Chapter 8

TUTORIAL FOR UNIX OPERATIONS

Akira Sawada, Hideyuki Usui and Masaki Okada

In the first week of the 4th International School for Space Simulation (ISSS-4), about 80 young scientists and students majoring space plasma gathered at Kyoto in Japan from all over the world to learn the basic concept of the space simulation and practice the simulation runs. For this tutorial course, various kinds of simulation code were voluntarily provided such as electromagnetic code, magnetohydrodynamic code and hybrid code. As hardware for the simulation runs, 50 UNIX workstations were prepared connected to three mini-supercomputers. Since the simulation runs were performed through the operation of these UNIX machines, the participants were required to have basic knowledge about UNIX operation. Therefore before starting the lecture and practice of the space simulation, tutorials of the basic operation of UNIX workstation, screen editor 'vi', and compilation of the simulation codes were given. The texts used for each tutorial are as follows.

8.1 Basic Operations in UNIX Operating System

8.1.1 Computer environments in ISSS–4

In the ISSS–4 tutorial course (first week), 50 desktop workstations (**OMRON LUNA**) and three super-graphic computers (**Stardent TITAN**) are available for exercises of several simulation codes. These computers run under the **UNIX operating system** (System V + BSD extension). All computers are connected each other by thin **Ethernet** wires using **TCP/IP** protocols. (The data transfer rate is 10 Mbit/sec.) For each two students, one LUNA computer is prepared on their desk. The LUNAs are devoted to terminals of TITANs using the **X Window System**, that is, you enter the TITANs from the LUNAs through the Ethernet network. Test runs of the simulation codes are performed on the TITANs.

The X Window System is a very popular window system on workstations and runs effectively under network environment. Diagnostic programs of your simulation results should be run on the TITANs and the graphic outputs can be seen on your LUNAs using the X Window System (even if machine architectures are much different between the TITAN and the LUNA).

Furthermore, one SONY NEWS workstation is prepared for a printer server to get printed graphic output of the diagnostics.

8.1.2 How to login/logout

First of all, you need to enter an interactive mode with the UNIX operating system, that is, to **login** into a host computer. To login into your LUNA, you input your **user name** for a login prompt

```
login:
```

In the ISSS 4, a common user name '`isss`' is used for every students, so you type '`isss`'(and press a RETURN key);

```
login: isss
Password:
```

and then the LUNA requires a **password** for the user '`isss`'. If you mistype the password, the system shows a login prompt again;

```
login: isss
Password:
Login incorrect
login:
```

Try again to input the user name and the password.

If you login into your LUNA successfully, the screen is cleared and the X Window System is automatically invoked. In a right half of your display, a large window is opened for an **xterm** (a terminal emulator on the X Window). The xterm is setupped to remote-login (login into a host computer from another computer through the network) into a TITAN at the beginning. In the ISSS 4, we can use three TITANs and the TITAN which you login into is selected by a host name of the LUNA you first login into. When you can see a prompt like

```
isss01@titan1[1]
```

all the login process is completed successfully.

To exit from the UNIX operating system, you have to **logout** from the host;

```
isss01@titan1[1] logout
```

or

```
isss01@titan1[1] exit
```

Exercise 1–1. Login into your workstation. When you login, use commands 'hostname' and 'whoami' to get a host name and a user name you are using now, respectively. You can see a relationship between the results of the above commands and the system prompt you can see on your xterm.

8.1.3 Operations for directories and listing the contents

In the UNIX operating system, the file system forms a tree. There are two types of file in the file system; one is a **regular file** which stores ordinary data, the other is a **directory file**. The directory file stores information of the files (they may be regular files as well as directory files) which are located under the directory.

The top of the tree of the file system is called the **root directory** and indicated by a **slash** ('/'). The slash is also used as a punctuation of each directories. The directory where a user is working is called a **current directory**. The current directory when you login into a host is a **home directory**. To get a name of the current directory, type

```
isss01@titan1[2] pwd
```

'pwd' is an abbreviation of 'Print Working Directory'. When you are just after the login, the 'pwd' command shows

```
isss01@titan1[2] pwd
/usr/isss/home/isss01
```

and you can see '/usr/isss/home/isss01' as your home directory.

