Unix Programming

UNIT-I

B.Tech(CSE)-V SEM

Syllabus

UNIT-I : Introduction to UNIX: The UNIX Operating System, A brief history of UNIX, The UNIX Architecture, Basic features of UNIX. General Purpose Utilities- cal, date, man, echo, bc, clear, passwd, who, whoami,uname Directory Handling Commands: pwd, cd, mkdir, rmdir. File Handling Utilities - cat, touch, cp, ls, rm, mv, nl, pg,tar,wc Displaying Commands: more,head,tail, simple filters and commands: cmp, comm., ulink, diff, find, cut, paste,sort, uniq, tr, finger. Disk Utilities– du, df, mount, umount.Process Utilities–ps, kill. Networking Utilities– ping, telnet, rlogin, ftp, finger.

UNIT-II : THE FILE SYSTEM : Types of Files, Directories and Files, UNIX File System, Absolute and relative pathnames, File Attributes and Permissions ,The File Command -knowing the File Type, Chmod Command-Changing File Permissions, Chown Command-Changing the Owner of a File, Chgrp Command- Changing the Group of a File. Vi editor-editing with vi, moving the cursor, editing, copying and moving text, pattern searching.

UNIT-III : Introduction to Shell Programming : Shell Variables-The Export Command-The Profile File a Script Run During Starting-The First Shell Script-The read Command-Positional parameters-The $? Variable knowing the exit Status-More about the Set Command-The Exit Command-Branching Control Structures-Loop Control Structures-The Continue and Break Statement-The Expr Command: Performing Integer Arithmetic-Real Arithmetic in Shell Programs-The here Document(<<)-I/O Redirection, The Sleep Command-Debugging Scripts-The Script Command-The Eval Command-The Exec Command. Command Line Structure-Met characters.

Syllabus

UNIT-IV :Regular Expressions:grep, egrep, fgrep, Sed- line addressing, context addressing, text editing,substitution.
Programming with awk: syntax of awk programming statement, structure of awk script, variables ,records fields, and special variables, patterns, operators ,simple input files, awk programming- simple awk programming, awk control structures, looping, functions in awk.

UNIT-V: Unix process: What is a process, process structure, process identifiers, starting new process, waiting for a process, zombie process, system call interface for process management - fork, vfork, exit, wait, waitpid, exec system call.

UNIT : VI Signals : Signal functions, unreliable signals, interrupted system calls, kill and raise functions, alarm, pause functions, abort, sleep functions.

Text Books:

Introduction to Unix and shell programming, M G venkateshmurthy, Pearson education
Advanced programming in the unix environment, W. Richard Stevens, 3rd Edition, Pearson education

REFERENCES

1. Unix and shell Programming, B.A. Forouzan& R.F. Giberg, ,Thomson, First Edition, NewDelhi, 2003.

Academic Year:2020-21

Course Title: Unix Programming
Year&Sem:III & V SEM

Syllabus Details

S.No.	COURSE OUTCOMES (After completion of the course, The Learner is able to)	KNOWLEDGE LEVEL
CO1	Illustrate the UNIX basics and the working of the built in commands in Unix	K2
CO2	Demonstrate the file system and change the permissions associated with files	K2
CO3	Develop basic programs using shell script	K3
CO4	Demonstrate the grep family and data transforming programs sed, and awk	K2
CO5	Construct programs for process system calls	K3
CO6	Explain the concept of signals and its system call	K2

Pre-Requisites towards Unix:

- What is an Operating System?
- What are the various functions of an Operating System?
- What are the different types of Operating systems?
- What is a multi-user operating system?
- What is a task or a process?
- What are the different types of processes?
- What is a multi-tasking operating system?

What is an Operating System?

- It is an interface between a computer user and computer hardware.

- Every computer must have at least one OS to run other programs.

Functions of an Operating system:

Types of an Operating Systems:

An Operating System can be categorized into:

1. Single User – Single Tasking operating system

2. Single User – Multi Tasking operating system

3. Multi User – Multi Tasking operating system

Types of an Operating Systems(Contd..)

1. Single User – Single Tasking operating system: In this type of operating system only one user can log into the system and can perform only one task at a time. Ex: MS-DOS
2. Single User – Multi Tasking operating system: This type of operating system supports only one user to log into the system but a user can perform multiple tasks at a time, browsing internet while playing songs etc. Ex: windows 98,xp,vista, seven etc.
3. Multi User – Multi Tasking operating system: These type of operating system provides multiple users to login into the system and also each user can perform various tasks at a time. In a broader term multiple users can logged into system and share the resources of the system at the same time. Ex: UNIX, LINUX etc.

UNIT-I

Academic year:2020-21
Year-Semester:III-I
Course Title: Unix programming

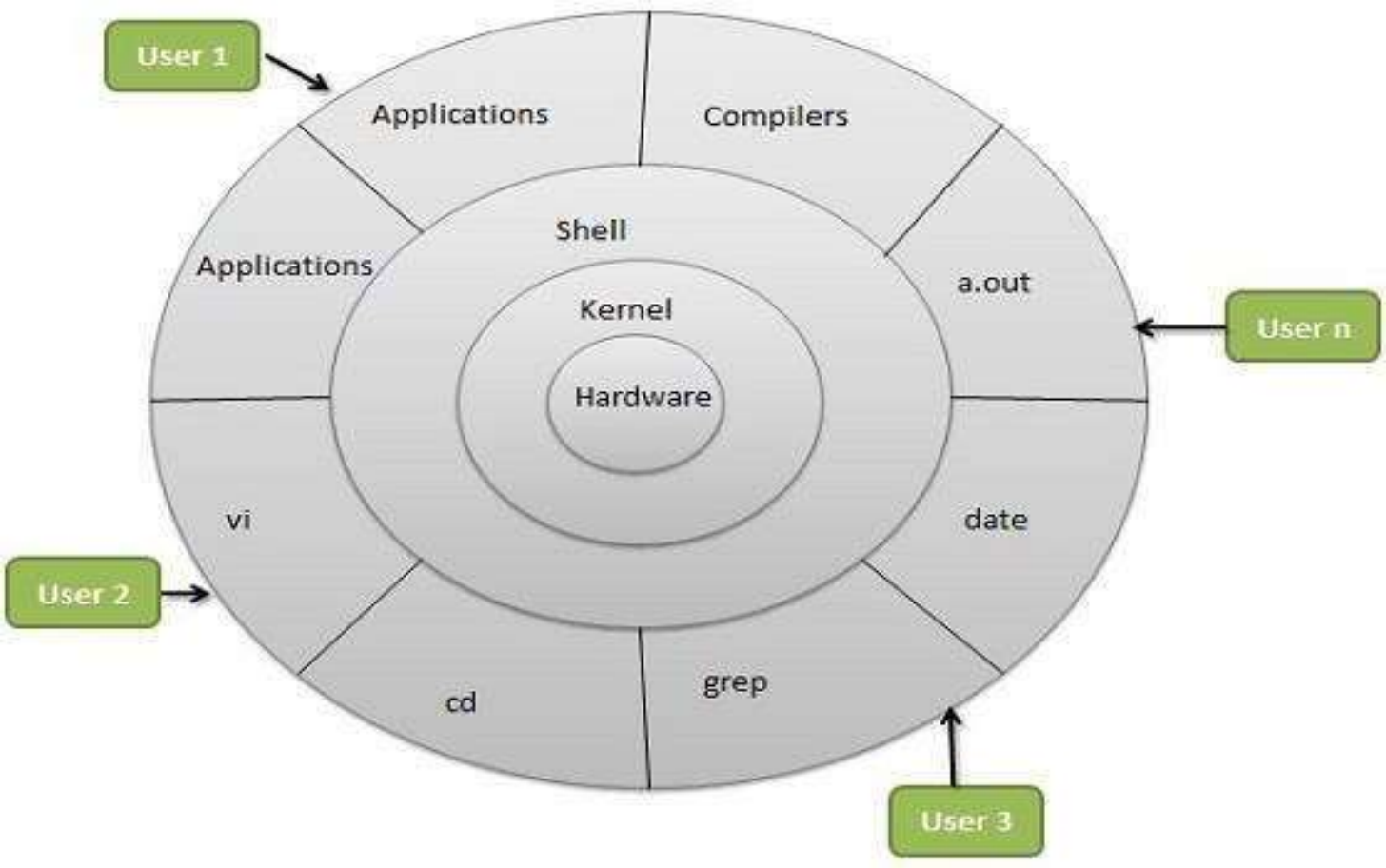
Introduction to Unix: The Unix OS

- Unix is a multi-user and multi-tasking Operating System.
- Unix was designed to let a number of programmers access at the same time and share its resources.
- The important features that make Unix favorite are:
  - Multitasking
  - Multiuser
  - Portability
  - Pattern Matching
  - Unix Communication

