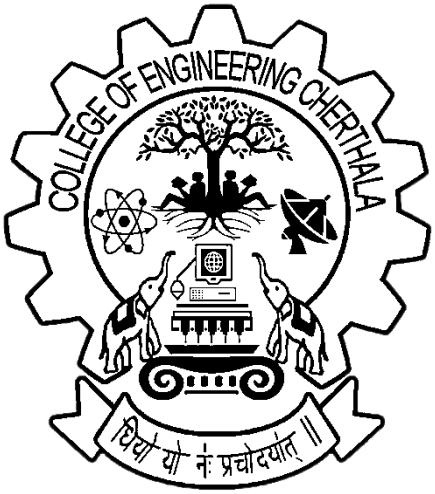
COLLEGE OF ENGINEERING CHERTHALA

LAB RECORD

20MCA136 – NETWORKING & SYSTEM ADMINISTRATION LAB



CERTIFICATE

This is certified to be bonafide works of Mr./Ms. ……..……., In the class ……...., Reg. No. ……., of College of Engineering Cherthala, during the academic year 2022-23.

Teacher In Charge **External Examiner**

Internal Examiner

INDEX

20	Shell script program to convert Decimal to binary	95	07/06/2023	
21	Shell script program to Reverse a number	97	15/06/2023	
22	Shell script program to find Average	99	15/06/2023	

Experiment No.1

Familiarisation with Computer Hardware

Aim: Familiarization of various Computer Hardware Components, ports/interfaces and related cables.

Materials required: -

Computer Hardware Components, Motherboard, CPU, RAM, power supply, etc and different ports/interfaces and related cables.

Theory: -

The computer is an electronic device that takes the input information from the input device and generates the output information and it will be displayed on the output. It enables arithmetic computations, data processing, information management (storage) and knowledge reasoning in an efficient manner.

The word computer is derived from the word compute which means „to calculate . So,“ a computer is generally considered to be the calculating device that performs operations at very faster rates.

1. Cabinet:

a. It is used to install all hardware devices like (motherboard, SMPS, HDD, CD ROM,FDD)
b. It has Start, Restart Button, LED s,Audio and USB Connecters are available on the front side.

2. Motherboard

A motherboard is the main printed circuit board in general-purpose computers and other expandable systems. It holds and allows communication between many of the crucial electronic components of a system, such as the central processing unit and memory, and provides connectors for other peripherals.

3. CPU

A central processing unit, also called a central processor, main processor or just processor, is the electronic circuitry that executes instructions comprising a computer program. The CPU performs basic arithmetic, logic, controlling, and input/output operations specified by the instructions in the program.

4. RAM

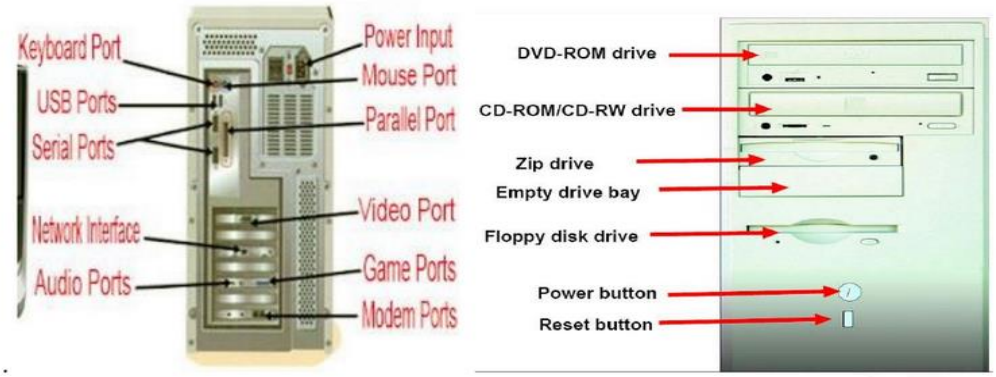
Random-access memory is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code.

5. Graphics Processing Unit (GPU)

A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device . GPUs are used in embedded systems, mobile phones , personal computers, workstations, and game consoles.

6. Computer Interface Cable & Ports

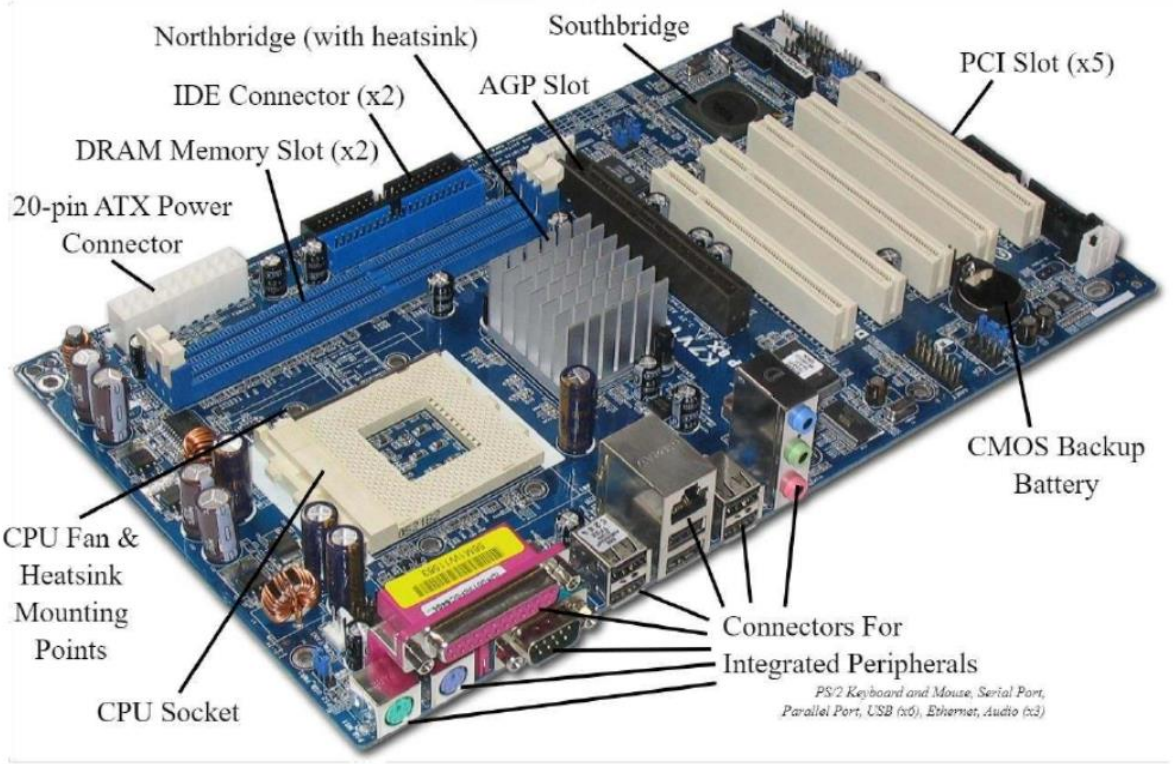
External devices are connected to a computer using cables and ports. Ports are slots on the motherboard into which a cable of an external device is plugged. Computer cables are used to connect monitors, keyboards, printers, hard drives, and other peripherals to computers. These are mainly VGA Cable, D-sub cable, analog video cable, DVI Cable, PS/2 Cable, Ethernet Cable, 3.5mm Audio Cable, USB Cable, Computer Power Cord


Keyboard Port
USB Ports
Serial Ports
Network Interface
Audio Ports
Power Input
Mouse Port
Parallel Port
Video Port
Game Ports
Modem Ports
DVD-ROM drive
CD-ROM/CD-RW drive
Zip drive
Empty drive bay
Floppy disk drive
Power button
Reset button


Ethernet
Mini USB
HDMI
USB Type A
Micro USB
USB Type B
USB Type C
USB 3.0


S-Video
VGA
HDMI
RCA/ Composite
DVI


COMPUTER CABLES
USB TO MICRO
USB TO PRINTER
USB TO USB
VGA TO VGA
DISPLAY TO VGA
MINI DISPLAY TO HDMI
USB TO VGA
VGA TO HDMI
MINI DISPLAY TO VGA


Northbridge (with heatsink)
Southbridge
PCI Slot (x5)
AGP Slot
IDE Connector (x2)
DRAM Memory Slot (x2)
20-pin ATX Power Connector
CMOS Backup Battery
CPU Fan & Heatsink Mounting Points
Connectors For Integrated Peripherals
PS/2 Keyboard and Mouse, Serial Port, Parallel Port, USB (x6), Ethernet, Audio (x3)
CPU Socket

Experiment No.2

Familiarisation of OS installation

Aim: Familiarisation of OS installation

1. windows

2. linux

Familiarization of OS installation - Windows and Linux

Materials required:

1. A computer which needs an OS.

2. A boot CD or DVD (or bootable USB thumbstick) of your chosen OS.

3. Drivers for all your hardware.

4. The motherboard chipset, if there is a special driver for it.

5. RAID card or other drive controller card. Some motherboards will have this built-in.

6. Network adapter.

7. Sound card.

8. Video card.

9. A good antivirus package

10. A suitable archiver. for Windows, you should consider 7-Zip or AlZip (and RARzilla for later). Linux and OSX will include gzip.

Theory:

An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer. The application programs make use of the operating system by making requests for services through a defined application program interface (API). In addition, users can interact directly with the operating system through a user interface, such as a command-line interface (CLI) or a graphical UI (GUI).

Installing Windows:

Each version of Microsoft Windows is installed on a computer using similar steps. While there are steps in the installation process that differ between versions of Windows, the following general steps and guidelines help you install Windows on your computer.

The steps below are for all recent versions of Windows, including Windows 98, Windows ME, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10, and Windows 11. These steps even work for earlier versions (e.g., Windows 95) as long as you use the disc version. The floppy diskette version is similar, but it requires additional steps.

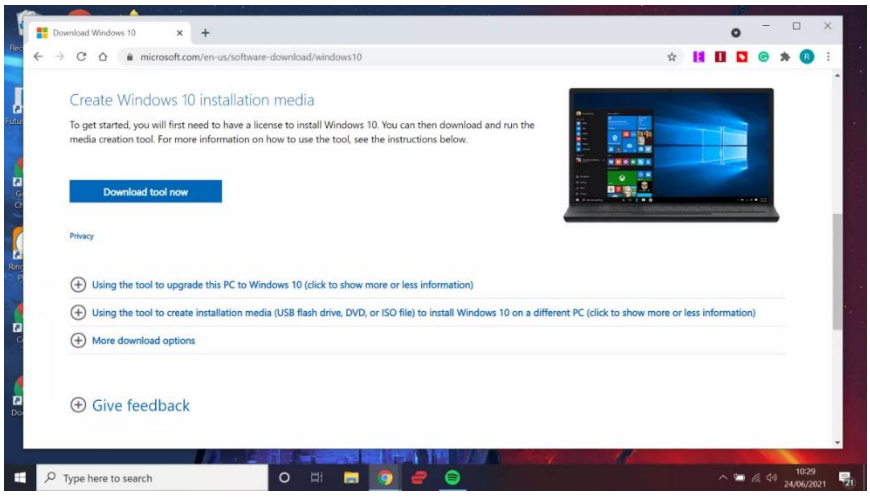
Check Hardware Compatibility:

Before installing or upgrading Windows on your computer, check the hardware in the computer to make sure it's compatible with that version of Windows. Microsoft provides a Windows Compatible Products List for checking if the hardware in your computer is compatible with the chosen version of Windows.

If one or more pieces of hardware is not compatible with the chosen Windows version, we recommend replacing that hardware with compatible hardware or purchasing a new computer. Having compatible hardware in your computer helps ensure the Windows install or upgrade process is successful.

Installing Windows:

To start the Windows install or upgrade process, you need to configure your computer to boot from a CD or DVD before booting to the hard drive. Changing the boot process forces the computer to look for the Windows installation disc before booting from the hard drive.



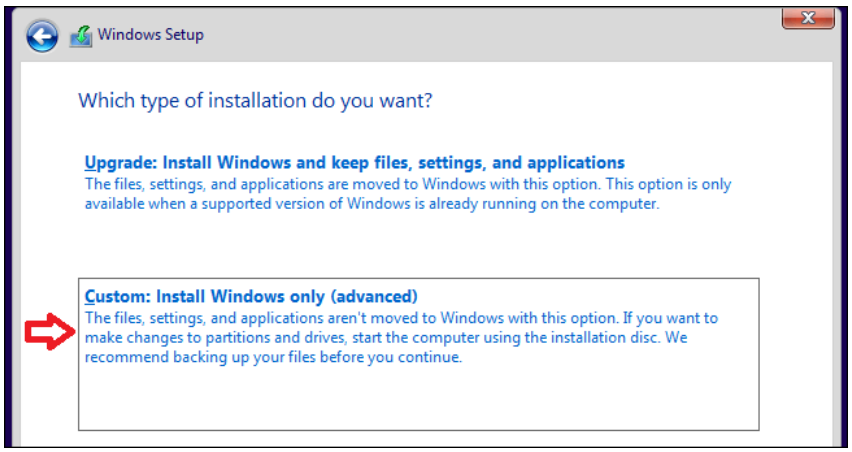
1. Open the CMOS setup.

2. Change the computer's boot order. Set the CD, DVD, or disc drive as the first boot device if you are trying to boot from a disc. Or, set the first boot device to your USB drive if you're trying to boot from a USB thumb drive. If the drive is not shown, keep the disc inserted and reboot the computer. With the disc in the drive, BIOS (basic input/output system) should recognize and include it in the list.

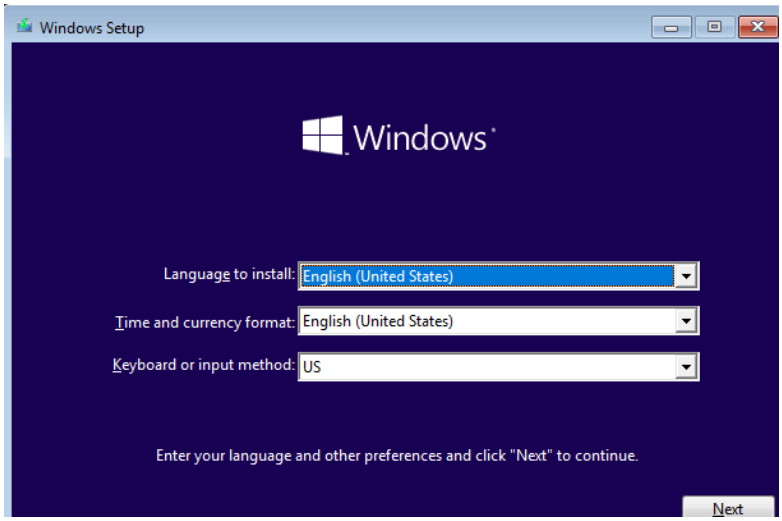
3. Save the settings change and exit BIOS.

