LAB MANUAL

Lab Name	Linux Shell Programming Lab
Lab Code	4CS4 – 24
Branch	Computer Science & Engineering
Year/Sem.	2nd Year/IV Sem.

Jaipur Engineering College and Research Center, Jaipur

Department of Computer Science & Engineering

(Rajasthan Technical University, KOTA)

JEERC JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE	**Jaipur Engineering College and Research Centre, Jaipur** **Department of Computer Science & Engineering**

INDEX

S.NO	CONTENTS	PAGE NO.
1	VISION/MISION	4
2.	PEO	4
3.	POS	5
4.	MAPPING OF PO & PEO	6
5.	COS	6
6.	MAPPING OF CO & PO	6
7.	SYLLABUS	7
8.	BOOKS	8
9.	INSTRUCTIONAL METHODS	8
10.	LEARNING MATERIALS	8
11.	ASSESSMENT OF OUTCOMES	8
	LIST OF EXPERIMENTS (RTU SYLLABUS)	
Exp:- 1	**EXPERIMENT-1 :**Use of Basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit.	16
Exp:- 2	**EXPERIMENT-2 :** Commands related to inode, I/O redirection and piping, process control commands, mails.	24
Exp:-3	**EXPERIMENT-3 :** Shell Programming: Shell script exercises based on following 3.1 Greatest among three numbers 3.2 To find a year is leap year or not 3.3 To input angles of triangle and finds out whether it is valid triangle or not	28

	3.4 To check whether a character is alphabet, digit or special character. 3.5 To calculate profit and loss	
Exp:-4	**EXPERIMENT-4:**Shell Programming : Looping , while, until, for loops 4.1 write a shell script to print all even and odd number from number from 1 to 10. 4.2 write a shell script to print table of given number. 4.3 write a shell script to calculate factorial of a given number. 4.4 write a shell script to print sum of all even numbers from 1 to 10. 4.5 write a shell script t print sum of digits of any number.	**34**
Exp:-5	**EXPERIMENT-5 :** Shell Programming - case structure, use of break 5.1 Write a shell script to make a basic calculator which performs addition, subtraction, multiplication , division 5.2 Write a shell script to print days of a week. 5.3 Write a shell script to print starting 4 months having 31 days	**37**
Exp:-6	**EXPERIMENT-6:**Shell programming- Functions 6.1 write a shell script to find a number is Armstrong or not. 6.2 write a shell script to find a number is palindrome or not. 6.3 write a shell script to print Fibonacci series. 6.4 write a shell script to find prime number 6.5 Write shell script to convert binary to decimal and decimal to binary.	**38**
Exp:-7	**EXPERIMENT-7:**Write a shell script to print different shapes- Diamond, Triangle, square, rectangle, hollow square etc.	**39**
Exp:-8	**EXPERIMENT-8:**Shell Programming – Arrays 8.1 Write a C program to read and print elements of array. 8.2 write a C program to find reverse of an array. 8.3 Write a C program to find sum of all array elements. 8.4 Write a C program to search an element in an array. 8.5 Write a C program to sort array elements in ascending or desending order.	**40**

VISION & MISSION

VISION: To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

MISSION:

M1: To impart outcome based education for emerging technologies in the field of computer science and engineering.

M2: To provide opportunities for interaction between academia and industry.

M3: To provide platform for lifelong learning by accepting the change in technologies

M4: To develop aptitude of fulfilling social responsibilities.

PEO

1. To provide students with the fundamentals of Engineering Sciences with more emphasis in **Computer Science &Engineering** by way of analyzing and exploiting engineering challenges.
2. To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.
3. To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.
4. To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self motivated life-long learning needed for a successful professional career.
5. To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge

PROGRAM OUTCOMES

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and Computer Science & Engineering specialization to the solution of complex Computer Science & Engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex computer Science & Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex Computer Science& Engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of Computer Science & Engineering experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern Computer Science& Engineering and IT tools including prediction and modeling to complex computer science engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Computer Science & Engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional Computer Science & Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the Computer Science & Engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in Computer Science & Engineering.
10. **Communication**: Communicate effectively on complex Computer Science & Engineering activities with the engineering community and with society at large, such as, being able to

comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the Computer Science & Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of Computer Science& Engineering change.

MAPPING OF PEOs & POs

PROGRAM OBJECTIVES	PROGRAM OUTCOMES											
	1	2	3	4	5	6	7	8	9	10	11	12
I	3	1										3
II	2	3	2	3		3					1	3
III	1	3	2	3	1						2	
IV				1	2		3	2	3		2	
V									2	2		

3- High

2- Medium

1- Low

COURSE OUTCOMES

Graduates would be able:

CO1: To apply basic commands of Linux and Commands related ti inode, I/O redirection and piping, process control and mails.

CO2: To analyze variety of problems of shell script using looping, case structures in shell script programming.

MAPPING OF CO & PO

COURSE OUTCOMES	PROGRAM OUTCOMES											
	1	2	3	4	5	6	7	8	9	10	11	12
I	3	2	2	2	2	1	1	1	1	2	2	3
II	3	3	3	2	2	1	1	1	1	1	2	3

3- High

2- Medium

1-Low

SYLLABUS

Objectives:

At the end of the semester, the students should have clearly understood and implemented the following:

4CS4-24: Linux Shell Programming Lab

Credit: 1 **Max. Marks: 50(IA:30, ETE:20)**

0L+0T+2P

List of Experiments:

1. Use of Basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit.
2. Commands related to inode, I/O redirection and piping, process control commands, mails.
3. Shell Programming: Shell script based on control structure- **If-then-fi, if-then-else-if, nested if-else, to find:**
   - 3.1 Greatest among three numbers.
   - 3.2 To find a year is leap year or not.
   - 3.3 To input angles of a triangle and find out whether it is valid triangle or not.
   - 3.4 To check whether a character is alphabet, digit or special character.
   - 3.5 To calculate profit or loss.
4. Shell Programming - Looping- while, until, for loops
   - 4.1 Write a shell script to print all even and odd number from 1 to 10.
   - 4.2 Write a shell script to print table of a given number
   - 4.3 Write a shell script to calculate factorial of a given number.
   - 4.4 Write a shell script to print sum of all even numbers from 1 to 10.
   - 4.5 Write a shell script to print sum of digit of any number.
5. Shell Programming - case structure, use of break
   - 5.1 Write a shell script to make a basic calculator which performs addition, subtraction, Multiplication, division
   - 5.2 Write a shell script to print days of a week.
   - 5.3 Write a shell script to print starting 4 months having 31 days.
6. Shell Programming - Functions
   - 6.1 Write a shell script to find a number is Armstrong or not.
   - 6.2 Write a shell script to find a number is palindrome or not.
   - 6.3 Write a shell script to print Fibonacci series.
   - 6.4 Write a shell script to find prime number.
   - 6.5 Write a shell script to convert binary to decimal and decimal to binary
7. Write a shell script to print different shapes- Diamond, triangle, square, rectangle, hollow square etc.
8. Shell Programming – Arrays
   - 8.1 Write a C program to read and print elements of array.
   - 8.2 Write a C program to find sum of all array elements.
   - 8.3 Write a C program to find reverse of an array.
   - 8.4 Write a C program to search an element in an array.
   - 8.5 Write a C program to sort array elements in ascending or descending order.

Reference

8. BOOKS:-

- UNIX Shell programming, By Stephen G. Kochan, Patrick H. Wood
- Behrouz A. Forouzan, Richard F. Gilberg, UNIX and Shell Programming, Thomson, 2003.
- Brian W. Kernighan, Rob Pike, The UNIX Programming Environment, PHI, 1996
- K. Srirengan, Understanding UNIX, PHI, 2002
- Sumitabha Das, Your UNIX- The Ultimate Guide, TMGH, 2002
- Sumitabha Das, UNIX Concepts and Applications, Second Edition, TMGH, 2002

INSTRUCTIONAL METHODS:-

- **Direct Instructions:**
  Through Projectors & White Board with Marker
- **Interactive Instruction:**
  Programs
- **Indirect Instructions:**
  Problem solving

LEARNING MATERIALS:-

- Text/Lab Manual

ASSESSMENT OF OUTCOMES:-

- End term Practical exam (Conducted by RTU, KOTA)
- Daily Lab interaction.

OUTCOMES WILL BE ACHIEVED THROUGH FOLLOWING:-

- Lab Teaching (through Projectors and White Board).
- Online Labs through google meet etc.
- Discussion on Programs.

JEERC JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE	**Jaipur Engineering College and Research Centre, Jaipur** **Department of Computer Science & Engineering**

INSTRUCTIONS OF LAB

DO's

1. Please switch off the Mobile/Cell phone before entering Lab.
2. Enter the Lab with complete source code and data.
3. Check whether all peripheral are available at your desktop before proceeding for program.
4. Intimate the lab In-charge whenever you are incompatible in using the system or in case software get corrupted/ infected by virus.
5. Arrange all the peripheral and seats before leaving the lab.
6. Properly shutdown the system before leaving the lab.
7. Keep the bag outside in the racks.
8. Enter the lab on time and leave at proper time.
9. Maintain the decorum of the lab.
10. Utilize lab hours in the corresponding experiment.
11. Get your CD / Pen drive checked by lab In-charge before using it in the lab.

DON'TS

1. No one is allowed to bring storage devices like Pan Drive /Floppy etc. in the lab.
2. Don't mishandle the system.
3. Don't leave the system on standing for long
4. Don't bring any external material in the lab.
5. Don't make noise in the lab.
6. Don't bring the mobile in the lab. If extremely necessary then keep ringers off.
7. Don't enter in the lab without permission of lab Incharge.
8. Don't litter in the lab.
9. Don't delete or make any modification in system files.
10. Don't carry any lab equipments outside the lab.

We need your full support and cooperation for smooth functioning of the lab.

INSTRUCTIONS FOR STUDENT

BEFORE ENTERING IN THE LAB

- All the students are supposed to prepare the theory regarding the next program.
- Students are supposed to bring the practical file and the lab copy.
- Previous programs should be written in the practical file.
- Any student not following these instructions will be denied entry in the lab.

WHILE WORKING IN THE LAB

- Adhere to experimental schedule as instructed by the lab in-charge.
- Get the previously executed program signed by the instructor.
- Get the output of the current program checked by the instructor in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and do not play games.
- If anyone caught red handed carrying any equipment of the lab, then he will have to face serious consequences.

EXPERIMENT NO.- 1

Use of Basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit.

ls command

The **ls** command lists all files in the directory that match the *name*. If name is left blank, it will list all of the files in the directory.

The syntax for the **ls** command is:

ls [options] [names]

Options:

-a	Displays all files.
-b	Displays nonprinting characters in octal.
-c	Displays files by file timestamp.
-C	Displays files in a columnar format (default)
-d	Displays only directories.
-f	Interprets each name as a directory, not a file.
-F	Flags filenames.
-g	Displays the long format listing, but exclude the owner name.
-i	Displays the inode for each file.
-l	Displays the long format listing.
-L	Displays the file or directory referenced by a symbolic link.
-m	Displays the names as a comma-separated list.
-n	Displays the long format listing, with GID and UID numbers.
-o	Displays the long format listing, but excludes group name.
-p	Displays directories with /
-q	Displays all nonprinting characters as **?**
-r	Displays files in reverse order.

