CS302
Operating System
Lab 1

Introduction to Linux Shell

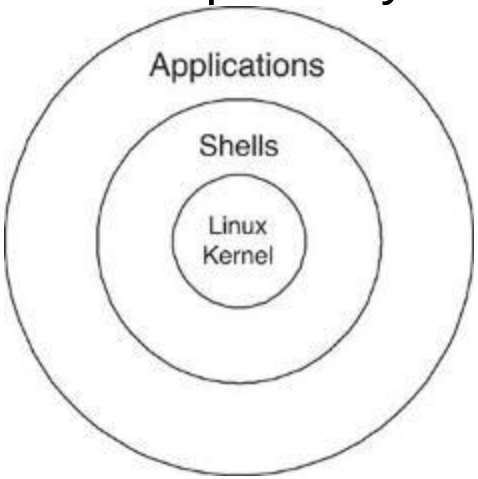
March 7th , 2018

Xiang Long

http://acm.sustc.edu.cn/cs302/

Linux Architecture

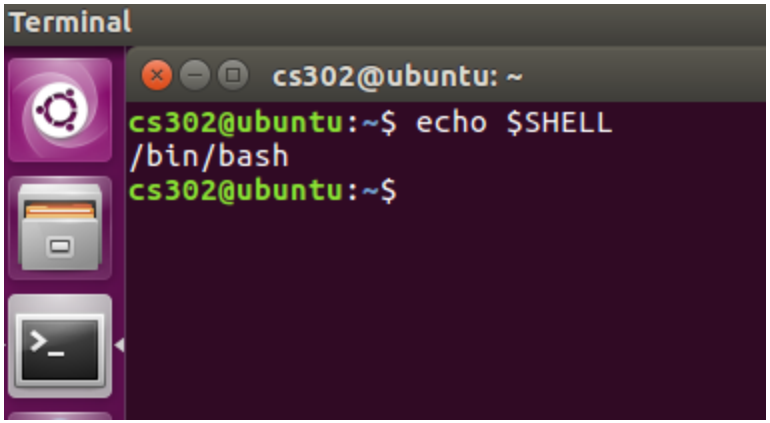
- A Linux Operating System has primarily three components



- **Kernel**: At the core is the Linux kernel, which mediates access to the underlying hardware resources such as memory, the CPU, and peripherals.
- **Shell**: The shell provides user access to the kernel. The shell provides command interpretation and the means to load user applications and execute them.
- **Applications**: These make up the bulk of the GNU/Linux operating system. These applications provide the useful functions for the operating system, such as windowing systems, web browsers, and, of course, programming and development tools.

Type of shell in-use

- There are two major types of shells in Linux:
  - Bourne shell(i.e. sh, ksh, bash)
  - C shell(i.e. csh,tcsh).
- We can find the type of shell in-use in a terminal in the environment variable **SHELL**



- In our Lab, the output is/bin/bash, which is the path to the shell executable.

