1. Programming at Linux

Data Structure and Algorithms

Hanjun Kim

Compiler **O**ptimization **Re**search **Lab**

Yonsei University

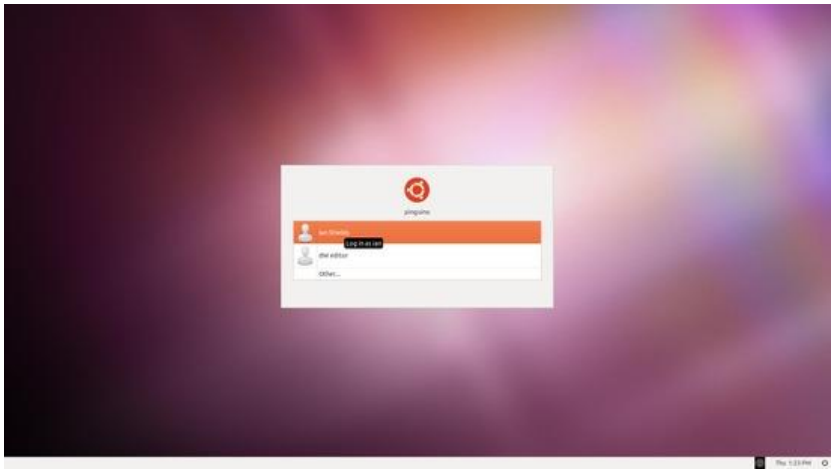
Basic of Linux Systems

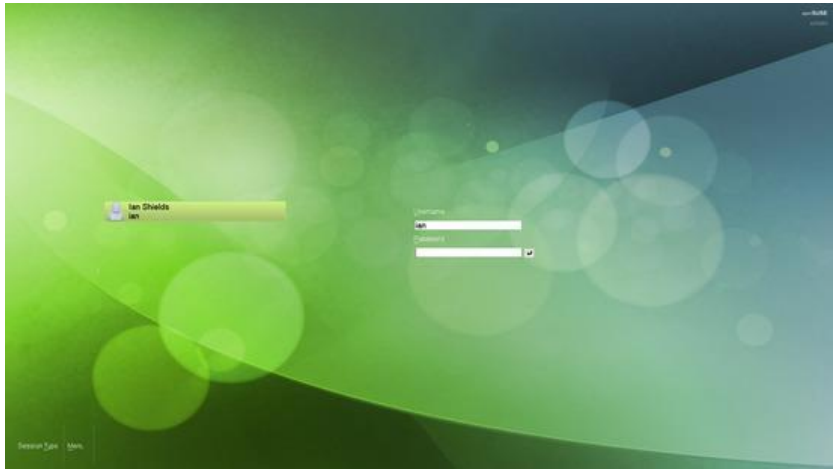
What is Linux?

- An Operating System
  - Unix-like and POSIX compliant
    - POSIX: **P**ortable **O**perating **S**ystem **I**nterface
  - Free and Open Source Software
  - Leading Operating System on Servers and Supercomputers
    - Since Nov. 2017, all the world's 500 fastest supercomputers run some variant of Linux

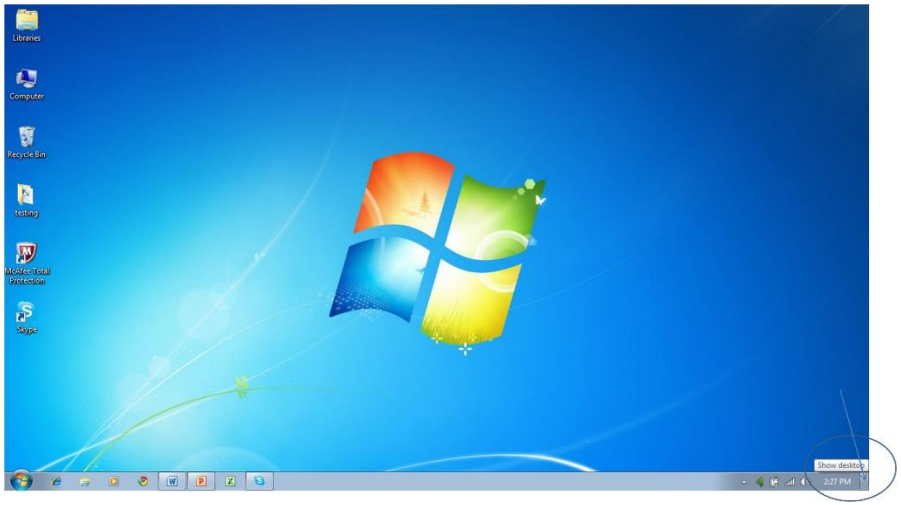
- Many popular distributions
  - Debian, Ubuntu, Linux Mint, Fedora, Arch Linux, …
  - Commercial: Red Hat Enterprise Linux, SUSE Linux Enterprise Server
  - Embedded Systems: Android, Tizen, …

