Index

Program 1:- Introduction to the Linux Server and understand the basic Directory Structure of Linux.

Solution:-

What is Linux?

Linux is a free and open-source family of operating systems that is resilient and flexible. In 1991, an individual by the name as Linus Torvalds constructed it. The system's source code is accessible to everyone for anyone to look at and change, making it cool that anyone can see how the system works. People from all across the world are urged to work together and keep developing Linux due to its openness. Since the beginning, Linux has grown into a dependable and safe OS that is used in an array of gadgets, including PCs, cell phones, and huge supercomputers. It is well-known for being cost-effective, which implies that employing it doesn't cost a lot, and efficient, which indicates it can complete a lot of jobs quickly. A lot of people love Linux, and

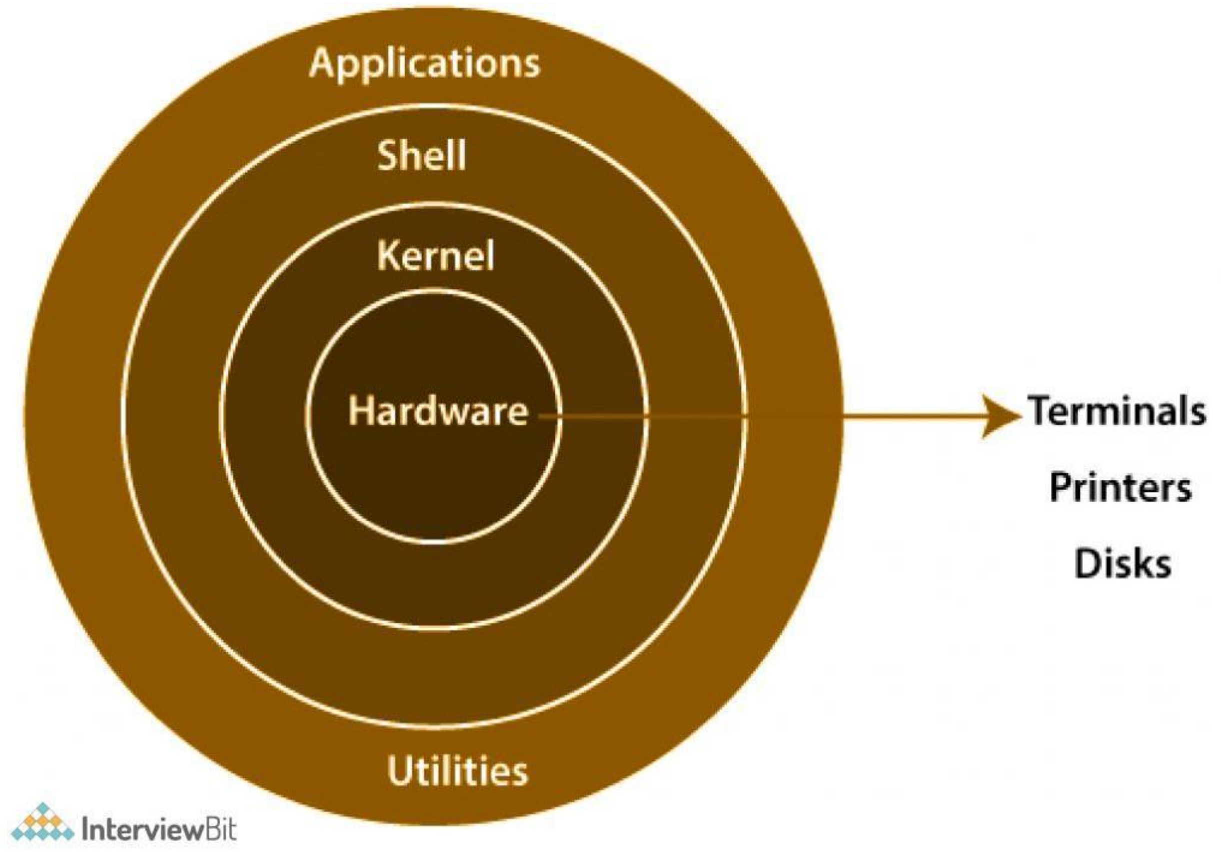
What is Linux Operating System?

Developed by Linus Torvalds in 1991, the Linux operating system is a powerful and flexible open-source software platform. It acts as the basis for a variety of devices, such embedded systems, cell phones, servers, and personal computers. Linux, that's well-known for its reliability, safety, and flexibility, allows users to customize and improve their environment to suit specific needs. With an extensive and active community supporting it, Linux is an appealing choice for people as well as companies due to its wealth of resources and constant developments.

Linux Architecture

A computer's operating system interface to the hardware is referred to as a software application. A number of software applications are run on operating systems to manage hardware resources on a computer.

The diagram illustrates the structure of the Linux system, according to the layers concept.



Linux Directory Structure

In Linux/Unix operating system everything is a file even directories are files, files are files, and devices like mouse, keyboard, printer, etc are also files. Here we are going to see the Directory Structure in Linux.

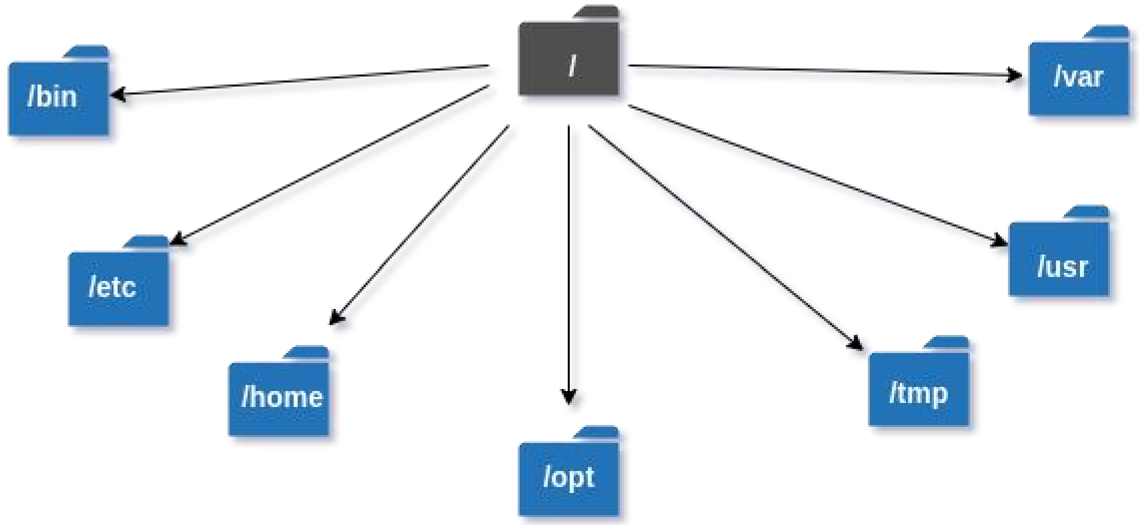
Types of files in the Linux system.

1. General Files – It is also called ordinary files. It may be an image, video, program, or simple text file. These types of files can be in ASCII or Binary format. It is the most commonly used file in the Linux system.
2. Directory Files – These types of files are a warehouse for other file types. It may be a directory file within a directory (subdirectory).

Device Files – In a Windows-like operating system, devices like CD-ROM, and hard drives are represented as drive letters like F: G: H

3. whereas in the Linux system devices are represented as files. As for example, /dev/sda1, /dev/sda2, and so on.

We know that in a Windows-like operating system, files are stored in different folders on different data drives like C: D: E: whereas in the Linux/Unix operating system files are stored in a tree-like structure starting with the root directory as shown in the below diagram.



The Linux/Unix file system hierarchy base begins at the root and everything starts with the root directory.

These are the common top-level directories associated with the root directory:

Directories	**Description**
/bin	**binary or executable programs.**
/etc	**system configuration files.**
/home	**home directory. It is the default current directory.**
/opt	**optional or third-party software.**

Directories	Description
/tmp	temporary space, typically cleared on reboot.
/usr	User related programs.
/var	log files.

Some other directories in the Linux system:

Directories	Description
/boot	It contains all the boot-related information files and folders such as conf, grub, etc.
/dev	It is the location of the device files such as dev/sda1, dev/sda2, etc.
/lib	It contains kernel modules and a shared library.
/lost+found	It is used to find recovered bits of corrupted files.
/media	It contains subdirectories where removal media devices are inserted.
/mnt	It contains temporary mount directories for mounting the file system.

Directories	Description
/proc	It is a virtual and pseudo-file system to contains info about the running processes with a specific process ID or PID.
/run	It stores volatile runtime data.
/sbin	binary executable programs for an administrator.

/srv	It contains server-specific and server-related files.
/sys	It is a virtual file system for modern Linux distributions to store and allows modification of the devices connected to the system.

Exploring directories and their usability:

We know that Linux is a very complex system that requires an efficient way to start, stop, maintain and reboot a system, unlike Windows operating system. In the Linux system some well-defined configuration files, binaries, main pages information files are available for every process.

Linux Kernel File:

- **/boot/vmlinux –** The Linux kernel file.

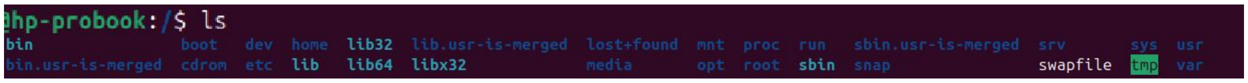
Device Files:

- **/dev/hda –** Device file for the first IDE HDD.
- **/dev/hdc –** A pseudo-device that output garbage output is redirected to /dev/null.

To check the Linux directories, open the terminal and execute **sudo -s** followed by system password to give root privilege. Then after changing the current home directory to the root directory and check the list of all available directories in the base directory using ls command

In Linux, the root directory is represented by `/`, and everything is organized under it. Here's a summary of key directories:

- / : The root directory of the entire file system.
- /bin : Contains essential command binaries (e.g., `ls`, `cp`, `mv`), needed for all users.
- /boot : Stores files needed to boot the system, including the Linux kernel.
- /dev : Contains device files, which represent hardware components (e.g., disks, USB drives).
- /etc : Stores configuration files for the system and applications (e.g., network settings, user login details).
- /home : Contains home directories for each user. User-specific files and personal data are stored here.
- /lib : Holds shared libraries needed by the binaries in `/bin` and `/sbin`.
- /media : A directory where removable media like USB drives are mounted.
- /mnt : A temporary mount point for mounting filesystems manually.
- /opt : Contains optional software and application packages.
- /proc : A virtual filesystem providing information about system processes and kernel information.
- /root : The home directory for the root (administrator) user.
- /sbin : Contains essential system binaries, typically for administrative tasks.
- /tmp : Stores temporary files, often cleared on system reboot.
- /usr : Contains user programs and utilities, further divided into `/usr/bin`, `/usr/lib`, etc.
- /var : Stores variable files like logs, databases, websites, etc., that change frequently.