Basic Bash Commands

- echo
  - **echo -** display a line of text

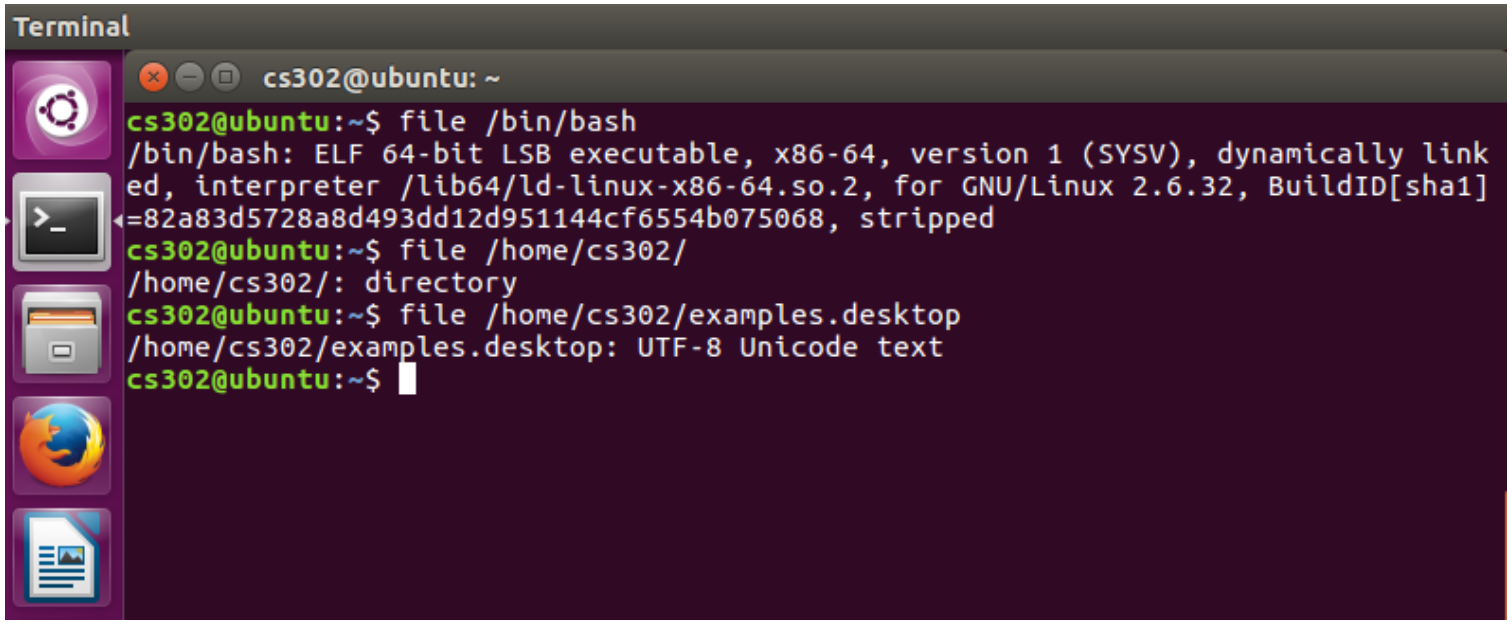
```
cs302@ubuntu: ~
cs302@ubuntu:~$ echo "hello cs302"
hello cs302
cs302@ubuntu:~$
```

  - **echo -** display a environment variable

```
cs302@ubuntu: ~
cs302@ubuntu:~$ echo $HOME
/home/cs302
cs302@ubuntu:~$
```

