UNIX Operating System

Module 1

Introduction to Unix

Objectives

Upon completing this module you should be able to understand the following:

- What is an Operating System
- History of unix Operating system
- Unix Architecture
- More features of unix
- Unix Flavors
- Linux Flavors

What Is an Operating System?

- Interface between Users and the Hardware
- Take care of Storage Management
- Take care of I/O device management

UNIX Variants

- UNIX Variants

Year	UNIX Variant	Features
1957	**BESYS**	• At Bell Labs • To run batch jobs
1965	**MULTICS** (Multiplexed Information and Computing Service)	• Adopted third generation computer equipments
1969	**UNICS** (UNiplexed Information and Computing Service)	• To play space travel on another smaller machine (DEC PDP-7)
1971	**UNIX**	• The first edition of the "UNIX PROGRAMMER'S MANUAL" – By K. Thompson [and] D. M. Ritchie; included over 60 commands

UNIX Variants

- UNIX Popular Variants

UNIX Variant	Features
AIX (Advanced Interactive eXecutive)	• Developed By IBM in 1990 • Shells available (Korn, Bourne, C) • Default Korn shell
BSD (Berkeley Software Distribution)	• Developed at the Computer System Research Group (CSRG)
Sun Solaris	• Sun company's UNIX variant operating system
MINIX	• A free UNIX clone written from scratch • Small size • Micro kernel-based design and simple documentation • Suited for personal computer

LINUX

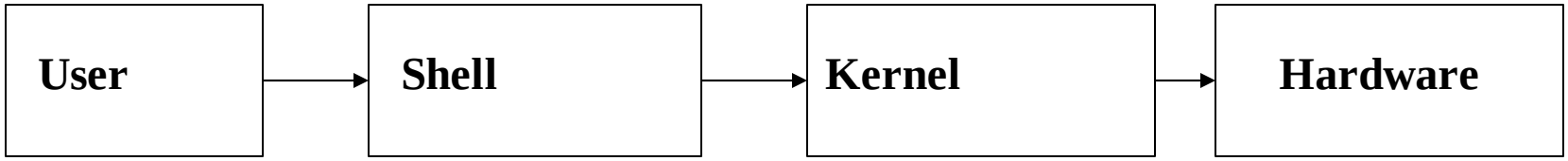
- Developed by Linus Torvalds
- Freely available multitasking and multi-user operating system
- Placed under General Public License
- Variants
  - Caldera Linux
  - Debian Linux
  - Kondara Linux
  - Red Hat Linux
  - Ubuntu Linux

UNIX Flavors

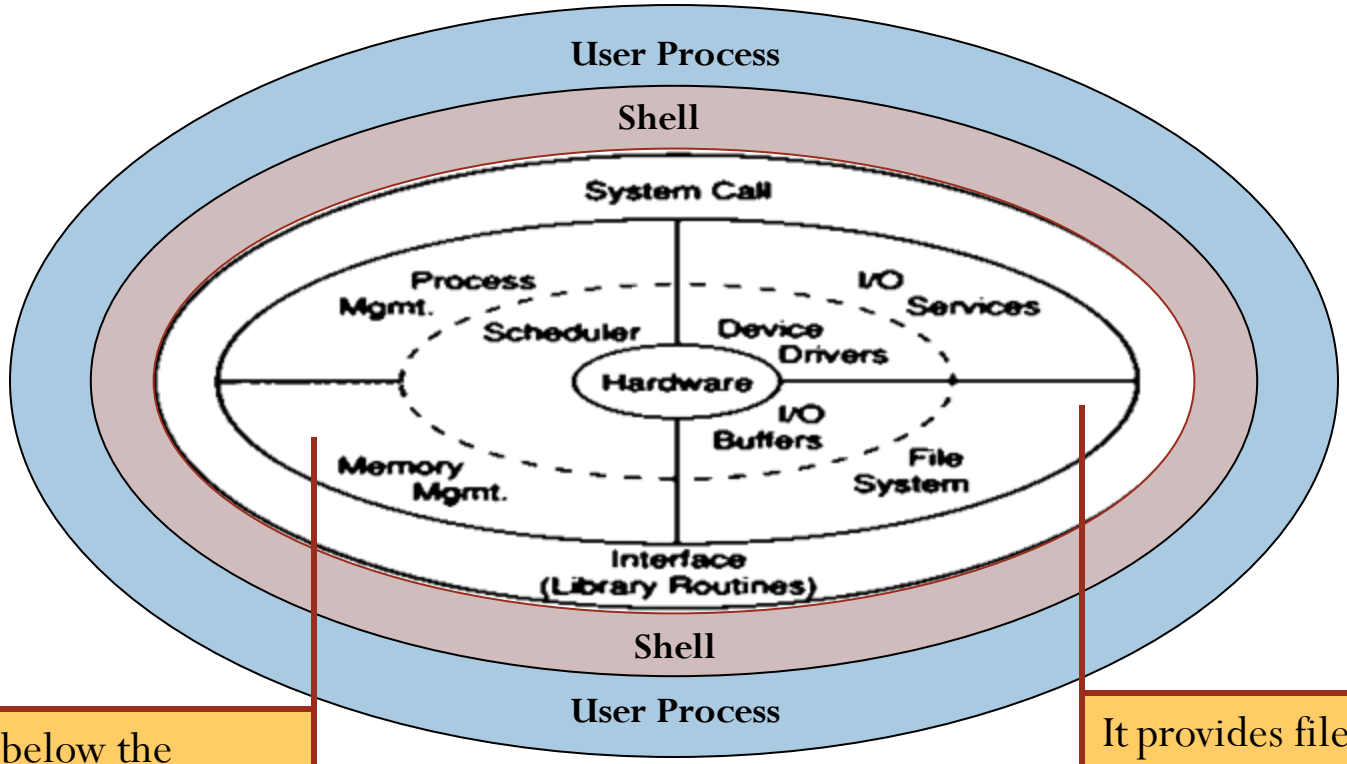
- All proprietary UNIX flavors and their names are respective trademarks of the originating entity/vendor.

Originator	Proprietary Name
U. Cal. Berkley	BSD
Sun	Solaris
IBM	AIX
HP	HP - UX
SGI	IRIX
Digital Equipment[2]	DEC UNIX
Compaq	Tru64 UNIX[3]
Apple	MacOS X[4]
L. Torvalds/GNU[5]	Linux[6]

UNIX Architecture – Kernel & Shell



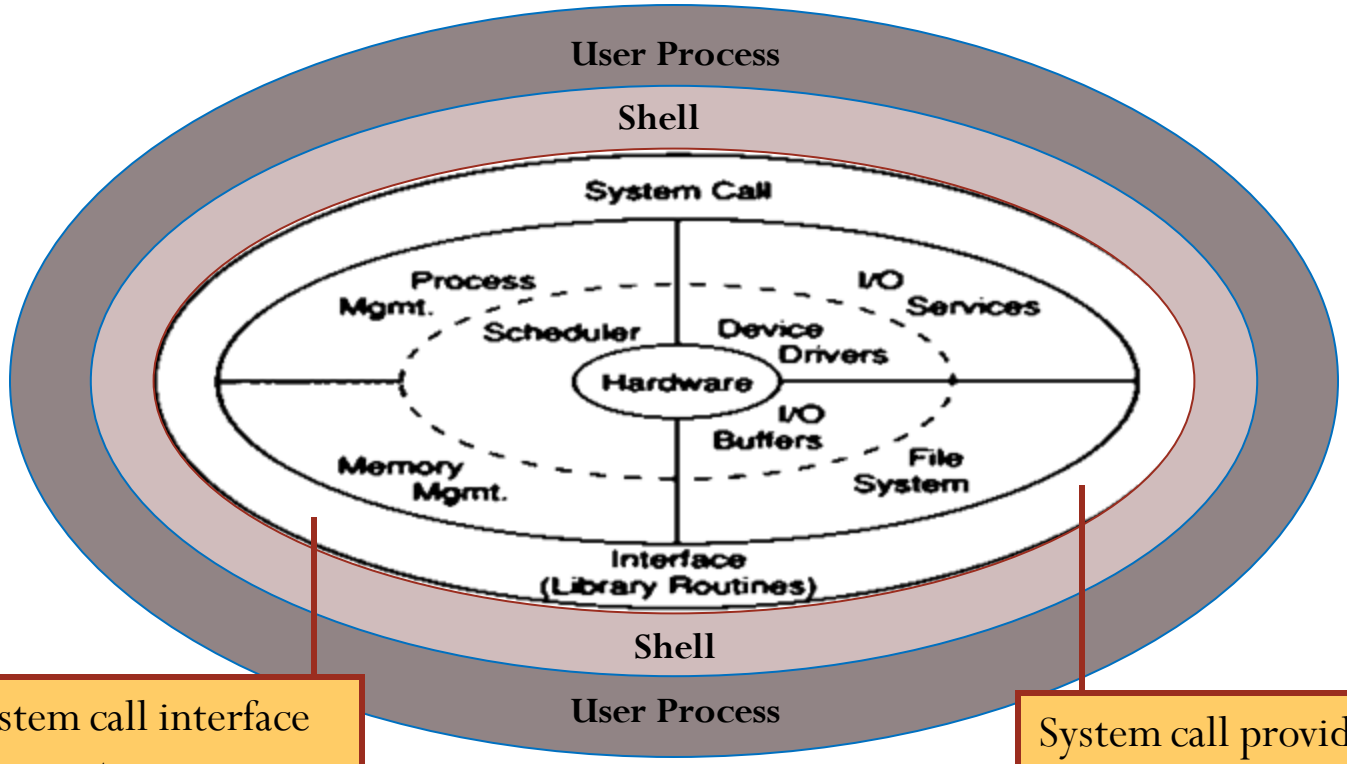
UNIX Onion Architecture (1 of 2)



Everything below the system call interface and above the physical hardware is the **KERNEL**.