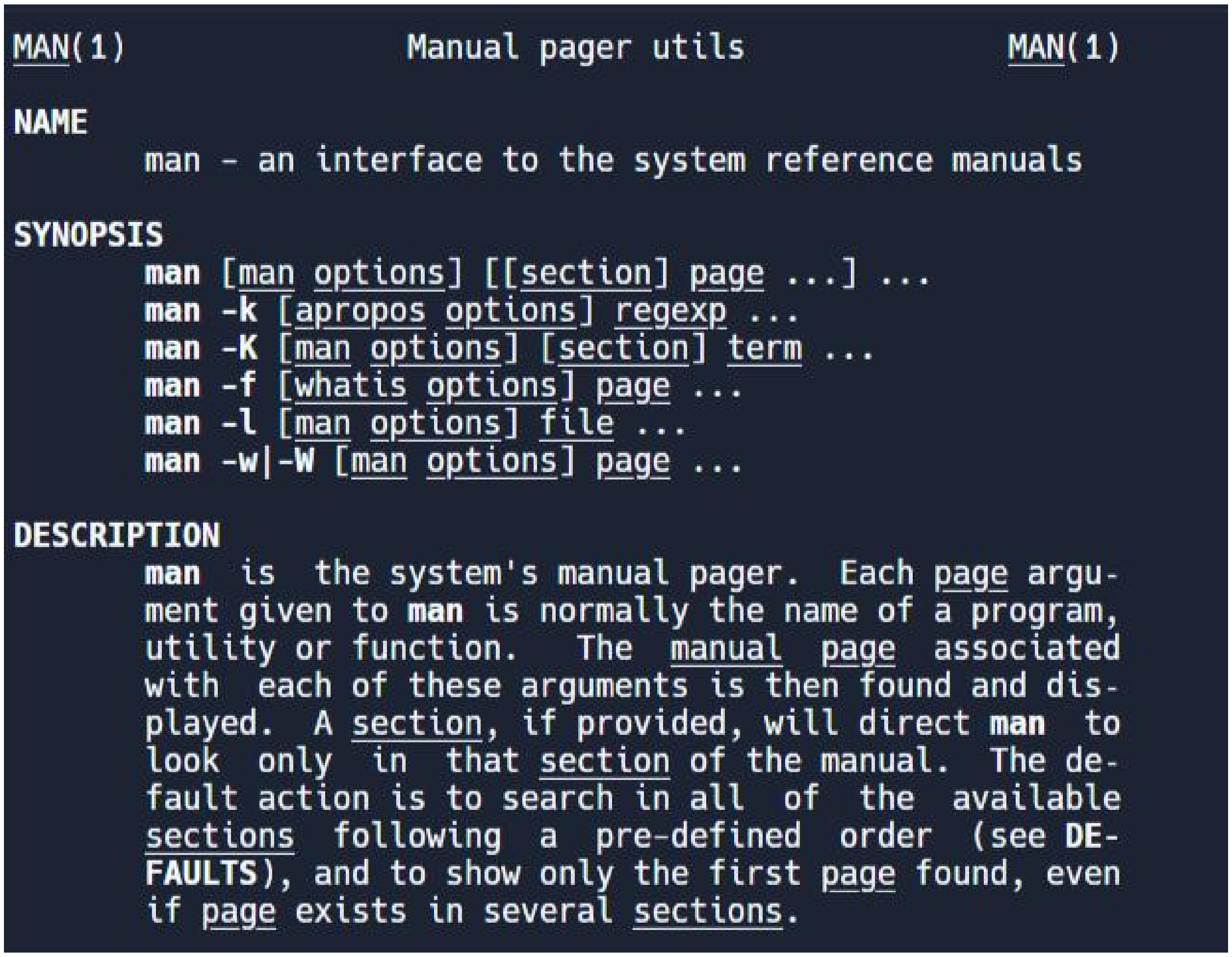


```
@hp-probook:/$ ls
bin                boot   dev  home  lib32  lib.usr-is-merged  lost+found  mnt  proc  run   sbin.usr-is-merged  srv       sys  usr
bin.usr-is-merged  cdrom  etc  lib   lib64  libx32             media       opt  root  sbin  snap                swapfile  tmp  var
```

Program 2:- To understand help command like:- man, info, help, whatis, apropos.

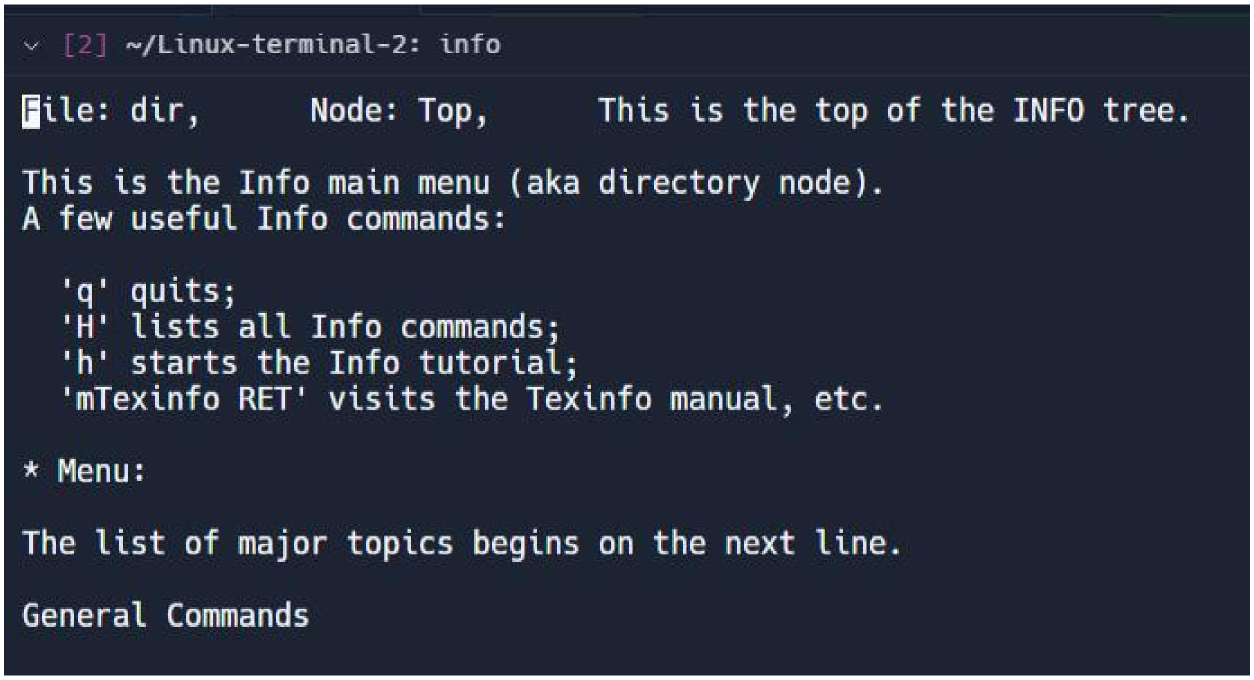
Solution :-

1- Man :-



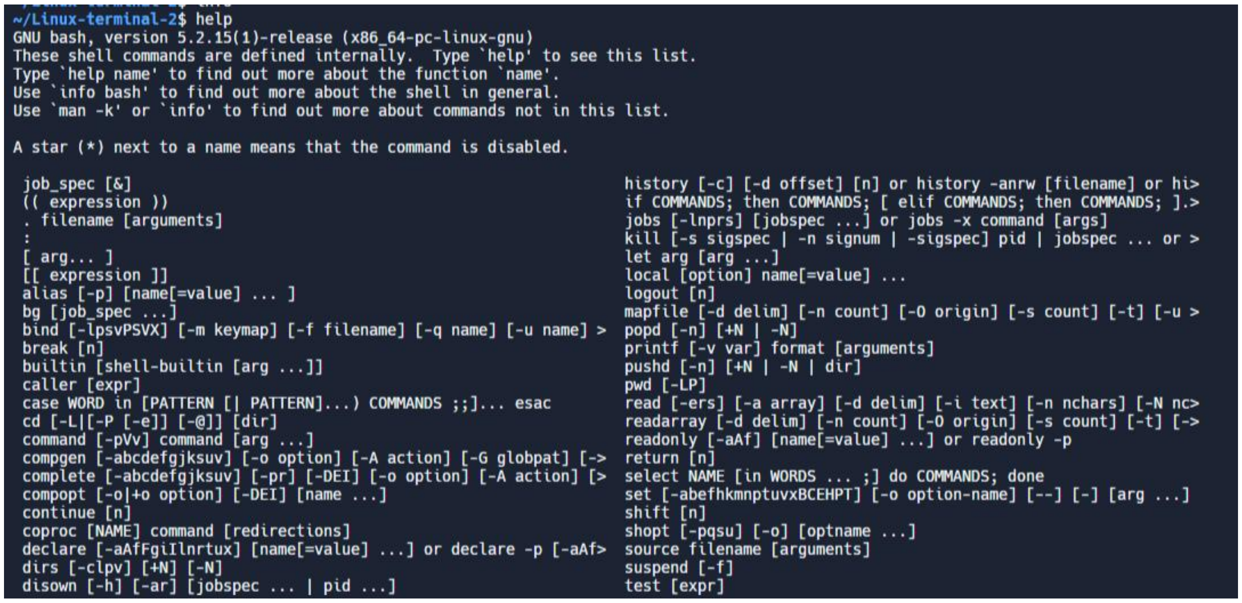
```
MAN(1)                    Manual pager utils              MAN(1)

NAME
       man - an interface to the system reference manuals

SYNOPSIS
       man [man options] [[section] page ...] ...
       man -k [apropos options] regexp ...
       man -K [man options] [section] term ...
       man -f [whatis options] page ...
       man -l [man options] file ...
       man -w|-W [man options] page ...

DESCRIPTION
       man  is  the system's manual pager.  Each page argu-
       ment given to man is normally the name of a program,
       utility or function.   The  manual  page  associated
       with  each of these arguments is then found and dis-
       played.  A section, if provided, will direct man  to
       look  only  in  that section of the manual.  The de-
       fault action is to search in all  of  the  available
       sections  following  a  pre-defined  order  (see DE-
       FAULTS), and to show only the first page found, even
       if page exists in several sections.
```

2-info :-



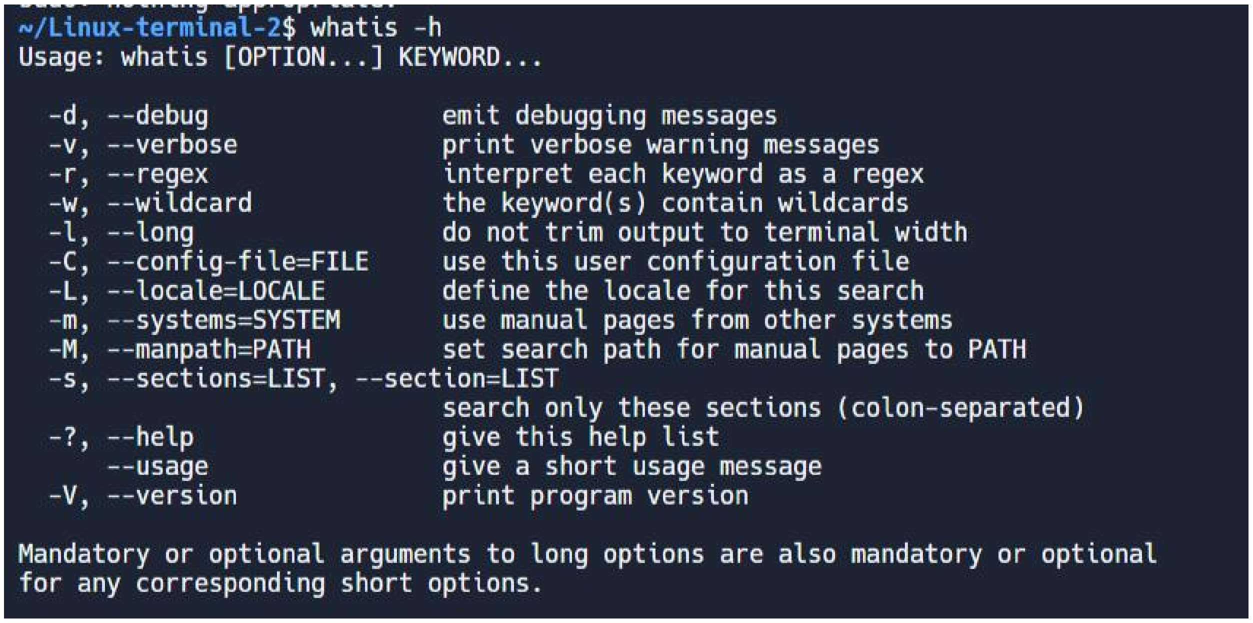
```
⌄ [2] ~/Linux-terminal-2: info

File: dir,      Node: Top,      This is the top of the INFO tree.

This is the Info main menu (aka directory node).
A few useful Info commands:

  'q' quits;
  'H' lists all Info commands;
  'h' starts the Info tutorial;
  'mTexinfo RET' visits the Texinfo manual, etc.

* Menu:

The list of major topics begins on the next line.

General Commands
```

