IISEE Lecture on Seismology and Tsunami Course

Computer - UNIX -

Lecturer
Yushiro Fujii

(This lecture was originally written by Y. Yagi and modified by Y. Fujii)

Why UNIX?

- Weakness
  - UNIX is difficult to use; it has cryptic commands and its interface is non-intuitive.
- Advantage
  - UNIX is stable, flexible, and powerful for ***multiple users*** and ***multitasking***. Many packages and libraries for seismology and geophysics have been developed in the UNIX system (e.g., *win*, *sac*, *GMT*, and *waveform inversion program*).

In this lecture, you will learn how to make use of UNIX's command and graphical tools, and familiarize yourself with the commands you can productively use.

Preparation for Lecture

Go to home directory

```
$ cd
```

Make “UNIX” directory

$ mkdir UNIX (***We use this directory for today’s lecture.***)

Open editor

```
$ cd UNIX
$ nedit &
```

Make a simple FORTRAN code like

```
(7 spaces) real a, b, c
      a=1.0
      b=2.0
      c=a+b
      write(6,*) 'c=',c
      stop
      end
```

and save as “program.f”

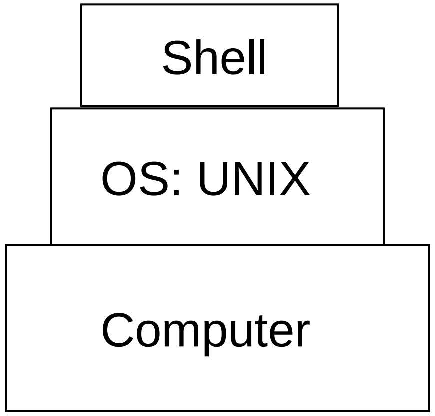
Confirm the program can be compiled and run.

```
$ g77 program.f
$ ./a.exe
  c=  3.
```

UNIX Shell

- Once you login, you are working with a program called a shell.
  - default shell for Cygwin, Linux (earth2, ocean): **bash**
  - default shell for SUN: **csh**
- The differences between the shells are slight. In this lecture, we will program the **csh** script.
- Shell prompt:

  Shell prompts usually contain $ or %.



Syntax of UNIX Commands

- UNIX commands are simple one-word entries such as the “ls” command. They can be more complex using various options. The general format for a UNIX command is

  *$ **command** (options) (file-names)*

  (e.g., *$ g77 –o program program.f*: you can find the program.exe file in the working directory).

Options

- Options modify the way in which a command works.
  - Syntax
    - Options are often single letters prefixed with a dash (-).
    - Multiple options in a single command line can be set off individually (-l -a) (in some cases, you can combine them after a single dash (like -la)).

```
$ ls  -l  -a
$ ls  -la
```

Unresponsive Terminal

- In case your terminal does not respond to a command (hung or frozen), please type (Ctrl)-c. The process is killed, and you will get a new command prompt.

UNIX File System

- Like other systems, a file is the unit of storage in UNIX. Files are organized into directories (folders). A directory is a special kind of file where the system stores information about other files.
- A directory is a place where files are said to be contained, and you are said to be working inside a directory.
  - If you want to check the working directory (Where am I?),

    Type

    $ pwd

    P.56, 57, Unix in 24 Hours

    Cygwin;

    /cygdrive/c/Documents and Settings/(user ID)/My Documents

    or

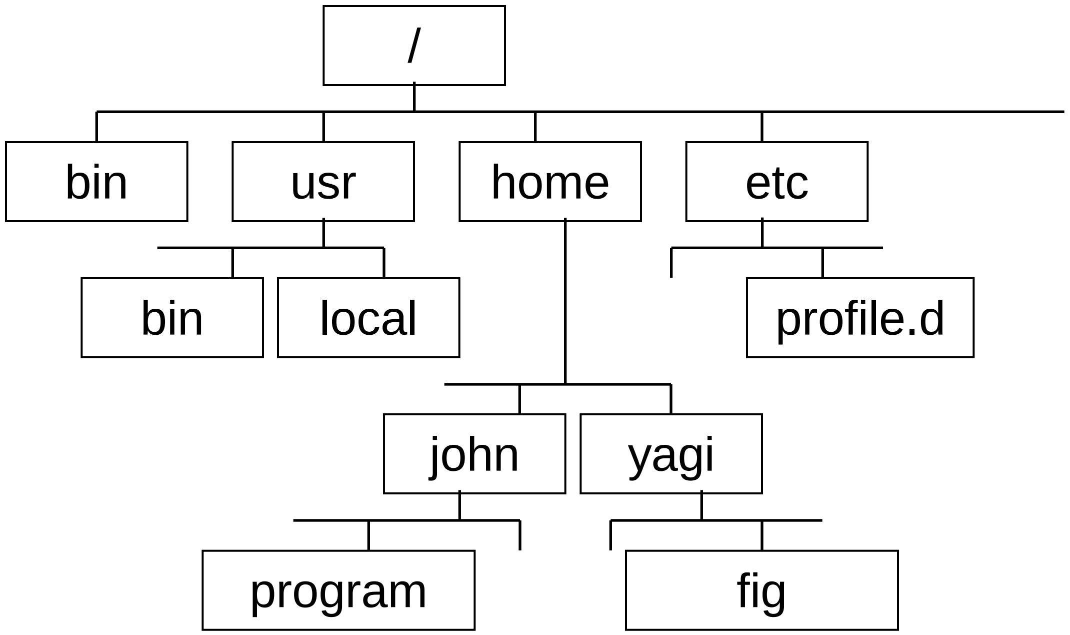
    /home/(user ID)

    Linux (earth2 or ocean):

    /home/(user ID)

Directory Tree

- All directories in a UNIX system are organized into a hierarchical structure, which is similar to a family tree. The parent directory is the root directory and is written as a forward slash (/).



Example of a directory tree.

In this case, the user directories are "/home/yagi" or "/home/john".

The *directory* named "bin" contain some programs.

In some cases, you cannot change the file without accessing your home directory.

Absolute and Relative Paths

Absolute path

An absolute pathname signifies the path of the directories you need to travel to get from the root to the required directory or file. In the path name, put slashes (/) between the directory names.

e.g., */home/yagi/tmp.ps*

- the root is the first “/”
- the directory “home” is a subdirectory of root “/”
- the directory “yagi” is a subdirectory of “home”
- the file of tmp.ps is located in the directory “/home/yagi/”

Type

```
$ pwd
/home/yagi
```

You can get the absolute pathname of the working directory.

Absolute and Relative Paths

Relative path

A *relative pathname* gives the location relative to your working directory. Unless you use an *absolute pathname (starting with a slash)*, UNIX assumes that you are using a relative pathname.

P.56, 57, Unix in 24 Hours

e.g.,

go to home directory

```
$ cd
```

go to “UNIX” directory

```
$ cd  UNIX
```

Make “work” directory

```
$ mkdir  work
```

P.110 - 112

go to “work” directory

```
$ cd  work
```

You can go up the tree by using the shorthand “..” for the parent directory. Evidently, you can also go down the tree by using the subdirectory name.

go up the tree

```
$ cd  ..
```

go up the tree and go down the directory

```
$ cd  ../UNIX/work
```

Listing Files

The “**ls**” command lists the entries in the directory tree. The syntax is

$ ls (*options*) (*directory-and-filename*)

Type

$ ls

P.50 - 54, 62 - 77, 86 - 93, Unix in 24 Hours

You can get the filenames in the working directory.