A Brief History of UNIX:

- First Version was created in Bell Labs in 1969.
- Some of the Bell Labs programmers who had worked on this project, Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy designed and implemented the first version of the Unix File System on a PDP-7 along with a few utilities. It was given the name UNIX by Brian Kernighan.
- 1973 Unix is re-written mostly in C, a new language developed by Dennis Ritchie.
- Being written in this high-level language greatly decreased the effort needed to port it to new machines
- 1980 BSD 4.1 (Berkeley Software Development)
- 1983 SunOS, BSD 4.2, System V
- 1988 AT&T and Sun Microsystems jointly develop System V Release 4 (SVR4). This later developed into UnixWare and Solaris 2.
- 1991 Linux was originated.

The UNIX Architecture:



- The UNIX operating system (OS) consists of :
  - a kernel layer
  - a shell layer
  - an application layer & files.

Kernel

- The kernel is the heart of the operating system.
- It interacts with the machine's hardware.
- It is a collection of routines written in C.
- It is loaded into memory when the system is booted.
- Main responsibilities:
  - Memory management
  - Process management (using commands: kill, ps, nohup)
  - File management (using commands: rm, cat, ls, rmdir, mkdir)
- To access the hardware, user programs use the services of the kernel via system calls.

Shell

- The shell interacts with the user.
- The shell is a command line interpreter (CLI).
- Main responsibilities:
  - interprets the commands the user types in and
  - dispatches the command to the kernel for execution
- There can be several shells in action, one for each user who's logged in.
- There are multiple shells that are used by the UNIX OS.
- For example: Bourne shell (sh), the C shell (csh), the Korn shell (ksh) and Bourne Again shell (bash).

Shell (Contd…)

- Meta characters provide short form representations and time also save.
- If shell sees any meta characters it expands that meta characters it expands that meta characters pass to kernel . If any blank spaces are appear they are removed at shell prompt and then passed to kernel.

Meta character	Description
*	To match 0 (or) more characters **Ex:** C*t It means ct,caat,cbt,……..
?	To match single character in a file **Ex:** C?t It means cat, cot,cit,cmt
[ ]	Specify a characters range or group of characters. **Ex:** [a-z] and [A-Z]

Types of Shells

- There are different types of shells available. Some of them are:

1. **The Bourne Shell(sh):** This can be used for any O.S. This is highly portable. It is developed by Stephen Bourn, so it is called Bourn Shell.
2. **C Shell:** This is created by Bill Joy. It has two advantages over the Bourne shell.
    - Aliasing of commands
    - History Commands
3. **Korn Shell:** It is a superset of Bourne shell. It offers a lot more capabilities and is decidedly more efficient than the other. It was designed to be so by David Korn of AT&T's Bell Labs.

List of Default Shell Prompts

Prompt	Shell
$(dollar)	Bourne and Korn Shells (sh, bash and ksh)
% (percent)	C Shells (csh and tcsh)
# (hash)	Any shell as root

Application and Files

- **Application**
  - This layer includes the commands, word processors, graphic programs and database management programs.
- **File**
  - A file is an array of bytes that stores information.
  - All the data of Unix is organized into files.
  - All files are then organized into directories.
  - These directories are further organized into a tree-like structure called the file system.

Utilities/ Commands in Unix:

- **Definition:**
- A **command** is a program that tells the unix system to do something.

  - Commands are generally issued by typing them in at the command line (i.e., the all-text display mode) and then pressing the ENTER key, which passes them to the shell.

Structure of a Unix Command:

- UNIX commands take the following general form:
  - **$Command [options] [arguments]**

- where an **argument** indicates on what the command is to perform its action, usually a file or series of files.
- An **option** modifies the command, changing the way it performs.
- Commands are case sensitive
- **command** and **Command** are not the same.
- We can provide more than one command in a single line using ;

Command structure (Contd…)

Options

- **Options** are generally preceded by a hyphen (-), and for most commands, more than one option can be strung together, in the form:

command -[option][option][option]

- **Example:** $ ls –l // -l option will perform a long list on all files in the current directory

- There must **not be any whitespaces between – and l.**
- Options are also arguments, but given a special name because they are predetermined.
- Options can be normally combined with only one – sign.
- Thus $ ls –l –a –t is same as $ ls –lat

Command Structure (Contd…)

Filename Arguments:

- Many UNIX commands use a filename as argument so that the command can take input from the file.
- If a command uses a filename as argument, it will usually be the last argument, after all options.

Example:

- **$ ls -lat chap01 chap02 chap03 # Multiple filenames as arguments**
- **$ rm file1 file2 file3**

- The command with its arguments and options is known as the command line.
- This line can be considered complete only after the user has hit [Enter].
- The complete line is then fed to the shell as its input for interpretation and execution.

Command Structure (Contd…)

Exceptions:

- There are some commands that **don't accept any arguments**.
- There are also some commands that may or may not be specified with arguments.

- **For Example:**
  - The ls command can run
    - → without arguments (ls)
    - → with only options (ls –l)
    - → with only filenames (ls f1 f2), or
    - → using a combination of both (ls –l f1 f2).
  - There are some commands compulsorily take options (cut).
  - There are some commands which can take an expression as an argument, or a set of instructions as argument. Ex: grep, sed

Getting Help On UNIX Commands

1. $man <Command name>

   Example: $man who

2. $<Command name> --help

   Example: $who --help

3. $info <Command name>

   Example: $info cat

man: Browsing The Manual Pages

- This command can be used to display manual (documentation) of a specified command.
- Syntax:
  - $ man command
- A pager is a program that
  - → displays one screenful information and
  - → pauses for the user to view the contents.
- The man command is configured to work with a pager.
- Following two commands can be used for navigation:
    - Spacebar or f – moves forward one screen
    - b – moves back one screen
- Finally, to quit the pager, press q. You'll be returned to the shell's prompt ($).

Understanding The man Documentation

- The man documentation is organized in eight (08) sections.
- Example:
  - $ man wc //Help on the wc command User Commands wc(1)

NAME

wc – display a count of lines, words and characters in a file

SYNOPSIS

wc [-c | -m | -C] [-lw] [files....]

DESCRIPTION

The wc utility reads one or more input files and, by default, writes the number of newline characters, words and bytes contained in each input file to the standard output.

OPTIONS

The following options are supported:

-c Count Bytes

-m Count Characters

-l Count Lines

-w Count Words

OPERANDS

The following operands are supported:

File - a path name of an input file. If no file operand are specified, the standard input will be used.

USAGE

See largefiles(5) for the behavious of wc when encountering files >= 2 Gbytes.