-R	Displays subdirectories as well.
-t	Displays newest files first. (based on timestamp)
-u	Displays files by the file access time.
-x	Displays files as rows across the screen.
-1	Displays each entry on a line.

pwd command.
pwd command will print your home directory on screen, pwd means present working directory.

/u0/ssb/sandeep
is output for the command when I use pwd in /u0/ssb/sandeep directory.

grepcommand

The **grep** command allows you to search one file or multiple files for lines that contain a pattern. Exit status is 0 if matches were found, 1 if no matches were found, and 2 if errors occurred.

The syntax for the **grep** command is:

grep [options] pattern [files]

Options:

-b	Display the block number at the beginning of each line.
-c	Display the number of matched lines.
-h	Display the matched lines, but do not display the filenames.
-i	Ignore case sensitivity.
-l	Display the filenames, but do not display the matched lines.
-n	Display the matched lines and their line numbers.
-s	Silent mode.
-v	Display all lines that do NOT match.
-w	Match whole word.

dd command
Copy a file, converting and formatting according to the options.

Syntax

dd [OPERAND]...
dd OPTION

Options :

bs=BYTES	force ibs=BYTES and obs=BYTES
cbs=BYTES	convert BYTES bytes at a time
conv=CONVS	convert the file as per the comma separated symbol list
count=BLOCKS	copy only BLOCKS input blocks
ibs=BYTES	read BYTES bytes at a time
if=FILE	read from FILE instead of stdin
iflag=FLAGS	read as per the comma separated symbol list
obs=BYTES	write BYTES bytes at a time
of=FILE	write to FILE instead of stdout
oflag=FLAGS	write as per the comma separated symbol list
seek=BLOCKS	skip BLOCKS obs-sized blocks at start of output
skip=BLOCKS	skip BLOCKS ibs-sized blocks at start of input
status=noxfer	suppress transfer statistics

mkdir command.

#mkdir sandeep will create new directory, i.e. here sandeep directory is created.

#cd command.

cd sandeep will change directory from current directory to sandeep directory. Use pwd to check your current directory and ls to see if sandeep directory is there or not. You can then use cd sandeep to change the directory to this new directory.

#rmdir command.

rmdir command will remove directory or directories if a directory is empty.

Options:

- rm -r directory_name will remove all files even if directory is not empty.
- rmdir sandeep is how you use it to remove sandeep directory.
- rmdir-p will remove directories and any parent directory that is empty.

#cat command

cat's general syntax is

cat [options] [filenames] [-] [filenames]

Reading Files

#cat file1

Concatenation

For example, the following command will concatenate copies of the contents of the three files file1, file2 and file3:

#cat file1 file2 file3

This output could just as easily be redirected using the output redirection operator to another file, such as file4, using the following:

#cat file1 file2 file3 > file4

File Creation

#cat > file1

If a file named file1 already exists, it will be overwritten (i.e., all of its contents will be erased) by the new, empty file with the same name. Thus the cautious user might prefer to instead use the append operator (represented by two successive rightward pointing angular brackets) in order to prevent unintended erasure. That is,

#cat >> file1

That is, if an attempt is made to create a file by using cat and the append operator, and the new file has the same name as an existing file, the existing file is, in fact, preserved rather than overwritten, and any new text is added to the end of the existing file.

Typing the following and then pressing ENTER creates a new file named file2 that contains copy of the contents of file1:

#cat file1 > file2

For example, to create a new file file6 that consists of text typed in from the keyboard followed by the contents of file5, first enter the following:

#cat - file5 > file6

Or to create a new file file8 that consists of the contents of file7 followed by text typed in from the keyboard, first enter the following:

cat file7 -> file8

#Banner command.
banner prints characters in a sort of ascii art poster, for example to print wait in big letters. I will type banner wait at unix command line or in my script. This is how it will look.

```
#   #   ##    #   #####
#   #  # #    #     #
#   # #   #   #     #
# ## # ######   #     #
## ## #   #   #     #
#   # #   #   #     #
```

 # touch command

touch command is used to create files with no contents.

#touch file1

#wc command
wc command counts the characters, words or lines in a file depending upon the option.

Options

- wc -l filename will print total number of lines in a file.
- wc -w filename will print total number of words in a file.

#du command.

- "**du**" stands for disk usage. This command is used to show the amount of disk space consumed by one or more directories

- **Syntax:**
- du [-a] [-k] [-s] [-d] [-L] [-o] [-r] [-x] directories

	Displays the space that each file is taking up.
-k	Write the files sizes in units of 1024 bytes, rather than the default 512-byte units.
-s	Instead of the default output, report only the total sum for each of the specified files.
-d	Do not cross filesystem boundaries. For example, du -d / reports usage only on the root partition.
-L	Process symbolic links by using the file or directory which the symbolic link references, rather than the link itself.
-o	Do not add child directories' usage to a parent's total. Without this option, the usage listed for a particular directory is the space taken by the files in that directory, as well as the files in all directories beneath it. This option does nothing if -s is used.
-r	Generate messages about directories that cannot be read, files that cannot be opened, and so forth, rather than being silent (the default).
-x	When evaluating file sizes, evaluate only those files that have the same device as the file specified by the file operand.
directories	Specifies the directory or directories.

#who command

who command displays information about the current status of system.
who options file
Who as default prints login names of users currently logged in.

Options:

- -a use all options.
- -b Report information about last reboot.
- -d report expired processes.
- -H print headings.

- -p report previously spawned processes.

