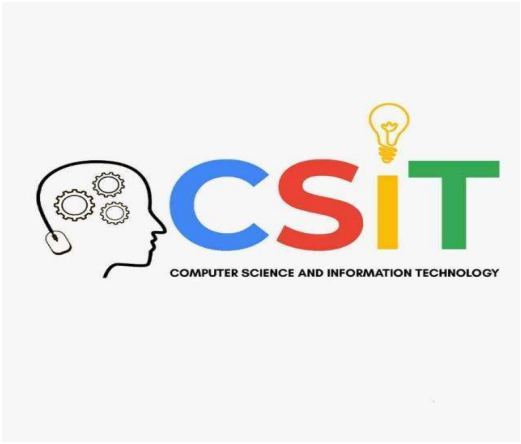
Acropolis Institute of Technology & Research, Indore



PRACTICAL FILE OF

LINUX LAB

Subject Code: CSIT - 505

B. Tech: III Year Vth Sem (CI-1)

Submitted To:

Prof.Garima Kumrawat

Submitted By:

Pooja Dalai
0827CI201133

Course Learning Objectives (CLO)

The Learning Objectives of **Linux** are such that the student will understand and learn the following things.

CLO1: To learn about the Basic Linux Concepts

CLO2: Students will become familiar with the Linux commands

CLO3: To learn Shell programming

CLO4: To learn about kernel

CLO5: Execute programs written in C under UNIX environment

Course Outcomes (CO)

At the end of the course, student would be able to

CO1: Understand the system calls

CO2: Compare between ANSI C AND C++ AND POSIX standards

CO3: To learn basic concepts of memory & scheduling

CO4: Mapping the relationship between UNIX Kernel support for files

CO5: Understand Kernel support for process creation and termination and memory allocation

Acropolis Institute of Technology and Research, Indore

Department of Computer Science and Information Technology

LIST OF PROGRAMS

S.NO	NAME OF EXPERIMENTS
1.	To Study basic & User status Unix/Linux Commands.
2.	Study & use of commands for performing arithmetic operations with Unix/Linux.
3.	Create a file called wlcc.txt with some lines and display how many lines, words and characters are present in that file.
4.	Append ten more simple lines to the wlcc.txt file created above and split the appended file into 3 parts. What will be the names of these split files? Display the contents of each of these files. How many lines will be there on the last file?
5.	Execute shell commands through vi editor.
6.	Write a Shell Script that accepts a filename, starting and ending line numbers as arguments and displays all lines between the given line numbers.
7.	Write a shell script that deletes all lines containing a specific word in one or more files supplied as arguments to it.
8.	Write a shell script that displays a list of files in the current directory to which the user has read, write and execute permissions.
9.	Write a shell script that accepts one or more file names as arguments and converts all of them to uppercase, provided they exist in the current directory.
10.	Write a shell script that computes the gross salary of an employee according to the following rules: i) If basic salary is <1500 then HRA=10% of the basic and DA=90% of the basic ii) If the basic salary is >=1500 then HRA=500/- and DA=98% of the basic
11.	Write an interactive file –handling shell programs. Let it offer the user the choice of copying, removing, renaming, or linking files. Once the

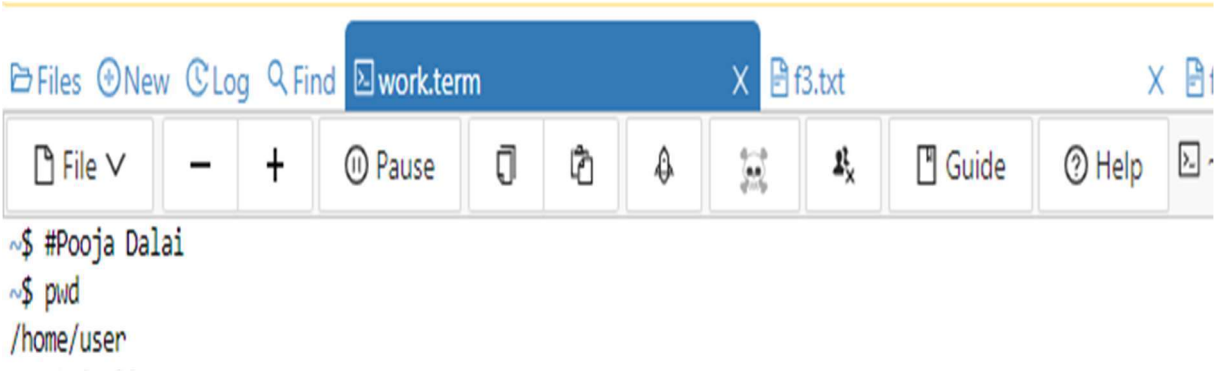
	user has made a choice, have the same program ask the user for the necessary information, such as the file name ,new name and so on.
12.	Write a shell script that accepts any number of arguments and prints them in the reverse order.
13.	Write a Shell script to count the number of lines in a file that do not contain vowels.
14.	Write a shell script which receives two file names as arguments. It should check whether the two file contents are the same or not. If they are the same then the second file should be deleted.
15.	Develop an interactive script that asks for a word and a file name and then tells how many times that word occurred in the file.
16.	Write a shell script to perform the following string operations: i. To extract a substring from a given string. ii. To find the length of a given string.
17.	Write a shell script that accepts two integers as its arguments and computes the value of the first number raised to the power of the second number.
18.	Write a shell script that takes a command –line argument and reports on whether it is directory, a file, or something else.
19.	Develop an interactive grep script that asks for a word and a file name and then tells how many lines contain that word.
20.	Write a shell script to list all of the directory files in a directory.
21.	Write and execute all process control commands.
22.	Study & Installation of SAMBA, APACHE, TOMCAT.
23.	Study & installation of Firewall & Proxy server.

EXPERIMENT N0.-01

AIM :- Study basic & User status Unix/Linux Commands

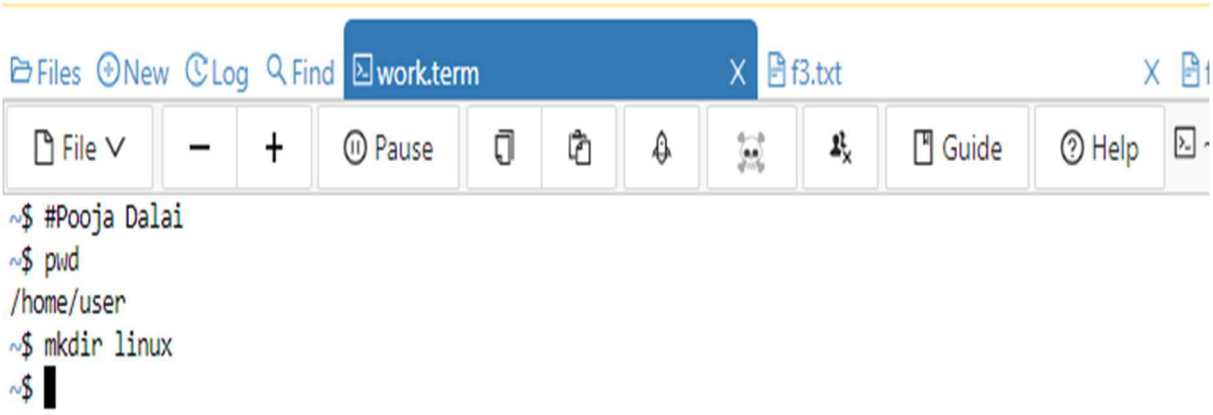
1. pwd Command

pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root.
pwd is shell built-in command(pwd) or an actual binary(/bin/pwd).
$PWD is an environment variable which stores the path of the current directory.



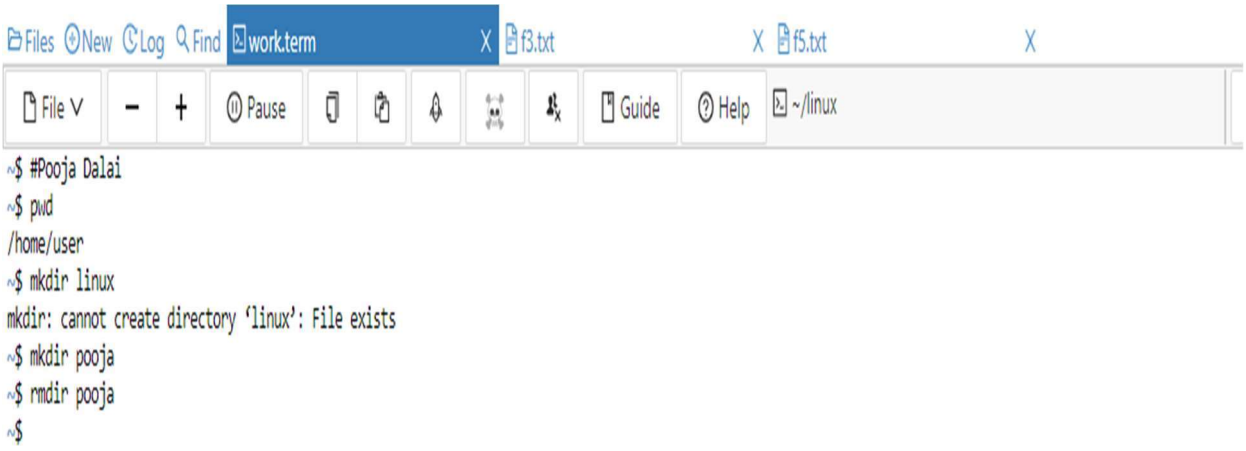
2. mkdir Command

mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems). This command can create multiple directories at once as well as set the permissions for the directories.



