CS302
Operating System
Lab 1

Introduction to Linux Shell

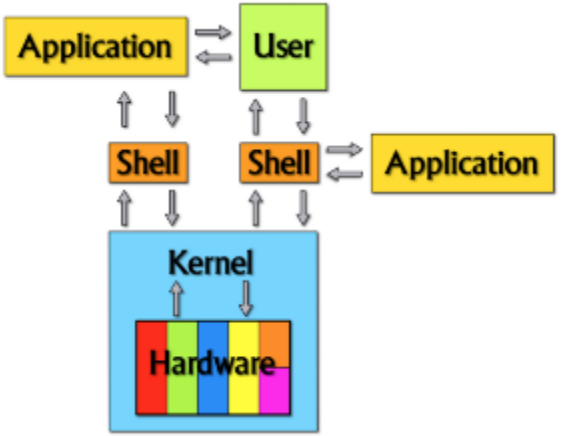
Feb. 27th , 2019

杨闯

鸣谢 向隆，歆勋，诗奇

Linux Architecture

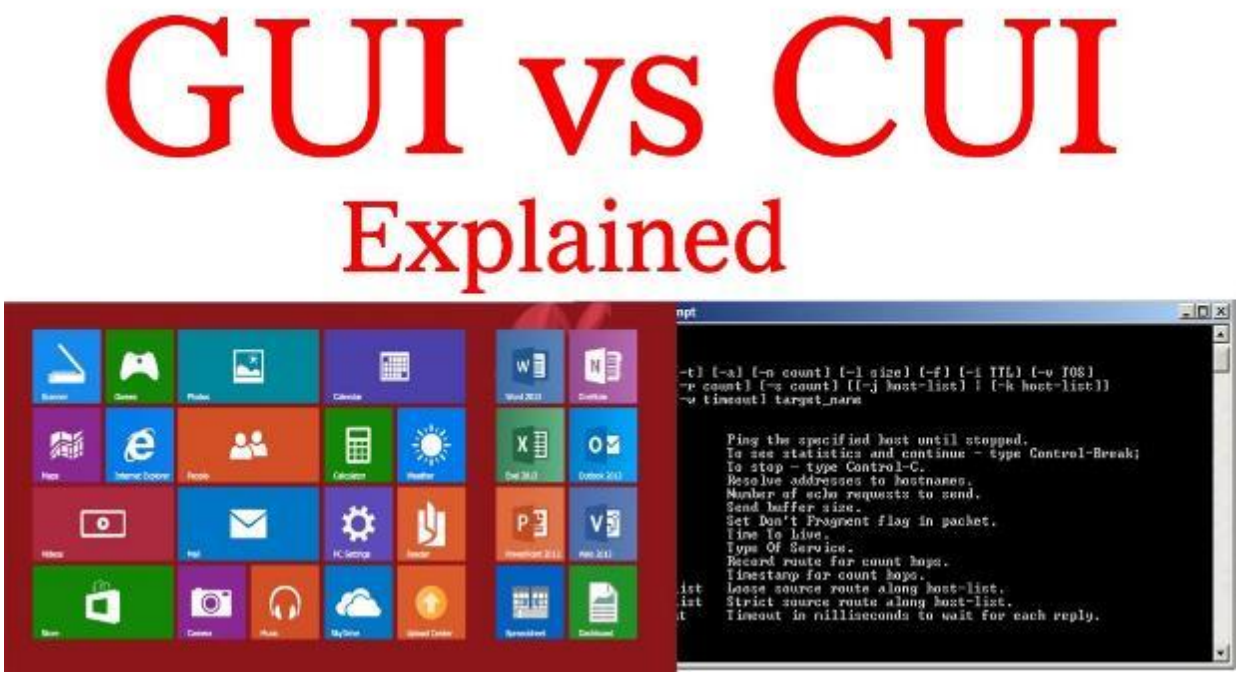
- A Linux Operating System has primarily three components



- **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.

