**INTRODUCTION**

**NAME:** KARTIKEYA NARIAN KHANNA

**BRANCH:** DEVOPS

**ROLL.NO:** R171219049

**SAP ID:** 500078285

**SEMESTER:** 3

**SUBJECT:** DEVELOPMENT AUTOMATION

**SUBMITTED TO:** Ms. NITIKA GOENKA

**DATE OF SUBMISSION:** 25TH NOV,2020

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Experiment** | **Pg.No** | **REMARKS** |
| 1 | Input | 4 |  |
| 2 | Output | 5 |  |
| 3 | Streams | 6 |  |
| 4 | Variable | 6 |  |
| 5 | Substitution | 7 |  |
| 6 | Functions | 8 |  |
| 7 | Sub-Shell | 8 |  |
| 8 | Conditional Statements | 8 |  |
| 9 | Loops | 12 |  |
| 10 | Archiving Log Files | 15 |  |
| 11 | Discarding Log Files | 16 |  |
| 12 | MYSQL Backup | 18 |  |
| 13 | Email Web Server | 20 |  |
| 14 | Email Running | 22 |  |
| 15 | User Command Validation | 24 |  |
| 16 | Monitor Disk Usage | 26 |  |
| 17 | Move Deleted Files To Recycle Bin | 28 |  |
| 18 | Restores The Deleted Files | 30 |  |
| 19 | Log Delete Actions | 33 |  |
| 20 | Bulk File Downloader | 34 |  |
| 21 | Install LAMP Stack | 36 |  |
| 22 | To Find Commands That Matches The Pattern. | 38 |  |
| 23 | Replace Given Text With New Text | 39 |  |
| 24 | Check Weather Input is Valid | 41 |  |
| 25 | Truncate The Given File | 43 |  |
| 26 | Colourize The Standard Output | 44 |  |
| 27 | Transform The Given Text | 46 |  |
| 28 | Basic Setup | 48 |  |
| 29 | Building Binary From Code | 50 |  |
| 30 | Transforming The Makefile | 51 |  |
| 31 | Archiving Logs With Make | 52 |  |
| 32 | Conditionals In Makefile | 53 |  |
| 33 | Standard Input | 54 |  |
| 34 | Append | 55 |  |
| 35 | Error | 56 |  |
| 36 | Error Redirection | 57 |  |
| 37 | Cat Command | 57 |  |
| 38 | Formatting The Terminal | 58 |  |
| 39 | Utilities Library To Reduce Logic Duplication | 61 |  |

**Experiment No. 2.1**

Introduction to bash and shell scripting

• Setting Up Bash, configurations

• Shell Scripts for various logical and arithmetic tasks

• Shell scripts for various system tasks

**Topics to be covered:**

Introduction to Bash and Shell scripting

1. Input

2. Output

3. Streams

3.1 Stdin

3.2 Stdout

3.3 Stderr

4. Variable

5. Substitution

5.1 Variable Substitution

5.2 Command Substitution

6. Function

7. Sub-Shell

8. Conditional statements

8.1 IF statement Nested IF Ladder IF

8.2 CASE statement

9. Loops

9.1 FOR loop

9.2 WHILE loop

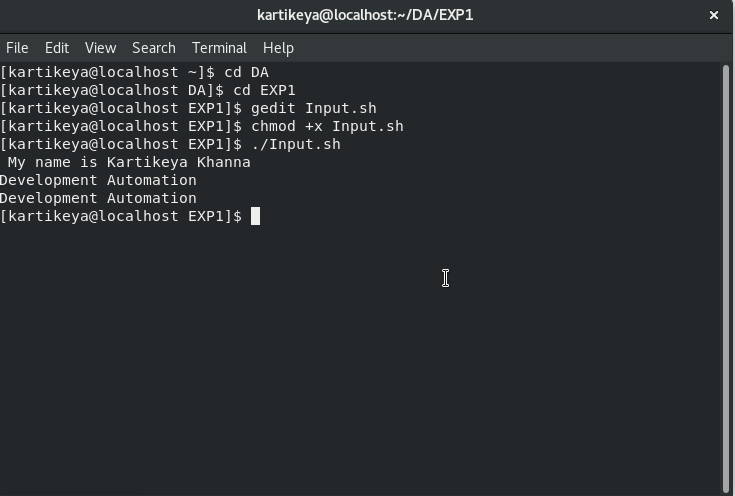
**1. Input**

A script can make use of the Input given to the script to Improve the user accessibility and also it enables the script to get data dynamically during the runtime.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | echo " My name is Kartikeya Khanna" |
|  | read course |
|  | echo "${course}" |

**Output**

****

|  |
| --- |
|  |

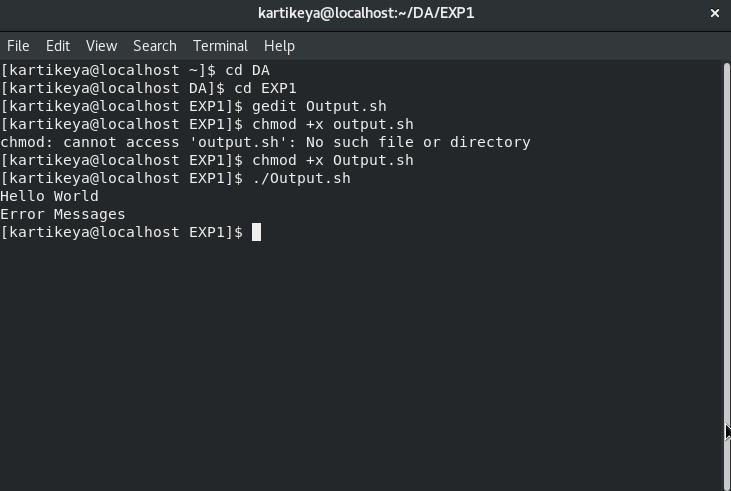
**2. Output**

In Linux, the output is given to the terminal as two streams namely, “stdout” and “stderr”. The former one is used to print any general information to the terminal while the latter one is used to display the error information in the terminal.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | echo "Hello World" |
|  | echo "Error Messages">&2 |

**Output**

****

**3. Streams**

Streams provide an abstraction of communication channels between a computer program and its execution environment. Primarily, there are 3 kinds of streams in most of the Linux operating systems, namely:

1. Stdin
2. Stdout
3. Stderr

1.Stdin

Standard Input is a stream of data that enters the program from the terminal. Unless any output is redirected to the program, the standard input is expected from the keyboard. This stream is generally used in programs to read the data dynamically.

##### 2.Stdout

##### Standard Output is the stream that is used to display an information on the terminal. By default, all the text printed in the terminal will be sent to stdout. This can be a help message, a warning or an information that makes the program easier to use and know the process going on.

##### 3.Stderr

##### Standard Error is a specific stream used to display the error information to the terminal. It is separated from stdout so that the error messages can be captured separately. With this separation, the general output of a program can be shown directly on the terminal while the error-related information can be separately written to a file.

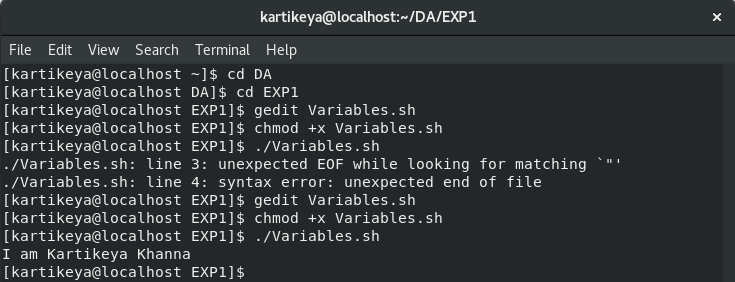
**4.Variable**

A variable is a space labelled by a name in memory that has some data inside. In other words, a variable is a labelled container that contains some data. The data of a variable can be changed wherever requires unless it is declared as read-only.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | myname="Kartikeya Khanna" |
|  | echo "I am ${myname}" |

**Output**

****

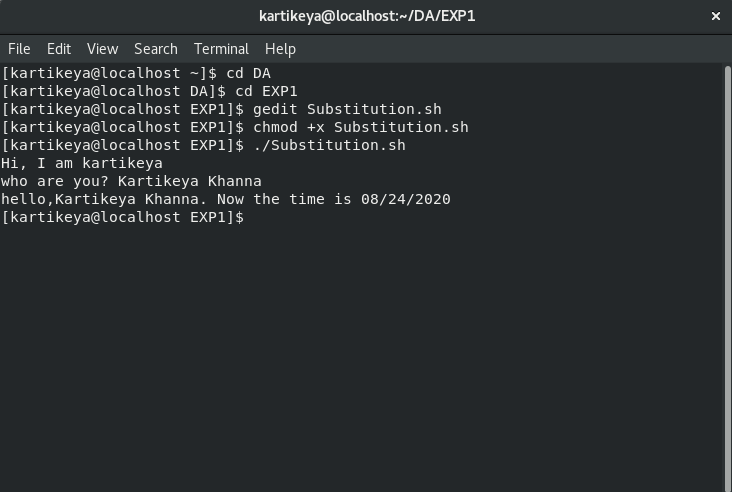
**5.Substitution**

Substitution generally refers to placing a text inside some other text. At a higher level, any command or a script is just a text in a particular format. This enables to use the constructed text in place of any commands.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | readonly greeting="hello" |
|  | current\_time=`date +%x` |
|  | echo "Hi, I am $(whoami)" |
|  | read -e -p "who are you?" myname |
|  | echo "${greeting},$myname. Now the time is $current\_time" |

**Output**

****

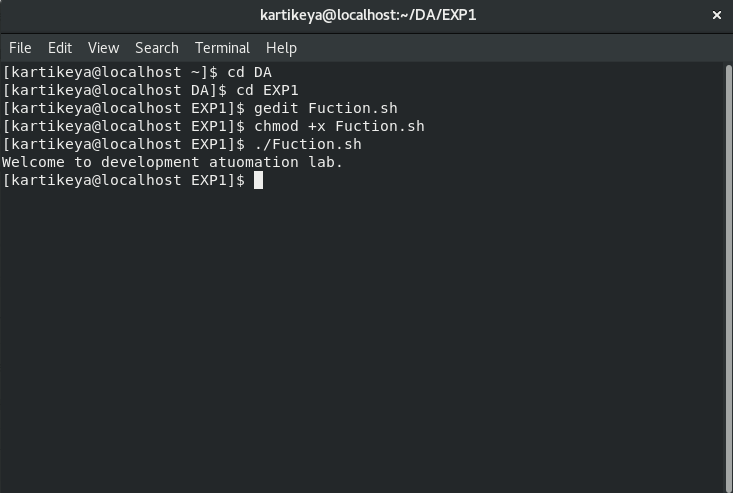
**6.Function**

Functions are used to define a set of actions and execute those actions multiple times across the program whenever they are called. Code duplication can be considerably reduced with the use of functions.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | function welcome { |
|  | echo "Welcome to $1" |
|  | } |
|  | welcome "development atuomation lab." |

**Output**

****

### **7.Sub-Shell**

### The shell is the terminal where we type the commands and execute the programs. This interactive shell can itself call a script that runs in a new process other than the parent shell’s process. Likewise, a running script can run some other script in a new process and this is called as a sub-shell.

