

LINUX SHELL SCRIPTING LABORATORY MANUAL

B.TECH (III SEM), AY: 2021-2022

Batch 2020-2024



SRI VASAVI ENGINEERING COLLEGE (Autonomous)

Pedatadepalli, TADEPALLIGUDEM-534 101, W.G. Dist.

Department of Computer Science & Engineering (Accredited by NBA)

SRI VASAVI ENGINEERING COLLEGE (Autonomous)
PEDATADEPALLI, TADEPALLIGUDEM.



Certificate

*This is to certify that this is a bonafide record of Practical Work done in
LINUX SHELL SCRIPTING Lab by Mr. /Miss _____
bearing RollNo. _____ of **CSE Branch** of
III Semester during the academic year **2021-22**.*

No. of Experiments Done:

Faculty In charge of the Laboratory

Head of the Department

EXTERNAL EXAMINER

LIST OF EXPERIMENTS:

1. Experiment the following Unix Commands:

- a) General Purpose Utilities: cal, date, man, who.
- b) Directory Handling Commands: pwd, cd, mkdir, rmdir.
- c) File Handling Utilities: cat, cp, ls, rm, nl, wc
- d) Displaying Commands: head, tail
- e) Filters: cmp, comm., diff, sort, uniq
- f) Disk Utilities: du, df

2. Develop a Shell Program to Display all the words which are entered as command line arguments.

3. Develop a shell script that Changes Permissions of files in PWD as rwx for users.

4. Develop a shell script to print the list of all sub directories in the current directory.

5. Develop a Shell Program which receives any year from the keyboard and determine whether the year is leap year or not. If no argument is supplied the current year should be assumed.

6. Develop a shell script which takes two file names as arguments-If their contents are same then delete the second file.

7. Develop a shell script to print the given number in the reversed order.

8. Develop a shell script to print first 25 Fibonacci numbers.

9. Develop a shell script to print the Prime numbers between the specified range.

10. Develop a shell script to delete all lines containing the word 'unix' in the files supplied as arguments.

11. Develop a shell script Menu driven program which has the following options.

- i) contents of /etc/passwd
- ii) list of users who have currently logged in.
- iii) present working directory.
- iv) exit.

Computer :

It is an electronic device used for storing and processing data in binary form.

Program :

Set of instructions used by computer to perform specific task.

Software :

It is a program and other operating information used by computer.

- System software: Used by OS .
- Application software: Used by users.

Operating System (OS):

OS is system software that manages computer hardware, software resources, and provides common services for computer programs.

or

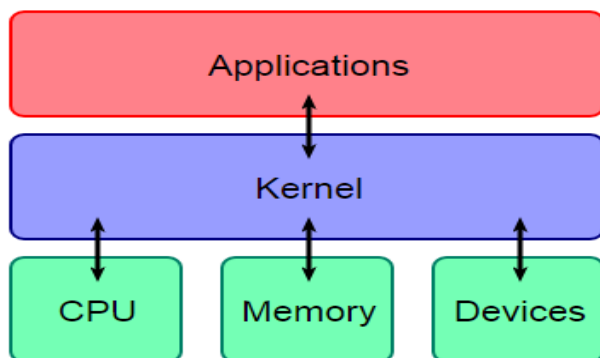
OS act as an interface between user and computer hardware.

Kernel :

It is the main core component in OS and it provides the most basic level of control over all of the computer's hardware devices. It manages memory access for programs in the RAM, it determines which programs get access to which hardware resources, it sets up or resets the CPU's operating states for optimal operation at all times, and it organizes the data for long-term non-volatile storage with file systems on such media as disks, tapes, flash memory, etc.

It has 5 services

- Memory management
- Device management
- Process management
- Interrupt handling
- Input/Output communication



User Interface or Shell :

Every computer that is to be operated by an individual requires a user interface. The user interface is usually referred to as a shell and is essential if human interaction is to be supported.

The user interface views the directory structure and requests services from the operating system that will acquire data from input hardware devices, such as a keyboard, mouse and requests operating system services to display prompts, status messages and such on output hardware devices, such as a video monitor or printer.

The two most common forms of a user interfaces are:

- Command Line Interface(CLI): where computer commands are typed out line-by-line.
- Graphical User Interface(GUI): where a visual environment is present. A mouse is used to navigate the computer.

UNIX :

UNICS(UNiplexed Information Computing System) is developed in 1969 by Ken Thompson at AT&T Bell labs, firstly developed in assembly language and then developed in C language by Dennis Ritchie.

Features of Unix OS :

- Unix is a portable, multi-user, multitasking operating system.
- It can be used as the master control program in workstations and servers.
- Hundreds of commercial applications are available.

Advantages :

- Portable, supports hierarchical file system
- Secure due to its strong server validation and authentication.

Disadvantages :

- CLI usage difficulty.
- Documentation of various Unix tools is hard to find.

Examples :

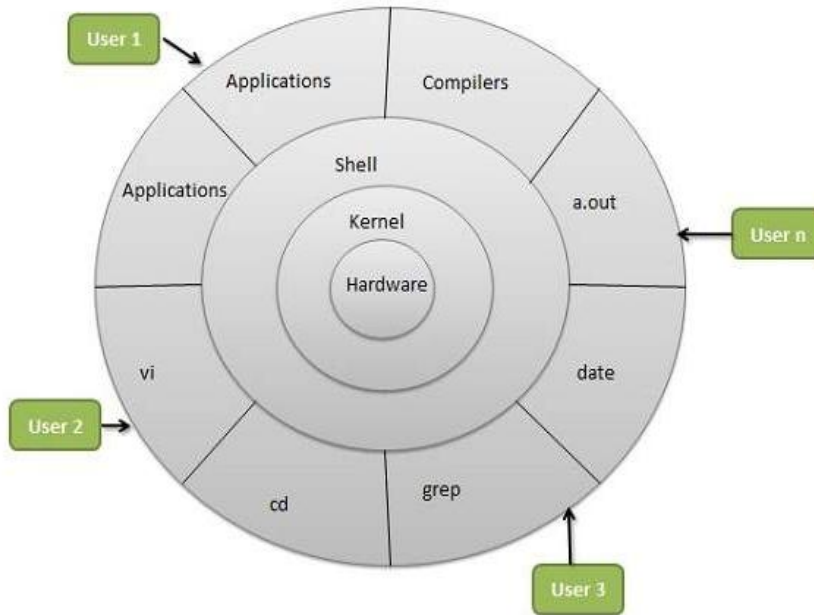
- Solaris
- AIX
- FreeBSD
- NetBSD
- Xenix

- MacOS

LINUX :

Loveable Intellect Not Using XP. It is developed by Linus Torvalds in 1991 at Linux community. It is Unix like OS developed by using C language.

Architecture of Linux :



Features of Linux OS :

- Portable, support multitasking.
- Programs consist of one or more processes, and each process has one or more threads.
- It can easily co-exist along with other Operating systems.
- Linux can run multiple user programs.
- Individual accounts are protected because of appropriate authorization.

Advantages :

- Compatible with large number of file formats.
- Open source software and is free of cost.
- User can make changes in Linux OS.
- Ensures privacy of user as it does not collect much user data.

Disadvantages :

- Use of CLI.
- Some graphic tools are not available.

- It doesnot have standard versions,so it is difficulty for users to choose the best version for their needs.

Examples :

- Fedora
- Ubuntu
- Slackware
- Debian.

Key differences between Unix and Linux :

- Linux source code is available to the general public whereas Unix source code is proprietary.
- Linux is a clone of Unix
- Linux default shell is BASH while the Unix shell is Bourne Shell.
- Linux threat detection and solution are very fast while Unix users require longer wait times to get the proper bug fixing patch.

Command :

A command is a directive to a computer program to perform a specific task. It may be issued via a command-line interface, such as a shell, or as input to a network service as part of a network protocol, or as an event in a graphical user interface triggered by the user selecting an option in a menu.

Shell Script :

A shell script is a computer program designed to be run by the Unix/Linux shell which could be one of the following:

- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

Use of LSS lab :

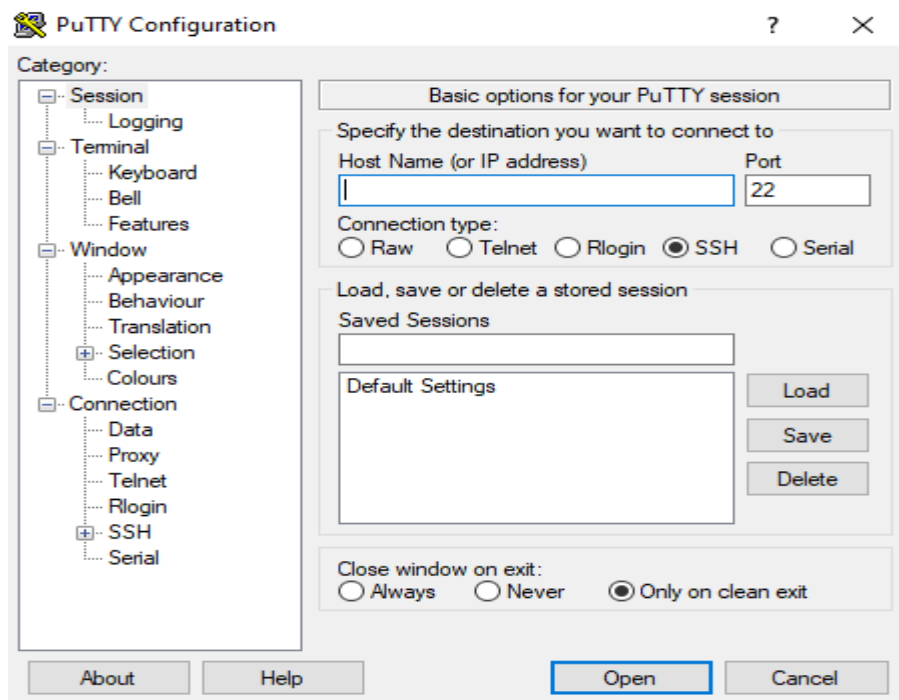
The Linux command is a utility of the Linux operating system. All basic and advanced tasks can be done by executing commands. The commands are executed on the Linux terminal. The terminal is a command-line interface to interact with the system, which is similar to the command prompt in the

Windows OS. Commands in Linux are case-sensitive. Many of the tasks that someone would like to perform on a computer are regular, require repetition, or are menial or tedious to do by hand. Shell scripting allows one to interact programmatically with a shell to do certain tasks.

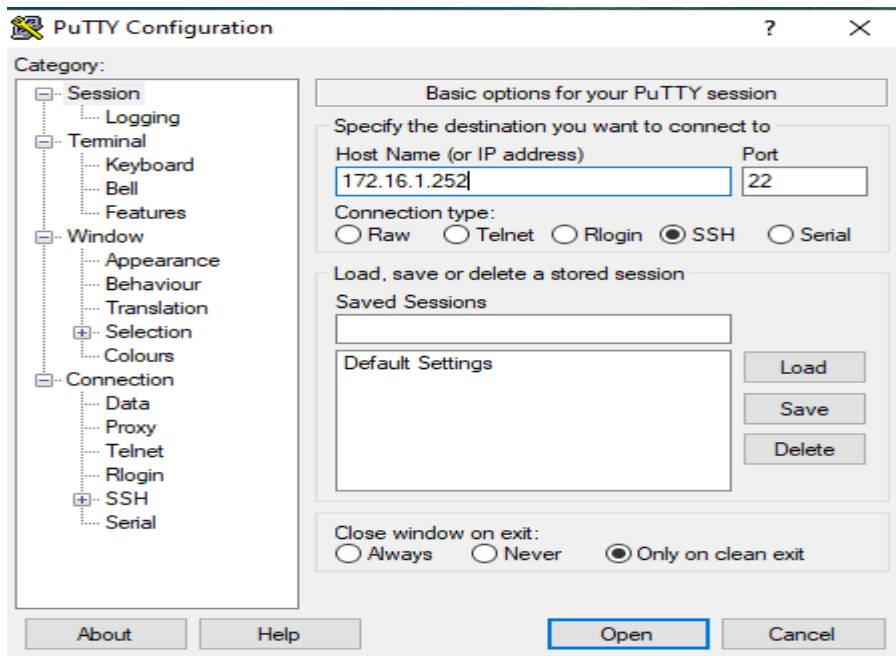
PuTTY allows the use of SSH (Secure Shell) to access a remote computer. It is a software terminal emulator that supports VT100 emulation, telnet, SSH, kerberos, and serial port connections.

Procedure to connect to LINUX:

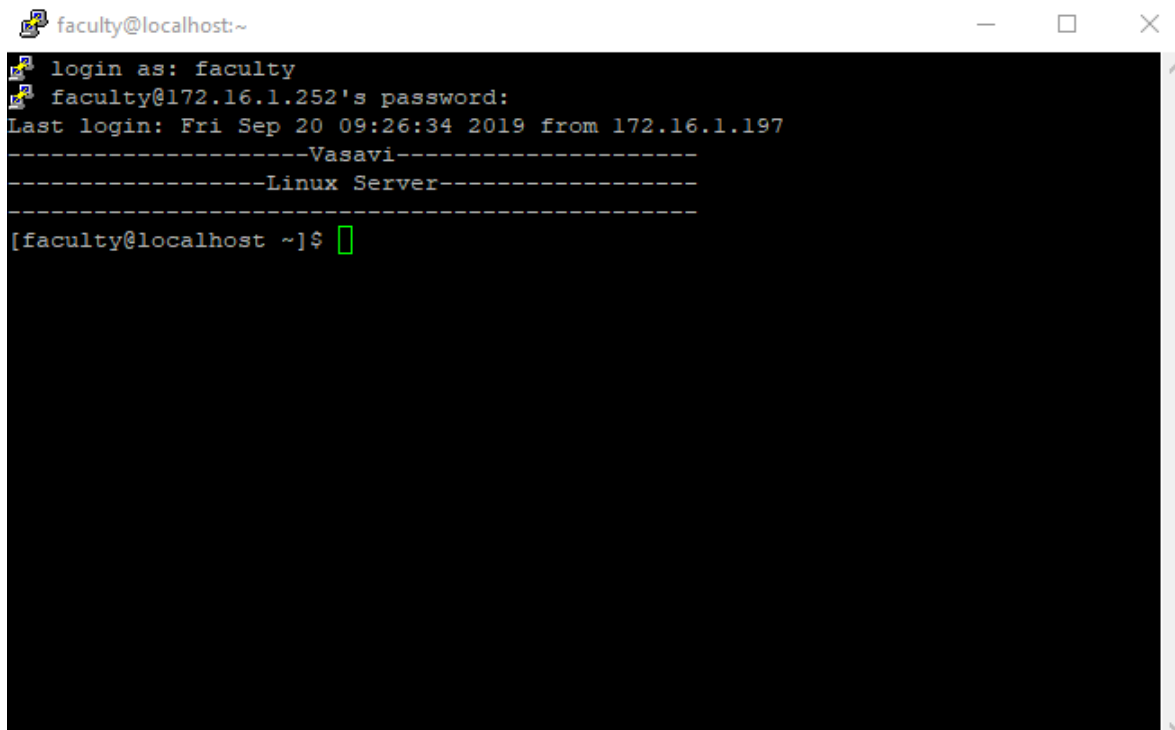
Step 1: Click on putty icon available on desk top. A window is opened



Step 2: Fill in IP address of linux server and click open.



Step 3: Provide login and password (nothing is displayed on screen while typing password) .



Step 4: Change the default password at your first login.(not for students).

Step 5: Logout from the system using either logout or exit command.

PROGRAM -1**Date:****Aim:** Experiment the following Unix Commands:

- a) General Purpose Utilities: cal, date, man, who.
- b) Directory Handling Commands: pwd, cd, mkdir, rmdir.
- c) File Handling Utilities: cat, cp, ls, rm, nl, wc
- d) Displaying Commands: head, tail
- e) Filters: cmp, comm., diff, sort, uniq
- f) Disk Utilities: du, df

a) General Purpose Utilities: cal, date, man, who.

cal(calendar) command: It displays calendar for specified month or a year. It has no options but uses arguments and the arguments are optional.