Once you have updated the boot order, you can begin the Windows installation process.

4. Place the Windows disc in the CD/DVD drive or USB thumb drive into the back of the computer.
5. Turn on or restart the computer. As the computer starts up, it should detect the installation disc or drive and show a message similar to *Press any key to boot from CD*. Press any key on the keyboard to have the computer boot from the Windows disc or drive.
6. After the Windows install begins, there are several prompts that you need to answer. Select either **Yes** or the appropriate option to install Windows.



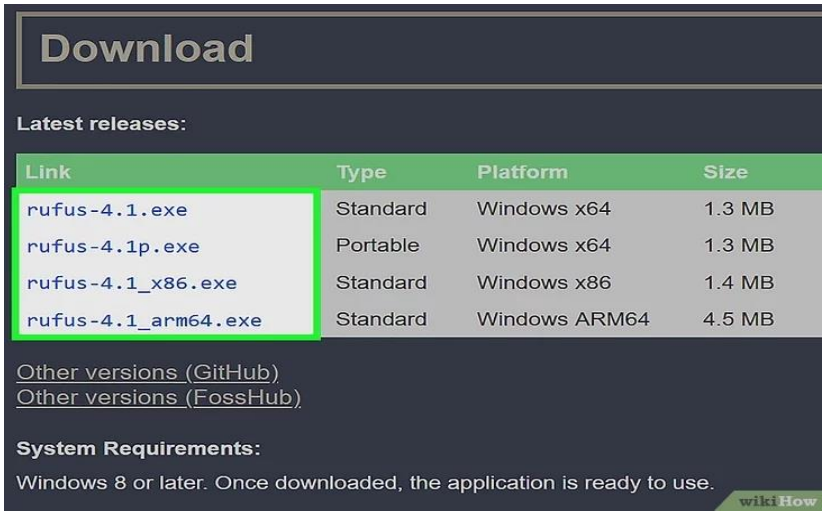
7. When asked which partition to install Windows onto, select the main partition, usually the C: drive or one labeled "Unallocated partition". If upgrading Windows, select the existing installation of Windows on the hard drive.
8. You may be asked if you want to erase all contents on the hard drive, then install Windows. We recommend you choose this option, as it also formats the hard drive to allow the Windows operating system to be installed.
9. The computer may need to restart several times during the Windows install process. The restarts are normal and if prompted to restart, select the **Yes** option.
10. When the install process is nearly complete, the Windows configuration option screens are shown. On these screens, you may be asked to select the time zone you live in, your preferred language, and the account's name you use to access Windows. Select the appropriate options and enter the appropriate information on each configuration screen.



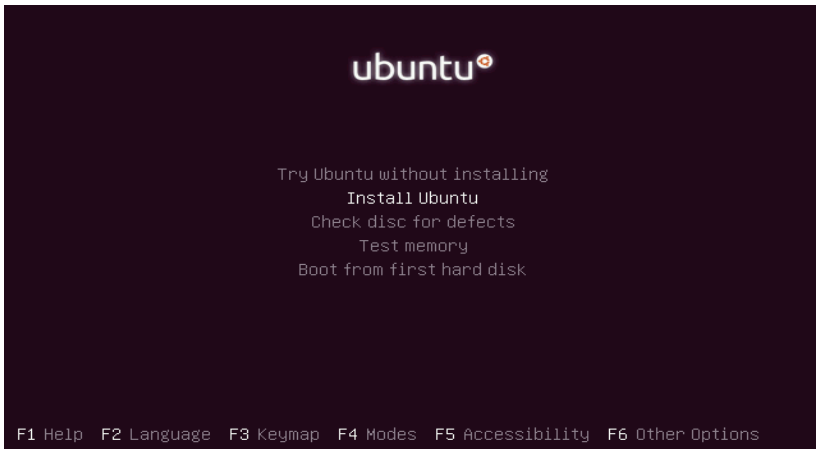
The Windows install process is completed when the computer prompts you to log in or when it loads into Windows.

Installing linux:

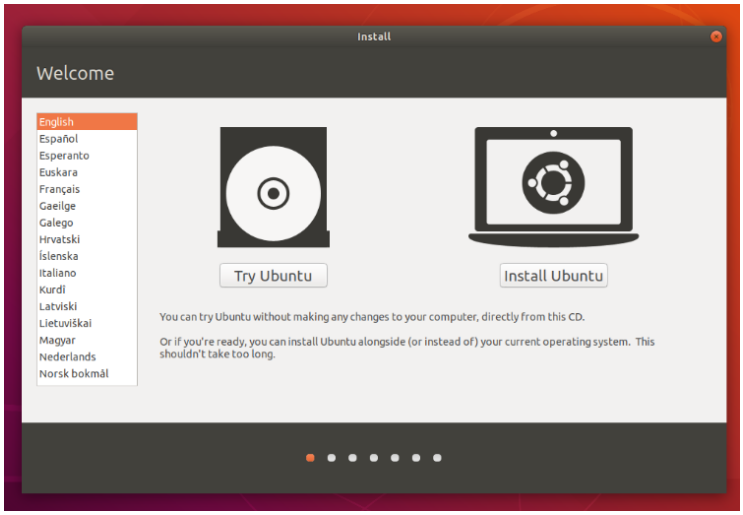
1. Choose a Linux Distribution:
    - There are many Linux distributions available, such as Ubuntu, Fedora, Debian, CentOS, and more. Choose one that suits your needs and download the installation ISO file from the official website of the distribution.



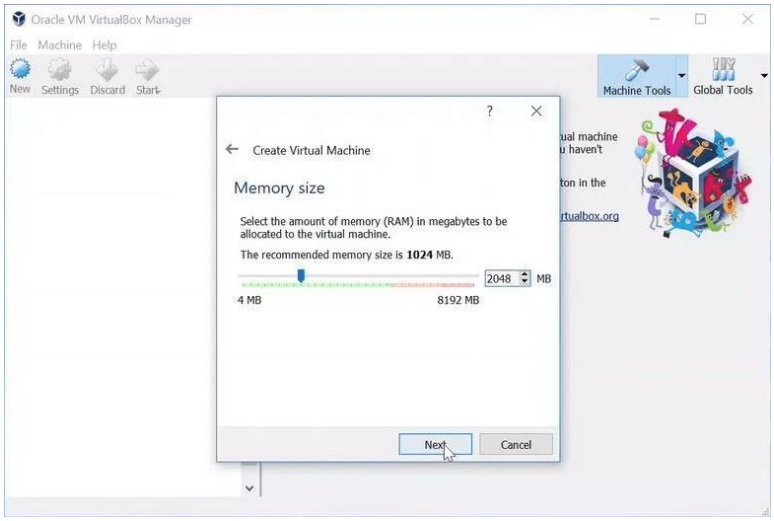
2. Create a Bootable USB Drive:
    - Use a tool like "Rufus" (on Windows) or "Etcher" (available for Windows, macOS, and Linux) to create a bootable USB drive from the downloaded ISO file.
3. Boot from the USB Drive:
    - Insert the bootable USB drive into your computer and restart it.
    - Access your computer's BIOS/UEFI settings to set the USB drive as the primary boot device. This process varies depending on your computer's manufacturer; check your computer's manual for instructions.
4. Start the Installation:
    - Once your computer boots from the USB drive, you'll see the Linux distribution's installation screen. Select "Install" to begin the installation process.



5. Choose Language and Keyboard Layout:
    - Select your preferred language and keyboard layout.



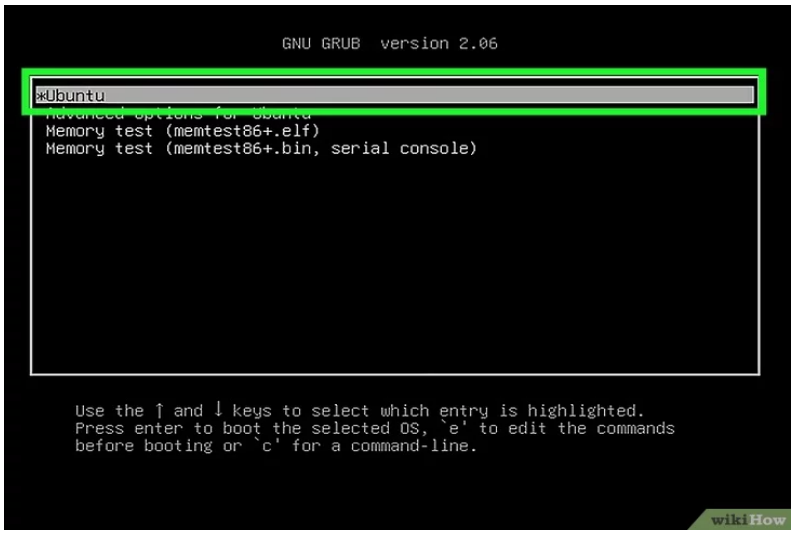
6. Select Installation Type:
    - You'll be prompted to choose an installation type. Options may include:
        - Erase Disk and Install: This option will erase everything on your disk and install Linux.
        - Install Alongside Existing OS: If you want to keep your current OS, you can install Linux alongside it.
        - Something Else: This option allows you to manually partition your disk.
7. Partitioning:
    - Depending on your choice in the previous step, you'll either set up partitions manually or let the installer handle it automatically.



8. Create a User Account:
    - You'll be asked to create a username and password for your Linux user account.

9. Install GRUB (Boot Loader):
    - If prompted, install the GRUB boot loader to the Master Boot Record (MBR) or EFI partition of your disk.

10. Complete the Installation:
    - The installer will copy files and packages to your system. Once it's finished, you'll be prompted to remove the installation media (USB drive) and press Enter to restart your computer.
11. Login to Your New Linux System:
    - After the reboot, you'll see the login screen. Enter the username and password you created during the installation.



12. Post-Installation Configuration:
    - Depending on the Linux distribution, you may need to perform post-installation configurations, such as installing updates, drivers, and additional software.

Experiment No.3

Study of a terminal based text editor such as Vim

a) Cursor operations

To start vim

```
user@gourisankaram:~$ vim
```

```
                        VIM - Vi IMproved

                         version 8.0.3741
                     by Bram Moolenaar et al.
          Modified by pkg-vim-maintainers@lists.alioth.debian.org
                Vim is open source and freely distributable

                       Become a registered Vim user!
             type  :help register<Enter>   for information

             type  :q<Enter>               to exit
             type  :help<Enter>  or  <F1>  for on-line help
             type  :help version8<Enter>   for version info
                                                          0,0-1         All
```

To create new file

:edit sample.txt

```
:edit sample.txt
```

To insert text

Press i

To save

:w

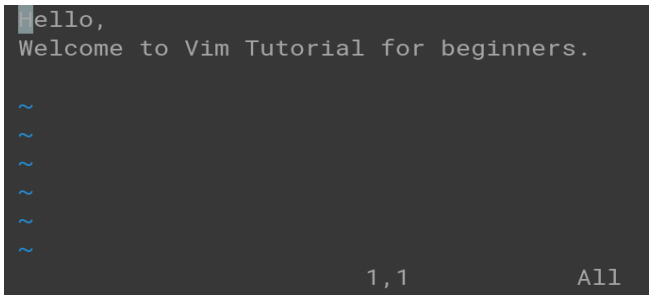
To quit

:q

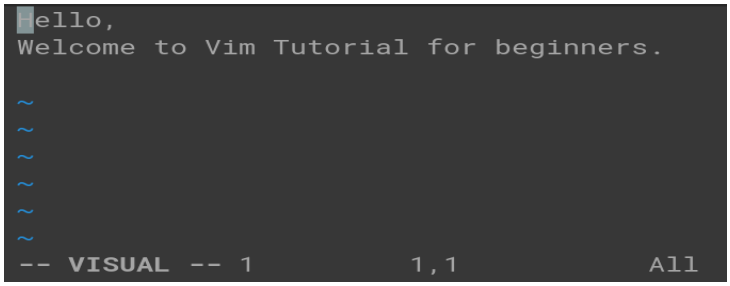
b) Manipulate text

To open sample.txt already created

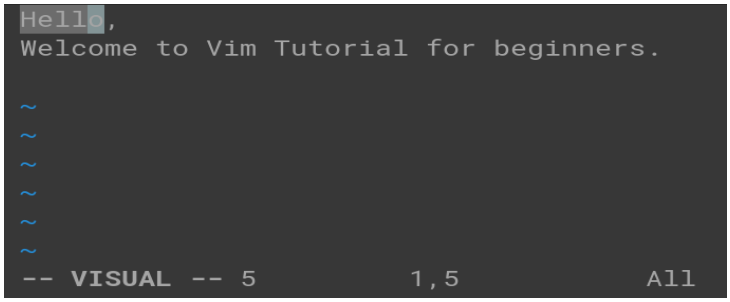
vim sample.txt



Enter Visual mode by pressing the v key on the keyboard

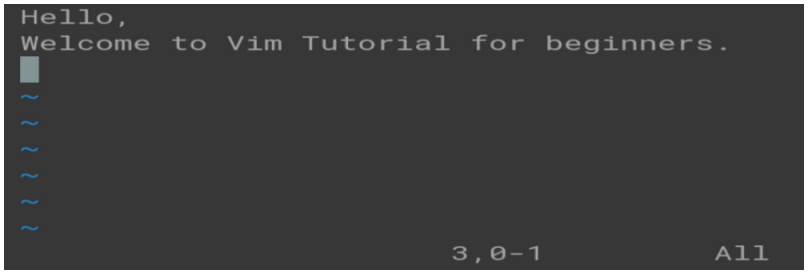


Moving the cursor to the letter “o”

