Department of Computer Science & Information Technology

III Year,V Semester(Batch 2022-2026)

Lab Record Submission

of

Linux Lab

(Subject Code – CSIT-505)

Submitted to:

Prof. Nidhi Nigam

Submitted by:

Ansh Yadav

Acropolis Institute of Technology and Research, Indore.

S.No.	Name of Experiment	Page No.	Date of Experiment	Date of Submission	Faculty Sign
01	Introduction of OS, function services of OS.Need for linux, History of linux and different services of linux.	**2-10**	**09/09/24**	**23/09/2024**	
02	Study of basic user abd shell command	**11-19**	**16/09/2024**	**23/06/2024**	
03	Study of linux architecture with different type of kernel shell.	**20-21**	**23/09/24**	**/11 /2024**	
04	Understanding linux file system	**22-24**	**23/09/24**	**/11 /2024**	
05	Study of linux editor with their command specification.	**25-34**	**7/10/2024**	**/11/2024**	
06	Create a file called wlcc.txt with some line and display how many lines words and characters are present in the file.	**35**	**10/10/24**	**/11/24**	
07	Append ten more simple line to the file ‘wlcc’ and display	**36**	**14/10/24**	**/11/24**	
08	Study and use of commands and for changing file permission	**37-40**	**21/10/24**	**/11/24**	
09	Write down shell script for following : 1. Write a shell script to print “Hello Learner”. 2. Write a bash script program to take run-time argument from the user as a name and print “Greetings <Your name>”. 3. Demonstrate the shell script with different usage of variables. 4. Write a shell script to perform arithmetic operations. 5. Write a shell script to take as an argument from the user and simply compare them. 6. Write a shell script to check whether a given input is even/odd. 7. Take a year as input and check if it's a leap year or not. 8. Check whether a no. is prime or not. 9. To find the area and circumference of a circle. 10. To check the given number and its reverse are same. 11.	**41-48**	**21/10/2024**	**/11/24**	

Experiment No.1

Introduction of operating System:

An operating system (OS) is essential for managing and facilitating interactions between a computer's hardware and software. It provides a structured environment for executing programs and handling tasks like file management. In the context of a file experiment, the OS plays a crucial role by allowing users to create, modify, and delete files, and by organizing these files into directories or folders. It also manages file permissions, which control access rights and ensure that only authorized users can perform certain actions on files. Additionally, the OS employs a file system, such as NTFS, FAT32, or ext4, to systematically organize and store files on storage devices. During your experiment, you will observe how the OS efficiently handles these file operations and maintains an organized structure for data management.

Function of Operating System:

Certainly! Here are the key functions of an operating system relevant to a file experiment:

1. **File Creation and Deletion**: Allows you to create new files and remove existing ones.
2. **File Organization**: Structures files in directories or folders for easy access and management.
3. **File Permissions**: Controls access rights, specifying who can read, write, or execute files.
4. **File System Management**: Uses a file system (like NTFS, FAT32, or ext4) to organize and store files on storage devices.
5. **File Access and Retrieval**: Manages how files are accessed and retrieved from storage efficiently.

Services of Operating System:

Here are the key services provided by an operating system:

1. **Process Management**: Handles the creation, scheduling, and termination of processes.
2. **Memory Management**: Allocates and manages the computer's memory resources.
3. **File Management**: Oversees file creation, deletion, and organization.
4. **Device Management**: Manages input and output devices, including drivers and interfaces.
5. **User Interface**: Provides a user interface (command-line or graphical) for interaction with the system.
6. **Security and Access Control**: Enforces user authentication and controls accessto system resources.

Need for linux operating system:

Linux operating system is widely used for several compelling reasons:

1. **Open Source**: Linux is open source, meaning its source code is freely available for anyone to view, modify, and distribute. This fosters innovation and customization.

2. Stability and Reliability: Known for its stability and reliability, Linux is often used in environments where uptime is critical, such as servers and embedded systems.
3. Security: Linux is designed with robust security features and has a strong community of developers who quickly address vulnerabilities and provide patches.
4. Cost-Effective: Linux is free to use, reducing costs associated with licensing fees compared to proprietary operating systems.
5. Flexibility and Customization: Linux can be tailored to suit specific needs, from lightweight distributions for older hardware to powerful configurations for advanced users and servers.
6. Performance: Linux typically has a smaller footprint and can be optimized for performance, making it suitable for a wide range of hardware.
7. Community Support: A vibrant community of users and developers provides extensive support and resources, including forums, documentation, and user guides.
8. Compatibility: Linux supports a wide range of hardware architectures and offers compatibility with various software applications, including many open-source tools and enterprise solutions.

History Of linux Operating System:

Linux originated in 1991 when Linus Torvalds, a Finnish student, released the first version of the Linux kernel as a free and open-source alternative to proprietary operating systems. Initially developed as a personal project, Linux quickly garnered support from a global community of developers. By adhering to the principles of open source, it allowed continuous improvements and adaptations. Over the years, Linux evolved into a robust and versatile operating system used across various domains, from personal computers to servers and embedded systems. Its development has been driven by contributions from both individual programmers and large organizations, leading to a diverse ecosystem of distributions tailored for different needs.

Different Services and Application Of Linux Operating system :

Here are some key services and applications of the Linux operating system:
1. Web Servers: Powers popular web servers like Apache and Nginx.
2. Database Management: Supports database systems such as MySQL, PostgreSQL, and MongoDB.
3. File Servers: Provides file sharing services with tools like Samba and NFS.
4. Network Management: Manages network services with tools like iptables and NetworkManager**.**
5. Development Environment: Used by developers for programming with support for various programming languages and development tools.
6. System Administration: Offers robust tools for system monitoring and management, such as top, htop, and systemd.
7. Security: Provides strong security features and tools, including SELinux and AppArmor.
8. Virtualization: Supports virtualization platforms like KVM, Docker, and VirtualBox.
9. Desktop Environments: Offers various desktop environments like GNOME, KDE, and XFCE for user interfaces.
10. Embedded Systems: Powers embedded systems and IoT devices due to its flexibility and low resource requirements.

Experiment No. 2

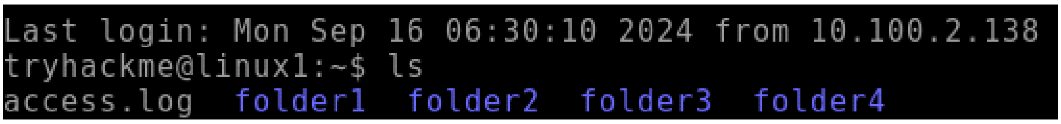
Aim :

To get the function of command of linux command\

1. **Name**: ls:- list directory contents

   Syntax : ls

   Description :

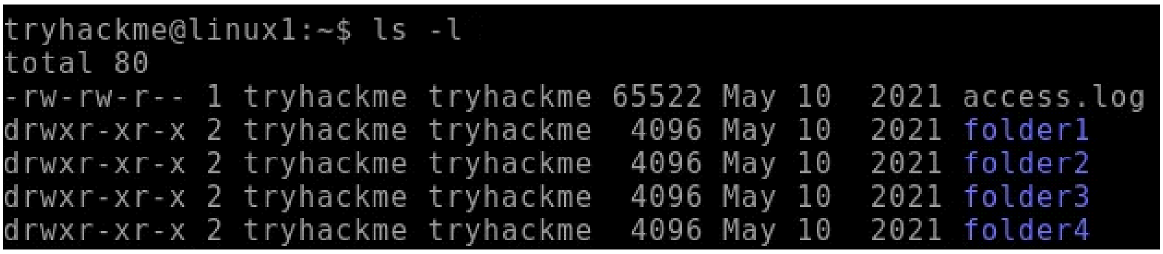


```
Last login: Mon Sep 16 06:30:10 2024 from 10.100.2.138
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4
```

2. **Name**: ls – l:- list directory contents

   Syntax : ls - l

   Description :

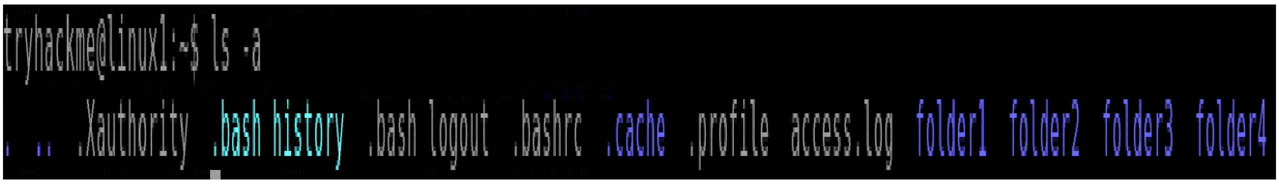


```
tryhackme@linux1:~$ ls -l
total 80
-rw-rw-r-- 1 tryhackme tryhackme 65522 May 10  2021 access.log
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder1
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder2
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder3
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder4
```

3. **Name**: ls – a:- list directory contents

   Syntax : ls - a

   Description :

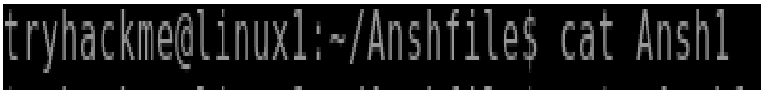


```
tryhackme@linux1:~$ ls -a
.  ..  .Xauthority  .bash_history  .bash_logout  .bashrc  .cache  .profile  access.log  folder1  folder2  folder3  folder4
```

4. **Name**: cat :- Concatenate files and print on the standard output

   Syntax : cat <filename>

   Description :

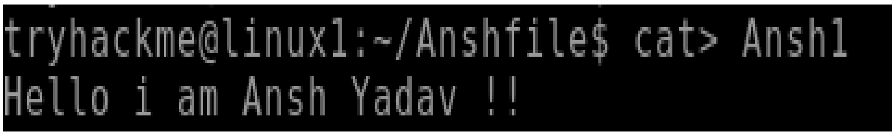


```
tryhackme@linux1:~/Anshfile$ cat Ansh1
```

5. **Name**: Cat > :- Write in a File

   Syntax : cat > filename

Description :

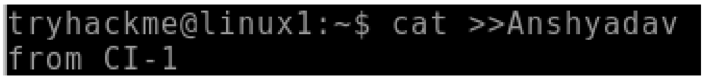


```
tryhackme@linux1:~/Anshfile$ cat> Ansh1
Hello i am Ansh Yadav !!
```

6. **Name**: Cat >> :- Add to already written file.

   Syntax : cat >> filename

   Description :

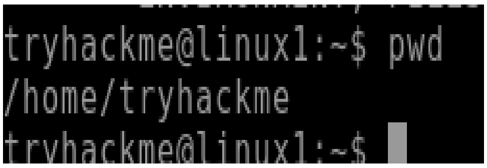


```
tryhackme@linux1:~$ cat >>Anshyadav
from CI-1
```

7. **Name**: pwd :- print name of current/working directory

   Syntax : pwd

   Description :

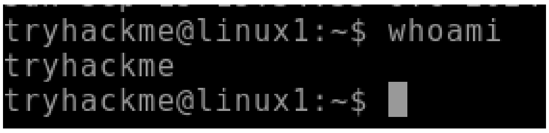


```
tryhackme@linux1:~$ pwd
/home/tryhackme
tryhackme@linux1:~$
```

8. **Name :** who am i :- displays username of current user interacting

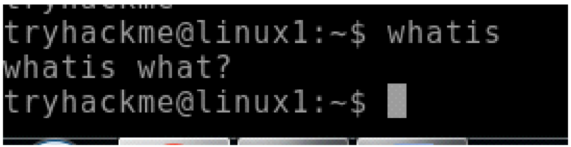
   Syntax : who am i

   Description :



```
tryhackme@linux1:~$ whoami
tryhackme
tryhackme@linux1:~$
```

9. **Name :** what is :- display one-line manual page descriptions
   Syntax : what is
   Description :

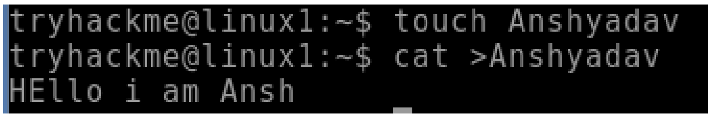


```
tryhackme@linux1:~$ whatis
whatis what?
tryhackme@linux1:~$
```

10. **Name :** Touch :- Create File

    Syntax : touch <file name>

Description :

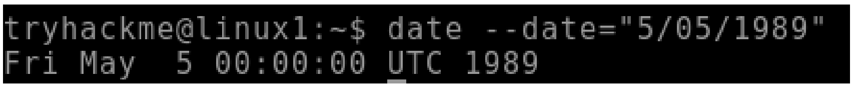


```
tryhackme@linux1:~$ touch Anshyadav
tryhackme@linux1:~$ cat >Anshyadav
HEllo i am Ansh
```

11. **Name : Date** :- date command is used to display the system date and time. date command is also used to set date and time of the system.

Syntax : date --date = "the date to be finded"

Description :

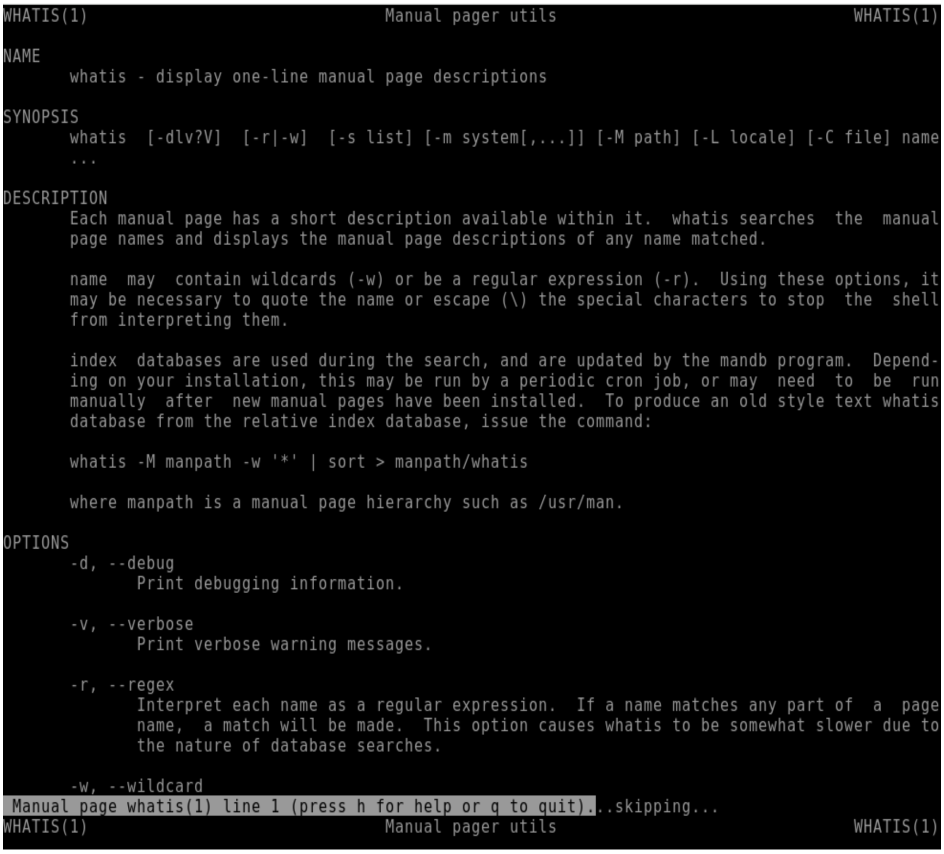


```
tryhackme@linux1:~$ date --date="5/05/1989"
Fri May  5 00:00:00 UTC 1989
```

12. Name : manual:- an interface to the system reference manuals

Syntax : man whatis

Description :

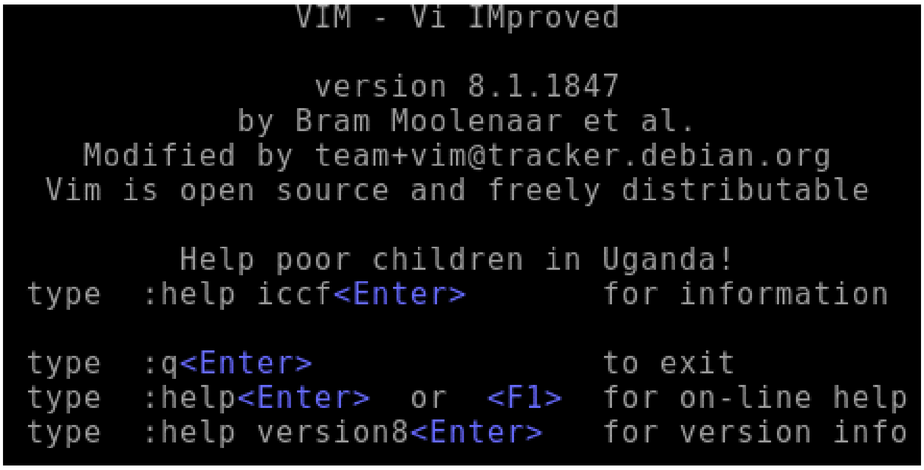


```
WHATIS(1)                      Manual pager utils                      WHATIS(1)

NAME
       whatis - display one-line manual page descriptions

SYNOPSIS
       whatis  [-dlv?V]  [-r|-w]  [-s list] [-m system[,...]] [-M path] [-L locale] [-C file] name
       ...

DESCRIPTION
       Each manual page has a short description available within it.  whatis searches  the  manual
       page names and displays the manual page descriptions of any name matched.

       name  may  contain wildcards (-w) or be a regular expression (-r).  Using these options, it
       may be necessary to quote the name or escape (\) the special characters to stop  the  shell
       from interpreting them.

       index  databases are used during the search, and are updated by the mandb program.  Depend-
       ing on your installation, this may be run by a periodic cron job, or may  need  to  be  run
       manually  after  new manual pages have been installed.  To produce an old style text whatis
       database from the relative index database, issue the command:

       whatis -M manpath -w '*' | sort > manpath/whatis

       where manpath is a manual page hierarchy such as /usr/man.

OPTIONS
       -d, --debug
              Print debugging information.

       -v, --verbose
              Print verbose warning messages.

       -r, --regex
              Interpret each name as a regular expression.  If a name matches any part of  a  page
              name,  a match will be made.  This option causes whatis to be somewhat slower due to
              the nature of database searches.

       -w, --wildcard
 Manual page whatis(1) line 1 (press h for help or q to quit)...skipping...
WHATIS(1)                      Manual pager utils                      WHATIS(1)
```

13. Name : Vi :-Vi IMproved, a programmer's text editor

Syntax : Vi

Description :

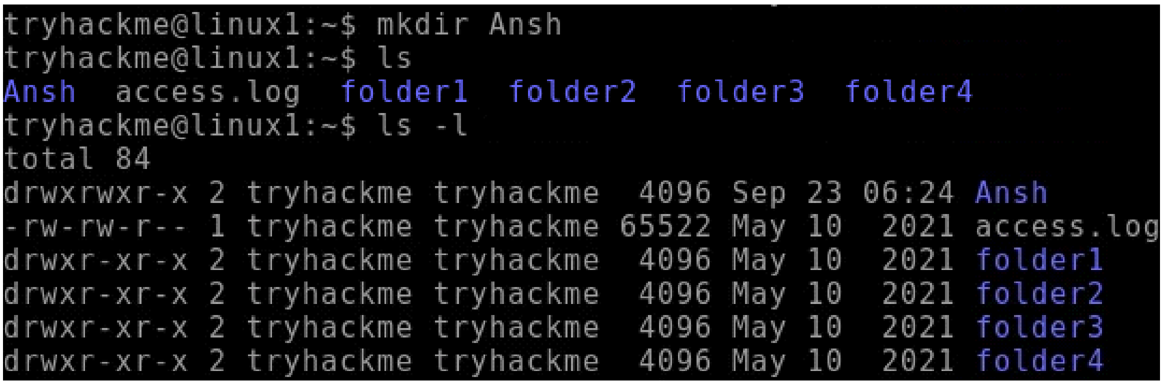


```
                VIM - Vi IMproved

                 version 8.1.1847
             by Bram Moolenaar et al.
   Modified by team+vim@tracker.debian.org
  Vim is open source and freely distributable

          Help poor children in Uganda!
 type  :help iccf<Enter>       for information

 type  :q<Enter>               to exit
 type  :help<Enter>  or  <F1>  for on-line help
 type  :help version8<Enter>   for version info
```

14. Name : Mkdir :- make a directory

Syntax : mkdir name

Description :

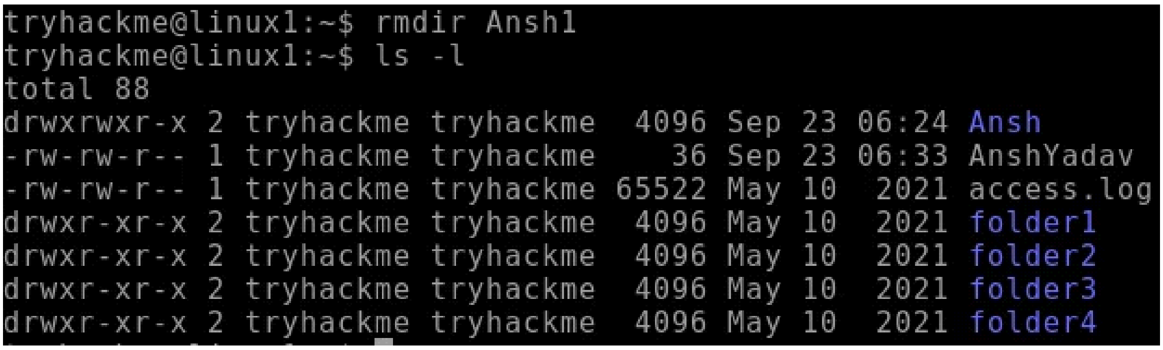


```
tryhackme@linux1:~$ mkdir Ansh
tryhackme@linux1:~$ ls
Ansh  access.log  folder1  folder2  folder3  folder4
tryhackme@linux1:~$ ls -l
total 84
drwxrwxr-x 2 tryhackme tryhackme  4096 Sep 23 06:24 Ansh
-rw-rw-r-- 1 tryhackme tryhackme 65522 May 10  2021 access.log
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder1
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder2
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder3
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder4
```

15. Name : rmdir :- remove a directory

Syntax : rmdir name

Description :

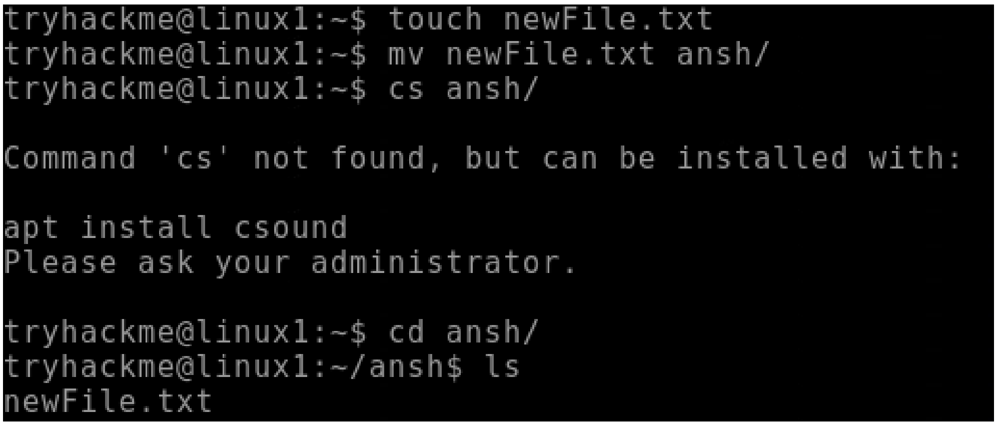


```
tryhackme@linux1:~$ rmdir Ansh1
tryhackme@linux1:~$ ls -l
total 88
drwxrwxr-x 2 tryhackme tryhackme  4096 Sep 23 06:24 Ansh
-rw-rw-r-- 1 tryhackme tryhackme    36 Sep 23 06:33 AnshYadav
-rw-rw-r-- 1 tryhackme tryhackme 65522 May 10  2021 access.log
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder1
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder2
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder3
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder4
```

16. Name : move :- move (rename) files

Syntax : mv <file name>.txt <directory name>/

Description :

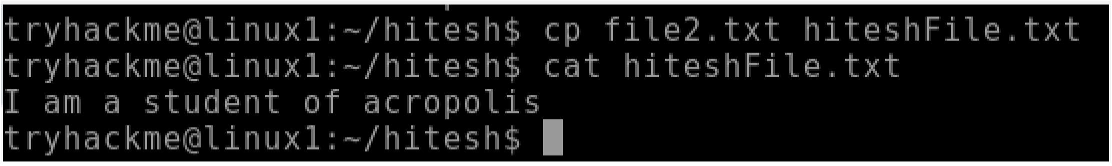


```
tryhackme@linux1:~$ touch newFile.txt
tryhackme@linux1:~$ mv newFile.txt ansh/
tryhackme@linux1:~$ cs ansh/

Command 'cs' not found, but can be installed with:

apt install csound
Please ask your administrator.

tryhackme@linux1:~$ cd ansh/
tryhackme@linux1:~/ansh$ ls
newFile.txt
```

17. Name : CP :- copy files and directories

Syntax : cp <file name>.txt <file name>.txt

Description :



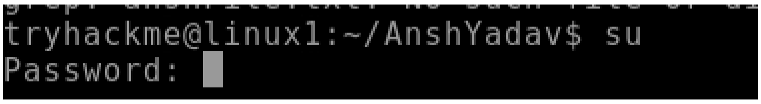
```
tryhackme@linux1:~/hitesh$ cp file2.txt hiteshFile.txt
tryhackme@linux1:~/hitesh$ cat hiteshFile.txt
I am a student of acropolis
tryhackme@linux1:~/hitesh$ 
```

18.