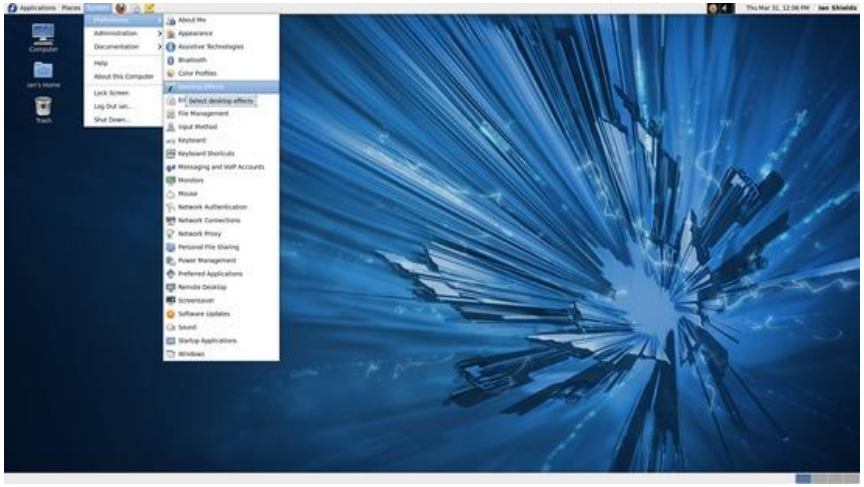
Is Linux Difficult?



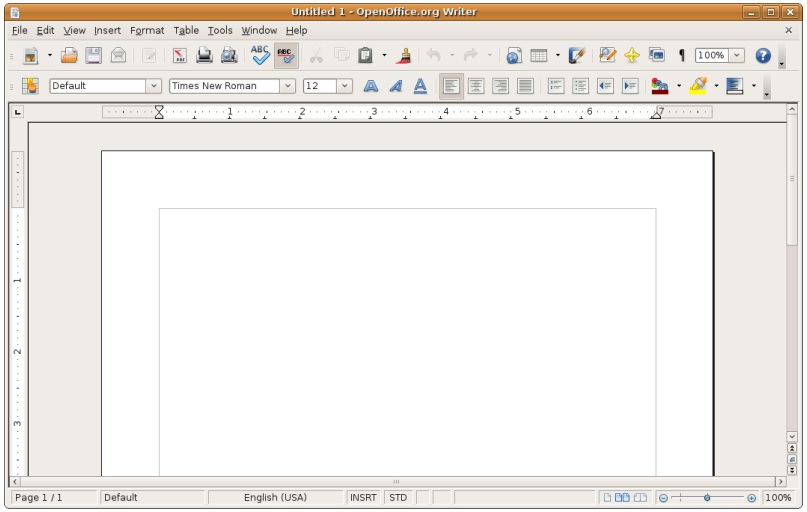


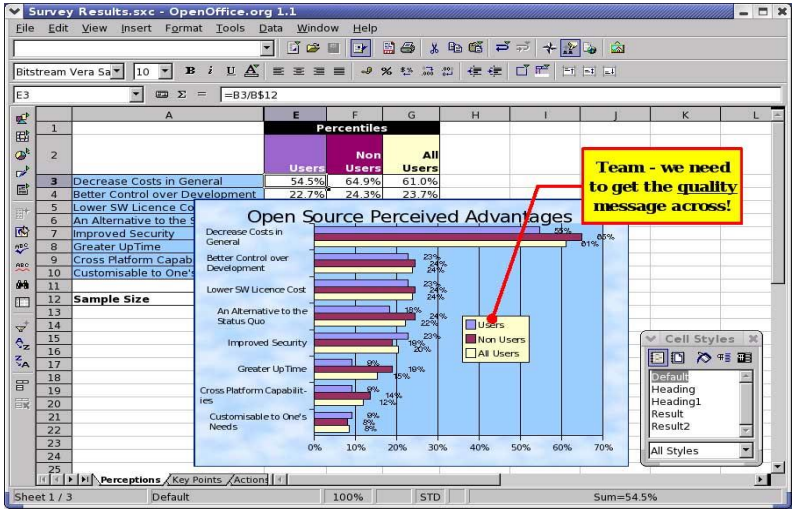
Is Linux Difficult?

