UNIX ワークステーションによる
データ解析入門
第 6.0 版

阿蘇司 [a]、 **石塚正則** [a]、**佐々木節** [b]、**白井みゆき** [a]、 **高清水直美** [a]

新潟大学高エネルギー研究室 [a]
高エネルギー物理学研究所 データ処理センター [b]

1995 年 7 月 13 日

まえがき

このテキストの目的は、高エネルギー実験のデータ解析でよく使われる HBOOK を用いて、ヒストグラムを作るための FORTRAN プログラムを書き、実行して、 PAW でディスプレイすることである。対象は、これまで UNIX を使った経験がない人を想定しており、ログインの仕方から簡単な UNIX のコマンドの使い方、エディタの使い方、デバッグの仕方や LaTeX[1]の使い方などを説明する。

このテキストで標準としている環境は、 X ウィンドウ・システムと (t)csh である。このマニュアルは、もとは SUN ワークステーション（SUN OS 4.1.x）を対象として書かれた。しかし、SUN 独自の機能についてはほとんど触れていない。マシンに依存しない環境を覚えた方が多くのところで役に立つ思われるので、多少不便でも標準的な環境にのみ触れる。

このテキストは、もともとは、新潟大学高エネルギー物理学研究室で、学部４年生を対象に UNIX ワークステーションの簡単な使い方と管理および簡単なプログラミングを教えるセミナーのテキストとして作成された。最近の急激な UNIX の普及とともに、新潟大学以外からも配布希望が多かったので、新潟大学に依存した部分をできるだけ取り除き、希望者に配布することになった。さらに、高エネルギー物理学研究所内で使用できるよう、一部内容に変更を加えた。

UNIX で FORTRAN プログラミングをする人口が少ないため、 FORTRAN プログラマ向けのよいテキストがほとんどない。また、 UNIX のオンライン・マニュアルは分かりやすくはなく、UNIX に関しては初心者向けの多数の本が出版されているにもかかわらず、買うに値するものは少ない。書籍やマニュアルを読むための手掛かりとして、より簡単でまとまりのよいテキストを目指して書かれた。参考になる本をできるだけ紹介し、 UNIX の理解の助けになればと希望する。

本書のうち Emacs のリファレンス・マニュアルは、 Free Software Fundation の著作物である。 また、本書の一部または全体を無断で流用することは禁ずる。このテキストに、誤りを発見したり、コメントがある場合には、ぜひ著者へ知らせてほしい。

最新版は、

```
bsun01.kek.jp:/pub/doc/unix_doc.ps.Z
```

にある。 anonymous ftp で入手することができる。

```
% uncompress unix_doc.ps
% lpr -s -P<printer_name> unix_doc.ps
または、
% lp -d<printer_name> unix_doc.ps
```

とすれば、ポストスクリプト・プリンタに出力される。 anonymous ftp の方法を以下に示す。

[1] このテキストは、 NTT 版日本語 LaTeX を使ってフォーマットされた。この素晴らしいシステムを作成し、無料で配布している人々に感謝する。

```
% ftp bsun01.kek.jp
Connected to bsun01.
220 bsun01 FTP server (Version wu-2.4(1) Sun Apr 17 02:18:07 JST 1994)
ready.
Name (bsun01:sasaki): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:    （ここに自分の E-mail アドレスを入力する。）
230-Welcome to the BELLE collaboration archive server. All your activity
230-is logged. If you do not prefer it, please logout now.
230-
230-Any comments and questions are wolcome. E-mail:sasaki@bsun01.kek.jp
230-
230 Guest login ok, access restrictions apply.
ftp> cd pub/doc
250 CWD command successful.
ftp> bin
ftp> get unix_doc.ps.Z
ftp> quit
```

また、同じディレクトリにある example.tar.gz は、このテキストで紹介した
例題を集めたものである。gunzip コマンドで解凍し、tar コマンドで
ファイルを取り出すことができる。

著者の電子メールアドレス：
unix4hep@ngthep.hep.sc.niigata-u.ac.jp

住所：
〒 950-21 新潟市五十嵐２の町
新潟大学理学部物理学教室
高エネルギー研究室

目次

第 1 章

とにかくワークステーションを使ってみよう！

とにかく何かやってみないと気がすまない人たちのために、簡単なワークステーションの使い方を紹介する。

1.1 アカウントをもらおう！

UNIX ワークステーションを使うには、まずそのワークステーションが使えるようにアカウントをもらわなければならない。システムの管理者に連絡し、ユーザ名とパスワードを入手する。UNIX はマッキントッシュ等のパーソナルコンピュータと違い、マルチユーザ・オペレーティング・システム（OS）なので、このような手続きが必要である。ユーザ名で、計算機上の資源の割り当てや、セキュリティの管理が行われる。また、他の人と資源を共有するので、もし間違ったことや自分勝手なことをすると、他の人の迷惑となることもあると覚えておこう。

1.2 UNIX ワークステーションの使い方

ここでは csh と UNIX のコマンドの初歩的な使い方を述べる。第 2 章と重複する内容も多いので、そちらも参照のこと。

1.2.1 ログインの方法

マルチユーザ OS である UNIX ワークステーションを使うには、まずログインという作業をしなてくてはならない。これは、システムがユーザの認証を行い、ユーザまたはシステムの指定した様々な仕事に必要な設定をし、システムを使用可能な状態にすることをいう。従って、使用を許可されていないワークステーションを用いることはできない。システム管理者から許可を得て、使用できるように設定をしてもらう必要がある。管理者は、ユーザ名と仮のパスワードを教えてくれるはずである。手続きが済んだら、さっそくログインしてみよう。

ワークステーション本体や X 端末[1]に、図 1.1 のようなウィンドウが表示さていたら、ワークステーションのリストから、ログインしたいマシーンをマウスで、ダブルクリックして選ぶ。 すると、図 1.2 のようなウィンドウが表示されるはずである。最初からこの状態になっている場合もある。そうしたら、

[1] X ウィンドウ用の端末

login:
の後に、ユーザ名を入れてリターン・キーを押し、
passwd:
の後に、パスワードを入力する。機種や設定によっては、派手に趣向をこらしたような、異なる画面が表示されていることもある。しかし、基本的に行うことは同じである。

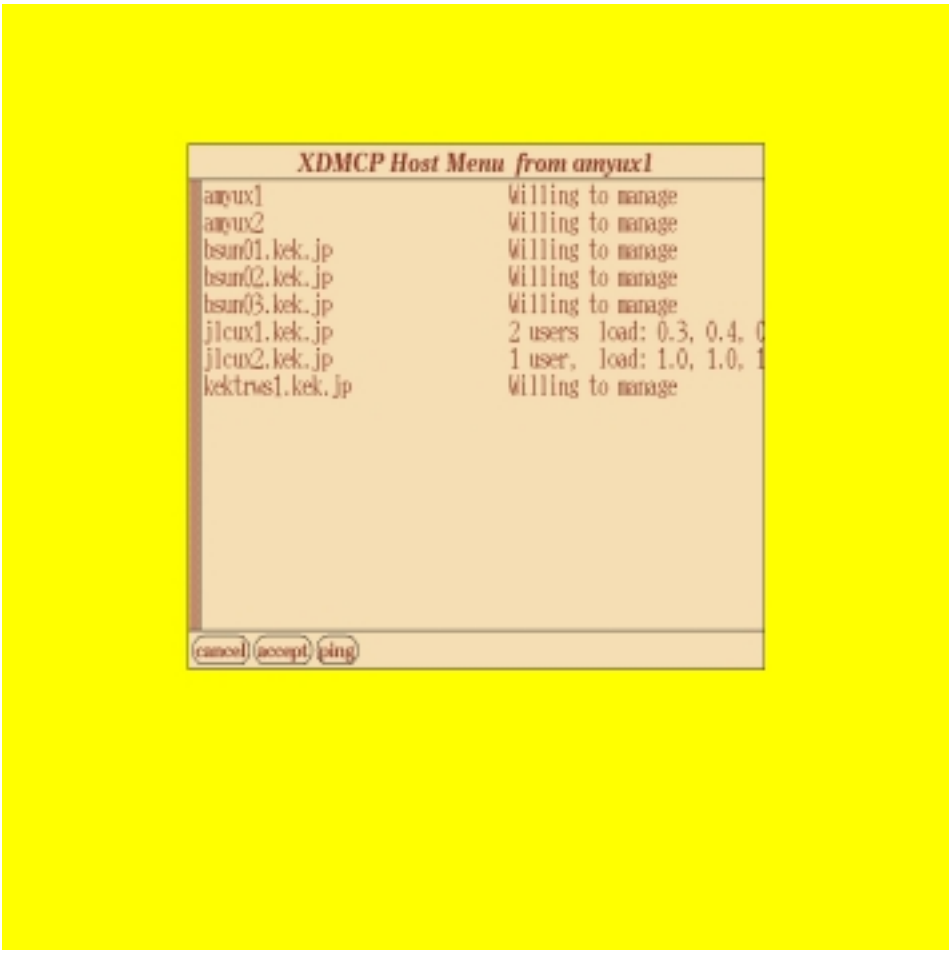


図 1.1: xdm chooser スクリーン

ログイン・ウィンドウの出ていない X 端末の場合には、 X 端末自身の機能を使わなければならず、一般的な方法はない。機種ごとに違うので、管理者と相談すること。多くの場合、 X 端末の操作メニューから、 telnet のウィンドウを開き、そこから telnet コマンドでログインする。

X 端末から、ログインに成功すると、そのシステム標準の画面が表れるはずである。少なくとも一つの端末エミュレータのウィンドウがあること多い。 % や $ 等の文字を含むプロンプト以外何も表示されていないシンプルな画面があれば、それがそうである。そのプロンプトの後にコマンドを書いて、リターン・キーを押すとコマンドが実行される。もし端末エミュレータの画面がなければ、マウスのボタンを背景の上で、右から順番に押してみよう。それらしいメニューがあるかもしれない。画面の一番下や上にそのためのボタンがあり、それをクリックすると表示され

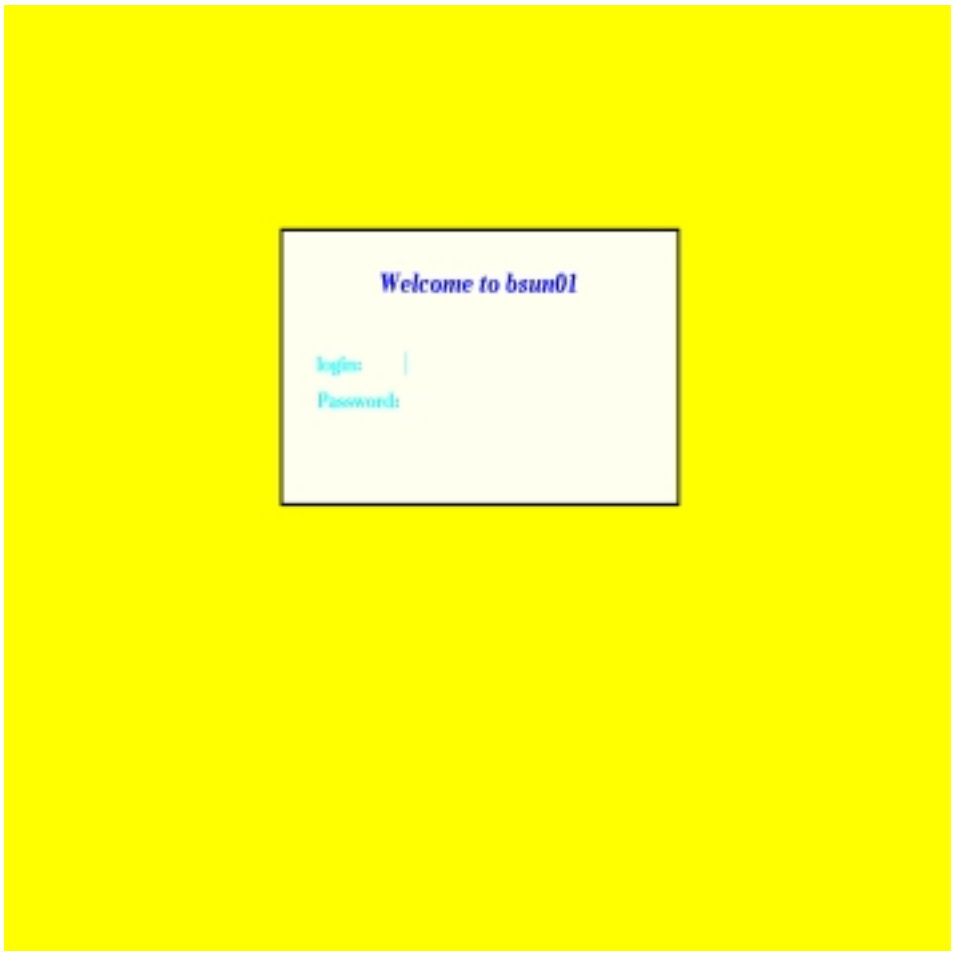


図 1.2: xdm によるログイン・ウィンドウ

るような機種もある。

次に、ログイン・ウィンドウの表示されていないコンソール (ワークステーション本体についているキーボードとディスプレイのこと) あるいは、すでに UNIX システムと接続されているターミナルからログインする方法を述べる。 UNIX システムと接続されているターミナルやコンソールには、

```
SunOS UNIX (bsun01)
login:
```

と表示されているはずである。 login:
のプロンプトのあとに、ユーザ名を入れてリターン・キーを押し、
passwd:
パスワードを要求されたらそれを入力する。すると、

```
Last login: Sat May 22 08:08:58 from bsun01
SunOS Release 4.1.3 (BFACTORY) #1: Fri Apr 30 20:43:14 JST 1993
Terminal type is sun
%
```

と表示される。途中でターミナル名を入力するためのプロンプトが出たら、正しいターミナル名を入力する。これで、システムにログインできたので、いろいろなことが試せる。

1.2.2 はじめてログインしたら

パスワードを変更する

```
% passwd
```

と入力する。ただし、 NIS[2]で管理さているマシンでは、代わりに

```
% yppasswd
```

を使う。パスワードは、自分の名前の一部を含んだものや、同じ文字の連続したもの、短いもの、辞書にある単語は避ける。ネットワークで世界中とつながっているので、侵入されないように気をつける。最近は、ユーザのパスワードをチェックする目的で、大きな辞書を使って相当に複雑なパスワードを類推できるプログラムが配布さているので、悪用されると非常に危険である。侵入者は、新しく侵入できた計算機を足掛かりに次のマシンを破ろうとするので、貴方の不用意なパスワードによって、他の多くのマシンの多くのユーザが影響を受けるかもしれない。

自分のディレクトリに必要なファイルをコピーする

シェル等の UNIX コマンドは、起動されると決まったファイルを読み込み、いろいろな設定を行う。それらは . で始まるファイル名でホーム・ディレクトリ (ログインした直後のディレクトリ) に置かれることが多い。それらをスタートアップ・ファイルまたはイニシャライズ・ファイルと呼ぶ。

[2]ネットワーク・インフォメーション・システム。複数の計算機で同一のアカウントが使えるシステム。例えば、KEK B ファクトリーの Sparc Station などで運用されている。

システム管理者は、サイトに応じた変更を加えたいろいろなプログラムを使うのに必要なスタートアップ・ファイルを用意しておき、それらを各自のホーム・ディレクトリにコピーしてあるはずである。

```
% ls -al
```

とやって確認してみよう。.cshrc や.login 等のファイルがあるはずである。もしなかったり間違えて消してしまった時は、

```
% cp  /usr/local/adm/skel/.??*  ./
```

としてコピーしよう。ただし、これらの置いてある場所はそれぞれ異なるので、もしこの場所になければ、システム管理者に聞いてみよう。システムで共通のものが用意されていない時のために、例を巻末に収録してある。

1.2.3 ログアウトの方法

作業が終わったら、ログアウトして、コンソールやターミナルを他のユーザのために開放しなくてはならない。これを怠ると、誰かに大切なファイルを消されてしまったり、メールを読まれてしまうかもしれない。

```
% logout  または
% exit
```

と入力する。もし、

```
% There are running job.
```

と表示されたら、バックグラウンドでなにかコマンドが実行されているので、

```
% jobs
```

と打ち、どういうジョブが実行中かを確認し、本当にログアウトしてよい場合には、`logout` か `exit` を二度繰り返せばよい。

一度ログアウトした後、新しいパスワードでログインできることを確認すること。もしできなければ、何かトラブル（あなた自身かシステム）があるので、管理者に連絡し、対処してもらう。

1.2.4 UNIX のコマンド

コマンドの入力

プロンプト（端末の現在表示されている一番下の行の左端に表示されている。何も設定していなければ、% か #。）の後ろに必要なら適当な引数を付けたコマンドをタイプし最後にリターンキーを押すと実行される。

```
% command_name [-options] [arguments]
```

オプションは、大抵の場合、”–” の後ろにアルファベット一文字を書く。必要ならば、コマンド名やオプションの後ろにスペースを一文字分おいて、引数を書く。途中でコマンドの実行を中止したいときは、 CTRL_C (control key と C を同時に押す) を打つ。また、ターミナルからの入力の終りを告げたいときは、 CTRL_D (control key と D を同時に押す) を用いる。これは標準的な場合で、設定を変えることもできる。もし、 CTRL_C 等がうまく働かない場合には、

```
% stty intr ^c eof ^d
```

と打ってみよう。

UNIX では、その他のオペレーティング・システムではあまり使わないキーをよく使用する。例えば、”^ ”は、端末によっては、”– ”と表示されることもある。” \ ” は、日本語端末では、” ¥ ”が表示されることもある。エスケープ・キーがもし働かなかったら、端末の設定を見直そう。 DEC の VT 端末では、 VT100 モードにするとうまくいく。また ”^ ”や ” ` ”のキーは、他のオペレーティング・システムではあまり使われないので、ターミナルによってはないこともある。

ファイル操作

UNIX のファイル・システムには、階層的なファイル構造が採用されている。 MS-DOS や VMS を使ったことがあればそれを思い浮かべればよい。各階層は、ディレクトリと呼ばれ、ユニークな名前を持っている。仕事の内容に応じて別のディレクトリで作業をしたほうが、ファイルの整理がしやすい。

ここではディレクトリの内容を確認するコマンドや、ディレクトリ間を移動するコマンドを簡単に紹介する。

- ls コマンド
  これはファイルの一覧を見るコマンドである。

  ```
  % ls
  ```

  を実行してみよう。またコマンドには種々のオプションを付けることができる場合が多い。例えば

  ```
  % ls -aF
  ```

  などを実行してみよう。

- cd コマンド
  これはディレクトリを移動するコマンドである。例えば

  ```
  % cd
  ```

  を実行するとルートディレクトリ (階層の頂点) に移動する。

- pwd コマンド
  これは現在自分がどのディレクトリにいるかをみる。

```
% pwd
```

ファイル操作法

今度はファイルの中身を確認したり、ファイルを作ったりといった作業について説明する。ここで、 MS-DOS と最も大きな違いがある。それはファイルには必ず所有者が存在するということである。その所有者の許可なしに、そのファイルを読んだり書いたり消したりできないのである。

- cat , more コマンド
  これは指定したファイルの内容を表示するコマンドである。

```
% cat file_name
```

```
% more file_name
```

more コマンドは画面一つ分表示したら、そこで一時停止する。リターン・キーで一行ずつスクロールし、スペース・キーで一画面ずつスクロールする。また、 q で終了する。

- cp コマンド
  これはファイルを複写するコマンドである。 test1 を test2 に複写するには

```
% cp test1 test2
```

を実行する。複数のファイルをまとめてコピーしたい時は、ワイルドカードが使える。例えば、

```
% cp a/* .
```

は、 a というディレクトリにある任意のファイルをカレントディレクトリ (.) にコピーする。このとき、

```
( 禁止  % cp a/* * )
```

としてはいけない。 UNIX においては、 ” * ”は、そのディレクトリにあるファイル名を展開するコマンドだからである。そのため、今すでにあるファイルの上に上書きされてしまうことになる。 VMS や MS-DOS に慣れているとこの失敗をしがちなので注意しよう。また、 VMS の様に、 ” . ” が特別な意味を持っていることもない。一文字だけ、任意の文字に展開したい時は、 ” ? ”が使える。前に出てきた

```
% cp  /usr/local/adm/skel/.??*  ./
```

の意味を考えてみよう。ワイルドカードは rm, mv など他の多くファイル操作のコマンドでも使うことができる。ワイルドカードや正規表現については、後の章でさらに述べる。

- mv コマンド
これはファイル名やディレクトリ名を変更するためのコマンドである。

```
% mv test2 work1
```

すると test2 は work1 というファイル名に変わる。

- rm コマンド
これはファイルを削除するためのコマンドである。

```
% rm work1
```

- mkdir , rmdir コマンド
これはディレクトリを作ったり削除したりするコマンドである。
例えば tmp というディレクトリを作るには、

```
% mkdir tmp
```

を実行する。またディレクトリを削除するには

```
% rmdir tmp
```

を実行する。 ただしディレクトリ内にファイルやディレクトリが存在すると、

そのディレクトリを削除できない。その時は、

```
% rm -rf tmp
```

コマンドを実行すると、ディレクトリごとファイルが全て削除される。不注意に使うと、必要なファイルまで消してしまう可能性があるので注意する。

オンライン・マニュアルの使いかた

UNIX のシステムにはオンライン・マニュアル機能があり、手軽に UNIX のマニュアルが読める。

- man コマンド
man コマンドを使って man 自身のマニュアルをみる。

```
% man man
```

あとは more コマンドの操作法でみていく。またキーワードでマニュアルを検索する機能がある。例えば、 ” manual ”というキーワードで検索するには

```
% man -k manual
```

を実行する。UNIX 関係のマニュアルや本には、rn(1) とか ctime(3) などの表記がよくある。これは、rn のマニュアルが第 1 章にあるという意味である。例えば、

```
% man 1 rn
```

というコマンドでマニュアルを読んでほしいという意味である。同じ名前で違うセクションに入っている場合があるので、区別するためにこう書かれる。マニュアルの各セクションは、大抵のマシンで、

```
     1       User-level  commands
2       System calls
3       Library functions
4       Devices and device drivers
5       File formats
6       Games
7       Various miscellaneous stuff - macro packages etc.
8       System maintenance and operation commands
```

となっている。たとえば、第 2 章のイントロダクションを、

```
% man 2 intro
```

で読むことができる。

その他のよく使われるコマンド

- w コマンド
  誰がワークステーションにログインして何をしているかをみる。

  ```
  % w
  ```

- last コマンド
  過去にそのワークステーションにログインした人のリストをみる。このリストは長いので、more コマンドと組み合わせて使うとよい。

  ```
  % last $|$ more
  ```

- mail コマンド

  ```
  % mail
  ```

  を実行すると自分が受け取ったメールのリストが表示される。リターン・キーで読むことができる。読み終わったら、” q ”を入力する。読み終わったメールは mbox というファイルに保存されるので再度読むことができる。メールを他人 (例えば石塚君) に送るには、

```
% mail ishizuka
```

を実行し、その後内容を書く。書き終えたら Ctrl-D を入力する。

```
% biff y
```

を実行すると、メールが届くとメッセージが端末に表示されるようになる。自分のホーム・ディレクトリにある .cshrc 等に書いておくとよい。

mail コマンドの他にもメールを読み書きするためのコマンドがある。 MH と呼ばれるメール・システムは、多機能なのでよく使われている。多少複雑なので、ここには紹介しないが、オンライン・マニュアル等で調べて欲しい。

1.2.5 シェル

UNIX システムにログインすると、ユーザは、シェル (shell) と呼ばれるプログラムの中にいる。シェルは、ユーザがタイプしたものを解釈し、コマンドをオペレーティング・システムに伝えるためのプログラムである。

シェルには、大きく分けて二つの系統がある。 B シェル (Bourne shell) と、 C シェルである。有名なものとしては、 B シェルの系統では、 sh (Bourne shell)、 bash (Bourne Again Shell)、ksh (Korn Shell)、 zsh (Super Korn Shell) があり、 C シェルの系統では、 csh (C シェル), tcsh がある。ユーザは、自分の好みに会わせてどのシェルを使うか選べばよい。 chsh コマンドで変更可能なオペレーティング・システムと、管理者に頼まなくてはならなオペレーティング・システムとがある。ログインした状態で、いまどのシェルを使っているか知りたい時は、

```
% echo $SHELL
```

と打つと、使用中のシェル名が表示される。ここでは、標準のシェルとして多くの場合使用されている csh の使い方について述べる。

PATH

PATH については、第 2章で、詳しく述べるが、どのディレクトリにあるファイルをコマンドとして実行するかシステムに教えるための機能である。もし前述のコマンドがうまく働かないときは、 第 2章の PATH に関する部分を読んでほしい。

C シェルのヒストリー機能

以前実行したコマンドをもう一度実行したいとか、修正して実行したいときに便利な機能である。ヒストリー機能を使うには、まずホーム・ディレクトリにある、.cshrc の中に

```
set history=10
```

と書いておく。あるいは、

```
% set history=10
```

と入力する。ただし、.cshrc に書いておかないとログインする度にのコマンドを実行しなくてはならない。この数字は、いくつ前のコマンドまで記憶しておくかということである。もっと多くしたければ大きな値を設定する。

```
% history
```

を実行させると、今までに実行したコマンドの一覧が表示される。

- コマンドの再実行
  一つ前のコマンドを実行させるには

  ```
  % !!
  ```

  ５つ前のコマンドを実行させるには

  ```
  % !5
  ```

  cat で始まるコマンドを実行させるには

  ```
  % !cat
  ```

- 一つ前のコマンドの文字の訂正
  例えば、”date” というコマンドを実行させるつもりが、間違えて”dato” と入力してしまったとする。

  ```
  % dato
dato: command not found
  ```

  と表示されてしまう。そこで、

  ```
  % ^o^e
  ```

  を実行すると、 o が e に入れ変わって、 date が実行される。

- いくつか前のコマンドの訂正

  % $\underbrace{!c}$: $s/old/new$
  ↗ コマンドの再実行の部分

  % !c : $\underbrace{gs}$ $/old/new$
  ↑ 変更を入力行全体に適用する

- 引数の再利用

  % emacs test.tex

  を実行して編集した後で、

  % jlatex $\underbrace{!\$}_{\uparrow\text{test.tex が渡される}}$

  とし、また引数が二つあって、一番目の引数を渡す場合は、

  % emacs test1.tex test2.tex

  % jlatex !^

  とやる。引数全部は、

  % jlatex !*

  とすればいいし、３つめの引数は、

  % jlatex !:3

  とすれば渡される。

csh にはこの他にも foreach 等いろいろなビルトイン・コマンドがある。例えば、そのディレクトリにあるすべてのファイルの拡張子を tmp にしたい時は、

```
foreach file (*)
mv $file $file:r.tmp
end
```

とすればよい。csh のオンライン・マニュアルまたは、[1] 等を参照のこと。

tcsh

このように、csh のヒストリー機能は決して使いやすくないので、それを改善した tsch というものがある。Emacs key binding と vi key binding（それぞれ UNIX で最も良く使われているエディタ）が選べ、それぞれのエディタのコマンドを用いてコマンドラインの編集ができる。また、カーソル・キーを使ってコマンドの再実行をすることもできる。その他の機能は csh と全く同じなのでこちらをログイン・シェルにすることを勧める。ただし、これは全てのワークステーションにインストールされているわけではない。バークレー系の UNIX であれば、

```
% chsh
Changing login shell for takosuke on amyux2.
Old shell: /bin/csh
New shell: /bin/tcsh
```

でログイン・シェルの変更ができる。ただし、 tcsh のある場所はサイトによる。また、 NIS で管理されているワークステーションの場合には、 ypchsh コマンドを用いなければならない。もし、 /bin になければ、 /usr/local/bin も見てみよう。システム V 系の UNIX の場合には、 chsh 等のコマンドがないことが多い。その場合は、システム管理者に頼んで変えてもらおう。

tcsh の一番簡単な使い方は、 CTRL_P を繰り返すことで、以前のコマンドを呼出し、 CTRL_F、CTRL_B、デリート・キーを使って直し、リターン・キーで実行する、というものである。デフォルトでは、インサート・モードになっているので、カーソルの後ろに任意の文字列を入力できる。また、途中までコマンド名やファイル名を書いて TAB キーを押すと、その先を勝手にタイプしてくれる補完機能もある。 emacs モードの時のそれ意外のキーのファンクションを挙げておく。詳しくは、 tcsh のオンライン・マニュアルを参照のこと。

```
^@                      ->  set-mark-command
^A                      ->  beginning-of-line
^B                      ->  backward-char
^C                      ->  tty-sigintr
^D                      ->  delete-char-or-list
^E                      ->  end-of-line
^F                      ->  forward-char
^G                      ->  is undefined
^H                      ->  backward-delete-char
^I                      ->  complete-word
^J                      ->  newline
^K                      ->  kill-line
^L                      ->  clear-screen
^M                      ->  newline
^N                      ->  down-history
^O                      ->  tty-flush-output
^P                      ->  up-history
^Q                      ->  tty-start-output
^R                      ->  redisplay
^S                      ->  tty-stop-output
^T                      ->  transpose-chars
^U                      ->  kill-whole-line
^V                      ->  quoted-insert
^W                      ->  kill-region
^X                      ->  sequence-lead-in
^Y                      ->  yank
^Z                      ->  tty-sigtsusp
^[                      ->  sequence-lead-in
^\                      ->  tty-sigquit
^]                      ->  tty-dsusp
```

コマンドのリコール機能を除けば、その他の機能は C シェルの上位互換なので、 csh と全く同じように使うことができる。

1.3 スタートアップ・ファイル

.cshrc は、 csh が起動されるたびに自動的に実行され、その内容は csh から実行した他のコマンドにおいても有効である。これと同じことを.cshrc 以外でやりたい時は、ファイルの中にコマ

ンドを書いておき、 source コマンドで実行する。例えば、 command_file という名前のファイルにコマンドを書いておき、

```
% source command_file
```

のようにする。.cshrc を書き換えたときも同様に source コマンドを実行しないと、変更が有効にならない。また、.login は、ログインするごとに実行されるので、ターミナルの設定等を書いておくとよい。

1.4 C シェルのエイリアス機能

csh/tcsh では、エイリアス機能を使ってコマンドの別名を定義したり、自分だけのコマンドの定義をができる。例えば cat を type に置き換えたいときは、次のようにする。 type に対して与えられた引数を全て cat に渡すという意味である。これを使うと、他のオペレーティング・システム流にコマンドを再定義することができる。

```
alias type 'cat  \!*'
```

これを .cshrc に書いておくと、毎回ログインするたびに実行する必要がなくなる。例えば、ディレクトリについてだけ長い形式で ls コマンドの出力を見るようなエイリアスは、

```
% alias dir 'ls -l |grep ^d'
```

1.5 ワークステーションを使用するための心得

パーソナルコンピュータと違い UNIX ワークステーションは、ネットワークを通じて複数の人が共用する。次のようなことに、気を付けよう。

- ディスク・スペースを使い過ぎない。ディスクの容量は有限なので、必要のないファイルはすぐに消そう。他の多くのオペレーティング・システムと同様に、 UNIX にもクォータ・システムが存在するが、使用していないことが多い。その場合ディスクが一杯になるまで使ってしまい、他の人の迷惑になることがある。
- 余りに大きなファイルをネットワークでコピーしない。できるだけ避けよう。コピーしている間他の人の仕事の妨げになるかもしれない。数 100MB 単位のファイルはできるだけテープでコピーするか、時間を選ぼう。とくに、大学と大学との間等の広域ネットワークを使う場合には気をつける。
- 一度にあまり多くのプロセスを作らない。たくさんの資源を使い、レスポンスの低下を招く。特に、 X のクライアントをあまり多く使い過ぎないようにしよう。 jobs コマンドや ps コマンドで、時々調べてみよう。
- 時間のかかるファイルをプリンタに出すときは、時間を選ぼう。
- 他人のユーザ名を使用しない。

