Operating Systems Lab (UCS-303) Lab Assignment No.1

Name: Shivam Garg
Roll No: 101703512

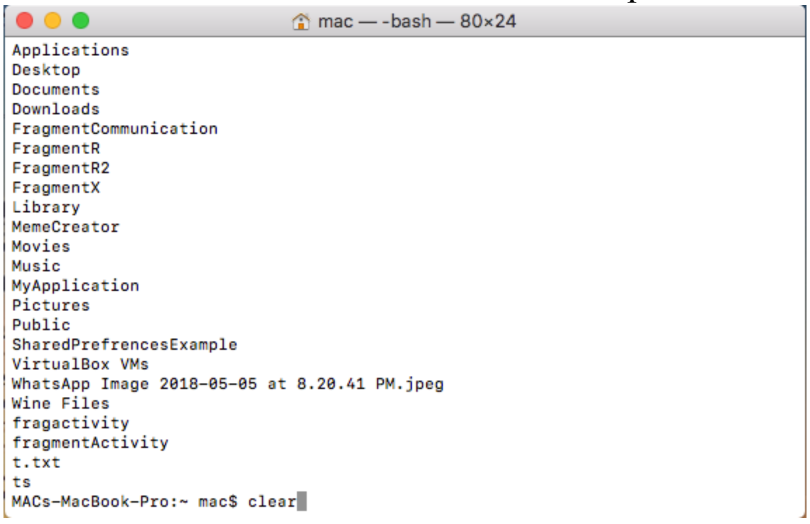
1. Introduction to Linux.

Linux is an operating system or a kernel. It is distributed under an open source license. Its functionality list is quite like UNIX. Linux germinated as an idea in the mind of young and bright Linus Torvalds when he was a computer science student. He used to work on the UNIX OS (proprietary software) and thought that it needed improvements. However, when his suggestions were rejected by the designers of UNIX, he thought of launching an OS which will be receptive to changes, modifications suggested by its users**.**

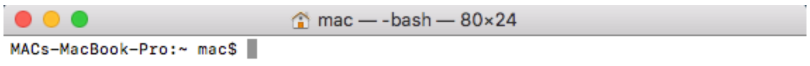
- Being open-source, anyone with programming knowledge can modify it.
- The Linux operating systems now offer millions of programs/applications to choose from.
- Once you have Linux installed you no longer need an antivirus! Linux is a highly secure system.
- Linux is the OS of choice for Server environments due to its stability and reliability (Mega-companies like Amazon, Facebook, and Google use Linux for their Servers). A Linux based server could run non-stop without a reboot for years on end.

2. Clear Command

The clear command is used to remove all previous commands and output from the console.



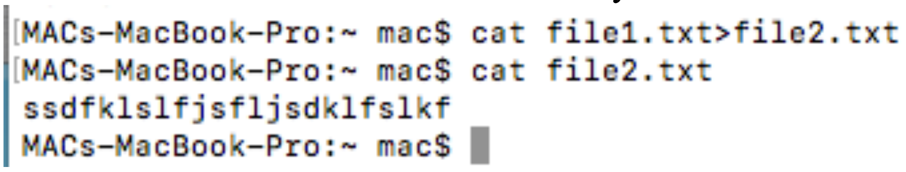
```
mac — -bash — 80×24
Applications
Desktop
Documents
Downloads
FragmentCommunication
FragmentR
FragmentR2
FragmentX
Library
MemeCreator
Movies
Music
MyApplication
Pictures
Public
SharedPrefrencesExample
VirtualBox VMs
WhatsApp Image 2018-05-05 at 8.20.41 PM.jpeg
Wine Files
fragactivity
fragmentActivity
t.txt
ts
MACs-MacBook-Pro:~ mac$ clear
```



```
mac — -bash — 80×24
MACs-MacBook-Pro:~ mac$
```

3. > option for directing the output of a command.

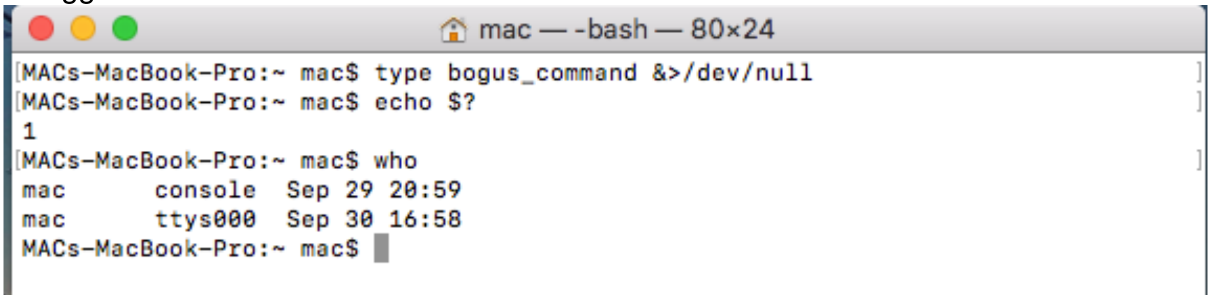
The “>” symbol redirects the output to of a command to the filename written along with the command. Overwrites the file if it already exists.



```
MACs-MacBook-Pro:~ mac$ cat file1.txt>file2.txt
MACs-MacBook-Pro:~ mac$ cat file2.txt
ssdfklslfjsfljsdklfslkf
MACs-MacBook-Pro:~ mac$
```

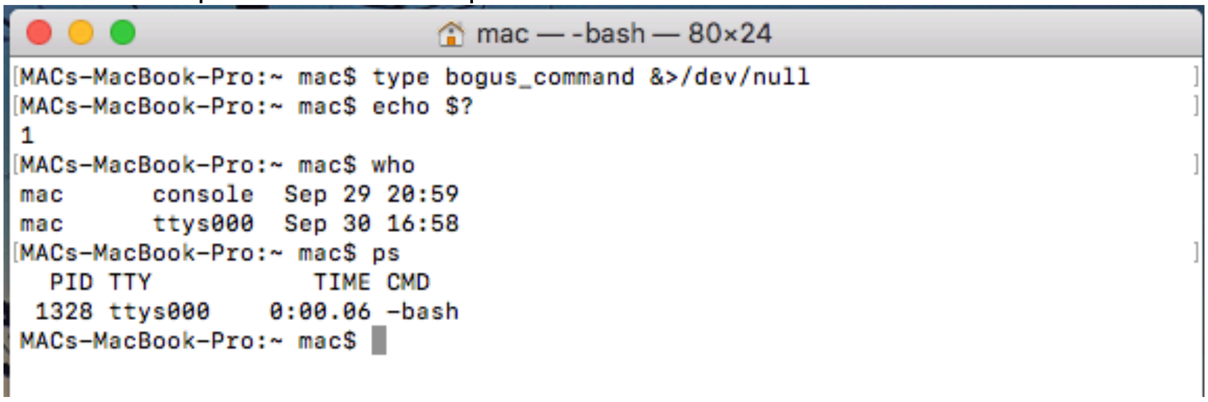
4. who, ps , adduser, man ,help, pwd commands.

who command is used to find out the time of last system boot, current run level of the system and List of logged in users and more.



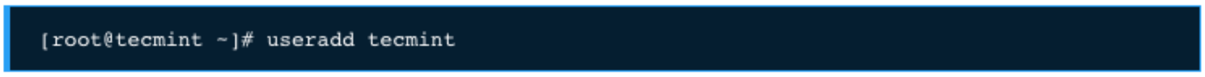
```
MACs-MacBook-Pro:~ mac$ type bogus_command &>/dev/null
MACs-MacBook-Pro:~ mac$ echo $?
1
MACs-MacBook-Pro:~ mac$ who
mac       console  Sep 29 20:59
mac       ttys000  Sep 30 16:58
MACs-MacBook-Pro:~ mac$
```

ps command is used to list the currently running processes and their PIDs along with some other information depends on different options.



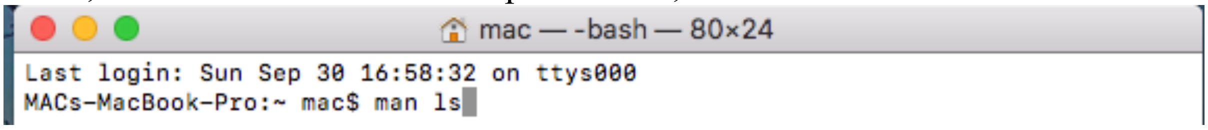
```
MACs-MacBook-Pro:~ mac$ type bogus_command &>/dev/null
MACs-MacBook-Pro:~ mac$ echo $?
1
MACs-MacBook-Pro:~ mac$ who
mac       console  Sep 29 20:59
mac       ttys000  Sep 30 16:58
MACs-MacBook-Pro:~ mac$ ps
  PID TTY           TIME CMD
 1328 ttys000    0:00.06 -bash
MACs-MacBook-Pro:~ mac$
```

adduser is used to To add/create a new user. Adduser or useradd both are the same commands.

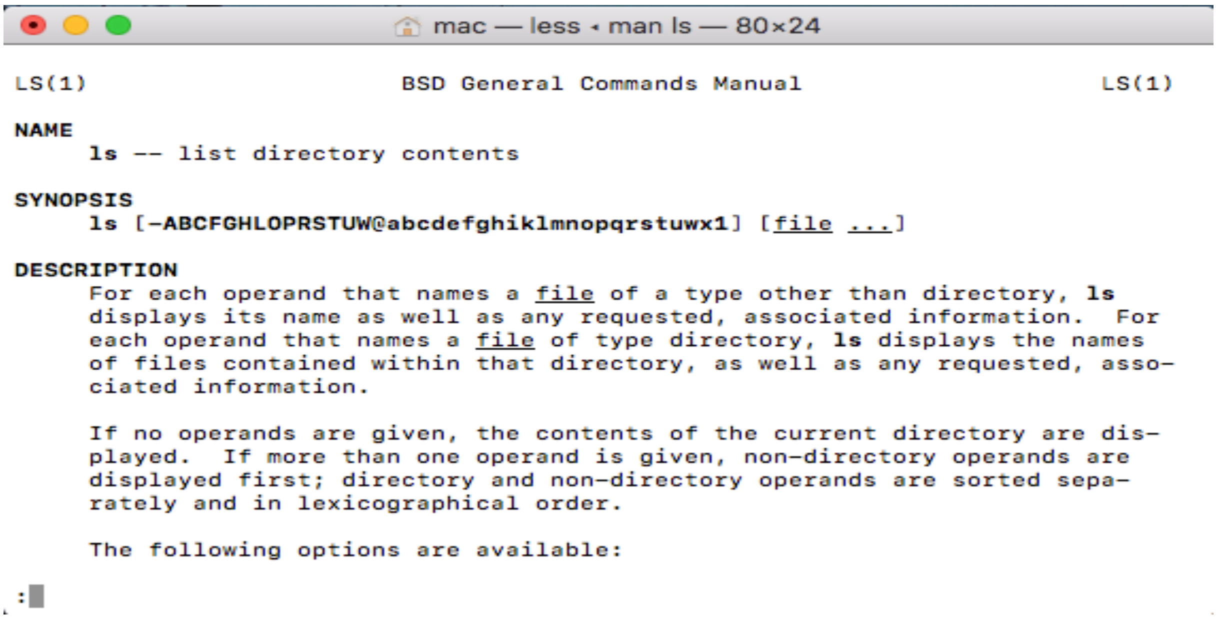


```
[root@tecmint ~]# useradd tecmint
```

man is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions.

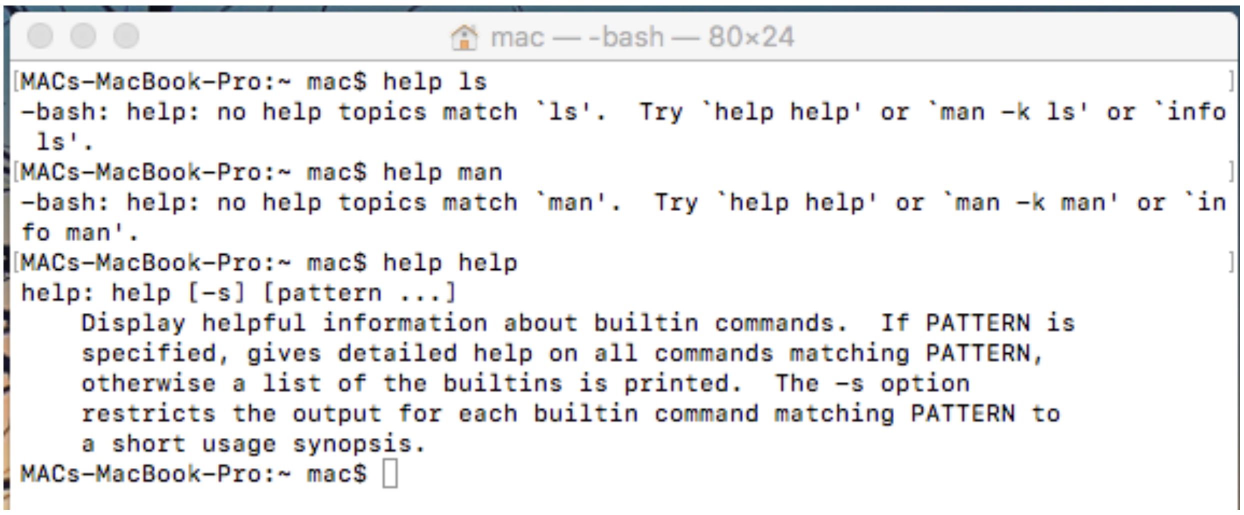


```
Last login: Sun Sep 30 16:58:32 on ttys000
MACs-MacBook-Pro:~ mac$ man ls
```



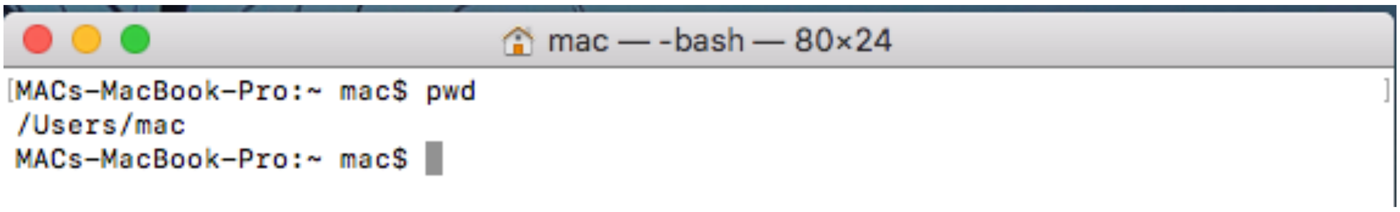
```
LS(1)                     BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls
     displays its name as well as any requested, associated information.  For
     each operand that names a file of type directory, ls displays the names
     of files contained within that directory, as well as any requested, asso-
     ciated information.

     If no operands are given, the contents of the current directory are dis-
     played.  If more than one operand is given, non-directory operands are
     displayed first; directory and non-directory operands are sorted sepa-
     rately and in lexicographical order.

     The following options are available:

:
```

help command displays information about shell built-in commands.



```
MACs-MacBook-Pro:~ mac$ help ls
-bash: help: no help topics match `ls'.  Try `help help' or `man -k ls' or `info
 ls'.
MACs-MacBook-Pro:~ mac$ help man
-bash: help: no help topics match `man'.  Try `help help' or `man -k man' or `in
fo man'.
MACs-MacBook-Pro:~ mac$ help help
help: help [-s] [pattern ...]
    Display helpful information about builtin commands.  If PATTERN is
    specified, gives detailed help on all commands matching PATTERN,
    otherwise a list of the builtins is printed.  The -s option
    restricts the output for each builtin command matching PATTERN to
    a short usage synopsis.
MACs-MacBook-Pro:~ mac$
```

Pwd command is used to print the current working directory



```
MACs-MacBook-Pro:~ mac$ pwd
/Users/mac
MACs-MacBook-Pro:~ mac$
```

5. Introduction to kernel, shell, types of shell.

The **kernel** is the innermost portion of the operating system. It is the core internals: the software that provides basic services for all other parts of the system, manages hardware, and distributes system resource.