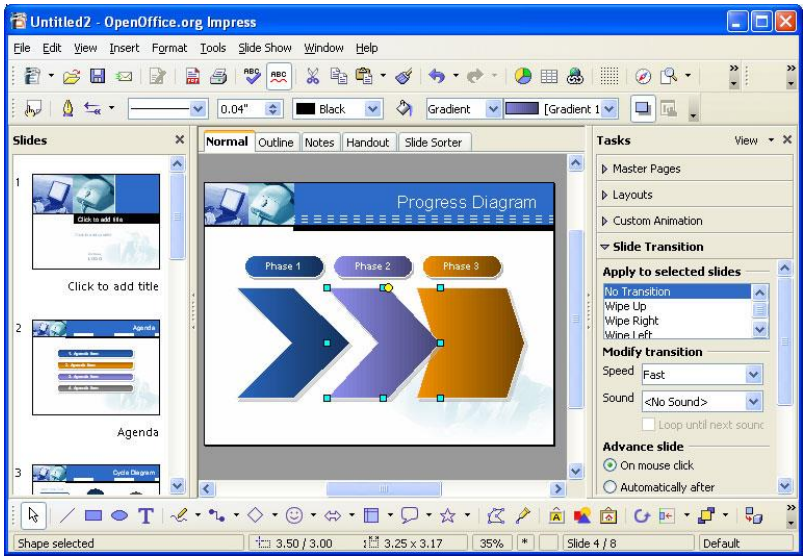


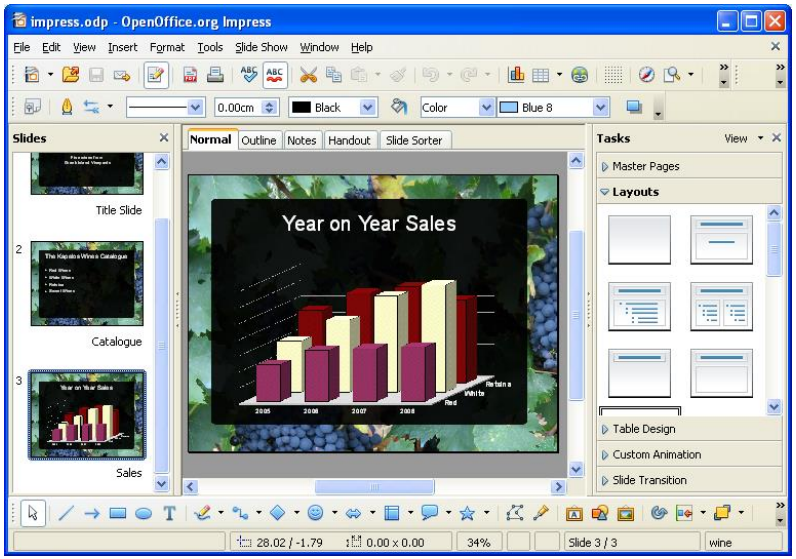


Is Linux Difficult?









Does Linux console look difficult?

```
drwxr-xr-x  25 root   root     4096 Oct  5 19:56 ./
drwxr-xr-x  25 root   root     4096 Oct  5 19:56 ../
-rwxr--r--   1 root   root     8192 Oct 13 00:17 aquota.user*
-rw-r--r--   1 root   root        0 Oct  5 19:56 .autofsck
-rw-r--r--   1 root   root        0 Jul  7 20:43 .autorelabel
drwxr-xr-x   2 root   root     4096 Oct  1 04:02 bin/
drwxr-xr-x   4 root   root     1024 Jul  8 09:43 boot/
drwxr-xr-x  10 root   root     3460 Oct  5 19:57 dev/
drwxr-xr-x  98 root   root    12288 Oct 13 13:25 etc/
-rw-r--r--   1 nobody nobody     19 Jul  8 11:13 .forward
drwx------   2 root   root     4096 Jul  7 15:54 .gnupg/
drwxr-xr-x  11 root   root     4096 Oct 12 11:53 home/
drwxr-xr-x  14 root   root     4096 Oct  4 04:02 lib/
drwx------   2 root   root    16384 Jul  7 15:30 lost+found/
drwxr-xr-x   2 root   root     4096 Oct 11  2006 media/
drwxr-xr-x   2 root   root        0 Oct  5 19:56 misc/
drwxr-xr-x   2 root   root     4096 Oct 11  2006 mnt/
drwxr-xr-x   2 root   root        0 Oct  5 19:56 net/
-rw-------   1 root   root      753 Jul  7 16:35 nohup.out
drwxr-xr-x   6 root   root     4096 Jul  8 10:01 opt/
dr-xr-xr-x 172 root   root        0 Oct  5 22:56 proc/
-rwxr--r--   1 root   root       32 Jul  8 09:46 quota.user*
-rw-------   1 root   root     1024 Jul  7 16:11 .rnd
drwxr-x---  10 root   root     4096 Oct 13 13:04 root/
drwxr-xr-x   2 root   root    12288 Oct  4 04:02 sbin/
drwxr-xr-x   5    500    500 24576 Oct 13 00:17 scripts/
drwxr-xr-x   2 root   root     4096 Jul  7 20:33 selinux/
drwxr-xr-x   2 root   root     4096 Oct 11  2006 srv/
drwxr-xr-x  11 root   root        0 Oct  5 22:56 sys/
drwxrwxrwt   5 root   root   733184 Oct 13 13:28 tmp/
drwxr-xr-x  17 root   root     4096 Jul  8 09:43 usr/
drwxr-xr-x  28 root   root     4096 Jul  8 09:43 var/
root@theserver [/]#
```

Windows are the same

C:\Windows\system32\cmd.exe

```
07/29/2009  08:50 PM    <DIR>          Desktop
09/01/2009  06:56 PM    <DIR>          Documents
09/01/2009  06:32 PM    <DIR>          Downloads
08/28/2009  02:11 PM    <DIR>          ErgoEmacs_Source
05/23/2009  09:56 PM    <DIR>          Favorites
05/23/2009  07:23 PM    <DIR>          Links
08/05/2009  03:40 PM    <DIR>          Mail
05/25/2009  10:31 AM    <DIR>          Movies
06/26/2009  04:54 PM    <DIR>          Music
06/26/2009  01:48 PM    <DIR>          music to sort
08/16/2009  06:18 PM    <DIR>          na_xruti
09/02/2009  10:31 AM    <DIR>          Pictures
07/28/2009  01:01 PM    <DIR>          PowerShell scripts
08/09/2009  05:01 PM    <DIR>          programs
07/28/2009  01:00 PM    <DIR>          ps_scripts
06/09/2009  05:17 PM    <DIR>          Saved Games
05/23/2009  07:23 PM    <DIR>          Searches
07/10/2009  08:24 PM    <DIR>          Tracing
06/17/2009  01:12 AM           129,063 unison.log
09/01/2009  12:32 PM    <DIR>          Videos
08/25/2009  05:37 PM    <DIR>          web
               18 File(s)        246,149 bytes
               36 Dir(s)  474,250,076,160 bytes free

C:\Users\xah>
```

Why Command line mode?

