Topic 13

Programming on Linux Environments

A Peek in the Real Engineering World

資料結構與程式設計

Data Structure and Programming

12.18.2019

1

What we are going to learn today...

- History of Linux OS
- Linux survival guide
  - Installation
  - User accounts
  - File/directory access privileges
  - Useful commands
  - Customize your environment
- Programming on Linux
  - Text editor (e.g. vi)
  - C/C++ compiler (e.g. g++)
  - Makefile
  - Debugger (e.g. ddd)

2

Proclaimer

- It's not possible to learn Linux in one day!!
- This slide acts more like a reference, or training tutorial.
- You definitely need to practice it on your own.

3

Well, I assume you know that Linux is an Operating System (OS), and the kernel of an OS contains ---

- **System call interface**
- **Process control**
- **Memory management**
- **File system management**
- **Device drivers**
- **... and more**

4

Why Linux?

1. Commonly used in industry
   - Stable, multi-user, multi-tasking workstations
   - Very similar to Unix OS, but PC is much cheaper

   ➔ A very useful skill to learn in school

2. Get to know your computer better
   - Windows for dummies
     - Easy to use, but difficult to fix
   - Forced to learn the logic behind the system admin

5

Some comments from VBird…

- 在 DOS 的年代，要學會 DOS 的指令雖然有點麻煩，不過卻可以讓人瞭解到每一個指令的背後含意，尤其是在寫簡單的批次檔時， 需要一些小小的邏輯概念，另外，在 DOS 年代玩 Game 的時候， 常常需要花上長時間去設定音效卡或顯示卡，這也造成了一些不喜歡碰電腦的人。
- 但是這一些麻煩在進入 Windows95 系統以後都不見了！不過這並不表示這是一件值得高興的事， 因為操作的介面越簡單，出錯的機率就越大！
- 怎麼說？因為妳可以一不小心就砍了很多的東西， 然後完全救不回來的當在那邊！不然就是當妳需要增加一些功能的時候，完全不知到從那邊著手去修改！ Windows 或許是方便，不過，在這種介面之下， 真的會讓人完全沒有機會去瞭解電腦的運作流程，如此也就讓哪些電腦維修廠商有生意作了！

6

Before talking about the history of Linux,

we need to take a look at the "Open Source Movement"…

7

Free = 免費 ?? or 自由??

- Many computer hobbyists / hackers advocate that software should be free from restrictions against copying or modification in order to make better and efficient computer programs

8

Open Source Movement

- Richard Stallman
  - A cult hero in the realm of computing
  - Started his awesome career in the famous Artificial Intelligence Laboratory at MIT
  - In 1983 started a open source movement where a “GNU” project began
- GNU project
  - Stallman wrote a (best-ever) C compiler (gcc) in 1984
  - By 1991 (the year Linux was born), lots of people contribute their source codes to GNU
  - http://www.gnu.org/

9

“Free as in Freedom” --- Philosophy of the GNU project

- “Free software” is a matter of liberty, not price.
  - To understand the concept, you should think of “free” as in “free speech,” not as in “free beer.”
  - Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software.
- *Copyleft* ⊚ is a general method for making a program or other work free, and requiring all modified and extended versions of the program to be free as well.

10

GNU General Public License (GPL)

- http://www.gnu.org/licenses/gpl.html
- The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.
  - For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

11

FSF/UNESCO Free Software Directory

- http://directory.fsf.org/
- Numerous free software are well archived
  - 16,236 as of 12/17/2019
- *The free software movement is one of the most successful social movements to emerge in the past 30 years, driven by a worldwide community of ethical programmers dedicated to the cause of freedom and sharing. But the ultimate success of the free software movement depends upon teaching our friends, neighbors and work colleagues about the danger of not having software freedom, about the danger of a society losing control over its computing.*

12

More Open Source Repositories

- GitHub – the largest collection of (open) source codes in the world
  - As of of May 2019, GitHub reports having over 37 million users and more than 100 million repositories (including at least 28 million public repositories).
  - In 2018, Microsoft acquired GitHub for 7.5B
- Other open source organizations: https://opensource.com/resources/organizations

13

Free OS?

- However, before Linux, there was no free OS source code in GNU
- MINIX
  - The only OS that was not commercial
    - Windows was too buggy
    - Unix was too expensive
  - Written from scratch by Prof. Andrew S. Tanenbaum
  - Anyone buys his OS book can get a free copy of the 12,000 line source codes (GitHub ref)
  - Inspired many students (~40,000-person newsgroup) for hands-on OS implementation

14

And one of them is

Linus Benedict Torvalds

15

New Baby in the Horizon

- MINIX was good, but still an OS for the students
  - Not an industry strength one
- In 1991, Linus Torvalds was a 2nd year CS student at the Univ. of Helsinki and a self-taught hacker
  - On Aug 25, he posted a message (about Linux) on MINIX news group
  - Linux 0.01 was released in mid-Sep
- A complete kernel (v1.0) was released in 1994

16

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat same physical layout of the file-system (due to practical reasons) among other things). I've currently ported bash(1.08) and gcc(1.40),and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)
Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

17

People like it.

18

Something about Linus Torvalds

- Born on 12.28.1969
- Also the original author of "git" (2005)
  - Learnt "BASIC" at the age of 11
  - Attended Universitas Helsingiensis in 1988.
  - Served in army in 1989 (for 11 months), where he studied "Minix"
  - Got to know Unix in 1990
  - Published Linux in 1991
  - Got master degree in 1996 (Thesis: Linux)
  - Received donations of stock options from RedHat and VA Linux in 1999 (~US$20M)
  - In order to manage the source code of Linux, he developed git (2005)

https://en.wikipedia.org/wiki/Linus_Torvalds

19

Basic Linux Survival Guide

- Getting access to Linux machines
- Using Linux
  - User accounts
  - File/directory access privileges
  - Useful commands
  - Customize your environment
- Writing programs on Linux

20

Getting access to Linux machines

- Install Linux OS on your computer
- Buy a Mac (Linux alike)
- Apply for an account on some Linux machine, and use "ssh" to remote login
  - On a text terminal, of course
- Install virtual machine (e.g. Virtual Box), and then install Linux on it
- Partition your HD, install dual-boot program, and install Linux on one of them

21

Basic Linux Survival Guide

- Installation
- Using Linux
  - User accounts
  - File/directory access privileges
  - Useful commands
  - Customize your environment
- Writing programs on Linux

22

What/Why should I care (about learning Linux)?