It provides file system, CPU scheduling, memory management, and other OS functions using system calls.

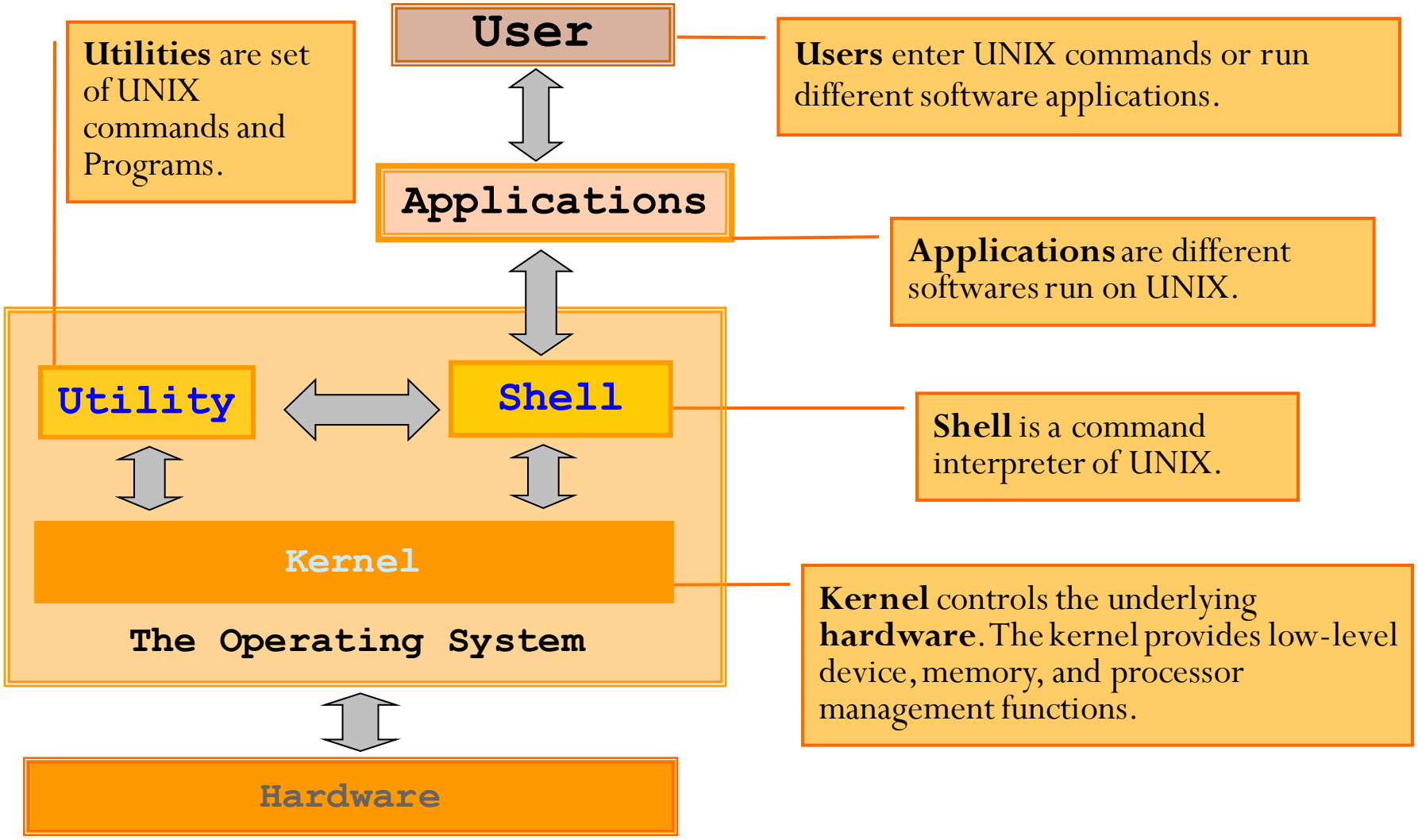
UNIX Onion Architecture (2 of 2)



The system call interface layer converts a process running in user mode to a protected kernel mode process.

System call provides a programming interface that allows user programs to access kernel functions.

User Application and Kernel Interaction



Utilities are set of UNIX commands and Programs.

Users enter UNIX commands or run different software applications.

Applications are different softwares run on UNIX.

Shell is a command interpreter of UNIX.

Kernel controls the underlying **hardware**. The kernel provides low-level device, memory, and processor management functions.

UNIX Architecture – System Calls

- Though there are over a thousand commands on the system, they all use a handful of functions, called system calls, to communicate with the kernel.

- All unix flavors use the same system calls and are described in the POSIX specification.

Kernel

- Kernel is the heart of OS and performs various operations on behalf of the user processes, like:
  - Permitting process controlled access
    - Devices like terminals
    - Disk and tape drives
    - Network drives
  - Controlling the process execution
    - Creation
    - Termination
    - Suspension
    - Communication

Kernel

- Efficient retrieval and storage of user data
  - By allocating storage
  - By reclaiming unused storage
  - Protecting
- Time sharing
  - Suspending the process
  - Rescheduling the processes
- Sharing of main memory
  - Swapping system

More Features of UNIX

- Hierarchical file system
- Multi tasking
- Multi user
- The building block approach
- Pattern matching (wildcard characters)
- Toolkit(Applications, RDBMSs, Languages etc..)
- Programming facility
- Documentation

Module 2

Logging In and Basic Commands

Objectives

Upon completing this module you should be able to understand the following:

- Logging In and Out
- Command Line Format
- The Secondary Prompt
- Online Manual Pages
- id Command
- who Command
- date Command
- cal, clear Commands
- passwd Command
- finger command

Processing Environment

- A program is a set of instructions written to perform a specific task.

- A process is:
  - An operation which takes the instructions given and does the manipulations or anything that is instructed in the code itself
  - The single executable module that runs concurrently with other executable modules
  - A program in execution

Shell as Command Interpreter

- Each line that shell reads from standard input is called a pipeline.
- Shell as a command interpreter:
  - Reads lines from standard input (each of these line is called pipeline)
  - Splits the command into tokens
  - Checks the token to see if it's a keyword
  - Checks the first word for alias
  - Performs command substitution for any $(string)
  - Performs wildcard expansion
  - Looks at first word as build-in command
  - Runs the command

A Typical Terminal Session

- Log in to identify yourself and gain access.
- Execute commands to do work.
- Log off to terminate your connection.

Command Line Format

Syntax:

$ command [-*options*] [*arguments*]

Separation : mail –f newmail not mail – f newmail

Order: mail –f newmail not mal newmail –f

Multiple options: who –m –u or who –mu not who –m u

Multiple arguments: mail team1 team2 not mail team1team2

The Secondary Prompt

$ echo 'hi

>*Good Morning‘*

> is the default secondary prompt

The Online Manual

man : With the man command, you can retrieve the information in the manual and display it as text output on your screen

$ man ls *Display the “ls" man page.*
$ man -k cat *Display entries with keyword "cat".*
$ man passwd *Display the "passwd" man page-Section 1.*
$ man 4 passwd *Display the "passwd" man page-Section 4.*

The id Command

Syntax:

id Displays effective user and group identification for session

Example:

$id

uid =303 (user3) gid=300 (class)

The who Command

Syntax:

who Reports information about users who are currently logged on to a system

Examples:

```
$ who
  root      tty1p5  Jul 01 08:01
  user11    tty1p4  Jul 01 09:59
  user12    tty0p3  Jul 01 10:01


$ whoami
  user12
```

The date Command

Syntax:

date Reports the date and time

Example:

```
$ date
Fri Jul  1 11:15:55 EDT 2005
```

Display Date from a String Value using –date Option

- date --date="12/2/2014“
- date --date="Feb 2 2014“
- date --date="next mon“
- date --date='3 seconds ago‘
- date --date="1 year ago“