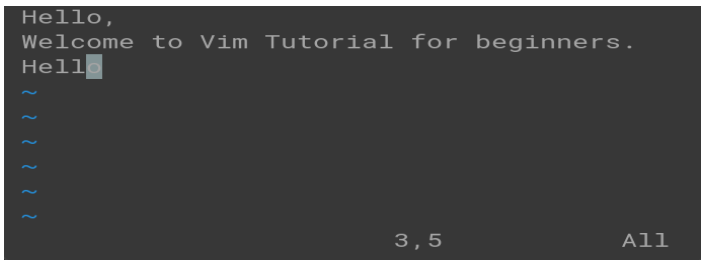


Press y to copy the text or press d to cut the text

Move cursor where to paste

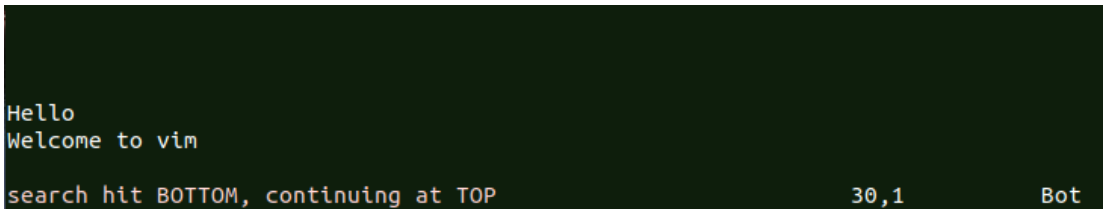


Press p to paste the text



c) Search for patterns

:/Hello

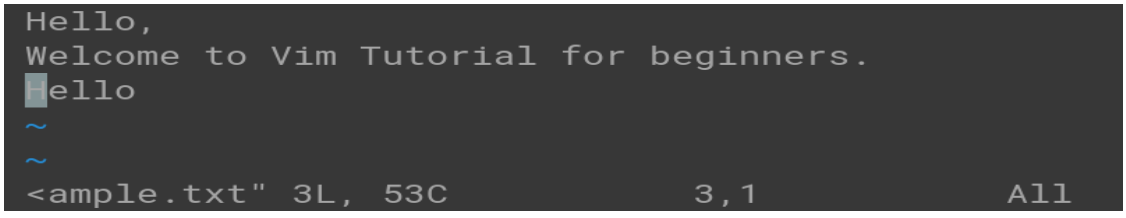


d) Global search and replace

:[range]s/{pattern}/{string}/[flags]

- [range] indicates that you can pass the range of lines. Pass % to find and replace in all lines. The range is separated by a comma. To find and replace between lines 5 to 10, pass 5,10. Use . to represent the current line and $ the last line of the file.
- {pattern} indicates the pattern to find the text. You can pass regex patterns here.
- {string} is the string to replace in the found text.
- [flags] indicates if you wish to pass any additional flags (for example, the flag c is passed to confirm before replacing the text). By default, this does a case-sensitive search. You can change it to do a case-insensitive search by passing a i flag.

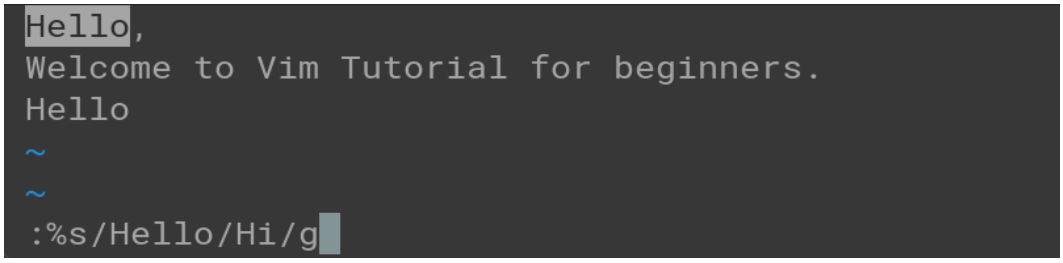
sample.txt file content

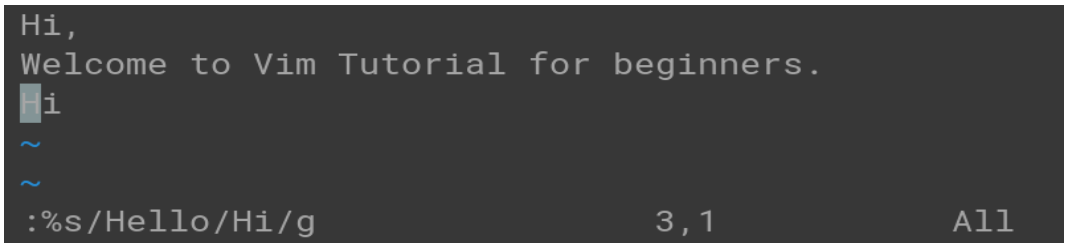


Replace “Hello” with “Hi”

:%s/Hello/Hi/g

- %s indicates replacing the content in the entire file
- Hello is the search text
- Hi is the text to replace
- g indicates making the change globally





Experiment No.4

Basic Linux commands, familiarity with following commands/operationse expected

1. man
2. Is, echo, read
3. more, less, cat,
4. cd, mkdir, pwd, find
5. mv, cp, rm,tar
6. wc, cut, paste
7. head, tail, grep, expr
8 chmod, chown
9. df,top, ps
10. useradd, usermod, userdel, passwd

1. man -Access manual for all Linux commands man <command name>

```
user@gourisankaram:~$ man ls
```

```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the  FILEs  (the  current  directory by
       default).  Sort entries alphabetically if none of  -cftuvSUX  nor
       --sort is specified.

       Mandatory  arguments  to  long  options  are  mandatory for short
       options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              scale  sizes  by  SIZE  before  printing  them;  e.g.,
              '--block-size=M' prints sizes in units of 1,048,576 bytes;
              see SIZE format below

       -B, --ignore-backups
              do not list implied entries ending with ~
Manual page ls(1) line 1 (press h for help or q to quit)
```

2. ls - Displays information about files in the current directory.

```
user@gourisankaram:~$ ls
 ceremony.svg            pavi        Reset_Settings
 Desktop                 pg14.sh     sample.txt
 Documents               pg16.sh    'siva 1.py'
 Downloads               pg17.sh    'siva .py'
 examples.desktop        pg18.sh    'Synfig Animation 1'
 isb.cfg                 pg19.sh     Templates
 itsuserguide.desktop    pg20.sh     TuxPaint-Pictures
 KeyboardShortcuts.txt   pg21.sh     VCD_Copy
 lios                    pg22.sh     Videos
 message.txt             Pictures
 Music                   Public
```

3. echo - Display active processes on the terminal

```
user@gourisankaram:~$ echo Hello
Hello
```

4. read - Read input from the user in a shell script

```
cec@cec-ThinkCentre-M83:~$ read
hello
```

5. more – view text files

```
cec@cec-ThinkCentre-M83:~$ mkdir mca
cec@cec-ThinkCentre-M83:~$ more mca

*** mca: directory ***
```

6. less – view text files

```
cec@cec-ThinkCentre-M83:~$ less mca
mca is a directory
```

7. cat - Display file contents on terminal

```
user@gourisankaram:~/mca$ gedit file1
user@gourisankaram:~/mca$ cat file1
Hello
how are you
```

8. cd - To navigate between different folders

```
user@gourisankaram:~$ mkdir mca
user@gourisankaram:~$ cd mca
user@gourisankaram:~/mca$
```

9. mkdir - Creates a directory.

```
user@gourisankaram:~/mca$ mkdir student
user@gourisankaram:~/mca$
```

10. Pwd - Displays the current working directory

```
cec@cec-ThinkCentre-M83:~/mca$ pwd
/home/cec/mca
```

11. find - command to search for files within a specific directory and perform subsequent operations.

```
user@gourisankaram:~/mca$ find file1
file1
user@gourisankaram:~/mca$
```

12. mv - Rename and Replace the files

```
user@gourisankaram:~/mca$ mv file1 student
user@gourisankaram:~/mca$ ls
student
```

13. cp - Moves files from one directory to another

```
user@gourisankaram:~/mca/student$ ls
file1
user@gourisankaram:~/mca/student$ cp file1 mca
```

14. rm - The rm command is used to delete files within a directory.

```
user@gourisankaram:~/mca/student$ rm file1
user@gourisankaram:~/mca/student$ ls
mca
```

15. tar – The tar command archives multiple files into a tar file

```
user@gourisankaram:~/mca$ tar -tvf myarchive.tar
```

16. wc - Check the lines, word count, and characters in a file using different options

```
user@gourisankaram:~/mca$ wc -l file1
1 file1
user@gourisankaram:~/mca$
```

17. cut

```
user@gourisankaram:~/mca$ cut -f1 file1
Hello
```

18. paste

```
user@gourisankaram:~/mca$ gedit file2
user@gourisankaram:~/mca$ paste file1 file2
Hello   welcome
```

19. head – The head command allows you to view the first ten lines of a text.

```
user@gourisankaram:~/mca$ head file2
welcome
```

20. Tail – The tail command displays the last ten lines of a file.

```
user@gourisankaram:~/mca$ gedit file2
user@gourisankaram:~/mca$ tail file2
welcome
You are here
```

21. grep - Search for a specific string in an output

```
user@gourisankaram:~/mca$ grep "You" file2
You are here
```

22. expr –

```
user@gourisankaram:~/mca$ expr a
a
```

23. chmod - **chmod** is a common command that modifies a file or directory's read, write, and execute permissions

```
user@gourisankaram:~/mca$ chmod u+rw file2
user@gourisankaram:~/mca$
```

24. chown - The **chown command** lets you change the ownership of a file, directory, or symbolic link to a specified username.

```
user@gourisankaram:~/mca$ chown guest file2
chown: invalid user: ‘guest’
```

25. Df - Use the **df command** to report the system's disk space usage, shown in percentage and kilobyte (KB)

```
user@gourisankaram:~/mca$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           384M  3.1M  381M   1% /run
/dev/sda5        36G   15G   20G  44% /
tmpfs           1.9G     0  1.9G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/loop0      128K  128K     0 100% /snap/bare/5
/dev/loop3       92M   92M     0 100% /snap/gtk-common-themes/1535
/dev/loop6       66M   66M     0 100% /snap/gtk-common-themes/1519
/dev/loop2      141M  141M     0 100% /snap/gnome-3-26-1604/104
/dev/loop1      119M  119M     0 100% /snap/core/15511
/dev/loop4      141M  141M     0 100% /snap/gnome-3-26-1604/111
/dev/loop5      106M  106M     0 100% /snap/core/15925
tmpfs           384M  160K  384M   1% /run/user/1000
```

26. top - The top **command** in Linux Terminal will display all the running processes and a dynamic real-time view of the current system. It sums up the resource utilization, from CPU to memory usage.

```
user@gourisankaram:~/mca$ top

top - 00:05:18 up  4:04,  1 user,  load average: 0.29, 0.47, 0.35
Tasks: 247 total,   1 running, 200 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.5 us,  0.7 sy,  0.0 ni, 97.7 id,  0.2 wa,  0.0 hi,  0.0 si,  0
KiB Mem :  3930676 total,   259512 free,  2218048 used,  1453116 buff/cach
KiB Swap:  2926588 total,  2806416 free,   120172 used.   919388 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+
  932 root      20   0 1106480  81848  59588 S   1.7  2.1   3:54.20
 3196 user      20   0 3138652 388672  99112 S   1.7  9.9   6:36.68
```

27. ps - The process status or ps command produces a snapshot of all running processes in your system

```
user@gourisankaram:~/mca$ ps
  PID TTY          TIME CMD
10218 pts/0    00:00:00 bash
12004 pts/0    00:00:00 ps
user@gourisankaram:~/mca$
```

28. useradd –

```
user@gourisankaram:~/mca$ sudo useradd guest
[sudo] password for user:
```

29. usermod –

```
user@gourisankaram:~/mca$ sudo usermod -l Guest guest
```

30. userdel –

```
user@gourisankaram:~/mca$ sudo userdel Guest
```

31. psswd -

```
user@gourisankaram:~/mca$ sudo passwd Guest
passwd: user 'Guest' does not exist
```

Output:

```
user@gourisankaram:~$ bash directory.sh
Enter folder name
mca
mca already exists
user@gourisankaram:~$ bash directory.sh
Enter folder name
dir1
Directory created
user@gourisankaram:~/dir1$ bash create.sh

File content :
Hello
How are you?
^Z
[1]+  Stopped                 bash create.sh
user@gourisankaram:~/dir1$ ls
create.sh  myFile
```

a.display the current directory

```
user@gourisankaram:~/dir1$ pwd
/home/user/dir1
```

b. listing files and folders in that directory

```
user@gourisankaram:~$ ls dir1
create.sh  myFile
```

c.list the content of directorty in an alphabetical order

```
user@gourisankaram:~$ ls -l dir1 | sort
-rw-rw-r-- 1 user user 19 Sep  6 11:33 myFile
-rw-rw-r-- 1 user user 46 Sep  6 11:32 create.sh
total 8
```

d.list the content of directory in reverse order

```
user@gourisankaram:~$ ls -l dir1 | sort -r
total 8
-rw-rw-r-- 1 user user 46 Sep  6 11:32 create.sh
-rw-rw-r-- 1 user user 19 Sep  6 11:33 myFile
```

e. create another directory dir2 move all the files in dirl to dir2

```
user@gourisankaram:~$ mkdir dir2
user@gourisankaram:~$ mv dir1/* dir2/
```

Experiment No.5

Shell Script and linux program questions

5.1. Create a new directory (dirl) and create new two files in that directory

Aim: Create a new directory (dirl) and create new two files in that directory

Algorithm:

1. Create a directory named 'dirl' using the 'mkdir' command.
2. Create a file named 'file1.txt' inside 'dirl' using the 'touch' command.
3. Create another file named 'file2.txt' inside 'dirl' using the 'touch' command.

Program:

```
echo "Enter folder name"
read dir
if [ ! -d $dir ]
then
mkdir $dir
echo "Directory created"
else
echo "$dir already exists"
fi
```

Result: Program executed successfully and output obtained.