Now, to list the contents of your home directory, you use a 'ls' (LiSt) command;

```
isss01@titan1[3] ls
./              .exrc           .uwmrc          kempo1/      tutorial/
../             .history        .xinitrc        mhd-2/
.cshrc          .login          hybrid-1/       tristan/
```

The file whose name is terminated with a '/' character in the outputs of the 'ls' command is a directory file. There are 6 files starting by a character '.' and 7 directories ('.', '..', 'hybrid-1', 'kempo1', 'mhd-2', 'tristan' and 'tutorial'). The directories '.' and '..' have special meanings; '.' indicates the current directory itself and '..' does a upper directory (a parent directory).

Using 'ls -l', you can get a detail information of the current directory;

```
isss01@titan1[4] ls -l
total 10
drwxrwxr-x  7 isss01   isss      512 Mar 21 02:26 ./
drwxr-xr-x  5 isssadm  isss     4096 Mar 21 02:25 ../
- - -rw-rw-r--  1 isssadm  isss        0 Mar 21 02:25 .cshrc
```

```
- - -rw-rw-r--  1 isssadm  isss        0 Mar 21 02:25 .exrc
- - -rw-rw-r--  1 isss01   isss        0 Mar 21 02:25 .history
- - -rw-rw-r--  1 isssadm  isss        0 Mar 21 02:25 .login
- - -rw-rw-r--  1 isssadm  isss        0 Mar 21 02:26 .uwmrc
- - -rw-rw-r--  1 isssadm  isss        0 Mar 21 02:26 .xinitrc
drwxrwxr-x  2 isss01   isss      512 Mar 21 02:26 hybrid-1/
drwxrwxr-x  2 isss01   isss      512 Mar 21 02:26 kempo1/
drwxrwxr-x  2 isss01   isss      512 Mar 21 02:26 mhd-2/
drwxrwxr-x  2 isss01   isss      512 Mar 21 02:26 tristan/
drwxrwxr-x  5 isss01   isss      512 Mar 21 02:37 tutorial/
```

The '`ls`' command can take a directory name as an argument to see the contents of the directory;

```
isss01@titan1[5] ls tutorial
./      ../     1/      2/      3/
```

In the tutorial directory, there are three directories that are '1', '2' and '3' and two special directories that are '.' and '..' as usual.

Exercise 1–2. Try '`pwd`', '`ls`' and '`ls -l`' in your home directory to see a location of your home directory and a list of the contents.

Exercise 1–3. Check the contents of each directories under your home directory using '`ls`' command with arguments.

You can change the current directory by a '`cd`' (Change Directory) command;

```
isss01@titan1[6] cd tutorial
isss01@titan1[7] pwd
/usr/isss/home/isss01/tutorial
```

In this case, you go down into the directory '`tutorial`'. To go back to the home directory which is an upper directory from the current directory, you use the '`cd`' command with '..' as an argument. (Remember that the directory '..' indicates the parent directory.)

```
isss01@titan1[8] cd ..
isss01@titan1[9] pwd
/usr/isss/home/isss01
```

Next, you go down into a directory '1' which is located under the directory '`tutorial`';

```
isss01@titan1[10] cd tutorial/1
isss01@titan1[11] pwd
/usr/isss/home/isss01/tutorial/1
```

Now, use the '`cd`' command without any arguments;

```
isss01@titan1[12] cd
isss01@titan1[13] pwd
/usr/isss/home/isss01
```

It lets the current directory go back to your home directory.

Exercise 1–4. Move around each directories under your home directory using the 'cd' command and check the contents of each directories using '`ls`' or '`ls -l`'.

To make a new directory or remove an old one, you can use '`mkdir`' (MaKe a DIRectory) and '`rmdir`' (ReMove a DIRectory), respectively. In the both commands, the target directory must be specified by an argument of the command. (Note: '`rmdir`' requires that the target directory is empty.)

