PERIYAR UNIVERSITY

(NAAC 'A++' Grade with CGPA 3.61 (Cycle - 3)
State University - NIRF Rank 56 - State Public University Rank 25
SALEM - 636 011, Tamil Nadu, India

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

MASTER OF COMPUTER APPLICATIONS
SEMESTER – I

CORE – IV: LINUX AND SHELL PROGRAMMING LAB
(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)
MCA 2024 admission onwards

Core Course – IV LAB

LINUX AND SHELL PROGRAMMING LAB

Prepared by:
Centre for Distance and Online Education (CDOE)
Periyar University
Salem – 636011.

SYLLABUS

LINUX AND SHELL PROGRAMMING - LAB

COURSE OBJECTIVES

- To enable the students to study and understand the efficiency of Linux shell script.
- To demonstrate the File Backup process.
- To develop and implement the shell script for GUI processing.
- To develop and implement the shell script for IPC and Networking.
- To demonstrate PostgreSQL.

LIST OF EXPERIMENTS

1. Write a Shell Script program to calculate the number of days between two dates.
2. Write a Shell Script program to check systems on local network using control structures with user input.
3. Write a Shell Script program to check systems on local network using control structures with file input.
4. Write a Shell Script program to demonstrate the script control commands.
5. Write a Shell Script program to demonstrate the Shell script function.
6. Write a Shell Script program to demonstrate the Regular Expressions.
7. Write a Shell Script program to demonstrate the sed and awk Commands.
8. Write a Shell Script program to demonstrate the File Backup process through creating a daily archive location.
9. Write a Shell Script program to create a following

   GUI tools. a) Creating text menus

   b) Building text window widgets

10. Write a Shell Script program to demonstrate to connect a PostgreSQL database and performing CRUD operations.

COURSE OUTCOMES

On the successful completion of the course, students will be able to

CO1:	To understand, apply and analyze the concepts and methodology of Linux shell programming	K1-K6
CO2:	To comprehend, impart and apply fundamentals of control structure and script controls	
CO3:	To understand, analyses and evaluate the functions, graphical desktop interface and editors	
CO4:	To collaborate, apply and review the concepts and methodology of regular expression and advanced gawk	
CO5:	To comprehend, use and analyze the advance concepts such as alternate	

K1- Remember, K2- Understand, K3- Apply, K4- Analyze, K5- Evaluate, K6- Create

MAPPING WITH PROGRAMME OUTCOMES

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10
CO1	H	H	H	L	H	L	L	L	M	L
CO2	H	H	H	L	H	L	L	L	M	L
CO3	H	H	H	L	H	L	L	L	M	H
CO4	H	H	H	L	H	L	L	L	M	L
CO5	H	H	H	L	H	L	L	L	M	H

H- High; M-Medium; L-Low

CONTENTS

S.NO	TITLE OF THE PROGRAM	PAGE NO
1.	CALCULATE THE NUMBER OF DAYS BETWEEN TWO DATES	6
2.	CHECK SYSTEM ON LOCAL NETWORK USING CONTROLS STRUCTURE WITH USER INPUT	9
3.	CHECK SYSTEM ON LOCAL NETWORK USING CONTROLS STRUCTURE WITH FILE INPUT	12
4.	THE SCRIPT CONTROL COMMAND	16
5.	SHELL SCRIPT FUNCTION	20
6.	REGULAR EXPRESSIONS	23
7.	SED AND GAWK COMMANDS	26
8.	DEMOSTRATE FILE BACKUP PROCESS THROUGH CREATING A DAILY ARCHIEVE LOCATIONS	31
9.	CREATE A FOLLOWING GUI TOOLS (A) CREATING TEXT MENUS. (B) BUILDING TEXT WINDOW WIDGETS	34
10.	CONNECT A POSTGRESQL DATABASE AND PERFORMING CRUD OPERATIONS	42

1 CALCULATE THE NUMBER OF DAYS BETWEEN TWO DATES

AIM:

To write the shell script program calculate the number of days between to two dates.

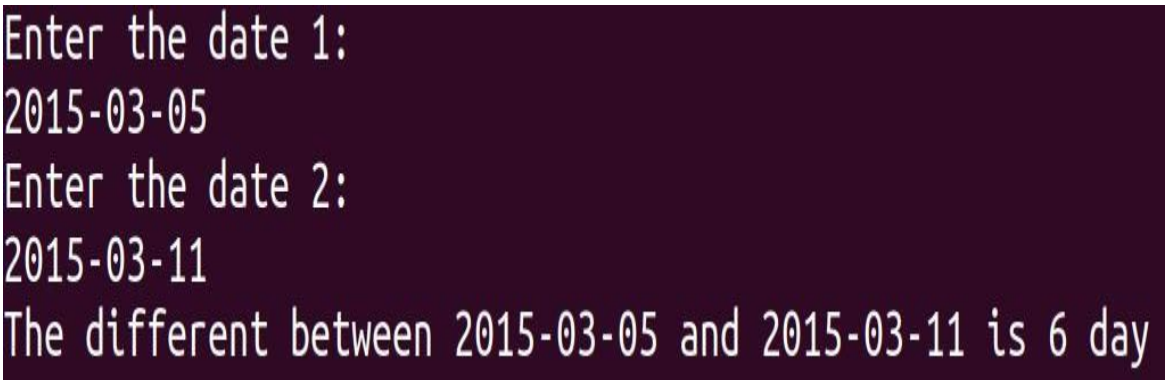
PROCEDURE:

STEP 1: Start the process.
STEP 2: Open the gedit editor with the filename.sh
STEP 3: Enter the date 1 and read by the user input and store in variable d1.
STEP 4: Enter the date 2 and read by the user input and store in variable d2.
STEP 5: Calculate the difference in days between the two days.
STEP 6: Print the values of dates and save.
STEP 7: Set the permission using $ chmod -R 777 .
STEP 8: Run the shell using ./filename.sh
STEP 9: Stop the process

SOURCE CODE:

```
echo "Enter the date 1: " read
d1
echo "Enter the date 2: " read
d2
days=$(( ($(date -d $d2 +%s) - $(date -d $d1 +%s)) / 86400))
echo "The different between $d1 and $d2 is $days day"
```

OUTPUT:



RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

2 CHECK SYSTEM ON LOCAL NETWORK USING CONTROLS STRUCTURE WITH USER INPUT

AIM:

To write the shell script program to check system on local network using structure with user input.