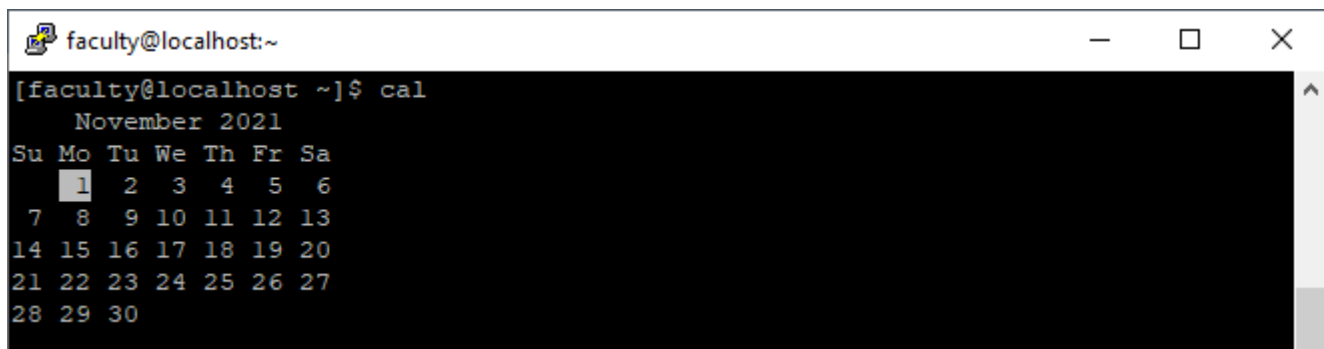
- a. if no arguments are entered, the calendar for the current month is printed

syntax:

\$ cal

Example:

\$ cal



```
faculty@localhost:~  
[faculty@localhost ~]$ cal  
    November 2021  
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
```

- b. if only one argument is entered, it is assumed to be a year and the calendar for that year is displayed.

Syntax:

\$ cal argument1

Example:

\$ cal 11

\$ cal 2021

```

faculty@localhost:~
[faculty@localhost ~]$ cal 11

      11

   January                February                March
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
   1  2  3                1  2  3  4  5  6  7       1  2  3  4  5  6  7
   4  5  6  7  8  9 10    8  9 10 11 12 13 14       8  9 10 11 12 13 14
  11 12 13 14 15 16 17    15 16 17 18 19 20 21      15 16 17 18 19 20 21
  18 19 20 21 22 23 24    22 23 24 25 26 27 28      22 23 24 25 26 27 28
  25 26 27 28 29 30 31    29 30 31

   April                  May                   June
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
   1  2  3  4                1  2                   1  2  3  4  5  6
   5  6  7  8  9 10 11    3  4  5  6  7  8  9       7  8  9 10 11 12 13
  12 13 14 15 16 17 18    10 11 12 13 14 15 16      14 15 16 17 18 19 20
  19 20 21 22 23 24 25    17 18 19 20 21 22 23      21 22 23 24 25 26 27
  26 27 28 29 30         24 25 26 27 28 29 30      28 29 30
                        31

   July                   August                September
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
   1  2  3  4                1                      1  2  3  4  5
   5  6  7  8  9 10 11    2  3  4  5  6  7  8       6  7  8  9 10 11 12
  12 13 14 15 16 17 18    9 10 11 12 13 14 15      13 14 15 16 17 18 19

```

- c. if both arguments are entered, then just one month is displayed

Syntax:

\$ cal argument1 argument2

example:

\$ cal 11 2021

\$ cal 01 21

\$ cal 11 2

```

faculty@localhost:~
[faculty@localhost ~]$ cal 11 2021

November 2021
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

```

date(date and time) command: It displays the system date and the time. If the system is local then it displays current system time. If the system is remote then it displays the time where the system is physically located. It can be used to set the date and time, but only by a system administrator.

Syntax:

\$ date options arguments

Options:

-u universal

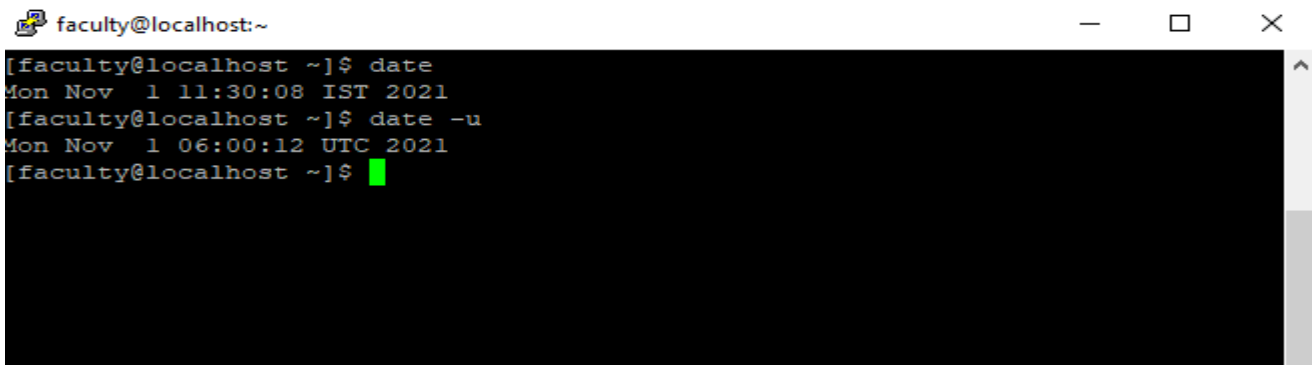
Arguments:

- a abbreviated weekday name
- A abbreviated full weekday name
- b abbreviated month name
- B abbreviated full month name
- d day of the month with two digits(leading zeros)
- e day of the month with spaces replacing leading zeros

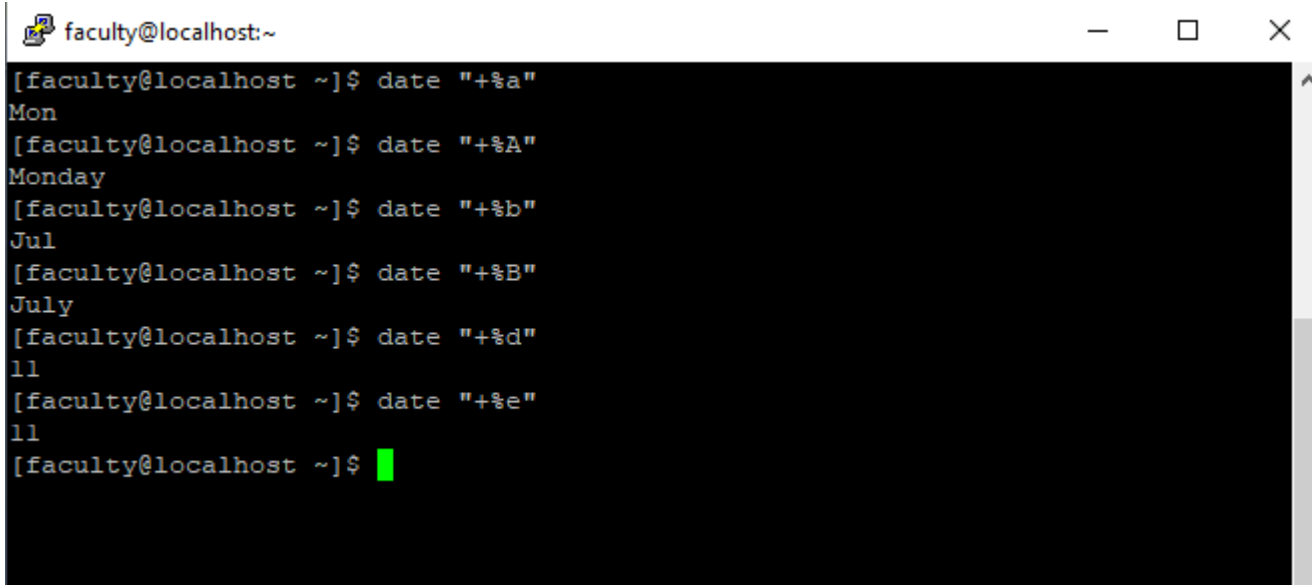
Example:

\$ date

\$ date -u



```
faculty@localhost:~  
[faculty@localhost ~]$ date  
Mon Nov  1 11:30:08 IST 2021  
[faculty@localhost ~]$ date -u  
Mon Nov  1 06:00:12 UTC 2021  
[faculty@localhost ~]$
```



```
faculty@localhost:~  
[faculty@localhost ~]$ date "+%a"  
Mon  
[faculty@localhost ~]$ date "+%A"  
Monday  
[faculty@localhost ~]$ date "+%b"  
Jul  
[faculty@localhost ~]$ date "+%B"  
July  
[faculty@localhost ~]$ date "+%d"  
11  
[faculty@localhost ~]$ date "+%e"  
11  
[faculty@localhost ~]$
```

man(online documentation) command:It is one of most important command and it displays online documentation of a command when you can't remember exactly what the options are for a command. There is even a manual explanation for the man command itself.

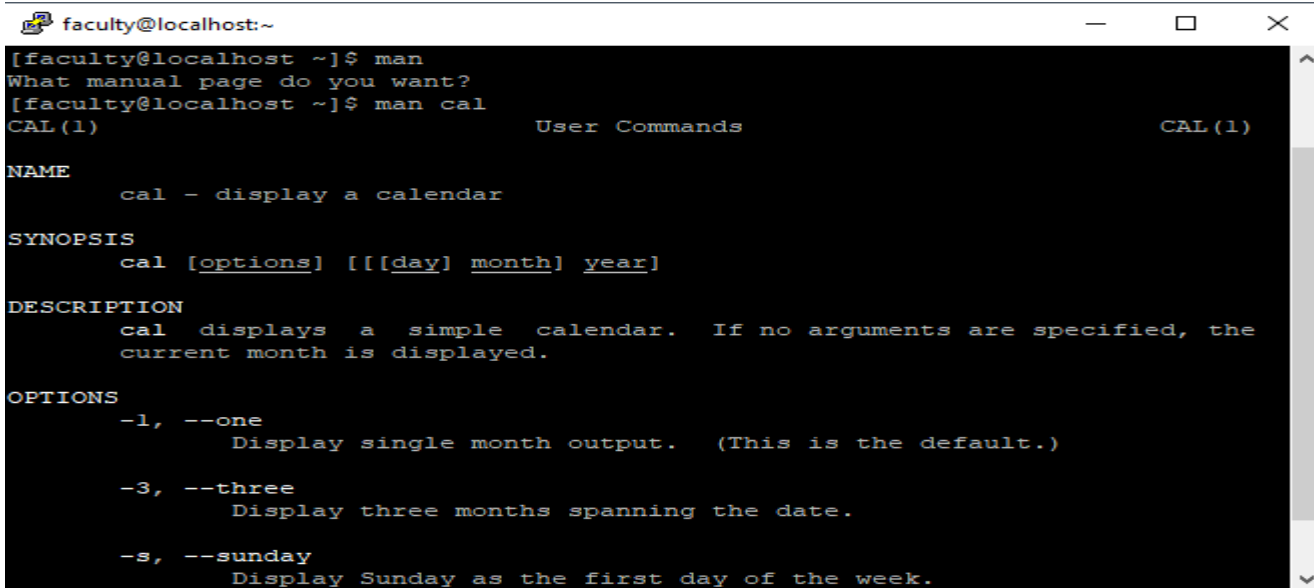
Syntax:

\$ man [any command]

Example:

\$ man cal

\$ man man



```
faculty@localhost:~  
[faculty@localhost ~]$ man  
What manual page do you want?  
[faculty@localhost ~]$ man cal  
CAL(1)                                User Commands                                CAL(1)  
  
NAME  
    cal - display a calendar  
  
SYNOPSIS  
    cal [options] [[[day] month] year]  
  
DESCRIPTION  
    cal displays a simple calendar.  If no arguments are specified, the  
    current month is displayed.  
  
OPTIONS  
    -1, --one  
        Display single month output.  (This is the default.)  
  
    -3, --three  
        Display three months spanning the date.  
  
    -s, --sunday  
        Display Sunday as the first day of the week.
```

who(who's online) command:It displays all users currently logged into the system. The who command returns the user's name(id), terminal and the time he/she logged in.

Syntax:

\$ who options

Options:

- u same as who command with idle times
- uH same as who -u with header

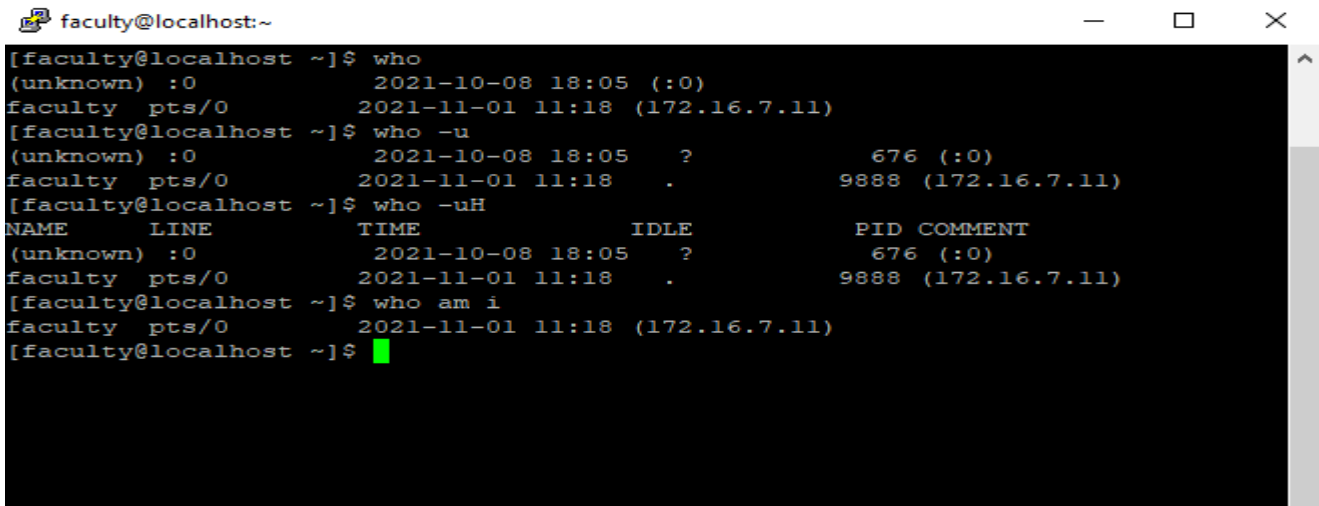
Example:

\$ who

\$ who -u

\$ who -uH

\$ who am i: It is used to ask about ourselves and it returns our user name(id).



```
faculty@localhost:~  
[faculty@localhost ~]$ who  
(unknown) :0          2021-10-08 18:05 (:0)  
faculty pts/0          2021-11-01 11:18 (172.16.7.11)  
[faculty@localhost ~]$ who -u  
(unknown) :0          2021-10-08 18:05 ?          676 (:0)  
faculty pts/0          2021-11-01 11:18 .          9888 (172.16.7.11)  
[faculty@localhost ~]$ who -uH  
NAME      LINE      TIME      IDLE      PID COMMENT  
(unknown) :0          2021-10-08 18:05 ?          676 (:0)  
faculty pts/0          2021-11-01 11:18 .          9888 (172.16.7.11)  
[faculty@localhost ~]$ who am i  
faculty pts/0          2021-11-01 11:18 (172.16.7.11)  
[faculty@localhost ~]$
```

b) Directory Handling Commands: pwd,cd,mkdir,rmdir.

pwd(locate directory / print working directory) command:It is used to determine the current directory is print working directory.It has no options and no attributes or arguments.When executed ,it prints the absolute path name for the current directory.