### **8.Conditional statements**

### Conditional statements are the vital components in any scripting or programming languages. They enable the ability to take decisions based on the given Boolean values. This allows us to logically do actions based on some criteria.

Much like regular programming languages, Bash supports two types of conditional statements:

1.IF statements

2.CASE statements

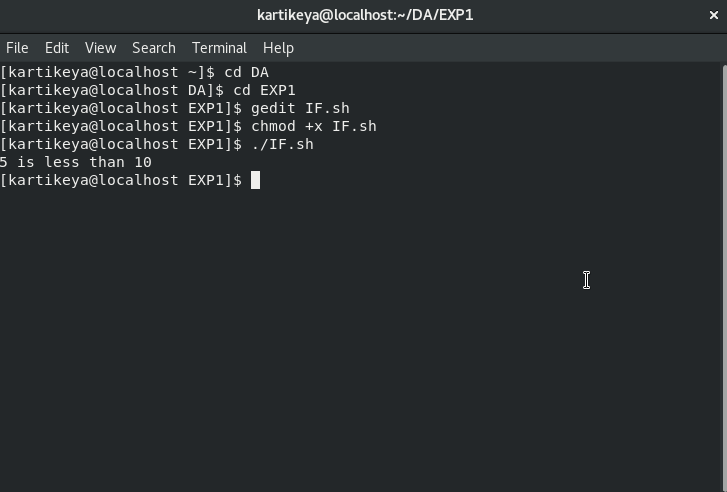
#### 1.IF statement

The IF statement takes up a Boolean value directly and executes the given code if the given condition is true. Expressions that gets evaluated to Boolean can be given as inputs to the IF statements.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | num1=5 |
|  | num2=10 |
|  | if [ $num1 -le $num2 ]; then |
|  | echo "$num1 is less than $num2" |
|  | else |
|  | echo "$num1 is greater than $num2" |
|  | fi |

**Output**

****

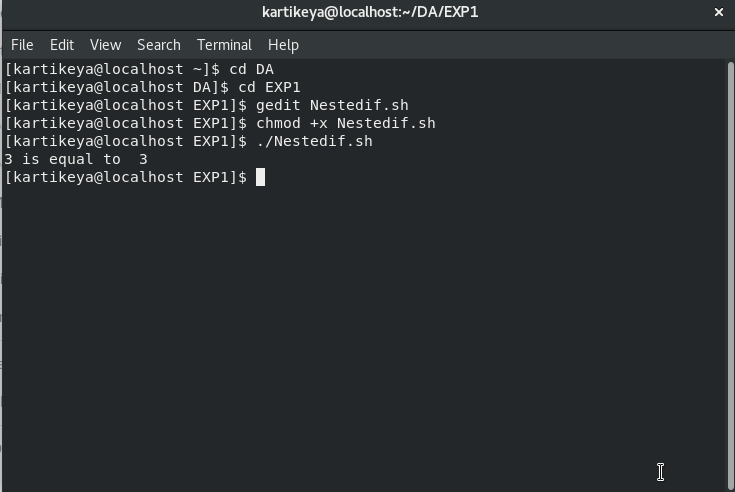
Nested IF

An IF statement can have multiple IF statements in its action and this type is generally called as “Nested IF” statement. If the actions don’t have any commands other than another IF statement, then the evaluation can be shrunk at one place rather than having a Nested IF statement.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | num1=3 |
|  | num2=3 |
|  | if [ $num1 -ne $num2 ]; then |
|  | if [ $num1 -gt $num2]; then |
|  | echo "$num1 is greater than $num2" |
|  | else |
|  | echo "$num1 is lesser than $num2" |
|  | fi |
|  | else |
|  | echo "$num1 is equal to $num2" |
|  | fi |

**Output**

****

Ladder IF

In some scenarios, multiple conditions need to be checked one after another till a condition that resolves to true is met. Instead of having multiple IF statements inside the “else” clause of each parent IF statement, multiple **“elif”** keyword with conditions and code for each can be used.

**Code**

Top of Form

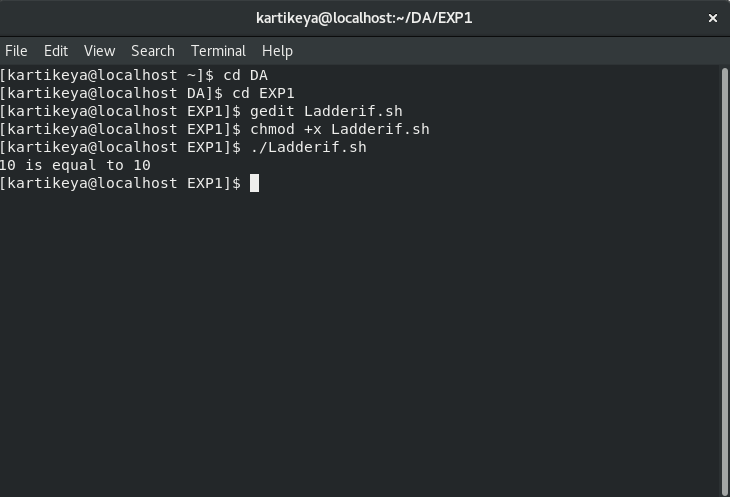
Bottom of Form

Top of Form

Bottom of Form

|  |  |
| --- | --- |
|  | #!/bin/bash |
|  | num1=10 |
|  | num2=10 |
|  | if [ $num1 -gt $num2 ]; then |
|  | echo "$num1 is greater then $num2" |
|  | elif [ $num1 -lt $num2 ]; then |
|  | echo "$num1 is lesser than $num2" |
|  | else |
|  | echo "$num1 is equal to $num2" |
|  | fi |

**Output**

**

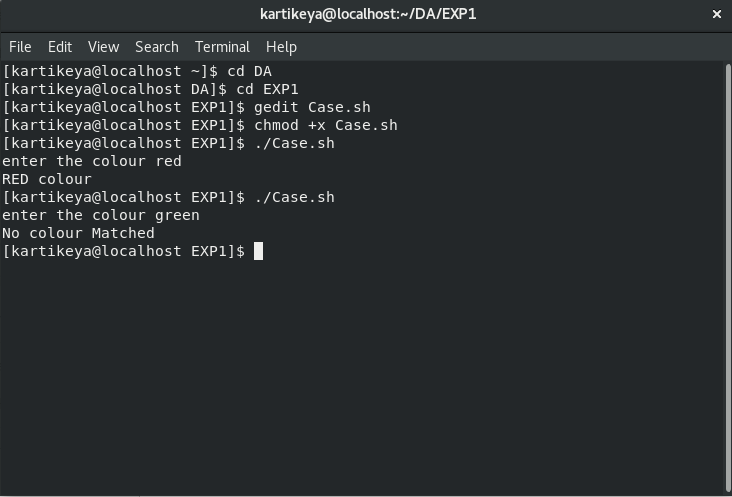
*2.*CASE statement

It is also a conditional statement that allows controlling the execution flow of a program. It is almost equivalent to the ladder IF statement. But the syntax and way of providing conditions are slightly different than the ladder IF statement.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | read -e -p "enter the colour " colour |
|  | case "$colour" in |
|  | red) echo "RED colour";; |
|  | black) echo "Black colour";; |
|  | \*) echo "No colour Matched";; |
|  | Esac |

**Output**

****

### **9.Loops**

### Loops are also a kind of control flow statement that enables the program to iterate a defined piece of code for a particular number of times or till a specified condition gets false. Loops are of two types:

1. For loop
2. While loop

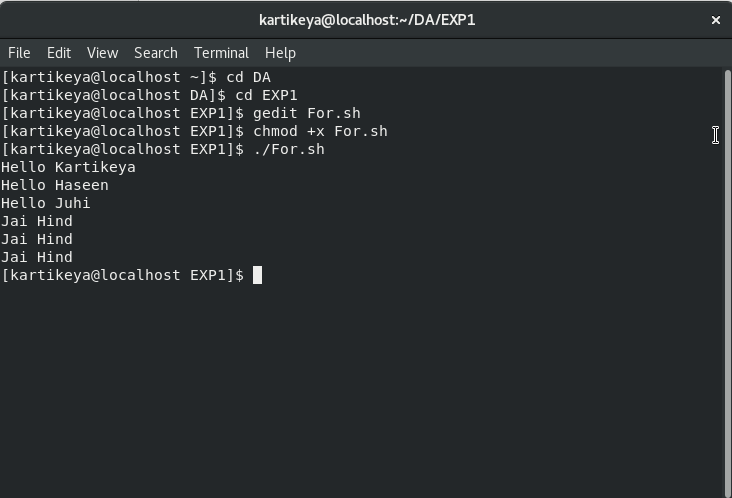
1.For loop

FOR loop is a common type of looping statement used to iterate over a given number of items. Usually, an array of items will be given as input to FOR statement. This statement starts with the first item and switches to the next item in each iteration till it goes through all of them in the given array.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | names="Kartikeya Haseen Juhi" |
|  | for name in $names; do |
|  | echo "Hello $name" |
|  | done |
|  | for number in seq 1 5; do |
|  | echo "Jai Hind" |
|  | done |

**Output**

****

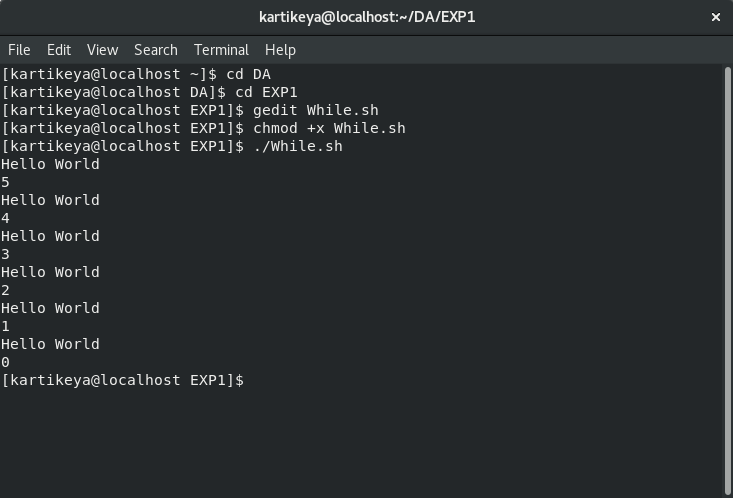
#### 2.While loop

#### WHILE loop is also a statement to do iterations same like FOR loop. The only difference is that WHILE loop does iteration till a given condition becomes false whereas the FOR statement does iteration over the given number of items.

**Code**

|  |
| --- |
| #!/bin/bash |
|  | names="Kartikeya Haseen Juhi" |
|  | for name in $names; do |
|  | echo "Hello $name" |
|  | done |
|  | for number in seq 1 5; do |
|  | echo "Jai Hind" |
|  | Done |

**Output**

****

**Experiment No. 2.2.1.1**

**Aim:**

To Archive, all the matching logs are given in the input and to move the archived files to the specified backup directory.

**Pre-Requisites:**

1.A set of sample log files in the file system.