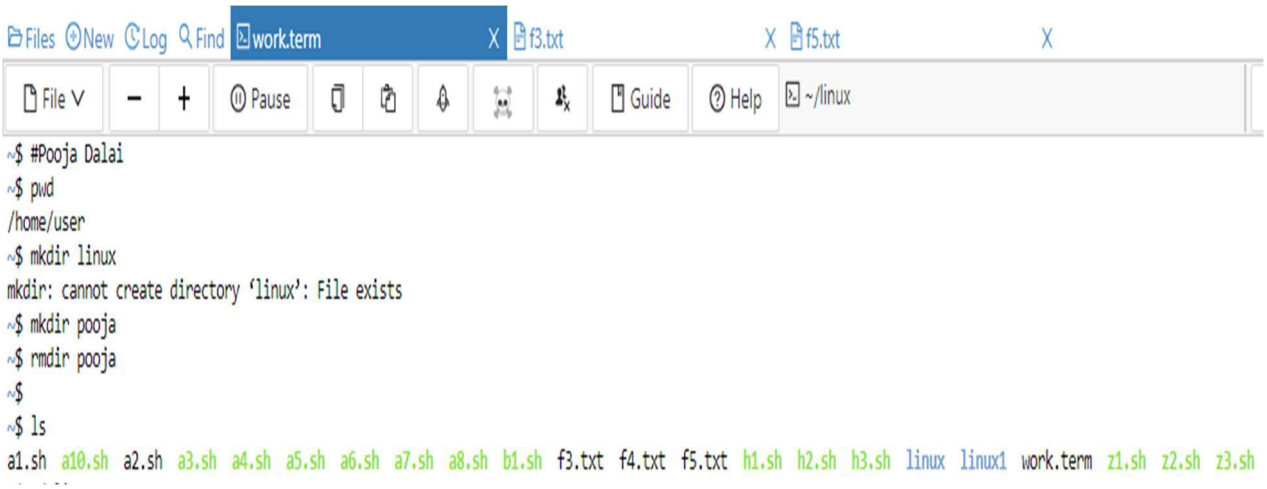
3. rmdir Command

rmdir command is used remove empty directories from the filesystem in Linux. The rmdir command removes each and every directory specified in the command line only if these directories are empty. So if the specified directory has some directories or files in it then this cannot be removed by rmdir command.



4. ls Command

The ls command is used to list files or directories in Linux and other Unix-based operating systems.



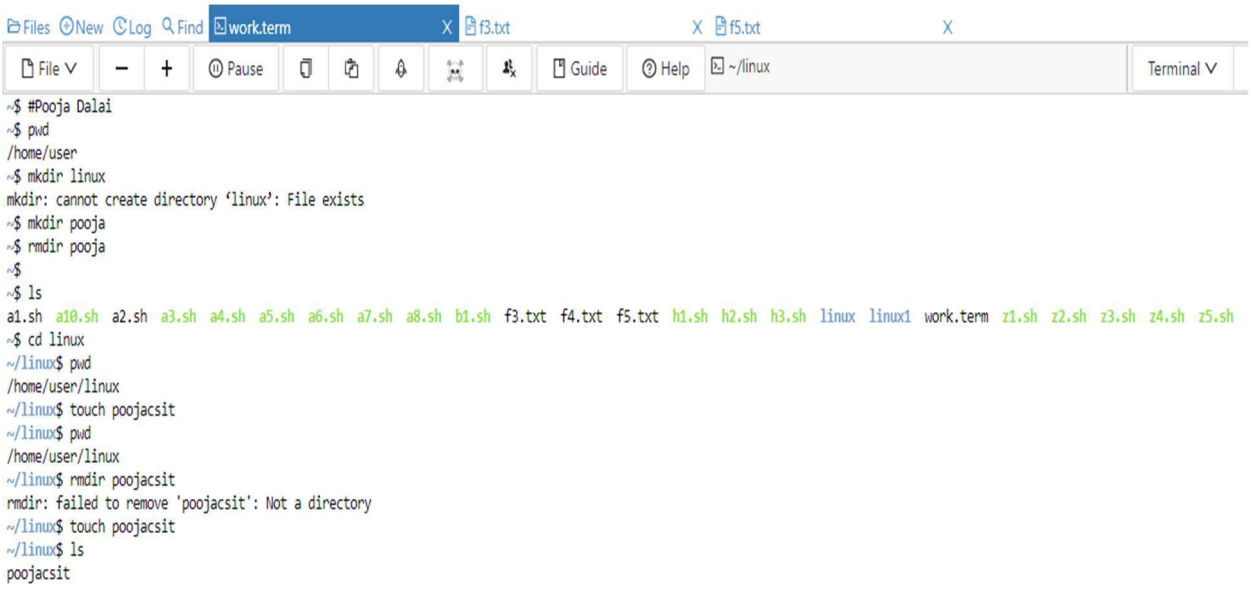
5. cd Command

The cd command is used to change the current directory.

· Change from current directory to new directory
· Change to the home directory
· Change to the previous directory
· Change to the parent directory
· Change to the root directory
· Change directory using absolute path
· Change directory using relative path
· Change to another user's home directory
· Change to directory having spaces
· Change to multiple sub directories

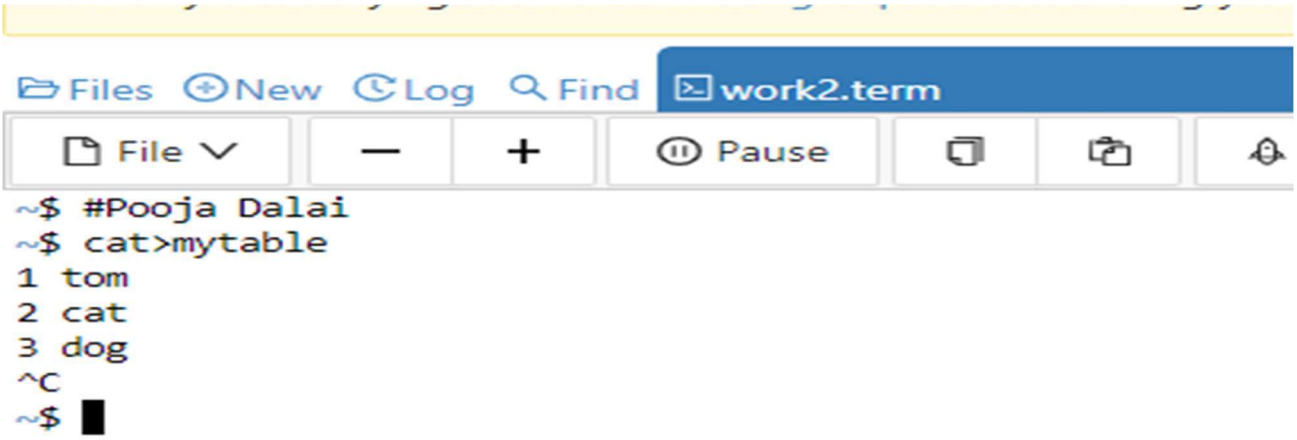
6. touch Command

The touch command is a standard command used in UNIX/Linux operating system which is used to create, change and modify timestamps of a file.



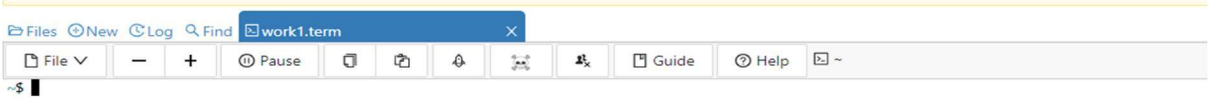
7. cat Command

cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files. So let us see some frequently used cat commands.



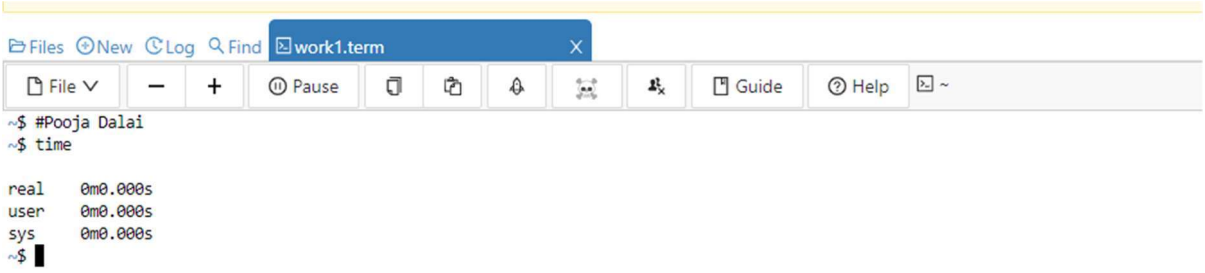
8. clear Command

Clear is a standard Unix computer operating system command that is used to clear the terminal screen. This command first looks for a terminal type in the environment and after that, it figures out the **t**erminfo database for how to clear the screen. And this command will ignore any command-line parameters that may be present. Also, the **clear** command doesn't take any argument and it is almost similar to **cls** command on a number of other Operating Systems.