- To survive in this class, you may only need to learn how to use the text editor, compiler (g++) and debugger (gdb or ddd).

➔ Why should you learn this much of Linux?

1. Most of you will need to use workstations in the future.
2. To get a better (engineering) sense of computer (science).
3. To support the spirit of freedom.

23

Well, most of the time you don't need to worry about creating user accounts.

(because you should already have one by now XD)

But to use Linux more smoothly and safely, it is important to know the types of user accounts, groups, and access privileges.

24

Remember…

- Linux is an OS for workstations
  - Multi-user, multi-task
  - Local host or remote login
  - Security is a big concern!
    - File ownership
    - File access privilege
- Administrator → *root*
- Ordinary users → others

25

What should we care about the file ownership and privileges?

- Who owns this file?
- Is this file read-only, writable, executable?
- Can others see the file? Can they modify the files?
- Can I open the file only to certain group of people?

26

Basic Designs for File Ownership and Privileges

- Each user has an login name and a unique id (UID)
  - root's UID = 0
- Each user is associated with one or more groups
  - Each group has its own id (GID)
- There are 3 kinds of access privileges for a file
  - Read / Write / Execute
- Each file/directory is associated with ---
  - An owner (UID) and a group (GID)
  - 3 sets of access privileges for different types of users (owner / group / others)

27

File/Directory Access Privilege

```
{ric@localhost}:/home/ric/classes/dspf06/bdd/bdd>ls -l
總計 60
lrwxrwxrwx  1 ric ric    7  1月  2  2007 bdd -> bin/bdd
drwxrwxr-x  2 ric ric 4096  1月  2  2007 bin/
drwxrwxr-x  2 ric ric 4096  1月 19  2007 dofiles/
drwxrwxr-x  2 ric ric 4096  1月 19  2007 include/
drwxrwxr-x  5 ric ric 4096  1月  2  2007 lib/
-rw-rw-r--  1 ric ric 1098 12月 31  2006 Makefile
drwxrwxr-x  5 ric ric 4096 12月 16 23:35 src/
drwxrwxr-x  2 ric ric 4096  1月  2  2007 testcases/
   ↑          ↑   ↑
 access     owner group
privilege
```


28

File/Directory Access Privilege

[e.g.] group

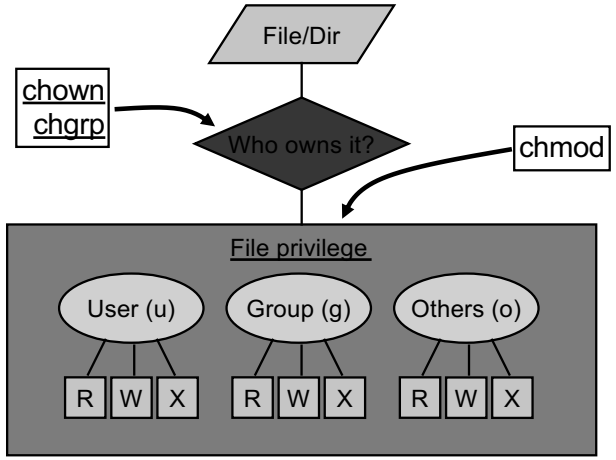
- **drwxrwxr-x 5 ric ric 4096 1月 2 2007 lib/**
  user others
  (yourself)

[e.g.]

- Suppose you are writing a proposal and you don't want the people outside your group to see this file ---
  - Yourself: Readable (r) and writable (w)
  - Group: Read only (i.e. write denied)
  - Others: Cannot read nor write

  ➔ **-rw-r----- 1 ric ric 736 12月 31 2006 proposal**

- Note:
  superuser (root) can read/write/execute all files



Changing File/Dir Ownership
(usually by *root* only)

- *chown* --- Change the owner of a file/dir
  - chown [-R] <Owner[:<Group>]> <Files>
    - "-R": recursive action
    - e.g. **chown jack proposal**
- *chgrp* --- Change the group of a file/dir

[Note] Each user can be associated with more than one groups

➔ Use "groups" command to see the group you are in

Changing File/Dir Access Privilege

- *chmod* --- Change the access privilege
  1. chmod [ugoa]<+-><rwx>
     - u: user, g: group, o: others, a: all
     - r: read, w: write, x: execute

     e.g. chmod g+rw myfile

     ➔ add read and write permission for users in my group
  2. chmod <0…7><0…7><0…7> <Files>
     - 3x3-bit numbers: <user><group><others>, 3 bit each

     ➔ 4: read, 2: write, 1: execute

     e.g. chmod 644 myfile

     ➔ for myself: read(4) + write(2)

     ➔ for group and others: read only (4)

     -rw-r--r-- 1 ric ric 908 12月 31 2006 myfile
     6 4 4

Useful Tip (1/2)

- Sometimes when you copy/download an executable file to the current directory, and you type it to run, you may get:
  > XXXX: Command not found.
- Most likely, it is due to the current directory is NOT included in the *search path* (more to cover later). Two way to resolve this:
  - Add current path (.) to the search path
  - Type "./" before the executable file name

33

Useful Tip (2/2)

- However, even after you do so, you may still get:
  > XXXX: Permission denied.
- This is very likely due to the access privilege error of the executable file (i.e. without "execution" permission)
  - "ls -l XXXX" to check
  - Use "chmod +x XXXX" to fix it

34

Basic Linux Survival Guide

- Installation
- Using Linux
  - User accounts
  - File/directory access privileges
  - Useful commands ⇦
  - Customize your environment
- Writing programs on Linux

35

Help!!! (Basic Commands for Query)

- man
  - Get the help (manual) message for a command
- who
  - Who are using this machine?
- whoami
  - Get the user name
- which <executable>
  - Where the executable located?
  - Depending on $path
- uname [-a]
  - Print system information
- file <filename>
  - What's the type of the file?

