Kernel Tasks

Processes

Create/destroy
Schedule
IPC
Input/ Output

Memory

Virtual memory allocation/ free space management

File systems

Virtual file systems/native file systems.
Maintaining inodes as well as data in a file system.

Devices

Device drivers to control all peripherals

Networking

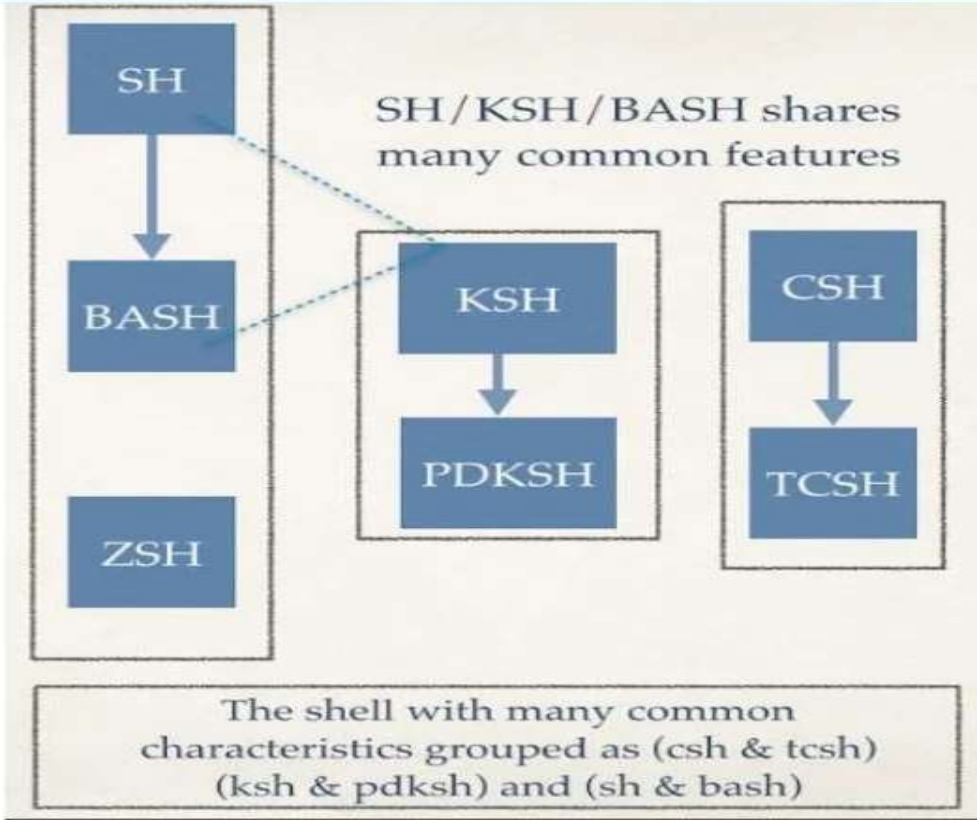
Receive packets – identify and dispatch Sending packets – Routing and Address Resolution

Shell Prompt

- Shell provides an interface to the user and UNIX OS.
- A shell is an environment in which we can run our commands, programs, and shell scripts.
- It gathers input from user and executes programs based on that input.

Different shells

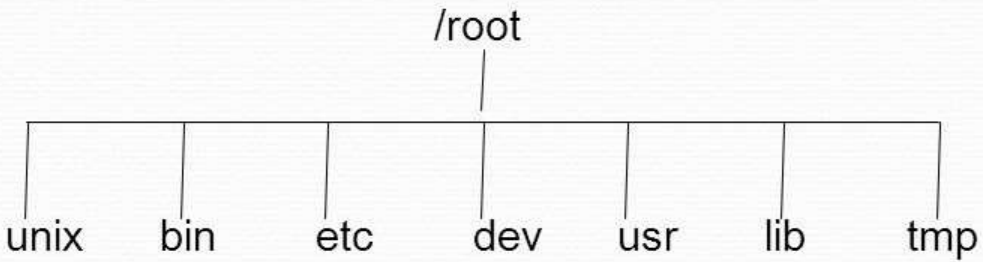
- Different flavor of shells, each has own set of recognized commands and functions
  - Borne shell(sh)
  - Korn shell(sh)
  - C shell
  - TENEX C shell(tcsh)
- Shells are usually installed in /bin/



Super User

- Unix is a case-sensitive operating system.
- A special account root that is reserved for the system administrator or super user.
- It has all system resources, including all files, directories, and data.
- The super user has full access and control over every thing and a slight mistake can cause irreparable damage to the system.

UNIX Hierarchical Structure



- Unix File system begins with the directory called the root. The root directory is denoted by /
- Branching from the root there are several other directories called bin,etc,usr,lib etc.. And ordinary file named UNIX which is the kernel program itself.
- This is very first file which is loaded into memory during system startup.
- The root directory is called the parent directory and others are child or subdirectories of the root.

(1) bin :-

- It contains the executable files for most unix commands

Eg. Cat, bash, date, chown, chgrp, chmod, ping etc.

- Unix commands are executable C programs or shell scripts devoted to carrying out single task at the user desk.
- Shell scripts are merely a collection of valid unix commands. Therefore the bin directory assumes a great amount of importance in a unix/linux installation.

(2) etc :-

- It contains other additional commands which are related to system maintenance and administration.
- It also contains several system files which store the relevant information about the users of the system and the terminals and devices connected to the system.

  Eg. apt, emacs, init, openoffice, Configuration files

(3) lib :-

- It contains all the libraries provided by Unix for programmers. When programs written under Unix need to make system call they can use libraries provided in this directory.

(4) dev :-

- In the dev directory it stores files that control character and block devices.
- Unix has a file for each device that it has to deal with. In fact all the information that is displayed on the terminal of each user comes from a terminal file in this directory.

(5) usr :-

- In the usr directory the user work area is provided. It is general practice to create each users home directory within this directory. So that users can store data and other files within them.

- The user is free to organize his home directory by creating other sub directories in it to contain functionally related files
- Within the usr directory there is another directory **bin** in which traditional UNIX commands are stored

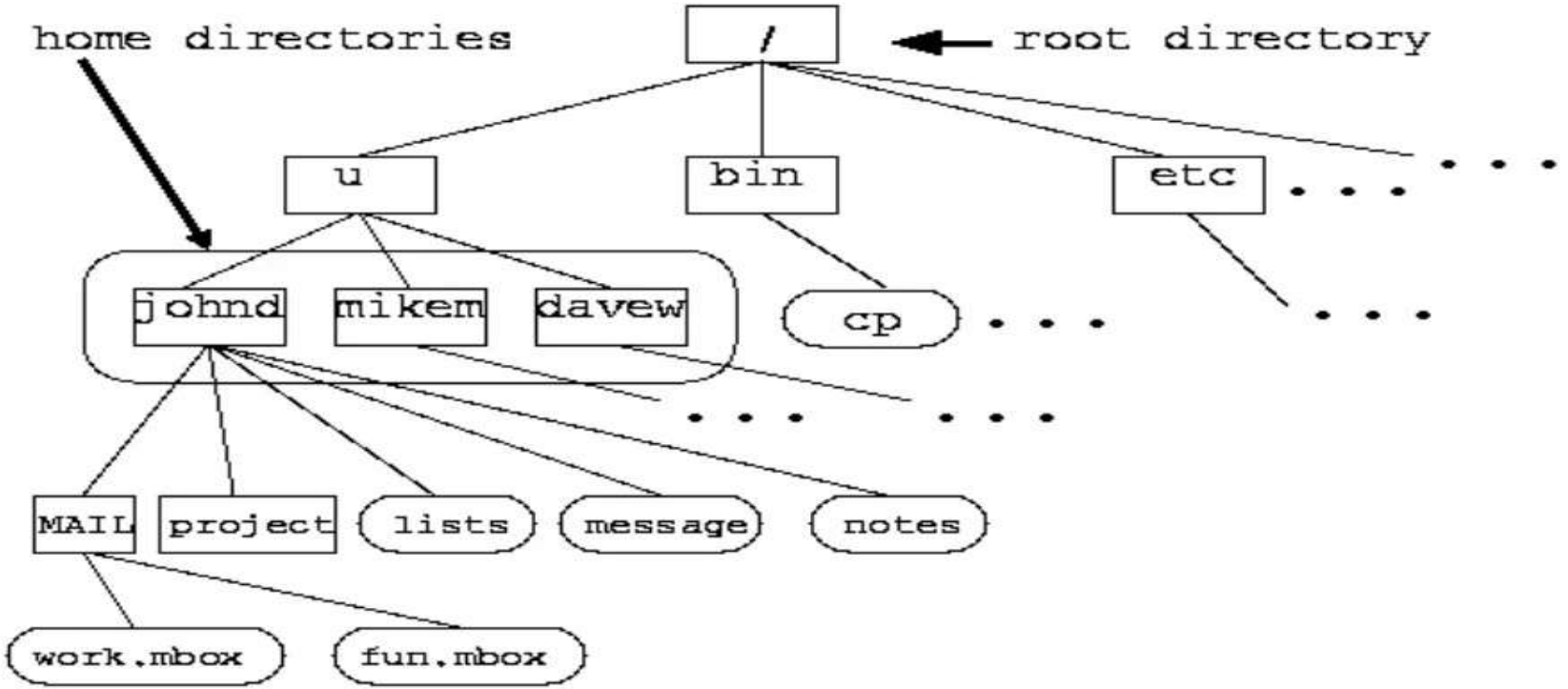
(6) tmp :-

- It is the temporary directory into which most of the temporary files are kept as compared to others. This directory assumes the least amount of importance both from the users and system point of view
- Files stored in this directory are automatically deleted when the system is shut down and restarted

- In UNIX there are 5 types of files:-
  1. ascii / ordinary files (-)
  2. directories (d)
  3. Character special files (c)
  4. Block special files (b)
  5. FIFO files or pipes (p)