Various Date Command Formats

- date +%<format-option>
- date +%a : Displays Weekday name in short (like Mon, Tue)
- date +%A :Displays Weekday name in full short (like Monday)
- date +%b : Displays Month name in short (like Jan)
- date +%B : Displays Month name in full short (like January)
- date +%d :Displays Day of month (e.g., 01)
- date +%D: Displays Current Date; shown in MM/DD/YY
- date +%F : Displays Date; shown in YYYY-MM-DD

The cal Command

Syntax:

cal Reports the calendar of 2005 September(Current month)

Example:

$ cal 8 2005 for Aug 2005

$ cal 2005 for the full calendar of year 2005

cal command with options

- cal: display the date in the current calendar format.
- ncal : display the date in the day horizontally.
- cal -h: Don't highlight today date.
- cal -3 :disply previous current and next month.
- cal -y : display all months.
- cal -y 2017 :Display all month of the 2017 year
  - etc

The passwd Command

Syntax:

passwd Assigns a login password

Example:

```
$ passwd
    Changing password for user1
    Old password:
    New password:
    Re-enter new password:
```

The clear Command

Syntax:

clear Clears terminal screen

finger

Finger: Displays information about the users currently logged on

Eg:

$ finger user1

Login name: user1
Directory: /export/home/user1 shell:/usr/bin/sh
On since Sep 05 09:10:12 on tty1
No plan

Module 3

File System

Objectives

Upon completing this module you should be able to understand the following:

- What Is a File System?
- File System structure
- Hard Link
- File Types
- Ordinary Files
- Directory Files
- Symbolic Links
- Device files
- The File System Hierarchy
- Basic Commands pwd, cd, ls, mkdir, rmdir, touch

What Is a File System?

collection of control structures and Data blocks that occupy the space defined by a partition and allow for the storage and management of data.

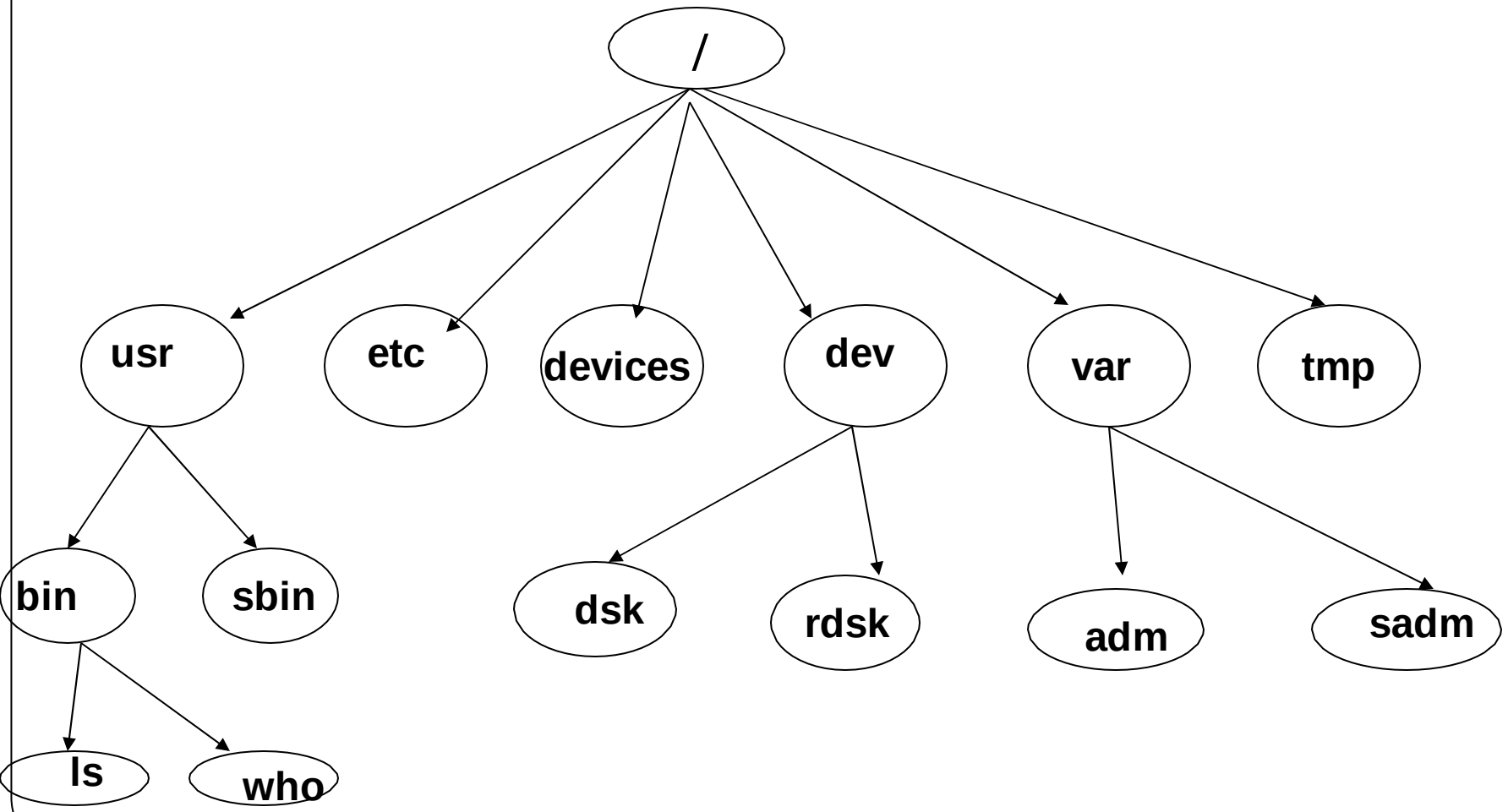
Data + Meta data

UNIX File System

- What is a file?
  - A collection of data items stored on a disk

- What is a file system?
  - A group of files and relevant information regarding them
  - Each file system is stored in a separate whole disk partition
  - UNIX supports variety of files
    - Ordinary file
    - Directory file
    - Special files
    - Character special files
    - Block special files

The File System Hierarchy



system directories

/ :root directory, top level directory

/bin : user commands like cp, cron, cmp etc..

/usr/sbin: admin commands

/dev : logical device files of all hardware devices

/devices: physical device files

/etc : System configuration files and user database

/tmp : to store temporary files

/usr : the binaries, shared libraries, shared documentation etc.

/var: stores the log files and dynamic files

File Path name

A sequence of file names, separated by slashes (/), that describes the path, the system must follow to locate a file in the file system

Absolute pathname (start from the /-directory):

Eg: /export/home/user1/file1

Relative pathname (start from the current directory)

./test1 (. = current directory)

../team03/.profile (.. = parent directory)

pwd - Present Working Directory

pwd prints the Current Directory

Displaying the OS Information

- uname : This command is use to display the operating name
- uname –r : This command is use to display the kernel version
- hostname: This command is use to display the host name
- hostname –i : This command is use to display the ip address of your machine.

Continue...

- who : This command is used to display the list of user connected to server.
- finger : It display the given current user information details.
- finger username: It display the given user information details.
- whoami : this command display the current user information.
- uptime : It display how long server is up and running, no of user connected, average load on the server.
- exit : This command is use to exit the terminal.

Cd Change Directory

syntax:

cd [dir_name]

Example:

$ pwd
/home/user3
$ cd memo; pwd
/home/user3/memo
$ cd ../..; pwd
/home
$ cd /tmp; pwd
/tmp

Directory related commands

- cd ~ : This command is use to move to the home directory.
- cd ~ username : This command is use to go the home directory for the user.
- cd - : This command is use to move to the last directory.
- cd .. : This command is use to come out from the current working directory.
- cd / : it changes to root directory
- ls : This command is use to display the all directory present in the current folder.

Continue...

- ls directoryName : This command is use to display the list of file and directory present in the directoryName folder.
- mkdir A: This command is used to make the directory
- mkdir A B C D : This command is used to create the more than one directory.
- rmdir FolderName : This command is use to remove or delete the folder.
- rmdir -r A : This command is use to remove or delete the folder which contains the files and folder.
- mv OldDirectory NewDirectory : This command is use to change the folder name.

Continue...

- mkdir .directoryName : This command is use to create the hidden folder or directory.
- mv abc .abc : Hide the directory
- mv .abc abc : Unhide the directory

Module 4

Managing Files

Objectives

Upon completing this module you should be able to understand the following:

- File Characteristics
- cat
- more
- tail
- wc
- cp
- mv
- rm

Creating the files in Unix

- We can create the file in Unix using different ways
  - Using Cat command
  - Using touch command
  - Using echo and print command
  - Using different text editor vi

cat commands

- cat >abc.txt
  - Welcome to Unix file
  - Cntr+D
    - This command is use to create the file and add the content.
- cat < abc.txt
  - or
- cat abc.txt
  - This command is use to read the content from a file.
- c. cat >> abc.txt
  - This command is use to append the content to the existing file.

touch command

- a. touch a.txt
  - This command is use to create the empty file
- b. touch b.txt c.txt d.txt
  - This command is use to create the multiple files.