36

How do I manage the files and directories in my account?

- Do I use “file browser”?
  - You may, but trust me, when you are using Linux as a programming/working environment, you don’t want to move your hand to the mouse and click on windows.
  - ➔ Command line is more convenient!!
- What should I know?
  - What are the files in a directory?
  - How to go to other directories?
  - How to create and delete a file or directory?
  - What are the system files in a typical Linux machine?

37

File System and Directory Structure

- To list the files in a directory
  - ➔ ls [options] [dir name]
  - e.g. “ ls / ”

```
bin   dev  home  lost+found  proc  sbin  usr
boot  etc  lib   mnt         root  tmp   var
```

- Special directory symbols
  - / : root directory of a disk partition
  - . : current directory
  - .. : previous directory
  - ~ : home directory
- To change directory
  - ➔ cd <dir name>

38

Useful Options for “ls”

- -a
  - do not hide entries starting with ‘.’
- -F
  - append indicator (one of */=@|) to entries
- -l
  - use a long listing format
- -r
  - reverse order while sorting
- -R
  - list subdirectories recursively
- -t
  - sort by modification time
- Tips: useful aliases
  - “ ls -alF ”
  - “ ls -lt ”

39

Important Directories under root dir ‘ / ’

- /bin
  - Store the basic commands that come with the OS
- /etc
  - System setup and configuration files
- /home (or /user)
  - Home directories for users
- /lib
  - System and application libraries, modules, etc
- /mnt (or /media)
  - Mount point entries

40

Important Directories under root dir ‘ / ’

- /root
  - Home directory for “root”
- /sbin
  - System admin commands
- /tmp
  - Temporary directory that everybody can read and write
- /usr
  - Root dir for applications
  - /usr/bin, /usr/local, /usr/include, /usr/lib, /usr/X, etc
- /var
  - Log message, mails, buffers, etc

41

Basic Commands for Processing File/Dir

- ls
- cd
- pwd :
- cp :
- mv :
- rm :
- ln [-s]
- mkdir :
- rmdir :
- cat/more/less :

42

Basic Commands for Processing String/Stream

- echo : display a line of text
- ‘ | ’ : Pipe to pass string from left to right
  - e.g. alias ll 'ls -al \!* | more'
- ‘ > ’ and ‘ >> ’ : redirect output
  - e.g. ls -alF > filelist
- ‘ & ’ (used with ‘ | ’ and ‘ > ’)
  - To include standard error (stderr), i.e. the error or warning messages
- grep/fgrep [options] <PATTERN> <FILEs>
  - Searches the named input FILEs for lines containing a match to the given PATTERN
  - -n : show line number
  - -v : invert match (exclusive)
  - e.g. alias ga 'grep -n \!* * |& grep -v grep | more'
- tee : read from standard input and write to standard output and files
  - e.g. make -f Makefile |& tee make.log
- sort

43

Symbols for (String) Pattern Matching

- ^ : beginning of line
  - e.g. grep ‘^assign’ *.v
- $: end of line
  - e.g. grep ’00);$’ *.v
- * : repeating or not
  - e.g. grep ‘^ *i’ *.v
- . : any character
  - e.g. grep ‘e.*module’ *.v
- \ : escape character
  - e.g. grep '\.v' *.v

44

Single or double quote? ‘ ’ or “ ”

- ‘ ’ : print as is

  “ ” : process special characters such as $, ` `
  - $: print the content of environment variable
    - e.g. $path, $DISPLAY, $cwd
  - ` `: execute the command
    - e.g. `whoami`, `pwd`
- echo “My name is `whoami`”
  - My name is ric

  echo ‘My name is `whoami`’
  - My name is `whoami`

45

Basic Commands for Processes

- ps
  - Report a snapshot of the current processes
  - (e.g.) ps -ef | grep ric
- top
  - Display process status
  - (e.g.) top -u ➔ Display processes evoked by you
- nice : run a program with modified scheduling priority
- renice : set nice values of running processes
- kill [options] : Kill a process
  - (e.g.) kill -9 <running process>
- ‘ & ’ (followed by a command)
  - Put the command to execute in background
- bg / fg
  - Set the command to execute in back or front ground

46

Basic Commands for Network

- rlogin : remote login
  - [Tips] Use with “$HOME/.rhosts”, you can login to remote machine (under same NIS) without being asked password
- rsh : remote shell
  - (e.g.) rsh cad16 cat /etc/password >> cad16_pswd
- telnet : user interface to the TELNET protocol
- ssh : OpenSSH SSH client
- ftp : Internet file transfer program
- sftp : Secure file transfer program
- traceroute <hostname>

47

Basic Commands for Administrations

- su <userID> : switch user
  - su - <userID> : to use <userID>’s shell schript
- sudo : execute a command as another user
  - Determines who is an authorized user by consulting the file “/etc/sudoers”.
    - e.g. “sudo apt-get install xcalc”
  - Recommended; don’t use “su root”
- visudo : edit the sudoers file
- ping <machine>
  - send ICMP ECHO_REQUEST to network hosts
  - To see if a machine is still alive
- finger <user>
  - user information lookup program

48

Basic Commands to Get Information

- date : print or set the system date and time
- cal : displays a calendar
- dmesg : print or control the kernel ring buffer
- ifconfig : show the network configuration
- traceroute[6] : trace the route of an IP(v6)
- nslookup : lookup the IP/domain name