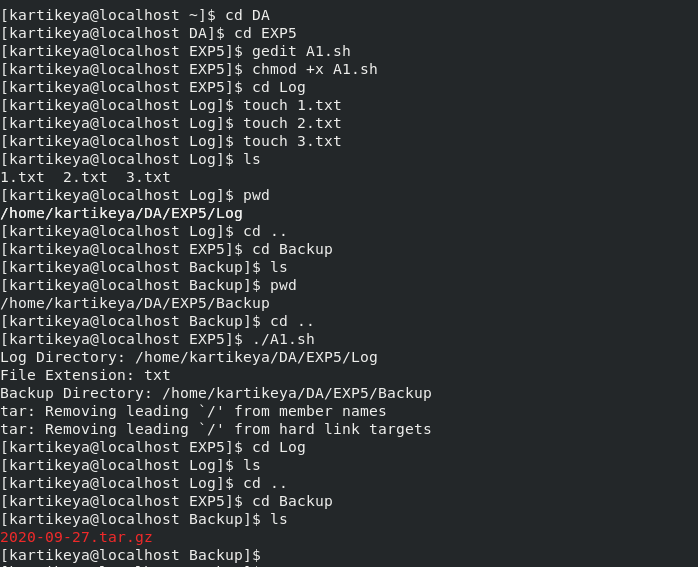
2.“tar” utility to create archives.

**Codes:**

**A1.sh**

|  |
| --- |
| #!/bin/bash |
|  | read -e -p "Log Directory: " log\_directory |
|  | read -e -p "File Extension: " extension |
|  | read -e -p "Backup Directory: " backup\_directory |
|  | tar czf archive.tar.gz $(find $log\_directory -name "\*.$extension") |
|  | mv archive.tar.gz $backup\_directory/$(date +%F).tar.gz |
|  | rm $(find $log\_directory -name "\*.$extension") |
|  | exit 0 |

**Output**

****

**Experiment No. 2.2.1.2**

###### **Aim:**

To automatically delete the archive files that are older than two days. Make use of task schedulers like Cron to execute the script at periodic intervals.

###### **Pre-Requisites:**

1.Archive files of the different timestamp.

2.Permissions to schedule Cron jobs.

**Codes:**

**A2.sh**

#!/bin/bash

if [ -z "$1" ]; then

echo "ERROR: No argument supplied" >&2; exit 1;

fi

archives\_directory= $(realpath $1)

if [ ! -d "$archives\_directory" ]; then

echo "ERROR: Archives directory does not exist" >&2;

exit 1;

fi

find $archives\_directory -type f -name '\*.tar.gz' -mmin +$((1)) - exec rm {}\;

path\_to\_script=$(realpath "$0")

if ! (crontab -l | grep -Fxq "0 0 \* \* \* $path\_to\_script $archives\_directory"); then

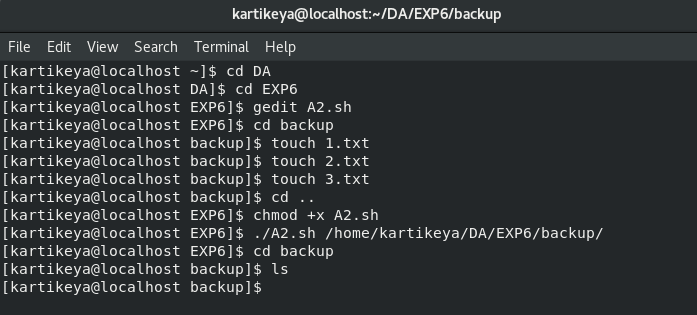
crontab -l | { cat; echo "0 0 \* \* \* $path\_to\_script $archives\_directory";

} | crontab - echo "Script added to Cron"

fi

exit 0

**Output**



**Experiment No. 2.2.1.3**

###### **Aim:**

To take backups of MySQL server in every 12 hours and to move them to the backup directory to keep the backups safe.

###### **Pre-Requisites:**

1.MySQL server with MySQL client in the lab environment.

2.Permissions to schedule Cron jobs.

**Codes:**

**A3.sh**

#!/bin/bash

if [ -z "$1" ]; then

echo "ERROR: Credentials file not specified" >&2; exit 1; elif [ -z "$2" ]; then

echo "ERROR: Backup directory not specified" >&2; exit 1;

fi

credentials\_file=$(realpath $1)

backup\_directory=$(realpath $2)

if [ ! -f "$credentials\_file" ]; then

echo "ERROR: Credentials file does not exist" >&2; exit 1; elif [ ! -d "$backup\_directory" ]; then

echo "ERROR: Backup directory does not exist" >&2; exit 1;

fi

source $credentials\_file

if [ -z ${hostname:+word} ]; then

echo "ERROR: hostname is not set" >&2; exit 1; elif [ -z ${username:+word} ]; then

echo "ERROR: username is not set" >&2; exit 1; elif [ -z ${password:+word} ]; then

echo "ERROR: password is not set" >&2; exit 1;

fi

mysqldump -h$hostname -u$username -p$password --all-databases > backup.sql

if [[ $? != 0 ]]; then

echo "ERROR: Error in taking mysql backup" >&2; exit 1;

fi

mv backup.sql $backup\_directory/$(date +%F\_%R).sql path\_to\_script=$(realpath "$0")

if ! (crontab -l | grep -Fxq "0 \*/12 \* \* \* $path\_to\_script $credentials\_file

$backup\_directory"); then

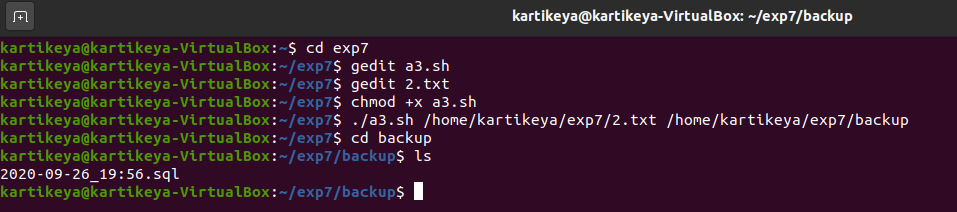
crontab -l | { cat; echo "0 \*/12 \* \* \* $path\_to\_script $credentials\_file

$backup\_directory"; } | crontab - echo "Script added to Cron"

fi

exit 0

**Output**



**Experiment No. 2.2.1.4**

**Aim:**

To email the summary of the web server such as the aggregates of all the HTTP status codes and the IP Addresses with top hits, to the given email ID every day.

###### **Pre-Requisites:**

1.MailUtils should be configured.

2.Permissions to schedule Cron jobs.

3.Apache log files should exist locally on the filesystem.

**Codes:**

**A4.sh**

#! /usr/bin/bash

if [ -z "$1" ]

then

echo "ERROR: Location of the web server's log is not specified" >&2; exit 1;

fi

log\_file=$(realpath $1)

echo "path of log file is $log\_file"

[email\_id=kartikeya532001@gmail.com](mailto:email_id=kartikeya532001@gmail.com)

if [ ! -f "$log\_file" ]

then

echo "ERROE: Log file does not exist" >&2; exit 1;

fi

(

echo -e 'Apache Web Server Access Logs - Summary\n'

echo -e 'STATUS\t\t - \tCOUNT'

cat $log\_file | sed 's/.HTTP\/1\.1" \(...\)./\1/g' | sort | uniq -c | awk '{printf " %s\t\t-\t %s\n", $2, $1}' echo -e 'nIP ADDRESS\t - \tHITS'

cat $log\_file | sed 's/ .\*//g' | sort | uniq -c | awk '{printf "%s\t - \t %s\n", $2, $1}'

) > /tmp/log\_summary

echo "log summary"

cat /tmp/log\_summary

cat /tmp/log\_summary | mail -s "Apache web server access logs - Summary" $email\_id

if [[ $? != 0 ]]

then

echo "ERROR: error in sending mail to $(email)" >&2; exit 1;

fi

path\_to\_script=$(realpath "$0")

if ! (crontab -l | grep -Fxq "0 \* \* \* \* $path\_to\_script $log\_file")

then

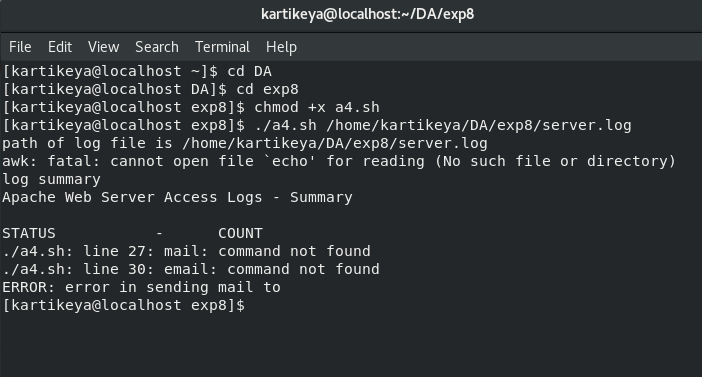
crontab -l { cat; echo "0 \* \* \* \* $path\_to\_script $log\_file"} |

crontab - echo "Script added to cron"

fi

exit 0

**Output**



**Experiment No. 2.2.1.5**

###### **Aim:**

###### To monitor the web server’s status continuously at particular intervals and to restart the webserver if it has stopped running.

###### **Pre-Requisites:**

Apache web server installed and configured.

Permissions to schedule Cron jobs.

**Codes:**

**A5.sh**

#!/bin/bash

if [ -z "$1" ]

then

echo "ERROR: Webserver port is not specified in the arguments" >&2

exit 1;

fi

listening\_port=$1

netstat -lnt | grep -q ":$1 "

if [[ $? != 0 ]]

then

echo "ERROR: Web server is not running";

/etc/init.d/apache2 restart

fi

path\_to\_script=$(realpath "$0")

if ! (crontab -l | grep -Fxq "\*/1 \* \* \* \* $path\_to\_script $listening\_port")

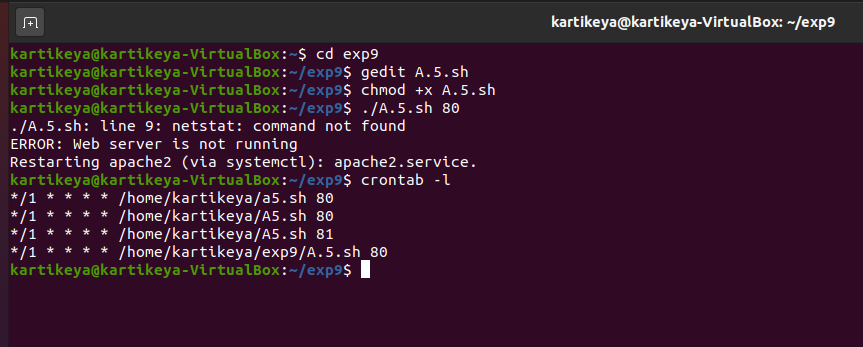
then

crontab -l | { cat; echo "\*/1 \* \* \* \* $path\_to\_script $listening\_port"; } | crontab - echo "Script added to Cron"

fi

exit 0

**Output**



**Experiment No. 2.2.1.6**

###### **Aim:**

To block the users from running forbidden commands by continuous validation of the given commands. Other commands should be executed normally whereas the forbidden commands should throw an error.

