UNIX LAB
SEM IV
MU, IT Dept.

By,

Himani Deshpande(TSEC)

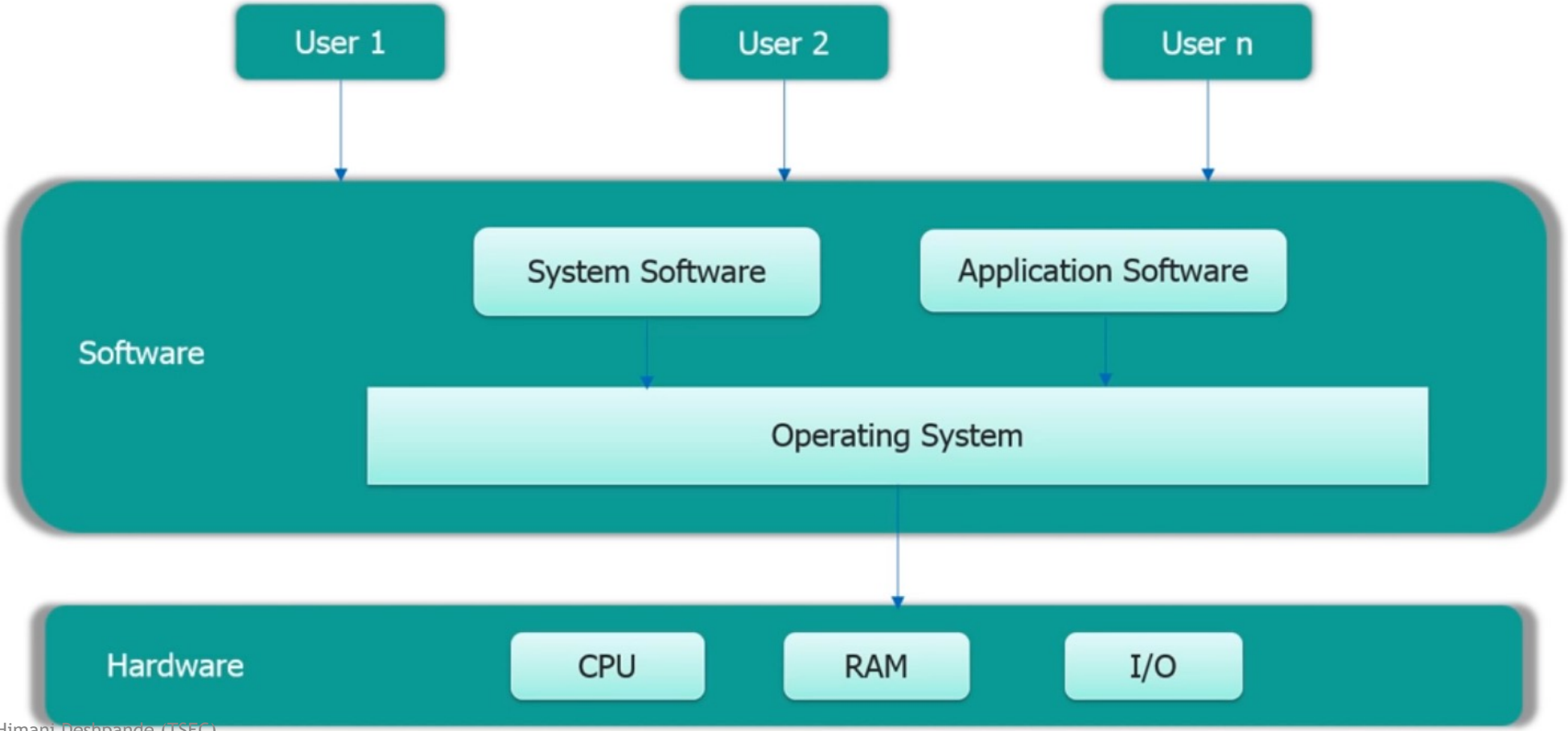
Unix Lab

- **LO 1 -:** Students will be able to understand the architecture and functioning of Unix
- **LO 2**-: Students will be able to identify the basic Unix general purpose commands.
- **LO 3-:** Students will be able to execute unix commands for system administrative tasks such as file management and user management.
- **LO 4-:** Students will be able to execute unix commands for system administrative tasks such as process management and memory management.
- **LO 5-:** Students will be able to implement shell scripts for different applications.
- **LO 3-:** Students will be able to implement **advance scripts** using the awk, grep, sed and perl scripts for performing various tasks.

Assignment List

Sr. No.	TITLE	Written/ Programming	LO
1	Basic concepts of operating system.	Written	LO1
2	Study of Unix general purpose utility commands.	Programming	LO1
3	Study of Vi Editor.	Programming	LO6
4	Study of unix file and directory permissions.	Programming	LO2
5	Commands for Basic System administrative task.	Programming	LO5
6	Programming using Shell Script (Basic).	Programming	LO4
7	Programming using Shell Script (Adv).	Programming	LO4
8	Study of awk commands.	Programming	LO3
9	Study of grep commands.	Programming	LO3
10	Programs using sed (Stream Editor).	Programming	LO2
11	Programming using Perl script.	Programming	LO1
12	Study of unix file system.	Written	LO1

UNIX OS


User 1
User 2
User n
Software
System Software
Application Software
Operating System
Hardware
CPU
RAM
I/O

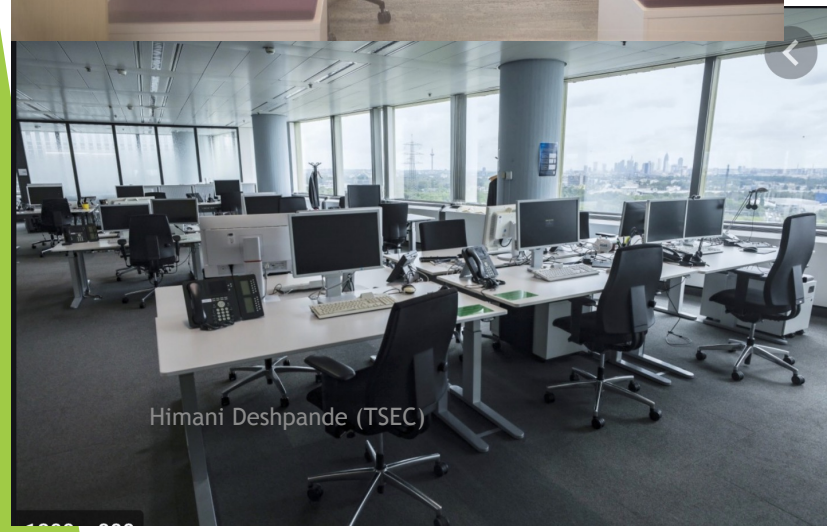
Operating System

- Operating system is a set of programs which acts as an interface between between users and computer.
- It is the heart of any machine's software and provides environment in which a user can execute programs.
- It controls the allocation of resources and services such as memory, processor, devices , information etc.

- Popular OS are
  DOS
  Windows
  MAC OS
  Unix/Linux

OS example

- Employees are application software they need to perform.
- Office building provides the environment.
- If the office has facilities like cleanliness, proper ventilation , lift, canteen, welding machine, other resources.
- Environment effects efficiency.



Without OS

If you want to play a game you need to design it to suit to your hardware.

You need to design whole environment.

Your code cant port that game to another machine .

Power of OS

- IBM launched its DOS coded by Bill Gates.
- Bill gates became the richest man, developing Windows Operating systems.

Assignment 1

- Q1. List operating system services.
- Q2. List and explain with a diagram different operating system structures with an example for each.

UNIX OS

- Unix is a multi-user, multi tasking and multi processing OS.
- It is a Command line Interpreter
- Developed at AT&T Bell Laboratories Research Center, USA in 1969 by Ken Thompson and Denis Ritchie.
- Earlier Unix was written in Assembly language and originally spelled as UNICS.
- Later it was re-written in ‘C’ language and named as Unix.
- Some Unix OS are open source and some are not.

Why UNIX

- Completely Free
- Comparatively secured
- Could run smoothly on any machine(hardware), even on your old computer .

• What Is UNIX?

- UNIX is a computer operating system, a control program that works with users to
  - run programs,
  - manage resources, and
  - communicate with other computer systems.

- Several people can use a UNIX computer at the same time; hence UNIX is called a multiuser system. Any of these users can also run multiple programs at the same time; hence UNIX is called multitasking.

Unix variants

- Unix has a number of variants but they all follow almost all the Unix terminal commands

Unix Operating System supports the following basic features and capabilities :

- **Multi-user** : More than one user can use the machine simultaneously supported via terminals
- **Multi-tasking** : Multiple programs can be run at a time
- **Multi-process** : Each user can execute several processes simultaneously
- **Hierarchical Structure** : Unix directories are present like a tree structure to support the organization and maintenance of files
- **Open System** : Some of the Unix OS are open-source. Users can modify the Unix source code
- **Portability** : It is the ability of the software that operates from one machine to another machine having different configuration Unix allows users to transfer data from one system to another
- **Programming Facility** : Unix Shell can be used as a Programming/Scripting Language
- **Communication Facility** : Unix allows communication between different users by providing some information
- **Security** : Unix has system level security controlled by system administrator and file level security controlled by owner of the file
- **Tools and Utilities** : Supports many of the tools, libraries and utilities to aid software development
- **Piping** : In Piping concept, the output of the first command becomes the input of the next command/process
- **Help Facility** : In Unix, 'man' command is used to view help content on any command
- **Modularity** : Unix consists of multiple number of independent modules or programs which perform different elementary tasks

Installation and Execution Options

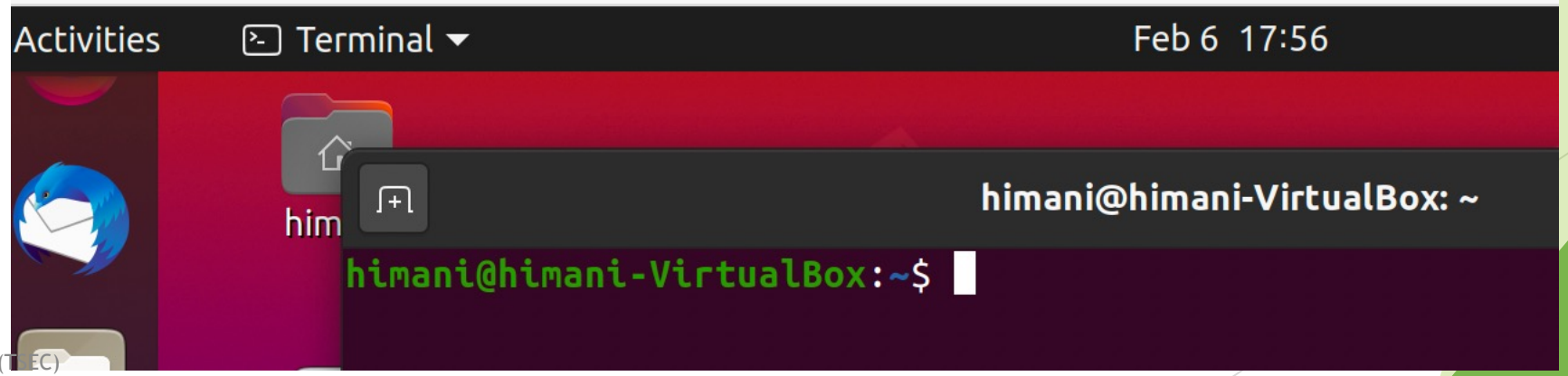
1. Host OS :-
   a. Single Boot - Ubuntu 20.04 LTS / BOSS Linux 8.0
   b. Dual Boot along with Windows - Ubuntu 20.04 LTS
2. Guest OS/Virtual OS
   a. VirtualBox
      Installation
      ISO IMAGE
   b.. VMware
      Installing VMware on Windows
      Ubuntu ISO image
      How to create ubuntu VM in VMware
3. Ubuntu Terminal App for Windows
4. Cloud OS - onworks.net
5. Termux App for Android - Google Play, F-Droid, Kali Nethunter Store
6. Virtual Labs

Terminal

Ctrl+Alt+T

Terminal provides an interface into which users can type commands and that can print text.

The terminal outputs the results of **commands** which are specified by the user itself. Execution of typed **command** is done only after you press the Enter key.



- Unix is also case-sensitive. This means that ***cat*** and ***Cat*** are different commands.
- The prompt is displayed by a special program called the **shell**.
- **Shells accept** commands, and **run** those commands.
- They can also be programmed in their own language. These programs are called "**shell scripts**".

BASIC UNIX COMMANDS

- echo,
- clear,
- exit,
- date,
- time,
- uptime: **Uptime** is a **command** that returns information about how long your system has been running together with the current time
- Cal
- man,

Cal

```
himani@himani-VirtualBox:~$ cal
   February 2021
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28
```

```
himani@himani-VirtualBox:~$ cal 2 2021
   February 2021
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28
```

```
himani@himani-VirtualBox:~$ date "+Date: %m%d%y%n Time: %H%M%S"
Date: 020621
 Time: 182005
```

```
himani@himani-VirtualBox:~$ cal 2000
                              2000
      January                 February                  March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1            1  2  3  4  5            1  2  3  4
 2  3  4  5  6  7  8   6  7  8  9 10 11 12   5  6  7  8  9 10 11
 9 10 11 12 13 14 15  13 14 15 16 17 18 19  12 13 14 15 16 17 18
16 17 18 19 20 21 22  20 21 22 23 24 25 26  19 20 21 22 23 24 25
23 24 25 26 27 28 29  27 28 29              26 27 28 29 30 31
30 31

      April                   May                     June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1         1  2  3  4  5  6                  1  2  3
 2  3  4  5  6  7  8   7  8  9 10 11 12 13   4  5  6  7  8  9 10
 9 10 11 12 13 14 15  14 15 16 17 18 19 20  11 12 13 14 15 16 17
16 17 18 19 20 21 22  21 22 23 24 25 26 27  18 19 20 21 22 23 24
23 24 25 26 27 28 29  28 29 30 31           25 26 27 28 29 30
```

Help

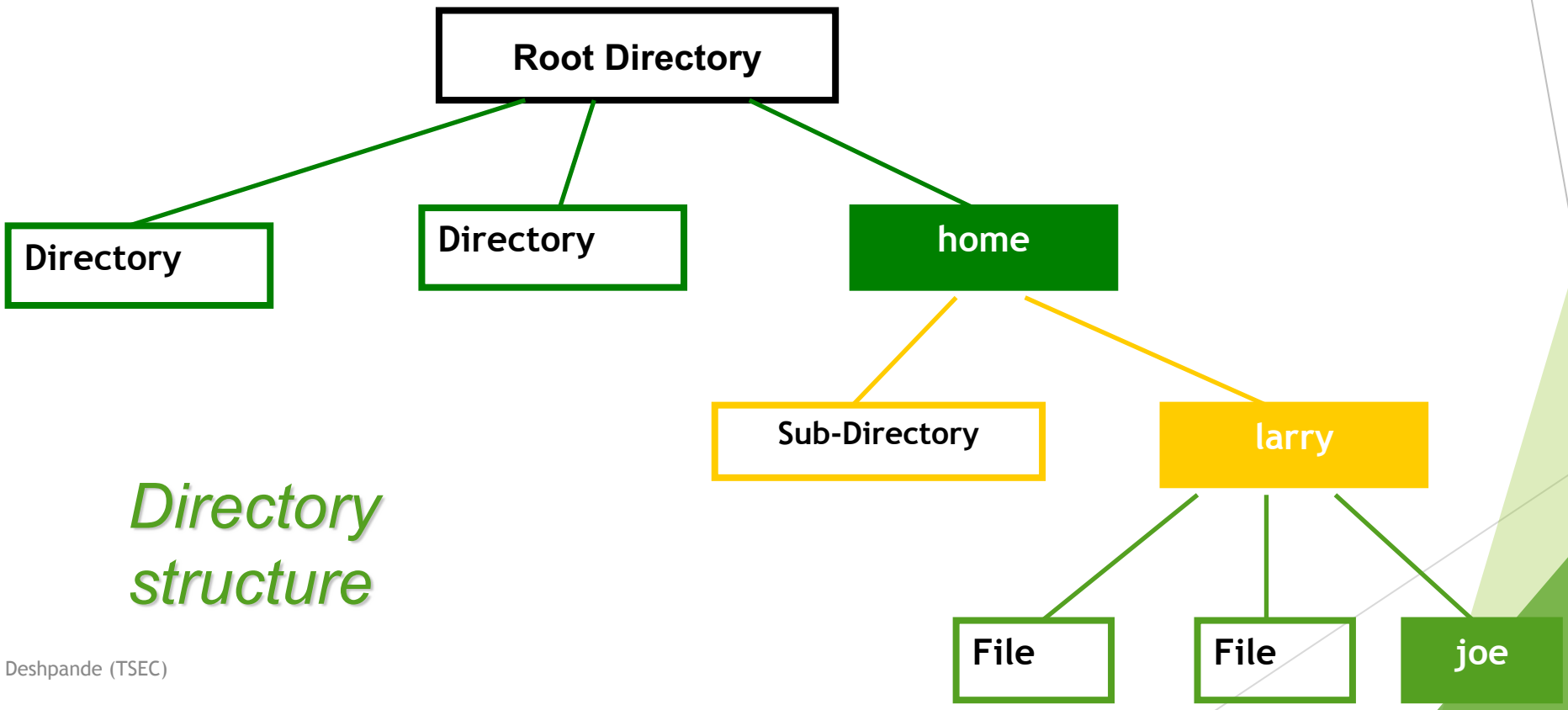

```
himani@himani-VirtualBox:~$ who
himani   :0           2021-02-06 17:40 (:0)
himani@himani-VirtualBox:~$ whoami
himani
himani@himani-VirtualBox:~$ who am i
himani@himani-VirtualBox:~$ passwd
Changing password for himani.
Current password:
New password:
Retype new password:
```

Storing information

- Unix provides files and directories.
- A directory is like a folder: it contains pieces of paper, or files.
- A large folder can even hold other folders-*directories can be inside directories*.
- In unix, the collection of directories and files is called the file system. Initially, the file system consists of one directory, called the **"root"** directory
- Inside "root" directory, there are more directories, and inside those directories are files and yet more directories.