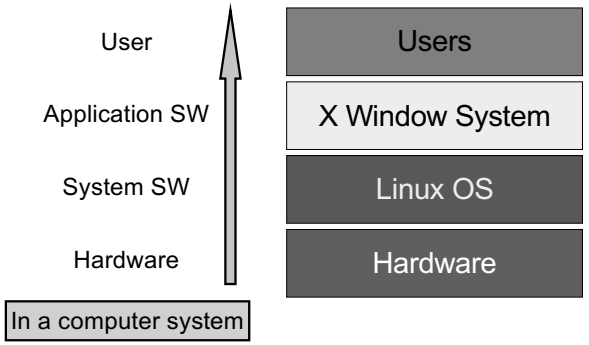
49

Basic Commands to Backup Files/Dirs

- tar [options] <target> [sources]
  - (e.g.) tar zcvf aaa.tgz aaa bbb ccc
  - (e.g.) tar ztvf aaa.tgz // test extract
  - (e.g.) tar zxvf aaa.tgz // extract
- gzip / gunzip
- bzip2 / bunzip2 / bzcat

50

X Window System

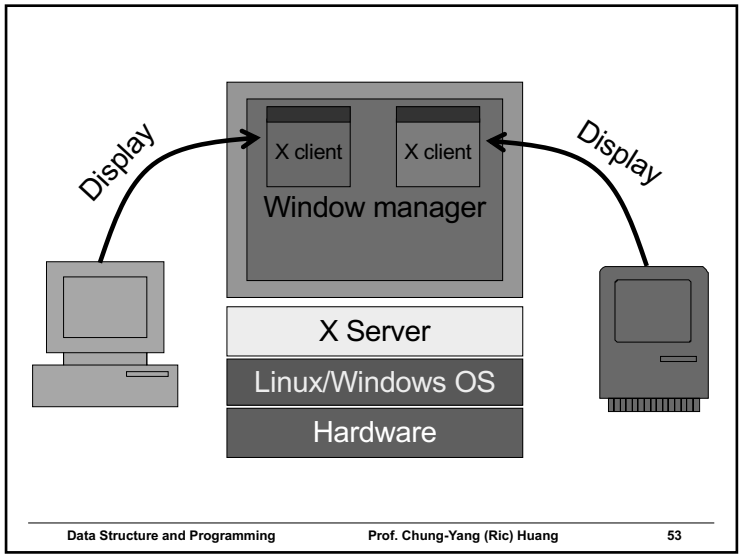


51

X Window System

- First developed by MIT for Unix OS (*free* source code)
- Usually simply mentioned as “X”
- What distinguishes X from most other graphical user interfaces (e.g. Windows) is the “server-client” mechanism

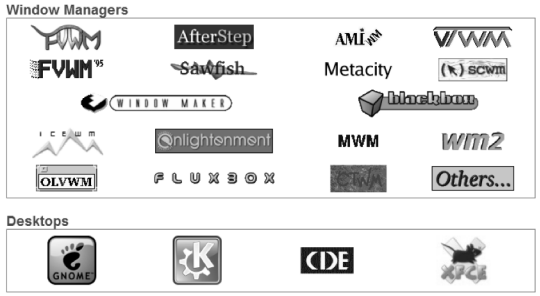
52



53

Various Window Managers

- Like GNU / Linux, many people contribute various window manager programs for *free*
- Many of them are very fancy (e.g. http://xwinman.org/)



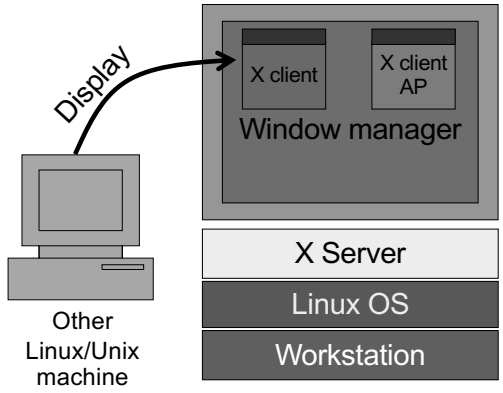
54

X Terminal (or called “xterm”)

- A special X client program that acts as an independent terminal
  - As an input/output terminal
  - Can also launch a new terminal by the command “xterm”
- So, how to execute X window and X terminal on Linux?

55

Scenario #1: Using a Linux machine



56

Using X on a Linux machine

1. Log on a Linux machine using “user name” and “password”
2. Start X window program by the “startx” command
   - Note: some “desktop environment (DE)” starts the X automatically
3. A window manager is executed by the startx script (now you see a GUI like Windows)
4. Execute some X client program (e.g. xterm) locally, or…

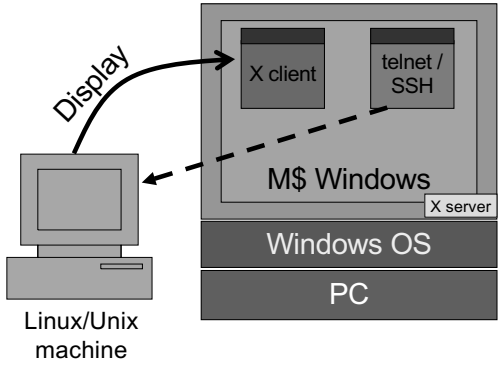
57

Using X on a Linux machine

5. Remote login (e.g. telnet) to other machine, execute a X client program there, and display back to the local X server
   - In local host (running X server program)
     - xhost +[remote machine]
   - In remote machine (assume csh)
     - setenv DISPLAY <local host>:0.0

58

Scenario #2: Using a Windows machine



59

Displaying X back to your Windows PC

- Login to Linux workstation from Windows
  - Start a X server program on your Windows
    - x-win32
    - Xming
    - xQuartz (for Mac)
  - telnet or ssh
- Displaying X back
  1. Enable X forwarding in the ssh setup or
  2. ssh -X username@hostname

60

Other X-related Commands/APs

- xterm : terminal emulator for X
- xdvitune : to tune the video mode under X
- xfig : Facility for Interactive Generation of figures under X11
- xv (or xview) : to view image files under X
- acroread: viewing pdf file
- gv : viewing ps file
- xcalc : calculator
- xclock: clock

61

Basic Linux Survival Guide

- Installation
- Using Linux
  - User accounts
  - File/directory access privileges
  - Useful commands
  - Customize your environment ⇦
- Writing programs on Linux

62

Think: after you log on to Linux…

1. You will see a terminal
2. A command prompt in current directory
   - e.g.
     [ric@localhost ~]$
3. Flashing cursor waiting for you to enter a command

Who is waiting for you?