###### **Pre-Requisites:**

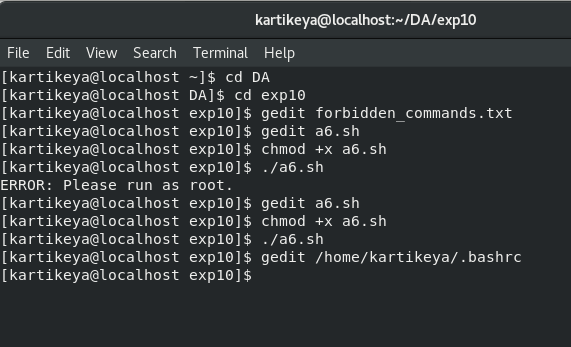
Root permission to the Linux Operating System.

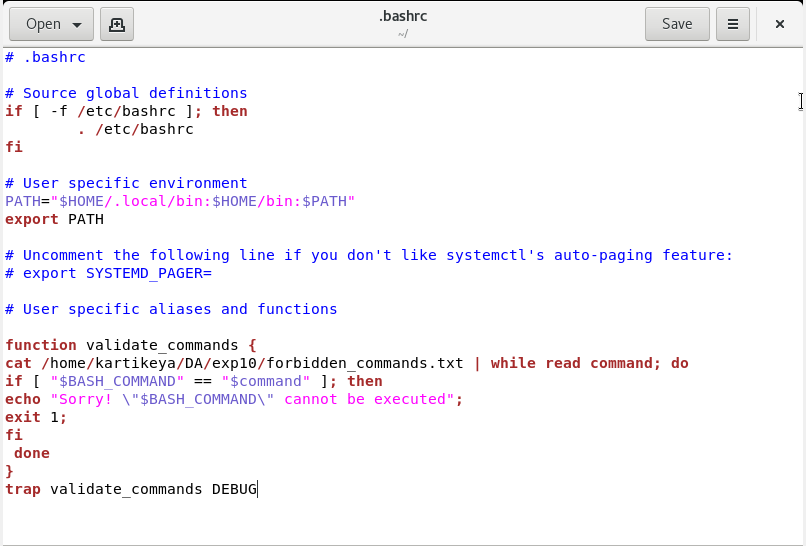
**Codes:**

**A6.sh**

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | #if [[ $(id -u) -ne 0 ]] ; then |
|  | #echo "ERROR: Please run as root."; exit 1; |
|  | #fi |
|  |  |
|  | if [ ! -f "/home/kartikeya/DA/exp10/forbidden\_commands.txt" ]; then |
|  | echo "ERROR: The file /opt/forbidden\_commands.txt does not exist" >&2; |
|  | exit 1; |
|  | Fi |
|  |  |
|  | for user in $(getent passwd | cut -d : -f 6 | grep '/home' | sed 's:$:/.bashrc:'); do |
|  | cat $user | grep -q 'validate\_commands' |
|  | if [[ $? != 0 ]]; then |
|  | echo ' |
|  | function validate\_commands { |
|  | cat /home/kartikeya/DA/exp10/forbidden\_commands.txt | while read command; do |
|  | if [ "$BASH\_COMMAND" == "$command" ]; then |
|  | echo "Sorry! \"$BASH\_COMMAND\" cannot be executed"; |
|  | exit 1; |
|  | Fi |
|  | Done |
|  | } |
|  | trap validate\_commands DEBUG' >> $user |
|  | Fi |
|  | Done |
|  | exit 0 |

**Output**

****

****

**Experiment No. 2.2.1.7**

###### **Aim:**

To monitor the usage of the given disk and alert the user if it is beyond the given threshold.

###### **Pre-Requisites:**

1.MailUtils should be configured.

2.Permissions to schedule Cron jobs.

**Codes:**

**A7.sh**

#!/bin/bash

if [ -z "$1" ]; then

echo "ERROR: Device name is not specified in the arguments" >&2; exit 1; elif [ -z "$2" ]; then

echo "ERROR: Threshold is not specified in the arguments" >&2; exit 1;

fi

device\_name=$1 thresold\_limit=$2

email\_id=kartikeya532001@gmail.com

percentage\_used=$(df -H | grep "$device\_name" | awk '{ print $5 }' | cut -d'%'

-f1)

if [ $percentage\_used -ge $thresold\_limit ]; then

echo "Running out of space \"$device\_name ($percentage\_used%)\" on

$(hostname) as on $(date)" |

mail -s "Disk usage breached the thresold limit" $email\_id

fi

path\_to\_script=$(realpath "$0")

if ! (crontab -l | grep -Fxq "\*/1 \* \* \* \* $path\_to\_script $device\_name

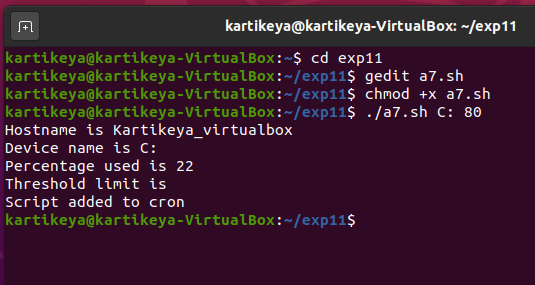
$thresold\_limit"); then

crontab -l | { cat; echo "\*/1 \* \* \* \* $path\_to\_script $device\_name

$thresold\_limit"; } | crontab - echo "Script added to Cron"

fi exit 0

**Output**



**Experiment No. 2.2.1.8**

###### **Aim:**

To move the files/folders to the recycle bin when deleted.

**Codes:**

**A8.sh**

#!/bin/bash

recycle\_bin="$HOME/.recycle\_bin"

rm="/bin/rm -r "

copy="/bin/cp -r "

if [ $# -eq 0 ]

then

echo "ERROR: Please enter the file path to delete." >&2

exit 1;

fi flags=""

while getopts "dfiPRrvW" args

do

case $args in

f ) exec $rm "$@" ;;

\* ) flags="$flags -$args" ;;

esac

done

shift $(( $OPTIND - 1 ))

if [ ! -d $recycle\_bin ]

then

mkdir $recycle\_bin

fi

for arg

do

newname="$recycle\_bin/$(date "+%S.%M.%H.%d.%m").$(basename "$arg")"

if [ -f "$arg" ]

then

$copy "$arg" "$newname"

elif [ -d "$arg" ]

then

$copy "$arg" "$newname"

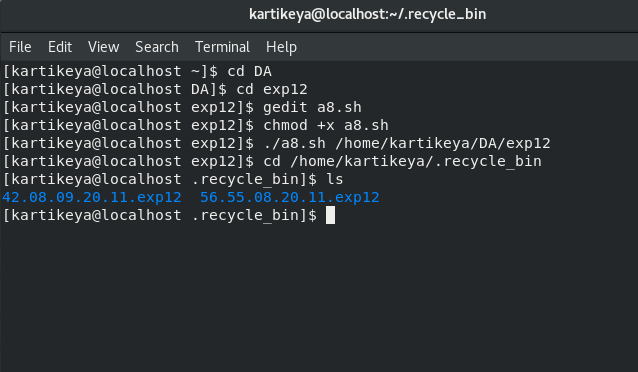
Fi

done

exec $rm $flags "$@"

exit 0

**Output**



**Experiment No. 2.2.1.9**

###### **Aim:**

To restore the deleted files/folders from the recycle bin.

**Codes:**

**A9.sh**

#!/bin/bash

recycle\_bin="$HOME/.recycle\_bin"

rm="/bin/rm"

move="/bin/mv"

destination=$(pwd)

if [ ! -d $recycle\_bin ]

then

echo "ERROR: Recycle Bin was not found in home directory" >&2

exit 1;

fi

cd $recycle\_bin

if [ $# -eq 0 ]

then

echo "Deleted files in the recycle bin:"

ls -FC | sed -e 's/\([[:digit:]][[:digit:]]\.\)\{5\}//g' -e 's/^/ /' exit 0

fi

pattern\_matches="$(ls \*"$1" 2> /dev/null | wc -l)"

if [ $pattern\_matches -eq 0 ]

then

echo "ERROR: No match for the pattern \"$1\"" >&2 exit 1;

fi

if [ $pattern\_matches -gt 1 ]

then

echo "More than one file or directory match in the archive:" index=1

for name in $(ls -td \*"$1")

do

datetime="$(echo $name | cut -c1-14| awk -F. '{ print $5"/"$4" at "$3":"$2":"$1 }')"

if [ -d $name ]

then

size="$(ls $name | wc -l | sed 's/[^0-9]//g')"

echo " $index) $1 (contents = ${size} items, deleted =

$datetime)"

else

size="$(ls -sdk1 $name | awk '{print $1}')"

echo " $index) $1 (size = ${size}Kb, deleted = $datetime)"

fi

index=$(( $index + 1))

done

echo ""

read -e -p "Which version of $1 to restore ('0' to quit)? [1] : " desired

if [ ${desired:=1} -ge $index ]

then

echo "ERROR: Restore cancelled by user: index value too big." >&2

exit 1;

fi

if [ $desired -lt 1 ]

then

echo "ERROR: Restore cancelled by user." >&2

exit 1;

fi

restore="$(ls -td1 \*"$1" | sed -n "${desired}p")"

if [ -e "$destination/$1" ]

then

echo "ERROR: Already exists in this directory. Cannot overwrite." >&2

exit 1;

fi

echo -n "Restoring file \"$1\" ..."

$move "$restore" "$destination/$1" echo "done."

read -e -p "Delete the additional copies of this file? [y] " answer

if [ ${answer:=y} = "y"]

then

$rm -rf \*"$1" echo "deleted."

else

echo "additional copies retained."

fi else

if [ -e "$destination/$1" ]

then

echo "ERROR: Already exists in this directory. Cannot overwrite." >&2

exit 1;

fi

restore="$(ls -d \*"$1")"

echo -n "Restoring file \"$1\" ... "

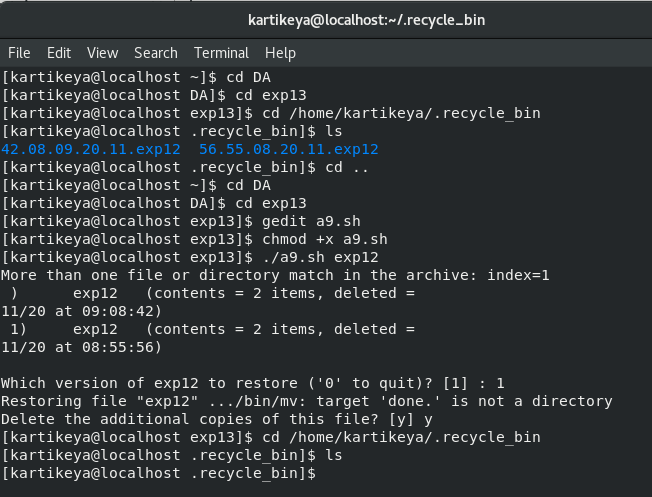
$move "$restore" "$destination/$1"

echo "done."

fi

exit 0

**Output**



**Experiment No. 2.2.1.10**

###### **Aim:**

To Log all the delete operations made through the script.