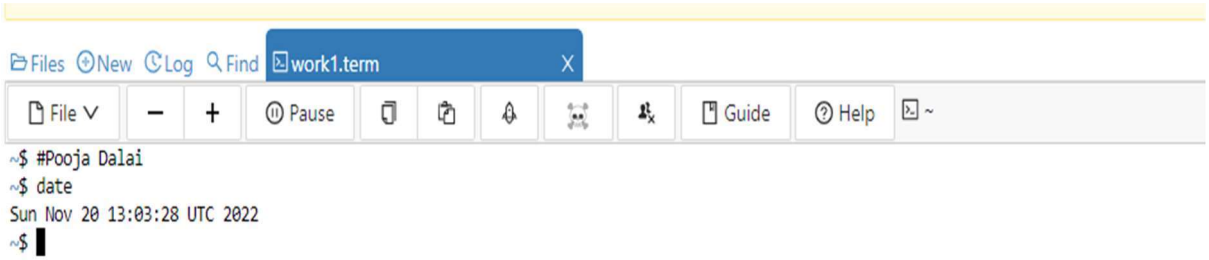
9. time Command

Time command in Linux is used to execute a command and prints a summary of real-time, user CPU time and system CPU time spent by executing a command when it terminates.



10.date Command

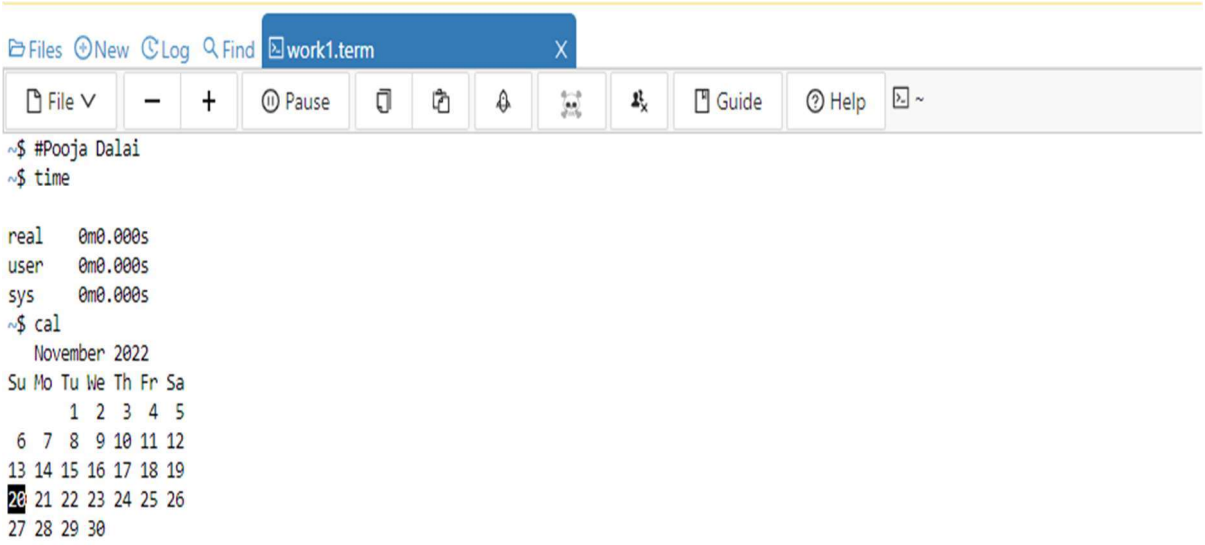
date command is used to display the system date and time. date command is also used to set date and time of the system. By default the date command displays the date in the time zone on which unix/linux operating system is configured.You must be the super-user (root) to change the date and time.



11.cal Command

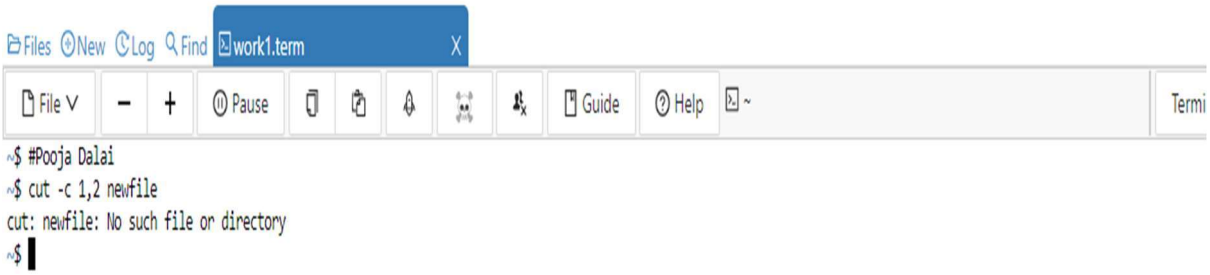
f a user wants a quick view of the calendar in the Linux terminal, cal is the command for you. By default, the cal command shows the current month calendar as output.

cal command is a calendar command in Linux which is used to see the calendar of a specific month or a whole year.



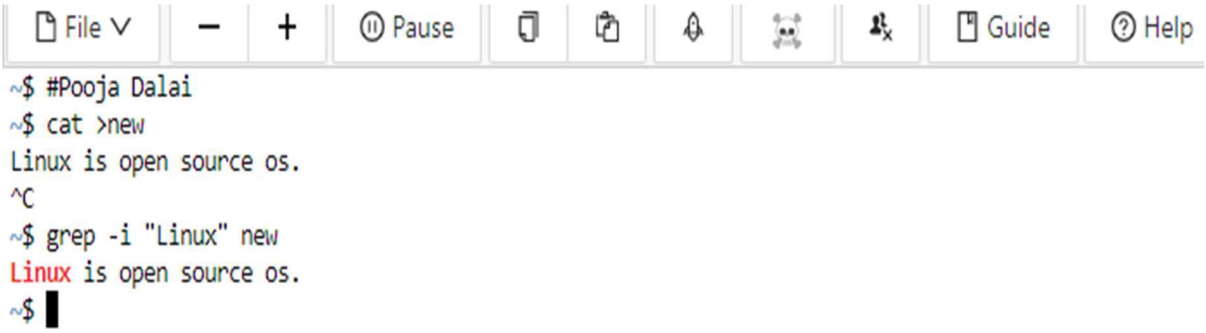
12. cut Command

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte** position, character and field. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is not precedes by its file name.



13. grep Command

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


File
Pause
Guide
Help
~$ #Pooja Dalai
~$ cat >new
Linux is open source os.
^C
~$ grep -i "Linux" new
Linux is open source os.
~$

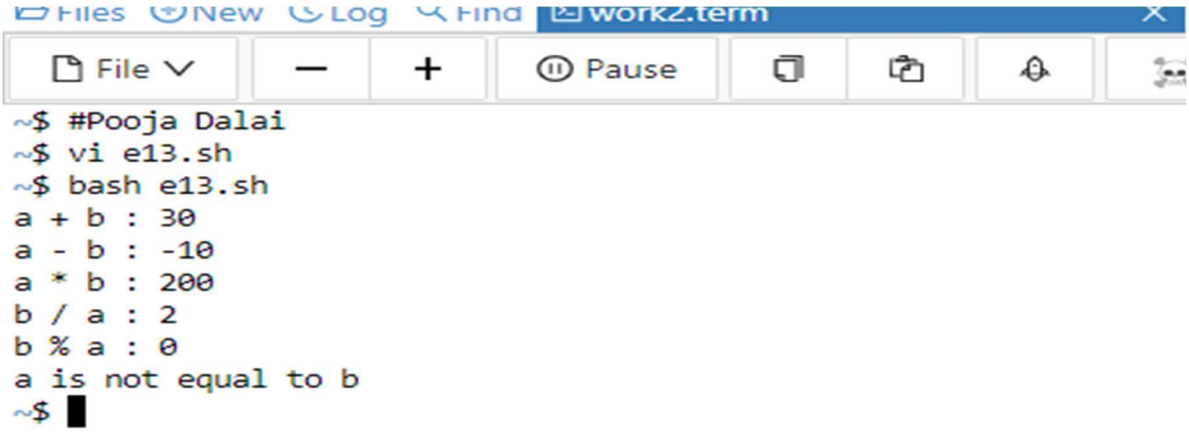
EXPERIMENT N0.-02

AIM :- Study & use of commands for performing arithmetic operations with Unix/Linux.