A shell program.

63

When you log on to a Linux system, a “shell script” will be executed immediately.

But,

what is a “shell”?

(有力可施外殼程式設計…)

64

So, a login shell script is to setup the common environmental variables, command aliases, search paths, etc.

It will be applied to all the terminals you open

65

Check list

1. What type of shell you are using?
2. How to customize your shell environment?
3. How to set up the environment variables?
4. How to create aliases?
5. How to set up your search path (for executable files)?
6. How to set up your "cd" path? (for "cd" command)?

66

Setup your shell (working environment)

1. Choose the shell you like
   - bash (Bourne-Again SHell)
     - Default by many Linux distributors now
   - csh (or tcsh --- turbo csh)
     - Used by many engineers in industry

   ➔ Specified in /etc/password
2. Edit .bashrc or .cshrc (or .tcshrc)
   - Define environment variables
   - Define shell attributes
   - Create command aliases

67

A .tcshrc Example

```
# .tcshrc
setenv PLATFORM `/bin/uname`
setenv LD_LIBRARY_PATH
  /usr/lib:/usr/local/lib
setenv erase ^H
set path=( \
  . /usr/local/bin /bin \
  /usr/bin /usr/sbin /etc )

# Platform-dependent path
if ($PLATFORM == "sunos5")
   then
  set path=( $path \
    /usr/openwin/bin \
    /usr/ucb/bin )
endif
```

```
# Shell attributes
set prompt="{%n@%m}:$cwd>"
set autolist
limit core 0

# Source other scripts
if (-f $HOME/.alias) then
    source $HOME/.alias
endif

if (-f $HOME/.path) then
    source $HOME/.path
endif

if (-f $HOME/.cdpath) then
    source $HOME/.cdpath
endif
```


68

What are environment variables?

- Variables that can be dynamically set to ---
  - Establish some component of the user's working environment
    - e.g. $HOME, $LD_LIBRARY_PATH
  - Affect the way running processes behave
    - e.g. using "getenv()" in C program

1. To set environment variables in shell script
   - *setenv <variable> <value>*
2. To list all environment variables
   - *env*
3. To show one environment variable
   - *echo $<variable>*

Creating command aliases

- Usefulness: (e.g.) aliasing commonly-used options into commands

  e.g.

  alias ls 'ls -F'
  alias ll 'ls -al \! | more'*
  alias k 'kill -9'
  alias so 'source ~/.tcshrc'
  alias rm 'rm -i'

- Where to define the aliases ---

  [option 1] Include "*alias*…" commands in shell script

  [option 2] Create a file (e.g. ".alias") and call it when running shell script ← recommended

Creating .alias

- In your shell script, do (for example):

  > if (-f $HOME/.alias) then
  > source $HOME/.alias
  > endif

  [Note] *source*: to execute a script

- In file .alias, list all the aliases you want

An Exemplar .alias file

```
alias ls    'ls -F'
alias ll    'ls -al \!* | more'
alias la    'ls -a'
alias gc    'grep -n \!* *.C *.h
             *.c *.cpp *.cxx | more'
alias grc   'grep -n \!* */*.C
            */*.h */*.c */*.cpp */*.cxx |&
            grep -v grep | more'
alias G++   'g++ -g -Wall -o \!*
             \!*.cpp'
alias k     'kill -9'
alias top   "top -i"
alias mps   'ps -ef | grep \!* |
             more'
alias cd    'cd \!*;set
prompt="{%n@%m}:$cwd>"'
alias pushd 'pushd \!*; set
prompt="{%n@%m}:$cwd>"'
alias popd  'popd; set
prompt="{%n@%m}:$cwd>"'
alias wcc   "wc *.h *.C *.c
             *.cpp *.cxx"
alias wcd   'foreach dir (\!*)
{ wc $dir/*.h $dir/*.C $dir/*.c
$dir/*.cpp $dir/*.cxx }'
alias vi    vim
```

Define Shell Attributes

- Goal: to improve shell look-and-feel and setup useful shell properties

  e.g.
  - set prompt = " {%n@%m}:$cwd> "
  - set autolist
  - limit core 0
- [Useful] 2 shell attributes to specify paths
  1. set path = ($path . /bin /usr/bin)
     - To find the commands or executable programs
  2. set cdpath = (. $HOME .. $HOME/tests)
     - Search path for command "cd"

  ➔ Can be specified in files ".path" and ".cdpath"

csh/tcsh vs. bash

csh/tcsh	bash	說明
setenv A yy	export A=yy	將環境變數 A 設為 yy
alias A yy	alias A=yy	將 A 設為 yy 的 alias
source xx	source xx . xx	執行 xx 這個 script
& 在 > 之後	>& or &>	redirect from stderr
& 在 | 之後	2>&1 |	pipe from stderr
\!*	(??)	取得所有 arguments, 如 alias G++ 'g++ -o \!* \!*.cpp'