- Much more control of the system
- Much less resource required
- Faster usage
- Faster remote access
- Less diverse – Just need to learn one system!
- Less movement of hands

Let’s Install SSH Terminal

- SSH
  - Cryptographic network protocol for secure data communication between two networked computers
  - Usages
    - Remote command-line login
    - Remote command execution
    - Other secure network services
- SSH Terminals
  - PuTTY
    - http://www.chiark.greenend.org.uk/~sgtatham/putty/
  - Xshell 6
    - http://www.netsarang.com/products/xsh_overview.html
    - Download the free license version for Home & School users

Login to the class server

ssh <yourID>@class.corelab.or.kr

- Ex: ssh –p 20202 y2019142001@class.corelab.or.kr
- It will ask your password
- Type your password (It does not show ******)

```
$ ssh hanjun@class.corelab.or.kr
hanjun@class.corelab.or.kr's password:
Welcome to Ubuntu 13.10 (GNU/Linux 3.11.0-12-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Mar  4 15:35:49 KST 2014

  System load:    0.0             Processes:          256
  Usage of /home: 0.1% of 10.83TB   Users logged in:    0
  Memory usage:   1%              IP address for em1: 141.223.99.108
  Swap usage:     0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

34 packages can be updated.
31 updates are security updates.

Last login: Tue Mar  4 15:35:51 2014 from www.corelab.or.kr
hanjun@ubuntu:~$
```

Shell

- Shell interprets the command and request service from kernel
- Similar to DOS but DOS has only one set of interface while Linux can select different shell
  - Bourne Again shell (Bash), TC shell (Tcsh), Z shell (Zsh)
- Different shell has similar but different functionality
- Bash is the default for Linux
- Graphical User Interface (GUI) of Linux is in fact an application program work on the shell

Change Password

$ passwd

```
$hanjun@ubuntu:~$ passwd
Changing password for hanjun.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
```

Logout

$ logout

```
hanjun@ubuntu:~$ logout
Connection to class.corelab.or.kr closed.
$
```

Fish & Fishing

- Manpage
  - $ man ls
  - $ man 2 mkdir
  - $ man man
  - $ man -k mkdir
- Googling! ☺
- Manpage sections
  - 1 User-level cmds and apps
    - /bin/mkdir
  - 2 System calls
    - int mkdir(const char *, …);
  - 3 Library calls
    - int printf(const char *, …);
  - 4 Device drivers and network protocols
    - /dev/tty
  - 5 Standard file formats
    - /etc/hosts
  - 6 Games and demos
    - /usr/games/fortune
  - 7 Misc. files and docs
    - man 7 locale
  - 8 System admin. Cmds
    - /sbin/reboot

Basic Commands

- ls
  - $ ls -l
  - $ ls -a
  - $ ls -la
  - $ ls -l --sort=time
  - $ ls -l --sort=size -r
- cd
  - $ cd /usr/bin
- pwd
  - $ pwd
- ~
  - $ cd ~
- ~*user*
  - $ cd ~hanun
- What will “cd ~/hanjun” do?

- which
  - $ which ls
- whereis
  - $ whereis ls
- locate
  - $ locate stdio.h
  - $ locate iostream
- find
  - $ find / | grep stdio.h
  - $ find /usr/include | grep stdio.h

Basic Commands

- echo
  - $ echo “Hello World”
  - $ echo -n “Hello World”
- cat
  - $ cat /etc/modules
  - $ cat /proc/cpuinfo
- cp
  - $ cp foo bar
  - $ cp –R foo bar
- mv
  - $ mv foo bar
- mkdir
  - $ mkdir foo
- rm
  - $ rm foo
  - $ rm -rf foo
  - $ rm -i foo
  - $ rm -- -foo
- chmod
  - $ chmod 755 ~/public_html
- chgrp
  - $ chgrp bar /home/foo
- chown
  - $ chown -R foo:bar /home/foo

Basic Commands

- tar
  - $ tar cvfp lab1.tar lab1
- gzip
  - $ gzip -9 lab1.tar
- untar & ungzip
  - $ gzip -cd lab1.tar.gz | tar xvf –
  - $ tar xvfz lab1.tar.gz
- touch
  - $ touch foo
  - $ cat /dev/null > foo
- Pipe
  - $ cal > foo
  - $ cat /dev/zero > foo
  - $ cat < /etc/passwd
  - $ who | cut -d’ ‘ -f1 | sort | uniq | wc –l
- echo
  - $ echo “The date is `date`”
  - $ echo `seq 1 10`
- Hard, soft (symbolic) link
  - ln vmlinuz-2.6.24.4 vmlinuz
  - ln -s firefox-2.0.0.3 firefox
- Disk usage
  - $ df -h /
- File space usage
  - $ du -sxh ~/