Syntax:

\$ pwd

Example:

\$ pwd

A terminal window titled 'faculty@localhost:~' with standard window controls. It shows the execution of the 'pwd' command, which returns '/home/faculty'. The prompt is '[faculty@localhost ~]\$' with a green cursor.

```
faculty@localhost:~  
[faculty@localhost ~]$ pwd  
/home/faculty  
[faculty@localhost ~]$
```

cd(change directory) command:cd command is used to change our working directory.If we have multiple directories,then we need some way to move among them.There are no options in cd command but have pathname

i.e either relative or absolute path.Generally it is relative.If there is no pathname argument,the target is the home directory.the home directory can also be targeted by using the home abbreviation(cd ~) or cd.

Syntax:

\$ cd pathname

Example:

\$ cd LSS

A terminal window titled 'faculty@localhost:~/LSS' with standard window controls. It shows the execution of the 'cd LSS' command, which changes the directory to 'LSS'. The prompt is '[faculty@localhost LSS]\$' with a green cursor.

```
faculty@localhost:~/LSS  
[faculty@localhost ~]$ cd LSS  
[faculty@localhost LSS]$
```

mkdir(make directory) command:It is used to make or create a new directory.

Syntax:

\$ mkdir options directoryname

Options:

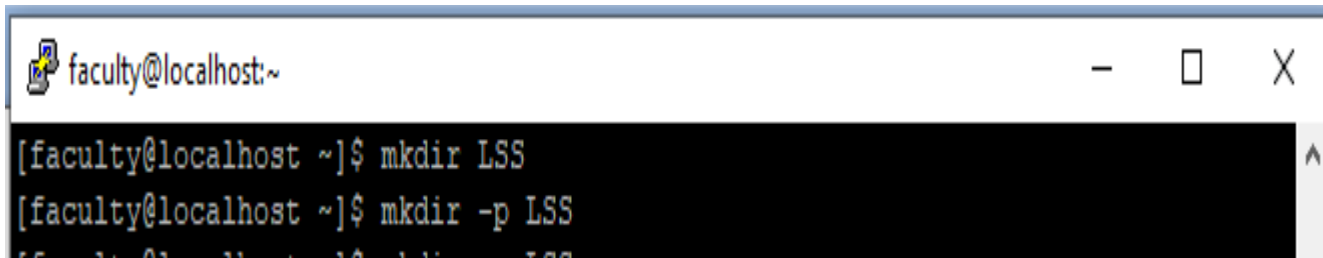
- m permission mode
- p parent directories

Example:

\$ mkdir LSS

\$ mkdir -m LSS

\$ mkdir -p LSS



```
faculty@localhost:~  
[faculty@localhost ~]$ mkdir LSS  
[faculty@localhost ~]$ mkdir -p LSS
```

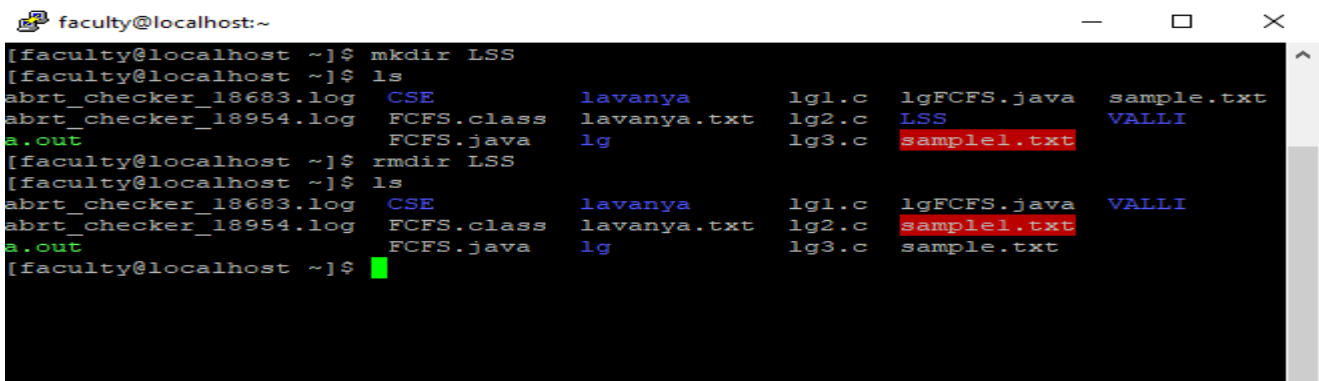
rmdir(remove directory) command:It is used to remove a directory which is no longer required.It cannot delete a directory unless it is empty.If it contains any files,then “directory not empty” error message is displayed.It has no options.

Syntax:

\$ rmdir directoryname

Example:

\$ rmdir LSS



```
faculty@localhost:~  
[faculty@localhost ~]$ mkdir LSS  
[faculty@localhost ~]$ ls  
abrt_checker_18683.log  CSE          lavanya      lg1.c      lgFCFS.java  sample.txt  
abrt_checker_18954.log  FCFS.class   lavanya.txt  lg2.c      LSS          VALLI  
a.out                 FCFS.java    lg          lg3.c      sample1.txt  
[faculty@localhost ~]$ rmdir LSS  
[faculty@localhost ~]$ ls  
abrt_checker_18683.log  CSE          lavanya      lg1.c      lgFCFS.java  VALLI  
abrt_checker_18954.log  FCFS.class   lavanya.txt  lg2.c      sample1.txt  
a.out                 FCFS.java    lg          lg3.c      sample.txt  
[faculty@localhost ~]$
```


c) File Handling Utilities: cat,cp,ls,rm,nl,wc

cat(catenate) command: cat linux command concatenates files and print it on the standard output.

syntax: cat options inputfiles

options:

- A Show all.
- b Omits line numbers for blank space in the output.
- e A \$ character will be printed at the end of each line prior to a new line.
- E Displays a \$ (dollar sign) at the end of each line.
- n Line numbers for all the output lines.
- s If the output has multiple empty lines it replaces it with one empty line
- T Displays the tab characters in the output.
- v Non-printing characters (with the exception of tabs, new-lines & form-feeds) are printed

visibly.

Operations With cat Command:

1. To Create a new file: \$cat > file1.txt

This command creates a new file file1.txt. After typing into the file press control+d (^d) simultaneously to end the file. Make sure you type '**Ctrl-d**' at the end to save the file.

2. To Append data into the file: \$cat >> file1.txt

To append data into the same file use append operator >> to write into the file, else the file will be overwritten (i.e., all of its contents will be erased). Make sure you type '**Ctrl-d**' at the end to save the file.

3. To display a file: \$cat file1.txt

This command displays the data in the file.

4. To concatenate several files and display: \$cat file1.txt file2.txt

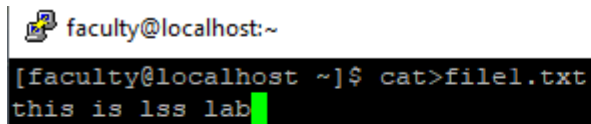
The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you can print those files in a new file or display the file using less command. cat file1.txt file2.txt | less

5. To concatenate several files and to transfer the output to another file. \$cat file1.txt file2.txt > file3.txt

In the above example the output is redirected to new file file3.txt. The cat command will create new file file3.txt and store the concatenated output into file3.txt

Example:

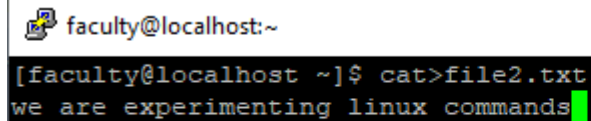
```
$ cat > file1.txt
```



```
faculty@localhost:~  
[faculty@localhost ~]$ cat>file1.txt  
this is lss lab
```

A terminal window titled 'faculty@localhost:~' with standard window controls. The prompt is '[faculty@localhost ~]\$'. The command 'cat>file1.txt' has been entered, and the text 'this is lss lab' is being typed on the next line. A green cursor is at the end of the text.

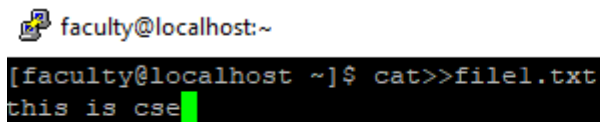
\$ cat > file2.txt



```
faculty@localhost:~  
[faculty@localhost ~]$ cat>file2.txt  
we are experimenting linux commands
```

A terminal window titled 'faculty@localhost:~' with standard window controls. The prompt is '[faculty@localhost ~]\$'. The command 'cat>file2.txt' has been entered, and the text 'we are experimenting linux commands' is being typed on the next line. A green cursor is at the end of the text.

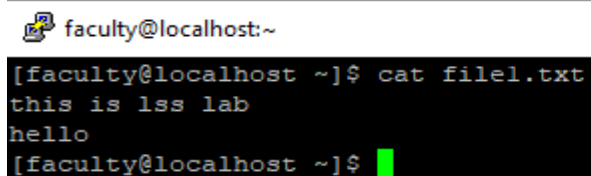
\$ cat >> file1.txt



```
faculty@localhost:~  
[faculty@localhost ~]$ cat>>file1.txt  
this is cse
```

A terminal window titled 'faculty@localhost:~' with standard window controls. The prompt is '[faculty@localhost ~]\$'. The command 'cat>>file1.txt' has been entered, and the text 'this is cse' is being typed on the next line. A green cursor is at the end of the text.

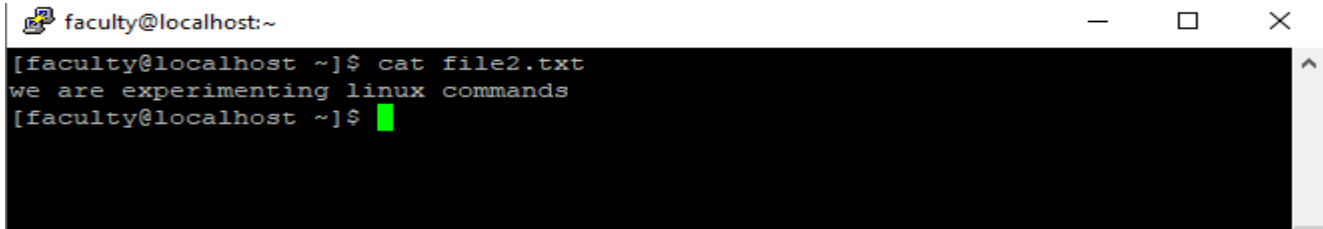
\$ cat file1.txt



```
faculty@localhost:~  
[faculty@localhost ~]$ cat file1.txt  
this is lss lab  
hello  
[faculty@localhost ~]$
```

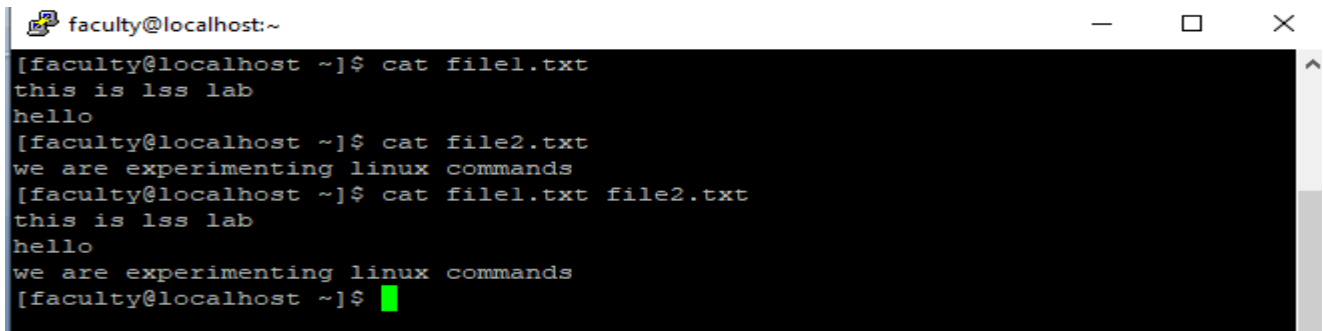
A terminal window titled 'faculty@localhost:~' with standard window controls. The prompt is '[faculty@localhost ~]\$'. The command 'cat file1.txt' has been entered, and the output 'this is lss lab' and 'hello' is displayed on the next two lines. The prompt '[faculty@localhost ~]\$' is shown on the third line with a green cursor.

\$ cat file2.txt



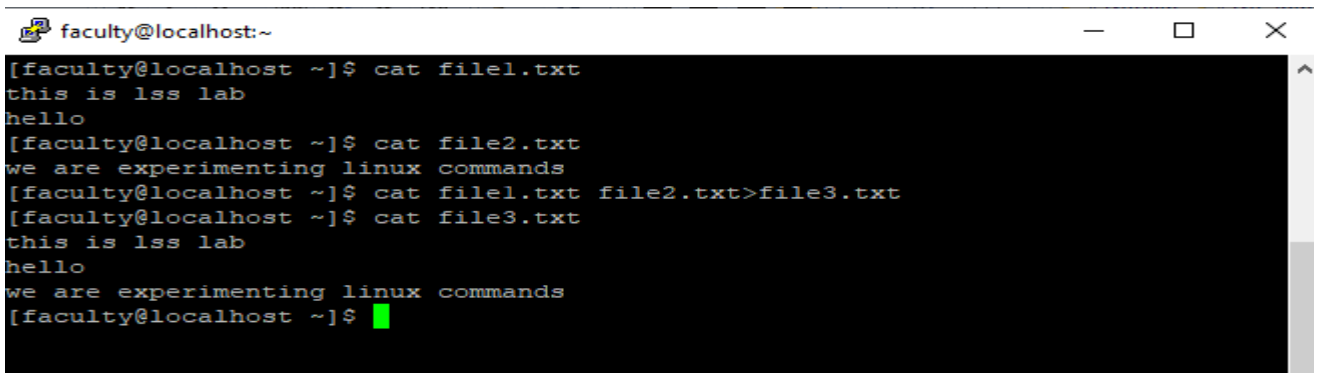
```
faculty@localhost:~  
[faculty@localhost ~]$ cat file2.txt  
we are experimenting linux commands  
[faculty@localhost ~]$
```

\$ cat file1.txt file2.txt



```
faculty@localhost:~  
[faculty@localhost ~]$ cat file1.txt  
this is lss lab  
hello  
[faculty@localhost ~]$ cat file2.txt  
we are experimenting linux commands  
[faculty@localhost ~]$ cat file1.txt file2.txt  
this is lss lab  
hello  
we are experimenting linux commands  
[faculty@localhost ~]$
```

\$ cat file1.txt file2.txt>file3.txt



```
faculty@localhost:~  
[faculty@localhost ~]$ cat file1.txt  
this is lss lab  
hello  
[faculty@localhost ~]$ cat file2.txt  
we are experimenting linux commands  
[faculty@localhost ~]$ cat file1.txt file2.txt > file3.txt  
[faculty@localhost ~]$ cat file3.txt  
this is lss lab  
hello  
we are experimenting linux commands  
[faculty@localhost ~]$
```

cp(copy) command:It creates a duplicate of a file ,a set of files or a directory.The cp command copies both text and binary files.If the source is a file,the new file contains an exact copy of the data in the source file.If the source is a directory,all of the files in the directory are copied to the destination,which must be a directory.If the destination file exists,its contents are replaced by the source file contents.