3-help



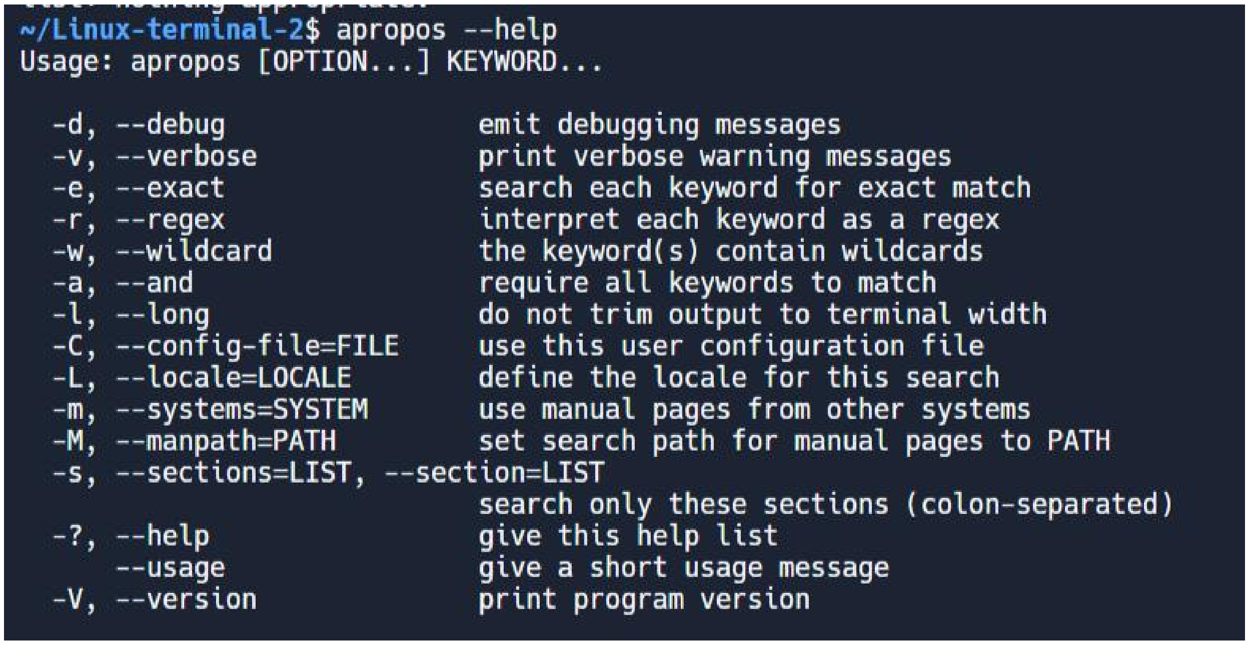
```
~/Linux-terminal-2$ help
GNU bash, version 5.2.15(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

 job_spec [&]                                                     history [-c] [-d offset] [n] or history -anrw [filename] or hi>
 (( expression ))                                                 if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ].>
 . filename [arguments]                                           jobs [-lnprs] [jobspec ...] or jobs -x command [args]
 :                                                                kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or >
 [ arg... ]                                                       let arg [arg ...]
 [[ expression ]]                                                 local [option] name[=value] ...
 alias [-p] [name[=value] ... ]                                   logout [n]
 bg [job_spec ...]                                                mapfile [-d delim] [-n count] [-O origin] [-s count] [-t] [-u >
 bind [-lpsvPSVX] [-m keymap] [-f filename] [-q name] [-u name] > popd [-n] [+N | -N]
 break [n]                                                        printf [-v var] format [arguments]
 builtin [shell-builtin [arg ...]]                                pushd [-n] [+N | -N | dir]
 caller [expr]                                                    pwd [-LP]
 case WORD in [PATTERN [| PATTERN]...) COMMANDS ;;]... esac       read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nc>
 cd [-L|[-P [-e]] [-@]] [dir]                                     readarray [-d delim] [-n count] [-O origin] [-s count] [-t] [->
 command [-pVv] command [arg ...]                                 readonly [-aAf] [name[=value] ...] or readonly -p
 compgen [-abcdefgjksuv] [-o option] [-A action] [-G globpat] [-> return [n]
 complete [-abcdefgjksuv] [-pr] [-DEI] [-o option] [-A action] [> select NAME [in WORDS ... ;] do COMMANDS; done
 compopt [-o|+o option] [-DEI] [name ...]                         set [-abefhkmnptuvxBCEHPT] [-o option-name] [--] [-] [arg ...]
 continue [n]                                                     shift [n]
 coproc [NAME] command [redirections]                             shopt [-pqsu] [-o] [optname ...]
 declare [-aAfFgiIlnrtux] [name[=value] ...] or declare -p [-aAf> source filename [arguments]
 dirs [-clpv] [+N] [-N]                                           suspend [-f]
 disown [-h] [-ar] [jobspec ... | pid ...]                        test [expr]
```

4-whatis -h :- The whatis command throws an error if no options, filenames, or arguments are passed. So, when we use the -h option, it gives the general syntax along with the various options that can be used.



```
~/Linux-terminal-2$ whatis -h
Usage: whatis [OPTION...] KEYWORD...

  -d, --debug                emit debugging messages
  -v, --verbose              print verbose warning messages
  -r, --regex                interpret each keyword as a regex
  -w, --wildcard             the keyword(s) contain wildcards
  -l, --long                 do not trim output to terminal width
  -C, --config-file=FILE     use this user configuration file
  -L, --locale=LOCALE        define the locale for this search
  -m, --systems=SYSTEM       use manual pages from other systems
  -M, --manpath=PATH         set search path for manual pages to PATH
  -s, --sections=LIST, --section=LIST
                             search only these sections (colon-separated)
  -?, --help                 give this help list
      --usage                give a short usage message
  -V, --version              print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.
```

5- apropos –help :-

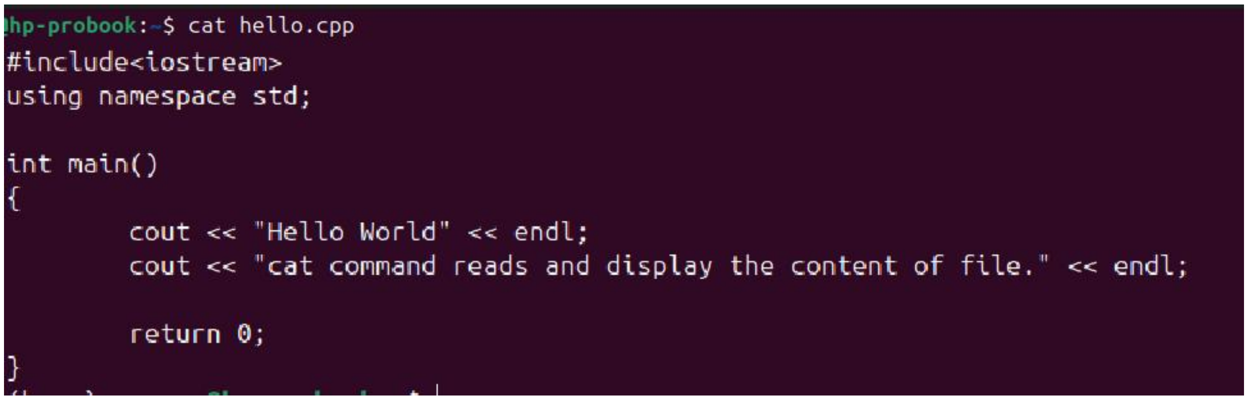


```
~/Linux-terminal-2$ apropos --help
Usage: apropos [OPTION...] KEYWORD...

  -d, --debug                emit debugging messages
  -v, --verbose              print verbose warning messages
  -e, --exact                search each keyword for exact match
  -r, --regex                interpret each keyword as a regex
  -w, --wildcard             the keyword(s) contain wildcards
  -a, --and                  require all keywords to match
  -l, --long                 do not trim output to terminal width
  -C, --config-file=FILE     use this user configuration file
  -L, --locale=LOCALE        define the locale for this search
  -m, --systems=SYSTEM       use manual pages from other systems
  -M, --manpath=PATH         set search path for manual pages to PATH
  -s, --sections=LIST, --section=LIST
                             search only these sections (colon-separated)
  -?, --help                 give this help list
      --usage                give a short usage message
  -V, --version              print program version
```

Program 3:- To understand basic directory navigation commands like cat, cd, mv, cp, rm, mkdir, rmdir, file, pwd command.

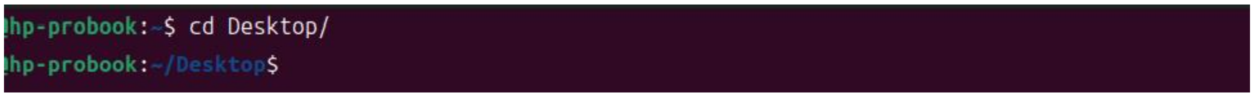
Solution:-

1-cat:- Reads and displays the content of files.

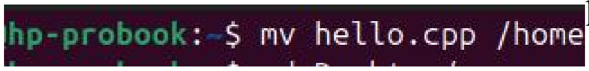


```
hp-probook:~$ cat hello.cpp
#include<iostream>
using namespace std;

int main()
{
        cout << "Hello World" << endl;
        cout << "cat command reads and display the content of file." << endl;

        return 0;
}
```

2-cd:- Change Directory.

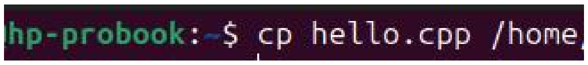


```
hp-probook:~$ cd Desktop/
hp-probook:~/Desktop$
```



```
hp-probook:~$ mv hello.cpp /home
```

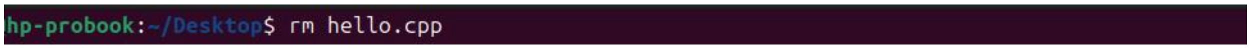
3-mv:- Moves a file or directory to a new location or renames it.



```
hp-probook:~$ cp hello.cpp /home
```

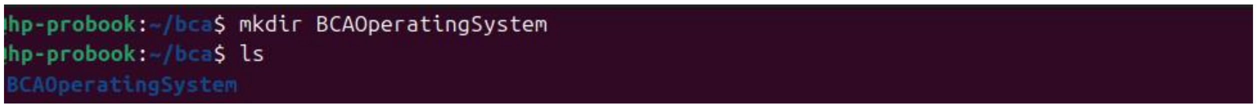
4-cp:- Copies files or directories.

5-rm:- Deletes files or directories.

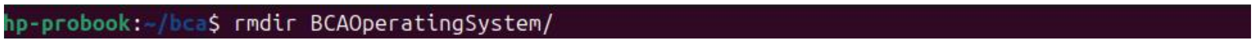


```
hp-probook:~/Desktop$ rm hello.cpp
```

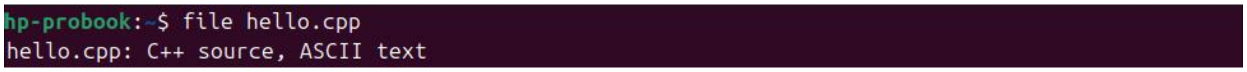
6-mkdir:- Creates a new directory.

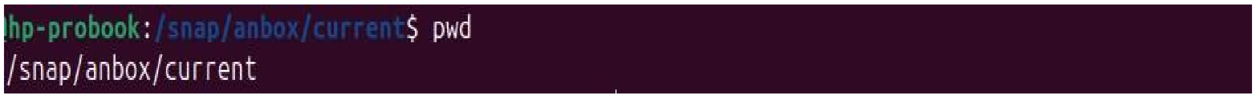

```
hp-probook:~/bca$ mkdir BCAOperatingSystem
hp-probook:~/bca$ ls
BCAOperatingSystem
```

7-rmdir:- Deletes an empty directory.


```
hp-probook:~/bca$ rmdir BCAOperatingSystem/
```

8-file:- Shows the file type (e.g., text, binary, image).


```
hp-probook:~$ file hello.cpp
hello.cpp: C++ source, ASCII text
```


```
hp-probook:/snap/anbox/current$ pwd
/snap/anbox/current
```

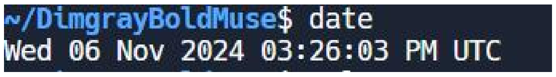
9-pwd:- Displays the current directory path.

Program 4:- To understand basic commands like:-

date , cal, echo, bc, ls, who, whoami, hostname, uname, tty, alias.