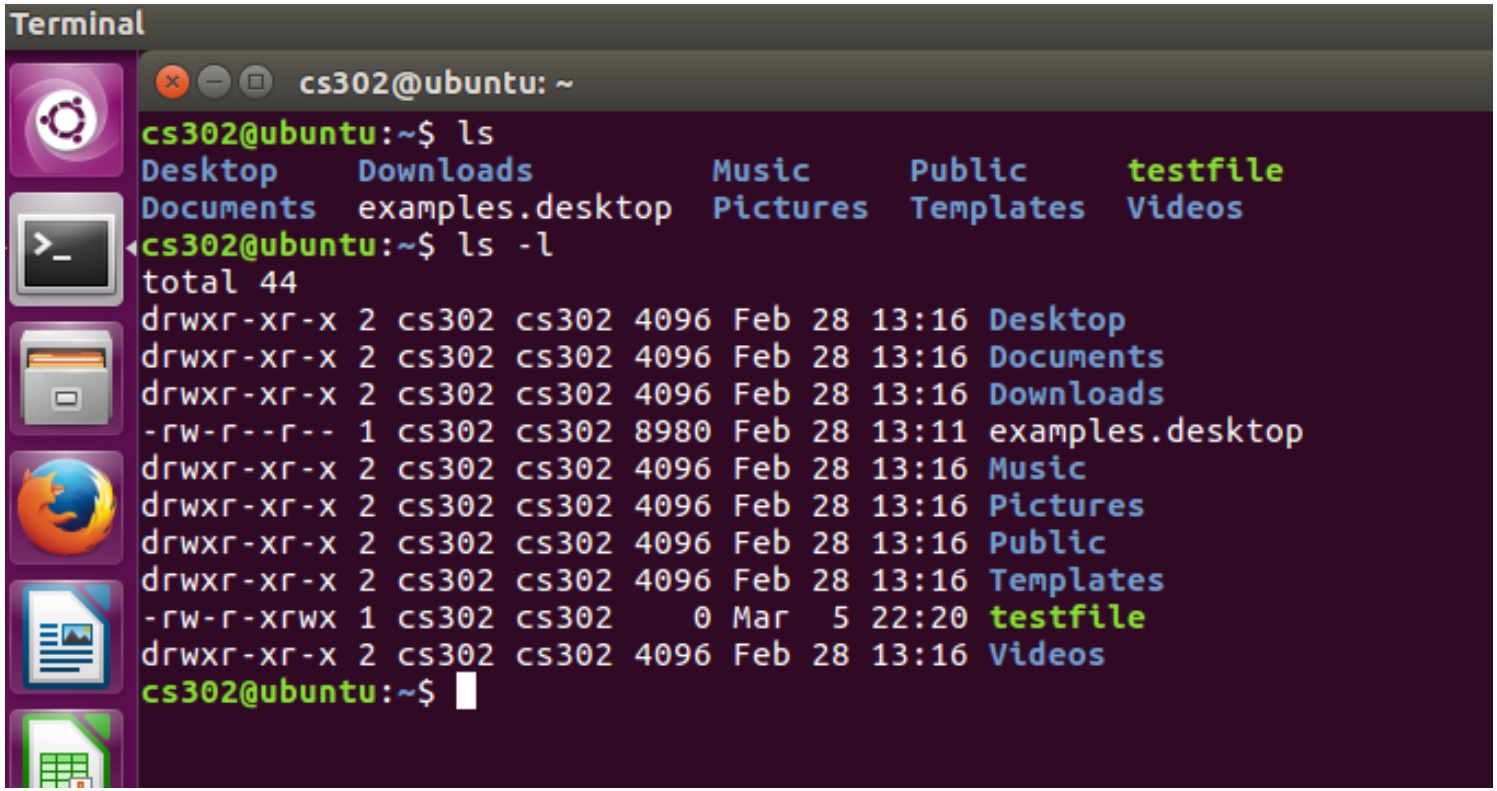
Basic operations on files

- File
  - **file** - determine file type



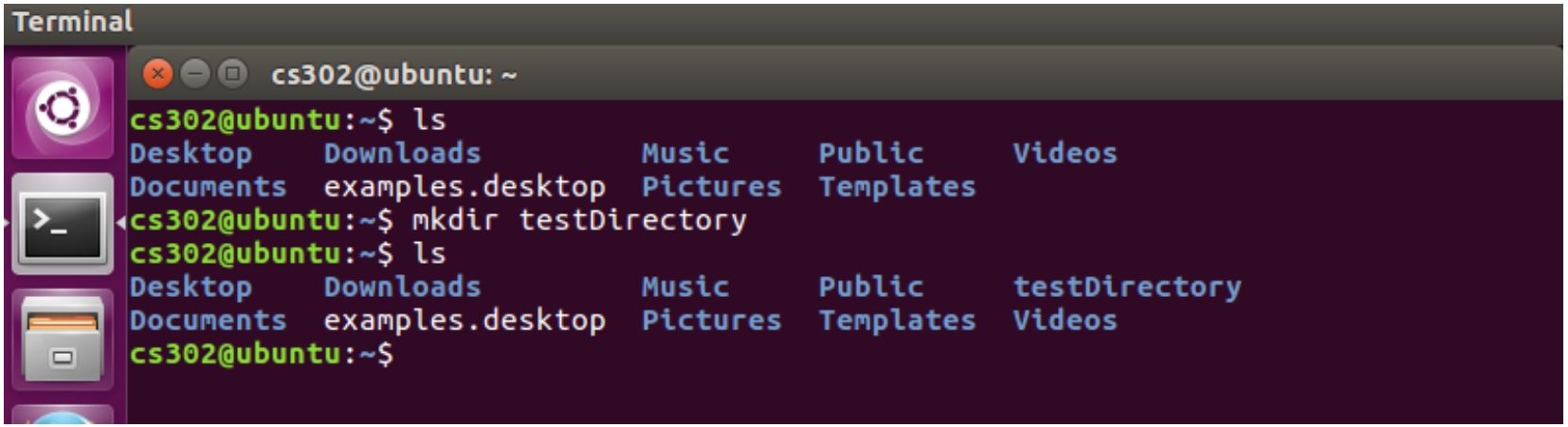
Basic operations on files

- ls
  - **ls** - list directory contents
  - common option： -l -a



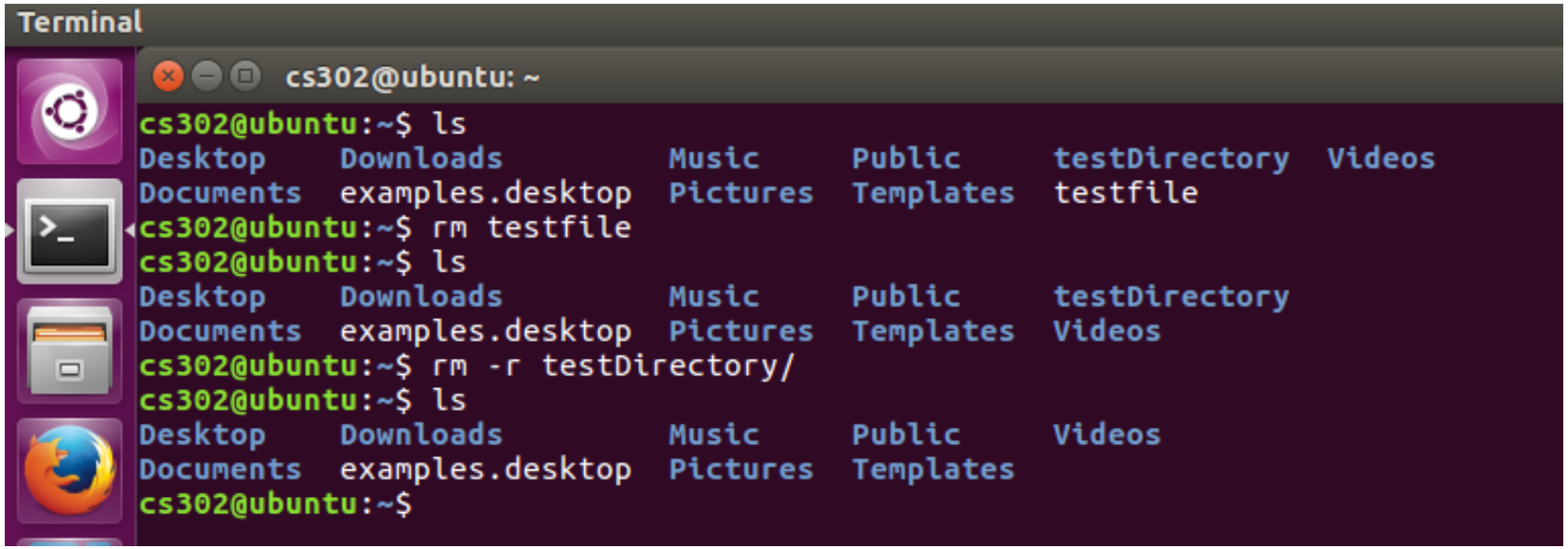
Basic operations on files

- mkdir
  - **mkdir** - make directories



Basic operations on files

- rm
  - **rm -** remove files or directories
  - common option: -r -f



Basic operations on files

- cp
  - cp - copy files and directories
  - common option: -r

```
cs302@ubuntu: ~
cs302@ubuntu:~/testdir$ ls
testfile
cs302@ubuntu:~/testdir$ cp testfile testfile.bak
cs302@ubuntu:~/testdir$ ls
testfile  testfile.bak
cs302@ubuntu:~/testdir$ cd ..
cs302@ubuntu:~$ ls
Desktop    Downloads         Music     Public     testdir
Documents  examples.desktop  Pictures  Templates  Videos
cs302@ubuntu:~$ cp -r testdir/ /home/cs302/Templates/
cs302@ubuntu:~$ ls /home/cs302/Templates/
testdir
cs302@ubuntu:~$
```

Basic operations on files

- mv
  - mv - move (rename) files

:

```
cs302@ubuntu: ~/testdir
cs302@ubuntu:~/testdir$ ls
testfile  testfile.bak
cs302@ubuntu:~/testdir$ mv testfile /home/cs302/
cs302@ubuntu:~/testdir$ ls /home/cs302/
Desktop    Downloads         Music     Public     testdir   Videos
Documents  examples.desktop  Pictures  Templates  testfile
cs302@ubuntu:~/testdir$
```

Showing file content

- cat/more/less/head/tail
  - **cat**, **more**, **less**, **head**, and **tail** are commonly used commands for showing file content in a terminal, i.e., printing files' content to the terminal.
  - **cat** concatenates files and prints all content to the terminal at once.

```
cs302@ubuntu:~$ ls
Desktop    Downloads         hello.txt  Pictures  Templates  testfile
Documents  examples.desktop  Music      Public    testdir    Videos
cs302@ubuntu:~$ cat hello.txt
hello world
hello cs302
hello ubuntu
hello termianl
hello shell
```

  - **more** and **less** are two other commands that do a similar job as **cat** once, **more** and **less** divide and print one screen at a time.