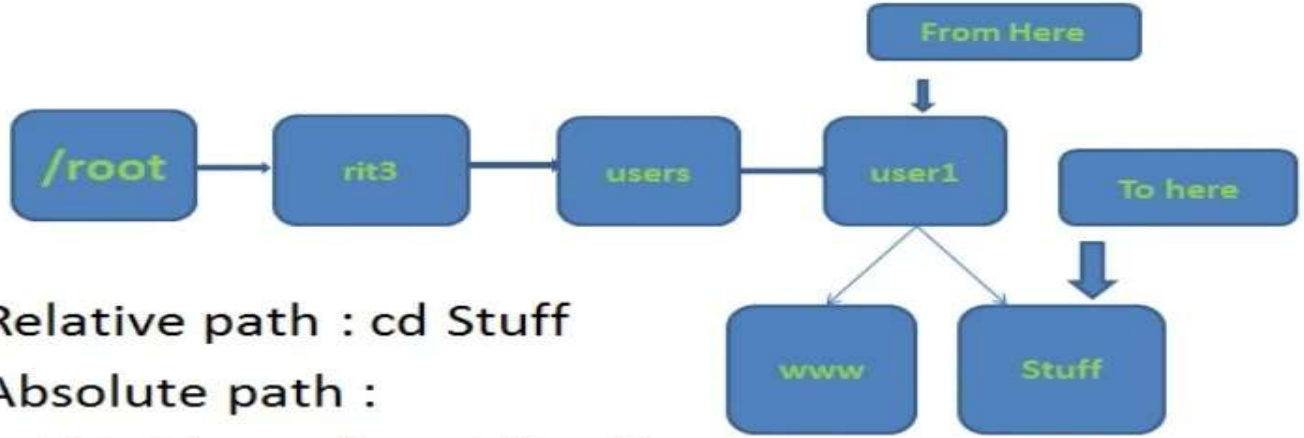
- The special block & character files are located in the /dev directory and are used to handle block and character devices
- If the file is the special file(b or c) ls command will display the major & minor devices numbers instead of the file size
- In normal practice this files are never found in the user directory

File Organization



Absolute path & Relative path

- Absolute path : starts from root(/) and follow the tree
- Relative path : starts from current directory



Relative path : cd Stuff

Absolute path :

cd/rit3/users/user1/Stuff

Unix Commands

- **Internal commands:**
  - commands built-into the shell are known as internal commands.
  - The shell does not create a separate process to run internal commands.
  - cd, echo, pwd etc.
  - Similarly, variable assignment with the statement x=5.
- **External commands:**
  - commands stored in **/bin** directory are known as external commands.
  - External commands require the shell to create a new process.
  - ls, cat, date etc.
- **Shell scripts:**
  - The shell executes these scripts by spawning(generating) another shell, which then executes the commands listed in the script.
  - The child shell becomes the parent of the commands that feature in the script.

Unix Command Line Structure

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

  Command-name ***[options] [arguments]***
  - **argument** indicates on what the command is to perform its action, usually a file or series of files.
  - **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]*

    e.g.: **ls** -alR
  - will perform a long list on all files in the current directory and recursively perform the list through all sub-directories.

- For most commands you can separate the options, preceding each with a hyphen, e.g.:

  command *-option1 -option2 -option3*

  e.g.: **ls** –a -l -R

- These are the standard conventions for commands. However, not all Unix commands will follow the standard. Some don't require the hyphen before options and some won't let you group options together.

pwd (Print/Present Working Directory)

- It identifies the current working directory and displays the full path-name for the current working directory.

  e.g. $pwd <enter>

  /usr/bin

  $

- Here /usr/bin is the directory in which the user is currently working.
- Purpose of *pwd* command is to determine where you are in the file system hierarchy.

who command

- It displays all the user currently logged into the UNIX system. This command produces following output:

  e.g.$who

  ```
  root     pts/0        Jan 6 10:42
  user1    tty01        Jan 6 14:45
  user2    tty02        Jan 6 16:10
  $
  ```

- In the above output, the first column indicates user name/login name, second column indicates device name of their respective terminals, and last column represents date and time of logging in.
- If you want to display header line, then –H option is used. –u option provides a more detailed list of user.

  e.g.$who –Hu <enter>

  ```
  NAME  LINE   TIME          IDLE  PID  COMMENT
  user1 tty01  Jan 6 14:45     .    30
  user2 tty02  Jan 6 16:10    0:40  31
  ```

- In the fourth column dot (.) shows that activity has occurred in the last one minutes before the command was invoked. user2 seems to be idling for the last 40 minutes.

- Suppose you want to know your own details, then ***am i*** option is used with ***who*** command.

  e.g.$who am I <enter>

  user1 tty01 Jan 6 14:45
- This command displays login details pertaining to the user who invoked this command.
- **Who** command is regularly used by the system administrator to monitor whether terminals are being properly utilized or not.
- Some times you want to send message to another user, who is currently login. For that first of all, you have to check his terminal is writable(+), not writable(-) or unknown(?). ***who – T*** command is used to see such things.

  e.g.$who –T

  user1 -tty1 Jan 7 10:45

  user2 +tty2 Jan 7 11:02

  user3 +tty3 Jan 7 11:20

  user4 -tty4 Jan 7 11:30
- In above display, it places a '+' next to the user who have allowed messages and a '-' sign indicates disallowed messages.

- ***-q*** option 'quick'. Display all login names / usernames and number of user logged in.

  e.g. $ who –q

  user1 user2 user3

  #users=3

  $

- ***-m*** option display only host name and user associated with stdin (terminal).

  e.g. $who -m

  linuxhost!user1 pts/1 jul 12 10:20

  $

- where in the first column it display hostname and user_id.

tty command

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

  $tty <enter>

  /dev/tty01

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

bc command

- Unix provides two types of calculators – a graphical object (Xcal command) and the character based (bc command).
- When you invoke *bc* without arguments, the cursor keeps on blinking and nothing seems to happen.

```
$bc
12+5
17
2*2;2^3
4
8
<ctrl+d> or quit   or halt   #indicates end of bc command.
```

- By default, bc performs truncated division, and you have to set ***scale*** to the number of digits of precision before you perform any division

  Scale=2

  17/7

  2.42
- bc is quite useful in converting numbers from one base to another, set ibase (input base) before you provide the number.

  ibase=2

  101

  #output in decimal base 10
- The reverse is also possible (i.e. decimal to binary)

  Obase=2

  5

  # binary of 5

- Similar to hexadecimal (base 16). If you give wrong base then by default it takes its default base.
- bc can also be used with variables that retain their values until you exit the program:

  e.g. x=3;y=5;z=1

  r=x+y+z

  r

  9
- bc also contains program language features such as arrays, functions, conditions (if) and loops (for and while)

  e.g. for(I=0;I<5;I++)I

  0

  1

  2

  3

  4

- ***bc*** also supports functions like sqrt, cosine, sine, tangent, exponent etc

  sqrt(4)

  2

  sqrt(11)

  3

  scale=2

  sqrt(11)

  3.31

- Another in-built function s(), e(), c() etc stand for sine, cosine, exponent respectively would work only when ***bc*** is invoked with the ***–l*** option. (i.e. bc –l)

  e.g. a=`echo “scale=2;5/3”|bc`

  echo $a -> 1.66

ls (list) command

- It displays files and directories in current working directory (if not specified).
  syntax:
  ls [options] argument-list
- There are various options used with ls command:

Option	Description
-x	Display multi-column output (sort line by line)
-F	Marks executables with *, directories with / and symbolic links with @
-a	shows all files including ., .. and those beginning with dot.
-A	shows all files and those beginning with dot but excluding . and ..
-R	Recursive listing of all files in subdirectories.
-L	Lists all files pointed to by symbolic links.
-l	Long listing showing 7 attributes of a file.
-n	Displays numeric user-id and GUID instead of their names (use with –l option).

Option	Description
-t	Sorts by last modification time.
-lt	Displays listing sorted by last modification time.
-u	Sorts by last access time.
-lut	Displays and sorts by last access time.
-i	Shows inode number.
-r	Sorts files in reverse order.

- ***ls –l*** gives long listing or detailed information about files or directories.

  -rw-r- -r-- 1 root root 4174 Jan 12 10:45 chap1

- The first column shows the type and permission associated with each file. In this column, the first character indicates the file type. It has any of the following characters:

Character	Meaning
-	Indicates ordinary files
d	Indicates directory files
c	Character special files
b	block special files
l	link file

- Then, you see a series of characters that can take the values r, w, x and –(hyphen) for owner, group and others (three character for each). In the Unix system, a file can have three types of permission – read, write and execute. –(hyphen) indicates that permission is denied.

- The second column indicates the number of links(alias) associated with the file. This is actually the number of filenames maintained by the system of that file.
- The third column shows the owner of the files. When you create a file, you automatically become its owner.
- The fourth column shows the group of owner. When the system administrator opens user account, he simultaneously assigns the user to some group.
- The fifth column shows the size of file in bytes.
- The sixth column indicates the last modification time and date of the file.
- The last column shows the name of the file/directory.

wc command

❑ It prints the number of lines, words and characters (bytes) for each file(s), and a total line if more than one file is specified.

Syntax:

wc [option] [filename(s)]

- Following are the options used with *wc* command:

-l (lines) - It prints the newline counts.

-L(max-line-length) - It prints the length of the longest line

-w (words) - It prints the word counts

-c (character/bytes) - It prints the byte/character counts

- If you do not specify any options, by default, it display number of lines, words and characters in file(s).

  e.g. $wc file1 <enter>

  1 4 15 file1

- The ***wc*** command is capable of accepting input directly from the keyboard.

  e.g. $wc <enter>

  This is wc command <enter>

  <ctrl+d>

  1 4 18

- You can use multiple option at a time such ***lc, lw, wc, lcw*** etc..