#dfspace Command:

- The dfspace command formats the output for the df command to make it easier to read.
- **sort command:**
- Sorts the lines in a text file.
- Syntax:
- sort [options]... [file]

-b	Ignores spaces at beginning of the line.
-c	Check whether input is sorted; do not sort
-d	Uses dictionary sort order and ignores the punctuation.
-f	Ignores caps
-g	Compare according to general numerical value
-i	Ignores nonprinting control characters.
-k	Start a key at POS1, end it at POS2 (origin 1)
-m	Merges two or more input files into one sorted output.
-M	Treats the first three letters in the line as a month (such as may.)
-n	Sorts by the beginning of the number at the beginning of the line.
-o	Write result to FILE instead of standard output
-r	Sorts in reverse order
-s	Stabilize sort by disabling last-resort comparison
-t	Use SEP instead of non-blank to blank transition
-T	Use DIR for temporaries, not $TMPDIR or /tmp; multiple options specify multiple directories
-u	If line is duplicated only display once
-z	End lines with 0 byte, not newline

EXPERIMENT NO.-2

Commands related to inode, I/O redirection and piping, process control commands, mails.

Inode:-

Inodes in Unix are data structures which contain all the properties of a file, metadata. The properties of the file include file size, file owner, the group to which the file belongs to, file access rights, hard link count, the location where the file contents are present and time stamps(last modified time, last accessed time, last changed time). In other words, the inode data structure contains all the information of the file except the file name and its contents.

Commands to access Inode numbers

Following are some commands to access the Inode numbers for files:

i)Ls -i Command

$ ls –i

```
1448240 a 1441807 Desktop 1447344 mydata 1441813 Pictures 1442737 testfile 1448145
worm 1448240 a1 1441811 Documents 1442707 my_ls 1442445 practice 1442739 test.py
1447139 alpha 1441808 Downloads 1447278 my_ls_alpha.c 1441810 Public 1447099
1447478 article_function_pointer.txt 1575132 google 1447274 my_ls.c 1441809 Templates
1442390 chmodOctal.txt 1441812 Music 1442363 output.log 1448800 testdisk.log 1575133
vlc
```

See that the Inode number for ‘a’ and ‘a1’ are same as we created ‘a1’ as hard link.

ii) Df -i Command

df -i command displays the inode information of the file system.

$ df –i

```
Filesystem      Inodes  IUsed   IFree IUse% Mounted on
/dev/sda1      1875968  293264 1582704   16% /
none            210613     764  209849    1% /dev
none            213415       9  213406    1% /dev/shm
none            213415      63  213352    1% /var/run
none            213415       1  213414    1% /var/lock
```

```
/dev/sda2          7643136  156663 7486473    3% /home
```

The flag -i is used for displaying Inode information.

iii) Stat Command

Stat command is used to display file statistics that also displays inode number of a file

$ stat a

```
File: `a'
Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 805h/2053d Inode: 1448240 Links: 2
Access: (0644/-rw-r--r--) Uid: ( 1000/himanshu) Gid: ( 1001/ family)
Access: 2012-01-14 16:30:04.871719357 +0530
Modify: 2012-01-14 16:29:50.918267873 +0530
Change: 2012-01-14 16:30:03.858251514 +0530
```

I/O redirection and piping:-

Standard Output

Most command line programs that display their results do so by sending their results to a facility called standard output. By default, standard output directs its contents to the display. To redirect standard output to a file, the ">" character is used like this:

[me@linuxbox me]$ ls > file_list.txt

In this example, the ls command is executed and the results are written in a file named file_list.txt. Since the output of ls was redirected to the file, no results appear on the display.

Each time the command above is repeated, file_list.txt is overwritten (from the beginning) with the output of the command ls. If you want the new results to be appended to the file instead, use ">>" like this:

[me@linuxbox me]$ ls >> file_list.txt

When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when you attempt to append the redirected output, the file will be created.

Standard Input

Many commands can accept input from a facility called standard input. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the "<" character is used like this:

[me@linuxbox me]$ sort < file_list.txt

In the above example we used the sort command to process the contents of file_list.txt. The results are output on the display since the standard output is not redirected in this example. We could redirect standard output to another file like this:

[me@linuxbox me]$ sort < file_list.txt > sorted_file_list.txt

As you can see, a command can have both its input and output redirected. Be aware that the order of the redirection does not matter. The only requirement is that the redirection operators (the "<" and ">") must appear after the other options and arguments in the command.

Pipes

The most useful and powerful thing you can do with I/O redirection is to connect multiple commands together with what are called pipes. With pipes, the standard output of one command is fed into the standard input of another.

[me@linuxbox me]$ ls -l | less

In this example, the output of the ls command is fed into less. By using this "| less" trick, you can make any command have scrolling output. I use this technique all the time.

By connecting commands together, you can acomplish amazing feats. Here are some examples you'll want to try:

Examples of commands used together with pipes	
Command	**What it does**
ls -lt | head	Displays the 10 newest files in the current directory.
du | sort -nr	Displays a list of directories and how much space they consume, sorted from the largest to the smallest.

find . -type f -print | wc -l	Displays the total number of files in the current working directory and all of its subdirectories.

Process control commands:-

Here "PID" is refer to the process ID, that you can get from command "ps -aux"

fg PID
Bring a background or stopped process to the foreground.

bg PID
Send the process to the background. Opposite to fg. The same can be accomplished with z. If you have stopped jobs, you have to type exit twice in row to log out.

any_command&
Run any command in the background (the symbol "&" means "run the proceeding command in the background").

batch any_command
Run any command (usually one that is going to take more time) when the system load is low. I can logout, and the process will keep running.

at 17:00
Execute a command at a specified time. You will be prompted for the command(s) to run, until you press d.

kill PID
Force a process shutdown. First determine the PID of the process to kill using ps.

killall program_name
Kill program(s) by name.

To force termination of a job whose process ID is 111, enter the command

kill -9 11

EXPERIMENT NO. - 3

Shell Programming: Shell script based on control structure- If-then-fi, if-then else-if, nested if-else, to find:

(A) Greatest among three numbers.