Output:

```
user@gourisankaram:~$ cat file
This contain abc
Here we get linux commands
```

```
user@gourisankaram:~$ bash gr1.sh file
This contain abc
```

5.2. Use grep to

a. Display all lines in a file that contain a string "abc"

Aim: Display all lines in a file that contain a string "abc"

Algorithm:

1. Start
2. Accept the filename as input from the user or as a command-line argument.
3. Check if the specified file exists:
    - If the file does not exist, display an error message and exit.
    - If the file exists, proceed to the next step.
4. Open the file for reading.
5. Loop through each line in the file: a. Read the current line. b. Check if the line contains the string "abc":
    - If it does, print the line to the output.
    - If it does not, continue to the next line.
6. Close the file.
7. End

Program:

```
#!/bin/bash

if [ $# -ne 1 ]; then
   echo "Usage: $0 <filename>"
   exit 1
fi

filename="$1"

if [ ! -f "$filename" ]; then
   echo "File not found: $filename"
   exit 1
fi
grep 'abc' "$filename"
```

Result: Program executed successfully and output obtained.

Output:

```
user@gourisankaram:~$ cat file
This contain abc
Here we get linux commands
```

```
user@gourisankaram:~$ gedit gr2.sh
user@gourisankaram:~$ bash gr2.sh file
Here we get linux commands
```

b. Display all the lines in a file that does not contain the string "abc"

Aim: Display all the lines in a file that does not contain the string "abc"

Algorithm:

1. Start
2. Accept the filename as input from the user or as a command-line argument.
3. Check if the specified file exists:
    - If the file does not exist, display an error message and exit.
    - If the file exists, proceed to the next step.
4. Open the file for reading.
5. Loop through each line in the file: a. Read the current line. b. Check if the line does not contain the string "abc":
    - If it does not contain "abc," print the line to the output.
    - If it contains "abc," continue to the next line.
6. Close the file.
7. End

Program:

```
#!/bin/bash

if [ $# -ne 1 ]; then
   echo "Usage: $0 <filename>"
   exit 1
fi

filename="$1"

if [ ! -f "$filename" ]; then
   echo "File not found: $filename"
   exit 1
fi

grep -v 'abc' "$filename"
```

Result: Program executed successfully and output obtained.

Output:

```
user@gourisankaram:~/mca$ bash pg3.sh
“Enter the first file name:”
file1
“Enter the second file name:”
file2
file1 file2 differ: byte 1, line 1
pg3.sh: line 6: [-e: command not found
“file1 in not existed”
```

5.3. Write shell script program which receives 2 file names as arguments

a. check wheather the file contents are same or not

b. if they are same delete the second file

Aim: Check wheather the file contents are same or not ,if they are same delete the second file

Algorithm:

1. Display the message "Enter the first file name:"
2. Read the input from the user and store it in the variable file1.
3. Display the message "Enter the second file name:"
4. Read the input from the user and store it in the variable file2.
5. Compare the contents of file1 and file2 using the cmp command.
    - If the files are identical, proceed to the next step.
    - If the files are not identical, skip the next step and continue to the if [-e $file1] block.
6. If the files are identical, remove file2 using the rm command.
7. Check if file1 exists:
    - If file1 exists, proceed to the next step.
    - If file1 does not exist, display the message "$file1 is not existed" and exit.
8. Check if file2 does not exist:
    - If file2 does not exist, display the message "The two file contents are the same. So, $file2 is deleted."
    - If file2 exists, display the message "The two file contents are not the same, and $file2 is not deleted."

Program:

```
echo “Enter the first file name:”
read file1
echo “Enter the second file name:”
read file2
cmp $file1 $file2 && rm file2
if [-e $file1]
then
        if [! -e $file2]
        then
                echo “The 2 file contents are same. So $file2 is deleted”
        else
                echo “The 2 file contents are not same and $file2 is not deleted”
        fi
else
echo “$file1 in not existed”
fi
```

Result: Program executed successfully and output obtained.

Output:

Enter the filename:

f1.txt

Roll_No.	Mark	Name
1	85	Anu
2	90	Meera
3	68	Sai
4	75	Achu

5.4.Create a file fl which contain data

roll no	name	mark
1	anu	85
2	meera	90
3	sai	68
4	achu	75

a. Use the cut and paste command to swap fields 2 and 3 of file

Aim: Use the cut and paste command to swap fields 2 and 3 of file

Algorithm:

1. Display the message "Enter the file name:"
2. Read the input from the user and store it in the variable file.
3. Use the cut command to perform the following operations on the contents of file:
    - Extract fields 1 and 3 from file.
    - Redirect the output to a temporary file named "cat."
4. Use the cut command again to extract field 2 from file.
    - Pipe the output to the paste command, combining it with the contents of the "cat" temporary file.
5. Redirect the combined output to a file named "result."
6. The script is now complete.

Program:

```
echo “Enter the file name:”
read file
cut -f1,3 $file > cat && cut -f2 $file | paste cat -> result
```

Result: Program executed successfully and output obtained.

Output:

```
Enter the filename:
f1.txt

No. of lines to print:
2

Roll_No.    Mark    Name
1           85      Anu
```

b. Print first 2 rows of the file fl

Aim: Print first 2 rows of the file fl

Algorithm:

1. Display the message "Enter the filename:"
2. Read the input from the user and store it in the variable file.
3. Display the message "No. of lines to print:"
4. Read the input from the user and store it in the variable no.
5. Use the head command to display the first n lines (where n is the value stored in the variable no) of the file specified by file.
6. The script is complete.

Program:

```
echo “Enter the filename:”

read file

echo “No. of lines to print:”

read no

head -n $no $file
```

Result: Program executed successfully and output obtained.

Output:

Enter the filename:

f1.txt

Line number to print:

3

2 90 Meera

c. Print the content of 3rd row only of the file fl

Aim: Print the content of 3rd row only of the file fl

Algorithm:

1. Display the message "Enter the filename:"
2. Read the input from the user and store it in the variable file.
3. Display the message "Line number to print:"
4. Read the input from the user and store it in the variable num.
5. Use the sed command to print the line specified by the value stored in the variable num from the file specified by file.
6. The script is complete.

Program:

```
echo “Enter the filename:”

read file

echo “Line number to print:”

read num

sed “${num}q;d” $file
```

Result: Program executed successfully and output obtained.

Output:

```
Enter the filename:
f1.txt
Column name to print:
3
Name
Anu
Meera
Sai
Achu
```

d. Print the names only from the file fl

Aim: Print the names only from the file fl

Algorithm:

1. Display the message "Enter the filename:"
2. Read the input from the user and store it in the variable file.
3. Display the message "Column number to print:"
4. Read the input from the user and store it in the variable num.
5. Use the awk command to print the column specified by the value stored in the variable num from the file specified by file.
6. The script is complete.

Program:

```
echo “Enter the filename:”

read file

echo “Column number to print:”

read num

awk “{print $ ${num}}”$file
```

Result: Program executed successfully and output obtained.

Output:

Enter the filename:

f1.txt

Number of lines: 5

Number of words: 15

Number of characters: 42

e. Count the number of lines,words and character from the file f1

Aim: Count the number of lines,words and character from the file f1

Algorithm:

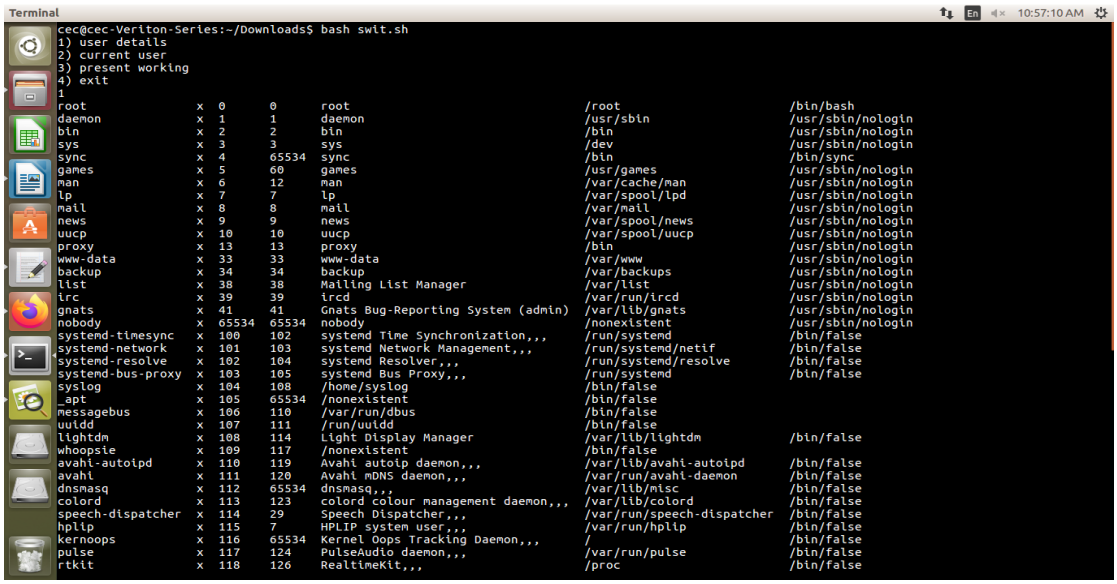
1. Display the message "Enter the filename:"
2. Read the input from the user and store it in the variable file.
3. Use the wc command to count the number of lines, words, and characters in the file specified by file. Store the results in variables nol (number of lines), now (number of words), and noc (number of characters).
4. Display the counts of lines, words, and characters as follows:
    - "Number of lines: <nol>"
    - "Number of words: <now>"
    - "Number of characters: <noc>"
5. The script is complete.

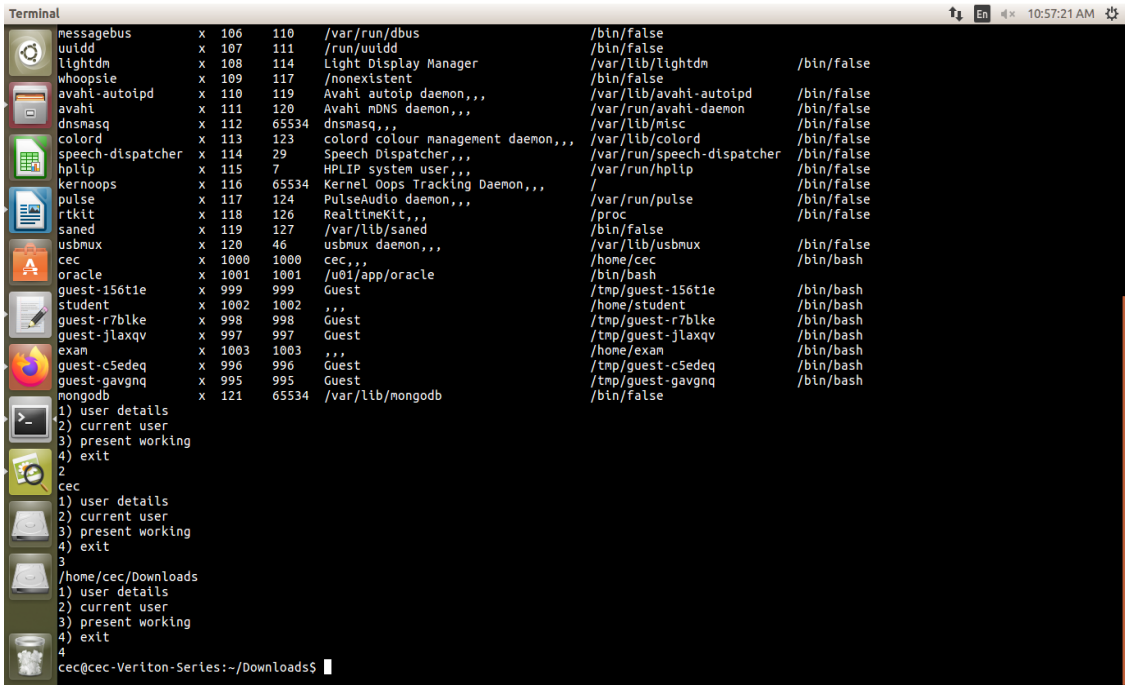
Program:

```
echo “Enter the filename:”

read file

nol =`wc - -lines<$file`

now = `wc - -word<$file`

noc = `wc - -m<$file`

echo “Number of lines: $nol”

echo “Number of words: $now”

echo “Number of characters: $noc”
```

Result: Program executed successfully and output obtained.

Output:


Terminal
10:57:10 AM
cec@cec-Veriton-Series:~/Downloads$ bash swit.sh
1) user details
2) current user
3) present working
4) exit
1
root x 0 0 root /root /bin/bash
daemon x 1 1 daemon /usr/sbin /usr/sbin/nologin
bin x 2 2 bin /bin /usr/sbin/nologin
sys x 3 3 sys /dev /usr/sbin/nologin
sync x 4 65534 sync /bin /bin/sync
games x 5 60 games /usr/games /usr/sbin/nologin
man x 6 12 man /var/cache/man /usr/sbin/nologin
lp x 7 7 lp /var/spool/lpd /usr/sbin/nologin
mail x 8 8 mail /var/mail /usr/sbin/nologin
news x 9 9 news /var/spool/news /usr/sbin/nologin
uucp x 10 10 uucp /var/spool/uucp /usr/sbin/nologin
proxy x 13 13 proxy /bin /usr/sbin/nologin
www-data x 33 33 www-data /var/www /usr/sbin/nologin
backup x 34 34 backup /var/backups /usr/sbin/nologin
list x 38 38 Mailing List Manager /var/list /usr/sbin/nologin
irc x 39 39 ircd /var/run/ircd /usr/sbin/nologin
gnats x 41 41 Gnats Bug-Reporting System (admin) /var/lib/gnats /usr/sbin/nologin
nobody x 65534 65534 nobody /nonexistent /usr/sbin/nologin
systemd-timesync x 100 102 systemd Time Synchronization,,, /run/systemd /bin/false
systemd-network x 101 103 systemd Network Management,,, /run/systemd/netif /bin/false
systemd-resolve x 102 104 systemd Resolver,,, /run/systemd/resolve /bin/false
systemd-bus-proxy x 103 105 systemd Bus Proxy,,, /run/systemd /bin/false
syslog x 104 108 /home/syslog /bin/false
_apt x 105 65534 /nonexistent /bin/false
messagebus x 106 110 /var/run/dbus /bin/false
uuidd x 107 111 /run/uuidd /bin/false
lightdm x 108 114 Light Display Manager /var/lib/lightdm /bin/false
whoopsie x 109 117 /nonexistent /bin/false
avahi-autoipd x 110 119 Avahi autoip daemon,,, /var/lib/avahi-autoipd /bin/false
avahi x 111 120 Avahi mDNS daemon,,, /var/run/avahi-daemon /bin/false
dnsmasq x 112 65534 dnsmasq,,, /var/lib/misc /bin/false
colord x 113 123 colord colour management daemon,,, /var/lib/colord /bin/false
speech-dispatcher x 114 29 Speech Dispatcher,,, /var/run/speech-dispatcher /bin/false
hplip x 115 7 HPLIP system user,,, /var/run/hplip /bin/false
kernoops x 116 65534 Kernel Oops Tracking Daemon,,, / /bin/false
pulse x 117 124 PulseAudio daemon,,, /var/run/pulse /bin/false
rtkit x 118 126 RealtimeKit,,, /proc /bin/false