Syntax:

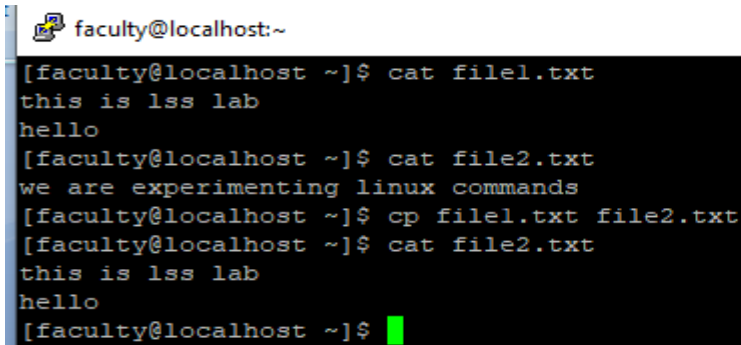
\$ cp options source destination

Options:

- p preserve time and permissions
- i interactive prompt
- r recursive copy subdirectories

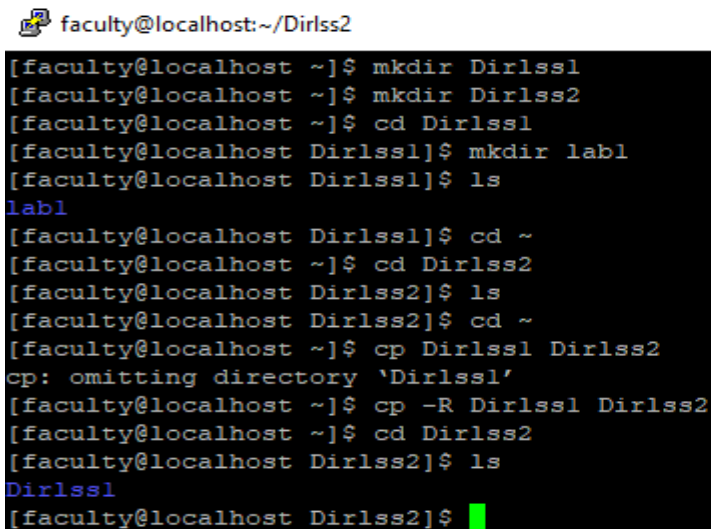
Example:

```
$ cp file1.txt file2.txt
```

A terminal window titled 'faculty@localhost:~' with standard window controls. It shows a sequence of commands: 'cat file1.txt' outputs 'this is lss lab' and 'hello'; 'cat file2.txt' outputs 'we are experimenting linux commands'; 'cp file1.txt file2.txt' copies the file; and a second 'cat file2.txt' shows the original content of file1.txt, 'this is lss lab' and 'hello'.

```
[faculty@localhost ~]$ cat file1.txt
this is lss lab
hello
[faculty@localhost ~]$ cat file2.txt
we are experimenting linux commands
[faculty@localhost ~]$ cp file1.txt file2.txt
[faculty@localhost ~]$ cat file2.txt
this is lss lab
hello
[faculty@localhost ~]$
```

```
$ cp Dirlss1 Dirlss2
```

A terminal window titled 'faculty@localhost:~/Dirlss2' with standard window controls. It shows a sequence of commands: 'mkdir Dirlss1' and 'mkdir Dirlss2' create directories; 'cd Dirlss1' changes to the first directory; 'mkdir lab1' creates a subdirectory; 'ls' lists 'lab1'; 'cd ~' returns to the home directory; 'cd Dirlss2' changes to the second directory; 'ls' lists the contents; 'cd ~' returns to the home directory; 'cp Dirlss1 Dirlss2' fails with an error about omitting the directory; 'cp -R Dirlss1 Dirlss2' successfully copies the directory; 'cd Dirlss2' changes to the destination directory; and 'ls' lists the copied 'Dirlss1' directory.

```
[faculty@localhost ~]$ mkdir Dirlss1
[faculty@localhost ~]$ mkdir Dirlss2
[faculty@localhost ~]$ cd Dirlss1
[faculty@localhost Dirlss1]$ mkdir lab1
[faculty@localhost Dirlss1]$ ls
lab1
[faculty@localhost Dirlss1]$ cd ~
[faculty@localhost ~]$ cd Dirlss2
[faculty@localhost Dirlss2]$ ls
[faculty@localhost Dirlss2]$ cd ~
[faculty@localhost ~]$ cp Dirlss1 Dirlss2
cp: omitting directory 'Dirlss1'
[faculty@localhost ~]$ cp -R Dirlss1 Dirlss2
[faculty@localhost ~]$ cd Dirlss2
[faculty@localhost Dirlss2]$ ls
Dirlss1
[faculty@localhost Dirlss2]$
```

```
$ cp -p file.txt file2.txt
```

```
$ cp -i file1.txt file2.txt
```

ls(list) command: ls command is used to list the files or contents present in a directory. Depending on the options it can list files, directories, subdirectories. If the name of the directory is provided then it displays the contents of the working directory.

Syntax:

```
$ ls
```

Options:

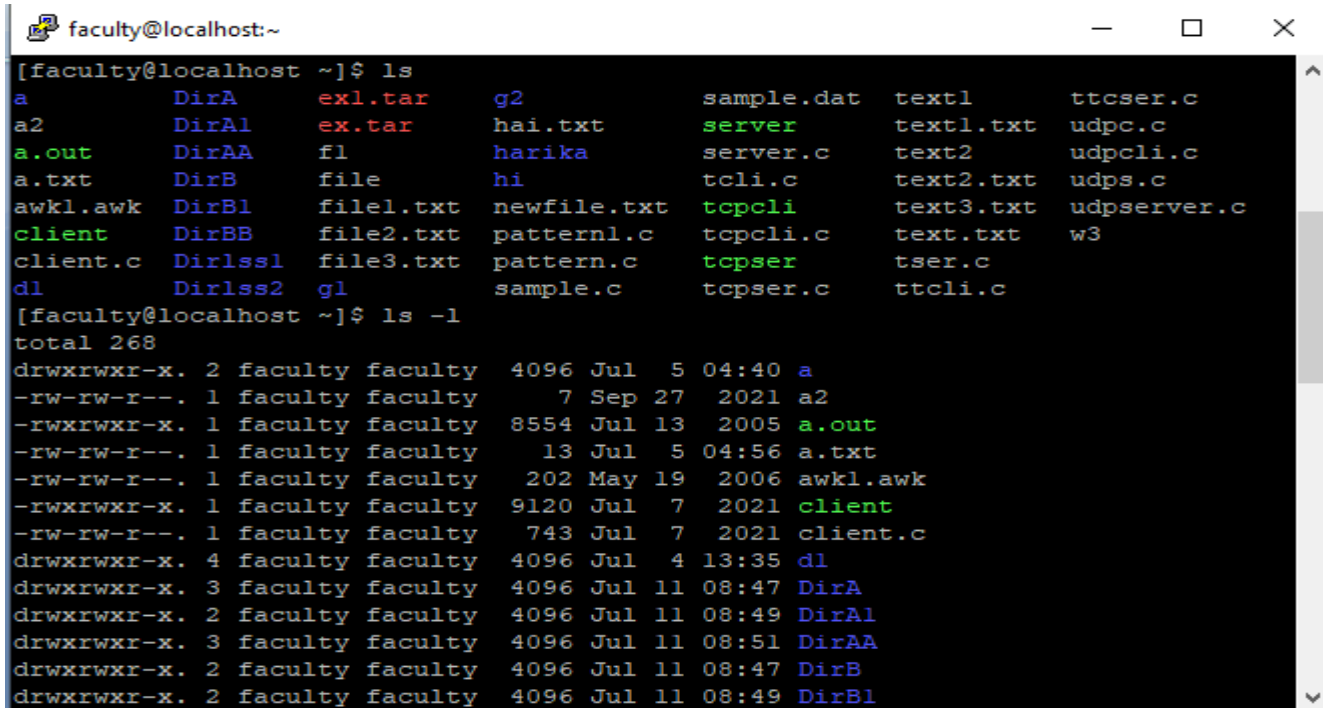
-l longlist

- d working directory
- n user/group id's
- r reverse order
- t time sequence
- p identify directories
- l(One) print one column
- i print inode
- c time inode date
- R recursive

Example:

```
$ ls
```

```
$ ls -l
```



```

[faculty@localhost ~]$ ls
a      DirA    ex1.tar  g2      sample.dat  text1      ttcser.c
a2     DirA1   ex.tar   hai.txt  server      text1.txt  udpc.c
a.out  DirAA   fl       harika   server.c    text2      udpcli.c
a.txt  DirB    file     hi       tcpcli.c    text2.txt  udps.c
awk1.awk DirB1   file1.txt newfile.txt tcpcli      text3.txt  udpserver.c
client DirBB   file2.txt pattern1.c tcpcli.c    text.txt   w3
client.c Dir1ss1 file3.txt pattern.c  tcpser      tser.c
dl     Dir1ss2 gl       sample.c  tcpser.c    ttcli.c

[faculty@localhost ~]$ ls -l
total 268
drwxrwxr-x. 2 faculty faculty 4096 Jul  5 04:40 a
-rw-rw-r--. 1 faculty faculty    7 Sep 27 2021 a2
-rwxrwxr-x. 1 faculty faculty 8554 Jul 13 2005 a.out
-rw-rw-r--. 1 faculty faculty   13 Jul  5 04:56 a.txt
-rw-rw-r--. 1 faculty faculty  202 May 19 2006 awk1.awk
-rwxrwxr-x. 1 faculty faculty 9120 Jul  7 2021 client
-rw-rw-r--. 1 faculty faculty  743 Jul  7 2021 client.c
drwxrwxr-x. 4 faculty faculty 4096 Jul  4 13:35 dl
drwxrwxr-x. 3 faculty faculty 4096 Jul 11 08:47 DirA
drwxrwxr-x. 2 faculty faculty 4096 Jul 11 08:49 DirA1
drwxrwxr-x. 3 faculty faculty 4096 Jul 11 08:51 DirAA
drwxrwxr-x. 2 faculty faculty 4096 Jul 11 08:47 DirB
drwxrwxr-x. 2 faculty faculty 4096 Jul 11 08:49 DirB1

```

rm(remove) command: It deletes an entry from a directory by destroying its link to the file. Remember, however, that there can be multiple links to a physical file. This means that a remove does not always physically delete a file. The file is deleted only if, after remove, there are no more links to it. To delete a file, we must have write permissions.

Syntax:

```
$ rm options filelist
```

Options:

- f force
- i interactive
- r recursive

Example:

```
$ rm file1
```

```
$ rm -r file2
```

```
[faculty@localhost ~]$ rm file1.txt
[faculty@localhost ~]$ ls
a          DirA      ex1.tar   hai.txt   server    text1.txt  udpc.c
a2         DirA1     ex.tar    harika    server.c  text2      udpcli.c
a.out      DirAA     fl        hi        tcli.c    text2.txt  udps.c
a.txt      DirB      file      newfile.txt tcpcli    text3.txt  udpserver.c
awk1.awk   DirB1     file2.txt pattern1.c tcpcli.c   text.txt   w3
client     DirBB     file3.txt pattern.c  tcpser    tser.c
client.c   Dir1ss1   g1        sample.c  tcpser.c  ttcli.c
dl         Dir1ss2   g2        sample.dat text1      ttcser.c
[faculty@localhost ~]$
```

nl(number line) command: The nl utility in Linux is used to give number lines of a file on console.

Syntax:

```
$ nl filename
```

Example: \$ nl file2.txt

```
faculty@localhost:~
[faculty@localhost ~]$ nl file2.txt
 1  this is lss lab
 2  hello
[faculty@localhost ~]$
```

wc(word count) command: It counts the number of characters, words and lines in one or more documents. The character count includes newlines(/n). Options can be used to limit the output to one or two of the counts.

Syntax:

```
wc options inputfiles
```

options:

- c character count
- l line count
- w word count

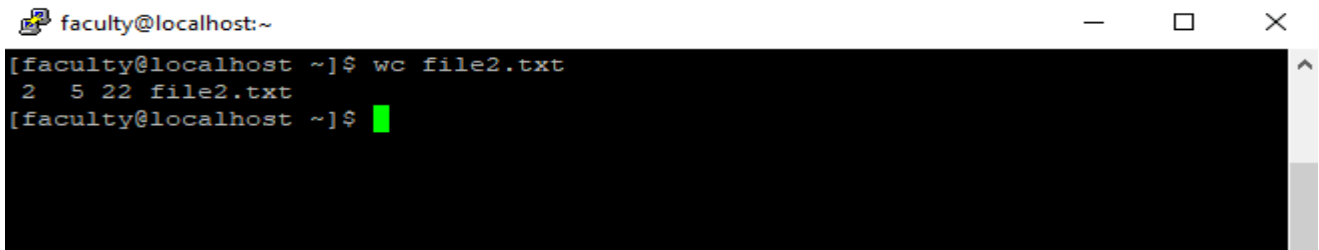
Example:

```
$ wc file2.txt
```

```
$ wc -c file2.txt
```

```
$ wc -l file2.txt
```

```
$ wc -w file2.txt
```

A terminal window titled 'faculty@localhost:~' with standard window controls. The terminal shows the command '[faculty@localhost ~]\$ wc file2.txt' and its output '2 5 22 file2.txt'. The prompt '[faculty@localhost ~]\$' is followed by a green cursor.

```
faculty@localhost:~  
[faculty@localhost ~]$ wc file2.txt  
2 5 22 file2.txt  
[faculty@localhost ~]$
```

d) Displaying Commands: head, tail

head command: It copies a specified number of lines from the beginning of one or more files to the standard output stream or to displays portions of files. If no files are specified, it gets the lines from standard input.

Syntax:

```
head options inputfiles
```

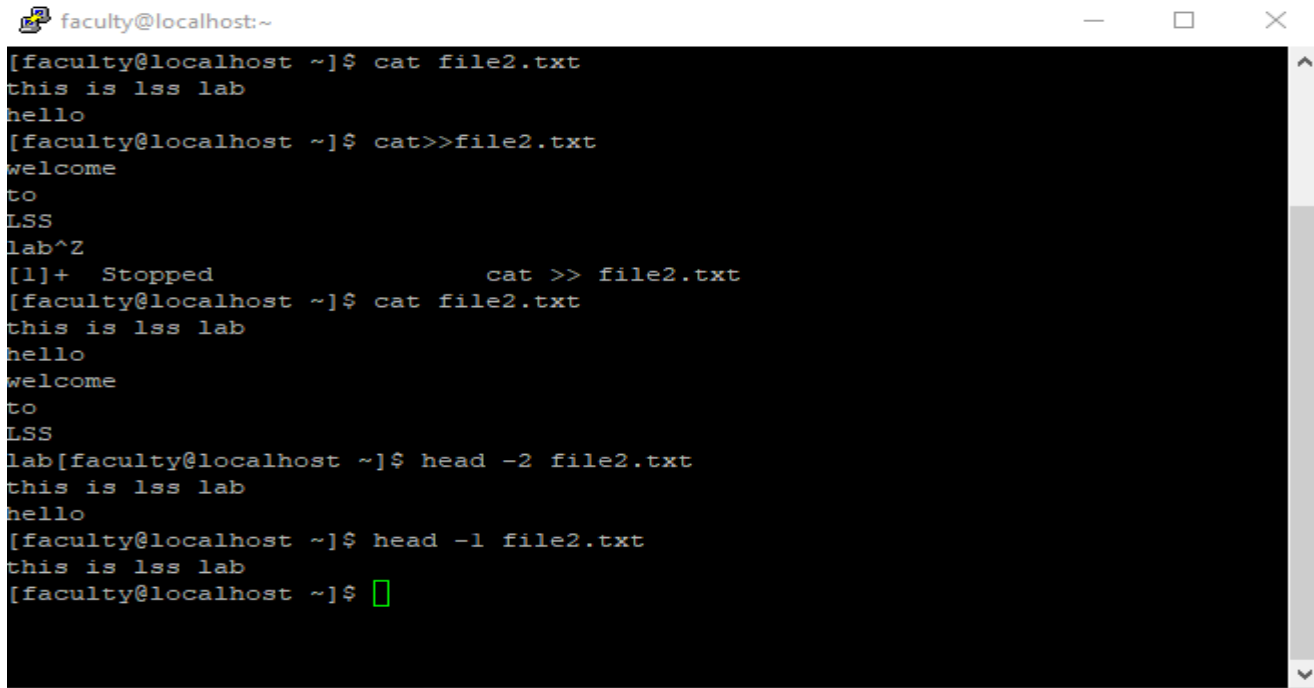
options:

- N number of lines

Example:

```
$ head -2 file2.txt
```

```
$ head -1 file2.txt
```

A terminal window titled 'faculty@localhost:~' with standard window controls. It shows a series of commands and their outputs. The 'cat' command is used to view and append to 'file2.txt'. The 'head' command is used to view the first two and then the last one line of the file. The prompt is '[faculty@localhost ~]\$' and the cursor is a green block.

```
[faculty@localhost ~]$ cat file2.txt
this is lss lab
hello
[faculty@localhost ~]$ cat>>file2.txt
welcome
to
LSS
lab^2
[!]+ Stopped          cat >> file2.txt
[faculty@localhost ~]$ cat file2.txt
this is lss lab
hello
welcome
to
LSS
lab[faculty@localhost ~]$ head -2 file2.txt
this is lss lab
hello
[faculty@localhost ~]$ head -1 file2.txt
this is lss lab
[faculty@localhost ~]$
```

tail command:It copies a specified number of lines from the ending of one or more files to the standard output stream or to displays portions of files.If no files are specified,it gets the lines from standard input.