Typical components of a kernel are:

- Interrupt handlers to service interrupt requests.
- A scheduler to share processor time among multiple processes.
- A memory management system to manage process address spaces.
- System services such as networking and interprocess communication.

A **shell** in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

There are two main shells in Linux:

1. The **Bourne Shell**: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also is known as sh
- Korn Shell also knew as sh
- **B**ourne **A**gain **SH**ell also knew as bash.

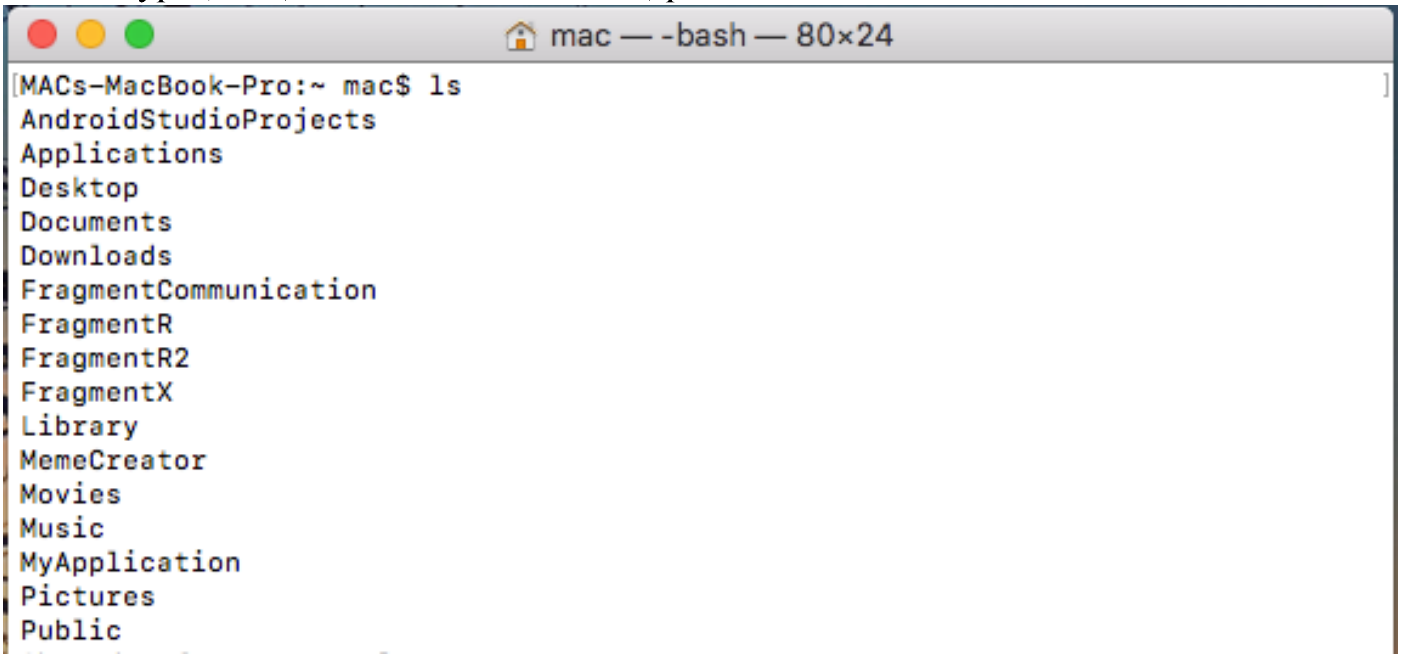
2. The C shell: The prompt for this shell is %, and its subcategories are:

- C shell also is known as csh
- Tops C shell also is known as tcsh

6. ls, ls D*, ls –l and explanation of the details of file in –l command

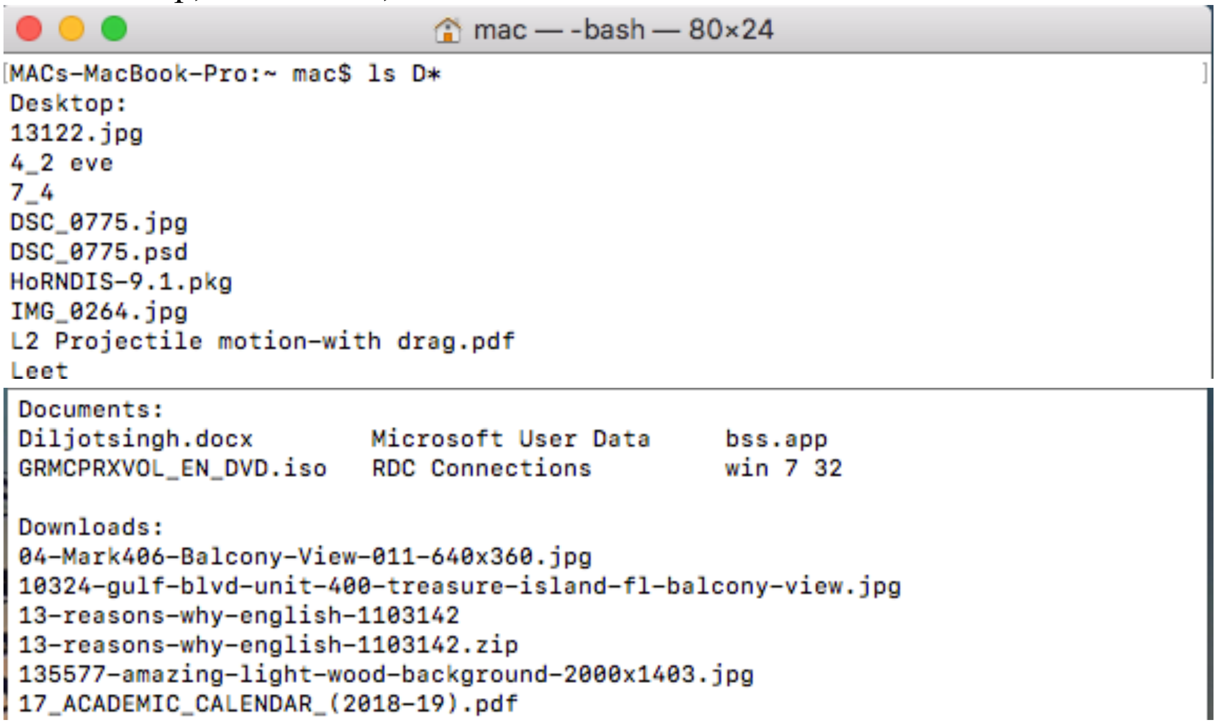
ls is a command that lists directory contents of files and directories.

with no option list files and directories in bare format where we won't be able to view details like file types, size, modified date and time, permission and links etc.



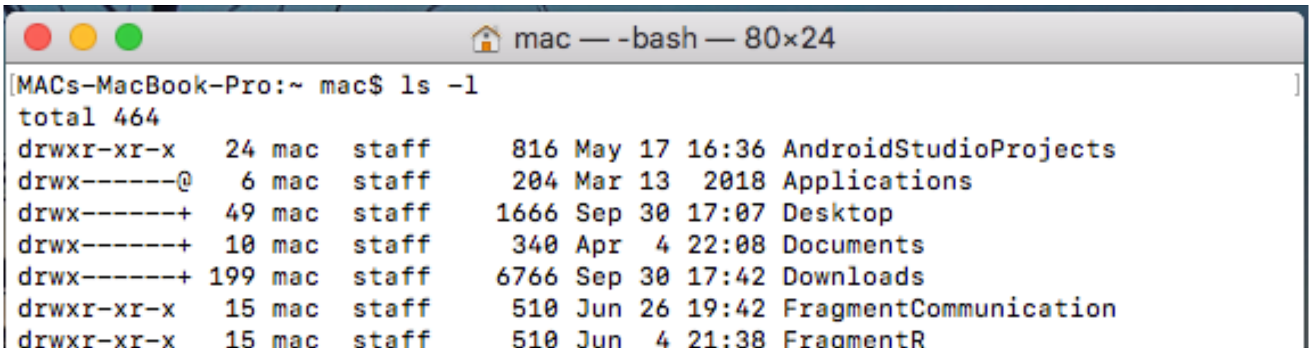
```
MACs-MacBook-Pro:~ mac$ ls
AndroidStudioProjects
Applications
Desktop
Documents
Downloads
FragmentCommunication
FragmentR
FragmentR2
FragmentX
Library
MemeCreator
Movies
Music
MyApplication
Pictures
Public
```

ls D* lists the lists directory contents of all files and directories whose name starts with "D" like Desktop, Documents, Downloads etc.



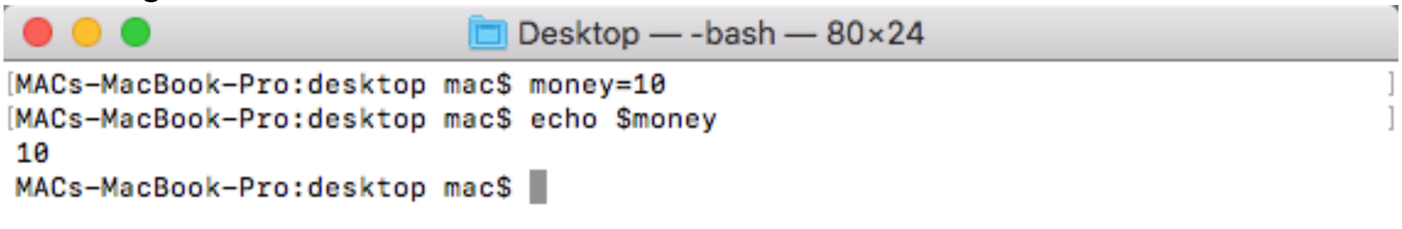
```
MACs-MacBook-Pro:~ mac$ ls D*
Desktop:
13122.jpg
4_2 eve
7_4
DSC_0775.jpg
DSC_0775.psd
HoRNDIS-9.1.pkg
IMG_0264.jpg
L2 Projectile motion-with drag.pdf
Leet
Documents:
Diljotsingh.docx        Microsoft User Data     bss.app
GRMCPRXVOL_EN_DVD.iso   RDC Connections         win 7 32

Downloads:
04-Mark406-Balcony-View-011-640x360.jpg
10324-gulf-blvd-unit-400-treasure-island-fl-balcony-view.jpg
13-reasons-why-english-1103142
13-reasons-why-english-1103142.zip
135577-amazing-light-wood-background-2000x1403.jpg
17_ACADEMIC_CALENDAR_(2018-19).pdf
```

ls-l shows file or directory, size, modified date and time, file or folder name and owner of file and its permission.



```
MACs-MacBook-Pro:~ mac$ ls -l
total 464
drwxr-xr-x   24 mac  staff    816 May 17 16:36 AndroidStudioProjects
drwx------@   6 mac  staff    204 Mar 13  2018 Applications
drwx------+  49 mac  staff   1666 Sep 30 17:07 Desktop
drwx------+  10 mac  staff    340 Apr  4 22:08 Documents
drwx------+ 199 mac  staff   6766 Sep 30 17:42 Downloads
drwxr-xr-x   15 mac  staff    510 Jun 26 19:42 FragmentCommunication
drwxr-xr-x   15 mac  staff    510 Jun  4 21:38 FragmentR
```

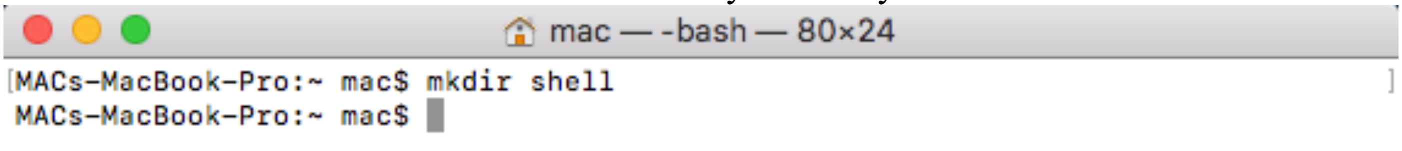
7. Printing variable values.



```
MACs-MacBook-Pro:desktop mac$ money=10
MACs-MacBook-Pro:desktop mac$ echo $money
10
MACs-MacBook-Pro:desktop mac$
```

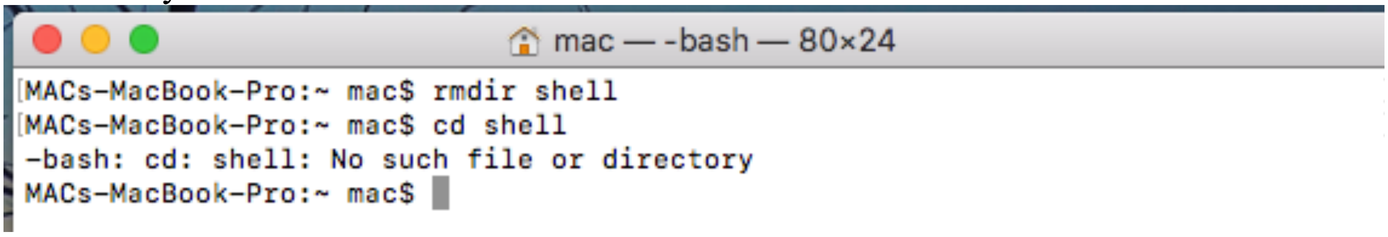
8. mkdir, rmdir, rm, rm –r, toach commands

The **mkdir** command creates new directories in your file system.