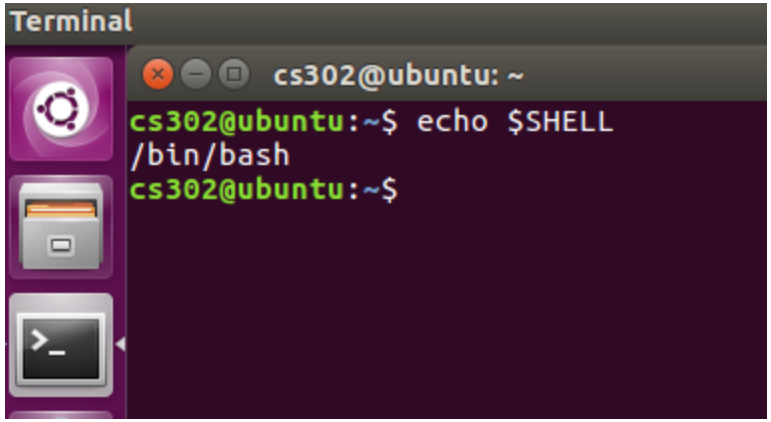
What is shell

- Shell:
  - A CUI (vs GUI) that connects the user and OS.
  - An interpreter to execute shell script, and execute the program in $PATH.



Type of shell in-use

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



Shell in windows?

Shell as CUI (Character User Interface)

Display a prompt,

Read a command,

Process the given command,

then Execute the command.

After which it starts the process all over again.

Basic Bash Commands

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

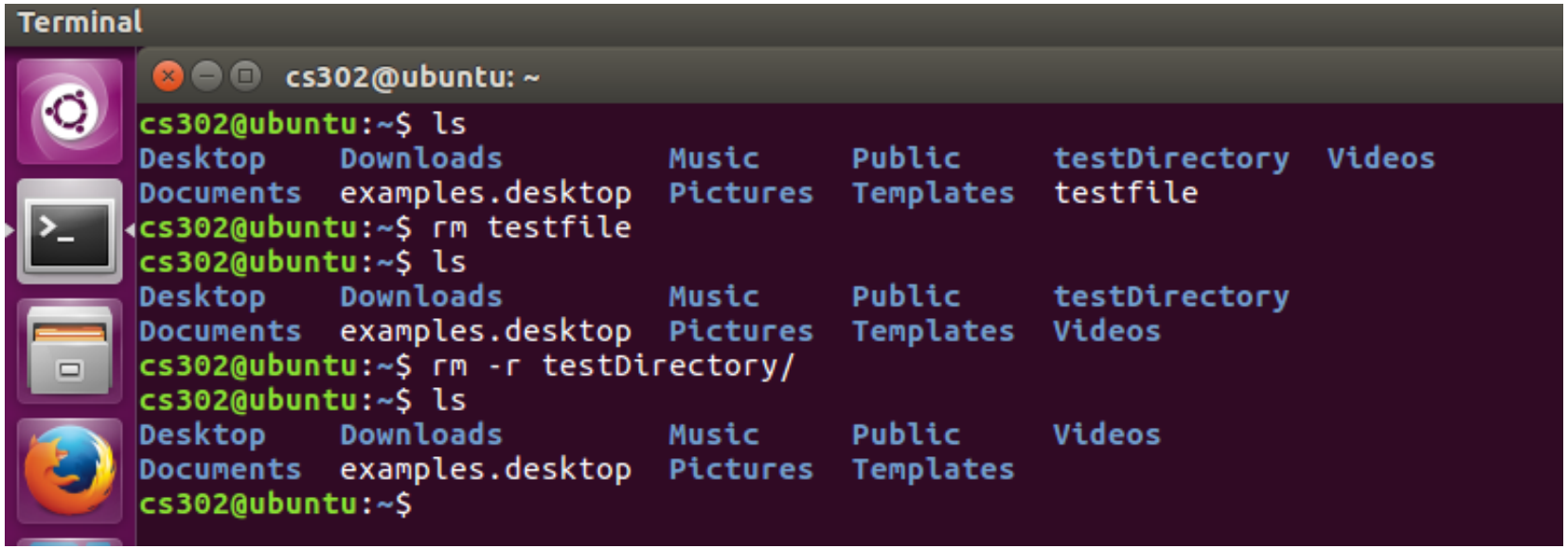
```
cs302@ubuntu: ~
cs302@ubuntu:~$ echo "hello cs302"
hello cs302
cs302@ubuntu:~$
```

  - **echo -** display a environment variable

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

Basic operations on files

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



Basic operations on files

- cp
  - **cp** - copy files and directories
  - common option: -r
  - Notice： widely use on backups

```
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
  - Notice： Widely used on rename

```
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$
```

Basic operations on files

- find
  - **find** find a file in given path
  - common option: -name -type -size -ctime

```
mark@marklinux:~$ find . -name '*.sh'
./up.sh
./cal.sh
./func.sh
./hello.sh
./while.sh
./test.sh
./if.sh
```

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 && Tail
  - **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:~$
```

Very useful parameter tail –f !!
Let's try!