(B) To find a year is leap year or not.

(C) To calculate profit or loss.

(D) To input angles of triangle and finds out whether it is valid triangle or not

(E) To check whether a character is alphabet, digit or special character.

(A) Greatest among three numbers. Code-

```
#!/bin/bash
echo "enter first number"
read first
echo "enter second number"
read sec
echo "enter third number"
read third
if [ $first -gt $sec ] ; then
if [ $first -gt $third ] ; then
echo -e " $first is greatest number "
else
echo -e " $third is greatest number "
fi
else
if [ $sec -gt $third ] ; then
echo -e " $sec is greatest number "
else
echo -e " $third is greatest number "
fi
```

```
fi
```

Output-

:-$ gedit great.sh

:-$ bash great.sh enter first number 34

enter second number 78

enter third number 10

78 is greatest number

:-$

(B) To find a year is leap year or not. Code-

```
echo "Enter Year:"
read y
year=$y
y=$(( $y % 4 ))
if [ $y -eq 0 ]
then
echo "$year is Leap Year!"
else
echo "$year is not a Leap Year!"
fi
```

Output-

:-$ gedit leap.sh

:-$ bash leap.sh enter year

2006

2006 is not a leap year

:-$

bash leap.sh enter year

2000

2000 is a Leap year

:-$

(C) To calculate profit or loss. Code-

```
clear

echo -e "Enter CostPrice:\c" readcp
echo -e "Enter
Selling Price:\c"
readsp
if [ $sp
-eq $cp
]; then
echo -e "\nNo profit or loss has
incurred.\n" elif [ $sp -lt $cp ];
then
echo -e "\nLoss of Rs.`expr $cp - $sp` has
incurred.\n" else
echo -e "\nProfit of Rs.`expr $sp - $cp` has
incurred.\n" fi
```

Output-

:-$ geditpl1.sh

:-$ bashpl1.sh

Enter Cost Price:1000 Enter Selling Price:1250

Profit of Rs.250 has incurred.

:-$

(D) To input angles of triangle and finds out whether it is valid triangle or not

```
bool is Valid Triangle (int a, int b, int c)
{
```

```
int longestSide = a;
if (b > longestSide )
longestSide = b;
if(c > longestSide )
longestSide = c;
return (longestSide < a + b + c - longestSide);
}
```

Output-

```
:-$ gedit tri.c
:-$ gcc -o tri tri.c
:-$ ./tri
Enter three angles of triangle:
30
60
90
Triangle is valid.
:-$
```

(E) To check whether a character is alphabet, digit or special character.

```
echo "enter a char"
read c
if [[ $c == [A-Z] ]];
then
echo "upper"
elif [[ $c == [a-z] ]];
then
echo "lower"
else
```

```
echo "Digit or special symbols!"
fi
```

Output-

```
:-$ gedit digit.c
:-$ gcc -o digit digit.c
:-$ ./digit
Please Enter any
character : d d is an
Alphabet
:-$ ./digit
Please Enter any
character : 7 7 is a
Digit
:-$ ./digit
Please Enter any
character : # # is a
Special Character
:-$
```

EXPERIMENT NO. - 4

Shell Programming - Looping- while, until, for loops

(A) Write a shell script to print all even and odd number from 1 to10.
(B) Write a shell script to print table of a given number
(C) Write a shell script to calculate factorial of a given number.
(D) Write a shell script to print sum of all even numbers from 1 to 10.
(E) Write a shell script t print sum of digits of any number.

(A) Write a shell script to print all even and odd number from 1 to 10. Code-

```
echo "First Odd and Even number till 30 are"
n=1
while [ $n -lt 30 ]; do
out=$(( $n % 2 ))
if [ "$out" -eq 0 ] then
echo "$n is even number"
else
echo "$n is ODD number"
fi
n=$(( $n + 1 ))
done
```

Output-

```
:-$ gedit evenodd.sh
:-$ bash evenodd.sh Enter a Number:
6
Number is even
:-$ bash evenodd.sh Enter a Number:
7
Number is odd
:-$
```

(B) Write a shell script to print table of a given number.

```
echo "Enter a Number"
read n
echo "Enter Range"
read r
i=0
while [ $i -le $r ]
do
echo " $n x $i = `expr $n \* $i`"
i=`expr $i + 1`
done
```

Output-
:-$ gedit table.sh
:-$ bash table.sh Enter a Number 4
Enter Range 10
4 x 0 =0
4 x 1 =4
4 x 2 =8
4 x 3 =12
4 x 4 =16
4 x 5 =20
4 x 6 =24
4 x 7 =28
4 x 8 =32
4 x 9 =36
4 x 10 = 40
:-$

(C) Write a shell script to calculate factorial of a given number.

```
#taking input from user echo -e "enter a number" read n
#if enter value less than 0 if [ $n -le 0 ] ; then
echo "invalid number" exit
fi
#factorial logic
if [ $n -gt 0 ] ; then for((i=$n;i>=1;i--)) do
fact=`expr $fact \* $i` done
fi
```

```
echo "The factorial of $n is $fact"
```

Output-

```
:-$ gedit fact.sh
:-$ bash fact.sh enter a number The factorial of 5 is 120
:-$
```

(D) Write a shell script to print sum of all even numbers from 1 to 10.

```
“Enter upper limit”
read n
$i=2
while [$i -lt $n]
do
expr ‘$sum=$sum+$i’
expr ‘$i=$i+2’
done
echo “Sum is : $sum”
```

(E) Write a shell script to print sum of digits of any number.

```
echo -n "Enter number : "
read n
# store single digit
sd=0
# store number of digit
sum=0
# use while loop to caclulate the sum of all digits
while [ $n -gt 0 ]
do
sd=$(( $n % 10 )) # get Remainder
n=$(( $n / 10 )) # get next digit
sum=$(( $sum + $sd )) # calculate sum of digit
done
echo "Sum of all digit is $sum"
```

Output-

```
:-$ gedit digit.sh
:-$ bash digit.sh
enter the
number
23456
the sum of 23456 is 20
:-$
```

EXPERIMENT NO. - 5

Shell Programming - case structure, use of break

(A) Write a shell script to make a basic calculator which performs addition, subtraction, multiplication, division.