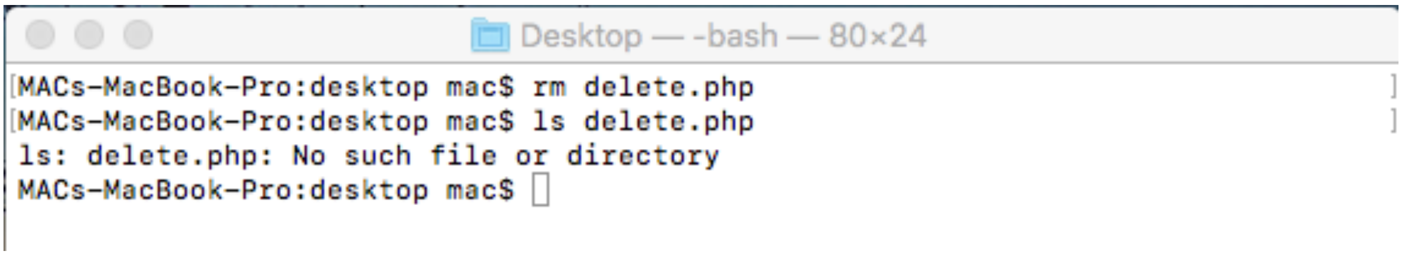
```
MACs-MacBook-Pro:~ mac$ mkdir shell
MACs-MacBook-Pro:~ mac$
```

The **rmdir** command removes each directory specified on the command line, if they are empty. That is, each directory removed must contain no files or directories, or it cannot be removed by rmdir.



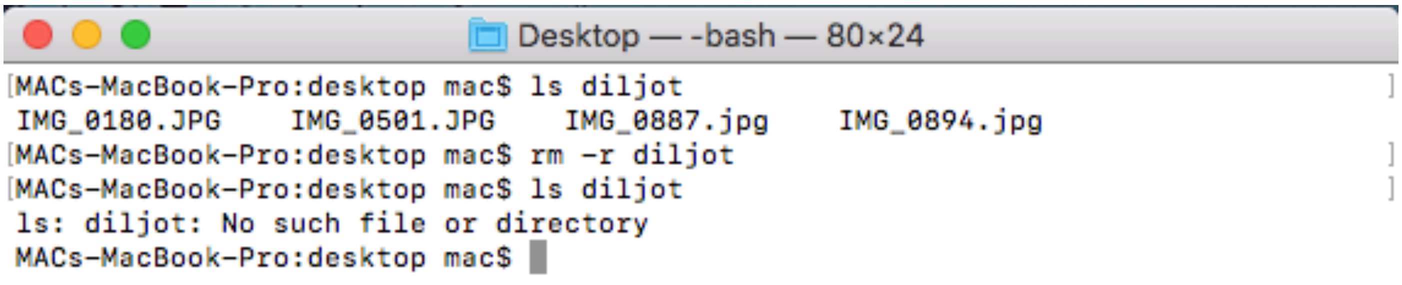
```
MACs-MacBook-Pro:~ mac$ rmdir shell
MACs-MacBook-Pro:~ mac$ cd shell
-bash: cd: shell: No such file or directory
MACs-MacBook-Pro:~ mac$
```

The **rm** ("remove") command is used to delete files. When used recursively, it may be used to delete directories.



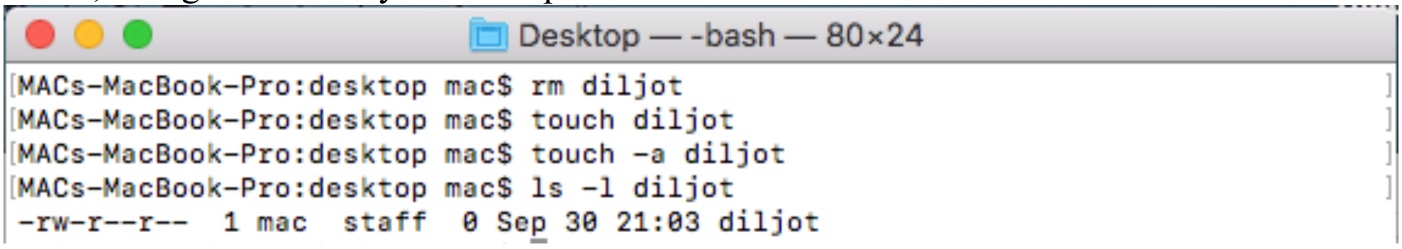
```
MACs-MacBook-Pro:desktop mac$ rm delete.php
MACs-MacBook-Pro:desktop mac$ ls delete.php
ls: delete.php: No such file or directory
MACs-MacBook-Pro:desktop mac$
```

rm-r: With **-r(or -R)** option rm command performs a tree-walk and will delete all the files and sub-directories recursively of the parent directory. At each stage it deletes everything it finds. Normally, **rm** wouldn't delete the directories but when used with this option, it will delete.



```
MACs-MacBook-Pro:desktop mac$ ls diljot
IMG_0180.JPG    IMG_0501.JPG    IMG_0887.jpg    IMG_0894.jpg
MACs-MacBook-Pro:desktop mac$ rm -r diljot
MACs-MacBook-Pro:desktop mac$ ls diljot
ls: diljot: No such file or directory
MACs-MacBook-Pro:desktop mac$
```

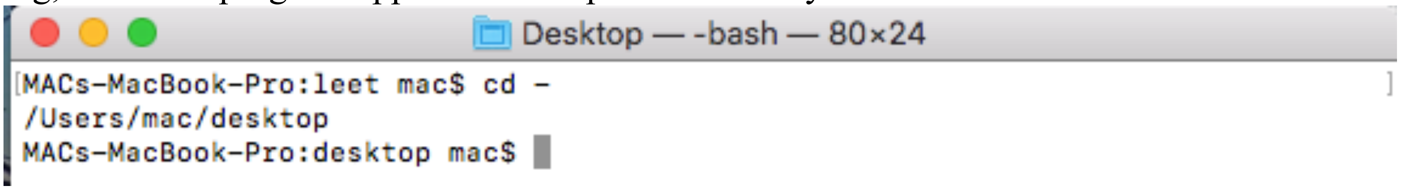
The **touch** command is a standard program for Unix/Linux operating systems, that is used to create, change and modify timestamps of a file.



```
MACs-MacBook-Pro:desktop mac$ rm diljot
MACs-MacBook-Pro:desktop mac$ touch diljot
MACs-MacBook-Pro:desktop mac$ touch -a diljot
MACs-MacBook-Pro:desktop mac$ ls -l diljot
-rw-r--r--  1 mac  staff  0 Sep 30 21:03 diljot
```

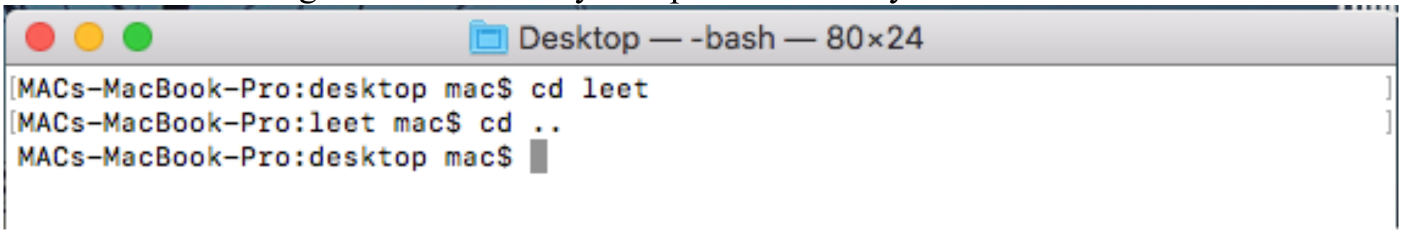
9. cd, cd .. commands

'cd' stands for 'change directory'. 'cd' is the only way to navigate to a directory to check log, execute a program/application/script and for every other task.



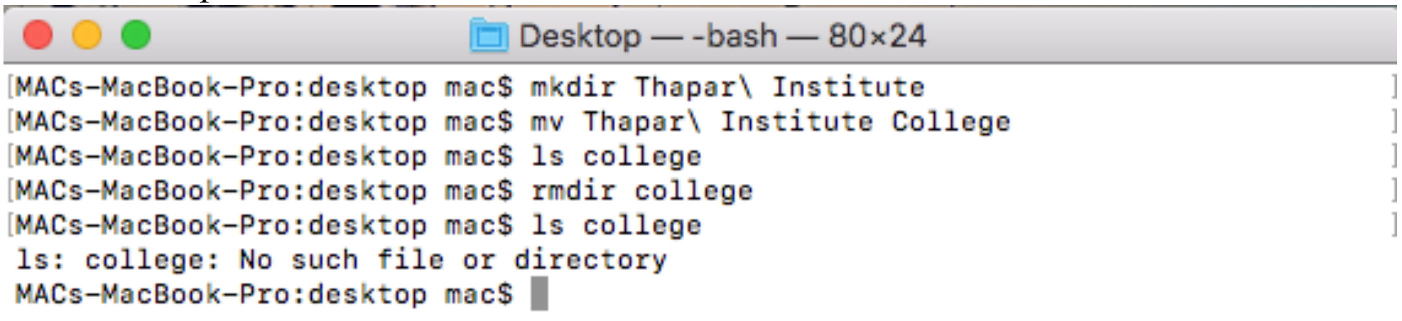
```
MACs-MacBook-Pro:leet mac$ cd -
/Users/mac/desktop
MACs-MacBook-Pro:desktop mac$
```

cd .. is used to change current directory into parent directory.



```
MACs-MacBook-Pro:desktop mac$ cd leet
MACs-MacBook-Pro:leet mac$ cd ..
MACs-MacBook-Pro:desktop mac$
```

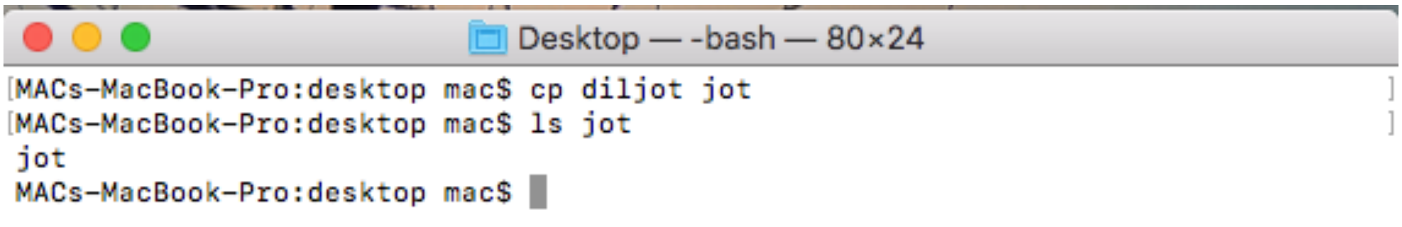
10. To make, remove and change directory for names including spaces eg. "Thapar Institute" as mkdir Thapar\ Institute.



```
MACs-MacBook-Pro:desktop mac$ mkdir Thapar\ Institute
MACs-MacBook-Pro:desktop mac$ mv Thapar\ Institute College
MACs-MacBook-Pro:desktop mac$ ls college
MACs-MacBook-Pro:desktop mac$ rmdir college
MACs-MacBook-Pro:desktop mac$ ls college
ls: college: No such file or directory
MACs-MacBook-Pro:desktop mac$
```

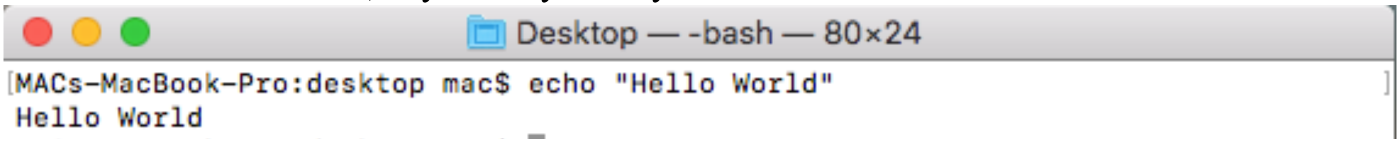
11. cp, mv, echo, cat, cal, date commands.

cp stands for **copy**. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name.



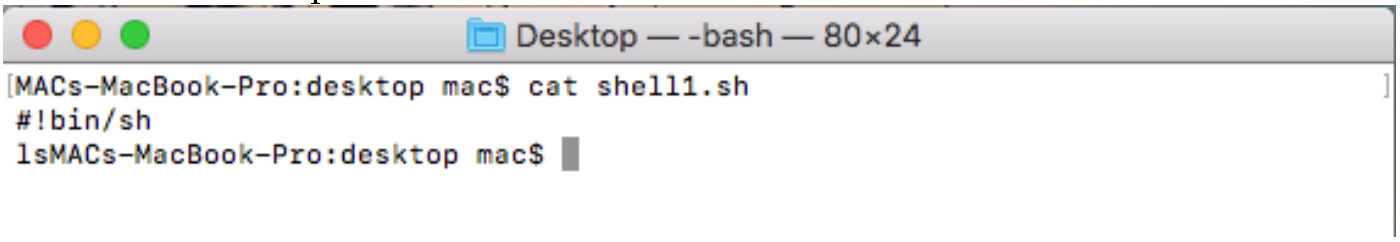
```
[MACs-MacBook-Pro:desktop mac$ cp diljot jot
[MACs-MacBook-Pro:desktop mac$ ls jot
jot
MACs-MacBook-Pro:desktop mac$ 
```

Echo command is used to display text. It is frequently used in scripts, batch files, and as part of individual commands; anywhere you may need to insert text.