第 2 章

UNIX 入門

前章で、簡単な UNIX の使い方は説明した。前の章と重複するところもあるが、この章ではもう少し詳しく UNIX の機能について紹介したい。しかしとても全ては無理なので、マニュアルを一度見て欲しい。いろいろな本が出版されているが、システムによる違いがあるので、一番頼りになるのは OS に付属のものである。

2.1 UNIX とは？

ATT により DEC 社のミニコンピュータ PDP シリーズ上で開発されたマルチユーザ、マルチタスクの OS である。 ATT 版は、 System V と呼ばれている。最近多く使われているのは、SVR4(System V Release 4) である。

後に、カリフォルニア大学バークレー校で、 ATT 版に対し特にネットワーク関連の機能に 改良が加えられた BSD 版 (Berkley Software Distribution) というバージョンが作られた。 BSD 版の最新は 4.4BSD であるが、商用の BSD 版ベースの OS は、その前の 4.2BSD か 4.3BSD をもとにしている場合が多い。例えば、 SUN OS 4.1.3 (Solaris 1.1) は、 4.2BSD をもとにし、 4.3BSD の多くの機能を取り込んでいる。

最近では、 System V をベースに BSD の機能の多くを取り入れている場合が多い。 とくにネットワーク関係のコマンドは、 BSD 版のコマンドが取り入れられいることが多い。

ATT 版、 BSD 版の間で基本的な所はほとんど変わらないが、同じ名前のコマンドでも、その書式が微妙に違うことがあるので注意がいる。 System V と BSD のコマンドが PATH [1]の設定により選べるものが多い (SUN OS、 NEWS OS 等)。 SunOS の場合、 /usr/ucb に Berkeley 版のコマンド、 /usr/5bin に System V のコマンドがある。

このように各社それぞれの機能をとり入れたり、様々な変更追加を行なっており、 UNIX といっても計算機の種類によって少しずつ違いがある。特に、システムの管理の方法は、システムにより大きく違うので注意がいる。一般ユーザにとっては、 PATH によって使いなれた方のコマンドを選べるのであればあまり問題はないかもしれない。また、 BSD のソース・コードの多くの部分は、既に自由に配布してもよいことになっているので、 System V ベースのワークステーションにそれらをインストールすることもできる。 BSD 版の場合、一部はすでに無料で配布されているが、 ATT 版、 BSD 版の両方とも決まった金額を支払えば、全体のソース・コードすら買うこともできる。

[1]後述

少し前までの UNIX の歴史については、「Life with UNIX」（アスキー出版）[9] に詳しく載っている。一時期、 OSF (Open Software Fundation、 IBM、 DEC 等がメンバー) と USL(ATT の関連会社) の 2 大勢力に分かれて UNIX 再統合を目指したが、失敗に終わった。 UNIX の版権が ATT から、 Novell に売却されてしまい、 USL は事実上空中分解した。 SUN など USL の主要メンバーだった会社も OSF に現在は参加している。 HP は OSF の OS の採用を取り止め、 SVR4 を採用した。 DEC だけが OSF の OS を採用した。 DEC 以外の主要メンバー、 IBM、 HP、 SUN は協力して COSE という標準環境を作ろうとしている。パーソナル・コンピュータが非常にパワフルになり、ローエンドのワークステーションを凌ぐ性能で、なおかつ低価格なので、ワークステーション・ベンダーはそれに対抗しようと必死になっている。

2.2 ファイル・システム

2.2.1 ファイルの種類

UNIX には、 3 種類のファイルが存在する。

通常ファイル	テキストや、ソース・コード、バイナリ等を含む
スペシャル・ファイル	周辺機器などのデバイスに対応する
ディレクトリ	これら 3 種類のファイルの入れもの

また見かけがファイルと同じ、リンクというものも存在するが、これについては後述する。

2.2.2 ファイル・システムの構造

UNIX ではファイル・システムはツリー構造をしている 2.1。一番根元 / はルート・ディレクトリと呼ばれる。このツリーの基本構造はだいたいのシステムでは同じようになっている。

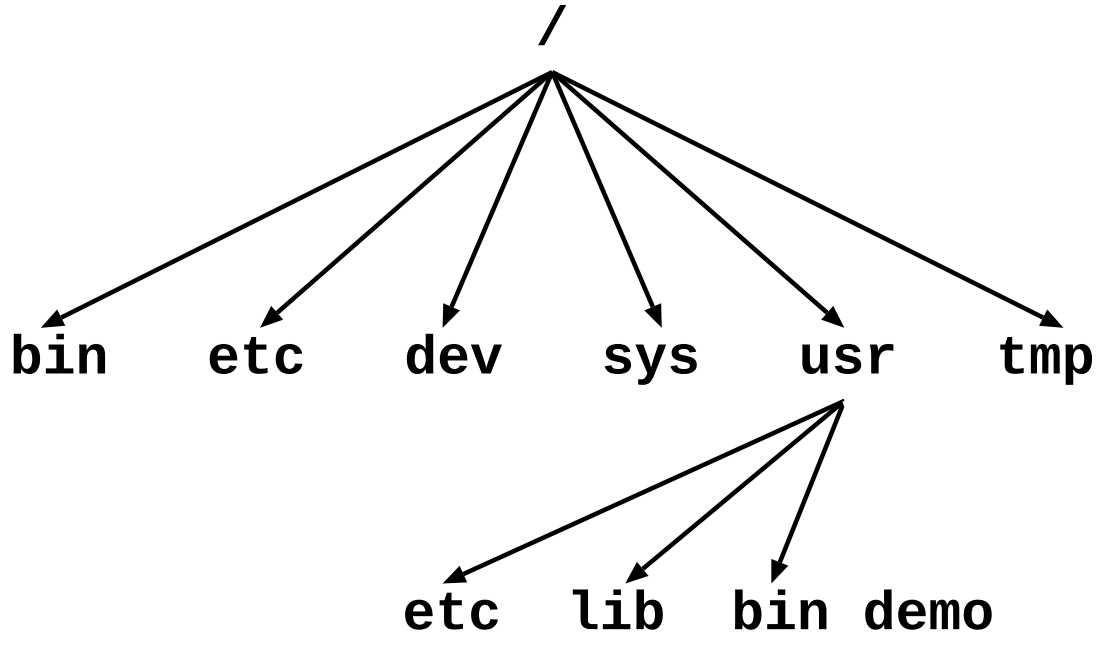


図 2.1: ファイル・ツリー

それぞれのディレクトリの区切りは、’/’で表される。例えば、ユーザのホーム・ディレクトリ (ログインした時いるディレクトリ) は、/home/users/someone と表される（サイトによる）。通常 /etc には、管理用のファイルが格納されている。また /bin には実行可能なバイナリ・ファイル、/dev にはデバイスのためのスペシャル・ファイルが格納される。/tmp は、ユーザやシステムが一時的にファイルを作るのに使われる。ただし、一般ユーザが長時間大きなファイルを保存してはならない。

2.2.3 リンク

リンクとは、いわばファイルの名前のことであり、UNIX では、一つのファイルが複数のリンクをもてる。この機能は、一つのファイルを複数のディレクトリで扱うときに便利である。リンクには、ハード・リンクとシンボリック・リンクの二種類があり、性格も異なる。

ハードリンク

```
% ln aaa bbb
```

とすると、aaa のハード・リンク bbb が作られる。ハード・リンクは、ディレクトリには作ることができないし、ディスクやパーティションをまたがっても作ることかできない。

```
% ls -l aaa bbb
-rw-r--r--  2 sasaki          0 Aug  4 23:42 aaa
-rw-r--r--  2 sasaki          0 Aug  4 23:42 bbb
```

リンクの数が 2 になったはずである。リンクの利点は、この二つは、名前が違っても同じファイルなので、コピーした場合に比べてディスク・スペースが半分節約できる。そこで、例えば bbb を編集してセーブすると、

```
-rw-r--r--  2 sasaki          0 Aug  4 23:42 aaa
-rw-r--r--  1 sasaki          6 Aug  4 23:42 bbb
```

となって、bbb のリンク・カウントが 1 になっていることに気付く。bbb は、すでに aaa と内容が変わったので、別のファイルになったのである。

シンボリック・リンク

シンボリック・リンク（ソフト・リンク）はハード・リンクと異なり、ファイルそのものではなく、ディレクトリにエントリを作って他のファイルを名前で参照する。シンボリック・リンクを作るには、

```
% ln -s aaa bbb
```

とする。例えば、/usr/local/bin を自分のホーム・ディレクトリの /bin にシンボリック・リンクしておくと、cd /bin で /usr/local/bin に簡単に移動できる。

2.2.4 ファイルの操作用コマンド

cd、mv、cat、more 等ファイル操作用コマンドについては、前の章ですでに述べたので、ここでは説明を省略する。

2.2.5 ファイルのパーミッション

ファイルについての情報は、ls コマンドで知ることができる。ファイルには様々な属性があり、実行可能かどうか、ディレクトリかそれとも普通のファイルか、 パーミッションはどうかなど、様々な属性情報を持っている。例えば ls コマンドで表示されるフィールドのそれぞれの意味は、

```
% ls -l .cshrc
-rw-r--r--  1 someone       2813 Feb  1 23:51 .cshrc
```

rw-r--r-- は、ファイルの所有者は read/write できるがそれ以外は read だけ、 1 はリンクの数、someone は所有者、 2813 はサイズ、 Feb 1 23:51 は最終更新履歴、.cshrc はファイルの名前である。また、ディレクトリの場合には、

```
drwx------269 sasaki       5632 Aug  4 15:34 Mail
```

と表示される。最初の一文字の意味は、

```
-    普通のファイル
d    ディレクトリ
l    シンボリック・リンク
```

また、ファイルのパーミッションは、 3 文字ずつ区切って読む。

```
user  group  Others
rwx   ---    ---
```

最初がファイルの所有者自身、次が同じグループに所属するユーザ、最後がそれ意外である。 rwx- は、それぞれ次の意味を持つ。

```
r    read を許す
w    write を許す（上書き、消去）
x    実行を許す
-    許さない
```

ファイルのパーミッションの変更

chmod を使用して、ファイルのパーミッションを変更できる。ただし、スーパーユーザを除く一般ユーザは、自分の所有するファイルのみ変更可能である。 chmod コマンドで使用する一文字の意味は、

```
u user(所有者)
g group
o other
a all

+ そのパーミッションを足す
- そのパーミッションを除く
= 与えられたパーミッションに設定する

r read
w write
x execute
```

となっている。例えば、あるファイルが、

```
-rw-------  1 hehehe        204 Mar 29  1994 .cshrc
```

の時、全ての人が read 可能にするには、

```
% chmod a+r .cshrc
```

というコマンドを実行する。

```
-rw-r--r--  1 hehehe        204 Mar 29  1994 .cshrc
```

のように変わったはずである。また、例えばユーザだけがこのファイルを実行可能にするには、

```
% chmod u+x .cshrc
```

とする。また、全てに rw を許すような設定にしたければ、

```
% chmod a=rw .cshrc
```

とすることができる。

2.3 コマンドの実行と PATH

UNIX のコマンドには二種類あり、シェルのビルトイン・コマンドと、ファイルになっているものがある。 シェルのビルトイン・コマンドは、シェル内部で実行される。 csh の foreach や setenv 等がそうである。それ以外のコマンドは全てファイルの名前そのものか、その別名である。 PATH という環境変数に定義されたディレクトリの中で実行可能形式のものが自動的に呼びだされる。 たとえば、 PATH に

```
PATH=~/bin:.
```

と設定してあったとすると、コマンドはカレント・ディレクトリおよび /bin からだけ探される。[2]普通の UNIX システムでは、コマンドの多くは、 /bin か /usr/bin にあるので、この二つは必ず PATH に設定されていなければならない。

前の例に戻って、例えば hello と打つと、 hello という名前の実行可能ファイルが PATH の中から探されて実行される。 PATH を設定するとき順番に気を付けないと、自分の予期しなかったものが起動されてしまうこともある。すでにあるコマンドと同じ名前で新しいコマンドを登録してしまうと混乱が起きる。実際にそのコマンド名で実行されるファイルの名前を知りたいときは、 which コマンドを使う。

[2]PATH に . を入れるのは非常に危険である。特に、最初に入れてはならない。もし、カレント・ディレクトリに、自分が実行したいコマンドと同名のファイルがあって実行可能なら、そちらが実行されてしまう。それが、もし自分のファイルを全部消すようになっていたりしたら...

2.4 パイプ

パイプ処理は、UNIX における特徴的な機能である。あるコマンドの出力を別のコマンドに渡すのに使われる。例えば、今いるディレクトリにたくさんファイルがあって、その中から xxx という文字列を含んだファイルを探したいときは、

```
% ls -l|grep xxx
```

とすればよい。縦棒がパイプを表す。もしパイプがないと、一旦 ls コマンドの出力をファイルに書いてから、そのファイルに対して grep コマンドを実行しなければならない [3]。パイプは、何段も繰り返して使用してよく、一般的には、

```
% command1 | commmand2 | command3.......
```

のように使用することができる。

2.5 リダイレクション

UNIX のコマンドの多くは、標準入力（普通は、端末からのキー入力）から読んで、標準出力（普通は、端末の画面）にその結果を出力するように作られている。また、エラー・メッセージは、標準エラー（普通は、端末の画面）に出力される。これらの入出力をファイルに切替えるのが、リダイレクションという機能である。例えば、ls コマンドの出力をファイルに書きたい時は、

```
% ls > out
```

とすると、out というファイルに書き込まれる。前のパイプの例は、

```
% ls > out
% grep xxx out
```

としても同じ結果が得られる。[4]

自分のコマンドに in というファイルから入力を読ませて、out というファイルに書くには、

```
%       a.out<in>out
```

とする。すでにあるファイルの後に付け足したい時は、

```
% a.out >> out
```

とする。すでにあるファイルに上書きしたい時は、

```
% a.out >! out
```

とする。最後に、表 2.1に、リダイレクションの機能をまとめておく。

[3] 実際多くの OS ではそうする

[4] この中間的なファイルを取っておく理由がないなら、パイプの例の方が便利であることは一目瞭然である

リダイレクトの指示	動作
>	標準出力をリダイレクトする
> &	標準出力と標準エラーをリダイレクトする
<	標準入力をリダイレクトする
>>	標準出力を付け足す
>> &	標準出力と標準エラーを付け足す
|	パイプ（ほかのコマンドの標準入力へ出力）

表 2.1: リダイレクション

2.6 正規表現

正規表現は、ファイル名などの文字列のパターン認識に用いられる。シェルだけでなく、 vi、 grep 等多くの UNIX のコマンド内で使用される。知っておいたほうがよい最小限のものを紹介する。

? 任意の一文字マッチング
* 任意の文字列マッチング
\ エスケープ 特殊文字の効果を打ち消す
$ 行末

2.7 環境変数と環境設定

シェルは、環境変数を持っており、それによってユーザのいろいろな環境設定を行える。コマンドやプログラムは、この変数によって機能や設定を変えることもできる。したがって正しく設定されていないと、コマンドの動きがおかしくなることもあり得る。例えば、 TERM という環境変数には、現在使用しているターミナルの形式を設定する。 emacs 等のプログラムは、この変数を見て、そのターミナルを制御するのに適したコードを送る。 csh では、環境変数は setenv コマンドで設定し、 env または printenv コマンドで表示する。 B-shell では、 export コマンドを使用する。例えば、デフォルトのプリンタの名前を登録したければ、

```
% setenv PRINTER  printer_name
```

とする。また、その設定値を確かめるには、

```
% printenv PRINTER
```

とする。

環境変数の設定は通常 .cshrc ファイルに書いておく。.cshrc ファイルは、そのシェルが起動されたときにしか実行されないので、.cshrc ファイルを変更したら、 source コマンドで変更した内容を csh に教えてやる必要がある。

```
% source .cshrc
```

とする。知っておくと便利な、環境変数を挙げておく。

```
HOME      ホーム・ディレクトリ
PATH      パス
PRINTER   デフォルトのプリンタ名
EDITOR    アプリケーション内で起動されるデフォルトのフルスクリーン・エディタ
VISUAL    アプリケーション内で起動されるデフォルトのライン・エディタ
```

2.8 プロセス

プロセスとは、 OS が資源を管理する単位である。 UNIX システムにログインすると、 OS は親のプロセスをスタートさせる。その親プロセスから子プロセスを作ることができる。これをフォーク (fork) するという。端末からの入出力を受けていないプロセスをバックグラウンド・プロセスと呼ぶ。 csh の場合、自分の子プロセスは、 jobs コマンドで表示される。子プロセスをフォークするには、コマンドの後ろに & を付けて実行する。例えば、 mule をバックグラウンドで実行するには、

```
% mule &
[1] 14947
```

とする。コマンド実行後画面に表示される数字は、最初の鍵括弧の中が、その親シェルからフォークされているバックグラウンド・プロセスの通し番号、次がプロセス ID である。これの実行を中止するには、 jobs コマンドで確かめて、

```
% jobs
[1]    Running                 mule

% kill %1
```

と、通し番号を % の後ろに付けて kill コマンドに与えるか、

```
% kill 14947
```

とプロセス id を与える。プロセス id を調べるには、 ps コマンドを用いることができる。 kill コマンドは、システムで決められたシグナルをプロセスに与える。そのシステムにどういうシグナルが用意されているかは、 man signal をするか /usr/include/signal.h の中身を見ればよい。よく使われるシグナルとして、

```
  1  HANGUP
  9  KILL
 15  TERMination
```

がある。あるプロセスを強制終了させるには、

```
% kill -KILL process_id
```

または、

```
% kill -9 process_id
```

とする。また、そのシェルの子プロセス以外の実行中のプロセスを確かめるには、 ps コマンドを用いる。 BSD 系のオペレーティング・システムなら、

```
% ps -axv
```

System V 系なら、

```
% ps -eal
```

で、全プロセスの詳しい情報が表示される。

2.9 ネットワーク機能

優れたネットワーク機能も UNIX の特徴の一つである。いろいろなコマンドが用意されているだけではなく、それを自分でプログラムするのに必要なライブラリも提供されてる。

一般的には、ネットワークには様々なプロトコル（通信のための約束）があるが、 UNIX では TCP/IP が主に使われている。このプロトコルでは、ネットワークにつながれた各ホストは、IP アドレスとホスト名の両方で識別される。また、広域ネットワークで通信を行うためにドメイン名を持つ。例えば、 KEK は kek.jp というドメイン名を使っている。ドメイン名を合わせたホスト名、または IP アドレスは重複させてはならない。

日本国内の IP アドレスとドメイン名は JP-NIC というところが管理しており、そこから使用許可を得なければならない。現在では、登録料と使用料を支払うようになっている。 KEK では、256*256 分の IP アドレスを確保しており、 KEK 内では、各グループに割り当てられた番号を許可を得て使うことができる。

勝手に IP アドレスを付けたり、許されていないアドレスを使ったりすることはしてはいけない。 KEK やそれぞれの大学のルールに従って、新しい機器を導入するようにしよう。もし勝手なことをすると、そのネットワーク全体が利用不可能になり、多くの人に迷惑をかけることになる。ワークステーションのネットワーク関係の設定は出来るだけ慎重に行い、分からないときは管理コマンドを実行したり、ネットワークに接続したりしてはならない。

UNIX には、ネットワークを使った様々なアプリケーションを開発できるように、豊富なライブラリが付属しており、ユーザが自分でアプリケーション・プログラムの開発を行えるようになっている。プログラミングについては専門書 [13] を見てもらうことにして、ネットワーク関連のいくつかのコマンドについて簡単に説明しよう。

リモート・ログイン

ほかのマシーンにログインするには、 rlogin か telnet コマンドを用いる。またネットワークを通してのコピーには、 ftp か rcp を使う。[5]別のホストでコマンドを実行するには、 rsh コマンドを使う。

KEKVAX にログインするには、

```
%telent  kekvax (or kekvax.kek.jp)
```

[5]rcp や rsh を使うには、ホーム・ディレクトリに .rhosts というファイルにアクセスを許すホストとユーザ名を書いておく。%man rsh を試せ。例えば、
amyux1.kek.jp takosuke
amyux1 takosuke
のように、ドメイン名付きとそうでないものの両方を書いておくとよい。

または、

```
%rsh  kekvax（or kekvax.kek.jp）
```

とする。通常、同じネットワーク・ドメインのホストに対しては、ドメイン名を省略できる。別のマシンでコマンドを実行するには、

```
%rsh amyux1 w
```

とする。この場合は amyux1 で w コマンドを実行した結果が表示さる。

他のユーザとのコミュニケーション

他のターミナルからログインしているユーザにメッセージを送りたいときは、 write コマンドが使える。

```
%write someone
Let’s have lunch!
CTRL_D （control key と D を一緒に押す。）
```

相手のターミナルには

```
Message from dareka@dokoka on ttyp3 at 12:10 ...
Let’s have lunch!
EOF
```

のように表示される。また、 mail や talk を使えば離れたところにいる人と簡単に連絡が取れる。

```
%mail someone
Hello,
CTRL_D （control key と D を一緒に押す。）
```

とすると、メッセージが someone というユーザに送られる。 talk はインタラクティブな筆談プログラムである。

```
%talk someone@somewhere
```

とすると、 somewhere というホストの someone という人を呼ぶ。もし、

```
Message from Talk_Daemon@host1 at 21:28 ...
talk: connection requested by someone@somewhere
talk: respond with:  talk someone@somewhere
```

というメッセージが来たら、誰かが talk したがっているので、

```
%talk someone@somewhere
```

として答えると、画面が二つに割れて筆談ができる。

2.9.1 ホスト間でのファイルのコピー

rcp

例えば、ホスト amyux2 から自分のホストのカレント・ディレクトリにファイルをコピーするには、

```
%rcp amyux2:.cshrc .
```

逆に、カレント・ディレクトリのファイルをホスト amyux2 にコピーするには、

```
%rcp .cshrc amyux2:.cshrc
```

とする。

FTP

File Transfer Program という名前の通り、ネットワークを通じファイルのコピーをするのに使う。例えば、今ログインしているホスト A と別のホスト B との間でファイルのコピーをしたいときは、次のようにする。まず、

```
%ftp B
```

とコマンドを打つとユーザ名とパスワードを聞かれるので、それを入力する。この時パスワードはエコーされない。もし ftp した先のユーザ名が今いるホストと同じならユーザ名に対しキャリッジ・リターンでよい。相手が SUN の時の表示の例。

```
% ftp dokoka
Connected to dokoka
220 dokoka FTP server (SunOS 4.1) ready.
Name (dokoka:watasi): anata
331 Password required for anata.
Password:
230 User anata logged in.
ftp>
```

もしパスワードやをユーザ名を間違えると、

```
530 Login incorrect.
Login failed.
```

と表示される。この時は、 user という ftp のサブ・コマンドを打って、もう一度ユーザ名とをパスワードを入力する。ログインに成功したら、 help と打つと使えるコマンドの一覧が表示される。ディレクトリのリストをみて、ファイルを取ってくる例。

```
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (130.87.117.1,1114) (0 bytes).
whet.doc
whet.f
whetston.c
226 ASCII Transfer complete.
```

```
204 bytes received in 0.22 seconds (0.91 Kbytes/s)
ftp> get whet.doc
200 PORT command successful.
150 ASCII data connection for whet.doc (130.87.117.1,1118) (313 bytes).
226 ASCII Transfer complete.
local: whet.doc remote: gomi.doc
319 bytes received in 0.05 seconds (6.2 Kbytes/s)
```

もし、違うファイル名でセーブしたい時は、

```
ftp> get whet.doc  test.doc
```

のように指定する。またファイル名にワイルドカードを使いたい時は、

```
ftp> mget whet.*
```

のようにする。

逆に、ファイルを相手に送りたい時は、 get,mget の代りに put,mput を使う。この時、ファイルごとにコピーするかどうかプロンプトされるのが煩わしいと思ったら、 mget,mput の前に prompt (noint の時もある) というコマンドを打つと、 プロンプトされなくなる。 ftp を終了するには、 quit と打つ。

```
ftp> quit
221 Goodbye.
```

2.10 その他のコマンド

UNIX は、カーネル部分をなるべく小さくしようという発想で設計されたので、コマンドの多くは後から付け加えられた。他数のユーザによってプログラムされ、命名されたので、コマンド名に規則性が全くない。オプションの与え方もまちまちで、複雑すぎることもある。これが、UNIX を使いにくいと思わせる最大の原因だろう。非常に多くのコマンドが用意されているにもかかわらず、使いこなすのは簡単ではない。ここにはコマンドの一覧を載せるが、詳しくはオンライン・マニュアルを参考にしてほしい。場合によっては、マニュアルを見てもよく分からないかもしれない。それが、 UNIX 関係の入門書 [7] が山のように出ている理由かもしれない。これから紹介するコマンドは、 /bin, /usr/bin, /usr/ucb にある。オペレーティング・システムによっては別の場所か、もともとないコマンドもあるかもしれない。特に BSD と書いてあるコマンドは、 System V にはないことがある。

apropos (1)	キーワード検索によってコマンドを探す
ar (1)	アーカイブとライブラリの維持管理プログラム
at (1)	コマンドを指定時刻に実行する
atq (1)	指定された時間にジョブを実行する
awk (1)	パターン検索と処理の為の言語
basename (1)	ファイル名の接辞を除去する
cal (1)	カレンダを表示する

cb (1)	C 言語プログラムの清書プログラム
cc (1)	C コンパイラ
cd (1)	作業ディレクトリを変更する
chfn, chsh, passwd (1)	パスワード・ファイル情報の変更 (BSD)
chgrp (1)	グループを変更する
chmod (2)	ファイルのモードを変更する
clear (1)	端末装置の画面をクリアする
col (1)	逆改ページためのフィルタ
colcrt (1)	nroff の出力を CRT で見るためのフィルタ
compress, uncompress, zcat (1)	データの圧縮と展開を行う
cp (1)	コピー
csh (1)	C 言語に似た構文を持つシェル (コマンド・インタープリタ)
dbx (1)	デバッガ
deroff (1)	nroff、 troff、 tbl、 eqn 構造を除去する
df (1)	ファイル・システムの使用状況を報告する
du (1)	ディスクの使用状況を表示する
error (1)	コンパイラのエラーメッセージを解析し表示する
expand, unexpand (1)	タブのスペースへの展開とその逆を行う
f77 (1)	Fortran 77 コンパイラ
false, true (1)	真理値を返す
finger (1)	ユーザ情報検索プログラム
fpr (1)	Fortran ファイルを表示する
fsplit (1)	Fortran のマルチルーチン・ファイルを別々のファイルに分割する
ftp (1C)	ARPANET ファイル転送プログラム
grep, egrep, fgrep (1)	ファイルをサーチしてパターンを探す
groups (1)	グループのメンバーを表示する
hostid (1)	現ホストシステムの識別名をセットまたは表示する
hostname (1)	現ホストシステムの名前をセットまたは表示する
indent (1)	C プログラム・ソースに字下げ処理を施して整形する
man (1)	キーワードによってマニュアルを検索したり、 マニュアルを出力する
man 3 intro	C ライブラリ関数の紹介
man 3f intro	FORTRAN ライブラリ関数の紹介
man 1 intro	コマンドの紹介
man 2 intro	システムコールとエラー番号の紹介
iostat (1)	入出力に関する統計を報告する
lastcomm (1)	実行されたコマンドを逆順に表示する
login (1)	UNIX へのログイン
lpq (1)	スプール・キューを調べるプログラム (BSD)
lpr (1)	オフライン・プリント (BSD)
lp (1)	オフライン・プリント (SYSV)

lptest (1)	ライン・プリンタの波状パターンの生成
ls (1)	ディレクトリの内容の表示
mkdir (1)	ディレクトリの作成
mt (1)	磁気テープ操作プログラム
mv (1)	ファイルの移動または改名
nice (1)	コマンドの実行優先順位の変更
netstat (1)	ネットワーク・ステータスの表示
nroff (1)	テキスト・フォーマッティング
od, hd (1)	8 進数、 10 進数、 16 進数のアスキー形式のダンプ
passwd (1)	パスワードの変更
pr (1)	ファイルの出力
printenv (1)	環境変数の表示
ps (1)	プロセス状態の表示
pwd (1)	作業ディレクトリ名の表示
quota (1)	ディスクの使用状況とリミットの表示
ranlib (1)	アーカイブをランダム・ライブラリへ変換する (BSD)
rcp (1C)	リモート・ファイルのコピー
rlogin (1C)	リモート・ログイン
rm,rmdir (1)	ファイルまたはディレクトリの削除 (リンクの解除)
rsh (1C)	リモート・シェル
ruptime (1C)	ローカル・マシンのホストの状態を表示する
rusers (1C)	ローカル・マシン上に誰がログインしているか表示
rwall (1C)	ネットワーク上のすべてのユーザへ write する
sed (1)	ストリーム・エディタ
sleep (1)	ある期間のプロセスの実行休止
spell, spellin, spellout (1)	スペル・エラーの検出
su (1)	他のユーザへの切り替え
tar (1)	テープのアーカイブ
telnet (1C)	TELNET プロトコルへのユーザ・インタフェース
tip, cu (1C)	リモート・システムに接続する
tn3270 (1)	IBM VM/CMS へのフルスクリーン・モードでの リモート・ログイン
touch (1)	ファイルの最終修正時刻を更新する
tr (1)	文字の変換
troff, nroff (1)	テキスト・フォーマッティング及びタイプセッティング
tty (1)	端末名を取得する
users (1)	ログイン中のユーザの簡単なリスト
uuencode, uudecode (1C)	バイナリ・ファイルをメールで転送できるよう エンコード・デコードする
vmstat (1)	仮想メモリの統計情報を報告する
w (1)	システムの使用者とその作業内容の表示

wc (1)	ワードカウント
whatis (1)	コマンドが何であるかを示す
whereis (1)	プログラムのソース、バイナリ、マニュアルの ファイルの位置を示す
which (1)	別名とパスを含むプログラム・ファイルの位置を示す
who (1)	システム上に誰がいるかを示す
whoami (1)	現在の実効ユーザ ID を表示する
whois (1)	DARPA インターネット・ユーザ名ディレクトリ・サービス
write (1)	別のユーザへメッセージを送る
yes (1)	引数または y を繰り返し出力する

これらは、ほんの一例である。 OS に標準的にバンドルされているもの以外に、大抵の場合管理者が多くのコマンドを /usr/local/bin 等にインストールしている。 UNIX 用の多くのフリー・ソフトウエアが使用可能なので、それらがどこにあるかは、それぞれのシステムの管理者に聞いてみよう。 KEK 内の多くのホストでは、 /kek/local/bin にインストールされている。どのコマンドをどう使うかは、それぞれの機種のマニュアルや関連図書、雑誌などに詳しく載っている。「Life with UNIX」や、「やさしい UNIX」等を読んでみるとよいだろう。

2.11 UNIX の学び方

UNIX に限らず計算機を上手に使えるようになろうと思えば、とにかくたくさん使って慣れるしかない。参考書がたくさん出版されているので、それらを片手に実際試してみよう。本の中には、あまり実用的でないオペレーティング・システムの研究者向けのような本もあるので、そういうものを避け、できるだけ例題の豊富な実用的なものを買うとよい。また、 UNIX magazine や UNIX user などの雑誌も役に立つ。雑誌のバックナンバーもできるだけそろえるとよいだろう。 KEK の情報資料室にもそろっている。

参考書をいくつか巻末に挙げておく。非常にたくさんの本がでているので、自分に合ったレベルのものを探して買うとよいだろう。自分の使おうとしている計算機が BSD か System V かは知った上で買うようにしよう。

第 3 章

テキストエディタ

この章では、代表的なテキスト・エディタである emacs と vi の簡単な使い方を紹介する。好みによりどちらを使っても構わない。 vi はほぼ全ての UNIX 計算機で使えるが、 emacs はインストールされていない場合もある。 X の下で使える xedit 等のエディタもあるが、テキスト端末でも使える vi か emacs の片方は知っておいた方がよい。システム管理者は、ある程度 vi が使えることが要求される。