(B) Write a shell script to print sum of all even numbers from 1 to10.

(C) WAP in C to sort array elements in ascending or descending order.

(A) Write a shell script to make a basic calculator which performs addition, subtraction, multiplication, division.

```
echo "Enter number1:\c"
read n1
echo "Enter operator(+, -, /, *):\c"
read op
echo "Enter number2:\c"
read n2
if [ "$op" = "+" ];
then
calc=`echo $n1 + $n2|bc`
echo "\n$n1 + $n2 = $calc\n"
elif [ "$op" = "-" ];
then
calc=`echo $n1 - $n2|bc`
echo "\n$n1 - $n2 = $calc\n"
 elif [ "$op" = "*" ];
then
calc=`echo $n1 \* $n2|bc`
echo "\n$n1 * $n2 = $calc\n"
elif [ "$op" = "/" ];
```

```
then
calc=`echo $n1 / $n2|bc`
echo "\n$n1 / $n2 = $calc\n"
else
echo "Invalid operator!\n"
fi
```

Output-

```
Enter operator(+, -, /, *):+
Enter number1:100
Enter number2:50
100 + 50 = 150
Enter operator (+, -, /, *):*
Enter number1:90
Enter number2:5
90 * 5 = 450
```

(B) Write a shell script to print days of a week

```
echo "enter a number"
read n
case $n in
1) echo "Sunday" ;;
2) echo "Monday" ;;
3) echo "Tuesday" ;;
4) echo "Wednesday" ;;
5) echo "Thursday" ;;
6) echo "Friday" ;;
7) echo "Saturday" ;;
```

```
*) echo "enter value between 1 to 7" ;;
esac
```

Output:

```
$sh a.sh
Enter a number: 4
Wednesday
```

(C) Write a shell script to print starting 4 months having 31 days.

```
month="$1"
if [ -z "$month" ] || [ ${#month} != 6 ]; then
echo must specify yyyymm, e.g. $(date +%Y%m)
exit 2
fi
year=${month:0:4}
month=${month:4:2}
echo $((
(
$(date -u -d "${year}-${month}-01 +1 month" +%s)
-
$(date -u -d "${year}-${month}-01" +%s)
)
/
86400
))
```

EXPERIMENT NO. - 6

Shell Programming - Functions

(A) Write a shell script to find a number is Armstrong or not.

(B) Write a shell script to find a number is palindrome or not.

(C) Write a shell script to print Fibonacci series.

(D) Write a shell script to find prime number.

(E) Write a shell script to convert binary to decimal and decimal to binary.

(A) Write a shell script to find a number is Armstrong or not.

```
echo "Enter a number: "
readc
while [ $x -gt 0 ]
do
r=`expr $x % 10`
n=`expr $r \* $r \* $r`
sum=`expr $sum + $n`
x=`expr $x / 10`
done
if [ $sum -eq $c ]
then
echo "It is an Armstrong Number."
else
echo "It is not an Armstrong Number."
fi
```

Output-

```
:-$ gedit armstrong.sh
```

```
:-$ bash armstrong.sh
Enter a number:
123
It is not an Armstrong Number.
:-$ bash armstrong.sh
Enter a number:
153
It is an Armstrong Number.
:-$
```

(B)Write a shell script to find a number is palindrome or not.

```
#!/bin/bash
# Shell script to find whether an input number is palindrome or not
#.....
echo -n "Enter number :"
read n
# store single digit
sd=0
# store number in reverse order
rev=""
# store original number
on=$n
while [ $n -gt 0 ]
do
sd=$(( $n % 10 ))
# get Remainder n=$(( $n / 10 ))
# get next digit
```

```
# store previous number and current digit in reverse
rev=$( echo ${rev}${sd} )
done
if [ $on -eq $rev ];
then
echo "Number is palindrome"
else
echo "Number is NOT palindrome"
fi
```

Output-

```
:-$ gedit pal11.sh
:-$ bash pal11.sh
Enter number : 123
Number is NOT palindrome
:-$ bash pal11.sh
Enter number : 121
Number is palindrome
:-$
```

(C) Write a shell script to print Fibonacci series.

```
# Program for Fibonacci
# Series
# Static input fo N
N=6
# First Number of the
# Fibonacci Series
a=0
```

```
# Second Number of the
# Fibonacci Series
b=1
echo "The Fibonacci series is : "
for (( i=0; i<N; i++ ))
do
echo -n "$a "
fn=$((a + b))
a=$b
b=$fn
done
# End of for loop
```

Output:

:-$ gedit fibonacci.sh

:-$ bash fibonacci.sh

Enter a Number :10

The Fibonacci sequence for the number 10 is :

0 1 1 2 3 5 8 13 21 34 55

:-$

(D) Write a shell script to find prime number.

```
echo "Enter a number: "
read num
 i=2
 f=0
while [ $i -le `expr $num / 2` ]
```

```
do
 if [ `expr $num % $i` -eq 0 ]
 then
f=1
fi
 i=`expr $i + 1`
done
 if [ $f -eq 1 ]
 then
 echo "The number is composite"
else
echo "The number is Prime"
 fi
```