To learn more about "bash": http://linux.vbird.org/linux_basic/0320bash.php

.(t)cshrc vs. .bashrc

設定游標前之提示字串為 "{username@hostname}:currentDirectory>" 的形式

- For .(t)cshrc
  - `set prompt="{%n@%m}:$cwd>"`
- For .bshrc
  - ```
    if [ -f /etc/bashrc ]; then
        . /etc/bashrc
        PS1="{\u@\h}:\w>"
    fi
    ```


OK, hopefully by now you've got the idea of how the Linux machine works

The next task is:

how to write program on it?
    ```

Writing Programs on Linux

- Text editor (e.g. vim)
- C/C++ compiler (e.g. g++)
- Makefile
- Debugger (e.g. gdb or ddd)

77

Text Editors under FSF

- There are many text editors under FSF...
- Categories within Text creation and manipulation
  - Fonts
  - Font related software
  - Editors
    - 127 projects(tools)
    - 4 subcategories
  - Dictionaries
  - Word processing
  - Documentation tools
  - Misc

78

The first thing you need to know about an editor…

- There are (at least) two modes
  - "insert" and "command" modes
  - ➔ Why??
- Insert mode
  - To insert or append text in the editor
- Command mode
  - To move your cursor
  - To copy, paste, delete text
  - To open, close, save a file
  - To search for strings
  - Others: such as MACROs

79

Switching between modes in vi/vim

- File open：vim test.txt
- Default is in "command mode"
- To enter "insert" mode
  - i : to insert at current cursor position
  - I : to insert at the beginning of line
  - o : to open a new line below for insertion
  - O : to open a new line above for insertion
  - a : to append text to the next of the current cursor position
  - A : to append text to the end of line
  - R : to replace text at current cursor position
- To go from insert mode to command mode
  - [Esc]

80

Command Mode: Browsing

- h(←) j(↓) k(↑) l(→) : Move cursor
- Ctrl + [f b d u] : Page scrolling
  - Forward / Backward 1 page
  - Down / Up half page
- + / - : to the next/prev non-empty line
- 0 / $: to the beginning / end of the line
- w / b : Forward / Backward to the beginning of the next word
- H / M / L : to the top / middle / bottom of the screen
- G : to the end of the file

81

"Number" in Command Mode

- Number in command mode usually means "times the command repeated"
  - e.g. <num> <space>
    - ➔ move <num> spaces to the right
  - e.g. <num> <enter>
    - ➔ move <num> lines forward

82

Command Mode : Delete and Replace

- x / X : delete / backspace
  - 3x : delete 3 characters
- dd : delete current line
- dw : delete until next word
- dG : delete till the end of file
- cw = dw → i
- r<char> : replace with this character
- R : replace until <Esc> is pressed

83

Command Mode : Copy and Paste

- yy : copy current line
  - 3yy : copy 3 lines
- p / P : paste under / above current line

84

Single line command mode (:)

➔ Single line command at the end of page

- w : write w! : force write
- q : quit q! : force quit
- <num>: go to <num> line
- $: go to last line
- <l1>, <l2> s/<word1>/<word2>/[gc]
  - Replace <word1> in lines l1 ~ l2 by <word2>
  - ^<word> : <word> must appear in the beginning of line
  - <word>$: <word> must appear at the end of line
  - g : allow more than 1 replaces in a line
  - c : confirm before replacing
  - e.g.: 1, $ s/torch/touch/g
- e <filename> : edit another file

85

Other Special Modes

- Visual mode
  - <Ctrl> + v : mark a rectangular block (by moving cursor)
  - <Shift> + v : mark a block (line-based)
  - v: mark a continuous block (by moving cursor)
- Search mode
  - / : search by string forward
  - ?: search by string backward
  - n / N : find the next /previous one

86

Other Commands

- J : join the next line
- u / <ctrl>+r : undo / redo
- q<char> : record command MACRO <char>
  - q : stop recording
- @<char> : replay recorded MACRO <char>
- :set (no)nu : (un)display line numbers
- :set (no)hlsearch : (no) highlight on the matched strings
- :!<shell command> : shell escape
- :sp : split parallel :vs : vertical split
  - <Ctrl>+ww : move between windows
  - :q : quit current window

87

Writing Programs on Linux

- Text editor (e.g. vi)
- C/C++ compiler (e.g. g++)
- Makefile
- Debugger (e.g. ddd)

88

Some Basics You Need to Know...

- In the coming homework assignments, you will be dealing with SW developments on multiple files ---
  - .h (header) files
    ➔ class declaration/definition, function prototype
  - .cpp (source) files
    ➔ class and function implementation
  - Makefiles
    ➔ scripts to build the project

Why header files?
Why multiple source files?

- Basic coding disciplines

1. Place the class, global variable and function definitions in header files
   - To share with different programmers
   - To facilitate class browsing and reuse
2. Break a long source code into multiple meaningful source files
   - Easier to read and maintain
   - Easier to debug
   - Faster compilation

A simple example