PROCEDURE:

Step 1: Start the process.
Step 2: Checking system on local network by using control structure net user input.
Step 3: Check for command line option here by using gets opts.
Step 4: Ping or ping 6 command is a quick wave to determine if a system is up and operating on the network.
Steps: After finishing the step 4 to get the permission using chmod -R 777 .
Step 6: Run the code using ./filename.sh
Step 7: The ip address parameters are accidently not included. The user to produce a message and exit (sh filename.sh -tIPv4 IPaddress) or IPv4 domain name or IPv6 IPaddress.
Step 8: Stop the process.

SOURCE CODE:

```
while getopts t: opt do
case "$opt" in t)
if [ $OPTARG = "IPv4" ]
then
        pingcommand=$(which ping)
elif [ $OPTARG = "IPv6" ]
then
        pingcommand=$(which ping6)
 fi;;

*) echo "Usage: -t IPv4 or -t IPv6" echo
"Exiting script..."
exit;;
esac
shift $(( $OPTIND - 1 ))
if [ $# -eq 0 ]
then
        echo "\nIP Address(es) parameters are missing." echo
        "\nExiting script..."
        exit
fi
for ipaddress in "$@" do
        echo "\nChecking system at $ipaddress..." echo
        $pingcommand -q -c 3 $ipaddress echo
done

exit
done
```

OUTPUT:

```
elanchezhian@elanchezhian-virtual-machine:~/Desktop/Linux$ sh po2.sh -t IPv4 192.168.240.1

Checking system at 192.168.240.1...

PING 192.168.240.1 (192.168.240.1) 56(84) bytes of data.

--- 192.168.240.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.744/2.015/2.421/0.292 ms
```

RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

3 CHECK SYSTEM ON LOCAL NETWORK USING CONTROLS STRUCTURE WITH FILE INPUT

AIM:
To write the shell script program to check system on local network wing controls structure with file input.

PROCEDURE:
Step 1: Start the process.
Step 2: Creating file with text and inserting with group of Ip address
Step 3: Enter the <filename. Txt> if the file is readable and is not empty. It check the system, Ip group file import.
Step 4: The ping or ping 6 command is quick way to determine if a system is up and Operating a network.
Step 5: If the file is either not a file is empty or did not readable file import and then exiting script.
Step 6: Stop the process

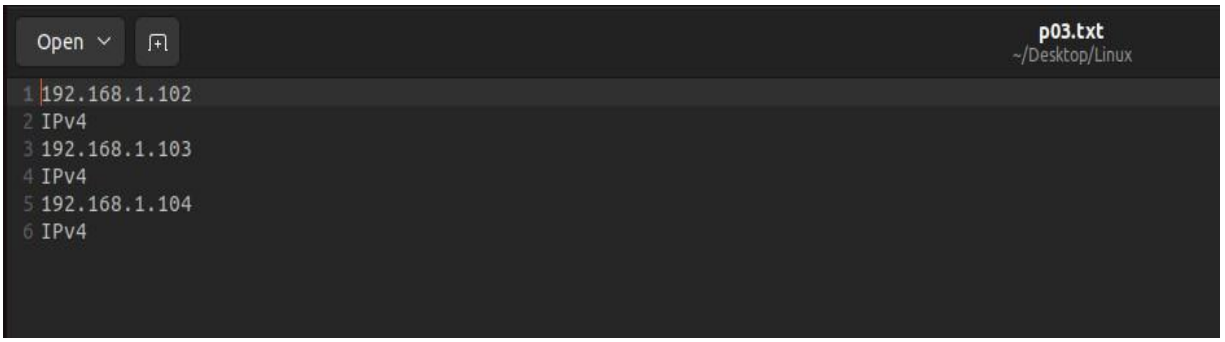
SOURCE CODE:

```
echo "\nPlease enter the file name with an absolute directory
reference...\n"
choice=0
while [ $choice -eq 0 ]
do
        read -p "Enter name of file:" filename if [ -z
        $filename ]
        then
                quitanswer=""
        else    choice=1

        fi
done



if [ -s $filename ] && [ -r $filename ]
then
        echo "$filename is a file, is readable, and is not empty." echo
        cat $filename | while read line do
        ipaddress=$line
        read line
        iptype=$line
        if [ $iptype = "IPv4" ]
        then
                pingcommand=$(which ping)
        else
                pingcommand=$(which ping6)

        fi

echo "Checking system at $ipaddress..."
$pingcommand -q -c 3 $ipaddress done
```

```
echo "\nFinished processing the file. All systems checked." else
echo "\n$filename is either not a file, is empty, or is not readable by you. Exiting
script..."
fi
exit
```

OUTPUT:

➢ P03.txt file




```
Please enter the file name with an absolute directory reference...

Enter name of file:p03.txt
p03.txt is a file, is readable, and is not empty.

Checking system at 192.168.1.102...
PING 192.168.1.102 (192.168.1.102) 56(84) bytes of data.

--- 192.168.1.102 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2035ms

Checking system at 192.168.1.103...
PING 192.168.1.103 (192.168.1.103) 56(84) bytes of data.

--- 192.168.1.103 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2055ms

Checking system at 192.168.1.104...
PING 192.168.1.104 (192.168.1.104) 56(84) bytes of data.

--- 192.168.1.104 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2054ms


Finished processing the file. All systems checked.
```

RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

4 THE SCRIPT CONTROL COMMAND

AIM:
To determine the script control command.

