**POORNIMA INSTITUTE OF ENGINEERING & TECHNOLOGY, JAIPUR**

**Department of Computer Science and Engineering**

**Lab Manual**

**Linux and Shell Programming Lab**

**4CS4-24**



|  |  |  |  |
| --- | --- | --- | --- |
| **Branch** | **CS** | **Name of Lab** | **Linux Shell Programming Lab** |
| **Session** | **2020-21** | **Subject Code** | **4CS4-24** |
| **Year** | **2nd Year** | **Faculty** | **Dr.Anil Kmar** |
| **Semester** | **4th Semester** | **Lab Assistant** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Document No.** | PIET/CS/CS/2019/V/LAB/LSP/  4CS4-24 | Created By |  |
| Version |  | Verified By |  |
| Authorized By (HOD) | Mr. Deepak Moud | | |

**LAB RULES**

|  |  |
| --- | --- |
| **DO’S** | **DON’TS** |
| Be regular to the lab. | Do not come late to the lab. |
| Follow proper dress code. | Do not throw the connecting wires on the floor. |
| Maintain Silence. | Do not operate µp/IC trainer kits unnecessarily. |
| Know the theory behind the experiment before coming to the lab. | Don't bring any external material inside the LAB. |
| Arrange the chairs/stools and equipment properly before leaving thelab. | Do not panic if you don’t get the output. |
| Avoid unnecessary talking while doing the experiment. | Don't carry any LAB equipment outside the lab. |
| Keep the Table clean. | Do not try to repair or tamper lab equipment. |



**Department of Computer Engineering**

**Academic Session 2019-20(Even)**

**4CS4-24 Linux Shell Programming Lab**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Contents** | **Experiments** | **Lab Turn** |
| 1 | **Understanding of Linux and Basic Shell Commands** | * ls, mkdir, rmdir, cd, cat, banner * touch, file, wc, sort, * cut, grep, dd, dfspace, du, ulimit. * Cal, date, mv, echo, diff * Uniq, passwd, | Turn-01 |
| 2 | **Understanding of Vi editor** | * To learn how to use vi editor and its operating keys and geditor | Turn-02 |
| 3 | **Shell script based on control structure** | * Greatest among three numbers. * To find a year is leap year or not. * To input angles of a triangle and find out whether it is valid triangle or not. | Turn-03 |
| * To check whether a character is alphabet, digit or special character. * To calculate profit or loss. * To count the number of vowels in a line of text. | Turn-04 |
| 4 | **Looping- while, until, for loops** | * Write a shell script to print table of a given number. * Write a shell script to calculate factorial of a given number. * Calculate Gross Salary from basic, DA=20% and HRA=10% | Turn-05 |
| 5 | * Write a shell script to print sum of all even numbers from 1 to 10. * Write a shell script to print sum of digit of any number. * Write a shell script to print all even and odd number from 1 to 10. | Turn-06 |
| 6 | * Write a shell script to print days of a week. * To find the sum of square of individual digits of a number. * Shell program to display student grades. | Turn-07 |
| * Write a shell script to print starting 4 months having 31 days * To find the sum of cube of individual digits of a number. * Shell program to find the smallest number from a set of numbers | Turn-08 |
| 7 | * Write a shell script to find a number is Armstrong or not * Write a shell script to find a number is palindrome or not. * Display of Greeting Message “Good Morning” “Good Evening” | Turn-09 |
| 8 | **Arrays & Shell Scripts for string Processing** | * Write a C program to read and print elements of array. * Write a C program to find sum of all array elements * Matrix Multiplication | Turn-10 |
| * Write a C program to search elements in an Array * Write a C Program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5 | Turn-11 |
| 9 | Certification | * Certification from Udemy/Neptel |  |

**INSTRUCTIONS**

**Before Entering in the Lab**

* All the students are supposed to prepare the theory regarding the next experiment/ Program.
* Students are supposed to bring their lab records as per their lab schedule.
* Previous experiment/program should be written in the lab record.
* If applicable trace paper/graph paper must be pasted in lab record with proper labeling.
* All the students must follow the instructions, failing which he/she may not be allowed in the lab.