```
// a.h
class A
{
public:
   void f();
};
```

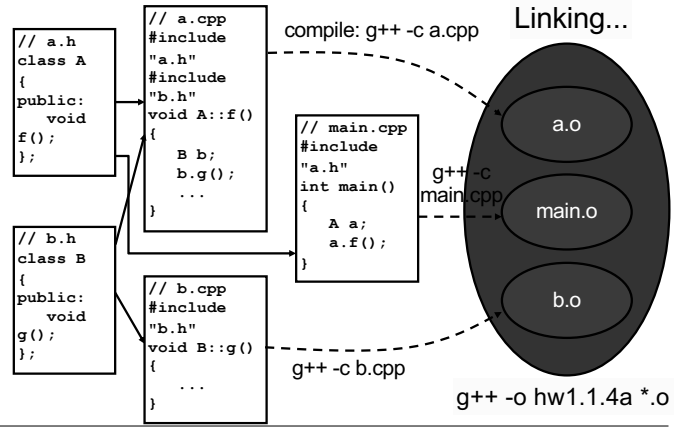
```
// a.cpp
#include "a.h"
#include "b.h"
void A::f() {
   B b;
   b.g();
   ...
}
```

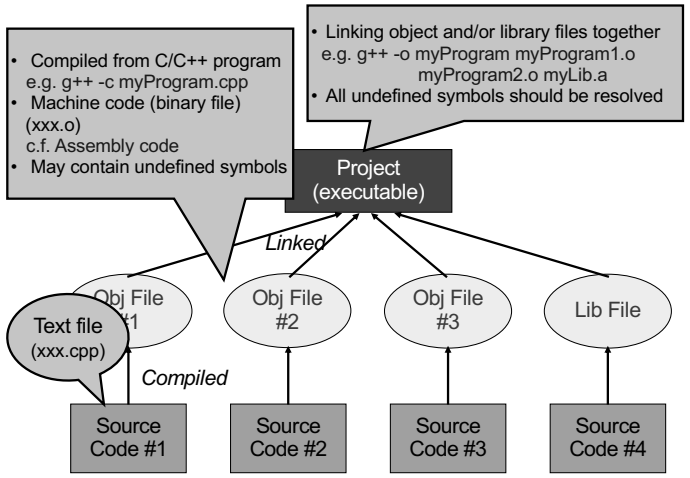
```
// main.cpp
#include "a.h"
int main()
{
   A a;
   a.f();
}
```

```
// b.h
class B
{
public:
   void g();
};
```

```
// b.cpp
#include "b.h"
void B::g() {
   ...
}
```


Compiling the simple example





93

C++ Compiler : g++

- g++ -c my_code.cpp
  compile only ➔ my_code.o
- g++ -o my_executable my_code1.o my_code2.o
  ➔ link objs to executable
  - "-o" specifies the output executable file name. Default is "a.out"
- g++ -o my_executable my_code.cpp
  ➔ Directly compile and build executable from source code
  [Recommend] alias G++ 'g++ -g -Wall -o \!* \!*.cpp'

94

g++ Flags

- -g : Debugging mode
- -D<MACRO> : Define MACRO
- -W : Show the compilation warning message
  - e.g. -Wall
- -L : Search directory path in linking
- -I : Search directory path for inclusion
- -l<libname>: Specify library for linking
- -O[num] : Optimization
  - Removing debugging information
  - Code will be smaller

95

Linking with library files

- **Think ---**
  You wrote a package (e.g. a matrix solver), and you want to share it with others. Clearly, they should have their own "main()" and then just include your package when creating the executable...
  ➔ Send them all the source codes? What if you don't want to...?
  ➔ Send them all objective files? Better creating a tarball?
  ➔ Create a library archive!!
- Command:
  - ar [cr] libraryName <objFiles>
- Instantiation:
  - g++ ... -L$(LIBPATH) -larchive ...
  - e.g.
    let libraryName = libabc.a in the same directory
    ➔ LIBPATH = ./
    ➔ g++ ...-L $(LIBPATH) -labc ...

96

Order of library and object files

- "man ld"...

1. **The linker will search an archive only once, at the location where it is specified on the command line.**
2. **If the archive defines a symbol which was undefined in some object which appeared before the archive on the command line, the linker will include the appropriate file(s) from the archive.**
3. **However, an undefined symbol in an object appearing later on the command line will not cause the linker to search the archive again.**

➔ In other words, say a obj xyz.o instantiates a function which is defined in the lib libabc.a.
[Question]
g++ ... xyz.o -labc ... or
g++ ... -labc xyz.o ... ??

97

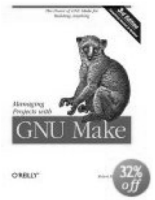
Creating executable from multiple source files

- With multiple header and source files, creating the executable seems to require multiple commands…
- Can we create a "project" and use a single command to generate the executable?

98

Building Up a Project Using "make"

- ***make*** is a command generator. Using a description file and some general templates, it creates a sequence of commands for execution by the UNIX shell.



- Advantages
  1. Save time in compilation (dependency checking & minimizing rebuilds)
  2. Simplify the command (macros & suffix rules)
  3. For better SW management

99

Format of a Basic Makefile

- The basic makefile contains five entries ---
  <objective>:<list of dependencies>
  <tab><Rule>
  - The first line is called dependency line or rules line
    - Objective is rebuilt only if any of the dependencies is newer
  - The second line is command line
- Example:

```
run: main.o window.o
   g++ -o run main.o window.o
main.o: main.cpp window.h
   g++ -c main.cpp
window.o: window.cpp window.h
   g++ -c window.cpp
```

To build ---
> make

100

Notes about Makefile

- The "<tab>" can't be replaced with "spaces"
  - If you want to check the tabs in your makefile, issue the command ---
    - $cat –v –t –e makefile
    - The -v and -t option cause all tabs to appear as ^ I, and the -e option places a dollar sign($) at the end of each line
- The filename of makefile
  - Makefile/makefile: just type "make"
  - Others (e.g. myMake): "make –f myMake"

101

Example of a Makefile (multiple targets)

```
plot_prompt : basic.o prompt.o
  cc –o plot_prompt basic.o prompt.o
plot_win : basic.o window.o
  cc –o plot_win basic.o window.o
basic.o : basic.c
  cc –c basic.c
prompt.o : prompt.c
  cc –c prompt.c
window.o : window.c
  cc –c window.c
```

```
%> make
%> make plot_prompt
%> make plot_win
```


102

More Rules for Makefile

- # : designates the beginning of a comment
- We can continue a long line by placing a backslash(\ ) at the end
- Target without prerequisites
  - e.g.