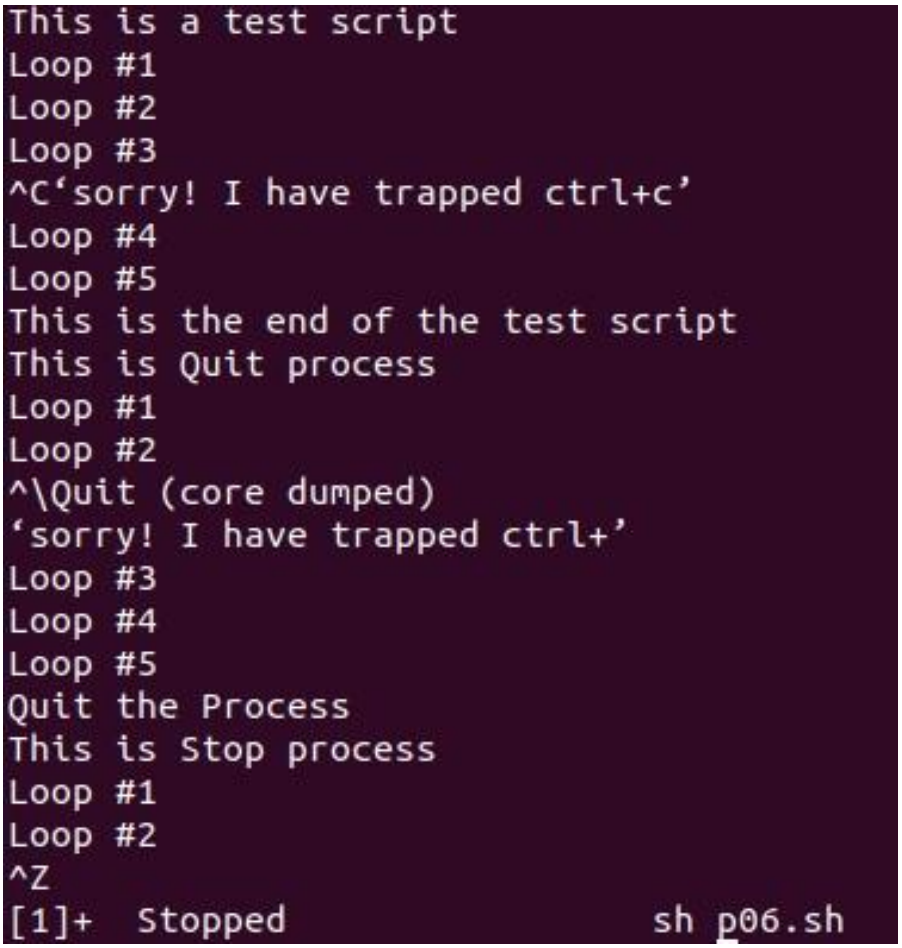
PROCEDURE:
Step 1: Start the process.
Step 2: The following script control commands are used..
a. sigint - interupt from keyboard
b. sigquit - quit from keyboard.
c. sigstop - stop process (ctrl + c and ctrl + z)
Step 3: This command which is used to response to hardware signal and other events by trap command
Step 4: Stop the process.

SOURCE CODE:

```
#SIGINT
trap "echo ‘sorry! I have trapped ctrl+c’" INT
echo "This is a test script"
count=1
while [ $count -le 5 ]
do
        echo "Loop #$count"
        sleep 2
        count=$(( $count+1 ))

done

echo "This is the end of the test script"

#SIGQUIT
trap "echo ‘sorry! I have trapped ctrl+\’" QUIT
echo "This is Quit process"
count=1
while [ $count -le 5 ]
do
        echo "Loop #$count"
        sleep 2
        count=$(( $count+1 ))

done

echo "Quit the Process"
```

```
#SIGSTOP
trap STOP
echo "This is Stop process"
count=1
while [ $count -le 5 ]
do
        echo "Loop #$count"
        sleep 2
        count=$(( $count+1 ))

done

echo "Stop the Process"
```

OUTPUT:



RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

5 SHELL SCRIPT FUNCTION

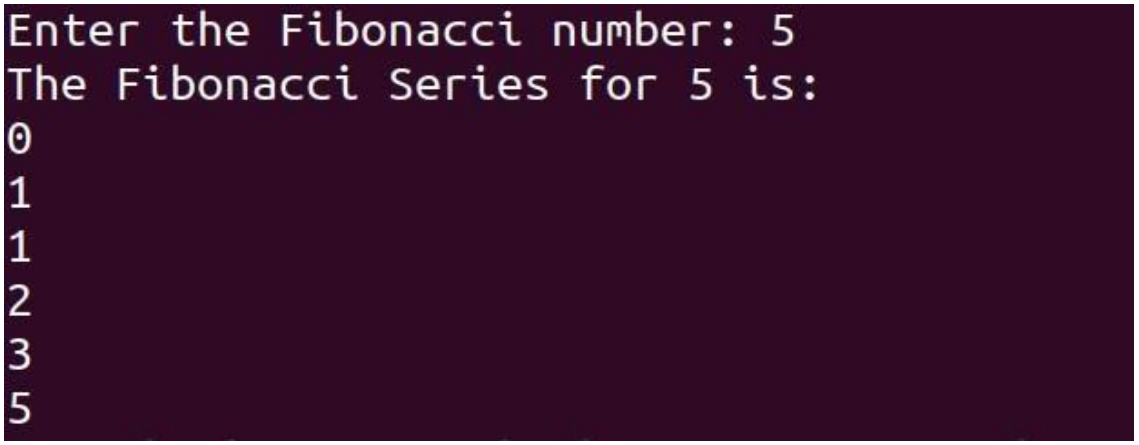
AIM:
To determine the shell script function.

PROCEDURE:
Step 1: Start the process.
Step 2: Declare the variables like
i = 0
f1 = 0
f2 = 1 and n
Step 3: Read the value by the user input and store the value n.
Step 4: Declare the function called fib().
Step 5: While loop to check the condition, if the condition is true, it will go to fibonacci calculation otherwise terminate.
Step 6: Stop the process.