**Codes:**

**A10.sh**

#!/bin/bash

removelog="/home/kartikeya/DA/exp14/remove.log"

if [ $# -eq 0 ]

then

echo "Usage: $0 [-s] list of files or directories"

exit 0

fi

if [ "$1" = "-s" ]

then

shift

else

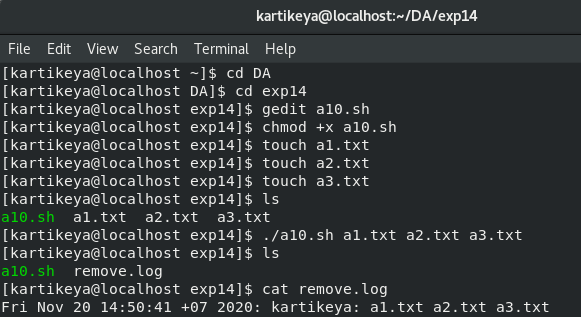
echo "$(date): ${USER}: $@" >> $removelog

fi

/bin/rm "$@"

exit 0

**Output**



**Experiment No. 2.2.1.11**

###### **Aim:**

To download a list of files from the given URL list.

**Codes:**

**A11.sh**

#!/bin/bash

downloads\_directory="$HOME/Downloads"

if [ -z "$1" ]

then

echo "ERROR: Download List file is not specified in the arguments" >&2

exit 1;

fi

if ! [ -z "$2" ]

then

downloads\_directory=$2

fi

if [ ! -d $downloads\_directory ]

then

mkdir -p $downloads\_directory

fi

download\_list=$1

cat $download\_list | while read url

do

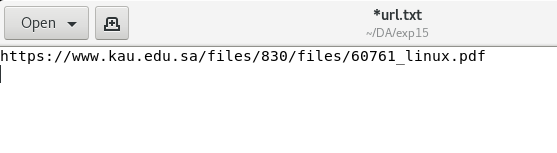
echo " "

echo "$url"

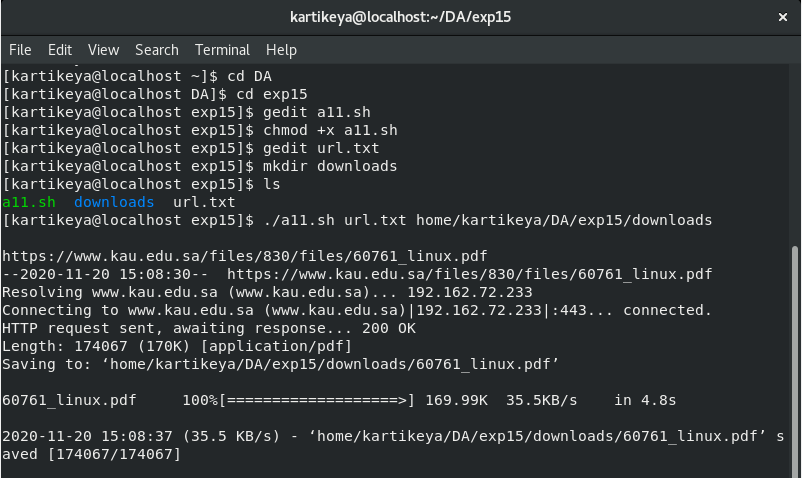
wget -P $downloads\_directory $url

done

exit 0



**Output**



**Experiment No. 2.2.1.12**

###### **Aim:**

To Install Apache, MySQL, PHP on Linux machine (LAMP stack).

###### **Pre-Requisites:**

1.Apt repository configured with valid mirrors

2.Root permission to the Linux Operating System

**Codes:**

**A12.sh**

#!/bin/bash

echo -e "\n\nUpdating Apt Packages and upgrading latest patches\n" sudo apt-get update -y && sudo apt-get upgrade -y

echo -e "\n\nInstalling Apache2 Web server\n"

sudo apt-get install apache2 apache2-doc apache2-mpm-prefork apache2-utils libexpat1 ssl-cert -y

echo -e "\n\nInstalling PHP & Requirements\n"

sudo apt-get install libapache2-mod-php7.0 php7.0 php7.0-common php7.0-curl php7.0-dev php7.0-gd php-pear php7.0-mcrypt php7.0-mysql -y

echo -e "\n\nInstalling MySQL\n"

sudo apt-get install mysql-server mysql-client -y

echo -e "\n\nPermissions for /var/www\n" sudo chown -R www-data:www-data /var/www echo -e "\n\n Permissions have been set\n"

echo -e "\n\nEnabling Modules\n" sudo a2enmod rewrite

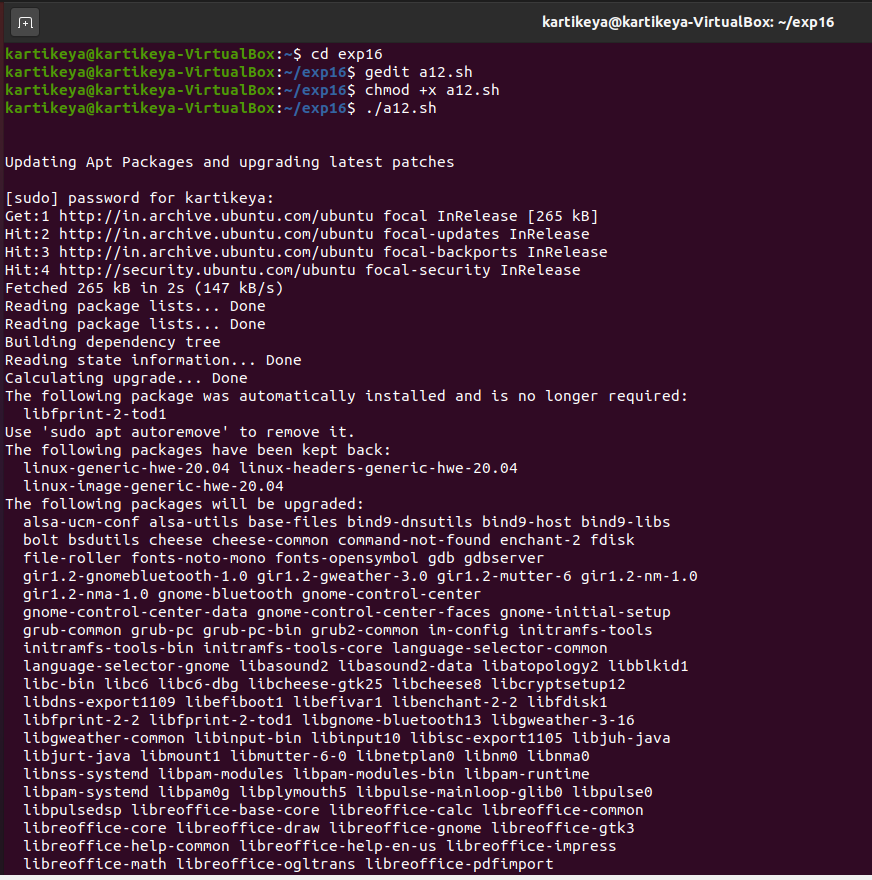
sudo phpenmod mcrypt

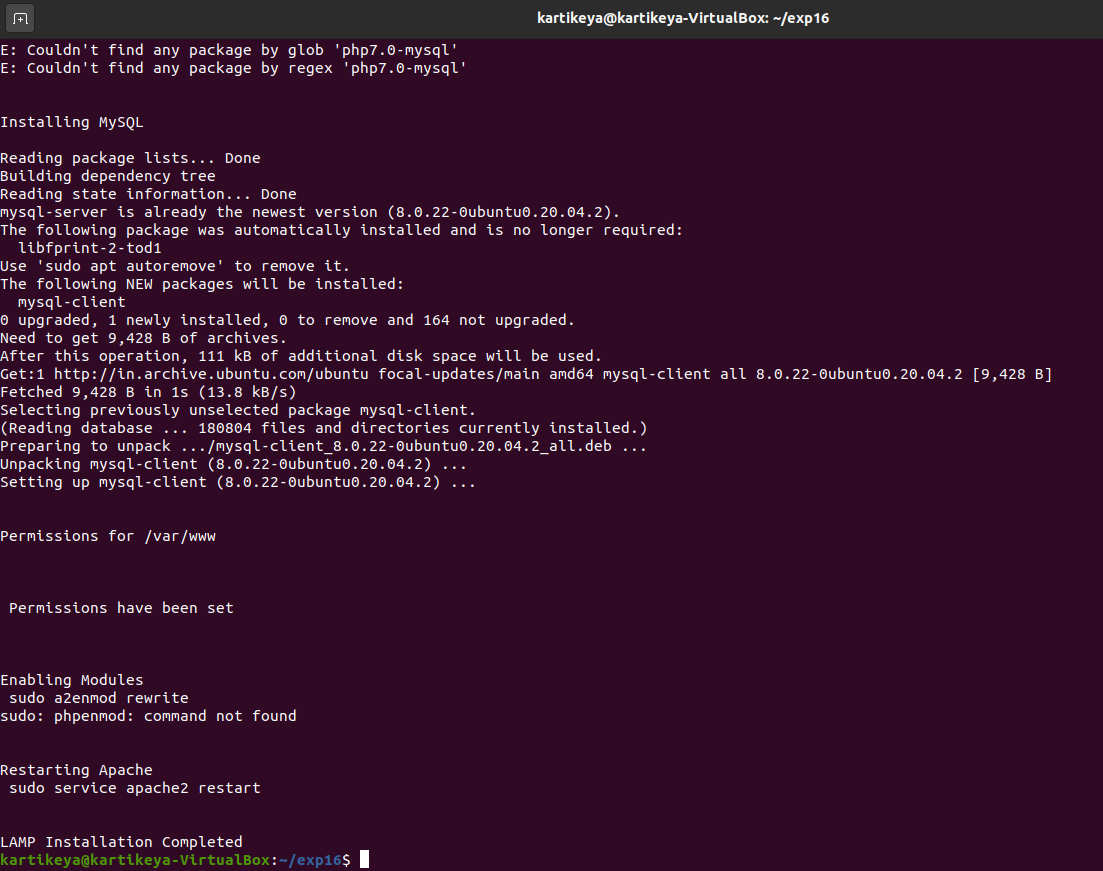
echo -e "\n\nRestarting Apache\n" sudo service apache2 restart

echo -e "\n\nLAMP Installation Completed"

exit 0

**Output**

****

****

**Experiment No. 2.2.2.1**

###### **Aim:**

Find the commands executed by all users that match the given pattern.

###### **Pre-Requisites:**

1.Root permission to the Linux operating system.