Terminal
10:57:21 AM
messagebus x 106 110 /var/run/dbus /bin/false
uuidd x 107 111 /run/uuidd /bin/false
lightdm x 108 114 Light Display Manager /var/lib/lightdm /bin/false
whoopsie x 109 117 /nonexistent /bin/false
avahi-autoipd x 110 119 Avahi autoip daemon,,, /var/lib/avahi-autoipd /bin/false
avahi x 111 120 Avahi mDNS daemon,,, /var/run/avahi-daemon /bin/false
dnsmasq x 112 65534 dnsmasq,,, /var/lib/misc /bin/false
colord x 113 123 colord colour management daemon,,, /var/lib/colord /bin/false
speech-dispatcher x 114 29 Speech Dispatcher,,, /var/run/speech-dispatcher /bin/false
hplip x 115 7 HPLIP system user,,, /var/run/hplip /bin/false
kernoops x 116 65534 Kernel Oops Tracking Daemon,,, / /bin/false
pulse x 117 124 PulseAudio daemon,,, /var/run/pulse /bin/false
rtkit x 118 126 RealtimeKit,,, /proc /bin/false
saned x 119 127 /var/lib/saned /bin/false
usbmux x 120 46 usbmux daemon,,, /var/lib/usbmux /bin/false
cec x 1000 1000 cec,,, /home/cec /bin/bash
oracle x 1001 1001 /u01/app/oracle /bin/bash
guest-156t1e x 999 999 Guest /tmp/guest-156t1e /bin/bash
student x 1002 1002 ,,, /home/student /bin/bash
guest-r7blke x 998 998 Guest /tmp/guest-r7blke /bin/bash
guest-jlaxqv x 997 997 Guest /tmp/guest-jlaxqv /bin/bash
exam x 1003 1003 ,,, /home/exam /bin/bash
guest-c5edeq x 996 996 Guest /tmp/guest-c5edeq /bin/bash
guest-gavgnq x 995 995 Guest /tmp/guest-gavgnq /bin/bash
mongodb x 121 65534 /var/lib/mongodb /bin/false
1) user details
2) current user
3) present working
4) exit
2
cec
1) user details
2) current user
3) present working
4) exit
3
/home/cec/Downloads
1) user details
2) current user
3) present working
4) exit
4
cec@cec-Veriton-Series:~/Downloads$

5.5.To print user details.

a) Current user. b) Current working directory.

Algorithm:

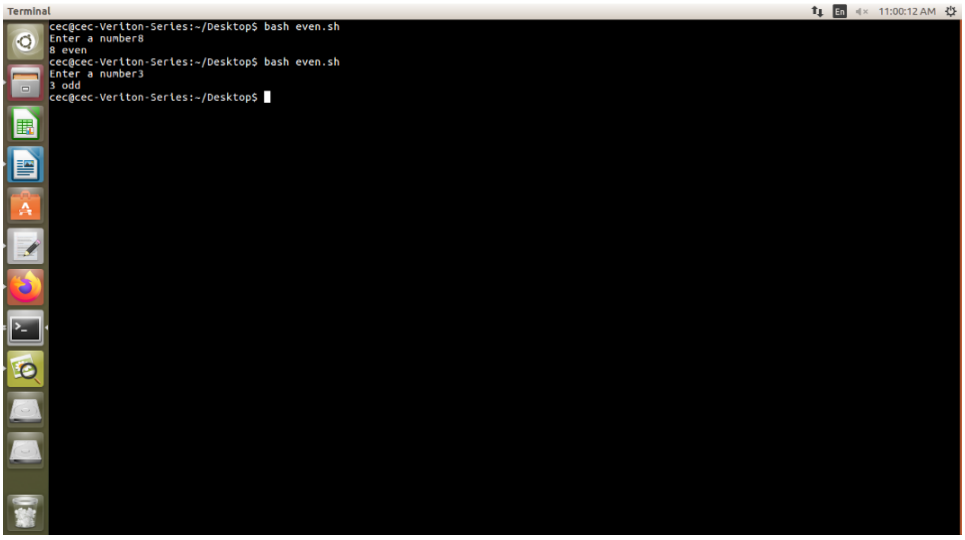
1. Start
2. Initialize a variable `con` with the value 1.
3. Start an infinite `while` loop using `while [ $con -ne 0 ]`.
   1. Display a menu with options:
      1) User details
      2) Current user
      3) Present working directory
      4) Exit
   2. Display "Enter your choice" and read the user's choice into the variable `con`.
   3. Use a `case` statement to handle the different choices:
      Case 1:
        1. Display user details by executing the command `cat /etc/passwd | column -s ":" -t`.
      Case 2:
        1. Display the current user's username by executing the command `whoami`.
      Case 3:
        1. Display the present working directory by executing the command `pwd`.
      Case 4:
        1. Display "Exit".
        2. Exit the script.
      Default:
        1. Display "Invalid choice".
   4. End the `case` statement.
4. End the `while` loop.
5. End

Program:

```
let con=1;
while [ $con -ne 0 ]
do
echo "1) user details"
echo "2) current user"
echo "3) present working"
echo "4) exit"
read con
case $con in
  1) cat /etc/passwd | column -s ":" -t;;
  2) whoami;;
  3) pwd;;
  4) exit;;
 esac
done
```

Result: Program executed successfully and output obtained.

Output:


Terminal
11:00:12 AM
cec@cec-Veriton-Series:~/Desktop$ bash even.sh
Enter a number8
8 even
cec@cec-Veriton-Series:~/Desktop$ bash even.sh
Enter a number3
3 odd
cec@cec-Veriton-Series:~/Desktop$

Experiment No. 6

Even or not

Aim: Write shell script program to check whether a number is even or not

Algorithm:

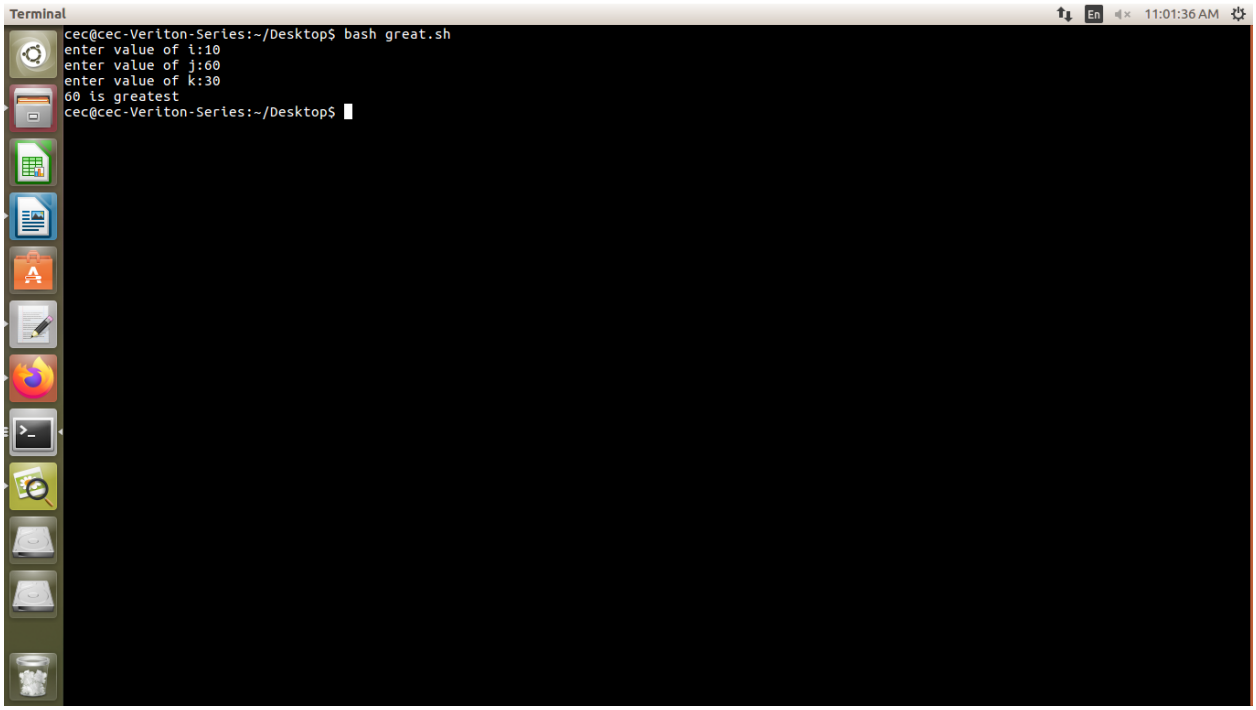
1. Start
2. Display "Enter a number" to prompt the user for input.
3. Read the user input and store it in a variable n.
4. Calculate the remainder when n is divided by 2 and store it in a variable rem.
5. Use an if statement to check if rem is equal to 0:
    - If true, execute the following block of code:
        1. Display "n even" indicating that the number is even.
    - If false, execute the following block of code:
        1. Display "n odd" indicating that the number is odd.
6. End

Program:

```
echo -n "Enter a number"
read n
rem=$((n%2))
if [ $rem -eq 0 ]
then
        echo "$n even"
else
        echo "$n odd"
fi
```

Result: Program executed successfully and output obtained.

Output:


Terminal
11:01:36 AM
cec@cec-Veriton-Series:~/Desktop$ bash great.sh
enter value of i:10
enter value of j:60
enter value of k:30
60 is greatest
cec@cec-Veriton-Series:~/Desktop$

Experiment No. 7

Largest among three numbers

Aim: Write shell script program to print largest among three numbers

Algorithm:

1. Start
2. Prompt the user to enter the value of `i`.
3. Read the user input and store it in a variable `i`.
4. Prompt the user to enter the value of `j`.
5. Read the user input and store it in a variable `j`.
6. Prompt the user to enter the value of `k`.
7. Read the user input and store it in a variable `k`.
8. Use nested `if` statements to compare the values of `i`, `j`, and `k` to find the greatest:
   - If `i` is greater than `j`:
     - If `i` is also greater than `k`, then display "`i` is greatest".
     - Else, display "`k` is greatest".
   - Else (if `i` is not greater than `j`):
     - If `j` is greater than `k`, then display "`j` is greatest".
     - Else, display "`k` is greatest".
9. End

Program:

```
read -p "enter value of i:" i
read -p "enter value of j:" j
read -p "enter value of k:" k
if [ $i -gt $j ]
then
 if [ $i -gt $k ]
 then
  echo "$i is greatest"
 else
  echo "$k is greatest"
 fi
else
 if [ $j -gt $k ]
 then
  echo "$j is greatest"
 else
echo "$k is greatest"
 fi
fi
```

Result: Program executed successfully and output obtained.

Experiment No. 8

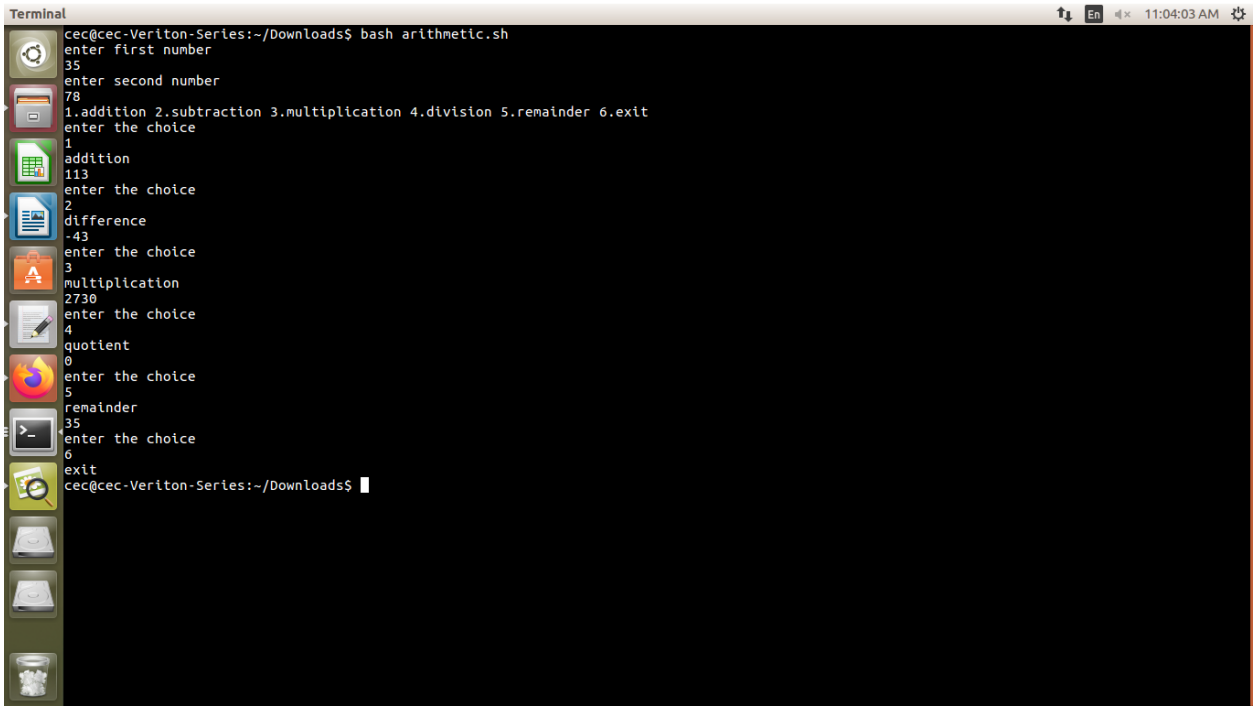
Arithmetic operations

Aim: Write menu driven shell script program to perform all arithmetic operations.