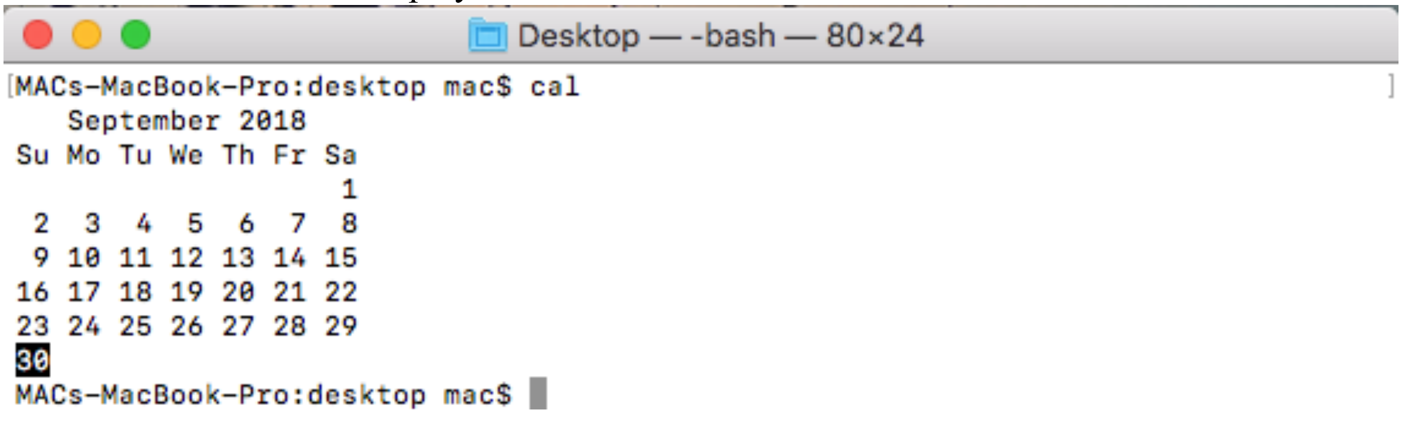
```
[MACs-MacBook-Pro:desktop mac$ echo "Hello World"
Hello World
```

cat command allows us to create single or multiple files, view content of file, concatenate files and redirect output in terminal or files.



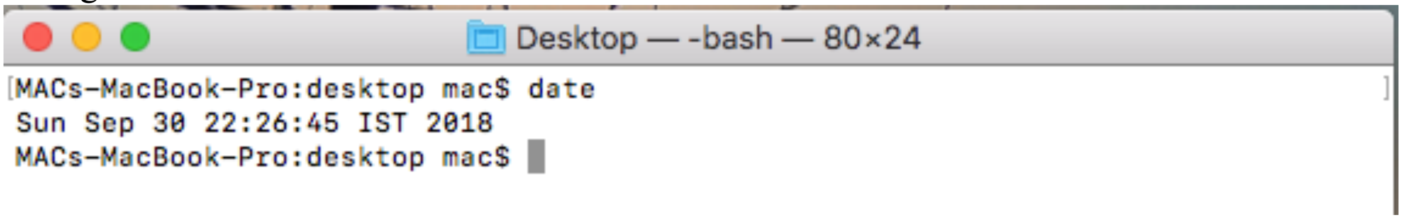
```
[MACs-MacBook-Pro:desktop mac$ cat shell1.sh
#!bin/sh
lsMACs-MacBook-Pro:desktop mac$ 
```

cal command is used to display calendar.



```
[MACs-MacBook-Pro:desktop mac$ cal
   September 2018
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
MACs-MacBook-Pro:desktop mac$ 
```

date command is used to display day, date, time and time zone with which linux computer is configured to.



```
[MACs-MacBook-Pro:desktop mac$ date
Sun Sep 30 22:26:45 IST 2018
MACs-MacBook-Pro:desktop mac$ 
```

Operating Systems Lab
(UCS-303)
Lab Assignment No.2

Name: Shivam Garg
Roll No: 101703512

1. **ls** command lists directory contents of files and directories.
   ls .. list root directory
   ls * list all the files first ans then directories in a cloumn format
   ls -a List all files including hidden file starting with '.'.
   ls -R option will list very long listing directory trees.
   ls -d */ list directories - with ' */'
   ls -l To show long listing information about the file/directory.
   ls -i list file's inode index number
   ls -t option flag sorts files/directories list by time/date.
   ls -S option flag sorts files/directories list by file size.
   ls -d $PWD/* command lists all files and directories along with the full path.
2. ex. Ls -al>listings
   here the output of commands is -al is redirected to file "listings" instead of the screen of terminal.
3. cat file1.txt file2.txt :command will concatenate copies of the contents of the two files file1, file2
   cat file1.txt > file2.txt : creates a new file named file2 that contains a copy of the contents of file1
   cat file1 file2 >file3 : The contents of each file will be displayed on the monitor screen starting on a new line and in the order that the file names appear in the command. This output could just as easily be redirected using the output redirection operator to file3
   cat file1 file2 | sort > file3

4. **echo** : The echo program displays text. It's a handy way to create customized output in your terminal.
   echo -e : Enable interpretation of backslash escape sequences.
5. **printf** : inserts arguments into a user-defined string of text, creating formatted output.
6. The cal command displays a simple, formatted calendar in your terminal.
   cal -m : Specify a month to display
   cal -y : Specify a year to display.
   cal -j : Display a Julian calendar, instead of the default Gregorian calendar. All days are numbered from January 1, rather than from the beginning of the month.
7. This cal command displays a simple, formatted calendar of February 2006 in your terminal.
8. **date** : This command is used to print out, or change the value of, the system's time and date information.
   date -d : Displays the given date string in the format of date. But this will not affect the system's actual date and time value.Rather it uses the date and time given in the form of string.

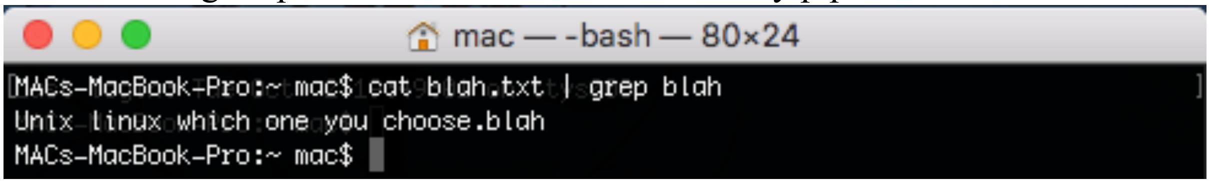
date -s : To set the system date and time -s or –set option is used.
date -u : Displays the time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time)time zone.
date -f : This is used to display the date string present at each line of file in the date and time format.This option is similar to –date option but the only difference is that in –date we can only give one date string but in a file we can give multiple date strings at each line.
date -r : This is used to display the last modified timestamp of a datefile .

Operating Systems Lab
(UCS-303)
LAB Assignment No.-3

Name: Shivam Garg
Roll No: 101703512

In details explanation of the following commands and topics:

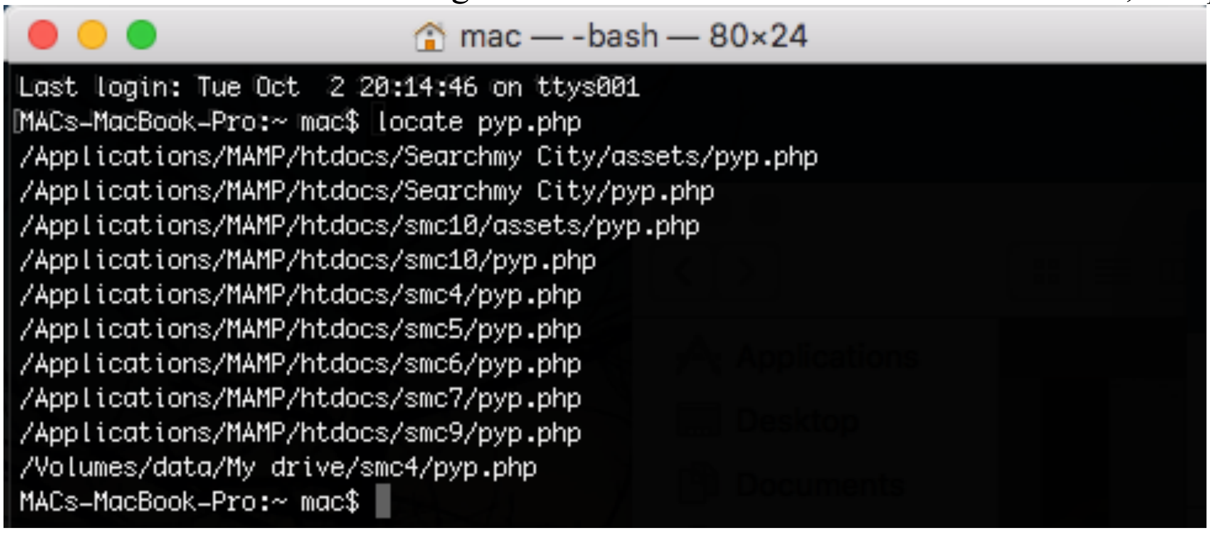
1. Feeding output of one command to another by pipeline.



```
MACs-MacBook-Pro:~ mac$ cat blah.txt | grep blah
Unix linux which one you choose.blah
MACs-MacBook-Pro:~ mac$
```

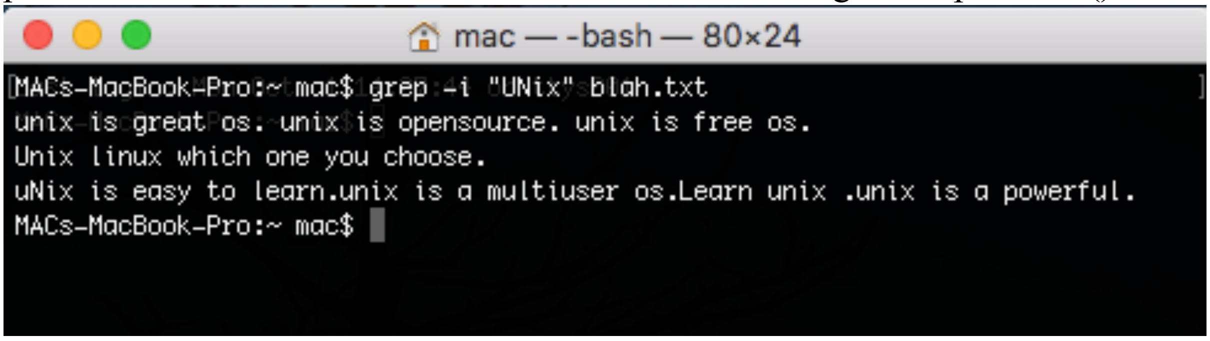
2. Locate, grep commands

Locate command finds files by name. It reads one or more databases created by updatedb and writes file names matching at least one of the PATTERNs to the screen, one per line.



```
Last login: Tue Oct  2 20:14:46 on ttys001
MACs-MacBook-Pro:~ mac$ locate pyp.php
/Applications/MAMP/htdocs/Searchmy City/assets/pyp.php
/Applications/MAMP/htdocs/Searchmy City/pyp.php
/Applications/MAMP/htdocs/smc10/assets/pyp.php
/Applications/MAMP/htdocs/smc10/pyp.php
/Applications/MAMP/htdocs/smc4/pyp.php
/Applications/MAMP/htdocs/smc5/pyp.php
/Applications/MAMP/htdocs/smc6/pyp.php
/Applications/MAMP/htdocs/smc7/pyp.php
/Applications/MAMP/htdocs/smc9/pyp.php
/Volumes/data/My drive/smc4/pyp.php
MACs-MacBook-Pro:~ mac$
```

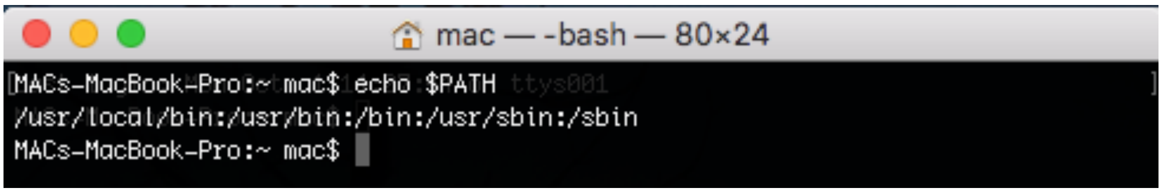
grep stands for globally search for regular expression and print out. The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression ().



```
MACs-MacBook-Pro:~ mac$ grep -i "UNix" blah.txt
unix is great os. unix is opensource. unix is free os.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
MACs-MacBook-Pro:~ mac$
```

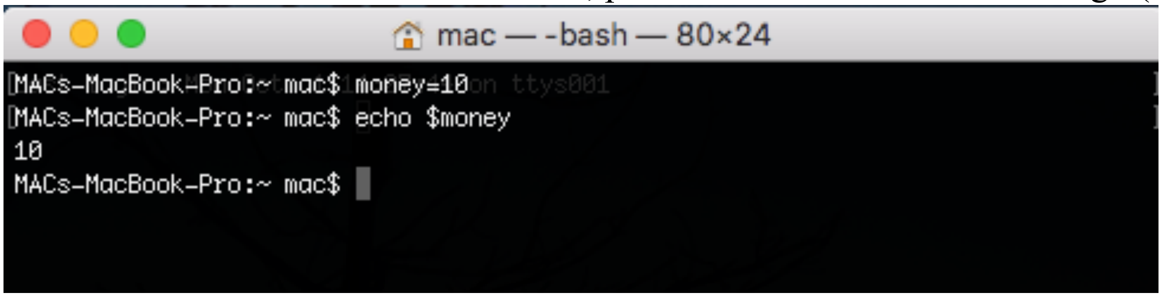
3. PATH and SHELL variable

The **path** variable stores a path where the linux looks for the commands that you type at the command-line.



```
MACs-MacBook-Pro:~ mac$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
MACs-MacBook-Pro:~ mac$
```

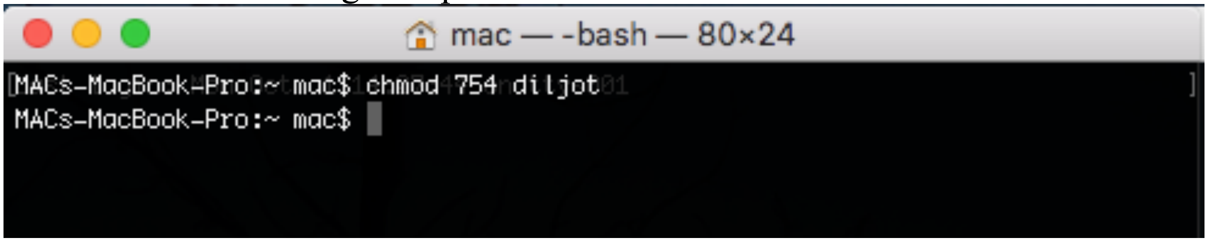
A shell variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.
A shell variable is nothing more than a pointer to the actual data. The shell enables you to create, assign, and delete variables.
Variables are defined as follows −
variable_name=variable_value
To access the value stored in a variable, prefix its name with the dollar sign (**$**)



```
MACs-MacBook-Pro:~ mac$ money=10
MACs-MacBook-Pro:~ mac$ echo $money
10
MACs-MacBook-Pro:~ mac$
```