8.1.4 Operations for regular files

Basic operations for regular files are to copy, to move and to remove. They correspond to '`cp`' (CoPy), '`mv` (MoVe)', and '`rm`' (ReMove) commands, respectively.

The '`cp`' command is used in a form '`cp` *source destination*' basically and it requires two arguments at least. To copy a file '`test`' in a directory '`from`' into the current directory, you use '`cp`' command like

```
isss01@titan1[14] cp from/test .
```

In this case, the new file is also named '`test`'. If the destination is a directory, the '`cp`' command allows you to specify files as the sources. For example, you have two directories named '`from`' and '`to`'. Use the '`cp`' command like

```
isss01@titan1[15] cp from/* to
```

and all the regular files in the directory '`from`' are copied into the directory '`to`'. (Strictly speaking, source files whose name starts with a letter '`.`' are not copied.) This is a mechanism so-called **wild cards** that the '`*`' is expanded to all files in the directory except some files starting with a letter '`.`'.

The '`mv`' command is also used in a form '`mv` *source destination*'. To move a file is equivalent to changing the file name, so that it is processed much faster than copying a file.

The '`rm`' is a command to remove a file and the target files are specified in the arguments.

8.2 Screen Editor (vi) Tutorial

This is a sequence of lessons in the use of the text-editing program called *'vi'* which stands for 'Visual'. The *vi* editor allows you to move the cursor to any point on the screen or in the file and create, change, or delete text from that point.

8.2.1 Basic commands

In this section, you will learn the basic commands you need from opening to closing a file. To edit a file you must be able to move the cursor, add, change, and delete text.

- **How to Open File**
  Type 'vi' and the name of the file you want to edit.

```
vi [filename]
```

  Then *vi* clears the screen and displays a window in which you can edit text.

- **How to Move the Cursor**
  To edit your text, you need to move the cursor to the point on the screen where you will begin the correction. This is easily done with four keys that are grouped together on the keyboard: 'h', 'j', 'k', and 'l'.

h .	Move the cursor one character to the left.
j .	Move the cursor down one line.
k .	Move the cursor up one line.
l .	Move the cursor one character to the right.

  Note: You may also use the arrow keys in the same way as the 'h', 'j', 'k' and 'l' commands.

- **How to Delete Text**
  If you want to delete a character, move the cursor to that character and press 'x'. To delete a line, type 'dd'. To delete multiple lines, specify the number before the command. If you delete something by mistake, type 'u'; it reappears on the screen.

x ...	Delete one character.
dd ..	Delete one line.
n dd .	Delete *n* lines.
u ...	Undo the last command.

- **How to Add Text**
  There are two basic commands for adding text: the insert('i') and append('a') commands. To add text with insert command at a point in your file, move the cursor to the point by using 'h', 'j', 'k', and 'l'. Then press

'i' and start entering text. The 'a' allows you to append text after the cursor. The *vi* editor continues to accept the characters you type until you press the 'ESC' key.

i .	Insert text before the character the cursor is on.
I .	Insert text at the beginning of the current line.
a .	Append text after the current character.
A .	Append text at the end of the current line.
o/O	Open a new line after/before the current line.

- **How to Quit Vi**
  When you have finished your text, you will want to save it and return to the shell. You can use the ':w' and ':q' commands of the line editor for saving and quitting file. (Line editor commands begin with a colon and appear on the bottom line of the screen.) The ':w' command saves a file. The ':q' command leaves the editor and returns to the shell. You can type these commands separately or combine them into the single command ':wq'.

:w	Save a file.
:q	Quit *vi*.
:wq	Save a file and quit *vi*.
:q!	Quit *vi* without saving the edited text to a file.
:w *newfile*	Save text to *newfile*

8.2.2 Positioning the cursor in undisplayed text

Basically 'h', 'j', 'k', and 'l' are enough to move a cursor in the current editing window. However, if you are editing a large file, you need to move the cursor to text that is not shown in the current window. The following ways are used to move around within the file:

- **By scrolling forward or backward in the file.**

^f (Ctrl-f) ..	Page forward one full screen.
^d (Ctrl-d) ..	Page forward one half screen.
^b (Ctrl-b) ..	Page backward one full screen.
^u (Ctrl-u) ..	Page backward one half screen.