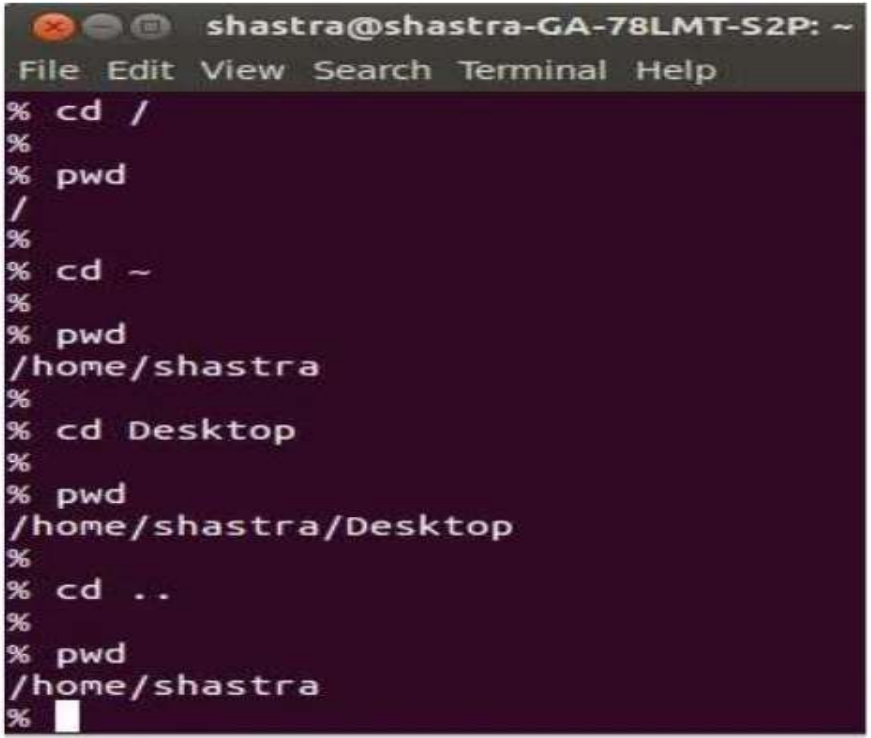
  e.g. $wc –lc file1<enter>

  1 15 file1

UNIX commands

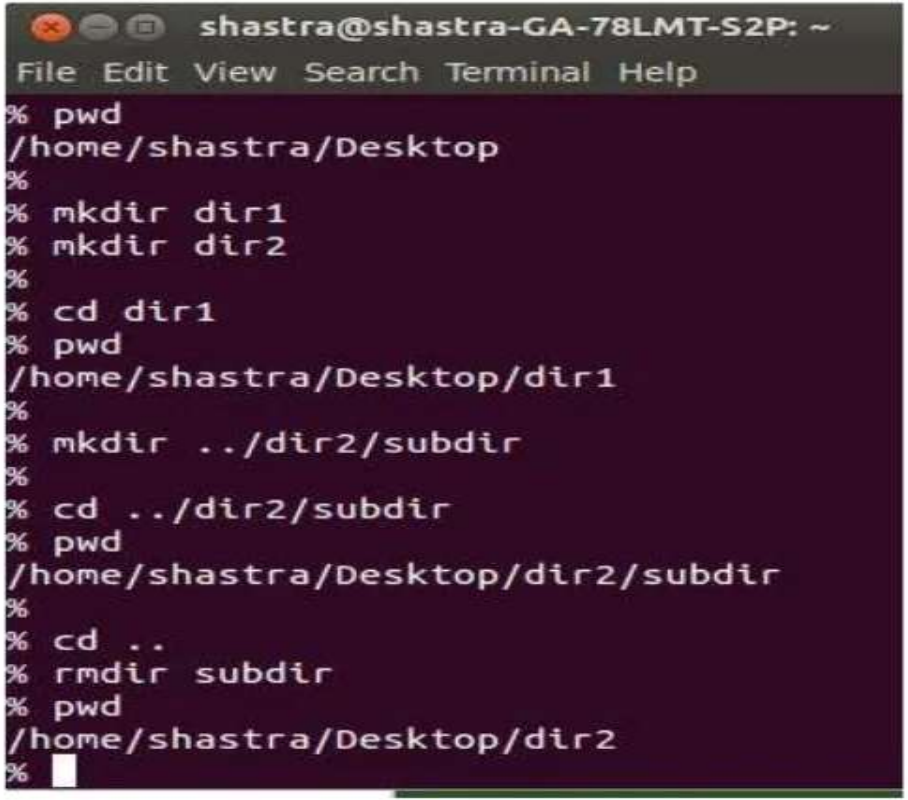
Changing directory

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



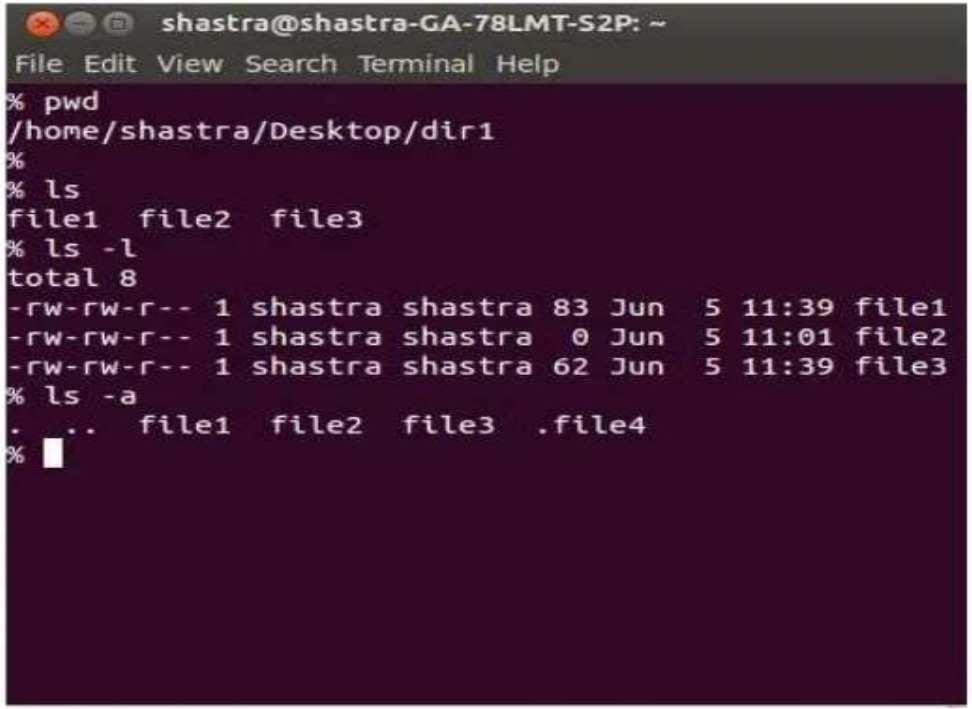
Creating and Removing Directories

- **mkdir** creates a directory
  - mkdir <dir_name>
  - mkdir path>/<dir_name>
- **rmdir** removes a directory
  - rmdir <dir_name>



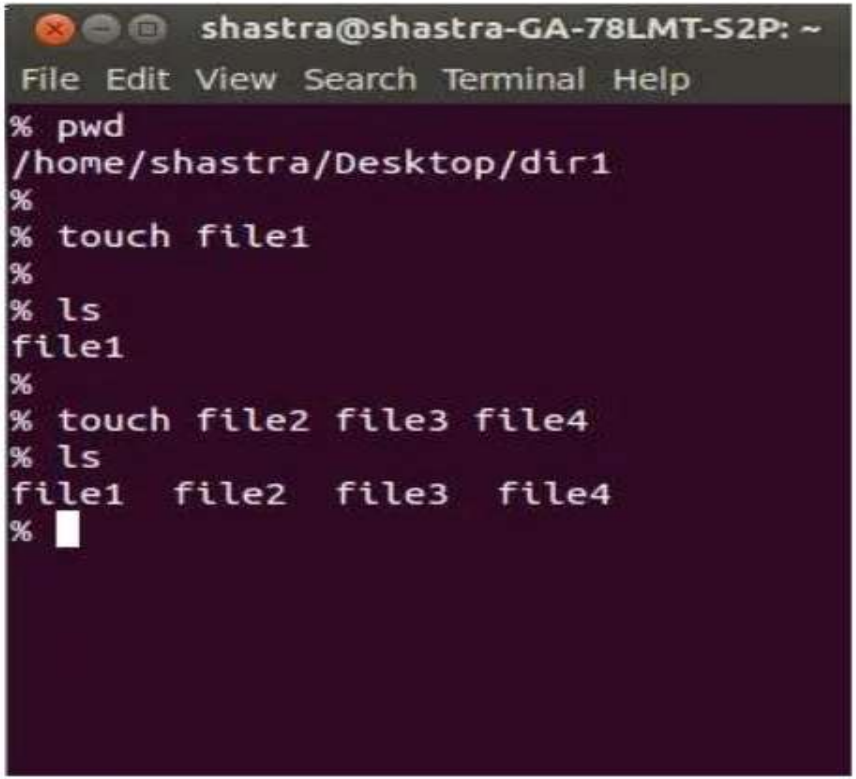
Listing Files

- ls list the files and directories in current directory
- ls –l → long listing the all details of files
- ls –a → hidden files
- ls –t → sort the files
- Ls -R recursively displays all files in subdirectories



Creating Files

- touch is used to create new, empty files
- touch <file_name>
  - The argument to the command taken as new file name
  - Touch <fie1><file2><file3>
- Can create any no.of files simultaneously
- This command can change timestamps for existing files



Creating and Editing files

- vi editor to create an ordinary files
  - vi <file_name>
    - Press i - come into insert mode
    - esc - come to command mode
    - :wq to save and come out of file completely.
  - vim, gvim, gedit, nedit, emacs are some of editors

vi Cheat Sheet

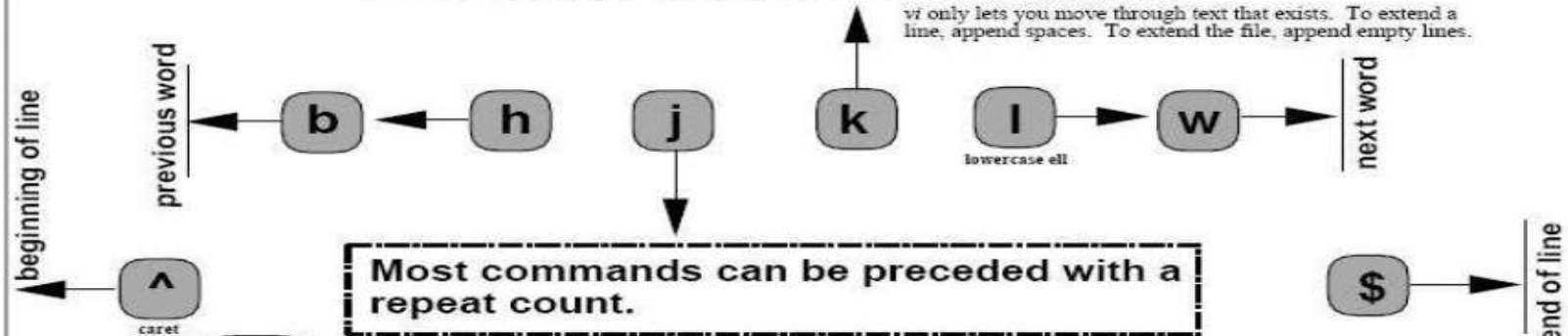
You're in command mode when you start *vi*.

Pressing ESC always takes you back to command mode.

Contact us at info@kcomputing.com
+1 512 858 0380

Cursor Movement Commands

vi only lets you move through text that exists. To extend a line, append spaces. To extend the file, append empty lines.



5w move ahead 5 words

16k move up 16 lines

1G go to line 1

76G go to line 76

G go to end of file

ctrl G where am I?

Changing Commands...can include any cursor movement command

c$ change to end of line

d5w delete 5 words

dG delete to end of file

dd delete current line

x delete this character

u undo last change

U restore line

Add-text Commands...press ESC when you're done

a append here

A append at end of line

i insert here

I insert at start of line (capital i)

R overstrike

Exiting *vi*...be in command mode! Press Enter to end these commands.