Algorithm:

1. Start
2. Read the user input and store it in a variable `a`.
3. Display "Enter second number" to prompt the user for input.
4. Read the user input and store it in a variable `b`.
5. Display a menu with options:
   1) Addition
   2) Subtraction
   3) Multiplication
   4) Division
   5) Remainder
   6) Exit
6. Start an infinite `while` loop using `while :`.
   1. Display "Enter your choice" and read the user's choice into the variable `choice`.
   2. Use a `case` statement to handle the different choices:
      Case 1:
        1. Display "Addition".
        2. Perform addition using `expr $a + $b`.
      Case 2:
        1. Display "Subtraction".
        2. Perform subtraction using `expr $a - $b`.
      Case 3:
        1. Display "Multiplication".
        2. Perform multiplication using `expr $a \* $b`.
      Case 4:
        1. Display "Division".
        2. Perform division using `expr $a / $b`.
      Case 5:
        1. Display "Remainder".
        2. Perform remainder calculation using `expr $a % $b`.
      Case 6:
        1. Display "Exit".
        2. Break out of the infinite loop.
      Default:
        1. Display "Invalid choice".
   3. End the `case` statement.
7. End

Output:


Terminal
11:04:03 AM
cec@cec-Veriton-Series:~/Downloads$ bash arithmetic.sh
enter first number
35
enter second number
78
1.addition 2.subtraction 3.multiplication 4.division 5.remainder 6.exit
enter the choice
1
addition
113
enter the choice
2
difference
-43
enter the choice
3
multiplication
2730
enter the choice
4
quotient
0
enter the choice
5
remainder
35
enter the choice
6
exit
cec@cec-Veriton-Series:~/Downloads$

Program:

```
echo "enter first number"
read a
echo "enter second number"
read b
echo "1.addition 2.subtraction 3.multiplication 4.division 5.remainder 6.exit"
while :
do
        echo "enter the choice"
        read choice
        case $choice in
        1)      echo "addition"
                expr $a + $b
                ;;
        2)      echo "difference"
                expr $a - $b
                ;;
        3)      echo "multiplication"
                expr $a \* $b
                ;;
        4)      echo "quotient"
                expr $a / $b
                ;;
        5)      echo "remainder"
                expr $a % $b
                ;;
        6)      echo "exit"
                break
                ;;
        *)      echo "invalid choice"
                ;;
        esac
done
```

Result: Program executed successfully and output obtained.

Experiment No. 9

Sum of natural numbers

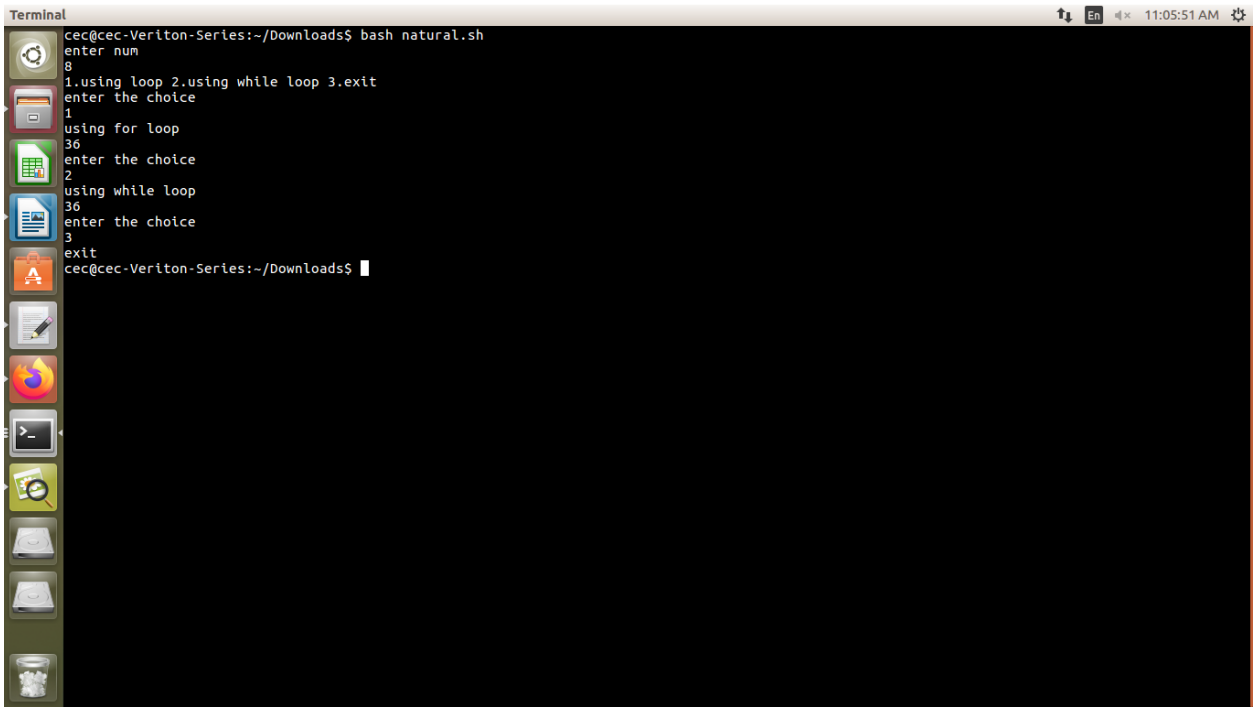
Aim: Write shell script program to find sum of natural numbers

a) Using for loop
b) Using while loop

Algorithm:

1. Start
2. Read the user input and store it in a variable num.
3. Display a menu with options:
    1. Using loop
    2. Using while loop
    3. Exit
4. Start an infinite loop (while :).
    - Read the user's choice and store it in a variable choice.
    - Use a case statement to handle the different choices:
        1. Case 1:
            1. Display "Using for loop".
            2. Initialize a variable a with the value of num.
            3. Initialize a variable total with 0.
            4. Use a for loop with the loop variable i ranging from a down to 0.
                - Inside the loop, update the value of total by adding i to it.
            5. Display the value of total.
        2. Case 2:
            1. Display "Using while loop".
            2. Initialize a variable a with the value of num.
            3. Initialize a variable total with 0.
            4. Use a while loop with the condition that 0 should be less than a.
                - Inside the loop, update the value of total by adding a to it.
                - Decrement the value of a by 1.
            5. Display the value of total.
        3. Case 3:
            1. Display "Exit".
            2. Break out of the infinite loop.

Output:


Terminal
11:05:51 AM
cec@cec-Veriton-Series:~/Downloads$ bash natural.sh
enter num
8
1.using loop 2.using while loop 3.exit
enter the choice
1
using for loop
36
enter the choice
2
using while loop
36
enter the choice
3
exit
cec@cec-Veriton-Series:~/Downloads$

4.Default:

1. Display "Invalid choice".

- End the case statement.

5.Stop.

Program:

```
echo "enter number"
read num
echo "1.using loop 2.using while loop 3.exit"
while :
do
	echo "enter the choice"
	read choice
	case $choice in
	1)	 echo "using for loop"
		a=$num
		total=0
		for((i=$a;i>=0;i--))
		do
			total=`expr $total + $i`
		done
		echo $total
		;;
	2)	 echo "using while loop"

		a=$num
		total=0
		while [ 0 -lt $a ]
		do
		total=`expr $total + $a`
		a=`expr $a - 1`

		done
		echo $total

		;;
	3)	 echo "exit"
		break
		;;
	*)	 echo "invalid choice"
		;;
	esac
done
```

Result: Program executed successfully and output obtained.

Experiment No. 10

Compare two strings

Aim: Write shell script program to compare two strings and check if the strings are empty

Algorithm:

1. Prompt the user to enter the first string.
2. Read the first string from user input and store it in 'string1'.
3. Prompt the user to enter the second string.
4. Read the second string from user input and store it in 'string2'.
5. Check if 'string1' is empty:
   5.1 If it is empty, print "The first string is empty."
   5.2 If it is not empty, print "The first string is not empty."
6. Check if 'string2' is empty:
   6.1 If it is empty, print "The second string is empty."
   6.2 If it is not empty, print "The second string is not empty."
7. Compare 'string1' and 'string2':
   7.1 If they are equal, print "The strings are equal."
7.2 If they are not equal, print "The strings are not equal."

Program:

```
#!/bin/bash
read -p "Enter the first string: " string1
read -p "Enter the second string: " string2
if [ -z "$string1" ]; then
   echo "The first string is empty."
else
   echo "The first string is not empty."
fi
if [ -z "$string2" ]; then
   echo "The second string is empty."
else
```

Output:


cec123@user:~/Desktop/networking$ gedit file1.sh
cec123@user:~/Desktop/networking$ bash file1.sh
Enter the first string: Hai hello
Enter the second string: Hai hello
The first string is not empty.
The second string is not empty.
The strings are equal.
cec123@user:~/Desktop/networking$ bash file1.sh
Enter the first string: Hai hello
Enter the second string: hello hai
The first string is not empty.
The second string is not empty.
The strings are not equal.
cec123@user:~/Desktop/networking$

```
    echo "The second string is not empty."
fi
if [ "$string1" = "$string2" ]; then
    echo "The strings are equal."
else
    echo "The strings are not equal."
fi
```

Result: Program executed successfully and output obtained

Experiment No. 11

Fibonacci series using function

Aim: Write shell script program to generate fibonacci series using function

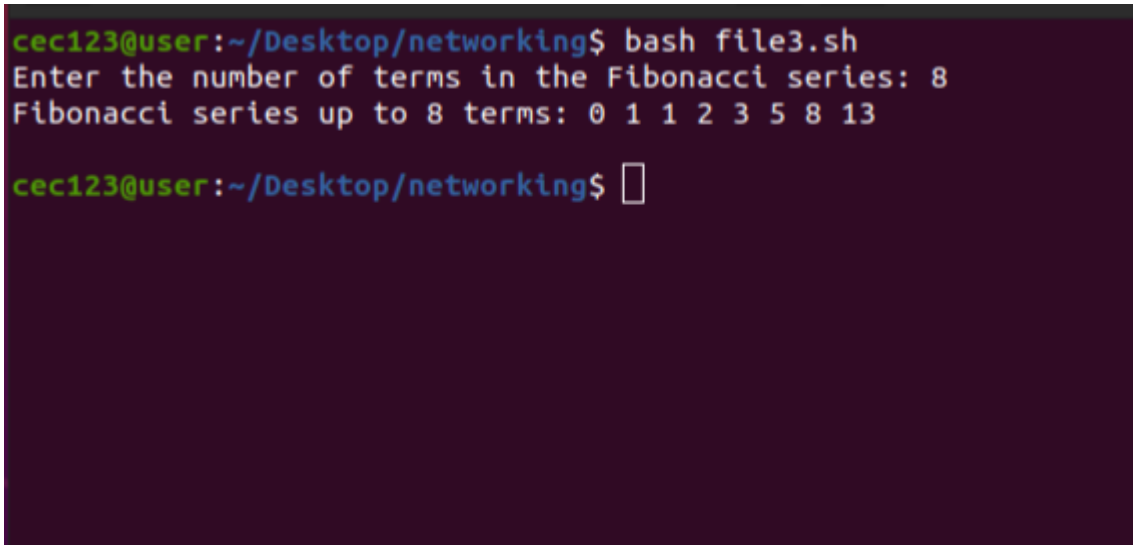
Algorithm:

1. Define a function 'generate_fibonacci':
   1.1 Accept a parameter 'limit' which represents the upper limit for the Fibonacci series.
   1.2 Initialize local variables 'a' and 'b' to store the first two numbers of the series.
   1.3 Print the first number 'a'.
   1.4 Use a loop while 'b' is less than or equal to 'limit':
      1.4.1 Print the current number 'b'.
      1.4.2 Calculate the next number 'next' as the sum of 'a' and 'b'.
      1.4.3 Update 'a' to be 'b'.
      1.4.4 Update 'b' to be 'next'.
2. Prompt the user to enter the limit for the Fibonacci series.
3. Read the entered limit and store it in the variable 'limit'.
4. Call the function 'generate_fibonacci' with the 'limit' as an argument.

Program:

```
fibonacci() {
n=$1
a=0
b=1
if [ $n -eq 1 ]; then
echo $a
elif [ $n -eq 2 ]; then
echo -n "$a $b"
else
echo -n "$a $b "
for (( i=3; i<=n; i++ )); do
c=$((a + b))
```

Output:


cec123@user:~/Desktop/networking$ bash file3.sh
Enter the number of terms in the Fibonacci series: 8
Fibonacci series up to 8 terms: 0 1 1 2 3 5 8 13
cec123@user:~/Desktop/networking$

```
echo -n "$c "
a=$b
b=$c
one
fi
echo
}
read -p "Enter the number of terms in the Fibonacci series: " num_terms
result=$(fibonacci $num_terms)
echo "Fibonacci series up to $num_terms terms: $result"
echo " “
```

Result: Program executed successfully and output obtained

Experiment No. 12

Palindrome

Aim: Write shell script program check wheather given number is palindrome

Algorithm:

1. Define a function 'is_palindrome':

   1.1 Accept a parameter 'num' which represents the number to be checked.

   1.2 Initialize local variables 'original' and 'reversed' to store the original and reversed versions of the number.

   1.3 Use a loop while 'num' is greater than 0:

      1.3.1 Get the last digit of 'num' using 'num % 10'.

      1.3.2 Append the digit to 'reversed' after multiplying it by 10.

      1.3.3 Remove the last digit from 'num' using 'num / 10'.

   1.4 Compare 'reversed' with 'original':

      1.4.1 If they are equal, return success (0) as the number is a palindrome.

      1.4.2 If they are not equal, return failure (1) as the number is not a palindrome.

2. Prompt the user to enter a number.

3. Read the entered number and store it in the variable 'number'.

4. Call the function 'is_palindrome' with the 'number' as an argument.

5. Check the return status of the function:

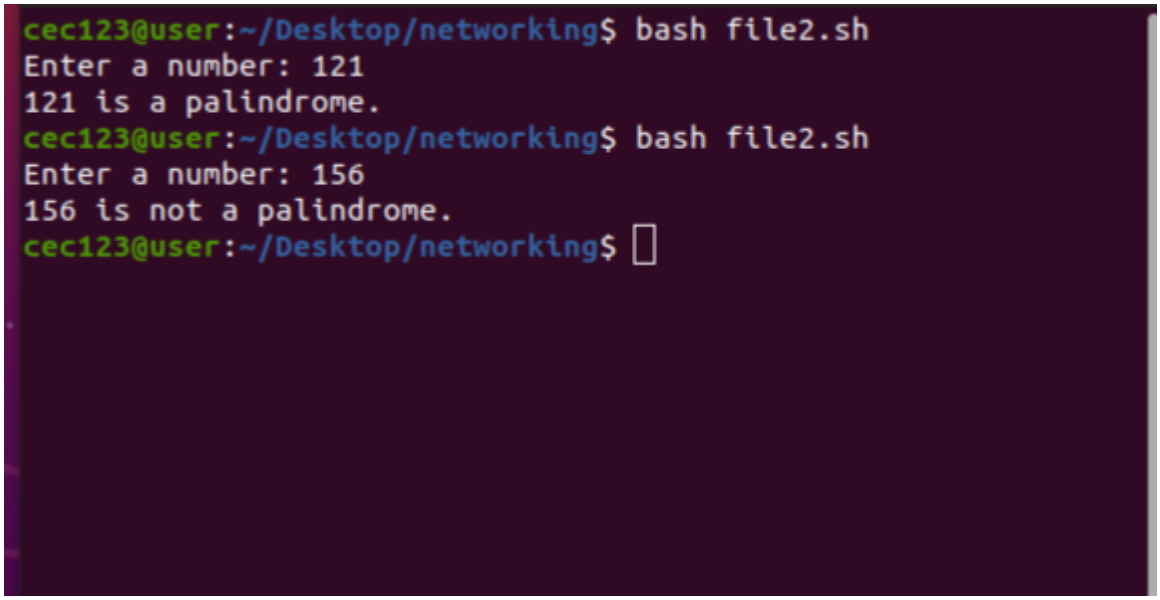
   5.1 If the return status is 0, print "[number] is a palindrome."