vi

- 2 modes
  - Input mode
    - ESC to back to cmd mode
  - Command mode
    - Cursor movement
      - h (left), j (down), k (up), l (right)
      - ^f (page down)
      - ^b (page up)
      - ^ (first char.)
      - $ (last char.)
      - G (bottom page)
      - :1 (goto first line)
    - Switch to input mode
      - a (append)
      - i (insert)
      - o (insert line after
      - O (insert line before)
- Delete
  - dd (delete a line)
  - d10d (delete 10 lines)
  - d$ (delete till end of line)
  - dG (delete till end of file)
  - x (current char.)
- Paste
  - p (paste after)
  - P (paste before)
- Undo
  - u
- Search
  - /
- Save/Quit
  - :w (write)
  - :q (quit)
  - :wq (write and quit)
  - :q! (give up changes)

HTML

- Example Code
  - <html>
    <body>

    <h1>My First Heading</h1>

    My first paragraph.

    </body>
    </html>

- Nested Structure

```
<html>
    <body>
        <h1> My First Heading </h1>

        <p> My first paragraph </p>
    </body>
</html>
```

HTML

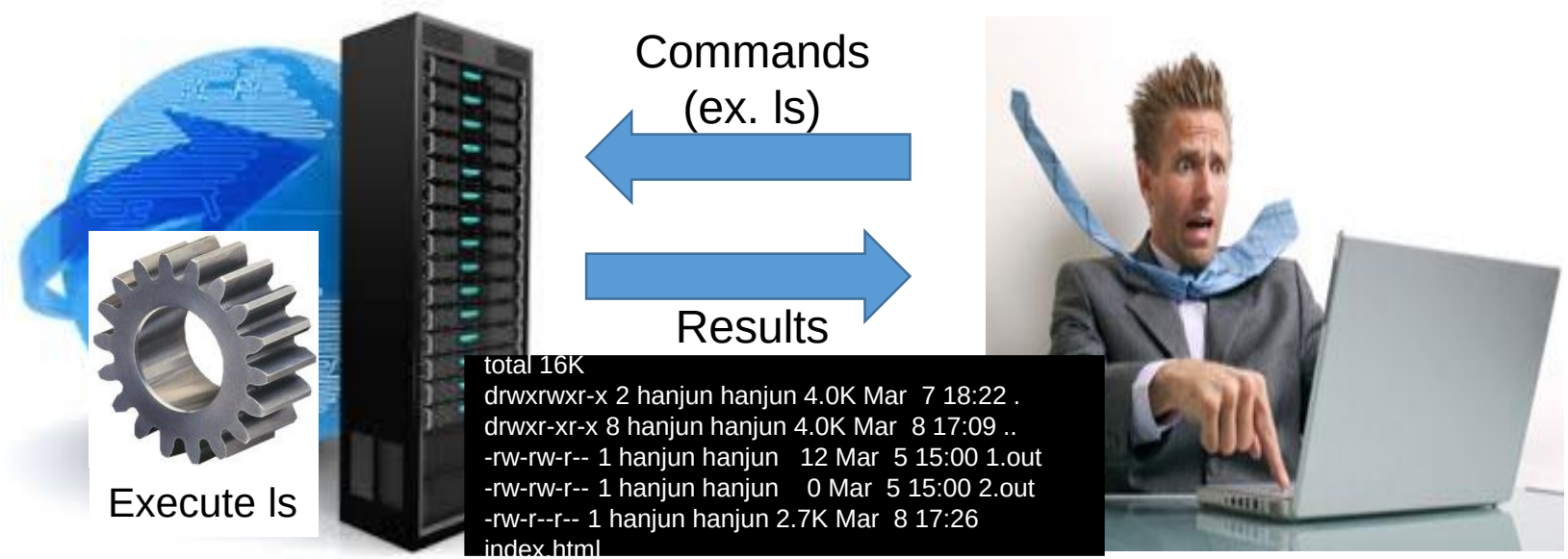
- HTML Head
  - <head>
    <title>Title of the document</title>
    </head>
- HTML Link
  - This is a link to EE
- HTML Image
  - <img src="your_image.jpg" alt="image_name" width="104" height="142">
- HTML Comment Tags
  - <!-- This is a comment -->

HTML

- HTML Table
  - <table style="width:300px">
    <tr>
      <td>Jill</td>
      <td>Smith</td>
      <td>50</td>
    </tr>
    <tr>
      <td>Eve</td>
      <td>Jackson</td>
      <td>94</td>
    </tr>
    </table>

Basic of Programming at Linux

How are Server & Client working?