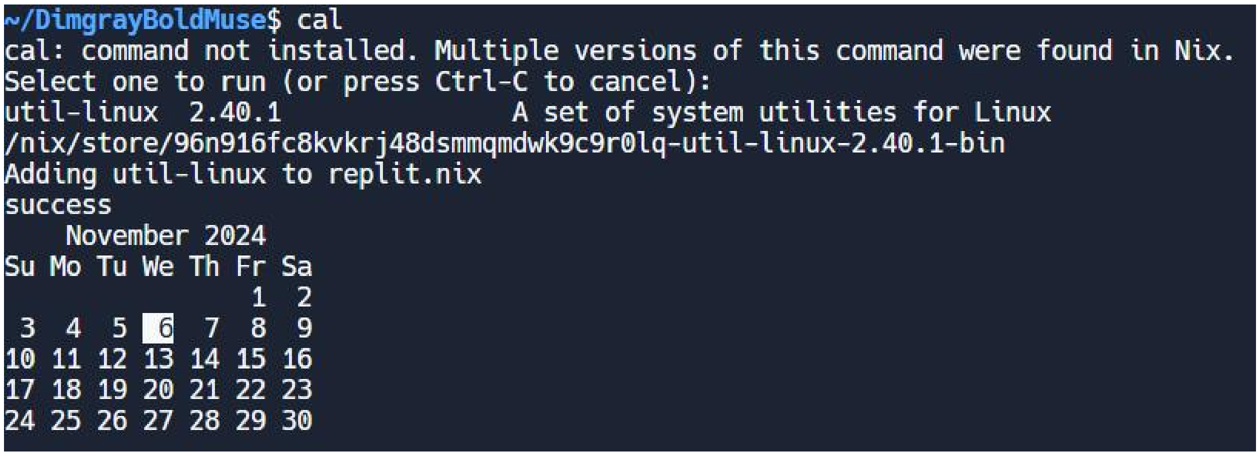
Solution:-

1-date:- The date command displays the current date and time on the system.



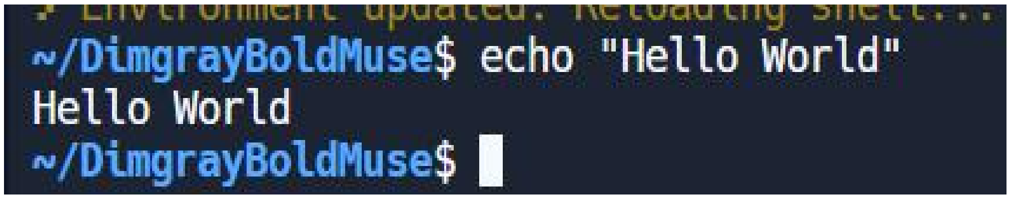
```
~/DimgrayBoldMuse$ date
Wed 06 Nov 2024 03:26:03 PM UTC
```

2-cal:- The cal command displays the calendar for the current month by default.



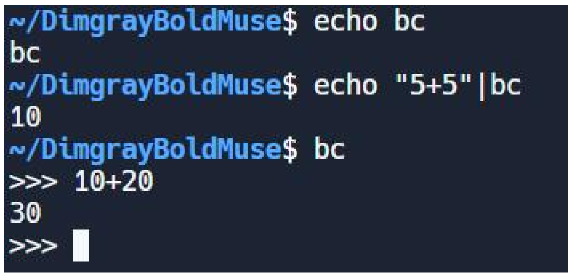
```
~/DimgrayBoldMuse$ cal
cal: command not installed. Multiple versions of this command were found in Nix.
Select one to run (or press Ctrl-C to cancel):
util-linux  2.40.1              A set of system utilities for Linux
/nix/store/96n916fc8kvkrj48dsmmqmdwk9c9r0lq-util-linux-2.40.1-bin
Adding util-linux to replit.nix
success
    November 2024
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
```

3-echo:- The echo command is used to print text to the terminal or to a file.



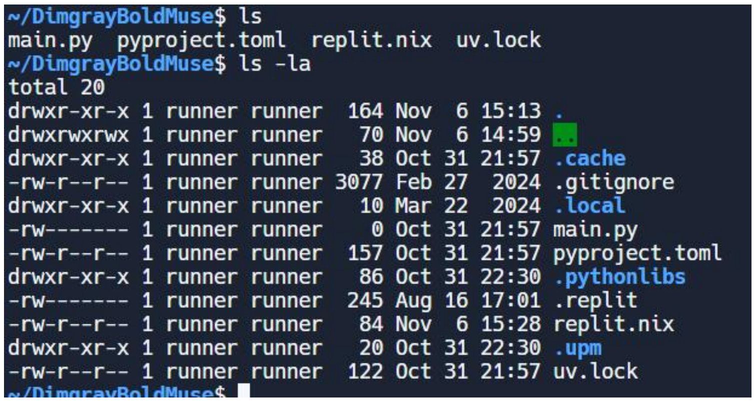
```
~/DimgrayBoldMuse$ echo "Hello World"
Hello World
~/DimgrayBoldMuse$
```

4-bc:- The bc command is a basic calculator that allows you to perform arithmetic operations in the terminal.



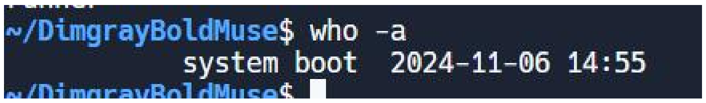
```
~/DimgrayBoldMuse$ echo bc
bc
~/DimgrayBoldMuse$ echo "5+5"|bc
10
~/DimgrayBoldMuse$ bc
>>> 10+20
30
>>>
```

5-ls:- The ls command lists the files and directories in the current directory.



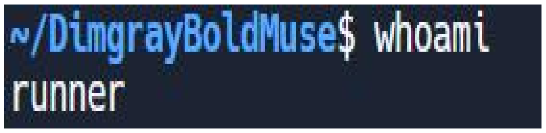
```
~/DimgrayBoldMuse$ ls
main.py  pyproject.toml  replit.nix  uv.lock
~/DimgrayBoldMuse$ ls -la
total 20
drwxr-xr-x 1 runner runner  164 Nov  6 15:13 .
drwxrwxrwx 1 runner runner   70 Nov  6 14:59 ..
drwxr-xr-x 1 runner runner   38 Oct 31 21:57 .cache
-rw-r--r-- 1 runner runner 3077 Feb 27  2024 .gitignore
drwxr-xr-x 1 runner runner   10 Mar 22  2024 .local
-rw------- 1 runner runner    0 Oct 31 21:57 main.py
-rw-r--r-- 1 runner runner  157 Oct 31 21:57 pyproject.toml
drwxr-xr-x 1 runner runner   86 Oct 31 22:30 .pythonlibs
-rw------- 1 runner runner  245 Aug 16 17:01 .replit
-rw-r--r-- 1 runner runner   84 Nov  6 15:28 replit.nix
drwxr-xr-x 1 runner runner   20 Oct 31 22:30 .upm
-rw-r--r-- 1 runner runner  122 Oct 31 21:57 uv.lock
~/DimgrayBoldMuse$
```

6-who:- The who command displays information about the users currently logged into the system.



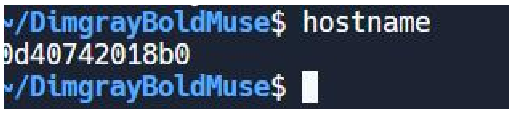
```
~/DimgrayBoldMuse$ who -a
           system boot  2024-11-06 14:55
~/DimgrayBoldMuse$
```

7-whoami :- The whoami command displays the username of the currently logged-in user.



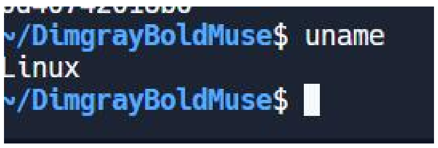
```
~/DimgrayBoldMuse$ whoami
runner
```

8- hostname :- The hostname command displays or sets the system's hostname.



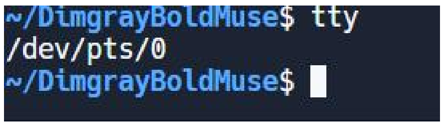
```
~/DimgrayBoldMuse$ hostname
0d40742018b0
~/DimgrayBoldMuse$ █
```

9- uname :- The uname command provides system information, such as the operating system name, kernel version, etc.



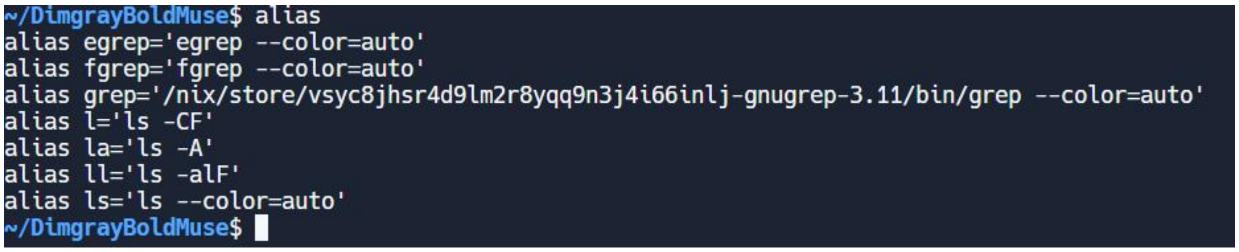
```
~/DimgrayBoldMuse$ uname
Linux
~/DimgrayBoldMuse$ █
```

10- tty :- The tty command displays the terminal name that you're using.



```
~/DimgrayBoldMuse$ tty
/dev/pts/0
~/DimgrayBoldMuse$ █
```

11- alias :- The alias command allows you to create shortcuts for longer commands.

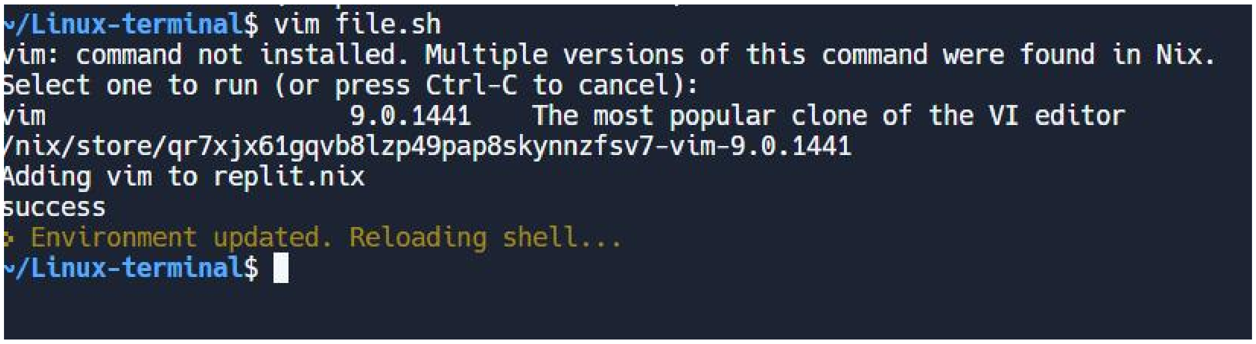


```
~/DimgrayBoldMuse$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='/nix/store/vsyc8jhsr4d9lm2r8yqq9n3j4i66inlj-gnugrep-3.11/bin/grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
~/DimgrayBoldMuse$ █
```

Program 5:- To understand vi basics, Three modes of vi Editor, how to write, save, execute a shell script in vi editor.

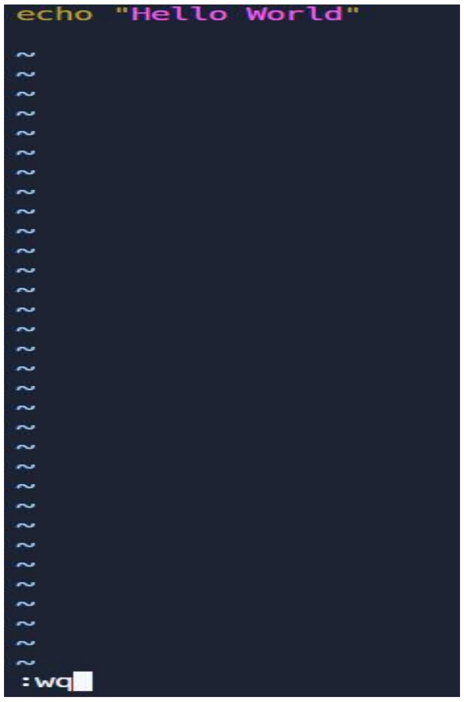
Solution:-

Create a file :-



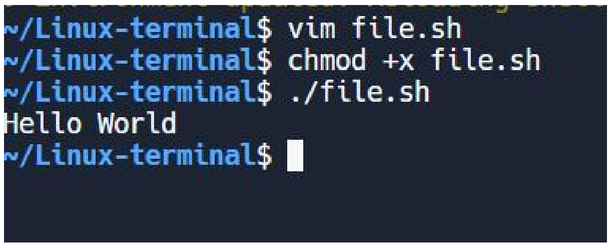
```
~/Linux-terminal$ vim file.sh
vim: command not installed. Multiple versions of this command were found in Nix.
Select one to run (or press Ctrl-C to cancel):
vim                    9.0.1441    The most popular clone of the VI editor
/nix/store/qr7xjx61gqvb8lzp49pap8skynnzfsv7-vim-9.0.1441
Adding vim to replit.nix
success
 Environment updated. Reloading shell...
~/Linux-terminal$
```

Write a file and save it :-



```
echo "Hello World"

~
~
~
:wq
```

Execute a shell script:-

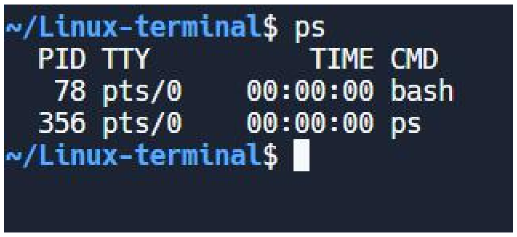


```
~/Linux-terminal$ vim file.sh
~/Linux-terminal$ chmod +x file.sh
~/Linux-terminal$ ./file.sh
Hello World
~/Linux-terminal$ 
```

Program 6:- To understand process related commands like: -ps, top, pstree, nice, renice in Linux.