5.2 If the return status is 1, print "[number] is not a palindrome."

Program:

```
#!/bin/bash
is_palindrome()
 {
   local num=$1
   local original=$num
   local reversed=0
   while [ $num -gt 0 ]; do
      digit=$((num % 10))
```

Output:


cec123@user:~/Desktop/networking$ bash file2.sh
Enter a number: 121
121 is a palindrome.
cec123@user:~/Desktop/networking$ bash file2.sh
Enter a number: 156
156 is not a palindrome.
cec123@user:~/Desktop/networking$

```
        reversed=$((reversed * 10 + digit))
        num=$((num / 10))
    done
    if [ $reversed -eq $original ]; then
        return 0  # Return success (0) if palindrome
    else
        return 1  # Return failure (1) if not palindrome
    fi
}
read -p "Enter a number: " number
is_palindrome $number
if [ $? -eq 0 ]; then
    echo "$number is a palindrome."
else
    echo "$number is not a palindrome."
fi
```

Result: Program executed successfully and output obtained

Experiment No. 13

Prime number between two limits using function

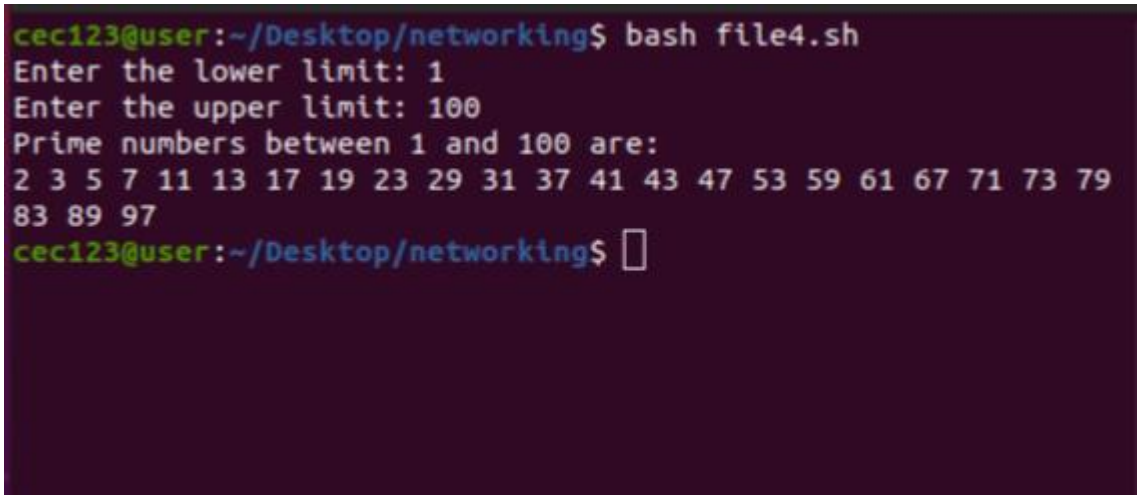
Aim: Write shell script program to print prime number between two limits using function

Algorithm:

1. Start
2. Define the `is_prime` Function:
   - Function `is_prime(num)`:
     - If `num` is less than 2:
       - Return 1 (indicating not prime)
     - For `i` starting from 2 up to the square root of `num`:
       - If `num` is divisible by `i`:
         - Return 1 (indicating not prime)
     - Return 0 (indicating prime)
3. Define the `print_primes` Function:
   - Function `print_primes(start, end)`:
     - For `num` starting from `start` up to `end`:
       - If `num` is prime (using `is_prime` function):
         - Print `num` without a newline
     - Print a newline
4. Get user input for the lower limit and upper limit:
   - Read the value of `lower_limit` from the user
   - Read the value of `upper_limit` from the user
5. Call the `print_primes` Function:
   - Print "Prime numbers between lower_limit and upper_limit are:"
   - Call `print_primes(lower_limit, upper_limit)`

6. End

Output:

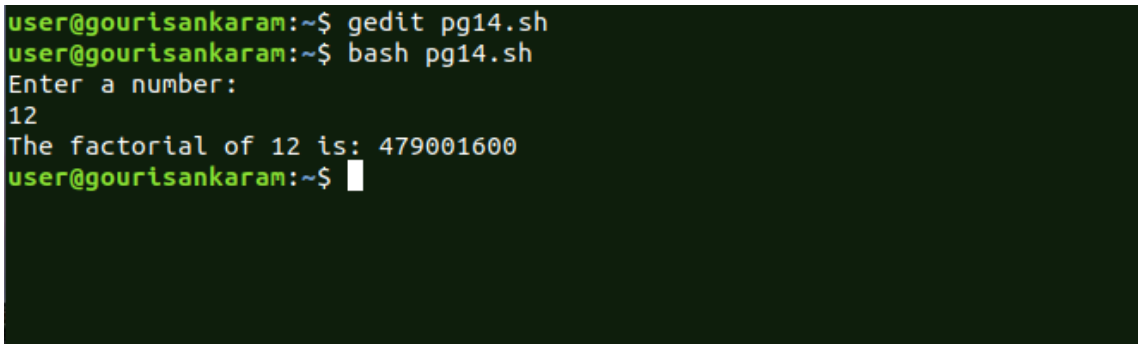

cec123@user:~/Desktop/networking$ bash file4.sh
Enter the lower limit: 1
Enter the upper limit: 100
Prime numbers between 1 and 100 are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97
cec123@user:~/Desktop/networking$

Program:

```
#!/bin/bash
is_prime()
{
 local num=$1
    if [ $num -lt 2 ]; then
        return 1
    fi
    for (( i=2; i*i<=num; i++ )); do
        if [ $((num % i)) -eq 0 ]; then
            return 1
        fi
    done
 return 0 }
print_primes() {
    local start=$1
    local end=$2
    for (( num=start; num<=end; num++ )); do
        if is_prime $num; then
            echo -n "$num "
        fi
    done
    echo
}
read -p "Enter the lower limit: " lower_limit
read -p "Enter the upper limit: " upper_limit
echo "Prime numbers between $lower_limit and $upper_limit are:"
print_primes $lower_limit $upper_limit
```

Result: Program executed successfully and output obtained.

Output:


user@gourisankaram:~$ gedit pg14.sh
user@gourisankaram:~$ bash pg14.sh
Enter a number:
12
The factorial of 12 is: 479001600
user@gourisankaram:~$

Experiment No. 14

Factorial of a given number using function

Aim: Write shell script program to generate factorial of a given number using function

Algorithm:
1. Define a function called `factorial` which takes one argument, `$1`, representing the number for which the factorial needs to be calculated.
2. Inside the `factorial` function:
- Check if `$1` is equal to 0. If yes, return 1 (since the factorial of 0 is 1).
- If `$1` is not 0:
- Subtract 1 from `$1` and store the result in a local variable `temp`.
- Call the `factorial` function recursively with the value of `temp` and store the result in a local variable `result`.
- Multiply `result` by `$1` and store the final result in a local variable `result`
- Return `result`.
3. Outside the function, print a message to the user to "Enter a number."
4. Read the user's input number and store it in a variable `num`.
5. Call the `factorial` function with the input `num` and store the result in a variable `result`.
6. Print the result, displaying "The factorial of $num is: $result."

Program:
```
factorial(){
   if [ $1 -eq 0 ]; then
      echo 1
   else
      local temp=$(expr $1 - 1)
      local result=$(factorial $temp)
      echo $(expr $result \* $1)
   fi
}
echo "Enter a number:"
read num
result=$(factorial $num)
echo "The factorial of $num is: $result"
```

Result: Program executed successfully and output obtained.

Experiment No. 15

Gross salary

Aim: Write shell script program that compute the gross salary of a employee according to following rules
a.if basic salary is <15000 then HRA-10% of basic and DA=90% of the basic
b.if basic salary is >15000 then HRA =rs 100 of basic and DA=98% of the basic

Algorithm:
1. Start

2. Define the `calculate_gross_salary` Function:

   - Function `calculate_gross_salary(basic)`:

     - If `basic` is less than 15000:

       - Calculate HRA as 10% of `basic` and DA as 90% of `basic`

     - Else:

       - Set HRA to 100 and calculate DA as 98% of `basic`

     - Calculate gross_salary as the sum of `basic`, HRA, and DA

     - Return `gross_salary`

3. Get user input for basic salary:

   - Read the value of `basic_salary` from the user

4. Call the `calculate_gross_salary` Function:

   - Call `calculate_gross_salary(basic_salary)`

   - Store the result in `gross_salary`

5. Display the calculated gross salary:

   - Print "Gross Salary: gross_salary"

6. End

Output:


cec123@user:~/Desktop/networking$ bash file5.sh
Enter the basic salary: 60000
Gross Salary: 118900.00
cec123@user:~/Desktop/networking$

Program:

```
#!/bin/bash
calculate_gross_salary() {
   local basic=$1
   if [ $basic -lt 15000 ]; then
      hra=$(echo "scale=2; $basic * 0.10" | bc)
      da=$(echo "scale=2; $basic * 0.90" | bc)
 else
      hra=100
      da=$(echo "scale=2; $basic * 0.98" | bc)
   fi
   gross_salary=$(echo "scale=2; $basic + $hra + $da" | bc)
   echo $gross_salary
}
read -p "Enter the basic salary: " basic_salary
gross_salary=$(calculate_gross_salary $basic_salary)
echo "Gross Salary: $gross_salary"
```

Result: Program executed successfully and output obtained

Output:

```
user@gourisankaram:~$ bash pg16.sh
Enter the first integer: 12
Enter the second integer: 2
21
The value of 12 raised to the power of 2 is 441
user@gourisankaram:~$
```

Experiment No. 16

Power

Aim: Write shell script program that accept two integers as its arguments and compute the value of first number reversed to the power of second number

Algorithm:

1. Prompt the user to enter the first integer and store it in the variable `first`.

2. Prompt the user to enter the second integer and store it in the variable `second`.

3. Reverse the first integer by using the `rev` command and store the result in the variable `reversed_first`.

4. Display the reversed first integer by echoing the `reversed_first` variable.

5. Calculate the result of raising the reversed first integer to the power of the second integer using the `bc` command. The expression `"reversed_first^second"` is passed to `bc`, and the result is stored in the variable `result`.

6. Display the result by echoing a message that includes the values of `first`, `second`, and `result`, showing "The value of $first raised to the power of $second is $result."

Program:

```
#!/bin/bash

read -p "Enter the first integer: " first

read -p "Enter the second integer: " second

reversed_first=$(echo $first | rev)

echo $reversed_first

result=$(echo "$reversed_first^$second" | bc)

echo "The value of $first raised to the power of $second is $result "
```

Result: Program executed successfully and output obtained.

Experiment No. 17

Calculator

Aim: Write shell script program for calculator

Algorithm:

1. Prompt the user to enter the first number.
2. Read the first number and store it in the variable 'a'.
3. Prompt the user to enter the second number.
4. Read the second number and store it in the variable 'b'.
5. Enter a while loop that keeps running until the user chooses to exit.
6. Inside the loop, display a menu of arithmetic operations:
   - 1 for Addition
   - 2 for Subtraction
   - 3 for Multiplication
   - 4 for Division
   - 5 for Remainder
   - 6 for Exit
7. Prompt the user to enter their choice.
8. Read the user's choice and store it in the variable 'choice'.
9. Use a 'case' statement to perform the corresponding operation based on the user's choice:
   - If 'choice' is 1, perform addition and display the result.
   - If 'choice' is 2, perform subtraction and display the result.
   - If 'choice' is 3, perform multiplication and display the result.
   - If 'choice' is 4, perform division and display the result.
   - If 'choice' is 5, perform remainder operation and display the result.
   - If 'choice' is 6, exit the loop and the script.
   - If 'choice' is anything else, display "Invalid choice."
10. Repeat the loop until the user chooses to exit (chooses 6).