Algorithm/Process/Source Code :-

```
#!/bin/sh
a=10
b=20
val=`expr $a + $b`
echo "a + b : $val"
val=`expr $a - $b`
echo "a - b : $val"
val=`expr $a \* $b`
echo "a * b : $val"
val=`expr $b / $a`
echo "b / a : $val"
val=`expr $b % $a`
echo "b % a : $val"
if [ $a == $b ]
then
   echo "a is equal to b"
fi
if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

Expected Output/Screenshot:-



EXPERIMENT - 6

AIM :- Write a Shell Script that accepts a file name, starting and ending line numbers as arguments and displays all lines between the given line numbers.

Algorithm/Process/Source Code :-

- First, create a file main.sh using cat or any file creation command.
- Take user input with the read command
- Now we will use a tool sed

We will use the tool sed which is used to output the contents to the console. Sed is passed a -n flag to choose which lines are output to the console.

$ vi file.sh

```
echo "file name"

read f

echo "starting line"

read s

echo "ending line"

read e

sed -n $s,$e\p $f
```

Syntax :-

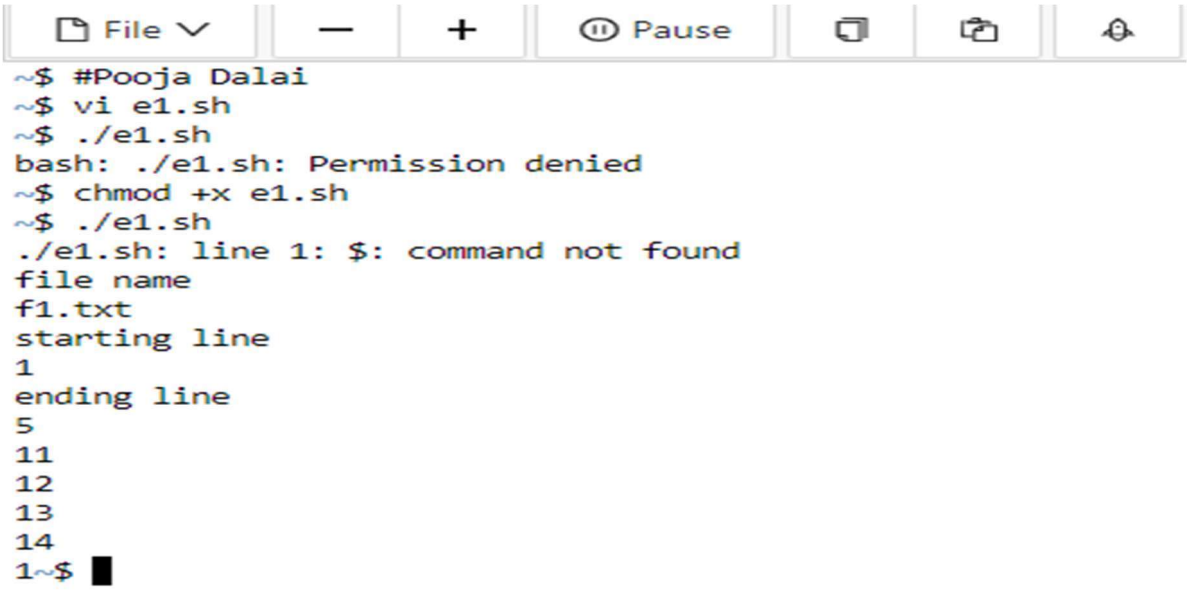
```
sed -n <start>,<end>\p <filename>
```

Execution :-

$ ./file.sh or bash file.sh

(for permission use chmod +x file.sh co

Expected Output/Screenshot :-



EXPERIMENT - 7

AIM :- Write a shell script that deletes all lines containing a specific word in one or more files supplied as arguments to it.

Algorithm/Process/Source Code :-

Create two files with some lines – f1.txt and f2.txt

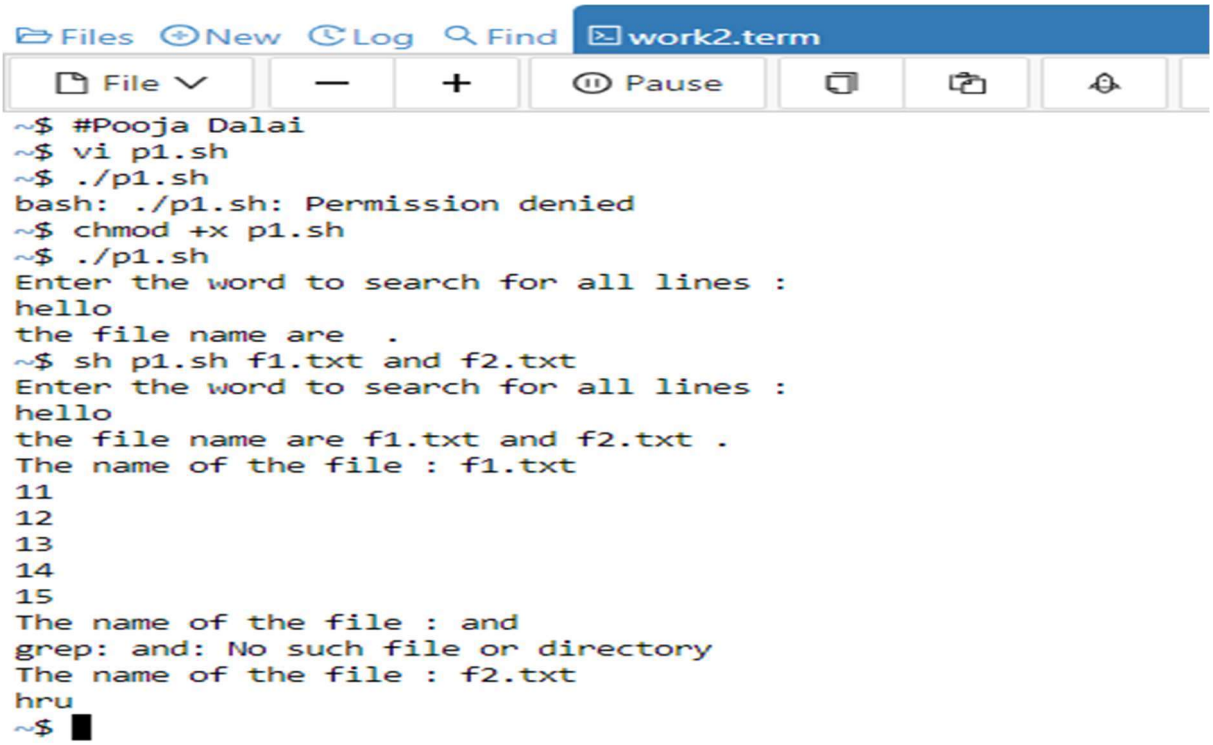
Create .sh file - delete.sh

```
echo "Enter the word to search for all lines :"
read word
echo "the file name are $* ."
for i in $*
do
echo "The name of the file :" $i
grep -v $word $i
done
```

Execution :-

$ sh delete.sh f1.txt and f2.txt

Expected Output/Screenshot :-



EXPERIMENT - 8

AIM :- Write a shell script that displays a list of files in current directory to which the user has read, write and execute permissions.

Algorithm/Process/Source Code :-

Create a .sh file

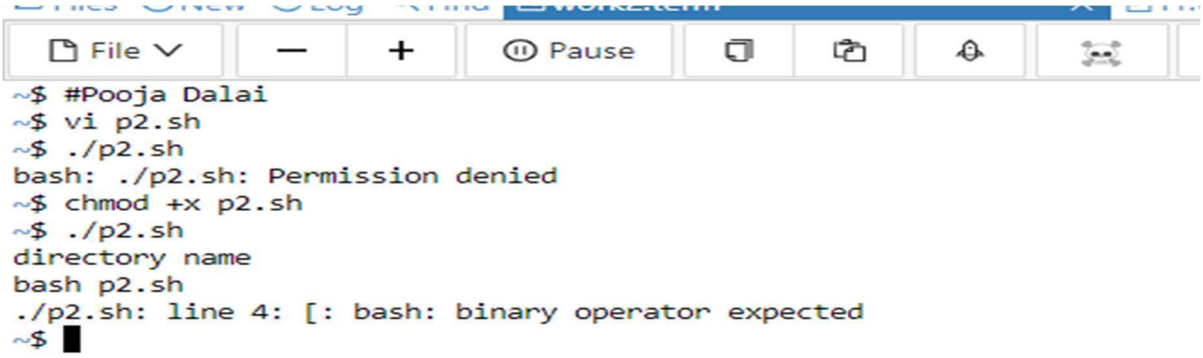
```
echo "directory name"
read D
if [ -d $D ]
then
cd $D
ls > f
exec < f
while read line
do
```

```
if [ -f $line ]
then
if [ -r $line -a -w $line -a -x $line ]
then echo "$line has all permission"
fi
fi
done
```

Execution :-

$ ./filename.sh or bash filename.sh

Expected Output/Screenshot :-



EXPERIMENT - 9

AIM :- Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.

Algorithm/Process/Source Code :-

Create a .sh file

```
for file in *

do

if [ -f $file ]

then

echo $file | tr '[a-z]' '[A-Z]'

fi

done
```

Execution :-

$ls (to check the all files/directories)

$ ./filename.sh

Expected Output/Screenshot :-