Servers (@school)

Clients (@home)

Execute Hello.c

```
#include <stdio.h>

int main(void)
{
    printf("Hello, world!₩n");
    return 0;
}
```

Compile

```
0101010101001101010….
…
…
…
1101010101101000101
```

Compile Process

```
#include <stdio.h>

int main(void)
{
        printf("Hello, world!₩n");
         return 0;
}
```

hello.c
Source code
C language
Contains preprocessor directives

```
…
int printf(char* format, …);
…
```

stdio.h

C Preprocessor

Preprocess
gcc –E hello.c > hello.i

```
…
int printf(char* format, …);
…

int main(void)
{
        printf("Hello, world!₩n");
         return 0;
}
```

hello.i
Source code
C language
Contains declaration of printf() function
Missing definition of printf() function

Compile Process

```
…
int printf(char* format, …);
…
int main(void) {
        printf("Hello, world!₩n");
         return 0;
}
```

hello.i
Source code
C language
Contains declaration of printf() function
Missing definition of printf() function

C Compiler

Compile
gcc –S hello.i

```
.section .rodata

cGreeting:
 .asciz "hello, world₩n"
 .section .text
 .globl main
 .type main,@function

main:
 pushl %ebp
 movl %esp, %ebp
 pushl $cGreeting
 call printf
 addl $4, %esp
 movl $0, %eax
 movl %ebp, %esp
 popl %ebp
 ret
```

hello.s
Source code
Assembly language
Missing definition of printf() function

Compile Process

```
.section .rodata

cGreeting:
 .asciz "hello, world₩n"
 .section .text
 .globl main
 .type main,@function

main:
 pushl %ebp
 movl %esp, %ebp
 pushl $cGreeting
 call printf
 addl $4, %esp
 movl $0, %eax
 movl %ebp, %esp
 popl %ebp
 ret
```

hello.s
Source code
Assembly language
Missing definition of printf() function

Assembler

Assemble
gcc –c hello.s

```
10010100011010010010 0…
```

hello.o
Object code
Machine language
Missing definition of printf() function

Compile Process

10010100011010010010 0···

hello.o
Object code
Machine language
Missing definition of printf() function

Link
gcc hello.o –lc -o hello

Linker

1110010101001100110011···

libc.a
Library containing machine language definition of printf() function (and many others)

1110010101001100110011···

hello
Executable code
Machine language

A Simpler Way

```
#include <stdio.h>

int main(void)
{
        printf("Hello, world!₩n");
         return 0;
}
```

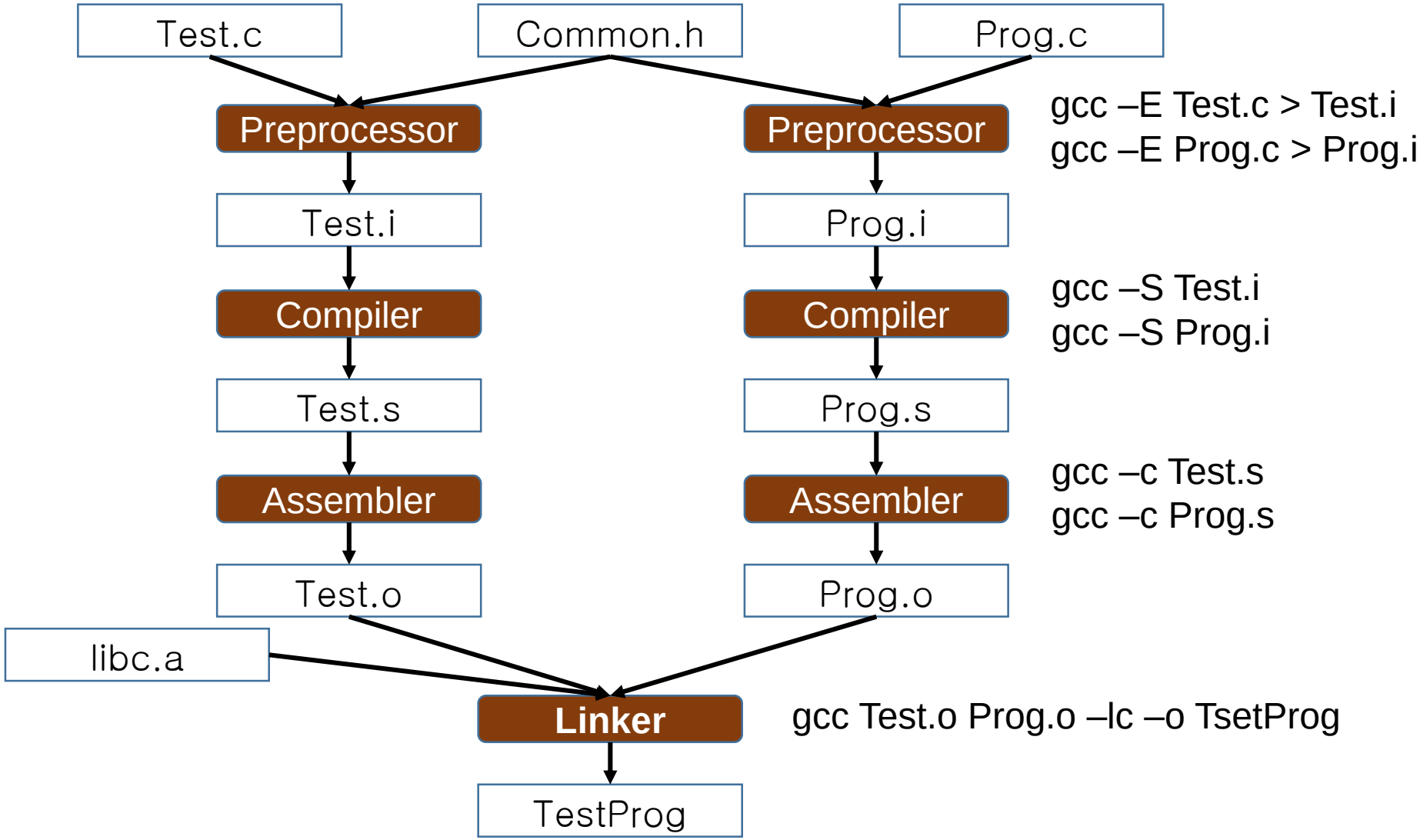
hello.c
Source code
C language
Contains preprocessor directives