4. chmod

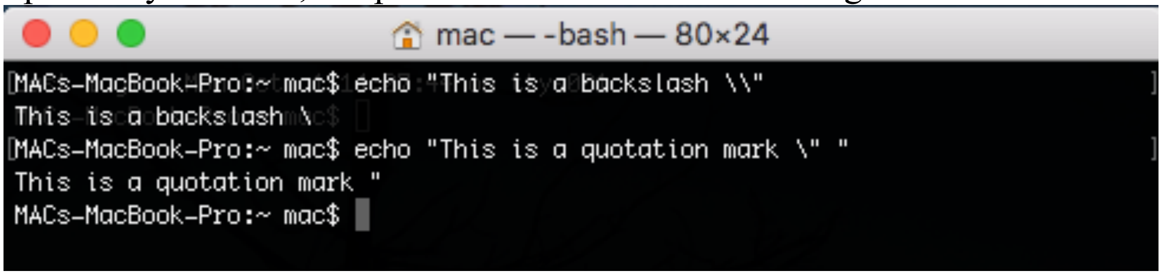
The command name **chmod** stands for "change mode", and it is used to define the way a file can be accessed.
chmod is used to change the permissions of files or directories.



```
MACs-MacBook-Pro:~ mac$ chmod 754 diljot
MACs-MacBook-Pro:~ mac$
```

5. Using escape sequence.

Most characters (*****, **'**, etc) are not interpreted (ie, they are taken literally) by means of placing them in double quotes (""). They are taken as is and passed on to the command being called. However, **"**, **$**, **`**, and \ are still interpreted by the shell, even when they're in double quotes. The backslash (\) character is used to mark these special characters so that they are not interpreted by the shell, but passed on to the command being run



```
MACs-MacBook-Pro:~ mac$ echo "This is a backslash \\"
This is a backslash \
MACs-MacBook-Pro:~ mac$ echo "This is a quotation mark \" "
This is a quotation mark "
MACs-MacBook-Pro:~ mac$
```

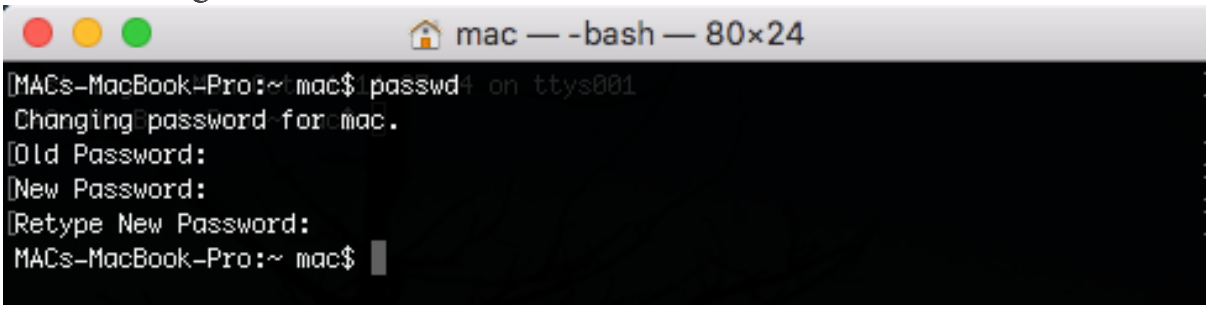
6. Internal and External commands.

- **Internal Commands :** Commands which are built into the shell. For all the shell built-in commands, execution of the same is fast in the sense that the shell doesn't have to search the given path for them in the PATH variable and also no process needs to be spawned for executing it.
  Examples: source, cd, fg etc.
- **External Commands :** Commands which aren't built into the shell. When an external command has to be executed, the shell looks for its path given in PATH variable and

also a new process has to be spawned and the command gets executed. They are usually located in /bin or /usr/bin. For example, when you execute the "cat" command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed.
Examples: ls, cat etc.

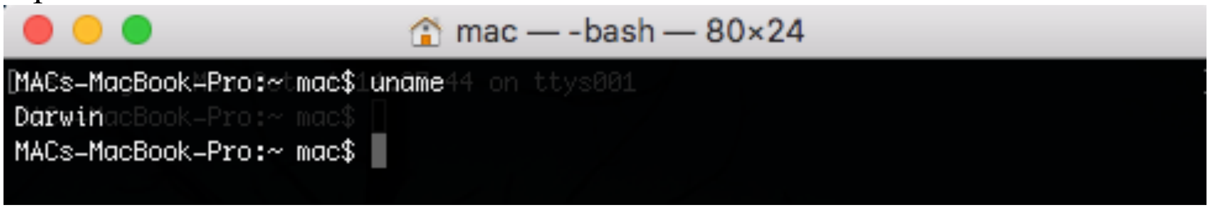
7. Commands - passwd, uname, tty, sty

The **passwd** command is used to change the password of a user account. A normal user can run passwd to change their own password, and a system administrator (the superuser) can use passwd to change another user's password, or define how that account's password can be used or changed.



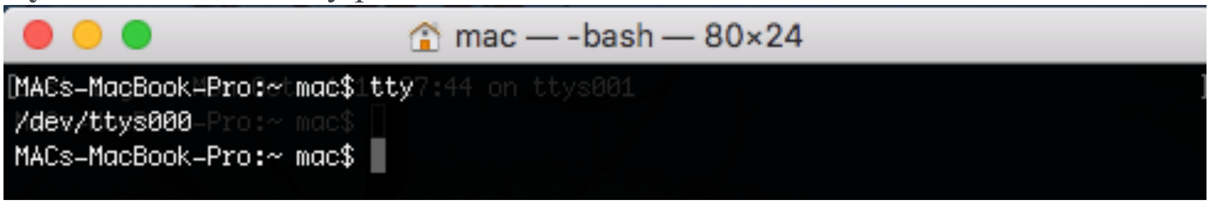
```
MACs-MacBook-Pro:~ mac$ passwd
Changing password for mac.
Old Password:
New Password:
Retype New Password:
MACs-MacBook-Pro:~ mac$
```

Uname command prints certain information about the current system. If no option is given, it prints the kernel name.



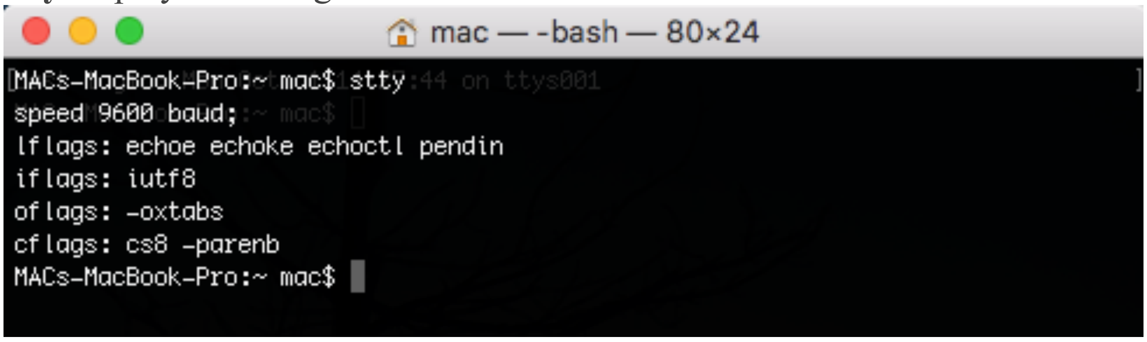
```
MACs-MacBook-Pro:~ mac$ uname
Darwin
MACs-MacBook-Pro:~ mac$
```

tty command basically prints the file name of the terminal connected to standard input.



```
MACs-MacBook-Pro:~ mac$ tty
/dev/ttys000
MACs-MacBook-Pro:~ mac$
```

stty displays or changes the characteristics of the terminal.



```
MACs-MacBook-Pro:~ mac$ stty
speed 9600 baud;
lflags: echoe echoke echoctl pendin
iflags: iutf8
oflags: -oxtabs
cflags: cs8 -parenb
MACs-MacBook-Pro:~ mac$
```

8. Types of files and file system in Linux.

There are seven different types of files in linux:

1. **-** : regular file
2. **d** : directory
3. **c** : character device file
4. **b** : block device file

5. **s** : local socket file
6. **p** : named pipe
7. **l** : symbolic link

Regular file
The regular file is a most common file type found on the Linux system. It governs all different files such us text files, images, binary files, shared libraries, etc. You can create a regular file with the touch command.
To remove a regular file you can use the rm command.

Directory
Directory is second most common file type found in Linux. Directory can be created with the mkdircommand.
directory can be identified by "d" symbol from the ls command output. To remove empty directory use the rmdir command.

Character device
Character and block device files allow users and programs to communicate with hardware peripheral devices.

Block Device
Block devices are similar to character devices.
They mostly govern hardware as hard drives, memory, etc.

Local domain sockets
Local domain sockets are used for communication between processes. Generally, they are used by services such as X windows, syslog and etc.
Sockets can be created by socket system call and removed by the unlink or rm commands.

Named Pipes
Similarly as Local sockets, named pipes allow communication between two local processes. They can be created by the mknod command and removed with the rm command.

Symbolic Links
With symbolic links an administrator can assign a file or directory multiple identities. Symbolic link can be though of as a pointer to an original file. There are two types of symbolic links:

- hard links
- soft links

The difference between hard and soft links is that soft links use file name as reference and hard links use direct reference to the original file. Furthermore, hard links cannot cross file systems and partitions. To create symbolic soft link we can use ln -s command. To remove symbolic link we can use unlink or rm command.

8. Directory structure in Linux

The following table provides a short overview of the most important higher-level directories you find on a Linux system.
Table 7.1. Overview of a Standard Directory Tree

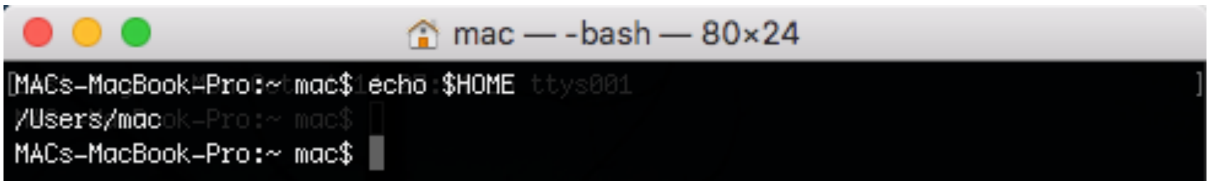
Director y	Contents
/	Root directory—the starting point of the directory tree.
/bin	Essential binary files, such as commands that are needed by both the system administrator and normal users. Usually also contains the shells, such as Bash.
/boot	Static files of the boot loader.
/dev	Files needed to access host-specific devices.
/etc	Host-specific system configuration files.
/lib	Essential shared libraries and kernel modules.
/media	Mount points for removable media.
/mnt	Mount point for temporarily mounting a file system.
/opt	Add-on application software packages.
/root	Home directory for the superuser root.
/sbin	Essential system binaries.
/srv	Data for services provided by the system.
/tmp	Temporary files.
/usr	Secondary hierarchy with read-only data.
/var	Variable data such as log files
/ windows	Only available if you have both Microsoft Windows* and Linux installed on your system. Contains the Windows data.

/bin
Contains the basic shell commands that may be used both by root and by other users. These commands include **ls**, **mkdir**, **cp**, **mv**, **rm**, and **rmdir**. /bin also contains Bash, the default shell in openSUSE.
/boot
Contains data required for booting, such as the boot loader, the kernel, and other data that is used before the kernel begins executing user mode programs.
/dev
Holds device files that represent hardware components.
/etc
Contains local configuration files that control the operation of programs like the X Window System. The /etc/init.d subdirectory contains scripts that are executed during the boot process.
/home/*username*
Holds the private data of every user who has an account on the system. The files located here can only be modified by their owner or by the system administrator. By default, your e-mail directory and personal desktop configuration are located here.

/lib
Contains essential shared libraries needed to boot the system and to run the commands in the root file system. The Windows equivalent for shared libraries are DLL files.
/media
Contains mount points for removable media, such as CD-ROMs, USB sticks, and digital cameras (if they use USB). /media generally holds any type of drive except the hard drive of your system. As soon as your removable medium has been inserted or connected to the system and has been mounted, you can access it from here.
/mnt
This directory provides a mount point for a temporarily mounted file system. root may mount file systems here.
/opt
Reserved for the installation of additional software. Optional software and larger add-on program packages, such as the KDE and GNOME desktop environments, can be found here.
/root
Home directory for the root user. Personal data of root is located here.
/sbin
As the s indicates, this directory holds utilities for the superuser. /sbin contains binaries essential for booting, restoring, and recovering the system in addition to the binaries in /bin.
/**srv**
Holds data for services provided by the system, such as FTP and HTTP.
/**tmp**
This directory is used by programs that require temporary storage of files. By default, the data stored in /tmp is deleted regularly.
/usr
/usr has nothing to do with users, but is the acronym for UNIX system resources. The data in /usr is static, read-only data that can be shared among various hosts compliant to the Filesystem Hierarchy Standard (FHS). This directory contains all application programs and establishes a secondary hierarchy in the file system. /usr holds a number of subdirectories, such as /usr/bin, /usr/sbin, /usr/local, and /usr/share/doc.
/usr/bin
Contains generally accessible programs.
/**usr/sbin**
Contains programs reserved for the system administrator, such as repair functions.
/usr/local
In this directory, the system administrator can install local, distribution-independent extensions.
/usr/share/doc
Holds various documentation files and the release notes for your system.
/var
Whereas /usr holds static, read-only data, /var is for data which is written during system operation and thus is variable data, such as log files or spooling data.
/windows
Only available if you have both Microsoft Windows and Linux installed on your system. Contains the Windows data available on the Windows partition of your system.

9. HOME variable

HOME variable is a linux variable which is the default path to the user's home directory.



```
MACs-MacBook-Pro:~ mac$ echo $HOME
/Users/mac
MACs-MacBook-Pro:~ mac$
```

10. Absolute and relative pathname

An **absolute path** is defined as specifying the location of a file or directory from the root directory(/). In other words,we can say that an absolute path is a complete path from start of actual file system from / directory.

Some examples of absolute path:

/var/ftp/pub
/etc/samba.smb.conf
/boot/grub/grub.conf

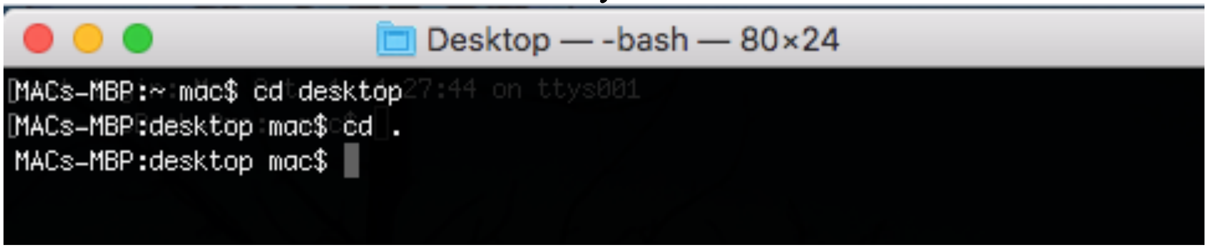
Relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and **never starts with a /** .