Program:

```
echo "Enter first number"
read a
echo "Enter second number"
read b
echo "1.Addition 2.Substraction 3.Multiplication 4.Division 5.Remainder 6.Exit"
while :
do
        echo "Enter the choice"
        read choice
        case $choice in
        1)      echo "Addition"
                expr $a + $b
                ;;
        2)      echo "Difference"
                expr $a - $b
                ;;
```

Output:

```
user@gourisankaram:~$ gedit pg17.sh
user@gourisankaram:~$ bash pg17.sh
Enter first number
23
Enter second number
56
1.Addition 2.Substraction 3.Multiplication 4.Division 5.Remainder 6.Exit
Enter the choice
1
Addition
79
Enter the choice
2
Difference
-33
Enter the choice
3
Multiplication
1288
Enter the choice
4
Quotient
0
Enter the choice
5
Remainder
23
Enter the choice
6
Exit
user@gourisankaram:~$
```

```
3)      echo "Multiplication"
            expr $a \* $b
            ;;
        4)      echo "Quotient"
            expr $a / $b
            ;;
        5)      echo "Remainder"
            expr $a % $b
            ;;
        6)      echo "Exit"
            break
            ;;
        *)      echo "Invalid choice"
            ;;
        esac
done
```

Result: Program executed successfully and output obtained.

Experiment No. 18

Binary to decimal

Aim: Write shell script program to convert binary to decimal

Algorithm:

1. The script starts by printing "Enter Binary Number -" and then reads the input value into the variable `n`.

2. There is a function named `binaryCon` defined. This function is used to convert a binary number
to its decimal equivalent. It starts by initializing variables `i` (for power of 2) and `num` (for storing the decimal result) to 0.

3. Inside the `binaryCon` function, there is a `while` loop that continues until the value of `n` becomes 0.

4. Inside the loop:
-The last digit of the binary number is extracted using the modulo operation (`expr $n % 10`), and stored in the variable `digit`.
- The decimal equivalent of the extracted binary digit is calculated using the formula `num + digit
* 2^i` and stored in the variable `num`.
- The value of `n` is updated by performing integer division (`expr $n / 10`) to remove the last digit.
- The variable `i` is incremented to keep track of the current power of 2.

5. Once the loop completes, the `binaryCon` function prints "Resultant Decimal Number" followed by the calculated decimal value stored in the variable `num`.

6. The main part of the script asks the user to enter a value for `n`, which represents the number of values the user will input.

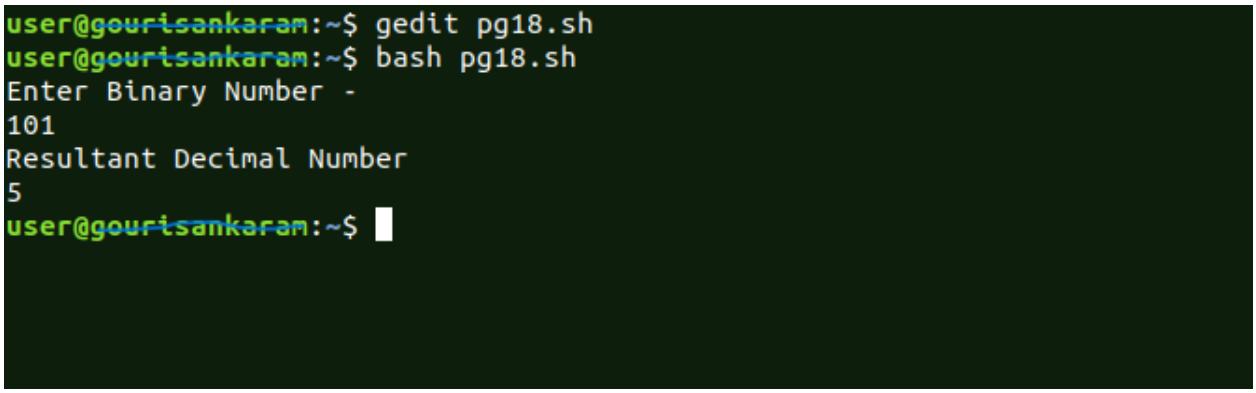
7. A `while` loop is used to read `n` numbers one by one. The loop runs from 1 to `n` (inclusive).

8. Inside the loop:
- The script reads a number from the user and stores it in the variable `num`.
- The variable `sum` is updated by adding the current input `num` to it.
- The loop counter `i` is incremented.

9. After reading all the input numbers and calculating their sum, the script calculates the average by dividing the sum (`sum`) by the total number of inputs (`n`). The `bc -l` command is used for floating- point division.

10. The calculated average (`avg`) is printed on the screen as "avg=<average value>".

Output:

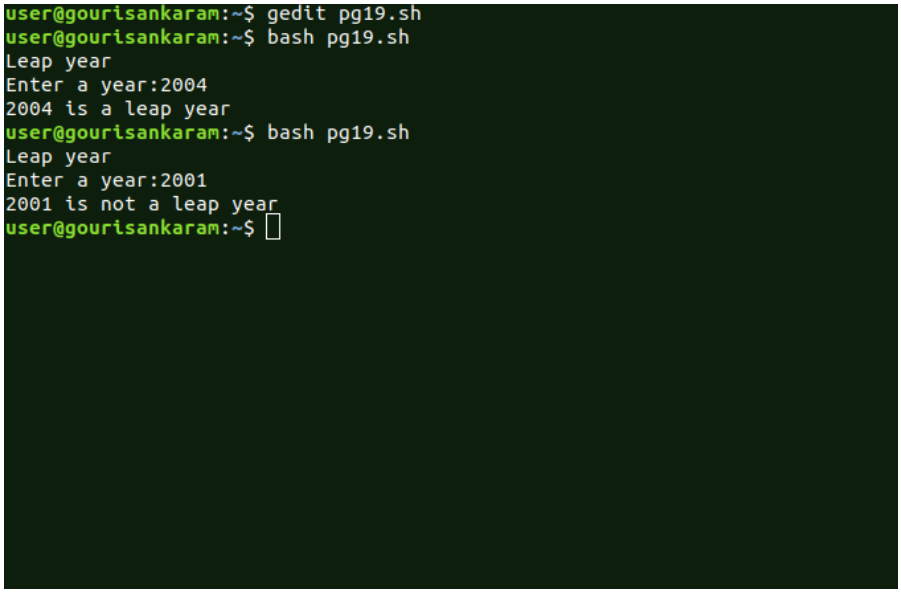

user@gouritsankaram:~$ gedit pg18.sh
user@gouritsankaram:~$ bash pg18.sh
Enter Binary Number -
101
Resultant Decimal Number
5
user@gouritsankaram:~$

Program:

```
echo "Enter Binary Number -"
read n
function binaryCon(){
i=0
local num=0
while [ $n != 0 ]
do
digit=`expr $n % 10`
num=$(( num + digit * 2**i ))
n=`expr $n / 10`
(( ++i ))
done
echo "Resultant Decimal Number"
echo "$num"
}
binaryCon
```

Result: Program executed successfully and output obtained.

Output:


user@gourisankaram:~$ gedit pg19.sh
user@gourisankaram:~$ bash pg19.sh
Leap year
Enter a year:2004
2004 is a leap year
user@gourisankaram:~$ bash pg19.sh
Leap year
Enter a year:2001
2001 is not a leap year
user@gourisankaram:~$

Experiment No. 19

Leap year or not

Aim: Write shell script program to check leap year or not

Algorithm:

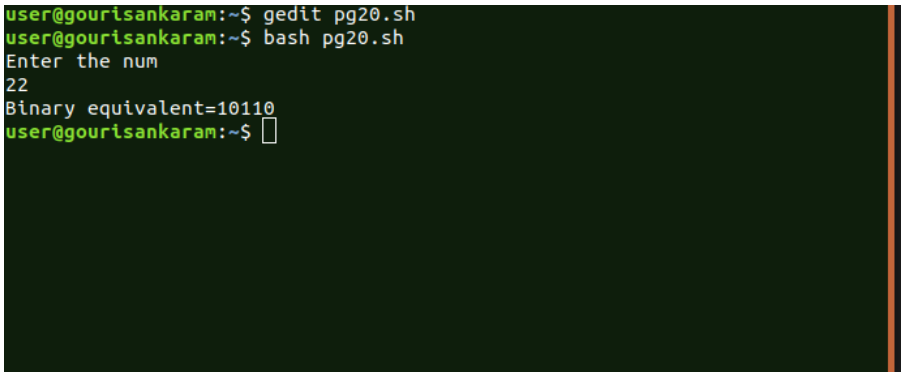
1. Print "Leap year" to the console.
2. Prompt the user to enter a year by printing "Enter a year:" and reading the input into the variable `year`.
3. Use the `expr` command to calculate the remainder when `$year` is divided by 4 using backticks (\` \`) or `$()`. This is done with the expression `expr $year % 4`.
4. Check if the result of the modulo operation is equal to 0 by using the conditional statement `[ `expr $year % 4` -eq 0 ]`.
5. If the result is equal to 0, print "$year is a leap year" to the console.
6. If the result is not equal to 0 (i.e., the year is not divisible by 4), print "$year is not a leap year" to the console.

Program:

```
echo "Leap year"
echo -n "Enter a year:"
read year
if [ `expr $year % 4` -eq 0 ]
then
echo "$year is a leap year"
else
echo "$year is not a leap year"
fi
```

Result: Program executed successfully and output obtained.

Output:


user@gourisankaram:~$ gedit pg20.sh
user@gourisankaram:~$ bash pg20.sh
Enter the num
22
Binary equivalent=10110
user@gourisankaram:~$

Experiment No. 20

Decimal to binary

Aim: Write shell script program to convert decimal to binary

Algorithm:

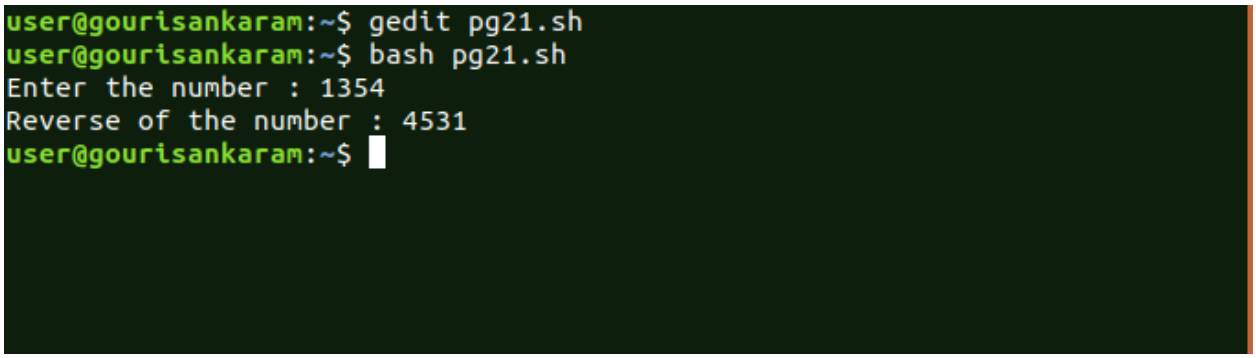
1. Print "Enter the num" to the console.
2. Read the decimal number entered by the user into the variable `n`.
3. Initialize two variables: `val` to 0 (which will store the binary equivalent) and `power` to 1 (used to calculate the place value of each binary digit).
4. Start a while loop that continues as long as `n` is not equal to 0:
   a. Calculate the remainder of `n` divided by 2 using the expression `r=expr $n % 2`.
   b. Update the `val` variable by multiplying the current remainder `r` by the current `power` and adding it to `val`. This effectively appends the binary digit to the result.
   c. Multiply `power` by 10 to update its value for the next iteration, representing the place value of the next binary digit.
   d. Divide `n` by 2 to prepare for the next iteration.
5. Once the loop finishes, print "Binary equivalent=$val" to display the binary equivalent of the entered decimal number.

Program:

```
echo "Enter the num"
read n
val=0
power=1
while [ $n -ne 0 ]
do
r=`expr $n % 2`
val=`expr $r \* $power + $val`
power=`expr $power \* 10`
n=`expr $n \/ 2`
done
echo "Binary equivalent=$val"
```

Result: Program executed successfully and output obtained.

Output:


user@gourisankaram:~$ gedit pg21.sh
user@gourisankaram:~$ bash pg21.sh
Enter the number : 1354
Reverse of the number : 4531
user@gourisankaram:~$

Experiment No. 21

Reverse a number

Aim: Write shell script program to reverse a number

Algorithm:

1. Prompt the user to enter a number and store it in a variable 'n'.
2. Initialize a variable 'num' to 0. This variable will store the reversed number.
3. Start a while loop with the condition that 'n' is greater than 0.
4. Inside the loop:
   a. Multiply 'num' by 10 (shifts the digits of 'num' to the left).
   b. Find the last digit of 'n' by taking the remainder when divided by 10 and store it in a variable 'k'.
   c. Add 'k' to 'num' (appends 'k' as the last digit of 'num').
   d. Update 'n' by removing its last digit (divide by 10).
5. Repeat the loop until 'n' becomes 0.
6. Print the reversed number, which is stored in 'num'.

Program:

```
read -p "Enter the number : " n
num=0
while [ $n -gt 0 ]
do
num=$(expr $num \* 10)
k=$(expr $n % 10)
num=$(expr $num + $k)
n=$(expr $n / 10)
done
echo Reverse of the number : $num
```

Result: Program executed successfully and output obtained.

Output:

```
user@gourisankaram:~$ gedit pg22.sh
user@gourisankaram:~$ bash pg22.sh
Enter size(n)
5
Enter numbers
1
2
3
4
5
avg=3.000000000000000000
user@gourisankaram:~$
```

Experiment No. 22

Average

Aim: Write shell script program to find average of n numbers

Algorithm:

1. Prompt the user to enter the size 'n' (the number of values to be averaged).
2. Initialize variables 'i' to 1 (to keep track of the input count), 'sum' to 0 (to store the sum of the numbers), and 'avg' to 0 (to store the average).
3. Display a message to the user: "Enter numbers".
4. Start a loop that runs from 'i' = 1 to 'n':
   a. Prompt the user to enter a number 'num'.
   b. Add 'num' to 'sum' to accumulate the sum of entered numbers.
   c. Increment 'i' by 1 to keep track of the number of inputs.
5. Calculate the average by dividing 'sum' by 'n' using the 'bc' command to handle decimal arithmetic.
6. Display the average to the user as "avg=<average>".

Program:

```
echo "Enter size(n)"
read n
i=1
sum=0
echo "Enter numbers"
while [ $i -le $n ]
do
read num
sum=$((sum+num))
i=$((i+1))
done
avg=$(echo $sum / $n | bc -l)
echo "avg=$avg"
```

Result: Program executed successfully and output obtained.