Showing file content

- Head
  - **head** prints a certain numbers of lines, 10 by default, from the beginning of a file.

```
cs302@ubuntu:~$ head -n 2 hello.txt
hello world
hello cs302
cs302@ubuntu:~$
```

  - **tail** is almost the same as **head**, except that it counts lines from the end of a file.

```
cs302@ubuntu:~$ tail -n 2 hello.txt
hello termianl
hello shell
cs302@ubuntu:~$
```

Searching file content

- Grep
  - **grep** searches files and prints the lines in which keywords are found, with keywords highlighted.

```
cs302@ubuntu:~$ cat hello.txt
hello world
hello cs302
hello ubuntu
hello termianl
hello shell
cs302@ubuntu:~$ grep cs302 hello.txt
hello cs302
cs302@ubuntu:~$ grep hello hello.txt
hello world
hello cs302
hello ubuntu
hello termianl
hello shell
cs302@ubuntu:~$
```

Basic operations on processes

- ps
  - **ps** displays information about a selection of the active processes

```
cs302@ubuntu:/home$ ps
   PID TTY          TIME CMD
  2646 pts/4    00:00:00 bash
  4587 pts/4    00:00:00 ps
```

  - To see every process on the system using BSD syntax:

```
cs302@ubuntu:/home$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.4 119560  4628 ?        Ss   Mar06   0:02 /sbin/init auto noprompt
root          2  0.0  0.0      0     0 ?        S    Mar06   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    Mar06   0:00 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<   Mar06   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S    Mar06   0:02 [rcu_sched]
root          8  0.0  0.0      0     0 ?        S    Mar06   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        S    Mar06   0:00 [migration/0]
root         10  0.0  0.0      0     0 ?        S    Mar06   0:00 [watchdog/0]
root         11  0.0  0.0      0     0 ?        S    Mar06   0:00 [kdevtmpfs]
```

Basic operations on processes

- Kill
  - **kill** Send a signal to a process, affecting its behavior or killing it.

```
cs302@ubuntu: ~
cs302@ubuntu:~$ ps
   PID TTY          TIME CMD
  4807 pts/4    00:00:00 bash
  4821 pts/4    00:00:00 ps
cs302@ubuntu:~$ kill -9 4807
```

Useful Operators for Bash Commands

- Pipe Operator (|)
  - By putting **|** between 2 commands, the output of the first command is piped to the second command as its input.

```
cs302@ubuntu: ~
cs302@ubuntu:~$ ls
Desktop    Downloads         hello.txt  Music     Public     testdir   Videos
Documents  examples.desktop  ls.out     Pictures  Templates  testfile
cs302@ubuntu:~$ ls | grep D
Desktop
Documents
Downloads
cs302@ubuntu:~$
```

Basic operations on processes

- Output Redirect Operator (>)
  - Adding a **>** to the end of a command, followed by a file name, redirects the standard output stream (stdout) (What are standard streams?)

```
cs302@ubuntu:~$ ls > ls.out
cs302@ubuntu:~$ cat ls.out
Desktop
Documents
Downloads
examples.desktop
hello.txt
ls.out
Music
Pictures
Public
Templates
testdir
testfile
Videos
```