To get more information about each file, type

```
$ ls  -al
total 104
d rwxr-xr-x    6          yagi  none      0   Sep 19 10:28  .
d rwxr-xr-x    4          yagi  none      0   Sep 19 10:28  ..
- rwxr-xr-x    1          yagi  none 1999   Sep 23 15:40  a.out
- rw-r--r--    1          yagi  none   256   Sep 23 15:40  program.f
d rw-r-xr-x    1          yagi  none      0   Sep 23 15:40  work
```

type | access modes | # of links | owner | group | size | modification date and time | name

Listing Files

Type

Tells whether the file is a directory “d” or a plain file “-”

Access modes

Specifies three types of users (yourself “u”, your group “g”, and all others “o”) who are allowed to read “r”, write “w”, or execute “x” files.

If you want to edit the access modes, the “**chmod**” command is useful.

(e.g., type

$ chmod u-x (file-name)

You cannot execute the “*file-name*”

$ chmod u+x (file-name)

You can again execute “*file-name*”

P.94 - 102, Unix in 24 Hours

Size (in bytes)

Size of the file

Modification date

Date when the file was last modified.

Listing Files

If you give the pathname to a directory, “ls” will list the directory, but it will not change your working directory.

$ ls /usr/local/

bin doc etc include lib man share

-F and --color options are useful for detecting the file type.

$ ls –F

a.out program.f work/*

“/” at the end of each directory name. Files with an execute status “x”, like programs, are marked with “*”.

$ ls --color

a.exe program.f work

A green filename signifies the execute status, and a blue filename signifies the directory name.

Looking Inside Files

The “**cat**” command lets you move forward in the files by any number of pages or lines. The syntax is

$ cat (files)

P.134 - 136, Unix in 24 Hours

……………………………………………

………Inside file…………………

……………………………………………

“cat” works for short files containing characters that can be displayed on a single screen or less. You cannot go back to view the previous screens.

Looking Inside File

The “**less**” and “**more**” commands let you move forward in the files by any number of pages or lines. The syntax is

P.136 - 140, Unix in 24 Hours

```
$ more  (files)
::::::::::::::::::::::::::::
(file-name)
::::::::::::::::::::::::::::
………Inside file…………………
::::::::::::::::::::::::::::
(file-name)
::::::::::::::::::::::::::::
………Inside file…………………
------More----(50%)
```

The prompt says that you are 50% of your way through the file.

Looking Inside Files

If you type “h” at the “more” prompt, you can get the useful “more” commands on your system.

Typical useful “more” commands are as follows:

Command	**Description**
(SPACE)	Display next page
z	Display next page
(ENTER)	Display next page
d or (Ctrl)-D	Scroll k lines (k depends on your system)
b or (Ctrl)-B	Skip backward k screens of text
q	End reading file

Managing Your Files

The previous lecture dealt with formulating FORTRAN programs; for separating the programming files with the others files, you can create a programming directory using the "**mkdir**" command.

P.110 - 112, Unix in 24 Hours

```
$ mkdir  (new-directory-names)
```

Before editing a file, you can save a copy using the "**cp**" command.

P.112 - 114

```
$  cp  (old-file-name)  (new-file-name)
```

e.g.,

```
$ cp   program.f   program2.f
$ cp   program2.f   program3.f
```

If you want to put a copy of the file into a subdirectory, use

```
$  cp  (file-name)   (subdirectory-name)
```

e.g.,

```
$ cp   program.f   work
```

Managing Your Files

You can change the filename using the “**mv**” command. Type

$ mv (old-file-name) (new-file-name)

If you want to put a file into a subdirectory, use

$ mv (file-name) (subdirectory-name)

If you do not want to overwrite any old files, use “-i” option for safety. The syntax is

$ mv -i (old-file-name) (new-file-name)

P.114 - 116, Unix in 24 Hours

Managing Your Files

You can remove files using the “**rm**” command. Type

$ rm (file-name)

Please use “-i” option for safety.

$ rm -i (file-name)

P.117 - 121, Unix in 24 Hours

e.g., *$ rm -i program2.f*

(press “n” for not deleting a file)

or use alias (like a shortcut in Windows)

$ alias rm=’rm -i’

$ alias mv=’mv -i’

$ alias cp=’cp -i’

P.294 - 296

You can remove directories using the “**rmdir**” command. Type

$ rmdir (*directory-name*)

The directory has to be empty before it is deleted.

P.116 - 117

Changing your Environment

On a UNIX system, you can change the environment. When you log into a UNIX computer, csh refers to the “/home/(*user-name*)/.cshrc” file that contains information about the environment, and bash refers to the “/home/(*user-name*)/.bcshrc” file. For changing the environment, type

```
$  pwd
/home/(user-name)
$  cp  /etc/bash.bashrc  .bashrc
$  nedit  .bashrc  &
```

In the nedit window, insert the following lines:

```
alias  rm=’rm -i’
alias  mv=’mv -i’
alias  cp=’cp -i’
```

```
$  source  .bashrc                    refer to new environment file

%  rehash                             refresh environment (for csh)
```

Useful Tip 1

If you mistyped on the command line, you do not need to type the same command again. Push the up arrow key and modify the command.

e.g.,

$ ***cpy*** *file-name1 file-name2 (wrong command for “cp”)*

$ (push up-arrow key)

$ cpy file-name1 file-name2 (edit command using left- or right-arrow key)

$ ***cp*** *file-name1 file-name2*

This tip is also useful in case you want to repeat the same command.

Useful Tip 2

Bash or tcsh has a function to interpolate a file or command name. You do not need to type all the characters of a long filename.

Push the tab key after you type a few beginning characters of a file or command name.

e.g.,

There is a file called program.f in the “work” directory.

$ cd work

$ less prog (push the Tab key)

$ less program.f (filename is interpolated!)

Managing Your Files

You can create a backup file using the “**tar**” command. The syntax is

Create a backup file from files or directory

```
$ tar  -cvf  (backup-file-name)  (files or directory)
```

Extract all the files from backup file

```
$ tar  -xvf  (backup-file-name)
```

P.376 - 381, Unix in 24 Hours

Exercise

You will create “tmp1” and “tmp2” directories in your “UNIX” directory; copy the FORTRAN programs into “tmp1”; go to “tmp1”; backup the Fortran programs in “tmp1”; copy the backup files into “tmp2”; and extract all the files in “tmp2”.

Exercise

Go to home directory; create “tmp1” and “tmp2” directories

```
$ cd  UNIX
$ mkdir  tmp1  tmp2
```

Copy the FORTRAN programs into “tmp1”; move “tmp1”

```
$ cp  *.f  tmp1/
$ cd  tmp1
```

Backup the Fortran programs in “tmp1”; copy backup file into “tmp2”

```
$ tar  -cvf  program.tar  *.f
$ cp  program.tar  ../tmp2
```

Go to “tmp2”; extract all files in “tmp2”

```
$ cd  ../tmp2
$ tar  -xvf  program.tar
```

- “*” (Asterisk) is replaced by any character in a filename (e.g., *.f representing all the FORTRAN files in the working directory)
- “?” (question) represents any single character in a filename

Redirecting Input/Output

Input

The shell takes whatever you type on your keyboard as the input to the command (after you press (*Return*) to start the command)

Input redirection

You can use a given file as the input to a command and/or a program that does not accept filenames by using the "**<**" operator. The syntax is

$ command < (input-file)

P.143 - 146, Unix in 24 Hours

Redirecting Input/Output

Sample program to use redirecting input (**<**)

Make a simple FORTRAN code and save it as “program_in.f”

```
(7 spaces) real a, b, c
           write(6,*) ’a=?’
           read(5,*) a
           write(6,*) ’b=?’
           read(5,*) b
           c=a+b
           write(6,*) ’c=’,c
           stop
           end
```

Make an input file and save it as “input.dat”

```
  1.0
  2.0
```

Confirm that the program can be compiled and run.

```
$ g77  program_in.f
$ ./a.exe  <   input.dat
 a=?
 1.0
 b=?
 2.0
 c=3.
```

Redirecting Input/Output

Output

As the command runs, the results are usually displayed on your terminal. The terminal is the command's standard output.