EXIT STATUS

0 Successful Completion

> 0 An Error Occurred

SEE ALSO

space(3C), iswalpha(3C), iswspace(3C), setlocale(3C), attributes(5), environ(5), largefile(5)

Understanding the man documentation:

- A man is divided into a number of compulsory and optional sections.
- Every command doesn't need all sections, but the first three (NAME, SYNOPSIS and DESCRIPTION) are generally seen in all man pages.
  - NAME presents a one-line introduction of the command.
  - SYNOPSIS shows the syntax used by the command.
  - DESCRIPTION provides a detailed description.
- The SYNOPSIS follows certain conventions and rules:
  - If a command argument is enclosed in rectangular brackets, then it is optional; otherwise, the argument is required.
  - The ellipsis (a set if three dots) implies that there can be more instances of the preceding word.
  - The | means that only one of the options shows on either side of the pipe can be used.
- All the options used by the command are listed in OPTIONS section.
- There is a separate section named EXIT STATUS which lists possible error conditions and their numeric representation.

General Purpose Utilities:

- cal
- date
- man
- echo
- bc
- clear
- passwd
- who
- whoami
- uname

The cal (Calender) command:

- The cal command is used to invoke the calendar of any specific month, or a complete year.

- Synopsis:

  $cal [[month]year]

- Everything within rectangular brackets is optional, so cal can be used without arguments, in which case it displays the calendar of the current month.

The cal command:

- To see the calendar of a specific month you need to specify two parameters with the cal command.
- **Ex:** To display the calendar of MARCH 2006 we can write

  $ cal 03 2006

- When we print calendar of the entire year it will not fit in the current screen page, it scrolls off too rapidly before you can use [ctrl-s] to pause it.

The cal command (Contd…)

- To make cal put in the same way man pauses , use cal with a pager(more or less) using the
- |(pipeline) symbol to connect them.
  - **Ex:** $ cal 2006 | more
- The | symbol connects two commands (in pipe line) where more takes input from the cal command.

date: Displaying the system date

- Date command displays the current date and time to the nearest second.
- **Synopsis:**

  $ date

- The command can also be used with suitable format specifier as arguments.
- Each format is **preceded by the+** symbol, followed by **the % operator**, and a single character describing the format.

date: Displaying the system date

- For instance, you can print only the month number using the format +%m

$ date +%m

- To print month name

$ date +%h

- If want to combine them in one command.

$ date +"%h %m"

date: Displaying the system date

- There are many other format specifiers:
  - d – The day of the month (1-31)
  - y – The last two digits of the year.
  - H, M and S – The hour, minute and second respectively.
  - D – The date in the format mm/dd/yy.
  - T – The time in the format hh:mm:ss.
- When you use multiple formats then you must enclose within double quotes.

echo: Displaying a message

- echo command is used often in shell scripts to display diagnostic messages on the terminal , or to issue prompts for taking user input.
- Originally echo was an external command but now all shells have echo built in.
- An escape sequence is generally two character string beginning with a \(backslash).
- An escape sequence is used at the end of the string and is used as an argument to echo.

echo: Displaying a message

- Example:
  - $ echo “Enter Filename: \c”
- Like \c there are other escape sequences.

Escape Sequence	Description
\t	Tab space
\n	New Line character

Escape sequence used by echo and printf

Escape Sequence	Description
\c	No new line (cursor in the sameline)
\f	Form feed
\r	Carriage return
\v	Vertical tab
\\	Backslash
\a	Bell

printf: an alternative for echo

- The printf command is available in unix and is an alternative for echo.
- Like echo it is an external command
- **Synopsis:**

  $ printf “No file name entered \n”
- Printf uses formatted strings in the same way as C language uses.
- Here are commonly used formatted string

printf : an alternative for echo

Formatted String	Description
%s	String
%30s	As above but printed in a space 30 character wide
%d	Decimal integer
%6d	As above but printed in a space 6 character wide
%o	Octal Integer
%x	Hexadecimal integer
%f	Floating Point Number

bc: The Calculator

- **bc** command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.

  Synopsys:

  $ bc

  2+2

  output:4

- bc can take multiple inputs each separated by a ;
- Example: $ bc

  12 *12 ; 2 ^3

  - Output:

    144

    8

bc: The Calculator

Examples:

1. Input : $ echo "12+5" | bc

   Output : 17

2.Input : $ echo "10^2" | bc

  Output : 100

3. Input: $ x=`echo "12+5" | bc` $ echo $x

Output:17

bc: The Calculator

- bc can perform only integer computations and truncates the decimal portion that it sees.
- For example :

  9/5 will produce 1 as output.

- To enable floating point computations, you have to set **scale** to the number of digits of precision before you key in the expression.

  Example:

  ```
  $ echo "scale=2;$x/$y" | bc
  ```

- For example:

  Scale=2

  17/7 *truncates to 2 decimal places*

  2.24

- bc has another use and that is converting numbers from one base to another.
- Set ibase(input base) to 2 before you provide the number

bc: The calculator

- Two special variables of bc : **ibase** & **obase**.
- These 2 variables are used to define the base of the numbers.
- ibase is used to specify the base of the input number, and obase for the output result.

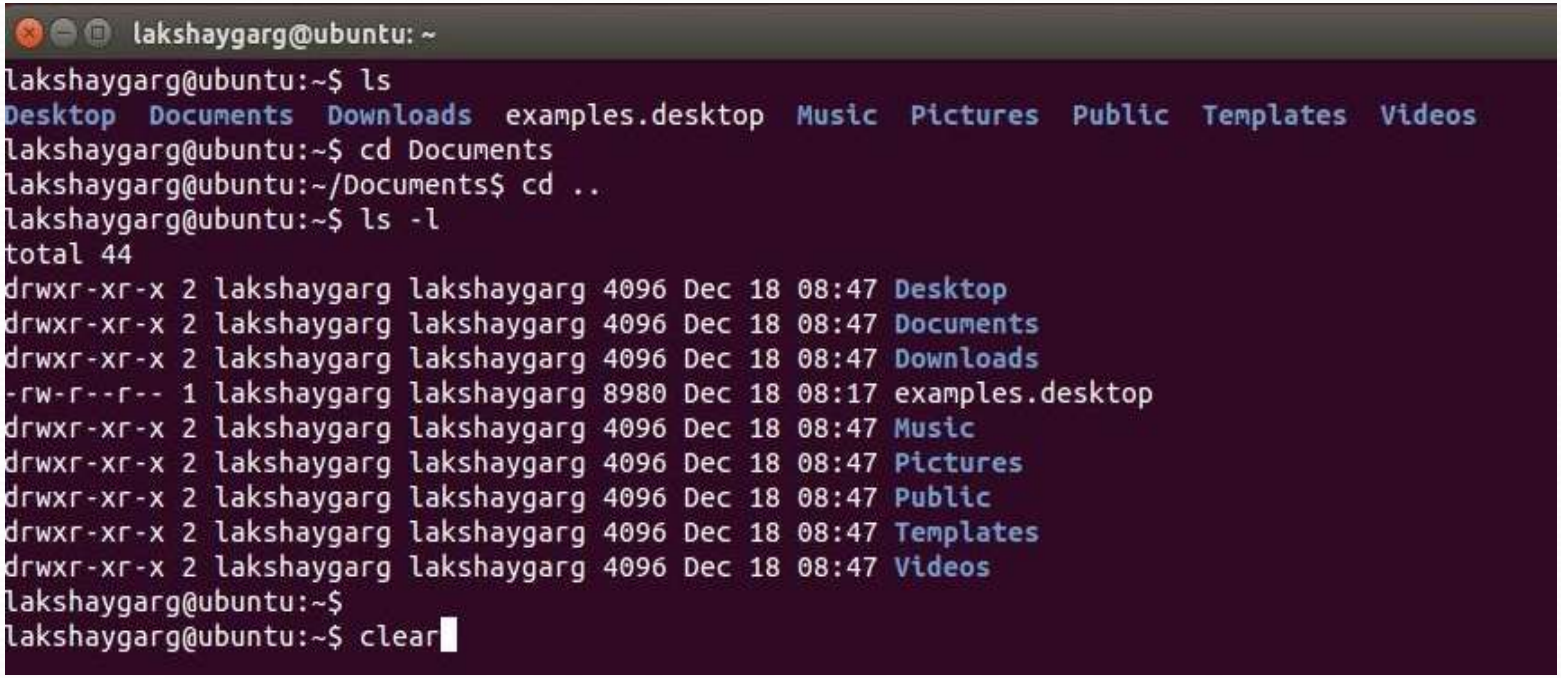
Examples:

1. $echo “obase=16; 80" | bc #**Decimal to Hex Number**
2. $ echo "obase=8;80" | bc #**Decimal to Octal Number**
3. $ echo "obase=2;80" | bc #**Decimal to Binary Number**
4. $ echo "ibase=2;1010000" | bc # convert a binary number to decimal
5. $ echo "obase=16;ibase=2;1010000" | bc #convert a binary number to hex

clear - clear the terminal screen

- *clear* is a standard Unix computer operating system command that is used to clear the terminal screen.
- The *clear* command doesn't take any argument and it is almost similar to *cls* command on a number of other operating systems.
  - $ clear

Terminal Before Executing clear command:



Terminal after executing clear command:

passwd: Changing your password

- To change the password of any unix user use the passwd command.
- **Synopsis:**

  $ passwd
- passwd expects you to respond three times.
  - First it prompts for the old password.
  - Next it checks whether you have entered a valid password, and if you have , it then prompts for the new password.

passwd: Changing your password

- Enter the new password with password naming rules applicable to your system.
- Finally , passwd asks you to re enter the new password.
- If everything goes smoothly , the new password is registered by the system.
- When you enter the password , the string is encrypted by the system.

Password Framing rules and Discipline

- There are some rules that you are expected to follow when handling your own password
  - Don’t choose a password similar to old one.
  - Don’t use commonly used names like names of friends, relatives, pets and so forth. A system may check its own directory and throw out those passwords that are easily guessed.
  - Use a mix of alphabetic and numeric characters. Enterprise unix don’t allow passwords that are wholly alphabetic or numeric
  - Do not write the password in an easily accessible document.
  - Change the password regularly.

who: Who are the users?

- It is more powerful and displays data about all the users who have logged into the system currently.
- **Synopsis:**

  $ who

who: Who are the users?

- The first column shows the usernames working on the system.
- The second column shows the device names of their respective terminals.
- The third , fourth and fifth columns show date and time of logging in.
- The last column shows the machine name from where the user logged in.

who: Who are the users?

- Most unix commands to avoid cluttering the display with header information, this command does have a header option (-H).
- This option prints the column headers, and when combined with –u option, provides a more detailed list.

whoami

•It is basically the concatenation of the strings **“who”,”am”,”i”** as **whoami**.

•It prints the username associated with the current effective userid when this command is invoked.

- **Synopsis:**

  $ whoami

whoami

- **Commands related to whoami command are as follows :**

  1. w — Show who is logged on and what they are doing.
  2. who — Report which users are logged in to the system.

uname: Knowing your machine's characteristics

- It Prints system Information
- **Synopsis:** $ uname

  Ex: $uname

  output: linux # without any option it prints the kernel name

option	Description
-a	Print all information
-r	It prints kernel release details
-m	It prints machine details
-o	It prints operating system details
-n	To know the machine name user

Directory Handling Commands:

- pwd
- cd
- mkdir
- rmdir

Directory handling commands:

- Directory handling commands are used for working with directories in UNIX.
- The typical operations may include:
  - Finding the path of current working directory.
  - Creating a new directory.
  - Changing or moving into the directories.
  - Removing or deleting the directory.

pwd: print working Directory

- The pwd command is a command line utility for printing the current working directory.

- It will print the full system path of the current working directory to standard output.
  - **$pwd**

    /home/cse

mkdir: make directory

mkdir: This command can be used to create a new directory.

Syntax:

- $ mkdir DIRECTORY_NAME

- **1. How to create a directory**
  - $ mkdir mydir
  - $ls

    mydir

  2. How to create multiple directory
  - $ mkdir dir1 dir2 dir3 # creates three directories "dir1", "dir2" and "dir3"
  - $ls
    - dir1 dir2 dir3

mkdir: make directory (contd…)

- **3. How to create parent directory: 2 ways**
  - a) without using option

    $mkdir maindir

    $cd maindir

    $mkdir sub1

    $cd sub1

    $mkdir sub2

    $tree maindir

  - **b)use the –p option**
    - $mkdir –p maindir/sub1/sub2
    - $tree maindir

cd: change directory

- cd (change directory) command is used to change the present working directory to the 'directory' specified.

Syntax:

$ cd <directory name>

$ cd jncs

The above command is used to change the current working directory to "jncs".

rmdir: removing a directory

- rmdir command will delete the empty directories. i.e directory without any sub-directories or files:
- **Syntax:**

$ rmdir <directory name>

Example 1:

$ rmdir jncs #removes the directory named “jncs”

Example-2: To Delete Nested Empty Directories in Linux:

- $ rmdir -p dir1/dir2/dir3

File Handling Utilities

- cat
- touch
- cp
- ls
- rm
- mv
- nl
- pg
- Tar
- wc

File Handling Utilities (Contd…) –touch command

- 1. Creating Files:
  a) touch b)cat

a) touch: By using this command we can create several empty files quickly. It also changes file timestamps**.**

Synopsys: $ touch filename(s)

Example1: $touch sample

Example2: $touch sample1 sample2 sample3 sample4

File Handling Utilities (Contd…) - cat command

b)cat:

- The word cat refers to concatenation.
- We can create a file with some content.
- This command is used in appending the contents of one or more files as described at the command prompt.
  - **Synopsys: $ cat options inputfile(s)**
  - **To create a file:**

```
$cat >test
    Hi
    Hello
```

presss ctrl+d [EOF (or) end of file character]

  - **To display the "test" file contents use:**

```
$cat   test
     Hi
     Hello
```

File Handling Utilities (Contd…)

- uptonow we are seeing the two cases of cat command:

  1. create a new file

  2. Display the contents of an existing file

- **Redirection Operators:**

  -Most Unix system commands take input from your terminal and send the resulting output back to your terminal. A command normally reads its input from the standard input, which happens to be your terminal by default. Similarly, a command normally writes its output to standard output, which is again your terminal by default.

  -Redirection is nothing but the file output is located to the destination file.

Redirection operators	Description
<	It refers to redirecting the standard input
>	It refers to redirecting the standard output
>>	It refers to appending output redirection operator

File Handling Utilities (Contd…) - cat command

- cat command concatenates the contents of two files and store them in the third file:
  - **$cat sample1 sample2 sample3 > newsample**

- Append output redirection operator(>>):

$cat sample1 sample2 sample3 >> newsample

Options:

Option	Description
-e	printing the dollar symbol ($) at the end of each line.
-n	Displaying the numbering in output
-v	To display non-printable characters

File Handling Utilities (Contd…) - cp command

- **cp command:**
- 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.