sudo

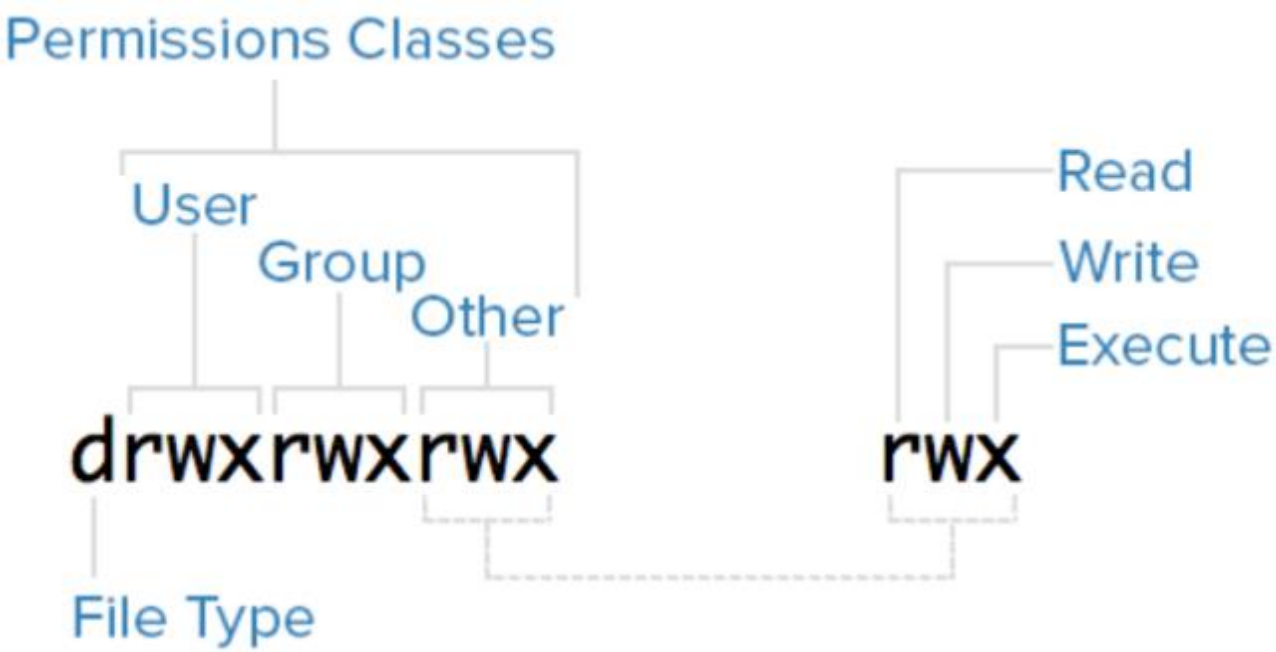
- sudo
  - **sudo** allows a permitted user to execute a command as the superuser or another user

```
cs302@ubuntu: ~
cs302@ubuntu:~$ ls
Desktop    Downloads         ls.out  Pictures  Templates  testfile
Documents  examples.desktop  Music   Public    testdir    Videos
cs302@ubuntu:~$ ls /
bin    dev   initrd.img  lost+found  opt   run   srv  usr
boot   etc   lib         media       proc  sbin  sys  var
cdrom  home  lib64       mnt         root  snap  tmp  vmlinuz
cs302@ubuntu:~$ mv ls.out /
mv: cannot move 'ls.out' to '/ls.out': Permission denied
cs302@ubuntu:~$ sudo mv ls.out /
cs302@ubuntu:~$ ls /
bin    dev   initrd.img  lost+found  mnt   root  snap  tmp  vmlinuz
boot   etc   lib         ls.out      opt   run   srv   usr
cdrom  home  lib64       media       proc  sbin  sys   var
cs302@ubuntu:~$
```

File Permission

- Linux is a multi-user system.
- The most common way to view the permissions of a file is **ls -l**

```
cs302@ubuntu:~$ ls -l
total 48
drwxr-xr-x 2 cs302 cs302 4096 Feb 28 13:16 Desktop
drwxr-xr-x 2 cs302 cs302 4096 Feb 28 13:16 Documents
```



Changing Permissions

- To change the file or the directory permissions, you use the **chmod** (change mode) command.
- There are two ways to use **chmod** — the symbolic mode and the absolute mode.
  - Using chmod in Symbolic Mode

Chmod operator & Description
+ Adds the designated permission(s) to a file or directory.
- Removes the designated permission(s) from a file or directory.
= Sets the designated permission(s).