Output redirection

You can write the results of a command or/and program to a named file using the "**>**" operator. The syntax is

```
$  command  >  (output-filename)
```

e.g.,

```
$  ls  –al  >  list.dat
$ cat  list.dat
  total 104
  d rwxr-xr-x       6             yagi  none      0       Sep 19 10:28  .
  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
$  echo  "Ohayou-gozaimasu"  >  greeting.dat
$ cat  greeting.dat
Ohayou-gozaimasu
```

echo: display a line of text

If you created a *new-file* consisting of *file1* followed by *file2*, type

```
$ cat   (file1)  (file2)  >  (new-file)
```

or

```
$ cat   (file1)  >  (new-file)
$ cat   (file1)  >>  (file2)
```

(if you want to add the contents of *file1* to the end of *file2)*

Comparing Text Files

“diff” command is useful to check the differences between text file A and text file B.

e.g.,

```
$ diff  program.f  program_in.f
2,3c2,5
<       a=1.0
<       b=2.0
---
>       write(6,*) 'a=?'
>       read(5,*) a
>       write(6,*) 'b=?'
>       read(5,*) b
```

$\longleftarrow$ program.f

$\longleftarrow$ program_in.f

Pipes and Filters

You can connect two commands together so that the output from one command (program) becomes the input to the next command (program) using the “**|**” (vertical bar) operator. Any two commands (programs) can form a pipe as long as the first program writes to the standard output and the second program reads from the standard input. The syntax is

$ command (options) (filename) | command (options)

If you use a filtering program, the pipe is very useful.

Filtering program: grep, sort, more, less

P.146 - 149, Unix in 24 Hours

Pipes and Filters

The “**grep**” program searches for files with lines that have a certain pattern. The syntax is

$ grep (pattern) (files)

e.g.,

```
$ grep  real  *.f
program.f:   real  a, b, c
program2.f: real  a, b, c
program3.f: real  a, b, c
filename:     lines that have the specified pattern
```

P.153 - 156, Unix in 24 Hours

e.g.,

```
$ ls  –al   |   grep  prog
-rwxr-xr-x  1  fujii  None 56057  10  11 10:59 program.exe
-rw-r--r--  1  fujii  None      100  10  11 10:55 program.f
-rw-r--r--  1  fujii  None      158  10  11 10:59 program_in.f
-rw-r--r--  1  fujii  None      100  10  11 10:55 program2.f
```

First, our example runs “**ls –al**” to list your directory; the standard output of “**ls –al**” is piped to grep, which only outputs the lines that contain the string “prog”.

Pipes and Filters

The “**sort**” program arranges the lines of text alphabetically or numerically. The syntax is

$ sort (options) (file)

option	Descriptions
-n	Sort numerically and ignore blanks and tabs.
-r	Reverse the order of sort.
-k	Specify the location or field (column) to sort.

e.g.,

$ ls –al | grep prog | sort -k 5 -n -r

P.147 - 149, Unix in 24 Hours

```
-rwxr-xr-x  1  fujii  None 56057  10  11  10:59 program.exe
-rw-r--r--  1  fujii  None   158  10  11  10:59 program_in.f
-rw-r--r--  1  fujii  None   100  10  11  10:55 program2.f
-rw-r--r--  1  fujii  None   100  10  11  10:55 program.f
```

Multitasking

Running a command as a background process

To run a command in the background, add the “&” character at the end of the command line. The syntax is

$ program < (input-file) > (output-file) &

[1] 12222

The process ID for this command is 12222. To check on a process, the “**ps**” command is useful. The syntax is

$ ps (options)

P.273, 274, Unix in 24 Hours

To cancel a process, the “kill” command is useful. The syntax is

$ kill (ID)

P.312, 317 - 319

Multitasking

e.g.,

```
$ nedit  &
[1]  3292
$ ps
      PID    PPID    PGID     WINPID  TTY  UID    STIME COMMAND
     2380       1    2380       2380       con 1006 18:05:18 /usr/bin/bash
      432    2380     432       2436       con 1006 19:17:56 /usr/bin/sh
     3764     432     432        168       con 1006 19:17:56 /usr/X11R6/bin/xinit
     3544    3764    3544        240       con 1006 19:17:56 /usr/X11R6/bin/XWin
     2808    3764    2808       1672       con 1006 19:17:59 /usr/bin/xterm
     2136    2808    2136       3904         0 1006 19:18:00 /usr/bin/bash
     3292    2136    3292       1960         0 1006 19:41:49 /usr/X11R6/bin/nedit
      984    2136     984       3040         0 1006 19:41:50 /usr/bin/ps
$ kill  3292
[1]+  Terminated              nedit
```

You can enter an entire sequence of commands separated using semicolons (;). The syntax is

```
$  command1 ; command2
```

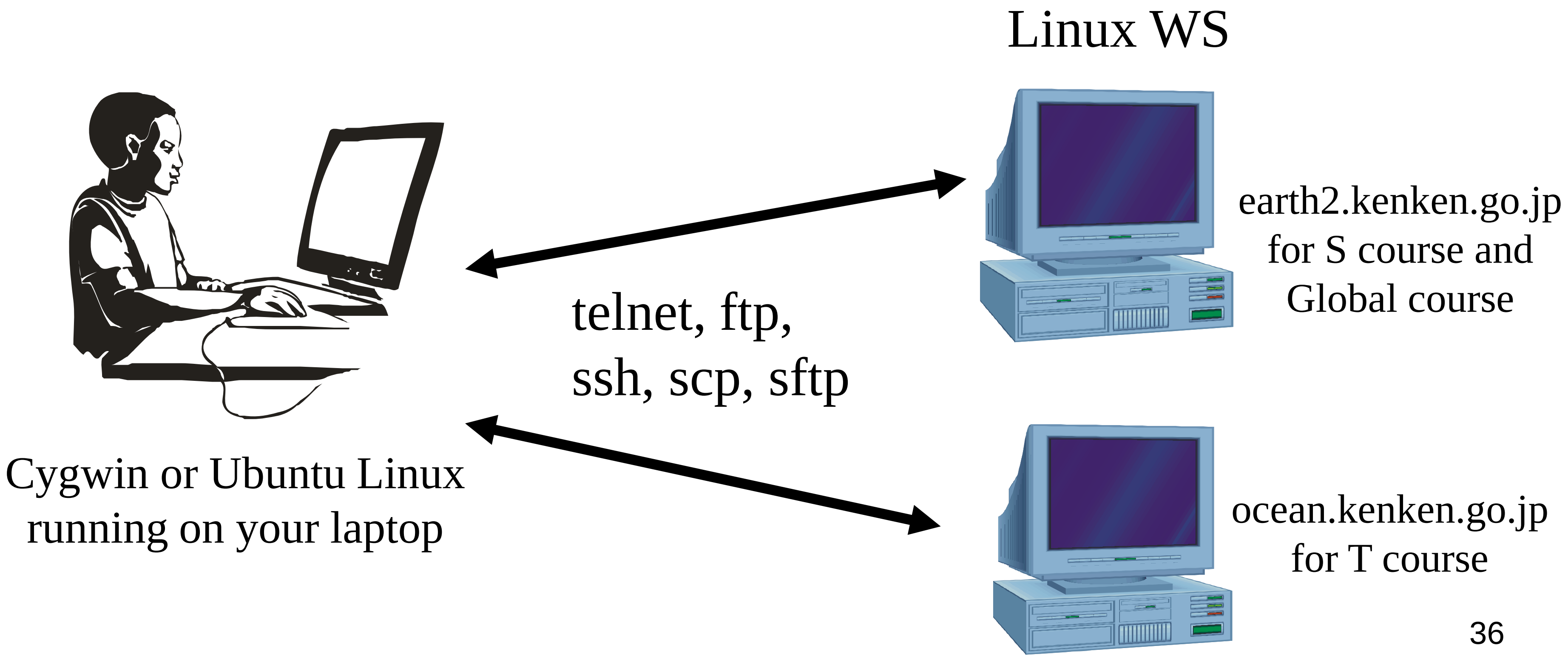
If you want to run a command line in the background, type

```
$ command  &
$ (command1  ;  command2)  &
```