3.1 emacs

GNU (Gnu is Not UNIX) プロジェクトにより作成配布されているフリー・ソフトウエア (無料で配布及び使用ができるソフトウェア) である。[1] Extended Macro からその名が取られた。非常に拡張性があり、ユーザの必要な環境にカスタマイズできる。 Emacs lisp という LISP 言語の一種でプログラムをする。単なるテキスト・エディタを越えて、一つのプログラミング環境に近いものとなっている。一度 emacs を立ち上げたら、その中で全てのことができるようなっている。

英語版のマニュアルは TEX のソース・コードが公開されており、日本語版も共立出版 [2] から出ている（ただし、少し版が古い）。また、参考書も多数出版されている。 [3]

3.1.1 日本語 GNU Emacs と Mule

日本語 GNU Emacs は、 GNU Emacs に日本語の表示入力機能を取り入れたものである。また Mule は、日本語に限らず他国語のサポートを取り入れた emacs である。日本語入力などを除いた基本的な機能は全く英語版と同じと考えてよい。日本語 Emacs を作った人々が Mule の開発も行っており、日本語 Emacs のヴァージョンは固定された。これから使い始めるなら、 Mule を使った方がよい。基本的な使い方はほぼ同じなので、この後の emacs というコマンド名は、NEmacs や Mule などそのホスト名で有効な名前に読み変えてほしい。

[1] ただし、 GNU 関係のソフトウエアには配布や使用には条件がついている。詳しくはソース・コードのファイルの中に含まれる著作件関係のファイルを読むこと。

3.1.2 Emacs の立ち上げと終了

ここで使うコマンドの記述方法であるが、 GNU Emacs のマニュアルに従う。 M-x は、エスケープ・キーの次に x を入力するという意味で、 C-x や C-c は、コントロール・キーを押したまま x または c を押すという意味である。ここではファイル名 foo.x の FORTRAN ソース・ファイルを編集するとして話を進める。 Emacs は、
%emacs foo.x
で立ち上がる。この画面で FORTRAN のプログラムを書いたり、文章を書いたりすることができる。画面の下の方を見ると日本語 Emacs では、

```
-----NEmacs:  foo.x               (-EE-:Text)-All--------------------
```

また、 Mule では、

```
[--]J_:-----Mule: foo               (Text Fill)--All-----------------
```

のように表示される。ここでとにかく foo.x と表示されていることだけは確認しておくこと。その他については設定により異なる場合があるのであまり気にしなくてよい。コマンド

```
M-x fortran-mode
```

で、 FORTRAN モードになる。

```
-----NEmacs:  foo.x               (-EE-:Fortran)-All--------------------
```

または、

```
[--]J.:-----Mule: foo.x   line#1      (Fortran)--All--------------------
```

に変わったはずである。通常は、 FORTRAN のソース・ファイルには、.f を最後に付けるが、.f で終るファイルを読み込むと、自動的に FORTRAN モードになるので、この作業は不要である。.f で終らない FORTRAN ソース・ファイルを編集する時に、上のようにする。このモードにすると字下げなどが容易に行なえる。この他にも、 Text モード、 TEX モード、 LATEX モードなどのモードがある。マニュアルを見て、目的にあったモードをセットすると編集作業は容易になる。

次に Emacs の終了の仕方であるが、これは Emacs の画面上で、

```
C-x C-c
```

とコントロール・キーを押しながら [x] そして [c] のキーを押せばよい。この方法は Emacs で編集したファイルを保存 (save) して終了する (保存するかどうか質問される）。

3.1.3 Emacs の機能

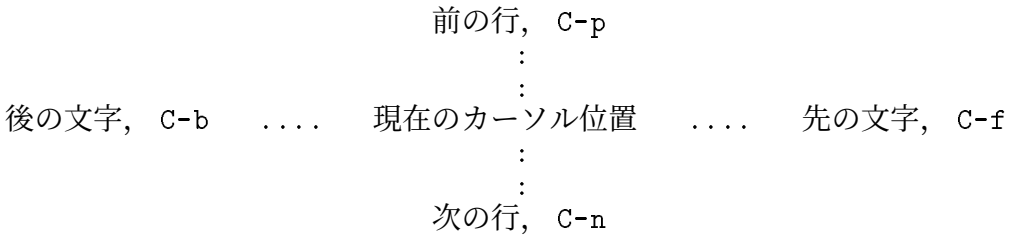
Emacs のコマンドについては、このマニュアルの最後に GNU Emacs コマンド・リファレンスカードとしてまとめられているので、見て欲しい。ここでは、その中の基本的なものを紹介する。

基本の基本 — カーソルの移動 etc..—-

Emacs での一番基本的なコマンドはここで説明するよりも、実際に自分で使ってみるのが一番よいので、自分で Emacs を立ち上げて、

```
CTRL-h t        （英語版）
または、
CTRL-h T        （日本語版）
```

を試して欲しい。このコマンドで Emacs のチュートリアルが始まる。指示に従ってコマンドを打って練習しもらいたい。
また、 C-h i で info を起動するとより詳しい情報が得られる。簡単に、チュートリアルのまとめを載せておく。カーソルの移動は、下図のようになってる。



また、次のような行を~/.emacs のどこかに書いておけば、 Sun ワークステーションでもカーソル・キーで移動できる。 emacs は、起動された時にこのファイルを読み込み、初期設定を行う。すでに、 emacs を使い込んでいそうな人のファイルをコピーさせてもらったほうがよいかもしれない。

```
; Make Sun arrow keys work
(defvar esc-bracket-map (make-sparse-keymap)
        ``*Keymap for ESC-[ encoded keyboard'')

(progn
  (define-key esc-map ``[`` esc-bracket-map)         ; Install esc-bracket-map
  (define-key esc-map ``[A'' 'previous-line)         ; R8
  (define-key esc-map ``[B'' 'next-line)             ; R14
  (define-key esc-map ``[C'' 'forward-char)          ; R12
  (define-key esc-map ``[D'' 'backward-char)         ; R10
  (define-key esc-map ``[[`` 'backward-paragraph))   ; the original esc-[
```

他の簡単な機能については、次の表にまとめてみた。

移動の方法	
C-v	前に一画面分進む
ESC v	後ろに一画面分戻る
C-l	画面を書き直す
ESC f	一単語先に進む
ESC b	一単語後に戻る
ESC]	段落の終わりに移動
ESC [	段落の先頭に移動
C-a	行の最初に移動
C-e	行の最後に移動
ESC <	ファイルの最初に移動
ESC >	ファイルの最後に移動

困ったときは？	
C-g	中止 (わけがわからなくなったら、これだ！)

削除の方法	
<Delete>	カーソルの直前の文字を削除
C-d	カーソルのある文字を削除
ESC <Delete>	カーソルの直前の単語を削除
ESC d	カーソル位置以降にある単語を削除
C-k	カーソル位置から行末までを削除
C-x u	UNDO(取り消し)

File 操作	
C-x C-f	別のバッファへのファイルの読み込み（Find）
C-x C-i	ファイルの挿入
C-x C-s	ファイルの保存（Save）
C-x C-b	バッファリストの表示
C-x C-c	エディタを終了する

カット & ペースト

ファイルを編集していて一文字だけでなく、行単位あるいはリージョン単位で削除する場合がある。このような時、行単位で削除する場合は、削除したい行の先頭にカーソルを移動させ、

```
C-k
```

とする。これでその一行は画面上から削除されるが、削除された一行は実は Emacs 内にあるバッファに取り込まれていて、

```
C-y
```

で画面上に復帰させることができる。

またリージョンを削除する場合は、削除したいリージョンの先頭にカーソルを移動させて、

```
ESC-Space
```

のように [ESC] を押し、その後に [Space] を押す。そしてリージョンの最後部にカーソルを移動させて、

```
C-w
```

と打つことによって削除される。削除せずにリージョンをコピーしたい場合は、 C-w の代わりに、

```
ESC w
```

を行う。この [ESC w] により Emacs 内のバッファにそのリージョンが読み込まれているので、これも

```
C-y
```

で画面上に復帰させることができる。

これらの便利な使い方は実際に使ってみて研究してみよう。

3.1.4 FORTRAN プログラムの編集

.f で終るファイルを読み込むと自動的に FORTRAN モードに入る。編集用に様々なコマンドが用意されているが、そのうちいくつか挙げておく。

```
TAB             fortran-indent-line
ESC             Prefix Command
C-c             Prefix Command
;               fortran-abbrev-start
C-c C-n         fortran-next-statement
C-c C-p         fortran-previous-statement
C-c C-r         fortran-column-ruler
C-c C-w         fortran-window-create
C-c ;           fortran-comment-region
ESC C-q         fortran-indent-subprogram
ESC LFD         fortran-split-line
ESC C-h         mark-fortran-subprogram
ESC ;           fortran-indent-comment
ESC C-e         end-of-fortran-subprogram
ESC C-a         beginning-of-fortran-subprogram
```

FORTRAN モードでは、 make コマンドを起動し、コンパイル・エラーの出た行を容易に見つけ出し、編集することもできる。

```
ESC x compile
```

と打つと、ミニバッファに次のような表示が出る。

```
Compile command: make -k
```

これでよければ、リターン・キーを押す。

```
f77 xx.f
```

と書き直すこともできる。もし、エラーが検出されたら、

```
C-x ‘  (next-error)
```

で、エラーのある行にカーソルを飛ばしてそこを編集する。他にも多くの機能があるので、 C-h m や C-h b で確認してみよう。

3.1.5 Mail

Emacs の中からいろいろなコマンドを呼び出すことができるが、メールもその一つである。いつくかの方法があるが、ここでは mh-mail の簡単な使い方を紹介する。次のようなエイリアスを定義しておくと便利である。日本語 emacs では、

```
alias rmail emacs -nw -f set-fileio-jis -f mh-rmail
alias smail emacs -nw -f set-fileio-jis -f mh-smail
alias scan emacs -nw -l mh-e -f set-fileio-jis -f mh-scan
```

Mule では、

```
alias rmail mule  -f mh-rmail
alias smail mule  -f mh-smail
alias scan mule  -f mh-scan
```

とする。コマンドの意味は次の節で説明する。 mh はメールを操作するためのパッケージとインターフェースである。通常 mh コマンドは、 /usr/new、 /usr/local/new、 /usr/loca/mh、 /usr/local/b /usr/local/bin/mh 等にインストールされている。どこにインストールされているか調べて、 PATH に加えておくこと。そして、 inc と打つと、必要な設定が行なわれる。これは、一度だけやればよく、次からはやる必要はない。もしインストールされていない場合には、 ftp.kek.jp:/pub/GNU からコピーしてインストールしよう。

メールを送りたい時は、 smail コマンドを用いる。すると、 emacs が立ち上がるので、送り先などを入力を行う。あとは、普通に emacs を使って、中身を編集し、最後に C-c C-c で送る。ただセーブしただけでは送られない。

メールを読みたい時には rmail を実行すると、 emacs が立ち上がって、新しいメールの一覧が表示される。ここで、スペース・キーを使って読みたいメールを選ぶ。古いメール（一度読んでしまったもの）を読みたいときは、 scan と打てば、メールの一覧が表示されるので、同じようにスペース・キーを使って選択し、読むことができる。

3.1.6 日本語の入力

かな漢字変換には Wnn + egg や SKK 等が使える。egg が多く使われているので、その使い方を述べる。まず、JSERVER という環境変数が正しく設定されているか調べよう。

```
% env|grep JSERVER
```

とする。何も表示されなければ、システム管理者に相談して、かな漢字変換のサーバが実行されているマシン名を設定する。

漢字変換を始めるには、C- \ を入力する。emacs のステータス行が、日本語 emacs では、

```
[----] ---- NEmacs: *scratch*          (Lisp Interaction)---All---[JJJ-]
```

から

```
[ a あ] ---- NEmacs: *scratch*          (Lisp Interaction)---All---[JJJ-]
```

に、MULE では、

```
[--]J_:-----Mule: *scratch*          (Lisp Interaction)--All------------
```

から

```
[あ]J_:-----Mule: *scratch*          (Lisp Interaction)--All------------
```

に変わったはずである。次に C- \ を入力するまでローマ字仮名変換が行なわれる。かなになったところで、スペース・キーを押すと漢字に変換される。このとき、かなは縦棒で囲まれている。このモードをフェンス・モードと呼ぶ。別の候補を選びたい時には、もう一度スペース・キーを押す。候補がなくなるまで、スペース・キーを押すたびに新しい候補が出てくる。候補の一覧を見たい時は、ESC s を打つ。

日本語の漢字コードとしては、EUC (DEC 漢字とほぼ同じ)、JIS、Shift JIS(MS-DOS 漢字コード) が広く使われている。いろいろなコードがあるので紛らわしいが、UNIX でよく使われているのは JIS と EUC 漢字コードである。どの漢字コードも Nemacs は扱えるが、そのサイトによって決めているコードでファイルに書いておかないと、日本語の TEX がうまく動かないなどの問題を起こす。特にメールは気を付けないと、文字化けして送ったメールが先方で読めなくなる。メールは、JIS コードで送る決まりになっているので気を付よう。これを避けるには、.emacs の中に、

```
(if (boundp 'MULE)
  (progn
     (define-program-coding-system nil ".*scan.*" (cons *junet* *junet*))
     (define-program-coding-system nil ".*inc.*" (cons *junet*  *junet*))))
(if (boundp 'NEMACS)
  (progn
     (define-program-kanji-code nil ".*scan.*" 2)
     (define-program-kanji-code nil ".*inc.*" 2)))
```

と定義しておく。または、

```
alias rmail /usr/local/bin/emacs -nw -f set-fileio-jis -f mh-rmail
alias smail /usr/local/bin/emacs -nw -f set-fileio-jis -f mh-smail
```

のように .cshrc に書いておいて、漢字コードを指定して emacs を立ち上げてもいいかもしれない。

デフォルトの漢字コードの設定は、システム管理者に相談し、自分の .emacs に書いておく。例えば、

```
;; Kanji codes --- 0: no conversion, 1: shift JIS, 2: JIS, 3: EUC
 (cond ((boundp 'NEMACS) (setq kanji-display-code 3))
           ((boundp 'MULE)
            (set-display-coding-system *euc-japan*)))
```

egg のその他のコマンドを挙げる。

アルファベット
: ローマ字を仮名に変換

‘SPC’
: 仮名漢字変換開始

‘C-a’
: フェンス内の先頭の文字へ移動

‘C-b’
: フェンス内で一文字分前へ

‘C-c’
: 入力を中止し，フェンス・モードから抜ける

‘C-d’
: 一文字削除

‘C-e’
: フェンス内の最後の文字へ移動

‘C-f’
: フェンス内で一文字分後ろへ

‘C-g’
: 入力をキャンセルし，フェンス・モードから抜ける

‘C-k’
: フェンス内のカーソルから後ろを削除

‘C-l’
: フェンス内の入力を確定し，フェンス・モードから抜ける

‘RET’
: フェンス内の入力を確定し，フェンス・モードから抜ける

‘C-T’
: フェンス内の文字の転置

‘C-^’
: コード入力、ギリシャ文字、記号のメニュー表示

‘M-h’
: フェンス内の文字をひらがなにする（コマンド *fence-hiragana*）

‘M-k’
: フェンス内の文字をカタカナにする（コマンド *fence-katakana*）

‘M->’
: フェンス内の文字を全角文字にする（コマンド *fence-zenkaku*）

‘M-<’
: フェンス内の文字を半角文字にする（コマンド *fence-hankaku*）

ユーザ辞書の管理

例えば，バッファ中の『文部省高エネルギー物理学研究所』という単語を辞書に登録する手順は次のようになる。

1. 登録文字列を指定する．登録する語の範囲をリージョンで指定し，‘M-X toroku-region’を実行する．

2. 読みを指定する．ミニバッファに

   辞書登録『文部省高エネルギー物理学研究所』 読み：

   が表示される．読みをローマ字で入力し（自動的にひらがなに変換する）リターンで読みの入力を終了する．

3. 登録する辞書を選ぶ。普通はただキャリッジ・リターン。

4. 品詞を指定する．ミニバッファに品詞一覧が表示される．

   品詞: 0. 普通名詞 / 1. 固有名詞 / 2. 動詞 / 3. 特殊な動詞 / 4. 動詞以外の用言 /

   ‘C-F’でカーソルが右に移動し，‘C-B’でカーソルが左に移動する．
   ‘C-N’で次の品詞一覧が表示される．
   ‘C-P’で前の品詞一覧に戻る．
   目的の品詞にカーソルを移動し，リターン・キーよって品詞を選択する．
   動詞にはいろいろな種類があるため，例えば「サ変（名詞型）語幹」を
   選択する
   には，まず「動詞」を選択し，次の選択リストから，「サ変（名詞型）
   語幹」を探し，選択する．
   例の場合は固有名詞なので，「固有名詞」を選ぶ．

3.1.7 その他

その他の機能については参考文献 [2] を見て欲しい。非常にたくさんの機能があって紹介しきれない（多分ほとんどは、一生使わないだろう）。ただし、日本語化の部分について書かれたもとしては、参考文献 [3]、[?] がある。これらはかな漢字変換に Canna とういソフトウエアを使う環境で書かれているが、共通に役に立つ部分も多い。 UNIX マガジンの連載、「Nemacs 入門」は、詳しく初心者向けに説明しているので、これが単行本になるのを待つか、 バックナンバーを集めるとよいだろう。付録に .emacs の例を付けておく。

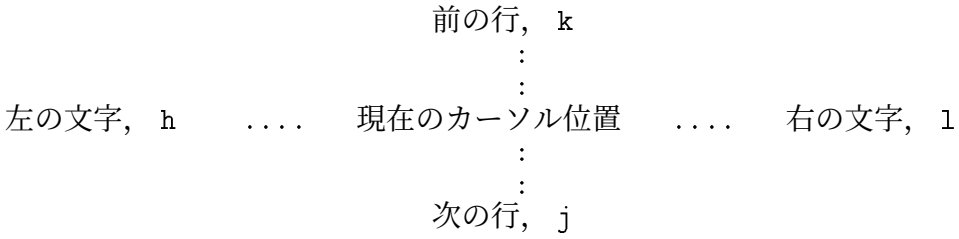
3.2 vi

vi[15] は標準で UNIX にバンドルされており、全ての UNIX 計算機で使えると思ってよい。(vee eye と発音する。)

emacs に比べ、立ち上がりが早い、メモリもあまり消費しないという利点がある。しかし、モードの切替えが存在するので、 emacs が好きな人は嫌う傾向がある。このモード切替えをいと

わない人は、 vi 使いになれるだろう。また、 emacs は、標準でバンドルされている場合は少なく自分でインストールしなくてはならない場合もあるので、 vi の最小限の機能位は覚えた方がよい。慣れてしまうと、非常に機能的にデザインされているので、どの他のエディターよりも効率的に編集が行える。

vi には、 emacs にはないモードという概念がある。コマンド・モードと入力モードの二つである。この切替えはエスケープ・キーで行ない、正しいモードが選択されていないと、思ったように動作しない。カーソルの移動の方向と、キーの定義は、次のようになる。



vi の終了の仕方は、

```
ESC :wq
```

または

```
ESC ZZ
```

でセーブして終了し、

```
ESC :q!
```

で、セーブせず強制終了する。コマンドモードから、入力モードに移行するには、 i コマンドを用いる。入力モードからコマンドモードに戻るには、 ESC キーを打つ。

巻末に vi のリファレンスカードが付いているので、他のコマンドはそれを参照のこと。

第 4 章

いろいろなソフトウェアの紹介

4.1 X ウィンドウ・システム

マサチューセッツ工科大学 (MIT) によって開発されたウィンドウ・システム [16, 17] であり、無料で配布されている。ネットワーク・ベースになっていて、自分のワークステーションだけではなく、別のワークステーションでプログラムを実行し、自分のワークステーションに表示させたりできる。多くの計算機に移植されおり、UNIX ワークステーションに限らず VMS 等の OS 上でも使用でき、UNIX ワークステーションの標準的なウィンドウ・システムとして使われている。X ウィンドウをベースにユーザ・インターフェースを改良したり、機能を付け加えたもの (OPEN-WINDOW、DCE WINDOW 等）が標準でついている場合が多い。サーバとして、それら OS に添付の物を使用し、クライアントは MIT 配布の物を用いて、機種に依存しない環境を作ることができる。いろいろなコンピュータ会社のものが交じっている状態では、機種に依存しない環境を覚えた方が良い。ここでは、MIT から配布されている X11R5 と X11R6 に基づいて説明する。

4.2 X ウィンドウの使い方

まず注意して欲しいのは、X ウィンドウを使うには特別な端末が必要だということである。**コンソールおよび X 端末以外から X ウィンドウを使おうとしてはならない。** X 端末では、普通はすでにサーバが立ち上がっているので、ログイン・ウィンドウが表示されていれば、そこにユーザ名とパスワードを入力してログインする。それ以外の場合は、telnet 等でログインする。コンソールの場合、ログイン・ウィンドウが出ていなければ、通常のようにログインして、xinit あるいは startx を実行するとサーバが立ち上がる。もしコマンドが見つからない場合は、PATH がきちんと設定されているか確かめる。デフォルトでは twm というウィンドウ・マネージャが起動される。ウィンドウ・マネージャが、キーボードやマウスからのイベントを読みとり、それぞれのウィンドウに対しいろいろな操作を可能にする。ウィンドウ・マネージャにはいろいろな種類があって操作性はみな異なる。また、twm もカスタマイズすることができるので、操作の仕方は一様ではない。twm 以外にも多くのウィンドウ・マネージャが入手可能である。twm のデフォルト設定では、ルート・ウィンドウ（背景）で、マウスの左ボタンを押すと、ウィンドウの操作に関するメニューが出る。この中の kill を選ぶと X ウィンドウが終了する。（左ボタンを押したままマウスを押し下げる。） 新しいウィンドウを起動したい時は、すでに開いているターミナル・

エミュレータ等のウィンドウで、 kterm あるいは xterm といったターミナル・エミュレータを起動すればよい。 twm をカスタマイズするには、ホーム・ディレクトリの.twmrc というファイルを変更する。色の指定や、操作の仕方を変更することができる。詳しい説明は省くが、 twm のオンライン・マニュアルを参照して自分の使いやすいように変更してみよう。もし自分で設定するのが面倒なら、誰かのをもらってそれに手を加えて見るのもよい。

X ウィンドウを立ち上げた時どのようなクライアントを起動するかは、.xinitrc および .xsession に書いておく。.xinitrc は、コンソールでサーバを立ち上げた時に用いられ、.xsession のは、xdm を使っている X 端末（ログイン用のウィンドウの開いている）からログインした時に自動的に実行される。 また .Xdefaults には、クライアントのリソースのデフォルトを書いておく。例えば xterm のスクロール領域を 100 行にしたいなら、

```
XTerm*saveLines: 100
```

という行を.Xdefaults に追加する。

システムによってはファイル・ブラウザが用意されていて、簡単にファイルの編集や実行ができるものもある。 SUN の標準的ウィンドウ・システムである OPEN ウィンドウには、 filemgr というファイル・ブラウザがついており、 X11 のサーバ上でも使用できる。このプログラムは、ウィンドウ・マネージャとして、 olwm を使用すること前提としているので、.xinitrc を書き替えて twm の代わりに olwm が立ち上がるようにしてから使用する。例えば、

```
/usr/openwin/bin/filemgr -geometry 80x20+0+15 &
/usr/openwin/bin/olwm -3d &
```

のように書いておく。また、ヒューレット・パッカードやソニーなど Motif を標準として採用している計算機にもそれぞれのファイル・ブラウザが用意されている。それぞれの使用法については、マニュアルを参照して欲しい。

ここで、サーバとクライアントという言葉について簡単に説明する。 X の場合のサーバとは、ワークステーション本体や、 X 端末で実行されている描画やいろいろな入力を処理するためのプログラムをいう。 xinit か startx で立ち上げるのがサーバである。これに対しクライアントは、xterm など実際にその上で働いているプログラムである。サーバは、クライアントの要求に応じて新しいウインドウを開いたり、変形したり、表示を行ったり、キーボードからの入力をクライアントに受渡したりする。クライアントは、サーバにいろいろな要求を送ってプログラムを実行する。

4.3 ネットワーク経由の X ウィンドウ・クライアントの利用

UNIX は、プログラム同士がネットワークを通じて情報交換することも他の OS 比べて簡単にできるようになっており、それを利用しているものの 1 つが X ウィンドウである。 X ウィンドウ のクライアントを別のワークステーションで動かし、自分のワークステーションで見ることもできる。自分のワークステーション (mymachine) で、

```
xhost  +aaaa
```

を実行し、クライアントのあるワークステーション (aaaa) で、（csh の時）

```
setenv DISPLAY mymachine:0
```

を実行してからプログラムを実行する。名前の代わりに IP アドレスでもよい。例えば、 IP アドレスが 130.87.1.1 の X 端末で、ワークステーション amyux1 上のクライアント xdvi を実行したいときは、 X のサーバに接続の許可を設定する。

```
xhost  +amyux1
```

次に、

```
setenv DISPLAY 130.87.1.1:0
```

とした後、 xdvi を実行する。

4.4 ターミナル・エミュレータ

xterm や kterm(xterm の日本語版) がよく使われている。 vt102 端末と TEKTRO 4010 をエミュレーションする。もし、字の大きさが小さすぎるとか大きすぎると思ったら、 xterm/kterm のウィンドウにマウスをスポットし、コントロール・キーとマウスの一番右のボタンを一緒に押し続けてみる。フォントのメニューが表示され、別の大きさのフォントを選ぶことができる。デフォルト以外のフォントが表示されなかったり、選択しても大きさが変わらない時は、 xterm/kterm のインストールの不備か、フォントがないことが考えられるので、システム管理者に相談してみる。

4.5 X のクライアント

X 用の多くのクライアントが開発されており、その多くは無償で入手できる。 X ウィンドウ・システムの配布テープには多くのクライアントが含まれているが、それ意外にも様々なものがネットワーク等で入手可能である。 X11R5 のオンライン・マニュアルにあるものを紹介する。但し、使用しているワークステーションに、全てがインストールされているとは限らない。また、 PEX (Phigs Extension for X) に対応していないサーバでは実行できないものもある。

appres (1)	list application resource database
auto_box (1)	display a rotating cube using PHIGS
bdftopcf (1)	convert font from Bitmap Distribution Format to Portable Compiled Format
beach_ball (1)	display a bouncing sphere using PHIGS
bitmap, bmtoa, atobm (1)	bitmap editor and converter utilities for the X Window System
display (1)	display an image on any workstation running X
editres (1)	a dynamic resource editor for X Toolkit applications
ico (1)	animate an icosahedron or other polyhedron
idraw (1)	drawing editor
imake (1)	C preprocessor interface to the make utility

import (1)	dump an image of an X window
kbd_mode (1)	recover the Sun console keyboard
kinput2 (1)	kanji input front end processor for X11
kterm (1)	kanji terminal emulator for X
listres (1)	list resources in widgets
maze (1)	an automated X11 demo repeatedly creating and solvinga random maze
oclock (1)	display time of day
plbpex (1)	picture level benchmark program
puzzle (1)	15 puzzle game for X
showrgb (1)	uncompile an rgb color name database
tvtwm (1)	Tom's Virtual Tab Window Manager for the X Window System
twm (1)	Tab Window Manager for the X Window System
viewres (1)	graphical class browser for Xt
xauth (1)	X authority file utility
xbiff (1)	mailbox flag for X
xboard (1)	Xt/Athena user interface for GNU Chess, version 1.2
xcalc (1)	scientific calculator for X
xcalendar (1)	calendar with a notebook for X11
xclipboard (1)	X clipboard client
xclock (1)	analog / digital clock for X
xco (1)	display X11 color names and colors
xconsole (1)	monitor system console messages
xcutsel (1)	interchange between cut buffer and selection
xditview (1)	display ditroff output
xdpr (1)	dump an X window directly to a printer
xdpyinfo (1)	display information utility for X
xdu (1)	display the output of "du" in an X window
xdvi (1)	DVI Previewer for the X Window System
xedit (1)	simple text editor for X
xev (1)	print contents of X events
xeyes (1)	a follow the mouse X demo
xfd (1)	display all the characters in an X font
xfed (1)	font editor for X window systems bdf fontfiles
xfig (1)	Facility for Interactive Generation of figures under X11
xfontsel (1)	point click interface for selecting X11 font names
xforecast (1)	X window system interface to the weather program
xgas (1)	animated simulation of an ideal gas
xgc (1)	X graphics demo
xhost (1)	server access control program for X
xlbiff (1)	mailbox message previewer for X
xload (1)	system load average display for X

xlock (1)	Locks the local X display until a password is entered
xlogo (1)	X Window System logo
xlsatoms (1)	list interned atoms defined on server
xlsclients (1)	list client applications running on a display
xlsfonts (1)	server font list displayer for X
xmag (1)	magnify parts of the screen
xman (1)	Manual page display program for the X Window System
xmessage (1)	display a message or query in a window (X based /bin/echo)
xmh (1)	send and read mail with an X interface to MH
xmkmf (1)	create a Makefile from an Imakefile
xmodmap (1)	utility for modifying keymaps in X
xphoon (1)	displays the PHase of the mOON on the root window
xpostit (1)	X window system Postit notes
xpr (1)	print an X window dump
xrdb (1)	X server resource database utility
xrefresh (1)	refresh all or part of an X screen
xrlogin (1)	start an xterm that uses rlogin or telnet to connect to a remote host
xrsh (1)	start an X program on a remote machine
xscope (1)	X Window System Protocol Monitor
xset (1)	user preference utility for X
xsetroot (1)	root window parameter setting utility for X
xstdcmap (1)	X standard colormap utility
xtalk (1)	talk to another user
xterm (1)	terminal emulator for X
xtetris (1)	X Window block dropping game
xwd (1)	dump an image of an X window
xwininfo (1)	window information utility for X
xwnmo (1)	Input Manager of the X Window System Version 11
xwud (1)	image displayer for X

例として、このクライアントを用いた画面のハードコピーの取り方を紹介する。いろいろな方法があるが、一例としては、

```
xwd -nobdrs| xpr -device ps | lpr -Pps &
```

という方法がある。 xwd で画面のダンプをとり、それをポストスクリプトに変換してプリンタに送る。このコマンドを打つと、カーソルの形が変わる。印刷したいウィンドウをマウスの左ボタンでクリックする。もちろんプリンタの名前は適当なものに変えて実行しなければならない。

4.6 LaTeX の使い方

D.Knuth (TeX の作者) による TeX のマニュアルが、 Addison Weseley 社 から出版されている (ISDN 0-201-13448-9) 。この翻訳は ASCII 出版から出ている。 L.Lamport(LaTeX の作者) に

よる LaTeX のマニュアルとその翻訳も同じ会社から出ている。 このほかにもいろいろな例題集が出版されている。

TeX(テックと発音する人が多い) とは、高品質の組版、特に数学のテキストを組むために設計されたプログラムである。 Knuth は数学者であるが、印刷会社から戻ってくる論文に関して、数式の印刷精度が低いのに我慢できず TeX をデザインした。 TeX は自由度が高く高機能であるが、マクロを作ったりするのはやさしくないので、 LaTeX を使うことを勧める。あるいは他人の書いた高機能のマクロがあればそれを使った方がよい。もともとは英語用であるが、いろいろな言語に移植されており、もちろん日本語もある。 LaTeX(ラテック、ラテフ) では、文書整形のためのコマンドよりもテキストの構造に神経を集中させることで、ユーザが組版にかける手間を省けるように一群のコマンドを TeX に追加した。ほとんどの UNIX ワークステーション上で TeX と LaTeX が使えるようになっている。日本語版には ASCII 版と NTT 版がある。ちなみにこのマニュアルは NTT 版日本語 LaTeX によって書かれている。両版には細かな違いがあるので注意が必要である。