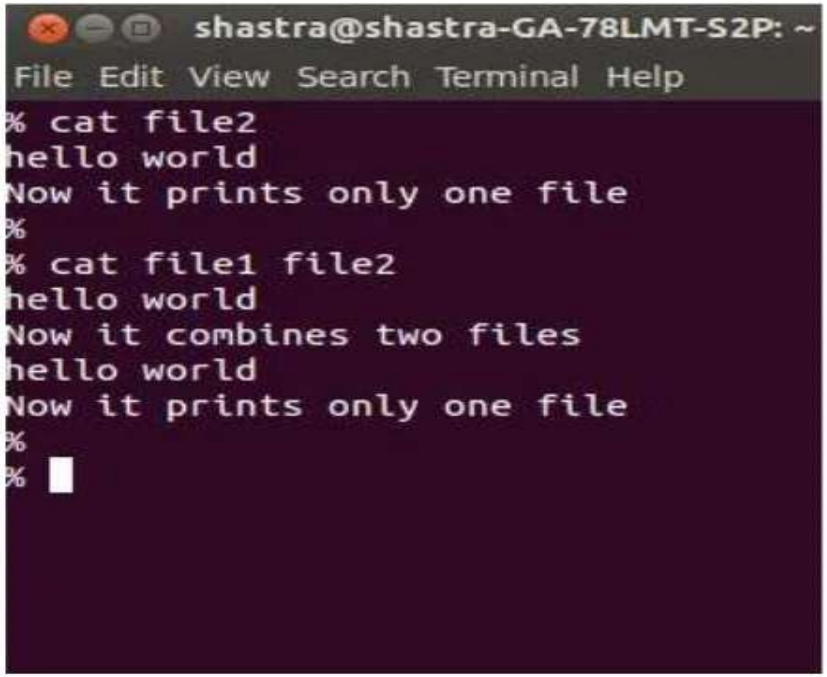
:wq write file then quit

:q! quit, forget changes

:q quit unless save needed

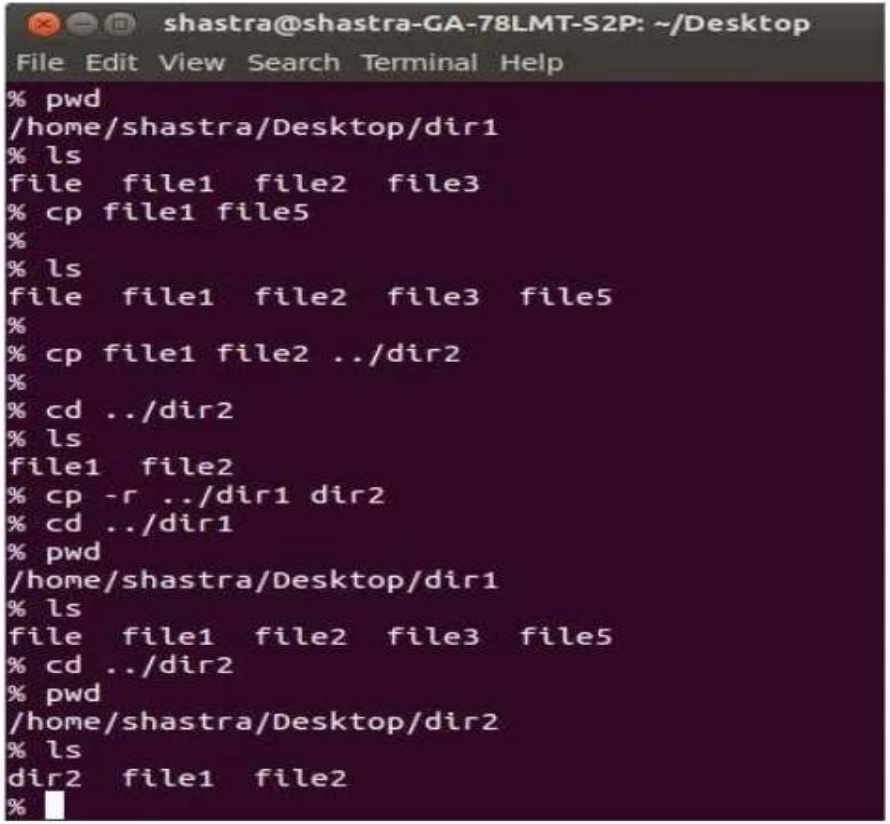
Display or concatenate files

- **cat** displays the file content on terminal.
  - Or display two files' contents together
- **cat** file1 file2
  - Combines and displays file1 and file2
- **cat** file1 file2 > filex
  - Moves file1 and file2 to filex



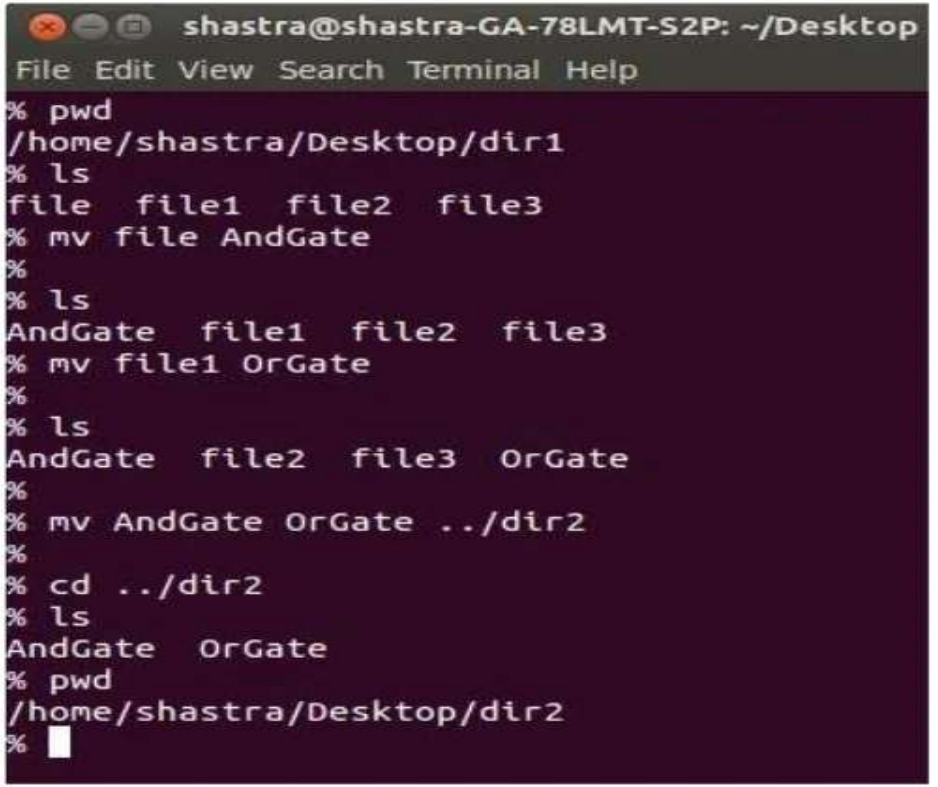
Copying and Moving files

- cp used to make copies of files and directories
  - Cp <source_file> <destination_file>
  - Cp file1 file2 dir1
    - Copies file1 and file2 into dir1 directory
  - Cp –r dir1 dir2
    - Copies directories using -r



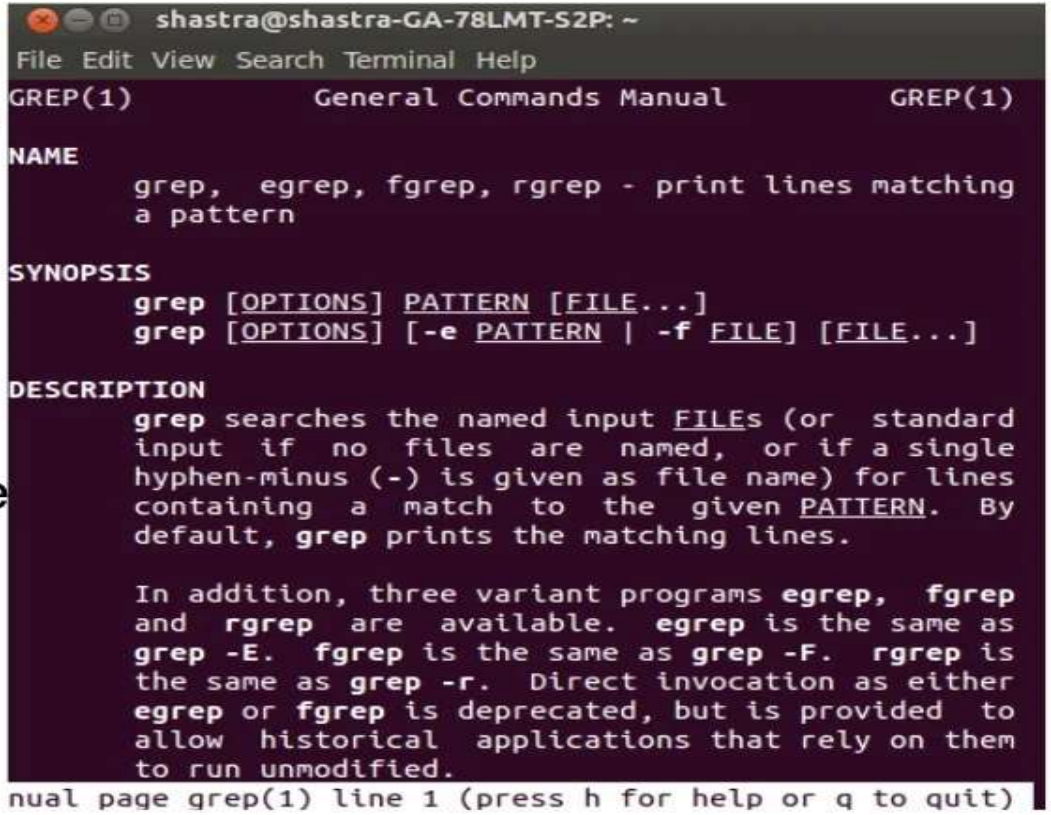
Moving and Renaming files

- mv used to change the name of the files or move the files into other directories
- mv file1 file2
  - Renames the file1 to file2
- mv file1 file2 dir1
  - Moves file1 and file2 to dir1 directory