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:

```
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 && killall （try to compare with windows）
  - **kill** Send a signal to a process, affecting its behavior or killing it. There are 64 kinds of signal to send to process. Only -9 can kill it unconditionally.

HUP	1	终端断线
INT	2	中断（同 Ctrl + C）
QUIT	3	退出（同 Ctrl + \）
TERM	15	终止
KILL	9	强制终止
CONT	18	继续（与STOP相反， fg/bg命令）
STOP	19	暂停（同 Ctrl + Z）

  - killall can kill all process by name, the signal number is as same as kill.

```
mark@marklinux:~$ ps
   PID TTY          TIME CMD
 14902 pts/0    00:00:00 bash
 14925 pts/0    00:00:00 vim
 14926 pts/0    00:00:00 vim
 14928 pts/0    00:00:00 vim
 14930 pts/0    00:00:00 ps
mark@marklinux:~$ kill -9 14925
mark@marklinux:~$ ps
   PID TTY          TIME CMD
 14902 pts/0    00:00:00 bash
 14926 pts/0    00:00:00 vim
 14928 pts/0    00:00:00 vim
 14931 pts/0    00:00:00 ps
[2]   Killed                  vim n
mark@marklinux:~$
```

```
mark@marklinux:~$ killall -9 vim
mark@marklinux:~$ ps
   PID TTY          TIME CMD
 14902 pts/0    00:00:00 bash
 14933 pts/0    00:00:00 ps
[3]-  Killed                  vim c
[4]+  Killed                  vim a
mark@marklinux:~$
```

Text processing command

- Text processing
  - **grep** searches and prints the lines in which keywords are found, with keywords highlighted.
  - **awk** process text row by row
  - **sed** replace text

```
mark@marklinux:~$ cat hello.txt
hello world
hello cs302
hello ubuntu
hello terminal
hello shell
```

```
mark@marklinux:~$ grep hello hello.txt
hello world
hello cs302
hello ubuntu
hello terminal
hello shell
```

```
mark@marklinux:~$ awk '{print $2}' hello.txt
world
cs302
ubuntu
terminal
shell
```

```
mark@marklinux:~$ cat hello.txt | sed 's/hello/hi/g'
hi world
hi cs302
hi ubuntu
hi terminal
hi shell
```

ps aux | grep python | grep -v ‘server.py’ | awk ‘{print $2}’ | xargs kill -9

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

- 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?)
  - Using >> to append the text.
  - Using < to redirect the standard input stream.

```
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
```

How oj work

./main < test.in > test.out

diff test.out answer.out

sudo

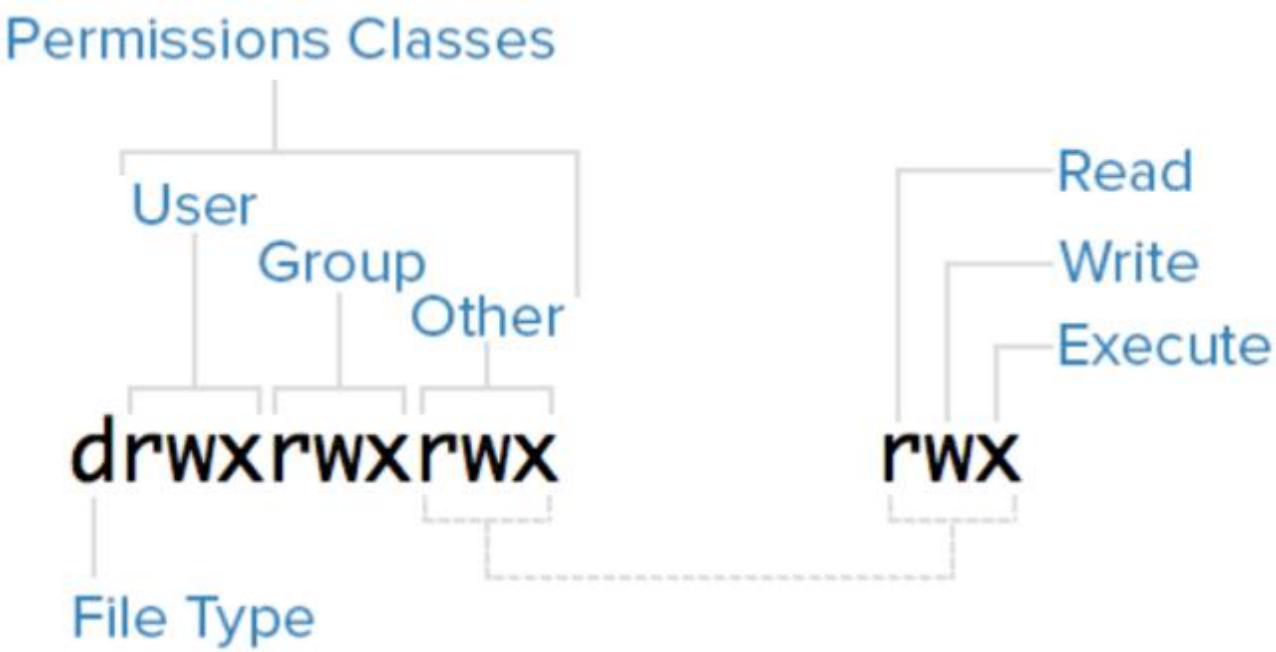
- sudo
  - **sudo** allows a permitted user to execute a command as the superuser.