- Each file and each directory has a name.
- A short name for a file could be joe,
- while it's "full name" would be **/home/larry/joe.** The full name is usually called the **path.**
- The path can be divide into a sequence of directories.
- For example, here is how **/home/larry/joe** is read:

- A path could refer to either a directory or a filename, so joe could be either.
- All the items before the short name must be directories.



Directory structure

Cat Command

- **cat filename**
  It will show content of given filename
- **$cat file1 file2**
  This will show the content of file1 and file2.
- **$cat -n filename**
  It will show content with line number example:-cat-n geeks.txt
  1)This is geeks
  2)A unique array
- **$ cat >newfile**
  Will create and a file named newfile
- **$cat -e file**
  '$' is shows at the end of line
- **$cat file1 > file2**
  Will copy the contents of one file to another file

tty command

- Since UNIX treats even terminals as files. It is tty (teletype) command, that tells you the filename of the terminal you are using. This command is simple and need no arguments:

  $tty <enter>

  /dev/tty01

- The terminal filename tty01 resident in the /dev directory. If the user logs in from another terminal next time, his terminal device name will be different.

tty – – version : Prints the version information

- which : which returns the pathnames of the files which would be executed in the current environment. Eg which cal
- history : history of executed commands
- pwd : present working directory
- whoami : user logged in

passwd

- With the passwd command, you can change the password associated with your individual account name.

- For example,

```
sariyer:~> passwd
Changing password for dag.
Old password:
New passwd:
Retype new passwd:
sariyer:~>
```

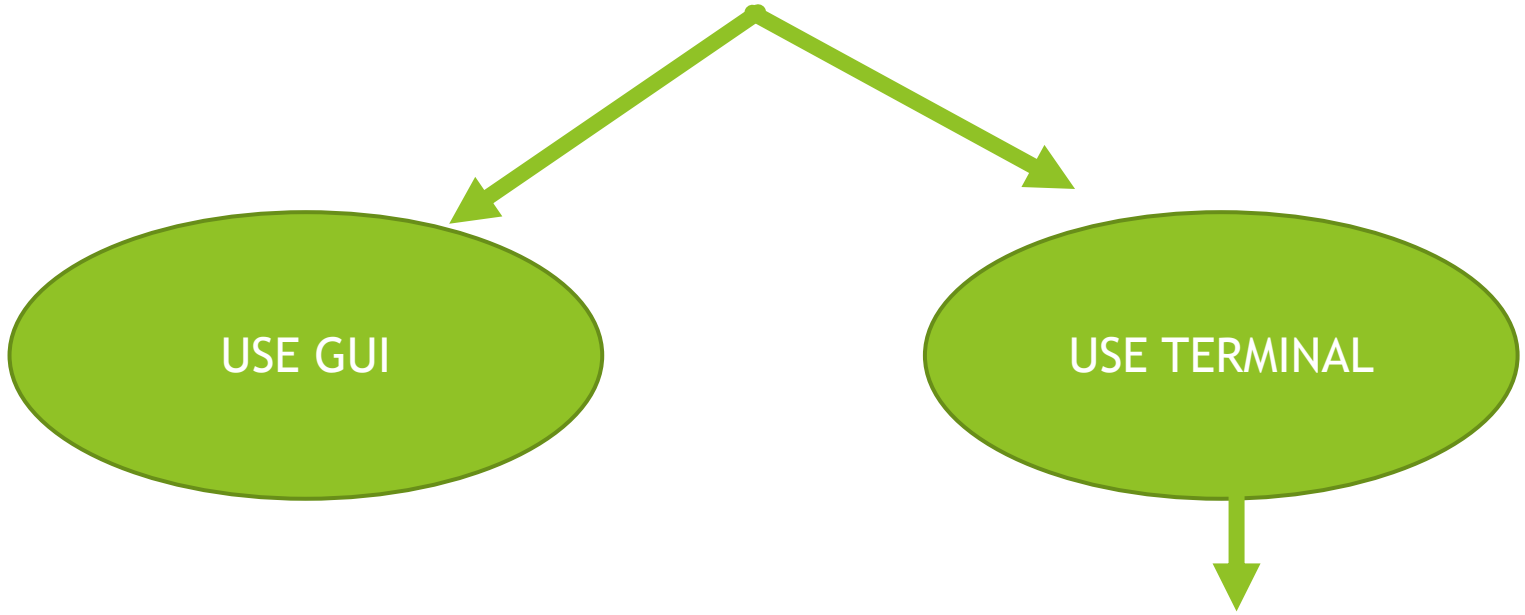
- pr: convert text file for printing
- lp: The lp command is used to print **files** on Unix and Linux systems. The name "lp" stands for "line printer".
- lpr: lpr submits **files** for printing.
- lpstat: **lpstat** displays status information about the current classes, jobs, and printers.
- lpq: shows printer queue status
- lprm: cancels printer queued job
- Cancel: cancel printer current job.
- mail, etc.

- id : id command in **Linux** is used to find out user and group names and numeric ID's (UID or group ID) of the current user or any other user in the server.
  This command is useful to find out the following information as listed below:
  1. User name and real user id.
  2. Find out the specific Users UID.
  3. Show the UID and all groups associated with a user.
  4. List out all the groups a user belongs to.
  5. Display security context of the current user.
- ping: **PING** (Packet Internet Groper) **command** is used to check the network connectivity
- Ifconfig:

EXPERIMENT – 3

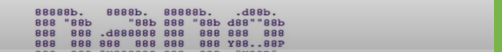
UNIX TEXT EDITOR
VI EDITOR

Editing/ creating file on Unix OS



Vi editor

Vi improved



VI Editor

- The VI editor is screen-based editor used by many unix users.
- VI is a screen-oriented editor written by Bill Joy in 1976.
- It is a text editor that uses command line on a Unix operating system
- VI lets you add, change and delete text, but does not provide such formatting capabilities as centering lines or indenting paragraphs.
- Vi is case-sensitive.

VI Editor

- It is advisable to work on vi editor as its
  - Feature rich.
  - Provide endless possibilities to work on files.
- Learning editor will benefit you to create scripts and editing files.

MODES

- Command Mode
- Insert Mode

Command Mode

- In command mode ,

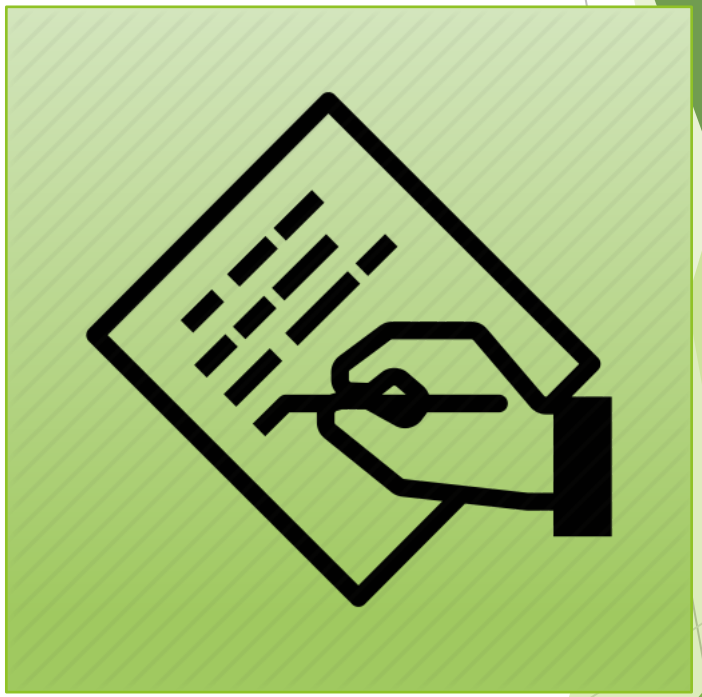
the characters you type are interpreted as commands.

Save changes to file .

Eg. Copy, cut, paste, search operations, moving the cursor etc.

Insert Mode

- In insert mode, everything you type is inserted into the file as text.
- Deals with writing on file.

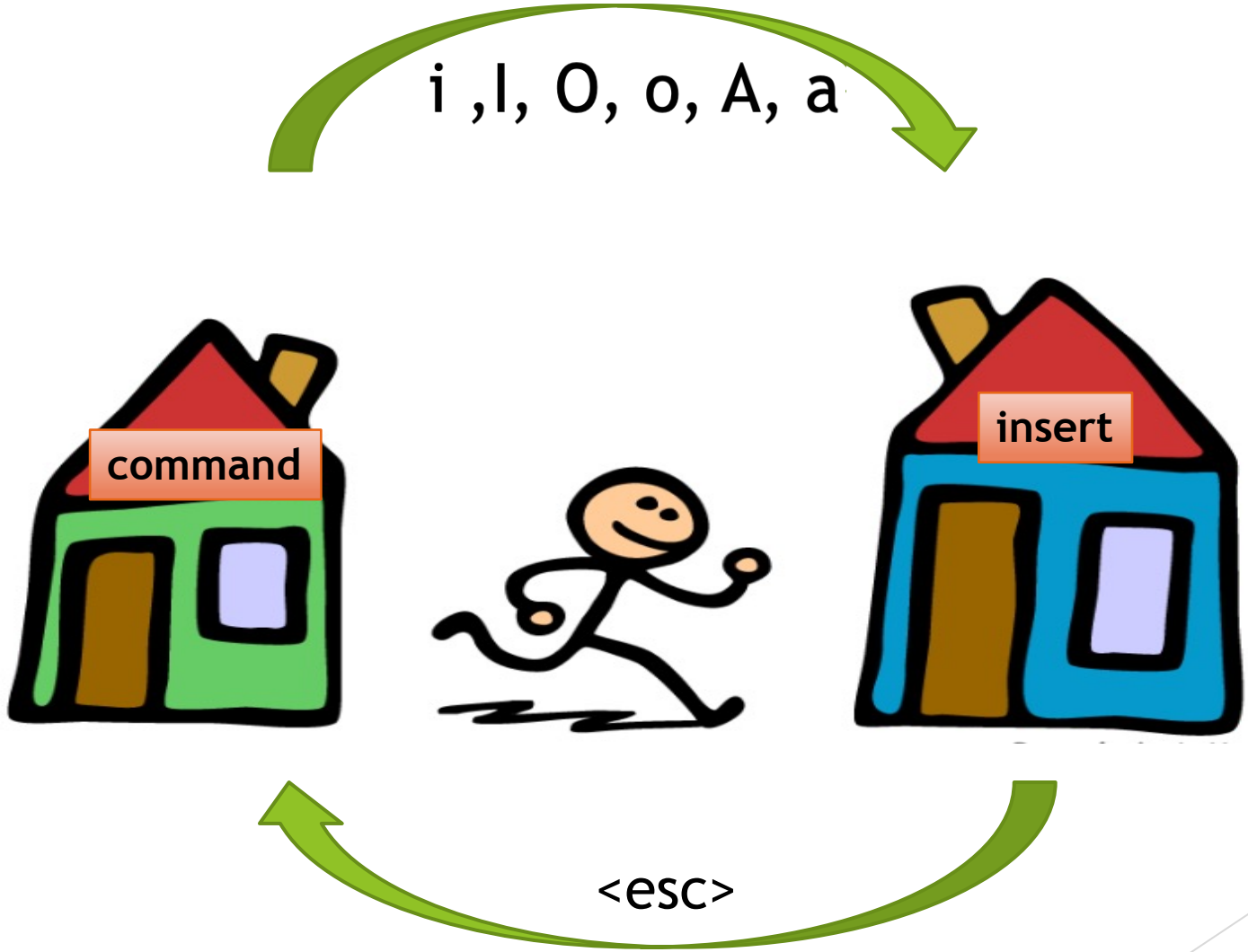


Switching between modes

- Vi starts in Command Mode by Default.
- Type <Esc> to change from insert Mode to Command Mode.
- To get into the insert mode from command mode press “i”.

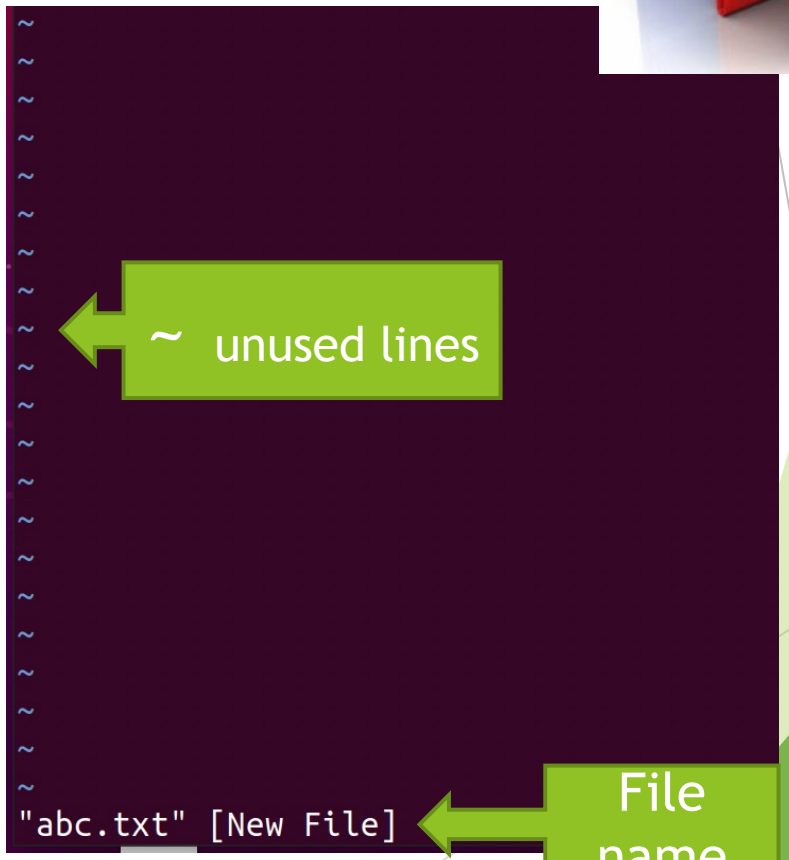
No visual cue to determine the mode you are currently in!

- If you forget which mode you are in , hit the **<Esc> key Twice** to get to Command Mode.


i ,I, O, o, A, a
command
insert
<esc>

Creating/opening a file

- Command:
  - $ vi <filename>
  - With enter key a new file names filename opens in command mode. If it already exists then it will open.



COMMAND MODE

INSERTING IN FILE

- I - insert text at the beginning of line
- i - Insert at cursor (goes into insert mode)
- A - Write at the end of line (goes into insert mode)
- a - Write after cursor (goes into insert mode)
- O - Open up a new line in front of the current line and add text there
- o - Open up a new line following the current line and add text there

Copy

- ‘yy’ will copy the entire line.
  - Similarly ‘3yy’ will copy 3 lines and so on.
- ‘yw’ will copy words,
  - ‘7wy’ copies 7 words from the current cursor location.

COMMAND MODE

Cut and paste

- Press ‘dd’ cut/delete the entire line.

  - Similarly ‘3dd’ will cut/delete 3 lines.
  - ‘dw’ is used to cut/delete a word.
  - ‘x‘ delete character at cursor
  - ‘X’ delete character to left of cursor
  - ‘D’ delete from cursor to end of line
  - Press ‘p’ for paste

COMMAND MODE

Searching

- Command: /{letter}

- • For example /ink searches ‘ink’ a in file.
  - n repeats search in same direction
  - N repeats search in opposite direction

COMMAND MODE

Replace

- ’r’ replace one character at a time.
- “R” keep replacing characters until <esc> was pressed.