Synopsys:

- **$cp [OPTION] Source Destination**
- **cp [OPTION] Source Directory**
- **cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory**

Examples:

copy the contents of 1st file to the 2nd file

```
$ ls
a.txt

$ cp a.txt b.txt

$ ls
a.txt  b.txt
```

copy the contents of 1st file and 2nd file to the directory

```
$ ls
a.txt  b.txt  new

Initially new is empty
$ ls new

$ cp a.txt b.txt new

$ ls new
a.txt  b.txt
```

File Handling Utilities (Contd…) - cp command

- **cp command options:**

Option	Description
-i	Interactive copying. Ex: Overwrite (Y/N)
-r (or)-R	Recursively Copying directory structures
-b	creates the backup of the destination file in the same folder with the different name and in different format.
-f	Copy the files and directory forcefully

Using –i option

```
$ cp -i a.txt b.txt
cp: overwrite 'b.txt'? y

$ cat b.txt
GFG
```

Using –b option

```
$ ls
a.txt  b.txt

$ cp -b a.txt b.txt

$ ls
a.txt  b.txt  b.txt~
```

File Handling Utilities (Contd…) - cp command

Using –f option

```
$ ls -l b.txt
-r-xr-xr-x+ 1 User User 3 Nov 24 08:45 b.txt

User, group and others doesn't have writing permission.

Without -f option, command not executed
$ cp a.txt b.txt
cp: cannot create regular file 'b.txt': Permission denied

With -f option, command executed successfully
$ cp -f a.txt b.txt
```

Using –r option

```
$ ls geeksforgeeks/
a.txt  b.txt  b.txt~  Folder1  Folder2

Without -r option, error
$ cp geeksforgeeks gfg
cp: -r not specified; omitting directory 'geeksforgeeks'

With -r, execute successfully
$ cp -r geeksforgeeks gfg

$ ls gfg/
a.txt  b.txt  b.txt~  Folder1  Folder2
```

File Handling Utilities (Contd…) - mv command

- **mv command:**

mv stands for **move**. 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. tem like UNIX.

No additional space is consumed on a disk during renaming. This command normally **works silently** means no prompt for confirmation.

Synopsys:

- **$mv [OPTION] Source Destination**
- **$mv [OPTION] Source Directory**
- **$mv [OPTION] Directory Source**

Example 1:

1. $mv sample sample1

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. By default, **mv** doesn't prompt for overwriting the existing file, So be careful

File Handling Utilities (Contd…) - mv command

- **mv command options**

Option	Description
-i	Interactive confirmation of overwrites. Ex: Overwrite (Y/N)
-r (or)-R	Recursively moving directory structures
-b	backup of an existing file that will be overwritten as a result of the **mv** command. This will create a backup file with the tilde character(~) appended to it.
-f	moving the files and directory forcefully

File Handling Utilities (Contd…) - rm command

- **rm** - remove files or directories. By default, it does not remove directories.

Synopsys:

- **$rm [OPTION] filename(s)**

Example: Let us consider 5 files having name **a.txt, b.txt** and so on till **e.txt**.

```
$ ls
a.txt  b.txt  c.txt  d.txt  e.txt
```

```
Removing one file at a time
$ rm a.txt

$ ls
b.txt  c.txt  d.txt  e.txt

Removing more than one file at a time
$ rm b.txt c.txt

$ ls
d.txt  e.txt
```

File Handling Utilities (Contd…) - rm command

Examples:

1.$rm xyz

2.$rm f1 f2 f3 f4 f5

3. $rm a*.c

4. $rm a?.c

5. $rm a[0-9]*.c

6. $rm a[!A-Z a-z 0-9]*.c

Option	Description
-i	Interactive deletion. Ex: Overwrite (Y/N)
-r (or)-R	Recursive Deletion. delete all the files and sub-directories recursively of the parent directory. At each stage it deletes everything it finds. Normally, rm wouldn't delete the directories but when used with this option, it will delete.
-f	Force deletion. **rm** prompts for confirmation removal if a file is **write protected.** The **-f** option overrides this minor protection and removes the file forcefully.

The ls command:

- The ls command is used to get a list of files and directories.
- Options can be used to get additional information about the files.

- **Syntax:**

  $ls [options] [path(s)/filename(s)]

Options of ls Command:

- **The ls command supports the following options:**
  - ls -a: list all files including hidden files. These are files that start with ".".
  - ls -A: list all files including hidden files except for "." and ".." – these refer to the entries for the current directory, and for the parent directory.
  - ls -R: list all files recursively, descending down the directory tree from the given path.
  - ls -l: list the files in long format i.e. with an index number, owner name, group name, size, and permissions.
  - ls – o: list the files in long format but without the group name.
  - ls -g: list the files in long format but without the owner name.
  - ls -i: list the files along with their index number.
  - ls -s: list the files along with their size.
  - ls -t: sort the list by time of modification, with the newest at the top.
  - ls -S: sort the list by size, with the largest at the top.
  - ls -r: reverse the sorting order.

The nl (numbering lines) command:

- The nl command is a command line text formatting utility in UNIX.
- It's main purpose is to display line numbers of a file or standard input.
- **Syntax:**

  $ nl [Options] filename

- **Using Styles (-b)**
  - There are three output styles to choose from:
  - a = Number all lines
  - t = Number only nonempty lines
  - n = Number no Lines
  - p = Number only lines that contain a match for the basic regular expression

  By default the nl command uses the (t) style and numbers only nonempty lines.

The nl (numbering lines) command:

- **Changing the Starting Line Number**
  - You can change the starting line number by using the -v option followed by the number you wish to start with.
- **Example:**

  $ nl -v 77 file1.txt

  the above command numbers the contents of file1.txt starting from 77.

-i option, line number increment at each line

Example:

$nl -i2 file1.txt //increments the line number by 2.

The pg (page) command: browse page wise through text files

- **pg** displays a text file on a CRT one screenful at once. After each page, a prompt is displayed.
- The user may then either press the newline key to view the next page or one of the keys described below.
- If no filename is given on the command line, **pg** reads from standard input.
- **Syntax:**

  pg [options] [+/Pattern/] [file_name]

Options of pg command:

-c	Moves the cursor to the home position and clears the screen before each page. This flag is ignored if the clear_screen field is not defined for your workstation type in the terminfo file.
-e	Does not pause at the end of each file.
-f	Does not split lines. Normally, the pg command splits lines longer than the screen width.
-n	Stops processing when a pg command letter is entered. Normally, commands must end with a new-line character.
-p *String*	Uses the specified string as the prompt. If the *String* contains a %d value, that value is replaced by the current page number in the prompt. The default prompt is : (colon). If the specified string contains spaces, you must enclose the string in quotation marks.
-r	Prevents shell escape when the "!" subcommand is used.
-s	Highlights all messages and prompts.
+*LineNumber*	Starts at the specified line number.
-*Number*	Specifies the number of lines in the window. On workstations that contain 24 lines, the default is 23.
+/*Pattern*/	Starts at the first line that contains the specified pattern.

Sub-commands of pg Command:

Page	Displays the page specified by the *Page* parameter.
+*Number*	Displays the page obtained by adding the *Number* value to the current page.
-*Number*	Displays the page as specified by the *Number* value before the current page.
l	(Lowercase L) Scrolls the display one line forward.
*Number*l	Displays at the top of the screen the line specified by the *Number* parameter.
+*Number*l	Scrolls the display forward for the specified number of lines.
-*Number*l	Scrolls the display backward for the specified number of lines.
d	Scrolls half a screen forward. Pressing the Ctrl-D key sequence functions the same as the d subcommand.
-d	Scrolls half a screen backward. Pressing the -Ctrl-D key sequence functions the same as the -dsubcommand.
Ctrl-L	Displays the current page again. A single . (dot) functions the same as the Ctrl-L key sequence subcommand.
$	Displays the last page in the file. Do not use this when the input is from a pipeline.

The tar (tape archive) Command:

- "**tar**" stands for tape archive, which is used by large number of **Linux/Unix** system administrators to deal with tape drives backup.
- The tar command used to rip a collection of files and directories into highly compressed archive file commonly called **tarball.**
- The tar is most widely used command to create compressed archive files and that can be moved easily from one disk to another disk or machine to machine.

Create tar Archive File

- The below example command will create a **tar** archive file **sample.tar** for a directory **/home/jncs** in current working directory.

  # tar -cvf sample.tar /home/jncs/

- **c** – Creates a new .tar archive file.
- **v** – Verbosely show the .tar file progress.
- **f** – File name type of the archive file.