```
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:~$
```

How many command in Linux?

- All in here: https://github.com/coreutils/coreutils/tree/master/src

coreutils / coreutils
mirrored from git://git.sv.gnu.org/coreutils.git

Watch 63 | Star 865 | Fork 240

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Insights

Branch: master | coreutils / src / | Create new file | Upload files | Find file | History

pixelb wc: treat non breaking space as a word separator ... Latest commit a5202bd 2 days ago

..		
blake2	all: prefer HTTPS in URLs	2 years ago
.gitignore	basenc: A new program complementary to base64/base32	2 months ago
basename.c	maint: update all copyright year number ranges	2 months ago
basenc.c	basenc: allocate buffers on heap	2 months ago
cat.c	maint: update all copyright year number ranges	2 months ago
chcon.c	maint: update all copyright year number ranges	2 months ago
chgrp.c	maint: update all copyright year number ranges	2 months ago
chmod.c	maint: update all copyright year number ranges	2 months ago
chown-core.c	maint: update all copyright year number ranges	2 months ago
chown-core.h	maint: update all copyright year number ranges	2 months ago
chown.c	maint: update all copyright year number ranges	2 months ago
chroot.c	maint: update all copyright year number ranges	2 months ago
cksum.c	all: detect --help and --version more consistently	2 days ago
comm.c	comm,join: ensure warnings are apparent upon exit	14 days ago
copy.c	maint: update all copyright year number ranges	2 months ago
copy.h	maint: update all copyright year number ranges	2 months ago

Shell as Interpreter

Programming Language and Scripting Language

- Programming Language:
  - Use compiler
  - Generate a binary program after parsing the whole code
  - Java, C, C++
- Scripting Language:
  - Use interpreter
  - Execute the code line by line, without program generation.
  - Shell, Python, JavaScript, PHP

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.

How if not executable?

```
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:~$
```

Elements in Bash Scripts

- Bash Script
  - Shebang: **#!/bin/bash**, at the beginning of a script. This shebang tells the OS to use **/bin/bash** to parse and run the script.

```
#!/bin/bash
echo hello
```

```
#!/usr/bin/python3
print("hello")
```

  - A list of commands

```
#! /bin/bash
a=100
b=0
for i in `seq 1 $a`
do
    b=$(expr $i + $b)
done
echo $b
```

`seq 1 10 `
And
$(seq 1 10)

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..
  - Variables definitio **No space**

```
NAME=cs302
```

  - Variables accessing ($, ${})

```
echo $NAME; echo ${NAME}
```

  - For example

```
V1=hello $NAME
V2='hello $NAME'
V3="hello $NAME"
V4="hello $NAMEabc"
V5="hello ${NAME}abc"
```

What's different?

Shell branch

Unix Shell supports following forms of if…else statement

- if...fi statement **Why need fi?**
- if...else...fi statement
- if...elif...else...fi statement