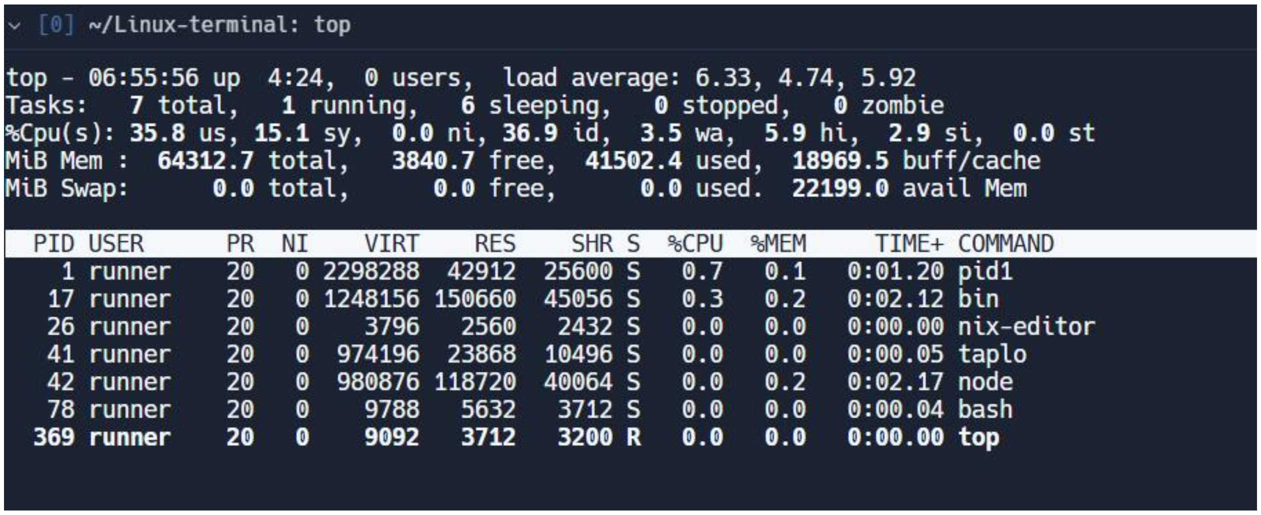
Solution :-

1-ps:- The ps command is used to display information about active processes running on the system.



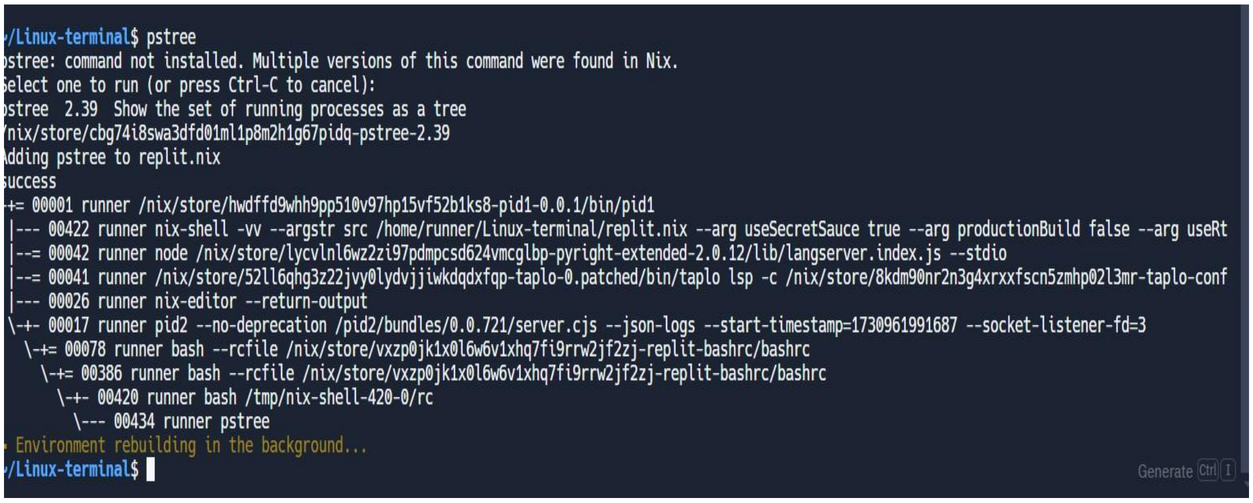
```
~/Linux-terminal$ ps
  PID TTY          TIME CMD
   78 pts/0    00:00:00 bash
  356 pts/0    00:00:00 ps
~/Linux-terminal$
```

2-top:- The top command is a dynamic, real-time view of the system's processes. It shows a summary of system information and updates the process list regularly.



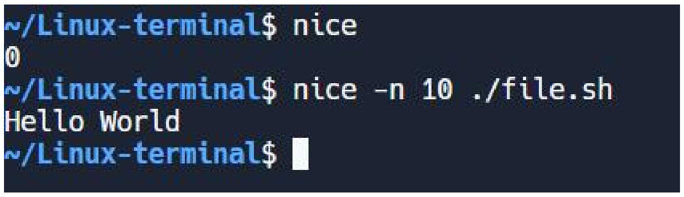
```
[0] ~/Linux-terminal: top

top - 06:55:56 up  4:24,  0 users,  load average: 6.33, 4.74, 5.92
Tasks:   7 total,   1 running,   6 sleeping,   0 stopped,   0 zombie
%Cpu(s): 35.8 us, 15.1 sy,  0.0 ni, 36.9 id,  3.5 wa,  5.9 hi,  2.9 si,  0.0 st
MiB Mem :  64312.7 total,   3840.7 free,  41502.4 used,  18969.5 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  22199.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 runner    20   0 2298288  42912  25600 S   0.7   0.1   0:01.20 pid1
   17 runner    20   0 1248156 150660  45056 S   0.3   0.2   0:02.12 bin
   26 runner    20   0    3796   2560   2432 S   0.0   0.0   0:00.00 nix-editor
   41 runner    20   0  974196  23868  10496 S   0.0   0.0   0:00.05 taplo
   42 runner    20   0  980876 118720  40064 S   0.0   0.2   0:02.17 node
   78 runner    20   0    9788   5632   3712 S   0.0   0.0   0:00.04 bash
  369 runner    20   0    9092   3712   3200 R   0.0   0.0   0:00.00 top
```

3-pstree:- The pstree command shows processes in a tree-like format, which illustrates how processes are related to one another (parent-child relationship).



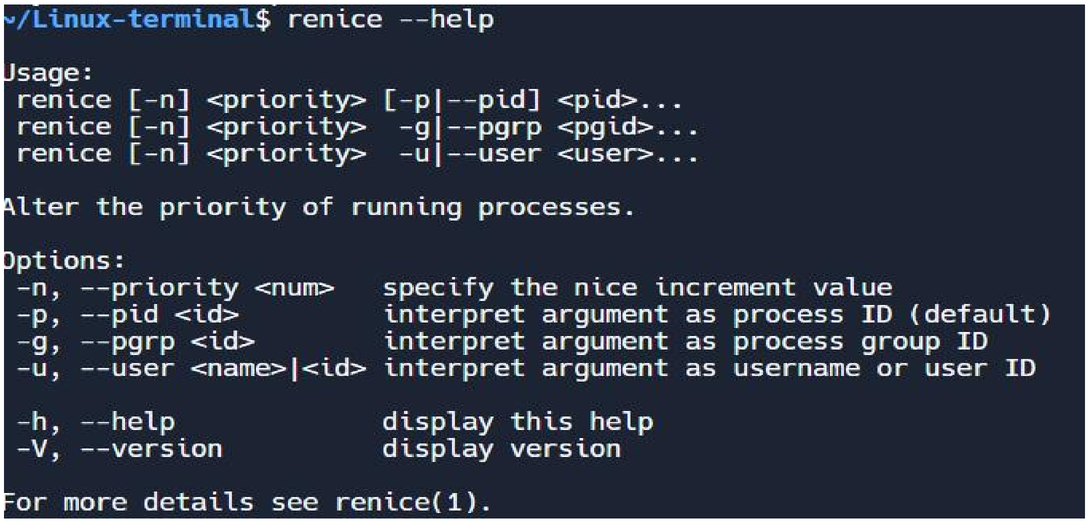
```
/Linux-terminal$ pstree
stree: command not installed. Multiple versions of this command were found in Nix.
elect one to run (or press Ctrl-C to cancel):
stree  2.39  Show the set of running processes as a tree
nix/store/cbg74i8swa3dfd01ml1p8m2h1g67pidq-pstree-2.39
dding pstree to replit.nix
uccess
+= 00001 runner /nix/store/hwdffd9whh9pp510v97hp15vf52b1ks8-pid1-0.0.1/bin/pid1
 |--- 00422 runner nix-shell -vv --argstr src /home/runner/Linux-terminal/replit.nix --arg useSecretSauce true --arg productionBuild false --arg useRt
 |--= 00042 runner node /nix/store/lycvlnl6wz2zi97pdmpcsd624vmcglbp-pyright-extended-2.0.12/lib/langserver.index.js --stdio
 |--= 00041 runner /nix/store/52ll6qhg3z22jvy0lydvjjiwkdqdxfqp-taplo-0.patched/bin/taplo lsp -c /nix/store/8kdm90nr2n3g4xrxxfscn5zmhp02l3mr-taplo-conf
 |--- 00026 runner nix-editor --return-output
 \-+- 00017 runner pid2 --no-deprecation /pid2/bundles/0.0.721/server.cjs --json-logs --start-timestamp=1730961991687 --socket-listener-fd=3
   \-+= 00078 runner bash --rcfile /nix/store/vxzp0jk1x0l6w6v1xhq7fi9rrw2jf2zj-replit-bashrc/bashrc
     \-+= 00386 runner bash --rcfile /nix/store/vxzp0jk1x0l6w6v1xhq7fi9rrw2jf2zj-replit-bashrc/bashrc
       \-+- 00420 runner bash /tmp/nix-shell-420-0/rc
         \--- 00434 runner pstree
 Environment rebuilding in the background...
/Linux-terminal$
```

4-nice:- The nice command is used to start a process with a specified priority (also known as "niceness"). The priority value affects how much CPU time the process will get.



```
~/Linux-terminal$ nice
0
~/Linux-terminal$ nice -n 10 ./file.sh
Hello World
~/Linux-terminal$
```

5-renice:- The renice command allows you to change the priority (niceness) of a running process.



```
~/Linux-terminal$ renice --help

Usage:
 renice [-n] <priority> [-p|--pid] <pid>...
 renice [-n] <priority>  -g|--pgrp <pgid>...
 renice [-n] <priority>  -u|--user <user>...

Alter the priority of running processes.

Options:
 -n, --priority <num>   specify the nice increment value
 -p, --pid <id>         interpret argument as process ID (default)
 -g, --pgrp <id>        interpret argument as process group ID
 -u, --user <name>|<id> interpret argument as username or user ID

 -h, --help             display this help
 -V, --version          display version

For more details see renice(1).
```

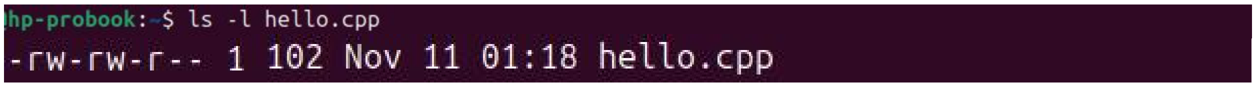
Program 7:- To understand how to examine and change File permissions.