The wc (word count) command:

- This command can be used to get a count of the total number of lines, words, and characters contained in a file.
  - Syntax:

    $wc FILENAME
- **Case 1:**
  - **Example:**

    $ cat P1.c

    WELCOME // contents of P1.c

    TO

    UNIX

    $ wc P1.c
  - LINE WORD CHARATCTER FILENAME

    3 3 15 P1.c

The wc (word count) command:

- The header includes the following attributes:
  - **LINE**

  This represents the total number of lines in the file.
  - **WORD**

  This represents the total number of words in the file (excluding space, tab and newline).
  - **CHARATCTER**

  This represents the total number of characters in the file (including space, tab and newline).
  - **FILENAME**

  This represents the name of the file.

The wc command:

- **Case 2:**
- This command can also accept more than one filename as arguments.
- Example:

```
$ wc p1.txt p2.txt
```

The above command displays the output as:

```
3    3    15    p1.txt
4    5    20    p2.txt
```

Displaying Commands:

- more
- head
- tail

The more command:

- **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.
- Another application of more is to use it with some other command after a pipe (|).
- When the output is large, we can use more command to see output one by one.

The more command:

- **Syntax:**

  $more [-options] [-num] [+/pattern] [+linenum] [file_name]

  - [-**options**]: any option that you want to use in order to change the way the file is displayed. Choose any one from the followings: (-d, -l, -f, -p, -c, -s, -u)
  - [-**num**]: type the number of lines that you want to display per screen.
  - [+/**pattern**]: replace the pattern with any string that you want to find in the text file.
  - [+**linenum**]: use the line number from where you want to start displaying the text content.
  - [**file_name**]: name of the file containing the text that you want to display on the screen.

The more command:

- ***While viewing the text file use these controls:***

  Enter key: to scroll down line by line.
  Space bar: To go to the next page.
  b key: To go to back one page.

- **Options:**
  - **-d :** Use this command in order to help the user to navigate.
  - It displays "[Press space to continue, 'q' to quit.]" and displays "[Press 'h' for instructions.]" when wrong key is pressed.
- **Example:**

  $more -d sample.txt

The more command: options

- **-f :** This option does not wrap the long lines and displays them as such.
- **Example:**

  $more -f sample.txt
- **-p :** This option clears the screen and then displays the text.
- **Example:**

  $more -p sample.txt
- **-c :** This option is used to display the pages on the same area by overlapping the previously displayed text.
- **Example:**

  $more -c sample.txt

The more command: options

- **-s :** This option squeezes multiple blank lines into one single blank line.
- **Example:**

  $more -s sample.txt
- **-u :** This option omits the underlines.
- **Example:**

  $more -u sample.txt
- **+/pattern :** This option is used to search the string inside your text document. You can view all the instances by navigating through the result.
- **Example:**

  $more +/reset sample.txt

The more command: options

- **+num :** This option displays the text after the specified number of lines of the document.
- **Example:**

  $more +30 sample.txt
- **Using more to Read Long Outputs:** We use more command after a pipe to see long outputs.
- For example, seeing log files, etc.

  $cat a.txt | more

The head command:

- This command is used to display the top of the file.
- By default, it displays the first 10 lines of the file.
- It is mainly useful to verify the contents of a file.
- **Syntax:**

  $head [count option] filename
- **Example:**

  $head state.txt # displays first 10 lines of state.txt
- **Displaying first n lines of a file (-n)**
- -n option is used to specify a line count and to display the first n lines of the file.
- **Example:**

  $ head -n 3 state.txt // displays first 3 lines of state.txt

The head command: Options

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

The head command: options

1. **-n num:** Prints the first 'num' lines instead of first 10 lines. **num** is mandatory to be specified in command otherwise it displays an error.

   $ head -n 5 state.txt

2. **-c num:** Prints the first 'num' bytes from the file specified. Newline count as a single character, so if head prints out a newline, it will count it as a byte. **num** is mandatory to be specified in command otherwise displays an error.

   $ head -c 6 state.txt

The head command : options

- **3. -q:** It is used if more than 1 file is given. Because of this command, data from each file is not precedes by its file name.

  $ head -q state.txt capital.txt

4. -v: By using this option, data from the specified file is always preceded by its file name.

$ head -v state.txt

The tail 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 [OPTION]... [FILE]...

The tail command: options

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

The tail command: options

- 1. **-n num:** Prints the last 'num' lines instead of last 10 lines. **num** is mandatory to be specified in command otherwise it displays an error.
- This command can also be written as without symbolizing 'n' character but '-' sign is mandatory.

$ tail -n 3 state.txt (or)

$ tail -3 state.txt

- Tail command also comes with an '+' option which is not present in the head command.
- With this option tail command prints the data starting from specified line number of the file instead of end.
- For command: **tail +n file_name**, data will start printing from line number 'n' till the end of the file specified.

The tail command: options

- 2. **-c num:** Prints the last 'num' bytes from the file specified.
  - Newline count as a single character, so if tail prints out a newline, it will count it as a byte.
  - In this option it is mandatory to write **-c** followed by positive or negative **num** depends upon the requirement.
  - By **+num**, it display all the data after skipping **num** bytes from starting of the specified file and by **-num**, it display the last **num** bytes from the file specified.
  - **With negative num** :

    $ tail -c -6 state.txt (or) $ tail -c 6 state.txt

    With positive num :

    $ tail -c +6 state.txt

The tail command: options

- 3. **-q:** It is used if more than 1 file is given. Because of this command, data from each file is not precedes by its file name.

  $ tail -q state.txt capital.txt

- **4. -f:** This option is mainly used by system administration to monitor the growth of the log files written by many Unix program as they are running.

  This option shows the last ten lines of a file and will update when new lines are added.

  As new lines are written to the log, the console will update with the new lines.

  $ tail -f logfile

The tail command: options

5. **-v:** By using this option, data from the specified file is always preceded by its file name.

   $ tail -v state.txt

6. **–version:** This option is used to display the version of tail which is currently running on your system.

   $ tail --version

Simple filters and Commands:

- Filters are programs that take plain text(either stored in a file or produced by another program) as standard input, transforms it into a meaningful format, and then returns it as standard output.
- UNIX has a number of filters.
- Examples for filtering commands are:
  - cat
  - head
  - tail
  - cmp
  - comm
  - diff
  - sort
  - wc
  - grep
  - cut
  - uniq
  - nl
  - sed

The cmp (compare) command:

- **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.
  - When cmp is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found *i.e* the files compared are identical.
  - cmp displays no message and simply returns the prompt if the the files compared are identical.
- **Syntax:** $cmp options… FromFile [ToFile]

The cmp command: Example

- **$cmp file1.txt file2.txt If the files are not identical :**
  - the output of the above command will be :

    $cmp file1.txt file2.txt

    file1.txt file2.txt differ: byte 9, line 2

    /*indicating that the first mismatch found in two files at byte 20 in second line*/
- **If the files are identical :**
  - you will see something like this on your screen:

    $cmp file1.txt file2.txt

    $ _

    /*indicating that the files are identical*/

The cmp command: options

- **1. -b(print-bytes) :** If you want cmp displays the differing bytes in the output when used with -**b** option.
  - **$cmp -b file1.txt file2.txt**

    file1.txt file2.txt differ: 12 byte, line 2 is 154 l 151 i

2. -i [bytes-to-be-skipped] : Now, this option when used with cmp command helps to **skip a particular number of initial bytes from both the files** and then after skipping it compares the files.

$cmp -i 10 file1.txt file2.txt

$_

/*indicating that both files are identical after 10 bytes skipped from both the files*/

The cmp command: options

- **3. -i [bytes to be skipped from first file] : [bytes to be skipped from second file] :**

  This option is very much similar to the above -i [bytes to be skipped] option but with the difference that now it **allows us to input the number of bytes we want to skip** from both the files separately.

  $cmp -i 10:12 file1.txt file2.txt

  $_

  /*indicating that both files are identical after 10 bytes skipped from first file and 12 bytes skipped from second file*/

The cmp command: options

- **4. -l option :** This option makes the cmp command print byte position and byte value for all differing bytes.