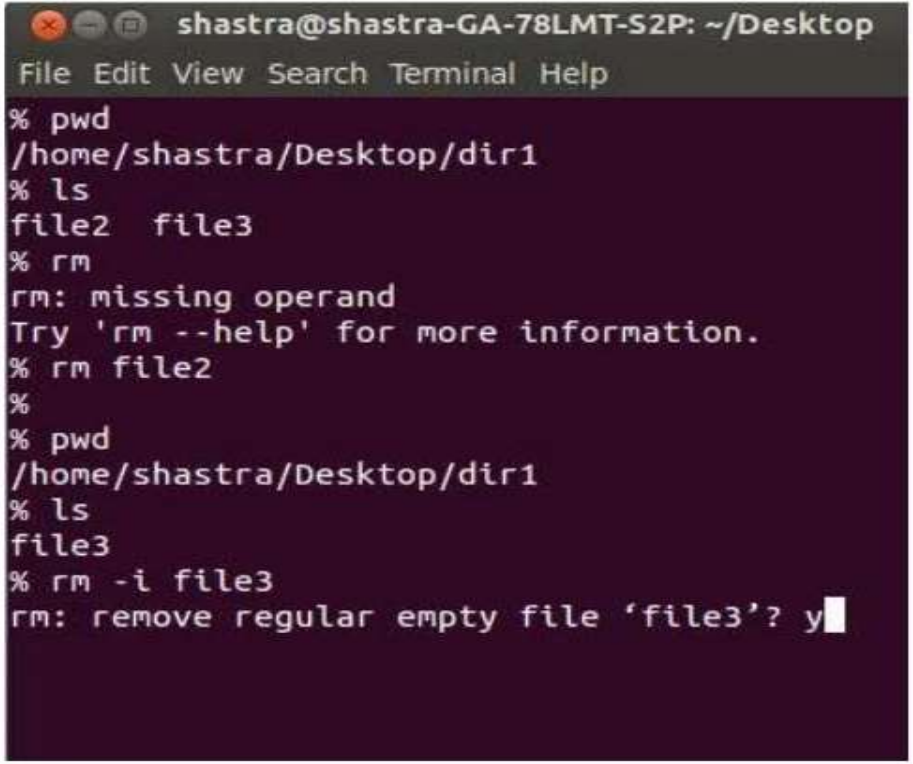
Getting help

- **man** used to get the manuals pages about the command
  - man <command>
  - (or)
  - <command> --help
- **Info** used to get the description of all available commands.
  - info
  - Info <command>



Removing files

- rm is used to remove files
  - rm <file1>
    - Deletes the existing file
  - rm -i <file1>
    - You will be asked that you wish to be delete or not
  - rm -r <directory>
  - Recursively deletes the contents of directory,
  - its subdirectories and deletes directory itself



Commands on files

- **Sort** this command sort and combines all the lines in file
  - Sort -d <file1>
    - Sorts based on dictionary order in which letters, digits, whitespaces considered
  - Sort -r <file1>
    - Reverse the order of the combining sequence
- **diff** display difference between two files
  - Diff <file1> <file2>
    - Reports line by line difference between the text files
- **file** tests named files to determine the categories their contents belongs to.
    - File <file1>

Display commands

- **echo** prints the given argument in standard output device
  - echo “type a string”
- **head** displays the head or start of the file
  - head -number <file1>
  - Head -10 file1
- **tail** displays the tail or end of the file
  - tail -number <file1>
  - Tail -20 <file1>

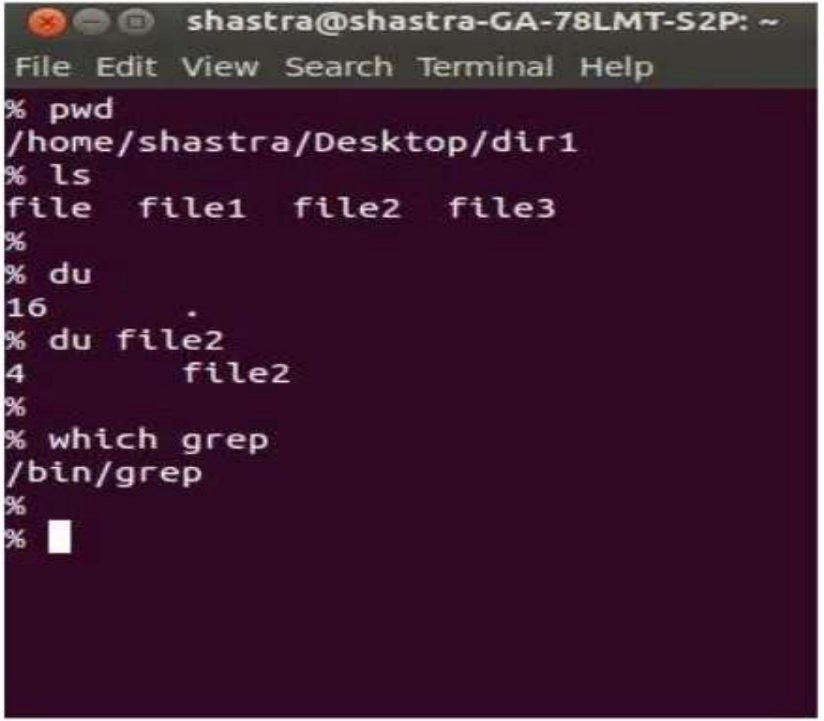
Display commands(more, less)

- more display the large file in one screenful at a time
  - more <file1>
  - Allows only forward control
- less similar to more
  - Less <file1>
  - Allows forward and backward movement

- more, less can also be used for pattern searching
  - more +/pattern <file1>
  - Less +/pattern <file1>

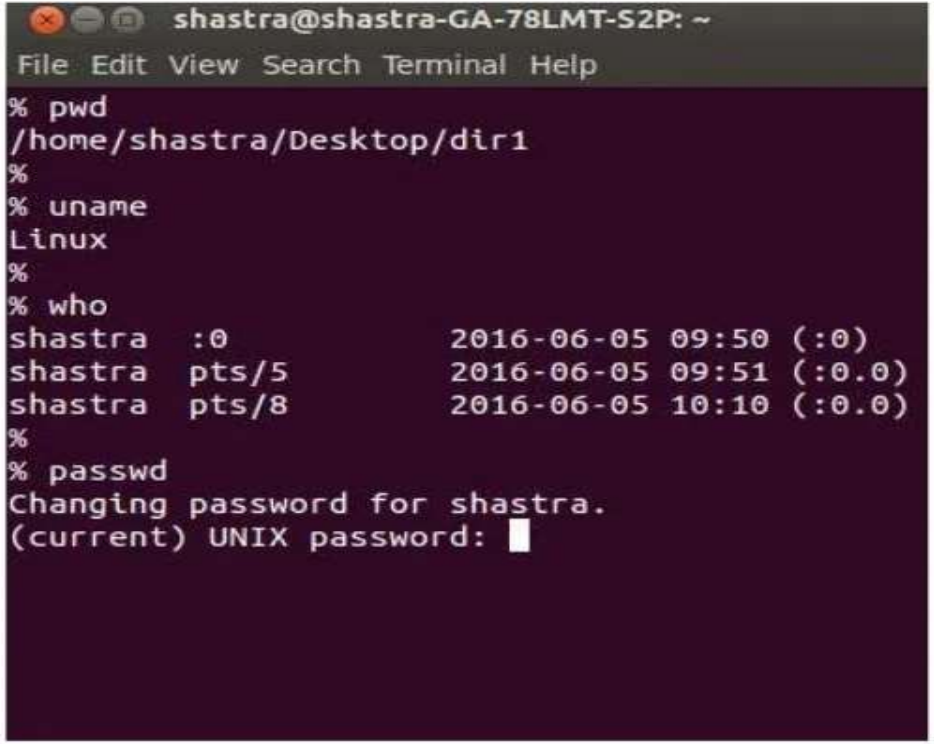
System resource commands

- **date** report current date and time
- **which** reports path for the specified command
  - which <command>
- **du** report the disk usage of specified file or directory
  - du <dirname>



System resource commands

- uname display or set the name of the current machine
- passwd change or set the password
- who report who is logged in and what processes are running
- script saves everything that appears on the screen to a file until exit is executed.
- script <file>



Find files of a specified name or type

- **find** searches for files in a named directory and all its subdirectories.

- find . -type f -name "*.txt"
- Searches current directory for the files end with .txt extension

find < > -type < > -name "pattern"

Path from where
You want to search

d – directory
f – files
l - link

Pattern Matching

- grep searches for lines containing specified pattern
- grep <pattern> <file_name>
- grep -i <pattern> <file_name>
  - For insensitive search
- grep -wocv <pattern> <file>
  - -w Prints line which have individual word
  - -o it prints only the word
  - -c it gives the count
  - -v prints lines which didn't match

Output Redirection Operator

The output redirection operator is used to redirect the output (from a command) that is supposed to go to a terminal by default, to a fi le instead. This process of diverting the output from its default destination is known as *output redirection*.

Syntax `command [> | >>] output_file`

Here, `output_file` is the name of the fi le where we wish to direct and save the output of the command.

Example `$ ls > kk`

On using this command, nothing will appear on the output screen and all output, that is, the list of files and directories from the `ls` command is redirected to the file `kk`.

On viewing the contents of the fi le `kk`, we get the list of files and directories in it.

Note: If the file `kk` does not exist, the redirection operator will first create it, and if it already exists, its contents will be overwritten.

In order to append output to the fi le, the append operator, `>>` is used as shown in the following command:

```
$ ls >> kk
```

Input Redirection Operator

In order to redirect the standard input, we use the input redirection operator, the < (less than) symbol.

Syntax `command < input_fi le`
Here, `input_file` is the name of the file from where the data will be supplied to the command for the purpose of computation.

Examples
(a) `$ sort < kk`
The `sort` command in the example receives the input stream of bytes from the file `kk`.
We can also combine input and output redirection operators.

(b) `$ sort < kk > mm`
On using the command, nothing will appear on the terminal screen; instead the content
of the file `kk` will be sorted and sent directly to the file `mm`.

The pipe operator, represented by the symbol '|', is used on the command line for the purpose of sending the output of a command as an input to another command. The pipe operator is different from the output redirection operator, '>' in a way that the output indirection operator '>' is mostly used for sending the output of a command to a file, whereas the pipe operator is used for sending output of a command to some other command for further processing.

Syntax command1 | command2 [| command3…]
Example $ cat notes.txt | wc
4 20 75
Here, the output of the cat command is sent as input to another command, wc. The wc command counts the lines, words, and characters in the file notes.txt whose content is passed to it.
We can combine several commands with pipes on a single command line as follows:
$ cat notes.txt | sort| lp
This command sorts the content of the file notes.txt and sends the sorted content to the printer for printing.
***Note*:** The pipe operator provides a one-way flow of data that is from left to right, whereas the redirection operator enables two-way flow of data.