**While Working in the Lab**

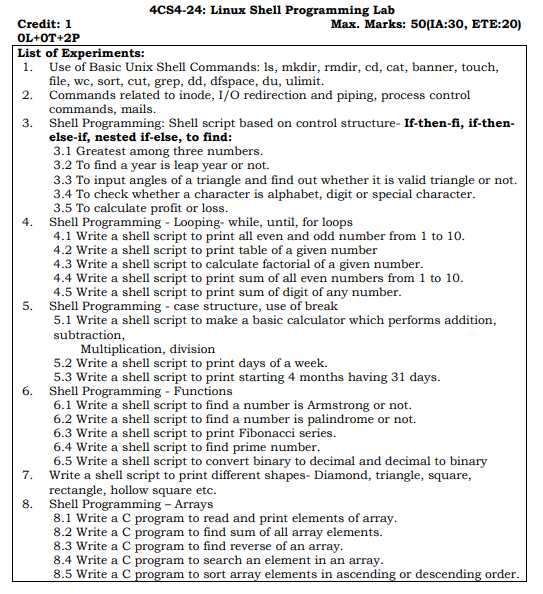
* Adhere to experimental schedule as instructed by the lab in-charge/faculty.
* Get the previously performed experiment/ program signed by the faculty/ lab in charge.
* Get the output of current experiment/program checked by the faculty/ lab in charge in the lab copy.
* Each student should work on his/her assigned computer at each turn of the lab.

Take responsibility of valuable accessories.

**RAJASTHAN TECHNICAL UNIVERSITY, KOTA**

**Syllabus**

**II Year-IV Semester: B.Tech. Computer Science and Engineering**



**MARKS SCHEME**

**RTU Marks Scheme**

|  |  |  |
| --- | --- | --- |
| Maximum Marks Allocation | | |
| Sessional | End-Term | Total |
| 30 | 20 | 50 |

**Marks Division**

|  |  |  |
| --- | --- | --- |
| Mid Term I & II | | |
| Practical | Viva | Total |
| 15 | 5 | 20 |
| Attendance & Performance | | |
| Performance | Attendance | Total |
| 15 | 5 | 20 |
| End-Term Practical | | |
| Practical | Viva | Total |
| 15 | 5 | 20 |

**Internal Assessment System**

Total Marks – 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attendance | Discipline | Performance | Record | Viva | Total |
| 2 | 2 | 2 | 2 | 2 | 10 |

**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 |

**mkdircommand.**

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

**cdcommand.**  
cdsandeep 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.

Cd ../.. : goes to the parent of present 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.
* rmdirsandeep is how you use it to remove sandeep directory.
* rmdir -p will remove directories and any parent directories that are empty.

**catcommand**

**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

file1 will be created as empty file.

**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.
* wc -c filename will print total number of characters 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

|  |  |
| --- | --- |
| -b | 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. |

**whocommand**

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.
* -u report terminal usage.

**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 |

**Cut command:**

Cut out selected fields of each line of a file.

Syntax

cut [-b] [-c] [-f] list [-n] [-d delim] [-s] [file]

|  |  |
| --- | --- |
| -b list | The list following -b specifies byte positions (for instance, -b1-72 would pass the first 72 bytes of each line). When -b and -n are used together, list is adjusted so that no multi-byte character is split. If -b is used, the input line should contain 1023 bytes or less. |
| -c list | The list following -c specifies character positions (for instance, -c1-72 would pass the first 72 characters of each line). |
| -f list | The list following -f is a list of fields assumed to be separated in the file by a delimiter character (see -d ); for instance, -f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -s is specified. If -f is used, the input line should contain 1023 characters or less. |
| list | A comma-separated or blank-character-separated list of integer field numbers (in increasing order), with optional - to indicate ranges (for instance, 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field)). |
| -n | Do not split characters. When -b list and -n are used together, list is adjusted so that no multi-byte character is split. |
| -d delim | The character following -d is the field delimiter (-f option only). Default is tab. Space or other characters with special meaning to the shell must be quoted. delim can be a multi-byte character. |
| -s | Suppresses lines with no delimiter characters in case of -f option. Unless specified, lines with no delimiters will be passed through untouched. |
| file | A path name of an input file. If no file operands are specified, or if a file operand is -, the standard input will be used. |

Examples

**name=`who am i | cut -f1 -d' '`**

Set name to current login name.

**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:

### 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.

### Df -i Command

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

$ df–i

|  |
| --- |
| FilesystemInodesIUsedIFreeIUse% 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.

### Stat Command

[Stat command](http://www.thegeekstuff.com/2009/07/unix-stat-command-how-to-identify-file-attributes/) 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](http://linuxcommand.org/man_pages/sort1.html) 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](http://linuxcommand.org/man_pages/head1.html) | Displays the 10 newest files in the current directory. |
| [du](http://linuxcommand.org/man_pages/du1.html) | sort -nr | Displays a list of directories and how much space they consume, sorted from the largest to the smallest. |
| [find](http://linuxcommand.org/man_pages/find1.html) . -type f -print | [wc](http://linuxcommand.org/man_pages/wc1.html) -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").  
  