  $cmp -l file1.txt file2.txt

5. -s option : This allows you to suppress the output normally produced by cmp command *i.e* it compares two files without writing any messages.

This gives an exit value of 0 if the files are identical, a value of 1 if different, or a value of 2 if an error message occurs.

$cmp -s file1.txt file.txt

The cmp command: options

- **6.** -n [number of bytes to be compared] option :

  This option allows you to limit the number of bytes you want to compare ,like if there is only need to compare at most 25 or 50 bytes.

  $cmp -n 50 file1.txt file2.txt

  $_

  /*indicating files are identical for starting 50 bytes*/

The comm (common) command:

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

  $comm [OPTION]... FILE1 FILE2

  $comm file1.txt file2.txt

The comm command: options

- **Options for comm command:**

  1. -1 :suppress first column(lines unique to first file).
  2. -2 :suppress second column(lines unique to second file).
  3. -3 :suppress third column(lines common to both files).
  4. – -check-order :check that the input is correctly sorted, even if all input lines are pairable.
  5. – -nocheck-order :do not check that the input is correctly sorted.
  6. – **-help :**display a help message, and exit.
  7. – **-version :**output version information, and exit.

The diff (difference) command:

- Diff command in Linux is used to compare the content of two files line by line and if the difference is found then it will also list differences along with line numbers.
  - **Syntax:**

    $ diff <options> file1 file2
- Output of diff command can be in following format:
  - Normal (default)
  - Context
  - Unified
- Symbols used in diff command output are:
  - a -> it indicates that something has been added
  - c -> it indicates that some text has been changed
  - d -> it indicates some text has been deleted

The diff command: Options

- By default, diff command output is in normal format, it means when contents of two files are identical then it will not produce any output but we will get the prompt.
- UNIX system offers two different ways to view the **diff** command output :
  - **context mode**
  - **unified mode**.

1. **-c (context) :** To view differences in context mode, use the **-c** option.

   $ diff -c file1.txt file2.txt

The diff command: Options

- **-u (unified) :** To view differences in unified mode, use the **-u** option. It is similar to context mode but it **doesn't display any redundant information** or it shows the information in concise form.

  $ diff -u file1.txt file2.txt

- **-i :** By default this command is *case sensitive*. To make this command *case in-sensitive* use **-i** option with **diff**.

  $ diff -i file1.txt file2.txt

- **–version :** This option is used to display the version of **diff** which is currently running on your system.$ diff --version

The cut command:

- The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output.
- It can be used to cut parts of a line by **byte position, character and field**.
- Basically the cut command slices a line and extracts the text.
- **Syntax:**

  $cut OPTION... [FILE]...

It is necessary to specify option with command otherwise it gives error.

The cut command: options

- **1. -b(byte):** To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma.
- Range of bytes can also be specified using the hyphen(-).
- **List without ranges**

  $ cut -b 1,2,3 state.txt

- **List with ranges**

  $ cut -b 1-3,5-7 state.txt

The cut command: options

- **2. -c (column):** To cut by character use the -c option. This selects the characters given to the -c option.
- This can be a list of numbers separated comma or a range of numbers separated by hyphen(-).

```
$ cut -c 2,5,7 state.txt
$ cut -c 1-7 state.txt
```

The paste command:

- Paste command is one of the useful commands in Unix or Linux operating system.
- It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by **tab** as delimiter, to the standard output.
- When no file is specified, or put dash ("-") instead of file name, paste reads from standard input and gives output as it is until a interrupt command [**Ctrl-c**] is given.
- **Syntax:**

  $paste [OPTION]... [FILES]...

The paste command: options

- **1. -d (delimiter):** Paste command uses the tab delimiter by default for merging the files.
- The delimiter can be changed to any other character by using the **-d** option.
- If more than one character is specified as delimiter then paste uses it in a circular fashion for each file line separation.
- **Only one character is specified**

  $ paste -d "|" number state capital

  More than one character is specified

  $ paste -d "|," number state capital

The paste command: options

- **2. -s (serial):** We can merge the files in sequentially manner using the -s option.
- It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab.
- And these single lines are separated by newline.

```
$ paste -s number state capital
```

The sort command:

- 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.
  - sort is a standard command line program that prints the lines of its input or concatenation of all files listed in its argument list in sorted order.
  - The sort command is a command line utility for sorting lines of text files. It supports sorting alphabetically, in reverse order, by number, by month and can also remove duplicates.
  - The sort command can also sort by items not at the beginning of the line, ignore case sensitivity and return whether a file is sorted or not. Sorting is done based on one or more sort keys extracted from each line of input.
  - By default, the entire input is taken as sort key. Blank space is the default field separator.

The sort command:

- **The sort command follows these features as stated below:**
  - Lines starting with a number will appear before lines starting with a letter.
  - Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
  - Lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.
  - **Syntax :**

    $ sort filename.txt

The sort command: options

- **-o Option :** Unix also provides us with special facilities like if you want to write the **output to a new file**, output.txt, redirects the output like this or you can also use the built-in sort option -o, which allows you to specify an output file.
  Using the -o option is functionally the same as redirecting the output to a file.
- **Syntax :**

  $ sort inputfile.txt > filename.txt

  $ sort -o filename.txt inputfile.txt

The sort command: options

- **-r Option: Sorting In Reverse Order** : You can perform a reverse-order sort using the -r flag. the -r flag is an option of the sort command which sorts the input file in reverse order i.e. descending order by default.
  - **Syntax:**

    $ sort -r inputfile.txt

The sort command: options

- **-n Option** : To sort a file **numerically** use –n option.
- This option is used to sort the file with numeric data present inside.
- **Syntax :**

  $ sort -n filename.txt

  -nr option : To sort a file with **numeric data in reverse order** we can use the combination of two options.

  $ sort -nr filename.txt

The uniq command:

- The **uniq** command in Linux is a command line utility that reports or filters out the repeated lines in a file.
- In simple words, **uniq** is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines.
- **uniq** filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file .
- **Syntax:**

  $uniq [OPTION] [INPUT[OUTPUT]]

The uniq command: options

- **c – -count :** It tells how many times a line was repeated by displaying a number as a prefix with the line.
- **-d – -repeated :** It only prints the repeated lines and not the lines which aren't repeated.
- **-f N – -skip-fields(N) :** It allows you to skip N fields(a field is a group of characters, delimited by whitespace) of a line before determining uniqueness of a line.
- **-i – -ignore case :** By default, comparisons done are case sensitive but with this option case insensitive comparisons can be made.
- **-s N – -skip-chars(N) :** It doesn't compares the first N characters of each line while determining uniqueness. This is like the -f option, but it skips individual characters rather than fields.
- **-u – -unique :** It allows you to print only unique lines.
- **-w N – -check-chars(N) :** It only compares N characters in a line.