Name : SU :- run a command with substitute user and group ID

Syntax : su

Description :



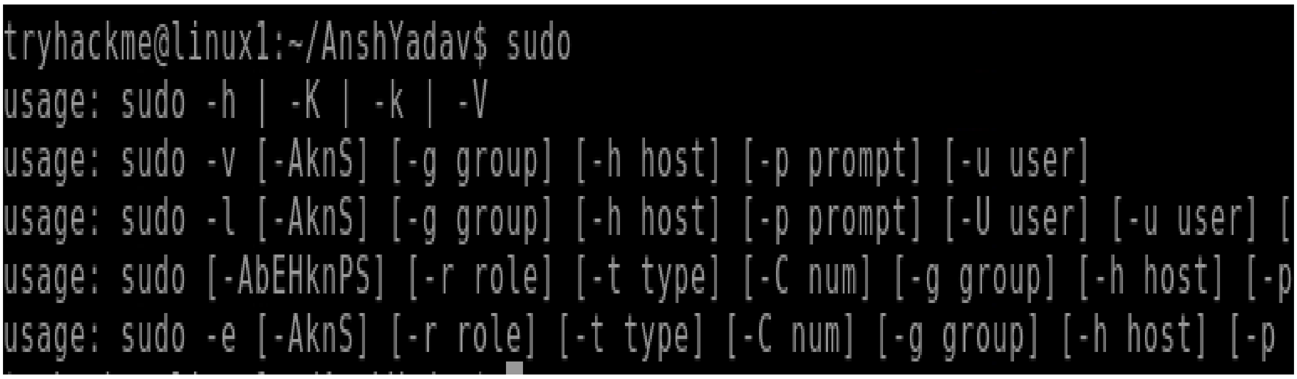
```
grep: anshFile.txt: No such file or di
tryhackme@linux1:~/AnshYadav$ su
Password: 
```

19.

Name : SUDO :-   execute a command as another user

Syntax : sudo

Description :

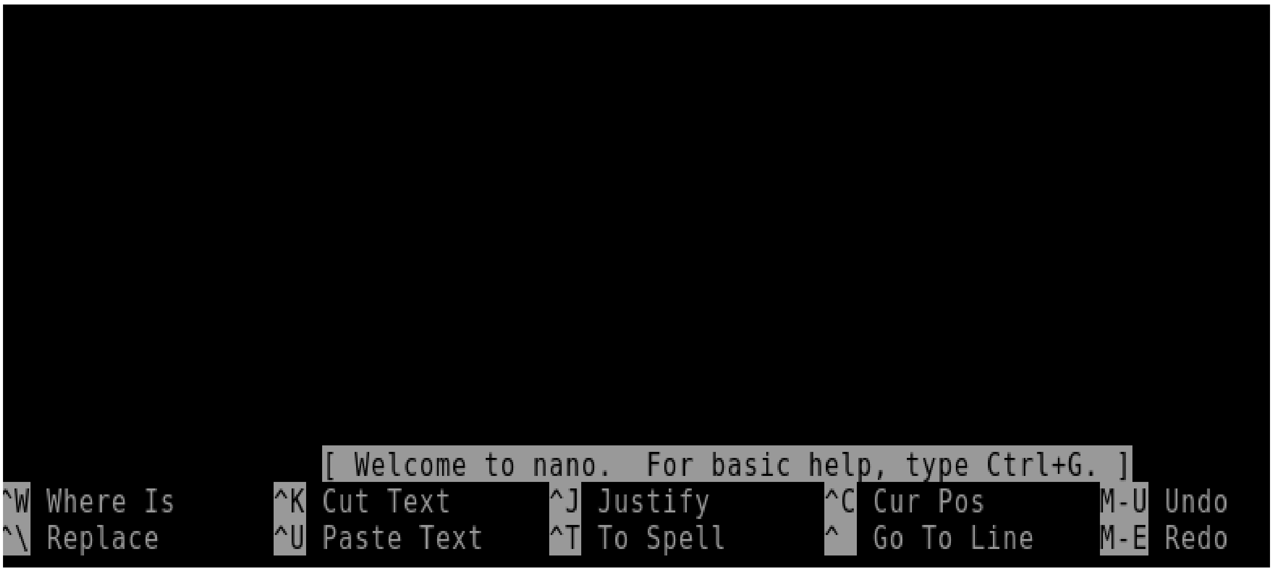


```
tryhackme@linux1:~/AnshYadav$ sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user] [
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
```

20. Name : Nano :- Nano's ANOther editor, inspired by Pico

Syntax : nano

Description :

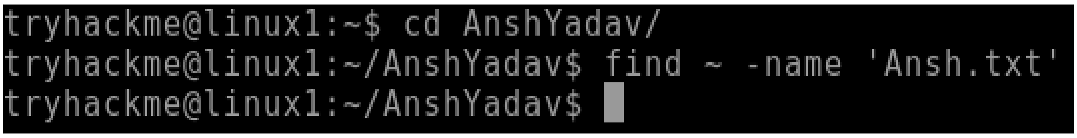


```
                    [ Welcome to nano.  For basic help, type Ctrl+G. ]
^W Where Is      ^K Cut Text      ^J Justify       ^C Cur Pos       M-U Undo
^\ Replace       ^U Paste Text    ^T To Spell      ^  Go To Line    M-E Redo
```

21. . Name : find :- search for files in a directory hierarchy

Syntax : find ~ -name 'file name.txt'

Description :

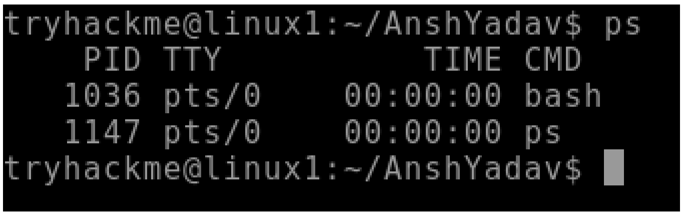


```
tryhackme@linux1:~$ cd AnshYadav/
tryhackme@linux1:~/AnshYadav$ find ~ -name 'Ansh.txt'
tryhackme@linux1:~/AnshYadav$ 
```

22. Name : PS :- report a snapshot of the current processes.

Syntax : ps

Description :

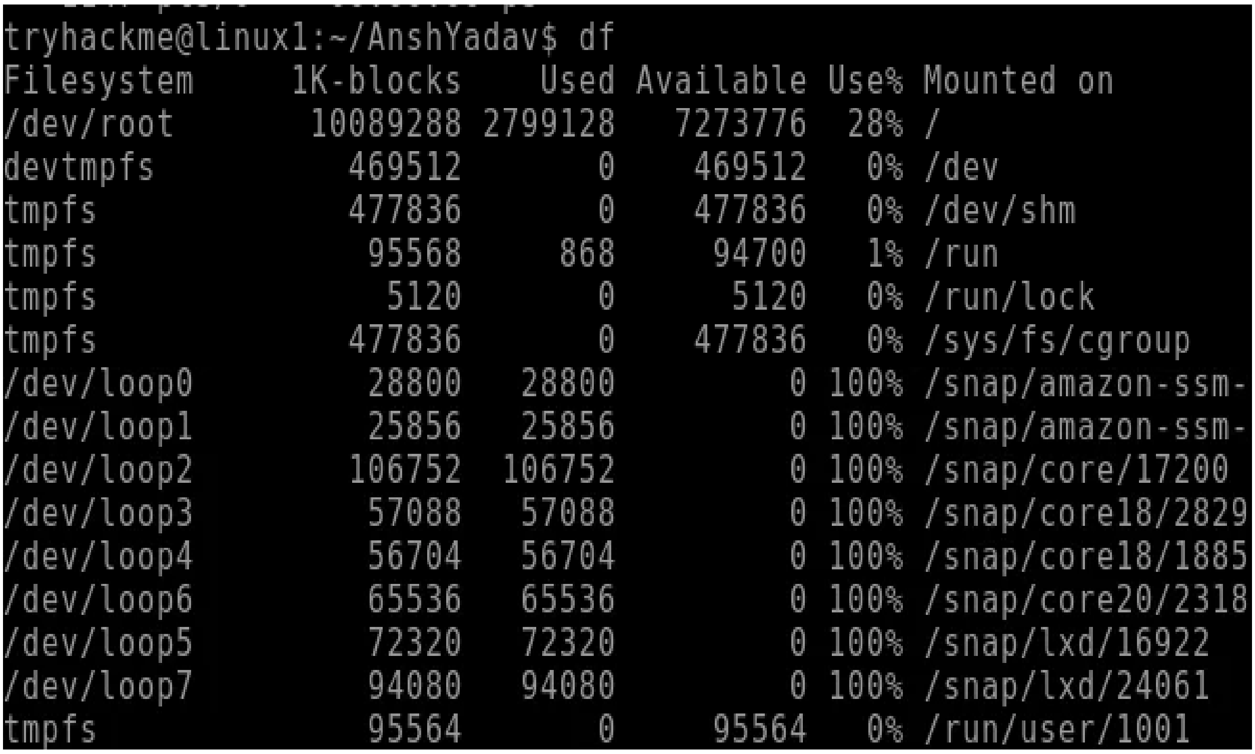


```
tryhackme@linux1:~/AnshYadav$ ps
    PID TTY          TIME CMD
   1036 pts/0    00:00:00 bash
   1147 pts/0    00:00:00 ps
tryhackme@linux1:~/AnshYadav$ 
```

23. Name : Df :- report file system disk usage space

Syntax : df

Description :

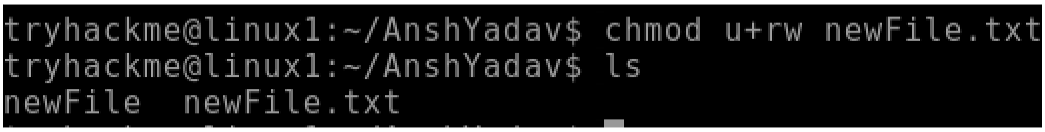


```
tryhackme@linux1:~/AnshYadav$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/root       10089288 2799128   7273776  28% /
devtmpfs          469512       0    469512   0% /dev
tmpfs             477836       0    477836   0% /dev/shm
tmpfs              95568     868     94700   1% /run
tmpfs               5120       0      5120   0% /run/lock
tmpfs             477836       0    477836   0% /sys/fs/cgroup
/dev/loop0         28800   28800         0 100% /snap/amazon-ssm-
/dev/loop1         25856   25856         0 100% /snap/amazon-ssm-
/dev/loop2        106752  106752         0 100% /snap/core/17200
/dev/loop3         57088   57088         0 100% /snap/core18/2829
/dev/loop4         56704   56704         0 100% /snap/core18/1885
/dev/loop6         65536   65536         0 100% /snap/core20/2318
/dev/loop5         72320   72320         0 100% /snap/lxd/16922
/dev/loop7         94080   94080         0 100% /snap/lxd/24061
tmpfs              95564       0     95564   0% /run/user/1001
```

24. . Name : Chmode :- change file mode bits

Syntax : chmod u+rw newFile.txt

Description :

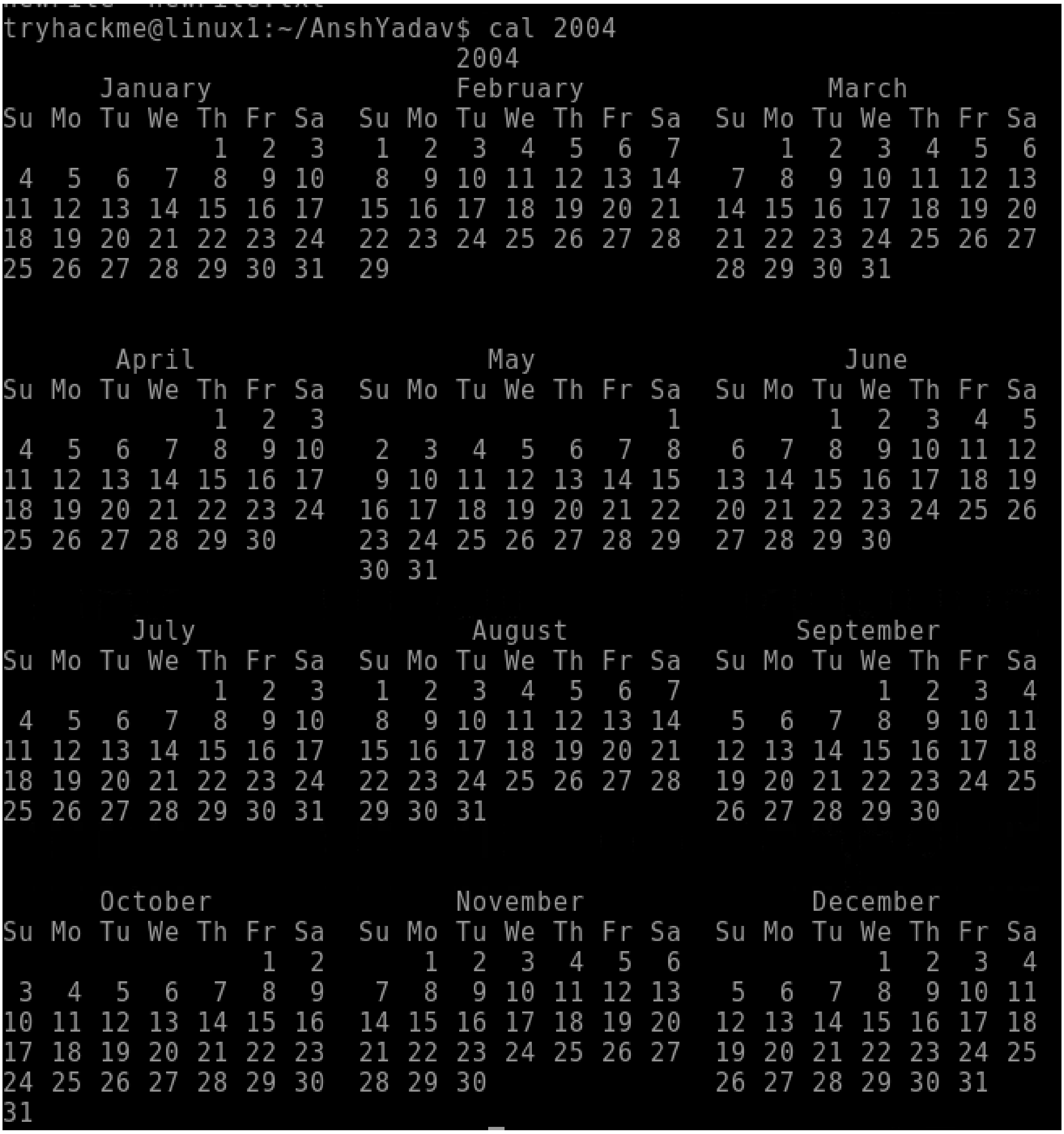


```
tryhackme@linux1:~/AnshYadav$ chmod u+rw newFile.txt
tryhackme@linux1:~/AnshYadav$ ls
newFile  newFile.txt
```

25. . Name : Cal :- gives the calendar for that year

Syntax : cal "year "

Description :

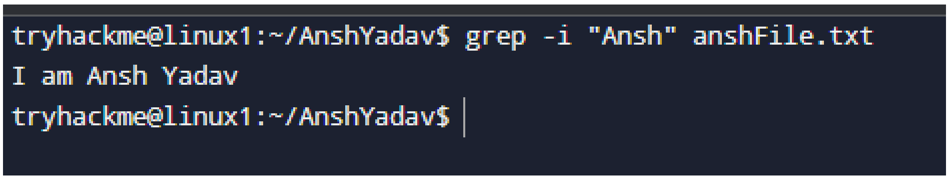


```
tryhackme@linux1:~/AnshYadav$ cal 2004
                            2004
      January               February                 March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7      1  2  3  4  5  6
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   7  8  9 10 11 12 13
11 12 13 14 15 16 17  15 16 17 18 19 20 21  14 15 16 17 18 19 20
18 19 20 21 22 23 24  22 23 24 25 26 27 28  21 22 23 24 25 26 27
25 26 27 28 29 30 31  29                    28 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                     1         1  2  3  4  5
 4  5  6  7  8  9 10   2  3  4  5  6  7  8   6  7  8  9 10 11 12
11 12 13 14 15 16 17   9 10 11 12 13 14 15  13 14 15 16 17 18 19
18 19 20 21 22 23 24  16 17 18 19 20 21 22  20 21 22 23 24 25 26
25 26 27 28 29 30     23 24 25 26 27 28 29  27 28 29 30
                      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   1  2  3  4  5  6  7            1  2  3  4
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   5  6  7  8  9 10 11
11 12 13 14 15 16 17  15 16 17 18 19 20 21  12 13 14 15 16 17 18
18 19 20 21 22 23 24  22 23 24 25 26 27 28  19 20 21 22 23 24 25
25 26 27 28 29 30 31  29 30 31              26 27 28 29 30


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

26. Name : Grep :- print lines that match patterns

Syntax : grep -i "Directory name" file name .txt

Description :