Multiuser

UNIX offers a “multiuser system” environment.

We can use the same UNIX computer (Linux WS) simultaneously from any computer (Windows PC).



Connecting to a UNIX Computer

- When you might have a workstation on your desk, but you need to work from the main computer at another location, remote login programs can be useful and powerful.
- We use "**ssh**" (secure shell) or "telnet" to log into another computer.
  - The syntax for most remote login programs is
    $ (program-name) (remote-hostname)
  - Type
    $ ssh earth2.kenken.go.jp –l (user ID)
    or
    $ ssh (user ID)@earth2.kenken.go.jp

P.423 - 424, Unix in 24 Hours

Connecting to a UNIX Computer

IP address

- Almost all machines on a LAN have an Internet Protocol (IP) address and host name, which are unique to each machine. IP address is a 32-bit number. The syntax is ???.???.???.??? (four sets of numbers separated by “.”).

Domain host name

- Domains and host names are usually used instead of IP addresses.
- e.g.,
- $\underset{\swarrow}{\text{earth2}}.\underline{\text{kenken.go.jp}}$

host name **domain name**

Connecting to a UNIX Computer

You can confirm whether another computer is alive or not using the “**ping**” command. The syntax is

```
$  ping   (host-name or IP-address)
```

e.g.,

```
$ ping  earth2.kenken.go.jp
Pinging earth2.kenken.go.jp [172.16.21.40] with 32 bytes of data:
Reply from 172.16.21.40 : bytes=32 time=4ms TTL=253
Reply from 172.16.21.40 : bytes=32 time=4ms TTL=253
Reply from 172.16.21.40 : bytes=32 time=4ms TTL=253
```

(type (Ctrl)-c to stop)

Connecting to a UNIX Computer

```
$ ssh  earth2.kenken.go.jp  –l  (your ID)
          or
$ ssh  (your ID)@earth2.kenken.go.jp
Connected to earth2.kenken.go.jp (172.16.21.40).

(user ID)@earth2's password:
[(user ID)@earth2 (user ID)]$


e.g.,
[fujii@earth2 fujii]$:
[fujii@earth2 fujii]$: ls  -a
(file names in earth2) …………………
[fujii@earth2 fujii]$: mkdir  UNIX

[fujii@earth2 fujii]$: exit
Connection closed by foreign host.
```

You can edit, compile, and run in “earth2” or “ocean” using ssh through the network.

Changing your Password

On a UNIX system, everyone can find your username. If you type “who”, you can get all the usernames. For your own safety, please change your password using the “**passwd**” command. Type

```
$ passwd
passwd:  changing password for yagi
Enter login(NIS) password: *************
New password: **************
```

- In general, a password should be something that is easy for you to remember but difficult for other people to guess. Please use a combination of alphabets, numbers, and symbols.

P.29 - 32, Unix in 24 Hours

Copying Files between Two Computers (1)

The command “**sftp**” or “**ftp**” (file transfer protocol) is a flexible way to copy files between two computers. The syntax is

```
$  sftp  (your ID)@(host-name)
```

P.424 - 429, Unix in 24 Hours

Type

```
$  sftp  fujii@earth2.kenken.go.jp
Connecting to earth2.kenken.go.jp...
fujii@earth2.kenken.go.jp's password:
password: **************
sftp>

sftp>  type “quit”, “bye”, or “exit” to quit
$
```

Copying Files between Two Computers (1)

Some **sftp** commands

command	Description
sftp> put (file-name)	Copies the file from your local computer to the remote computer.
sftp> get (file-name)	Copies the file from the remote computer to your local computer.
sftp> mput (file-names)	Copies the files from your local computer to the remote computer.
sftp> mget (file-name)	Copies the file from the remote computer to your local computer.
sftp> cd (path-name)	Changes the working directory on the remote machine to *path-name*.
sftp > ls	Lists the remote directory.
sftp > !ls	Lists the local directory

Some **ftp** commands

ftp> bin	Tells the ftp to copy the following file without translation.
ftp> asc	Transfers plain text file data if needed.

Copying Files between Two Computers (2)

The command "**scp**" is a simple way to copy files between two computers.

You have to know in advance where the file you want is located in the remote computer.

The syntax is

$ scp   (your ID)@(host-name):(file name with remote directory path) (local directory)

Type

$ scp fujii@ocean:/home/fujii/work/program.f .

If you want to transfer the entire directory to the current directory, type

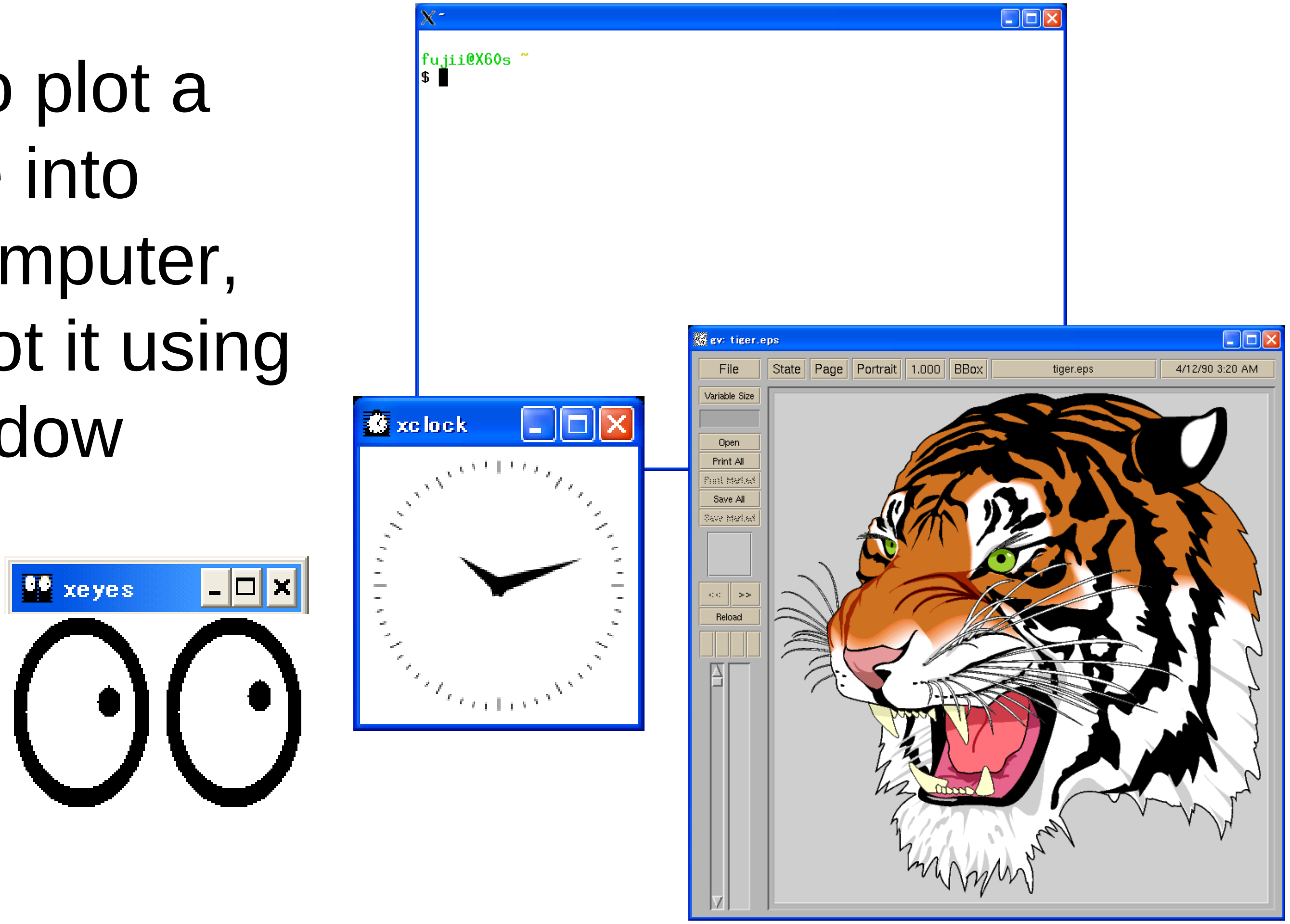
$ scp –r fujii@ocean:/home/fujii/work .