Syntax:

tail options inputfiles

options:

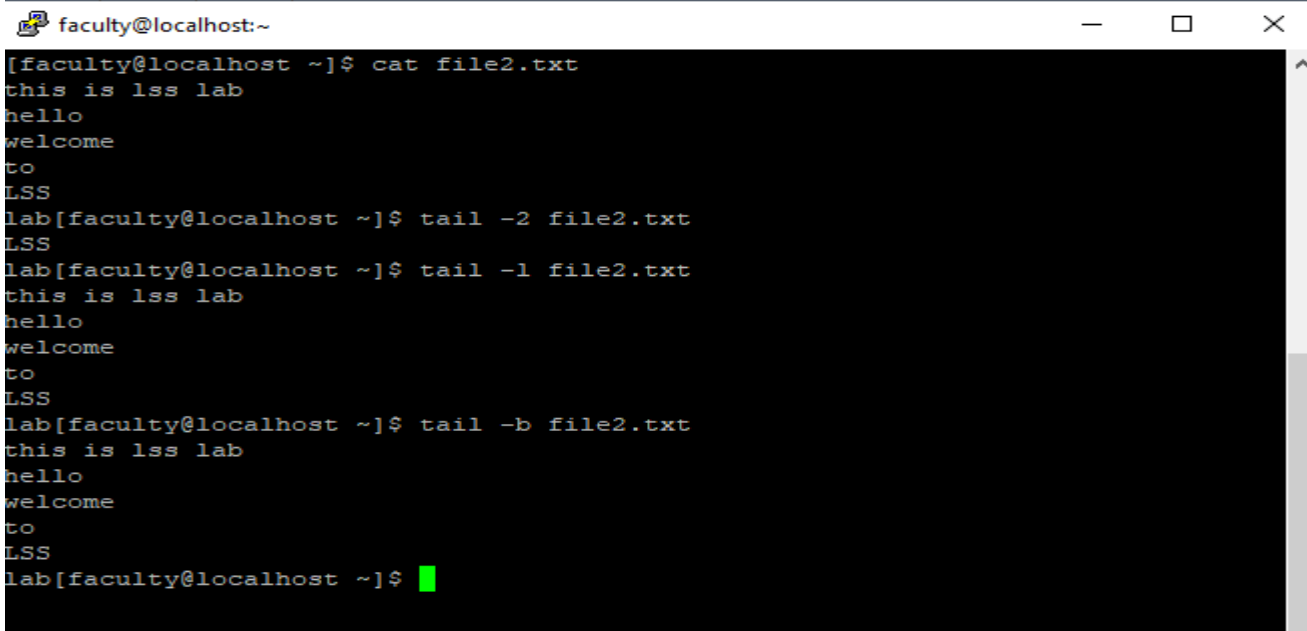
- +N skip N-1 lines
- N N lines from end
- l measured by lines(default)
- c measured by characters
- b measured by blocks
- r output in reverse order

Example:

\$ tail -2 file2.txt

\$ tail -l file2.txt

\$ tail -b file2.txt

A terminal window titled 'faculty@localhost:~' with standard window controls. It shows a sequence of commands and their outputs. The 'cat' command displays the contents of 'file2.txt'. The 'tail' command is used twice: first with '-2' to show the last two lines, and then with '-1' to show the last line. Finally, 'tail -b' is used to show the last byte of the file. The prompt changes from '[faculty@localhost ~]' to 'lab[faculty@localhost ~]' after the first 'tail' command.

```
faculty@localhost:~  
[faculty@localhost ~]$ cat file2.txt  
this is lss lab  
hello  
welcome  
to  
LSS  
lab[faculty@localhost ~]$ tail -2 file2.txt  
LSS  
lab[faculty@localhost ~]$ tail -1 file2.txt  
this is lss lab  
hello  
welcome  
to  
LSS  
lab[faculty@localhost ~]$ tail -b file2.txt  
this is lss lab  
hello  
welcome  
to  
LSS  
lab[faculty@localhost ~]$
```

e) Filters: cmp,comm.,diff,sort,uniq

cmp(compare) command:It examines two files byte by byte.the action it take depends on thye options used.

Syntax:

cmp options inputfiles

options:

-l list

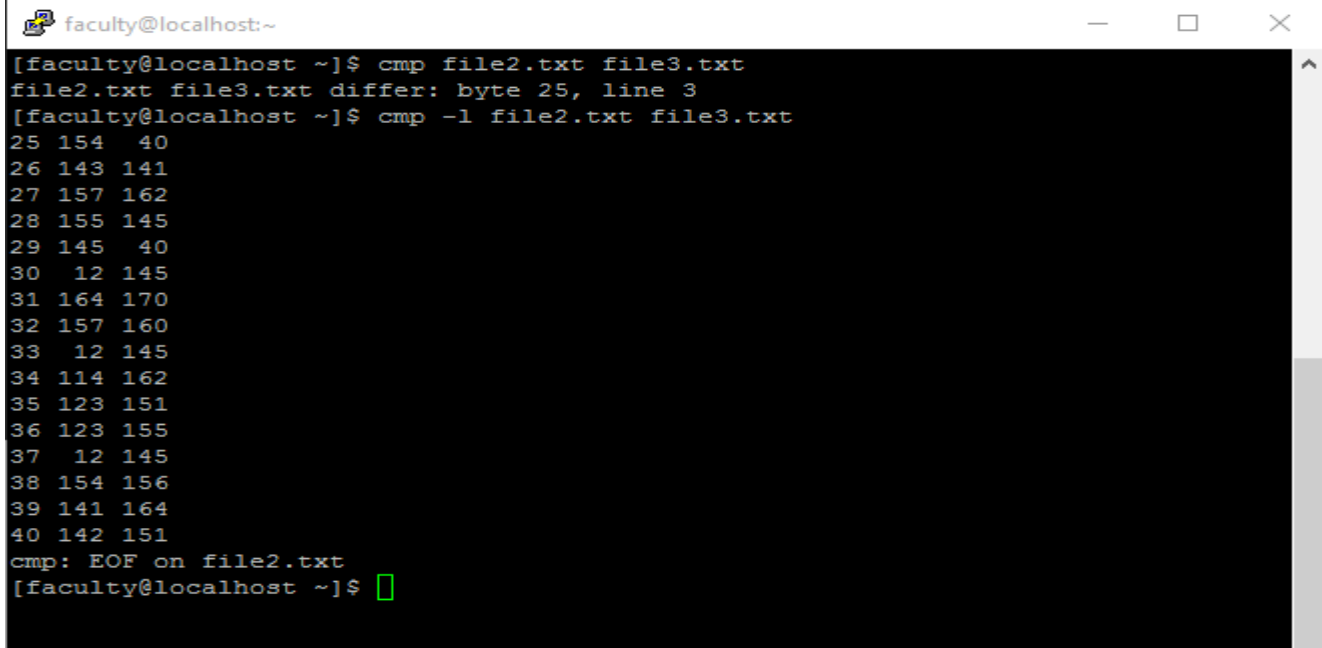
-s suppress list

Example:

\$ cmp file2.txt file3.txt

\$ cmp -l file2.txt file3.txt

\$ cmp -s file2.txt file3.txt



```
faculty@localhost:~  
[faculty@localhost ~]$ cmp file2.txt file3.txt  
file2.txt file3.txt differ: byte 25, line 3  
[faculty@localhost ~]$ cmp -l file2.txt file3.txt  
25 154 40  
26 143 141  
27 157 162  
28 155 145  
29 145 40  
30 12 145  
31 164 170  
32 157 160  
33 12 145  
34 114 162  
35 123 151  
36 123 155  
37 12 145  
38 154 156  
39 141 164  
40 142 151  
cmp: EOF on file2.txt  
[faculty@localhost ~]$
```

comm(common) command: It finds lines that are identical in two files. It compares the files line by line and displays the result in three columns. The left column contains unique lines in file1, the center column contains unique lines in file2, the right column contains lines found in both files.

Syntax:

comm inputfiles

Example:

comm file4.txt file5.txt

```
faculty@localhost:~  
[faculty@localhost ~]$ cat>file4.txt  
LSS Lab  
welcome  
[faculty@localhost ~]$ cat file4.txt  
LSS Lab  
welcome  
[faculty@localhost ~]$ cat>file5.txt  
hello  
LSS Lab  
[faculty@localhost ~]$ cat file5.txt  
hello  
LSS Lab  
[faculty@localhost ~]$ comm file4.txt file5.txt  
      hello  
LSS Lab  
welcome  
[faculty@localhost ~]$
```

diff(difference) command:It shows line by line differences between two files.The first file is compared to the second file.The differences are identified such that the first file could be modified to make it match the second file.The diff command always works on files.

Syntax:

diff options [files or directories]

options:

- b ignore trailing blanks
- w ignore whitespace
- i ignore case

Example:

\$ diff file4.txt file5.txt

\$ diff Dirlss1 Dirlss2

```
faculty@localhost:~  
[faculty@localhost ~]$ diff file4.txt file5.txt  
0a1  
> hello  
2d2  
< welcome  
[faculty@localhost ~]$ diff Dirlss1 Dirlss2  
Only in Dirlss2: Dirlss1  
Only in Dirlss1: lab1  
[faculty@localhost ~]$
```

sort(sorting) command:It sorts the data lines in a text file using standard sorting rules.It interprets numbers as characters and performs a standard character sort,producing output that might not be what you want.By using `-n` parameter the sort command recognize numbers as numbers instead of characters,and sort them based on their numerical values.

Syntax:

`sort options inputfiles`

options:

- `-n` recognizes numbers as numbers instead of characters.
- `-b` ignore leading blocks
- `-C` don't sort,but don't report if data is out of sort order
- `-c` don't sort,but check if the input data is already sorted.report if not sorted
- `-d` dictionary order
- `-f` ignore case
- `-g` general numeric sort
- `-i` ignore nonprinting
- `-M` month sort
- `-o` output file
- `-r` reverse
- `-u` unique
- `-z` zero terminated
- `-m` merge two already sorted data files

Example:

```
$ sort file9.txt
```

```
faculty@localhost:~  
[faculty@localhost ~]$ cat>file9.txt  
hai  
hello  
welcome  
to  
lss  
lab  
[faculty@localhost ~]$ cat file9.txt  
hai  
hello  
welcome  
to  
lss  
lab  
[faculty@localhost ~]$ sort file9.txt  
hai  
hello  
lab  
lss  
to  
welcome  
[faculty@localhost ~]$
```

\$ sort -n file8.txt

```
faculty@localhost:~  
4 lab  
[faculty@localhost ~]$ rm file7.txt  
[faculty@localhost ~]$ rm file8.txt  
[faculty@localhost ~]$ cat>file8.txt  
50  
40  
10  
100  
[faculty@localhost ~]$ cat file8.txt  
50  
40  
10  
100  
[faculty@localhost ~]$ sort file8.txt  
10  
100  
40  
50  
[faculty@localhost ~]$ sort -n file8.txt  
10  
40  
50  
100  
[faculty@localhost ~]$
```

uniq(unique) command:It deletes duplicate lines,keeping the first and deleting the others.To be deleted,the lines must be adjacent.Duplicate lines that are not adjacent are deleted when the file is in sorted order.

Syntax:

uniq options inputfile

options:

- u only nonrepeated lines output
- d only repeated lines output
- c add count to lines
- f field
- s character

Example:

\$ uniq file6.txt

```
faculty@localhost:~  
[faculty@localhost ~]$ cat file6.txt  
hai  
hai  
this is lss lab  
this is lss lab  
[faculty@localhost ~]$ cat file7.txt  
hai  
this is lss lab  
hai  
this is lss lab  
[faculty@localhost ~]$ uniq file6.txt  
hai  
this is lss lab  
[faculty@localhost ~]$ uniq file7.txt  
hai  
this is lss lab  
hai  
this is lss lab  
[faculty@localhost ~]$
```

\$ uniq -u file6.txt

\$ uniq -d file6.txt

```
faculty@localhost:~  
[faculty@localhost ~]$ cat file6.txt  
hai  
hai  
this is lss lab  
this is lss lab  
welcome  
[faculty@localhost ~]$ uniq -u file6.txt  
welcome  
[faculty@localhost ~]$ uniq -d file6.txt  
hai  
this is lss lab  
[faculty@localhost ~]$ uniq -c file6.txt  
  2 hai  
  2 this is lss lab  
  1 welcome
```

f) Disk Utilities: du,df

du(disk usage) command:It is used to find disk usage for files and directories.By default du command displays all of the files,directories and subdirectories under the current directory,and it shows how many disk blocks each file or directory takes.

Syntax:

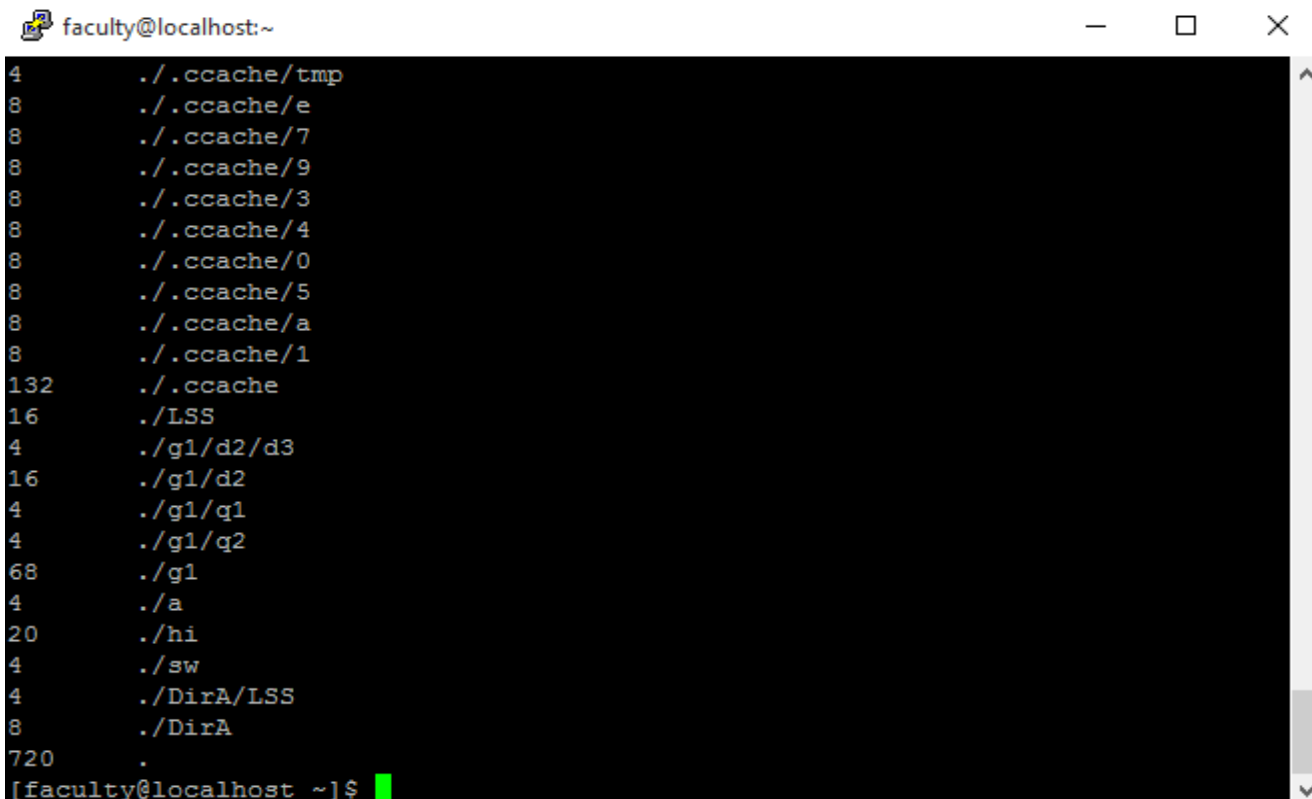
du options inputfile

options:

- c produce a grand total of all the files listed
- h print size in human readable form,using K for kilobyte,M for megabyte,G for gigabyte
- s summarize each argument

Example:

\$ du



```
faculty@localhost:~  
4      ./ccache/tmp  
8      ./ccache/e  
8      ./ccache/7  
8      ./ccache/9  
8      ./ccache/3  
8      ./ccache/4  
8      ./ccache/0  
8      ./ccache/5  
8      ./ccache/a  
8      ./ccache/1  
132    ./ccache  
16     ./LSS  
4      ./g1/d2/d3  
16     ./g1/d2  
4      ./g1/q1  
4      ./g1/q2  
68     ./g1  
4      ./a  
20     ./hi  
4      ./sw  
4      ./DirA/LSS  
8      ./DirA  
720    .  
[faculty@localhost ~]$
```

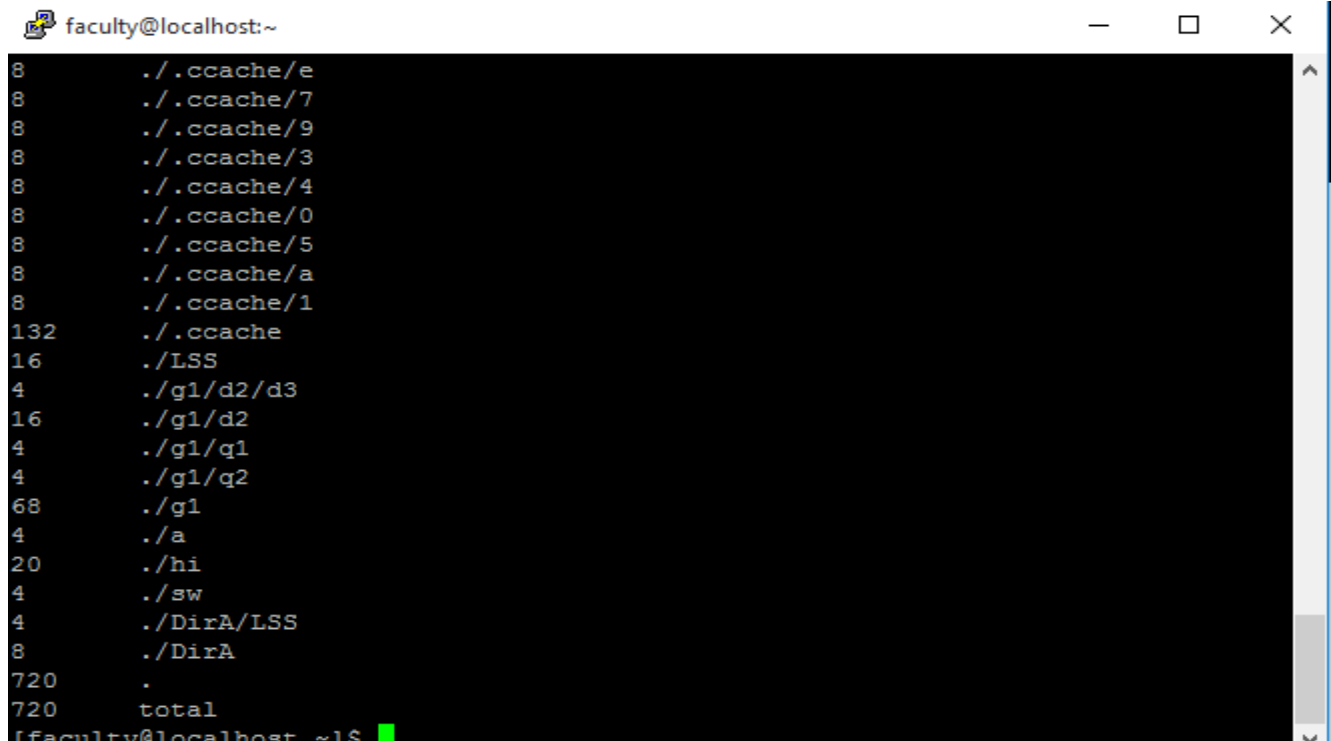
\$ du -s



A terminal window titled 'faculty@localhost:~' with standard window controls. The command '[faculty@localhost ~]\$ du -s' has been executed, resulting in the output '720 .' followed by a new prompt '[faculty@localhost ~]\$'.

```
faculty@localhost:~  
[faculty@localhost ~]$ du -s  
720 .  
[faculty@localhost ~]$
```

\$ du -c



A terminal window titled 'faculty@localhost:~' with standard window controls. The command '[faculty@localhost ~]\$ du -c' has been executed, displaying a detailed disk usage report for various directories and a final 'total' line. The output is as follows:

```
faculty@localhost:~  
[faculty@localhost ~]$ du -c  
8      ./ccache/e  
8      ./ccache/7  
8      ./ccache/9  
8      ./ccache/3  
8      ./ccache/4  
8      ./ccache/0  
8      ./ccache/5  
8      ./ccache/a  
8      ./ccache/1  
132    ./ccache  
16     ./LSS  
4      ./g1/d2/d3  
16     ./g1/d2  
4      ./g1/q1  
4      ./g1/q2  
68     ./g1  
4      ./a  
20     ./hi  
4      ./sw  
4      ./DirA/LSS  
8      ./DirA  
720    .  
720    total  
[faculty@localhost ~]$
```


\$ du -h

```

[faculty@localhost ~]$ du -h
4.0K    ./DirLss2/DirLss1/lab1
8.0K    ./DirLss2/DirLss1
12K     ./DirLss2
4.0K    ./DirA1
4.0K    ./local/share/systemd
8.0K    ./local/share
12K     ./local
4.0K    ./DirBB/DirAA/LSS1
8.0K    ./DirBB/DirAA
12K     ./DirBB
4.0K    ./d1/d2
4.0K    ./d1/d3
60K     ./d1
4.0K    ./DirB1
4.0K    ./DirAA/LSS1
8.0K    ./DirAA
4.0K    ./mozilla/extensions
    
```

df(disk filesystem) command:It is used to show disk utilization for a system or disk space available on an individual device.

Syntax:

df options inputfile

options:

- h information in human readable format
- hT information of particular partition in human readable format

Example:

\$ df

```

[faculty@localhost ~]$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/mapper/fedora-root 51475068 17801628 31035616 37% /
devtmpfs         2015668      0    2015668  0% /dev
tmpfs            2024376      80    2024296  1% /dev/shm
tmpfs            2024376    4904    2019472  1% /run
tmpfs            2024376      0    2024376  0% /sys/fs/cgroup
tmpfs            2024376     12    2024364  1% /tmp
/dev/sda1        487652     80720    377236 18% /boot
/dev/mapper/fedora-home 99511800 8549456 85884328 10% /home
    
```

\$ df -h

\$ df -hT

```

[faculty@localhost ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/fedora-root  50G   17G   30G   37% /
devtmpfs        2.0G    0    2.0G    0% /dev
tmpfs           2.0G   80K    2.0G    1% /dev/shm
tmpfs           2.0G   4.8M    2.0G    1% /run
tmpfs           2.0G    0    2.0G    0% /sys/fs/cgroup
tmpfs           2.0G   12K    2.0G    1% /tmp
/dev/sda1       477M   79M   369M   18% /boot
/dev/mapper/fedora-home  95G   8.2G   82G   10% /home
[faculty@localhost ~]$ df -hT
Filesystem      Type      Size  Used Avail Use% Mounted on
/dev/mapper/fedora-root ext4       50G   17G   30G   37% /
devtmpfs        devtmpfs  2.0G    0    2.0G    0% /dev
tmpfs           tmpfs     2.0G   80K    2.0G    1% /dev/shm
tmpfs           tmpfs     2.0G   4.8M    2.0G    1% /run
tmpfs           tmpfs     2.0G    0    2.0G    0% /sys/fs/cgroup
tmpfs           tmpfs     2.0G   12K    2.0G    1% /tmp
/dev/sda1       ext4      477M   79M   369M   18% /boot
/dev/mapper/fedora-home ext4      95G   8.2G   82G   10% /home
[faculty@localhost ~]$

```

PROGRAM -2

Date:

Aim: Develop a Shell Program to Display all the words which are entered as command line arguments.

Overview of Unix Command Line Arguments:

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.

How to write, compile and run shell programs:

Step1: By using vi command create shell script/program with .sh extension.

\$vi filename.sh

Step2: In the vi editor type the program and for saving & quitting from vi, press **Esc :wq** then press enter

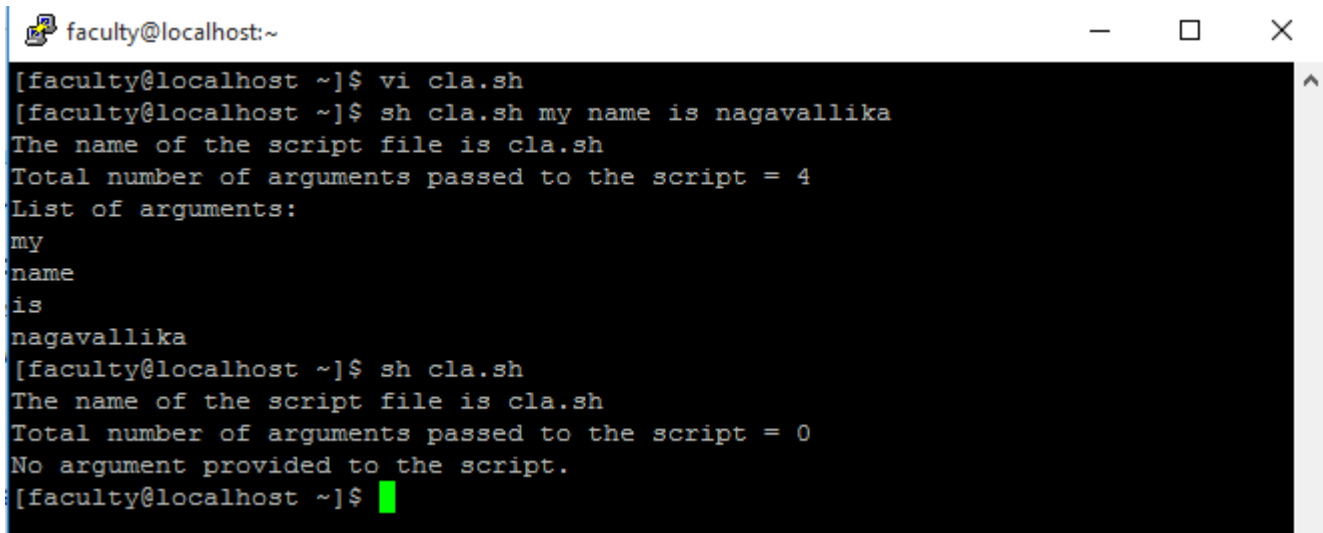
Step3: Compile and execute the shell script using following command

\$sh filename.sh

Shell Script:

```
echo "The name of the script file is $0"
echo "Total number of arguments passed to the script = $#"
```

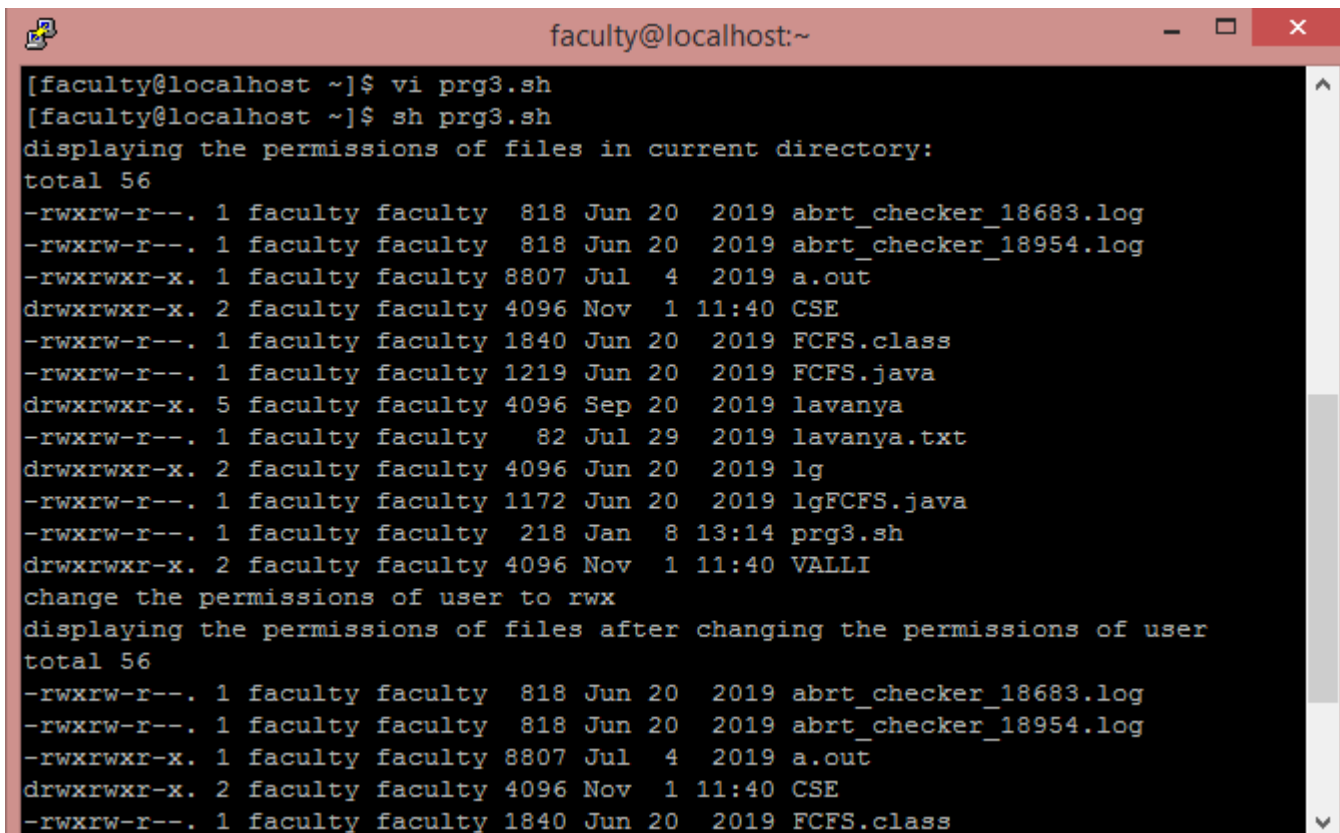
```
if [ $# -gt 0 ]
then
    echo "List of arguments:"
    for arg in $@
    do
        echo "$arg"
    done
else
    echo "No argument provided to the script"
fi
```

Output:

```
faculty@localhost:~
[faculty@localhost ~]$ vi cla.sh
[faculty@localhost ~]$ sh cla.sh my name is nagavallika
The name of the script file is cla.sh
Total number of arguments passed to the script = 4
List of arguments:
my
name
is
nagavallika
[faculty@localhost ~]$ sh cla.sh
The name of the script file is cla.sh
Total number of arguments passed to the script = 0
No argument provided to the script.
[faculty@localhost ~]$
```

PROGRAM -3**Date:****Aim:** Develop a shell script that Changes Permissions of files in PWD as rwx for users.**Shell Script:**

```
echo "displaying the permissions of files in current directory:"  
ls -l  
echo "change the permissions of user to rwx"  
chmod u=rwx *  
echo "displaying the permissions of files after changing the permissions of user"  
ls -l
```