Cat vs Touch

- This command is used to create the file with zero byte data. Cat command one can create or update one file at a time but using touch command user can only create multiple zero types files but can not update multiple files at time.

Creating file using echo

- echo “Welcome to Unix OS”
  - It display the output on the console.
- echo "Hi, How r you " > a1.txt
  - It create the file and copy the contents
- printf "Hi, How r you \n">a2.txt
  - This command create the file and copy the content to the a2.txt file.

more - Display the Contents of a File

Syntax:
more [*filename]...* Display files one screen at a time

Example:

```
  $ more funfile
          .
          .
          .
--funfile (20%)--
```

Q *or q* — *Quit more*
Return — *One more line*
Space bar — *One more page*

tail - Display the End of a File

Syntax:

tail [-*n]* *[filename]*... Display the end of file(s)

Example:

```
$ tail -1 test
soon as it is available.
```

wc commands

- wc -l a3.txt
  - It is use to display the number of lines in the file.
- wc -w a3.txt
  - It is use to display the number of words in the file.
- wc -lwc a3.txt
  - It is use to display the number of line, world and character in the file.

cp - Copy Files

Syntax:

cp [-i] ***file1 new_file*** Copy a file

cp [-i] ***file [file...] dest_dir*** Copy files to a directory

cp -r [-i] ***dir [dir...] dest_dir*** Copy directories

Example:

cp file1 d1 copies file1 to d1 directory

cp file2 file3 create a copy of file2 as file3 in the same directory

mv - Move or Rename Files

Syntax:

mv [-i] ***file new_file*** Rename a file
mv [-i] ***file [file...] dest_dir*** Move files to a directory
mv [-i] ***dir [dir...] dest_dir*** Rename or move directories

Example:

```
$ ls -F
f1  f2* memo/ note remind
$ mv f1 file1
$ ls -F
file1  f2* memo/ note remind
$ mv f2 memo/file2
$ ls -F
file1 memo/ note remind
$ ls -F memo
file2*
```

```
$ mv note remind memo
$ ls -F
      file1    memo/
$ls -F memo
file2* note remind
      $ mv memo letters
$ ls -F
      file1    letters/
```

Deleting the file

- rm a1.txt
  - This command is use to delete the a1.txt file.
- rm -i a2.txt
  - This command is use to delete the file with permission /confirmation
- rm -f a3.txt
  - This command is use to delete the file forcefully
- rm –r d1 remove the directory.
  - This command is use to delete the directory

Module 5

Filter commands

Flat files

- In a file data entered by using the delimiter is known as flat file.
- The default delimiter is tab.
- Delimiter means fields separated by , pipe or space.
- Enter key means records separated

Sample file

- cat > Employee

  1 Raj 12000

  2 Seeta 14000

  3 Reeta 16000

  4 Veeta 18000

- cat > Manager

  100 Raju 42000

  200 Ramesh 64000

  300 Ajay 66000

  4000 Mahesh 68000

Cut command

- cut -c 1 Employee
  - This command help to display the first column values
- cut -c 3-6 Employee
  - This command help to display the contents from 3 to 6 columns values.
- cut -c 3, 6 Employee
  - This command is use to display the 3 and 6 position values.

Paste commands

- Paste : Paste command is one of the useful and most used unix command which is used to merge the character or lines of 2 different files.
- Paste command sequentially writes the corresponding lines from each specified file.

Continue...

- paste Employee
  - It display all the contents from the file
- paste -s Employee (means serial)
  - Join all the lines in the same files.
- paste -s -d ',' Employee
  - Join all the lines in the same files using the , delimiter
- paste Empoyee Manager
  - This command is use to display the both the file contents.
- paste -s Employee Manager
  - This command is use to display the first file data and second file data.

Continue...

- paste -s -d ',' Employee Manager
  - This command is use to join First file data after that second file data with the delimiter as ','
- paste - - < Employee
  - Merge a file by pasting the data into 2 columns

Translate commands (tr)

- Tr command is also mostly used in Unix filter command which stands for translate and used to translate file character by character.
- If both the SET1 and SET2 are specified and -d option is not specified then tr command will replace each character in SET1 with
- each character in same position in SET2
- So first create the file and store some info

Continue...

- tr abcd ABCD < test
  - So in the test file wherever abcd contents is present it will replaced by the ABCD contents.
- tr a-z A-Z < test
  - So in this command it will convert lower case a to z contents to upper case A to Z contents.
- tr [:lower:] [:upper:] < test
  - So this command is same as convert lower case contents to upper case contents.
- tr [:space:] ',' <test
  - This command is use where the space is available replace by comma symbol.
- tr -d 'a' < test
  - This command is use to delete the a character from the file.

Banner command

- This command is use to display the contents in the form of banner
  - Banner Welcoem to Unix

Sort Command

- Sort command in Unix is basically used to order the element in the file. Sort command sorts the element in the file by ASCII values. Sort command sort numerical values as well as it sorts the string n to the file.

Continue…

- sort sortDemo
  - It will display the contents in the ascending order.
- sort -r sortDemo
  - It will display the contents in the descending order.
- sort -u sortDemo
  - It will use to display remove the duplicate data from the file and sort the file in the ascending order.
- sort -n numberFile
  - It will use to sort the file using the numeric values other wise it will sort using the ASCI keys.

Continue...

- sort -k2 Employee
  - It will sort the data for the second column in ascending order (By default delimiter is consider as space)
- sort -r -k3 Employee
  - It will sort the data for the third column in descending order (By default delimiter is consider as space)
- sort -r -k2 -t ',' Employee
  - It will sort the data in reverse order for the second column with the delimiter is comma

Module 6

vi Editor

Objectives

Upon completing this module you should be able to understand the following:

- What Is vi?
- Starting and Ending a vi Session
- Cursor Control Commands
- Input Mode: i, a, O, o
- Deleting Text: x, dw, dd, dG
- Copying, moving and changing Text
- Searching for Text: /, n, N
- Find and Replace

What Is vi?

- A screen-oriented text editor
- Included with most UNIX system distributions
- Command driven
- Categories of commands include
  - General administration
  - Cursor movement
  - Insert text
  - Delete text
  - Paste text
  - Modify text

Vi editor mode

- command mode
- Input or insert mode
- Exit command mode

Starting and Ending a vi Session

vi [filename] Start a vi edit session of file

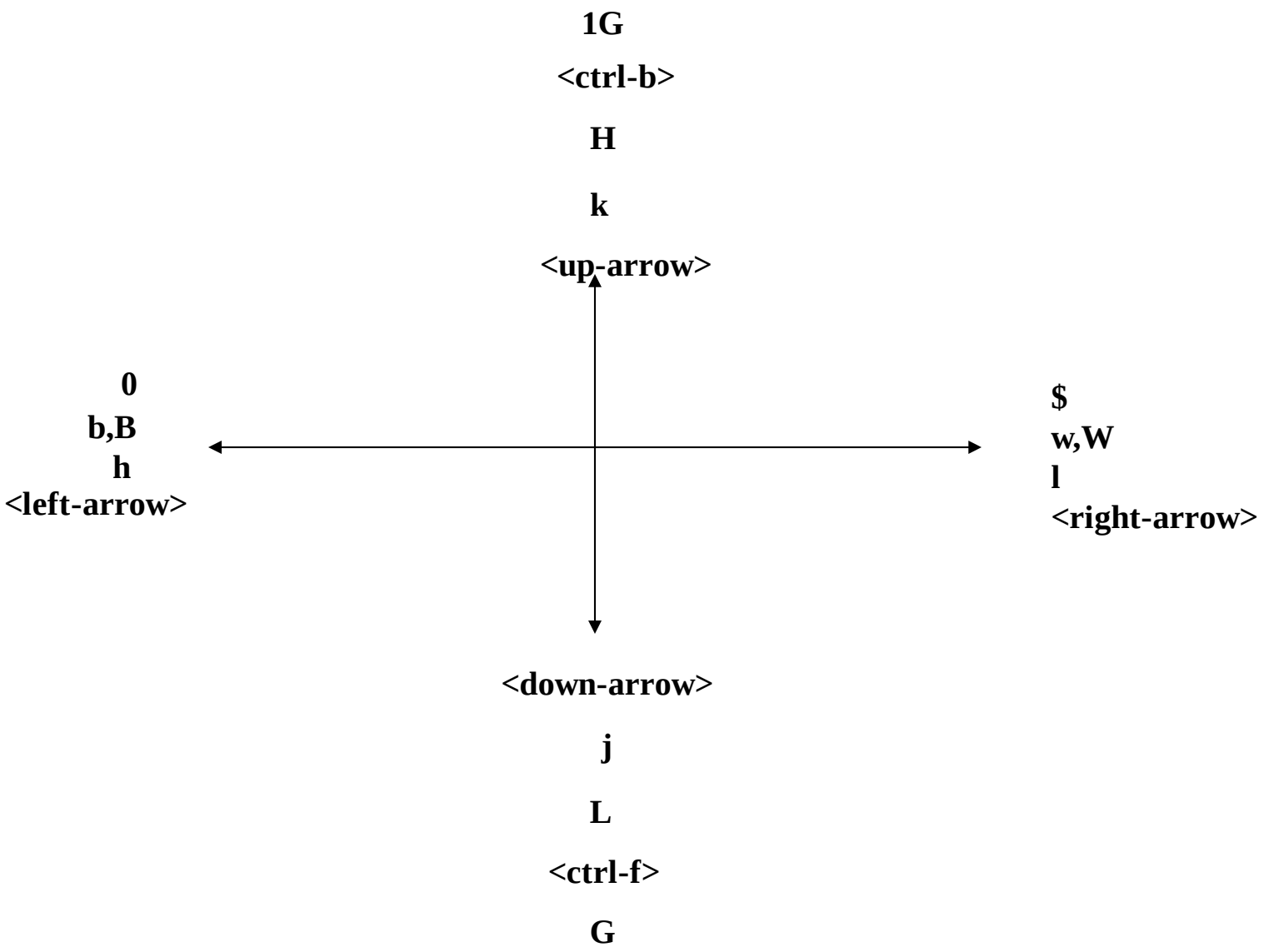
Example:

$ vi testfile
- If the file doesn't exist, it will be created
- Otherwise vi will open the existing file

All modifications are made to the copy of the file brought into memory.

:wq or :x or <shift-zz>	write and quit
:w	write
:q	quit
:q!	Quit without saving

Cursor Control Commands



Input Mode: i, a, O, o

i will insert character at the present cursor position
I will insert character at the beginning of the line
a will append character at the present cursor position
A will append character at the end of the line

o will insert a blank line below
O will insert a blank line above

Deleting Text: x, dw, dd, dG

x	deletes the current character
dw	deletes the current word
dd	deletes the current line
dG	delete all lines to end of file, including current line.

With any of these you can prefix a number

Example: 3dd will delete 3 lines

d$	to delete to the end of the line
d0	to to delete the start of the file

Copying, moving and changing Text

yy	copy the line
yw	copy the word
dw	cut the word
dd	cut the line

p will paste the yanked lines below

P will paste the yanked lines above

r	will overwrite the current character
R	will replace all text on the right of the cursor position
cw	will replace only the current word

Searching for Text: /, n, N

/director will locate for the first occurrence of the pattern ‘director’

n to locate the next occurrence

N to locate the previous occurence

Find and Replace

To find and replace the word **director** with **member :**

:1,$s/director/member/g

1,$ represents all lines in the file

g makes it truly global. g will ensure that all occuences in each line is replaced. Without g only the first occurrence of each line will be replaced.

Module 7

Grep commands

Grep commands

- Grep command actually searches the file which has the given pattern. Grep command is also used to search the string or regular expression in given file or files.

Continue...

- grep 'I' grepDemo
  - This will display the line contents which contains the I character
- grep 'a' grepDemo a1.txt
  - It will display the lines which contains both the file like grepDemo as well as a1.txt file.
- grep -i 'a' grepDemo a1.txt
  - It will ignore the case sensitive data.
- grep -c 'a' grepDemo
  - It will display the count of a character present in the grepDemo file.

Continue...

- grep -n -i 'a' grepDemo
  - It will display the content with the line number.
- grep -n -v 'a' grepDemo
  - It will display the contents apart from 'a' character line.
- grep -o [abc] a1.txt
  - It will display the first character must be start with a or b or c character.

Word pattern

- \< : start with
- >\ :end with

Grep world pattern

- grep 'a' grepDemo
  - It will display the a character must be contains
- grep '\<a' grepDemo
  - It will display only those line which start with a character
- grep 'a\>' grepDemo
  - It will display only those line which end with a character
- grep '\<a\>' grepDemo
  - It will display only those line which start and end with a character
- grep '\<[0-9]' grepDemo
  - It will display only those which start with 0 to 9 digits only.

Line pattern

- a ^ : Start of the line
- b $: End of the line

Grep line pattern commands

- --> grep '^a' grepDemo
  - Display only those line which start with a character
- --> grep "s$" grepDemo
  - Display only those line which end with a character
- --> grep "^[^TH]" grepDemo
  - Display only those line which doesn't start with the TH character
- --> grep "^....$" grepDemo
  - Display the line which contains only has four character words.
- --> grep -v "^$" grepDemo
  - This is use to delete the empty file.

Module 8

SED Commands

SED commands

- SED commands in Unix which will help to modify the files in Unix.
- SED command in Unix is basically used to replace and find the text but sed command also used to do many things apart from replacing the text.
- It is used to search and replace a string and it is multipurpose filter string.
- In SED commands we will use
  - s : means substitution
  - g : means every line all occurrences
  - without g : means every line 1st occurrences

Continue...

- sed s/sql/mysql/ sed_file
  - It is used to replace the sql content to mysql contents.
- echo day | sed s/day/night/
- It will convert day to night
- echo abc | sed s/[abc]/ABC/
  - or
- echo bbc | sed s/[abc]/ABC/
  - or
- echo cbc | sed s/[abc]/ABC/
  - It will display the ABCbc

Other operator in SED with regular expression

- ? --> zero or one
- * ---> zero or many
- + ----> 1 or many
- ^ --> Start with
- $ ---> end with

Continue...

- echo raj | sed s/raj/RajDeep/g
  - This will replace raj by RajDeep
- echo raj | sed s/Raj/RajDeep/i
  - This will ignore the case sensitive character
- echo raj | sed s/^/ /g
  - This will give the left side two spaces.
- sed '$d' a1.txt
  - This command will help to delete the last content from the file.
- sed '2d' a1.txt
  - This command is use to delete the 1 line in the file.

Continue...

- sed 1,3d a1.txt
  - This command is use to delete the content from file from 1 to 3 line
- sed -n '2p' a1.txt
  - This command is use to display the content from the 2nd line
- sed -n '1,3p' a1.txt
  - This command is use to display the content from the 1 to 3rd line
- sed -i 1d a.txt
  - This command is use to display first line contents from the file.

Continue...

- sed -e s/a/A/g -e s/b/B/g a1.txt
  - We can execute the mutliple command using the option as -e
- sed '2,6 s/a/A/g' a1.txt
  - We can change the content from 2 to 6 from from a to A in a1.txt file.
- sed p a.txt
  - This command is use to display the content twice from the file.

Module 9

AWK

AWK Commands

- Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file.
- Awk has built in string functions and associative arrays.
- Awk support most of the operators, conditional blocks and loops available in C language.

AWK Basic Syntax

- awk 'begin {start_action} {action} end {stop_action}' filename

Sample file

- cat > Employee

  1 Raj 12000

  2 Seeta 16000

  3 Veeta 18000

  4 Ramesh 19000

AWK Commands

- -->awk '{print $1}' Employee.txt
  - It will display the 1 column from the file with delimiter is default space operator
- -->awk –F ',' '{print $1}' Employee.txt
  - It will display the 1 column from the file with delimiter is comma operator
- -->awk –F ',' '{print $1,$2,$3}' Employee.txt
  - It will display the 1,2 and 3 column from the file

Sum of two number using awk

- awk "BEGIN {a=10;b=20} {sum=a+b} END {print sum}" Employee.txt
  - The output will display as 30
- awk "BEGIN {a=10;b=20; sum print sum}"
  - The output will display as 30 without any file

Awk with looping

- awk 'BEGIN {for(i=0;i<10;i++) print i}‘
  - It will display the number start from 0 to 9
- awk '{print NF}' Employee.txt
  - It will display the number of column
- awk -F ',' '{print NF}' Employee.txt
  - NF : Number of Field
  - NR : Number of Record
- awk -F ',' '{print $2}' Employee.txt

Awk pre-defined function

- upperCase
- lowerCase

Grep Vs Sed Vs Awk

- Grep : Grep is useful if you want to quickly search for lines that match in a file. It can also return some other simple information like matching line numbers, match counts, and file name lists.
- Awk : It is an entire programming language. Built around reading CSV style files, processing the records, and optionally printing out a result data set. It can do many thing but it is not the easiest tool to use for simple task.
- Sed : It is useful when you want to make changes to a file based on regular expression. It allow you to easily match parts of line, makes modifications and print out results It is less expressive than awk that lends to somewhat easier for simple task.

Module 10

File Access Permissions

Objectives

Access to files is dependent on a user's identification and the permissions associated with a file. This module will show how to Understand the read, write, and execute access to a file Permissions

- ls (11, ls -l)
- chmod
- umask
- chown
- chgrp
- su
- setuid, setgid, sticky bit

File Permission

- Unix file permissions, directories, how to give the permission using chmod command. File permissions are most important component of Unix operating system because it secures the files by giving the specific permissions.
- **Owner Permission :**
  - Owner permissions are Unix file permissions given to the specific owner to open, read and write or execute the file.
- **Group Permission :**
  - Group permission are Unix file permissions to specific group of users to open, read and execute the file.
- **World Permissions :**
  - These are Unix file permissions to other user to perform open, read, write, execute file permission.