```
~$ #Pooja Dalai
~$ vi p3.sh
~$ ./p3.sh
bash: ./p3.sh: Permission denied
~$ chmod +x p3.sh
~$ ./p3.sh
A.SH
E1.SH
E12.SH
E13.SH
F1.TXT
F2.TXT
F3.SH
M1.SH
M2.SH
M3.SH
M5.SH
M6.SH
MYTABLE
NEW
P1.SH
P2.SH
P3.SH
WORK2.TERM
~$
```

EXPERIMENT - 10

AIM :- write a shell script that computes the gross salary of an employee according to the following rules:

i. If basic salary is <1500 then HRA=10% of the basic and DA=90% of the basic

ii. If the basic salary is>=1500 then HRA=500/- and DA=98% of the basic

Algorithm/Process/Source Code :-

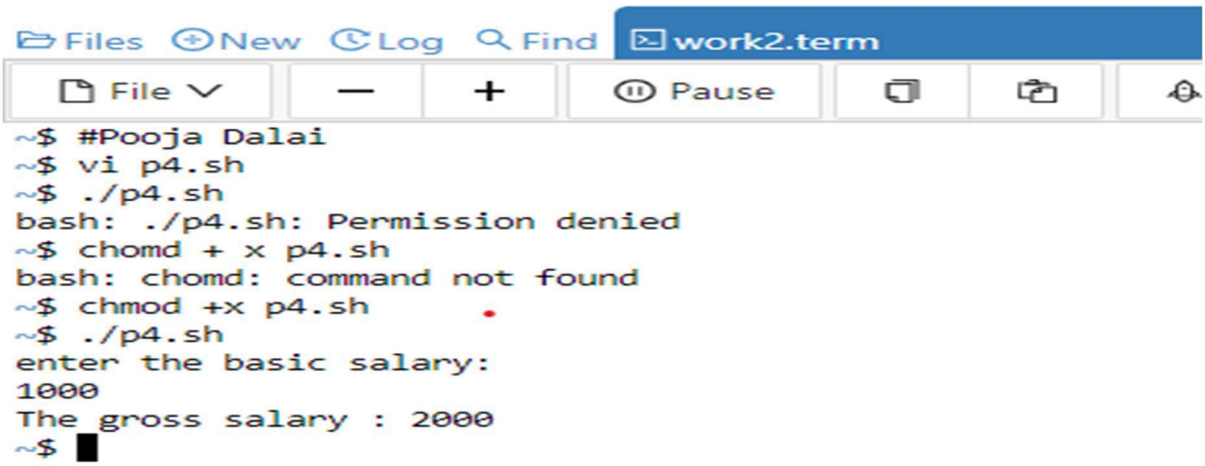
Create a file gsalary.sh

```
echo "enter the basic salary:"
read bsal
if [ $bsal -lt 1500 ]
then
gsal=$((bsal+((bsal/100)*10)+(bsal/100)*90))
echo "The gross salary : $gsal"
fi
if [ $bsal -ge 1500 ]
then
gsal=$(((bsal+500)+(bsal/100)*98))
echo "the gross salary : $gsal"
fi
```

Execution :-

```
$ ./gsalary.sh
```

Expected Output/Screenshot :-



EXPERIMENT - 11

AIM :- Write an interactive file –handling shell programs. Let it offer the user the choice of copying, removing, renaming, or linking files. Once the user has made a choice, have the same program ask the user for the necessary information, such as the file name ,new name and so on.

Algorithm/Process/Source Code :-

```
while true
do
echo "*******MENU*********"
echo "
1. List of files.
2. Copying files.
3. Removing files.
4. Renaming files.
5. Linking files.
press [CTRL+C] TO EXIT"
echo "enter your choice "
```

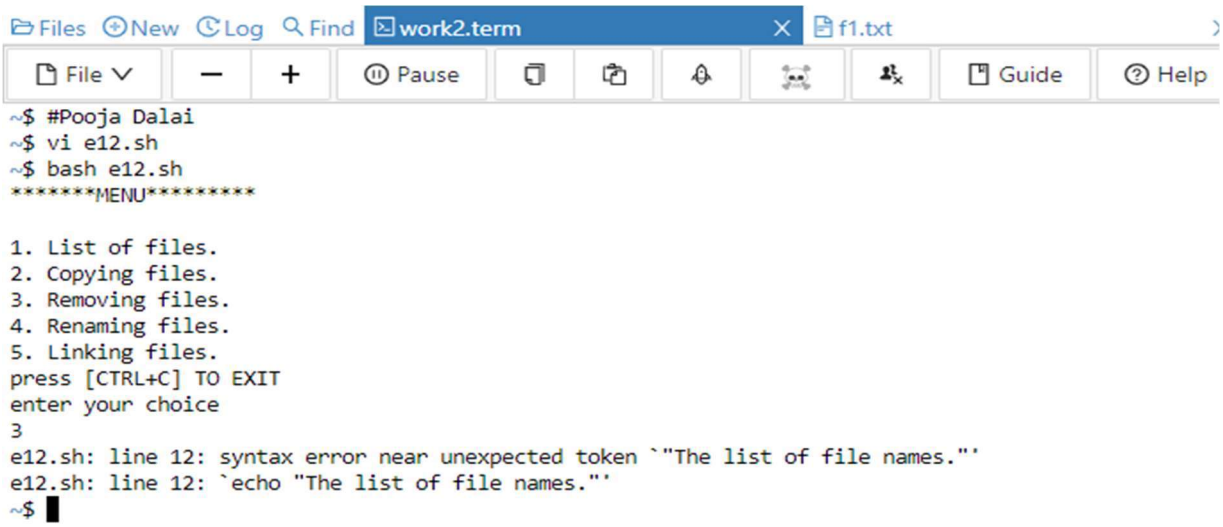
```
read ch
case "$ch" in
echo "The list of file names."
ls -l || echo "These are file";;
echo "Enter the old filename."
read ofile
echo "Enter the new file name."
read nfile
cp $ofile $nfile && echo "Copied sucessfully." || echo "Copied is not possible." ;;
 echo "Enter the file name to remove."
read rfile
rm -f $rfile && echo "Successfully removed." ;;
echo "Enter the old file name."
read ofile
echo "Enter the new file name."
read nfile
mv $ofile $nfile && echo "The file $ofile name renamed to $nfile." || echo "You
cann't Rename the file.".;;
echo "Enter the original filename."
read ofile
echo "Enter the new filename to link a file."
read lfile
ln $ofile $lfile && echo "Creat the linking file Sccessfully." || echo "You cann't
Linking the file.";; * )
echo "Invalid option."
Echo " Enter correct choice."
esac
done
```

Execution: bash script_filename.sh

Execution :-

Bash file_name.sh

Expected Output/Screenshot :-



EXPERIMENT - 12

AIM :- Write a shell script that accepts any number of arguments and prints them in the reverse order.

Algorithm/Process/Source Code :-

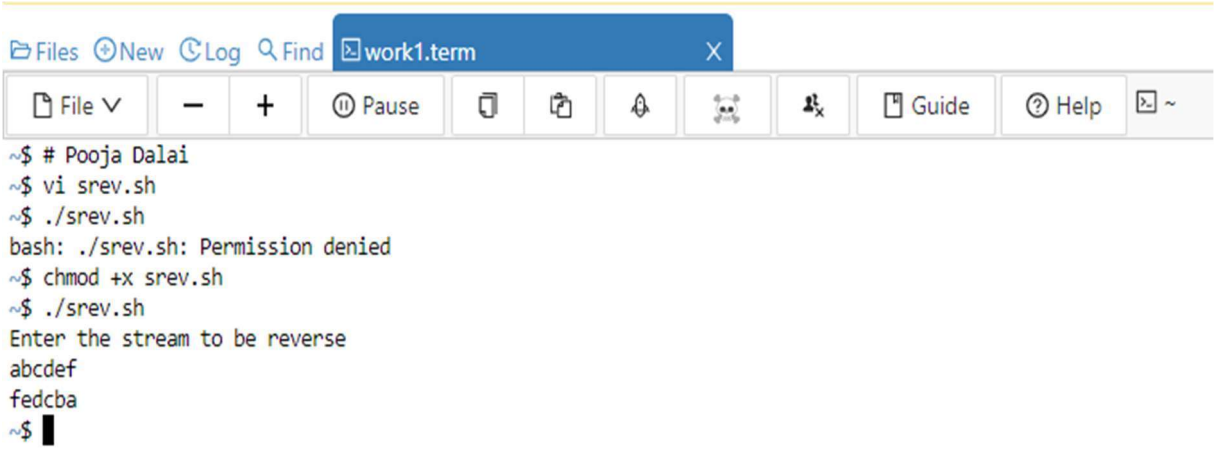
Create a file – srev.sh

```
#!/bin/bash
echo "Enter the stream to be reverse"
read str
echo $str | rev
```

Execution :-

$ ./srev.sh

Expected Output/Screenshot :-



EXPERIMENT - 13

AIM :- Write a Shell script to count the number of lines in a file that do not contain vowels.

Algorithm/Process/Source Code :-

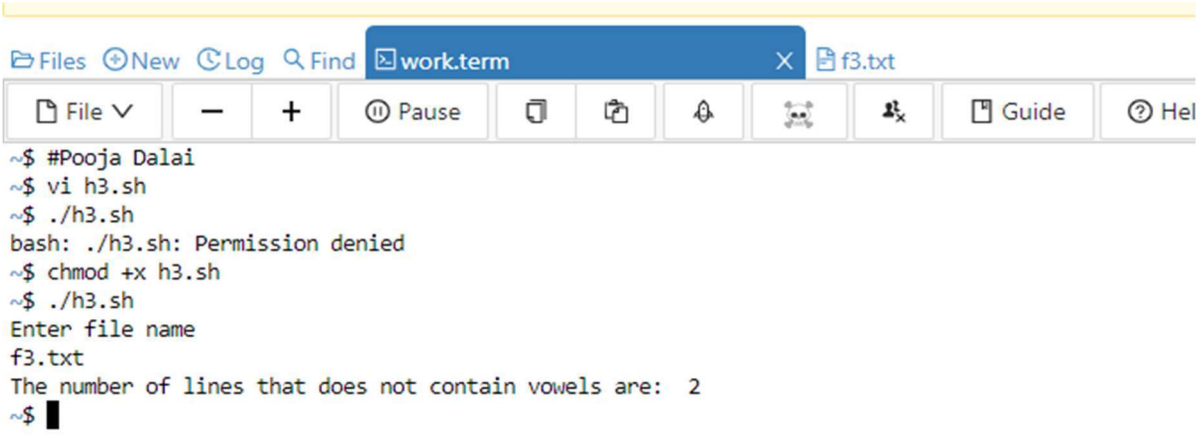
Create two files (.sh and txt)