Some examples of absolute path:

pub
samba.smb.conf
grub.conf

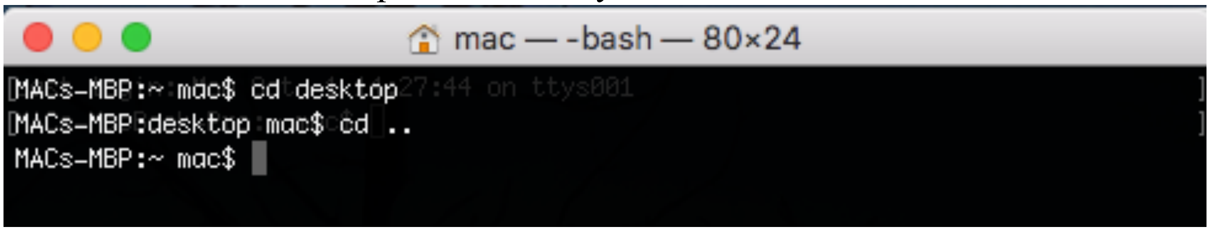
11. Using . and ..

'.' is used to refer to the current directory.



```
MACs-MBP:~ mac$ cd desktop
MACs-MBP:desktop mac$ cd .
MACs-MBP:desktop mac$
```

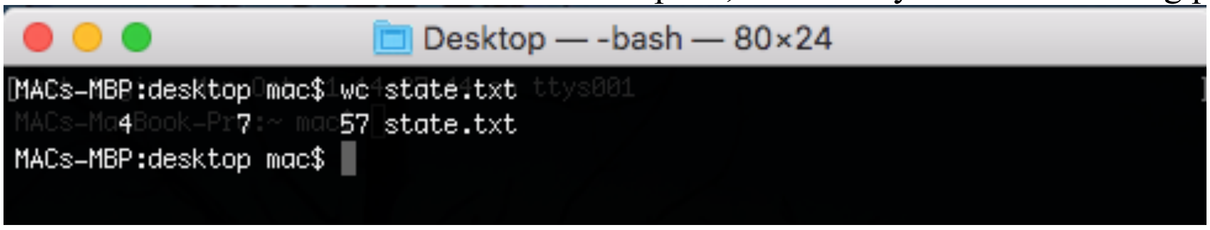
'..' is used to refer to the parent directory.



```
MACs-MBP:~ mac$ cd desktop
MACs-MBP:desktop mac$ cd ..
MACs-MBP:~ mac$
```

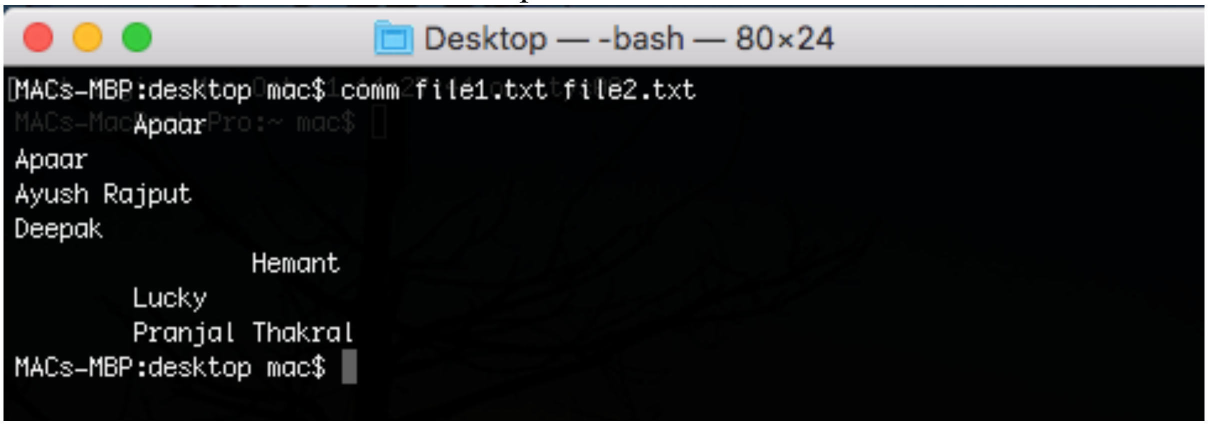
12. Commands - wc, comm, cmp, diff

wc stands for **word count**. As the name implies, it is mainly used for counting purpose.



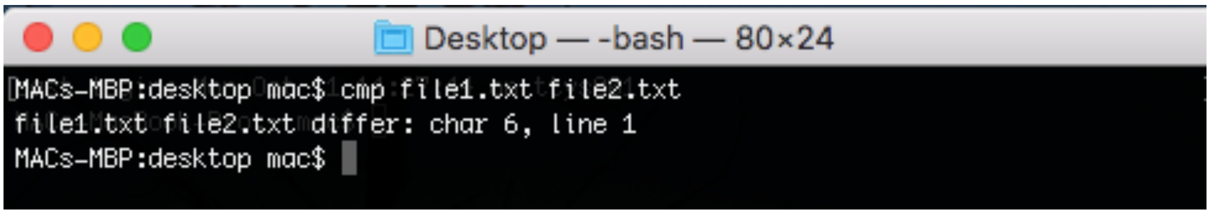
```
MACs-MBP:desktop mac$ wc state.txt
       4       7      57 state.txt
MACs-MBP:desktop mac$
```

comm compare two sorted files line by line and write to standard output; the lines that are common and the lines that are unique.



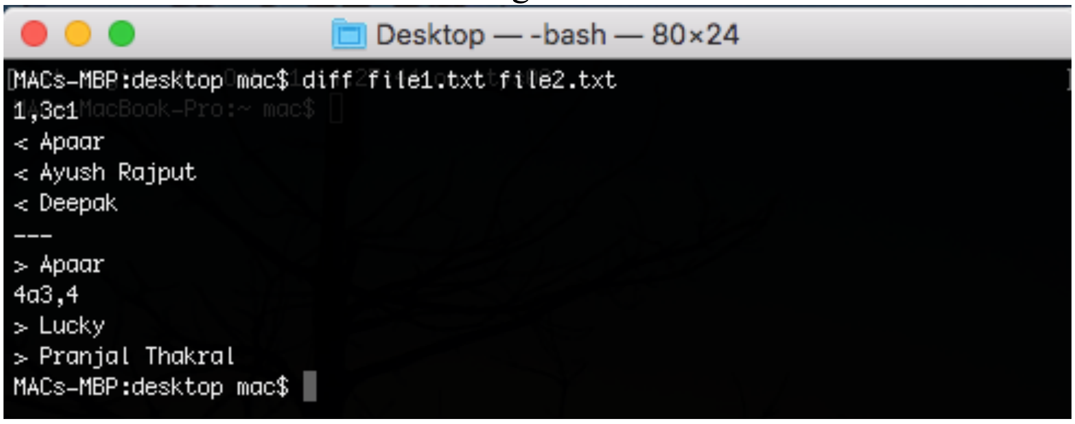
```
MACs-MBP:desktop mac$ comm file1.txt file2.txt
        Apaar
Apaar
Ayush Rajput
Deepak
                Hemant
        Lucky
        Pranjal Thakral
MACs-MBP:desktop mac$
```

cmp command in Linux/UNIX is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.



```
MACs-MBP:desktop mac$ cmp file1.txt file2.txt
file1.txt file2.txt differ: char 6, line 1
MACs-MBP:desktop mac$
```

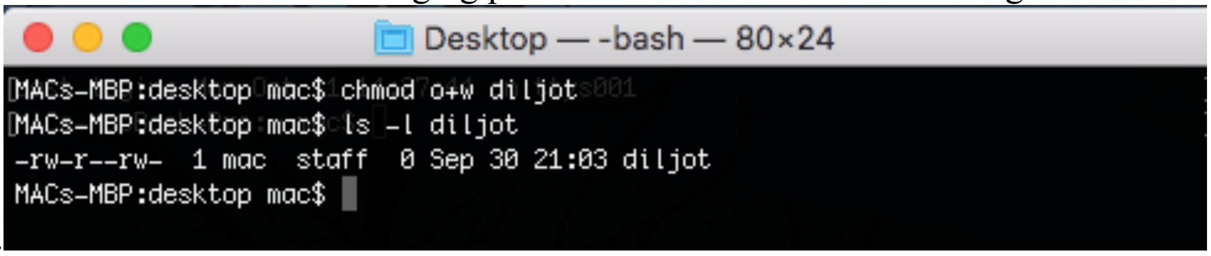
diff stands for difference. This command is used to display the differences in the files by comparing the files line by line. Unlike its fellow members, cmp and comm, it tells us which lines in one file have is to be changed to make the two files identical.



```
MACs-MBP:desktop mac$ diff file1.txt file2.txt
1,3c1
< Apaar
< Ayush Rajput
< Deepak
---
> Apaar
4a3,4
> Lucky
> Pranjal Thakral
MACs-MBP:desktop mac$
```

13. Adding permissions

Adding permissions is similar to changing permissions and can be done through chmod command.



```
MACs-MBP:desktop mac$ chmod o+w diljot
MACs-MBP:desktop mac$ ls -l diljot
-rw-r--rw-  1 mac  staff  0 Sep 30 21:03 diljot
MACs-MBP:desktop mac$
```

14. File permissions and changing the access rights.

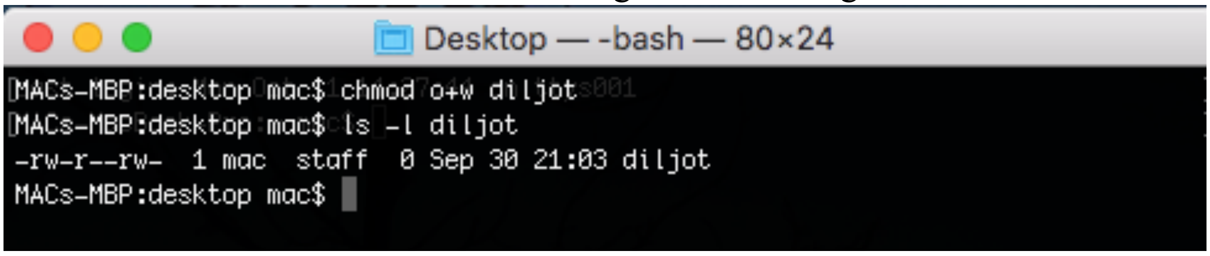
File Permissions

Every file and directory in your Linux system has following 3 permissions:

Read: This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.

Write: The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory.

Execute: In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.
We can use the chmod command to change the access rights in Linux.



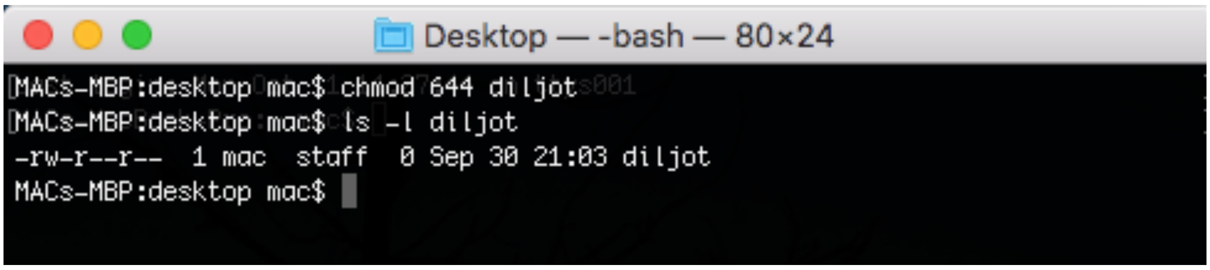
```
MACs-MBP:desktop mac$ chmod o+w diljot
MACs-MBP:desktop mac$ ls -l diljot
-rw-r--rw-  1 mac  staff  0 Sep 30 21:03 diljot
MACs-MBP:desktop mac$
```

16. Relative and absolute permission.
i. Absolute Permission   ii. Relative Permission
i. Absolute Permission: Each permission setting can be represented by a numerical value:

- read = 4
- write = 2
- execute = 1

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

For example, if you want read and write permissions, you would have a value of 6; 4 (read) + 2 (write) = 6.



```
MACs-MBP:desktop mac$ chmod 644 diljot
MACs-MBP:desktop mac$ ls -l diljot
-rw-r--r--  1 mac  staff  0 Sep 30 21:03 diljot
MACs-MBP:desktop mac$
```

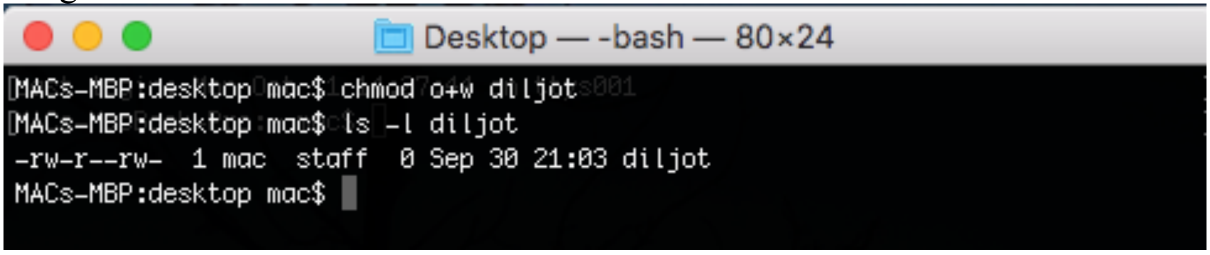
ii. Relative Permission: you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

Operator	Description
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as –

User Denotations	
u	user/owner
g	group
o	other
a	all

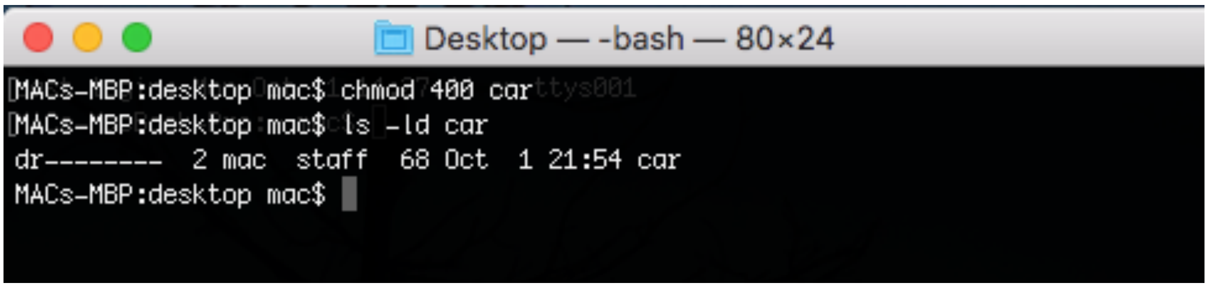
The chmod command takes as its argument an expression comprising somw letters and symbols that completely describe the user category and the type of permission being assigned or removed.



```
MACs-MBP:desktop mac$ chmod o+w diljot
MACs-MBP:desktop mac$ ls -l diljot
-rw-r--rw-  1 mac  staff  0 Sep 30 21:03 diljot
MACs-MBP:desktop mac$
```

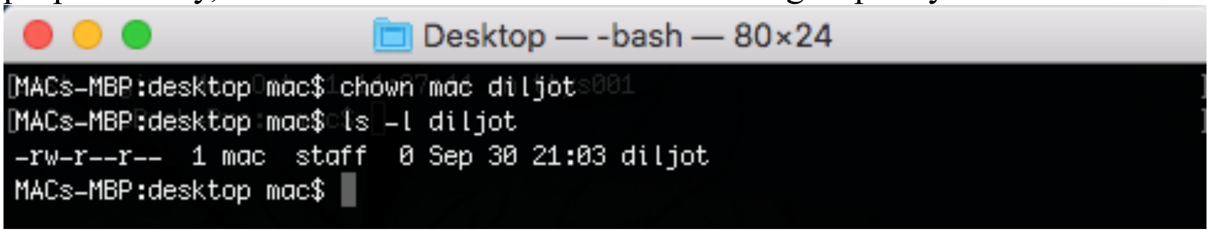
17. Directory permission.
The same series of permissions may be used for directories but they have a slightly different behaviour.