Solution:-

In Linux, file permissions are crucial for security and access control. Each file and directory has specific permissions that determine who can read, write, or execute it. Here's a guide to examining and changing these permissions:

Step 1: Understanding File Permissions

Permissions are displayed with the `ls -l` command :



```
hp-probook:~$ ls -l hello.cpp
-rw-rw-r-- 1 102 Nov 11 01:18 hello.cpp
```

This output can be broken down as follows:

- The first character (−) indicates the type of file:
    - − for a regular file
    - `d` for a directory
    - `l` for a symbolic link
- The next nine characters are divided into three sets, each representing the permissions for the owner, group, and others:
    - `r`: read permission
    - `w`: write permission
    - `x`: execute permission
    - `−`: no permission

Example Breakdown:

- `rwx`: Owner has read, write, and execute permissions.
- `r−x`: Group has read and execute permissions (no write).
- `r−−`: Others have read-only permission.

Step 2: Changing File Permissions with `chmod`

The `chmod` command is used to modify permissions, either with symbolic (letters) or numeric (octal) modes.

1. Using Symbolic Mode

In symbolic mode, permissions are modified by specifying:

- User (u), Group (g), Others (o), or All (a)

- Add (+), Remove (-), or Set (=) permissions
- Permission types: Read (r), Write (w), Execute (x)

2. Using Numeric (Octal) Mode

In numeric mode, permissions are represented by a three-digit number, with each digit representing the permissions for owner, group, and others respectively.

Permission Values:

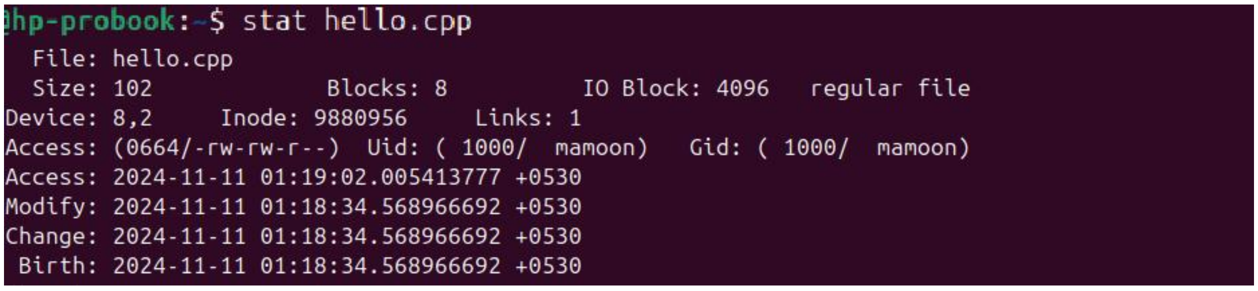
- `4` = Read (`r`)
- `2` = Write (`w`)
- `1` = Execute (`x`)
- Add these values to set multiple permissions (e.g., `7` = `4+2+1` = `rwx`).

Full Permission Examples:

- `chmod 777 filename`: All users (owner, group, others) have full permissions (`rwxrwxrwx`).
- `chmod 600 filename`: Only the owner has read and write (`rw`-------------).

Step 3: Viewing Permissions with `stat`

For a more detailed view of permissions and other file properties, use:



```
hp-probook:~$ stat hello.cpp
  File: hello.cpp
  Size: 102             Blocks: 8          IO Block: 4096   regular file
Device: 8,2     Inode: 9880956     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  mamoon)   Gid: ( 1000/  mamoon)
Access: 2024-11-11 01:19:02.005413777 +0530
Modify: 2024-11-11 01:18:34.568966692 +0530
Change: 2024-11-11 01:18:34.568966692 +0530
 Birth: 2024-11-11 01:18:34.568966692 +0530
```

Step 4: Changing File Ownership with `chown`

The `chown` command changes the owner and group of a file or directory.

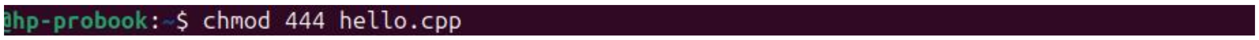
```
Ex:- chown owner:group filename
```

Program 8:- Set a file to be read-only with the chmod command. Interpret the file permissions.

Solution:-

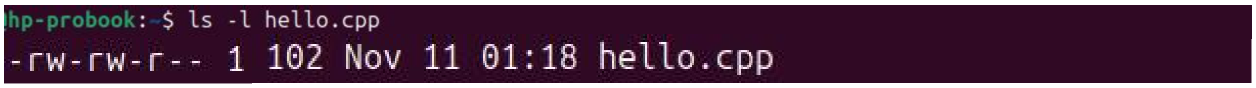
Step 1: Setting a File to Read-Only

To make a file read-only, set its permissions to `444`. This removes write and execute permissions for everyone (owner, group, and others).



```
@hp-probook:~$ chmod 444 hello.cpp
```

Step 2: Interpreting the File Permissions:



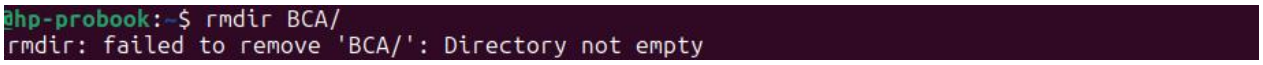
```
hp-probook:~$ ls -l hello.cpp
-rw-rw-r-- 1 102 Nov 11 01:18 hello.cpp
```

Program 9:- Delete one or more directories with the rmdir command. See what happens if the directory is not empty. Experiment (carefully!) with the rm -r command to delete a directory and its content.

Solution:-

Using `rmdir` to Delete Directories

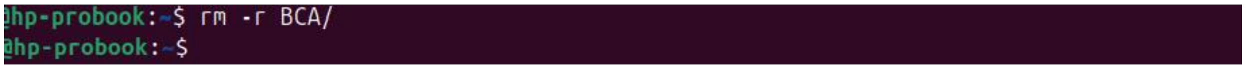
The `rmdir` command removes empty directories only. If the directory is not empty, `rmdir` will not work and will return an error.



```
@hp-probook:~$ rmdir BCA/
rmdir: failed to remove 'BCA/': Directory not empty
```

Deleting a Directory and Its Contents with `rm -r`

The `rm -r` (recursive) command removes a directory **and all its contents**, including subdirectories and files. This is helpful for deleting non-empty directories but should be used carefully, as deleted files cannot be recovered.

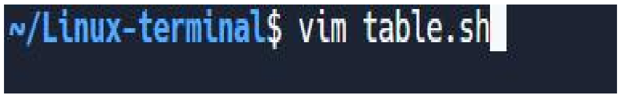


```
@hp-probook:~$ rm -r BCA/
@hp-probook:~$
```

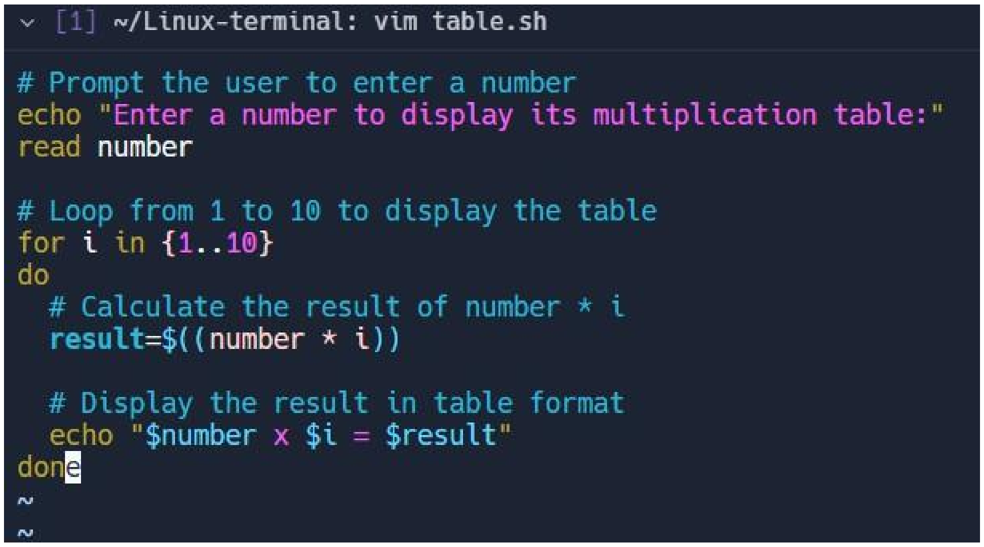
Program 10:- Write basic shell script to display the table of a number.

Solution :-

1. Save the script to a file, e.g., table.sh.



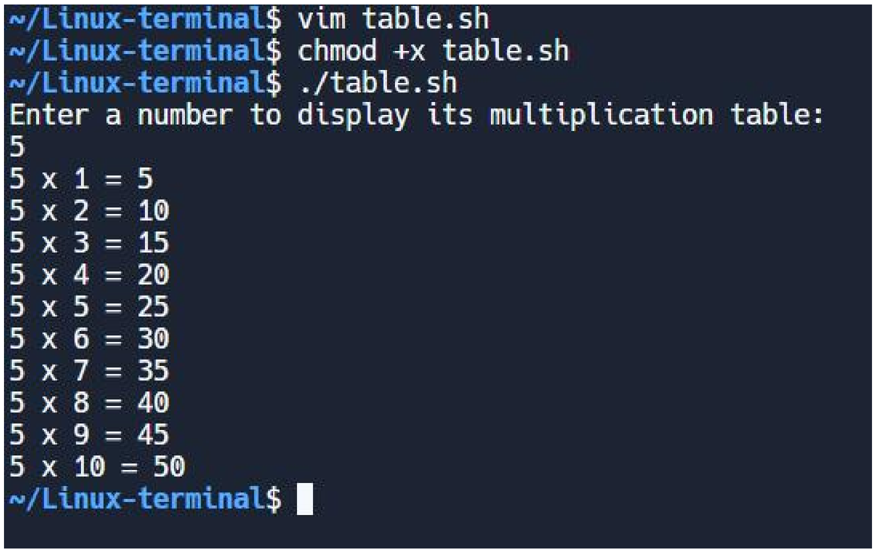
```
~/Linux-terminal$ vim table.sh
```



```
⌄ [1] ~/Linux-terminal: vim table.sh

# Prompt the user to enter a number
echo "Enter a number to display its multiplication table:"
read number

# Loop from 1 to 10 to display the table
for i in {1..10}
do
  # Calculate the result of number * i
  result=$((number * i))

  # Display the result in table format
  echo "$number x $i = $result"
done
~
~
```

2. Make the script executable

Output:-

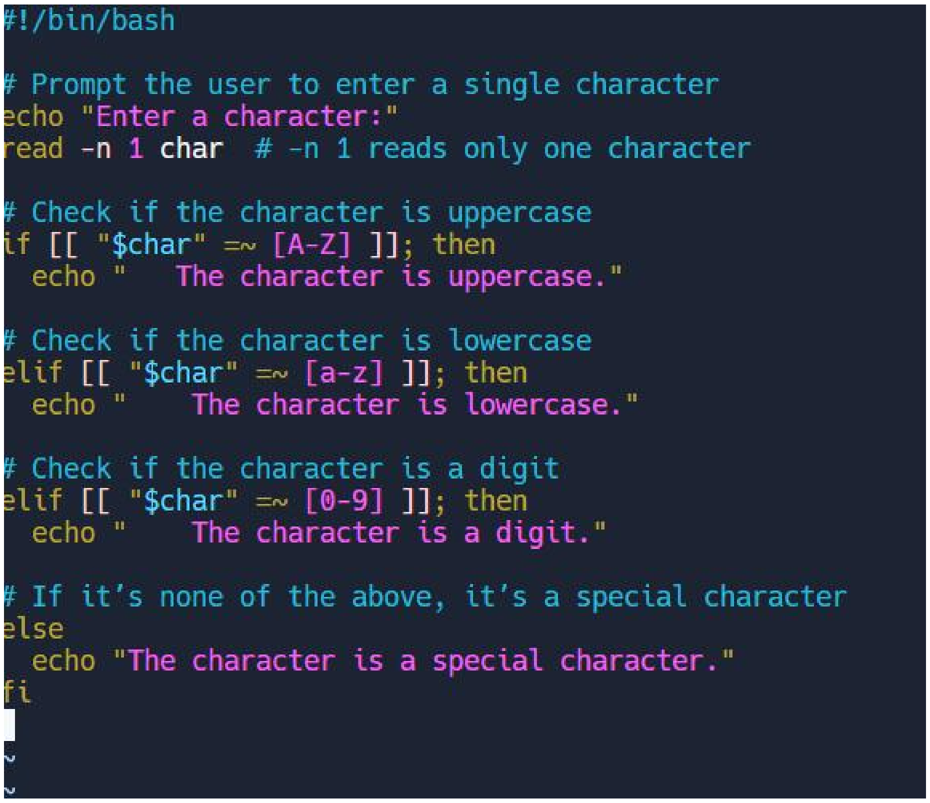


```
~/Linux-terminal$ vim table.sh
~/Linux-terminal$ chmod +x table.sh
~/Linux-terminal$ ./table.sh
Enter a number to display its multiplication table:
5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
~/Linux-terminal$
```

Program 11:- Write basic shell script to input a character from user and then check whether it is uppercase, lowercase or digit.