- **By going a specified line in the file.**

`G` .	Go to the last line of the file.
n`G` .	Go to the *n*th line of the file.

- **By searching for a pattern in the file.**

`/`*pattern* .	Search forward for the next occurrence of the *pattern*.
`?`*pattern* .	Search backward for the first occurrence of the *pattern*.
`n`	Repeat the last search command.

8.2.3 Modifying text

The delete commands and text input commands provide one way for you to modify text. Another way you can change text is by using commands that let you replace and substitute text.

- **By replacing text.**

`cc` .	Replace all the characters in the line.
`r` .	Replace the current character. No need to be followed by pressing '`ESC`' key.
`R` .	Replace characters typed over until '`ESC`' key is pressed.

- **By substituting text.**

`s` .	Delete the character under the cursor and place you in text input mode.
`S` .	Delete the entire line and place you in text input mode.

8.2.4 Yanking and putting text

vi provides a set of commands that cut and paste text in a file. Use command '`yy`' to yank line(s) and paste it by command '`p`'

`yy` .	Yank a copy of the current line.
n`yy`	Yank *n* lines.
`p` .	Put the most-recently deleted or yanked text
..	after the current position.

8.2.5 Other commands

`J .`	Joint the line below the current line with the current line.
`^l .`	Clear and redraw the screen.
`. .`	Repeat the last command.
`Esc`	Whenever you are not sure what state *vi* is in, hit this key a few times until it beeps.

This concludes the basic tutorial of *vi*. There are other useful commands. In this school, however, you are not required to know more than the commands stated so far. The recommendation at this point is that you get used to using *vi* on a regular basis.

8.3 How to Compile Fortran Programs

All the source programs are distributed with Fortran77 source file format. You will find Fortran source files and one Makefile in each directory ``hybrid-1'', ``kempo1'', ``macroEM'', ``mhd-2'', ``tristan'' and ``tutorial''. The files with `.f` extension are the Fortran source files. These Fortran source files can be compiled by a command `fc`[1] or `f77`. In order to minimize the CPU load, we compile each subroutines separately. `fc` command creates object module with extension `.o` from a source file `.f`. After all source files are compiled, you have to link all the modules to make an executable file. These compiling and linking steps are so complicated that you had better use a command `make`. `Make` command refers the `Makefile`, compiles the source files only which you need to compile and links the object modules automatically. Makefile is a file which describes how to make a target command, i.e., Makefile describes the dependencies of the target file, object modules and source programs. We register all the source files, object modules, executable files and their dependencies to the `Makefile` in advance. For instance, you can make an executable file `tut1` by typing.

```
make tut1 ..... (a)
```

It will take a few minutes to compile all the source programs and link. Then you will find an executable file `tut1` in the directory. A symbol $*$ after the file name indicates that the file `tut1` is the executable format. If you found some error messages on the screen during executing `make` command, you need to debug the source program by the editor. If you didn't find any error messages, you can execute the file `tut1` by typing,

[1]Command name depends on the system. In this ISSS-4 tutorial course, `fc` is used on titan1 and titan2, f77 on titan3. As for the TRISTAN, we use `xy77` command.

```
tut1 ..................................................... (b)
```

Then you will see the results on your display. If you want to change or debug a source program with `vi`, type,

```
vi (file name) ........................................... (c)
```

Hereafter, you can remake the executable file, execute and modify with these commands (a), (b) and (c), respectively.

In this ISSS-4 tutorial course, we have prepared an appropriate `Makefile` for each tutorial materials beforehand. All you have to do is to make, execute and edit. Change directory to the exercise directory `$HOME/tutorial/3` and execute the following operations.