Output-

```
:-$ gedit prime.sh
:-$ bash prime.sh
enter number
7
it is prime
:-$ bash prime.sh
enter number
4
it is not prime
:-$
```

(E) Write a shell script to convert binary to decimal and decimal to binary.

```
echo "Conversion of decimal to Binary and Binary to Decimal"
echo "1. Convert Decimal to Binary"
echo "2. Convert Binary to Decimal"
echo "3. Exit"
echo "Enter ur choice:"
read ch
case $ch in
1) echo "Enter any decimal no:"
read num
rem=1
bno=" "
while [ $num -gt 0 ]
do
rem=`expr $num % 2 `
bno=$bno$rem
num=`expr $num / 2 `
done
i=${#bno}
final=" "
while [ $i -gt 0 ]
do
rev=`echo $bno | awk '{ printf substr( $0,'$i',1 ) }'`
final=$final$rev
i=$(( $i - 1 ))
done
echo "Equivalent Binary no:" $final ;;
2) echo "Enter any Binary no;"
read bino
```

```
len=${#bino}
i=1
pow=$((len - 1 ))
while [ $i -le $len ]
do
n=`echo $bino | awk '{ printf substr( $0,'$i',1 )}' `
j=1
p=1
while [ $j -le $pow ]
do
p=$(( p * 2 ))
j=$(( j + 1 ))
done
dec=$(( n * p ))
findec=$(( findec + dec ))
pow=$((pow - 1 ))
i=$(( i + 1 ))
done
echo "Equivalent Decimal no:"$findec ;;
3) echo "Enter correctly:" ;;
esac
```

Output-

Conversion of decimal to Binary and Binary to Decimal

1. Convert Decimal toBinary
2. Convert Binary toDecimal
3. Exit

Enter ur choice:

2

Enter any Binary no; 1100

Equivalent Decimal no:12

:-$

Conversion of decimal to Binary and Binary to Decimal

1. Convert Decimal toBinary

2. Convert Binary toDecimal

3. Exit

Enter ur choice:1

Enter any decimal no:12

Equivalent Binary no: 1100

:-$

EXPERIMENT NO. - 7

Write a shell script to print different shapes- Diamond, triangle, square, rectangle, hollow square etc.

Rectangle-

Code-

```
/**

* C program to print rectangle starpattern
*/
#include <stdio.h>
int main()
{
int i, j, rows, columns;
/* Input rows and columns from user */
printf("Enter number of rows: ");
scanf("%d", &rows);
printf("Enter number of columns: ");
scanf("%d", &columns);
/* Iterate through each row */
for(i=1; i<=rows; i++)
{
/* Iterate through each column */
for(j=1; j<=columns; j++)
{
/* For each column print star */
printf("*");
}
/* Move to the next line/row */
printf("\n");
}
return 0;
}
```

Output-

```
:-$ gedit rectangle.c
:-$ gcc -o rectangle rectangle.c

:-$ ./rectangle

Enter number of rows: 5

Enter number of columns: 10

**********

**********

**********

**********

**********

:-$
```

Diamond-

Code-

```
/**

* C program to print diamond starpattern
*/
#include <stdio.h>
int main()
{
int i, j, rows;
int stars, spaces;
printf("Enter rows to print : ");
scanf("%d", &rows);
stars = 1;
```

```
spaces = rows - 1;
/* Iterate through rows */
for(i=1; i<rows*2; i++)
{
/* Print spaces */
for(j=1; j<=spaces; j++)
printf("");
/* Print stars */
for(j=1; j<stars*2; j++)
printf("*");
/* Move to next line */
printf("\n");
if(i<rows)
{
spaces--;
stars++;
}
else
{
spaces++;
stars--;
}
}
return 0;
}
```

Output-

:-$ gedit diamond.c

:-$ gcc -o diamond diamond.c

:-$ ./diamond

Enter rows to print : 5

*

***

*****

****

****

****

*****

***

*

:-$

Triangle-

Code-

```
/*
* C program to print right triangle star patternseries
*/
#include <stdio.h>
int main()
{
int i, j, n;
/* Input number of rows from user */
printf("Enter value of n: ");
scanf("%d", &n);
for(i=1; i<=n; i++)
{
/* Print i number of stars */
for(j=1; j<=i; j++)
{
printf("*");
}
/* Move to next line */
```

```
printf("\n");
}
return 0;
}
```

Output-

```
:-$ gedit triangle.c

:-$ gcc -o triangle triangle.c

:-$ ./triangle

Enter value of n: 5

*

**

***

****

*****

:-$
```

Square-

Code-

```
/**

* C program to print square starpattern
*/
#include
<stdio.h>intmain()
{
int i, j,N;
/* Input number of rows from user */
printf("Enter number of rows: ");
```

```
scanf("%d", &N);
/* Iterate through N rows */
for(i=1; i<=N;i++)
{
/* Iterate over columns*/
for(j=1; j<=N;j++)
{
/* Print star for each column */
printf("*");
}
/* Move to the next line/row */
printf("\n");
}
return 0;
}
```

Output-

```
:-$ gedit sqr.c

:-$ gcc -o sqr sqr.c

:-$ ./sqr

Enter number of rows: 5

*****

*****

*****

*****

*****

:-$
```

Hollow Square

Code-

```
/**

* C program to print hollow square starpattern
*/
#include <stdio.h>
intmain()
{
int i, j,N;
/* Input number of rows from user */
printf("Enter number of rows: ");
scanf("%d", &N);
/* Iterate over each row */
for(i=1; i<=N; i++)
{
/* Iterate over each column */
for(j=1; j<=N; j++)
{
if(i==1 || i==N || j==1 || j==N)
{
/* Print star for 1st, Nth row and column */
printf("*");
}
else
{
printf("");
}
}
/* Move to the next line/row */
printf("\n");
}
return 0;
}
```

Output-

:-$ gedit hsqr.c

:-$ gcc hsqr hsqr.c gcc: error: hsqr: No such file

or directory

:-$ gcc -o hsqr hsqr.c

:-$ ./hsqr

Enter number of rows: 5

*****

* *

* *

* *

*****

EXPERIMENT NO. - 8

Shell Programming- Arrays

(A) WAP in C to read and print elements of array.