SOURCE CODE:

```
read -p "Enter the Fibonacci number: " n fib(){
        i=0
        f1=0
        f2=1
        echo "The Fibonacci Series for $n is:" while [
        $i -le $n ]
        do
                echo "$f1"
                temp=$(($f1+$f2))
                f1=$f2
                f2=$temp
                i=$((i+1))


        done
}
 fib
```

OUTPUT:



RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

6 REGULAR EXPRESSION

AIM:
To write the Regular Expression.

PROCEDURE:
Step 1: Stop the process.
Step 2: Create a file with .txt and store fruits name in it
Step 3: Using grep command to identity the conditions to be given in the echo command.
Step 4: By using pipe symbol (|) to grep the given command.
Step 5: Basic regular expression.
a. '.' - find all Original word
b. '*' - find out the match upto 0 or more occurrence.
c. '^' - find out the starting of the string (character)
d. '\' - find out single space.
e. '?' - find out exactly one character in string.
f. '[]' - find out the word
Step 6: Stop the process

SOURCE CODE:

```
fruits_file=$(cat fruit.txt | grep App.e)
echo "\n1. Using '.' to find out all the original word wheres given word is 'App.e'"
echo "Output:\n$fruits_file" fruits_file=$(cat
fruit.txt | grep Ap*le)
echo "\n2. Using '*' to find out all the fruits name of 'Ap' one after another in it"
echo "Output:\n$fruits_file"
fruits_file=$(cat fruit.txt | grep ^B)
echo "\n3. Using '^' to find out all the words that start with the letter
'B'"
echo "output:\n$fruits_file" fruits_file=$(cat
fruit.txt | grep "\ ")
echo "\n4. Using '\' to find out all the fruits name that has single space in their full
name"
echo "Output:\n$fruits_file" fruits_file=$(cat
fruit.txt | grep -E Ch?)
echo "\n5. Using '?' to find out all the fruits name that has ‘Ch’ in it"
echo "Output:\n$fruits_file" fruits_file=$(cat fruit.txt |
grep -E "(fruit)")
echo "\n6. Using ‘()’ to find out all the fruits name that has word
‘fruit’ in it"
echo "Output:\n$fruits_file"
```

OUTPUT:

➢ fruit.txt file


```
1. Using '.' to find out all the original word wheres given word is 'App.e'
Output:
Apple
custard Apple

2. Using '*' to find out all the fruits name of 'Ap' one after another in it
Output:
Apple
custard Apple

3. Using '^' to find out all the words that start with the letter 'B'
output:
Banana
Bil Berry
Black Berry

4. Using '\' to find out all the fruits name that has single space in their full name
Output:
Bil Berry
Black Berry
custard Apple
Chico Fruit
Goji Berry
Juniper Berry
Passuib Fruit
Star Fruit
Salal Berry
Ugli Fruit

5. Using '?' to find out all the fruits name that has 'Ch' in it
Output:
Currant
Cherimoya
Chico Fruit

6. Using '()' to find out all the fruits name that has word 'fruit' in it
Output:
Drangonfruit
```

RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

7 SED AND GAWK COMMANDS

AIM:
To determine the SED and AWK commands.

PROCEDURE:
Step 1: start the process.
Step 2: Create file with .txt and type appropriate content to perform SED command.
Step 3: Use the command for substituting occurrence, string, duplicating replaced line, printing replace line, range and delete line.
Step 4: Use the gawk command (to be install the following command, sudo apt install gawk)
used to find the grade for mark which entered by user input.
Step 5: Stop the process

SOURCE CODE (sed command) :

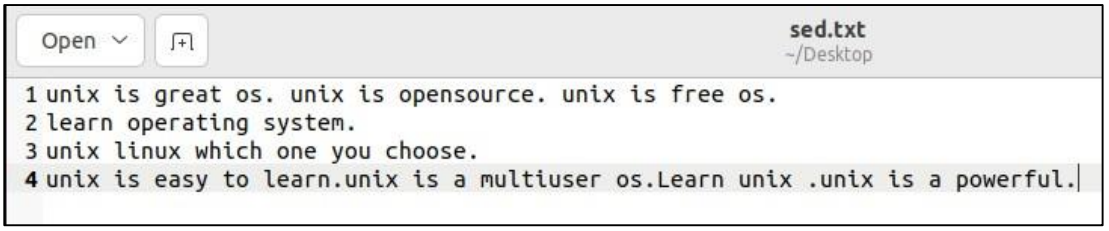
```
echo "1.Replacing or substituting string:" echo "-----------------------------------
------" sed 's/unix/linux/' sed.txt
echo
echo "2.Replacing the nth occurrence of a pattern in a line:" echo "----------
--------------------------------------------------" sed 's/unix/linux/2' sed.txt
echo
echo "3.Replacing all the occurrence of the pattern in a line:" echo "----------
--------------------------------------------------" sed 's/unix/linux/g' sed.txt
echo
echo "4.Replacing from nth occurrence to all occurrences in a line:" echo "--
---------------------------------------------------------------------" sed 's/unix/linux/3g'
sed.txt
echo
echo "5.Replacing string on a specific line number:" echo "----------------------
--------------------------------" sed '3 s/unix/linux/' sed.txt
echo
echo "6.Duplicating the replaced line with /p flag:" echo "-------------------------
---------------------------" sed 's/unix/linux/p' sed.txt
echo
echo "7.Printing only the replaced lines:" echo "-----------------------------------
----" sed -n 's/unix/linux/p' sed.txt
echo
echo "8.Replacing string on a range of lines:" echo "-------------------------------
---------------" sed '2,$ s/unix/linux/' sed.txt
echo

echo "9.Deleting lines from a particular file:" echo "-------------------------------
-------------" sed '2,4d' sed.txt
```

SOURCE CODE (gawk command) :

```
gawk 'BEGIN { print "Enter the mark:" getline
mark < "-"
if (mark >= 90) print "A+" else if( mark
>= 80) print "A" else if( mark >= 70)
print "B+" else if( mark >= 60) print "B"
else if( mark >= 50) print "C+" else
print "Fail" }'
```