Exercise 3–1. Compile and link all the source files by a following command.

```
fc main.f sub1.f sub2.f -L/usr/isss/lib -loption
-lcX11 -lX11
```

Execute a command `a.out`, which you will get with the previous `fc` command.

Exercise 3–2. Make an executable command tut1 and tut2 by typing a command `make`. Tut1 is a sample program, which outputs a sample graphic data directly to your display. Tut2 outputs to a graphic data file named `cgraph.dat`. Execute the previous two commands `tut1` and `tut2`.

8.4 What Are `make` and `Makefile`

The make command allows the programmer to maintain, update and regenerate groups of computer programs. Makefile contains a sequence of entries that specify dependencies of the target files. The first line of an entry is a blank-separated, non-null list of targets, then a ":", then a (possibly null) list of prerequisite files or dependencies. Text following a ";" and all following lines that begin with a "[tab]" are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the "\[return]" sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. The following `Makefile` says that tut1 depends on three files main.o, sub1.o and sub2.o, and that they in turn depend on the their corresponding source files, main.f sub1.f and sub2.f.

```
tut1:   main.o sub1.o sub2.o
[tab]   fc main.o sub1.o sub2.o -o tut1
main.o: main.f
[tab]   fc -c main.f
sub1.o: sub1.f
[tab]   fc -c sub1.f
sub2.o: sub2.f
```

```
[tab]   fc -c sub2.f
```

Entries with the form `string1 = string2` are macro definitions. String2 is defined as all characters up to a comment character or an unescaped newline. The macro definition `OBJS=main.o sub1.o sub2.o` allows you to simplify the previous example as follows.

```
OBJS    = main.o sub1.o sub2.o
tut1:   ${OBJS}
[tab]   fc ${OBJS} -o tut1
main.o: main.f
[tab]   fc -c main.f
sub1.o: sub1.f
[tab]   fc -c sub1.f
sub2.o: sub2.f
[tab]   fc -c sub2.f
```

When the dependencies between an object module and a corresponding source file does not listed in the Makefile, the command `make` searches fortran source program whose file name has the same main part as the object module and an extension `.f` in order to make the object module with `.o`. We can reduce the `Makefile` as following.

```
OBJS    = main.o sub1.o sub2.o
tut1:   ${OBJS}
[tab]   fc ${OBJS} -o tut1
```

For convenience, an entry `clean` is added at the end of the Makefile. This entry removes all the object modules, executable files and graphic data. If you want to re-compile all source programs, type `make clean`.

Exercise 4–1. Edit and change letters "ISSS-4 " to your own name in the sample subroutine `sub2.f`. Remake the command `tut1`. Execute the command `tut1`.

Exercise 4–2. Edit and reduce the `Makefile` as the third sample `Makefile`. **Do not** delete the entry `clean`. Remove all the object modules by typing `make clean`, and check whether the `make` command is executed properly.

8.5 What IS X Window System

The X Window System is a network-transparent window system which runs under a number of operating systems. It was developed at MIT. The X Window system supports overlapping windows, fully recursive subwindows, and both text and graphics operations within the windows.

The LUNA in front of you and the host computer correspond to the X server machines and client, respectively. The host computer displays the graphic data on your LUNA's display. You have to indicate which server now you are using

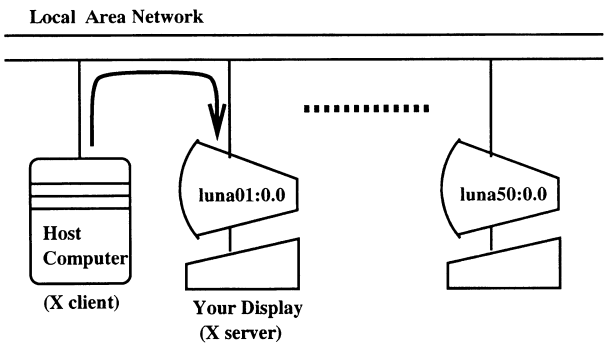

Figure 8.1: Relation between X window servers and client.

by setting the environment variable "DISPLAY". This environment variable has already set in the initializing process when you first logged in. You can check your environment variables by the command `env`.