**Codes:**

**B1.sh**

#!/bin/bash

if [[ $(id -u) -ne 0 ]] ; then

echo "ERROR: Please run as root." ; exit 1 ;

fi

if [ -z "$1" ]; then

echo "ERROR: Search pattern not specified" >&2; exit 1;

fi

pattern=$1

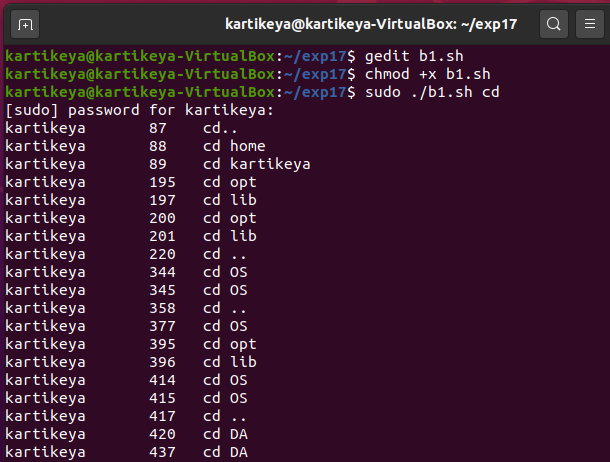
getent passwd | cut -d : -f 6 | sed 's:$:/.bash\_history:' | xargs -d '\n' grep

-s -n -e "$pattern" > /tmp/search

cat /tmp/search | sed 's/.\/\(.\)\/\.bash\_history\:\(.\*\)\:/\1 \t \2 \t/g'

exit 0

**Output**

****

**Experiment No. 2.2.2.2**

###### **Aim:**

To Search and Replace a given text with a new text across multiple files.

**Codes:**

**B2.sh**

#!/bin/bash

if [ "$#" -eq 0 ]

then

echo "ERROR: Files/Folders to search & replace is not specified" >&2

exit 1;

fi

item\_paths=""

for i in $@

do

item\_path=$(realpath $i)

if [ -f "$item\_path" ]

then

item\_paths+="$item\_path "

elif [ -d "$item\_path" ]

then

item\_paths+="$item\_path/\* "

else

echo "ERROR: The given path \"$item\_path\" is not a suitable type" >&2

exit 1;

fi

done

read -e -p "Search For: " search\_text

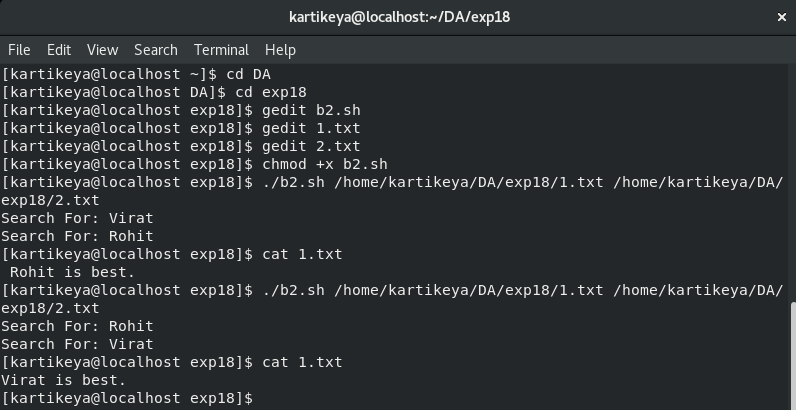
read -e -p "Replace With: " replace\_text

sed -i "s/$search\_text/$replace\_text/g"

$(grep -lz "$search\_text" $item\_paths)

exit 0

**Output**



**Experiment No. 2.2.2.3**

**Aim:**

To check whether the given input is a valid AlphaNumeric Character.

**Codes:**

**B3.sh**

#!/bin/bash

is\_valid\_alpha\_numeric\_text()

{

filtered\_input="$(echo $1 | sed -e 's/[^[:alnum:]]//g')"

if [ "$filtered\_input" == "$1" ]

then

return 0

else

return 1

fi

}

read -e -p "Enter Input to validate: " input\_text

if [[ "$input\_text" == '' ]]

then

echo "ERROR: Please enter any text to validate." >&2

exit 1;

fi

if ! is\_valid\_alpha\_numeric\_text "$input\_text"

then

echo "Input contains characters other than alphabets and numbers"

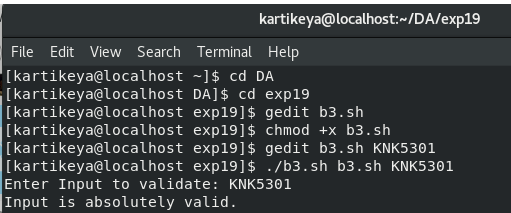
else

echo "Input is absolutely valid."

Fi

exit 0

**Output**



**Experiment No. 2.2.2.4**

###### **Aim:**

To truncate the given file to the specified size.

**Codes:**

**B4.sh**

#!/bin/bash

if [ "$#" != "2" ]

then

( echo "

Usage: `basename $0` <size> <path>

Truncate the <path> to exactly <size> bytes.

\* If <path> doesn't exist it is created.

\* <size> is a number which can be optionally followed by K, M, G, etc." ) >&2

exit 1;

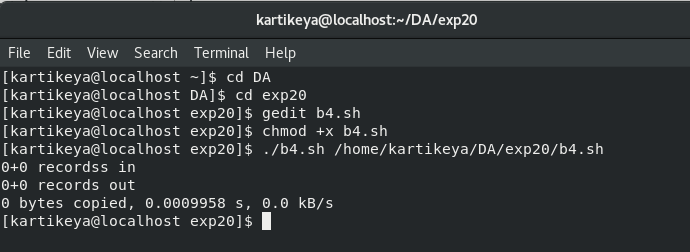
fi

size=$1 file="$2"

dd bs=1 seek=$size if=/dev/null of="$file"

exit 0

**Output**



**Experiment No. 2.2.2.5**

###### **Aim:**

To colourize the standard output by the traditional diff tool.

**Codes:**

**B5.sh**

#!/bin/bash

if [ "$#" -ne "2" ]

then

echo "ERROR: Two files/directories must be passed as arguments" >&2

exit 1;

fi

removed=$(echo -en '\033[31m')

added=$(echo -en '\033[32m')

changed=$(echo -en '\033[34m')

file=$(echo -en '\033[47m')

reset=$(echo -en '\033[0m')

diff -Naru "$@" | sed " s/^\\{3\}.\\*\{4\}/$changed&$reset/;t

s/^-\{3\}.\*-\{4\}/$changed&$reset/;t

s/^@./$changed&$reset/;t

s/^[0-9]./$changed&$reset/;t

s/^!.\*/$changed&$reset/;t

s/^-./$removed&$reset/;t

s/^<./$removed&$reset/;t

s/^\./$added&$reset/;t

s/^\+./$added&$reset/;t

s/^>./$added&$reset/;t

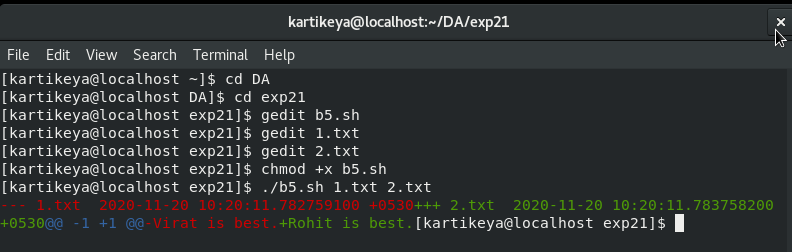
s/^Only in./$file&$reset/;t

s/^Index: ./$file&$reset/;t

s/^diff .\*/$file&$reset/;t

"exit 0

**Output**



**Experiment No. 2.2.2.6**

###### **Aim:**

To transform the given input text to lowercase, uppercase and first letter uppercase text.

**Codes:**

**B6.sh**

#!/bin/bash

first\_letter\_uppercase()

{

echo $1 | awk '{print toupper(substr($0,0,1))tolower(substr($0,2))}'

}

everything\_uppercase()

{

echo $1 | awk '{print toupper($0)}'

}

everything\_lowercase()

{

echo $1 | awk '{print tolower($0)}'

}

if [ -z "$1" ]

then

echo "ERROR: Input text is not supplied in the arguments" >&2

exit 1;

fi

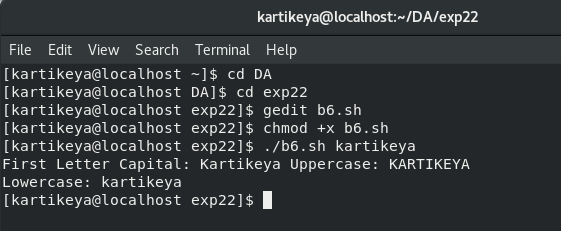
input=$1

echo "First Letter Capital: $(first\_letter\_uppercase "$input") Uppercase: $(everything\_uppercase "$input")

Lowercase: $(everything\_lowercase "$input")"

exit 0

**Output**



**Experiment No. 2.4.5.1**

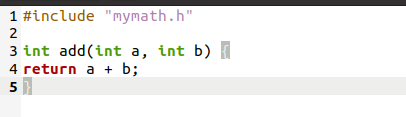
###### **Aim:**

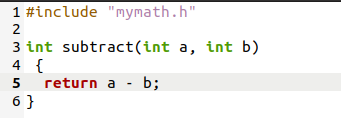
To create the C source code for building them with Make.

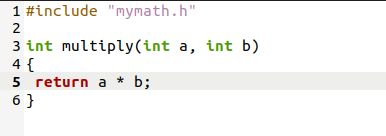
###### **Pre-Requisites:**

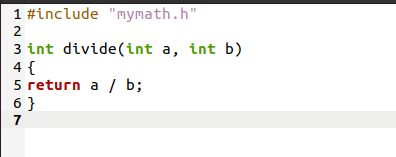
GCC compiler.

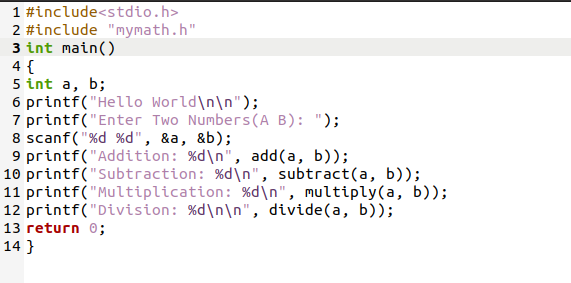
**Codes:**



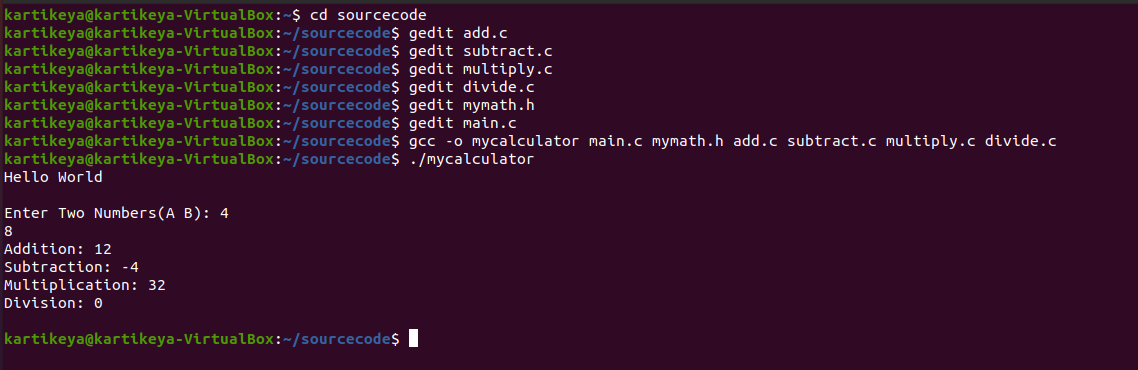








**Output**



**Experiment No. 2.4.5.2**

###### **Aim:**

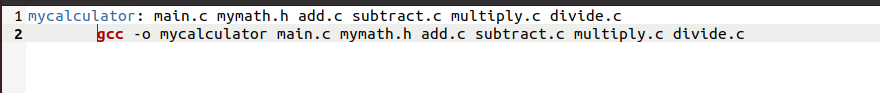
To build the binary file from the C source code.