- **r** - you have the ability to read the contents of the directory (ie do an ls)
- **w** - you have the ability to write into the directory (ie create files and directories)
- **x** - you have the ability to enter that directory (ie cd)



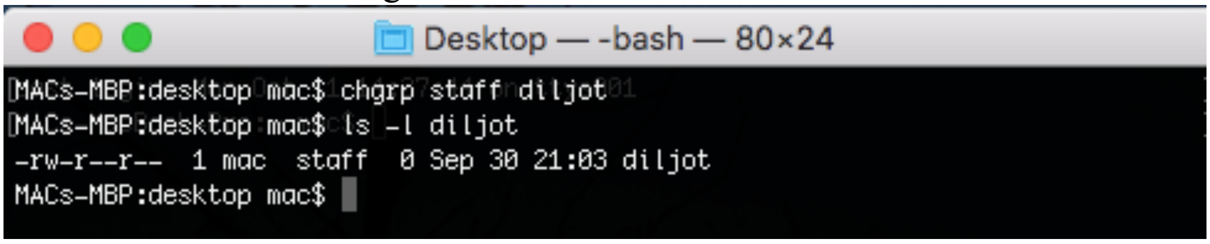
```
MACs-MBP:desktop mac$ chmod 400 car
MACs-MBP:desktop mac$ ls -ld car
dr--------  2 mac  staff  68 Oct  1 21:54 car
MACs-MBP:desktop mac$
```

18. Changing file ownership.
You can change the ownership of a specific file using the *chown command.* For security purposes only, the root user or members of the sudo group may transfer ownership of a file.



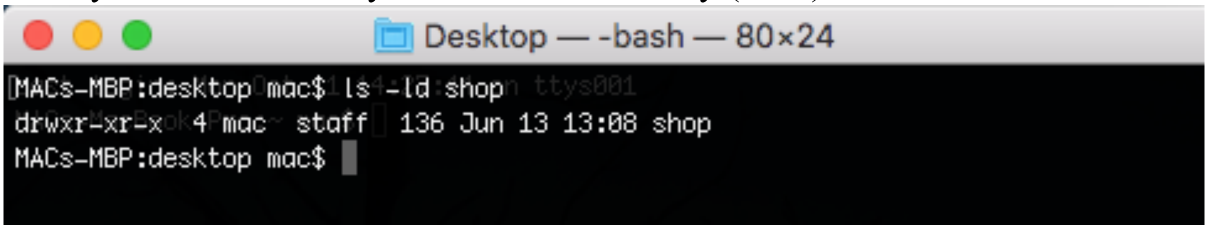
```
MACs-MBP:desktop mac$ chown mac diljot
MACs-MBP:desktop mac$ ls -l diljot
-rw-r--r--  1 mac  staff  0 Sep 30 21:03 diljot
MACs-MBP:desktop mac$
```

19. Changing group of a particular file.
Through **chgrp** command, a user can change the group owner of a file but only to a group to which the user also belongs.



```
MACs-MBP:desktop mac$ chgrp staff diljot
MACs-MBP:desktop mac$ ls -l diljot
-rw-r--r--  1 mac  staff  0 Sep 30 21:03 diljot
MACs-MBP:desktop mac$
```

20. r/w/x permissions for directory.
The r/w/x permissions for the directory are:
  r - you have the ability to read the contents of the directory (ie do an ls)
  w - you have the ability to write into the directory (ie create files and directories)
  x - you have the ability to enter that directory (ie cd)



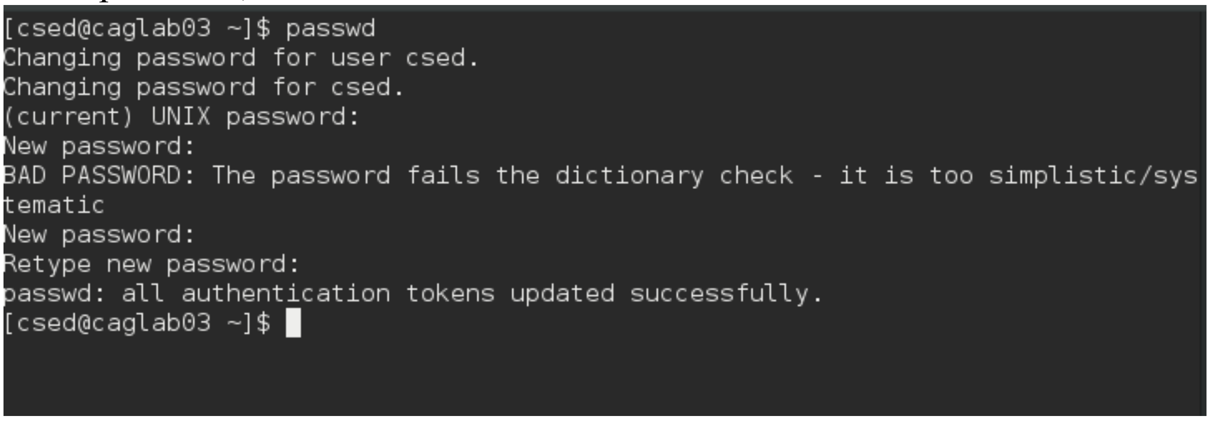
```
MACs-MBP:desktop mac$ ls -ld shop
drwxr-xr-x  4 mac  staff  136 Jun 13 13:08 shop
MACs-MBP:desktop mac$
```

Operating Systems Lab
(UCS-303)
Lab Assignment No.4

Name: Shivam Garg
Roll No: 101703512

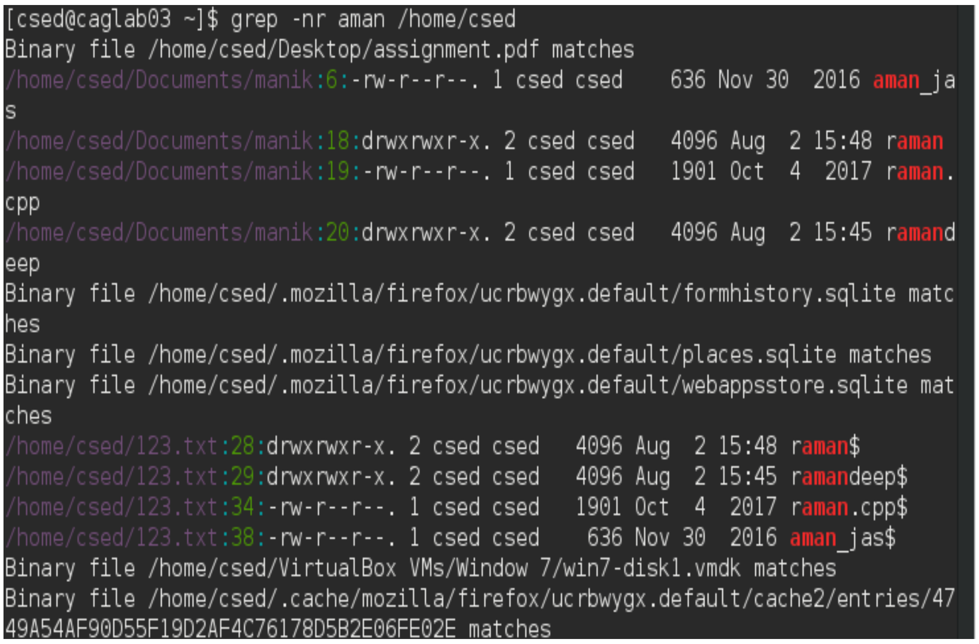
Write and execute the following UNIX commands:

1. To change the password.
   passwd command is used to change a user's password. The password entered by the user is run through a key derivation function to create a hashed version of the new password, which is saved.



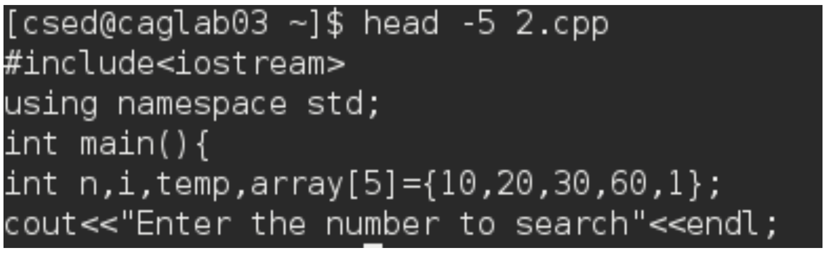
```
[csed@caglab03 ~]$ passwd
Changing password for user csed.
Changing password for csed.
(current) UNIX password:
New password:
BAD PASSWORD: The password fails the dictionary check - it is too simplistic/sys
tematic
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[csed@caglab03 ~]$
```

2. To search files in the current directory/subdirectory for lines that match a particular string pattern given as input.



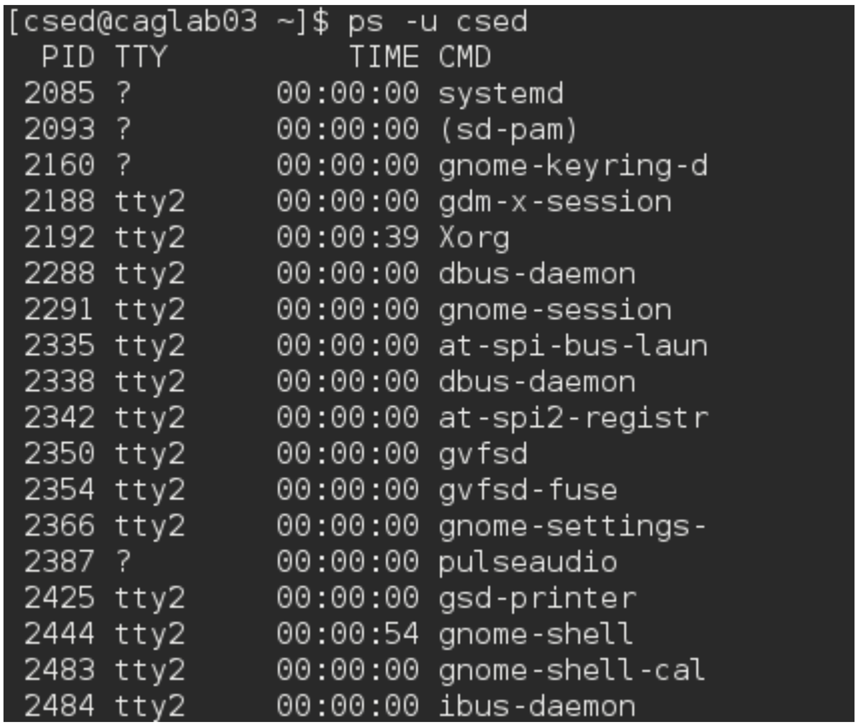
```
[csed@caglab03 ~]$ grep -nr aman /home/csed
Binary file /home/csed/Desktop/assignment.pdf matches
/home/csed/Documents/manik:6:-rw-r--r--. 1 csed csed    636 Nov 30  2016 aman_ja
s
/home/csed/Documents/manik:18:drwxrwxr-x. 2 csed csed   4096 Aug  2 15:48 raman
/home/csed/Documents/manik:19:-rw-r--r--. 1 csed csed   1901 Oct  4  2017 raman.
cpp
/home/csed/Documents/manik:20:drwxrwxr-x. 2 csed csed   4096 Aug  2 15:45 ramand
eep
Binary file /home/csed/.mozilla/firefox/ucrbwygx.default/formhistory.sqlite matc
hes
Binary file /home/csed/.mozilla/firefox/ucrbwygx.default/places.sqlite matches
Binary file /home/csed/.mozilla/firefox/ucrbwygx.default/webappsstore.sqlite mat
ches
/home/csed/123.txt:28:drwxrwxr-x. 2 csed csed   4096 Aug  2 15:48 raman$
/home/csed/123.txt:29:drwxrwxr-x. 2 csed csed   4096 Aug  2 15:45 ramandeep$
/home/csed/123.txt:34:-rw-r--r--. 1 csed csed   1901 Oct  4  2017 raman.cpp$
/home/csed/123.txt:38:-rw-r--r--. 1 csed csed    636 Nov 30  2016 aman_jas$
Binary file /home/csed/VirtualBox VMs/Window 7/win7-disk1.vmdk matches
Binary file /home/csed/.cache/mozilla/firefox/ucrbwygx.default/cache2/entries/47
49A54AF90D55F19D2AF4C76178D5B2E06FE02E matches
```

3. To print the first 5 lines of a file.
   Head command is used.



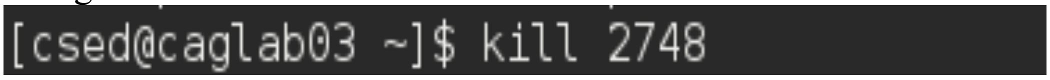
```
[csed@caglab03 ~]$ head -5 2.cpp
#include<iostream>
using namespace std;
int main(){
int n,i,temp,array[5]={10,20,30,60,1};
cout<<"Enter the number to search"<<endl;
```

4. To print the number of processes run by a particular user.
The ps command reports information on current running processes, outputting to standard output.