In this ISSS-4 tutorial course, there are three graphic supercomputers, which are named titan1, titan2 and titan3. One host supercomputer is shared with 6 to 24 LUNAs. All graphic terminals, X window servers, are named `luna01` to `luna50`. User identification name is determined from the terminal name. The user name is from `isss01` to `isss50`. The one who uses the terminal from luna01 to luna06 logs in `titan1`. The one who uses the terminal from luna07 to luna18 logs in `titan2`. The others logs in `titan3`. The host name that you are using and your user identification name is found in the prompt. Such as,

```
isss01@titan1[1]
```

You can get a graphic data in two different ways. One is to put out directly to your display and the other is to save in the data file `cgraph.dat`. You can replay the graphic data file on your display by typing `repx11`. When you display the graphic data, one graphic window is opened in the upper-left corner on the display. The size and the location of the graphic window is determined by an environment variable "GEOMETRY". The default value of the "GEOMETRY" is "1000x800+0+0". This means that the window has 1000 and 800 pixels in X and Y direction, respectively and that the upper-left corner is the origin of the graphic window.

Exercise 5–1. Check your environment variables by typing `env`. Change the size and location of the graphic window by a command `setenv GEOMETRY 500x400-0+0`.

Exercise 5–2. Check the terminal name next to yours. Set the name to the environment variable "DISPLAY " such as `setenv DISPLAY luna??:0.0`.

Execute the command `tut1`. **Note that you must reset the variable `DISPLAY` to your own terminal name, when you finished this exercise.**

8.6 How to Make Your Own Graphic Library

Graphic library is necessary for viewing the simulation results. But computer environment of the simulationists is not always the same and graphical environment is highly depend on the computer system you use. All simulation codes listed in this book are using very well-known graphic interface, CALCOMP. For convenience of most simulationists, we listed whole source program of graphic CALCOMP–X interface library in appendix. We briefly describe how to make your own graphical interface using CALCOMP standard.

CALCOMP interface has originally introduced for pen plotter device. The library consists of six subroutines Function of each subroutine is so primitive that any graphic devices we would have nowadays are able to achieve its function very easily. In order to get graphic results of our simulation codes, Six subroutines we need to implement are listed in the Table 8.1

Table 8.1: List of subroutine calls used in the simulation codes.

CALL PLOTS	Initializes graphic device.
CALL PLOT(x, y, ipen)	Moves pen down/up. ipen= 2/3 Closes device. ipen=999.
CALL CHART	Advances to a new page.
CALL NEWPEN(npen)	Changes thickness of the line.
CALL FACTOR(fact)	Changes scaling factor.
CALL WHERE(x, y)	Returns current position of the pen.

In subroutine 'PLOTS', it connects to ther server and creates a new window whose geometry is 300x250+0+0. After initializing the background color of the window, it waits for the window to be mapped.

Subroutine 'PLOT' draws a line from previous pen position to the current position if ipen equals to 2 or -2. Previous position is sotred in variables `xold` and `yold`. If this routine is called with ipen equals to 3 or -3, it just put the current pen position to variables `xold` and `yold`. You can move the origin of the coordinate by giving a negative value to the ipen argument. If 999 is given to the argument, it destorys the window and close the connection with the X window server.

The third program 'CHART' flushes all the buffer, and waits for the user to press one of the three mouse buttons. After receiving the ButtonPress event from the X server, it clears the window and return to the main program.

Program 'NEWPEN' changes the thickness of the line by calling Xlib module `XSetLineAttributes`. If you are using color X server and want to change the color of the line, you can replace `XSetLineAttributes` with `XSetForeground`. Colormap should be defined in order to use colors in 'PLOTS' beforehand.

With CALCOMP interface, coordinate system is defined at the head of the program. Page size is defined as 34.5cm wide × 36.5cm height for compatibility. You can change the scaling factor by calling program 'FACTOR'. 1.0 is set to the variable `factor` as default.