COMMAND MODE

Undo and Redo

- • u undo last change made in file.
- • U restores the current line.
- • Ctrl +r redo

COMMAND MODE

SAVING AND EXITING:

:wq	save the file and quits (same as ZZ)
:w	save file but not quit VI
:q	quits without saving
:q!	force quit
ZZ	quits VI and save edits

COMMAND MODE

MOVING THE CURSOR:

h, ← move cursor one position to the left

j ,↓ move cursor one position down

k ,↑ move cursor one position up

l ,→ move cursor one position to the right

COMMAND MODE

Keystrokes	Action
i	Insert at cursor **(goes into insert mode)**
a	Write after cursor **(goes into insert mode)**
A	Write at the end of line **(goes into insert mode)**
ESC	Terminate insert mode
u	Undo last change
U	Undo all changes to the entire line
o	Open a new line **(goes into insert mode)**
dd	Delete line
3dd	Delete 3 lines.
D	Delete contents of line after the cursor

COMMAND MODE

Keystrokes	Action
C	Delete contents of line after the cursor and insert new text.
dw	Delete word
4dw	Delete 4 words
cw	Change word
x	Delete character at cursor
r	Replace character
R	Overwrite characters from cursor onward
s	Substitute one character under cursor continue to insert
S	Substitute entire line and begin to insert at beginning of the line
~	Change case of individual character

COMMAND MODE

EXPERIMENT -4

FILE AND DIRECTORY PERMISSIONS IN UNIX

commands

- Study of Unix file system (tree structure), file and directory permissions, single and multiuser environment.
- b) Execution of File System Management Commands like ls, cd, pwd, cat, mkdir, rmdir, rm, cp, mv, chmod, wc, piping and redirection, grep, tr, echo, sort, head, tail, diff, comm, less, more, file, type, wc, split, cmp, tar, find, vim, gzip, bzip2, unzip, locate, etc.
- c) Execution of User Management Commands like who, whoami, su, sudo, login, logout, exit, passwd, useradd/adduser, usermod, userdel, groupadd, groupmod, groupdel, gpasswd, chown, chage, chgrp, chfn, etc.

commands

- ls,
  cd,
  pwd,
  cat,
- mkdir
  - creates a new directory
- rmdir
  - removed empty directory The **rmdir** command removes each and every directory specified in the command line only if these directories are empty.
- rm
  - **rm** is the **UNIX** command to **delete fil**es and, sometimes, directories.
  - It's short for “remove”.
  - Be very careful when deleting stuff with this command, as **UNIX** usually has no recycle bin or trash can

cd

- **cd** is used to change from one directory to another directory
- cd ~ → to home directory
- cd / → to root directory
- cd - → to last directory
- cd .. → to immediate parent directory
- cd <dir_name> or cd <path>
- pwd → know the path for current directory

- cp
  - This command is used to **copy** files or group of files or directory. It creates an exact image of a file on a disk with different file name.
  - **cp** command require at least two filenames in its arguments.
  - $ cp sample1.txt sample2.txt
- mv
  - mv is used to move one or more files or directories from one place to another in file system like UNIX.
  - It has two distinct functions:
    - **(i)** It rename a file or folder.
    - **(ii)** It moves group of files to different directory.
  - **$ mv a.txt geek.txt**
  - If the destination file **doesn't exist**, it will be created.
  - If the destination file **exist**, then it will be **overwrite** and the source file will be deleted.

  - mv works on files as well as directories

File Permissions

- The permissions of a file are the first line of defence in the security of a Unix system.

- **Owner permissions**
  - The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions**
  - The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions**
  - The permissions for others indicate what action all other users can perform on the file.

File Ownership

- Each file and/or directory in Linux is owned by a single user and belongs to a single group.
- The ownership details are assigned at the time the file or directory are created.

- user and group ownerships distinct;
  - it is possible for a user to own a file but not be a member of the owning group.

Users and Groups cont.

→ User information in `/etc/passwd`

→ Password info is in `/etc/shadow`

→ Group information is in `/etc/group`

Access Modes

- There are three access modes.
  - Read “r”
  - Write “w”
  - Execute “x”

The meanings of the above access modes differ for files and directories:

Files:

- Read: Access to view the file’s contents.
- Write: Access to change the contents.
- Execute: Access to execute the file (binary or shell script).

Directories:

- Read: Access to view the directory’s contents.
- Write: Access to change the directory’s contents (create or delete files)
- Execute: Access to enter the directory (with the “cd” command).

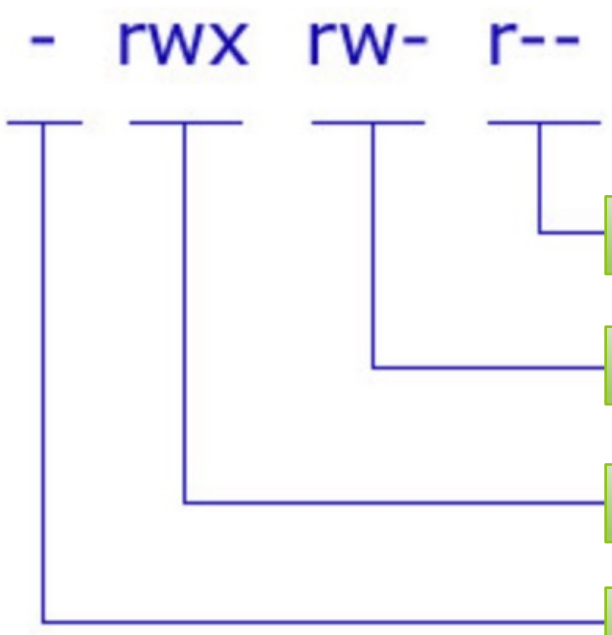
File permissions

chmod (change file modes)
chown (change file owner)
chgrp (change file group owner)

ls -l	all file permissions
ls -l myfile.txt	permissions for a particular file

Owner | Size in bytes | Date , time

```
drwxrwxr-x 2 himani himani 4096 Feb 21 19:44 lab
drwxrwxr-x 2 himani himani 4096 Feb 21 19:45 lab1
drwxrwxr-x 3 himani himani 4096 Feb 21 19:45 lab2
-rw-rw-r-- 1 himani himani   39 Feb 21 19:35 xyz.txt
```



- Only read permissions for all other users
- Read, write permissions for members of the group owning the file
- Read, write and execute permissions for the owner of the file
- FIle type: " - " means a file. " d " means a directory

Changing ownership

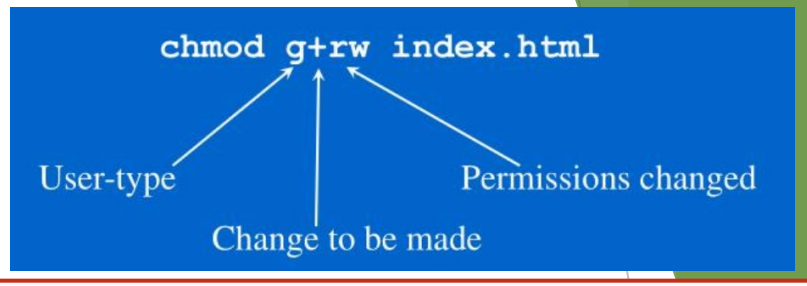
- The only ones allowed to change access modes on files and directories are the owners and the super-user (root).

  - In order to change Group ownership only, we'd use the following command:
    - **chgrp [groupname] [filename(s)]**
  - If we wish to change both user and group ownerships, we'd use:
    - **chown [username]:[groupname] [filename(s)]**

chmod (change mode)

- chmod changes the permissions of each given file according to mode, where mode describes the permissions to modify.
- Mode can be specified with octal numbers or with letters.

chmod [reference][operator][mode] file...



Reference	Class	Description
u	owner	file's owner
g	group	users who are members of the file's group
o	others	users who are neither the file's owner nor members of the file's group
a	all	All three of the above, same as ugo

Operator	Description
+	Adds the specified modes to the specified classes
-	Removes the specified modes from the specified classes
=	The modes specified are to be made the exact modes for the specified classes

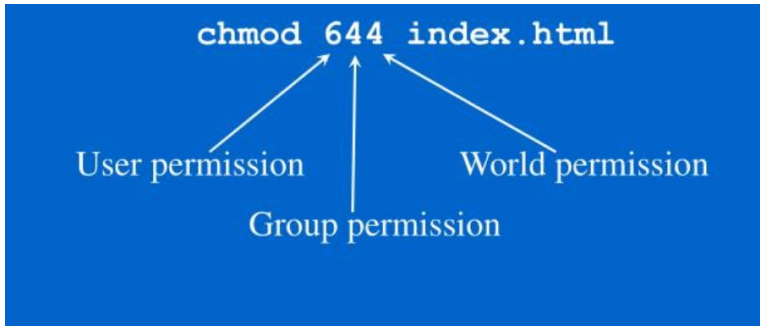
r	Permission to read the file.
w	Permission to write (or delete) the file.
x	Permission to execute the file, or, in the case of a directory, search it.

chmod

```
himani@himani-VirtualBox:~/Desktop/unix$ ls -l
total 24
-rw-rw-r-- 1 himani himani    8 Feb 21 20:22 abc.txt
drwxrwxr-x 2 himani himani 4096 Feb 21 19:44 lab
drwxrwxr-x 4 himani himani 4096 Feb 21 20:30 lab1
drwxrwxr-x 4 himani himani 4096 Feb 21 20:26 lab2
drwxrwxr-x 2 himani himani 4096 Feb 21 20:25 lab4
-rw-rw-rwx 1 himani himani   39 Feb 21 19:35 xyz.txt
himani@himani-VirtualBox:~/Desktop/unix$ chmod o-wx xyz.txt
himani@himani-VirtualBox:~/Desktop/unix$ ls -l
total 24
-rw-rw-r-- 1 himani himani    8 Feb 21 20:22 abc.txt
drwxrwxr-x 2 himani himani 4096 Feb 21 19:44 lab
drwxrwxr-x 4 himani himani 4096 Feb 21 20:30 lab1
drwxrwxr-x 4 himani himani 4096 Feb 21 20:26 lab2
drwxrwxr-x 2 himani himani 4096 Feb 21 20:25 lab4
-rw-rw-r-- 1 himani himani   39 Feb 21 19:35 xyz.txt
```

```
himani@himani-VirtualBox:~/Desktop$ ls -l hello.c
-rw-rw-r-- 1 himani himani 58 Feb 22 15:30 hello.c
himani@himani-VirtualBox:~/Desktop$ chmod o+wx hello.c
himani@himani-VirtualBox:~/Desktop$ ls -l hello.c
-rw-rw-rwx 1 himani himani 58 Feb 22 15:30 hello.c
```

Octal chmod



Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

- Any combination of the above numbers would set the file's permissions:
  - **644** = rw-r--r--
  - **755** = rwxr-xr-x
  - **700** = rwx------
  - **777** = rwxrwxrwx

default file permission : 644
644 :
6(110) (rw-) for user;
4(100) (r—) for group
4(100) (r—) for others (**general user**)

directory permission. : 755

Piping or redirection

```
}
himani@himani-VirtualBox:~/Desktop$ cat harsh.c | head -5
#include<stdio.h>
void X(char c)
{

char a= 'J';
himani@himani-VirtualBox:~/Desktop$ cat harsh.c | head -5 |tail -3
{

char a= 'J';
himani@himani-VirtualBox:~/Desktop$
```

- A pipe is a form of redirection .
- pipe send the output of one command/program/process to another command/program/process for further processing.
- Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command.

- Syntax :
  - command_1 | command_2 | command_3 | .... | command_N
  - $ ls -l | more (The more command takes the output of $ ls -l as its input.)
  - $ sort record.txt | uniq (sort the given file and print the unique values only.)
  - $ cat sample2.txt | head -7 | tail -5
    - (select first 7 lines through (head -7) command and that will be input to (tail -5) command which will finally print last 5 lines from that 7 lines.

wc command

- It is used to find out **number of lines**, **word count**, **byte and characters count** in the files specified in the file arguments.
- By default it displays **four-columnar output.**
  - First column shows number of lines present in a file specified,
  - second column shows number of words present in the file,
  - third column shows number of characters present in file and
  - fourth column itself is the file name which are given as argument.

```
himani@himani-VirtualBox:~/Desktop$ wc abc.txt
 49  48 220 abc.txt
```

grep

- Grep is used to search for a string of characters in a specified file.
- When it finds a match, it prints the line with the result.
- The grep command is handy when searching through large log files..
- Syntax:
  - **grep [options] pattern [files]**

Options Description

-c : This prints only a count of the lines that match a pattern
-h : Display the matched lines, but do not display the filenames.
-i : Ignores, case for matching
-l : Displays list of a filenames only.
-n : Display the matched lines and their line numbers.
-v : This prints out all the lines that do not matches the pattern
-e exp : Specifies expression with this option. Can use multiple times.
-f file : Takes patterns from file, one per line.
-E : Treats pattern as an extended regular expression (ERE)
-w : Match whole word
-o : Print only the matched parts of a matching line,with each such part on a separate output line.
-A n : Prints searched line and n lines after the result.
-B n : Prints searched line and n line before the result.
-C n : Prints searched line and n lines after before the result.

```
himani@himani-VirtualBox:~/Desktop$ cat hello.c
 #include<stdio.h>
void main()
printf("HELLO !! ");
{
}
himani@himani-VirtualBox:~/Desktop$ grep -c "(" hello.c
2
himani@himani-VirtualBox:~/Desktop$ grep -h "void" hello.c
void main()
himani@himani-VirtualBox:~/Desktop$ grep -i "vOId" hello.c
void main()
himani@himani-VirtualBox:~/Desktop$ grep -n "vOId" hello.c
himani@himani-VirtualBox:~/Desktop$ grep -n "void" hello.c
2:void main()
```

tr

- The tr command in UNIX is a command line utility for translating or deleting characters.

  -c : complements the set of characters in string. i.e.,
  operations apply to characters not in the given set
  -d : delete characters in the first set from the output.
  -s : replaces repeated characters listed in the set1 with single occurrence
  -t : truncates set1

- **convert lower case to upper case**
  - **$cat filename | tr “[a-z]” “[A-Z]”**
  - **or “[**
  - **$cat filename | tr “[:lower:]” “[:upper:]”**

```
himani@himani-VirtualBox:~/Desktop$ cat hello.c | tr "[a-z]" "[A-Z]"
 #INCLUDE<STDIO.H>
VOID MAIN()
PRINTF("HELLO !! ");
{
}
```

tr

- translate white-space to tabs
  - $ echo "Welcome To GeeksforGeeks" | tr [:space:] '\t’
- translate braces into parenthesis
  - $ tr '{}' '()' newfile.txt
- delete specified characters using -d option
  - $ echo "Welcome To GeeksforGeeks" | tr -d 'w’
- remove all the digits from the string
  - $ echo "my ID is 73535" | tr -d [:digit:]
- using -c option , remove all characters except digits
  - $ echo "my ID is 73535" | tr -cd [:digit:]

sort

- SORT command is used to sort a file, arranging the records in a particular order.
- By default, the sort command sorts file assuming the contents are ASCII.
- Using options in sort command, it can also be used to sort numerically.
- SORT command sorts the contents of a text file, line by line.
- It supports sorting alphabetically, in reverse order, by number, by month and can also remove duplicates.

```
himani@himani-VirtualBox:~/Desktop$ cat se.txt
Nidhi
Darshan
Dhairya
swati
Bhagvati
Uday
himani@himani-VirtualBox:~/Desktop$ sort se.txt
Bhagvati
Darshan
Dhairya
Nidhi
swati
Uday
himani@himani-VirtualBox:~/Desktop$ sort -r se.txt
Uday
swati
Nidhi
Dhairya
Darshan
Bhagvati
```

Sort options

- sort -b: Ignore blanks at the start of the line.
- sort -r: Reverse the sorting order.
- sort -o: Specify the output file.
- sort -n: Use the numerical value to sort.
- sort -M: Sort as per the calendar month specified.
- sort -u: Suppress lines that repeat an earlier key.
- sort -k POS1, POS2: Specify a key to do the sorting. POS1 and POS2 are optional parameters and are used to indicate the starting field and the ending field indices. Without POS2, only the field specified by POS1 is used. Each POS is specified as “F.C” where F represents the field index, and C represents the character index from the start of the field.
- sort -t SEP: Use the provided separator to identify the fields.

head

- print the top N number of data of the given input.
- By default, it prints the first 10 lines of the specified files.
- Syntax: **$ head state.txt**

- **-n num:**
  - Prints the first ‘num’ lines instead of first 10 lines.
- **-c num:**
  - Prints the first ‘num’ bytes from the file specified.
- **-q:**
  - It is used if more than 1 file is given.
- **-v:**
  - By using this option, data from the specified file is always preceded by its file name.