簡単な例

簡単な文を日本語 LaTeX で書いてみる。

まず最初にエディタを使って入力ファイルを編集する。 (emacs での日本語の入力については別の章を見よ。) 他の file と区別が付くように、入力ファイル名は test.tex のように拡張子を.tex にする。

LaTeX にはスタイル・ファイルというものがあり、書こうとしている文章に合わせて選ぶ。例えば、この文書のように長いものには book か report、短いものには article というスタイルが適している。また、手紙のためのスタイルもある。

これは xxx.tex というファイルに日本語 LaTeX で文章を書いた例である。

% **cat xxx.tex**

```
\documentstyle{j-report} %% どういうスタイルで書くか。
\title{てすと}           %% 題名
\author{すけきよ}        %% 著者
\begin{document}         %% ここから文章が始まる。
\maketitle               %% 題名のページの出力
\chapter{始め}           %%chapter の題名
\section{始まるよ}
Hello world !
今日は。今日は良い天気です。
\section{セクション２}
セクション２
\chapter{次の章}         %%chapter の題名
\section{セクション壱}
\end{document}           %% 文章の終り。
```

これを jlatex コマンドでコンパイルする。

```
%jlatex xxxx
This is JTeX, Version 1.06, based on TeX C Version 2.95 (no format
```

```
preloaded)
(xxx.tex
** JLaTeX ver.1.2......Isozaki
(/usr/local/lib/jtex/inputs/j-report.sty
Document Style ‘j-report’ <26 Apr 88>.
(/usr/local/lib/jtex/inputs/j-rep10.sty)
(/usr/local/lib/jtex/inputs/j-titlepage.sty))
No file xxx.aux.
[0]
第 \lower 0.1ex\hbox {1} 章.
[1]
第 \lower 0.1ex\hbox {2} 章.
[2] (xxx.aux)
Output written on xxx.dvi (3 pages, 1896 bytes).
Transcript written on xxx.log.
```

すると、 xxx.dvi という file ができる。これは device-independent ファイルと呼ばれ、出力機器に依存しない。これを別のコマンドを使い、それぞれの機器用のファイルを作る。もしコンパイル時に、エラーが出たら e と入力してその個所を修正する。エラーが出なくなったら、印刷するかプレヴューする。 X ウィンドウか OPEN ウィンドウでプレヴューするには xdvi コマンド、印刷するには dvi2ps と lpr コマンドを使う。また、クロス・リファレンスを使った時には、 LaTeX コマンドを二度実行しなければならないことに注意する。

```
%xdvi xxx
```

or

```
%dvi2ps xxx|lpr -Pps
```

とする。 dvi2ps の出力はポストスクリプトであるからポストスクリプト・プリンタに出力されなければならない。

TeX や LaTeX プログラム本体はもちろん英文のフォントはすべて無料配布されている。ただし、大日本印刷の日本語フォントだけは有料で、 1 プリンタあたり約 10 万円を支払わなければならない。 プリンタ内臓のフォントを使用すれば、このフォントは必要ない。

PAW 等で作った EPSF (Encapsulated PostScript File) で書かれた絵を取り込むこともできる。このマニュアルに載っている挿絵は EPSF を用いて取り込まれたものである。ポストスクリプト・ファイルによる絵の取り込み方はいくつもあるが、このマニュアルを作成するために用いたのは epsf.sty と dvi2psj の組合せである。このほかにも、 psfig と dvips を組み合わせる方法がある。ポストスクリプト・ファイルを取り込むためのマクロと dvi ファイルをポストスクリプトに変換するプログラムは、対応したものを使わないと正常に動作しない。 PAW など EPSF をサポートしているプログラムの出力は簡単に取り込むことができ、糊とハサミで切り貼りする必要がない。このマニュアルでは、次のようにして絵を取り込んでいる。

```
\documentstyle{jreport}  %% どういう style で書くか。
\begin{document}         %% ここから文章が始まる。
文章
\leavevmode
\epsfxsize=5.cm
```

```
\epsfysize=15cm
\epsfbox{test_tex.eps}

文章
\end{document}              %% ここで文章が終る。
```

4.7 emacs の TEX、LATEX モード

emacs の TEX、LATEX モードを使うと編集が楽にできる。また、デフォルトで付いてくるものを改良した AUC 版もある。これは anubis.ac.hmc.edu から anonymous ftp により入手できる。/usr/local/emacs/lisp に auc-tex.el がなければ、便利なので是非入手を勧める。

また、LATEX や TEX のコマンドを無視してくれるスペリング・チェッカーもある。M-x ispell-buffer でバッファ全部、M-x ispell-word で単語のスペリングのチェックができる。もし、ispell がインストールされていないときは、M-x spell-buffer や、M-x spell-work を使用する（これらは、標準であるはず）。ispell の方がユーザ・インターフェースも優れており、辞書も大きいのでお勧めである。

4.8 その他のツール

標準に UNIX にバンドルされているわけではないが、多くのサイトで使えるソフトウェアを紹介する。これらは全てフリーウェアであり、ネットワーク等により無料配布されている。これらはユーザ各自がインストールするものなので、サイトによっては、インストールされていなかったり、置いてあるディレクトリが異なる可能性があるので注意してほしい。これ以外にも非常に多くのものが入手可能である。

4.9 GNUPLOT

関数やデータを 2D 及び 3D でグラフィック表示するためのソフトウエアである。いろいろな出力デバイスをサポートしている。使い方は簡単で、丁寧なオンライン・ヘルプもある。デバイスとして、xfig というグラフィック・エディタ用の出力フォーマットを選べば、書いたグラフにコメントを書いたり、好きな場所にタイトル等を書くことができる。

```
%gnuplot
```

と打つと立ち上がる。例えば 3 次関数のグラフを書きたいときは、

```
gnuplot> plot x**3
```

でよい。plot の後に関数名や領域の指定ができる。非常に高い機能を備えており、いろいろなグラフが簡単に描ける。TEX で書かれた詳細なマニュアルや多彩な例題が沢山付いているので、システム管理者にマニュアルについて問い合わせ、試してみよう。

4.10 idraw

スタンフォード大学と シリコングラフィックス社により開発されている interviews という一連のソフトウエア・パッケージの一部である。C++ で記述されおり、X11 と共に無料配布されている。MacDraw のようなグラフィック・エディタだと思えばよい。日本語の扱える国際化版もある。

4.11 その他

その他の有名なフリー・ソフトウエアを挙げておく。

CAP (8)	Columbia AppleTalk Package for Unix
XGKS (3)	A GKS library for the X Window System
ansir (1)	read ANSI standard labelled tape
ansiw (1)	write ANSI standard labelled tape
bash (1)	GNU Bourne-Again SHell
bibtex (1)	make a bibliography for TeX
bison (1)	GNU Project parser generator (yacc replacement)
cptp (1)	copy from tape to tape
cvs (L)	Concurrent Versions System
dvi2ps (1)	convert a DVI file to PostScript
dvi2tty (Local)	preview a dvi file on an ordinary ascii terminal
dviconcat (1)	concatenate DVI files
dvipage (1)	display DVI files from TeX and LaTeX.
dvips (1)	convert a TeX DVI file to PostScript (PostScript is a trademark of Adobe Systems, Inc.)
dviselect (1)	extract pages from DVI files
dvitype (1)	translate a dvi file for humans
emacs (1)	GNU project Emacs
fig (1)	Facility for Interactive Generation of figures
flex (1)	fast lexical analyzer generator
floppy (1)	Fortran coding convention checker and code tidier
fortc (1)	Unidata utility to generate C code with fortran-callable functions
gcc (1)	GNU project C Compiler
geqn (1)	groff equation formatter
gnuplot (1)	an interactive plotting program
gpic (1)	compile pictures for troff or TeX
groff (1)	front end for the groff document formatting system
gtbl (1)	table formatter for groff
gtroff (1)	format documents
ident (1)	identify files

idraw (1)	drawing editor
inc (1)	incorporate new mail
inc (1)	incorporate new mail
ispell (local)	Correct spelling for a file
kbanner (1)	display kanji files in large letters
latex (1)	intensional text formatting and typesetting
lwf (1)	ASCII to PostScript filter
mag (1)	generalized magnetic tape
gmake (1L)	GNU make utility to maintain groups of programs
mf, inimf (1)	Metafont, a language for alphabet design
mh (1)	Message Handler
mkmf (1)	makefile editor
msh (1)	MH shell (and BBoard reader)
nkf (1L)	Network Kanji code conversion Filter
patch (1)	a program for applying a diff file to an original
perl (1)	Practical Extraction and Report Language
phone (1)	communicate with other users in real-time
psbb (1)	extract bounding box from PostScript document
rcs (1)	change RCS file attributes
shar (1net)	create file storage archive for extraction by /bin/sh
slitex (1)	make LaTeX slides
tcsh (1)	C shell with file name completion and command line editing
tex, initex, virtex (1)	text formatting and typesetting
top (1)	display and update information about the top cpu processes
transfig (1)	creates a makefile for portable LaTeX figures
xtacho (1)	load average tachometer for X
xtiff (1)	view a TIFF file in an X window
xvttool ()	An X program that provides VT100 or tn3270 emulation with function keys
xwd2ps (l)	convert X11 window dumps to PostScript

ポストスクリプト・ファイルのプレヴューができる gs 等は大変便利なツールである。これらのコマンドを使う前に man コマンドで確認してから実行すること。使っているウィンドウ・システムの環境によっては実行できないものもある。このほかにも数多くのコマンドがある。

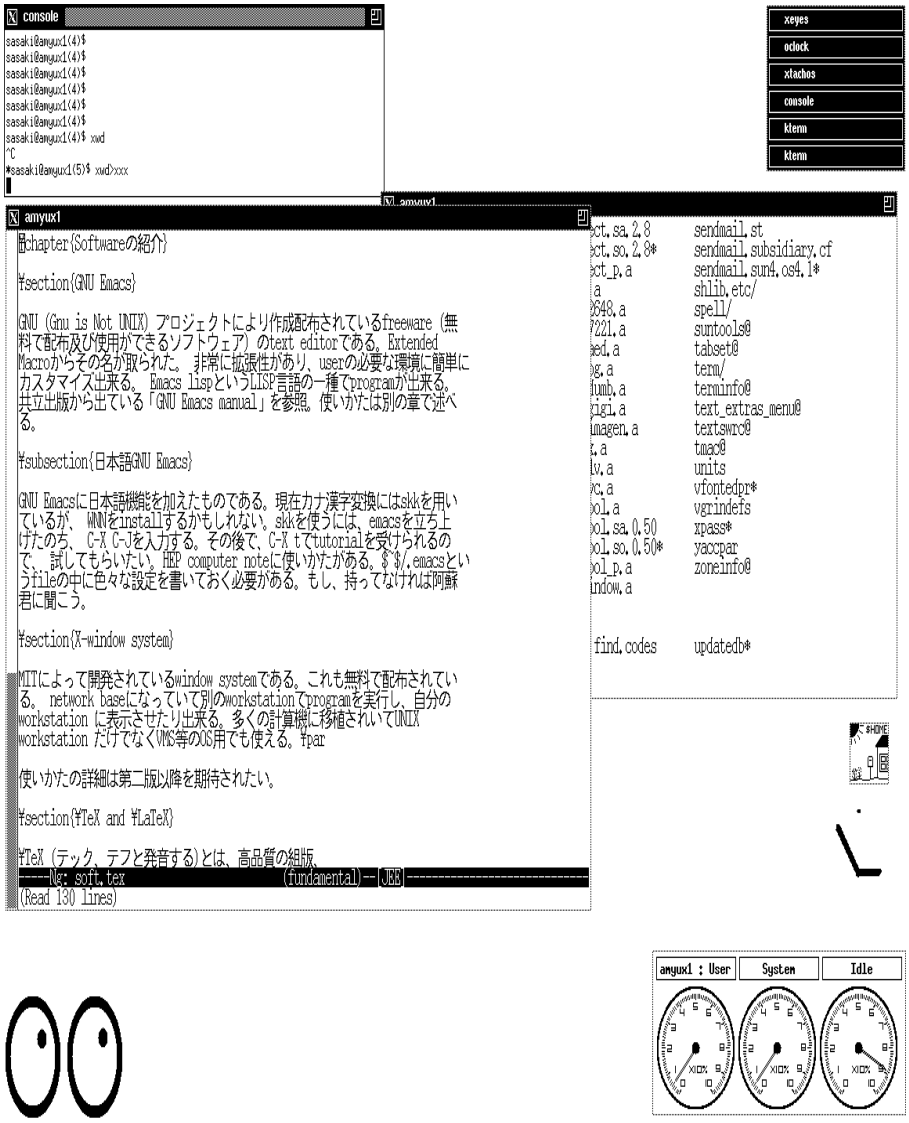


図 4.1: X ウィンドウを使っている画面の例。

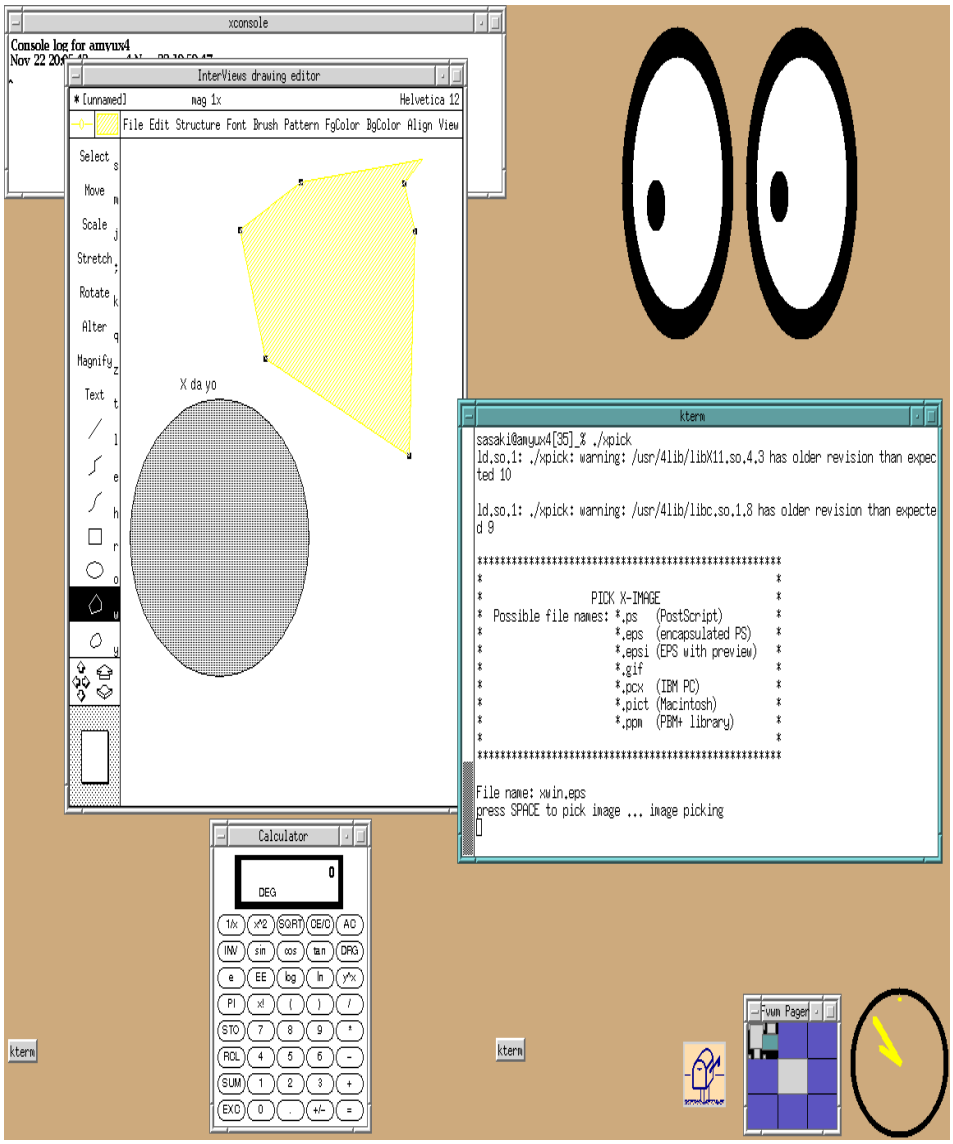


図 4.2: X ウィンドウを使っている画面の例。

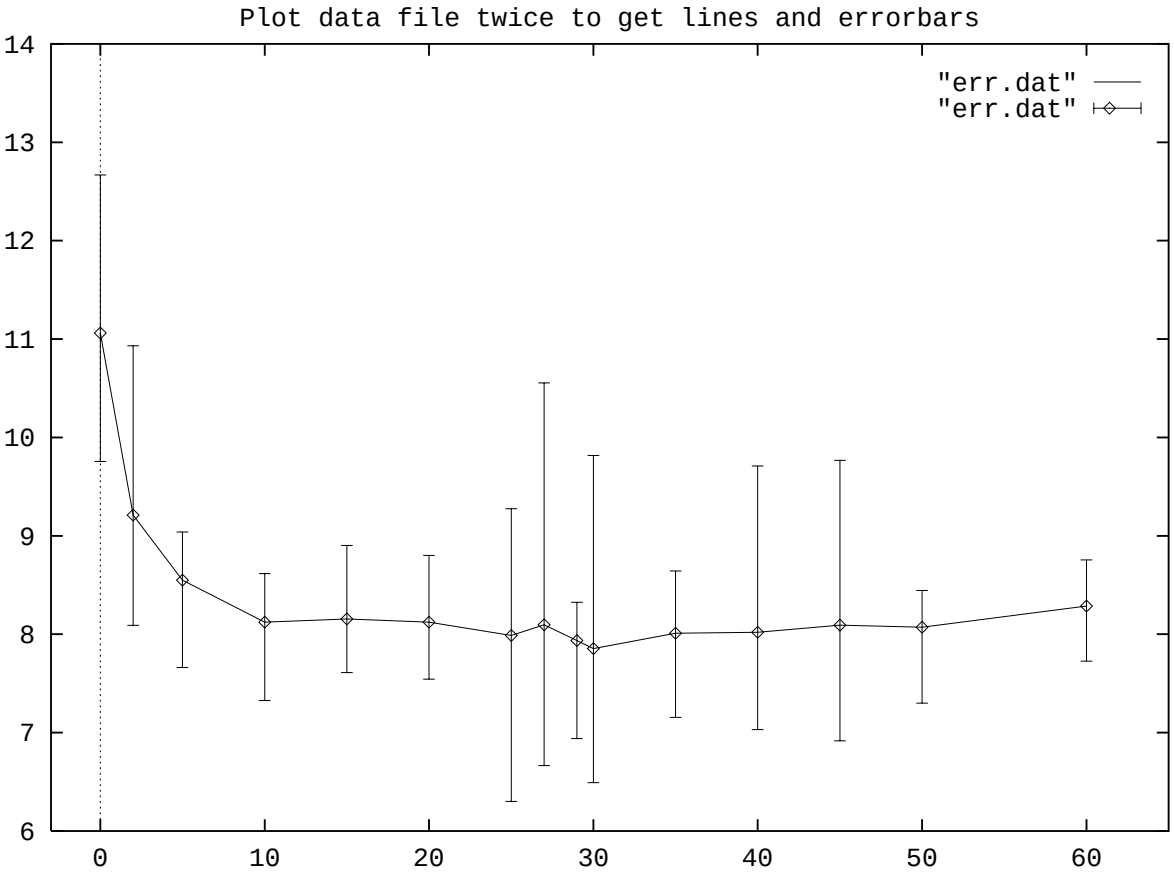

Plot data file twice to get lines and errorbars
"err.dat"
"err.dat"
14
13
12
11
10
9
8
7
6
0
10
20
30
40
50
60

図 4.3: GNUPLOT

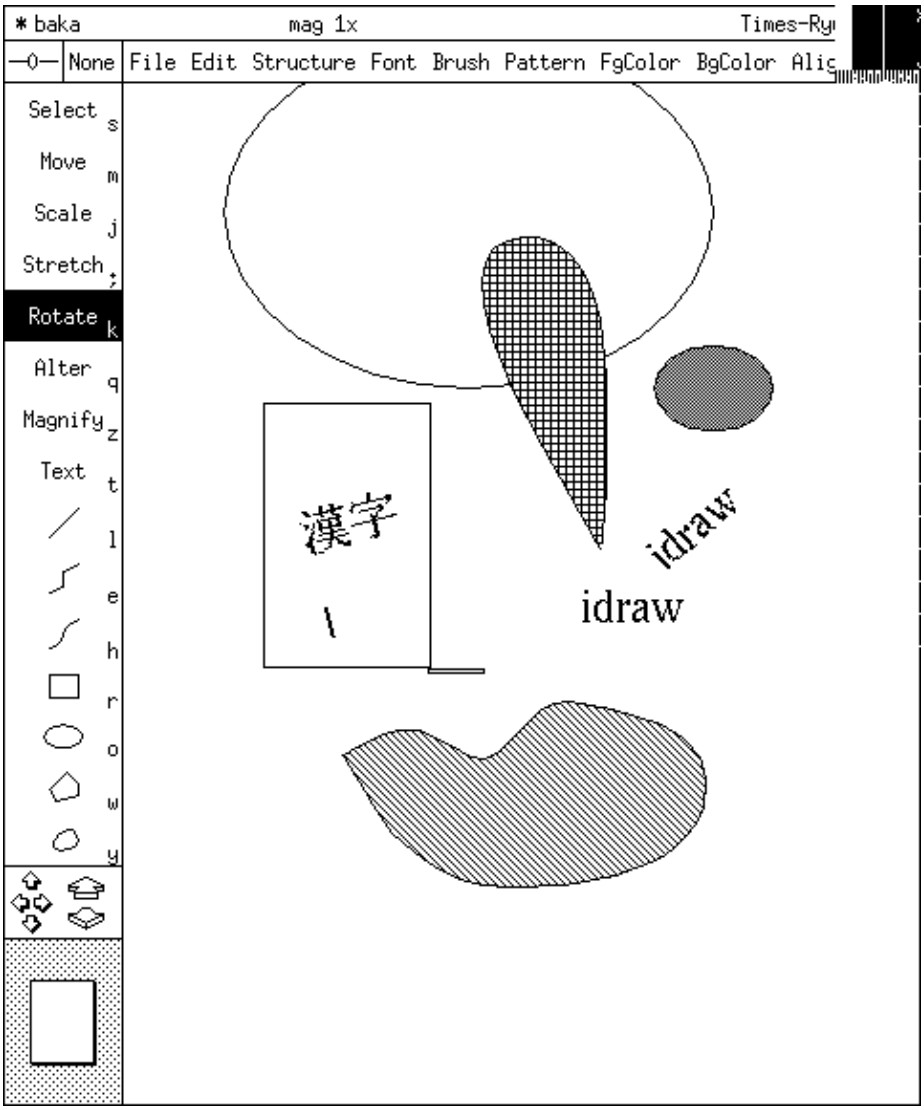

* baka
mag 1x
Times-Ry
None
File Edit Structure Font Brush Pattern FgColor BgColor Alig
Select s
Move m
Scale j
Stretch ;
Rotate k
Alter q
Magnify z
Text t
l
e
h
r
o
w
y
漢字
idraw
idraw

図 4.4: idraw

第 5 章

FORTRAN によるプログラミング

5.1 コーディング

emacs や vi 等、自分の好きなエディタを使って FORTRAN の文法に従い入力する。この時ソース・コードの入っているファイルには、普通 '.f' という拡張子を付けて作成する。これを大文字にすると多くのシステムでは、 cpp という C 言語のプリプロセッサを通してからコンパイルされる（システムによっては、明示的にオプションを指定しないとならない。）。大抵の UNIX の FORTRAN コンパイラは、いわゆる VMS FORTRAN 互換機能を取り入れていることが多い。変数名や関数名の 6 文字制限等が廃止されている。大型計算機などとソフトウエアのやり取りをする場合には注意が必要である。

また、入力の際のタブの扱いには気をつける。処理系によっては、タブが行頭にあると 6 カラムの空白と同等に扱うものもある。

5.2 コンパイルとリンク

FORTRAN ソース・コードをコンパイルしてリンクするには、

```
%f77  foo.f
```

とすればよい。[1]もし、リンクせずコンパイルだけの時は、

```
%f77 -c  foo.f
```

とする。 foo.o というオブジェクト・ファイルが生成される。もし、ライブラリをリンクしたい時は、

```
%f77 foo.f -lfxgks -lxgks -lX11 -L/usr/local/lib
```

等とする。 –l はライブラリの名前 –L はライブラリのあるディレクトリを示す。 /usr/lib は自動的に捜すので –L で指定する必要はない。この場合実際には、 /usr/local/lib/libfxgks.a 、 /usr/local/l
そして、 /usr/local/lib/libX11.a というライブラリがリンクされる。

例として、よく使う HBOOK をリンクするコマンドは、

[1]FORTRAN コンパイラを起動するコマンド名やオプションの与え方はマシンにより異なるので、それぞれのマシンのマニュアルを調べること。 fort77 や、 xlf 等機種により異なる。

```
%f77 foo.f -lpacklib -L/kek/cern/94a/lib
```

のように書かれる。

このほか多くのオプションが f77 コマンドには用意されていて、それらはマシンに依存する。コンパイルを実行しようとするマシンのマニュアルを一読することを勧める。どれでも共通のものとしては、デバッグ・オプションの −g やオプティマイズ・フラグの −O、a.out を別の名前に変えるのに使う −o <file> 等がある。オプティマイズにレベルがあるときは、−O の後ろに数字を書いて指定する。

5.3 実行の仕方

UNIX では実行可能ビットの立ったファイル名を打つと、それが実行される。もし、なにもオプションを付けずにコンパイルとリンクを行なうと、a.out というファイルができるので、

```
%a.out
```

と打てばよい。ユニット 5 にファイルから入力させ、ユニット 6 の出力をファイルに書きたい時は、

```
%a.out <input>output
```

とする。もし、a.out 以外の名前にリンカの出力を書きたいときは、

```
%f77 -o test x.f
```

と、−o の後に名前を書く。

ユニット N にファイル名を指定せず WRITE すると、SUN FORTRAN の場合 fort.N というファイルができるが、これを実際のファイル名にしたい時には、シンボリック・リンクを用いると便利である。例えば、

```
%ln -s test.out fort.2
```

のようにしておき、プログラムでユニット 2 に書くと、test.out にその内容が書かれる。入力に対しても同様である。（HP FORTRAN の場合には、fort.22 等の代わりに ftn22 になる。機種によって異なるので、それぞれの機種用のマニュアルを参照すること。）

5.4 ライブラリの使い方

複数のオブジェクト・ファイルをリンクする場合、ライブラリを作ると便利である。例えば、libtest.a というライブラリを作りたい時は、（ライブラリの拡張子は、.a である。）

```
% ar r libtest.a a.o b.o c.o
```

のようにする。モジュールを入れ換えたい時は、

```
% ar r libtest.a a.o b.o c.o
```

とする。BSD 系オペレーティング・システムの場合、この後忘れずに ranlib コマンドを使って、リンカが認識できるように変換しなければならない。

```
%ranlib libtest.a
```

ar コマンドのオプションは、システムによって異なることもある。
　このようにして作ったライブラリは、

```
%f77 x.f a.o b.o c.o
```

とする代わりに、

```
%f77 x.f libtest.a
```

や、L オプションの後ろにライブラリのあるパスを書いて、

```
%f77 x.f -ltest -L/usr/local/lib
```

のようにして使用する。

5.5　プログラムのデバッグの仕方

　まず他人を疑う前に自分を疑うことが必要である。多くの UNIX の FORTRAN コンパイラにはバグが多くあることが知られているが、初歩的なレベルでの明らかなものは少ない。始めに自分の間違いではないかよく調べる。

5.5.1　エラーの対処方法

　よく見かけるエラーとその対処方法について述べる。UNIX のエラー・メッセージは不親切でよく分からないことが多い。結局いつも同じメッセージが出ることも多い。

引数のミスマッチ

　サブルーチンや関数など副プログラムの仮引数とそれを呼ぶ側の引数の数の不一致によって起きる。

```
      PROGRAM MAIN

      CALL SUB0(A,B,C)

      END

      SUBROUTINE SUB0(A,B,C,D)

      END
```

D を省略すること自体はエラーにならないが、サブルーチン SUB0 の内部で、D を参照しようとすると、D は実際のアドレスが割り当てられていないので、エラーを起こす (FORTRAN はアドレスで引数を渡すので。)。 不要な引数を削除するか、D をきちんと渡すようにする。
バグの例。

```
      PROGRAM MAIN0
      CALL SUB1(A,B,C)
      END
      SUBROUTINE SUB1(A,B,C,D)
      A=1
      B=2
      C=4
      D=5                          <== ここで core dump
      RETURN
      END
```

こういうプログラムを実行すると、次のようなメッセージがでる。

```
*** Segmentation Violation = signal 11 code 3

segmentation fault.    core dumped

Bus error.
```

いい加減なメモリ管理

宣言したよりも大きな配列の添字を参照した時や、実際にはゼロから配列が宣言されていないのに、添字がゼロになった時にエラーになる。

```
      PROGRAM MAIN
      REAL DIM(10)

      R = DIM(IN)          <=== IN=0

      END
```

このように配列の添字が変数のとき、何らかの理由 (バグ) で引数がゼロになるとエラーを起こすので配列の引数を確認する。

こういうプログラムを実行すると、次のようなメッセージがでる。

```
*** Segmentation Violation = signal 11 code 3

segmentation fault.    core dumped
```

浮動小数点の演算例外

SUN FORTRAN の場合は次のようなメッセージがでる。

```
 Warning: the following IEEE floating-point arithmetic exceptions
          occurred in this program and were never cleared:
 Division by Zero;
```

Warning と書いてあるが、これは結果にかなり重大な影響を及ぼすこともある。

```
      INTEGER N, IEEE_HANDLER, SIGFPE_ABORT

      N = IEEE_HANDLER( 'SET', 'COMMON', SIGFPE_ABORT )
```

という行をプログラムに入れてコアダンプさせ、プログラムのどこでエラーが起きているか探す。他の FORTRAN コンパイラについては、マニュアルを参照してほしい。

引数のタイプのミスマッチ

VAX 等他の計算機で使っていたプログラムを移植するときに問題になることが多い。この厄介な点は一見正常に動いているのだが、結果が合わないということである。IEEE 浮動小数点を使っている計算機（多くの UNIX ワークステーション）の場合、単精度の仮引数に倍精度の実引数を渡したり、その逆をしてはならない。VAX や IBM 互換汎用大型計算機などでは問題にならないが（精度が落ちるだけ）、IEEE 浮動小数点を使っている計算機の場合には、正しい値が渡されない。一見正常に動いているのに値がオカシイ場合はこの原因が考えられる。

```
% cat exampl.f
      PROGRAM ARGMCH
      REAL AA
      AA = 3.0
      PRINT *,AA
      CALL  SUBR(AA)
      END
      SUBROUTINE SUBR(CC)
      REAL*8 CC
      PRINT *,CC
      END
% f77 exampl.f
% a.out
    3.00000
    32.000000000000
```

5.5.2 デバッガ (dbx) の使い方

dbx デバッガは一般的なマシンのほとんどで使えるが、ヒューレット・パッカード社のように xdb というコマンドを代わりに用意している場合もある。マシンによっては、標準ではない場合もある。