###### **Pre-Requisites:**

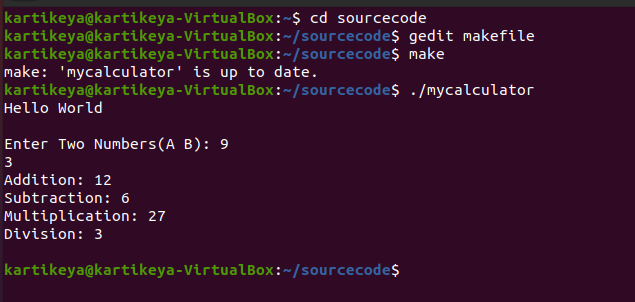
1.GNU Make utility

2.C compiler

**Codes:**



**Output**



**Experiment No. 2.4.5.3**

###### **Aim:**

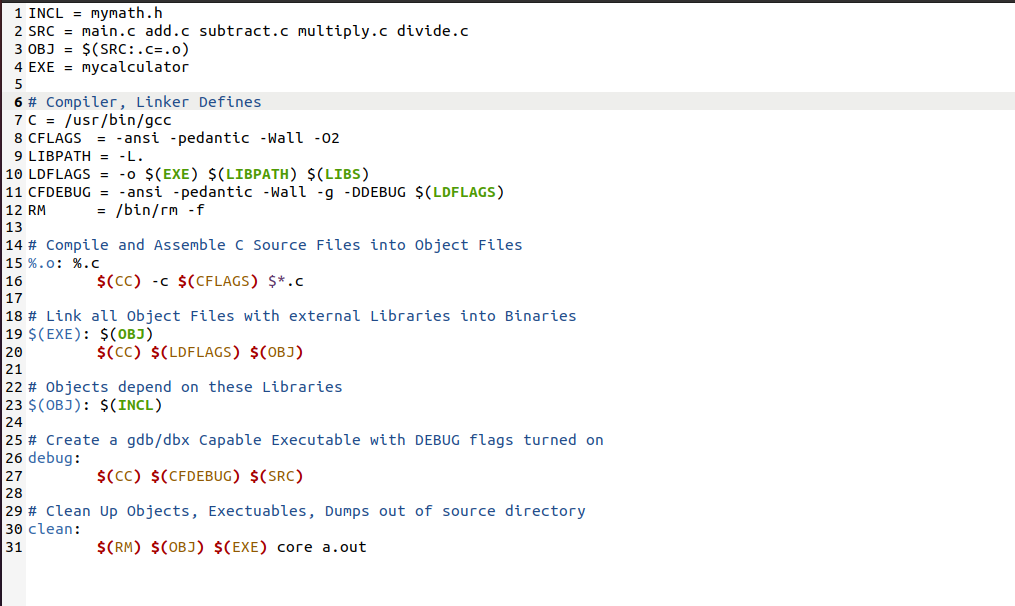
###### To transform the Makefile from the previous lab to implement the best practices.

###### **Pre-Requisites:**

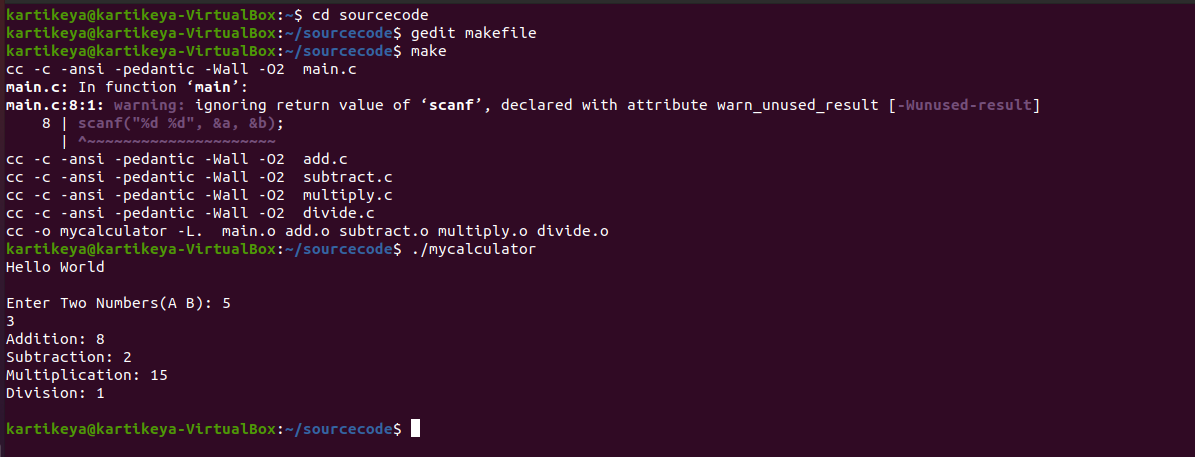
1.GNU Make utility

2.C compiler

**Codes:**



**Output**



**Experiment No. 2.4.5.4**

###### Aim:

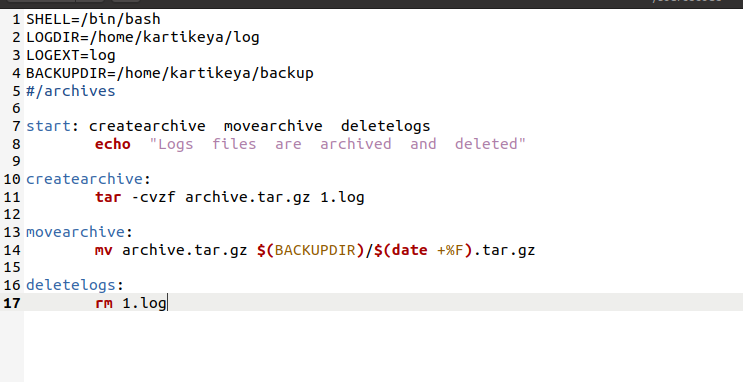
To archive the log files and to move them to separate directory.

###### Pre-Requisites:

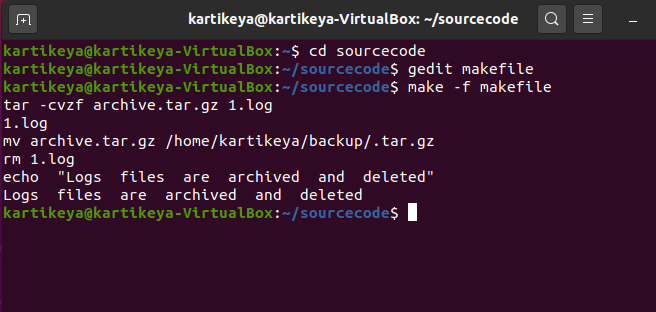
1.GNU Make utility

2.Tar utility

**Codes:**



**Output**



**Experiment No. 2.4.5.5**

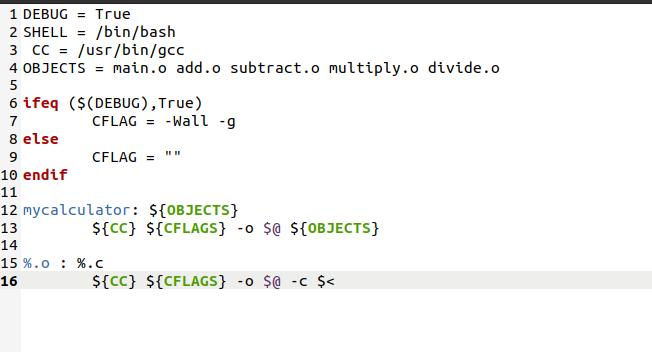
###### **Aim:**

To use the conditional in Makefile to enable/disable the debug mode based on the value of a macro.