Short Options	Long Options
-n	--lines
-c	--bytes
-q	--quiet
-v	--verbose

```
himani@himani-VirtualBox:~/Desktop$ head -3 hello.c
 #include<stdio.h>
void main()
printf("HELLO !! ");
himani@himani-VirtualBox:~/Desktop$ tail -3 hello.c
printf("HELLO !! ");
{
}
```

tail

- It is the complementary of head command.
- The tail command, as the name implies, print the last N number of data of the given input.
- By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.
- Syntax: **$ tail state.txt**
  - **-n num:** Prints the last 'num' lines instead of last 10 lines.
  - **-c num:** Prints the last 'num' bytes from the file specified.
  - **-q:** It is used if more than 1 file is given. **$ tail -q state.txt capital.txt**
  - **-v:** By using this option, data from the specified file is always preceded by its file name.
  - **-version:** This option is used to display the version of tail which is currently running on your system.

Short Options	Long Options
-n	--lines
-c	--bytes
-q	--quiet
-v	--verbose
-f	--follow

head and tail

- **$ head -n 20 state.txt | tail -10**
- Display all recently modified or recently used files. **$ ls -t**
- Cut three most recently used file. **$ ls -t | head -n 3**
- **$ tail -n 7 state.txt | sort -r**
- **$ cat state.txt | head -n 20 | tail -n 5 > list.txt**

diff

- diff stands for **difference**.
- This command is used to display the differences in the files by comparing the files line by line.
- It tells us which lines in one file have is to be changed to make the two files identical.

$ cat a.txt
Gujarat
Uttar Pradesh
Kolkata
Bihar
Jammu and Kashmir

$ cat b.txt
Tamil Nadu
Gujarat
Andhra Pradesh
Bihar
Uttar pradesh

$ diff a.txt b.txt
0a1
➢ Tamil Nadu
2,3c3
➢ Uttar Pradesh
➢ Andhra Pradesh
5c5
➢ Uttar pradesh

0a1
means **after** lines 0(at the very beginning of file) you have to add **Tamil Nadu** to match the second file line number 1.

2,3c3 which means from line 2 to line 3 in the first file needs to be changed to match line number 3 in the second file. It then tells us those lines with the above symbols.

diff

4c2,means “Lines 4 in the first file needs to be changed in order to match lines 4 in the second file”

- The options of the result should be like this –
- **a** -Added the text to file
- **c** -Changes are made in the file
- **d** -Deletion operation is performed
- **<** Lines from the first file
- **>** Lines from the second file

2c2,means “Lines 2 in the first file needs to be changed in order to match lines 2 in the second file”

```
himani@himani-VirtualBox:~/Desktop$ cat a
I need to buy apples
I need to run the laundry
I need to wash the dog
I need to get the dog detailed
himani@himani-VirtualBox:~/Desktop$ cat b
I need to buy apples
I need to do the laundary
I need to wash the dog
I need to get the car detailed
himani@himani-VirtualBox:~/Desktop$ diff a b
2c2
< I need to run the laundry
---
> I need to do the laundary
4c4
< I need to get the dog detailed
---
> I need to get the car detailed
```

comm

- comm compare two sorted files line by line and write to standard output;
  - the lines that are common and the lines that are unique.

```
himani@himani-VirtualBox:~/Desktop$ cat se.txt
Nidhi
Darshan
Dhairya
swati
Bhagvati
Uday
himani@himani-VirtualBox:~/Desktop$ cat se1.txt
Nidhi
Raj
Dhairya
Tina
Shubh
Bhagvati
himani@himani-VirtualBox:~/Desktop$ comm <(sort se.txt) <(sort se1.txt)
                    Bhagvati
Darshan
                    Dhairya
                    Nidhi
            Raj
            Shubh
swati
            Tina
Uday
himani@himani-VirtualBox:~/Desktop$
```

The first is lines that are only in file one (se.txt), the second is lines only in file two (se1.txt), the third is line that are in both files.

comm options

Options	Function
-1	Does not display column 1 (does not display lines found only in file1).
-2	Does not display column 2 (does not display lines found only in file2).
-3	Does not display column 3 (does not display lines found in both files).
-i	Case insensitive comparison of lines.
– -check-order	Check the order of the input, even if all input lines are pairable
– -nocheck-order	Ignore the order of the input
– -output-delimiter=STR	delimates columns with delimeter “STR”
– -help	Displays a help menu
– -version	Display command version information

more

- **more** command is used to view the text files in the command prompt, displaying one screen at a time in case the file is large .
- The more command also allows the user do scroll up and down through the page.
- Options :
  - -d
  - -f
  - -p
  - -c
  - -s
  - -u
  - +num : This option displays the text after the specified number of lines of the document.

```
himani@himani-VirtualBox:~/Desktop$ more +3 hello.c
printf("HELLO !! ");
{
}
```

less

- *Less* is a program similar to *more* (1), but which allows backward movement in the file as well as forward movement.
- *less* does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like *vi*

-E : causes less to automatically exit the first time it reaches end of file.

-f : forces non-regular file to open.

-F : causes less to exit if entire file can be displayed on first screen

-g : highlight the string which was found by last search command

-G : suppresses all highlighting of strings found by search commands

-i : cause searches to ignore case

-n : suppresses line numbers

-p pattern : it tells less to start at the first occurrence of pattern in the file

-s : causes consecutive blank lines to be squeezed into a single blank line

file

- **file command** is used to determine the type of a file.

- *It has three sets of tests as follows:*
  - **filesystem test:** This test is based on the result which returns from a *stat* system call. The program verifies that if the file is empty, or if it's some sort of special file. This test causes the file type to be printed.
  - **magic test:** These tests are used to check for files with data in particular fixed formats.
  - **language test:** This test search for particular strings which can appear anywhere in the first few blocks of a file

type

- The **type** command is used to describe how its argument would be translated if used as commands.
- It is also used to find out whether it is built-in or external binary file.
- type [Options] command names
- Options
  - -a : This option is used to find out whether it is an alias, keyword or a function and it also displays the path of an executable, if available.
  - -t : This option will display a single word as an output.alias – if command is a shell alias
    - keyword – if command is a shell reserved word
    - builtin – if command is a shell builtin
    - function – if command is a shell function
    - file – if command is a disk file
  - -p :This option displays the name of the disk file which would be executed by the shell. It will return nothing if the command is not a disk file.

WC

print the number of newlines, words, and bytes in files

Tag	Description
-c, **--bytes**	
	print the byte counts
-m, **--chars**	
	print the character counts
-l, **--lines**	
	print the newline counts
-L, **--max-line-length**	
	print the length of the longest line
-w, **--words**	
	print the word counts
--help	display this help and exit
--version	
	output version information and exit

commands

- split : Split command in Linux is used to **split large files into smaller files**.
- cmp: **cmp** command in Linux/UNIX is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.
- tar: The **tar command** stands for tape achieve, which is the most commonly used tape drive backup **command** used by the Linux/**Unix** system. It allows for you to quickly access a collection of files and placed them into a highly compressed archive file commonly called tarball, or **tar**, gzip, and bzip Linux.
- find: The **find** command in UNIX is a command line utility for walking a file hierarchy.
- vim: **Vim** is a text editor that is upwards compatible to Vi. It can be used to edit all kinds of plain text. It is especially useful for editing programs.
- gzip: gzip command compresses files.
- bzip2: **bzip2 command** in Linux is used to compress and decompress the files i.e. it helps in binding the files into a single file which takes less storage space as the original file use to take. It has a slower decompression time and higher memory use.
- unzip,
- locate : *locate* command in Linux is used to find the files by name.

Experiment – 5

Execution of User Management Commands

commands

Experiment – 6

Basics of Shell Scripting

Shell

- A shell is special user program which provide an interface to user to use operating system services.
- Shell accept human readable commands from user and convert them into something which kernel can understand.
- It is a command language interpreter that execute commands read from input devices such as keyboards or from files.
- The shell gets started when the user logs in or start the terminal.

Shell Scripting

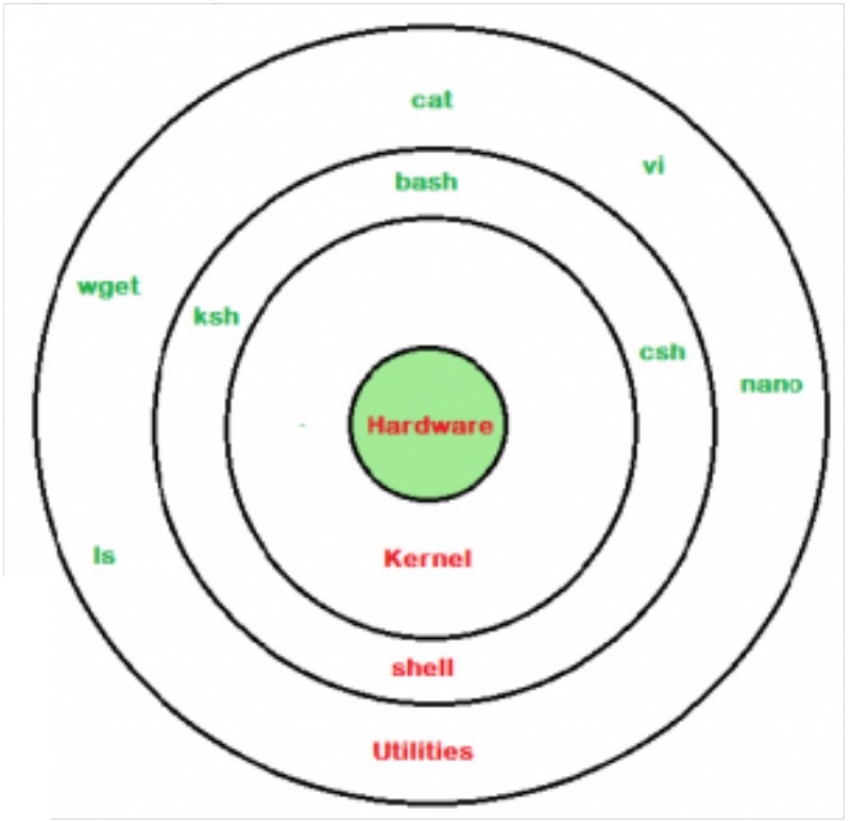
- Shell is broadly classified into two categories –
  - Command Line Shell
  - Graphical shell

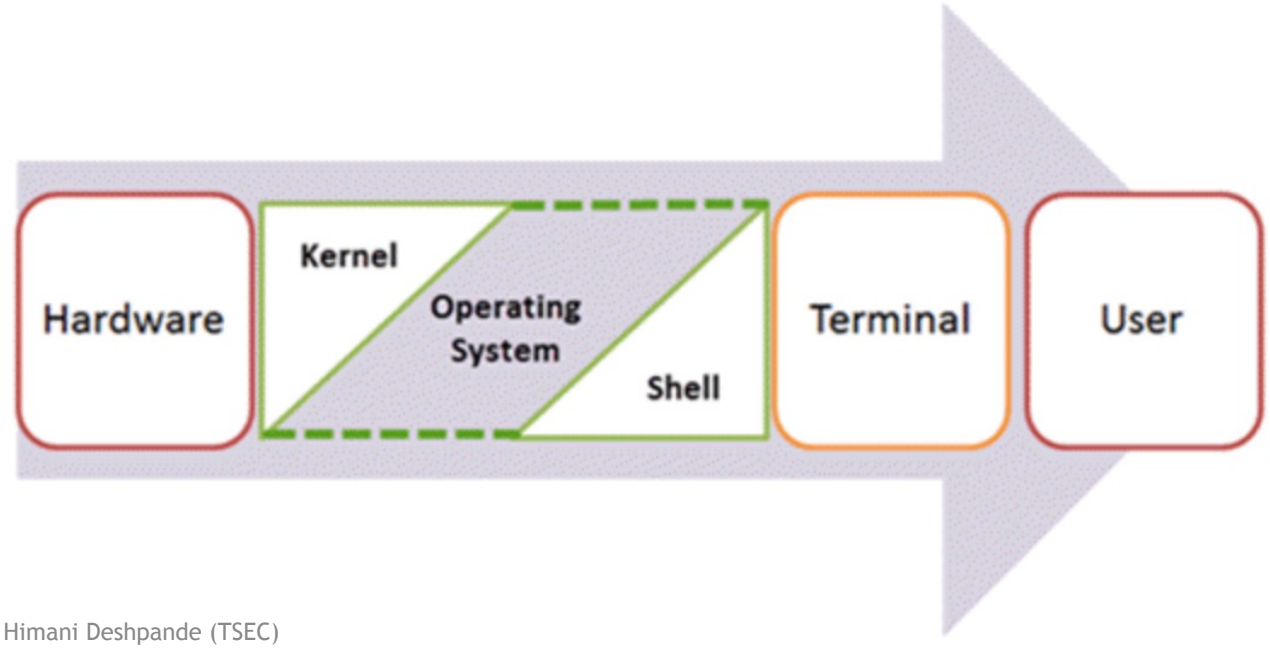
Kernel

- The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system.

- It manages following resources of the Linux system –
  - File management
  - Process management
  - I/O management
  - Memory management
  - Device management etc.

shell





Components of Shell Program

Shells

- There are several shells are available for Linux systems like –
  - ● BASH (Bourne SHell)
  - ● CSH (C SHell)
  - ● KSH (Korn SHell)

Bounrne / BASH shell

- It is most widely used shell in Linux systems.
- It was developed in 1977 by Stephen Bourne of AT&T and introduced in UNIX Version 7, replacing the Mashey shell (sh).
- The Bourne shell is considered the primary shell in scripts. All UNIX systems have it
- The prompt for this shell is $.
- It is used as default login shell in Linux systems and in macOS.
- It can also be installed on Windows OS.

CSH (C SHell)

- The C shell's syntax and usage are very similar to the C programming language.
- The prompt for this shell is %,

KSH (Korn SHell)

- The Korn Shell also was the base for the POSIX Shell standard specifications etc.

Shell Scripting

- ● Usually shells are interactive that mean, they accept command as input from users and execute them.

- ● However some time we want to execute a bunch of commands routinely, so we don’t want to type in all commands each time in terminal.

Shell Scripting

- As shell can also take commands as input from file we can write these commands in a file and can execute them in shell to avoid this repetitive work.
- These files are called Shell Scripts or Shell Programs. Shell scripts are similar to the batch file in MS-DOS.
- Each shell script is saved with .sh file extension eg. myscript.sh
- A shell script have syntax just like any other programming language.

Shell script

- ● A shell script comprises following elements –
  - ● Shell Keywords – if, else, break etc.
  - ● Shell commands – cd, ls, echo, pwd, touch etc.
  - ● Functions
  - ● Control flow – if..then..else, case and shell loops etc.

Shell Scripts (1)

- Basically, a shell script is a text file with Unix commands in it.

- Shell scripts usually begin with a #! and a shell name
  - For example: **#!/bin/bash**
  - If they do not, the user's current shell will be used

- Any Unix command can go in a shell script
  - Commands are executed in order or in the flow determined by control statements.

Why Shell Script

There are many reasons to write shell scripts –

- To avoid repetitive work and automation.
- System admins use shell scripting for routine backups.
- System monitoring.
- Adding new functionality to the shell etc.

Advantages of shell scripts

- The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax.
- ● Writing shell scripts are much quicker.
- ● Quick start.
- ● Interactive debugging etc.

Disadvantages of shell scripts :

- Prone to costly errors, a single mistake can change the command which might be harmful.
- Slow execution speed.
- Design flaws within the language syntax or implementation.
- Not well suited for large and complex task.
- Provide minimal data structure unlike other scripting languages.

expr

The **expr command** in Unix evaluates a given **expression** and displays its corresponding output.

It is **used** for: Basic operations like addition, subtraction, multiplication, division, and modulus on integers.

```
Himanis-MacBook-Pro-2:UNIXLab himanideshpande$ x=4
Himanis-MacBook-Pro-2:UNIXLab himanideshpande$ y=6
Himanis-MacBook-Pro-2:UNIXLab himanideshpande$ expr $x + $y
10
```

- The expr command supports the following operators:
  - **for integer:** addition, subtraction, multiplication, division, and modulus.
  - **for strings:** regular expression, set of characters in a string.

- expr length Hello

The example below increments the $count variable value to 1 inside the shell script.

- echo $count
  count=`expr $count + 1`

Compare two number

```
Himanis-MacBook-Pro-3:~ himanideshpande$
expr $x = $y
0
```

String matching

```
Himanis-MacBook-Pro-3:~ himanideshpande$
expr Himani : Hima
4
```

expr

- x= 15
- y=`expr $x + 10`
- echo $y

```
Himanis-MacBook-Pro-3:~ himanideshpande$ x=10
Himanis-MacBook-Pro-3:~ himanideshpande$ y=`expr $x + 2`
Himanis-MacBook-Pro-3:~ himanideshpande$ echo $y
12
```

Scripting key rules

- Comments should be preceded with #.
- Comment split over multiple lines must have # at the beginning of each line.
- Multiplication symbol must always be prceded by \.