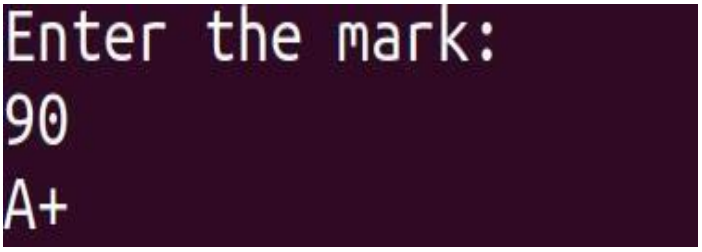
OUTPUT (sed command):

➢ Sed.txt file:




```
1.Replacing or substituting string:
---------------------------------------
linux is great os. unix is opensource. unix is free os.
learn operating system.
linux linux which one you choose.
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

2.Replacing the nth occurrence of a pattern in a line:
-------------------------------------------------------------
unix is great os. linux is opensource. unix is free os.
learn operating system.
unix linux which one you choose.
unix is easy to learn.linux is a multiuser os.Learn unix .unix is a powerful.

3.Replacing all the occurrence of the pattern in a line:
-------------------------------------------------------------
linux is great os. linux is opensource. linux is free os.
learn operating system.
linux linux which one you choose.
linux is easy to learn.linux is a multiuser os.Learn linux .linux is a powerful.

4.Replacing from nth occurrence to all occurrences in a line:
-------------------------------------------------------------------------
unix is great os. unix is opensource. linux is free os.
learn operating system.
unix linux which one you choose.
unix is easy to learn.unix is a multiuser os.Learn linux .linux is a powerful.

5.Replacing string on a specific line number:
-----------------------------------------------------
unix is great os. unix is opensource. unix is free os.
learn operating system.
linux linux which one you choose.
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

6.Duplicating the replaced line with /p flag:
-----------------------------------------------------
linux is great os. unix is opensource. unix is free os.
linux is great os. unix is opensource. unix is free os.
learn operating system.
linux linux which one you choose.
linux linux which one you choose.
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

7.Printing only the replaced lines:
---------------------------------------
linux is great os. unix is opensource. unix is free os.
linux linux which one you choose.
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

8.Replacing string on a range of lines:
-----------------------------------------------
unix is great os. unix is opensource. unix is free os.
learn operating system.
linux linux which one you choose.
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

9.Deleting lines from a particular file:
-----------------------------------------------
unix is great os. unix is opensource. unix is free os.
```

OUTPUT (gawk command) :



RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

8 DEMOSTRATE FILE BACKUP PROCESS THROUGH CREATING A DAILY ARCHIEVE LOCATIONS

AIM:
To write the demonstrate file backup process through creating a daily archieve locations.

PROCEDURE:
Step 1: Start the process.
Step 2: First select the files to backup and store txt format
Step 3: Give the file name to the archieve file and read the file name and destination path
Step 4: The file will be archieve in destination path and archieved file is backup.
Step 5: To display archieve file in terminal user using tar -tzvf
Step 6: Stop the process.

SOURCE CODE:

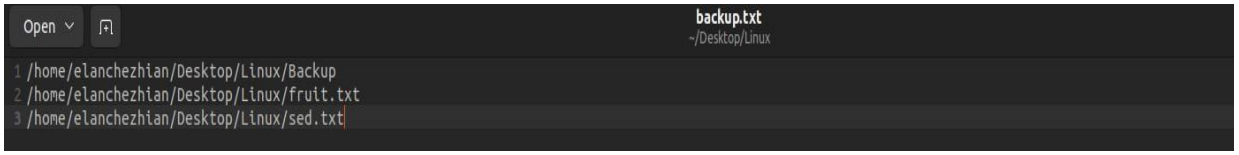
```
DATE=$(date +%y%m%d)
read -p "Give name to the archive file:" file
FILE=$file$DATE.tgz
read -p "Enter the Filename: " SOURCE read -p
"Enter the Destination path: " des
DESTINATION=$des/$FILE
if [ -f $SOURCE ]
then
        echo
else
        echo "$SOURCE doesn't exist, BACKUP INCOMPLETE" exit


fi

FILE_NO=1
exec < $SOURCE read
FILE_NAME while [ $?
-eq 0 ]
do
        if [ -f $FILE_NAME -o -d $FILE_NAME ]
        then
                FILE_LIST="$FILE_LIST $FILE_NAME"
         else
                echo "$FILE_NAME doesn’t exit, thus it is not included" echo
                "BACKUP is still on process"
                echo


         fi

FILE_NO=$[$FILE_NO+1]
read   FILE_NAME
done
echo "Starting Archive..."
tar -czf $DESTINATION $FILE_LIST 2>/dev/null echo "Archive
COMPLETED at $DESTINATION" exit
```

OUTPUT:

➢ Creating the file to Store the backup file:

```
elanchezhian@elanchezhian-virtual-machine:~/Desktop/Linux$ gedit backup.txt
```

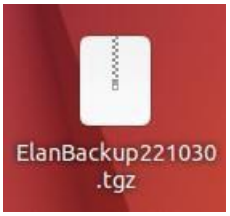
➢ Backup Files:



➢ Running the Script to Backup the Files

```
Give name to the archive file:ElanBackup
Enter the Filename : backup.txt
Enter the Destination path: /home/elanchezhian/Desktop

Starting Archive...
Archive COMPLETED at /home/elanchezhian/Desktop/ElanBackup221030.tgz
```

➢ Archive file (ElanBakup221030.tgz):