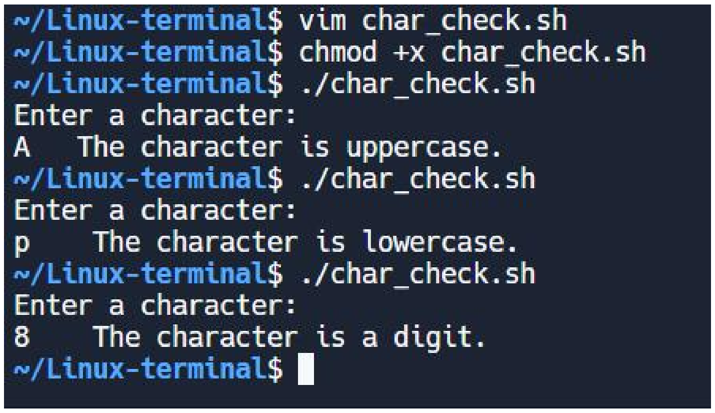
Solution:-

Create a file :- vim char_check.sh



```
#!/bin/bash

# Prompt the user to enter a single character
echo "Enter a character:"
read -n 1 char  # -n 1 reads only one character

# Check if the character is uppercase
if [[ "$char" =~ [A-Z] ]]; then
  echo "   The character is uppercase."

# Check if the character is lowercase
elif [[ "$char" =~ [a-z] ]]; then
  echo "    The character is lowercase."

# Check if the character is a digit
elif [[ "$char" =~ [0-9] ]]; then
  echo "    The character is a digit."

# If it’s none of the above, it’s a special character
else
  echo "The character is a special character."
fi

~
~
```

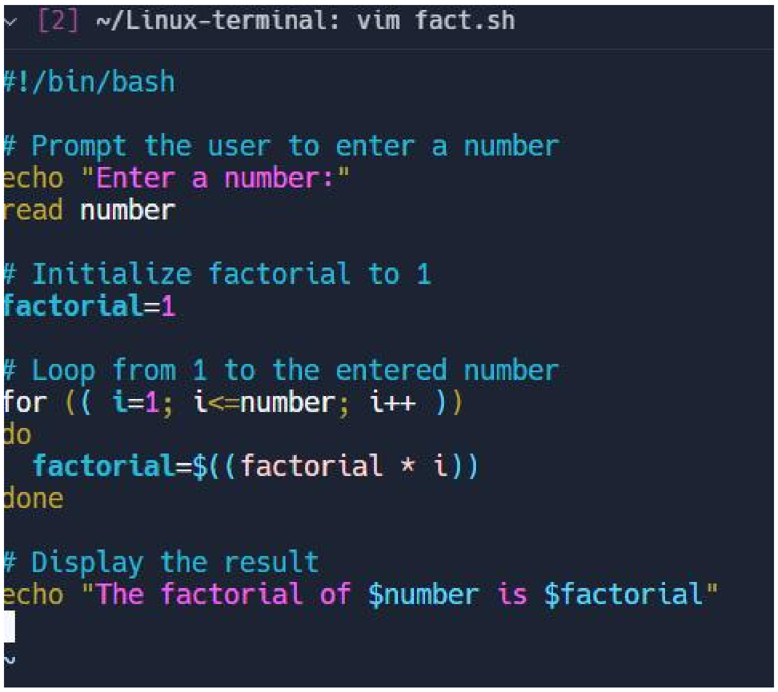
Output:-



```
~/Linux-terminal$ vim char_check.sh
~/Linux-terminal$ chmod +x char_check.sh
~/Linux-terminal$ ./char_check.sh
Enter a character:
A   The character is uppercase.
~/Linux-terminal$ ./char_check.sh
Enter a character:
p    The character is lowercase.
~/Linux-terminal$ ./char_check.sh
Enter a character:
8    The character is a digit.
~/Linux-terminal$
```

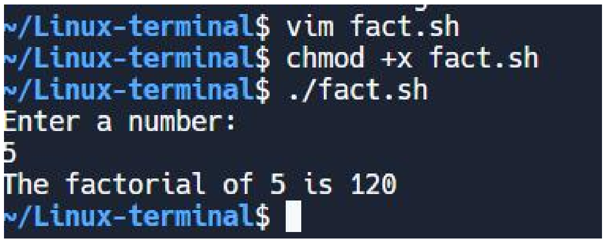
Program 12 :- Write basic shell script to calculate factorial of a number.

Solution:-



```
[2] ~/Linux-terminal: vim fact.sh
#!/bin/bash

# Prompt the user to enter a number
echo "Enter a number:"
read number

# Initialize factorial to 1
factorial=1

# Loop from 1 to the entered number
for (( i=1; i<=number; i++ ))
do
  factorial=$((factorial * i))
done

# Display the result
echo "The factorial of $number is $factorial"

~
```

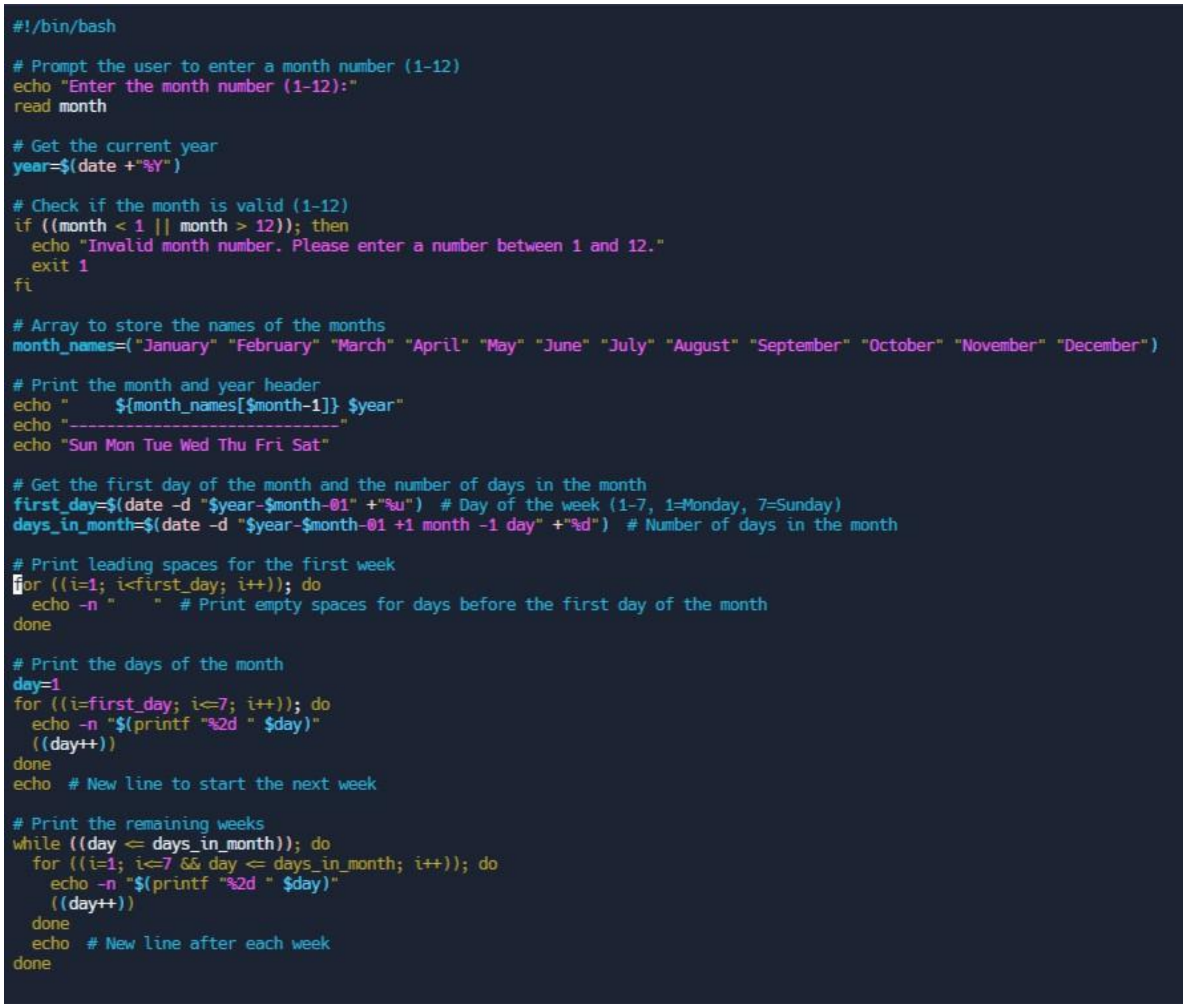
Output:-



```
~/Linux-terminal$ vim fact.sh
~/Linux-terminal$ chmod +x fact.sh
~/Linux-terminal$ ./fact.sh
Enter a number:
5
The factorial of 5 is 120
~/Linux-terminal$
```

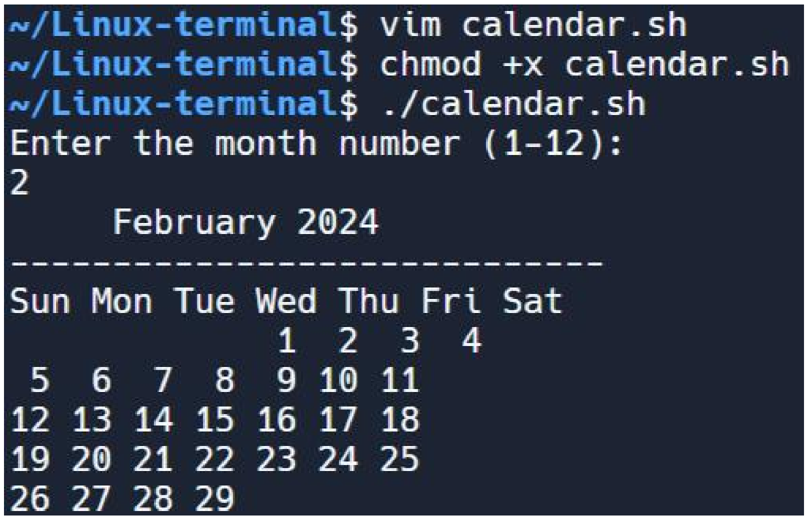
Program 13:- Write basic shell script to input the month number and generate corresponding calendar.