```
#!/bin/bash

echo "Enter file name"

read fileawk '$0!~/[aeiou]/{ count++ }

END{print "The number of lines that does not contain vowels are: ",count}' $file
```

Execution :-

$ ./filename.sh (then give the txt file name as input)

Expected Output/Screenshot :-



EXPERIMENT - 14

AIM :- Write a shell script which receives two file names as arguments. It should check whether the two file contents are same or not. If they are same then second file should be deleted.

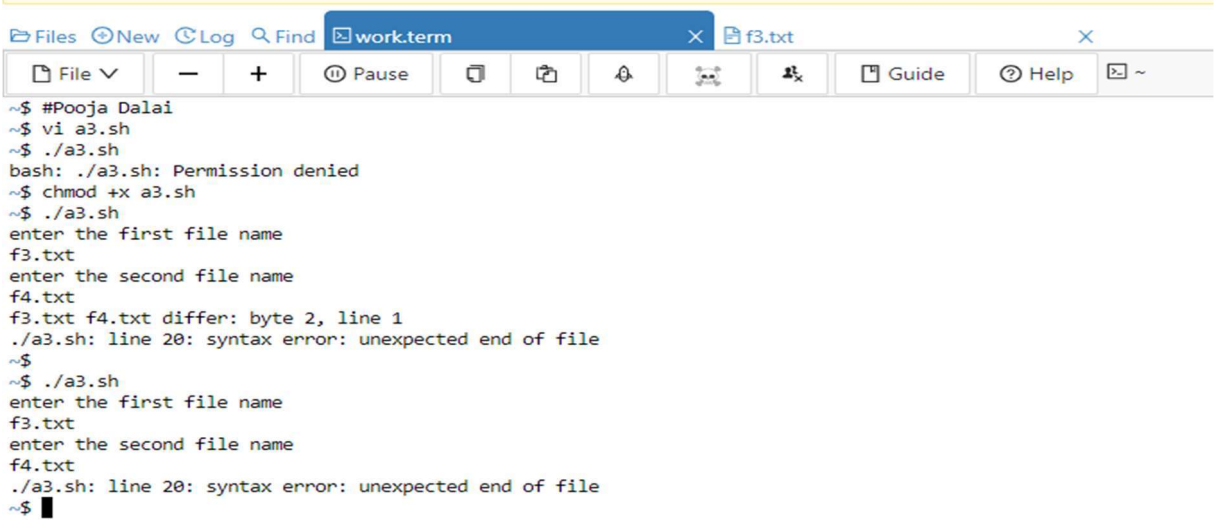
Algorithm/Process/Source Code :-

```
#!/bin/bash
echo "enter the first file name"
read file1
echo "enter the second file name"
read file2
cmp $file1 $file2 && rm $file2

if [ -e $file1 ]
then
if [ ! –e $file2 ]
then
echo "The two files contents are same."
echo "The second file is deleted successfully."
else
echo "The two files contents are not same."
echo "You cann't remove the file '$file2' ."
fi
else
echo "You should enter the existing file names."
Fi
```

Execution :-

```
$ ./filename.sh
```

Expected Output/Screenshot :-



EXPERIMENT - 15

AIM :- Develop an interactive script that ask for a word and a file name and then tells how many times that word occurred in the file.

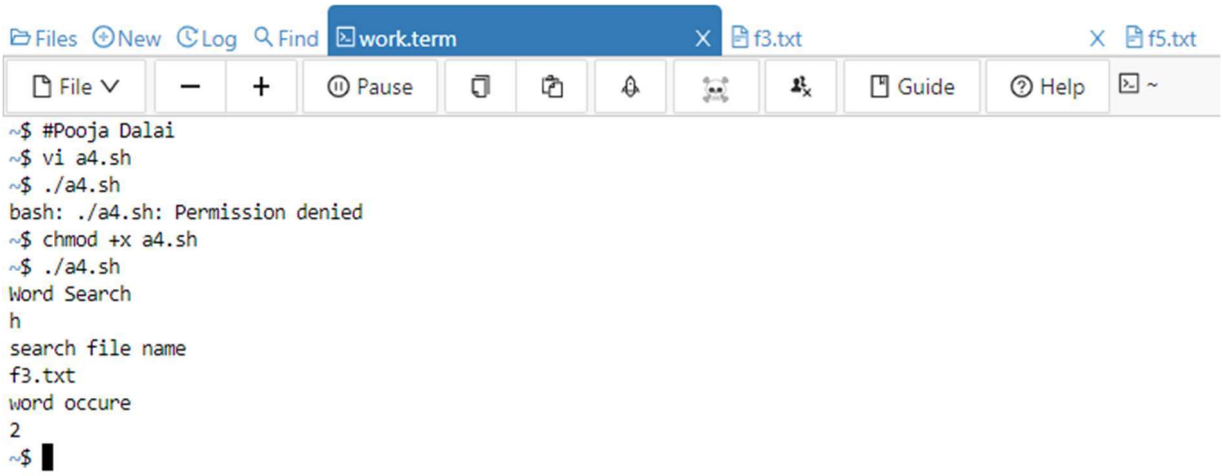
Algorithm/Process/Source Code :-

```
#!/bin/bash
echo "Word Search"
read w
echo "search file name"
read f
echo "word occure"
grep -c $w $f
```

Execution :-

$ ./filename.sh

Expected Output/Screenshot :-



EXPERIMENT - 16

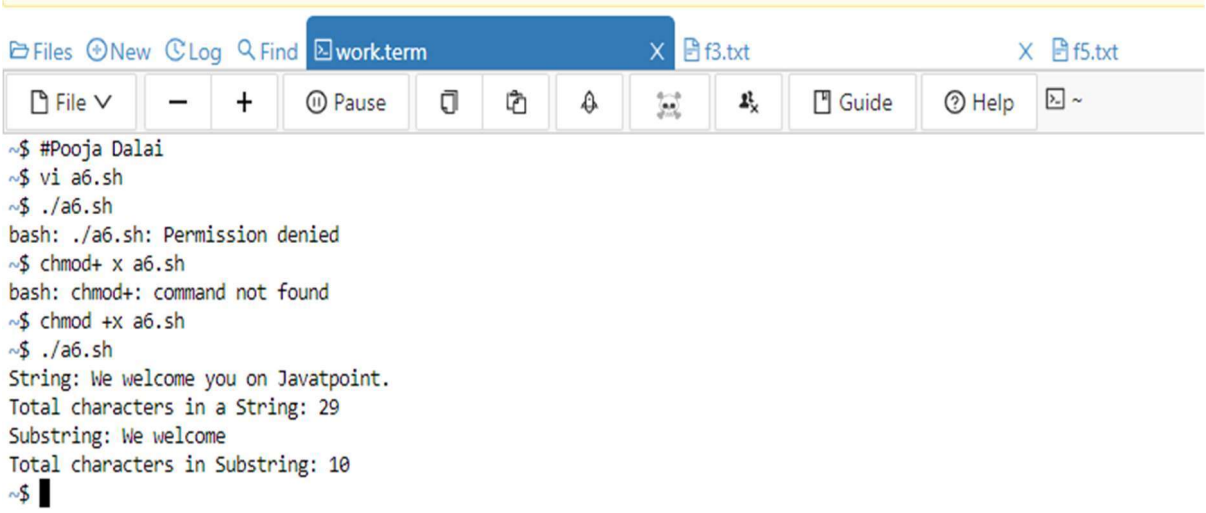
AIM :- Write a shell script to perform the following string operations:

i. To extract a sub-string from a given string.

Algorithm/Process/Source Code :-

```
#!/bin/bash
echo "String: We welcome you on Javatpoint."
str="We welcome you on Javatpoint."
echo "Total characters in a String: ${#str} "
substr="${str:0:10}"
echo "Substring: $substr"
echo "Total characters in Substring: ${#substr} "
```

Expected Output/Screenshot :-

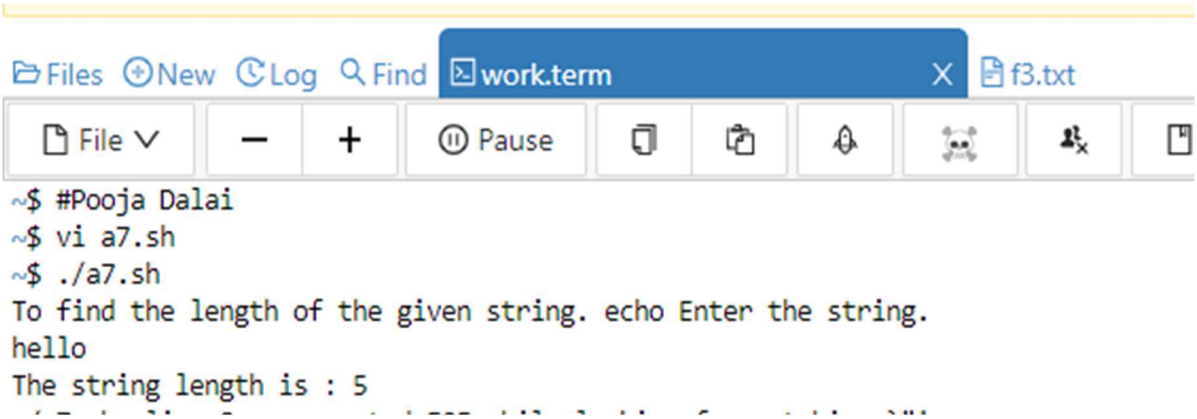


ii. To find the length of a given string.

Algorithm/Process/Source Code :-