Default permission for regular file

- Open file — Read
- Write or Modify — Write
- Execute — Read and Execute

Default file Information

ls -l a1.txt

This command is use to check the default file permission

When you run this command it will display the information as

Field1	Field2	Field3	Field4	Field5	Field6	Field7	Field8	Field9	Field10
-	rw-	r--	r--	1	root	root	209	Jun 16	printcap

All fields information

- The first field could be
  - - for file, d for directory , l for link
- The second, third and fourth field
  - Second - The owner has over the file
  - Third -- the group has over the file
  - Fourth -- Everybody else has over the file
- Fifth
  - This field specifies the number of links or directories inside this directory

Continue…

- Sixth
  - The user that owns the file or directory
- Seventh
  - The group that file belongs to, and any user in that group will have the permissions given in the third filed over that file.
- Eight
  - The size in bytes, you may modify this by using the -h option together with -l will have the output in k,M,G for a better understanding.
- Ninth
  - The date of last modification
- The tenth field
  - The name of the file.

Chmod commands

- chmod u+x a1.txt
  - This command is use to add the execute permission to the file.
- chmod u-r a1.txt
  - This command is use to remove the read permission to the file.
- chmod u-w a1.txt
  - This command is use to remove the write permission to the file
- chmod u+w a1.txt
  - This command is use to add the permission to the file.
- chmod u-wx,g-x,o-x a1.txt
  - This remove permission of Write, execute for user and execute for group and others

Chmod with numbers

- chmod 754 a1.txt
  - Here the digits 7,5 and 4 each individually represents the permissions for the user, group and others.
  - Each digits is a combination of the numbers 4, 2, 1 and 0.
    - 4 stands for "read"
    - 2 stands for "write
  - 1 stands for "execute and
  - 0 stand for no permission
- So 7 is the combination of permission 4+2+1(read, write and execute), 5 4+0+1 (read, no write and execute) and 4 is 4+0+0 (read, no write no execute).

Types of Access

There are three types of access for each file and directory:

Read

files:	contents can be examined.
directories:	contents can be examined.

Write

files:	contents can be changed.
directories:	contents can be changed.

Execute

files:	file can be used as a command.
directories:	can become current working directory.

Change Permissions (Symbolic Notation)

Syntax:

chmod ***mode_list file...*** Change permissions of file(s)

mode_list [who[operator]permission] [,...]

who	user, group, other or all(u/g/a)
operator	+ (add), - (subtract), = (set equal to)
permission	read, write, execute

Example:

Original permissions: mode user group other

rw-r--r-- rw- r-- r--

$ chmod u+x,g+x,o+x file *or $ chmod +x file*

Final permissions: mode user group other

rwxr-xr-x rw-x r-x r-x

Change Permissions(Octal Notation)

File and directory permissions can also be specified as an octal number:

Read Permission :4

Write Permission :2

Execute Permission:1

We can just add the numbers to specify the permission for each category

Example: 6 means read and write, 7 means read, write and execute

eg:

$ chmod 664 f1 will give read and write permissions for owner and group while only read for others

Default Permissions

The default protections for newly created files and directories are:

File	-rw-r—r--	644
Directory	drwxr-xr-x	755

These default settings may be modified by changing the **umask** value.

umask - Permission Mask

Umask specifies what permission bits will be set on a new file or directory when created.

New Directory:	777 – 022	755	→ rwxr-xr-x
New File :	666 – 022	644	→rw-r—r—

The default value of umask is set in /etc/profile. This can be Changed for all the user or a particular user

chown - Change File Ownership

Syntax:
chown ***owner [:group] filename…***

Changes owner of a file(s) and, optionally, the group ID

Example:

```
$ id
uid=101(user1),  gid=101(group1)
$ cp f1 /tmp/user2/f1
$ ls -l /tmp/user2/f1
-rw-r----- 1 user1 group1 3967 Jan 24 13:13 f1
$ chown user2 /tmp/user2/f1
$ ls -l /tmp/user2/f1
-rw-r----- 1 user2 class 3967 Jan 24 13:13 f1
```

Only the owner of a file (or root) can change the ownership of the file.

The chgrp Command

Syntax:

chgrp *newgroup filename ...*

Example:

```
$ id
uid=303 (user3),   gid=300 (class)
$ ls -l f3
-rw-r----- 1 user3 class 3967 Jan 24 13:13 f3
$ chgrp class2 f3
$ ls -l f3
-rw-r----- 1 user3 class2 3967 Jan 24 13:13 f3
$ chown user2 f3
$ ls -l f3
-rw-r----- 1 user2 class2 3967 Jan Raj 24 13:13 f3
```

su - Switch User Id

Syntax:

su [*user_name]*

Change your effective user ID and group ID

Example:

```
$ ls -l  f1
-rwxr-x---  1 user1   group1 3967 Jan 24 23:13 class_setup
$ id
uid=303 (user1), gid=300 (group1)

$ su – user2
Password:
$ id
uid=400 (user2), gid=300 (group1)
```

Special File Permissions – setuid, setgid, sticky bit

- **setuid** – changes the effective user id of the user to the owner of the program
  - chmod u+s f1 or
  - chmod 4744 f1
- **setgid** – changes the effective group id of the user to the group of the program
  - chmod g+s f1 or
  - chmod 2744 f1
- **sticky bit** – ensures the deletion of files by only file owner in a public writable directory
  - chmod +t f1 or
  - Chmod 1744 f1

Module 7

Shell Features

Objectives

Upon completing this module you should be able to understand the following:

- Shell functionalities
- Commonly Used Shells
- BASH Shell Features
- Aliasing, Command History
- Re-entering Commands The
- User Environment Setting
- Shell Variables Variable
- Substitution Command
- Substitution
- Transferring Local Variables to the Environment
- Functions of a Shell before Command execution.

Shell functionalities

- command execution
- environment settings
- variable assignment
- variable substitution
- command substitution
- filename generation
- I/O redirection
- Interpretive programming language

Commonly Used Shells

/bin/bash
/bin/ksh
/bin/sh
/bin/csh
/bin/rksh
/bin/rsh
/bin/tcsh
/bin/zsh

- Bash shell Korn shell
  Bourne shell C Shell
- Restricted Korn shell
  Restricted Bourne shell
- An amended version of the C shell The Z shell

BASH Shell Features

- A shell user interface with some advanced features:
  - Command aliasing
  - File name completion
  - Command history mechanism
  - Command line recall and editing
  - Job control
  - Enhanced cd capabilities
  - Advanced programming capabilities

Aliasing

Syntax:

alias [*name[=string]]*

Examples:

```
$ alias dir=ls
$ alias mroe=more
$ alias mstat=/home/tricia/projects/micron/status
$ alias laser="lp -dlaser"
$ laser fileX
request id is laser-234 (1 file)
$ alias       displays aliases currently defined
$ alias mroe  displays value of alias mroe

mroe=more
```

Command History

- The shell keeps a history file of commands that you enter.
- The history command displays the last 16 commands.
- You can recall, edit, and re-enter commands previously entered.

Syntax:

history [-*n*| *a z]* Display the command history.

Example:

$ history -2 *list the last two commands*

15 cd

16 more .profile

$ history 3 5 list command numbers 3 through 5

3 date

4 pwd

5 ls

Re-entering Commands

- You type **r *c*** to re-enter command number ***c***.

Example:

$ history 3 5 *list command numbers 3 through 5*
3 date
4 pwd
5 ls

$ r 4 run command number 4
pwd
/home/kelley

The User Environment

- Your environment describes your session to the programs you run.

Syntax:

env

Example:

```
$ env
HOME=/home/gerry
PWD=/home/gerry/develop/basics
...
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin:\
/home/gerry/bin
```

Two Important Variables

- The *PATH* variable
  - A list of directories where the shell will search for the commands you type
- The *TERM* variable
  - Describes your terminal type and screen size to the programs you run

```
$ env
  ...
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
$ TERM=70092
```

Setting Shell Variables

Syntax: ***name=value***

Example:

```
color=blue
PATH=$PATH:/usr/ucb
```

.
.
.

Variable Substitution

$ echo $PATH
/usr/bin:/usr/contrib/bin:/usr/local/bin
$ PATH=$PATH:$HOME:.
$ echo $PATH
/usr/bin:/usr/contrib/bin:/usr/local/bin:/home/user3:.
$ echo $HOME
/home/user3
$ file_name=$HOME/file1
$ more $file_name
<contents of /home/user3/file1>

Command Substitution

Syntax:

$(*command)*

Example:

$ dt=date This will store the string date in the variable dt not the value

$ dt=$(date) This will do command substitution and store the result of date command in the variable dt

note: instead of $(date) we can also use `date`

Displaying Variable Values

```
$ echo $HOME
/home/user3

$ env
HOME=/home/user3
….
SHELL=/usr/bin/sh

$ set
HOME=/home/user3
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
….
color=lavender
count=3
dir_name=/home/user3/tree


$ unset dir_name
```

Transferring Local Variables to the Environment

Syntax:

export ***variable*** Transfer ***variable*** to environment

Functions of a Shell before Command execution.