**batchany\_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.  
  
**killallprogram\_name**  
Kill program(s) by name.  
  
To force termination of a job whose process ID is 111, enter the command

**kill -9 111**

**EXPERIMENT NO.-3**

**Greatest among three numbers**

echo "Enter three Integers:"

read a b c

if [ $a -gt $b -a $a -gt $c ]

then

echo "$a is Greatest!"

elif [ $b -gt $c -a $b -gt $a ]

then

echo "$b is Greatest!"

else

echo "$c is Greatest!"

fi

**EXPERIMENT NO.-3 .2**

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

**EXPERIMENT NO.-3 .4**

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

**EXPERIMENT NO.-4.1**

#!/bin/bash

counter=1

while [ $counter -le 100 ]

do

if [ $(($counter % 2)) == 0 ]

then

echo $counter

#echo "more magic would happen here"

fi

((counter++))

done

**for odd numbers**

i=1

while [ $i -le 100 ]; do

printf "${i} "

i=$(( i + 2 ))

done

printf "\n"

**EXPERIMENT NO.-4.2**

clear

echo "Type any number to generate Mathematical Table"

read number

i=1

while [ $i -le 10 ]

do

echo " $number \* $i =`expr $number \\* $i ` "

i=`expr $i + 1`

done

**EXPERIMENT NO.-4.3**

# Shell Program to Print Factorial of a Number

clear

i=1

fact=1

echo "Enter any number: "

read n

for (( i=1; i<=n; i++ ))

do

fact=`expr $fact \\* $i`

done

echo "Factorial = $fact"

**EXPERIMENT NO.-5.1**

# Shell Program to Implement Small Calculator

clear

echo "Enter 1st number: "

read a

echo "Enter 2nd number: "

read b

echo -e "1. Add\n2. Subtract\n3. Multiply\n4. Divide"

echo "Enter your choice: "

read ch

case $ch in

1) c=`expr $a + $b`;;

2) c=`expr $a - $b`;;

3) c=`expr $a \\* $b`;;

4) c=`expr $a / $b`;;

\*) echo "Invalid choice."

exit 0;;

esac

echo "Result = $c"

**EXPERIMENT NO.-5.2**

days=(mon tue wed thurs fri sat)

for ((i=0; i<${#days[@]}; i++))

do

if (( i < 4 ))

then

echo "weekday: ${days[$i]}"

else

echo "weekend: ${days[$i]}"

fi

done

**EXPERIMENT NO.-5.3**

Date\_m=2014/09/15

Days=25

Seconds=$(echo "$Days"\*60\*60\*24| bc -l)

Tp2date=$(date -d "$Date\_m" +%Y/%m/%d)

Tp2sec=$(date -d "$TD" +%s)

Tp1sec=$(echo "$TDsec"-"$Seconds"| bc -l)

Tp1date=$(date -d @$Tp1sec)

echo "$Tp1date"

**EXPERIMENT NO.-6.1**

echo "Enter a number: "

read c

x=$c

sum=0

r=0

n=0

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

**EXPERIMENT NO.-6.2**

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

**EXPERIMENT NO.-6.3**

clear

echo "How many number of terms to be generated ?"

read n

x=0

y=1

i=2

echo "Fibonacci Series up to $n terms :"

echo "$x"

echo "$y"

while [ $i -lt $n ]

do

i=`expr $i + 1 `

z=`expr $x + $y `

echo "$z"

x=$y

y=$z

done

**EXPERIMENT NO.-6.4**

echo -n "Enter a number: "

read num

i=2

while [ $i -lt $num ]

do

if [ `expr $num % $i` -eq 0 ]

then

echo "$num is not a prime number"

echo "Since it is divisible by $i"

exit

fi

i=`expr $i + 1`

done

echo "$num is a prime number "

a=$1

b=$2

c=$3

if[ $# -lt 3 ]

then

echo "$0 n1 n2 n3"

exit 1

fi

if[ $a -gt $b -a $a -gt $c ]

then

echo "$a is largest integer"

elif[ $b -gt $a -a $b -gt $c ]

then

echo "$b is largest integer"

elif[ $c -gt $a -a $c -gt $b ];

then

echo "$c is largest integer"

else

echo "Sorry cannot guess number"

fi

**EXPERIMENT NO.-6.5**

tput clear

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