➢ Listing of the Archive contents from a Terminal Prompt Type:

```
elanchezhian@elanchezhian-virtual-machine:~/Desktop/Linux$ tar -tzvf /home/elanchezhian/Desktop/ElanBackup221030.tgz
drwxrwxr-x elanchezhian/elanchezhian 0 2022-10-30 09:25 home/elanchezhian/Desktop/Linux/Backup/
-rw-rw-r-- elanchezhian/elanchezhian 239 2022-10-29 12:06 home/elanchezhian/Desktop/Linux/Backup/sample1.sh
-rw-rw-r-- elanchezhian/elanchezhian 166 2022-10-28 22:06 home/elanchezhian/Desktop/Linux/fruit.txt
-rw-rw-r-- elanchezhian/elanchezhian 133 2022-10-28 22:21 home/elanchezhian/Desktop/Linux/sed.txt
```

RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

9 CREATE A FOLLOWING GUI TOOLS (A) CREATING TEXT MENUS. (B) BUILDING TEXT WINDOW WIDGETS

AIM:
To write the shell script to create the following GUI Tools.
a) Creating text windows
b) Building text windows widget

PROCEDURE:
PART A: Creating text windows
Step 1: Start the process.
Step 2: Define “diskspace”, " memusage", “menu functions”.
Step 3: While I condition is true, mem function will be executed.
Step 4: User must select option from the menu list.
1. Display diskspace (of -k)
2. Display logged on space (who)
3. Display memory reusage (mem info)
4. Exit program.
Step 5: Using case command to select the options.
Step 6: Stop the process.
PART B: Building text window widgets
Step 1: start the process.
Step 2: Install dialog box by using.
‘sudo apt-get install -y dialog" command.
Step 3: Insert dialog commands in the function “diskspace”, "whoseon", "memusage":
Step 4: Use select command to display menu list and get input from user by using case
commands.
Step 5: Stop the process.

SOURCE CODE:

A) Creating Text Menus

```
diskspace() {


} diskspace whoseon() {


        } whoseon memusage() {


        } memusage menu(){


        }
        menu
clear df -k

clear who

clear
cat /proc/meminfo


clear echo
echo “\t\t\t\tSys Admin Menu\n"
echo “\t\t1. Display disk space"
echo “\t\t2. Display logged on users" echo “\t\t3. Display memory usage"
echo “\t\t0. Exit program\n\n"
echo
echo "\t\t\tEnter option: " read option
echo
```

```
while [ True ]
do
        menu
        case $option in
        0)
        break ;;
        1)
        diskspace ;;
        2)
        whoseon ;;
        3)
        memusage ;;
        *)
        clear
        echo "Sorry, wrong selection";;
        esac
echo "\n\n\t\t\tHit any key to continue" read
line
do
ne
cle
ar
```

B) Text window widgets temp=$(mktemp -t test.XXXXXX) temp2=$(mktemp -t test2.XXXXXX) function diskspace {

```
        clear
        df -k> $temp
        dialog --textbox $temp 20 50
}
function whoseon {
        clear
        who> $temp
        dialog --textbox $temp 20 50
```

```
}
 function memusage {
        clear
        cat /proc/meminfo> $temp dialog --
         textbox $temp 20 50
 }
while [ 1 ]
do
        clear
        dialog --menu "Sys Admin Menu" 20 30 10 1 "Display diskspace" 2
        "Display users" 3 "Display memory usage"
         2> $temp2
         if [ $? -eq 1 ]
         then
                break
         fi
selection=$(cat $temp2)
case $selection in
         1) diskspace ;;
         2) whoseon ;;
         3) memusage ;;
         *) dialog --msgbox "Sorry, invalid selection" 10 30
esac
done
clear
rm -f $temp 2> /dev/null rm -f
$temp2 2> /dev/null
```

OUTPUT:

A) Text Menus

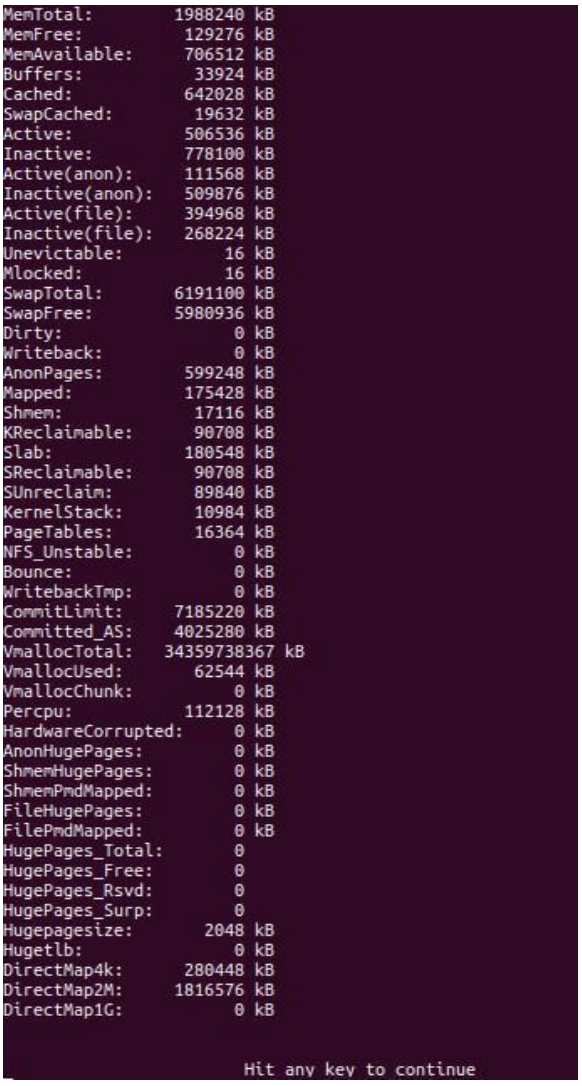
```
                        Sys Admin Menu

              1. Display disk space
              2. Display logged on users
              3. Display memory usage
              0. Exit program


                     Enter option:
```