- Searches for a command
- Substitutes Shell variable values.
- Command substitution, I/O redirection
- Interpreted programming interface

Module 8

Standard I/O Redirection

Objectives

Upon completing this module you should be able to understand the following:

- The Standard Files
- Input Redirection
- Output Redirection
- Creating a file with cat
- Error Redirection
- Combined Redirection
- Split Outputs

The Standard Files

Standard Input File (0)	Keyboard
Standard Output File (1)	Monitor
Standard Error File (2)	Monitor

operators:

standard input redirection	0<	or <
standard output redirection	1>	or >
standard Error redirection	2>	

Input Redirection

Default standard input

```
$ mail user1
subject: Hi
Test mail
<ctrl-d>
```

Redirect Input from a file: <

```
$ mail user1 < letter
```

Output Redirection

Default standard output:

$ ls

f1 f2 f3

Redirect output from a file: >

$ ls > ls.out

Redirecting and appending output to a file: >>

$ who >> who.out

Creating a file with cat

While normally used to list the contents of files, using **cat** with redirection can be used to create a file

```
$ ls
file1    file2    file3
```

Using redirection

```
$ cat > newfile
file created by using the output redirection
<ctrl-d>

$ ls
file1    file2    file3    newfile
```

Error Redirection

Default standard error:

$ cat file1 file3

This is file1

cat: cannot open file2

Redirecting error output to a file: 2> (To append: 2>>)

$ cat file1 file3 2>err

This is file1

$ cat err

cat: cannot open file2

$ cat file1 file3 2> /dev/null

Combined Redirection

Combined redirects:

$ command > outfile 2>errfile <infile

$ command >> appendfile 2 >> errorfile <infile

Association Examples:

$ command > outfile 2 >&1

note: This is NOT the same as above

$ command 2>&1 >outfile

Split Outputs

The **tee** command reads standard input and sends the data to both standard output and a file.

Example:

```
$ ls | tee ls.save | wc –l
```

Module 9

Filters & other commands

Objectives

Upon completing this module you should be able to understand the following:

- grep, sort, find ,cut , tr
- ftp
- paste command
- split command
- uniq command
- diff(Differential file comparator) Command
- cmp command
- file command
- tar (tape archive program)
- Restoring Files

grep

Search for lines matching specified **pattern**

syntax:

grep [options] pattern [file1 file2 …..]

Example:

emp.dat

Robin Bangalore

John Chennai

Rina Bangalore

$ grep Bangalore emp.dat

Robin Bangalore

Rina Bangalore

grep with Regular Expressions

grep ‘regular_expression’ file

Valid metacharacters

.	Any single character
*	Zero or more occurences of the preceding character
[aA]	Enumeration: a or A
[a-f]	Any one of the characters in the range of a through f
^a	Any lines that start with a
z$	Any lines that end with z

grep options

-v	print lines that **do not match**
-c	print only a **count** of matching lines
-l	print only the **names** of the files with matching lines
-n	**number** the matching lines
-i	**ignore the case** of letters when making comparisons
-w	do a **whole word** search

sort command

- The sort command sorts lines and writes the result to standard output:

- $ sort [-t delimiter] [+field[.column]] [options]

- **Options:**

- -d sorts in dictionary order. Only letters, digits and spaces are considered in comarisons

-r reverses the order of the specified sort

-n sorts numeric fields in arithmetic value

sort examples

```
$ cat animals
dog.2
cat.4
elephant.10
rabbit.7

$ sort animals
cat.4
dog.2
elephant.10
rabbit.7

$ cat animals | sort +0.1
rabbit.7
cat.4
elephant.10
dog.2

$ cat animals | sort -t. -n +1
dog.2
cat.4
rabbit.7
elephant.10
```

find command

Search one or more directory structures for files that meet certain specified criteria

Display the names of matching files or execute commands against those files

find path expression

Examples:

$ find / -name file1 ---search for files in whole system with name file1

$ find / -user user1 –**exec** rm { } \; ---search for the files owned by user1 and delete them

$ find / -user user1 –**ok** rm { } \; ---search for the files owned by user1 and delete them interactively

find options

-type	f	ordinary file
	d	directory
-size	+n	larger than "n" blocks
	-n	smaller than "n" blocks
	n	equal to "n" blocks
-mtime	+x	modified more than "x" days ago
	-x	modified less than "x" days ago
-perm	onum	access permissions match "onum"
	mode	access permissin match "mode"
-user	user1	finds files owned by "user1"
-o		logical "or"
-newer	ref	searches for files that are newer than the reference file.

cut command

Used to cut fields from each line of a file or columns of a table

cut –d: -f1,5 /etc/passwd
cut –c2 test second character of each line
cut –c-2 test first 2 characters of each line

tr command

Used to substitute the values

tr [a-z] [A-Z] will convert each small letter to caps

cat test | tr [a-z] [A-Z] will convert the small letters to caps in display

tr –s " " will squeeze multiple space occurences of space into one

ftp

The ftp Command Syntax :

$ ftp hostname

get	Gets a file from the remote computer.
put	Sends a local file to the remote system
mget	get multiple files from the remote system
mput	Sends multiple files to the remote system
cd	changes the directory in the remote system
lcd	changes the directory in the local system
ls	Lists files on the remote computer.
?	Lists all ftp commands
help command	Display a very brief help message for command.
bye	

The paste command

Used to paste files vertically

eg. file1 file2 paste file1 file2

name	**address**	**age**	**name**	**address**	**age**
ram	mvm	29	ram	mvm	29
raghu	rjnagar	25	raghu	rjnagar	25

(To rearrange f1 f2 f3 to f3 f2 f1, cut each field,put into file then paste)

By default paste uses the **tab** character for pasting files, but –d to specify one

eg. paste –d \| file1 file2

name	**address|age**
ram	mvm|29
raghu	rjnagar|25

The *split* command

Used to split up the files:

This command breaks up the input into several equi line segments.

by defalult, split breaks a file into 1000 line pieces(or whatever is left)

split creates files like xaa,xab,xac …. xaz then xba, xbb …. xbz and … xzz . So there can be 676(26*26) files

split –5 chap (here the chap file is broken into files of 5 lines size)

split chap small (the file names will be smallaa,smallab …. smallzz)

The uniq command

- used in EDP environment
- to clear the duplicate entries by mistakes

eg. dept.list

```
01|accounts|6213
01|accounts|6213
02|admin|5423
03|marketing|6521
03|marketing|6521
```

uniq dept.list

```
01|accounts|6213
02|admin|5423
03|marketing|6521
```

The input to uniq must be ordered data. So sort and send the data

sort dept.list | uniq -uniqlist (uniqlist is the output file)

The *diff(Differential file comparator)* Command

Analyzes text files

Reports the differences between files

diff [-options] file1 file2

The *cmp* command

```
$ cmp names names.old
names names.old differ: byte 6, line 1

$cmp -l names names.old
6   12  151
7  102  156
8  157  145
...
...
...
cmp: EOF on names
```

The *file* command

The file command tells the type of file

Example:

```
$ file /usr/bun/vi
/usr/bin/vi:executable (RISC system/6000) or object
module

$ file c1
c1:      ascii text

$ file /usr/bin
/usr/bin:  directory
```

tar (tape archive program)

create a disk archive that contains a group of files or an entire directory structure

-c copy
-x extract
-t list [only one can be present at a time]
-f to specify the device name

tar [–]cvf etc.tar /etc/*

Restoring Files

tar –xvf tarfile

Example:
tar –xvf etc.tar

selective extraction
tar –xvf etc.tar /etc/passwd

tar –tvf etc.tar will display the archive

Module 10

Controlling Processes

Objectives

Upon completing this module you should be able to understand the following:

- Monitoring Process
- Controlling Processes
- Terminating Processes
- Kill Signals
- Running Long Processes
- Job Control

Monitoring Process

The **ps** command dislays process status information

$ ps -f

UID	PID	PPID	...	TTY	...	COMMAND
john	202	1	...	tty0	...	-ksh
john	204	202	...	tty0	...	ksh
john	210	204	...	yyu0	...	ls -R /
john	212	204	...	tty0	...	ps -f

