tr
xxxxxxxxxxxxxxxxxxxxxxxxxx a-z
```

試してみたいのならば

「不思議の国のアリス」で試す

```
a308-05% cp -pr ~re00018/sample-angou .
a308-05% cd sample-angou
a308-05% head -30 sample-angou.txt
a308-05% head -30 sample-angou.txt | ./decode1.sh
a308-05% cat decode1.sh
a308-05% cat encode1.sh
```

解読に成功した人は、(1) 文章 (かなり有名なお話です) のタイトル、(2) 復号化するためのコマンドを報告してください。解読したという報告は mk@math.meiji.ac.jp の方に連絡して下さい。

5 その他

5.1 Perl のすすめ

単機能のコマンドを組み合わせて使うのが「UNIX らしい」のですが、便利さで選べば Perl 一つに詳しくなる方が良いかもしれません。Perl でやるとどうなるか、誰かレポートしてみませんか？

5.2 tcsh のすすめ

http://www.math.meiji.ac.jp/~mk/syori2-2005/How-to-tcsh/ を見て下さい。

[16] http://www.math.meiji.ac.jp/~mk/syori2-2005/angou.txt にも置いてある。

A　古典 (文書) の電子化

知的財産権についての議論が盛んである。マルチメディアやインターネットなど、コンピューターに絡んだ問題が多い。

- 著作権にも有効期限があり、それが切れた著作物は自由に利用できる。
  有効期限は、日本の場合は作家の死後 50 年。世界的にどんどん長くする傾向にある (例: イギリス 50 年 → 70 年、アメリカ 95 年. そうするのが本当に良いことなのだろうか?)。
- 既に活字になっている文書を電子化するには **OCR** (optical character recognition, 光学的文字認識) などが使え、作業は半自動的に行える。
- ひとたび電子化してしまえば、完全なコピーが簡単に出来る。
- 電子化された文書は、さまざまな利用が出来る。
  再び整形して本にする、読み上げソフトで音声にする、機械翻訳する、定量分析する、etc.

諸君に考える材料を提示する意味で、Gutenberg プロジェクトを紹介しよう。

A.1　Gutenberg プロジェクト

Gutenberg プロジェクトは、1971 年にイリノイ大学の Michael S. Hart の思いつきからスタートしたプロジェクトで、その行動計画は次のようなものである[17]。

> もっとも利用頻度の高い 1 万タイトルの書物を、順次、電子テキスト化していって、二十一世紀のはじめの年、すなわち 2001 年の終りまでに「グーテンベルク電子公共図書館」を作り上げること

このプロジェクトの成果は、CD-ROM や WWW ページ `http://www.promo.net/pg/` などで公開されている。

今では、大きな機関 (アメリカ議会図書館、日本の国会図書館などの図書館) が、古典の電子化に取り組んでいるので、ボランティア・ベースの Gutenberg プロジェクトに、実際的な意味はもしかするとあまりないのかもしれないが、最初に一石を投じた営みは記憶しておくべきであろう。

A.2　日本語テキストの電子化

日本でも Gutenberg プロジェクトのような試みがある。詳しくは次回説明するが、例えば、

『青空文庫』 `http://www.aozora.gr.jp/`
特に『青空文庫早わかり』`http://www.aozora.gr.jp/guide/nyuumon.html`
(ここは古典だけでなく、著者が著作権を放棄した作品なども置いてある。)

[17]参考文献: 津野海太郎、本はどのように消えてゆくのか、晶文社 (1996) から引用した。