If you want to preserve the time stamp of the file, type

$ scp –p fujii@ocean:/home/fujii/work/program.f .

Connecting to a UNIX Computer Using X Window System

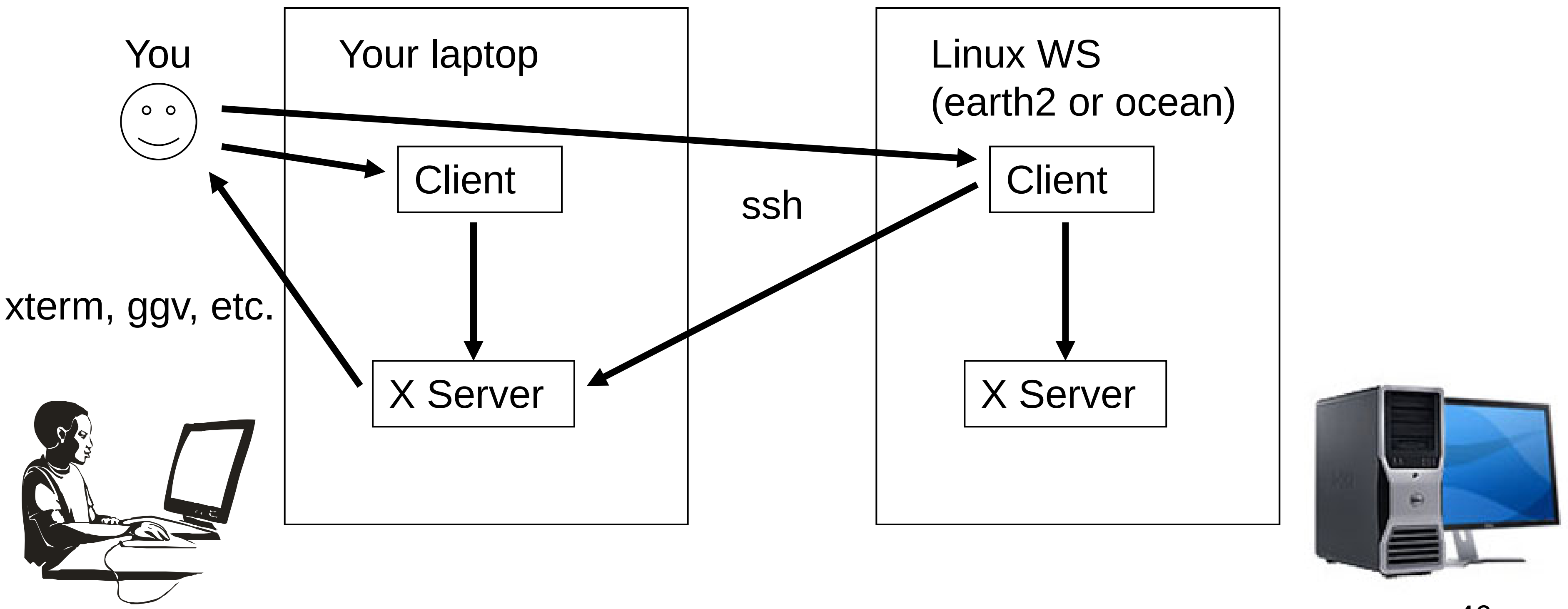
If you want to plot a graphic file into another computer, you can plot it using the “X Window System”.



```
fujii@X60s ~
$ 
```

X Window System

X Window System is a client-server system. The "client" asks the "server" to offer graphics, and the "server" can yield a graphical environment.



Connecting to a UNIX Computer Using X Window System

To start X Window System,
select “Xwin Server” icon.

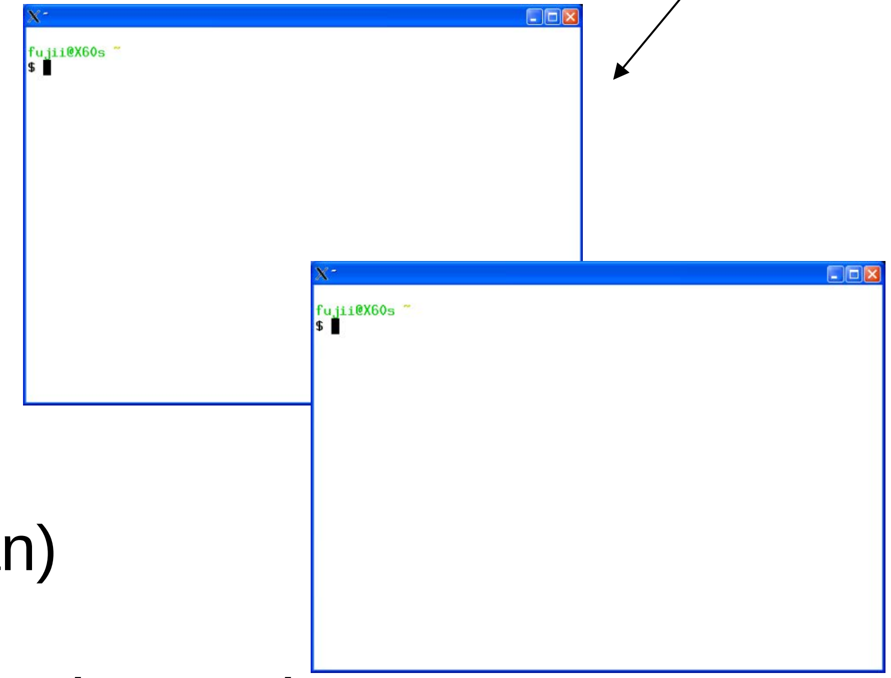
Open another terminal window

$ xterm &

Log into Linux WS (earth2 or ocean)
in a “new command window”