Program 'WHERE' returns current position of the pen in arguments `x` and `y`.

These subroutines can be designed so easily that you can implement your own graphic device driver by yourself not only for X Window System, but for any kinds of graphic devices. For example, a graphic library which generates PostScript file would be very useful for many simulationists.

Appendix

```
/*
 *    Calcomp - X Window Protocol converter.
 *    Programed by Masaki Okada
 *    (Radio Atmospheric Science Center, Kyoto University)
 *    $Revision: 1.2 $
 *    $Date: 1993/03/22 11:54:29 $
 */

#include <stdio.h>
#include <X11/Xlib.h>
#define HEIGHT  250
#define WIDTH   300
#define RHEIGHT 26.5
#define RWIDTH  34.5
#define DISPLAY "unix7:0.0"

Display *display;
int screen;
Window  win;
GC  gc;

int xold, yold;
float   xorg, yorg, xscale, yscale, factor = 1.0;
unsigned int    page = 1;

void    plots_()
{
    XEvent  event;

    display = XOpenDisplay(DISPLAY);
    screen = DefaultScreen(display);
    win = XCreateSimpleWindow(display,
        RootWindow(display, screen), 0, 0, WIDTH, HEIGHT, 1,
        BlackPixel(display, screen),
        BlackPixel(display, screen));
    xscale = WIDTH  / RWIDTH;
    yscale = HEIGHT / RHEIGHT;
    gc = DefaultGC(display, screen);
    XSetForeground(display, gc, WhitePixel(display, screen));
    XSelectInput(display, win,
        ExposureMask | StructureNotifyMask);
```

```
    XMapWindow(display, win);
    XNextEvent(display, &event);
    XSelectInput(display, win,
        ButtonPressMask | ButtonReleaseMask);
}

void    plot_(x, y, ipen)
float   *x, *y;
int *ipen;
{
    int x1, y1, x2, y2;
    XEvent  event;

    switch (abs(*ipen)) {
    case 2:
        x1 = xold * xscale * factor;
        y1 = HEIGHT - yold * yscale * factor;
        x2 = (xorg + *x) * xscale * factor;
        y2 = HEIGHT - (yorg + *y) * yscale * factor;
        XDrawLine(display, win, gc, x1, y1, x2, y2);
    case 3:
        xold = xorg + *x;
        yold = yorg + *y;
        break;
    case 999:
        XFlush(display);
        fputs("Last Page\n",stderr);
        while (1) {
            XNextEvent(display, &event);
            if ( event.type == ButtonPress ) break;
        }
        XDestroyWindow(display, win);
        XCloseDisplay(display);
    }
    if ( *ipen < 0 ) {
        xorg += *x;
        yorg += *y;
    }
}

void    chart_()
{
    XEvent  event;
```

```
    XFlush(display);
    fprintf(stderr, "Page %d\n", page);
    page ++;
    while (1) {
        XNextEvent(display, &event);
        if ( event.type == ButtonPress ) break;
    }
    XClearWindow(display, win);
}

void factor_(f)
float   *f;
{
    factor = *f;
}

void newpen_(ipen)
int *ipen;
{
    unsigned int    line_width;

    line_width = *ipen;
    XSetLineAttributes(display, gc, line_width,
        LineSolid, CapRound, JoinRound);
}

void where_(x, y)
float   *x, *y;
{
    *x = xold * xscale;
    *y = yold * yscale;
}
```

References

Nye, A., *The Definitive Guides to the X Window System, Volume One, Xlib Programming Manual for Version 11*, O'Reilly & Associates, Inc., 1990.

Nye, A., *The Definitive Guides to the X Window System, Volume Two, Xlib Reference Manual for Version 11*, O'Reilly & Associates, Inc., 1990.

William Joy and Mark Horton, An Introduction to Display Editing with Vi, *UNIX Programmer's manual Vol. 2c*, Sep. 1980.