```
tryhackme@linux1:~/AnshYadav$ grep -i "Ansh" anshFile.txt
I am Ansh Yadav
tryhackme@linux1:~/AnshYadav$
```

Experiment-3

Aim: Study of Linux Architecture different types of kernel and shell

1. Linux Architecture Overview

Linux architecture is built on a layered model, with each layer having a distinct role in managing system operations.Linux follows a **monolithic architecture**, meaning most of the operating system services run in the kernel space. The key layers in Linux architecture are:

a) Hardware Layer

This is the physical hardware like CPU, memory, and I/O devices. The operating system interacts directly with the hardware through device drivers.

b) Kernel

The core part of Linux, responsible for managing hardware, system calls, memory, and process management. The kernel is the intermediary between hardware and software.

c) System Libraries

Libraries like the GNU C Library (glibc) provide essential functions to interact with the kernel via system calls. These libraries make it easier for programs to perform tasks like file operations, memory management, etc.

d) System Utilities

These are user-space utilities that manage system tasks. Examples include disk management, process monitoring, etc. Common utilities include ps, top, ls, and grep.

e) Shell

The shell acts as an interface between the user and the kernel, interpreting commands and executing them via the kernel. Users can directly interact with the shell to run commands or scripts.

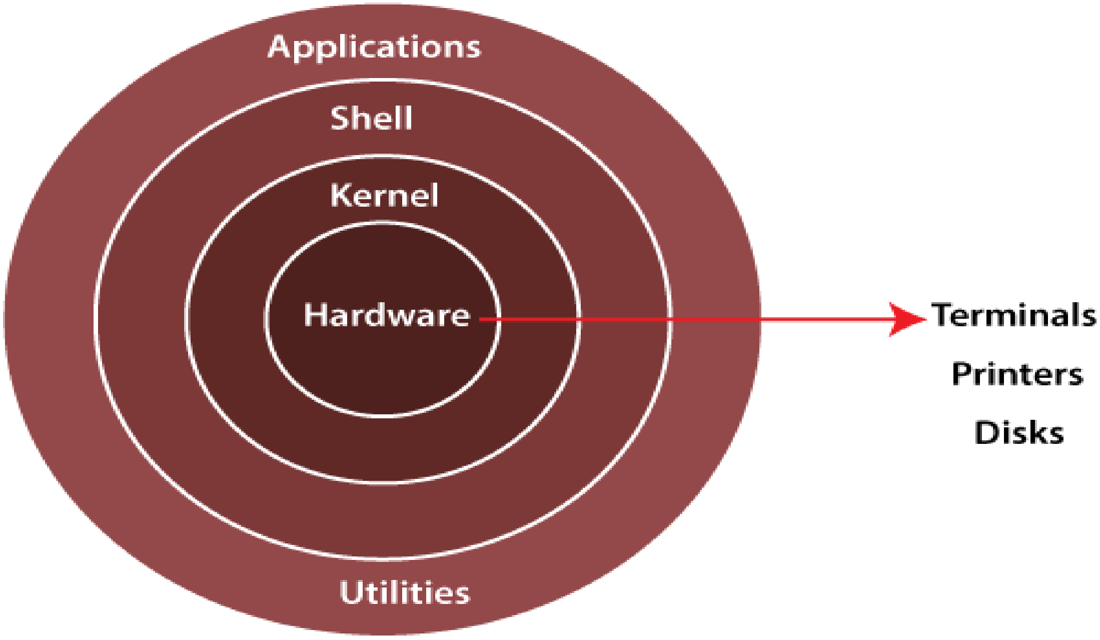


Fig.5

2. Types of Kernels

There are several types of kernels based on their design and the roles they play in an operating system:

a) Monolithic Kernel

- **Description:** In a monolithic kernel, all OS services (like device drivers, file system management, etc.) run in kernel space. It integrates many functionalities directly in the kernel.
- **Example:** Linux Kernel.
- **Advantages:** Faster access to hardware and services since everything is tightly integrated.
- **Disadvantages:** Any bug in the kernel can crash the entire system, making it less modular.

b) Microkernel

- **Description:** Microkernel architecture separates basic services from the kernel and runs them in user space. Only essential services (like inter-process communication, basic scheduling) reside in the kernel space.
- **Example:** Minix, QNX.
- **Advantages:** Increased system stability and security. Bugs in user space services don't crash the entire system.

- **Disadvantages:** Slower due to more context switching between user and kernel spaces.

c) Hybrid Kernel

- **Description:** A combination of monolithic and microkernel. The kernel runs essential services in kernel space but offloads others to user space.
- **Example:** Windows NT, macOS (XNU kernel).
- **Advantages:** Better stability than monolithic kernels with less overhead than microkernels.
- **Disadvantages:** Increased complexity in development and debugging.

d) Exokernel

- **Description:** Provides minimal abstraction and lets applications manage resources directly. Unlike other kernels, it focuses on efficiency by avoiding unnecessary abstractions.
- **Example:** MIT's Exokernel project.
- **Advantages:** Highly efficient, giving full control to the applications.
- **Disadvantages:** Complex and difficult to program, since the applications need to manage hardware resources directly.

e) Nanokernel

- **Description:**A nano kernel handles only the most basic hardware functions, such as interrupt handling and context switching. It leaves higher-level services, like memory management and IPC, entirely to user space, making it even smaller than a microkernel.
- **Example :**Used in specialized embedded systems and real-time operating systems (RTOS) for efficiency.
- **Advantages:** High Efficiency: Minimal overhead and fast performance,Modular: Easy to modify system services without affecting the kernel.
- **Disadvantages :** Limited Functionality: More complex user-space implementations are needed for basic services.

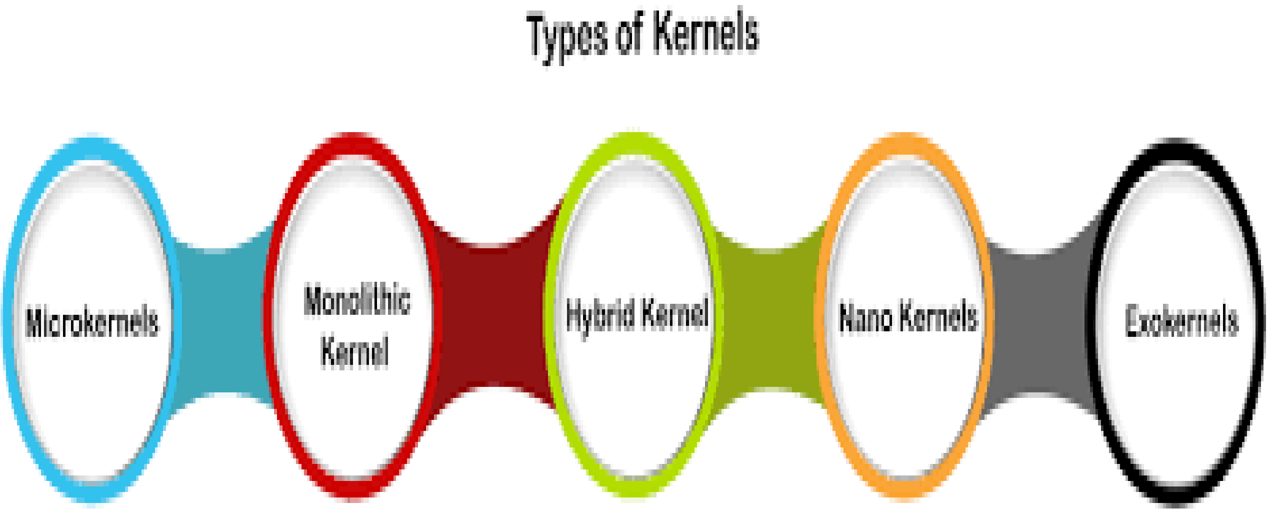


Fig.6

3. Types of Shells

Shells provide the user interface to the Linux system. There are several types of shells based on how they interpret user commands:

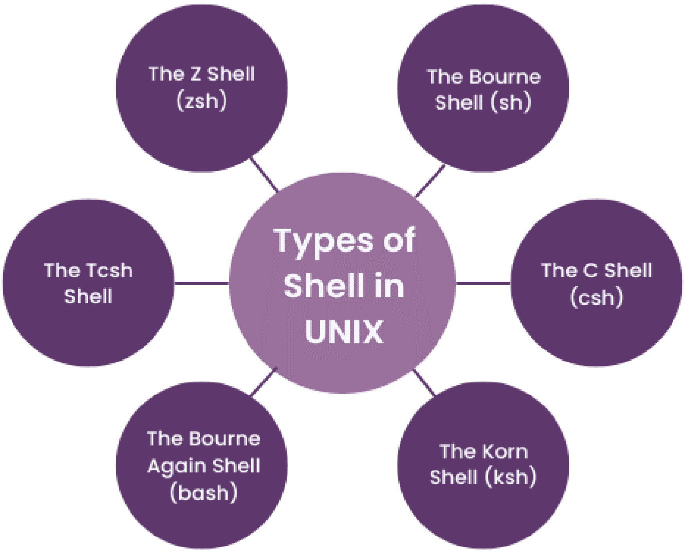


Fig.7

a) Bash (Bourne Again Shell)

- **Description:** The most widely used shell in Linux, derived from the original Bourne Shell (sh).
- **Features:** Supports scripting, variables, loops, and a variety of built-in commands. It is default on most Linux distributions.

- **Example:**$ bash, used in many Linux systems by default.

b) Zsh (Z Shell)

- **Description:** An extended version of Bash with additional features like spell-checking, auto-completion, and enhanced scripting capabilities.
- **Features:** More powerful configuration options and customization than Bash. Popular for power users.
- **Example:**$ zsh.

c) Ksh (Korn Shell)

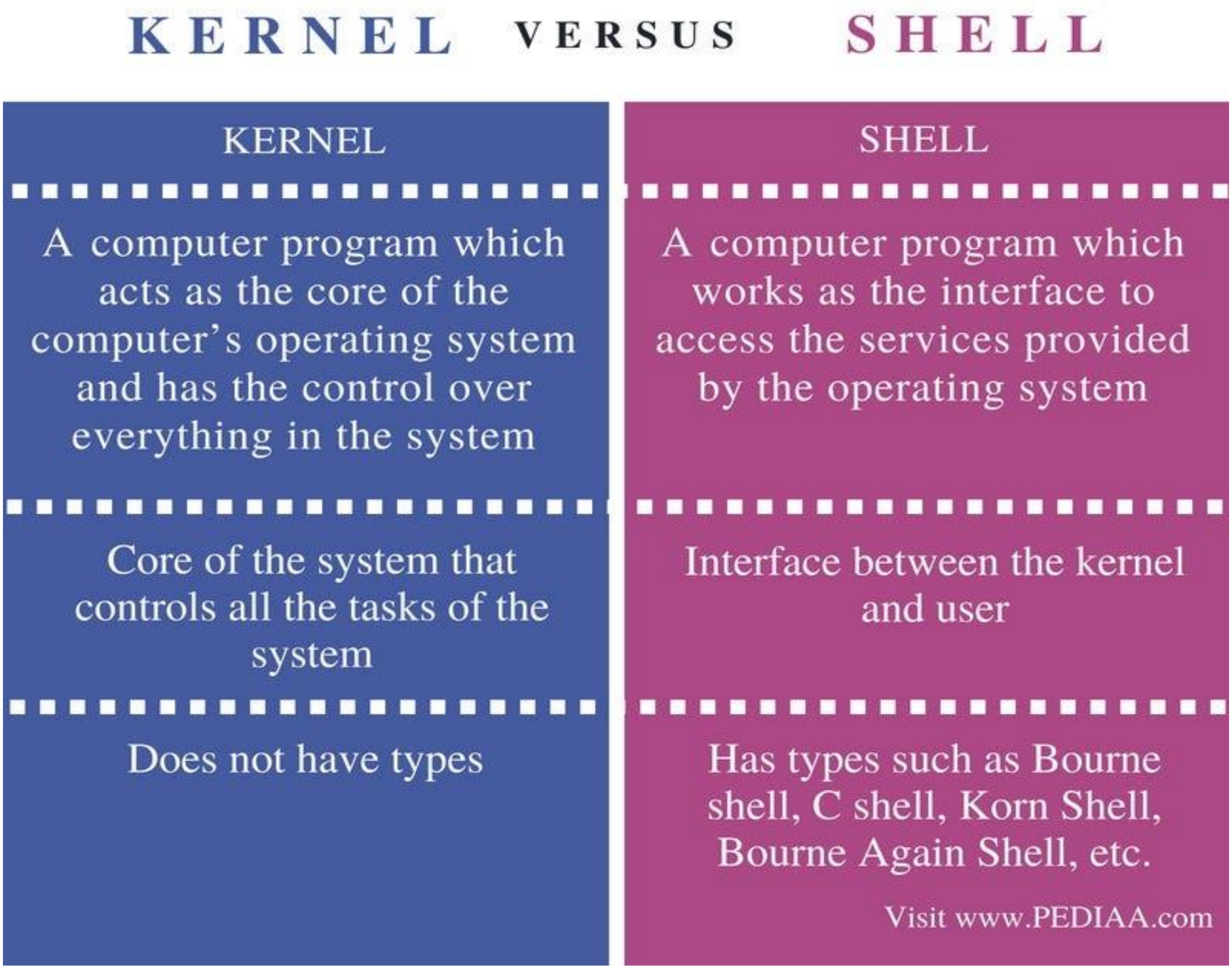
- **Description:** Combines features of both Bourne Shell and C Shell, with advanced scripting features and programming capabilities.
- **Features:** Better performance in scripting and built-in math functions.
- **Example:**$ ksh.

d) Tcsh (C Shell)

- **Description:** An enhanced version of the C Shell (csh). It uses syntax similar to the C programming language.
- **Features:** Useful for C programmers, with features like auto-completion and job control.
- **Example:**$ tcsh.

e) Fish (Friendly Interactive Shell)

- **Description:** A modern shell designed to be user-friendly and interactive. It offers features like syntax highlighting, smart auto-suggestions, and tab completion.
- **Features:** No need for complex configuration. It is designed to be more intuitive and easier to use.
- **Example:**$ fish.



KERNEL VERSUS SHELL

KERNEL	SHELL
A computer program which acts as the core of the computer's operating system and has the control over everything in the system	A computer program which works as the interface to access the services provided by the operating system
Core of the system that controls all the tasks of the system	Interface between the kernel and user
Does not have types	Has types such as Bourne shell, C shell, Korn Shell, Bourne Again Shell, etc.

Visit www.PEDIAA.com

Fig.8

Conclusion :

- **Linux Architecture** is monolithic, with layers including hardware, kernel, system libraries, utilities, and shell.
- **Types of Kernels**: Monolithic (like Linux), Microkernel, Hybrid, and Exokernel.
- **Types of Shells**: Bash, Zsh, Ksh, Tcsh, and Fish are popular options for command-line interaction.

Each kernel and shell has its strengths, and their choice depends on the use case, whether it's for efficiency, stability, or user experience.

Experiment-4

Aim: To Study the Linux file system

Introduction to Linux File System

A **Linux file system** is the foundation of how data is stored, organized, and accessed on Linux-based operating systems. It determines how files and directories are structured, managed, and retrieved on storage devices such as hard drives, SSDs, and external drives. Each file system type has its specific structure, features, and optimizations for various tasks, like maximizing speed, ensuring data security, or supporting large file systems.

File System Structure

The Linux file system uses a hierarchical **directory tree** structure, where the root directory (/) serves as the starting point. Everything in Linux, including files, directories, devices, and other system resources, is represented as part of this tree

Directory	Description
/	Root directory, the starting point for all paths in the Linux file system.
/bin	Essential binaries (executables) like system commands (ls, cp, mv) used by all users and administrators.
/etc	Configuration files for system-wide applications and services, such as fstab and networking settings.
/home	Home directories for users. Each user has a personal directory under /home (e.g., /home/username).
/var	Variable files such as logs, caches, and temporary files that change during system runtime.
/dev	Device files representing hardware devices such as hard drives, USBs, printers, etc.

/usr	Secondary hierarchy for user binaries, libraries, and documentation. Includes subdirectories like /usr/bin.
/lib	Essential shared libraries required by binaries located in /bin and /sbin.
/opt	Optional software installed on the system. Often used for third-party applications.
/tmp	Temporary files used by applications during system runtime. Automatically cleared upon reboot.

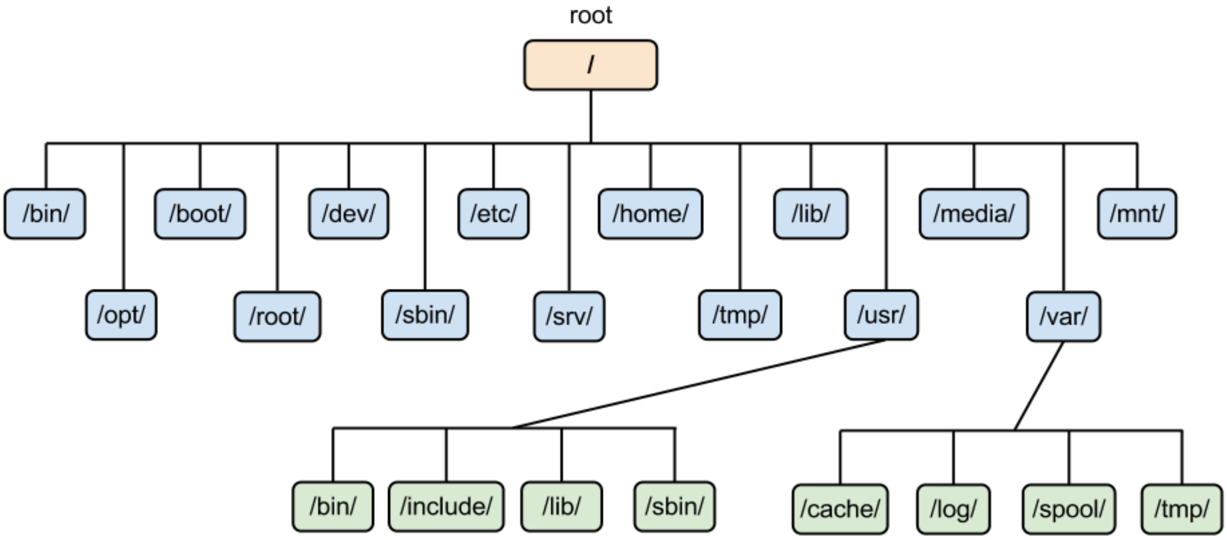


Fig.9

Characteristics & Features of a Linux File System

1. **File Organization**
   Linux organizes files into directories and subdirectories. Everything in Linux is treated as a file, including directories, hardware devices, and even processes.
2. **File Permissions**
   Linux file permissions control who can read, write, or execute a file or directory. The permissions are broken into three categories:
   - **Owner**: The user who owns the file.
   - **Group**: Users who belong to the file's group.
   - **Others**: All other users.

   Permissions are represented as rwx (read, write, execute).

3. **File Ownership**
   Each file and directory is owned by a user and a group. The ownership controls which users and groups have permissions to modify or execute files.
4. **Mounting**
   To access a file system in Linux, it must be **mounted**. A mounted file system is attached to a directory in the tree structure (usually under /mnt or /media). This makes external drives or partitions accessible from within the Linux environment.
5. **Inodes**
   Linux uses **inodes** to store metadata about files, such as:
   - File permissions.
   - File size.
   - Number of links (hard links).
   - User and group ownership.
   - Timestamps (creation, modification). An inode does not store the file's name or actual data but provides pointers to the data blocks where the file is stored.
6. **Journaling**
   Some Linux file systems (like Ext4, XFS, and Btrfs) use **journaling** to record changes before they are committed. This helps prevent data corruption and speeds up recovery after a system crash by maintaining a log of recent changes.
7. **File Types**
   Linux supports multiple types of files:
   - **Regular files**: Standard files like text files, images, or executables.
   - **Directories**: Special files that contain other files or directories.
   - **Symbolic links**: Pointers or shortcuts to other files or directories.
   - **Device files**: Represent hardware devices.
   - **Named pipes**: Files for inter-process communication.
   - **Sockets**: For network communication between processes.

Types of File Systems in Linux

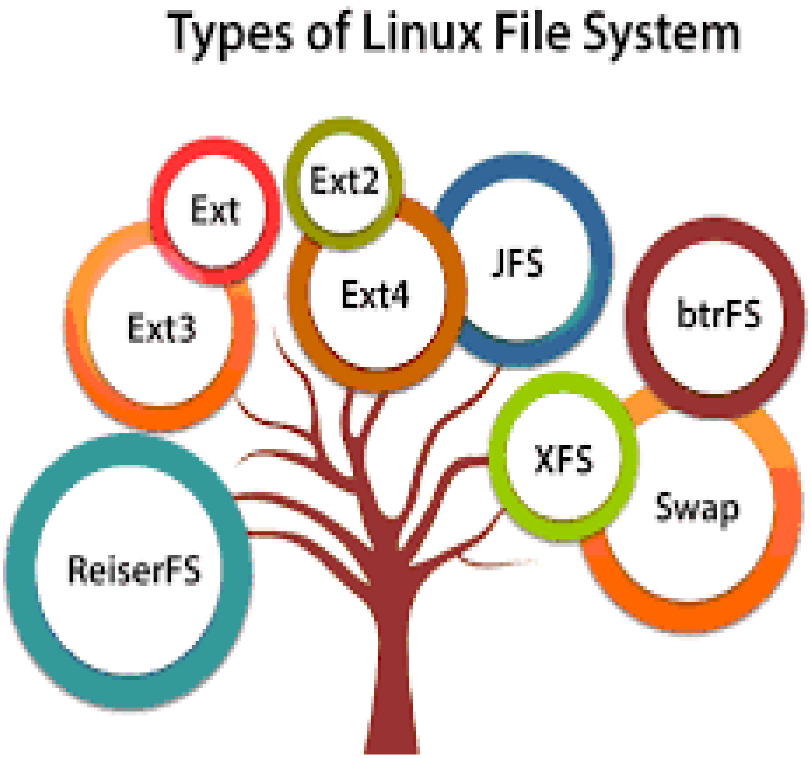


Fig.10

a) Ext (Extended File System)

- **Description**: The oldest file system family for Linux, designed specifically for this OS. It comes in several versions.
- **Versions**:
  - **Ext2**: Does not support journaling but is fast and efficient.
  - **Ext3**: Adds journaling to Ext2, improving reliability in case of crashes.
  - **Ext4**: The most widely used version, offering support for larger file sizes, journaling, and performance enhancements.
- **Features**:
  - **Support for large files** (up to 16 TB) and volumes (up to 1 EB).
  - **Improved journaling** for faster recovery from crashes.
  - **Backward compatibility** with Ext2 and Ext3.

b) XFS

- **Description**: A high-performance journaling file system initially developed by Silicon Graphics (SGI) for their IRIX operating system.
- **Features**:
  - **Excellent scalability** for large files and file systems.
  - **Efficient for large, high-throughput systems**, making it ideal for servers.
  - Supports **64-bit** systems and files up to 8 exabytes.

c) Btrfs (B-tree File System)

- **Description**: A modern Linux file system designed for advanced data management.
- **Features**:
  - **Snapshots**: Allows point-in-time copies of the file system.
  - **Subvolumes**: Enables organizing different parts of the file system into independently managed units.
  - **Compression**: Supports transparent file compression.
  - **Data integrity features**: Built-in error detection and automatic repair using checksums.

d) ReiserFS

- **Description**: A journaling file system known for its efficiency in handling small files.
- **Features**:
  - **Quick file lookups**: Well-suited for environments with many small files.
  - **Space efficiency**: Uses space more efficiently than Ext3 for small file storage.
  - Though it was popular for a time, development has stagnated, and it has largely been replaced by Ext4 or Btrfs in modern systems.

e) FAT32/NTFS

- **Description**: These are file systems used by Windows, but Linux supports them for compatibility.
- **Features**:
  - **FAT32**: Widely used for USB drives and older storage media. Limited support for large files (max file size of 4 GB).
  - **NTFS**: The primary file system for modern Windows systems, supporting larger files and better performance than FAT32.

f) Swap File System

- **Description**: A special file system type in Linux used to extend physical memory by using disk space as virtual memory.
- **Features**:
  - Provides **swap space** for system memory management, helping prevent out-of-memory errors.
  - **Important for systems with low physical RAM**, as it allows the operating system to move inactive data from RAM to disk.

Conclusion:

The Linux file system offers a flexible, scalable, and robust way to organize and manage data. With a wide range of file systems like Ext4, XFS, Btrfs, and others, Linux can support various use cases—from desktop computers and servers to embedded systems and mobile devices. Each file system has its own strengths, whether it's performance, data integrity, or compatibility, making Linux adaptable to different storage needs.

Experiment No. 5

Aim: Introduction to linux editors.

Linux text editor is kind of a computer program that can edit text like notepad software. Text

editors provide services in software development and operating system softwares which can be

utilized to include programming languages, source code, documentation files etc.

Mostly Used Text Editors in Linux

- Vi Text Editor
- Vim Editor
- Nano Editor
- Kate Editor
- Sublime Editor
- Atom Editor
- Emacs Editor

Linux text editors can be used for editing text files, writing codes and updating user instruction

files and many more.

Linux Text editors come in two forms-

(i) Command Line Text editors (such as Vi, nano, pico and more) and

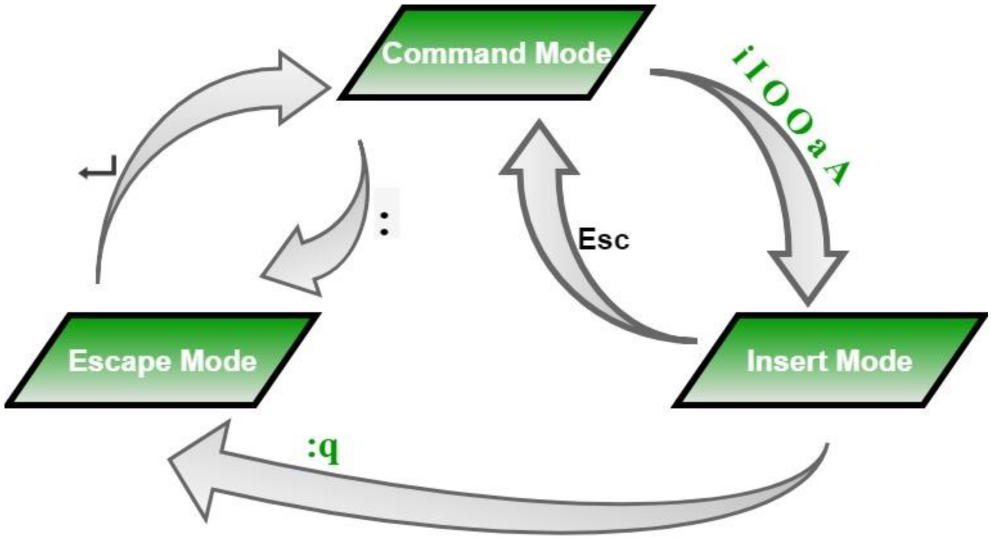
(ii) GUI based text editors (such as GNOME, KDE, etc)

Introduction of Vi editor :-

The default editor that comes with a UNIX os is Vi editor(Visual editor).

The unix editor is a full screen editor used to perform a variety of tasks. The unix architecture comes in three modes:

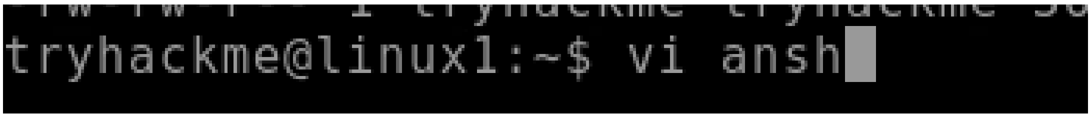
(i) Command, **(ii) insert** and **(iii) Esc mode**



Three modes of unix editor

(i) Command:-

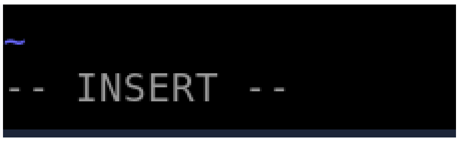
In Command mode each character is typed as a command that does something to the text file. The



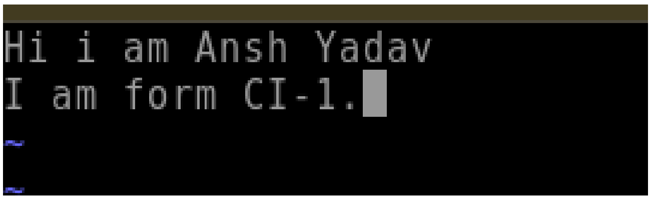
```
tryhackme@linux1:~$ vi ansh
```

basic vi editor commands are,

(a) vi <file name>:- edit filename starting at line-1.



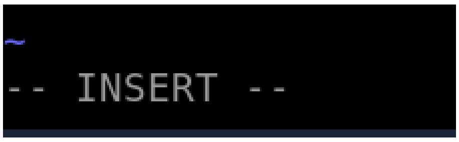
```
~
-- INSERT --
```



```
Hi i am Ansh Yadav
I am form CI-1.
~
~
```

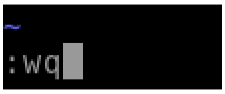
(a) To insert text:-

(i) i (before cursor) :-



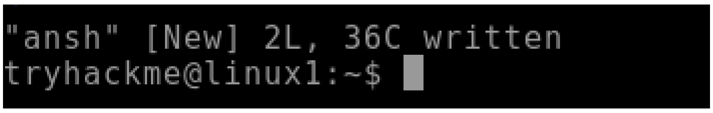
```
~
-- INSERT --
```

(ii) a (after cursor) :-

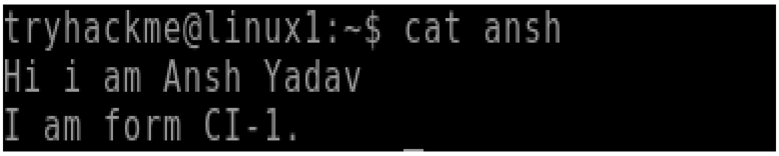


```
~
:wq
```

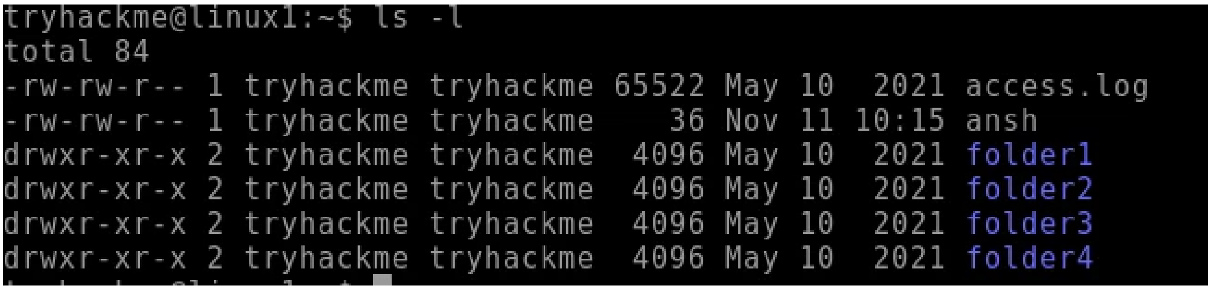
(iii) A (At the end of the line) :-



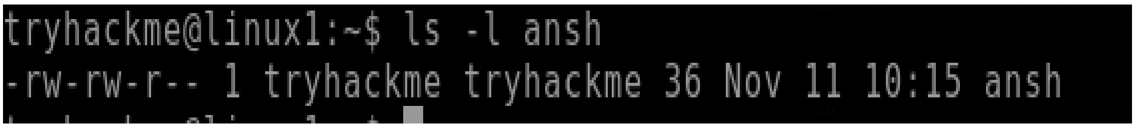
```
"ansh" [New] 2L, 36C written
tryhackme@linux1:~$
```



```
tryhackme@linux1:~$ cat ansh
Hi i am Ansh Yadav
I am form CI-1.
```



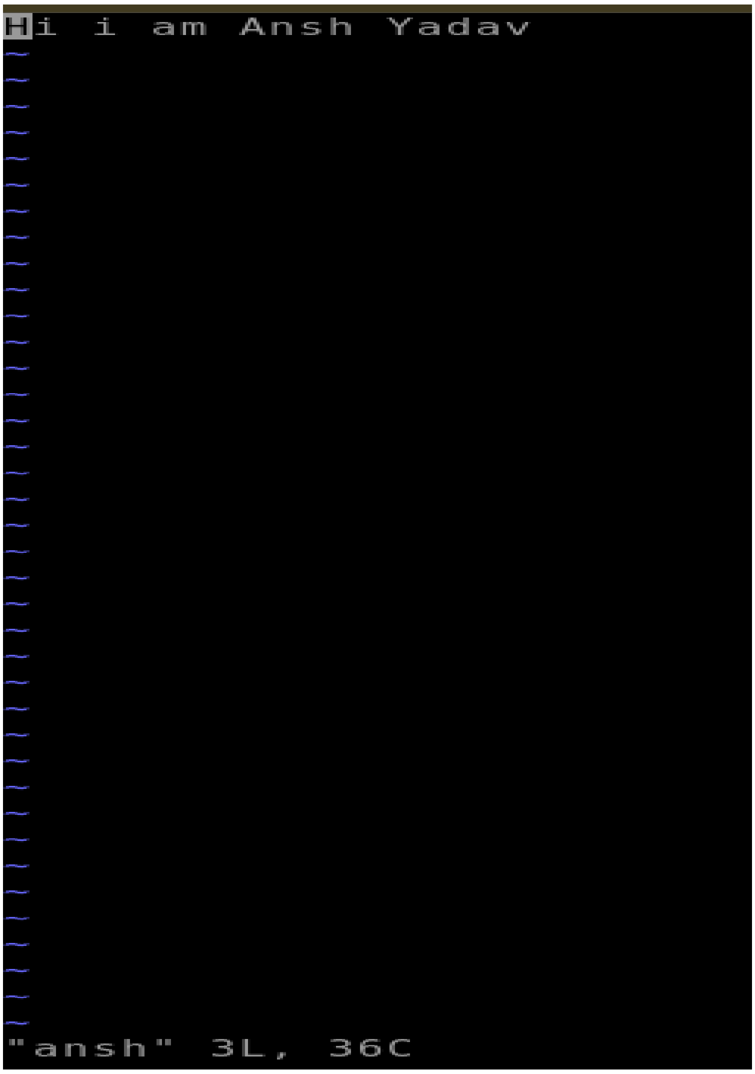
```
tryhackme@linux1:~$ ls -l
total 84
-rw-rw-r-- 1 tryhackme tryhackme 65522 May 10  2021 access.log
-rw-rw-r-- 1 tryhackme tryhackme    36 Nov 11 10:15 ansh
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder1
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder2
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder3
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder4
```