The tr (translate) command:

- The tr command in UNIX is a command line utility for translating or deleting characters.
- It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace.
- **Syntax :**

  $ tr [OPTION] SET1 [SET2]

The tr command: options

- -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

The tr command: Examples

- **How to convert lower case to upper case:**

  $cat filename | tr “[a-z]” “[A-Z]”

  $cat filename | tr “[:lower:]” “[:upper:]”

- **How to translate white-space to tabs**

  $ echo "Welcome To UNIX" | tr [:space:] '\t‘

find command

- This command is used to locate files in the unix directory tree.
- It searches the named dir's and its subdir's for files.
- **Most frequently called like this;**

  $ find ./ -name "t*" -print

- **Examples:**

  1. $ find . -name sample -print
  2. $ find . -name [ab]* -print

- **To search Dir's called backup from /usr dir :**

  $find /usr –type d –name backup -print

find command contd…

- To search normal files called backup from /usr dir :

```
$find   /usr   –type   f   –name   backup  -print
$find   /usr   –type   c   –name   backup  -print
$find   /usr   –type   b   –name   backup  -print
```

- To search root dir downwards all files which have exactly 2 links:
  - $find / -links 2 -print
  - $find / -links -2 -print
  - $find / -links +2 -print

finger command:

- **finger** command is a **user information lookup** command which gives details of all the users logged in.
- This tool is generally **used by system administrators**.
- It provides details like login name, user name, idle time, login time, and in some cases their email address even.
- **Examples:**
  - **To finger or get details of a user.**

    $finger cse
  - **To get idle status and login details of a user.**

    $finger -s cse

Disk Utilities: du, df, mount, umount

- A **disk utility** is a utility program that allows a user to perform various functions on a computer disk, such as:
  - disk partitioning and logical volume management
  - changing drive letters and other mount points
  - renaming volumes
  - disk checking
  - disk formatting
- which are otherwise handled separately by multiple other built-in commands.

du (disk usage) command:

- This command displays disk usage (usually in multiplies of 1k blocks).
- Du command without any argument displays disk usage of all files, sub dir's of current working dir.

- **Syntax :**

  $du [OPTION]... [FILE]...

Example:

$du dirname

$du filename

The du command: options

- ***-0, –null :*** *end each output line with NULL*
- ***-a, –all :*** *write count of all files, not just directories*
- ***-B, –block-size=SIZE :*** *scale sizes to SIZE before printing on console*
- ***-c, –total :*** *produce grand total*
- ***-d, –max-depth=N :*** *print total for directory only if it is N or fewer levels below command line argument*
- ***-h, –human-readable :*** *print sizes in human readable format*
- ***-S, -separate-dirs :*** *for directories, don't include size of subdirectories*
- ***-s, –summarize :*** *display only total for each directory*
- ***–time :*** *show time of last modification of any file or directory.*
- ***–exclude=PATTERN :*** *exclude files that match PATTERN*

df (disk free)command:

- The **df** command (short for disk free), is used to display information related to file systems about total space and available space.

- **Syntax :**

  $df [OPTION]... [FILE]...

- If no file name is given, it displays the space available on all currently mounted file systems.

The df command: options

- ***-a, –all :*** *includes pseudo, duplicate and inaccessible file systems.*
- ***-B, –block-size=SIZE :*** *scales sizes by SIZE before printing them.*
- ***-h, –human-readable :*** *print sizes in power of 1024*
- ***-H, –si:*** *print sizes in power of 1000*
- ***-i, –inodes :*** *list inode information instead of block usage*
- ***-l, –local :*** *limit listing to local file systems*
- ***–total :*** *elide all entries insignificant to available space, and produce grand total*
- ***-t, –type=TYPE :*** *limit listing to file systems of type TYPE*
- ***-T, –print-type :*** *print file system type*

The mount and umount commands:

- All files in a **UNIX/Linux** filesystem are arranged in form of a big tree rooted at '/'.
- These files can be spread out on various devices based on your partition table, initially your parent directory is mounted(i.e attached) to this tree at '/', others can be mounted manually using GUI interface(if available) or using **mount** command.
- **mount** command is used to mount the filesystem found on a device to big tree structure(**Linux** filesystem) rooted at '/'.
- Conversely, another command **umount** can be used to detach these devices from the Tree.

The mount and umount commands:

- **Syntax:**

  $mount -t type device dir

- **Some Important Options:**
  - **l** : Lists all the file systems mounted yet.
  - **h** : Displays options for command.
  - **V** : Displays the version information.
  - **a** : Mounts all devices described at **/etc/fstab**.
  - **t** : Type of filesystem device uses.
  - **T** : Describes an alternative fstab file.
  - **r** : Read-only mode mounted.
- **Syntax for umount:**

  $umount devicedir

Process utilities: ps, kill

- The process utilities in UNIX are used for performing process management tasks like:
  - Creating a process.
  - Executing the process.
  - Knowing the status of the process.
  - Terminating or killing the process.

ps (process status) command:

- ps command is used to list the currently running processes and their PIDs along with some other information depends on different options.
- **Syntax:**

  $ps [options]

- **Simple process selection :** Shows the processes for the current shell –

```
$ ps
PID          TTY          TIME         CMD
12330        pts/0        00:00:00     bash
21621        pts/0        00:00:00     ps
```

The ps command

- Result contains four columns of information.
  Where,
  PID – the unique process ID
  TTY – terminal type that the user is logged into
  TIME – amount of CPU in minutes and seconds that the process has been running
  CMD – name of the command that launched the process.
- **View Processes :** View all the running processes use either of the following option with ps –

  $ps –A

  $ ps -e

The ps command: options

- **View all processes associated with this terminal :**

  $ps -T

- **View all the running processes:**

  $ps -r

- **View all processes owned by you :**

  Processes i.e same EUID as ps which means runner of the ps command

  $ps -x

The kill command:

- *kill* command in UNIX/Linux, is a built-in command which is used to **terminate processes manually**.
- *kill* command sends a signal to a process which terminates the process.
- To kill a particular process, we use the pid of that process as:

  $kill <pid of the process to be terminated/killed>

Networking utilities: ping,telnet,rlogin,ftp

- Networking is an essential part of Unix and it offers lots of tools and commands to diagnose any networking problem.

- These utilities enables you to quickly troubleshoot connection issues.

- **Example:**
  - whether another system is connected or not
  - whether another host is responding or not

The ping command:

- PING (Packet Internet Groper) command is used to check the network connectivity between host and server/host.
- This command takes as input the IP address or the URL and sends a data packet to the specified address with the message “PING” and get a response from the server/host this time is recorded which is called latency.
- Fast ping low latency means faster connection.
- Ping uses **ICMP(Internet Control Message Protocol)** to send an **ICMP echo message** to the specified host if that host is available then it sends **ICMP reply message**.
- Ping is generally measured in millisecond every modern operating system has this ping pre-installed.
- **The syntax for ping command is:**

  $ping ipaddress (or)url

The telnet command:

- The **telnet** program is a user interface to the TELNET protocol.
- The **telnet** command is used for interactive communication with another host using the TELNET protocol. It begins in command mode, where it prints a telnet command prompt ("**telnet>**").
- If **telnet** is invoked with a *host* argument, it performs an **open** command implicitly.
- **Syntax**

  $telnet [-468ELadr] [-S *tos*] [-b *address*] [-e *escapechar*] [-l *user*] [-n *tracefile*] [*host* [*port*]]

The telnet command: options

- **-4**Force IPv4 address resolution.
- **-6**Force IPv6 address resolution.
- **-8**Request 8-bit operation.
- **-E**Disables the escape character functionality.
- **-L**Specifies an 8-bit data path on output.
- **-a**Attempt automatic login.
- **-b** *address*Use bind on the local socket to bind it to a specific local address.
- **-r**Emulate **rlogin**.
- **-S** *tos*Sets the IP TOS (type-of-service) option for the **telnet** connection to the value *tos*.
- ***host***Specifies a host to contact over the network.
- ***port***Specifies a port number or service name to contact. If not specified, the telnet port (**23**) is used.

The rlogin (remote login) command:

- The "*rlogin*" command is similar to telnet. The rlogin starts a terminal session on a remote host.
- Syntax :

  $rlogin hostname
- Example :

  $ rlogin jncs

  Password:

  Last login: Mon Jul 13 11:41:38 on pts/1

The ftp command:

- **ftp** is the user interface to the Internet standard File Transfer Protocol.
- The program allows a user to transfer files to and from a remote network site.
- **Syntax:**

  $ftp [-pinegvd] [host]

The ftp command: options

- **-A** Use active mode for data transfers.
- **-p** Use passive mode for data transfers.
- **-i** Turns off interactive prompting during multiple file transfers.
- *-4* Use only IPv4 to contact any host.
- *-6* Use IPv6 only.
- *-e* Disables command editing and history support.