Changing Permissions

- Then each example chmod command from the preceding table is run on the testfile, followed by ls –l, so you can see the permission changes

```
cs302@ubuntu: ~
cs302@ubuntu:~$ ls -l testfile
-rw-rw-r-- 1 cs302 cs302 0 Mar  7 04:56 testfile
cs302@ubuntu:~$ chmod o+wx testfile
cs302@ubuntu:~$ ls -l testfile
-rw-rw-rwx 1 cs302 cs302 0 Mar  7 04:56 testfile
cs302@ubuntu:~$ chmod o-x testfile
cs302@ubuntu:~$ ls -l testfile
-rw-rw-rw- 1 cs302 cs302 0 Mar  7 04:56 testfile
cs302@ubuntu:~$ chmod g=rx testfile
cs302@ubuntu:~$ ls -l testfile
-rw-r-xrw- 1 cs302 cs302 0 Mar  7 04:56 testfile
cs302@ubuntu:~$
```

Changing Permissions

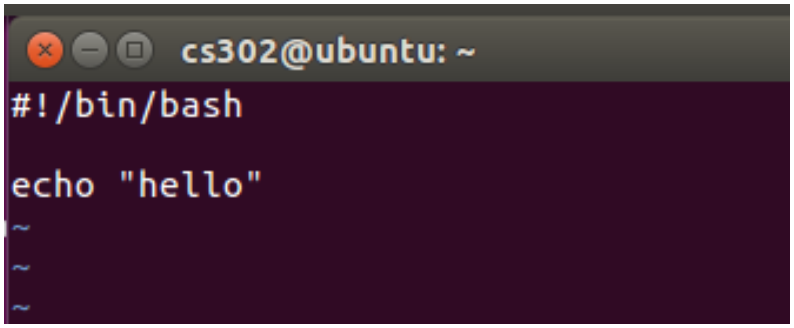
- **chmod** with Absolute Permissions

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

```
franklin@ubuntu:~$ ls -l testfile.txt
-rw-rw-r-- 1 franklin franklin 5 Mar  7 14:41 testfile.txt
franklin@ubuntu:~$ chmod 775 testfile.txt
franklin@ubuntu:~$ ls -ls testfile.txt
4 -rwxrwxr-x 1 franklin franklin 5 Mar  7 14:41 testfile.txt
franklin@ubuntu:~$
```

Elements in Bash Scripts

- Bash Script
  - ➢To create the Bash *script*, you may gather a list of commands, put it into a file



  - ➢Alternatively, we may put a shebang, **#!/bin/bash**, at the beginning of a script. This shebang tells the OS to use **/bin/bash** to parse and run the script.

Create a shell script

To create a shell script:

1. Use a text editor such as vim. Write required Linux commands and logic in the file.
2. Save and close the file (exit from vim).
3. Make the script executable.
4. Script can be run directly.

```
franklin@ubuntu:~$ vim hello.sh
franklin@ubuntu:~$ ls -l hello.sh
-rw-rw-r-- 1 franklin franklin 32 Mar  7 14:55 hello.sh
franklin@ubuntu:~$ chmod u+x hello.sh
franklin@ubuntu:~$ ls -l hello.sh
-rwxrw-r-- 1 franklin franklin 32 Mar  7 14:55 hello.sh
franklin@ubuntu:~$ ./hello.sh
hello cs302
franklin@ubuntu:~$
```

Shell Variables

- You can use variables as in shell. There are no data types. A variable in bash can contain a number, a character, a string of characters..
  - Variable Names

The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

  - Variables are defined as follows

```
variable_name=variable_value
```

  - For example

```
NAME=“cs302"
```

Accessing Values

- To access the value stored in a variable, prefix its name with the dollar sign ($)

```
#!/bin/bash

NAME="cs302"
echo $NAME
```

```
franklin@ubuntu:~$ chmod u+x varable.sh
franklin@ubuntu:~$ ./varable.sh
cs302
franklin@ubuntu:~$
```

Shell Decision Making

Linux Shell supports conditional statements which are used to perform different actions based on different conditions.

- The if...else statements

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

Unix Shell supports following forms of if…else statement

- if...fi statement
- if...else...fi statement
- if...elif...else...fi statement

Shell Decision Making

The **if...elif...fi** statement is the one level advance form of control statement that allows Shell to make correct decision out of several conditions.