Shell types in OS

cat /etc/shells

To check which shells your OS supports

```
user@instant-contiki:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
```

```
Himanis-MacBook-Pro:~ himanideshpande$ cat /etc/shells
# List of acceptable shells for chpass(1).
# Ftpd will not allow users to connect who are not using
# one of these shells.

/bin/bash
/bin/csh
/bin/dash
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
Himanis-MacBook-Pro:~ himanideshpande$ which bash
/bin/bash
```

Shell script

- ***Shell Script are interpreted and not compiler.***
- Extension -- > .sh
- It is optional to use .sh extension with shell files .
- It is good practise to use .sh extension as it gets easy for editor to understand the type of file.

Execute abc.sh file :

```
./abc.sh
or
bash abc.sh
```

No of lines in a file

```
#!/bin/bash
echo " the number of lines in unix.txt : "
echo $(wc unixlab.txt)
```

Shell script

```
#!/bin/bash
echo -n "Hello!" #-n suppresses the new line print
echo "You are welcome"
echo "we are working in directory $(pwd)"
echo "todays date is $(date)"
```

```
Himanis-MacBook-Pro-2:UNIXLab himanideshpande$ ./abc.sh
Hello!You are welcome
we are working in directory
/Users/himanideshpande/Desktop/Himani/OS_2021/UNIXLab
todays date is Mon Feb 14 21:03:00 IST 2022
```

Arguments

- **$0** is the name of the script itself (script.sh)
- **$1** is the first argument (filename1)
- **$2** is the second argument (dir1)
- **$9** is the ninth argument
- **${10}** is the tenth argument and must be enclosed in brackets after $9.
- **${11}** is the eleventh argument.

- **$#** is the count of the number of arguments.
- **$*** represents all command line arguments.

Name of executable script is stored in $0.

```
#!/bin/bash
echo "Script Name : $0"
echo "The number of parameters are $#"
echo "The parameter are $*"
echo "The parameters are $1 $2 $3"
echo "The shell script command is $0"
```

```
Himanis-MacBook-Pro:UNIXLab himanideshpande$ bash var.sh 1 2 3 4
Script Name : var.sh
The number of parameters are 4
The parameter are 1 2 3 4
The parameters are 1 2 3
The shell script command is var.sh
```

Add the values passed as arguments to file

```
#!/bin/bash
echo "Script Name : $0"
echo "The number of parameters are $#"
echo "The parameter are $*"
echo "The parameters are $1 $2 $3 $4"
echo "Sum is $(expr $1 + $2 + $3 + $4)"
echo "The shell script command is $0"
```

Reading input

```
#!/bin/bash
echo -n "Enter your first name "
read f
echo -n "Enter your last name "
read l
echo "Your name is $f $l"
```

```
Himanis-MacBook-Pro-2:UNIXLab
himanideshpande$ ./read.sh
Enter your first name Himani
Enter your last name Deshpande
Your name is Himani Deshpande
```

Write a shell script to print current system date.

```
#!/bin/bash
m=$(date +%d/%m/%y)
echo "Current system date is $m"
```

IF statement in shell programming

- syntax:
  if [logical expression]
- then
  ..
- .. else
- .. fi

for Loops

Syntax:

```
for variable in list-of-variables
do
    command1
    command2 ..
done
```

For loop

```
for x in 1 2 3 4 5
do
    echo "The value of x is $x"
done
```

For loop different syntax

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
 echo "Welcome $c times"
done
```

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

```
#!/bin/bash for (( ; ; ))
do
echo "infinite loops [ hit CTRL+C to stop]"
done
```

```
#0 to 10  with a gap of two
for i in {0..10..2}
do
    echo "Welcome $i times”
 done
```

Welcome 0 times
Welcome 2 times
Welcome 4 times
Welcome 6 times
Welcome 8 times
Welcome 10 times

operators

- = for string comparisons,
- -eq is for numeric ones.
- -eq is in the same family as -lt, -le, -gt, -ge, and -ne

While loop

while loop in Shell programming:

syntax:

while [logical expression]

do

..

.. done

While loop

```
#!/bin/bash
num=0
while [ $num –lt 10 ]
do
        echo $num
         num=$[$num+1]
done
```

Test brackets [ ], [[ ]]

- **Test - return the binary result of an expression:**
  [[ ]] ,
  [[expression]]

- [ ] : -eq, -lt etc for comparison
- [[ ]] : can use operators <,> ,<= ,>= etc.

brackets

- **Group commands in a sub-shell: ( )**
  (list)

- **Group commands in the current shell: { }**
  { list; }

- **Arithmetic expansion**
  The format for Arithmetic expansion is:
  $((expression))

- The format for a simple Arithmetic Evaluation is:
  ((expression))

- **Combine multiple expressions**
  (expression)
  ((expr1 && expr2))

Arithmetic Operations using double parenthesis (( ))

- Double Parenthesis is a built-in arithmetic feature of Bash shell.
- It is used to perform integer arithmetic operations.
- In this, you do not have to use $variable to access a value of the variable inside it.
- It does not misinterpret multiplication and other symbols.
- It ignores the importance of spaces.

Arithmetic Operations using square brackets [ ]

- You can only perform integer arithmetic operations using square brackets.
- In this, it is not necessary to prefix a variable with $ to access a value of the variable inside it.
- The advantage of using square brackets is that you do not need to use escape character to prevent misinterpretation of multiply and other symbols.
- It also ignores the importance of spaces.

Untill loop

- The untill loop is used for repeating the set of instructions **for the time the specified logical expression is false.** The moment the logical expression becomes true, the control will come outof loop.

Syntax:

until logical_expression do
..
... done

Write a shell script to print sum of even numbers upto 50

```
#!/bin/bash
s=0
n=2
until [ $n -gt 50 ]
do
s=$(( $s + $n ))
((n+=2))
echo $n
done
echo $s
```

```
Himanis-MacBook-Pro:UNIXLab himanideshpande$ bash untill.sh
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
650
```

Write a shell script to display numbers from 1 to 10

```
#!/bin/bash
n=1
while [ $n -le 10 ]
do
    echo $n
   (( n++ ))
done
```

Assignment :

Write a shell script to print sum of odd numbers upto 21

Print table of a number entered as file argument using while loop

```
1 echo "Enter a number: "
2 read number
3 i=1
4 while [ $i -le 10 ]
5 do
6   echo "$number x $i = $((number*i))"
7   i=$((i+1))
8 done
```

```
Enter a number: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Switch case

```
case word in
   pattern1)
      Statement(s) to be executed if pattern1 matches
      ;;
   pattern2)
      Statement(s) to be executed if pattern2 matches
      ;;
   pattern3)
      Statement(s) to be executed if pattern3 matches
      ;;
   *)
     Default condition to be executed
     ;;
esac
```

```
case "$1" in

1)  echo "Sending SIGHUP signal"
    kill -SIGHUP $2
    ;;
2)  echo  "Sending SIGINT signal"
    kill -SIGINT $2
    ;;
3)  echo  "Sending SIGQUIT signal"
    kill -SIGQUIT $2
    ;;
9) echo  "Sending SIGKILL signal"
   kill -SIGKILL $2
   ;;
*) echo "Signal number $1 is not processed"
   ;;
esac
```

expr command

- The **expr** command in Unix evaluates a given expression and displays its corresponding output.

- It is used for:
  - Basic operations like addition, subtraction, multiplication, division, and modulus on integers.
  - Evaluating regular expressions, string operations like substring, length of strings etc.

calculator

```
#!/bin/bash
sum=0 i="y"
echo " Enter one no."
read n1
echo "Enter second no."
read n2
while [ $i = "y" ]
do
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "Enter your choice"
read ch
```

```
case $ch in
 1)sum= $(expr $n1 + $n2)
 echo "Sum =" $sum;;

 2)sum=$(expr $n1 - $n2)
 echo "Sub = "$sum;;

 3)sum=$(expr $n1 \* $n2)
 echo "Mul = " $sum;;

 4)sum=$(expr $n1 / $n2)
 echo "Div = "$sum;;

*)echo "Invalid choice";;
esac

echo "Do u want to continue ?"
read i

if [ $i != "y" ]
then
  exit
fi
done
```

```
[Himanis-MacBook-Pro:UNIXLab himanides
 Enter one no.
4
Enter second no.
2
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice
3
Mul =  8
Do u want to continue ?
y
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice
1
calculator.sh: line 16: 6: command not found
Sum = 8
Do u want to continue ?
n
Himanis-MacBook-Pro:UNIXLab himanideshpande$
```

Write a program to print all files/directories in a current working directory.

```
#!/bin/bash
d=$(pwd)
echo "current working directory is $d"
l= $(ls $d)
for x in $l
do
        echo "The file name is $x"
done
```

```
Himanis-MacBook-Pro:UNIXLab himanideshpande$ bash files.sh
CUrrent working directory is /Users/himanideshpande/Desktop/OS/UNIXLab
The file name is AWK
The file name is in
The file name is UNIX.pdf
The file name is Assessment_UnixLab.xlsx
The file name is Assg1.docx
The file name is Experiment
The file name is List_CO_Mapped-OS.docx
The file name is Grep
The file name is command
The file name is in
The file name is Unix.pdf
The file name is Shell
The file name is Programing
The file name is in
The file name is UNIX.pdf
The file name is UNIX
The file name is LAB.pptx
The file name is UNIXCommands.pdf
The file name is Unix
The file name is administrative
The file name is tasks
The file name is commands-1.pdf
The file name is Unix_Ex4.pdf
The file name is Unix_Lab.pdf
The file name is abc.sh
The file name is count
The file name is count.sh
The file name is files.sh
The file name is installingLinuxVirtualBox.docx
The file name is installingLinuxVirtualBox.pdf
The file name is linuxcommands.pdf
The file name is read.sh
The file name is sed
The file name is in
The file name is Unix.pdf
The file name is unixlab.txt
The file name is var.sh
The file name is vi-Editor.pptx
The file name is vi_Editor.pdf
The file name is ~$Assessment_UnixLab.xlsx
The file name is ~$UNIX
```

Write a program to print all files/directories in a current working directory.

```
#!/bin/bash
l=$(pwd|ls)
echo "the list of files and directories in current working directory are $l"
```

OR

```
#!/bin/bash
for l in `ls $pwd` do
echo "$l"
done
```

OR

```
#!/bin/bash for l in $(ls)
do
echo "$l"
done
```

```
#!/bin/bash
d=$(pwd)
echo "current working directory is $d" l= $(ls $d)
for x in $l
do
        echo "The file name is $x"
done
```

Write a shell script to display all files and directories starting with letter 'b'

```
#!/bin/bash
for l in ls b*
do
echo "the list of files and directories in current working directory are $l"
done
```

Write a shell script to display names of .sh files starting with h

```
#!/bin/bash
for l in `ls h*.sh` do
echo "$l"
done
```

Write a shell script to display contents of .sh files starting with h

```
#!/bin/bash
for l in ls h*.sh
 do
cat $l
done
```

Arrays in script

```
#!/bin/bash

array=( dog cat bat mouse );

echo "${array[1]}"
```

cat

```
#!/bin/bash

array=( dog cat bat mouse );

echo "${array[0]}"
echo "${array[2]}"
echo "${array[3]}"
echo "${array[1]}"
```

Dog
Bat
Mouse
cat

```
echo "${array[@]}"
```

Reading elements of an array

```
#!/bin/bash
index_array=(1 2 3 4 5 6 7 8 9 0)

for i in ${index_array[@]}
do
        echo $i
done
```

```
root@ubuntu:~# bash for-loop.sh
1
2
3
4
5
6
7
8
9
0
```

echo ${assoc_array[@]}	Access All Elements of an Array
echo ${#index_array[@]}	Count the Number of Elements in an Array
unset index_array[1]	Delete Individual Array Elements

Display contents of pwd

```
#!/bin/bash
d=$(pwd)
echo "CUrrent working directory is $d"
l=$(ls $d)


for x in $l
do
echo "The file name is $x"
done
```

```
Himanis-MacBook-Pro-2:UNIXLab himanideshpande$ ./files.sh
CUrrent working directory is
/Users/himanideshpande/Desktop/Himani/OS_2021/UNIXLab
The file name is AWK
The file name is in
The file name is UNIX.pdf
The file name is Assessment_UnixLab.xlsx
The file name is Assg1.docx
The file name is Experiment
The file name is List_CO_Mapped-OS.docx
The file name is Grep
The file name is command
The file name is in
The file name is Unix.pdf
The file name is PERL
The file name is Scripting.pdf
The file name is Shell
The file name is Programing
```

Create an array of strings for months of an year. Enter the number of month and display the name of month.

Process Management

a) Execution of Process Management Commands like ps, pstree, nice, kill, pkill, killall, xkill, fg, bg, pgrep, renice, etc.

b) Execution of Memory Management Commands like free, /proc/meminfo, top, htop, df, du, vmstat, demidecode, sar, pagesize, etc.

ps

- "ps" command in Linux is an abbreviation of "process status"
- **check the status of active processes on a system**

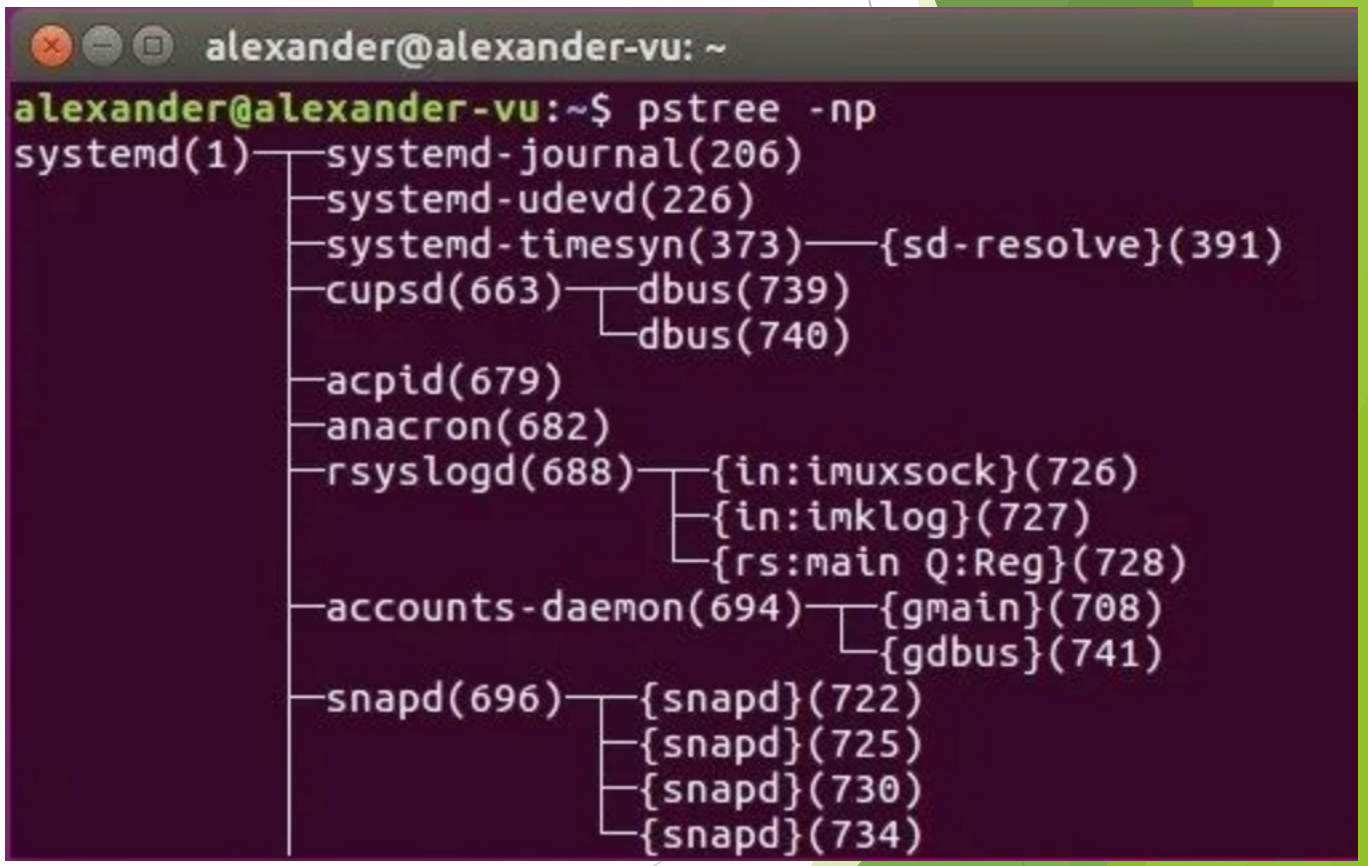
```
Himanis-MacBook-Pro-3:~ himanideshpande$
ps
  PID TTY           TIME CMD
33339 ttys000    0:00.03 -bash
```

- **Display All the Processes Associated with the Current Terminal**

```
Himanis-MacBook-Pro-3:~ himanideshpande$ ps -T
  PID TTY           TIME CMD
33338 ttys000    0:00.02 login -pf
himanideshpande
33339 ttys000    0:00.03 -bash
33376 ttys000    0:00.00 ps -T
```

pstree

- pstree is a **Linux command that shows the running processes as a tree.**
- It is used as a more visual alternative to the ps command.
- The root of the tree is either init or the process with the given pid.
- It can also be installed in other Unix systems.



nice

- The nice command lets **you run a command at a priority lower than the command's normal priority**.
- **nice** runs command *COMMAND* with an adjusted "niceness", which affects process scheduling.
- A process with a lower niceness value is given higher priority and more CPU time.
- Niceness values range from **-20** (most favorable to the process) to **19** (least favorable to the process)

nice

- To check the nice value of a process.
  - ps -el | grep terminal

```
manav@ubuntulinux:~$ ps -el | grep terminal
0 S  1000   77982    1632  1  80   0 - 241446 poll_s ?       00:00:00 gnome-term
inal-
```

- The eight highlighted value is the nice value of the process.

```
Himanis-MacBook-Pro-3:~ himanideshpande$ ps -el | grep terminal
  501 33526 33339        6   0  46  0 34153072    608 -      U+                0 ttys000    0:00.00 grep
terminal
```

renice

- changing priority of the running process.
  - sudo renice -n 15 -p 77982
- To change the priority of all programs of a specific group.
  - renice -n 10 -g 4

```
manav@ubuntulinux:~$ sudo renice -n 10 -g 1
1 (process group ID) old priority 0, new priority 10
```

- To change the priority of all programs of a specific user.
  - sudo renice -n 10 -u 2

```
manav@ubuntulinux:~$ sudo renice -n 10 -u 0
0 (user ID) old priority 0, new priority 10
```

Setting priority of process

- To set the priority of a process
  - nice -10 gnome-terminal
- To set the negative priority for a process
  - nice --10 gnome-terminal

nice & renice

- **nice command** in Linux helps in execution of a program/process with modified scheduling priority.
- It launches a process with a user-defined scheduling priority.
- In this, if we give a process a higher priority, then Kernel will allocate more CPU time to that process.