Controlling Processes

Foreground Processes:

```
$ ls -lR / >lsout
```

Background Processes:

```
$ ls -lR / >lsout &
```

Terminating Processes

Foreground Processes:

ctrl-c Interrupt key, cancels a forground process

kill Sometimes the kill command is used to terminate foreground processes

Background Processes:

kill kill is the only way to terminate background processes

Kill Signals

Sig **Meaning**

1.hangup- you logged out while the process was still running

2.interrupt- you pressed the interrupt(break) key sequence ctrl-c

03 quit -you pressed the quit key sequence ctrl-\

09 Kill signal:
The most powerful and risky signal that can be sent
Signal cannot be avoided or ignored!

15 Termination signal(Default): Stop a process
Signal can be handled by programs

Running Long Processes

The nohup command will prevent a process from being killed if you log off the system before it completes:

```
$ nohup ls -lR / >lsout &
  [1]    100
```

If you do not redirect output, nohup will redirect output to a file nohup.out

```
$ nohup ls -lR / &
  [1]    102
Sending output to nohup.out
```

Job Control

• jobs	Lists all jobs
• <ctrl-z>	Suspends foreground task
• fg %jobnumber bg	Execute job in foreground
%jobnumber stop	Execute job in background
%jobnumber	Suspends background task

Job Control Examples

```
$ ls -R / > out 2> errfile &
 [1]        273

$ jobs
 [1]   +  Running ls -R / > out 2>errfile &

$ fg %1
ls -R / > out 2>errfile
<ctrl-z>
[1] + Stopped (SIGTSTP) ls -R / >out 2>errfile &

$ bg %1
% jobs
[1] + Running                    ls -R / >out 2>errfile &

$ kill %1
[1] + Terminating  ls -R / > out 2> errfile &
```

Module 11

Shell Programming Concepts

Objectives

Upon completing this module you should be able to understand the following:

- echo & read commands
- Command Line Arguments
- The logical operators && and ||
- The if Conditional
- Numeric comparison with test
- numeric, string and file comparison
- The case CONDITIONAL
- expr: Computation
- Command Substitution
- sleep & wait
- while loop, until and for loops

echo & read commands

echo used to display messages on the screen

read used to accept values from the users, make programming interactive

eg.

```
echo “Enter ur name “

read name

echo “Good Morning  $name”
```

Command Line Arguments

Shell programs can accept values through

1. read [Interactive –used when there are more inputs]

2.From the **command Line** when u execute it[Non interactive- used when only a few inputs are there]

For eg. sh1 20 30 Here 20 & 30 are the command Line arguments.

Command Line args are stored in Positional parameters

$1 contains the first argument, **$2** the second, **$3** the third etc.

$0 contains the name of the file, **$#** stores the count of args
$* displays all the args

An example of command Line args.

```
#! /bin/bash
echo “Program: $0”
echo “The number of args specified is $#”
echo “The args are $*”
```

```
sh sh1 prog1 prog2
```

The parameter $?

$? can be used to know the exit value of the previous command . This can be used in the shell scripts:

The logical operators && and | |

These operators can be used to control the execution of command depending on the success/failure of the previous command

eg.

grep ‘director’ emp1.lst && echo “Pattern found in file”

grep ‘manager’ emp2.lst || echo “pattern not found”

or grep ‘director’ emp.lst’ && echo “Pattern found” || echo “Pattern not found”

exit : Script Termination

exit command used to prematurely terminate a program. It can even take args.

eg. grep “$1” $2 | exit 2

echo “Pattern found – Job over”

The if Conditional

if condition is true

then

  execute commands

else

  execute commands

fi

else is not a must :

if condition is true

then

execute commands

An example of if

eg1:

```
if grep “director” emp.lst
then echo “Pattern found”
else echo “Pattern not found”
fi
```

*Every **if** must have an accompanying **then** and **fi**, and optionally **else***

Numeric comparison with *test*

test is used to check a condition and return true or false

Relational operators used by if

Operator	Meaning
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Gfreater than or equal to
-lt	Less than
-le	Less than or equal to

An example of *test* with numeric

```
echo “Enter age”
read age
if test $age –ge 18
then echo “Major”
else echo “Minor”
fi
```

[ ] shorthand for *test*

```
echo “Enter age”
        read age
        if [ $age –ge 18 ]
        then echo “Major”
        else echo “Minor”
        fi
```

if –elif :Multi-way Branching

```
if condition
then execute commands
elif condition
then execute commands
elif condition
then execute commands
else execute conditions
fi
```

```
echo “Enter marks”
read marks
if [ $marks –ge 80 ]
then echo“Distinction”
elif [ $marks –ge 60 ]
then echo “First Class”
else echo “Pass”
fi
```

can have as many elif , else is optional

test String comparison

test with Strings uses = and !=

String Tests used by **test**

Test	**Exit Status**
-n str1	true if str1 is not a null string
-z str1	true if str1 is a null string
s1 = s2	true if s1 = s2
s1 != s2	true if s1 is not equal to s2
str1	true if str1 is assigned a not null

file tests

File related Tests with *test*

Test	**Exit Status**
-e file	True if file exists
-f file	True if fie exists and is a regular file
-r file	True if file exists and is readable
-w file	True if file exists and is writable
-x file	True if file exists and is executable
-d file	True if file exists and is a directory
-s file	True if the file exists and has a size >0

The *case* CONDITIONAL

case used to match an expression for more than one alternatives and uses a compact construct to permit multi way branching

case expression in

pattern1) execute commands;;

pattern2) execute commands

esac

An example for *case*

```
echo "1. List of files  2. Processes of user  2. Quit"
echo "Enter choice"
read choice
case "$choice" in
        1) ls ;;
        2) ps –f ;;
        3) exit;;
        *) echo "Wrong Choice"
esac
```

case Matching Multiple patterns

```
echo “Do u want to continue (y/n) “
read answer
case “$answer” in
y|Y) echo “Good” ;;
n|N) exit ;;
esac
```

case can also use wild cards like [yY][eE]* or [nN][oO]

expr: Computation

Shell doesn't have any compute features-depend on *expr*

expr 3 + 5

expr $x + $y

expr $x - $y

expr $x | $y*

expr &x / $y

expr $x % $y **division gives only integers** as expr can handle only integers

Command Substitution

```
x=5
x=`expr $x +1`
echo $x
```

it will give 6

sleep & wait

sleep 100 the program sleeps for 100 seconds

wait wait for completion of all background processes

wait 138 wait for completion of the process 138

while loop

while condition is true

do

execute commands

done

eg.

```
x=1
while test $x -le 10
do
echo $x
x=`expr $x + 1`
done (while true will set infinite condition)
```

until : while's complement

```
x=1
until test $x –gt 10
do
echo $x
x=`expr $x + 1`
done
```

For loop

for variable in list

do

execute commands

done

eg. for x in 1 2 3 4 5

do

echo “Value of x is $x”

done

The list in for loop

1. List from variables

```
for var in $PATH $HOME $MAIL
do
echo $var
done
```

2. List from Wild cards

```
for file in *.c
do
cc $file –o $file{x}
done
```

3. List from positional parameters

```
for pattern in $*
do
grep"$pattern" emp.lst || echo "pattern $pattern      not found"
done
```

4. List from command substitution

```
for file in `cat wdwiswt`
```

Labs

1. develop a script that accepts a file name as argument and displays the last modification time if the file exists and a suitable message if it doesn't.
2. write a script that accepts one or more file names as arguments, and converts them all to upper case provided they exist.
3. Devise a script that lists files by modification time when called with lm and by access time when called with la. By default the script should show the listing of all files in the current directory
4. Write a script that uses find to look for a file and echo a suitable message if the file is not found. You must not store the output of the find to a file.
5. use the "if then" construct to write a script that will check for at least two parameters present on the command line. Output an appropriate

   error message.
6. Write a shell script which will compare two numbers passed as command line arguments and tell which one is bigger.
7. Write a shell script which will add all the numbers between 0 and 9 and display the result.
8. Write a script which will give 4 choices to the user 1. ls 2. pwd 3. who 4. exit and execute the command as per the users choice

Labs contd.

9. Execute the date command with the proper arguments so that its output is in a mm-dd-yy format.
10. From your home directory, make the following directories with a single command line

    d1, d1/d2, d1/d3, d1/d2/d4
11. create a file f1 in the home directory. Change the permissions to 777.
12. create a file f2 in your home directory and set setuid and sticky bit permissions on the file.
13. Write the command which will display all the files in the system having setuid set.
14. write the command to find all the files in your home directory hierarchy modified 1yr back and having a size more than 10 mb and delete the matching files interactively.
15. write two commands which will display all the ordinary files in the system

Labs contd.

16. Write a script that lists all the start scripts in a specified rc#.d directory
    a. Name the script rcscripts
    b. Write the script so that it doesn't require command-line arguments
    c. For all start scripts in /etc/rc3.d directory print
- The script name
- The inode number of the script
- All other files under /etc with the same inode number

17. Print all entries from /etc/passwd for users who have ksh as their login shell
18. Print the lines from /etc/group that contain the pattern root preceded by the line number
19. Print the number of users in the system.
20. Print the users whose account is locked