```
tryhackme@linux1:~$ ls -l ansh
-rw-rw-r-- 1 tryhackme tryhackme 36 Nov 11 10:15 ansh
```

(c) To delete text:-

(i) dd



```
Hi i am Ansh Yadav
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"ansh" 3L, 36C
```

Experiment 6

AIM: Create a file called wlcc.txt with some line and display how many lines,

words and characters are present in the file

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

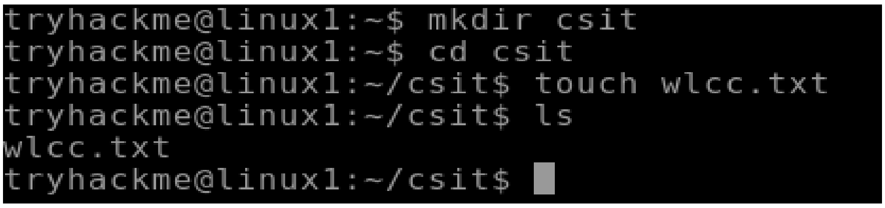
- It is used to find out the number of lines, word count, byte and characters count in the files specified in the file arguments.
- By default it displays four-columnar output.
- First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.

Syntax:

wc [OPTION]... [FILE]...

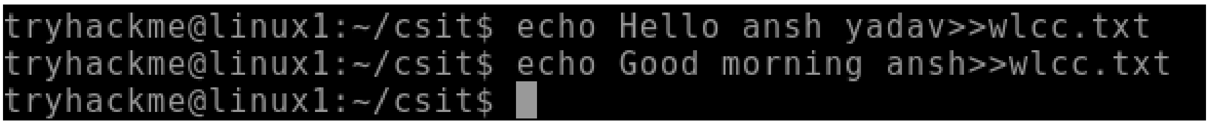
To create a file called `wlcc.txt` and display the number of lines, words, and characters it contains, you can follow these steps:

Step 1:Create the File wlcc.txt by using touch command.



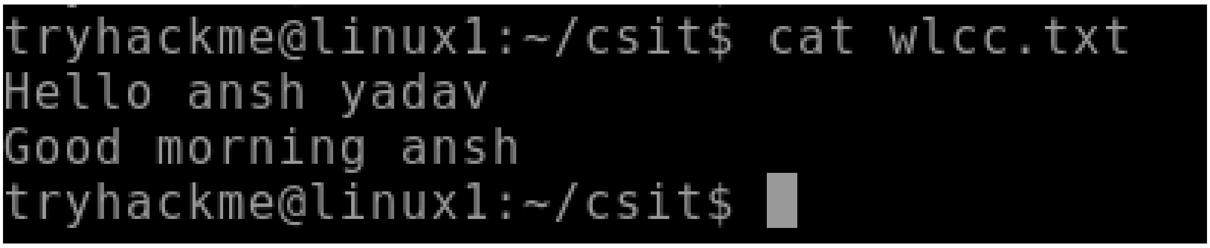
```
tryhackme@linux1:~$ mkdir csit
tryhackme@linux1:~$ cd csit
tryhackme@linux1:~/csit$ touch wlcc.txt
tryhackme@linux1:~/csit$ ls
wlcc.txt
tryhackme@linux1:~/csit$ 
```

Step 2: Write in the text file wlcc.txt by using echo command.



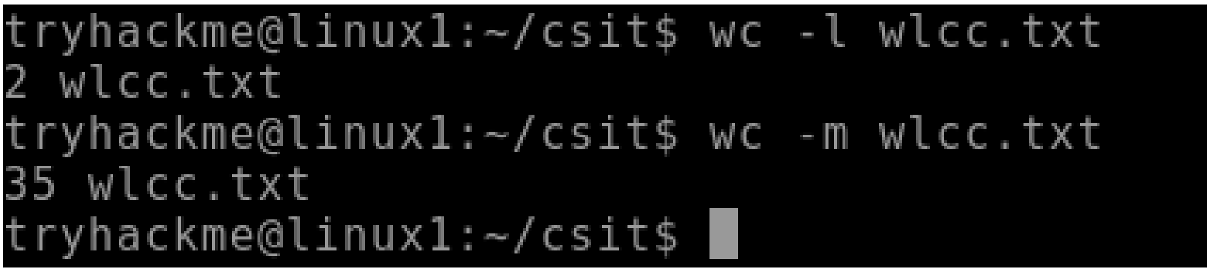
```
tryhackme@linux1:~/csit$ echo Hello ansh yadav>>wlcc.txt
tryhackme@linux1:~/csit$ echo Good morning ansh>>wlcc.txt
tryhackme@linux1:~/csit$ 
```

Step 3: Display the text which is in the wlcc.txt by using cat command.



```
tryhackme@linux1:~/csit$ cat wlcc.txt
Hello ansh yadav
Good morning ansh
tryhackme@linux1:~/csit$ 
```

Step 4: Display how many lines, words and characters are present in the file by using wc command.



```
tryhackme@linux1:~/csit$ wc -l wlcc.txt
2 wlcc.txt
tryhackme@linux1:~/csit$ wc -m wlcc.txt
35 wlcc.txt
tryhackme@linux1:~/csit$ 
```

Experiment 7

<u>AIM:-</u> Append ten more simple line to the file and display.

In the wide world of Linux, learning simple yet powerful commands is key to becoming a proficient

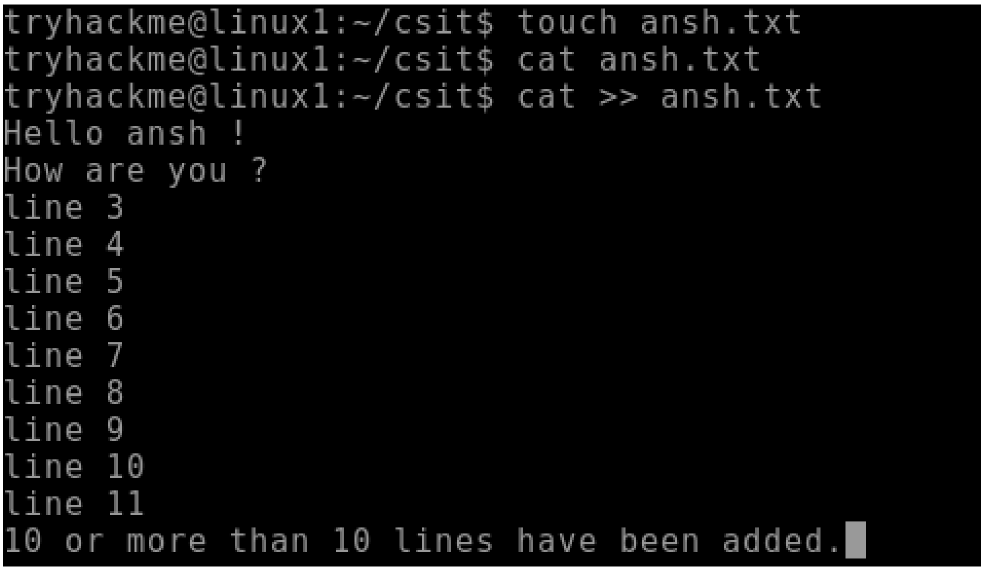
user. One such essential skill is appending lines to a file, a fundamental operation for adding

information or modifying configurations.

In Linux, text files store information in plain text, and each line typically represents a piece of data. Appending lines involves adding new information to the end of an existing file, preserving its current content.

In this tutorial, we'll explore several commands for adding one or more lines to a file

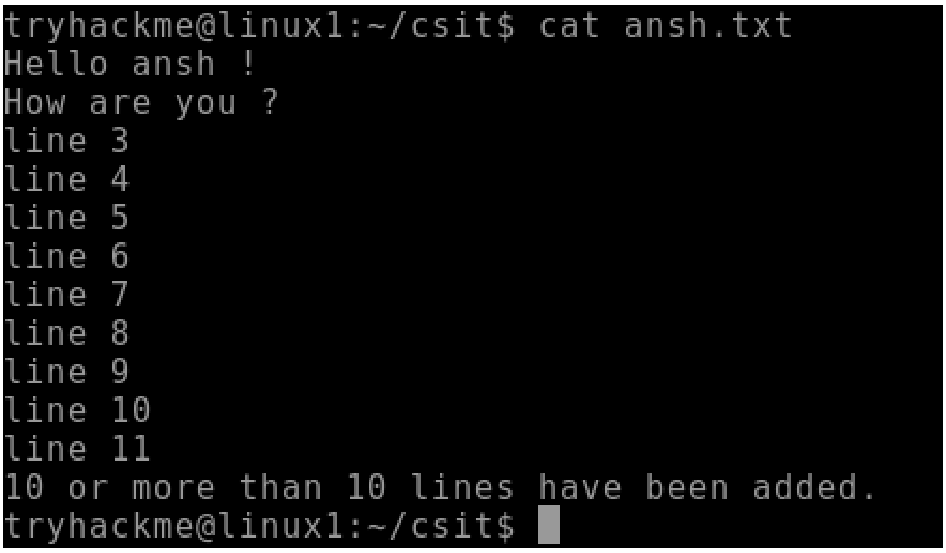
Step 1: Append Lines to the File



```
tryhackme@linux1:~/csit$ touch ansh.txt
tryhackme@linux1:~/csit$ cat ansh.txt
tryhackme@linux1:~/csit$ cat >> ansh.txt
Hello ansh !
How are you ?
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
line 11
10 or more than 10 lines have been added.
```

Step 2: Display the Contents of the File

After appending the lines, you can display the entire contents of isha.txt using the cat command:



```
tryhackme@linux1:~/csit$ cat ansh.txt
Hello ansh !
How are you ?
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
line 11
10 or more than 10 lines have been added.
tryhackme@linux1:~/csit$ 
```

Experiment 8

AIM:- Study and use of commands and for changing file permission

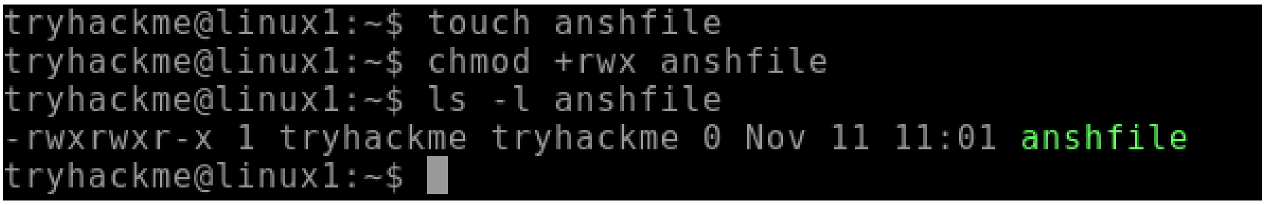
Every file and directory on your Linux system has permissions assigned to it. These permissions are what determines which users are allowed to read, write to, and/or execute the file. In this tutorial, you will learn about how to change Linux file permissions with the chmod command.

Changing File Permissions – Symbolic Mode:-

There are three types of permissions, read, write, and execute. The most user-friendly way of adding or removing permissions from a file or directory is with the chmod command and the +rwx or -rwx syntax (this is called symbolic mode) as shown in the examples below.

Example 1. Add read, write, and execute permissions to a file:

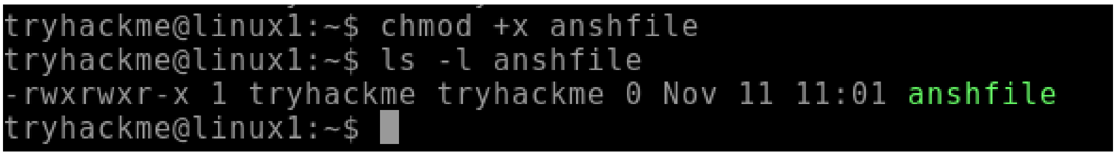
Syntax:- $ chmod +rwx filename



```
tryhackme@linux1:~$ touch anshfile
tryhackme@linux1:~$ chmod +rwx anshfile
tryhackme@linux1:~$ ls -l anshfile
-rwxrwxr-x 1 tryhackme tryhackme 0 Nov 11 11:01 anshfile
tryhackme@linux1:~$ 
```

Example 2. Add execute permissions to a file:

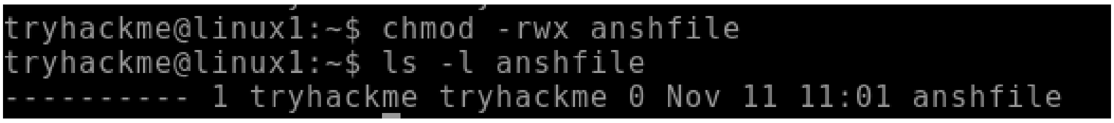
Syntax:-$ chmod +x filename



```
tryhackme@linux1:~$ chmod +x anshfile
tryhackme@linux1:~$ ls -l anshfile
-rwxrwxr-x 1 tryhackme tryhackme 0 Nov 11 11:01 anshfile
tryhackme@linux1:~$ 
```

Example 3. Remove all permissions from a file:

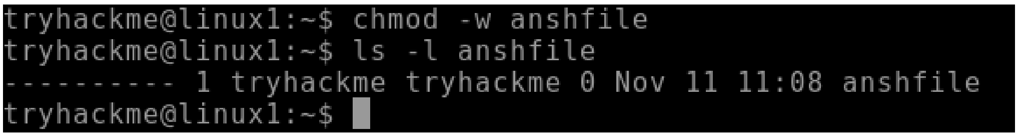
Syntax:- $ chmod -rwx filename



```
tryhackme@linux1:~$ chmod -rwx anshfile
tryhackme@linux1:~$ ls -l anshfile
---------- 1 tryhackme tryhackme 0 Nov 11 11:01 anshfile
```

Example 4. Remove write permissions from a file:

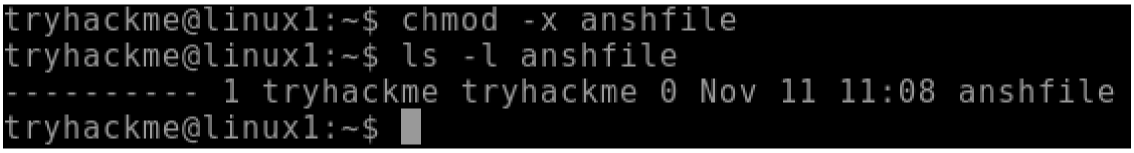
Syntax:- $ chmod -w filename



```
tryhackme@linux1:~$ chmod -w anshfile
tryhackme@linux1:~$ ls -l anshfile
---------- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 5. Remove execute permissions from a file:

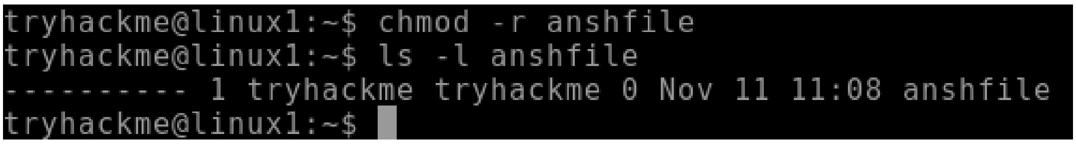
Syntax:-$ chmod -x filename



```
tryhackme@linux1:~$ chmod -x anshfile
tryhackme@linux1:~$ ls -l anshfile
---------- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 6. Remove read permissions from a file:

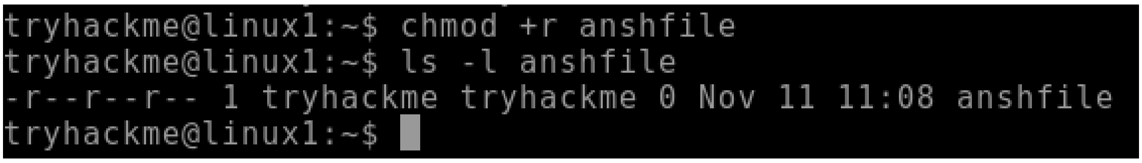
Syntax:-$ chmod -r filename



```
tryhackme@linux1:~$ chmod -r anshfile
tryhackme@linux1:~$ ls -l anshfile
---------- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 7. Add read permissions from a file:

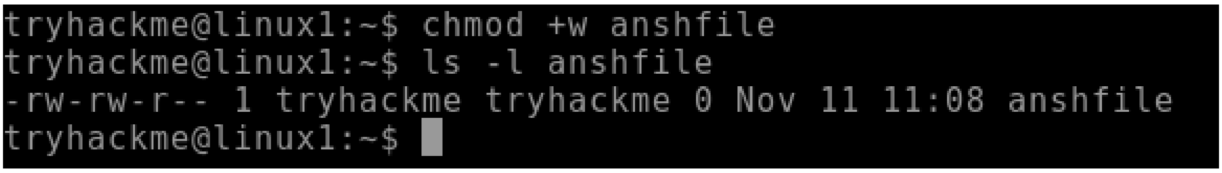
Syntax:-$ chmod +r filename



```
tryhackme@linux1:~$ chmod +r anshfile
tryhackme@linux1:~$ ls -l anshfile
-r--r--r-- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 8. Add write permissions from a file:

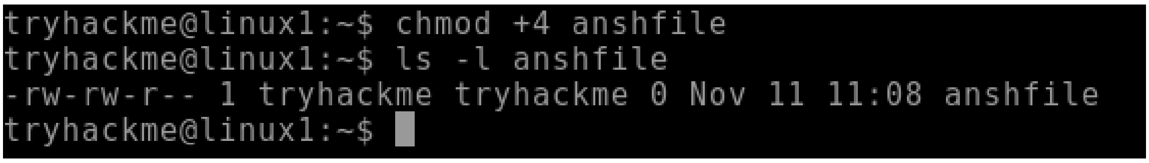
Syntax:-$ chmod +w filename



```
tryhackme@linux1:~$ chmod +w anshfile
tryhackme@linux1:~$ ls -l anshfile
-rw-rw-r-- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 9. Add 4(read) permissions from a file:

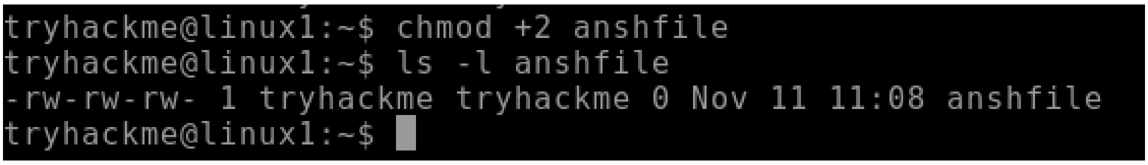
Syntax:-$ chmod +4 filename



```
tryhackme@linux1:~$ chmod +4 anshfile
tryhackme@linux1:~$ ls -l anshfile
-rw-rw-r-- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 10. Add 2(write) permissions from a file:

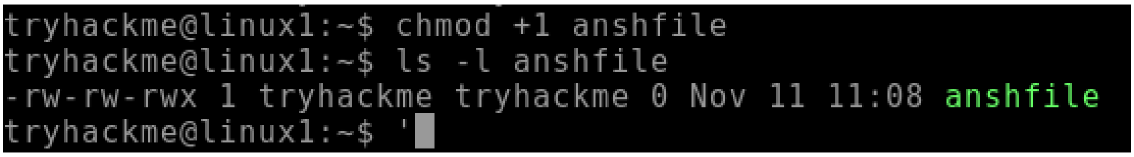
Syntax:-$ chmod +2 filename



```
tryhackme@linux1:~$ chmod +2 anshfile
tryhackme@linux1:~$ ls -l anshfile
-rw-rw-rw- 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 11. Add 1(execute) permissions from a file:

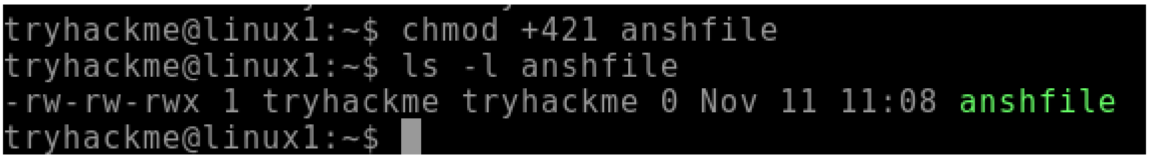
Syntax:-$ chmod +1 filename



```
tryhackme@linux1:~$ chmod +1 anshfile
tryhackme@linux1:~$ ls -l anshfile
-rw-rw-rwx 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ '
```

Example 12. Add 421(read,write,execute) permissions from a file:

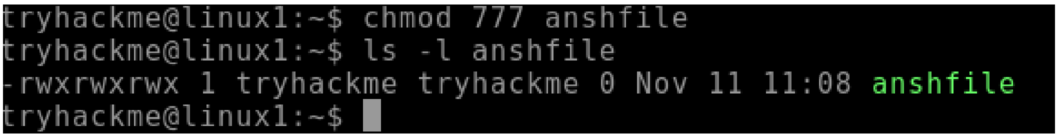
Syntax:-$ chmod +421 filename



```
tryhackme@linux1:~$ chmod +421 anshfile
tryhackme@linux1:~$ ls -l anshfile
-rw-rw-rwx 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Example 13. Add 777 permissions from a file:

Syntax:-$ chmod +777 filename



```
tryhackme@linux1:~$ chmod 777 anshfile
tryhackme@linux1:~$ ls -l anshfile
-rwxrwxrwx 1 tryhackme tryhackme 0 Nov 11 11:08 anshfile
tryhackme@linux1:~$ 
```

Viewing File Permissions:

Running the ls -l command on a file or directory will reveal everything you need to know about its current permissions.

Syntax :

```
$ ls -l script.sh
```

The output you see will look something like this:

```
-rwxrw-r-- 1 linuxnightly admins 42 Jan 20 14:19 script.sh
```

The first character of the output tells us what type of file it is.

- - = regular file
- d = directory
- l = symbolic link

You may see other file types throughout your system, but directories and regular files will comprise the vast majority of what you find.

The next set of characters we see are rwxrw-r--. These nine characters can be split into three different blocks. Each block of characters represents the permissions for a different user or group. In our example:

- rwx = user permissions
- rw- = group permissions
- r-- = other permissions

What each permission means:

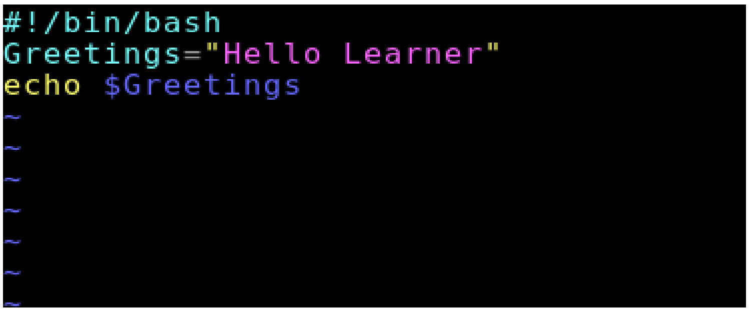
- r = read permissions
- w = write permissions
- x = execute permissions
- - = no permissions

Experiment 9

AIM:- SHELL Script

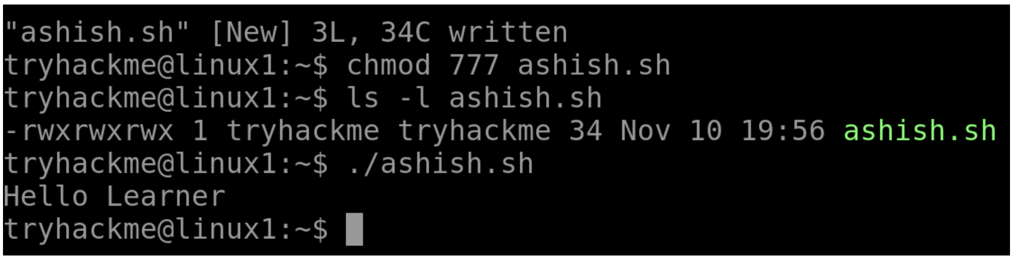
9.1 Write Shell script to print Hello Learner

Program



```
#!/bin/bash
Greetings="Hello Learner"
echo $Greetings
~
~
~
~
~
~
~
```

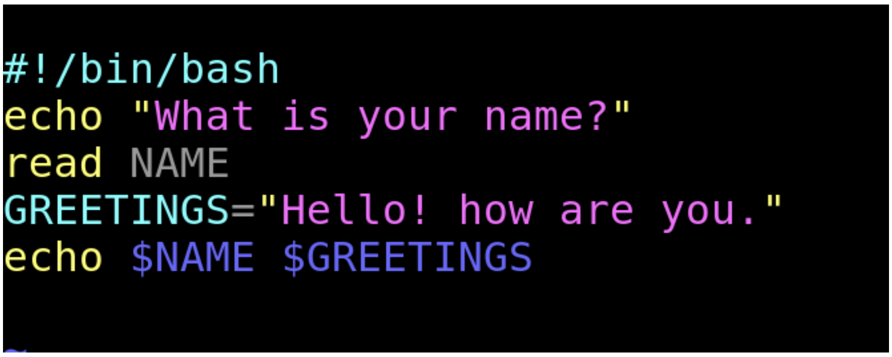
Output