- Whereas the **renice command** allows you to change and modify the scheduling priority of an already running process.
- Linux Kernel schedules the process and allocates CPU time accordingly for each of them.

kill

- It is used for manually terminating the processes.
- We can use the kill command along with the **-l** option for getting the list of every available signal:

```
Himanis-MacBook-Pro-3:~ himanideshpande$ kill -l
 1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL
 5) SIGTRAP 6) SIGABRT 7) SIGEMT 8) SIGFPE
 9) SIGKILL 10) SIGBUS 11) SIGSEGV 12) SIGSYS
13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGURG
17) SIGSTOP 18) SIGTSTP 19) SIGCONT 20) SIGCHLD
21) SIGTTIN 22) SIGTTOU 23) SIGIO 24) SIGXCPU
25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH
29) SIGINFO 30) SIGUSR1 31) SIGUSR2
```

pkill

kill pkill killall

- kill terminates processes based on Process ID number (PID),
- while the killall and pkill commands terminate running processes based on their names and other attributes.

fg

- fg command, **short for the foreground**,
- is a command that moves a background process on your current Linux shell to the foreground.
- This contrasts the bg command, short for background, that sends a process running in the foreground to the background in the current shell.

bg

- bg command in linux is **used to place foreground jobs in background.**

EXPERIMENT : 8

AWK

AWK

- Awk is a scripting language used for **manipulating data and generating reports.**
- The awk command programming language requires no compiling, and allows the user to use variables, numeric functions, string functions, and logical operators.
- Awk is mostly used for pattern scanning and processing.
- It searches one or more files to see if they contain lines that matches with the specified patterns and then performs the associated actions.
- Awk is abbreviated from the names of the developers – Aho, Weinberger, and Kernighan.

WHAT CAN WE DO WITH AWK ?

- **1. AWK Operations:**
  (a) Scans a file line by line
  (b) Splits each input line into fields
  (c) Compares input line/fields to pattern
  (d) Performs action(s) on matched lines
- **2. Useful For:**
  (a) Transform data files
  (b) Produce formatted reports
- **3. Programming Constructs:**
  (a) Format output lines
  (b) Arithmetic and string operations
  (c) Conditionals and loops

Syntax:
awk options 'selection _criteria {action }' input-file > output-file

Bank.txt

101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina 30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving

awk '{print}' bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{print}' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Print first 3 fields (acc no., name and balance) from bank.lst
(field 1 is referred by $1 and so on.)

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{print $1 $2 $3}' bank.txt
101ADITYA0
102Anil10000
103Naman0
104Ram10000
105Jyotsna5000
106Mukesh14000
107Vishal14500
108Chirag0
109Arya16000
110Priya130
201Bina3000
202Diya4000
203Gargi2000
204Hina30000
205Kalpana4000
301Nikhil7777
```

awk '{print $1 $2 $3}' bank.txt

Separate fields by tabs.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{print $1 "\t" $2 "\t" $3}' bank.txt
101     ADITYA  0
102     Anil    10000
103     Naman   0
104     Ram     10000
105     Jyotsna 5000
106     Mukesh  14000
107     Vishal  14500
108     Chirag  0
109     Arya    16000
110     Priya   130
201     Bina    3000
202     Diya    4000
203     Gargi   2000
204     Hina    30000
205     Kalpana 4000
301     Nikhil  7777
```

awk '{print $1 "\t" $2 "\t" $3}' bank.txt

Format specifiers for the field can be specified as below:

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{printf "%3d \t %-15s \t %7d \n", $1, $2, $3}' bank.txt
101      ADITYA                          0
102      Anil                        10000
103      Naman                           0
104      Ram                         10000
105      Jyotsna                      5000
106      Mukesh                      14000
107      Vishal                      14500
108      Chirag                          0
109      Arya                        16000
110      Priya                         130
201      Bina                         3000
202      Diya                         4000
203      Gargi                        2000
204      Hina                        30000
205      Kalpana                      4000
301      Nikhil                       7777
```


awk '{printf "%3d \t %-15s \t %7d \n", $1, $2, $3}' bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{printf "%7d \t %-7s \t %-7d \n",$1, $2, $3}' bank.txt
    101          ADITYA          0
    102          Anil            10000
    103          Naman           0
    104          Ram             10000
    105          Jyotsna         5000
    106          Mukesh          14000
    107          Vishal          14500
    108          Chirag          0
    109          Arya            16000
    110          Priya           130
    201          Bina            3000
    202          Diya            4000
    203          Gargi           2000
    204          Hina            30000
    205          Kalpana         4000
    301          Nikhil          7777
```

awk '{printf "%7d \t %-7s \t %-7d \n",$1, $2, $3}' bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{printf "%-13d \t %-10s \t %-7d \n",$1, $2, $3}' bank.txt
101           ADITYA        0
102           Anil          10000
103           Naman         0
104           Ram           10000
105           Jyotsna       5000
106           Mukesh        14000
107           Vishal        14500
108           Chirag        0
109           Arya          16000
110           Priya         130
201           Bina          3000
202           Diya          4000
203           Gargi         2000
204           Hina          30000
205           Kalpana       4000
301           Nikhil        7777
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '{printf "%13d \t %-10s \t %-7d \n",$1, $2, $3}' bank.txt
          101    ADITYA        0
          102    Anil          10000
          103    Naman         0
          104    Ram           10000
          105    Jyotsna       5000
          106    Mukesh        14000
          107    Vishal        14500
          108    Chirag        0
          109    Arya          16000
          110    Priya         130
          201    Bina          3000
          202    Diya          4000
          203    Gargi         2000
          204    Hina          30000
          205    Kalpana       4000
          301    Nikhil        7777
```

Print only those records having 'current' account.

- awk '/current/ {print}' bank.txt

  OR

  awk '/current/' bank.txt

  OR

- awk /current/ bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '/current/ {print}' bank.txt
103 Naman 0 20/08/2009 current
203 Gargi 2000 21/01/2015 current
```

Printing records having 'current' account. $0 means entire line.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '/current/ {print $0}' bank.txt
103 Naman 0 20/08/2009 current
203 Gargi 2000 21/01/2015 current
```

```
awk '/current/ {print $0}' bank.txt
```

Printing individual fields of file.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '/current/ {print $1}' bank.txt
103
203
```

awk '/current/ {print $1}' bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '/current/ {print $2}' bank.txt

Naman
Gargi
```

awk '/current/ {print $2}' bank.txt

Print records having balance less than 5000. (here $3 represents the 3rd field balance)

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$3<5000' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
103 Naman 0 20/08/2009 current
108 Chirag 0 15/12/2012 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
205 Kalpana 4000 8/9/2007 Current
```

awk '$3<5000' bank.txt

'OR'ing two conditions.
Print records having balance less than 5000 or more than 10000

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$3<5000 || $3>10000' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
```

awk '$3<5000 || $3>10000' bank.txt

Print records whose acc. type is not 'saving'

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$5 !~/saving/' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
205 Kalpana 4000 8/9/2007 Current
```

awk '$5 !~/saving/' bank.txt

Print records whose date of opening is 20 and the year is 09.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$4 ~ /^20.*09/' bank.txt
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
```

awk '$4 ~ /^20.*09/' bank.txt

NR : Gives the record No.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$4 ~ /^1/{print NR,$0}' bank.txt
1  101 ADITYA 0 14/11/2000 CURRENT
4  104 Ram 10000 15/08/2010 saving
5  105 Jyotsna 5000 16/06/2012 saving
8  108 Chirag 0 15/12/2012 Current
9  109 Arya 16000 14/12/2010 Current
10 110 Priya 130 16/11/2009 Saving
11 201 Bina 3000 11/03/2010 saving
12 202 Diya 4000 13/04/2018 Saving
14 204 Hina  30000 14/02/2014 saving
```

awk '$4 ~ /^1/{print NR,$0}' bank.txt

NF: count of fields/column for each record

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$4 ~ /^1/{print NF,$0}' bank.txt
5 101 ADITYA 0 14/11/2000 CURRENT
5 104 Ram 10000 15/08/2010 saving
5 105 Jyotsna 5000 16/06/2012 saving
5 108 Chirag 0 15/12/2012 Current
5 109 Arya 16000 14/12/2010 Current
5 110 Priya 130 16/11/2009 Saving
5 201 Bina 3000 11/03/2010 saving
5 202 Diya 4000 13/04/2018 Saving
5 204 Hina  30000 14/02/2014 saving
```

```
awk '$4 ~ /^1/{print NF,$0}' bank.txt
```

Performing arithmetic operation: Print customer name, balance, date and 5% interest on balance for “saving” account.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk '$5 == "saving" { printf "%20s
%d %20s %f \n",$2, $3, $4, $3*0.05}' bank.txt
                Anil 10000            20/05/2011 500.000000
                 Ram 10000            15/08/2010 500.000000
             Jyotsna 5000            16/06/2012 250.000000
              Vishal 14500            30/11/2011 725.000000
                Bina 3000            11/03/2010 150.000000
                Hina 30000            14/02/2014 1500.000000
              Nikhil 7777              8/9/1999 388.850000
```

awk '$5 == "saving" { printf "%20s %d %20s %f \n",$2, $3, $4, $3*0.05}’ bank.txt

BEGIN and END keywords:

- If u have something to print before processing the first line, for e.g. **a heading** then the BEGIN section is used.
- Similarly, the END section is useful in printing something after processing is over. They are optional.

- Syntactical form:
  awk ' BEGIN{actions}

  /pattern/ {actions}

  END{actions}' file_name

BEGIN

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk 'BEGIN{ printf "Records in the
bank are :\n"}{print $1, $2, $3}' bank.txt
Records in the bank are :
101 ADITYA 0
102 Anil 10000
103 Naman 0
104 Ram 10000
105 Jyotsna 5000
106 Mukesh 14000
107 Vishal 14500
108 Chirag 0
109 Arya 16000
110 Priya 130
201 Bina 3000
202 Diya 4000
203 Gargi 2000
204 Hina 30000
205 Kalpana 4000
301 Nikhil 7777
```

awk '**BEGIN{ printf "Records in the bank are :\n"}**
{print $1, $2, $3}' bank.txt

BEGIN and END

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ awk 'BEGIN{ printf "Records in the
bank are :\n"}{print $1, $2, $3} END{print "\n we displayed all records"}' bank.
txt
Records in the bank are :
101 ADITYA 0
102 Anil 10000
103 Naman 0
104 Ram 10000
105 Jyotsna 5000
106 Mukesh 14000
107 Vishal 14500
108 Chirag 0
109 Arya 16000
110 Priya 130
201 Bina 3000
202 Diya 4000
203 Gargi 2000
204 Hina 30000
205 Kalpana 4000
301 Nikhil 7777
```

```
awk 'BEGIN{ printf "Records in the bank are :\n"}
{       print $1, $2, $3}
END{print "\n we displayed all records"}' bank.txt
```

totalbal.awk :

```
{total+=$3}
END{print"total balance is = ", total}
```

```
root@MUM084:~/Desktop# awk -f totalbal.awk bank.lst total
balance is = 120407
```

Create file countrec.awk which contains actions to count no. of records, to calculate total balance and average balance. Apply commands in this file to bank.txt

```
BEGIN{printf "Records are: \n"}
{
print $0
c++
sum+=$3
}
END{printf "\n Number of records are: %d", c ; printf "\n Total balance is %d", sum; printf "\n Average balance is  %f", sum/c }
```

awk -f countrec.awk bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ cat countrec.awk
BEGIN{printf "Records are: \n"
}
{
print $0
c++
sum+=$3
}
END{printf "\n Number of records are: %d", c printf "\n Total balance is %d", sum
printf "\n Average balance is " %f", sum/c
}
```

Practical Execution

Create a student marklist file “bank.txt” having the following details
ID, name, Balance in Acc., Date of opening, Type of account

For the above data file:
1.Display all the records
2.Display the name and balance for all records
3.Display only Saving accounts.
4.Print records having balance less than 5000 or more than 10000
5.Print records whose account type is not saving.
6.Demonstrate use of NF and NR variables.
7.Calculate and print interest for each record as 0.05 % of balance.
8.Demonstrate use of END and BEGIN with awk.
9. scripting with AWK:
Create file countrec.awk which contains actions to count no. of records, to calculate total balance and average balance.
Apply commands in this file to bank.txt

Practical Execution

Create a student marklist file "studentmarks.lst" with maximum 80 marks per subject(at least 10 students). The record structure is:
RollNo Name EM DSA DMS PC PCPF Total Percentage

For the above data file:
1.Display full list
2.Display Roll No, Name and Total
3.Display details of students having overall distinction(>=75%)
4.Display details of students who have failed in any subject(< 32)

EXPERIMENT : 9

grep

Grep

- **Grep** is an acronym that stands for Global Regular Expression Print.
- **Grep** is a Linux / Unix **command**-line tool used to search for a string of characters in a specified file.

“Search Character Pattern”

Options with grep

- -e: pattern
- -i: Ignore uppercase vs. lowercase.
- -v: Invert match.
- -c: Output count of matching lines only.
- -l: Output matching files only.
- -n: Precede each matching line with a line number.
- -b: A historical curiosity: precede each matching line with a block number.
- -h: Output matching lines without preceding them by file names.
- -s: Suppress error messages about nonexistent or unreadable files.
- -f file: Take regexes from a file.
- -o: Output the matched parts of a matching line.

Search lines containing the pattern 'Mu' in file bank.txt

```
[Himanis-MacBook-Pro:UnixLab himanideshpande$ grep mu bank.txt
[Himanis-MacBook-Pro:UnixLab himanideshpande$ grep Mu bank.txt
 106 Mukesh 14000 20/12/2009 Current
```

O R

```
[Himanis-MacBook-Pro:UnixLab himanideshpande$ grep 'Mu' bank.txt
 106 Mukesh 14000 20/12/2009 Current
```

Bank2.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ cat bank2.txt
101 aditya 0 14/11/2000 CURRENT
102 anil 10000 20/05/2011 saving
103 naman 0 20/08/2009 current
104 ram 10000 15/08/2010 saving
105 jyotsna 5000 16/06/2012 saving
106 mukesh 14000 20/12/2009 Current
107 vishal 14500 30/11/2011 saving
108 chirag 0 15/12/2012 Current
109 arya 16000 14/12/2010 Current
110 priya 130 16/11/2009 Saving
201 bina 3000 11/03/2010 saving
202 diya 4000 13/04/2018 Saving
203 gargi 2000 21/01/2015 current
204 hina  30000 14/02/2014 saving
205 kalpana 4000 8/9/2007 Current
301 nikhil 7777 8/9/1999 saving
```

Search lines containing the pattern 'Mu' in file bank.txt and bank2.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep Current bank.txt bank2.txt
bank.txt:106 Mukesh 14000 20/12/2009 Current
bank.txt:108 Chirag 0 15/12/2012 Current
bank.txt:109 Arya 16000 14/12/2010 Current
bank.txt:205 Kalpana 4000 8/9/2007 Current
bank2.txt:106 mukesh 14000 20/12/2009 Current
bank2.txt:108 chirag 0 15/12/2012 Current
bank2.txt:109 arya 16000 14/12/2010 Current
bank2.txt:205 kalpana 4000 8/9/2007 Current
```

Count the number of lines containing the pattern 'Current' in file bank.lst

grep **-c** Current bank.txt
grep **-c** Current bank.txt bank2.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -c Current bank.txt
4
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -c Current bank.txt bank2.txt
bank.txt:4
bank2.txt:4
```