怪しそうなプログラムを全て −g オプション付きでコンパイルし、リンクしてから、

```
dbx a.out
```

または、a.out を実行可能プログラムの名前に変えて dbx を実行する。

```
% f77 -g bug1.f                                  <== コンパイルとリンク
bug1.f:
 MAIN main0:
        sub1:
```

```
% dbx a.out                                    <== デバッガの起動
Reading symbolic information...
Read 275 symbols
(dbx) run                                      <== プログラムの実行
Running: a.outsignal SEGV (no mapping at the fault address) in sub1 at
line 7 in file ‘‘bug1.f                        <==7 行目でエラー
    7           C=D
(dbx) where                                    <== エラーがどこででかたか？
sub1(a = 1.0, b = 2.0, c = 0.0, d = bad data address=0.0), line 7 in
‘‘bug1.f’’
MAIN(), line 2 in ‘‘bug1.f’’
main() at 0xf7780db4
(dbx) list 1,10                                <=1 行目から 10 行目のリストを表示
    1           PROGRAM MAIN0
    2           CALL SUB1(A,B,C)
    3           END
    4           SUBROUTINE SUB1(A,B,C,D)
    5           A=1
    6           B=2
    7           C=D                            <= これがエラーの出た行だ！
    8           D=5
    9           RETURN
   10           END
(dbx) print a                                  <= 変数 A の値のチェック
(dbx) quit                                     <== デバッガの終了
```

このようにして原因を探し出しプログラムを修正する。この他にも、デバッグする関数またはサブルーチンを指定するための func コマンド、サブルーチンの call tree の間を移動するための up, down コマンドを知っていると便利である。

デバッガのコマンドは他にも多くあるが、 run、 where、 list の最低三つ知っていれば一応は使える。詳しくはマニュアル [5] を読むこと。

GNU のデバッガ gdb はさらに高い機能を持っている。これも多くのサイトで利用可能である。多くの機種でサポートされているので、こちらを覚えると別のベンダーのマシンでも仕事がしやすい。しかし、 FORTRAN プログラムのデバッグに用いる場合不満な点も多い。

5.6 実行効率の良いプログラムを書くために

UNIX ワークステーションは、仮想記憶を備えているので、一見無限に記憶を使えるような錯覚に陥り、大きな配列をたくさん作ったりしがちである。しかし、仮想記憶は、実際にはディスクを使って実現されているので、大きな記憶容量を必要とするプログラムの実行は遅くなってしまう。出来るだけ余分な配列を作るのを避け、コンパクトなプログラムを作るように心がける。次のようなことに気をつけると、実行効率を落とさずにすむ。

- 余分に大きな配列を作らない。
- 配列を作るときは、左の添え字ほど大きくなるようにする。

- DO LOOP の入れ子内で多次元の配列を使うときは、配列の左側の添え字に関する LOOP が入れ子の内側にくるようにする。
- 必要のあるとき以外倍精度の実数の計算をしない。
- 構造化プログラミングを心がける。人間に取って理解しにくいプログラムは、コンパイラに取っても理解しにくいので最適化されにくい。
- 計算型 GO TO 文等、 FORTRAN77 で廃止された機能を使用しない。

以上は、ほんの一例である。気をつけてプログラムを書くことで、時には実行時間が 10 倍も変わる事がある。早い計算機を買う以上の効果があることになるので、効率良いプログラムを書く努力をするべきである。

第 6 章

make コマンド

6.1 イントロダクション

make とは、プログラムのコンパイル、リンク等のバッチ処理を行うためのコマンドである。独自の文法に従って書かれたファイルからコマンドを読み込み実行する。その際、いろいろなルールに従って処理が自動的に行われる。

make コマンドほど、 UNIX の思想をよく表しているコマンドはないと思われるほど、 UNIX と make は密接な関係にある。 make を使って非常に多くのことができるが、ここではその簡単な使い方について説明する。機種によって仕様が異なるので、 FSF (Free Software Fundation) の GNU-make を取り上げる。これは自由に配布が許されているフリー・ソフトウエアである。

6.2 GNU make のインストール

最近の GNU のツールは gzip というコマンドで圧縮された状態で配布されているので、始めに gzip コマンドのインストールが必要である。 ftp.kek.jp から anonymous ftp でこのコマンドを入手しインストールする。

```
% ftp ftp.kek.jp
Connected to ftp.kek.jp.
220 ftp FTP server (Version 4.1 Sun Mar 25 22:59:11 EST 1990) ready.
Name (ftp.kek.jp:sasaki): anonymous
331 Guest login ok, send ident as password.
Password: (自分のメールアドレスを入力する。エコーされない。)
230 Guest login ok, access restrictions apply.
ftp> cd pub/GNU
250 CWD command successful.
ftp> bin
ftp> get gzip-1.2.4.tar
ftp> quit
```

新しい版が出るとファイル名が変わるので、もしこの名前でなければ ls コマンドで調べて最新のものを入手する。このファイルを展開し、ソース・コードを取り出す。

```
% tar xvf gzip-1.2.4.tar
```

次に、そのディレクトリに移り、 configure スクリプトを実行する。この際コマンドをインストールするディレクトリを指定する。この例の場合は、 /kek/local/bin にインストールされる。特殊な機種以外はこれでインストールできるが、サポートされていない機種の場合には、手作業が必要となるので、マニュアルを熟読してから作業する。

```
% cd gzip-1.2.4
 ./configure --prefix=/kek/local
```

次に、システムにインストールされている make コマンドを使ってコンパイル、リンクを行い、インストールする。

```
% make
% make  install
```

これで gzip のインストールが完了した。 PATH にインストールしたディレクトリが入っているか確かめ、 csh を使っているなら rehash を行う。

make の配布ファイルを入手する。

```
% ftp ftp.kek.jp
Connected to ftp.kek.jp.
220 ftp FTP server (Version 4.1 Sun Mar 25 22:59:11 EST 1990) ready.
Name (ftp.kek.jp:sasaki): anonymous
331 Guest login ok, send ident as password.
Password: (自分のメールアドレスを入力する。エコーされない。)
230 Guest login ok, access restrictions apply.
ftp> cd pub/GNU
250 CWD command successful.
ftp> bin
ftp> get make-3.70.tar.gz
ftp> get make-doc-3.70.tar.gz
ftp> quit
```

make-doc-3.70.tar.gz にはマニュアルが入っている。 make-3.70.tar.gz からファイルを取り出し、インストールを行う。メジャーなマシンでは gzip 同様に簡単にインストールできる。 configure に同様にインストールするディレクトリを指定する。

```
% gunzip -c make-3.70.tar.gz | tar xvf -
% cd make-3.70
% ./configure --prefix=/kek/local
% make
% make install
```

これで、 GNU make のインストールが終了した。 GNU make のあるディレクトリがシステムのデフォルトの make のあるディレクトリより前に PATH に入っていることを確認し、そうでなければ修正する。 csh の場合は rehash し、 which コマンドで、 make が GNU make と同じか確認する。 make-doc-3.70.tar.gz から、マニュアルの dvi ファイルを取り出し、プリンタに打ち出しておくと便利である。

6.3 make コマンドの基礎

make は、 C、 FORTRAN 等、言語によらず、プログラム開発に使うことができる。オブジェクト・ファイルとソース・ファイルの最終変更時間を比べて、オブジェクトより新しいソース・ファイルだけコンパイルするので、ユーザはいくつかのファイルを変更した後でもただ make と打つだけでよく、効率の良い開発が行える。 make を使わずにシェルスクリプトを用いると、全てのファイルをコンパイルしてしまうか、または、ユーザが自分でどのファイルをコンパイルすべきかを指定しなければならない。ヘッダ・ファイルのように、他のファイルと依存性のあるファイルを変更した場合など、これらの作業は非常に面倒なものになる。しかし、 make を用い依存関係を makefile に書いておけば、 make コマンドが必要なファイルだけを選び出し、自動的にコンパイルの必要なファイルを選んで実行する。

make コマンドは、デフォルトで makefile または Makefile というファイルからコマンドを読み込む。もし、両方がそのディレクトリにあった場合には、 makefile の方が優先される。また、 −f オプションで異なるファイルを指定することもできる。ここではごく簡単な例しか挙げないが、 Makefile の書き方次第でかなり複雑な処理を行えるので、マニュアルを参照し各自工夫してほしい。

6.3.1 make の文法

詳しくは、 GNU make のマニュアルを参照すること。ここではごく簡単に説明する。

マクロ

実行されるコマンドの名前や、ファイル名、ディレクトリ名などを定義するのに用いる。 等号で結ばれた式の形をしており、例えばコンパイラの名前を定義するには、

```
CC=/usr/local/bin/gcc
```

と書く。コンパイラやリンカの名前など、 make がもともと持っているマクロもある。それらは、マニュアルに記載があるので参照して欲しい。例えば、 C コンパイラの名前のデフォルトは cc であるが、上記のようにして変更できる。また、コマンドラインからマクロ定義を与えることもできる。コンパイラ、リンカ両方の名前を変えたいなら、

```
% make "CC=/usr/local/bin/gcc LD=/usr/local/bin/gld"
```

のとすればよい。また、 SUN OS4.X の make なら /usr/include/make/default.mk に暗黙のルールが全て書いてある。 GNU make の場合は細部で異なる。また、シェルの環境変数をマクロとして使うこともできる。例えば、上の場合、

```
% setenv CC /usr/local/bin/gcc
% setenv LD /usr/local/bin/gld
% make
```

としても同じ効果が得られる。 (GNUmake 以外の場合、 −e オプションを付けないと、シェル変数を参照しないものもある。) make には、多くの組み込みマクロがあり、それらを利用すると、効率よくプログラム開発が行える。いくつか後ほど紹介するが、詳しくはマニュアルを参照してほしい。

ターゲット

ターゲットとは、例えばソース・コードから実行ファイルを作る場合、その実行ファイルの名前である。すでに makefile があったとすると、 a.out を作るには、

```
%make a.out
```

を実行する。ターゲットは、 makefile の中で指定することもできる。

依存関係

ターゲットに対する依存関係が定義できる。例えば、 x.c は、 x.h と依存関係があるというのは、次のように表せる。

```
x.c:x.h
```

複数のものに依存するような関係も許される。

```
x.c:x.h y.h z.h
```

ターゲットの右側に書かれたファイル名を、そのターゲットのコンポーネントと呼ぶ。

make がコマンドラインからのオプションなしで起動されたときのデフォルトのターゲットは all という名前である。 makefile に、

```
all:a.out
```

と書いておけば、 make と打つだけで a.out が生成される。 all というターゲットは a.out と依存関係があると定義したからである。そのターゲットを生成するためのコマンドをその次の行から複数行にまたがって書く。この時コマンドはタブに続いて書かなければならない。

```
all:a.out
  (tab) f77 x.o
```

ルール

ターゲットをどうやって作るか指定するために用いる。 makefile に次のような行があったとすると、

```
a.out: x.o y.o z.o
      f77 x.o y.o z.o
```

a.out が、 x.o、 y.o、 z.o と依存関係があり、 a.out を作るには f77 コマンドでオブジェクト・ファイルをリンクする、という意味になる。

サフィックス・ルール

言語によってソース・コードは決まったサフィックスを持っている。 make では、あるサフィックスを持ったファイルから別のファイルを生成する仕組みを定義できる。多くのコンパイラに対しては、ユーザが定義しなくても、内部に既に定義を持っている（暗黙のルール）。例えば、 FORTRAN のソース・コードから、 オブジェクト・ファイルを生成するためのルールは、

```
.f.o:$(FC) -c $(FFLAGS)
```

である。また、 LaTeX のソース・ファイルから、 DVI ファイルを生成するルールを作ると、

```
.tex.dvi:tex
```

となる。このルールは、同じサフィクスを持つ全てのファイルに適用される。

特別な意味を持つマクロ

いくつか、よく使うものだけを紹介する。

- $@ 処理中のターゲット名
- $? 処理中のターゲットより新しいコンポーネントのリスト
- $¡ 処理中のターゲットより古いコンポーネントのリスト
- $* 処理中のターゲットより古いコンポーネントからサフィックスを取ったもの。

6.3.2 文法のまとめ

まとめると、マクロの定義は、

```
MARCO = xxx yyy zzz
```

の形で行われ、

```
$(MARCO)
```

の形で参照される。ターゲットの定義は、

```
 target...  [:|::] [依存関係] ...  [; command] ...
    TAB           [command]
```

のようにできる。

6.3.3 例題１

例として、 a.f、 b.f、 c.f の三つのファイルと x.h、 y.h の二つのインクルード・ファイルから、ソース・コードをコンパイル、リンクし、プログラム aaa を生成するための Makefile を考えてみる。この中で、 x.h は三つの FORTRAN ソース・ファイルにインクルードされていて、 c.f だけが y.h をインクルードしている場合、 Makefile は次のようになる。

```
OBJS = a.o b.o c.o
all: aaa
aaa: $(OBJS)
     f77 -o aaa $(OBJS)
a.o: x.h
b.o: x.h
c.o: x.h y.h
```

１行目と２行目はそれぞれ、ソース・ファイルと作られるオブジェクト・ファイルを定義している（このような定義をマクロと呼ぶ）。

３行目は、 make と打った時に作られるプログラムの名前（ターゲット）を定義する。４行目と５行目では、 aaa が OBJS に定義されたオブジェクト・ファイルから作られ、次の行のコマンドで aaa が作られるということを定義している。

残りの行はインクルード・ファイルに対する依存関係を示している。すなわち、全てのオブジェクト・ファイルは x.h に依存しており、 c.f だけは y.h にも依存していることを示す。もし x.h が修正されると、全てのソースファイルがコンパイルしなおされる。それに対し y.h だけが修正されたときには、 c.f だけがコンパイルしなおされる。

この例ではマクロとして SRCS、 OBJS を使ったが、このキーワードは何でもよく、自分で定義できる。 $(OBJS) のように、マクロを $ マークと括弧で囲むと、 OBJS に定義された文字列が代入される。上の例では、

```
     f77 -o aaa $(OBJS)
```

は、実際には、

```
     f77 -o aaa a.o b.o c.o
```

という処理が行われる。

例えば、コンパイル時にデバッグ・オプションを付けたい時には、

```
FFLAGS = -g
```

という行を最初の方に付け加える。 FFLAGS というマクロは f77 を実行する時のオプションを定義するもので、 make の暗黙のルールの一つである。同様に C コンパイラに対しては、 CFLAGS というマクロがある。

ソース・ファイルからオブジェクト・ファイルを作るためのルールがどこにもないことに気づいたかもしれない。 C、 FORTRAN 等よく使われる言語のソース・ファイルとオブジェクト生成のルールは、すでに make の暗黙のルールで定義されているからである。それをそのまま使うなら書く必要がない。もし、そのルールを変更するために明示的に書きたければ、

```
.f.o: f77 -c $(FFLAGS)  $<
```

と書く。

一つの Makefile に複数のターゲットが定義されていても構わない。 make は、引数なしで起動された時には all というターゲットを生成するようになっている。上の例では、 make のターゲットは aaa であった。つまり、 all は aaa に依存しており、 aaa を作るためのルールがその次の行に書いてあった。 make はターゲットの名前を引数としてとれるので、

```
make aaa
```

としても、同じように aaa が作られる。

6.3.4 例題 2

少し複雑な例を考えてみる。 aaa が a.f、 b.f、 c.f から作られ、 bbb が b.f、 d.f、 e.f から作られる場合、

```
# examle 2
OBJ1 = a.o b.o c.o
OBJ2 = e.o  \
       b.o  \
       d.o
all: aaa bbb
aaa: $(OBJ1)
     f77 -o aaa $(OBJ1)
bbb： $(OBJ2)
     f77 -o bbb $(OBJ2)
a.o: x.h
b.o: x.h
c.o: x.h y.h
d.o: x.h y.h
e.o: x.h
```

というように書く。 # の後ろは、 Makefile の中では全てコメントとみなされる。バックスラッシュは行の継続を表す。前に述べたように、 make は最初から、 FORTRAN のソース・ファイルからオブジェクトを作る方法を知っているので、ソースからオブジェクトをつくるためのルールは不用である。この例では、 make または make all と打つと aaa 及び bbb が作られる。もし aaa もしくは bbb のみ作りたい時は、 make aaa 又は make bbb と打つ。

6.3.5 例題 3

前の例に、ソース・コードをプリンタに打ち出すためのターゲット print を付け加えてみる。

```
SRCS = a.f b.f c.f d.f e.f
OBJ1 = a.o b.o c.o
OBJ2 = e.o  \
       b.o  \
       d.o
all: aaa bbb
aaa: $(OBJ1)
     f77 -o aaa $(OBJ1)
bbb： $(OBJ2)
```

```
     f77 -o bbb $(OBJ2)
a.o: x.h
b.o: x.h
c.o: x.h y.h
d.o: x.h y.h
e.o: x.h
print:; lpr $(SRCS)
```

ターゲット print の後ろのセミコロンは依存関係の終了を表しており、このターゲットには何も依存関係がないということを表している。 make print とすると、ソース・ファイルがプリントされる。

6.4 make 応用編

6.4.1 Make の仕様

make を使いこなそうと思ったら、自分の make の持っている暗黙のルールを知っていなければならない。これを知る一番簡単な方法は、次のコマンドを使うことである。

```
%make -p -</dev/null
```

これにより、どのようなマクロが定義済みか分かり、コンパイルのフラグなどの変更が行える。

6.4.2 もっとマクロ

環境変数とマクロ

環境変数をマクロとして使えることは、すでに述べた。マクロ定義は、暗黙のルール、環境変数、そして makefile の中にも書けるので、その優先順位を知っておく必要がある。デフォルトでは、次のようになっている。

1. コマンドラインから与えられたもの
2. メイクファイル内で書かれているもの
3. 環境変数
4. 内部定義されているマクロ

しかし、時に都合で、環境変数を優先させたい時もある。そのような時は、”−e” フラグを付けてメイクを起動する。その場合は、

1. コマンドラインから与えられたもの
2. 環境変数
3. メイクファイル内で書かれているもの
4. 内部定義されているマクロ

と、優先順位が変更できる。コンパイラの名前や、コンパイル・オプションを一時的に変えたい時に便利である。

マクロ文字列の置換

ファイル名の一部などを置き換えてマクロ定義することもできる（古いヴァージョンの make では使えないので注意。)。下の例はソース・コードから、デバッグ・オプション付きでコンパイルしたものと、そうでない両方のオブジェクト・ファイルを生成し、名前の違うライブラリを作る場合の makefile である。

```
SRCS=a.f b.f c.f d.f
OBJ = $(SRCS:.c=.o)
OBJ_DEBUG = $(SRCS:.c=_dbg.o)
LIB = lib.a
LIB_DEBUG = $(LIB:.a=_dbg.a)
all: $(LIB) $(LIB_DEBUG)
$(LIB): $(OBJ)
        ar r $(LIB) $(OBJ)
        ranlib $(LIB)
$(LIB_DEBUG): $(OBJ_DEBUG)
        ar r $(LIB_DEBUG) $(OBJ_DEBUG)
        ranlib $(LIB_DEBUG)
a.o: x.h
b.o: x.h
c.o: x.h y.h
d.o: x.h y.h
```

.SUFFIXES マクロ

.SUFFIXES には、 make が扱うことのできるサフィックスのリストが定義されている。もし、標準と違うサフィックス・ルールを付け加えたい場合には、これも変更しなければならない。

```
.SUFFIXES: .ps
```

のようにすると、標準のルールに新しいサフィックス・ルールを付け加えられるようになる。また、自分でサッフィクス・ルールを作る際はデフォルトのルールと衝突する可能性があるので、追加するのではなく一度消去してしまった方が安全である。

```
.SUFFIXES:
.SUFFIXES:  .prt .nr
```

では、.nr というサフィクスを持つファイルを、 nroff コマンドで処理して、.prt という読めるファイルを作る場合を考えてみる。

```
.SUFFIXES:
.SUFFIXES:  .prt .nr
.nr.prt: nroff  -man $< > $(<:.nr=.prt)
```

.PHONY マクロ

不用なファイルを消去するためのターゲットのようなコンポーネントのないターゲットの場合、偶然そのターゲットと同じ名前のファイルがあるために make に失敗するということがありえる。そのような時は、.PHONY マクロを用いそれがコンポーネントなしのターゲットであると宣言することが可能である。

```
.PHONY:clean
clean:
        rm *.o
```

このようにしておけば、もし clean というファイルが合っても失敗することはない。

.SILENT マクロ

make は、どのコマンドを実行しているか画面に書きながら処理を進めるが、そのコマンドを表示させたくない場合には、このマクロを用いる。

```
.SILENT:clean
clean:
        rm *.o
```

6.5 サブディレクトリの make

複数のサブディレクトリをもつプログラムを make でコンパイル・リンクする方法を 2 通り紹介する。より洗練されたやり方もあるが、簡単な方法のみを紹介する。

6.5.1 シェル・コマンドを用いる方法

makefile の中には、シェル・コマンドを書くこともできる。それを利用した方法を紹介する。

```
SHELL=/bin/sh
MFLAGS =
CURRENT_DIR = .
SUBDIRS = dir1 dir2
all::
        @case '${MFLAGS}' in *[ik]*) set +e;; esac; \
        for i in $(SUBDIRS) ;\
        do \
        (cd $$i ; echo ``making'' all ``in $$(CURRENT_DIR)/$$i...''; \
        $(MAKE) $(MFLAGS)  all); \
        done
```

デフォルトで make が実行するするシェルは、/bin/sh であるが、ユーザのデフォルトのシェルを起動してしまうヴァージョンもあるので、SHELL マクロを定義しておいたほうが無難である。

6.5.2 VPATH マクロを用いる方法

オブジェクトを作るためのソース・コードのあるディレクトリを VPATH マクロに指定しておく。するとカレント・ディレクトリにソース・ファイルがない場合、VPATH で指定したディレクトリを参照しソース・ファイルをコンパイルしてくれる。

```
CFLAGS= $(SWAP) $(REMOTE) -Idir1 -Idir2
LFLAGS=
OBJS=vmsbackup.o getopt.o match.o rmtlib.o
VPATH= dir1 dir2
#
```

```
all:vmsbackup
vmsbackup: $(OBJS)
        cc $(LFLAGS) -o vmsbackup $(OBJS)
clean:
        rm -f vmsbackup *.o core *~
vmsbackup.o:vmsbackup.c
getopt.o:getopt.c
match.o:match.c
rmtlib.o:rmtlib.c
```

6.6 make のためのツール

6.6.1 mkmf

mkmf は、ソース・コードから makefile を生成するためのツールである。 OS には、標準でバンドルされていないので、 ftp.kek.jp:/pub/unix/mkmf.tar.Z を anonymous ftp してインストールすることが必要である。このツールには Makefile を生成するテンプレートがあり、ソース・コードのあるディレクトリで mkmf と打つだけで、それに従った Makefile を作ってくれる。 KEK のファイル・サーバの場合は、 /kek/local/bin にインストールされている。また、テンプレート・ファイルは、 /kek/local/lib に、ライブラリ生成用が l.Makefile、実行可能ファイル生成用が p.Makefile としておいてある。ユーザ自身が作ったテンプレートで makefile を作ることもできる。

6.6.2 makedepend

マサチューセッツ工科大学 (MIT) の X11 に含まれて配布されている。インクルード・ファイル等の複雑な依存関係を調べるのは面倒な作業であるが、このコマンドを使えば自動的に依存関係を調べて Makefile に追加してくれる。

6.6.3 imake

MIT が X11 の開発用に作ったコマンド。マルチプラットホーム用のソフトウエアを開発するには、コンパイルのフラグなどをマシンごとに変えなくてならず、 Makefile の扱いに注意が必要となる。それを解決するため、マシン毎のテンプレートに従い、 Imakefile というマシン依存性を取り除いた独自のフォーマットのファイルから imake が Makefile を生成する。 Imakefile から Makefile を作るには、通常 imake のフロントエンドである xmkmf を用いる。

6.7 この章の終りに

簡単な make の使い方を説明してきたが、ここに書いていない機能も数多くある。マニュアルを読み、 make の達人になれれば、プログラム開発の効率も必ず向上することであろう。また、マルチベンダー環境で、サブディレクトリが何層にも渡ってあるような大規模ソフトウエアの開発をするためには多くの困難があり、 それらの解決策の一つとして imake を紹介した。

第 7 章

CERN ライブラリ

CERN(欧州原子核研究所) では、膨大な数の数値計算、データ解析用のライブリやプログラムを蓄積している。 CERN または文部省高エネルギー物理学研究所と共同研究している機関では無料で、それ意外の非営利団体は 1000 フランで入手できる。そのうち、統計解析に役に立つ、 HBOOK と PAW について紹介する。

7.1 CERNLIB のインストール

KEK と共同研究している機関は、 KEK から再配布を受けられる。 KEK 内で新たにインストールする場合も同様に、データ処理センターに連絡して、申請用紙に必要事項を記入して提出しなければならない。著作権のないソフトウエアと勘違いされている人もなかにはいるようであるが、そうではないので、扱いに気を付けるべきである。もちろん、勝手に再配布することは、禁止されている。最近は、年に 2 回程バージョンアップがある。

KEK 所内の UNIX ワークステーションで使用する場合、共用のファイル・サーバを NFS マウントして使用するのが簡単である。 /kek/cern という PATH の下にバージョンごとに置かれている。 autumount を使っている場合には、 /etc/auto.master に次の行をいれる。

```
/kekfs /etc/auto.kekfs
```

/etc/auto.kekfs には、マシンに応じて、次のどれかをいれる。

```
%cat /etc/auto.kekfs.hp
kek     -ro,intr,suid           kiwifs:/public/hp/700/9.0

%cat /etc/auto.kekfs.sun
kek     -ro,intr,nosuid         kiwifs:/public/sun/4.1

%cat /etc/auto.kekfs.sol2
kek     -ro,intr,nosuid         kiwifs:/public/sun/Solaris2
```

そして、

```
ln -s /kekfs/kek /kek
```

とシンボリック・リンクを張る。

7.2 マニュアル

CERNLIB のマニュアルを X 端末上で見たいときは、 NCSA MOSAIC を利用するのが便利である。例えば、

```
%Mosaic http://wwwcn.cern.ch/asdoc/Welcome.html
```

を実行してみる。 Navigate menu の中の Hotlist コマンドで、このページを登録しておけば、次回からはメニューの中から選択できる。次の HTML は、 CERNLIB を使用するうえで役に立つであろう。

```
<TITLE> CENRLIB manaul pages</TITLE>
<BODY>
<BODY BGCOLOR="#AFE4EF">
<ul>
<h1> CERN library related pages</h1>
<h2>
<A href="http://asdwww.cern.ch/pl/"> CERN Program library home page</a>
<p>
<A href="http://asdwww.cern.ch/pl/paw/index.html"> PAW and PAW++</a>
<p>
<A href="http://wwwcn.cern.ch/asdoc/"> Library writeups</a>
<p>
<A href="http://asdwww.cern.ch/pl/paw/index.html"> Library information</a>
</ul>
</h2>
```

これを、 cernlib.html というファイルに落として、

```
%Mosaic cernlib.html
```

として、 Mosaic を起動する。見たい項目をマウスでクリックすると、必要な情報が得られるであろう。[1]

また、印刷物としてデータ処理センターにも置いて物もある。

7.3 CERNLIB の使い方

HBOOK を始め様々なライブラリが CERNLIB に含まれる。 CERNLIB を使うには、環境変数として、 CERN、 CERN_LEVEL を定義しておく。それぞれ、格納場所と使いたいバージョンを意味する。別のバージョンを使いたいときは、 CERN_LEVEL を変更する。.cshrc の中に次の様に書いておけばよい。

```
setenv CERN /kek/cern
setenv CERN_LEVEL 94a
source ${CERN}/${CERN_LEVEL}/mgr/plienv.csh
set path=($path ${BIN})
```

[1] Mosaic を利用すると様々な情報を世界中から得ることが可能である。 Mosaic http://www.kek.jp を試してみて欲しい。世界中のいろいろな情報へのポインタが用意されている。

3 行目のコマンドを実行することで、色々な定義が自動的に行わる。4 行目のコマンドにより、paw 等のコマンドへの PATH が設定される。ライブラリは、環境変数 LIB に定義されたディレクトリに用意されている。

```
ls $LIB
```

をやってみよう。どのライブラリをリンクするべきかは、使い方によるが、数値ライブラリと HBOOK 程度の使い方なら、 packlib、 pawlib、 mathlib で充分である。

7.4 HBOOK Ver.4 の使い方

実験のデータを処理するためヒストグラムを作り、データの統計処理を行なうことがよくある。このような場合、 HBOOK と言う CERN で作られたライブラリが最も適している。 HANDY PACK という物もあるが、最近はあまり使われなくなった。 HBOOK の Ver.3 と 4 では多少使い方が違う部分もあるが Ver.4 について簡単な使い方とプログラムのサンプルを紹介する。

HBOOK Ver.4 では、 1 次元、 2 次元のヒストグラムの他に、 n 次元のヒストグラムを取り、各変数の間の関係を見たり、一つの変数にカットを入れると他の変数の分布がどの様に変わるか見ることができる (N-tuple)。特に、 PAW を使って、会話的にカットを入れたり、任意の関数でフィティングを行ったり出来るの非常に強力である。 PAW の使い方は別の章で述べることにする。

7.4.1 サブルーチンの説明

HBOOK は多数の FORTRAN サブルーチンから構成され、ユーザが自分のプログラムからそれらを参照することで使用できる。最小限必要なものを紹介する。

HBOOK1

１次元のヒストグラムを定義するのに用いる。全ての変数を指定しなければならない。

```
CALL HBOOK1(ID,TITLE,NCH,L,H,MX)
     ID:Integer          ID
     TITLE:character     タイトル
     NCH:interger        ビンの数
     L: real             histgram の下限
     H: real             histgram の上限
     MX:real             各ビンのバッファサイズ、０を与えると
                         デフォルト値を取る。
```

それぞれのヒストグラムは、 ID として整数値をもっており、その ID 番号でアクセスされる。

HBOOK2

2 次元のヒストグラムを定義するのに用いる。

```
CALL HBOOK2(ID,TITLE,NXCH,XL,XH,NYCH,YL,YH,MX)
     ID:Integer
     TITLE:character     タイトル
```

```
NXCH:interger       X 軸のビンの数
XL: real            X の下限
XH: real            X の上限
NYCH:interger       Y 軸のビンの数
YL: real            Y の下限
YH: real            Y の上限
MX:real             各ビンのバッファサイズ、0 を与えると
                    デフォルト値を取る。
```

HFILL

ヒストグラムを fill する。

```
CALL HFILL(ID,X,Y,W)
     ID:Integer          fill したいヒストグラムの ID
     X:real              fill したい値
     Y:real              fill したい値 ( 1 次元のときは無視される。)
     W:real              weight
```

HISTDO

すべてのヒストグラムをテキスト表示する。 PAW 等でグラフィック表示するときは不用。

HRPUT

ヒストグラムのデータをファイルに書き込む。このデータを PAW で読み込んで処理することができる。

```
CALL HRPUT(ID,FILE,OPT)
    ID:integer        0 for all
    FILE: character   ファイル名
    OPT:character     オプション、'N' で普通はよい。詳しくはマニュ
                      アルをみよ。
```

7.4.2 コーディング

次に、これらをどのように使ったらいいのか見てみよう。 HBOOK で重要なのは、作業領域のためのメモリ確保である。それは、 COMMON 文によって行われるので、サブルーチンが沢山あろうと、どこかで一回だけすればよい。 Ver.4 では、 PAWC という名前付きコモン・ブロックを用いる。どの位のサイズを確保したらよいかは、どのようなヒストグラムを作るかによる。特に、多次元のものは多くのメモリを必要とする。