*$ ssh **–XC** (user ID)@earth2.kenken.go.jp*

Old window (Local PC)



```
fujii@X60s ~
$ 
```

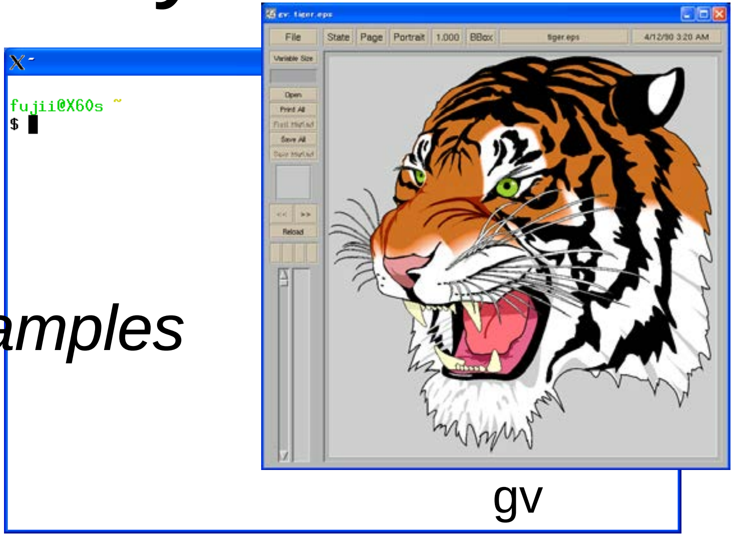
```
fujii@X60s ~
$ 
```

New window (for remote WS)

Connecting to a UNIX Computer Using X Window System

In the "old command window"
 (local laptop)

```
$ cd  /usr/share/ghostscript/9.06/examples
$ gv  tiger.eps  &
```

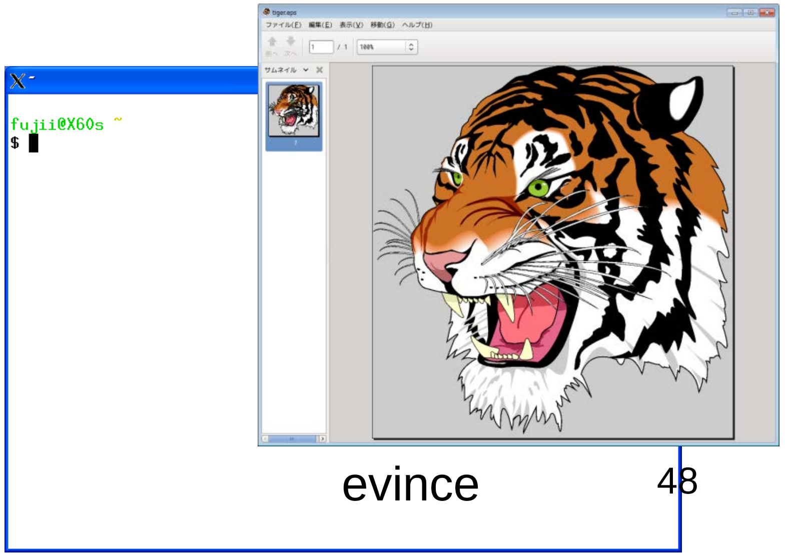


```
fujii@X60s ~
$
```

gv

In the "new command window"
 (remote Linux WS: earth2 or ocean)

```
$ evince  tiger.eps  &
```



```
fujii@X60s ~
$
```

evince

Exercise

You will put a *PS* file (*e.g., tiger.eps*) into the Linux WS (earth2 or ocean) using “sftp” or “scp” command, and plot the *PS* file in the Linux WS on your display.

Exercise

On the local computer window, type

```
$ cd  /usr/share/ghostscript/9.06/examples
$ sftp  (your ID)@earth2.kenken.go.jp
sftp> cd UNIX
sftp> put (PS-file)
sftp> quit
$ ssh  -XC  (your ID)@earth2.kenken.go.jp
```

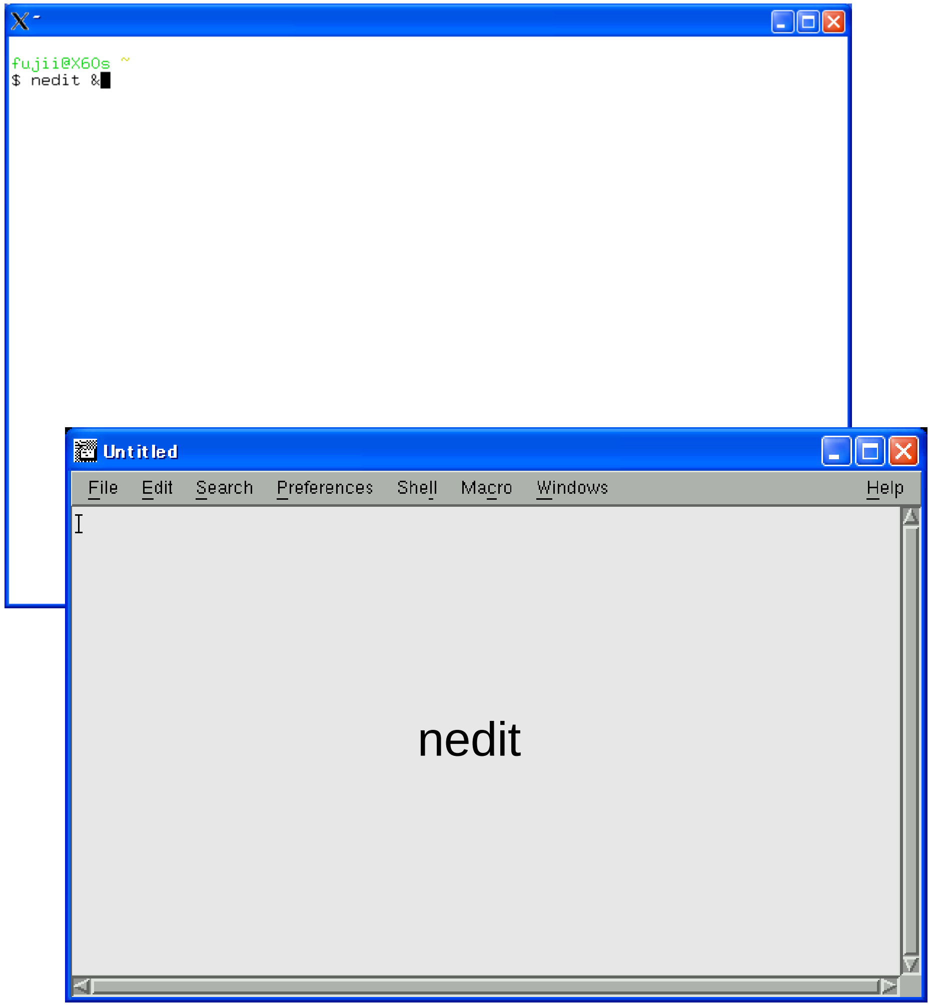
On earth2 or ocean command line, type

```
$ cd  UNIX
$ evince  (PS-file)
```

Basic Shell Scripting

"**Shell script**" is powerful for automation and customization.