Display line numbers along with lines containing the pattern 'current' in file bank.lst

grep **-n** current bank.lst

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -n current bank.txt
3:103 Naman 0 20/08/2009 current
13:203 Garai 2000 21/01/2015 current
```

Display lines **not** containing the pattern 'current' in file bank.lst

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -v current bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

grep **-v** current bank.txt

List the file names containing the pattern 'current' in files ending with .txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -l current *.txt
bank.txt
bank2.txt
```

grep -l current *.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -l bash *.sh
abc.sh
calculator.sh
count.sh
files.sh
first.sh
read.sh
unix.sh
untill.sh
var.sh
```

List the file names not containing the pattern 'current' in file bank.lst

grep -i current bank.txt

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -i current bank.txt
101 ADITYA 0 14/11/2000 CURRENT
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
203 Gargi 2000 21/01/2015 current
205 Kalpana 4000 8/9/2007 Current
```

- Here -e is used to indicate OR.

```
grep -e current -e Current bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -e current -e Current bank.txt
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
203 Gargi 2000 21/01/2015 current
205 Kalpana 4000 8/9/2007 Current
```

Prints 2 lines before and after the line containing pattern 'Jyotsna'

grep **-2** **-i** Jyotsna bank.lst

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep -2 -i
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
```

Prints lines containing letter ‘H'

```
grep H bank.txt
        OR
 grep ‘H’ bank.txt
        OR
 grep “H” bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep H bank.txt
204 Hina  30000 14/02/2014 saving
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep "H" bank.txt
204 Hina  30000 14/02/2014 saving
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep 'H' bank.txt
204 Hina  30000 14/02/2014 saving
```

Display all records that have or do not have letter 'D'

Display all records starting with 1

Append few records as shown below to bank.lst file

Display all records containing pattern current. Five dots (.....) implies that any five characters having 'nt' at the end are displayed.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep .....nt bank.txt
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
203 Gargi 2000 21/01/2015 current
205 Kalpana 4000 8/9/2007 Current
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep .....l bank2.txt
102 anil 10000 20/05/2011 saving
107 vishal 14500 30/11/2011 saving
205 kalpana 4000 8/9/2007 Current
301 nikhil 7777 8/9/1999 saving
```

Display all records containing pattern that starts with A, ends with a and inbetween contains any two letters.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep G...i bank.txt
203 Gargi 2000 21/01/2015 current
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ grep A..a bank.txt
109 Arya 16000 14/12/2010 Current
```

Question for implementation

Create a student mark list file "studentmarks.txt" with maximum 100 marks per subject(at least 10 students). The record structure is:RollNo Name EM DSA DMS PC PCPF Total Percentage
For the above data file:

- Display records of students having u in their name
- Display record of a specific student
- Display details of students having 60 in any subject
- Display line number and details of students who have 30 in any subject

EXPERIMENT : 10

sed

SED

SED command in UNIX is stands for stream editor and it can perform lot's of function on file like, searching, find and replace, insertion or deletion.

SED

- Syntax: **sed options 'address action' file(s)**

- **options:**
  - -n suppresses duplicate line printing
  - -f Reads instruction from a file
  - -e Interpretes the next string as an instruction or a set of instructions.

options

Sed command line options		
Sed syntax: *sed [options] sed-command [input-file]*		
-n	Suppress default pattern space printing	*sed -n '3 p' employee.txt*
-i	Backup and modify input file directly	*sed -ibak 's/John/Johnny/' employee.txt*
-f	Execute sed script file	*sed -f script.sed employee.txt*
-e	Execute multiple sed commands	*sed -e 'command1' -e 'command2' input-file*

Q. Print first 3 line from bank.txt

```
sed '3q' bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed '3q' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
```

Repeate line no 5

```
sed '5p' bank.txt
```

The /p print flag prints the replaced line twice on the terminal.

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed '5p' bank.tx
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed '3p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
103 Naman 0 20/08/2009 current
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Repeate two lines 3rd and 5th

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed '3,5p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Print 5th line from file

```
sed –n '5p' bank.txt
```

-n will suppresses duplicate line printing,
prints specific line

```
Himanis–MacBook–Pro:UnixLab himanideshpande$ sed –n '5p' bank.txt
105 Jyotsna 5000 16/06/2012 saving
```

```
Himanis–MacBook–Pro:UnixLab himanideshpande$ sed –n '11p' bank.txt
201 Bina 3000 11/03/2010 saving
```

Print 2nd to 5th line

```
sed –n ‘2,5p' bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed –n '2,5p' bank.txt
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
```

$ represents the last record in file.

```
root@MUM084:~/Desktop# sed -n '$p' bank.lst
110     Priya   130     16/11/2009      Saving
```

From 8th to last line

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '8,$p' bank.txt
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Lines except m to n

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '5,$!p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '7,$!p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
```

Print all lines except a few

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '2,9!p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Print all records

```
Himanis-MacBook-Pro:Unixlab himanideshpande$ sed '$q' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Print all records except last line

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '$!p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
```

```
sed -n '$!p' bank.txt
```

Exclude line 2nd to 5th

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '2,5!p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

```
sed -n '2,5!p' bank.txt
```

Combine multiple commands. (-e)

```
sed -n -e '1,2p' -e '7,9p' -e '$p' bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n -e '1,2p' -e '7,9p' -e '$p'
bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
```

Sed script

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ cat instruction.txt
1,2p
7,9p
$p
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n -f instruction.txt bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
```

Find occurrence of a word

```
sed –n '/Current/p' bank.txt
```

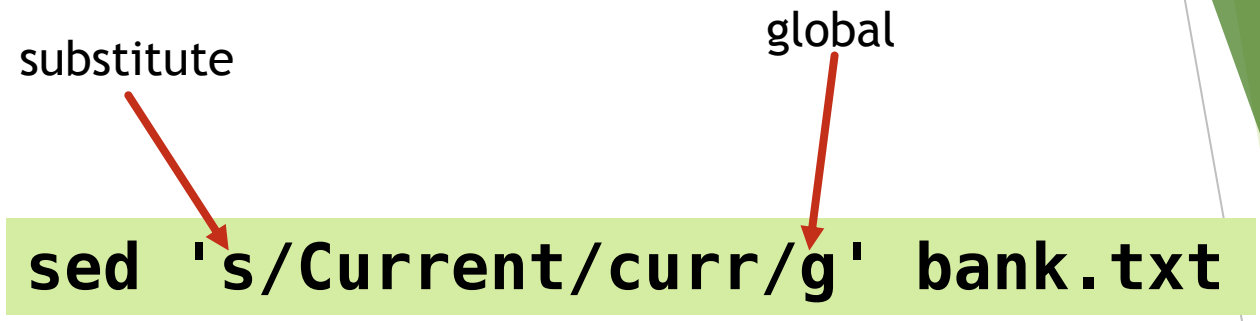
```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '/Current/p' bank.txt
106 Mukesh 14000 20/12/2009 Current
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
205 Kalpana 4000 8/9/2007 Current
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '/current/p' bank.txt
103 Naman 0 20/08/2009 current
203 Gargi 2000 21/01/2015 current
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '/CURRENT/p' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
```

Print occurrences of current or saving

sed -n '/current/,/saving/p' bank.lst

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed -n '/current/,/saving/p' bank.t
xt
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
```

replace




```
sed 's/Current/curr/g' bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed 's/Current/curr/g' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 curr
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 curr
109 Arya 16000 14/12/2010 curr
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 curr
301 Nikhil 7777 8/9/1999 saving
```

Writing to a file

```
sed –n '/[cC]urrent/w clist.txt' bank.txt
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed –n '/[cC]urrent/w clist.txt' bank.txt
Himanis-MacBook-Pro:UnixLab himanideshpande$ cat clist.txt
103 Naman 0 20/08/2009 current
106 Mukesh 14000 20/12/2009 Current
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
203 Gargi 2000 21/01/2015 current
205 Kalpana 4000 8/9/2007 Current
```

Delete second line

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed  '2d'   bank.txt
101 ADITYA 0 14/11/2000 CURRENT
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Deleting line 3 to 11

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed '3,11d' bank.txt
101 ADITYA 0 14/11/2000 CURRENT
102 Anil 10000 20/05/2011 saving
202 Diya 4000 13/04/2018 Saving
203 Gargi 2000 21/01/2015 current
204 Hina  30000 14/02/2014 saving
205 Kalpana 4000 8/9/2007 Current
301 Nikhil 7777 8/9/1999 saving
```

Delete lines other than the specified range

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed  '3,11!d'   bank.txt
103 Naman 0 20/08/2009 current
104 Ram 10000 15/08/2010 saving
105 Jyotsna 5000 16/06/2012 saving
106 Mukesh 14000 20/12/2009 Current
107 Vishal 14500 30/11/2011 saving
108 Chirag 0 15/12/2012 Current
109 Arya 16000 14/12/2010 Current
110 Priya 130 16/11/2009 Saving
201 Bina 3000 11/03/2010 saving
```

Printing with skipping

‘1~2 p’
start with 1st line
print every second line

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ sed  ‘1~2 p'   bank.txt
```

Practical for execution

Create a student marklist file "studentmarks.lst" with maximum 100 marks per subject(at least 10 students). The record structure is:
RollNo Name EM DSA DMS PC PCPF Total Percentage

For the above data file:
1.Display full list
2.Display last record
3.Display details of students with odd numbered record
4.Display details of students except record Nos 2,3,6,7and 8

EXPERIMENT : 11

PERL SCRIPTING

PERL

- PERL is Practical Extraction Reporting Language
- Used for text processing and automation.

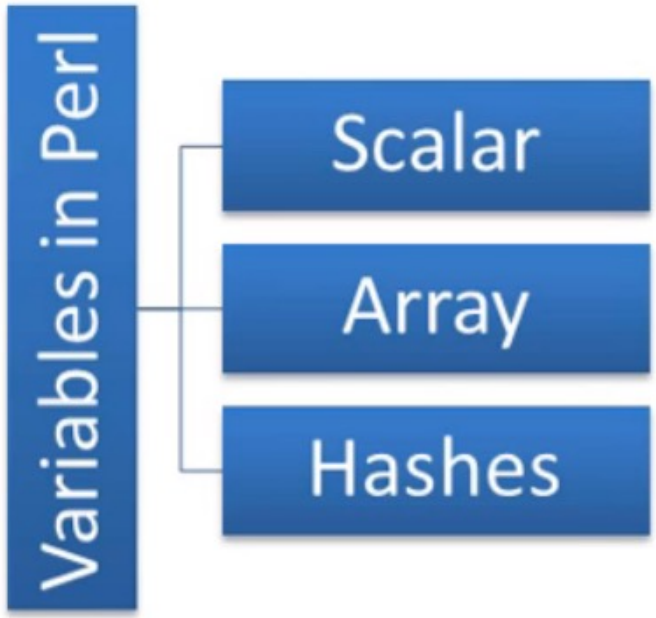
- Developed by Larry Wall in late 1980s as a way to make report processing easier.

Features of Perl

- Take features from C, awk, sed, sh.
- Supports OOP concept
- One of the most popular web programming language
- Case sensitive.

- PERL is interpreted language

Variables in Perl



perl

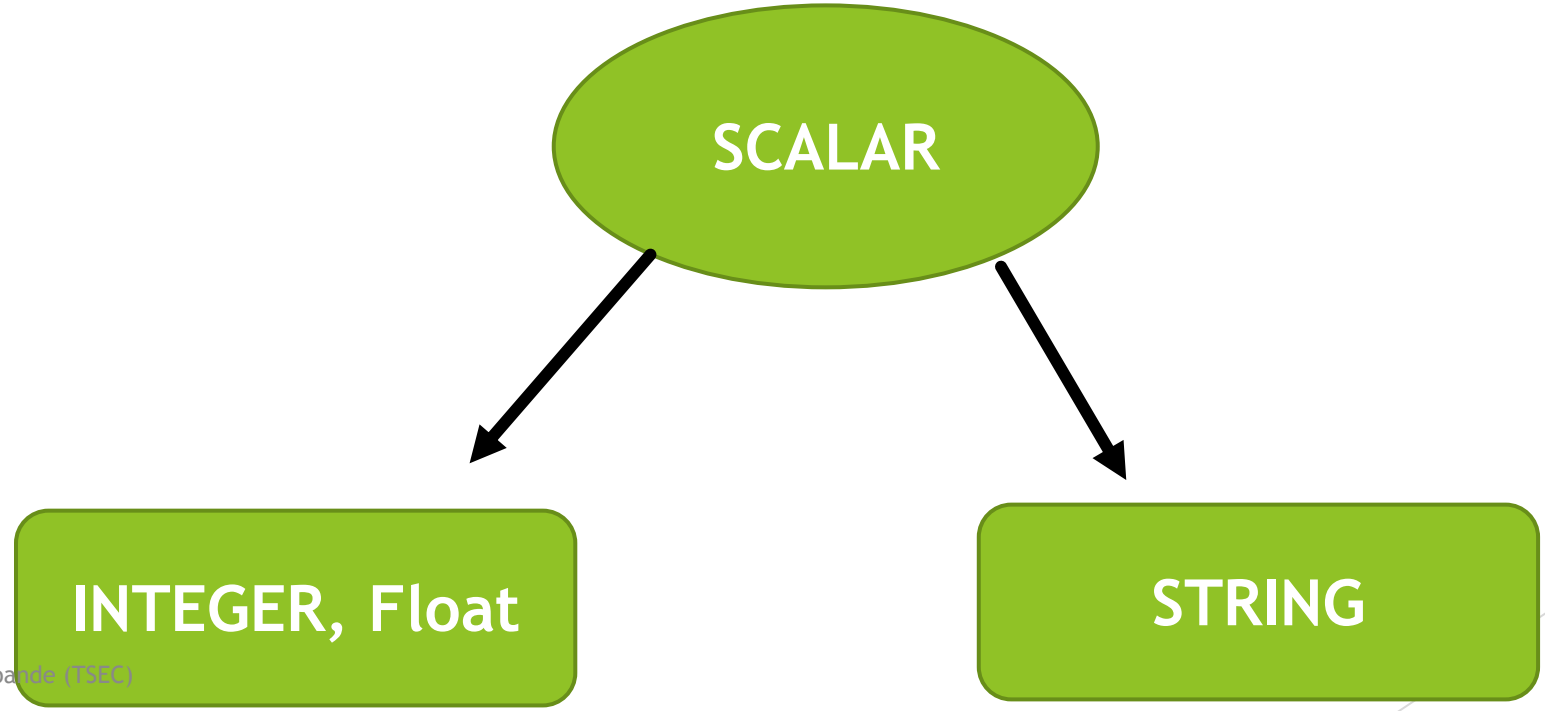
- There is no requirement of specifying the data type in Perl
- It only has three types – scalar, array and hashes

- Scalar must start with $
- Array must start with @
- Hashes must start with %

Scalar data type

Scalar is automatically taken as int or string (For Perl both are scalar)

All scalar values are initialized to 0



Scalar Variables

- $employee_name = “Sagar”; # String assignment
- $employee_age = 23; # An integer assignment
- $employee_salary = 440.5 # Floating point number

```
print "Age = $employee_age\n";
print "Name = $employee_name\n";
print "Salary = $employee_salary\n";
```

Age = 23
Name = Sagar
Salary = 440.5

Arrays in PERL

- Array variables are preceded by “@”
- @names = (“Shobhit”, ”Ekansh", ”Henna");
- @ages = (25, 28, 31);

```
print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

$ages[0] = 25
$ages[1] = 28
$ages[2] = 31
$names[0] = Shobhit
$names[1] = Ekansh
$names[2] = Henna

Hash Data

Set of **key/value** pair is called a Hash.
Each key in a hash structure are unique and of type strings.
The values associated with these keys are **scalar**.
Each key is associated with a single value.

- **Hash** variables are preceded by "%" sign
- %data = ('Danish', 28, 'Raju', 40, 'Ritesh', 25);


```
print "\$data{'Danish'} = $data{Danish'}\n";
print "\$data{'Raju'} = $data{'Raju'}\n";
print "\$data{'Ritesh'} = $data{'Ritesh'}\n";
```



```
$data{'Danish'} = 28
$data{'Raju'} = 40
$data{'Ritesh'} = 25
```

PERL

- Perl file extension is .pl or .pm
- Shebang at the beginning of script.
  - #!/usr/bin/prl (use which perl command to find location)
- Execute
  - Perl scriptname.pl

    Or
  - ./scriptname.pl

* A **shebang** is the character sequence consisting of the characters number sign and exclamation mark (#!) at the beginning of a script. It is also called sha-bang, **hashbang**, pound-bang, or hash-pling.