Piping

- Connect two commands together so that the output from one program becomes the input of the next program.
- ls –l | grep “Aug”
  - This would extract all the files created in August month

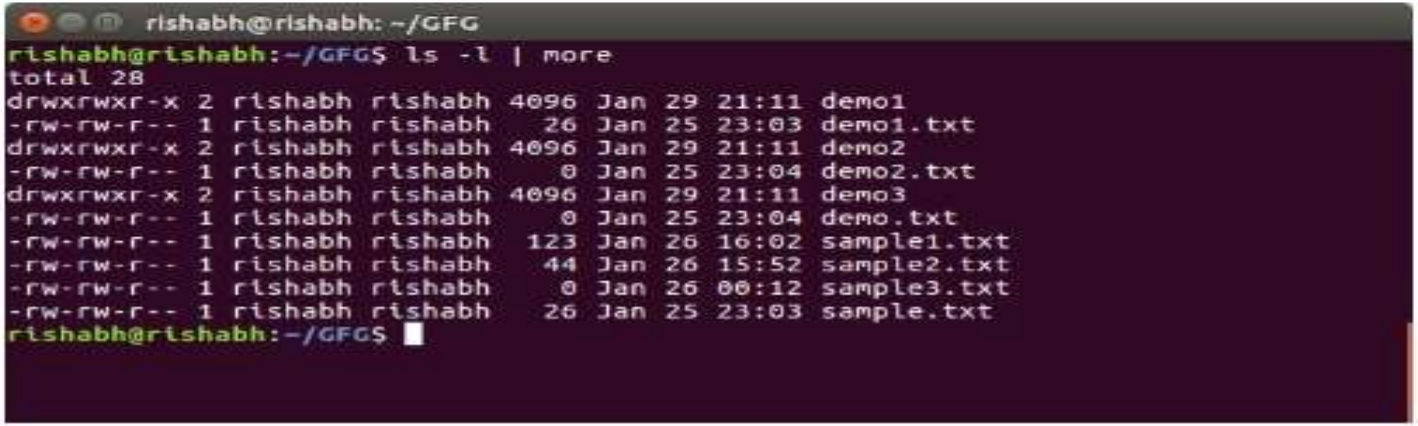
```
$ls -l | grep "Aug"
-rw-rw-rw-   1 john  doc      11008 Aug  6 14:10 ch02
-rw-rw-rw-   1 john  doc       8515 Aug  6 15:30 ch07
-rw-rw-r--   1 john  doc       2488 Aug 15 10:51 intro
-rw-rw-r--   1 carol doc       1605 Aug 23 07:35 macros
$
```

Example :

1. Listing all files and directories and give it as input to more command.

```
$ ls -l | more
```

Output :

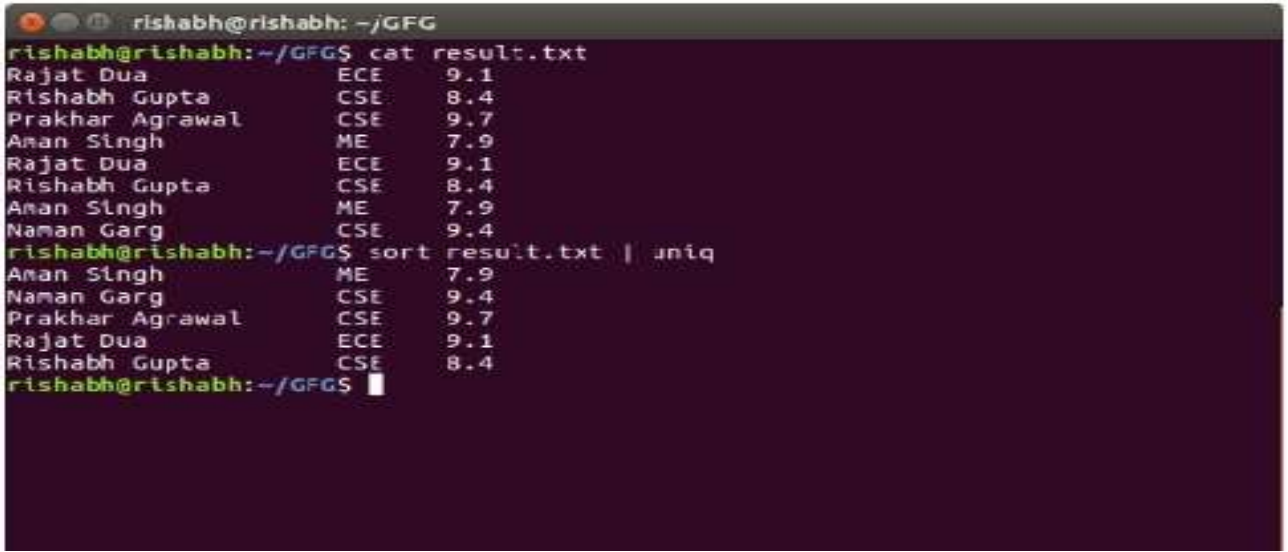


2. Use sort and uniq command to sort a file and print unique values.

```
$ sort record.txt | uniq
```

This will sort the given file and print the unique values only.

Output :



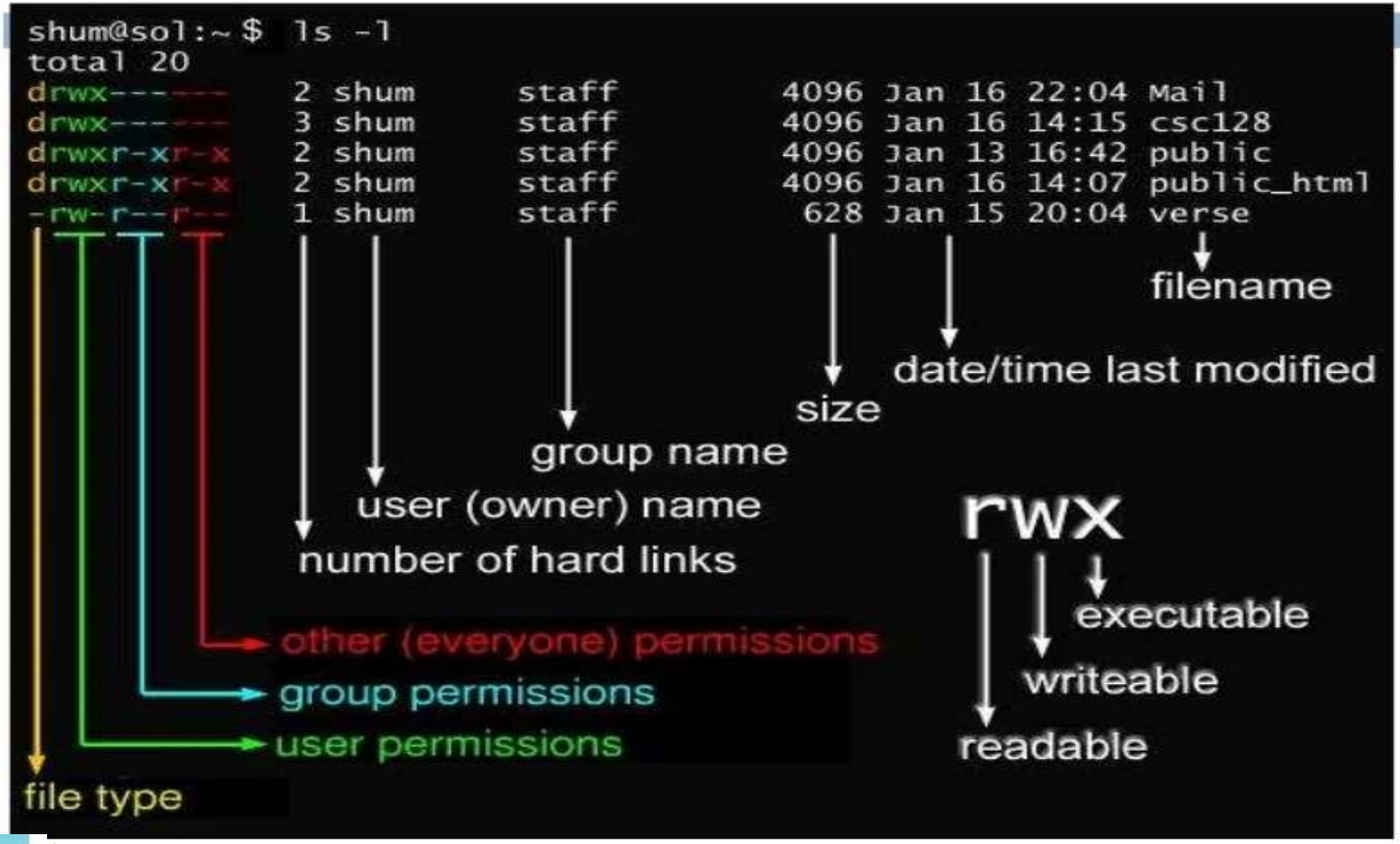
FILE ACCESS PERMISSIONS

1. *User* refers to the system user who created the fi le and is also sometimes called owner.
2. *Group* refers to one or more users who may access the fi le as a group.
3. *Other* refers to any other users of the system.

Access modes and permissions

Access mode	Ordinary file	Directory file
Read	Allows examination of file contents	Allows listing of files within the directory
Write	Allows changing of contents of the file	Allows creation of new files and removal of old ones
Execute	Allows execution of the file as a command	Allows searching of the directory

File Permissions



How file permissions are assigned

- File permissions, in UNIX, are assigned numerical values from 0 to 7. Individually however they have the following values:

  Read permission : 4

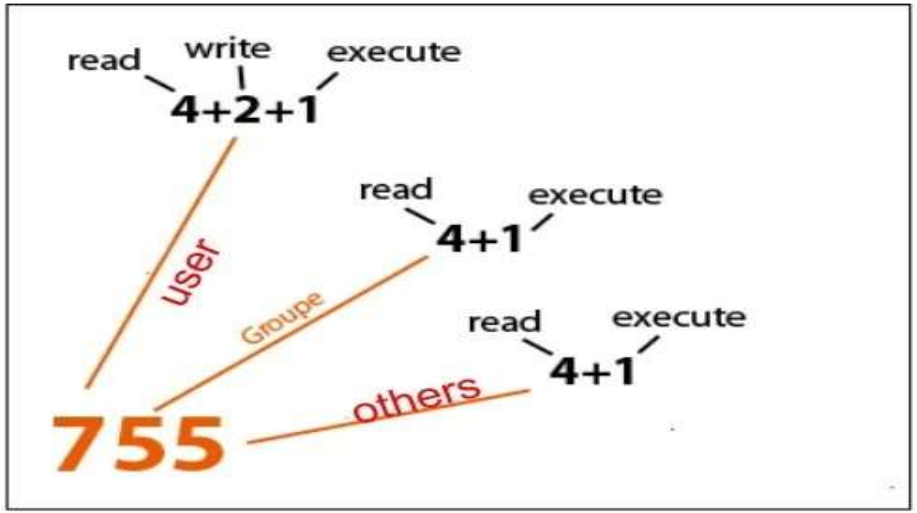
  Write permission : 2

  Execute permission : 1

- Thus, for a file that has a permission field like -rwxrwxrwx, the permissions are said to be 777(4+2+1) for the owner, group and others. Similarly, a file with permissions –rw_rw_rw_ is said to have 666 permission.
- A zero (0) value in the permission field is treated as a complete denied (removal) of the read, write and execute permission for all (owner, group and others)

Change permissions

- **chmod** alters the permissions on files and directories using either symbolic or octal numeric codes.
  - + to add a permission
  - - to add a permission
  - = to assign a permission
- chmod u=rw <file1>
- chmod u+x, g+w, o-w <file1>



chmod : Changing file permissions

- The ***chmod*** command enables you to modify the permission on a file. The syntax of the command is as follow:
  chmod [ugoa] [+ - =] [rwx] filename(s)
- The following options are available to the ***chmod*** command:
  - **u** applies to user
  - **g** applies to group
  - **o** applies to other
  - **a** applies to all (default: not specified any of these options)
  - **=** Instructs to add the specified permission and take away all others, if present.
  - **+** Gives specified permission.
  - **-** Takes away specified permission.
  - **r** read permission
  - **w** write permission
  - **x** execute permission

- **e.g.** Assume that file1 has access permission is –r_ _r_ _r_ _ then

  (1) $chmod +w file1

  Gives write permission to all i.e. –rw_rw_rw_

  (2) $chmod go+r, go-w file1

  Gives a read permission to group and others and take away their write permission.