```
"ashish.sh" [New] 3L, 34C written
tryhackme@linux1:~$ chmod 777 ashish.sh
tryhackme@linux1:~$ ls -l ashish.sh
-rwxrwxrwx 1 tryhackme tryhackme 34 Nov 10 19:56 ashish.sh
tryhackme@linux1:~$ ./ashish.sh
Hello Learner
tryhackme@linux1:~$ █
```

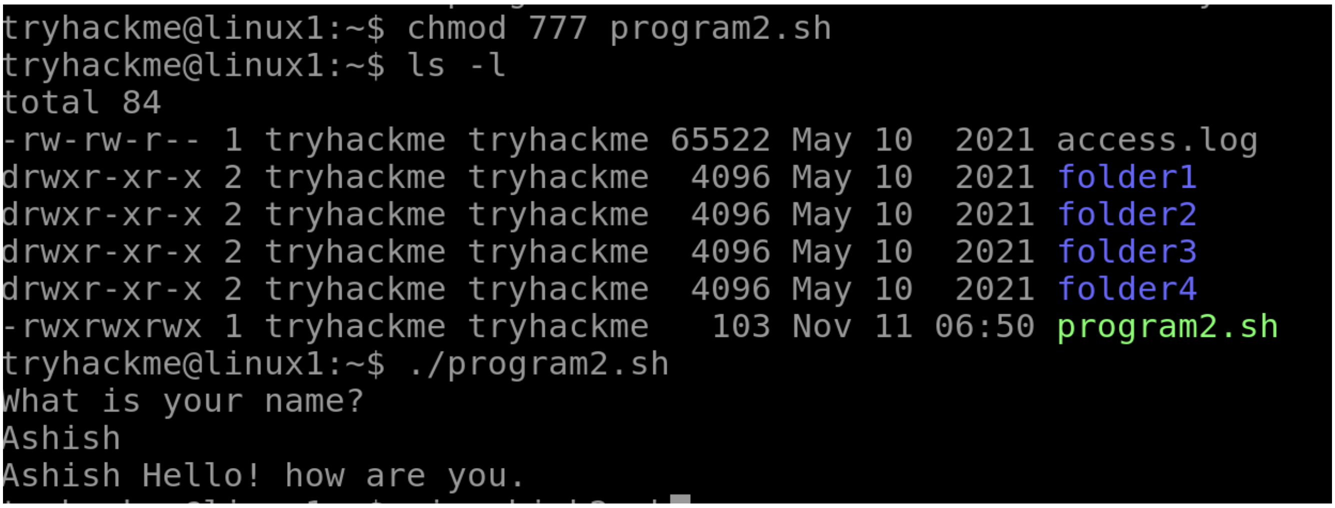
9.2 Write a bash script to print run time argument from user as a name and print Greetings <Your Name>

Program



```
#!/bin/bash
echo "What is your name?"
read NAME
GREETINGS="Hello! how are you."
echo $NAME $GREETINGS

~
```

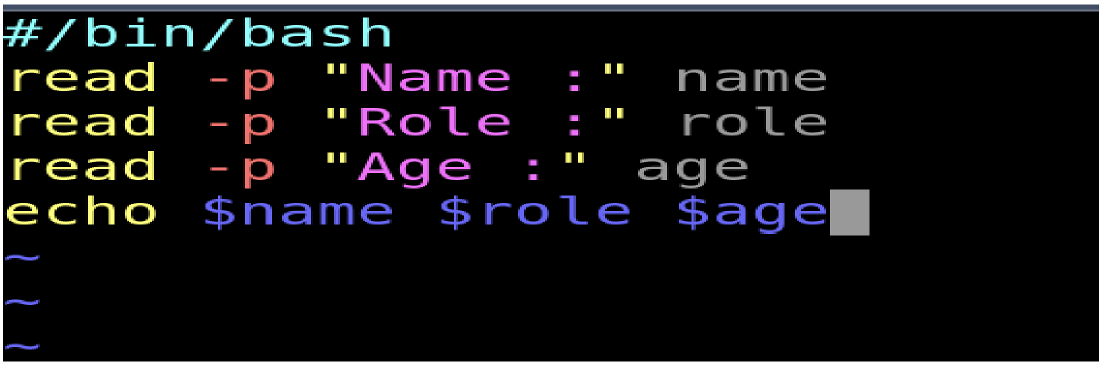
Output



```
tryhackme@linux1:~$ chmod 777 program2.sh
tryhackme@linux1:~$ ls -l
total 84
-rw-rw-r-- 1 tryhackme tryhackme 65522 May 10  2021 access.log
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder1
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder2
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder3
drwxr-xr-x 2 tryhackme tryhackme  4096 May 10  2021 folder4
-rwxrwxrwx 1 tryhackme tryhackme   103 Nov 11 06:50 program2.sh
tryhackme@linux1:~$ ./program2.sh
What is your name?
Ashish
Ashish Hello! how are you.
```

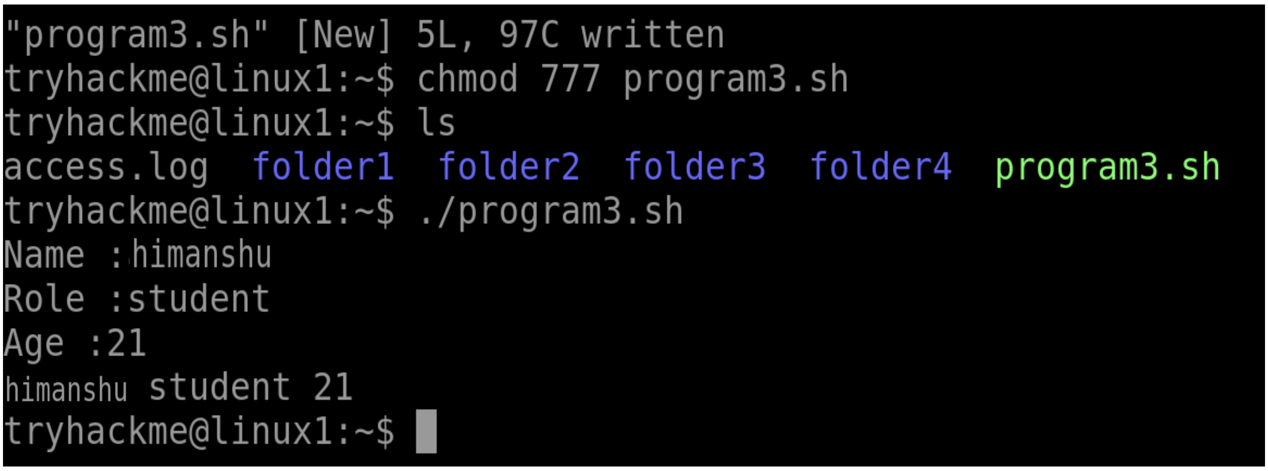
9.3 **Demonstrate the shell script with different usage of variable.**

Program



```
#/bin/bash
read -p "Name :" name
read -p "Role :" role
read -p "Age :" age
echo $name $role $age
~
~
~
```

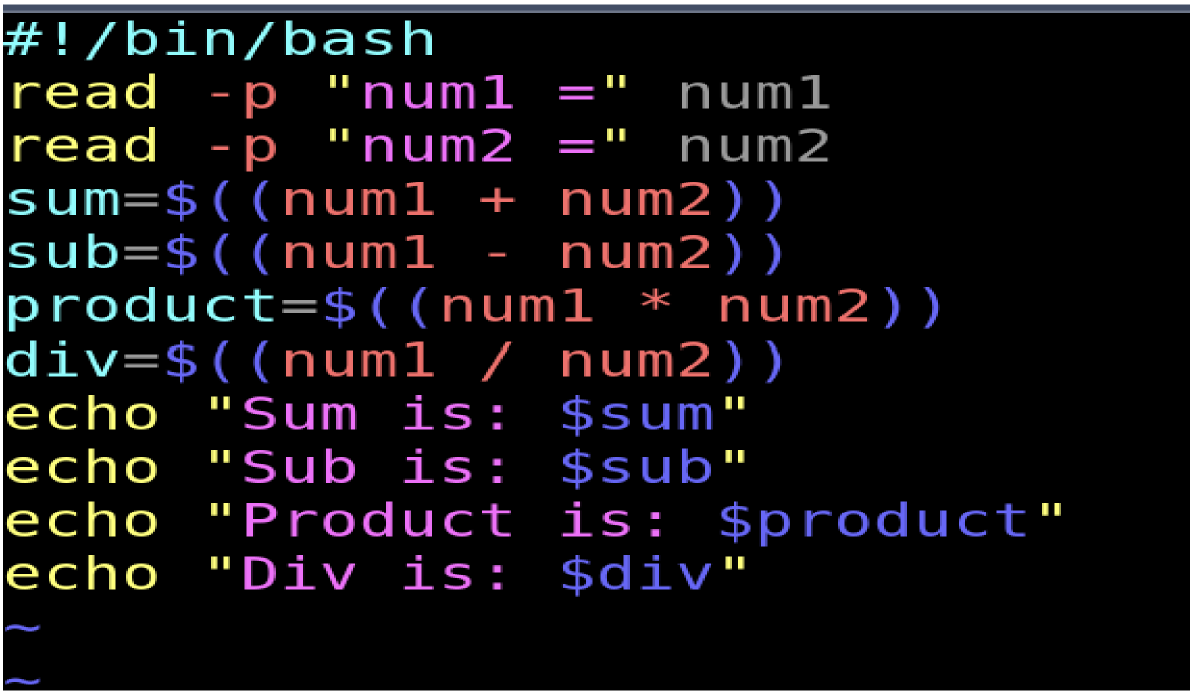
Output



```
"program3.sh" [New] 5L, 97C written
tryhackme@linux1:~$ chmod 777 program3.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  program3.sh
tryhackme@linux1:~$ ./program3.sh
Name :himanshu
Role :student
Age :21
himanshu student 21
tryhackme@linux1:~$
```

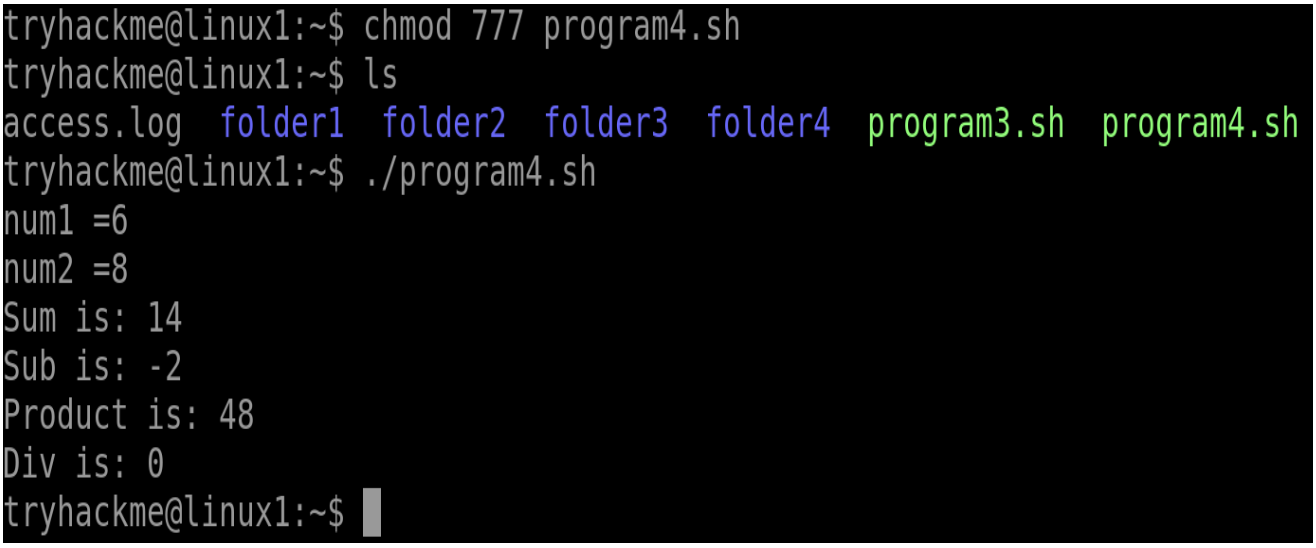
9.4 Write a shell script to perform arithmetic operations

Program



```
#!/bin/bash
read -p "num1 =" num1
read -p "num2 =" num2
sum=$((num1 + num2))
sub=$((num1 - num2))
product=$((num1 * num2))
div=$((num1 / num2))
echo "Sum is: $sum"
echo "Sub is: $sub"
echo "Product is: $product"
echo "Div is: $div"
~
~
```

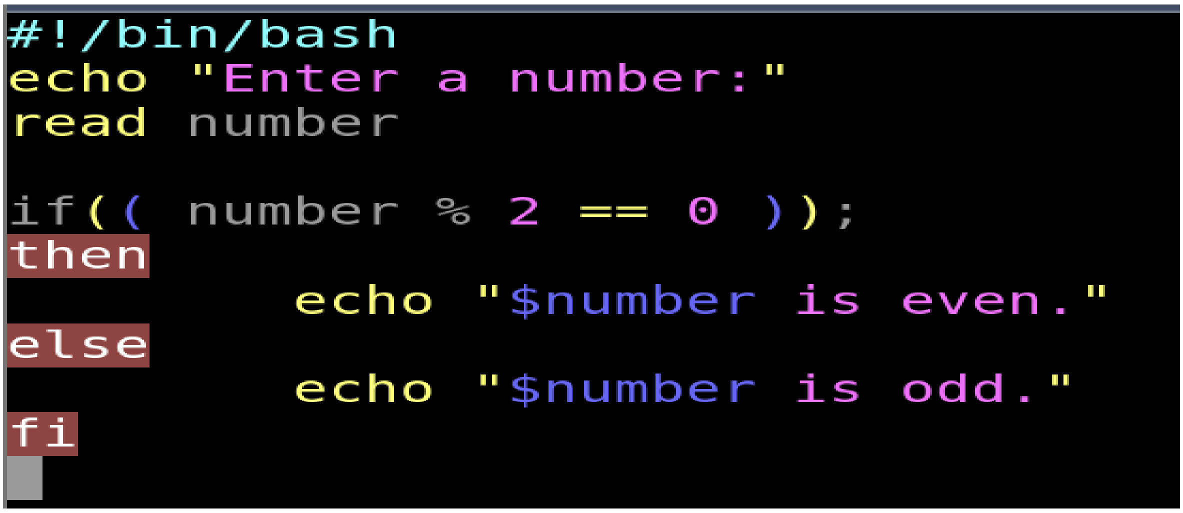
Output



```
tryhackme@linux1:~$ chmod 777 program4.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  program3.sh  program4.sh
tryhackme@linux1:~$ ./program4.sh
num1 =6
num2 =8
Sum is: 14
Sub is: -2
Product is: 48
Div is: 0
tryhackme@linux1:~$
```

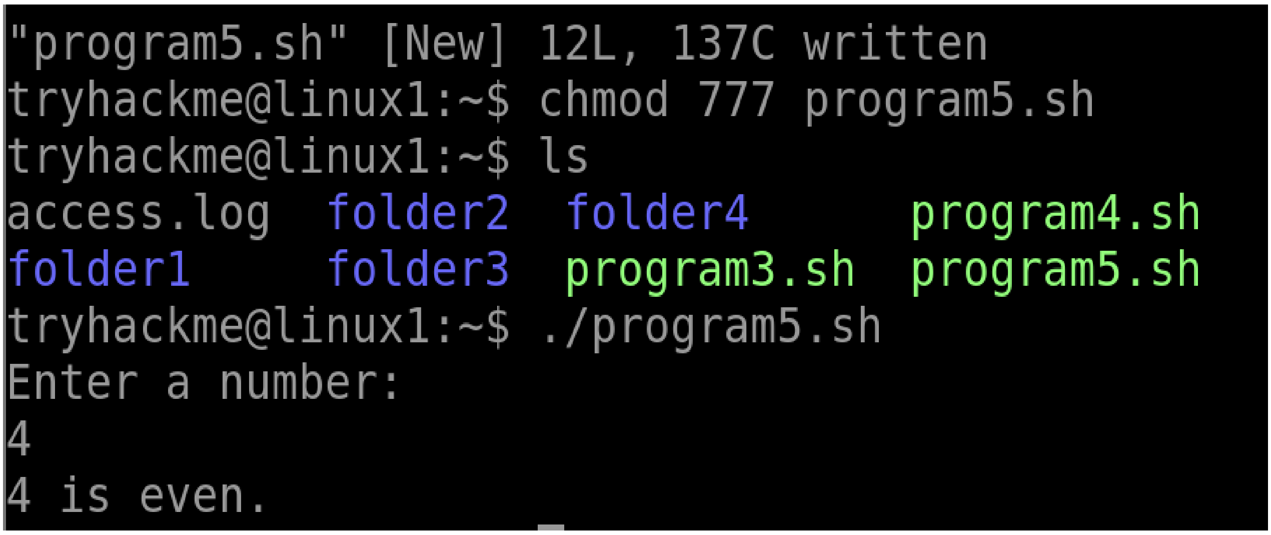
9.5 Write a shell script to check whether no is even or odd.

Program



```
#!/bin/bash
echo "Enter a number:"
read number

if(( number % 2 == 0 ));
then
        echo "$number is even."
else
        echo "$number is odd."
fi
```

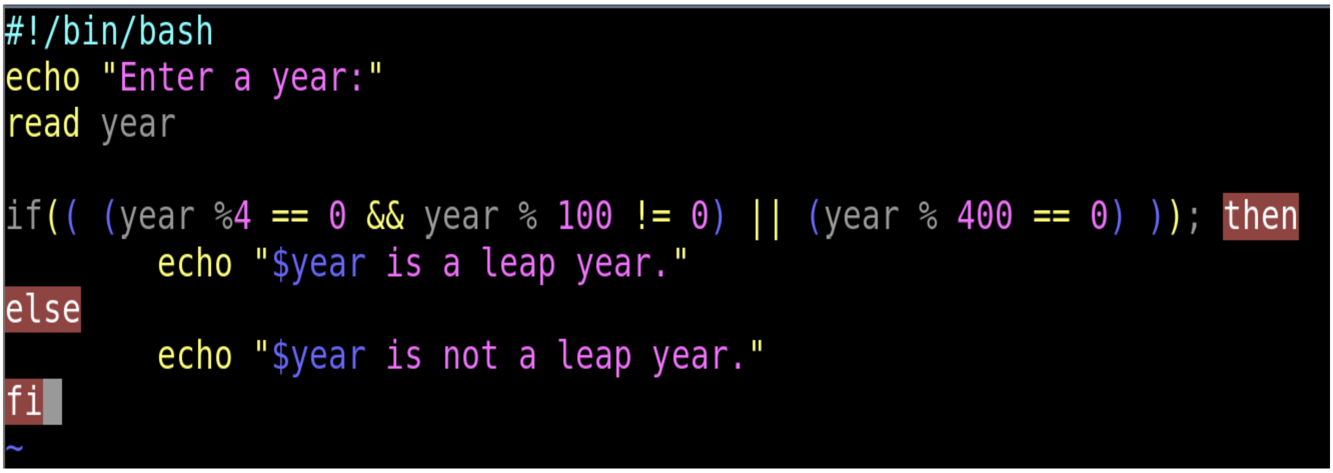
Output



```
"program5.sh" [New] 12L, 137C written
tryhackme@linux1:~$ chmod 777 program5.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4      program4.sh
folder1     folder3  program3.sh  program5.sh
tryhackme@linux1:~$ ./program5.sh
Enter a number:
4
4 is even.
```

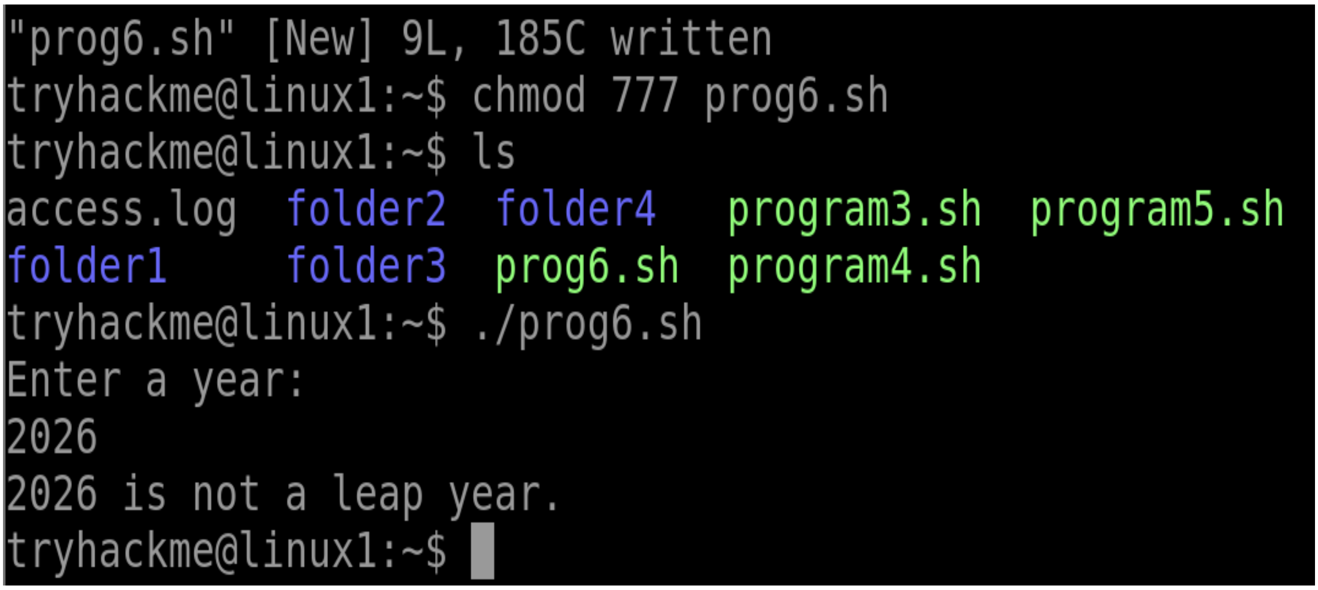
9.6 Write a shell script to check any input year is leap or not.

Program



```
#!/bin/bash
echo "Enter a year:"
read year

if(( (year %4 == 0 && year % 100 != 0) || (year % 400 == 0) )); then
        echo "$year is a leap year."
else
        echo "$year is not a leap year."
fi
~
```

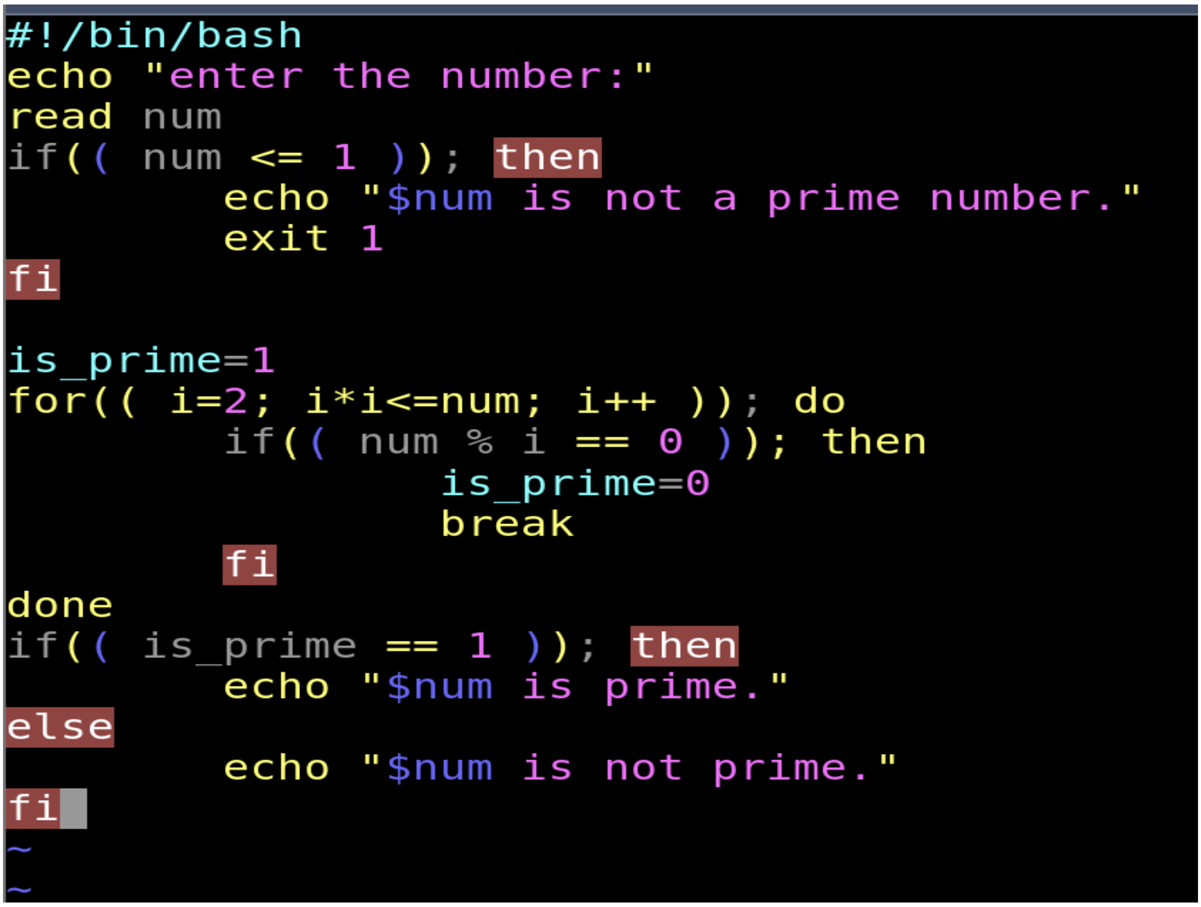
Output