Solution :-



```bash
#!/bin/bash

# Prompt the user to enter a month number (1-12)
echo "Enter the month number (1-12):"
read month

# Get the current year
year=$(date +"%Y")

# Check if the month is valid (1-12)
if ((month < 1 || month > 12)); then
  echo "Invalid month number. Please enter a number between 1 and 12."
  exit 1
fi

# Array to store the names of the months
month_names=("January" "February" "March" "April" "May" "June" "July" "August" "September" "October" "November" "December")

# Print the month and year header
echo "     ${month_names[$month-1]} $year"
echo "-----------------------------"
echo "Sun Mon Tue Wed Thu Fri Sat"

# Get the first day of the month and the number of days in the month
first_day=$(date -d "$year-$month-01" +"%u")  # Day of the week (1-7, 1=Monday, 7=Sunday)
days_in_month=$(date -d "$year-$month-01 +1 month -1 day" +"%d")  # Number of days in the month

# Print leading spaces for the first week
for ((i=1; i<first_day; i++)); do
  echo -n "    "  # Print empty spaces for days before the first day of the month
done

# Print the days of the month
day=1
for ((i=first_day; i<=7; i++)); do
  echo -n "$(printf "%2d " $day)"
  ((day++))
done
echo  # New line to start the next week

# Print the remaining weeks
while ((day <= days_in_month)); do
  for ((i=1; i<=7 && day <= days_in_month; i++)); do
    echo -n "$(printf "%2d " $day)"
    ((day++))
  done
  echo  # New line after each week
done
```

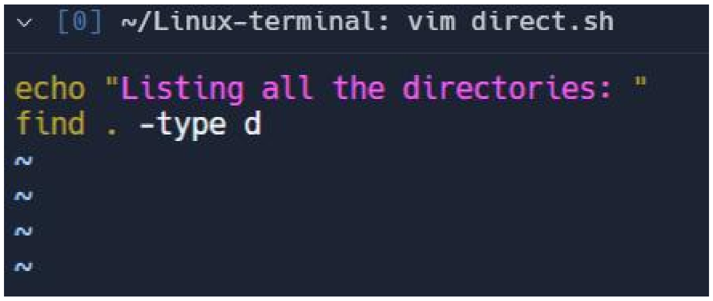
Output:-



```
~/Linux-terminal$ vim calendar.sh
~/Linux-terminal$ chmod +x calendar.sh
~/Linux-terminal$ ./calendar.sh
Enter the month number (1-12):
2
     February 2024
-----------------------------
Sun Mon Tue Wed Thu Fri Sat
             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
```

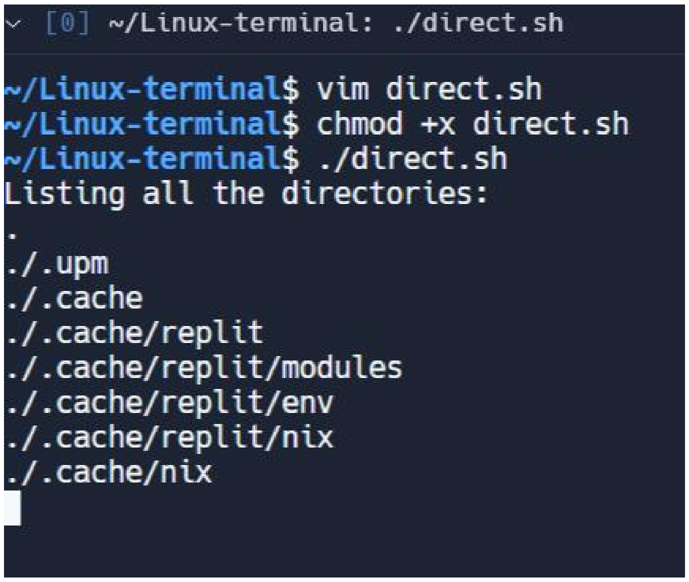
Program 14:- Write basic shell script to list all directories.

Solution:-



```
⌄ [0] ~/Linux-terminal: vim direct.sh

echo "Listing all the directories: "
find . -type d
~
~
~
~
```

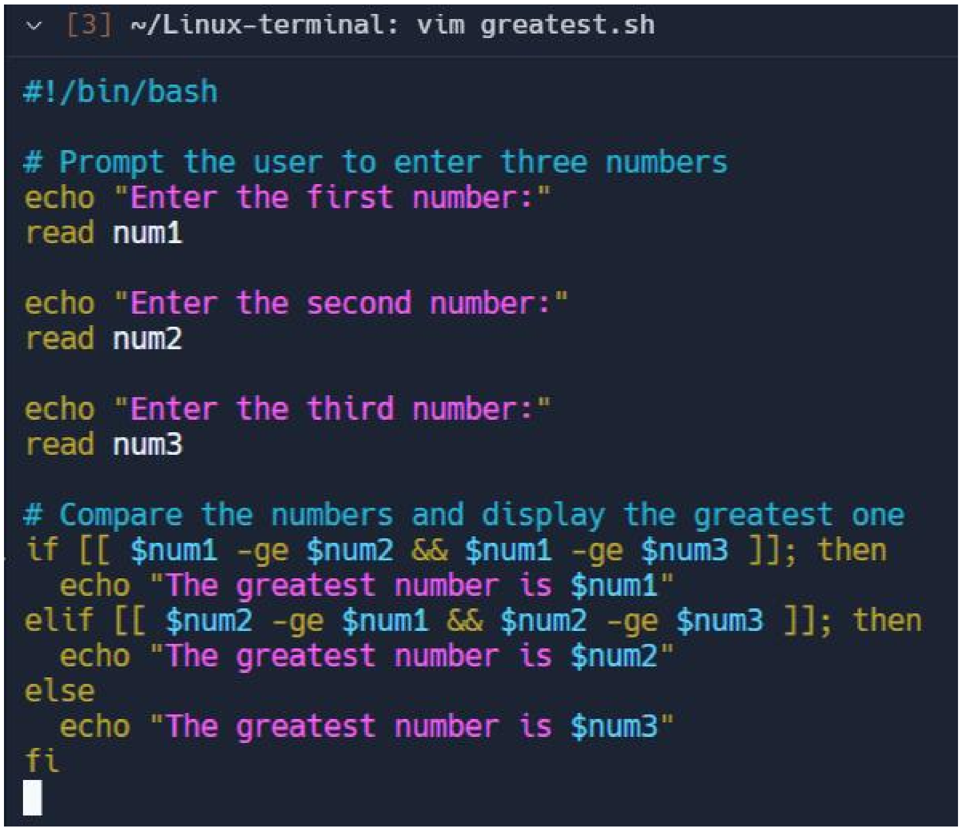
Output:-



```
⌄ [0] ~/Linux-terminal: ./direct.sh

~/Linux-terminal$ vim direct.sh
~/Linux-terminal$ chmod +x direct.sh
~/Linux-terminal$ ./direct.sh
Listing all the directories:
.
./.upm
./.cache
./.cache/replit
./.cache/replit/modules
./.cache/replit/env
./.cache/replit/nix
./.cache/nix
```

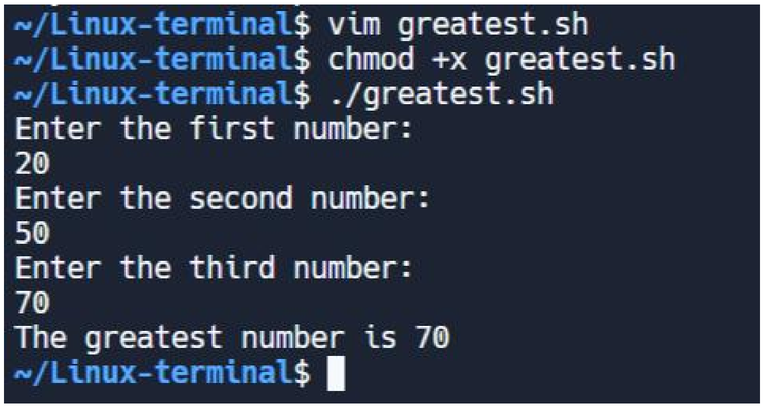
Program 15:- Write basic shell script to display greatest of three numbers.

Solution:-



```
⌄ [3] ~/Linux-terminal: vim greatest.sh

#!/bin/bash

# Prompt the user to enter three numbers
echo "Enter the first number:"
read num1

echo "Enter the second number:"
read num2

echo "Enter the third number:"
read num3

# Compare the numbers and display the greatest one
if [[ $num1 -ge $num2 && $num1 -ge $num3 ]]; then
  echo "The greatest number is $num1"
elif [[ $num2 -ge $num1 && $num2 -ge $num3 ]]; then
  echo "The greatest number is $num2"
else
  echo "The greatest number is $num3"
fi
```

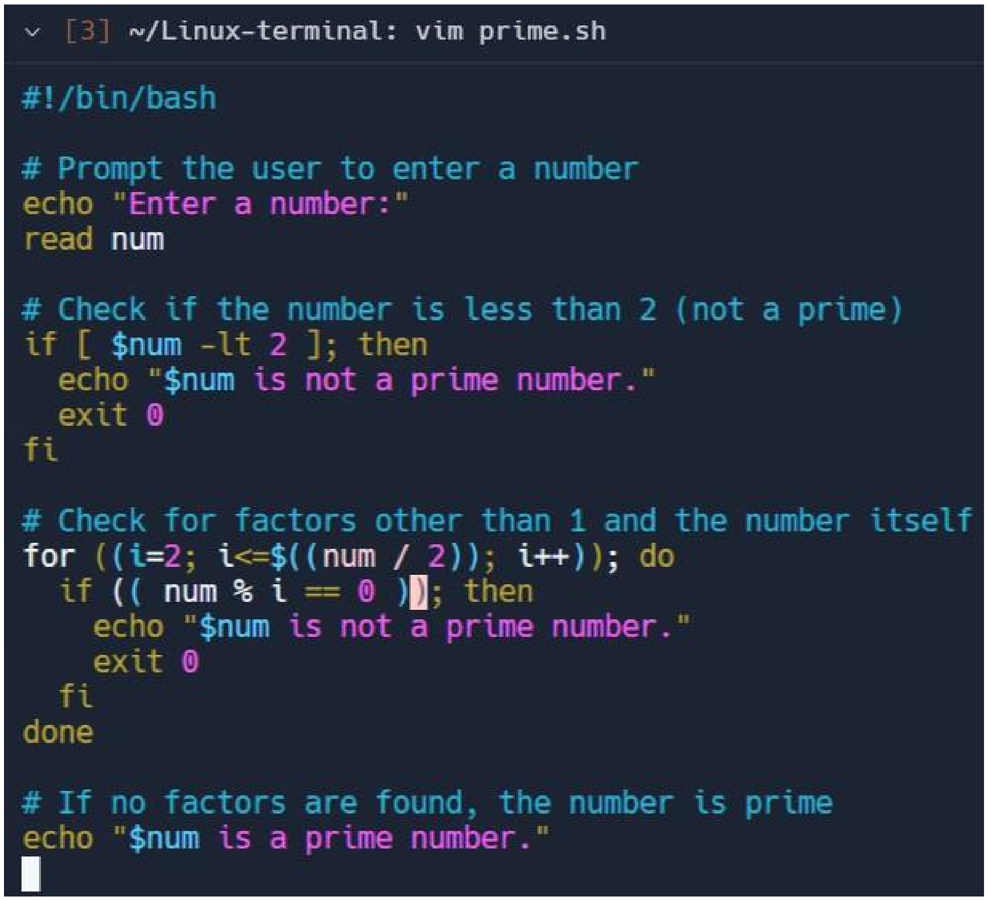
Output:-



```
~/Linux-terminal$ vim greatest.sh
~/Linux-terminal$ chmod +x greatest.sh
~/Linux-terminal$ ./greatest.sh
Enter the first number:
20
Enter the second number:
50
Enter the third number:
70
The greatest number is 70
~/Linux-terminal$
```

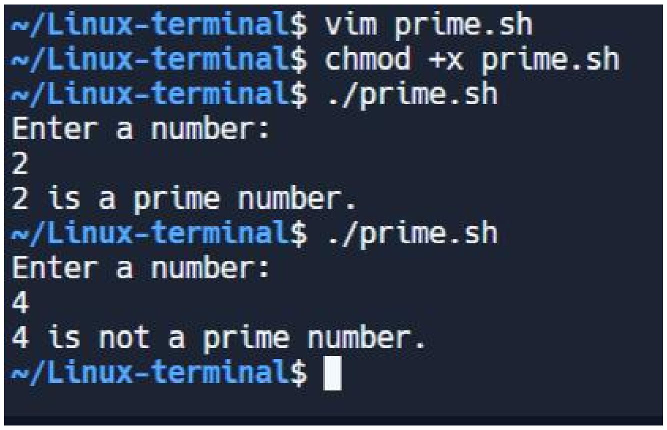
Program 16:- Write basic shell script to check whether the number entered by user is prime or not.

Solution:-



```
⌄ [3] ~/Linux-terminal: vim prime.sh

#!/bin/bash

# Prompt the user to enter a number
echo "Enter a number:"
read num

# Check if the number is less than 2 (not a prime)
if [ $num -lt 2 ]; then
  echo "$num is not a prime number."
  exit 0
fi

# Check for factors other than 1 and the number itself
for ((i=2; i<=$((num / 2)); i++)); do
  if (( num % i == 0 )); then
    echo "$num is not a prime number."
    exit 0
  fi
done

# If no factors are found, the number is prime
echo "$num is a prime number."
```

Output:-



```
~/Linux-terminal$ vim prime.sh
~/Linux-terminal$ chmod +x prime.sh
~/Linux-terminal$ ./prime.sh
Enter a number:
2
2 is a prime number.
~/Linux-terminal$ ./prime.sh
Enter a number:
4
4 is not a prime number.
~/Linux-terminal$
```