実際に例題を使って説明しよう。左の行番号は説明の為のものであり、実際には入れない。アスタリスクのあるカラムが本来 1 カラム目になる。

```
001
002          PROGRAM  EXAM1
003    *.==========>
004    *.            HBOOK BASIC EXAMPLE USING 1-DIM HISTOGRAM,
005    *.            SCATTER-PLOT AND TABLE.
006    *..=========> ( R.Brun )
```

```
007    *
008          COMMON/PAWC/H(80000)
009          CALL HLIMIT(80000)
010    *             Set global title
011          CALL HTITLE('EXAMPLE NO = 1')
013    *             Book 1-dim, scatter-plot and table
014    *
015          CALL HBOOK1(10,'EXAMPLE OF 1-DIM HISTOGRAM',100,1.,101.,0.)
016          CALL HBOOK2(20,'EXAMPLE OF SCATTER-PLOT',100,0.,1.,40,1.,41.,30.)
017          CALL HTABLE(30,'EXAMPLE OF TABLE',15,1.,16.,40,1.,41.,1000.)
019    *             Fill 1-dim histogram
020    *
021          DO 10 I=1,100
022             W=10*MOD(I,25)
023             CALL HFILL(10,FLOAT(I)+0.5,0.,W)
024       10 CONTINUE
026    *             Fill scatter-plot
027    *
028          X=-0.005
029          DO 30 I=1,100
030             X=X+0.01
031             DO 20 J=1,40
032                Y=J
033                IW=MOD(I,25)*MOD(J,10)
034                IWMAX=J-MOD(I,25)+10
035                IF(IW.GT.IWMAX)IW=0
036                CALL HFILL(20,X,Y,FLOAT(IW))
037       20    CONTINUE
038       30 CONTINUE
040    *             Fill table
041    *
042          DO 50 I=1,20
043             DO 40 J=1,40
044                CALL HFILL(30,FLOAT(I)+0.5,FLOAT(J)+0.5,FLOAT(I+J))
045       40    CONTINUE
046       50 CONTINUE
048    *             Print all histograms with an index
049    *
050          CALL HISTDO
051          CALL HRPUT(0,'hexam.dat','N')
052    *
053          END
```

８行目では、前述の作業領域を確保している。９行目は、作業領域として取った以上のメモリを使わないよう制限するためのもので、必ず行った方がよい。さもないと、メモリを破壊し思わぬ結果となる。１１行目は、グローバル・タイトルを設定するためのものである。１５行目から１７行目では、ヒストグラムの定義をしている。この例では、１次元と２次元にヒストグラムの他に、テーブルも作る。それ以下４６行目までで実際にヒストグラムをフィルし、５０行目で印字し、データは、５１行目で、hexam.dat というファイルに書いている。このプログラムをコンパイル・リンクするには、次のコマンドを用いる。

```
% f77 exam.f -lpacklib -L/kek/cern/94a/lib
```

ただし、ライブラリのおいてある場所は、ホストの事情に応じて変更する必要がある。

PAW を用い、このデータを表示したり加工したりできる。一番簡単な例としては、 PAW に対し、

```
PAW> hist/file 1 hexam.dat
PAW> ldir
PAW> hist/plot 10
```

というコマンドを与えると、 ID が 10 のヒストグラムをグラフィック表示する。もう少し詳しい PAW の使い方は後述する。

HBOOK や PAW のマニュアルは、 KEK の計算機センターが CERN と再配布の契約を結んだので、トリスタン計算機棟の事務室で入手できる。（その他の CERNLIB のマニュアルもある。）かなり高度な機能を持っているので、マニュアルが必要である。ここには、説明しなかった機能が沢山ある。たとえば、フィッティング等も会話的に簡単に行える。

7.4.3 フィッティング

ガウス分布 (HFITGA) や、対数分布（HFITEX）、多項式 (HFITPO) 等よく使う関数に対しては、専用のサブルーチンが用意されており、簡単に行うことができる。また、任意の関数に対して、フィッティングを行う為のサブルーチン (HFITL/HFITS) も用意されている。 また、ビンサイズが等しくないときに使えるサブルーチン HFITN も用意されている。

PAW の中で、会話的にこれらのフィッティングを行うこともできる。各変数の初期値を変えたり、関数の形を変えてフィッティングを行える。 PAW には、会話的 FORTRAN が用意されているので、毎回コンパイル、リンクをする必要がないので、便利である。

7.4.4 Ver.3 との違い

Ver.3 との大きな違いだけ述べる。作業領域の名前は、 Ver.3 では、名前なしであった。（ブランク・コモンと呼ぶ。）

```
      COMMON//H(80000)
```

のようにする。また、ヒストグラムのデータをファイルに書き込むには、 HSTORE というサブルーチンを用いた。 Ver.4 からは、 ZEBRA を使うように変わったので、 Ver.3 の HSTORE サブルーチンはサポートされなくなった。（代わりに、 HRPUT を使う。ダミーの HSTORE ルーチンが用意されているので、リンク はできるが、何もしてくれない。）しかし、互換性を保つ為、 Ver.3 で作られたデータ・ファイルを読み込めるようにはなっており、 PAW でも読める。

また、 n 次元のヒストグラム (N-TUPLES) を扱えるのは、 Ver.4 からである。

7.5 PAW の使い方

UNIX ワークステーションで使える PAW は、 CERN の HIGZ というグラフィック・パッケージに基づいているので、 HIGZ がサポートしているデバイスは全て使用可能である。 X ウィンドウ・システム、ポストスクリプト・プリンタ、 TEKTRONIX 4010 等をサポートしている。 そ

の他に、 GKS 版というのも VAX/VMS 用等にあり、それ意外のグラフィック・デバイスをサポートしていることもある。

PAW を使えば、ほとんど、どのようなグラフでも書くことができる。これは HBOOK と同様に CERN CN によって配布されているものである。 PAW について説明するといっても、その内容は幅広いので、例を取り上げていく。詳しくは、 PAW のマニュアルをみて各自研究してほしい。 PAW のマニュアルや、例題などは http://asdwww.cern.ch/pl/paw/index.html という URL を Mosaic などの WWW ブラウザを使ってアクセスすれば、見つかるだろう。

7.5.1 立ち上げと終了

PAW の立ち上げは、

```
%paw
```

と打つ。すると PAW 用のウィンドウが現れ、 PAW のコマンド待ち状態になる。最近、グラフィカル・ユーザ・インターフェースを備えた paw++ が公開された。メモリに余裕のあるシステムを使用している人には、そちらを勧めたい。 この場合は、

```
%paw++
```

と打てば、起動される。

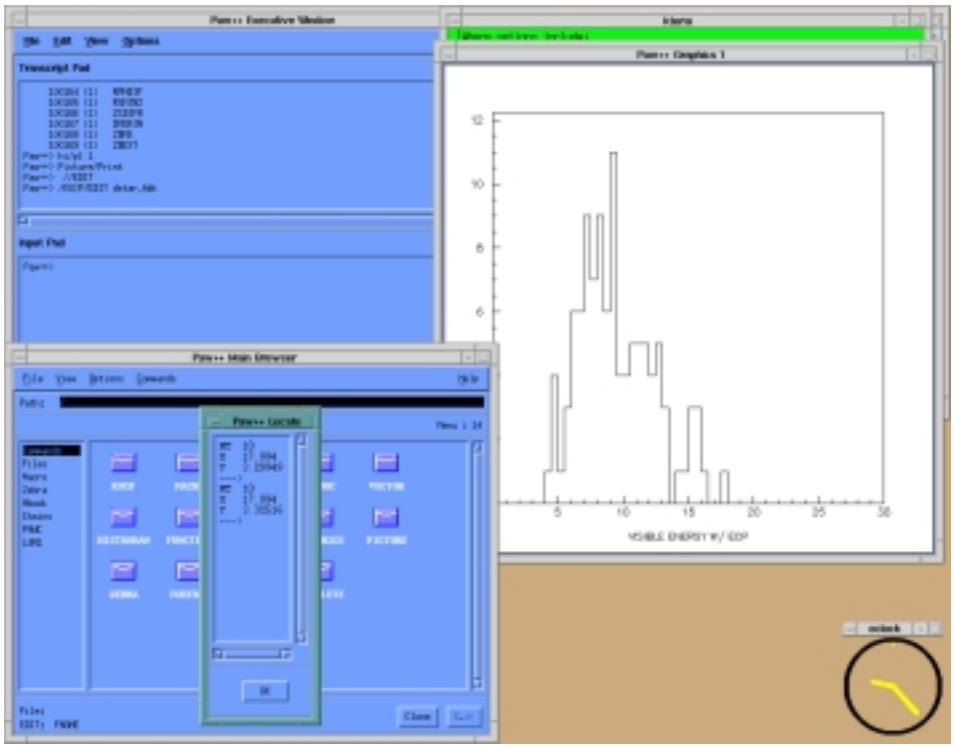


図 7.1: paw++

PAW が立ち上がる際に、カレント・ディレクトリに pawlogon.kumac という名のファイルがあるとこれを実行する。とりあえず、次のように書いてみよう（プロンプトを変えているだけ）。

```
%cat pawlogon.kumac

set/prompt 'PAWdayo[]>'
```

PAW を終了するときには、

```
PAWdayo[10]> q
```

とする。

7.5.2 ベクター・プロット

この方法は、データの値が分かっている時、単にそれをグラフ上にプロットしたいときなどに便利である。以下に例を示す。

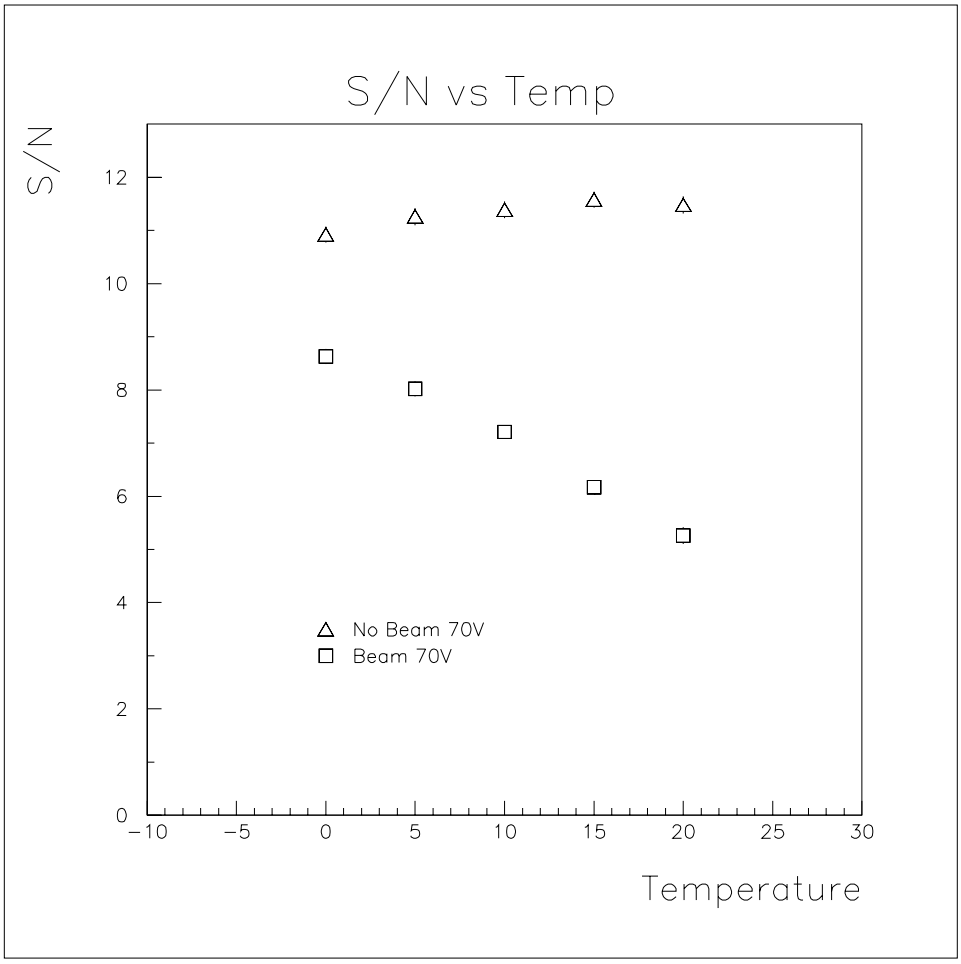


図 7.2: ベクター・プロット

このようなグラフは、データ点の情報を X や Y のベクターにして、プロットさせていくわけだが、ベクターへの値の代入などは、 PAW を立ち上げて一つひとつ打ち込んでいってもよい。

しかし、マクロという便利なものがあるので、これを利用しない手はない。マクロに関しては PAW のマニュアルの第四章 The KUIP Interface に書いてある。先に紹介した図も、マクロを使用して作成した。これは、カレント・ディレクトリに.kumac と言う拡張子のファイルを作りこのファイルに PAW で実行するコマンド群を書いておく。以下は、先程紹介した絵を書かせたファイルで、ファイル名が、 snvstmp.kumac の中身である。
説明のために、！で始まるコメントを書いておいた。実際には、！以降は書く必要はない。

```
*********************     ! '*' で始まる行は、コメントとみなされる。
* S/N vs Temp       *
*        1991-1-31  *
*********************
set xmgl 3.00             ! この set で始まる行は、 PAW のマニュアルの
set ymgl 3.00             ! 第 8 章. Graphics の 113page の絵
set asiz 0.50             ! にでている設定をきめる。グラフの題名
set ylab 1.50             ! を付けたりしないなら、余り気にしな
set xlab 2.00             ! くてよい。
set gsiz 0.6              !
set ymgu 2.50             !
title 'S/N vs Temp'       ! グラフのタイトル。
ve/cr x(5)                ! ve/cr は、ベクターを定義している。
ve/cr y(5)                ! 例えば、 ve/cr x(5) は、
ve/cr z(5)                ! vector/create x(5) の略で、
ve/cr w(5)                ! x (5ヶの配列) という名のベクターを
ve/cr a(5)                ! 定義している。
ve/cr b(5)
ve/cr c(5)
ve/cr d(5)
ve/inp a 20 15 10 5 0     ! 実際に定義したベクターの中に、数値を代入する。
ve/inp b 5.26 6.17 7.21 8.02 8.63 !ve/inp は /vector/input の略。
ve/inp c 0.16 0.13 0.13 0.14 0.14
ve/inp d 0 0 0 0 0
ve/inp x 20 15 10 5 0
ve/inp y 11.48 11.58 11.39 11.26 10.92
ve/inp z 0.16 0.15 0.16 0.16 0.15
ve/inp w 0 0 0 0 0
*
null -10 30 0 13          ! 座標軸の設定、左から、 Xmin,Xmax,Ymin,Ymax
*
atitle 'Temperature' 'S/N' ! X 軸,Y 軸のタイトル。
*
gra/hplot/errors a b d c 5 851 ! ベクターをプロットしていく。
gra/hplot/errors x y w z 5 852 ! エラー・バーも付ける。
*
key 0 3.5 852 'No Beam 70V'    ! key は、座標とコメントを指定すると、
key 0 3   851 '   Beam 70V'    ! グラフ上に書き込んでくれる。
```

この様に snvstmp.kumac を作成しておけば、 paw を立ち上げた後、

```
PAWdayo[1]> exec snvstmp
```

と打つだけで、実行してくれる。

7.5.3 HBOOK により作成されたヒストグラム

ここでは、 HBOOK Ver.4 によって作成された PAW 用のファイルを PAW 上で読み込み、絵を書かせるやり方を紹介する。 HBOOK で作成した PAW 用のファイルを foo.paw と言う名前だとして話を進める。以下に例を示した。

```
PAWdayo[1]> hi/file 1 foo.paw
PAWdayo[2]> ldir           ! これでヒストグラムの一覧がわかる。
PAWdayo[3]> hi/plot 60     ! 60 はヒストグラムの ID ナンバー。
PAWdayo[4]> hi/close 1
```

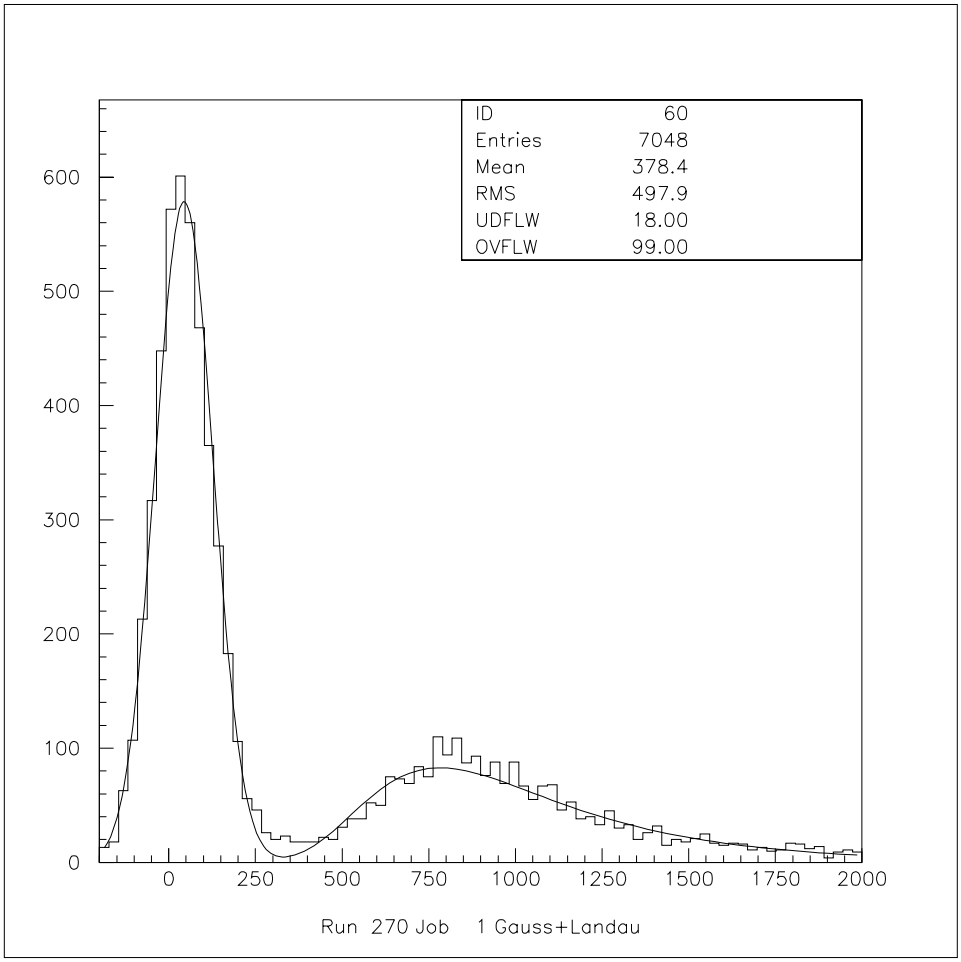


図 7.3: ヒストグラムの表示

7.5.4 ポストスクリプト・ファイルの作り方

PAW で書かせたグラフをポストスクリプト・プリンタに出力するために、ポストスクリプト・ファイルを作る。ポストスクリプト・ファイルを作る場合、 PAW 上で次のようにする。

```
PAWdayo[5]> fortran/file 66 paw.ps  !paw.ps はポストスクリプト・ファイル名。
PAWdayo[6]> Graphics/meta 66 -111   !-111 for portrait （デバイス名）
```

```
PAWdaya[7]> hi/plot 60                ! これは前節の場合。
PAWdayo[8]> exec snvstmp              ! マクロなら、こうなる。
PAWdayo[9]> fortran/close 66
```

また、 /.pawlogon.kumac の中に、

```
alias/creat psopen 'fort/file 66 paw.ps ; gra/meta 66 -111'
alias/creat psclose 'fort/close 66'
```

などと定義しておくと便利である。

デバイスとしては、この他にも、

```
METAFL=   4 Appendix E GKS
METAFL=-111 HIGZ/PostScript (Portrait).
METAFL=-112 HIGZ/PostScript (Landscape).
METAFL=-113 HIGZ/Encapsulated PostScript.
METAFL=-114 HIGZ/PostScript Color (Portrait).
METAFL=-115 HIGZ/PostScript Color (Landscape).
METAFL=-777 HIGZ/LaTex Encapsulated.
METAFL=-778 HIGZ/LaTex.
```

がある。こうして作られたポストスクリプト・ファイルをポストスクリプト・プリンタに送ることでヒストグラムを印刷できる。さらに、 Encapsulated PostScript を選べば、 TEX の文章のなかに簡単に挿絵を取り込める（例題の絵はそのようにして取り込みをしてある）。

7.5.5 フィッティング

PAW を使えば、好きな関数で簡単にフィティングができる。次の例では、ヒストグラムをガウス分布＋多項式でフィットしている。

```
PAW > hi/file 1 exam1.hbk
PAW > ve/cr par(6)
PAW > hi/fit 10(0:40) g ! 0 par(1:3)        ! 前半をガウス・フィット
PAW > hi/fit 10(40:100) p2 0 0 par(4:6)     ! 後半を多項式でフィット
PAW > hi/plot 10 sfunc                      ! 関数を同じ絵に重ねて描く
PAW > hi/fit 10(0:100) g+p2 ! 6 par         ! 全体をフィット
```

上記のコマンドは、

```
hi/fit id func option np par
```

という形をしている。 func には関数の名前を指定する。関数の形は、ガウス (g) や多項式 (pn) の他に、各自で定義することもできる。 np はパラメータの数、 par はパラメータの入ったベクターを意味する。 par にはフィットの前は初期値が、フィット後は結果の値が入る。例では np=0 となっており、この場合は初期値として par は使われないが、結果の値は par に代入される。例では、前半と後半を別々にフィットして大体のパラメータの値を決め、それを初期値にして全体のフィッティングを行なっている。詳細はマニュアル (Version 2.03) の 6 章を見てほしい。

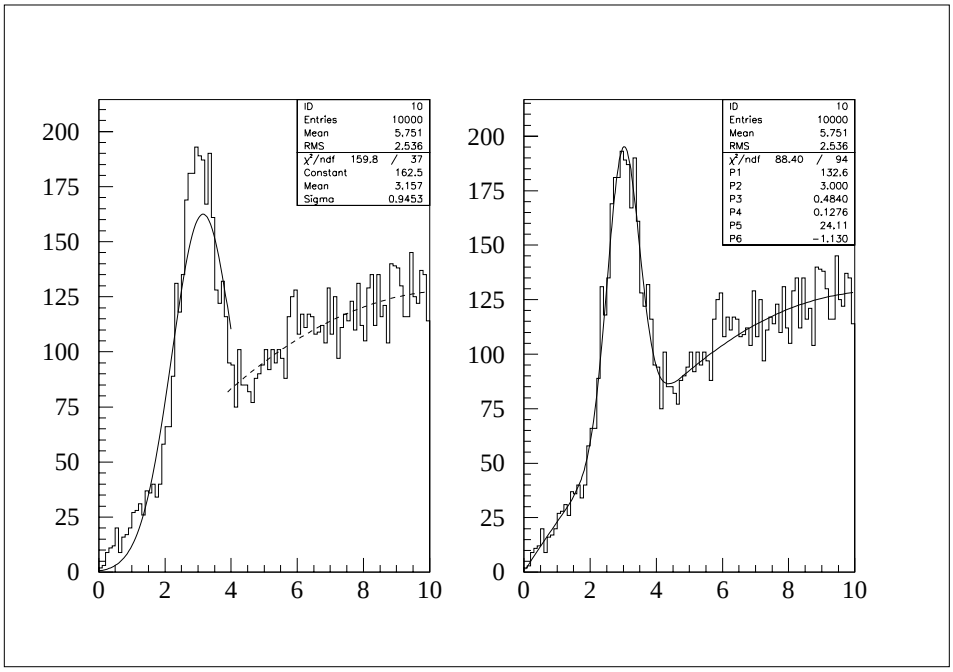


図 7.4: PAW を使ったフィッティングの例

第 8 章

UNIX システム管理

システム管理は、スーパーユーザの特権を持った管理者が主に行う。ワークステーションは一人一台の時代に入っているので、いつまでもお客さんではいられない。明日からは、管理者にならなくてはならないかもしれない。簡単に、システムの管理に必要な仕事について触れよう。その他システム管理者がしなくてはならない仕事については、[10] を参照のこと。

8.1 バックアップ

この節ではバックアップの方法について説明する。バックアップは、いざというときのための保険と言ってもよいであろう。予期せぬ事態により、ハードディスクが故障したり、システム自身が破損してしまった場合、私たちが日頃眠ずに働いた結果が一瞬のうちに消えさってしまうかもしれない。そのためにも、まめにバックアップを取り、システムに異常が発生したとき最小限の損失に抑えることができるように心がけよう。システム管理者の一番重要で、なおかつ、疎かにしがちな仕事である。

しかし、あまりシステム管理者に頼らず、大事な仕事をした後は、自分のディレクトリのバックアップは各自で取るようにしたほうが安全だろう。

バックアップを行うにあたって注意しなければならない点は、使用しているハードディスクの容量である。テープにバックアップを取る際に、テープの容量が十分あるかどうかが問題になってくるからである。それぞれのハードディスクがどのようなパーティションに分割されていて、どのディレクトリにマウントしてあるのかも、知っていなければならない。これらは、df コマンドを使って調べることができる。(パーティションに関しては、それぞれの OS のマニュアルや入門書等を参照すること。パーティションという概念が存在しない OS もある。)

8.1.1 tar コマンドを用いたバックアップ

/usr というディレクトリ以下のファイル全てのバックアップをディスクに取る場合のコマンドを示そう。

```
% cd /usr
% tar cvf - .  | compress > /scratch/usr.tar.Z
```

tar で複数のファイルを単一ファイルにまとめ、compress で圧縮している。ここで tar 及び compress についてはオンライン・マニュアルを参照のこと。バックアップのファイル名としては拡張

子を '.tar.Z ' とするとよいだろう。また、直接テープに書きたいときは、

```
%tar cvf /dev/rst1 /usr
```

のようにする。このときテープ・デバイスの名前は各システムで適当なものに変える必要がある。エグザバイト 8200 で 112m テープを用いた場合 2.3GB の容量がある。 SUN OS 4.1.2 の場合には、エグザバイト 8500 を使い、デバイス名として、 /dev/rst(0-7) を使うと約 2.3GB、 /dev/rst(8-15) を使えば、 5GB まで書き込むことができる。また、テープ・ドライブがローカルにつながっていない場合でも、 LAN のどこかにドライブのつながったワークステーションがあれば、

```
tar cvfb  -  20 /usr | rsh host dd of=/dev/rst0  obs=20b
```

のように行える。このコマンドは簡単に行え、一般ユーザの個人的バックアップに適している。ただし、このバックアップから一つのファイルだけとりだすのは楽ではないので、システム全体のバックアップには dump コマンドがよく使われている。 dump コマンドは、 System V 系のオペレーティング・システムには、 ufsdump 等という名前である場合や存在しない場合がある。その場合は、 cpio コマンド等を用いる。

8.1.2 dump を用いたバックアップ

スーパーユーザのみがこの作業を行える。シャットダウン・コマンドはシステムにより異なるので、それぞれのマニュアルを参照すること。まずシングル・ユーザ・モードに移行する。たとえば /home を /dev/st1 にバックアップする時は、

```
dump 0usf 120000 /dev/rst1 /home
```

とする。 パラメータはテープの長さが 120000 feet という意味である。これは 2.5GB の Exabyte テープに相当する長さである。最初の 0 はダンプ・レベルを表し、 0、 1、 3、 5、 3、 6 というふうにインクリメンタル・バックアップを取る。このほうが、 0、 1、 2、 3、 4、 5、 6 と連続したレベルで取るよりも復元が楽になる。テープからファイルを復元するには、 restore コマンドを用いる。通常は –i オプションとともに用い、簡単にファイルの復元が行える。

ネットワーク上の他のワークステーションに接続されているテープ・ドライブを使うには、 rdump/rrestore コマンドを使う。これを使うには、テープ・ドライブのつながっているマシンの自分（もちろん、 root でも可）のホームディレクトリに、.rhosts というファイルを作り、バックアップを取ろうとしているマシンの root がそのアカウントにアクセスできるように、エントリを追加する。作業後、このエントリは消して置いく。テープ・ドライブのつながっているマシンの名前と自分のアカウント名が、 ngthep、 aso だとすると、

```
dump 0usf  120000 aso@ngthep:/dev/rst1 /home
```

となる。

これらのバックアップを元に戻したいときは、 restore/rrestore コマンドを用い、 dump の時と同様に、

```
restore /dev/rst1
```

または、

```
rrestore aso@ngthep:/dev/rst1
```

とする。この時、 SUN OS 4.1.x では、 –i オプションを使うことで会話的に好きな数のファイルだけ取り出すことができ非常に便利である。

テープ・ドライブを使う上での一般的な注意としては、デフォルトのブロッキング・ファクタやレコード・フォーマットは、 OS のバージョンやマシンによって異なるということである。これは、異機種間でメディアのやり取りができるとは限らないことを意味している。異機種間でメディアのやり取りが予想される時は、細かいパラメタを与えてどのマシンでも読めるフォーマットで書かなければならない。そのような必要があるときは、 tar や dump のマニュアル、および、%man st、 %man mtio を参照して、間違いのないよう気を付けねばならない。一般的には、固定ブロック長で書いておくのがよいようである。

8.2 ユーザの登録

OS によっては、最初から便利なツールが付いてきている場合もある。 SUN の場合も、一応標準であるがあまり使いやすくない。もし、ユーザ登録用のツールに満足いくものがないときは、シェル・スクリプト adduser を使うと簡単に行える。”UNIX SYSTEM ADMINISTRATION HANDBOOK”、 Prentice Hall に載っているものである。 NIS を使っている時には、変更が必要である。

8.2.1 adduser の使い方

/etc/adduser の使い方は、スーパーユーザになり、

```
#/etc/adduser
```

と打ち、後は指示に従って入力していけばよい。 /etc/passwd,/etc/group の書き変え、及び新ユーザのホーム・ディレクトリの作成、.cshrc、.login 等のファイルのコピーを自動的に行ってくれる。使い方については指示に従って入力すればいいので、それほど難しいものではない。以下にその例を示す。

```
ngthep#/etc/adduser
enter login name, must be <= 8 characters
rie [return]
enter unix group
hep_niigata [return]
enter location of home dir
home/users [return]
enter users full name, campus address, campus phone, home phone
Rie Miyazawa , Niigata Univ [return], 263-XXXX

login                   rie
group                   hep
uid                     108
gid                     100
home                    /home/users/rie
```

```
passwd entry is:
rie::108:100:Rie Miyazawa,Niigata Univ,263-XXXX:/home/users/rie:/bin/csh

continue?(y/n) (last chance before scribbling on files)
y [return]
making passwd entry
making group entry
making phonelist entry
making home directory
/home/users/rie
chown rie.hep /home/users/rie
Changing password for rie on ngthep.
New password:
Retype new password:
No match
ngthep#exit
```

最後の No match というメッセージは、ワークファイルを消している時に出るが、無視して構わない。このシェル・スクリプトは、このテキストの置いてある場所から anonymous ftp で取得できる。

付録 A

スタートアップ・ファイルの例

A.1 (t)csh のスタートアップファイル

csh や tcsh の場合、シェルが起動されるたびに.cshrc が実行される。また、それがシステムへのログインを伴う場合には、.login も実行される。

A.1.1 .cshrc

```
#.cshrc example

# PATH の設定
#       X11 のバイナリのある場所、ローカルのバイナリのある場所
#       を自分のシステムに合わせて変更すること。
set path=(/usr/ucb  /bin /usr/bin /etc /usr/lib /usr/sbin  \
           /usr/lang  /kek/local/bin /usr/local/bin  \
           /kek/local/X11R6/bin )

# man コマンドのサーチパスの設定 ローカライズが必要
setenv MANPATH /usr/man:/usr/lang/man:/kek/local/X11R6/man:\
/kek/local/man:/usr/local/man

# interactive セッションのときのみ実行される
if ($?prompt) then
        set history=100
        set savehist=20
        set filec
        set cdpath=$home
        set notify ignoreeof
        set prompt="$user@`hostname`[\!]_% "
        set noclobber
        alias Mosaic 'Mosaic-2.5-l10n-0.sparc-cjk  \!*'
        alias cernman 'Mosaic http://asis01.cern.ch/CN/CNASDOC/Overview.html'
endif

# CERNLIB の環境の設定    CERN の値は、ローカライズすること
setenv CERN /kek/cern
setenv CERN_LEVEL 94a
source ${CERN}/${CERN_LEVEL}/mgr/plienv.csh
```

```
set path=($path ${BIN})
```