```
"prog6.sh" [New] 9L, 185C written
tryhackme@linux1:~$ chmod 777 prog6.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4   program3.sh  program5.sh
folder1     folder3  prog6.sh  program4.sh
tryhackme@linux1:~$ ./prog6.sh
Enter a year:
2026
2026 is not a leap year.
tryhackme@linux1:~$
```

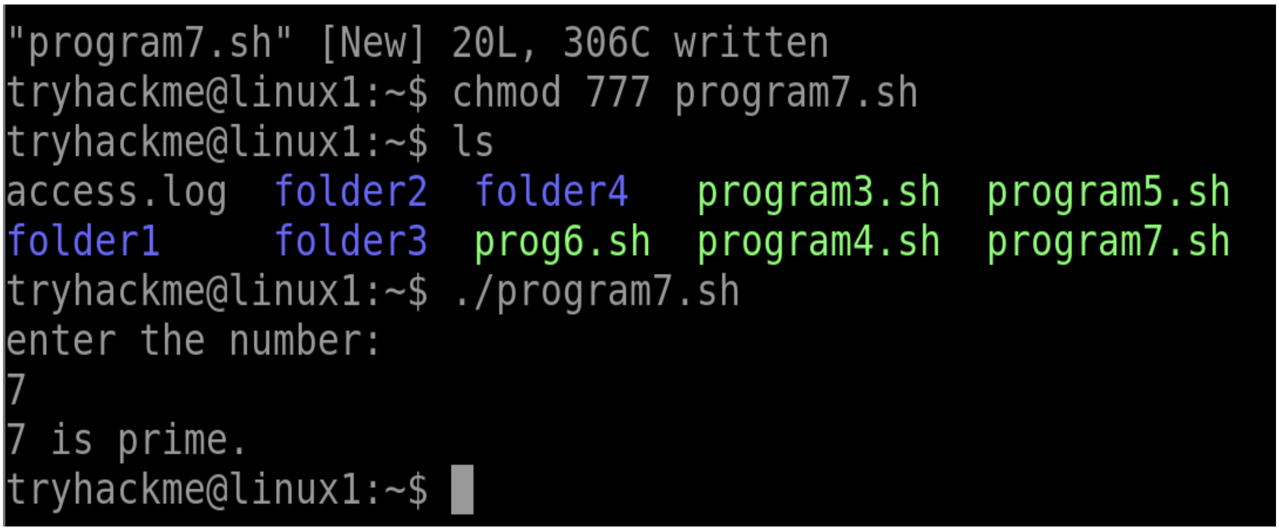
9.7 Write a shell script to check no is prime or not.

Program



```
#!/bin/bash
echo "enter the number:"
read num
if(( num <= 1 )); then
        echo "$num is not a prime number."
        exit 1
fi

is_prime=1
for(( i=2; i*i<=num; i++ )); do
        if(( num % i == 0 )); then
                is_prime=0
                break
        fi
done
if(( is_prime == 1 )); then
        echo "$num is prime."
else
        echo "$num is not prime."
fi
~
~
```

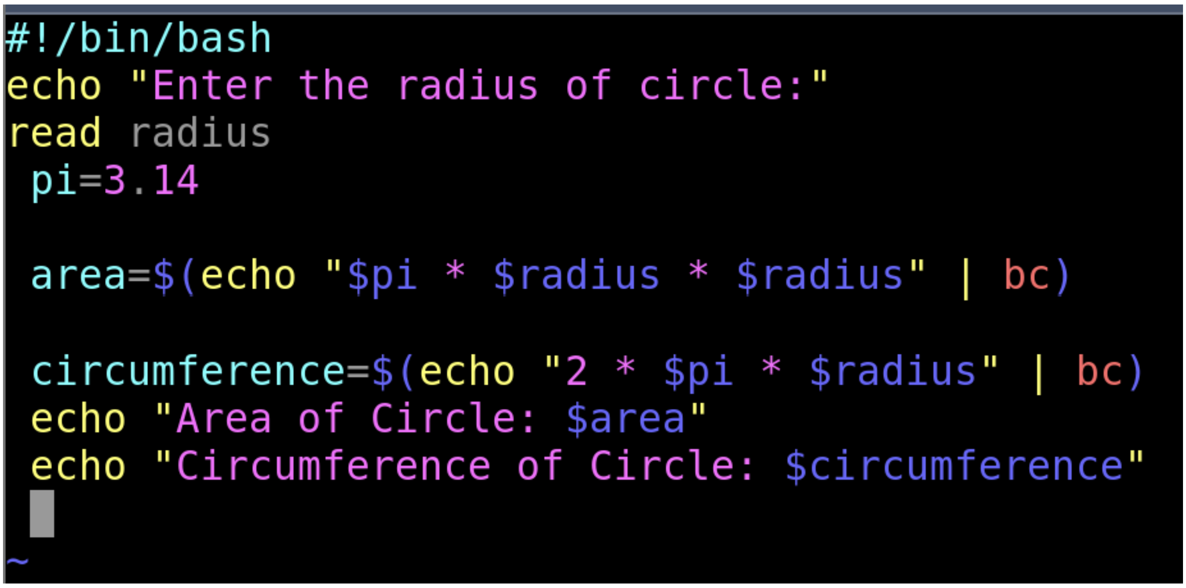
Output



```
"program7.sh" [New] 20L, 306C written
tryhackme@linux1:~$ chmod 777 program7.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4   program3.sh  program5.sh
folder1     folder3  prog6.sh  program4.sh  program7.sh
tryhackme@linux1:~$ ./program7.sh
enter the number:
7
7 is prime.
tryhackme@linux1:~$
```

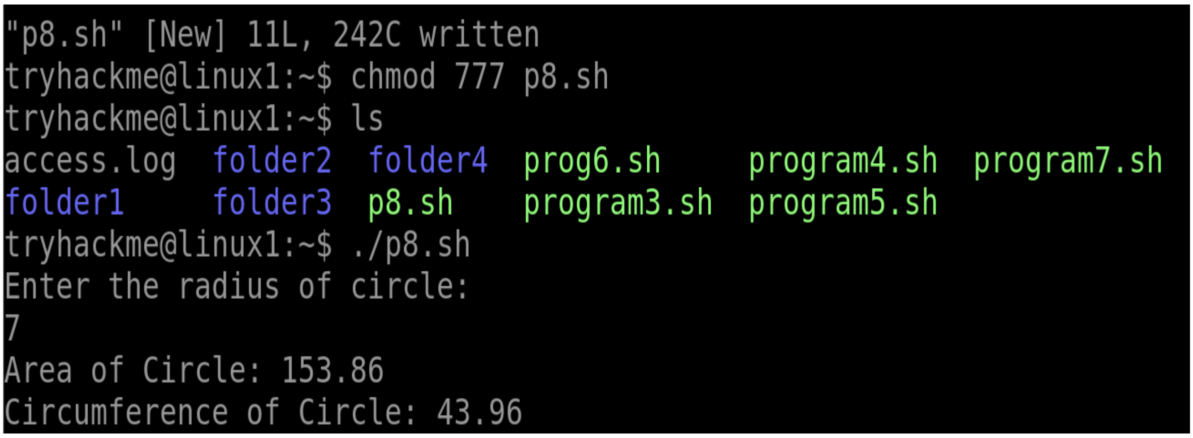
9.8 Write a shell script to find the area and circumference of a circle.

Program



```
#!/bin/bash
echo "Enter the radius of circle:"
read radius
 pi=3.14

 area=$(echo "$pi * $radius * $radius" | bc)

 circumference=$(echo "2 * $pi * $radius" | bc)
 echo "Area of Circle: $area"
 echo "Circumference of Circle: $circumference"

~
```

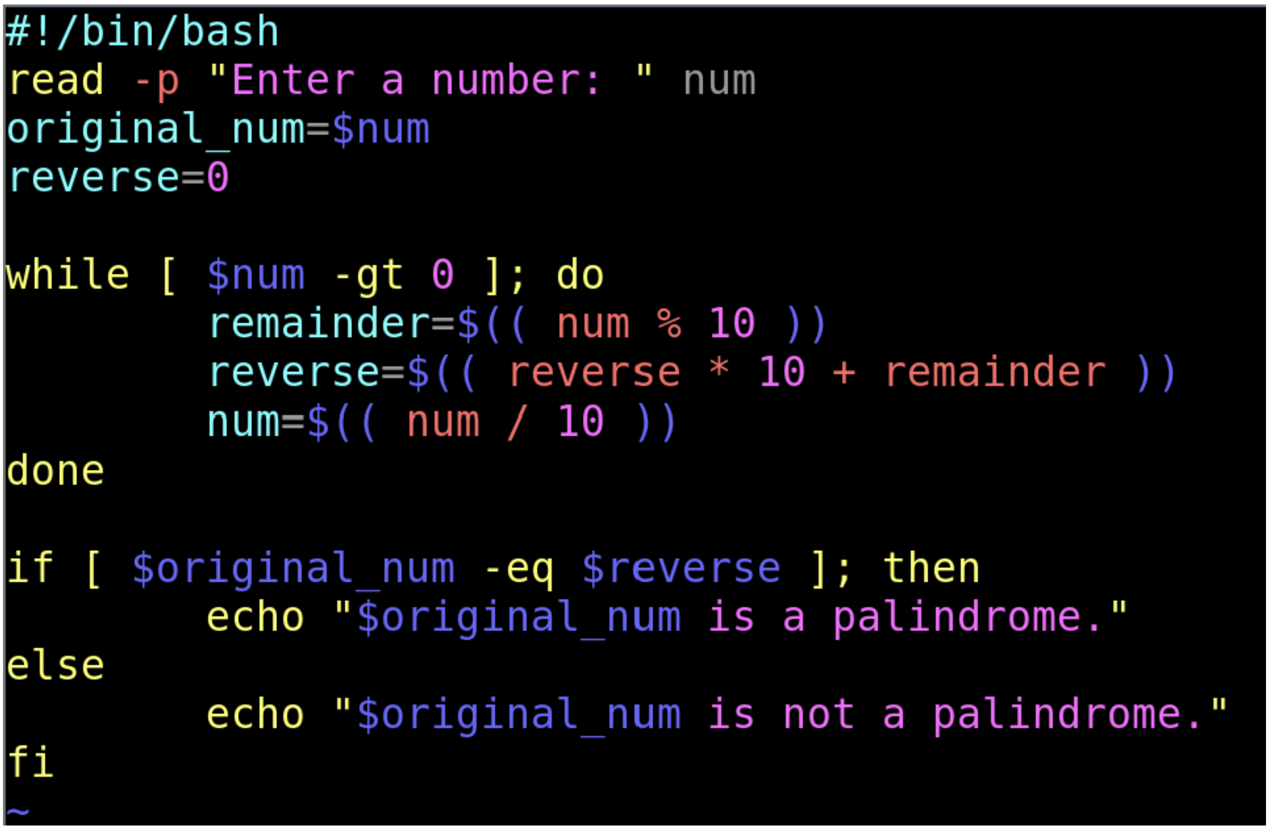
Output



```
"p8.sh" [New] 11L, 242C written
tryhackme@linux1:~$ chmod 777 p8.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4  prog6.sh     program4.sh  program7.sh
folder1     folder3  p8.sh    program3.sh  program5.sh
tryhackme@linux1:~$ ./p8.sh
Enter the radius of circle:
7
Area of Circle: 153.86
Circumference of Circle: 43.96
```

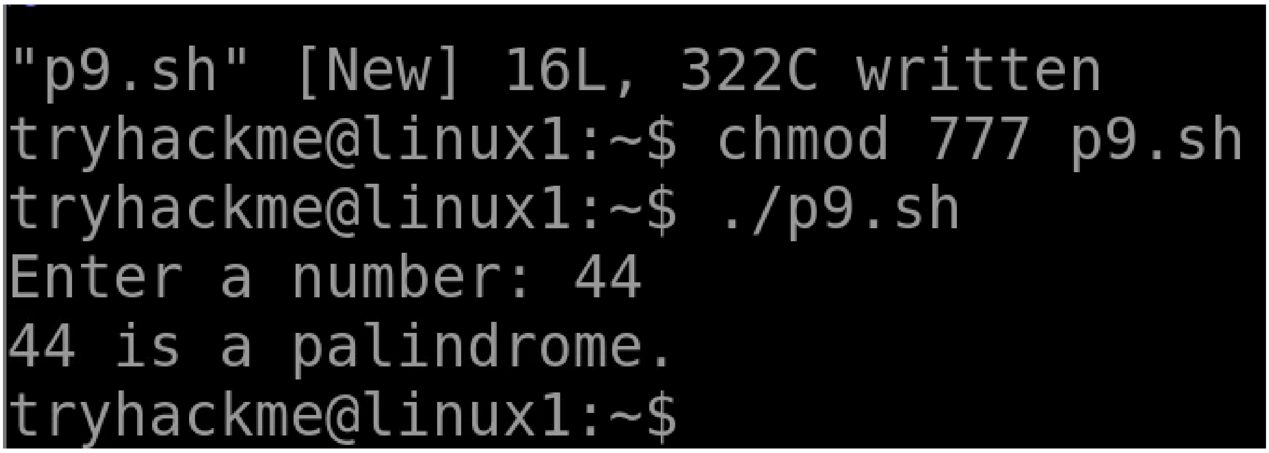
9.9 Write a shell script to check the given number and its reverse are same.

Program



```
#!/bin/bash
read -p "Enter a number: " num
original_num=$num
reverse=0

while [ $num -gt 0 ]; do
        remainder=$(( num % 10 ))
        reverse=$(( reverse * 10 + remainder ))
        num=$(( num / 10 ))
done

if [ $original_num -eq $reverse ]; then
        echo "$original_num is a palindrome."
else
        echo "$original_num is not a palindrome."
fi
~
```

Output

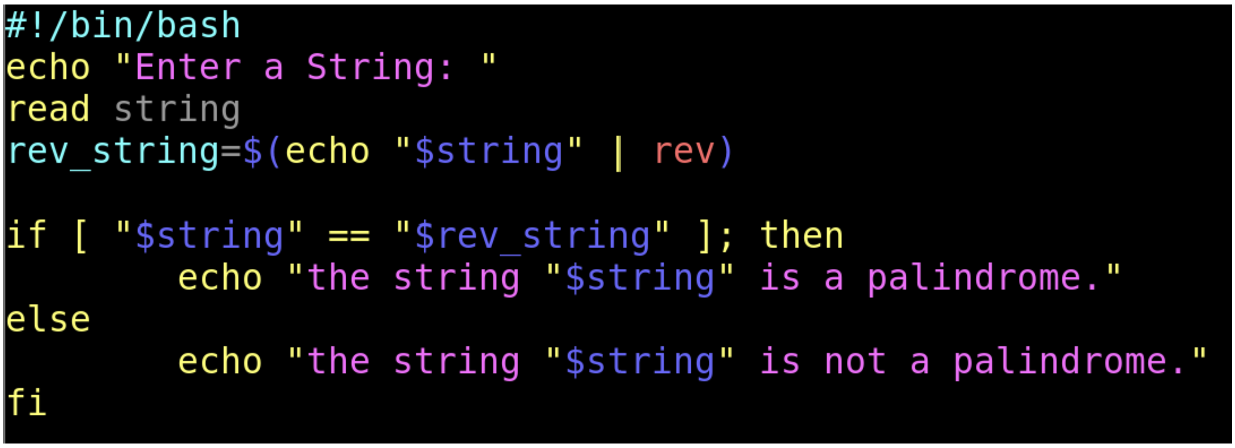


```
"p9.sh" [New] 16L, 322C written
tryhackme@linux1:~$ chmod 777 p9.sh
tryhackme@linux1:~$ ./p9.sh
Enter a number: 44
44 is a palindrome.
tryhackme@linux1:~$
```

9.10 Write a shell script to check the given string is palindrome or not.

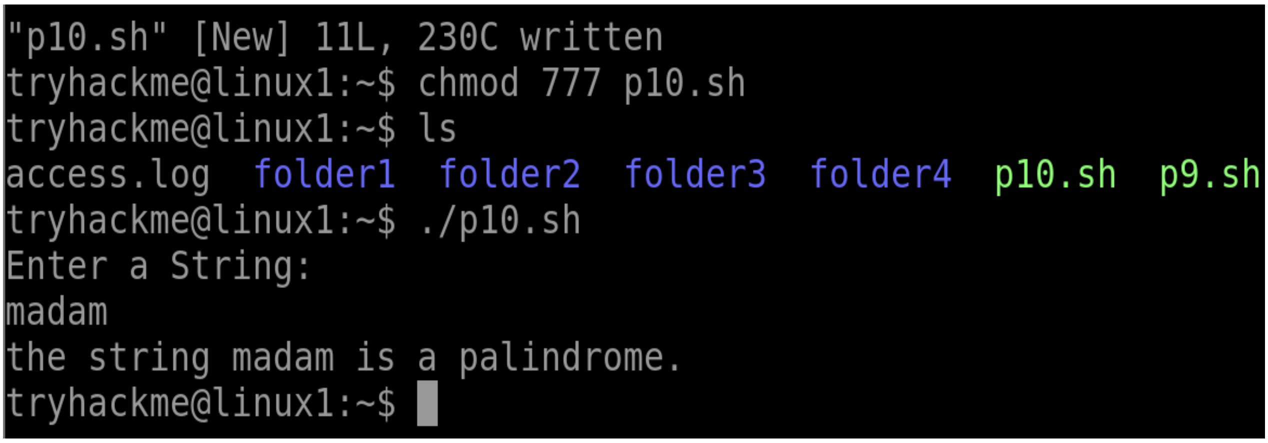
Example :madam.

Program



```
#!/bin/bash
echo "Enter a String: "
read string
rev_string=$(echo "$string" | rev)

if [ "$string" == "$rev_string" ]; then
        echo "the string "$string" is a palindrome."
else
        echo "the string "$string" is not a palindrome."
fi
```

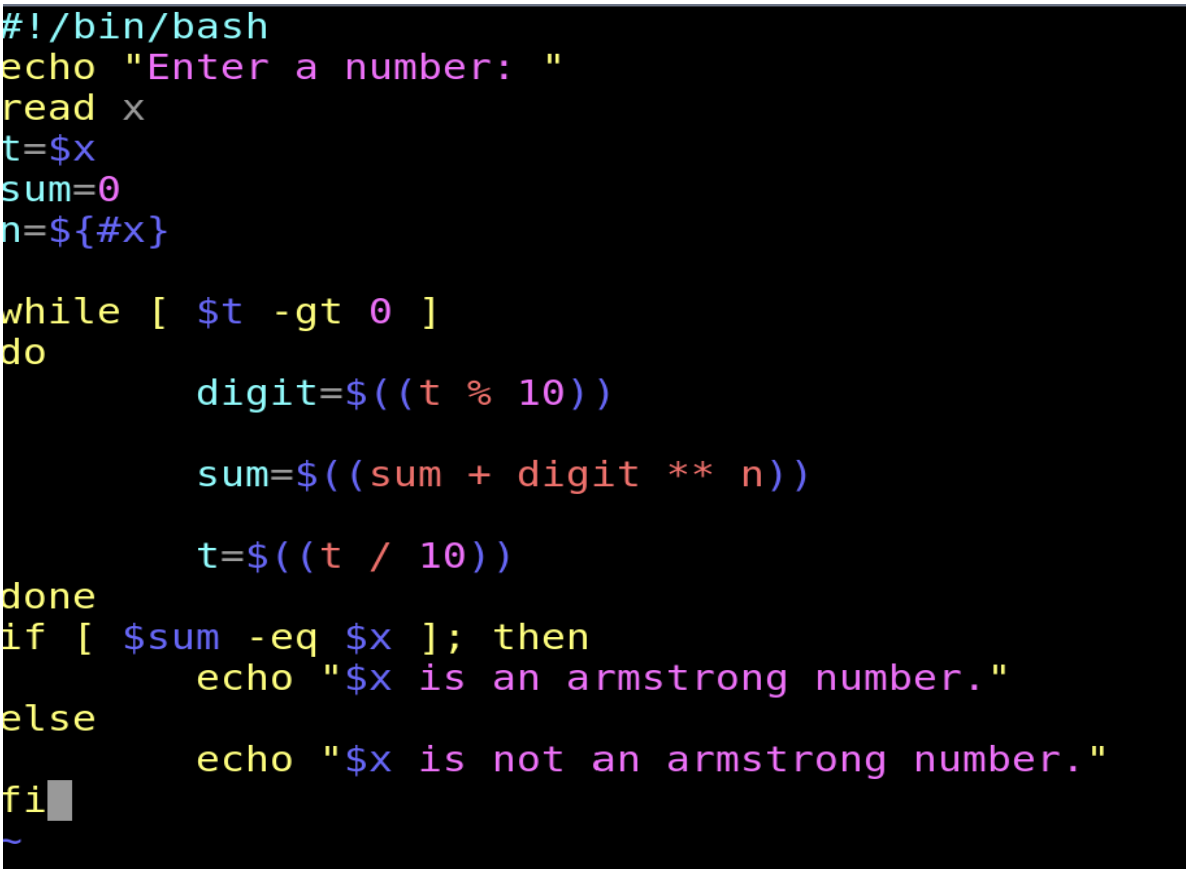
Output



```
"p10.sh" [New] 11L, 230C written
tryhackme@linux1:~$ chmod 777 p10.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  p10.sh  p9.sh
tryhackme@linux1:~$ ./p10.sh
Enter a String:
madam
the string madam is a palindrome.
tryhackme@linux1:~$ 
```

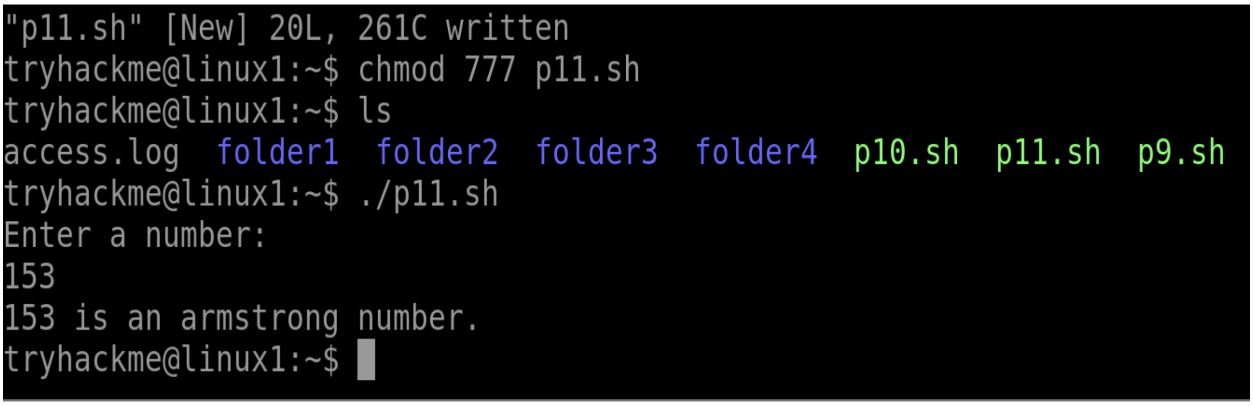
9.11 Write a shell script to check the given integer is Armstrong number or not.

Program



```
#!/bin/bash
echo "Enter a number: "
read x
t=$x
sum=0
n=${#x}

while [ $t -gt 0 ]
do
        digit=$((t % 10))

        sum=$((sum + digit ** n))

        t=$((t / 10))
done
if [ $sum -eq $x ]; then
        echo "$x is an armstrong number."
else
        echo "$x is not an armstrong number."
fi
~
```

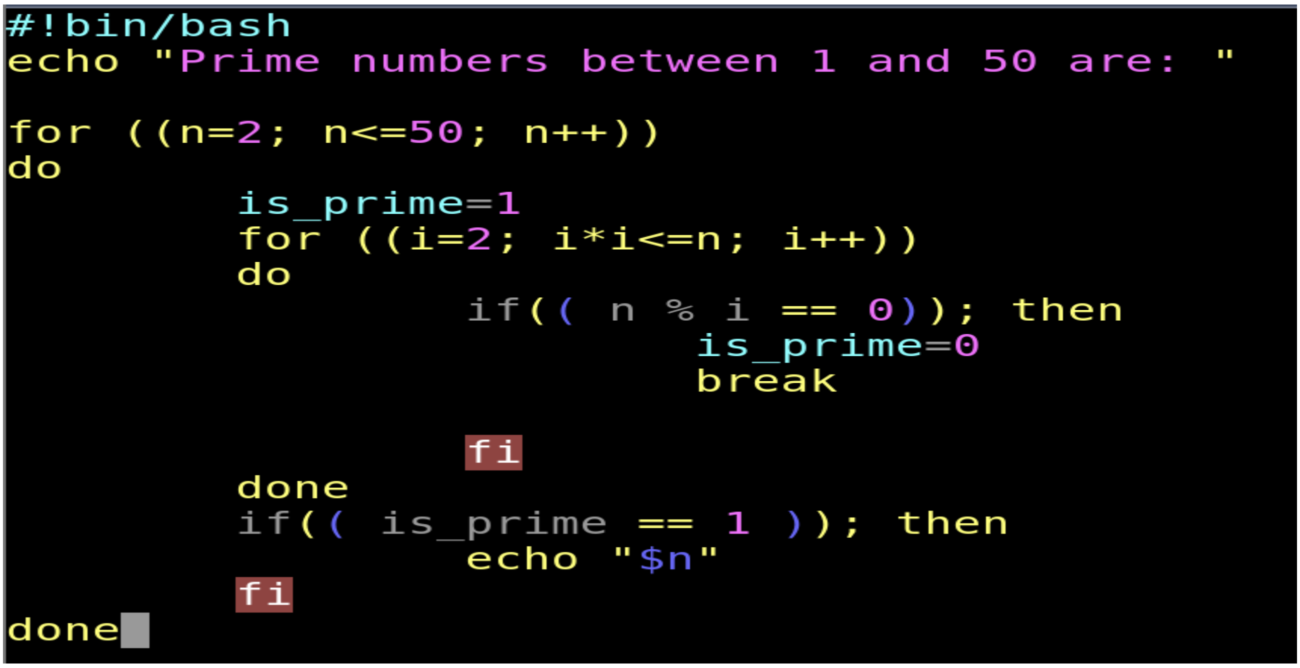
Output