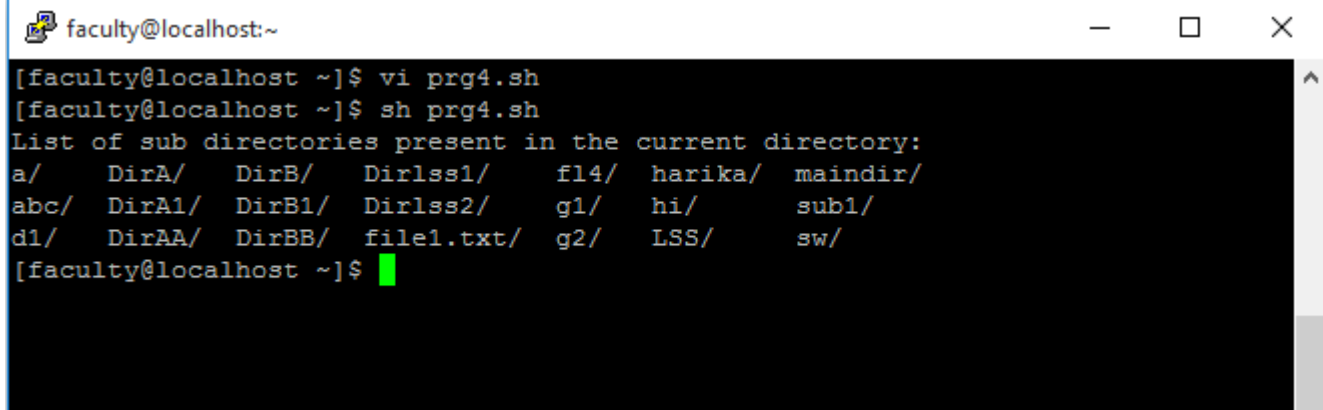
Output:

```
faculty@localhost:~  
[faculty@localhost ~]$ vi prg3.sh  
[faculty@localhost ~]$ sh prg3.sh  
displaying the permissions of files in current directory:  
total 56  
-rwxrw-r--. 1 faculty faculty 818 Jun 20 2019 abrt_checker_18683.log  
-rwxrw-r--. 1 faculty faculty 818 Jun 20 2019 abrt_checker_18954.log  
-rwxrwxr-x. 1 faculty faculty 8807 Jul 4 2019 a.out  
drwxrwxr-x. 2 faculty faculty 4096 Nov 1 11:40 CSE  
-rwxrw-r--. 1 faculty faculty 1840 Jun 20 2019 FCFS.class  
-rwxrw-r--. 1 faculty faculty 1219 Jun 20 2019 FCFS.java  
drwxrwxr-x. 5 faculty faculty 4096 Sep 20 2019 lavanya  
-rwxrw-r--. 1 faculty faculty 82 Jul 29 2019 lavanya.txt  
drwxrwxr-x. 2 faculty faculty 4096 Jun 20 2019 lg  
-rwxrw-r--. 1 faculty faculty 1172 Jun 20 2019 lgFCFS.java  
-rwxrw-r--. 1 faculty faculty 218 Jan 8 13:14 prg3.sh  
drwxrwxr-x. 2 faculty faculty 4096 Nov 1 11:40 VALLI  
change the permissions of user to rwx  
displaying the permissions of files after changing the permissions of user  
total 56  
-rwxrw-r--. 1 faculty faculty 818 Jun 20 2019 abrt_checker_18683.log  
-rwxrw-r--. 1 faculty faculty 818 Jun 20 2019 abrt_checker_18954.log  
-rwxrwxr-x. 1 faculty faculty 8807 Jul 4 2019 a.out  
drwxrwxr-x. 2 faculty faculty 4096 Nov 1 11:40 CSE  
-rwxrw-r--. 1 faculty faculty 1840 Jun 20 2019 FCFS.class
```

PROGRAM -4**Date:****Aim:** Develop a shell script to print the list of all sub directories in the current directory.**Shell Script:**

```
echo "List of sub directories present in the current directory :"
```

```
ls -d */
```

Output:A terminal window titled 'faculty@localhost:~' with standard window controls. The user enters 'vi prg4.sh' and then 'sh prg4.sh'. The script outputs 'List of sub directories present in the current directory:' followed by a grid of directory names: a/, DirA/, DirB/, Dirlss1/, fl4/, harika/, maindir/, abc/, DirA1/, DirB1/, Dirlss2/, g1/, hi/, sub1/, d1/, DirAA/, DirBB/, file1.txt/, g2/, LSS/, and sw/. The prompt returns to the user.

```
faculty@localhost:~  
[faculty@localhost ~]$ vi prg4.sh  
[faculty@localhost ~]$ sh prg4.sh  
List of sub directories present in the current directory:  
a/   DirA/  DirB/  Dirlss1/  fl4/  harika/  maindir/  
abc/ DirA1/  DirB1/  Dirlss2/  g1/   hi/      sub1/  
d1/  DirAA/  DirBB/  file1.txt/ g2/   LSS/     sw/  
[faculty@localhost ~]$
```

Shell script:

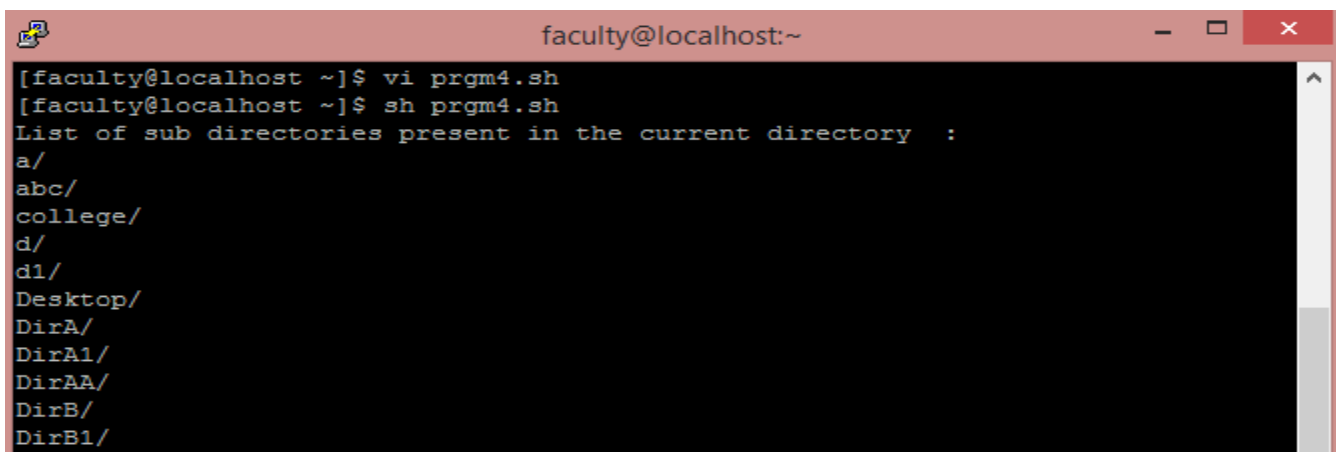
```
echo "List of sub directories present in the current directory :"
```

```
for d in */
```

```
do
```

```
echo $d
```

```
done
```

Output:A terminal window titled 'faculty@localhost:~' with standard window controls. The user enters 'vi prgm4.sh' and then 'sh prgm4.sh'. The script outputs 'List of sub directories present in the current directory :' followed by a list of directory names: a/, abc/, college/, d/, d1/, Desktop/, DirA/, DirA1/, DirAA/, DirB/, and DirB1/. The prompt returns to the user.

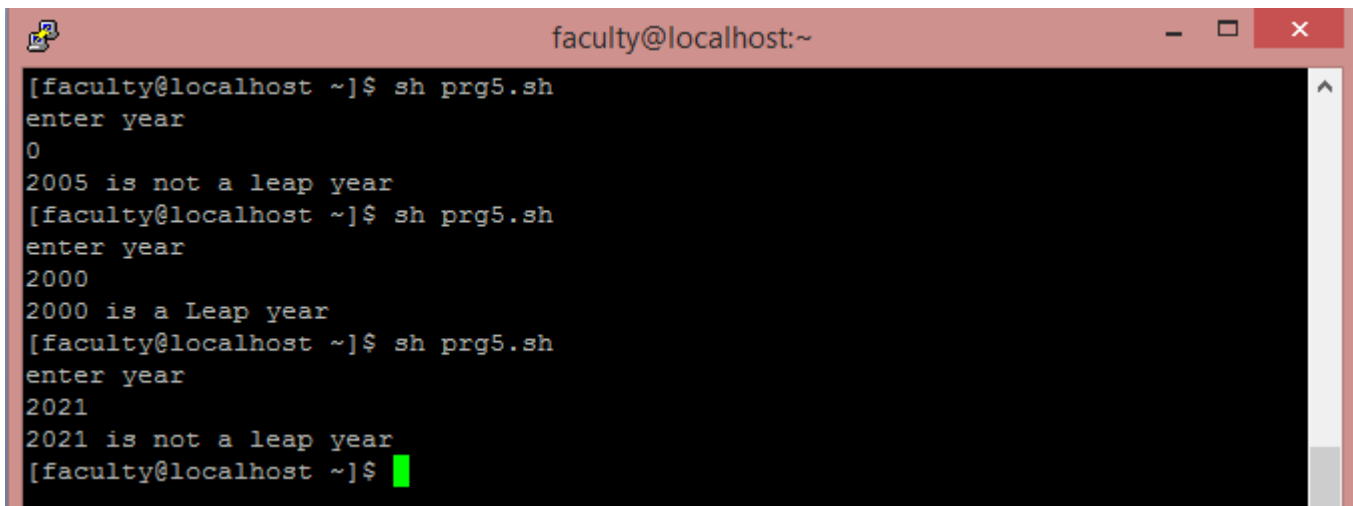
```
faculty@localhost:~  
[faculty@localhost ~]$ vi prgm4.sh  
[faculty@localhost ~]$ sh prgm4.sh  
List of sub directories present in the current directory :  
a/  
abc/  
college/  
d/  
d1/  
Desktop/  
DirA/  
DirA1/  
DirAA/  
DirB/  
DirB1/
```

PROGRAM -5**Date:**

Aim: Develop a Shell Program which receives any year from the keyboard and determine whether the year is leap year or not. If no argument is supplied the current year should be assumed.

Shell Script:

```
echo "enter year "  
read year  
#if year is less than or equal to 0 then current year is assigned to year variable  
if [ $year -le 0 ]  
then  
year=`date | cut -d " " -f6`  
fi  
if [ `expr $year % 400` -eq 0 ]  
then  
echo "$year is a Leap year"  
elif [ `expr $year % 100` -eq 0 ]  
then  
echo "$year is not a Leap year"  
elif [ `expr $year % 4` -eq 0 ]  
then  
echo "$year is a Leap year"  
else  
echo "$year is not a Leap year"  
fi
```

Output:

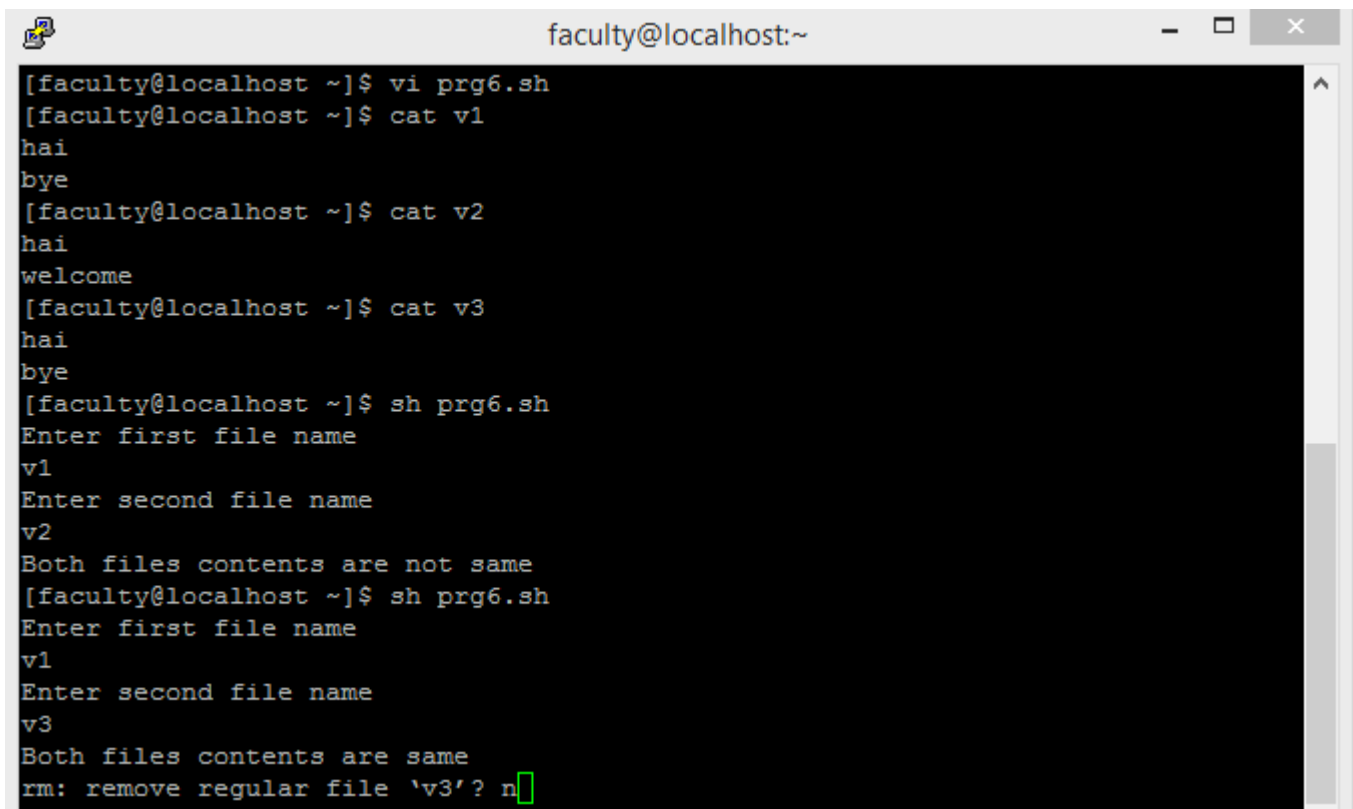
```
faculty@localhost:~  
[faculty@localhost ~]$ sh prg5.sh  
enter year  
0  
2005 is not a leap year  
[faculty@localhost ~]$ sh prg5.sh  
enter year  
2000  
2000 is a Leap year  
[faculty@localhost ~]$ sh prg5.sh  
enter year  
2021  
2021 is not a leap year  
[faculty@localhost ~]$
```

PROGRAM -6**Date:**

Aim: Develop a shell script which takes two file names as arguments-If their contents are same then delete the second file.