A.1.2 .login

```
#
# .login
#                                                25/Oct/90 T. Sasaki

# Set terminal name.
set noglob
eval `/usr/ucb/tset  -e -k^U -rsQ -m network:?vt100 -m unknown:?vt100`
unset noglob

# For terminal emulater on X.
if ($term == xterm || $term == kterm || $term == hpterm) then
        set noglob
        eval `resize`
        unset noglob
endif
# tty setting  CTRL_C で、割り込み
stty intr ^C  pass8
```

A.2 X11 のスタートアップ・ファイル

コンソールから、 startx としたときには、.xinitrc の中身が実行される。 X 端末などから、 xdm でログインしたときは、.xinitrc が実行されるシステムと、. xsession の場合がある。 HP の VUE などの用に X11 の大きく手をいれている場合には、ファイル名は異なる。

X11 のアプリケーションのデフォルトの様々な設定は、.Xdefaults というファイルに書いておく。

A.2.1 .xinitrc および.xsession の例

```
#PATH の設定
PATH=/kek/local/X11R6/bin:/usr/ucb:/usr/openwin/bin
export PATH

#SUN の場合のみ　シェアードライブラリのサーチパスの設定
#LD_LIBRARAY_PATH=/usr/local/X11R6/usr/lib:/usr/openwin/lib:/usr/lib
#export LD_LIBRARY_PATH

xterm -geometry 85x39+143+155 -T kterm@`hostname` \
-ms goldenrod -cr white   -bg "black" -fg floralwhite  \
-bd yellow -fk kanji16 -fn 8x16 &

# xconsole が無い場合 xterm -C -geometry 60x12+0+0& と置き換える
xconsole -geometry 600x125+0+0 &
# 時計
oclock -geometry 120x120--0-0 -trans -jewel red -fg yellow &
```

```
# マウスのポインタを追いかける目玉
xeyes -geometry +0-90 -fg blue&
# メイルが来たら知らせてくれる
xbiff -geometry +1080+580 -fg blue4 -bg navajowhite -update 15&
# 背景の色の設定
xsetroot -solid burlywood3
#window manager の実行
exec fvwm
wait
exit
```

A.2.2 .Xdefaults の例

```
Mosaic*tmpDirectory: ~/

kterm*VT100*translations: #override Meta<Key> : string("\033") insert()
kterm*vt100*translations: #override Shift<Key>space: begin-conversion()
KTerm*saveLines:        500
KTerm*scrollLines:      10
XTerm*scrollBar:        on
XKTerm*saveLines:       500
XTerm*scrollLines:      10
XTerm*scrollBar:        on

Mwm*buttonBindings:             DefaultButtonBindings
Mwm*Kinput.clientDecoration:    None
Mwm*InputServer.clientDecoration:       None
Mwm*vjeTransient.clientDecoration:      None
Mwm*keyBindings:                DefaultKeyBindings
Mwm*windowMenu:                 DefaultWindowMenu
Mwm*useIconBox:         true
Mwm*iconBoxGeometry:    1x5-1+1
Mwm*iconPlacement:      bottom left

Scrollbar.JumpCursor:   True

Ghostscript*yResolution:        75
Ghostscript*geometry:   650x850-10-10
Ghostscript*xResolution:        75

XDvi.psSpesial: true
```

A.3 .emacs の例

```
; mule と emacs で共用する例

; 標準のメジャーモードを text-mode にする。
(setq default-major-mode 'text-mode)

; Lisp ライブラリの load path に "~/elisp" を追加する。
(if (boundp 'MULE)
```

```
 (setq load-path (cons (expand-file-name "~/elisp" "/kek/local/lib/mule/site-lisp")
(if (boundp 'MULE)
(load "japanese")
)
(if (boundp 'NEMACS)
 (setq load-path (cons (expand-file-name "~/elisp" "/kek/local/emacs/site-lisp")
                       load-path)))
; ディスプレイに表示するときの文字コードを日本語 EUC にする。
 (cond ((boundp 'NEMACS) (setq kanji-display-code 3))
           ((boundp 'MULE)
            (set-display-coding-system *euc-japan*)))

; 新バッファ作成時のファイル文字コードを *euc-japan* にする。
     (if (boundp 'MULE)
         (set-default-file-coding-system *euc-japan*))

; MH のプログラムとやりとりするときの文字コードを JIS に指定する。
     (if (boundp 'MULE)
         (progn
           (define-program-coding-system nil ".*scan.*" (cons *junet* *junet*))
           (define-program-coding-system nil ".*inc.*" (cons *junet*
     *junet*))))
     (if (boundp 'NEMACS)
         (progn
           (define-program-kanji-code nil ".*scan.*" 2)
           (define-program-kanji-code nil ".*inc.*" 2)))

; Text モードにするとき、常に auto-fill-mode にする。
     (setq text-mode-hook
           '(lambda () (auto-fill-mode 1)))

; mail-mode にしたときに、ファイル文字コードを JIS にセットするように
; する。
     (setq mail-mode-hook
           '(lambda ()
         (setq kanji-fileio-code 2)  ; for Nemacs
         (if (boundp 'MULE)
             (set-file-coding-system *junet*))))

;;; for MH
(setq mh-progs "/kek/local/bin" )
(defun mh-scan ()
  (autoload 'mh-find-path "mh-e" "MH" t)
  (interactive)
  (mh-find-path)
  (call-interactively 'mh-visit-folder)
)
;; gnus
(autoload 'gnus "gnus" "Read network news." t)
(autoload 'gnus-post-news "gnuspost" "Post a news." t)
 (setq gnus-nntp-server "keknews.kek.jp")
 (setq gnus-local-domain "kek.jp")
 (setq gnus-local-organization "KEK, Nat'l Lab. for HEP, ...")
```

```
;; (setq nntp-buggy-select "T")
;;

;;
;; Make AUC TeX be loaded automatically when one of the following
;; three commands are invoked.
(load-file "/kek/local/mule/site-lisp/auctex/tex-site.elc")
(autoload 'tex-mode "auc-tex" "Automatic select TeX or LaTeX mode" t)
(autoload 'plain-tex-mode "auc-tex" "Mode for Plain TeX" t)
(autoload 'latex-mode "auc-tex" "Mode for LaTeX" t)
;;;;
(load-file "/kek/local/lib/mule/19.25/lisp/hilit19.elc")
;;  Xcal Mode
(autoload 'xcal "xcal-19")
(autoload 'day-of-week "cal")
(autoload 'generate-month "cal")
(autoload 'hilit-lookup-face-create "hilit19")
;
;
(substitute-key-definition 'rmail 'mh-rmail menu-bar-file-menu)
(substitute-key-definition 'smail 'mh-smail menu-bar-file-menu)
;;

;; キーバインドの変更
; 矢印キーが使えるようにする。(透過モード, 入力文字変換モード用)
(global-set-key "\eOD" 'backward-char)  ; ←
(global-set-key "\eOC" 'forward-char)   ; →
(global-set-key "\eOB" 'next-line)      ; ↓
(global-set-key "\eOA" 'previous-line)  ; ↑

;               〃              (フェンスモード用)
(define-key fence-mode-map "\eO" nil)
(define-key fence-mode-map "\eOD" 'fence-backward-char) ; ←
(define-key fence-mode-map "\eOC" 'fence-forward-char)  ; →
(define-key fence-mode-map "\eOB" 'undefined)           ; ↓
(define-key fence-mode-map "\eOA" 'undefined)           ; ↑

;               〃              (漢字変換モード用)
(define-key henkan-mode-map "\eO" nil)
(define-key henkan-mode-map "\eOD" 'henkan-backward-bunsetu)    ; ←
(define-key henkan-mode-map "\eOC" 'henkan-forward-bunsetu)     ; →
(define-key henkan-mode-map "\eOB" 'henkan-next-kouho)          ; ↓
(define-key henkan-mode-map "\eOA" 'henkan-previous-kouho)      ; ↑

; BS (C-h) キーと DEL キーを入れ替える。
;;;(load-library "term/bobcat")


;;sj3 kana-kanji
;(set-default-sys-dic-directory "/kek/local/dict/sj3")
;(set-default-usr-dic-directory "/usr/local/dict/sj3/user/$USER")
;(setq *sj3-service-name* 3000)
```

```
; jserver（仮名漢字変換サーバ）のホスト名の設定
(setq wnn-host-name "kektrws1")

; "nn" で 「ん」が入力できるようにする。
(setq enable-double-n-syntax t)

(load "egg-keymap")

; C-n で無駄な空白行が増えてしまうのを防ぐ。
(cond ((eq (string-to-int emacs-version) 18)
       (defun next-line (arg)
         (interactive "p")
         (next-line-internal arg)
         nil))
      ((eq (string-to-int emacs-version) 19)
       (setq next-line-add-newlines nil)))
```

A.4 vi のスタートアップファイル

.exrc が vi のスタートアップファイルである。

```
set autoindent
set autoprint
set noignorecase
set nomesg
set noslowopen
set noterse
set nonumber
set showmode
"
set report=2
set tabstop=8
set wrapmargin=0
"
"map ; :
"map g :%
"map v ~
"map m !} fmt -c
"map T !} sort
"
```

付録 B

Q & A

初心者のよく出会うトラブルとその対処法について述べる。

- “-” で始まるファイルを消す。

```
rm ./-filename
```

- おかしなキャラクタの入ったファイルができて消せない。

```
rm -i some*pattern*that*matches*only*the*file*you*want
rm -ri .
```

- ディレクトリの再帰的表示。

```
ls -R                   (not all versions of ‘‘ls’’ have -R)
find . -print           (should work everywhere)
du -a .                 (shows you both the name and size)
```

- ファイルの拡張子を foo から bar に直す。

```
foreach f ( *.foo )
     set base=‘basename $f .foo‘
     mv $f $base.bar
end
```

- rsh （SYSV 系マシンでは、 remsh) がうまく働かない。
  .cshrc の中に、

```
stty erase ^H
biff y
```

  があると、

```
% rsh some-machine date
stty: : Can't assign requested address
Where are you?
Tue Oct  1 09:24:45 EST 1991
```

  のようになる。

```
if ( $?prompt ) then
    biff y
endif
```

  と直す。

- 間違ってファイルを消してしまった。
  通常のシステムでは、取り返しがきない。バックアップがなければ、あきらめるしかない。
  転ばぬ先のバックアップ。

- talk コマンドが違う機種同士の間で働かない。
  SUN のように古いバージョンの talk しかサポートしていないマシンと他のマシンの間でよく起きるトラブルである。 SUN に新しい talk をインストールする。 ntalk というコマンド名で新しい talk を入れてある場合があるので、探してみよう。

- 誰が自分を finger したか知りたい。
  基本的には、できない。

- 他の人は、カーソル・キーでコマンドのリコールができるのに、自分はできない。
  使っているシェルが違うか、ターミナルの設定がカーソル・キーが働くようになっていない。

- Emacs を使うと画面が固まる。
  端末が、 XON/XOFF プロトコルでフロー制御しているとこの問題が起る。別の端末を使うか、 XON/XOFF 以外のフロー制御を使うように設定する。

- 画面のスクロールがおかしい。。
  eval ‘resize‘を実行してみよう。

付録 C

GNU Emacs Reference Card

GNU Emacs のソースコードやマニュアルと一緒に配布されているリファレンスカードである。最新版は Emacs のソースコードとともに配布されている。通常は、 /usr/local/emacs/etc/refcard.tex というファイル名である筈である。 2 column や 3 column でも出力できるので、自分で出力して持っていると便利である。 plain TEX でフォーマットする。

```
                GNU Emacs Reference Card

                   (for version 19)


Starting Emacs


To enter GNU Emacs 19, just type its name:  emacs

To read in a file to edit, see Files, below.


Leaving Emacs


suspend Emacs (or iconify it under X)            C-z
exit Emacs permanently                           C-x C-c


Files


read a file into Emacs                           C-x C-f
save a file back to disk                         C-x C-s
save all files                                   C-x s
insert contents of another file into this buffer C-x i
```

```
replace this file with the file you really want   C-x C-v
write buffer to a specified file                  C-x C-w
```

Getting Help

The Help system is simple. Type C-h and follow the directions. If you are a first-time user, type C-h t for a tutorial.

```
remove Help window                                C-x 1
scroll Help window                                ESC C-v

apropos: show commands matching a string          C-h a
show the function a key runs                      C-h c
describe a function                               C-h f
get mode-specific information                     C-h m
```

Error Recovery

```
abort partially typed or executing command        C-g
recover a file lost by a system crash             M-x recover-file
undo an unwanted change                           C-x u or C-_
restore a buffer to its original contents         M-x revert-buffer
redraw garbaged screen                            C-l
```

Incremental Search

```
search forward                                    C-s
search backward                                   C-r
regular expression search                         C-M-s
reverse regular expression search                 C-M-r

select previous search string                     M-p
select next later search string                   M-n
exit incremental search                           RET
undo effect of last character                     DEL
abort current search                              C-g
```

Use C-s or C-r again to repeat the search in either direction. If Emacs is still searching, C-g cancels only the part not done.

Motion

entity to move over	backward	forward
character	C-b	C-f
word	M-b	M-f
line	C-p	C-n
go to line beginning (or end)	C-a	C-e
sentence	M-a	M-e
paragraph	M--	M-"
page	C-x [	C-x]
sexp	C-M-b	C-M-f
function	C-M-a	C-M-e
go to buffer beginning (or end)	M-<	M->

scroll to next screen	C-v
scroll to previous screen	M-v
scroll left	C-x <
scroll right	C-x >
scroll current line to center of screen	C-u C-l

Killing and Deleting

entity to kill	backward	forward
character (delete, not kill)	DEL	C-d
word	M-DEL	M-d
line (to end of)	M-0 C-k	C-k
sentence	C-x DEL	M-k
sexp	M-- C-M-k	C-M-k

kill region	C-w
copy region to kill ring	M-w
kill through next occurrence of char	M-z char
yank back last thing killed	C-y
replace last yank with previous kill	M-y

Marking

set mark here	C-@ or C-SPC
exchange point and mark	C-x C-x
set mark arg words away	M-@
mark paragraph	M-h
mark page	C-x C-p
mark sexp	C-M-@
mark function	C-M-h
mark entire buffer	C-x h

Query Replace

interactively replace a text string	M-%
using regular expressions	M-x query-replace-regexp

Valid responses in query-replace mode are

replace this one, go on to next	SPC
replace this one, don't move	,
skip to next without replacing	DEL
replace all remaining matches	!
back up to the previous match	^
exit query-replace	ESC
enter recursive edit (C-M-c to exit)	C-r

Multiple Windows

delete all other windows	C-x 1
delete this window	C-x 0
split window in two vertically	C-x 2
split window in two horizontally	C-x 3
scroll other window	C-M-v
switch cursor to another window	C-x o
shrink window shorter	M-x shrink-window
grow window taller	C-x ^
shrink window narrower	C-x -
grow window wider	C-x "
select buffer in other window	C-x 4 b
display buffer in other window	C-x 4 C-o
find file in other window	C-x 4 f
find file read-only in other window	C-x 4 r
run Dired in other window	C-x 4 d
find tag in other window	C-x 4 .

Formatting

indent current line (mode-dependent)	TAB
indent region (mode-dependent)	C-M-"
indent sexp (mode-dependent)	C-M-q
indent region rigidly arg columns	C-x TAB
insert newline after point	C-o
move rest of line vertically down	C-M-o

```
delete blank lines around point          C-x C-o
join line with previous (with arg, next) M-^
delete all white space around point      M-"
put exactly one space at point           M-SPC

fill paragraph                          M-q
set fill column                         C-x f
set prefix each line starts with        C-x .
```

Case Change

```
uppercase word                          M-u
lowercase word                          M-l
capitalize word                         M-c

uppercase region                        C-x C-u
lowercase region                        C-x C-l
capitalize region                M-x capitalize-region
```

The Minibuffer

The following keys are defined in the minibuffer.

```
complete as much as possible           TAB
complete up to one word                SPC
complete and execute                   RET
show possible completions              ?
fetch previous minibuffer input        M-p
fetch next later minibuffer input      M-n
regexp search backward through history M-r
regexp search forward through history  M-s
abort command                          C-g
```

Type C-x ESC ESC to edit and repeat the last command that used the minibuffer. The following keys are then defined.

```
previous minibuffer command            M-p
next minibuffer command                M-n
```

Buffers

```
select another buffer                  C-x b
list all buffers                       C-x C-b
kill a buffer                          C-x k
```

Transposing

```
transpose characters                    C-t
transpose words                         M-t
transpose lines                         C-x C-t
transpose sexps                         C-M-t
```

Spelling Check

```
check spelling of current word          M-$
check spelling of all words in region   M-x ispell-region
check spelling of entire buffer         M-x ispell-buffer
```

Tags

```
find a tag (a definition)               M-.
find next occurrence of tag             C-u M-.
specify a new tags file          M-x visit-tags-table

regexp search on all files in tagsMtable-x tags-search
run query-replace on all the filesM-x tags-query-replace
continue last tags search or query-replaceM-,
```

Shells

```
execute a shell command                 M-!
run a shell command on the region       M-_
filter region through a shell commandC-u M-_
start a shell in window *shell*    M-x shell
```

Rectangles

```
copy rectangle to register              C-x r r
kill rectangle                          C-x r k
yank rectangle                          C-x r y
open rectangle, shifting text right     C-x r o
blank out rectangle               M-x clear-rectangle
prefix each line with a string    M-x string-rectangle
```

Abbrevs

add global abbrev	C-x a g
add mode-local abbrev	C-x a l
add global expansion for this abbrev	C-x a i g
add mode-local expansion for this abbrev	C-x a i l
explicitly expand abbrev	C-x a e
expand previous word dynamically	M-/

Regular Expressions

any single character except a newline. (dot)

zero or more repeats	*
one or more repeats	+
zero or one repeat	?
any character in the set	[: :]:
any character not in the set	[^ : :]:
beginning of line	^
end of line	$
quote a special character c	"c
alternative ("or")	"_
grouping	"(: :":)
nth group	"n
beginning of buffer	"‘
end of buffer	"’
word break	"b
not beginning or end of word	"B
beginning of word	"<
end of word	">
any word-syntax character	"w
any non-word-syntax character	"W
character with syntax c	"sc
character with syntax not c	"Sc

Registers

save region in register	C-x r s
insert register contents into buffer	C-x r i
save value of point in register	C-x r SPC
jump to point saved in register	C-x r j

Info

```
enter the Info documentation reader  C-h i

Moving within a node:

  scroll forward                     SPC
  scroll reverse                     DEL
  beginning of node                  .  (dot)

Moving between nodes:

  next node                            n
  previous node                        p
  move up                              u
  select menu item by name             m
  select nth menu item by number (1-5)   n
  follow cross reference (return with l) f
  return to last node you saw          l
  return to directory node             d
  go to any node by name               g

Other:

  run Info tutorial                    h
  list Info commands                   ?
  quit Info                            q
  search nodes for regexp              s



Keyboard Macros


start defining a keyboard macro      C-x (

end keyboard macro definition        C-x )
execute last-defined keyboard macro  C-x e
append to last keyboard macro        C-u C-x (
name last keyboard macro             M-x name-last-kbd-macro
insert Lisp definition in buffer     M-x insert-kbd-macro



Commands Dealing with Emacs Lisp


eval sexp before point               C-x C-e
eval current defun                   C-M-x
eval region                          M-x eval-region
eval entire buffer                   M-x eval-current-buffer
read and eval minibuffer             M-ESC
```

```
re-execute last minibuffer command   C-x ESC ESC
read and eval Emacs Lisp file        M-x load-file
load from standard system directory  M-x load-library
```

Simple Customization

Here are some examples of binding global keys in Emacs Lisp. Note that you cannot say ""M-#"; you must say ""e#".

```
(global-set-key ""C-cg" 'goto-line)
(global-set-key ""C-x"C-k" 'kill-region)
(global-set-key ""e#" 'query-replace-regexp)
```

An example of setting a variable in Emacs Lisp:

```
(setq backup-by-copying-when-linked t)
```

Writing Commands

```
(defun command-name (args)
  "documentation"
  (interactive "template")
  body)
```

An example:

```
(defun this-line-to-top-of-window (line)
  "Reposition line point is on to top of window.
With ARG, put point on line ARG.
Negative counts from bottom."
  (interactive "P")
  (recenter (if (null line)
                0
              (prefix-numeric-value line))))
```

The argument to interactive is a string specifying how to get the arguments when the function is called interactively. Type C-h f interactive for more information.

For copies of the GNU Emacs manual, write to the Free Software Foundation, Inc., 675 Massachusetts Ave, Cambridge MA 02139.

付録 D

vi リファレンス

kekux.kek.jp の anonymous FTP から入手したファイルを収録した。

```
////////////////////////////////////////////////////////////////////////
/                          VI REFERENCE (by maart@cs.vu.nl)            /
////////////////////////////////////////////////////////////////////////

default values          : 1
^X                      : <ctrl>x
[*]                     : * is optional
<*>                     : * must not be taken literally
<sp>                    : space
<cr>                    : carriage return
<lf>                    : linefeed
<ht>                    : horizontal tab
<esc>                   : escape
<del>                   : delete
<a-z>                   : an element in the range
N                       : number (* = allowed, - = not used)
CHAR                    : char unequal to <ht>|<sp>
WORD                    : word followed by <ht>|<sp>|<lf>

/////////////////
/ move commands /
/////////////////

 N | Command             | Meaning
---+---------------------+-----------------------------------------------
 * | h | ^H              | <*> chars to the left
 * | j | <lf> | ^N       | <*> lines downward
 * | l | <sp>            | <*> chars to the right
 * | k | ^P              | <*> lines upward
 * | $                   | to the end of line <*> from the cursor
 - | ^                   | to the first CHAR of the line
 * | _                   | to the first CHAR <*> - 1 lines lower
 * | -                   | to the first CHAR <*> lines higher
 * | + | <cr>            | to the first CHAR <*> lines lower
 - | 0                   | to the first char of the line
 * | |                   | to column <*> (<ht>: only to the endpoint !)
 * | f<char>             | <*> <char>s to the right (find)
```

```
 * | t<char>            | till before <*> <char>s to the right
 * | F<char>            | <*> <char>s to the left
 * | T<char>            | till after <*> <char>s to the left
 * | ;                  | repeat latest "f"|"t"|"F"|"T" <*> times
 * | ,                  | idem in opposite direction
 * | w                  | <*> words forward
 * | W                  | <*> WORDS forward
 * | b                  | <*> words backward
 * | B                  | <*> WORDS backward
 * | e                  | to the end of word <*> forward
 * | E                  | to the end of WORD <*> forward
 * | G                  | go to line <*> (default EOF)
 * | H                  | to line <*> from top of the screen (home)
 * | L                  | to line <*> from bottom of the screen (last)
 - | M                  | to the middle line of the screen
 * | )                  | <*> sentences forward
 * | (                  | <*> sentences backward
 * | }                  | <*> paragraphs forward
 * | {                  | <*> paragraphs backward
 - | ]]                 | to the next section (default EOF)
 - | [[                 | to the previous section (default begin of file)
 - | `<a-z>            | to the mark
 - | '<a-z>            | to the first CHAR of the line with the mark
 - | ``                | to the cursor position before the latest absolute
                       | jump (of which are examples "/" and "G")
 - | ''                | to the first CHAR of the line on which the cursor
                       | was placed before the latest absolute jump
 - | /<string>         | to the next occurrence of <string>
 - | ?<string>         | to the previous occurrence of <string>
 - | n                  | repeat latest "/"|"?" (next)
 - | N                  | idem in opposite direction
 - | %                  | find the next bracket and go to its match
                       | (also { } and [ ])

//////////////////////
/ searching (see above) /
//////////////////////

^]                     | search in the tags file where the function under the
                       | cursor is defined (file, line) and go to it
:[x,y]g/<string>/<cmd> | search globally [from line x to y] after <string>
                       | and execute the "ex" <cmd> on each occurrence

//////////////////
/ undoing changes /
//////////////////

u                      | undo the latest change
U                      | undo all changes on a line, while not having moved
                       | off it (unfortunately)
:q!                    | quit vi without writing
:e!                    | re-edit a messed-up file
```

```
////////////////////////////////////
/ appending text (end with <esc>) /
////////////////////////////////////

 * | a                  | <*> times after the cursor
 * | A                  | <*> times at the end of line
 * | i                  | <*> times before the cursor (insert)
 * | I                  | <*> times before the first CHAR of the line
 * | o                  | on a new line below the current (open)
                        | the count is only useful on a slow terminal
 * | O                  | on a new line above the current
                        | the count is only useful on a slow terminal
 * | ><move>            | shift the lines described by <*><move> one
                        | shiftwidth to the right (layout)
 * | >>                 | shift <*> lines one shiftwidth to the right
 * | .                  | repeat latest command <*> times
 * | ["<a-z1-9>]p       | put the contents of the (default undo) buffer <*>
                        | times after the cursor
                        | a buffer containing lines is put only once, below
                        | the current line
 * | ["<a-z1-9>]P       | put the contents of the (default undo) buffer <*>
                        | times before the cursor
                        | a buffer containing lines is put only once, above
                        | the current line

/////////////////
/ deleting text /
/////////////////

Everything deleted can be stored into a buffer. This is achieved by putting a
" and a letter <a-z> before the delete command. The deleted text will be in
the buffer with the used letter. If <A-Z> is used as buffer name, the adjugate
buffer <a-z> will be augmented instead of overwritten with the text. The undo
buffer always contains the latest change. Buffers "<1-9> contain the latest 9
LINE deletions ("1 is most recent).

 * | x                  | delete <*> chars under and after the cursor
 * | X                  | <*> chars before the cursor
 * | d<move>            | from begin to endpoint of <*><move>
 * | dd                 | <*> lines
 - | D                  | the rest of the line
 * | <<move>            | shift the lines described by <*><move> one
                        | shiftwidth to the left (layout)
 * | <<                 | shift <*> lines one shiftwidth to the left
 * | .                  | repeat latest command <*> times

////////////////////////////////
/ changing text (end with <esc> /
////////////////////////////////

 * | r<char>            | replace <*> chars by <char> - no <esc>
 * | R                  | overwrite the rest of the line, append <*> - 1 times
 * | s                  | substitute <*> chars
```

```
 * | S                   | <*> lines
 * | c<move>             | change from begin to endpoint of <*><move>
 * | cc                  | <*> lines
 * | C                   | the rest of the line and <*> - 1 next lines
 * | =<move>             | if the option 'lisp' is set, this command will
                         | realign the lines described by <*><move> as though
                         | they had been typed with the option 'ai' set too
 - | ~                   | switch lower and upper cases
 * | J                   | join <*> lines (default 2)
 * | .                   | repeat latest command <*> times ("J" only once)
 - | &                   | repeat latest "ex" substitute command, e.g.
                         | ":s/wrong/good"

//////////////////////////////
/ remembering text (yanking) /
//////////////////////////////

With yank commands you can put "<a-z> before the command, just as with
delete commands. Otherwise you only copy to the undo buffer. The use of buffers
<a-z> is THE way of copying text to another file: see the ":e <file>" command.

 * | y<move>             | yank from begin to endpoint of <*><move>
 * | yy                  | <*> lines
 * | Y                   | idem (should be equivalent to "y$" though)
 - | m<a-z>              | mark the cursor position with a letter

//////////////////////////////////////
/ commands while in append|change mode /
//////////////////////////////////////

^@                       | if typed as the first character of the insertion, it
                         | is replaced with the previous text inserted (max. 128
                         | chars), after which the insertion is terminated
^V                       | deprive the next char of its special meaning
                         | (e.g. <esc>)
^D                       | one shiftwidth to the left
0^D                      | remove all indentation on the current line
                         | (there must be no other chars on the line)
^^D                      | idem, except that it is restored on the next line
^T                       | one shiftwidth to the right
^H                       | one char back
^W                       | one word back
^U                       | back to the begin of the change on the current line
                         | (generally your kill char)
<del>                    | like <esc>

////////////////////////////////////////////////
/ writing, editing other files, and quitting vi /
////////////////////////////////////////////////

:q                       | quit vi after writing
:q!                      | quit vi without writing
:w                       | write the file
```

```
:w <name>               | write to the file <name>
:w >> <name>            | append the buffer to the file <name>
:w! <name>              | overwrite the file <name>
:x,y w <name>           | write lines x through y to the file <name>
:wq                     | write the file and quit vi
ZZ                      | write if the buffer has been changed, and quit vi
:x                      | idem
:x!                     | "w!" and "q"
:e <file>               | edit another file without quitting vi - the buffers
                        | are not changed (except the undo buffer), so text can
                        | be copied from one file to another this way
:e! <file>              | idem, without writing the current buffer
:e#                     | edit the previous file
^^                      | idem
:rew                    | edit the first file (when "vi file1 file2 ...")
:rew!                   | idem, without writing the current buffer
:n [<file>]             | edit the next file
:n! [<file>]            | idem, without writing the current buffer

////////////////////
/ display commands /
////////////////////

^G                      | give current line number and relative position
^L                      | refresh the screen (sometimes "^P" or "^R")
^R                      | sometimes vi replaces a deleted line by a '@', to be
                        | deleted by "^R" (also with option 'noredraw')
[*]^E                   | scroll <*> lines downward
[*]^Y                   | scroll <*> lines upward
[*]^D                   | scroll <*> lines downward
                        | (default the number of the previous scroll;
                        | initialization: half a page)
[*]^U                   | scroll <*> lines upward
                        | (default the number of the previous scroll;
                        | initialization: half a page)
[*]^F                   | <*> pages forward
[*]^B                   | <*> pages backward (in older versions only ^B works)

If in the next commands the field <wi> is present, the windowsize will change
to <wi>. The window will always be displayed at the bottom of the screen.

[*]z[wi]<cr>            | put line <*> at the top of the window
                        | (default the current line)
[*]z[wi]+               | put line <*> at the top of the window
                        | (default the first line of the next page)
[*]z[wi]-               | put line <*> at the bottom of the window
                        | (default the current line)
[*]z[wi].               | put line <*> in the centre of the window
                        | (default the current line)

////////////////////////////
/ mapping and abbreviation /
////////////////////////////
```

```
When mapping turn off the option 'timeout' by ":set noto", and with 'map!'
turn on 'remap' by ":set remap".

:map <string> <seq>     | <string> is interpreted as <seq>, e.g.
                        | ":map ^C :!cc %^V<cr>" to compile from within vi
                        | (vi replaces % by the current file name)
:map                    | show all mappings
:unmap <string>         | deprive <string> of its mapping
                        | when vi complains about non-mapped macros (whereas
                        | no typos are made), first do something like
                        | ":map <string> Z", followed by ":unmap <string>"
                        | ('Z' must not be a macro itself)
:map! <string> <seq>    | mapping in append mode, e.g.
                        | ":map! \be begin^V<cr>end;^V<esc>O<ht>"
                        | when <string> is preceded by ^V, no mapping is done
:map!                   | show all append mode mappings
:unmap! <string>        | deprive <string> of its mapping (see ":unmap")
:ab <string> <seq>      | whenever in append mode <string> is followed by a
                        | breakpoint (e.g. <sp> or ','), it is interpreted as
                        | <seq>, e.g. ":ab p procedure"
                        | a char preceded by ^V is not considered a breakpoint
:ab                     | show all abbreviations
:unab <string>          | do not consider <string> an abbreviation anymore
                        | (see ":unmap")
@<a-z>                  | consider the contents of the named register a
                        | command, e.g.:
                        |       o0^D:s/wrong/good/<esc>"zdd
                        | explanation:
                        |       o             - open a new line
                        |       0^D           - remove indentation
                        |       :s/wrong/good/ - this input text is an "ex"
                        |                       substitute command
                        |       <esc>         - finish the input
                        |       "zdd          - delete the line just created,
                        |                       into register 'z'
                        | now you can type @z to substitute 'wrong' by 'good'
                        | on the current line
@@                      | repeat last register command

////////////////////////////
/ switch and shell commands /
////////////////////////////

Q | <del><del>          | switch from vi to "ex"
:                       | an "ex" command can be given
:vi                     | switch from "ex" to vi
:sh                     | execute a subshell, back to vi by ^D
:!<cmd>                 | execute a shell <cmd>
:!!                     | repeat the last shell command
[*]!<move><cmd>         | the shell executes <cmd>, with as standard input the
                        | lines described by <*><move>, next the standard
                        | output replaces those lines
```

```
                        | (think of cb, sort, nroff, etc.)
[*]!!<cmd>              | give <*> lines as standard input to the shell <cmd>,
                        | next let the standard output replace those lines
:x,y w !<cmd>           | let lines x to y be standard input for <cmd>
                        | (notice the space between 'w' and '!')
:r!<cmd>                | put the output of <cmd> onto a new line
:r <name>               | read the file <name> into the buffer

//////////////
/ vi startup /
//////////////

vi [file]               : edit the file and display the first page

The editor can be initialized by the shell variable EXINIT, which looks like:

        EXINIT='<cmd>|<cmd>|...'
        <cmd>:  set options
                map ...
                ab ...
        export EXINIT (in the Bourne shell)

However, the list of initializations can also be put into a file. If this
file is located in your home directory, and is named ".exrc" AND the
variable EXINIT is NOT set, the list will be automatically excuted at
startup time. If the 3 conditions are not met, you have to give the execute
command yourself:

        :source file
or
        :so file

On-line initializations can be given with "vi +<cmd> file", e.g.:

vi +x file              : the cursor will immediately jump to line x
vi +/<string> file      : ~ to the first occurrence of <string>

Sometimes (e.g. if the system crashed while you were editing) it is possible
to recover files lost in the editor by "vi -r file".
If you just want to view a file by using vi, and you want to avoid any change,
instead of vi you can use the "view" command: the option 'readonly' will be set
automatically (with ":w!" you can override this option).
The most important options are:

ai                      | autoindent - in append mode after a <cr> the cursor
                        | will move directly below the first CHAR on the
                        | previous line.
                        | however, if the option 'lisp' is set, the cursor
                        | will align at the first argument to the last open
                        | list.
aw                      | autowrite - write at every shell escape
                        | (useful when compiling from within vi)
dir=<string>            | directory - the directory for vi to make temporary
```

```
                        | files (default /tmp)
eb                      | errorbells - beeps when you goof
                        | (not on every terminal)
ic                      | ignorecase - no distinction between upper and lower
                        | cases when searching
lisp                    | redefine the following commands:
                        | "(", ")"   - move backward (forward) over
                        |             S-expressions
                        | "{", "}"   - idem, but don't stop at atoms
                        | "[[", "]]" - go to previous (next) line beginning
                        |             with a '('
                        | see option 'ai'
list                    | <lf> is shown as '$', <ht> as '^I'
magic                   | some metachars can be used when searching:
                        | ^<string>    - <string> must begin the line
                        | <string>$    - <string> must end the line
                        | .            - matches any char
                        | [a-z]        - matches any char in the range
                        | [<string>]   - matches any char in <string>
                        | [^<string>]  - matches any char not in <string>
                        | <char>*      - 0 or more <char>s
                        | \<<string>\> - <string> must be a word
nu                      | number - numbers before the lines
para=<string>           | paragraphs - every pair of chars in <string> is
                        | considered a paragraph delimiter nroff macro (for "{"
                        | and "}").
                        | a <sp> preceded by a '\' indicates that the previous
                        | char is a single letter macro.
                        | ":set para=P\ bp" introduces '.P' and '.bp' as
                        | paragraph delimiters.
                        | furthermore completely empty lines and section
                        | boundaries are paragraph boundaries too.
redraw                  | the screen remains up to date
report=<*>              | vi reports whenever e.g. a delete
                        | or yank command affects <*> or more lines
ro                      | readonly - the file is not to be changed
                        | however, ":w!" will override this option
sect=<string>           | sections - gives the section delimiters (for "[[" and
                        | "]]"); see option 'para', however a '{' as first
                        | char on a line also starts a section (C functions!)
sh=<string>             | shell - which program is to be used for shell escapes
sw=<*>                  | shiftwidth - gives the swiftwidth (default sw=8)
sm                      | showmatch - whenever you append a ')', vi tries to
                        | show its match by putting for a moment the cursor
                        | onto it (also with { })
terse                   | short error messages
ts=<*>                  | tabstop - the length of a <ht>;
                        | warning: this is only IN the editor, outside of it
                        | <ht>s have their normal length (default ts=8)
wa                      | writeany - no checks when writing (dangerous)
warn                    | warn you when you try to quit without writing
wi=<*>                  | window - the number of lines vi is to show default
wm=<*>                  | wrapmargin - when in append mode vi automatically
```

```
                        | puts a <lf> whenever there is a breakpoint (e.g. <sp>
                        | or ',') within <wm> columns from the right margin
ws                      | wrapscan - when searching, the end is considered
                        | 'stuck' to the begin of the file

:set option             | turn option on
:set no option          | turn option off; no <sp> between "no" and the option
:set option=value       | give an option a value
:set                    | show all non-default options and their values
:set option?            | show an option's value
:set all                | show all options and their values
```