```
clean:
        /bin/rm -f *.o
```


103

MACROs for "make"

- <MACRO> = <definition>
  - Usually help to simplify the makefile
  - e.g.
    - objs= main.o input.o data.o
    - CC=/usr/fred/bin/cc
- MACRO Priority (least to highest)
  1. Internally Defined Macros (default)
  2. Shell Variables (setenv, set in xx.rc file)
  3. Description file itself (makefile)
  4. Macros Definitions on the Command Line (make –DXXX)

104

Use of MACRO

- Enclose the macros either in parentheses or in curly braces (i.e. ”()”or”{}”), and precede it with a dollar sign($)
  - $(abc), $(AdG_fd55),${as_Ek478yy}…etc.
    - ➔ p.s. The curly braces”{ }” is recommended.
  - Single-character macro names do not require either parentheses or braces, but it is good habit to use them (e.g. $d, $(d), ${d})
  - If you want to use dollar sign($), you must use “$$”
    - p.s.“$” will be used in shell command.

105

Example of Using MACRO

```
# This is the Makefile of menu
CC = gcc
CFLAGS = -DDEBUG -c
LIBS = -lncurses
INCLUDE = -I/usr/src/menu/include
EXEC = menu

all: clean install
install: $(EXEC)
   chmod 750 $(EXEC)
   cp $(EXEC) /usr/bin
$(EXEC): menu.o
   $(CC) -o $@ $? $(LIBS)
menu.o:
   $(CC) $(CFLAGS) -o $@ menu.c $(INCLUDE)
utils.o:
   $(CC) $(CFLAGS) -o $@ utils.c $(INCLUDE)
clean:
   -rm *.o
   -rm *~
```

What will happen if we call “make -DEXEC=hihi”

106

Internally Defined Macros(default)

- ${CC} : C compiler.
- $@ : the current target.
  - plot_prompt : basic.o prompt.o
  - ${CC} –o $@ basic.o prompt.o
- $? : a list of prerequisites that are newer than the current target.
  - libops :interact.o sched.o gen.o
  - ar r $@ $?

107

Suffix Rules

- Observation:
  Most of the activities that it describes—compiling and assembling—are governed by certain conventions, and therefore can be carried out by make under a set of default rules. Relying upon these rules, we can simplify our makefile.
  - .SUFFIXES : .o .c .s
  - .c.o :
  - ${CC} ${CFLAGS} -c $<
  - .s.o :
  - ${AS} ${ASFLAGS} -o $@ $<
  - OBJS=main.o iodat.o dorun.o lo.o
  - LIB=/usr/proj/lib/crtn.a
  - program : ${OBJS} ${LIB}
  - ${CC} -o $@ ${OBJS} ${LIB}
- p.s. Here $< has a meaning akin to $?, except that $< can be used only in suffix rules.

108

More about Makefile...

- Will be covered in Homework and Final Project.
- It's much more complicated, but very useful. Will talk about it later...

109

Writing Programs on Linux

- Text editor (e.g. vi)
- C/C++ compiler (e.g. g++)
- Makefile
- Debugger (e.g. ddd)

110

Don't say that you have taken DSnP…

If you never use the debugger!!!!

111

Debugger vs. cout

- Using "cout" to debug is easy, and sometimes (often) useful
  - But it cannot work well for tricky corner-case bugs
  - ➔Cannot set break point
  - ➔Cannot undo
  - ➔Cannot display the value of an arbitrary variable
  - ➔Cannot force some value of an variable
  - ➔Always need to recompile

112

C/C++ Debuggers on Linux

- gdb: a text mode debugger
  - Can work with xemacs for editing, compiling, debugging, etc
  - GUI: ddd (Make sure you select this package when installing Linux)
- ddd: a graphic mode debugger
- [Note] When you use debugger, make sure you compile with "-g" flag and no "strip"
- Basic usage
  - run, step, next, break, continue, until, finish, up/down, print, (graphical) display, list, undo/redo, quit
  - (mouse over variable) value pop-up
- Advanced usage
  - watch, break if, (used with cout/printf) exam the watch point status, exam the memory contents, etc...

113

gdb commonly used commands

- r [arguments]: run the program with [arguments]
- n: execute the next line of the program
- s: step into the function under current line
- b [funcName | lineNo]:
  set break point on <funcName> or <lineNo>, break on current line if no parameter is specified.
  - e.g.
    b BSTree::insert
    b adtTest.h:106
- del <breakPointId>: delete the break point by ID
- c: continue the execution to the next break or watch point
- info breakpoins: show the information of breakpoints (e.g. how many times breakpoints have been visited)
- p <var>: print the content of the variable
- p* <var>: print the content of the memory pointed by the variable

114

gdb commonly used commands

- display <var>: display the content of the variable
- display* <var>: display the content of the memory pointed by the varaible
- undisplay <displayId>: undisplay by ID
- up: move up one level of source code (for viewing; won't affect the execution)
- down: move down one level of source code (for viewing)
- where: show the calling trace to the current line
- dir <dirName>: add <dirName> for the look-up of the source code
- <Ctrl-c>: force to break the execution
- q: quit the debugger

115

gdb commonly used commands

- call <funcName>: call the function
- set <varName> = <value>: force to set the variable value
- watch <expression>: set the watch point (used with 'continue')
  - e.g. watch (idx = 10) // note: not "=="
- tbreak [funcName | lineNo]: set the temporary break point. Voided after the first encounter.
- b if <expression>: conditional break point
- p/display $<varId>: print/display the <varId> just printed

116

gdb tips

- You can add a number to most of the commands to specify the "repeating times".
  - For example, "n 10" will execute the next 10 lines of codes. "c 100" means
  - you will ignore the next 99 break points and stop at the 100th one.
- What happens if you don't know how many times of breakpoints are encountered before program crashes?
  - Set a break point; continue a BIG number and make it crash;
  - [ddd] open "data watch" window
  - [gdb] "info breakpoints"

117

What we have learned today...

- History of Linux OS
- Linux survival guide
  - Installation
  - User accounts
  - File/directory access privileges
  - Useful commands
  - Customize your environment
- Programming on Linux
  - Text editor (e.g. vi)
  - C/C++ compiler (e.g. g++)
  - Makefile
  - Debugger (e.g. ddd)

118