- Syntax

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
else
   Statement(s) to be executed if no expression is true
fi
```

Shell Loop Types

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly.

- The while loop
- The for loop
- The until loop

The while loop

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs.

- Syntax

```
while command
do
        Statement(s) to be executed if command is true
done
```

- Example

```
#!/bin/sh
a=0
while [ $a -lt 10 ]
do
        echo $a a=`expr $a + 1`
done
```

The for loop

The **for** loop operates on lists of items. It repeats a set of commands for every item in a list.

- Syntax

```
for var in word1 word2 ... wordN
do
        Statement(s) to be executed for every word.
done
```

- Example

```
#!/bin/sh
for var in 0 1 2 3 4 5 6 7 8 9
do
    echo $var
done
```

The until loop

The while loop is perfect for a situation where you need to execute a set of commands while some condition is true. Sometimes you need to execute a set of commands until a condition is true.

- Syntax

```
until command
do
    Statement(s) to be executed until command is true
done
```

- Example

```
#!/bin/sh
a=0 until [ ! $a -lt 10 ]
do
    echo $a a = `expr $a + 1`
done
```

Task

- Write the Linux bash script to view the number of files and subdirectories contained in the home directory and export it to the file.info file as follows.
- File.info store in `~/lab1/student_id/file.info`

[Directory1]

Directory1/Sub_Dir

Directory1/file2

…

[Sub_Dir]

Directory1/Sub_Dir/file1

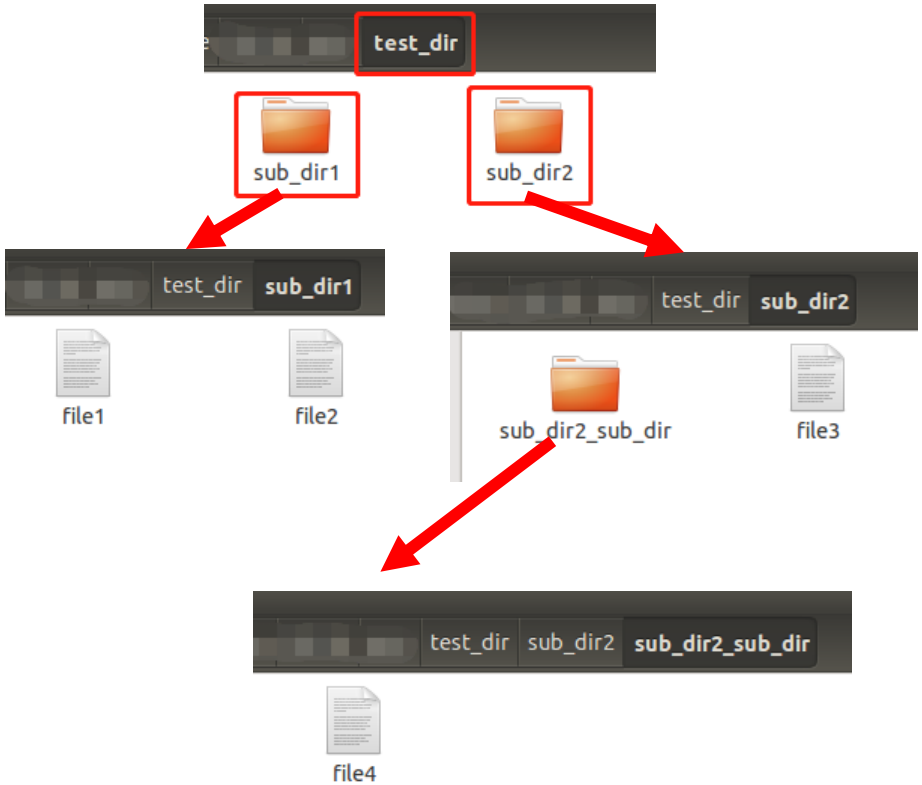
Directory2/Sub_Dir/file2

…

[Directories Count] : xx

[File Count] : xx

Task 1 Example



file.info

[test_dir]
/test_dir/sub_dir1
/test_dir/sub_dir2

[sub_dir1]
/test_dir/sub_dir1/file1
/test_dir/sub_dir1/file2

[sub_dir2]
/test_dir/sub_dir2/file3
/test_dir/sub_dir2/sub_dir2_sub_dir

[sub_dir2_sub_dir]
/test_dir/sub_dir2/sub_dir2_sub_dir/file4

[Directories Count] : 3

[Files Count] : 4

Thanks