Shell script:

```
echo "Enter first file name "  
read file1  
echo "Enter second file name "  
read file2  
if cmp -s "$file1" "$file2"  
then  
    echo "Both files contents are same"  
    rm -i "$file2"  
else  
    echo "Both files contents are not same"  
fi
```

Output:

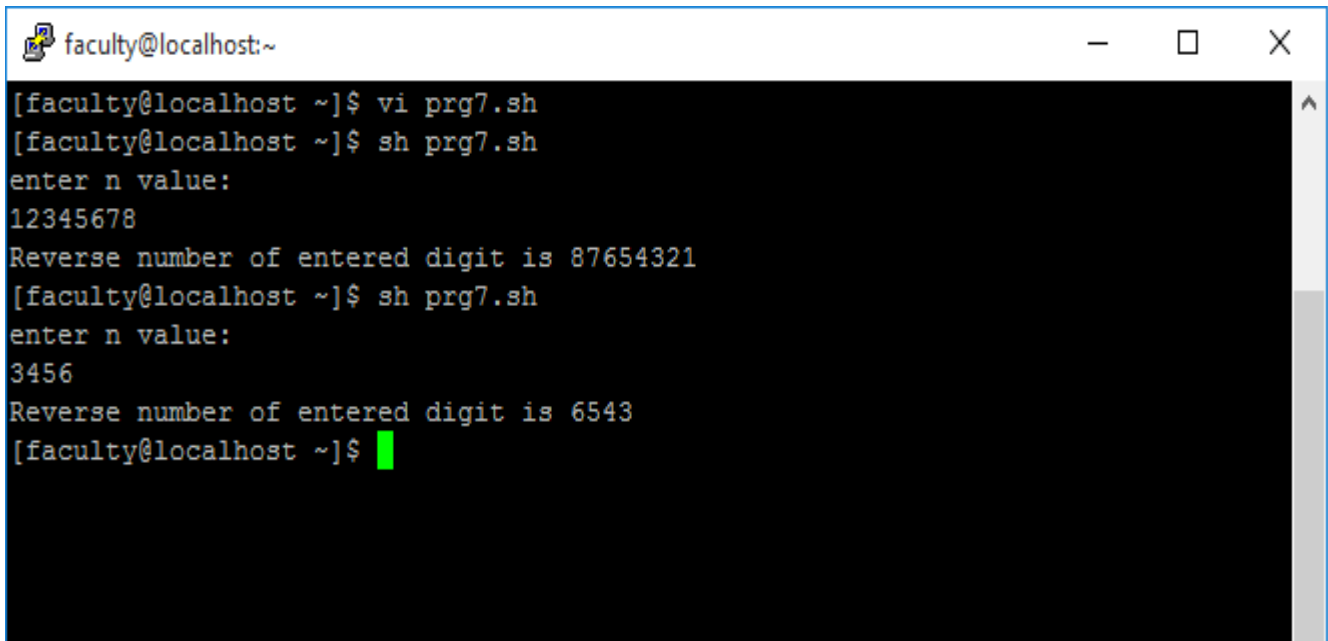
```
faculty@localhost:~  
[faculty@localhost ~]$ vi prg6.sh  
[faculty@localhost ~]$ cat v1  
hai  
bye  
[faculty@localhost ~]$ cat v2  
hai  
welcome  
[faculty@localhost ~]$ cat v3  
hai  
bye  
[faculty@localhost ~]$ sh prg6.sh  
Enter first file name  
v1  
Enter second file name  
v2  
Both files contents are not same  
[faculty@localhost ~]$ sh prg6.sh  
Enter first file name  
v1  
Enter second file name  
v3  
Both files contents are same  
rm: remove regular file 'v3'? n
```

PROGRAM -7**Date:**

Aim: Develop a shell script to print the given number in the reversed order.

Shell script:

```
echo "enter n value:"
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:"
```

Output:

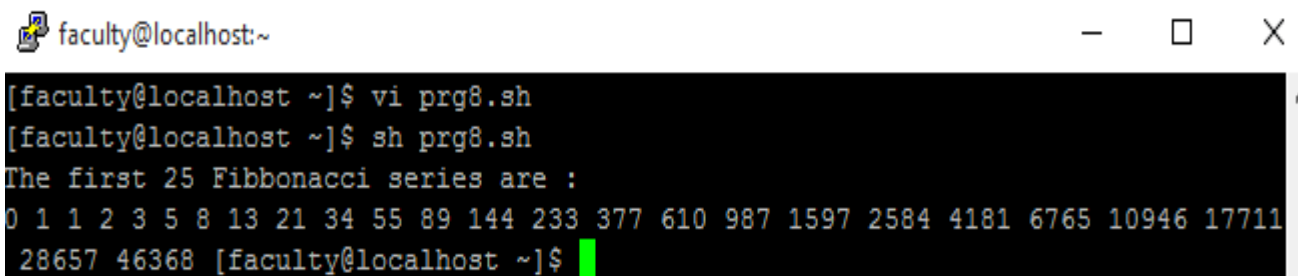
```
faculty@localhost:~
[faculty@localhost ~]$ vi prg7.sh
[faculty@localhost ~]$ sh prg7.sh
enter n value:
12345678
Reverse number of entered digit is 87654321
[faculty@localhost ~]$ sh prg7.sh
enter n value:
3456
Reverse number of entered digit is 6543
[faculty@localhost ~]$
```


PROGRAM -8**Date:**

Aim: Develop a shell script to print first 25 Fibonacci numbers.

Shell script:

```
n=25
a=0
b=1
echo "The first 25 Fibonacci series are : "
for((i=0;i<n i++))
do
    echo -n "$a "
    fn=$((a + b))
    a=$b
    b=$fn
done
```

Output:

The screenshot shows a terminal window with the following content:

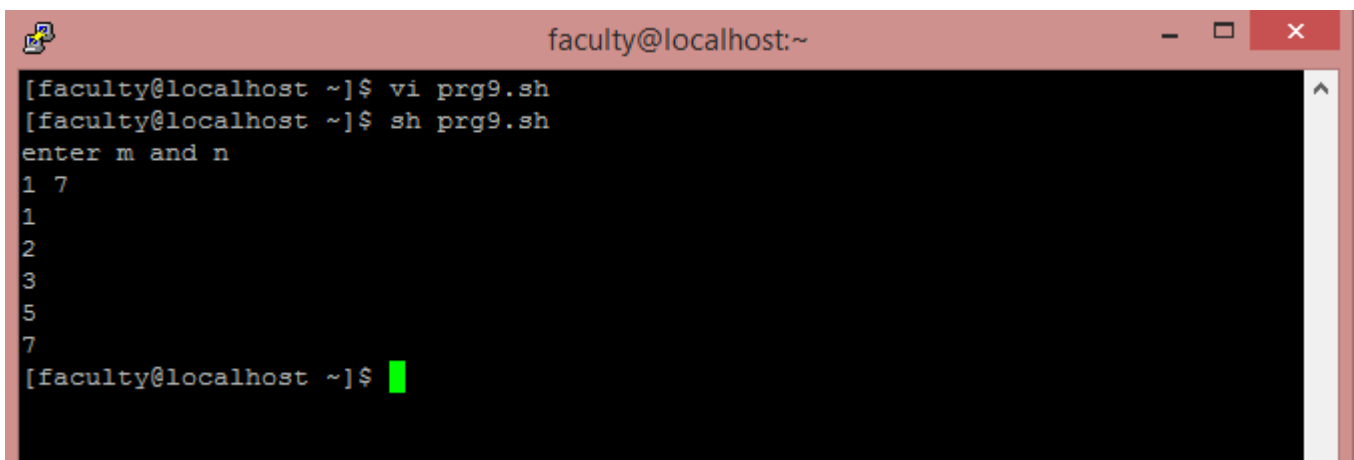
```
faculty@localhost:~
[faculty@localhost ~]$ vi prg8.sh
[faculty@localhost ~]$ sh prg8.sh
The first 25 Fibonacci series are :
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711
28657 46368 [faculty@localhost ~]$
```

PROGRAM -9**Date:**

Aim: Develop a shell script to print the Prime numbers between the specified range.

Shell script:

```
echo "enter m and n"
read m n
for a in $(seq $m $n)
do
    k=0
    for i in $(seq 2 $(expr $a - 1))
    do
        if [ $(expr $a % $i) -eq 0 ]
        then
            k=1
            break
        fi
    done
    if [ $k -eq 0 ]
    then
        echo $a
    fi
done
```

Output:

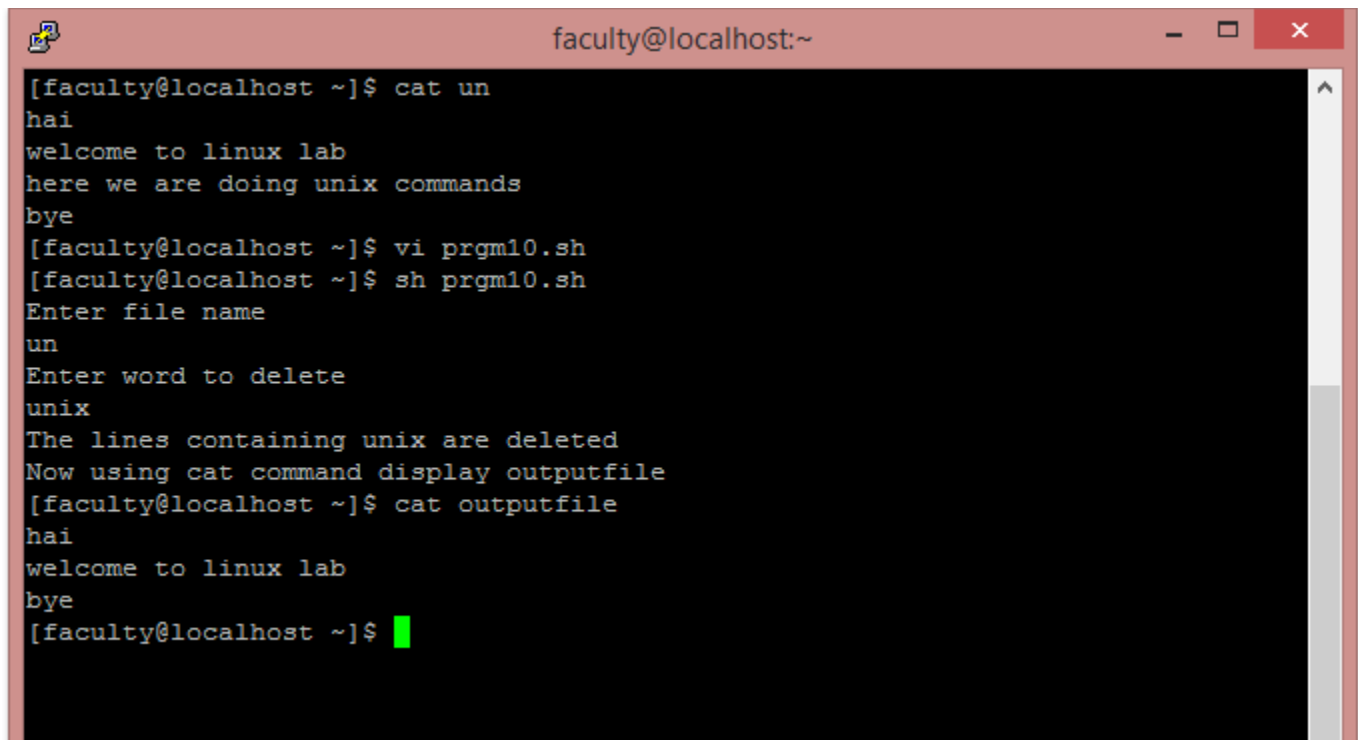
```
faculty@localhost:~  
[faculty@localhost ~]$ vi prg9.sh  
[faculty@localhost ~]$ sh prg9.sh  
enter m and n  
1 7  
1  
2  
3  
5  
7  
[faculty@localhost ~]$
```

PROGRAM -10**Date:**

Aim: Develop a shell script to delete all lines containing the word 'unix' in the files supplied as arguments.

Shell script:

```
echo "enter file name"
read FILE
echo "Enter word to delete"
read word1
sed -e '/unix/d' $FILE>outputfile
echo "The lines containing unix $word1 are deleted"
echo "now using cat display outputfile"
```

Output:

```
faculty@localhost:~
[faculty@localhost ~]$ cat un
hai
welcome to linux lab
here we are doing unix commands
bye
[faculty@localhost ~]$ vi prgm10.sh
[faculty@localhost ~]$ sh prgm10.sh
Enter file name
un
Enter word to delete
unix
The lines containing unix are deleted
Now using cat command display outputfile
[faculty@localhost ~]$ cat outputfile
hai
welcome to linux lab
bye
[faculty@localhost ~]$
```

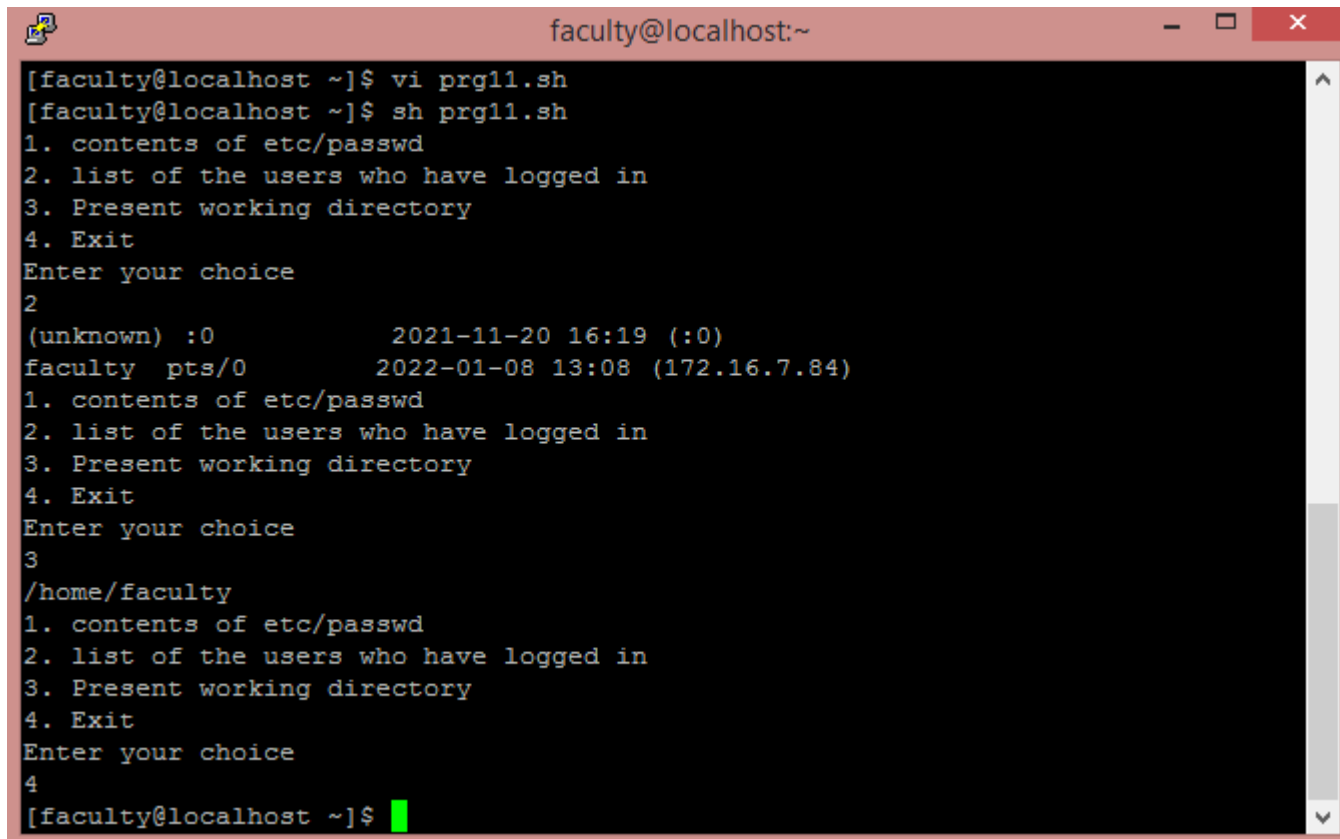
PROGRAM -11**Date:**

Aim: Develop a shell script Menu driven program which has the following options.

- i) contents of /etc/passwd
- ii) list of users who have currently logged in.
- iii) present working directory.
- iv) exit.

Shell script:

```
choice=0
while [ $choice -lt 4 ]
do
tput clear
echo "1. contents of etc/passwd"
echo "2. list of the users who have logged in"
echo "3. Present working directory"
echo "4. Exit"
echo "Enter your choice "
read choice
if [ $choice -eq 1 ]
then
cat /etc/passwd
elif [ $choice -eq 2 ]
then
who
elif [ $choice -eq 3 ]
then
pwd
fi
done
```

Output:

```
faculty@localhost:~  
[faculty@localhost ~]$ vi prg11.sh  
[faculty@localhost ~]$ sh prg11.sh  
1. contents of etc/passwd  
2. list of the users who have logged in  
3. Present working directory  
4. Exit  
Enter your choice  
2  
(unknown) :0          2021-11-20 16:19 (:0)  
faculty pts/0        2022-01-08 13:08 (172.16.7.84)  
1. contents of etc/passwd  
2. list of the users who have logged in  
3. Present working directory  
4. Exit  
Enter your choice  
3  
/home/faculty  
1. contents of etc/passwd  
2. list of the users who have logged in  
3. Present working directory  
4. Exit  
Enter your choice  
4  
[faculty@localhost ~]$
```