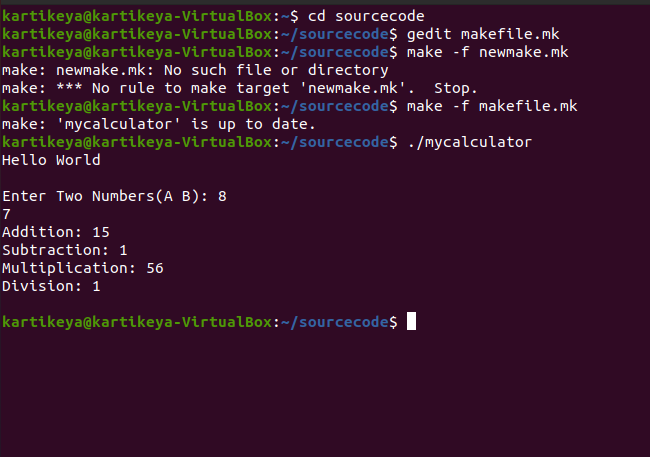
###### **Pre-Requisites:**

1.GNU Make utility

**Codes:**

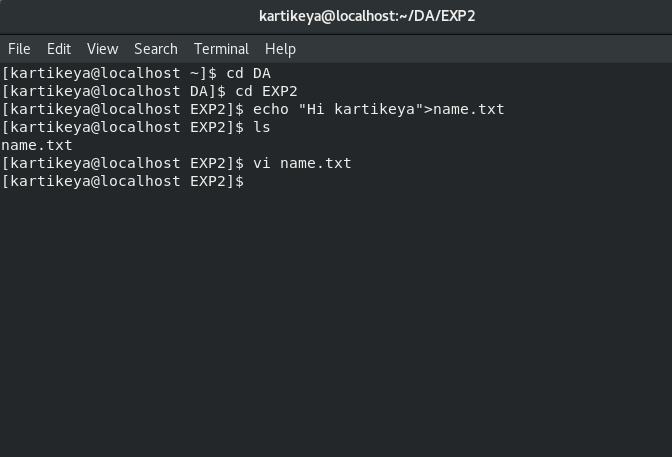


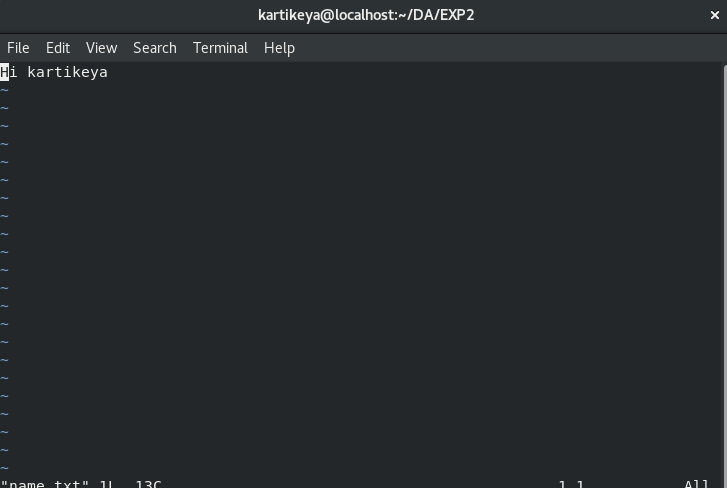
**Output**



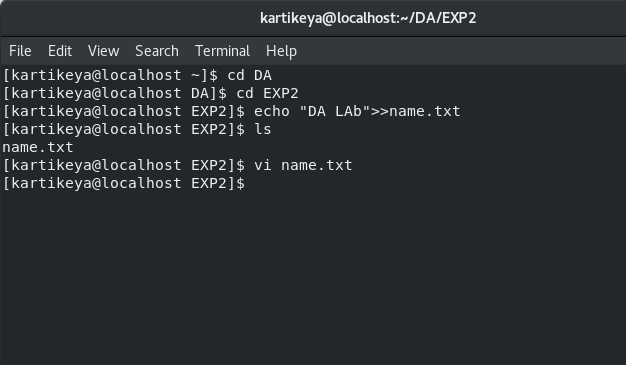
**Experiment No.2.5**

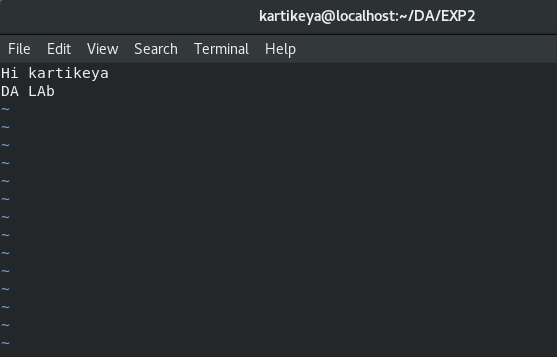
**1.Standard Input**

****

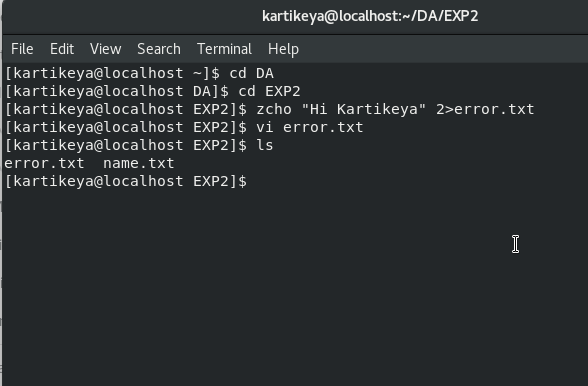
****

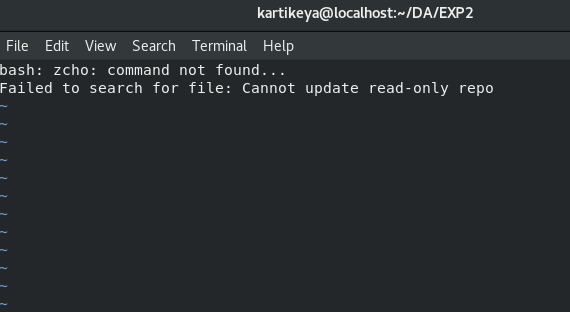
**2.Append**

****

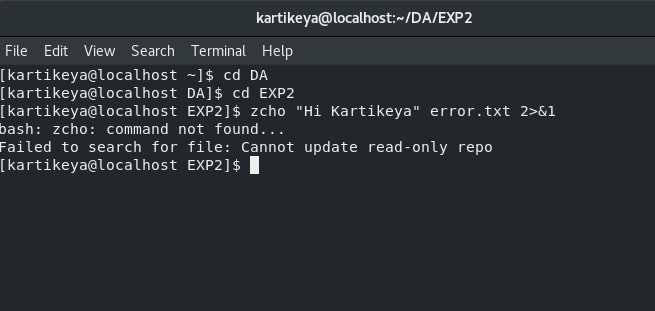
****

**3.Errors**

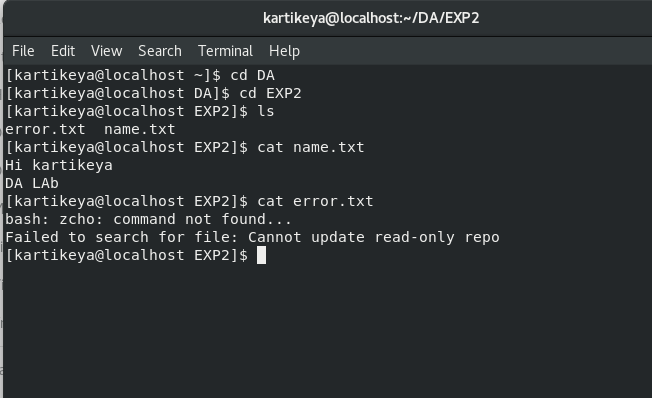
****

****

**4.Error Redirection**

****

**5.Cat Command**

****

**Experiment No. 2.6.1.1**

###### **Aim:**

To write a script that provides a set of functions to format the terminal as listed below:

1.Colorize the given text

2.Show progress bar that fills over a specified time

3.Transform the case of the given text

**Codes:**

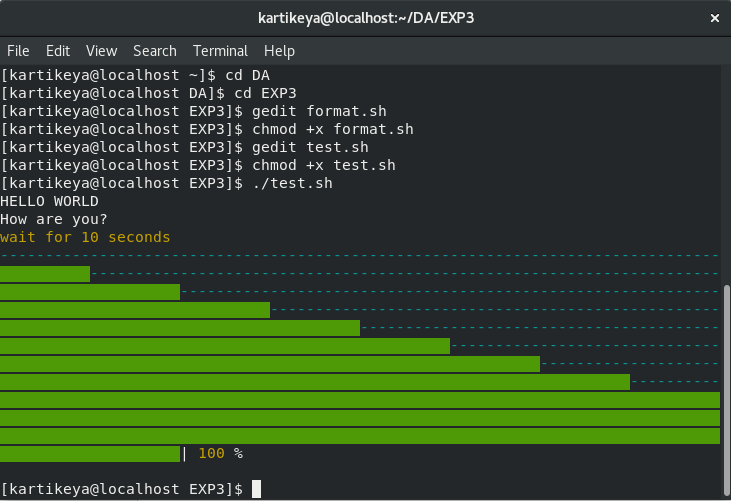
**1.Format.sh**

|  |
| --- |
| #!/bin/bash |
|  | style() |
|  | { |
|  | echo "\033[$1m" |
|  | } |
|  | red\_text() |
|  | { |
|  | printf "$(style "31")$1$(style "0")"; |
|  | } |
|  | green\_text() |
|  | { |
|  | printf "$(style "32")$1$(style "0")"; |
|  | } |
|  | yellow\_text() |
|  | { |
|  | printf "$(style "33")$1$(style "0")"; |
|  | } |
|  | blue\_text() |
|  | { |
|  | printf "$(style "34")$1$(style "0")"; |
|  | } |
|  | cyan\_text() |
|  | { |
|  | printf "$(style "36")$1$(style "0")"; |
|  | } |
|  | progress\_bar() |
|  | { |
|  | if [ -z "$1" ] ; then |
|  | echo "ERROR: The duration of the progress bar is required" >&2; exit 1; |
|  | Fi |
|  | local duration=$1 |
|  | local increment=$((100/$duration)) |
|  | for (( elapsed=0; elapsed<=100; elapsed=elapsed+increment )); do for ((done=0; done<elapsed; done=done+1)); do |
|  | printf "$(green\_text "▇")"; done |
|  | for ((remain=elapsed; remain<100; remain=remain+1)); do printf "$(cyan\_text "-")"; |
|  | Done |
|  | printf "| $(yellow\_text "$elapsed") %%" ; sleep 1 |
|  | printf "\r"; done |
|  | echo -e "\n"; |
|  | } |
|  |  |
|  | camelcase() |
|  | { |
|  | echo $1 | awk '{print toupper(substr($0,0,1))tolower(substr($0,2))}' |
|  | } |
|  |  |
|  | uppercase() |
|  | { |
|  | echo $1 | awk '{print toupper($0)}' |
|  | } |
|  | lowercase() |
|  | { |
|  | echo $1 | awk '{print tolower($0)}' |
|  | } |

**2.Test.sh**

|  |
| --- |
| #!/bin/bash |
|  | . /home/kartikeya/DA/EXP3/format.sh |
|  | uppercase "hello world" |
|  | camelcase "how are you?" |
|  | yellow\_text "wait for 10 seconds" |
|  | echo " " |
|  | progress\_bar 10 |

**Output**



**Experiment No. 2.6.1.2**

###### **Aim:**

To write a script that provides a list of utilities to reuse across the scripts. The utilities are listed below:

1.Check whether the given user exists

2.Check whether the root permission is given

3.Exit the function with an error message

4.Check whether a given path is valid and is a file

**Codes:**

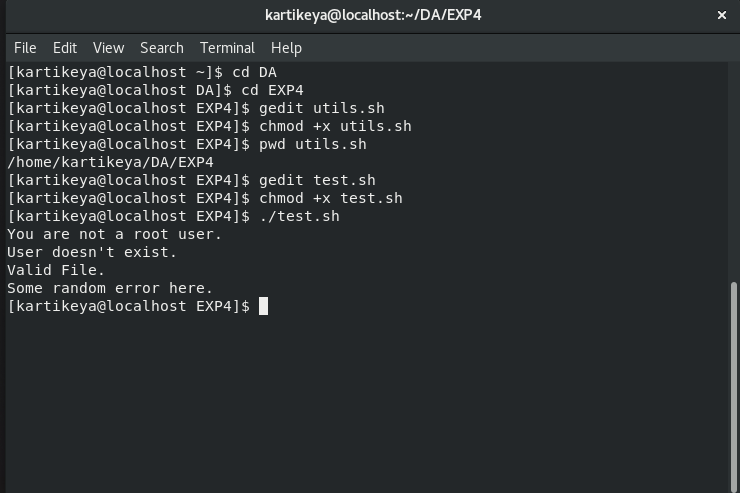
**1.Utils.sh**

|  |
| --- |
| #!/bin/bash |
|  | declare -r TRUE=0 |
|  | declare -r FALSE=1 |
|  | die() |
|  | { |
|  | echo "$1">&2; |
|  | exit 1 |
|  | } |
|  | has\_root\_permission() |
|  | { |
|  | [ $(id -u) -eq 0 ] && return $TRUE || return $FALSE |
|  | } |
|  | does\_user\_exit() |
|  | { |
|  | local username="$1" |
|  | grep -q "^${username}" /etc/passwd && return $TRUE || return $FALSE |
|  | } |
|  | is\_valid\_file() |
|  | { |
|  | local given\_path="$1" |
|  | [ -f "$(realpath $given\_path)" ] && return $TRUE || return $FALSE |
|  | } |

**2.Test.sh**

|  |
| --- |
| #!/bin/bash |
|  | . /home/kartikeya/DA/EXP4/utils.sh |
|  | has\_root\_permission && echo "You are a root user." || echo "You are not a root user." |
|  | does\_user\_exit "Redhat" && echo "User exist." || echo "User doesn't exist." |
|  | is\_valid\_file "/etc/passwd" && echo "Valid File." || echo "Invalid File." |
|  | die "Some random error here." |

**Output**

****