gcc hello.c –o hello

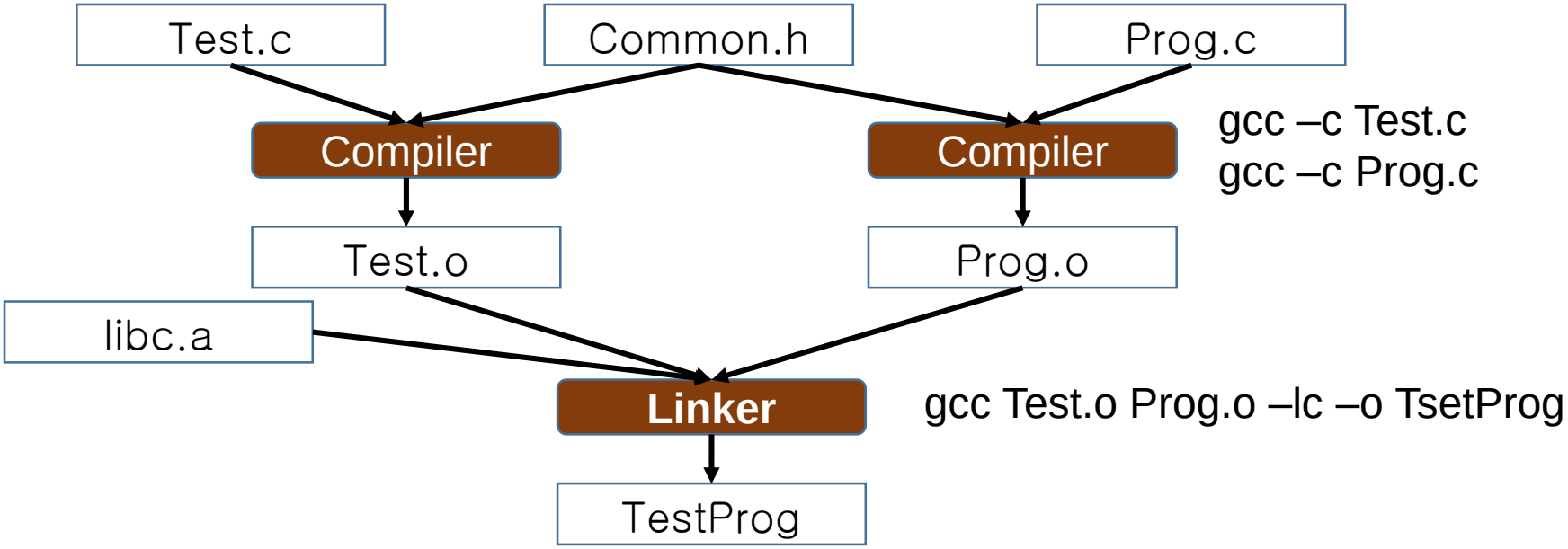
```
1110010101001100110011…
```

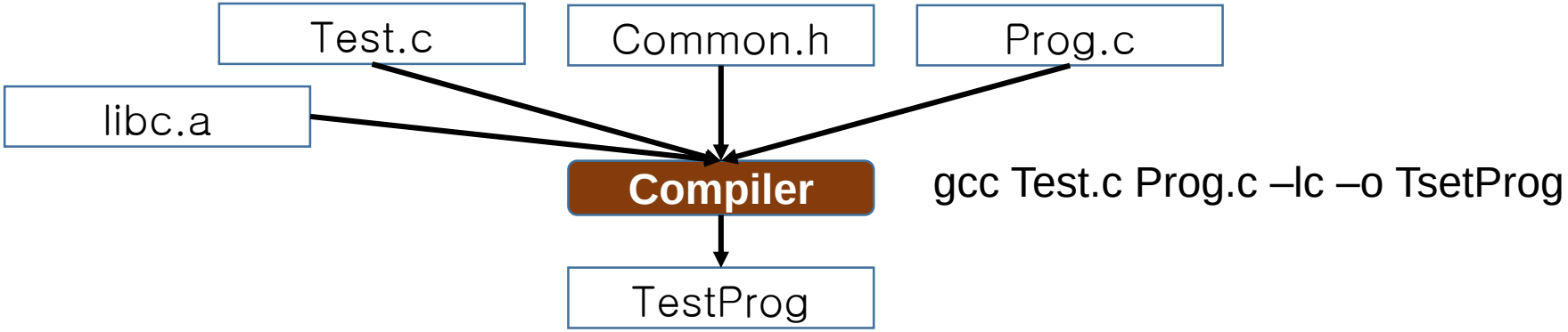
hello
Executable code
Machine language

Compiling Multiple Files



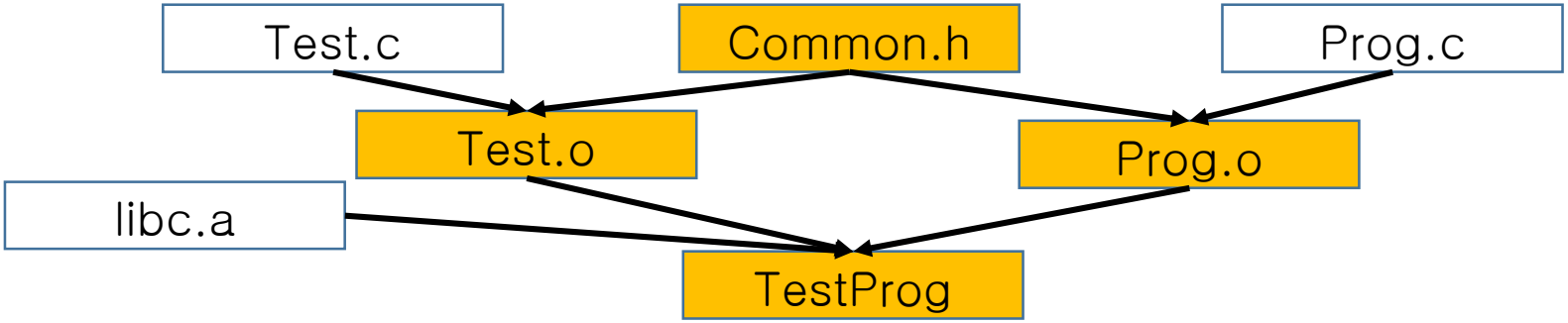
Simpler Ways?





How to manage many files?

- In big projects, thousands of files exist in a project
- Compiling all of them at once is ridiculous
  - gcc thousands.c of.c … files.c –lc –lothers –o bicProg.exe
  - If a file is changed, all the file should be compiled again
- Manually compiling all of them requires a lot of efforts
  - You need to track dependences of changed files



  - Can it be automatically managed?

Makefile

- Makefile
  - Provide a way for separate compilation
  - Describe the dependences among the project files
- Naming
  - *Makefile* or *makefile* are standard
  - Other names can be possible
- Running make
  - make
  - make –f filename
    - If the name is not “makefile” or “Makefile”
  - make target_name
    - If you want to make a target that is not the first one

Sample makefile

- Format

```
target : prerequisites
TAB     commands
```

- Example

```
testProg : test.o prog.o
       gcc test.o prog.o -lc -o testProg

test.o : test.c common.h
       gcc -c test.c

prog.o : prog.c common.h
       gcc -c prog.c
```

Variables

- Original Example

```
testProg : test.o prog.o
        gcc test.o prog.o -lc -o testProg

test.o : test.c common.h
        gcc -c test.c

prog.o : prog.c common.h
        gcc -c prog.c
```

- Example with Variables

```
CC=gcc
OBJS = test.o prog.o
HDRS = common.h

testProg : $(OBJS)
        $(CC) $(OBJS) -lc -o testProg
test.o : test.c
        $(CC) -c test.c
prog.o : prog.c
        $(CC) -c prog.c
$(OBJS) : $(HDRS)