```
"p11.sh" [New] 20L, 261C written
tryhackme@linux1:~$ chmod 777 p11.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  p10.sh  p11.sh  p9.sh
tryhackme@linux1:~$ ./p11.sh
Enter a number:
153
153 is an armstrong number.
tryhackme@linux1:~$
```

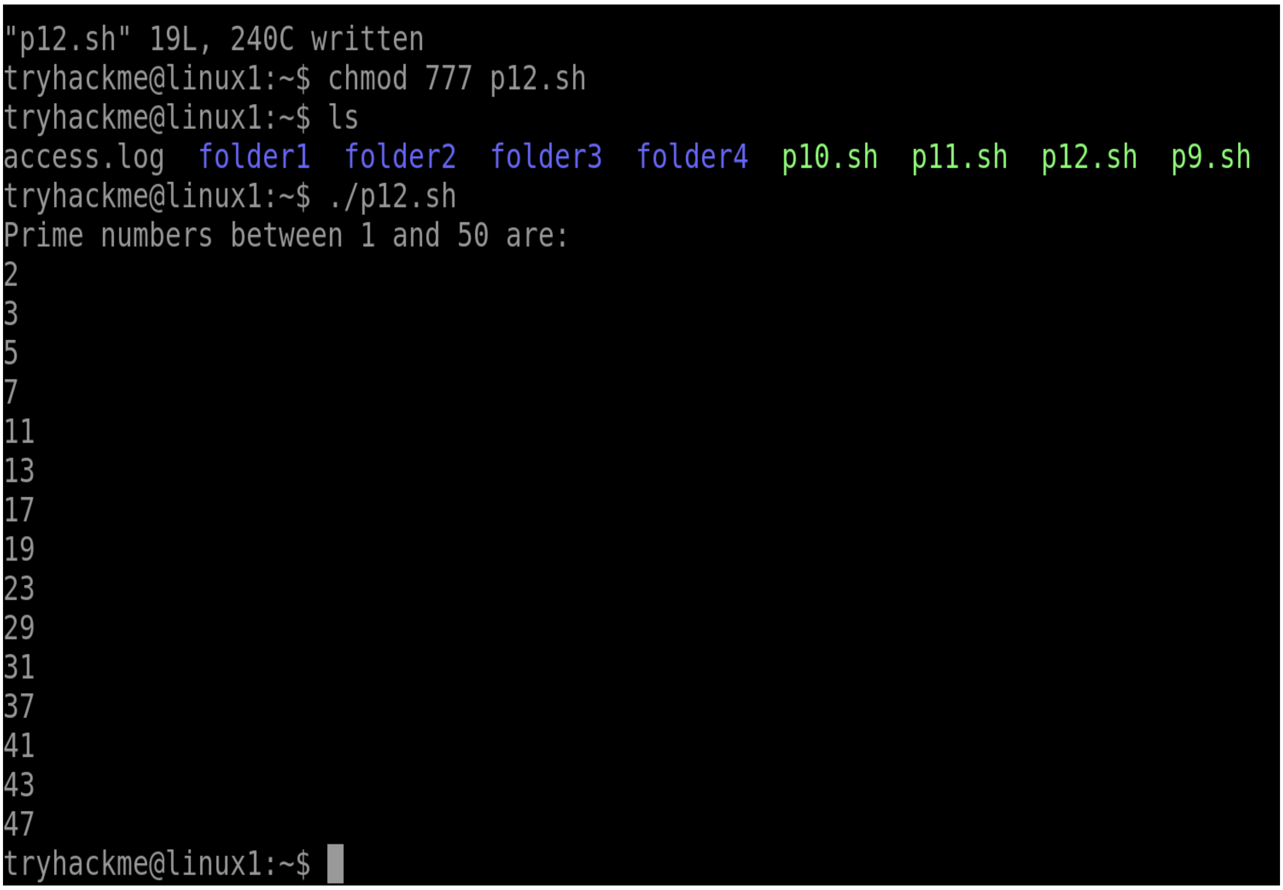
9.12 Write a Shell Script to generate prime numbers between 1 and 50.

Program



```
#!bin/bash
echo "Prime numbers between 1 and 50 are: "

for ((n=2; n<=50; n++))
do
        is_prime=1
        for ((i=2; i*i<=n; i++))
        do
                if(( n % i == 0)); then
                        is_prime=0
                        break

                fi
        done
        if(( is_prime == 1 )); then
                echo "$n"
        fi
done
```

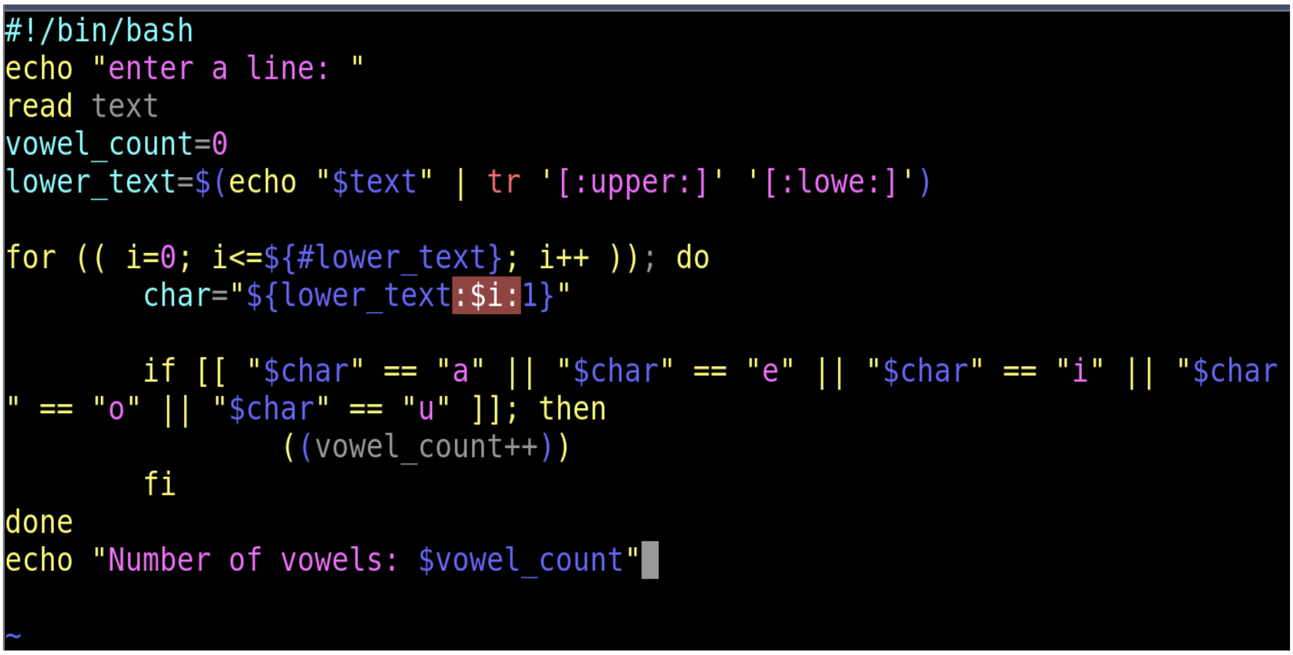
Output



```
"p12.sh" 19L, 240C written
tryhackme@linux1:~$ chmod 777 p12.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  p10.sh  p11.sh  p12.sh  p9.sh
tryhackme@linux1:~$ ./p12.sh
Prime numbers between 1 and 50 are:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
tryhackme@linux1:~$
```

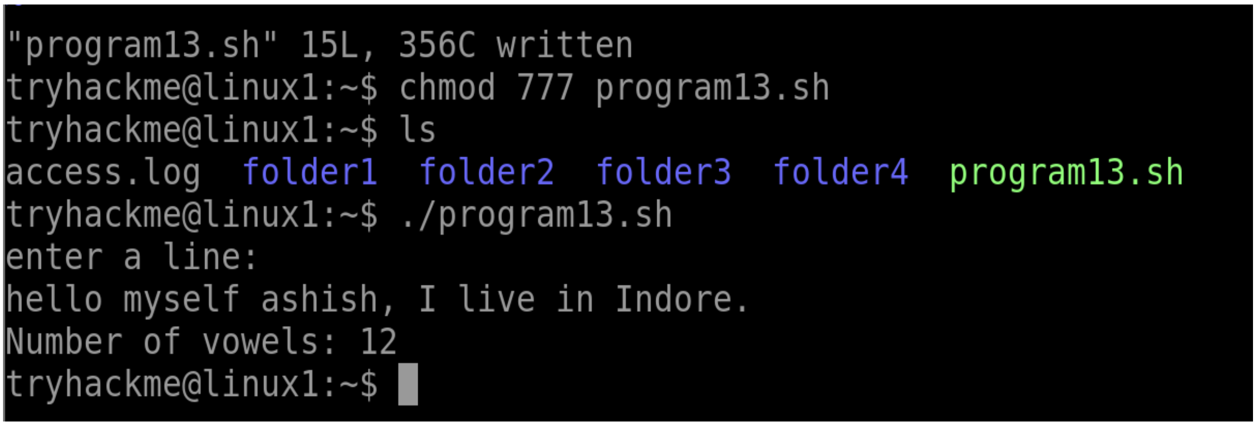
9.13 Write a Shell Script to count the number of vowels in a line of text.

Program



```
#!/bin/bash
echo "enter a line: "
read text
vowel_count=0
lower_text=$(echo "$text" | tr '[:upper:]' '[:lowe:]')

for (( i=0; i<=${#lower_text}; i++ )); do
        char="${lower_text:$i:1}"

        if [[ "$char" == "a" || "$char" == "e" || "$char" == "i" || "$char
" == "o" || "$char" == "u" ]]; then
                ((vowel_count++))
        fi
done
echo "Number of vowels: $vowel_count"

~
```

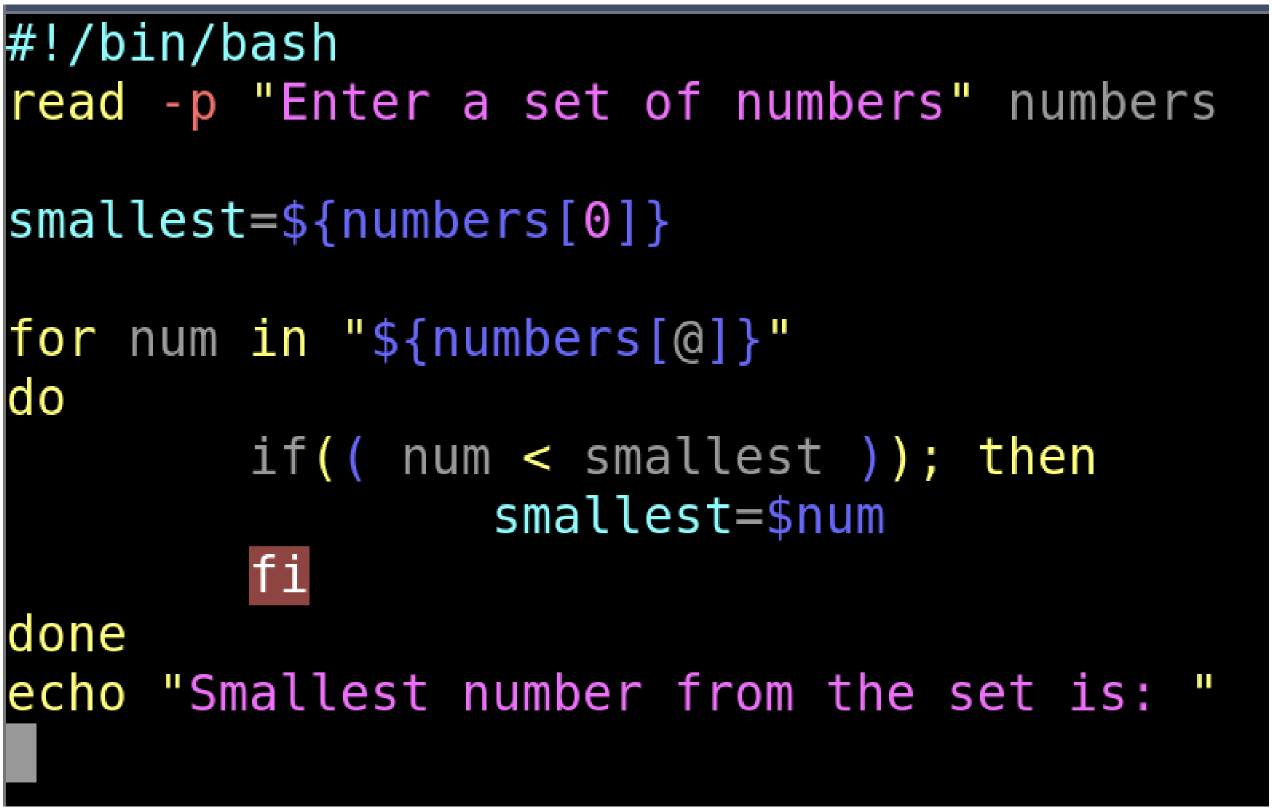
Output



```
"program13.sh" 15L, 356C written
tryhackme@linux1:~$ chmod 777 program13.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  program13.sh
tryhackme@linux1:~$ ./program13.sh
enter a line:
hello myself ashish, I live in Indore.
Number of vowels: 12
tryhackme@linux1:~$
```

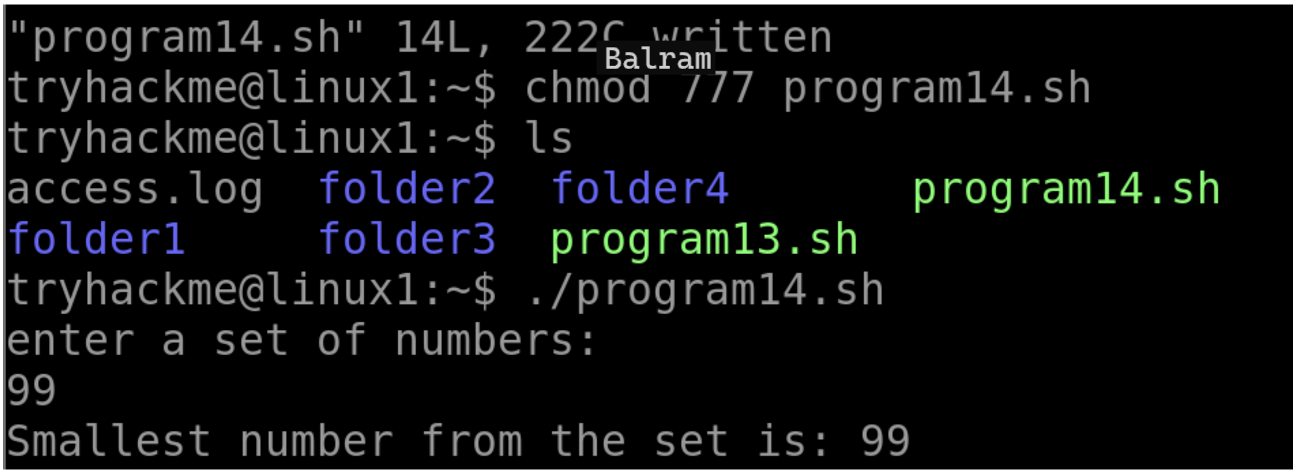
9.14 Write a Shell Script to find the smallest number from a set of numbers.

Programs



```
#!/bin/bash
read -p "Enter a set of numbers" numbers

smallest=${numbers[0]}

for num in "${numbers[@]}"
do
        if(( num < smallest )); then
                smallest=$num
        fi
done
echo "Smallest number from the set is: "
```

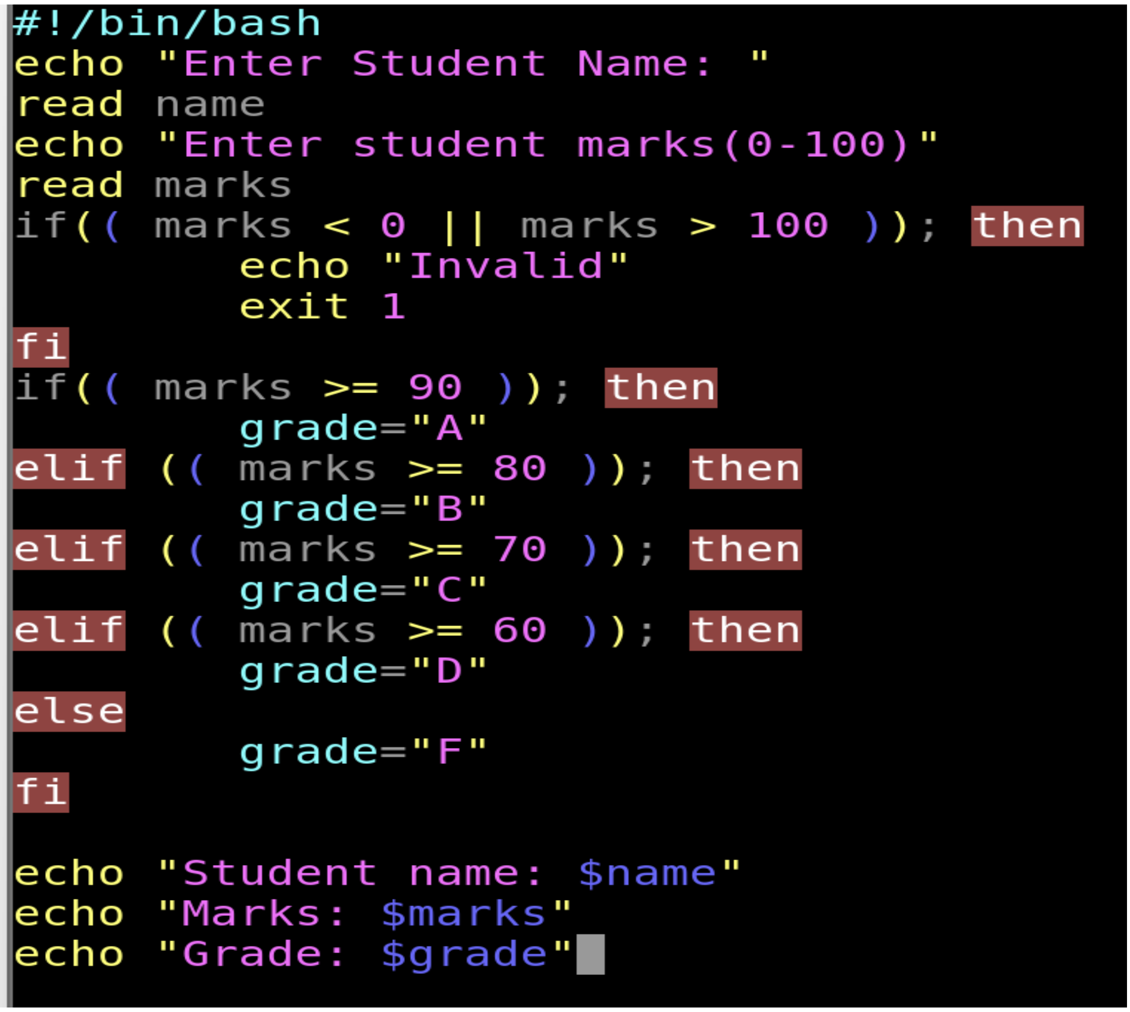
Output



```
"program14.sh" 14L, 222C written
tryhackme@linux1:~$ chmod 777 program14.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4       program14.sh
folder1     folder3  program13.sh
tryhackme@linux1:~$ ./program14.sh
enter a set of numbers:
99
Smallest number from the set is: 99
```

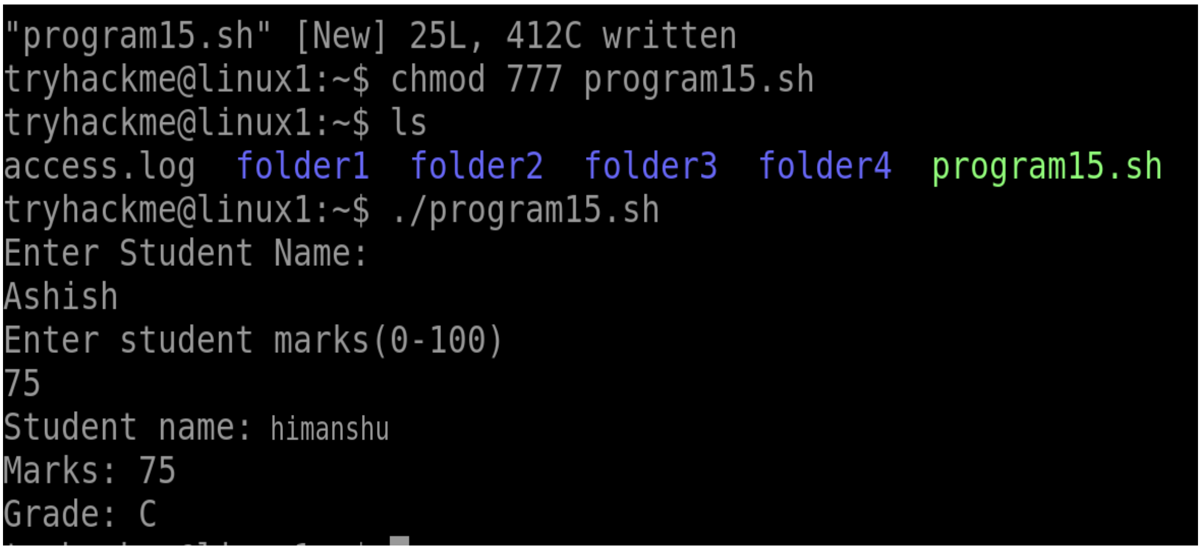
9.15 Write a Shell Script to display student grades

Program



```
#!/bin/bash
echo "Enter Student Name: "
read name
echo "Enter student marks(0-100)"
read marks
if(( marks < 0 || marks > 100 )); then
        echo "Invalid"
        exit 1
fi
if(( marks >= 90 )); then
        grade="A"
elif (( marks >= 80 )); then
        grade="B"
elif (( marks >= 70 )); then
        grade="C"
elif (( marks >= 60 )); then
        grade="D"
else
        grade="F"
fi

echo "Student name: $name"
echo "Marks: $marks"
echo "Grade: $grade"
```

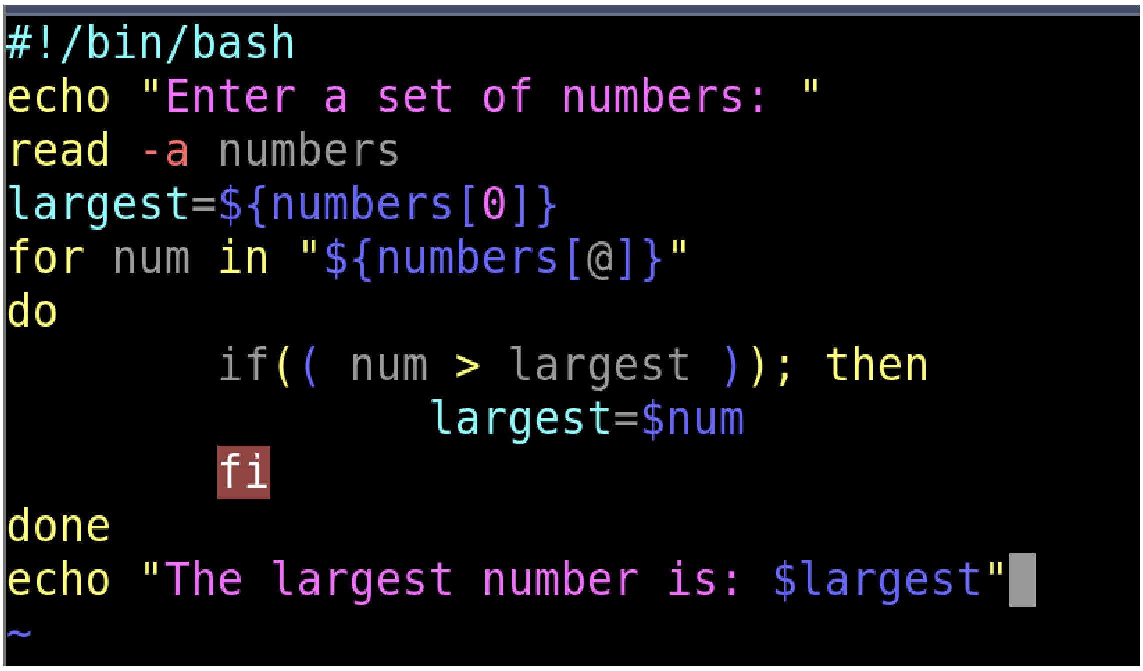
Output



```
"program15.sh" [New] 25L, 412C written
tryhackme@linux1:~$ chmod 777 program15.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  program15.sh
tryhackme@linux1:~$ ./program15.sh
Enter Student Name:
Ashish
Enter student marks(0-100)
75
Student name: himanshu
Marks: 75
Grade: C
```

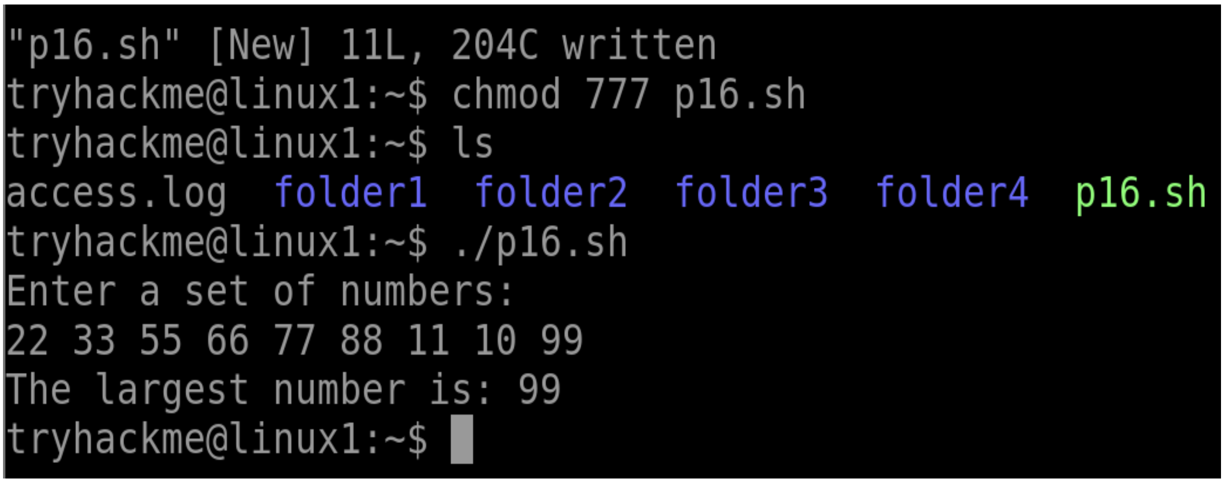
9.16 Write a Shell Script to find the largest number from a set of numbers.

Program