```
#!/bin/bash
echo "To find the length of the given string."
echo "Enter the string."
read string
strlen=${#string}
echo "The string length is : $strlen"
echo "substring”
read substr
substr=${#string:5:10}
echo "Total characters in substring: $substr "
```

Expected Output/Screenshot :-



EXPERIMENT – 17

AIM :- Write a shell script that accepts two integers as its arguments and computes the value of first number raised to the power of the second number.

Algorithm/Process/Source Code :-

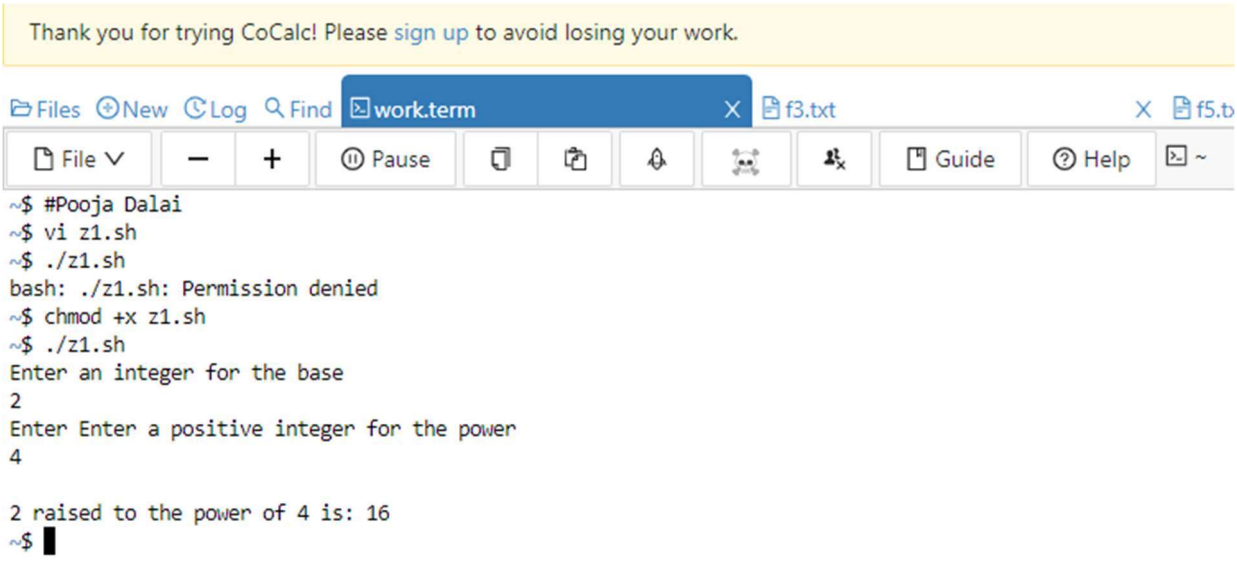
```
# Bash Program to find
# A to the power B

# Subroutine to find A
# to the power B
pow()
{
  # value of A
  a=$1

  # value of B
  b=$2

  # c to count counter
  c=1

  # res to store the result
  res=1

  #
  if((b==0));
  then
    res=1
  fi

  if((a==0));
  then
    res=0
  fi

  if((a >= 1 && b >= 1));
  then
    while((c <= b))
    do
      res=$((res * a))
      c=$((c + 1))
    done
```

```
  fi

  # Display the result
  echo "$1 to the power $2 is $res"
}

# Driver Code

# input
A=2
B=4

# calling the pow function
pow $A $B
```

Execution :-

bash file_name.sh

Expected Output/Screenshot :-



EXPERIMENT – 18

AIM :- Write a shell script that takes a command –line argument and reports on whether it is directory, a file, or something else.

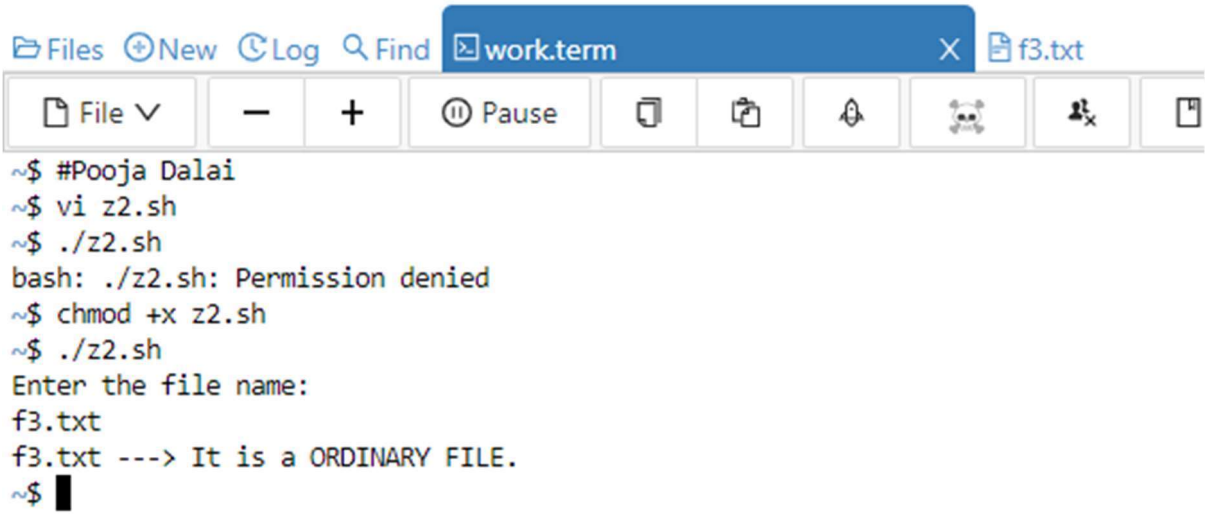
Algorithm/Process/Source Code :-

```
echo”enter file”
read str
if test -f$str
then echo “file exists n it is an ordinary file"
elif test -d $str
then echo “directory file”
else
echo”not exists”
fi
if test -c $str
```

Execution :-

```
bash file_name.sh
```

Expected Output/Screenshot :-



EXPERIMENT – 19

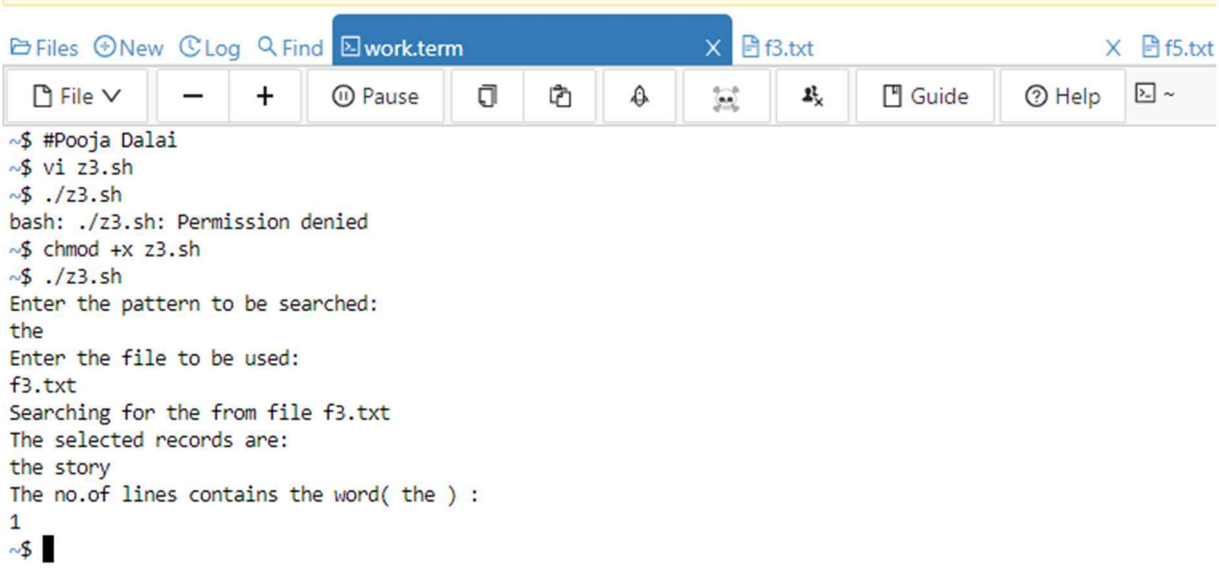
AIM :- Develop an interactive grep script that asks for a word and a file name and then tells how many lines contain that word.

Algorithm/Process/Source Code :-