1. Displaying Disk Space

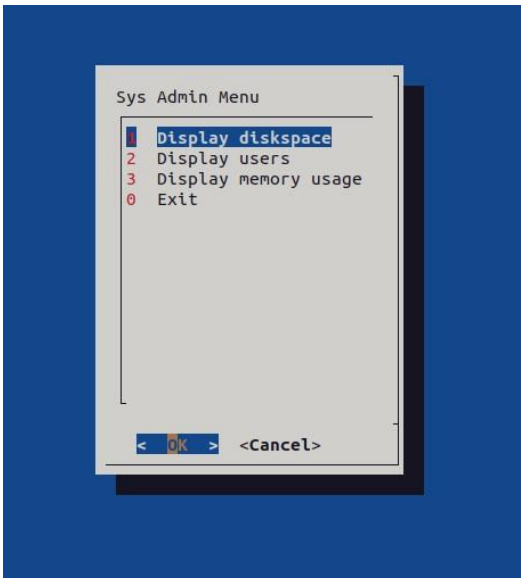
```
                    Enter option:
1

Filesystem     1K-blocks     Used Available Use% Mounted on
tmpfs             198824     1820    197004   1% /run
/dev/sda3       91789000 14707824  72372620  17% /
tmpfs             994120        0    994120   0% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
/dev/sda2         524252     5364    518888   2% /boot/efi
tmpfs             198824     4728    194096   3% /run/user/1000
/dev/sr0          129778   129778         0 100% /media/elanchezhian/CDROM
/dev/sr1         3737140  3737140         0 100% /media/elanchezhian/Ubuntu 22.04.1 LTS amd64
/dev/fd0            1424        9      1415   1% /media/floppy0


                    Hit any key to continue
```

2. Displaying Logged Users

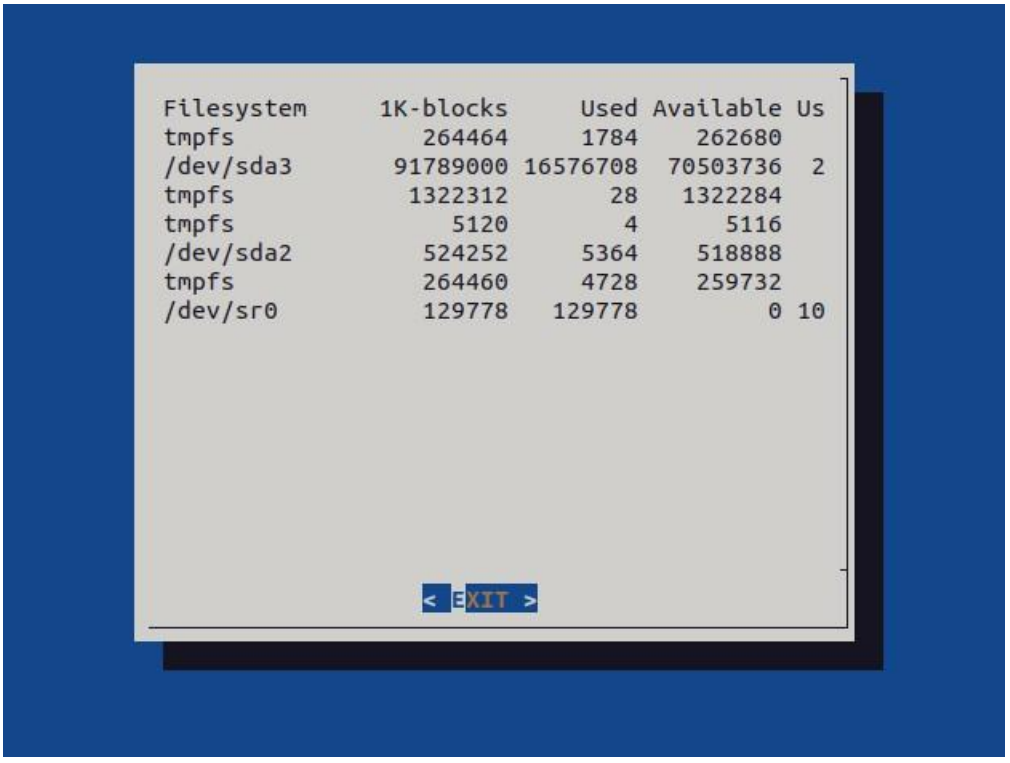
3. Displaying Memory Usage



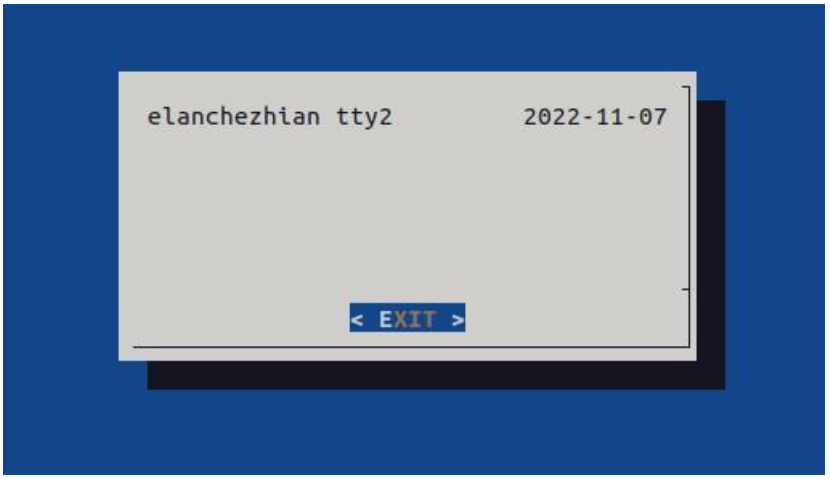
B) Text Window Widgets



1. Displaying Disk Space:



2. Displaying Logged Users:



3. Displaying Memory Usage

RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

10 CONNECT A POSTGRESQL DATABASE AND PERFORMING CRUD OPERATIONS

AIM:
To write the shell script to demonstrate to connect it program postgresql database and performing.