```
a=10
b=20
if [ $a -eq $b ]
then
    echo $a is equal $b
elif [ $a -lt $b ]
then
    echo $a is less than $b
else
    echo $a is larger than $b
fi
```

-eq(==)	equal
-ne(!=)	not equal
-gt	greater than
-ge	greater or equal
-lt	less than
-le	less or equal

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/bash
a=0
while [ $a -lt 10 ]
do
        echo a=$a
        a=$(($a+1))
done
```

space after and before [ ]

$(()) for calculation

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/bash
for var in 0 1 2 3 4 5 6 7 8 9
do
    echo $var
done
```

Function

Shell also support function, but it's very similar to a program. Different from other language's function

```
#!/bin/bash
funWithParam(){
    echo "函数第一个参数为 $1"
    echo "函数第二个参数为 $2"
    echo "函数第十个参数为 $10"
    echo "函数第十个参数为 ${10}"
    echo "函数参数总数有 $# 个"
    echo "作为一个字符串输出所有参数 $* "
    return $((255+$1))   #between 0-255
}

echo "程序名为 $0  程序第一个参数为 $1"
funWithParam 1 two 3 4 5 6 7 8 9 ten
echo $?
```

Powerful shell

- Shell is powerful when mixing Linux command.

```
#!/bin/bash
for i in {1..200}
do
    user=test$i
    userdel $user -r
    useradd -d /home/$user -s /bin/bash -g test $user -m
    echo $user:123456 | chpasswd
    rm /home/$user/*
    cp add_user.sh /home/$user/
done
```

Task

- Write the Linux bash script to view the number of files and subdirectories contained in given directory and export it to a given filename.
- Bash name: lab1-xxxx.sh (xxxx is student id, such as lab1-11210162.sh)
- Command line: ./lab1-xxxx.sh test_dir file.info (test_dir is target directory, can be absolute path or relative path. file.info is output file's path and name)
- Output format:

[Directory1]

Directory1/Sub_Dir ← Output path all in absolute path

Directory1/file2

…

[Sub_Dir]

Directory1/Sub_Dir/file1

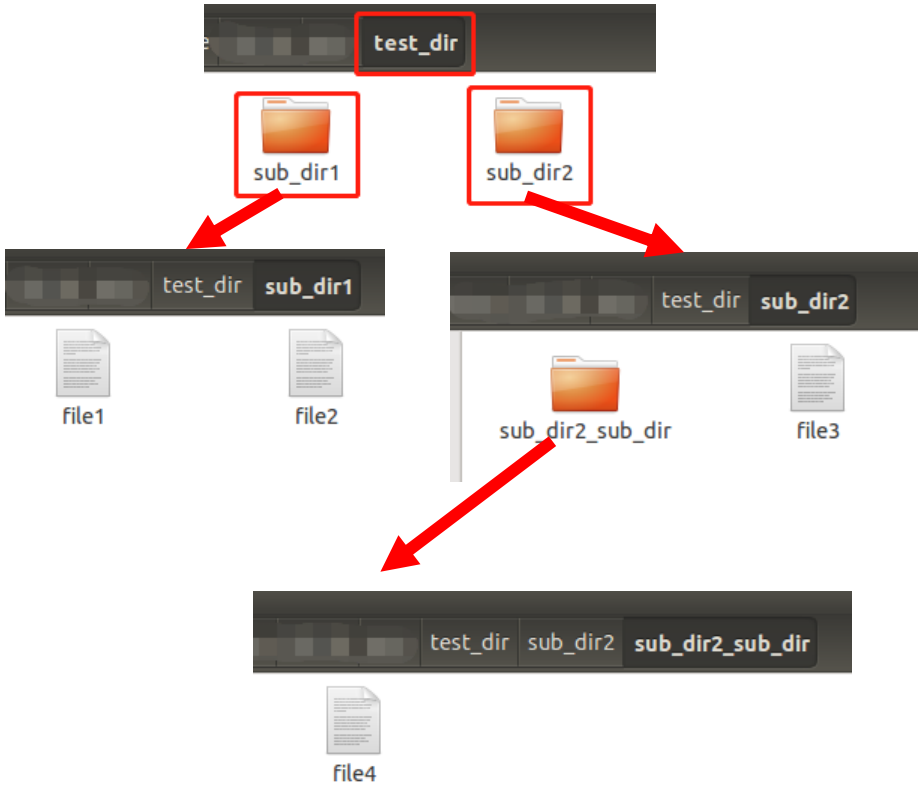
Directory2/Sub_Dir/file2

…

[Directories Count]:xx

[Files Count]:xx

Task 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