Print " Hello World !! "

```
#!/bin/perl
# This will print "Hello, World"
print "Hello, world\n";
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ perl hello.pl
Hello, world
```

Perl script for an array

```
#!/usr/bin/perl
@names = ('Kevin','Arpita','Yash');
@copy=@names;
$size=@names;
print "Given names are  @copy \n";
print "Number of names are :$size\n";
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ perl first.pl
Given names are  Kevin Arpita Yash
Number of names are :3
```

Program to

Perl treats same variable differently based on Context, i.e., situation where a variable is being used.

```
#!/bin/perl
@names = ('John Paul', 'Lisa', 'Kumar');
@copy = @names;
$size = @names;
print "Given names are : @copy\n";
print "Number of names are : $size\n";
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ perl variableuse.pl
Given names are : John Paul Lisa Kumar
Number of names are : 3
```

Array

```
@ages = (25, 30, 40);
```

```
print "\$ages[0] = $ages[0]\n";          25
print "\$ages[1] = $ages[1]\n";          30
print "\$ages[2] = $ages[2]\n";          40

print "\$ages[-1] = $ages[-1]\n";        40
print "\$ages[-2] = $ages[-2]\n";        30
print "\$ages[-3] = $ages[-3]\n";        25
```

Assign values to variables

```
#!/bin/perl
$age = 25;              # An integer assignment
$name = "John Paul";    # A string
$salary = 1445.50;      # A floating point
@ages = (25, 30, 40);
@names = ("John Day", "Lisa", "Kumar");
print "Age = $age\n";
print "Name = @names\n";
print "Salary = $salary\n";
print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$ages[-1] = $ages[-1]\n";
print "\$ages[-2] = $ages[-2]\n";
print "\$ages[-3] = $ages[-3]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```



```
Himanis-MacBook-Pro:UnixLab himanideshpa
Age = 25
Name = John Day Lisa Kumar
Salary = 1445.5
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$ages[-1] = 40
$ages[-2] = 30
$ages[-3] = 25
$names[0] = John Day
$names[1] = Lisa
$names[2] = Kumar
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

Arithmetic Operations

```
print 10 + 20, "\n"; # 20
print 20 - 10, "\n"; # 10
```

```
print 10 * 20, "\n"; # 200

print 20 / 10, "\n"; # 2
```

```
print 2**3, "\n"; # = 2 * 2 * 2 = 8.

print 3**4, "\n"; # = 3 * 3 * 3 * 3 = 81.
```

Power

Comparison operators for numbers

Equality	Operators
Equal	==
Not Equal	!=
Comparison	<=>
Less than	<
Greater than	>
Less than or equal	<=
Greater than or equal	>=

```
#!/bin/perl
$roll = 6;
@names = ('Ameya', 'Vivek', 'Shirsat');
$age = 20;
%gpa = ('Sem1',8.83,'Sem2',9.08,'Sem3',8.65);
print "Student Details \n";
print "Name: @names \n";
print "Roll No.: $roll \n";
print "Age: $age\n";
print "Sem 1: $gpa{'Sem1'}\n";
print "Sem 2: $gpa{'Sem2'}\n";
print "Sem 3: $gpa{'Sem3'}\n";
```

simple operations on variables:

```
#!/bin/perl
$str = "hello" . " world";  # Concatenates strings.

$num = 5+10;                # adds two numbers.
$mul = 4*5;                 # multiplies two numbers.
$mix = $str . $num;         # concatenates string and number.

print "str = $str\n";
print "num = $num\n";
print "mul = $mul\n";
print "mix = $mix\n";
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ perl arithmetic.pl
str = hello world
num = 15
mul = 20
mix = hello world15
```

Power of a number

<> operator is used to take input

```
#!/bin/perl
print " Enter base number: ";
$base=<>;

print " Enter power to be calculated: ";
$power=<>;

$result = $base ** $power;
print " Result is : " . $result;
```

```
Himanis-MacBook-Pro:UnixLab himanideshpande$ perl power.pl
 Enter base number: 4
 Enter power to be calculated: 2
 Result is : 16
```

Check whether a number is prime

```
#!/bin/perl
print " Enter number to check: ";
$num=<STDIN>;
$flag=0;
chomp($num);
for($i=2; $i<$num; $i++)
{
    if ($num%$i ==0)
    {   $flag=1;
        break; }
}

if ($flag==1)
{
   print " $num is not prime\n ";
}
else
{ print "$num is prime\n"; }
```

STDIN in Perl is used to take input from the keyboard

Standard input

```
Himanis-MacBook-Pro:UnixLab
himanideshpande$ perl prime.pl
 Enter number to check: 17
17 is prime
```

The chomp() function in Perl is used to remove the last trailing newline from the input string.

Sorting an array

```
#!/bin/perl
print "Enter 10 numbers: \n";
@num;
for ($i=0; $i<10; $i++)
{
   $num[$i] = <>;
}
for ($i=0; $i<10; $i++)
{
   for ($j=$i; $j<10; $j++)
   {
      if($num[$i] > $num[$j])
      {
         $temp = $num[$i];
         $num[$i] = $num[$j];
         $num[$j] = $temp;
      }
   }
}
print "After Sorting : \n";
for ($i=0; $i<10; $i++)
{
   print $num[$i];
}
```

Declare Football Tournament Winner using Hash

```
%score = ('IT',9,'CS',6,'EXTC',5,'CHEM',3,'MECH',8);
@value = values %score;
@key = keys %score;
$big = $value[0];
$winner;
for($i=0;$i<5;$i++)
{
    if($value[$i]>$big){
        $big = $value[$i];
      $winner = $key[$i];
    }
}
print "##############################################\n\n";
print "Winner : $winner\n\n";
print "##############################################\n";
```

Assignment

Write a perl script to:

A. Assign values for datatypes as indicated in brackets and display
   Student details:
   Roll No (Scalar), Name(Array of 3), Age(scalar),
   GPA for 3 semesters(Hashes(Sem,GPA)).
B. Check if a number is even or odd
C. Display power table of a number taken as input from user.
D. Calculate SI with value of Rate as 8% and P , T given as input.

Assignment

Write a perl script to:

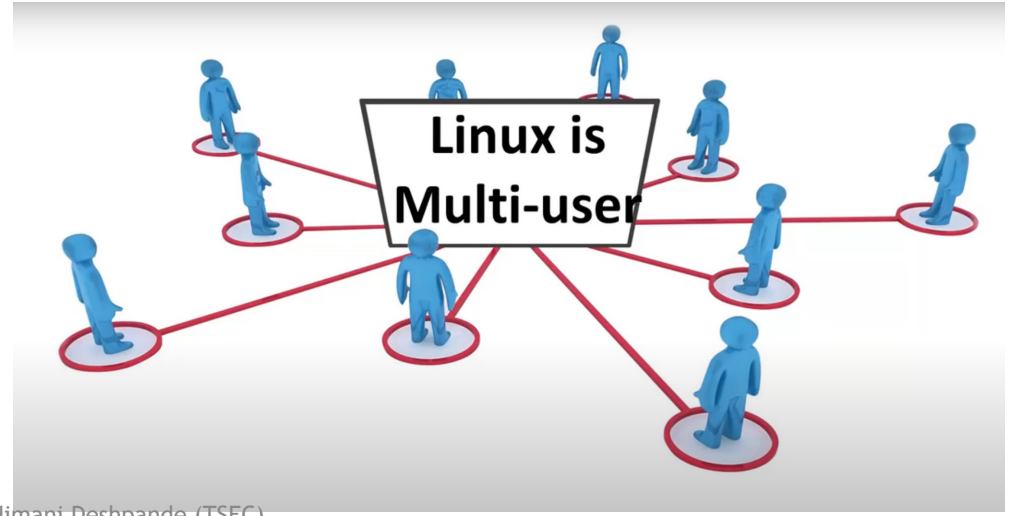
A. Sorting of an array.
B. Check if a number is prime or not.

EXPERIMENT : 12

UNIX FILE SYSTEM

Security

Security is a big concern on Unix OS being a multi user OS.



Unix is used on servers , its vital to protect files

Authorization Levels

- Ownership

- Permissions

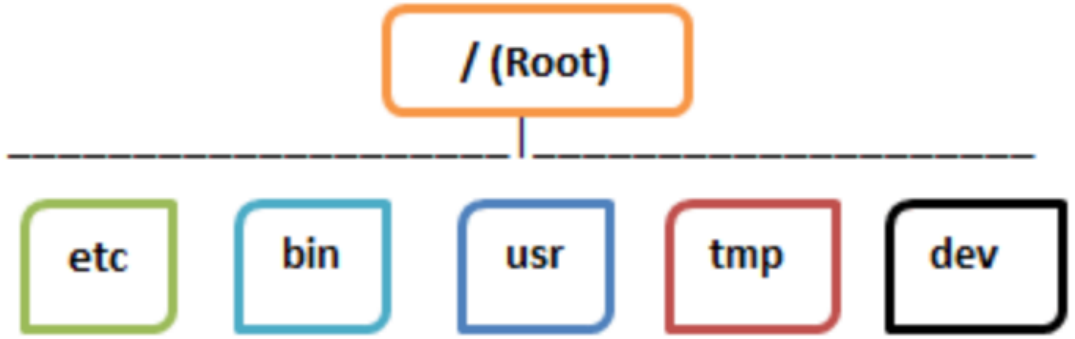
Introduction

- What is File System?
  - The abstraction used by kernel to represent and organize the storage resources.
- UNIX File System in general
  - File system is organized in tree structure.
  - File tree can be arbitrarily deep.
  - File name must NOT LONGER than 256 chars.
  - Single path name must NOT LONGER than 1023 chars.

Creating File System

- Mounting File System
  - File tree is composed of File System
  - Use *mount* command to map a directory within the existing file tree (mount point) to the root of the new file system.
    - *mount /dev/hda2 /usr*
  - Use *umount* command to detach the file system.
    - Detaching will fail if the file system is busy.

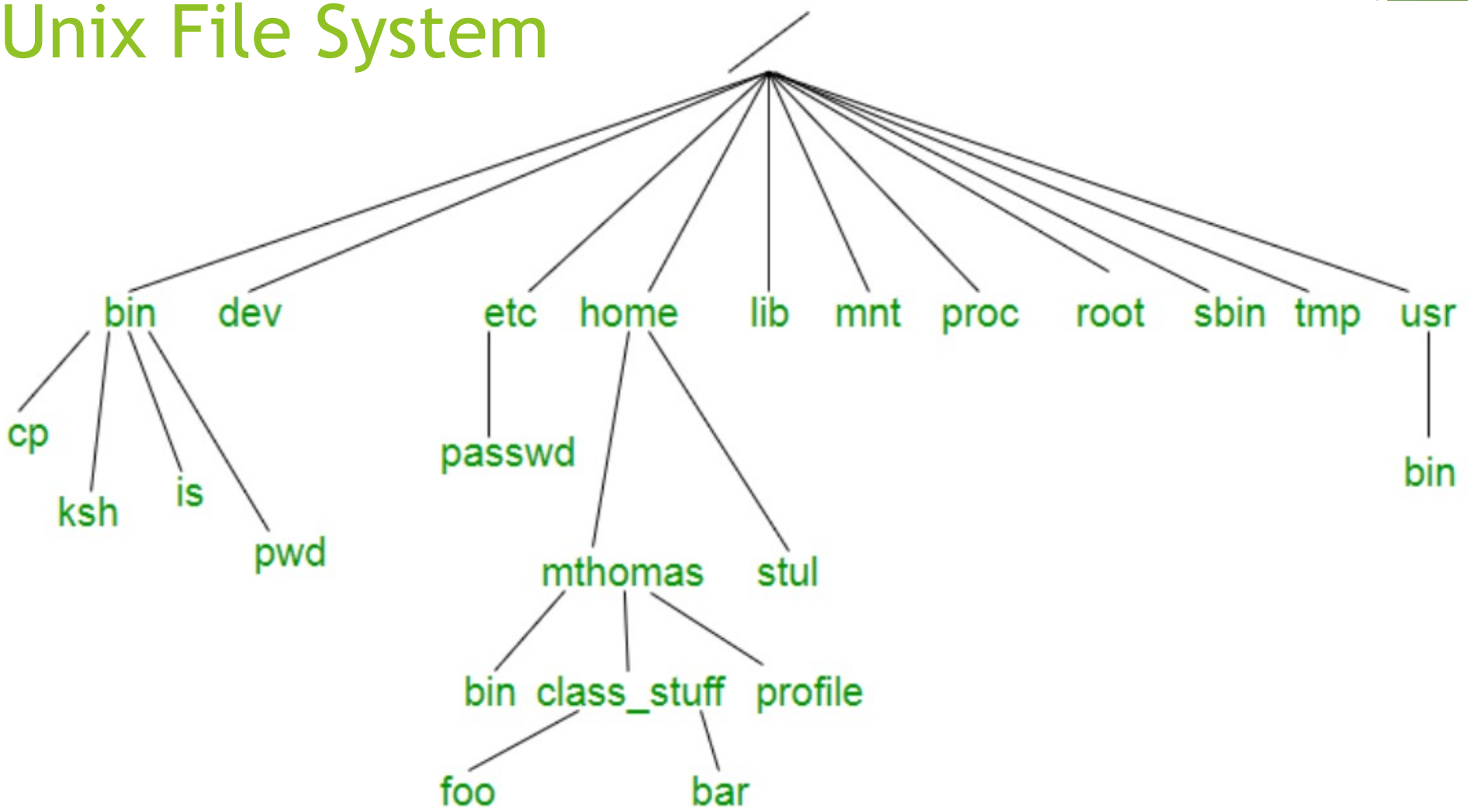
File System



Organizing of The File System (cont.)

/	The root directory
/bin or /sbin	Commands for basic system operation
/dev	Device entries
/etc	Critical startup and configuration files.
/lib	Library for the C compiler
/tmp	Temporary files
/var/adm or /var/log	Accounting file, log files

Unix File System

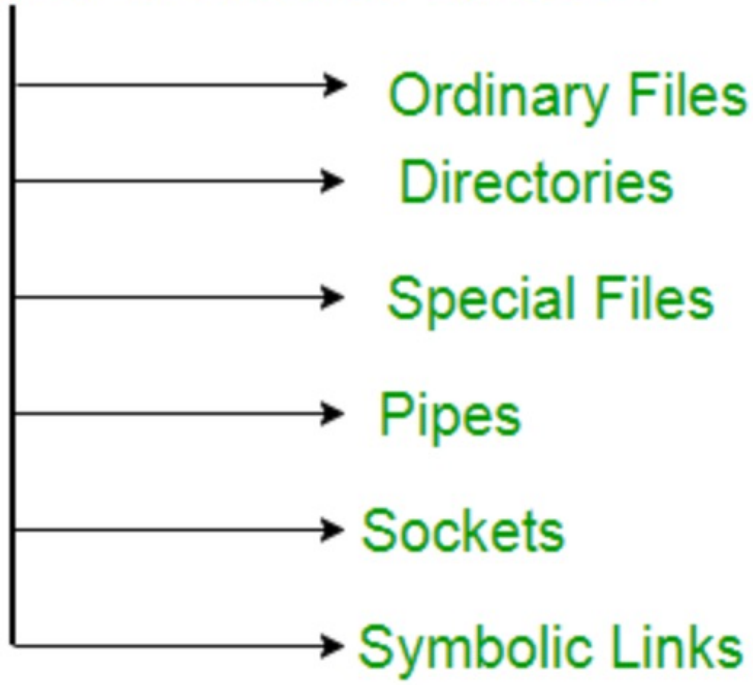


- **/ :** The slash / character alone denotes the root of the filesystem tree.
- **/bin :** Stands for “binaries” and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
- **/boot :** Contains all the files that are required for successful booting process.
- **/dev :** Stands for “devices”. Contains file representations of peripheral devices and pseudo-devices.
- **/etc :** Contains system-wide configuration files and system databases. Originally also contained “dangerous maintenance utilities” such as init,but these have typically been moved to /sbin or elsewhere.
- **/home :** Contains the home directories for the users.
- **/lib :** Contains system libraries, and some critical files such as kernel modules or device drivers.
- **/media :** Default mount point for removable devices, such as USB sticks, media players, etc.
- **/mnt :** Stands for “mount”. Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.
- **/proc :** procfs virtual filesystem showing information about processes as files.

- **/root :** The home directory for the superuser “root” – that is, the system administrator. This account’s home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- **/tmp :** A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- **/usr :** Originally the directory holding user home directories,its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).
- **/usr/bin :** This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
- **/usr/include :** Stores the development headers used throughout the system. Header files are mostly used by the **#include** directive in C/C++ programming language.
- **/usr/lib :** Stores the required libraries and data files for programs stored within /usr or elsewhere.
- **/var :** A short for “variable.” A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files.
- **/var/log :** Contains system log files.

- **/var/mail :** The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
- **/var/spool :** Spool directory. Contains print jobs, mail spools and other queued tasks.
- **/var/tmp :** A place for temporary files which should be preserved between system reboots.

Classification of Unix File System :



ALL THE BEST