PROCEDURE:
Step 1: Start the process.
Step 2: Install to connect postgresql by using “sudo apt install postgresql postgresql-contrib".
Step 3: Launch sql shell (psqd program tool by using “sudo -i -u postgre”)
Step 4: Create database and name it.
Steps: Then create table and insert one field and insert the values by using INSERT operation.
Step 6: Update the table by using UPDATE operation.
Step 7: Delete the values from table by using DROP operation.
Step 8: Display the values by using QUERY select from < table-name >
Step 9: Delete the database by using drop < database name >
Step 10: Stop the process.

SOURCE CODEAND OUTPUTS:

➢ To view the list of databases by using \l command:

```
elanchezhian@elanchezhian-virtual-machine:~$ sudo -i -u postgres
[sudo] password for elanchezhian:
postgres@elanchezhian-virtual-machine:~$ psql
psql (14.5 (Ubuntu 14.5-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 postgres  | postgres | UTF8     | en_IN   | en_IN |
 template0 | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(3 rows)
```

➢ Creating Database:

```
postgres=# CREATE DATABASE  bank_details;
CREATE DATABASE
```

➢ Listing the Database and Checking Database Which Created by User:

```
postgres=# \l
                                  List of databases
     Name     |  Owner   | Encoding | Collate | Ctype |   Access privileges
--------------+----------+----------+---------+-------+-----------------------
 bank_details | postgres | UTF8     | en_IN   | en_IN |
 postgres     | postgres | UTF8     | en_IN   | en_IN |
 template0    | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
              |          |          |         |       | postgres=CTc/postgres
 template1    | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
              |          |          |         |       | postgres=CTc/postgres
(4 rows)
```

➢ Changing Path to the Created Database(bank_details):

```
postgres=# \c bank_details;
You are now connected to database "bank_details" as user "postgres".
```

➢ Creating Table:

```
bank_details=# CREATE TABLE BankDetails(acc_no integer, name text, balance numeric, acc_type text);
CREATE TABLE
```

➢ Inserting Values to Table:

```
bank_details=# INSERT INTO BankDetails VALUES(50706,'Elanchezhian',1000.00,'Savings');
INSERT 0 1
bank_details=# SELECT * FROM BankDetails;
 acc_no |     name     | balance | acc_type
--------+--------------+---------+----------
  50706 | Elanchezhian | 1000.00 | Savings
(1 row)
```

➢ Inserting Multiple Values to Table:

```
bank_details=# INSERT INTO BankDetails VALUES(50707,'Hariharan',500.00,'Savings'),(50708,'Lachu',5000.00,'Current'),(50709,'Sanjai',800
0.00,'Current'),(50710,'Mahadevan',7000.00,'Savings');
INSERT 0 4
bank_details=# SELECT * FROM BankDetails;
 acc_no |     name     | balance | acc_type
--------+--------------+---------+----------
  50706 | Elanchezhian | 1000.00 | Savings
  50707 | Hariharan    |  500.00 | Savings
  50708 | Lachu        | 5000.00 | Current
  50709 | Sanjai       | 8000.00 | Current
  50710 | Mahadevan    | 7000.00 | Savings
(5 rows)
```

➢ Updating the Colum in Table:

```
bank_details=# UPDATE BankDetails SET balance=3000.00 WHERE balance=500.00;
UPDATE 1
bank_details=# SELECT * FROM BankDetails;
 acc_no |     name     | balance | acc_type
--------+--------------+---------+----------
  50706 | Elanchezhian | 1000.00 | Savings
  50708 | Lachu        | 5000.00 | Current
  50709 | Sanjai       | 8000.00 | Current
  50710 | Mahadevan    | 7000.00 | Savings
  50707 | Hariharan    | 3000.00 | Savings
(5 rows)
```

➢ Deleting the Colum in Table:

```
bank_details=# DELETE FROM BankDetails WHERE acc_no=50710;
DELETE 1
bank_details=# SELECT * FROM BankDetails;
 acc_no |     name     | balance | acc_type
--------+--------------+---------+----------
  50706 | Elanchezhian | 1000.00 | Savings
  50708 | Lachu        | 5000.00 | Current
  50709 | Sanjai       | 8000.00 | Current
  50707 | Hariharan    | 3000.00 | Savings
(4 rows)
```

➢ Deleting the Table:

```
bank_details=# DROP TABLE  BankDetails;
DROP TABLE
```

➢ Checking the Table if Exist or Not:

```
bank_details=# SELECT * FROM BankDetails;
ERROR:  relation "bankdetails" does not exist
LINE 1: SELECT * FROM BankDetails;
                      ^
```

➢ Deleting the Database and Listing of Databases:

```
bank_details=# \c postgres;
You are now connected to database "postgres" as user "postgres".
postgres=# DROP DATABASE bank_details;
DROP DATABASE
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 postgres  | postgres | UTF8     | en_IN   | en_IN |
 template0 | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(3 rows)
```

➢ Quit from Database:

```
postgres=# \q
```

➢ Logout from psql:

```
postgres@elanchezhian-virtual-machine:~$
logout
elanchezhian@elanchezhian-virtual-machine:~$
```

RESULT:

THUS THE ABOVE SHELL SCRIPT HAS BEEN EXECUTED SUCESSFULLY

Reference Books:

1. Michael Kerrisk, “The Linux Programming Interface”, No Starch Press, Inc., 2010.
2. Richard Blum and Christine Bresnahan, “Linux Command Line and Shell Scripting Bible”, Third Edition, Wiley, 2015.
3. W. Richard Stevens, Stephen A. Rago, “Advanced Programming in the UNIX Environment”, Third Edition, Addison-Wesley Professional Computing Series, 2013.
4. Shantanu Tushar and Sarath Lakshman, “Linux Shell Scripting Cook Book”, 2nd edition, Packt Publishing, 2013.
5. Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley and Dan Mackin, “UNIX and Linux System Administration Handbook”, 5th Edition, Pearson Education, Inc. 2018.

Website References:

1. https://linuxcommand.org/
2. https://tldp.org/
3. https://www.shellcheck.net/
4. https://www.explainshell.com/
5. https://www.pluralsight.com/cloud-guru