  (3) $chmod go=r, u=rw file1

  This removes all existing permission and replaces them with read permission for group and others and read & writes permission for owner of the file file1.

  (4) $chmod a+w file1 OR $chmod +w file1

  Gives write permission to all (owner, group and others)

  (5) $chmod a-w file1 OR $chmod –w file1

  Take away write permission from all.

  (6) You can also specify the permission with octal numbers.

  $chmod 744 file1

  This would assign the permission -rwxr_ _r_ _ to file1.

  (7) $chmod 777 file1

  Assign all permission to owner, group and others i.e. –rwxrwxrwx.

- ***chmod*** works with multiple files also. i.e. you can assign the same set of permissions to a group of files using a single ***chmod*** command.

  e.g. $chmod u+x file1 file2 file3

- It is possible to apply the ***chmod*** command recursively to all files and subdirectories. This is done with the –R (recursive) option and needs only the directory name or the meta-character * as argument.

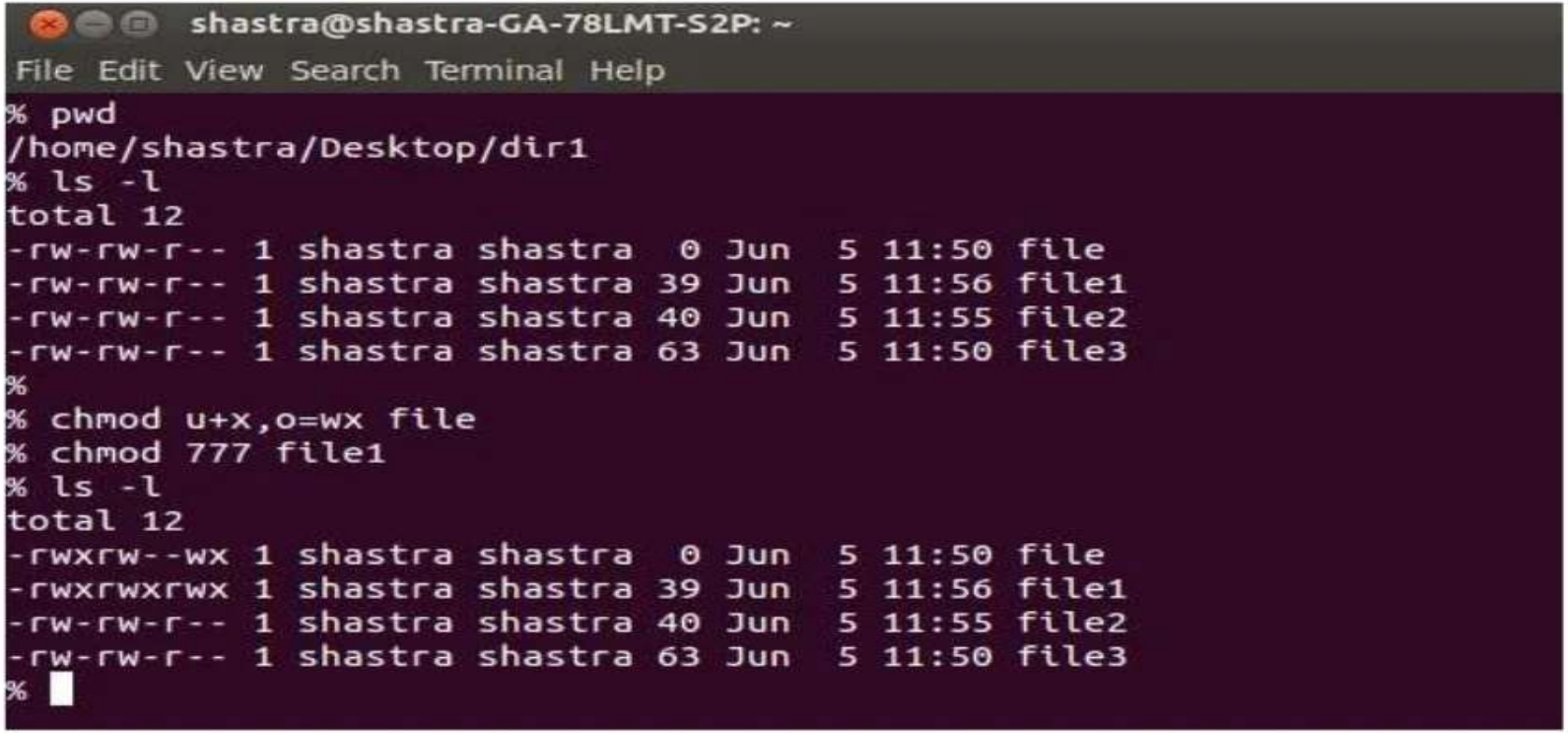
  e.g. (1) $chmod –R a+x mydir

  (2) $chmod –R a+x *

  OR

  $chmod –R a+x .

Change Permissions



chown and chgrp

- Chown command change the ownership of a file or directory.
  Syntax :

  chown options new-user file(s)

  e.g. user1 is owner of file file1. We can change the owner of the file as follow:

  e.g. chown user2 file <enter>
- similarly, chgrp command changes the group owner of a file. The general syntax is:

  chgrp options new-group file(s)

  e.g. chgrp bcasem1 file1
- Like chmod, both chown and chgrp also work the –R option to perform their operations in a recursive manner. All three commands place no restrictions whatsoever when used by the super user.
- In Linux, the chown command can only be used by the super user. A Linux user can use chgrp, but he can use it only for changing the group to one to which he also belongs.

Examples

By default, when we create or copy a file, we become its owner. For example, suppose we have a file named notes.txt and we want to change its ownership to another person named Ravi.

Let us first view the current owner of the file by giving the following command:

(a) $ ls –l notes.txt

-rwxrwxr-x 1 chirag it 120 Mar 15 12:20 notes.txt

We can see that chirag is the current owner of the fi le. Now chirag can give the following command to give the ownership of the file notes.txt to Ravi.

(b) $ chown ravi notes.txt

To see whether the ownership is changed, let us again give the ls –l command.

(c) $ ls –l notes.txt

-rwxrwxr-x 1 ravi it 120 Mar 15 12:20 notes.txt

We can see that the owner of the file notes.txt is changed from chirag to ravi.

Now, chirag will no longer be able to change the permissions of the file notes.txt and only ravi can do so.

For example, if we belong to the group *it*, our file will also have the same group ownership, as can be seen by the following command:

$ ls –l notes.txt
-rwxrwxr-x 1 chirag it 120 Mar 15 12:20 notes.txt
The following command is used to change the group ownership of a file named notes.txt from group it to group hospital:

$ chgrp hospital notes.txt

***Note*:** The group hospital must exist before giving this command.

Now, the group ownership may appear as follows:
$ ls –l notes.txt
-rwxrwxr-x 1 chirag hospital 120 Mar 15 12:20 notes.txt

***Note*:** Since we are still the owner of the file, we can again change its group ownership any time.

Displaying Group Membership

The groups command is used for finding the group to which a user belongs.
Syntax groups username1 [username2 [username3 …]]

Example
(a) % groups chirag
mba

This example asks the group name of the user, chirag. The output mba signifies that the user chirag belongs to the group named mba.

We can also find the group membership of more than one user simultaneously as follows:
(b) % groups chirag ravi
chirag : mba
ravi : other
This command asks the group names of the two users, chirag and ravi. The output indicates that the user chirag belongs to the mba group and the user ravi belongs to the other group.

Sharing Files Among Groups

Files can be shared with a group of users so that they can simultaneously read, work, and operate the files(s). For this to happen, a group needs to be created by the system administrator.

To create a group, we give the command with the following syntax:
Syntax groupadd group_name

Example % groupadd bankproject

This will create a group by the name bankproject. After creating a group, the next step is to set the group ownership of the file(s) to the given group using the chgrp command.

Syntax $ chgrp groupname fi lename

Example

(a) $ chgrp bankproject accounts.txt

This will set the group owner of the fi le accounts.txt to our newly created group bankproject. Similarly, we need to change the group ownership of all the fi les that we wish to share with the users of our group *bankproject*. Thereafter, we need to set the file permissions so that everybody in the group can read and write the fi le through the following syntax:

Syntax $ chmod g+rw filename

We can also assign access permissions to the group in the following way:

Syntax $ chmod 770 fi lename
This example assigns read, write, and execute permissions to the *owner* and *group* members of the fi le and no permission to the *other* users.

CUTTING DATA FROM FILES

The `cut` command is used for slicing (cutting) a fi le vertically.
Syntax `cut [-c –f] fi le_name`
Here, `–c` refers to columns or characters and `–f` refers to the fi elds, that is, words delimited by whitespace or tab.

file `bank.lst` has the following content.

```
101 Anil
102 Ravi
103 Sunil
104 Chirag
105 Raju
```

```
$ cut -f2 bank.lst
Anil
Ravi
Sunil
Chirag
Raju
```

The *Names* fi le consists of employee codes and names as follows:

```
101 Anil
102 Ravi
103 Sunil
104 Chirag
105 Raju
```

The *Telephone* fi le consists of employee codes and telephone numbers as follows:

```
101 2429193
102 3334444
103 7777888
104 9990000
105 5555111
```

```
$ cut -f2 Name > names.txt
$ cut -f2 Telephone > numbers.txt
```

PASTING DATA IN FILES

It is used to join textual data together and is very useful if we want to put together textual information located in various files.

Syntax `paste [-s] [-d "delimiter"] files`

Option Description

`-s` The `paste` command usually displays the corresponding lines of each specified fi le. The -s option refers to a serial option and is used to combine all the lines of each fi le into one line and display them one below the other.