付録 E

Glossary

UNIX を使っていると聞き慣れない多くの言葉に出会い、戸惑います。 brendan@cs.widener.edu 氏の作った用語集を日本語化したものを中心に、良く見る単語を集めてみました。

E.1 コンピュータ・ネットワーク関連

- **Address Resolution**
  インターネットアドレスと対応する実際の名前との変換

- **address** アドレス
  電子メイルのアドレスと言う意味か、ホストのインターネットアドレスの意味のどちらかで使われる。

- **alias** 別名
  別の名前に翻訳可能な名前。

- **American National Standards Institute (ANSI)**
  アメリカ合衆国のいろいろな標準を作るお役所。

- **American Standard Code for Information Interchange (ASCII)**
  8 進法の数字と文字の対応付の標準。 EBCDIC というもう一つ有名な標準もある。

- **Anonymous FTP**
  匿名 ftp。アカウントを持っていないマシンに、"anonymous" とか"guest" というユーザ ID を使って ftp でログインしてファイル操作をすること。このとき、パスワードの変わりに、自分のメイルアドレスを必ず入力する決まりになっている。

- **ANSI**
  "American National Standards Institute"を見よ。

- **Appletalk**
  アップル社の作ったネットワークの規格。 Localtalk を使用すると 235K bit/second、 Ethernet では、 10M bit/second の最大転送速度が得られる。

- **application** アプリケーション
  ユーザにとり、なんらかの役目をしてくれるプログラム。ワードプロセッサ等。

- **archie**
  どこの anoymous ftp に自分の欲しいファイルがあるか探す為のコマンド。

- **ASCII**
  "American Standard Code for Information Inter- change" を見よ。

- **backbone**
  階層的な分散システムの一番基本となる接続の仕組み。 backbone 上の中継システムと接続されている全てのシステムは、互いに接続を持つことを保証されている。これは、コスト, パフォーマンス, セキュリティなどの理由によって backbone を回避するという個別の設定をすることが出来ないということではない。

- **bandwidth**
  技術的には、伝送経路での最高, 最低周波数の差を（ヘルツ,Hz で）いう。しかし典型的には、与えられた伝達回路を通って送ることのできるデータの総計をいう。

- **BBS**
  "Bulletin Board System" を見よ。

- **BITNET**
  元来は IBM メインフレームシステムをもとにした学術研究用コンピュータネットワーク。 IBM NJE プロトコルに基づいたバケツリレー方式による電子メイルとファイル転送サービスを提供する。 BITNET と Internet は、幾つかのゲートウエイ経由で、相互に乗り入れできる。このネットワークは現在 the Corporation for Research and Educational Networking (CREN) の一部となっている。

- **Berkeley Source Distribution (BSD)**
  この３文字は UNIX オペレーションシステムのこのバージョンを表すのに使われており、そのソフトウエアはカルフォルニア大学バークレイ校によって開発され配付された。 "BSD" は通常配付 (OS) のヴァージョンナンバーの後に置かれる。例えば "4.3 BSD" は Berkeley UNIX distribution の version 4.3 である。たくさんの インターネットホスト上で BSD は稼働しており、多くの商業 UNIX implementations の御先祖であった。

- **big-endian**
  最も意味のある bit (or byte) が最初に来る、データの貯蔵または転送のためのフォーマット。逆のフォーマットは little-endian と呼ばれる。

- **broadcast**
  与えられたパケットのコピーがネットワークに接続されている全てのホストに同時に送られるような、パケット配達システム。

- **BSD**
  "Berkeley Software Distribution" を見よ。

- **cache**
  ディスクへフラッシュされることなく、メインメモリに保たれている小量の情報。情報を cashe へとどめておくことで、データをディスクに取りに行く必要が軽減される。

- **CCITT**
  "Comite Consultatif International de Telegraphique et Telephonique" を見よ。

- **Checksum**
  メッセージまたはパケットの全体の内容に依存したある計算された量。この値は通常、メッセージが転送されるときそれとともに送られる。受けとり側のシステムは受けとったデータに基づいて新しい Checksum を計算し、パケットとともに送られてきた値と比較する。もし二つの値が同じであれば、受けとり側はデータを正しく受けとれたということについて高い信頼性を持つことができる。

- **client**
  他のコンピュータシステムまたはプロセスのサービスを要求するコンピュータシステムまたはプロセス。ファイルサーバからのファイルの内容を要求しているワークステーションは、ファイルサーバの client である。

- **client-server model**
  ネットワークサービスや、プログラムのサービスのモデルを記述する一般的な方法。例えば DNS の典型である name-server/name-resolver や、 NFS と diskless ホストのような関係である、 file-server/file-client などである。

- **CRC**
  cyclic redundancy check

- **DARPA**
  Defense Advanced Research Projects Agency

- **DCE**
  Distributed Computing Environment

- **DECnet**
  Digital Equipment Corporation によって設計されたネットワークプロトコル。実装のそれぞれのフェーズでの機能は、 (たとえば、 Phase IV と Phase V) 異なる。

- **Domain Name System (DNS)**
  The Domain Name System はインターネット上で使用されている、ホストコンピュータの名前をアドレスに変換する機能である。 DNS はまた、ホストコンピュータがインターネット上に直接に同一形式で登録された名前を持たなくてもよいことになっている。重要なド

メインをいくつか挙げると、.COM (commercial), .EDU (educational), .NET (network operations), .GOV (U.S. government), .MIL (U.S. military), and .US, .UK, etc (national) などである。

- ドメイン
  インターネットでの階層的な名前の一部。構文上では、インターネットドメインネームは、ピリオド (dots) によって分けられた名前 (ラベル) の連なりである。例えば"tundra.mpk.ca.us"

- **EBCDIC**
  "Extended Binary Coded Decimal Interchange Code" を見よ。

- 電子メイル
  コンピュータのユーザがネットワークを通じて他のユーザ (又はユーザのグループ) とメッセージを交換できるというシステム。電子めいる はインターネットの最もポピュラーな使用法の一つである。

- **email** アドレス
  指定された宛先に電子メイルを送るために使われる、 UUCP または ドメイン名に基づいたアドレス。例えば筆者のアドレスは brendan@cs.widener.edu である。

- **email**
  "電子メイル"を見よ。

- **Ethernet**
  ローカルエリアネットワーク (LAN) のための 10 メガビット毎秒 の規格で、最初は Xerox によって開発され、後に Xerox, DEC そして Intel によって改善された。全てのホストは同軸ケーブルにつながれ、 CSMA/CD プロトコルによってネットワークへアクセスする。種々の上位のプロトコル DECnet, TCP/IP, and XNS は、基本的な通信手段として Ethernet を使用する。

- **Extended Binary Coded Decimal Interchange Code (EBCDIC)**
  IBM 関係で使用されている character code scheme。 "ASCII" の項も参照のこと。

- **FDDI**
  Fiber Distributed Data Interface

- **Fiber Distributed Data Interface (FDDI)**
  高速の LAN 用 (100Mb/sec) のネットワークの規格。光ファイバーを媒体として用い、トポロジーは、 2 重リングかトークンリングである。

- **File Transfer Protocol (FTP)**
  あるインターネットホスト上のユーザにインターネットのようなネットワークを通じて他のホストとアクセスしたり、ファイルを転送することを許すプロトコル。 FTP は通常、プロトコルの名前というだけではなく、ユーザがプロトコルを実行するために起動したプログラムの名前でもある。 (例えば "ftp host.bbn.com") このプロトコルは IP の上位層にあ

る TCP の上位に存在する。FTP は種々の OS で稼働している。ftp command を使って、ソフトウェア, 書類, マップなどの様な様々な情報を含んだコンピュータファイルをコピーすることができる。

- **finger**
  システムまたはリモートシステムにログインしている、全ての (又は特定の) ユーザを表示するプログラム。典型例では、full name, last login time, idle time, terminal line, and terminal location (where applicable) が表示される。ユーザによって置かれている plan file も表示する。

- **gateway**
  gateway については、両方ともネットワークで使われているにも関わらず、少し矛盾した二つの定義がある。一つは、一つかそれ以上のネットワークを相互接続するために使われているコンピュータのことである。このコンピュータは、そのコンピュータが接続されているネットワークからは一台のコンピュータとして見えるが、あるネットワークから、別のネットワークへパケットを転送する機能を持つ。ゲートウエイは、インターネット内の別のゲートウエイに経路情報を提供したり、もらったりする機能もあるので、ネットワーク間でパケットを送るときに、どの経路が一番良いかということを知ることができる。もう一つの意味は、TCP/IP と ISO のような異なるプロトコル間の変換を行うシステムと言うことである。

- **GNU**
  "Gnu's Not Unix"を見よ。

- **Gnu's Not Unix (GNU)**
  Free Software Foundation によって開発された UNIX で使用可能なソフトウェア。

- **HEPnet**
  High Energy Physics NETwork

- **HIPPI**
  High Performance Parallel Interface

- **host**
  ユーザが、ネットワーク上の他のホストコンピュータと通信することができるコンピュータ。個々のユーザは 電子メイル, TELNET , FTP などのアプリケーションプログラムを使って、通信をすることが出来る。

- **hostname**
  マシンに与えられた名前。"Fully Qualified Domain Name" の項も参照のこと。

- **hub**
  種々の他の機器をつなげる機器。ARCnet では、hub は種々のコンピュータ同志をつなげるのに使われる。message handling service では、hub はネットワーク上でのメッセージの転送のために使われる。

- **Integrated Services Digital Network (ISDN)**
  世界中の電信電話会社よって提供され始めている新興技術。ISDN は音声とデジタルのネットワークサービスを単媒体で結合させており、それは一本のワイヤーを通して音声と同じくらい良いデジタルデータサービスを顧客に提供することを可能にしている。ISDN を定義する基準は CCITT によって指定されている。

- **Internet Address**
  インターネットホストをユニークに認識する 32bit の数。このアドレスは典型的には、ドットによって分けられ 8bit の数 (8 進数) で言い替えられる。例えば 128.89.1.132。Internet address は network number , host number, class A, B, or C address に一致する。class A のネットワークアドレスは N.H.H.H. という形式で、ネットワークナンバーが 7bit、ホストナンバーが 24bit と規定されている。(例えば 26.0.0.117 は ネット 26 の ホスト 117 を表す) Class B では N.N.H.H. という形式で、ネットワークナンバーが 14bit、ホストナンバーが 16bit(例えば 128.89.1.132)。Class C では N.N.N.H. という形式で、ネットワークナンバーが 22bit、ホストナンバーが 8bit (例えば 192.1.14.28 は ネットワークナンバー 192.1.14 のホスト 28 をあらわす。)。

- **Internet Protocol (IP)**
  Internet を越えて送られる情報の単位として Internet datagram を決定している Internet 標準プロトコル。Internet 接続と非常に効果的なパケット集配サービスの基礎をもたらしている。RFC 791 で決められているデータグラムプロトコルに由来している。

- **internet**
  ゲイトウェイやルータで互いに接続されているネットワークの集団。インターネットはいくつかのネットワークプロトコルに基づいたコンピュータネットワークの集団である。インターネットはあたかも集団全体が一つのおおきな仮想的なネットワークであるかのように機能する。

- **Internet**
  (大文字の “I” に注意) 世界最大の internet であり、 MILNET, NSFNET, CREN などの大きな国際バックボーンネットや、世界中の無数の地方または局地的な キャンパスネットワークによって構成されている。 Internet は Internet プロトコルを使用している。 Internet に乗るには、例えばリモートネットワーク上の他のシステムに正規の IP パケットで (例えば Telnet や パケットを”ping” する) たどり着けるような IP 接続性を持っていなければならない。電子メイルでだけ接続されているネットワークは Internet 上にあるとは分類されない。

- **IP address**
  TCP/IP を用いて Internet に参加したいホストに assign される 32-bit アドレス。

- **IP**
  ”Internet Protocol”を見よ。

- **kernel**
  アプリケーションプログラムのための標準 API を供給するソフトウェアの本体。一般的には、 kernel には OS の構造や設計理念が反映されている。狭義では、 kernel は利用可能ないくつかのハードウェア資源に対する programmatic interface を供給する。 UNIX システムでは、 kernel は device driver, memory management routines, scheduler, system call などを含むプログラムである。このプログラムはシステムが稼働している間常に走っている。

- **LAN**
  "Local Area Network"を見よ。

- **little-endian**
  最も重要でない byte (bit) が始めに送られるようなバイナリーデータの蓄積, 輸送の形式。 "big-endian" の項も見よ。

- **Local Area Network (LAN)**
  数平方キロメートルかそれ以下の領域を受け持つデータネットワーク。なぜならネットワークとは、今日では 10 Mbps から 100Mbps というレンジのデータレートが可能なネットワークシグナルプロトコルで optimization が作られるようなごく小さいエリアをカバーするものとして知られているからである。広い領域の交流は、 metropolitan area networks (MANs) 又は wide-area networks (WANs) を媒体として LANs を互いに結合することでできる。 Ethernet も FDDI も local area networks である。

- **LocalTalk**
  Apple Computer によって開発された local area network (LAN) プロトコル。このネットワークは、電話線のツイストペアケーブル上で動き、データレート 235Kbps を持つよう設計されている。全てのマッキントッシュコンピュータは LocalTalk インターフェイスを装備している。

- **Network Address**
  ネットワーク上のホストをユニークに特定する数、または数のグループ。例えば 128.89.1.178 は nnsc.nsf.net に対するネットワークアドレスである。非公式には 電子メイルアドレスのこともいう。例えば nnsc@nnsc.nsf.net は NSF Network Service Center (NNSC) のネットワークアドレスである。

- **Network File System (NFS)**
  NFS は、 Sun Microsystems で開発された、それがあたかもローカルディスクにあるかのようにネットワークを越えてファイルにアクセスすることをコンピュータシステムに許すプロトコルを記述している。このプロトコルは 200 以上の会社によって製品化され、事実上の標準となっている。

- **Node**
  ネットワークに取り付けられたコンピュータ。ホストとも呼ばれる。

- **NSF**
  "National Science Foundation"を見よ。

- **Open Software Foundation (OSF)**
  開かれたソフトウェア環境のための仕様を開発するために Digital, IBM, そして 4 つの他のベンダーが設立した、営利を目的としない組織。

- **Open Systems Interconnection (OSI)**
  コンピュータやネットワークと異なった、接続の方法の国際基準となるべく設計されたプロトコルのセット。ヨーロッパは OSI の開発をほとんど終らせたので、おそらく間もなく使用されるであろう。

- **OSF**
  "Open Software Foundation"を見よ。

- **OSI**
  "Open Systems Interconnection"を見よ。

- **Packet Assembler/Disassembler (PAD)**
  パケットネットワークにターミナルを接続するよう設計されたネットワークホスト。

- **packet**
  パケット交換網上を送られるデータの単位。この言葉は幅広く使われる。いくつかの Internet 文献は、物理的なネットワークを通して送られたデータを詳細に参照するためにこれを使う一方、他の文献は Internet をパケット交換網のように見ており、 IP データグラムをパケットのように記述する。

- **protocol**
  二つのコンピュータがメッセージの交換をする際、従わなければならない、メッセージのフォーマットとルールに関する正式な記述。プロトコルは、マシンマシンインターフェースの低いレベルの詳細の記述をすることもでき、プログラム間の高レベルの交換も記述できる。

- **public domain**
  無料で公開されている知的財産。大学で開発されたコンピュータソフトウェアの多くは public domain 上にある。

- **Remote Procedure Call (RPC)**
  分散コンピューティングの client-server model を実装する際に、簡単で良く使われている方法。一般には、供給された argument を使って指定された手順を実行するようリモートシステムにリクエストが送られ、結果が戻ってくる。 RPC には、相容れないいろいろな実装がある。

- **repeater**
  あるケーブルから他のケーブルへ routing の決定をしたり、パケットの filtering をしたりせ

ずに、電気信号を伝搬させる機器。OSI terminology では、 repeater はシステムを媒介する物理的な階層をいう。 bridge や router の項を参照のこと。 Ethernet 上では、 repeater は二つまたはそれ以上のケーブルの区画を互いに繋げるのに使われる。 repeater は、ある区画で受けとった信号をほかの全ての区画へ送る前に、タイミングの狂いを補正し、増幅しなおす。

- **Reverse Address Resolution Protocol (RARP)**
  ARP の逆引き機能を持っている TCP/IP プロトコル。 RARP は Internet アドレスに対して物理的な (ハードウェアの) アドレスを作成する。ディスクレスシステムが初期化する際、自分の Internet アドレスを探すためにしばしば使われる。

- **rlogin**
  Berkeley UNIX (BSD を参照) rcmd プロトコルを元にした、あるコンピュータ上のユーザに他のコンピュータへログインすることを許すプログラム。このプロトコルは UNIX マシンのあいだで広く用いられているが、 UNIX でないマシンももちろん使うことが出来る。

- **router**
  パケットのために使用可能なルートのなかから最も良いネットワーク経路を探すための機器。これは、 bridges, LAN routers や WAN gateways といった、異なった機器を結び付ける機器である。 Bridge は、物理的なネットワークから他へパケットを受け渡すかどうかを選ぶという、ルータの原始的な形態をとっている。 Router は必要に応じてパケットの分解と最構成を行なう。信頼性を保証するために、その特殊な目的のためにつくられたコンピュータを使用することが多い。 Router は Network Layer の OSI Level 3 上で稼働する。

- **server**
  資源の供給者。ネットワーク用語としては、 fileserver(クライアントマシンにファイルを供給する), nameserver(ホスト名のアドレスを供給したり、その逆を行なったりする) や他の多種多様な server がある。

- **Simple Mail Transfer Protocol (SMTP)**
  コンピュータ間で電子メイルを転送するのに使われる、 TCP/IP の系列のプロトコル。 SMTP メイルは RFC-822 に準拠している。 SMTP は二つのメイルシステムがどのように関係するのか、そしてそれらがメイル転送のために交換するコントロールメッセージのフォーマットについて記述している。これは client/server system 用に設計されてはいないので、その場合は他のプロトコル (POP を参照) が使われる。しかし、 server は他の server とメッセージを交換する必要がある場合普通は SMTP を使用する。

- **Simple Network Management Protocol (SNMP)**
  もともとは router や bridg のようなネットワーク機器に基づいて、 IP を扱うために開発されたプロトコルで、いまでは hubs, workstations, toasters, jukeboxes, etc. を結び付けるという用途にまで拡大している。 IPX と AppleTalk のための SNMP も開発中であり、広く用いられているプロトコルである。 RFC 1098 で定義されている。”CMIP” も参照のこと。

- **subnet**
  そこに繋がれている全てのノードがあちこち飛び移れるようにするネットワーク技術を表す言葉。言い替えると、 subnet のユーザは subnet 上の全ての他のノードにダイレクトに交信できる。 subnet は X.25, Ethernet, トークン リング, ISDN, 又は point-to-point link で可能である。 subnet の集団は、 routing や network layer でもってお互いに結合し、ネットワークを形作っている。

- **TCP**
  Transmission Control Protocol

- **TCP/IP**
  Internet Protocol 上の Transmission Control Protocol。 IP 上で走る一連のアプリケーションと転送プロトコルに関する共通のきまりである。 FTP, Telnet, SMTP, and UDP (a transport layer protocol) などを含んでいる。

- **TELNET**
  Telnet は ある場所のコンピュータユーザが他の場所のコンピュータで働けるようにするプログラムである。 remote terminal connection service のための Internet 標準プロトコル。 TELNET は、 Internet access つまり Internet につながっている TCP/IP ネットワーク上にいる、ということが必要である。 FTP や 電子メイルとは違って、 Telnet はリモートホストのプログラムやコマンドを実際に使うことができる。

- **terminal emulator**
  コンピュータが端末の真似することのできるようにするプログラム。ワークステーションはホストの端末のように見える。

- **terminal server**
  あるネットワーク結合を通して LAN に多くの terminal を接続する小さく特殊なネットワークコンピュータ。ネットワーク上のどのユーザも様々なネットワークホストに TELNET できる。 terminal server は非同期ポートで多くのユーザと接続することもできる。

- **TN3270**
  TELNET プログラムの変種で、あたかもユーザが 3270 もしくはにたような terminal を持っているかのように、 IBM mainframe と接続したり使ったりできる。

- **transceiver**
  Transmitter-receiver。 Ethernet のようなローカルエリアネットワークとホストを結び付ける器械。 Ethernet transceiver にはケーブルに信号を送ったり、その衝突を感じたりする機器が備わっている。

- **Twisted Pair**
  少なくとも家とオフィスの間のような距離を、電話を繋ぐのに電話会社によって使われているワイヤーのタイプ。それは二つの捻られた伝導線でできている。この捻れが重要で、他では出来ないような通信が可能になるような電気的な特徴をもたらしている。通常、電話線はシールドされていない。 (“Shielded twisted Pair” を参照)

- **UDP**
  "User Datagram Protocol"を見よ。

- **Universal Time (UT)**
  Greenwich Mean Time (GMT) である。

- **UNIX-to-UNIX CoPy (UUCP)**
  これは初期においては、 dial-up 方式で ある UNIX システムから他の UNIX システムへファイルを送ることを許す、 UNIX OS(BSD を参照) 下で走るプログラムだった。現在では一般的に、 netnews や電子メイルを送るための UUCP プロトコルを用いたマシン群で作られた、大きな国際ネットワークを表すのにこの語は使われている。

- **User Datagram Protocol (UDP)**
  Internet 上で使用される transport プロトコル。 TCP と同じように IP を配達に使用するが、 TCP と異なり 保証と承認なしにデータグラムの交換を行なう。

- **UT**
  "Universal Time"を見よ。

- **UUCP**
  "UNIX-to-UNIX CoPy"を見よ。

- **Wide Area Network (WAN)**
  通常 serial lines で構成されている大きな地理的領域をカバーするネットワーク。

- **World Wide Web (WWW)**
  より多くの学術情報が誰にでも自由に使えるべきであるという理念に基づいたプロジェクト。 WWW プロジェクトは情報検索の技術と簡単だがパワフルな global information system とを融合させている。

- **worm**
  自己増殖するコンピュータプログラム。 Internet worm がおそらくもっとも有名である。それは首尾良く (または突然に)Internet を渡ってシステム中で自己を複製する。

- **WWW**
  "World Wide Web"を見よ。

- **WYSIWYG**
  "What You See is What You Get"を見よ。

- **Xerox Networking System (XNS)**
  DECnet や TCP/IP に代わるものとして、 Xerox によって開発されたネットワーキングプロトコル仕様。

- **XNS**
  "Xerox Network System"を見よ。

- **Yellow Pages(YP)**
  "NIS"を見よ。

- C
  低レベルのプログラミング言語。

- FORTRAN
  非常に初期から用いられてきたプログラミング言語。科学計算ではいまだに広く使われている。

- LSE
  Language Sensitive Editor。 VMS のために DEC によって供給されたテキストエディター。

- LUN
  Logical Unit Number。 FORTRAN program からファイルを参照するための小さな正の integer。

- UNIX
  AT&T による OS。様々な亜種が多くの会社から発売され、コンピュータ上で稼働している。

- VM/CMS
  大きい IBM computer への媒体のための OS。

- VMS
  DEC VAX computer の OS。

あとがき

このテキストの最初の版を作って以来、4 年が経った。高エネルギー物理学で UNIX 計算機が使われるのは、ごく当たり前となり、当時と比べると隔世の感すらある。当時 UNIX をこの分野で使っていたのは、ごく小数の人達で、データ解析のために使っていたのは、ほぼ我々だけであった。現在では、ほとんどの実験グループが UNIX 計算機を用いて解析を行っている。 急速な状況の変化に戸惑っている人たちも多いであろう。そういう人たちに幾らかでも、このテキストが役に立てばと願う。

高エネルギー物理学の分野での計算機の使い方も急速に変化しつつあるが、プログラミング言語も、 FORTRAN から、 C++ への移行が CERN を中心に始まっている。 FORTRAN を使うユーザの比率が、ほかの言語を使うユーザに比べて下がり続けていることもあり、 FORTRAN ユーザにとって、 UNIX での開発環境の向上は、あまり望めなくなってきている。今後数年内に、我々のデータ解析の環境は、大きく変わって行くかもしれない。ソフトウエアを作ったり、計算機の環境を整える仕事は、素人が片手間にできる仕事と考えられがちであった。しかし、今後ますます複雑化するであろうソフトウエア、そして計算機システムを使いこなすには、 expert の養成が不可欠である。

新潟大学の高エネルギー研究室も、 4 年の間に学生も代替わりを繰り返したがが、このマニュアルのアップデートの作業は続けられてきた。今後いつまで続けられるかわからないが、できるだけ続ける努力をしたいと思う。いつの日か、 FORTRAN にかわり、 C++ でのプログラミングを取り入れた版ができるかもしれない。

このテキスト完成には、筆者に名前を連ねた人以外にも多くの人たちの協力があった。新潟大学の宮野氏と宮田氏、その多くの大学院生と卒研生、日本歯科大学の山下氏には、いろいろな面で多大な協力をして頂いた。特に山下さんには、最初の UNIX ワークステーションの立ち上げから、 CERN ライブラリや解析プログラムの移植まで、いろいろな面でご尽力頂いた。ここに名前の出てこない方も含めて、協力して戴いた全ての方々に感謝する。

参考文献

[1] 「UNIX C SHELL フィールドガイド」、パーソナルメディア

[2] Richard Stallman 著、竹内、天海訳 「GNU Emacs マニュアル」 共立出版

[3] 大木敦雄著、「入門 Mule」 アスキー出版局

[4] 松田・暦本著、「UNIX 日本語環境」 アスキー出版局

[5] SUN OS マニュアル 53SP-3849-1。「デバッギングツール説明書」

[6] “UNIX for VMS Users”, Philip E. Bourne, Digital Press, ISBN 1-55558-034-3

[7] 「楽しい UNIX」、坂本文著、アスキー出版

[8] 「続・楽しい UNIX」、坂本文著、アスキー出版

[9] 「Life with UNIX」アスキー出版

[10] 「UNIX SYSTEM ADMINISTRATION HANDBOOK」 Prentice Hall

[11] 「UNIX ユーティリティライブラリ make」啓学出版

[12] 「UNIX ツール ガイドブック」共立出版

[13] 「UNIX C プログラミング」アスキー出版

[14] 「UNIX システムコール・プロラミング」アスキー出版

[15] 「vi 入門」アスキー出版

[16] “Xlib Reference Manual”, O’Reilly & Associates, Inc.

[17] “Xlib Programming Manual”, O’Reilly & Associates, Inc.

[18] ”The TeXbook”、 ADDISON-WESLEY PUBLISHING,ISBN 0-201-13448-9

[19] 「文書処理システム LaTeX」、アスキー出版