```
$ vi grep.sh echo "Enter the pattern to be searched: "
read pattern echo "Enter the file to be used: "
 read filename
echo "Searching for $pattern from file $filename"
 echo "The selected records are: "
 grep "$pattern" $filename
echo "The no.of lines contains the word( $pattern ) :"
 grep -c "$pattern" $filename
```

Execution :- bash file_name.sh

Expected Output/Screenshot :-



EXPERIMENT – 20

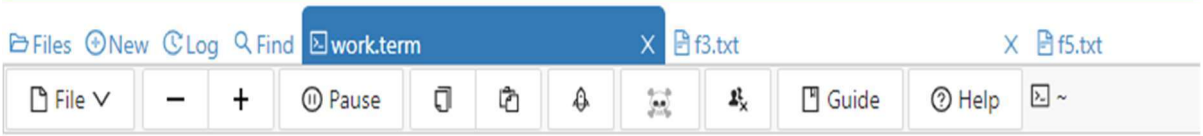
AIM :- Write a shell script to list all of the directory files in a directory.

Algorithm/Process/Source Code :-

```
# Shell Script to list all sub-directories
# present in a current folder

# echo prints a message for user on screen
echo "List of sub-directories present in this Folder - "

# Following command lists all sub directories
# '*/' will only match directories under the current directory
ls -d */
```

Execution :- bash file_name.sh

Expected Output/Screenshot :-




```
~$ #Pooja Dalai
~$ vi z5.sh
~$ ./z5.sh
bash: ./z5.sh: Permission denied
~$ chmod +x z5.sh
~$ ./z5.sh
enter directory name
Labwork3
./z5.sh: line 3: if[ -d Labwork3]: command not found
./z5.sh: line 4: syntax error near unexpected token `then'
./z5.sh: line 4: `then'
~$
```

EXPERIMENT – 22

AIM :- Study & Installation of SAMBA, APACHE, TOMCAT.

I. Study & Installation of SAMBA

A Samba file server enables file sharing across different operating systems over a network. It lets you access your desktop files from a laptop and share files with Windows and macOS users.

- To install Samba, we run:

sudo apt update
sudo apt install samba
We can check if the installation was successful by running:

whereis samba

The following should be its output:

samba: /usr/sbin/samba /usr/lib/samba /etc/samba /usr/share/samba
/usr/share/man/man7/samba.7.gz /usr/share/man/man8/samba.8.gz

- Setting up Samba

Now that Samba is installed, we need to create a directory for it to share:

mkdir /home/<username>/sambashare/
The command above creates a new folder sambashare in our home directory which we will share later.

The configuration file for Samba is located at /etc/samba/smb.conf. To add the new directory as a share, we edit the file by running:

sudo nano /etc/samba/smb.conf
At the bottom of the file, add the following lines:

[sambashare]
comment = Samba on Ubuntu
path = /home/username/sambashare
read only = no
browsable = yes
Then press Ctrl-O to save and Ctrl-X to exit from the *nano* text editor.

What we've just added

comment: A brief description of the share.
path: The directory of our share.

read only: Permission to modify the contents of the share folder is only granted when the value of this directive is no.

browsable: When set to yes, file managers such as Ubuntu's default file manager will list this share under "Network" (it could also appear as browseable).

Now that we have our new share configured, save it and restart Samba for it to take effect:

sudo service smbd restart
Update the firewall rules to allow Samba traffic:

sudo ufw allow samba

- Study & Installation of APACHE

  Overview
  Apache is an open source web server that's available for Linux servers free of charge.

  Installing Apache
  To install Apache, install the latest meta-package apache2 by running:

```
sudo apt update
sudo apt install apache2
```

After letting the command run, all required packages are installed and we can test it out by typing in our IP address for the web server.

- Study & Installation of SAMBA, APACHE, TOMCAT.

Step 1 - Create A Tomcat-Specific User and User Group

It's a bad idea to run Tomcat as the root user, especially if you're going to be starting Tomcat automatically. It's much more secure to create a new group and user specifically to run Tomcat. You can do so with the following commands (in this example, we have created a user group named tomcat, and a user named tomcat with the password tomcat; you can certainly be more creative if you wish):

```
$ groupadd tomcat

$ useradd -s /sbin/nologin -g tomcat -d /path/to/tomcat tomcat

$ passwd tomcat
```

Step 2 - Adjust Ownership For New Users And Groups

Now that you have created a user to run Tomcat, you'll need to give them access to the correct directories. Use the following commands, substituting your own usernames and groups as necessary:

```
# chown -R tomcat.tomcat /path/to/tomcat
# chmod 775 /path/to/tomcat/webapps
```

The first gives ownership of the Tomcat directories to the Tomcat user, and the second gives the user write access for the web apps directory.

Step 3 - Relay Traffic For Non-Root Tomcat User

When running Tomcat as a user other than the root user, you will not be able to bind to port 80, which is where Tomcat listens for HTTP requests. To get around this, you can use Netfilter, which is packaged with all major Linux distributions:

```
# iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080
# iptables -t nat -I OUTPUT -p tcp --dport 80 -j REDIRECT --to-ports 8080
```

To preserve these rules through re-boot, save them with the "ip-tables-save" command, and then follow the procedure appropriate for your Linux distribution (for most distributions, this means editing the iptables init script; Debian users should load the configuration via a script called by if-up.d or pre-up.d).

Step 3 - Create A Custom init Script

To start Tomcat at Linux boot time, we'll need to create an init script that calls the startup.sh and shutdown.sh scripts included with Tomcat.

The actual creation of this script is outside the scope of this article, but there are many useful resources available online. All you need to know in order to use the basic init script format to call Tomcat is how the startup.sh and shutdown.sh scripts work.

EXPERIMENT-23

AIM :- Study & installation of Firewall & Proxy server.

Step 1 : Beef-up basic Linux security:

While this blog is titled to address firewall configuration, the first step is to ensure that the firewall has all the support it needs with a 100% secure Linux machine. To do this, ensure you have all the latest security updates installed for your version of Linux.

Step 2: Decide how you want to protect your server:

While Iptables is generally where the linux community looks to configure a firewall, there are easier options available that are also free for use. Here are some that we would recommend:

1.ClearOS:

-ClearOS is extremely easy-to-use. It is suitable those who prefer an easy-to-follow UI AND also for geeks who would like to talk to it through the command-line interface

Post a 10-minute installation time, you are asked to reboot and are given all the information and support required to manage your firewall as easily as possible

2.OPNsense:

-OPNsense offers several advanced features not usually found in free firewalls like ‘forward caching proxy’ and ‘intrusion detection’.

-It supports the use of OpenVPN. To know how useful OpenVPN is, read more here

-It uses an Inline Intrusion Prevention System which is a powerful form of Deep Packet Inspection. Here, instead of just blocking an IP address or port, the firewall inspects individual data packets or connections and stops them before they reach the sender if found malicious

3.ConfigServer Firewall (CSF):

-CSF is an advanced firewall suite for Linux systems has the Login Failure Daemon (LFD) process that regularly scans for failed login attempts (or “Brute-force attacks”) on your Server and takes action against the offending IP Addresses very quickly

-CSF can be managed through the Command Line Interface and its front-end is accessible by the root account through cPanel, DirectAdmin and Webmin which makes configuring and managing the firewall very simple

IPTABLES:

Understand Iptables and how it works:

The Linux kernel has the capacity to filter incoming and outgoing packages with a filtering tool known as 'Iptables'. The Iptables tool is in charge of deciding which packages can come in and go out based on the rules it is configured to follow.

First, how to configure the firewall manually:

Working with iptables manually can be complicated. We have a quick fix at the bottom of our section you can try. Read on for more.

Step 1: Retrieve the Iptables firewall:

Iptables is pre-installed on almost every Linux distribution. You can use this command to retrieve the package:

sudo apt-get install iptables

Step 2: Discover what Iptables is already configured to do by default:

Run the iptable L command

Step 3: You can decide to modify the existing rules or instead start afresh:

To start afresh, run this command

iptables-F

Step 4: Decide which firewall ports to close:

First block all lines of attack by running the following commands:

Block XMAS Packets: iptables -A INPUT -p tcp –tcp-flags ALL ALL -j DROP

Block null packets: iptables -A INPUT -p tcp –tcp-flags ALL NONE -j DROP

Block syn-flood packets: iptables -A INPUT -p tcp ! –syn -m state –state NEW -j DROP

Step 5: Decide which firewall ports to leave open:

Step 6: Save your firewall configuration

Type the following command to save the settings you've configured and restart your firewall:

iptables -L -n

iptables-save | sudo tee /etc/sysconfig/iptables

service iptables restart

Tools to assist you with the iptables configuration:

If this is too complicated for you, you can use tools such as fwbuilder or UFW. Here, we will run you through the UFW Uncomplicated Firewall.

The UFW is a front-end for iptables that makes configuring the firewall easier while working with iptables.

Step 1: Type this command into the terminal to install UFW:

apt-get install ufw

Step 2: Next, enable the firewall:

ufw enable

Step 3: enable the default settings.

ufw default deny incoming
ufw default allow outgoing

This will deny all incoming connections. To specify which ones to allow – do the following:

Step 4: To allow specific connections. For example, SSH-

ufw allow ssh

Step 5: ensure the firewall is saved:

ufw status verbose

Rules may be deleted with the following command:

ufw delete allow ssh