(B) WAP in C to find sum of all array elements.

(C) WAP in C to find reverse of array elements.

(D) Write a C program to search an element in an array.

(E) Write a C program to sort array elements in ascending or descending order.

(A) WAP in C to read and print elements of array. Code-

```
#include <stdio.h>
void main()
{
int arr[10];
int i;
printf("\n\nRead and Print elements of an array:\n");
printf("----------------------------------------\n");
printf("Input 10 elements in the array :\n");
for(i=0; i<10; i++)
{
printf("element - %d : ",i);
scanf("%d", &arr[i]);
}
printf("\nElements in array are: ");
for(i=0; i<10; i++)
{
printf("%d ", arr[i]);
```

```
}
printf("\n");
}
```

Output-

```
:-$ gedit arr1.c
:-$ gcc -o arr1 arr1.c
:-$ ./arr1
Read and Print elements of an array:
------------------------------------------------
Input 10 elements in the array :
element - 0 :9
element - 1 :4
element - 2 :6
element - 3 :3
element - 4 :4
element - 5 :56
element - 6 :23
element - 7 :65
element - 8 :78
element - 9 :90
Elements in array are: 9 4 6 3 4 56 23 65 78 90
:-$
```

(B) WAP in C to find sum of all array elements. Code-

```
#include <stdio.h>
void main()
{
int a[100];
int i, n, sum=0;
printf("\n\nFind sum of all elements of array:\n");
printf("---------------------------------------\n");
printf("Input the number of elements to be stored in the array :");
scanf("%d",&n);
printf("Input %d elements in the array :\n",n);
for(i=0;i<n;i++)
{
printf("element - %d : ",i);
scanf("%d",&a[i]);
}
for(i=0; i<n; i++)
{
sum += a[i];
}
printf("Sum of all elements stored in the array is : %d\n\n", sum);
    }
```

Output-

:-$ gedit sum1.c

:-$ gcc -o sum1 sum1.c

```
:-$ ./sum1
Find sum of all elements of array:
-------------------------------------------
Input the number of elements to be stored in the array :4
Input 4 elements in the array :
element - 0 :6
element - 1 :5
element - 2 :8
element - 3 :3
Sum of all elements stored in the array is : 22
:-$
```

(C) WAP in C to find reverse of array elements. Code-

```
// Iterative C program to reverse an array
#include<stdio.h>
/* Function to reverse arr[] from start to end*/
void rvereseArray(int arr[], int start, int end)
{
int temp;
while (start < end)
{
temp = arr[start];
arr[start] = arr[end];
arr[end] = temp;
start++;
```

```
end--;
}
}
/* Utility that prints out an array on a line */
void printArray(int arr[], int size)
{
int i;
for (i=0; i < size; i++)
printf("%d ", arr[i]);
printf("\n");
}
/* Driver function to test above functions */
int main()
{
int arr[] = {1, 2, 3, 4, 5, 6};
printArray(arr, 6);
rvereseArray(arr, 0, 5);
printf("Reversed array is \n");
printArray(arr, 6);
return 0;
}
```

Output-

```
:-$ gedit reverse1.c
:-$ gcc -o reverse1 reverse1.c
:-$ ./reverse1
Enter size of the array: 5
Enter elements in array: 70
```

45

87

32

09

Array in reverseorder:9 32 87 45 70

(D) Write a C program to search an element in an array.

inux Shell Programming Page 38

```
/* C Program to Search an Element in an Array */
#include<stdio.h>
int main()
{
int arr[10], Size, i, Search, Flag;
printf("\n Please Enter the size of an array : ");
scanf("%d",&Size);
printf("\n Please Enter %d elements of an array: \n", Size);
for(i = 0; i < Size; i++)
{
scanf("%d",&arr[i]);
}
printf("\n Please Enter the Search Element : ");
scanf("%d",&Search);
Flag = 0;
for(i = 0; i < Size; i++)
{
if(arr[i] == Search)
```

```
{
Flag = 1;
break;
}
}
if(Flag == 1)
{
printf("\n We found the Search Element %d at Position %d ", Search, i + 1);
}
else
{
printf("\n Sorry!! We haven't found the the Search Element %d ", Search);
}
return 0;
}
```

Output-

```
:-$ gedit search.c
:-$ gcc -o search search.c
:-$ ./search
Enter number of elements in array
5
Enter 5 integer(s)
4
6
8
2
```

```
4
Enter a number to search
6
6 is present at location 2.
```

(E) Write a C program to sort array elements in ascending or descending order.

```
1. #include <stdio.h>
2. #define MAXSIZE 10
3. void main()
4. {
int array[MAXSIZE];
int i, j, num, temp;
printf("Enter the value of num \n");
scanf("%d", &num);
printf("Enter the elements one by one \n");
for (i = 0; i < num; i++)
{
scanf("%d", &array[i]);
}
printf("Input array is \n");
for (i = 0; i < num; i++)
{
printf("%d\n", array[i]);
}
/* Bubble sorting begins */
for (i = 0; i < num; i++)
{
```

```
for (j = 0; j < (num - i - 1); j++)
{
if (array[j] > array[j + 1])
{
temp = array[j];
array[j] = array[j + 1];
array[j + 1] = temp;
}
}
}
printf("Sorted array is...\n");
for (i = 0; i < num; i++)
{
printf("%d\n", array[i]);
}
}
```