- Shell scripts are little programs written in the shell language, which are similar to *BATCH* files on a conventional PC.
- Starting editor
- Type
- *$ nedit &*



```
fujii@X60s ~
$ nedit &
```

Basic Shell Scripting

Write the following in the nedit window:

```
#!/bin/csh  -f
echo “ +++++  working directory +++++ “
pwd
echo “ FILE in working directory, arranging in file size order ”
ls  –l  |  sort  -k 5 -n
```

Save as “print_file.csh”; change to the execute mode; run csh script

```
$ chmod  u+x  print_file.csh
$ ./print_file.csh
+++++ working directory +++++
/home/yagi
FILE in working directory, arranging in file size order
-rw-r--r--     1 yagi     none          578 May 15 18:21 csh.cshrc
-rwxr-xr-x    1 yagi     none         1390 Sep 19 14:21 gp7.sh
-rwxr-xr-x    1 yagi     none         1392 May 23 12:29 dsmnote.sh
-rwxr-xr-x    1 yagi     none         1392 May 23 19:35 test.sh
-rwxr-xr-x    1 yagi     none         1576 May 16 09:44 wmaker.sh
-rwxr-xr-x    1 yagi     none         1581 May 15 16:45 startxwin.sh
```

Basic Shell Scripting

If you program using FORTRAN, the “*do-loop*” command is useful to avoid repetitive tasks. In the shell script, “*foreach*” is similar to “*do-loop*”.

Write the following in the nedit window:

```
    #!/bin/csh  -f
    foreach  i (1  2  3)
    echo $i
    end
```

Save as “loop.csh”; change mode; and run

```
$ chmod  u+x  loop.csh

$ ./loop.csh
1
2
3
```

Basic Shell Scripting

If you program using FORTRAN, "*if*" is useful, which is also useful in shell scripting. Write the following in the nedit window:

```
#!/bin/csh  -f
foreach  i  (1  2  3 )
echo  $i
if  ($i  ==  2)  echo  " two"
end
```

Overwrite and run

```
$ ./loop.csh
1
2
two
3
```

Basic Shell Scripting

The following are the other relational operators used in “if”.

<	less than
< =	less than or equal to
==	equal to
!=	not equal
> =	greater than or equal to
>	greater than

Awk Programming (1)

Awk is a programming language designed to search for, match patterns, and perform actions on files. Awk programs are generally quite small and are interpreted.

Awk scans the input lines one after the other, searching each line to see if it matches a set of patterns. The action is performed when the pattern matches that of the input line. The simple syntax is

patterns “{ *action* }”

P.168 - 173, Unix in 24 Hours

Awk Programming (1)

When *awk* scans an input line, *awk* divides the line into a number of fields. The fields are separated by a space, tab, or special character (you can define special characters using the -F option). The fields are numbered beginning at one, and the dollar symbol ($) is used to represent a field.

For instance, the following line in a file

"I am seismologist."

has three fields.

$1	"I"
$2	"am"
$3	"seismologist."

Field zero ($0) refers to the entire line.

Awk scans the lines from files.

Awk Programming (1)

We will consider the most simple awk program. Open nedit and type

{ print $0 }

There is no pattern to match: only an action is expressed. This means that for every line encountered, perform the specified action. The action *prints* field 0 (the entire line).

Save as "print_infile.awk" and run this program. The syntax is

```
$ awk –f  print_infile.awk  (file-name)
```

Awk interprets the actions specified in the program file print_infile.awk, and applies it to each line read from the file (*file-name*). The effect is to print out each input line read from the file, thereby displaying the file on the screen (same as the Unix command *cat*).

As an example, we input "list.dat" that has been created in the previous step. Type

```
$ awk –f  print_infile.awk  list.dat
total 104
d rwxr-xr-x    6          yagi  none     0    Sep 19 10:28.
……………………………………………………………
……………………………………………………………
```

Awk Programming (1)

Simple Pattern Selection

This involves specifying a pattern to match for each input line scanned. The following awk program (print_infile.awk) compares the fifth field ($5); if the field is greater than or equal to "50", the specific action is performed (the entire line is printed). Modify print_infile.awk as

```
$5 >= 50 { print $0 }
```

Note: The operation of the ">=" symbol is the same as the csh script.

Save and run

```
$ awk -f print_infile.awk list.dat
```

P.171, Unix in 24 Hours

```
-rwxr-xr-x   1 fujii none 80493 Oct 20 19:19 a.exe
-rw-r--r--    1 fujii  none 69574 Oct 21 21:57 cmt.gmt
-rwxr-xr-x   1 fujii none 80493 Oct 20 19:19 program.exe
-rw-r--r--    1 fujii  none     100 Oct 20 19:19 program.f
-rwxr--r--   1 fujii  none       66 Oct 22 11:08 loop.csh
```

The program prints out all the input lines where the size is greater than and equal to "**50**".

Awk Programming (1)

Combining Patterns

Patterns can be combined to provide more complex matching. The following symbols are used to combine patterns.

||	logical “**or**” (either pattern can match)
&&	logical “**and**” (both patterns must match)
!	logical “**not**” (patterns not matching)

e.g.,

$5 >= 50 && $5 <= 2000 { print $0 }

Awk Programming (1)

Printing A Text String

Let us examine how to print some simple text. Consider the following statement: printf(“size : ”);

The *printf* statement is terminated by a semicolon. The brackets are used to enclose the argument, and the text is enclosed using double quotes. Now, let us combine it into an actual awk program that displays the location of all 286-type computers. Modify print_infile.awk as

```
{  printf  (“file name: ");  printf  $9;  printf  (“, size: ”);  print  $5  }
```

Save and run

```
$ awk  –f  print_infile.awk  list.dat

file name: , size:
file name:. , size: 0
file name:.. , size: 0
file name: a.exe, size: 80493
..............................................
..............................................
```

Exercise

You will get information of Global CMT (Harvard CMT) solutions from the Web site (http://www.globalcmt.org/CMTsearch.html) and select the event using the awk program.

Exercise

Input starting date (2001/1/1)

and ending date (2001/12/31)

Exercise

Select *“GMT psmeca input” format* and click *Done.*

Exercise

You can get the CMT catalog for 2001.

Please copy the results and save as "cmt.gmt" using nedit.

Useful Tip 3

Copy and paste the text in X Window, between X Window (e.g., xterm or nedit) and Windows software (e.g., Word, Excel, or IE).

	Copy	Paste
X Window	Just select text	Middle click
Windows software	Select area and (Ctrl)-c	(Ctrl)-v

Exercise

Please check the CMT file:

```
$ less cmt.gmt
120.42 19.20 77 -2.08 0.59 1.49 -4.47 -2.02 -9.11 23 X Y 010101A
127.07 6.73 44 1.48 0.19 -1.67 0.57 0.09 -0.37 27 X Y 010101B
127.13 7.12 44 1.00 0.26 -1.26 1.10 -0.96 -0.27 26 X Y 010101D
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Information for each field.

$1: longitude of centroid

$2: latitude of centroid

$3: depth of centroid

$4-9: Mrr, Mtt, Mff, Mrt, Mrf, Mtf (moment tensor components in 10*expo. (dyncm))

$10 expo in $4-9

Exercise

1. Please search earthquakes near Japan, and save it as file “*japan.cmt*”.

   Longitude: 120E–150E

   Latitude: 25N–45N

2. Please search earthquakes near your country, and save it as file “*country.cmt*”.
3. Please search deep earthquakes (over 400 km), and save it as file “*deep.cmt*”.

Exercise

1. Open nedit; type

   ```
   $1 >= 120 && $1 <= 150 && $2 >= 25 && $2 <= 45 {print $0}
   ```

   Save as “*japan.awk*” and run

   ```
   $ awk –f japan.awk cmt.gmt > japan.cmt
   ```

2. Refer to exercise 1.
3. Open nedit; type

   ```
   $3 >= 400 {print $0}
   ```

   Save as “*deep.awk*” and run

   ```
   $ awk –f deep.awk cmt.gmt > deep.cmt
   ```

Awk Programming (2)

BEGIN and END Statements

The keywords BEGIN and END are used to perform specific actions relative to the program’s execution.

BEGIN: The action associated with this keyword is executed before the first input line is read.

END: The action associated with this keyword is executed after all the input lines have been processed.

The **BEGIN** keyword is normally associated with printing titles and setting default values, whilst the **END** keyword is normally associated with printing the totals. Consider the following awk program, which uses BEGIN to print a title.

Modify “deep.awk”