```
#!/bin/bash
echo "Enter a set of numbers: "
read -a numbers
largest=${numbers[0]}
for num in "${numbers[@]}"
do
        if(( num > largest )); then
                largest=$num
        fi
done
echo "The largest number is: $largest"
~
```

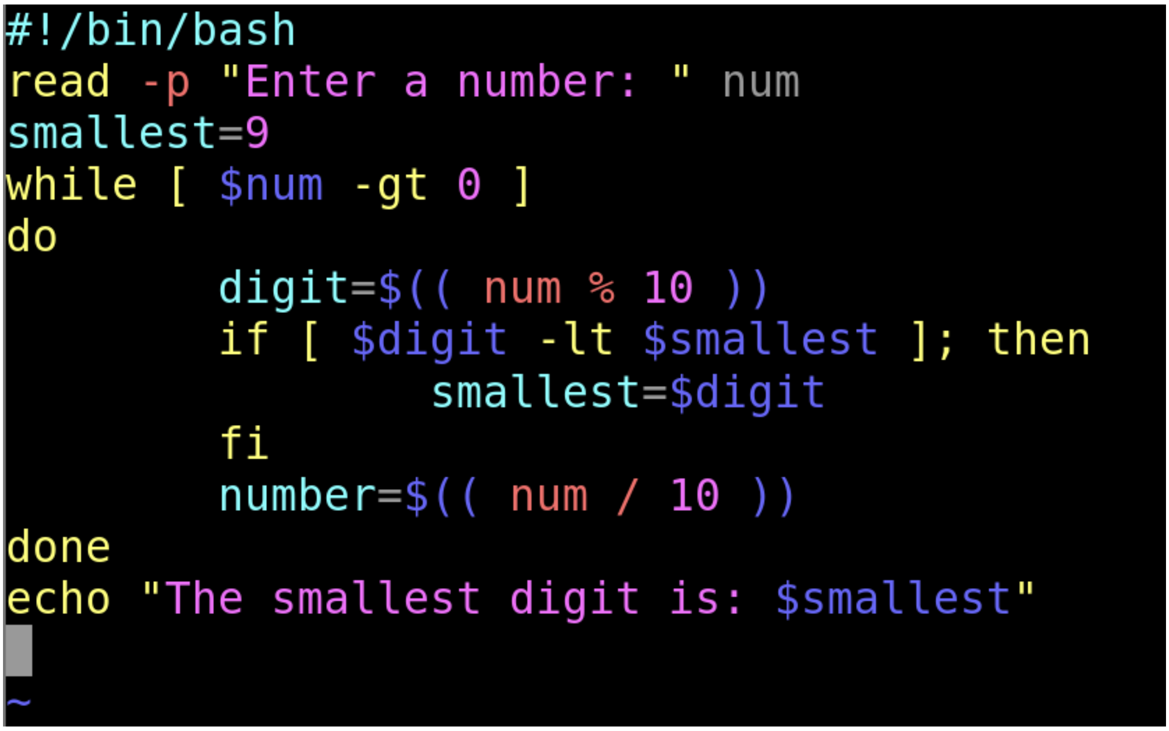
Output



```
"p16.sh" [New] 11L, 204C written
tryhackme@linux1:~$ chmod 777 p16.sh
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4  p16.sh
tryhackme@linux1:~$ ./p16.sh
Enter a set of numbers:
22 33 55 66 77 88 11 10 99
The largest number is: 99
tryhackme@linux1:~$
```

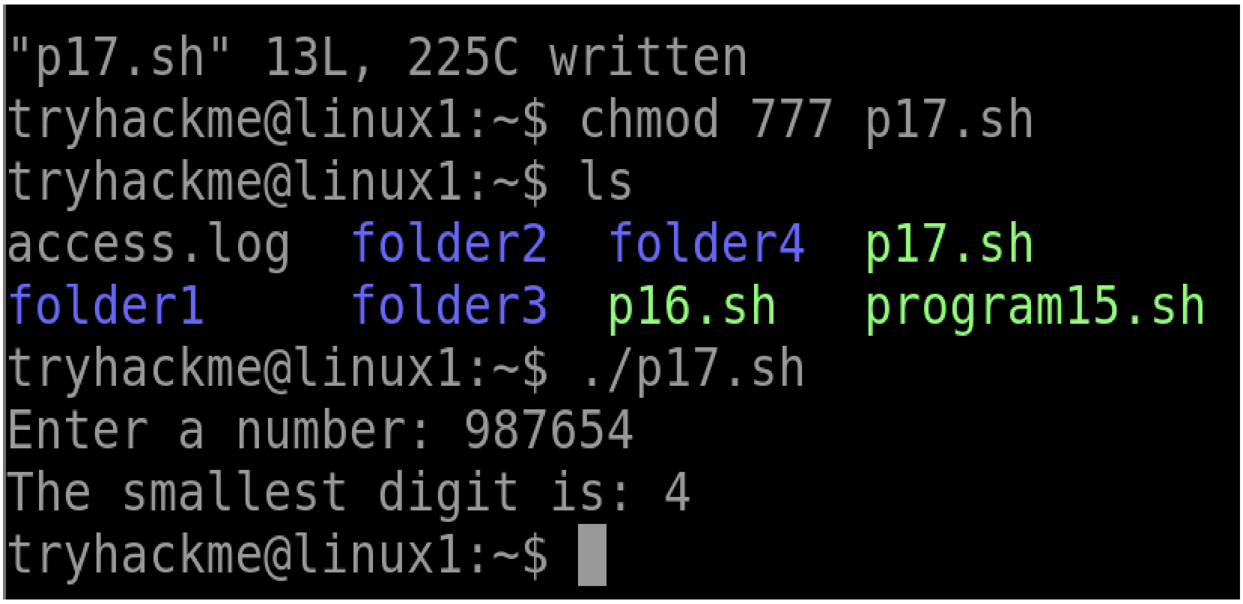
9.17 Write a Shell Script to find the smallest digit from a number.

Program



```
#!/bin/bash
read -p "Enter a number: " num
smallest=9
while [ $num -gt 0 ]
do
        digit=$(( num % 10 ))
        if [ $digit -lt $smallest ]; then
                smallest=$digit
        fi
        number=$(( num / 10 ))
done
echo "The smallest digit is: $smallest"

~
```

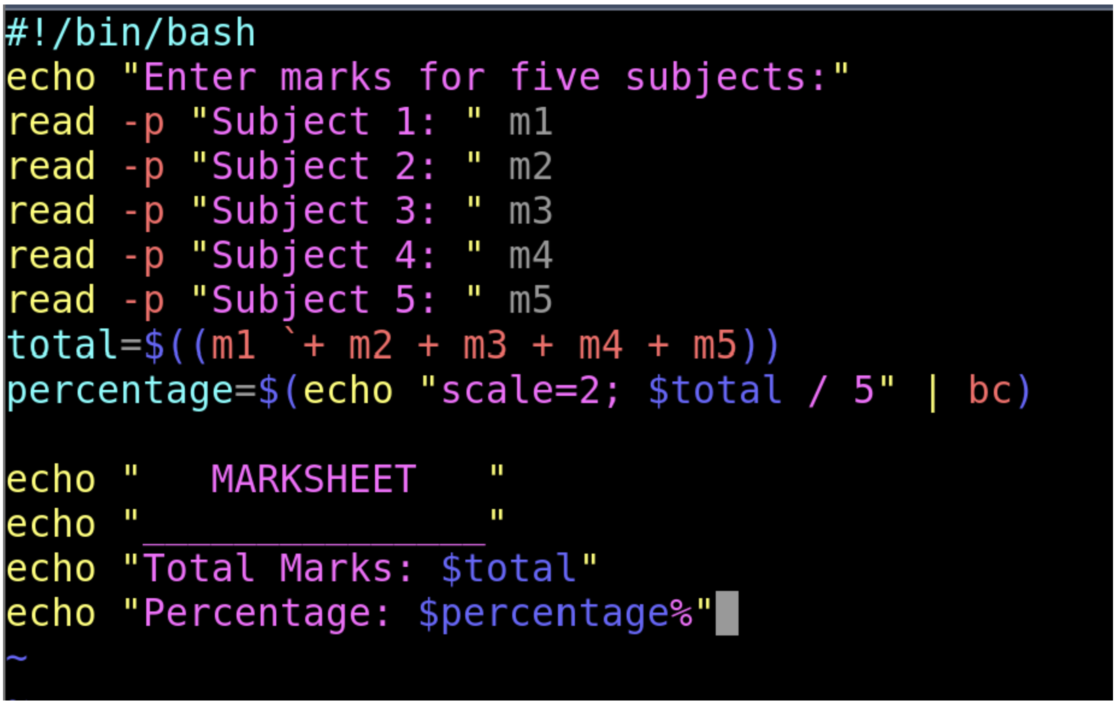
Output



```
"p17.sh" 13L, 225C written
tryhackme@linux1:~$ chmod 777 p17.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4  p17.sh
folder1     folder3  p16.sh   program15.sh
tryhackme@linux1:~$ ./p17.sh
Enter a number: 987654
The smallest digit is: 4
tryhackme@linux1:~$
```

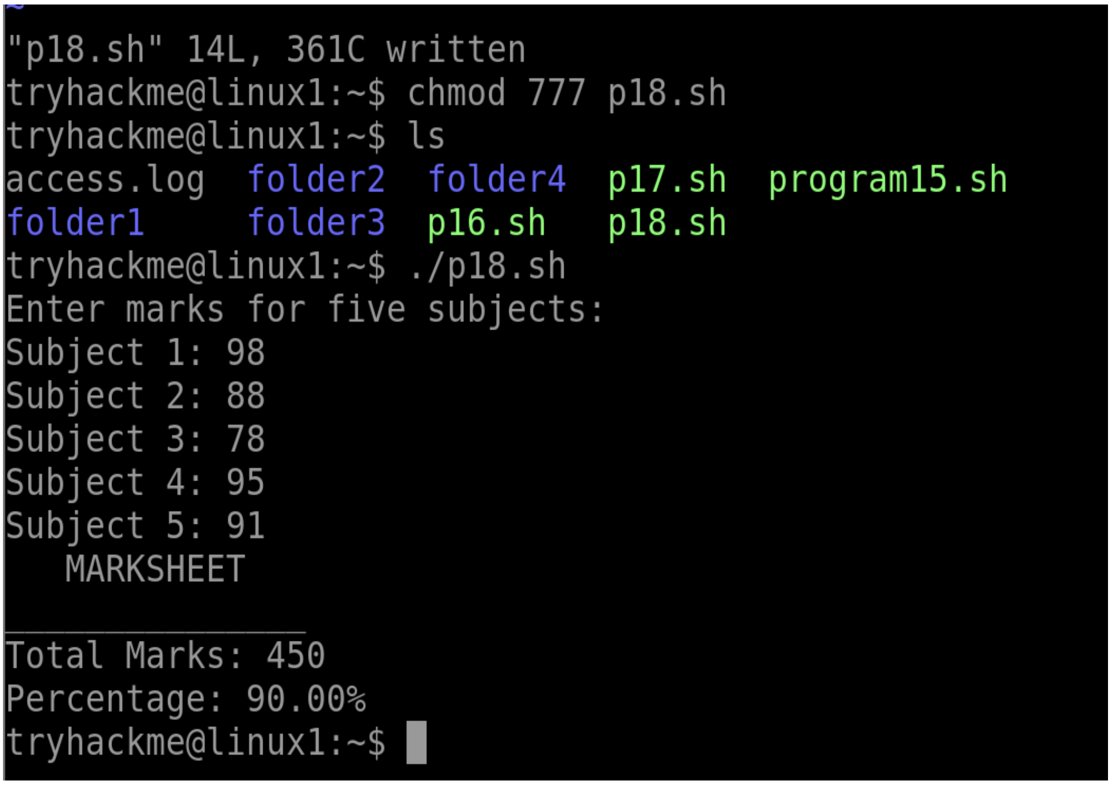
9.18 Write a Shell Script to print marksheet, that takes five subjects marks at runtime and calculate the percentage of a student

Program



```
#!/bin/bash
echo "Enter marks for five subjects:"
read -p "Subject 1: " m1
read -p "Subject 2: " m2
read -p "Subject 3: " m3
read -p "Subject 4: " m4
read -p "Subject 5: " m5
total=$((m1 `+ m2 + m3 + m4 + m5))
percentage=$(echo "scale=2; $total / 5" | bc)

echo "   MARKSHEET    "
echo "_______________ "
echo "Total Marks: $total"
echo "Percentage: $percentage%"
~
```

Output



```
"p18.sh" 14L, 361C written
tryhackme@linux1:~$ chmod 777 p18.sh
tryhackme@linux1:~$ ls
access.log  folder2  folder4  p17.sh  program15.sh
folder1     folder3  p16.sh   p18.sh
tryhackme@linux1:~$ ./p18.sh
Enter marks for five subjects:
Subject 1: 98
Subject 2: 88
Subject 3: 78
Subject 4: 95
Subject 5: 91
   MARKSHEET

_______________
Total Marks: 450
Percentage: 90.00%
tryhackme@linux1:~$
```

Experiment 10

Aim:- Case Study: Setting Up and Configuring Apache Tomcat, Samba, DNS/LDAP Services, Firewall, and Proxy Server Configuration in a Networked Environment

Overview

This case study focuses on the implementation of a networked environment that includes Apache Tomcat for web applications, Samba for file sharing, DNS and LDAP for directory services, firewall configurations for security, and proxy server settings for controlled internet access. The environment is designed to support a medium-sized organization with a focus on security, performance, and ease of management.

Objectives

1. Deploy Apache Tomcat for hosting Java-based web applications.
2. Set up Samba for file sharing across different operating systems.
3. Implement DNS and LDAP for centralized user management and resource location.
4. Configure a firewall to secure the network.
5. Set up a proxy server for controlled internet access and monitoring.

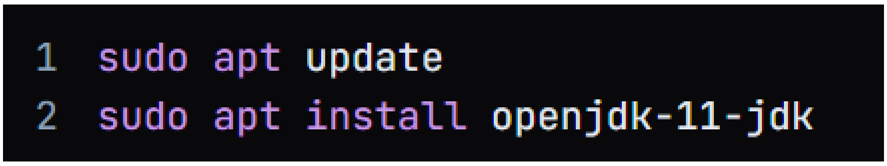
Environment Setup

- **Hardware**: A server with at least 16GB RAM, 4 CPU cores, and 500GB of storage.
- **Operating System**: Ubuntu Server 22.04 LTS.
- **Network Configuration**: Static IP addressing for servers, DHCP for client machines.

1. Deploying Apache Tomcat

Installation Steps:

1. **Install Java Development Kit (JDK)**

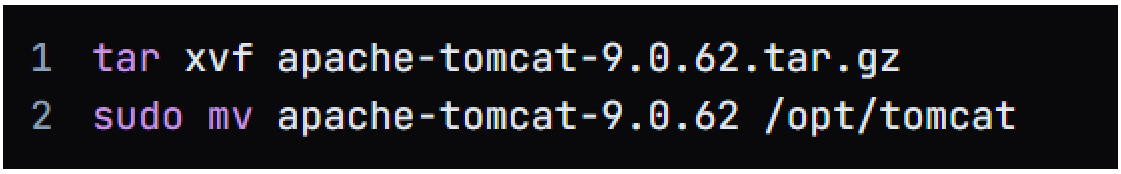


```
sudo apt update
sudo apt install openjdk-11-jdk
```

2. **Download Apache Tomcat**:

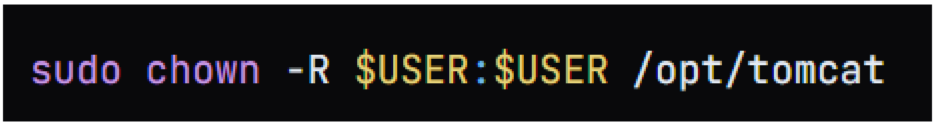
1. wget https://downloads.apache.org/tomcat/tomcat-9/v9.0.62/bin/apache-tomcat-9.0.62.tar.gz

3. **Extract and Configure**:



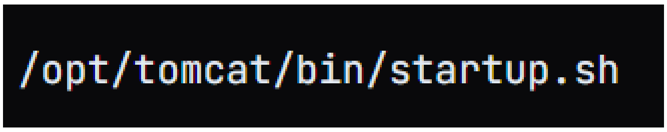
```
tar xvf apache-tomcat-9.0.62.tar.gz
sudo mv apache-tomcat-9.0.62 /opt/tomcat
```

4. **Set Permissions**



```
sudo chown -R $USER:$USER /opt/tomcat
```

5. **Start Tomcat**:



```
/opt/tomcat/bin/startup.sh
```

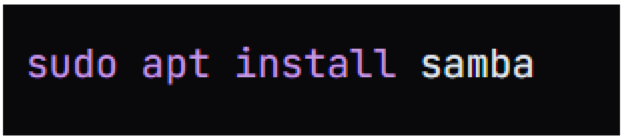
Configuration:

- Edit “server.xml” to configure the port and context paths.
- Ensure proper security settings in “web.xml” for web applications.

2. Setting Up Samba

Installation Steps:

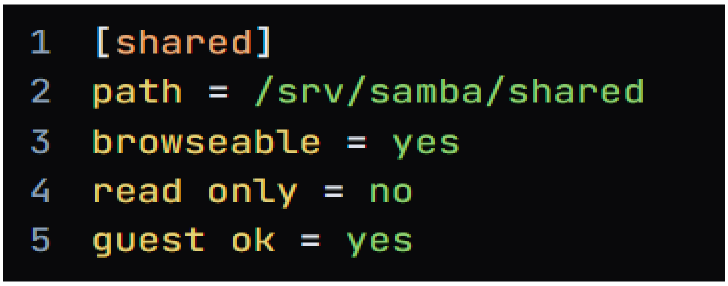
1. **Install Samba**



```
sudo apt install samba
```

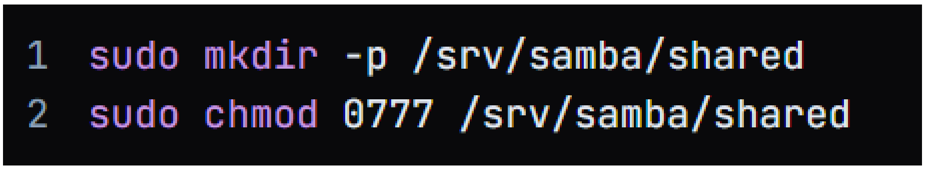
2. **Configure Samba:**

Edit /etc/samba/smb.conf to create a share**:**



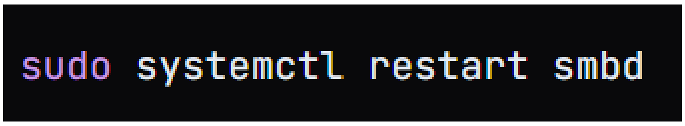
```
[shared]
path = /srv/samba/shared
browseable = yes
read only = no
guest ok = yes
```

3. **Create Shared Directory**:



```
sudo mkdir -p /srv/samba/shared
sudo chmod 0777 /srv/samba/shared
```

4. **Restart Samba Service**

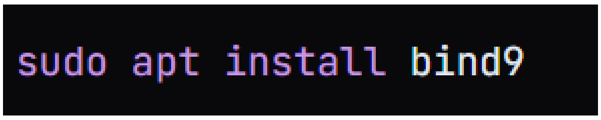


```
sudo systemctl restart smbd
```

3. Implementing DNS and LDAP

Installation Steps for DNS (Bind9):

1. **Install Bind9**

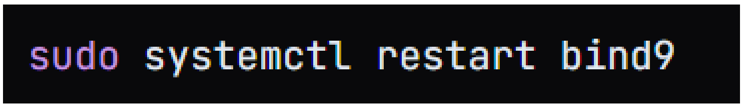


```
sudo apt install bind9
```

2. **Configure Zone Files:**

   Edit /etc/bind/named.conf.local to add zones.

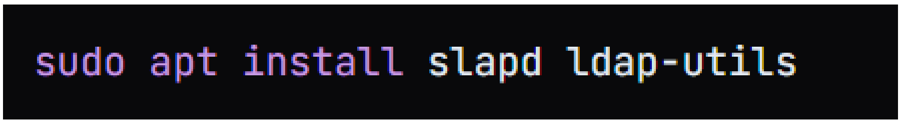
3. **Restart Bind9**



```
sudo systemctl restart bind9
```

Installation Steps for LDAP (OpenLDAP):

1. **Install OpenLDAP**



```
sudo apt install slapd ldap-utils
```

2. **Configure LDAP**:

   Use dpkg-reconfigure slapd to set up the domain and admin password.

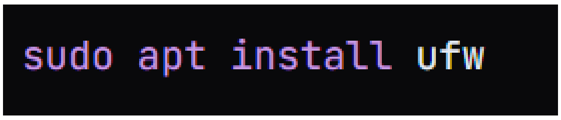
3. **Add Users:**

   Create LDIF files and use ldapadd to populate the directory.

4. Configuring the Firewall

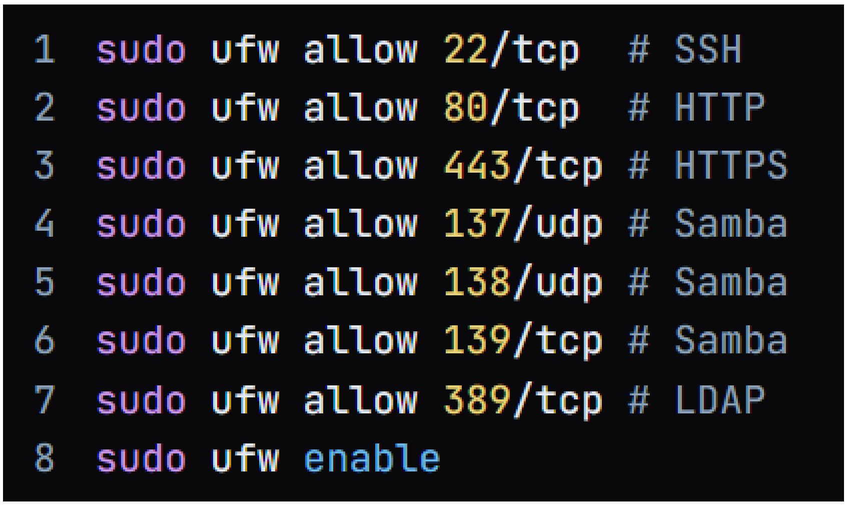
Installation and Configuration:

1. **Install UFW**



```
sudo apt install ufw
```

2. **Allow Necessary Ports**

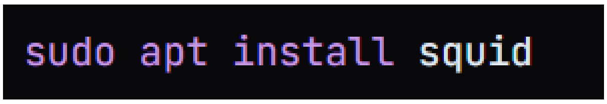


```
sudo ufw allow 22/tcp  # SSH
sudo ufw allow 80/tcp  # HTTP
sudo ufw allow 443/tcp # HTTPS
sudo ufw allow 137/udp # Samba
sudo ufw allow 138/udp # Samba
sudo ufw allow 139/tcp # Samba
sudo ufw allow 389/tcp # LDAP
sudo ufw enable
```

5. Setting Up a Proxy Server

Installation Steps (Squid Proxy):

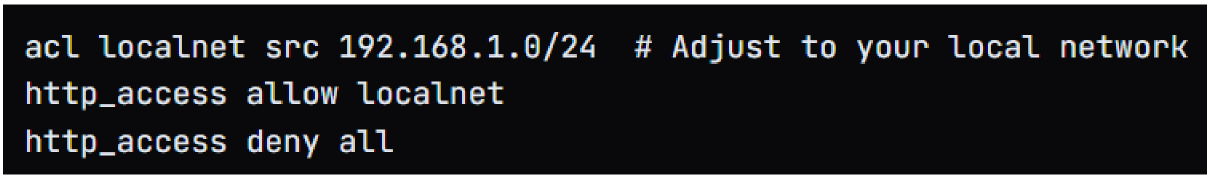
1. **Install Squid**



```
sudo apt install squid
```

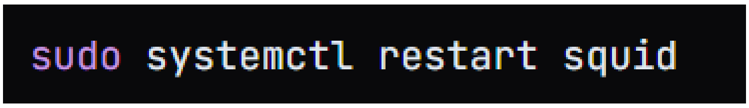
2. **Configure Squid:**

Edit /etc/squid/squid.conf to set up ACLs and control access



```
acl localnet src 192.168.1.0/24  # Adjust to your local network
http_access allow localnet
http_access deny all
```

3. **Restart Squid Service:**



```
sudo systemctl restart squid
```

Summary

This setup provides a secure, scalable infrastructure with Apache Tomcat for web applications, Samba for file sharing, DNS and LDAP for centralized resource and user management, and firewall/proxy configurations for network security. Following best practices in securing access and monitoring logs helps in maintaining a reliable and compliant environment.