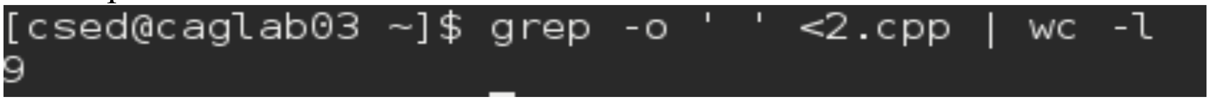
```
[csed@caglab03 ~]$ ps -u csed
  PID TTY          TIME CMD
 2085 ?        00:00:00 systemd
 2093 ?        00:00:00 (sd-pam)
 2160 ?        00:00:00 gnome-keyring-d
 2188 tty2     00:00:00 gdm-x-session
 2192 tty2     00:00:39 Xorg
 2288 tty2     00:00:00 dbus-daemon
 2291 tty2     00:00:00 gnome-session
 2335 tty2     00:00:00 at-spi-bus-laun
 2338 tty2     00:00:00 dbus-daemon
 2342 tty2     00:00:00 at-spi2-registr
 2350 tty2     00:00:00 gvfsd
 2354 tty2     00:00:00 gvfsd-fuse
 2366 tty2     00:00:00 gnome-settings-
 2387 ?        00:00:00 pulseaudio
 2425 tty2     00:00:00 gsd-printer
 2444 tty2     00:00:54 gnome-shell
 2483 tty2     00:00:00 gnome-shell-cal
 2484 tty2     00:00:00 ibus-daemon
```

5. To kill a process that is running at the background.
Kill command in UNIX and Linux is normally used to kill a process running at the background.



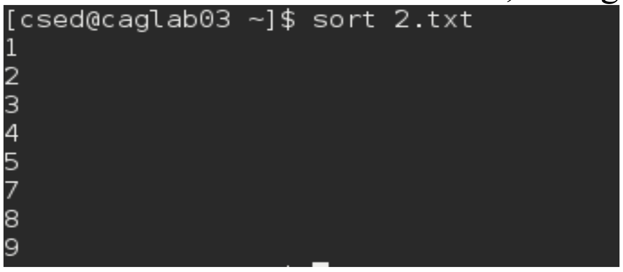
```
[csed@caglab03 ~]$ kill 2748
```

6. To display the count of no. of blank spaces in a given file.
Grep command is used.



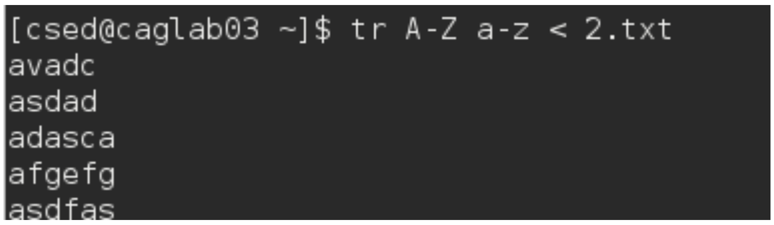
```
[csed@caglab03 ~]$ grep -o ' ' <2.cpp | wc -l
9
```

7. To sort alphabetically, a list of numbers stored in a data file in an ascending order.
SORT command is used to sort a file, arranging the records in a particular order.



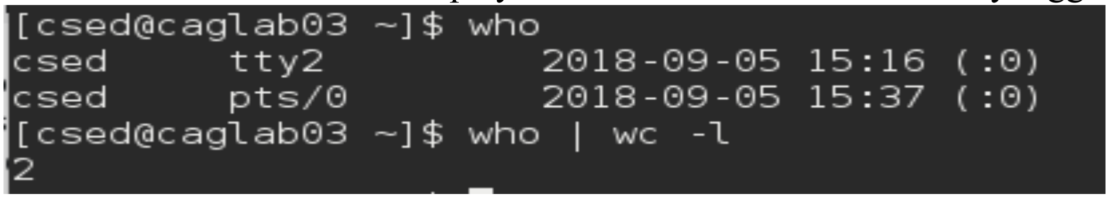
```
[csed@caglab03 ~]$ sort 2.txt
1
2
3
4
5
7
8
9
```

8. To convert the upper case letters to corresponding lower case letters in a text file.
tr command IS USED to convert all incoming text / words / variable data from upper to lower case or vise versa



```
[csed@caglab03 ~]$ tr A-Z a-z < 2.txt
avadc
asdad
adasca
afgefg
asdfas
```

9. To count the number of users currently logged on.

The standard UNIX command is who which displays a list of users who are currently logged into the computer. 

```
[csed@caglab03 ~]$ who
csed     tty2         2018-09-05 15:16 (:0)
csed     pts/0        2018-09-05 15:37 (:0)
[csed@caglab03 ~]$ who | wc -l
2
```

In details explanation of the following commands and topics:

1. Shell script

A **shell script** is a computer program designed to be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text

2. Read operation in shell script

Reading can be done through accepting user input, A common use for user-created variables is storing information that a user enters in response to a prompt.
Also using read, scripts can accept input from the user and store that input in variables

3. Using command line argument

The Unix shell is used to run commands, and it allows users to pass run time arguments to these commands. These arguments, also known as command line parameters, that allows the users to either control the flow of the command or to specify the input data for the command.

4. exit and EXIT status of commands

The **exit command** terminates a script, just as in a C program. It can also return a value, which is available to the script's parent process.

5. Logical operators &&, ||

   Logical AND Operator:- If both the operands are true, then the condition becomes true otherwise false. Logical AND in bash script is used with operator -a.
   Example: [$a -lt 20 -a $b -gt 100] is false.

   Logical OR Operator:- If one of the operands is true, then the condition becomes true. Logical OR in bash script is used with operator -o.
   Example: [$a -lt 20 -o $b -gt 100] is true.

-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

6. Conditional construct – If

If statements allow us to make decisions in our Bash scripts. They allow us to decide whether or not to run a piece of code based upon conditions that we may set. If statements, combined with loops allow us to make much more complex scripts which may solve larger tasks.

Example:

```
#!/bin/bash
# Basic if statement
if [ $1 -gt 100 ]
then
echo Hey that\'s a large number.
pwd
fi
```

7. Using test AND [ ] to evaluate an expression

we **test** is used as part of the conditional execution of shell commands.

test exits with the status determined by EXPRESSION. Placing the EXPRESSION between square brackets (**[** and **]**) is the same as testing the EXPRESSION with **test**. To see the exit status at the command prompt, echo the value "**$?**" A value of 0 means the expression evaluated as true, and a value of 1 means the expression evaluated as false.

8. Numeric comparison

Through the following comparison operators, numeric comparisons can be performed.

▪ **num1 -eq num2**	check if 1st number is equal to 2nd number
▪ **num1 -ge num2**	checks if 1st number is greater than or equal to 2nd number
▪ **num1 -gt num2**	checks if 1st number is greater than 2nd number
▪ **num1 -le num2**	checks if 1st number is less than or equal to 2nd number
▪ **num1 -lt num2**	checks if 1st number is less than 2nd number
▪ **num1 -ne num2**	checks if 1st number is not equal to 2nd number

9. String comparison

The various string comparisons are:

▪ **var1 = var2**	checks if var1 is the same as string var2
▪ **var1 != var2**	checks if var1 is not the same as var2
▪ **var1 < var2**	checks if var1 is less than var2
▪ **var1 > var2**	checks if var1 is greater than var2
▪ **-n var1**	checks if var1 has a length greater than zero
▪ **-z var1**	checks if var1 has a length of zero

10.Conditional construct – Case

The **case…esac Conditional construct statement** is used to check for multiple conditions for a decision making process in a shell script.

- It is better alternative for Ladder if-elif conditional construct statement.
- It evaluates a value of the variable and compares it with case specified by the user.
- The commands under a particular case value must be followed by a pair of semicolon (;;) to delimit it from set of commands under the next case value.

Syntax

```
case $variable_name in
                case1_pattern)
                        <commands>
                ;;
                case2_pattern)
                        <commands>
                ;;
                case3_pattern)
                        <commands>
                ;;
                .
                .
                *)                                                    //default_case
                        <commands>
                ;;
                esac
```

11.Matching multiple patterns
Multiple patters can be matched using
grep and egrep commands:

```
$ grep "PATTERN1\|PATTERN2" FILE
$ grep -E "PATTERN1|PATTERN2" FILE
$ grep -e PATTERN1 -e PATTERN2 FILE
$ egrep "PATTERN1|PATTERN2" FILE
```

Using awk command:

```
$ awk '/PATTERN1|PATTERN2/' FILE
```

Using sed command:

```
$ sed -e '/PATTERN1/b' -e '/PATTERN2/b' -e d FILE
```

12.expr – Computational and String handling
expr is a command line Unix utility which evaluates an expression and outputs the corresponding value. expr evaluates integer or string expressions, including pattern matching regular expressions. Most of the challenge posed in writing expressions is preventing the invoking command line shell from acting on characters intended for expr to process. The operators available for integers: addition, subtraction, multiplication, division and modulus for strings: find regular expression, find a set of characters in a string; in some versions: find substring, length of string for either: comparison (equal, not equal, less than, etc

13.Looping – While, For
While loop is used where a set of commands are to be executed till a condition remains true. Condition is checked at the beginning of the loop.
Example:

```
#!/bin/sh
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
```

```
do
  echo "Please type something in (bye to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done
```

for loops iterate through a set of values until the list is exhausted:

Example:

```
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

Operating Systems Lab
(UCS-303)
Lab Assignment No.5

Name: Shivam Garg
Roll No: 101703512

Write shell programs for the following:

1. To find second largest number among the 5 numbers given.

```
read -p "Enter first Number:" n1

read -p "Enter second Number:" n2

read -p "Enter third Number:" n3

read -p "Enter fourth Number:" n4

read -p "Enter fourth Number:" n5

if[ [ n1 -gt n2 ] && [ n1 -gt n2 ] && [ n1 -gt n3 ] && [ n1 -gt n4 ] && [ n1 -gt n5 ]] ; then

echo "$n1 is a Greatest Number"

elif[ [ n2 -gt n3 ] && [ n2 -gt n3 ] && [ n2 -gt n4 ] && [ n2 -gt n5 ]] ; then

echo "$n2 is a Greatest Number"

elif[ [ n3 -gt n4 ] && [ n3 -gt n5 ] ] ; then

echo "$n3 is a Greatest Number"

elif[ n4 -gt n5 ] ; then

echo "$n4 is a Greatest Number"

else

echo "$n5 is a Greatest Number"
```

```
fi
```

2. To find sum of all the alternate digits in a given 7 digit number.

```
echo “Enter a 7 digit number”
read NUM
N=1
while [ $N -le 7]
do
a = echo $NUM | cut –c $N
echo $a
N =`expr $N + 2`
done
```

3. To count number of vowels in a given string.

```
clear

echo -n "Enter the Name / String:"

while :

do

read name

echo $name | awk '{print gsub(/[aeiou]/,"")}'
done
```

4. (d) To take 2 strings as input, concatenate them and display

```
echo "enter string"
echo Enter first string : read s1

echo Enter second string : read s2
s3 = $s1$s2

len = `echo $s3 | wc -c`
len = `expr $len -1`

echo Concatinated string is $s3 of length $ len
```

5. To take 2 strings as input, concatenate them and display the length of the resultant string.

```
echo Enter first string:
read s1
echo Enter second string:
```

```
read s2
s3=$s1$s2
len=`echo $s3 | wc -c`
len=`expr $len - 1`
echo Concatinated stringis $s3 of length $len
```

6. To display the reverse of a given number.

```
 echo "Enter a number"
read n
sd=0
rev=0
while [ $n -gt 0 ]
do
   sd=$(( $n % 10 ))
   rev=$(( $rev *\ 10 + $sd ))
   n=$(( $n / 10 ))
done
echo "Reverse number of entered digit is $rev"
```

7. Write a shell program to count the number of special symbols, end of line characters and blank-spaces present in a text file. Redirect the output to a file called as output.

```
#!/bin/sh
if [ -d "$@" ]; then

echo "Number of files is $(find "$@" -type f | wc -l)"

echo "Number of directories is $(find "$@" -type d | wc -l)"
else

echo "[ERROR] Please provide a directory."

exit 1

fi
```

8. Write a shell program to count no. of characters, vowels, special symbols and blank spaces in a given file provided by the user as input and individually display the count.

```
Echo “Enter a text”

Read text

w = `Echo $text | wc -w`
```

```
w = `expr $w`

c = `Echo $text | wc -c`

c = `expr $c – 1`

s=0

alpha = 0

j=``

n=1

while [ $n -le $c ]

do

ch = `Echo $text | cut -c $n`
if test $ch = $j

then

s = `expr $s + 1`
fi
case $ch in

a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z) alpha=`expr $alpha + 1`;;
esac
n = `expr $n + 1`

done

special = `expr $c – $s – $alpha`

Echo "Words = $w"

Echo "Characters = $c"

Echo "Spaces = $s"

Echo "Special symbols = $special"
```

9. Write a shell script to check whether the year given as input is a leap year or not.

```
echo -n “Enter the year(yyyy) to find leap year :- ”
read year

d=`expr $year % 4`

b=`expr $year % 100`
c=`expr $year % 400`

if [ $d -eq 0 -a $b -ne 0 -o $c -eq 0 ]
then
echo “year is leap year”

else
echo “not leap year”

fi
```

10. Write a shell script to add two matrices A and B of size 3 x 2 matrix.

```
#!/bin/bash

read -p "Enter the matrix order [mxn] : " t

m=${t:0:1}
n=${t:2:1}

echo "Enter the elements for first matrix"

for i in `seq 0 $(($m-1))`

do

for j in `seq 0 $(($n-1))`

Do

read x[$(($n*$i+$j))]

done

done
```

```
echo "Enter the elements for second matrix"

for i in `seq 0 $(($m-1))`

do
for j in `seq 0 $(($n-1))`

do

read y[$(($n*$i+$j))] z[$(($n*$i+$j))]=$((${x[$(($n*$i+$j))]}+${y[$(($n*$i+$j))]}))
done

done

echo "Matrix after addition is"

for i in `seq 0 $(($m-1))`

do

for j in `seq 0 $(($n-1))`
do

echo -ne "${z[$(($n*$i+$j))]}\t"
done

echo -e "\n"

done

exit 0
```