```
BEGIN { print “deep earthquake (over 400 km)” }
$3  >=  400  { print $0 }
```

Run “deep.awk”

```
$ awk  -f  deep.awk  cmt.gmt
deep earthquake (over 400 km)
:::::::::::::::::::::::::::::::::::::::::::::::::::
```

Awk Programming (2)

User-defined Variables

Awk programs support the use of variables. We will count the number of deep earthquakes (over 400 km). Variables are explicitly initialized to zero by awk, so there is no need to explicitly assign a value of zero. When each input line is read, field three (***$3***) is checked to see if it is greater than *400*. If so, awk variable is incremented (the "**symbol** ++" means increment by one).

Open nedit; type

```
$3  >=  400  {  nde++  }
END { printf ("The number of deep earthquakes is "); print (nde) }
```

Save as "count.awk"; run "count.awk";

```
$ awk  –f  count.awk  cmt.gmt
The number of deep earthquakes is ??
```

Awk Programming (2)

Awk's Assignment Operators

The following is a summary of awk assignment operators.

+	add
-	subtract
*	multiply
/	divide
++	increment
--	decrement
^	exponential
+=	plus equals
-=	minus equals
*=	multiply equals
/=	divide equals
^=	exponential equals

e.g.,

sum= sum + 10 # same as "sum += 10"

sum= sum /10

Exercise

1. Please count the number of earthquakes for each depth range:
   - 1: (0-30 km)
   - 2: (30-100 km)
   - 3: (100-250 km)
   - 4: (250-400 km)
   - 5: (400-600 km)
2. Please calculate the average depths.

Exercise

1. Refer to the previous example.
2. Open nedit; type

```
{ ne  ++;  sumdepth  +=  $3 }
END { av  =  sumdepth / ne;  printf  ("The average depth is " );  print av }
```

Save as “calave.awk; run

```
$ awk  –f  calave.awk  cmt.gmt
The average depth is 57.9521
```

Awk Programming (Appendix)

Awk Built-in Variables

Awk provides a number of internal variables useful to process files. These variables are accessible by the programmer.

ARGC	number of command-line arguments
ARGV	array of command-line arguments
FILENAME	name of current input file
FNR	record number in current file
FS	input field separator
NF	number of fields in input line
NR	number of input lines

Awk Programming (Appendix)

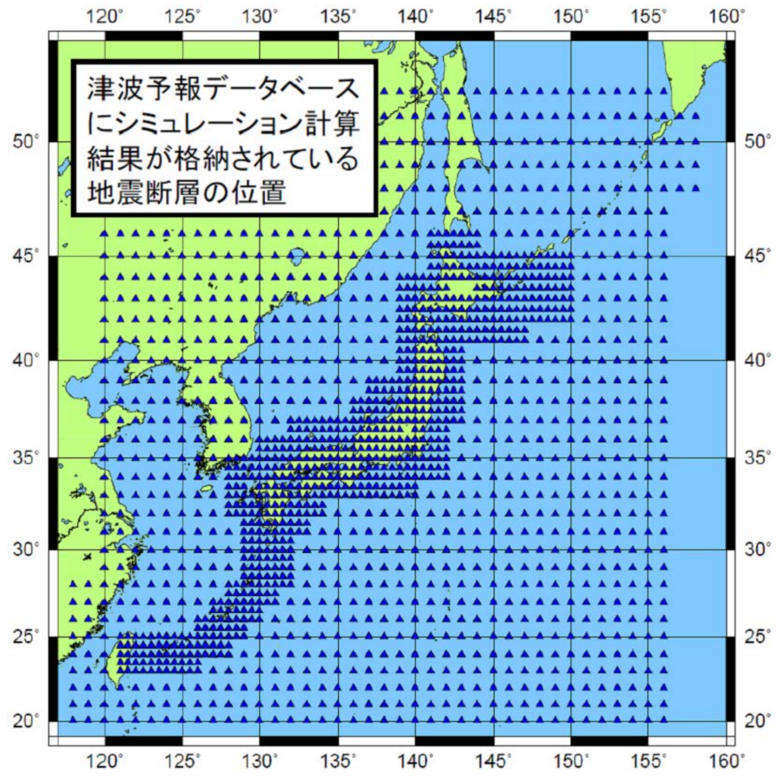
Awks Built-in Functions

sqrt(x)	square root of x
cos(x)	cosine of x (x in radians)
sin(x)	sine of x (x in radians)
atan2(y,x)	arctangent of y (x in radians)
exp(x)	exponential function of x
int(x)	integer part of x truncated toward 0
log(x)	natural logarithm of x
rand()	random number between 0 and 1

An example of shell script

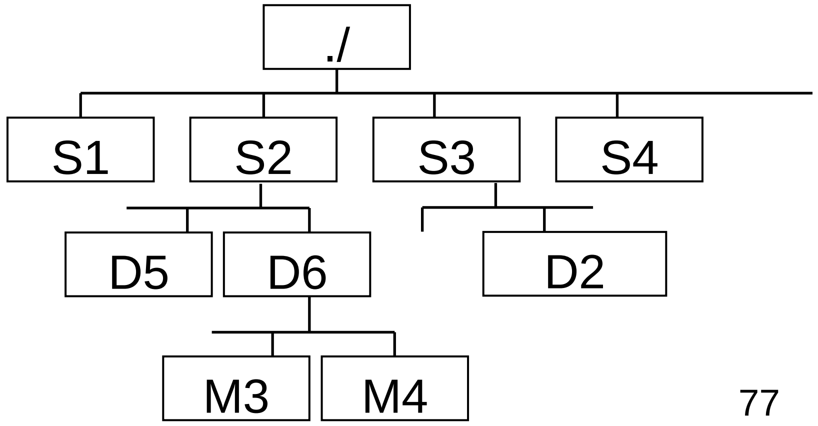
Computations using several or many parameters

e.g. Tsunami simulations for tsunami database



Source points: 1,500
Depths: 6 (0 – 100 km)
Magnitudes: 4 (6.5, 7.0, 7.5, 8.0)

Total: 36,000 cases !!



Source: http://www.data.jma.go.jp/svd/eqev/data/tsunami/ryoteki.html
(Japan Meteorological Agency's Web site)

An example of shell script

Without shell script

```
$cd  S1
$cd  D1
$cd  M1
$/home/fujii/tsunami.exe

$cd  ../M2
$/home/fujii/tsunami.exe

$cd  ../M3
$/home/fujii/tsunami.exe

$cd  ../M4
$/home/fujii/tsunami.exe

$cd  ../../D2
$cd  M1
$/home/fujii/tsunami.exe
 .
 .
 .
```

With shell script

$./TDB.csh &

```
#!/bin/csh

@ i = 1
while ($i  <=  1500)
set dirS = S$i ; cd $dirS
   @ j = 1
   while ($j  <=  6)
   set dirD = D$j ; cd $dirD
      @ k = 1
      while ($k  <=  4)
         set dirM = M$k ; cd $dirM
         echo 'Now,'  $dirS $dirD $dirM
         /home/fujii/tsnami.exe
         cd ..
      @ k++
      end
      cd ..
   @ j++
   end
   cd ..
@ i++
end
```