`-d` This option is for specifying the delimiter to be used for pasting lines from the specified fi les. The default delimiter used to separate the lines from the fi les is the *Tab* character.

```
$ paste names.txt numbers.txt
```

```
Anil 2429193
Ravi 3334444
Sunil 7777888
Chirag 9990000
Raju 5555111
```

```
$ paste -d"|" names.txt numbers.txt
Anil:2429193
Ravi:3334444
Sunil:7777888
Chirag:9990000
Raju:5555111
```

```
$ paste -s names.txt numbers.txt
Anil    Ravi    Sunil   Chirag Raju
2429193 3334444 7777888 9990000 5555111
```

sort: SORTING FILES

It is used for sorting fi les either line-wise or on the basis of certain fields, where the fields refer to the words that are separated by one of the following: white space, tab, or special symbol.

Syntax `sort [-n][-r][-f][-u] filename`

head: DISPLAYING TOP CONTENTS OF FILES

The `head` command is used for selecting the specified number of lines from the beginning of the file and displaying them on the screen.

Syntax `head -[n] file name`
Here, `n` is the number of lines that we want to select.

Example `head bnk.lst`
When used without an option, this displays the *first ten records* (*lines*) of the specified file.

`head -3 bnk.lst` It displays the first three lines of the file `bnk.lst`.

`head -3 bnk.lst notes.txt`
It will display the first three lines of both the files, `bnk.lst` and `notes.txt`, one after the other.

tail: DISPLAYING BOTTOM CONTENTS OF FILES

The `tail` command is used for selecting the specified number of lines from the end of the fi le and displaying them on the screen.

Syntax `tail –[ncbr] filename`

Option Description

`-n` Selects the last `n` lines
`-c` Selects the last `c` number of characters
`-b` Selects a specifi ed number of disk blocks
`-r` Sorts the selected lines in reverse order

```
$ tail +10 bnk.lst
```

It will start extracting from the tenth line (it will skip nine lines) up to the end of the file.

```
$ tail –50c bnk.lst
```

It will display the last 50 characters of the file `bnk.lst`.

diff: FINDING DIFFERENCES BETWEEN TWO FILES

The `diff` command is used for comparing two files. If there are no differences between the two files being compared, the command does not display any output.

Syntax `diff file1 file2`

Assume we have two fi les, `users.txt` and `customers.txt`, with the following content.

```
users.txt        customers.txt
John             John
Peter            Charles
Troy             Troy
```

Now on comparing the two fi es, we get the following output.

```
$ diff users.txt customers.txt
2c2
< Peter
--
> Charles
```

***Note*:** The < character precedes the lines from the first file and > precedes the lines from the second file.

cmp: COMPARING FILES

The `cmp` command compares two fi les and indicates the line number where the first difference in the files occurs. The `cmp` command does not display anything if the files being compared are exactly the same.

Syntax `cmp [[-l][-s]] file1 file2 [skip1] [skip2]`

```
users.txt          customers.txt
John               John
Peter              Charles
Troy               Troy
```

The following are examples of commands that are used to compare the two files.

```
$cmp users.txt customers.txt
```

The `cmp` command compares the files `users.txt` and `customers.txt` and displays the following output.

```
users.txt customers.txt differ: byte 6, line 2
```

The output indicates that the byte location where the first difference between the two files (`users.txt` and `customers.txt`) occurs is 6.

lp: PRINTING DOCUMENTS

The term `lp` stands for line printer and the command is used for printing files.

Syntax `lp [ -d printer_destina tion ] [ -n number_of_copies] [ -q priority ] [ -H ] [ -P page_list ] fi le(s)`

(a) `$lp notes.txt`
This command prints the file `notes.txt` on the default printer.

(b) `$lp -d Deskjet1001 notes.txt`
This command prints the file `notes.txt` on the printer named `Deskjet1001`.

(c) `$lp -d Deskjet1001 -n 2 notes.txt`
This command prints two copies of the file `notes.txt` on the printer named
`Deskjet1001`.

cancel: CANCELLING PRINT COMMAND

The `cancel` command cancels existing print jobs.

Syntax `cancel [ id ] [ printer_destination ]`

The options and arguments used in this command are

Option	Description
id	It indicates the print job ID that we wish to cancel.
printer_destination	It removes all jobs from the specified printer destination.

Examples

(a) `$cancel Deskjet1001`

This command cancels all print jobs sent for printing at the Deskjet1001 printer.

(b) `$cancel 1207`

This command cancels the print job with ID 1207.

script: RECORDING SESSIONS

The `script` command is used for recording our interaction with the Unix system. It runs in the background recording everything that is displayed on our screen.

Syntax `script [-a] filename`

The options and arguments used in the command are

Option	Description
`-a`	It appends the session into the filename. If this option is not specified, the filename will be overwritten with the new data.
`filename`	This gives the name of the fi le where our session will be recorded. If we do not provide a filename to the `script` command, it places its output in a default file named transcript.

Example The following example will begin recording the session in the file `transact.txt`:

```
$ script transact.txt
```

To exit from the scripting session, either press Ctrl-d or write `exit` on the command prompt followed by the Enter key.

CONVERSIONS BETWEEN DOS AND UNIX

1. `dos2unix`: Converts text fi les from DOS to Unix format
2. `unix2dos`: Converts text fi les from Unix format to DOS format

The syntax for the `dos2unix` command is as follows:

Syntax `dos2unix [-b] file1 [file2]`

Syntax `unix2dos [-b] file1 [file2]`

COMPRESSING AND UNCOMPRESSING FILES

The `zip` command compresses a set of files into a single archive. The syntax for zipping a set of files into a compressed form is as follows:

Syntax `zip [-g][-F][-q][-r] file_name files`

Examples

(a) `$ zip abc *`

All the files in the current directory are compressed into a single file `abc.zip`.

(b) If we wish to add a fi le(s) that we forgot to add in the `zip` file, the following statement will solve the purpose.

```
$ zip -g abc a.txt
```

This example adds the fi le `a.txt` to an existing `zip` fi le `abc.zip`.

The following is the option to correct the damaged zip file.

```
$ zip -F abc –out pqr
```

This example fixes the `zip` file `abc.zip` if damaged, and copies the fixed version into another `zip` file `pqr.zip`.

In order to compress the fi les of subdirectories, we use the `-r` option.

```
$ zip –r abc projects
```

This example compresses all the files in the projects directory as well as in its
subdirectories and saves them in the `abc.zip` fi le.

unzip Command

The `unzip` command is used to unzip the archive and extract all the files that were compressed in it.

(a) To unzip a zipped archive, we use the following `unzip` command.

```
$ unzip abc
```

This example extracts all the files stored in the zip fi le `abc.zip` into the current directory.

(b) `$ unzip –d temp abc`

This command extracts the files in the archive `abc.zip` into the temporary directory.

(c) `$ unzip –p abc`

This command extracts the files in the archive `abc.zip` on the screen. The file content is displayed on the screen.

compress Command

The `compress` command compresses the specifi ed fi le. It replaces the original file with its compressed version that has the same filename with a `.z` extension added to it.

Syntax `compress [-c] [-f] [-v] fi le`

(a) `$ compress transact.txt`

This example compresses the fi le `transact.txt` and renames it `transact.txt.z`.

***Note*:** The original fi le is replaced by another file, which has the same name with a `.z` extension added to it
(i.e., `transact.txt` is replaced by the fi le `transact.txt.z`).

(b) `$ compress –c customers.txt`
It displays the compressed format of the fi le `customers.txt` on the screen, but does not compress it.

uncompress Command

This command is used to get the compressed fi le back to its original form. The uncompressed file will have the same filename with the extension `.z` removed.

Syntax `uncompress [-c] [-f] fi le`

7-zip—Implementing Maximum Compression

Besides `zip`, `bzip`, `gzip`, and other similar commands, Unix also supports the `7-zip` command.
The `7-zip` command is the fi le archiver command that compresses fi les at the highest compression ratio (around 30–50% more than the other zip formats).

Syntax `7z [a][d][e][x] [l][t] compressed_file [files_to_compress]`

(a) The following example compresses all the fi les with extension `.txt` in the current directory into the fi le `data.7z`.

```
$ 7z a data.7z *.txt
```

(b) The following example displays the list of fi les compressed in the fi le `data.7z`.

```
$ 7z l data.7z
```

(c) The following example tests whether the fi les in the compressed fi le `data.7z` are OK or not. If the fi les are found to be OK, the fi lenames are displayed along with a of information about the compressed fi les: size and number of fi les and folders compressed in it.

```
$ 7z t data.7z
```

FUNCTION SPECIFICATION

Command	Function
chmod	It changes file/directory permissions.
umask	It stands for user file creation mask and sets the default permissions of the files that will be created in the future.
chown	It transfers ownership of a file to another user. Once the ownership of a file is transferred to another user, one cannot change its permissions until they become the owner again.
chgrp	It changes the group ownership of the file.
groups	It creates a group.
sort	It sorts files either line-wise or on the basis of certain fields.

Command	Function
cut	It slices (cuts) a file vertically. Files can be cut on the basis of characters and fields too.
paste	It joins content from different files.
wc (word count)	It calculates the number of characters, words, and lines in a file.
head	It selects the specified number of lines and characters from the beginning of the given file; the default number of lines selected is 10.
tail	It selects a specified number of lines and characters from the bottom of the specified file; the default number of lines selected is 10.
pg	It displays a long file page-wise, that is, one screen page at a time.

Command	Function
man	It displays the online documentation or manual of the given Unix command.
diff	It displays the difference between two files in a format that consists of two numbers and a character in between. The number to the left of the character represents the line number in the first file, and the number to the right of the character represents the line number in the second file.
uniq	It finds and displays duplicate lines in a file. In addition, it can be used to display only the unique lines in a file. The -u option of the uniq command removes all duplicate lines from a file. The -d option of the uniq command is used to display all duplicate lines in a file.
split	It splits a file into a specified number of lines or bytes.
cmp	It compares two files and indicates the line number where the first difference in the files occurs. It does not display anything if the files being compared are exactly the same.

Command	Function
	amount of time that the command takes to execute its own code. The sys time represents the time taken by Unix to invoke the command.
lp	It stands for line printer and prints files. Using the lp command, we can define the printer we wish to use through the -d option, the number of copies through the -n option, the pages to print through the -p option, and priority through the -q option.
cancel	It cancels existing print jobs.
profile file	It exists in the home directory and is the start-up file that automatically executes when we log in to the Unix system. It can be used to customize our environment. It can also be used to write the commands and scripts that we wish to execute automatically when we log in.
calendar	It reads the calendar file and displays appointments and reminders for the current day.

`zip`	Used to compress a set of files into a single file.
`unzip`	Used to unzip a zipped archive.
`compress`	Used to compress a specified file. It replaces the original file with its compressed version that has the same filename with a `.Z` extension added to it.
`uncompress`	Used to uncompress the compressed file back to its original form.
`7-zip`	Used to compress files at the highest compression ratio (around 30–50% more than the other `zip` formats).