```

Unsatisfied?

```
CC=gcc
OBJS = test.o prog.o
HDRS = common.h

testProg : $(OBJS)
        $(CC) $(OBJS) -lc -o testProg

test.o : test.c
        $(CC) -c test.c

prog.o : prog.c
        $(CC) -c prog.c

$(OBJS) : $(HDRS)
```

Problem: There are numerous rules for making ***.o*** files

Implicit rules

```
CC=gcc
OBJS = test.o prog.o
HDRS = common.h

testProg : $(OBJS)
        $(CC) $(OBJS) -lc -o testProg

%.o : %.c
        $(CC) -c $<

$(OBJS) : $(HDRS)
```

* Special Variables

$@ – The name of the target of the rule

$< – The name of the first prerequisite

$^ – The names of all the prerequisites

$? – The names of all the prerequisites that are newer than the target

Still Unsatisfied? ☹

```
CC=gcc
OBJS = test.o prog.o
HDRS = common.h

testProg : $(OBJS)
        $(CC) $(OBJS) -lc -o testProg

%.o : %.c
        $(CC) -c $<

$(OBJS) : $(HDRS)
```

Shell commands & Variable modifiers

```
CC=gcc
SRCS = $(shell ls *.c)
OBJS = $(SRCS:.c=.o)
HDRS = $(shell ls *.h)

testProg : $(OBJS)
        $(CC) $(OBJS) -lc -o testProg

%.o : %.c
        $(CC) -c $<

$(OBJS) : $(HDRS)
```

Common Targets

- Commonly used targets
  - `all`
    - make all the top level targets
    - `all: my_prog1 my_prog2`
  - `clean`
    - delete all files that are normally created by `make`
  - `print`
    - print listing of the source files that have changed

Conditional Statements

- Conditional commands
  - `if  ifeq  ifneq  ifdef  ifndef`
  - All of them should be closed with `endif`.
  - Complex conditionals may use `elif` and `else`.

- Example

```
libs_for_gcc = -lgnu
normal_libs =
ifeq ($(CC),gcc)
libs=$(libs_for_gcc)     #no tabs at the beginning
else
libs=$(normal_libs)      #no tabs at the beginning
endif
```