Operating System definition

An OS is a collection of s/w programs. It mainly controls the allocation and usage of h/w resources such as memory, CPU time, hard disk space etc.

All application programs use the OS to gain access to these hardware resources as and when they are needed. The OS is the 1st program to be loaded into the computer when it boots & it remains in memory at all times thereafter.

Types of Operating System

- **Single user OS** – this OS supports 1 user at a time. The user can perform one task as well as multiple tasks on the OS. **For ex,** MS – DOS, MS – Windows
- **Multi - user OS** – in this OS, multiple users can execute single task or multiple tasks at any given point of time. **For ex,** UNIX, LINUX, Windows Server OS

Differences between Windows & UNIX

UNIX	WINDOWS
1) All UNIX OS are multi – user OS.	**1)** It is a single user OS
2) It is a **command line interfaced based** OS i.e, any task is performed with commands.	**2)** It is a **GUI based OS**. Thus, it is easy to understand and also user friendly.
3) Security is very high.	**3)** Security is low compared to UNIX OS.
4) File Size – can extensively grow i.e, there is no limit as such for the file size.	**4)** File size is limited
5) Open source OS	**5)** Licensed OS
6) Not user friendly – because it is command based.	**6)** Very user friendly
7) The pathname looks like shown below, **/ home / demo2 / batch1 / class1** All are forward slash	**7)** The pathname looks like this, **C :\>home\demo2\batch1\class1** All are backward slash

ARCHITECTURE of an OS

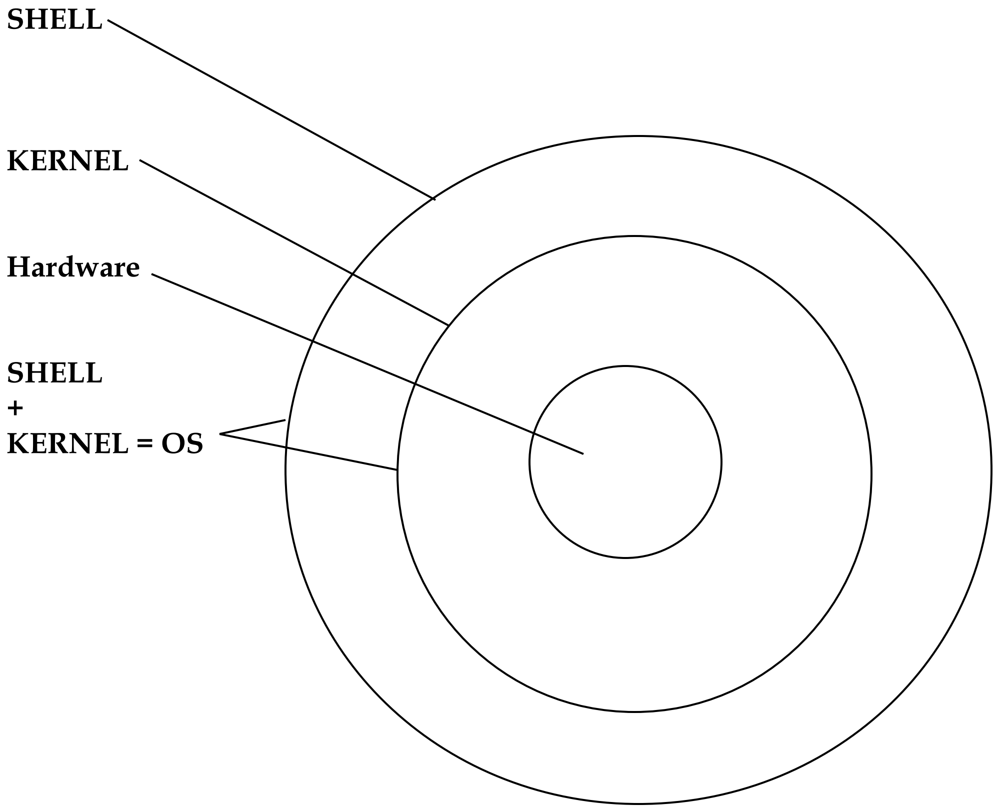
An OS is made up of 2 components, known as,

a) Shell

b) Kernel

The kernel is the core of an OS which manages the entire system resources

The shell acts as an interface between kernel and end user or application.



In UNIX OS, the shell is very protective – hence it is more secure. Whereas, in WINDOWS, the shell is less protective and more good in usability.

UNIX was originally called MULTICS (Multiplexed Information & Computing Systems). Then it changed to UNICS (Uniplexed Information & Computing Systems). Then it changed to UNIX – Extended version of UNICS.

<u>**Flavours of OS**</u>

- **AT & T** – Unix
- **IBM** – AIX
- **HP** – HP-UX
- **Sun Microsystems** – Sun Solaris
- **BSD** – BSD UNIX – BSD stands for Berkeley Software Distribution
- **SCO** – SCO UNIX – SCO stands for Santa Cruz Operations

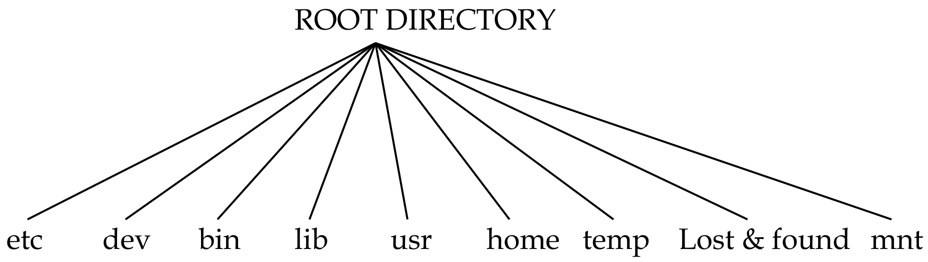
UNIX is a character based OS. It has been modified into a GUI based OS called LINUX. LINUX was developed by a person named Lynx.

Different versions of LINUX are,

- Red Hat
- Mandrake (HP)
- Fedora Core
- Ubuntu
- SUSE

NOTE :- Since it is difficult to install a LINUX Operating System. We can download a s/w named CYGWIN software – which simulates the UNIX like operating system in Windows itself.
Google and free download CYGWIN software.

UNIX File System



All files in UNIX are related to one another. The file system in UNIX is a collection of all these related files organized in a hierarchical (inverted tree like structure).
Every UNIX file system has a top, which serves as the reference point for all files. This top is called **root** and is represented by a frontslash (/). Root is actually a directory.
The root directory has a number of subdirectories under it. These subdirectories in turn have other sub directories under them.
Every file, apart from the root , must have a parent and thus it should be possible to trace the ultimate parentage of a file to a root.

Root – it is the starting directory for Unix OS. Denoted by **/**

Etc – contains all configuration files of Unix OS.

Dev – contains all device files like drivers.
A device file or special file is an interface for a device driver that appears in a file system as if it were an ordinary file. There are also special device files in MS-DOS and Microsoft Windows. They allow software to interact with a device driver using standard input/output system calls, which simplifies many tasks and unifies user-space I/O mechanisms.
Device files often provide simple interfaces to peripheral devices, such as printers.

Bin – stands for 'binary'. Contains binary files. Binary files are files which can be run i.e, they are executable files.

Lib – stands for 'library files'. It contains all re-usable programs/data.

Temp – just for temporary storage. Similar to recycle bin where we can store unwanted files.

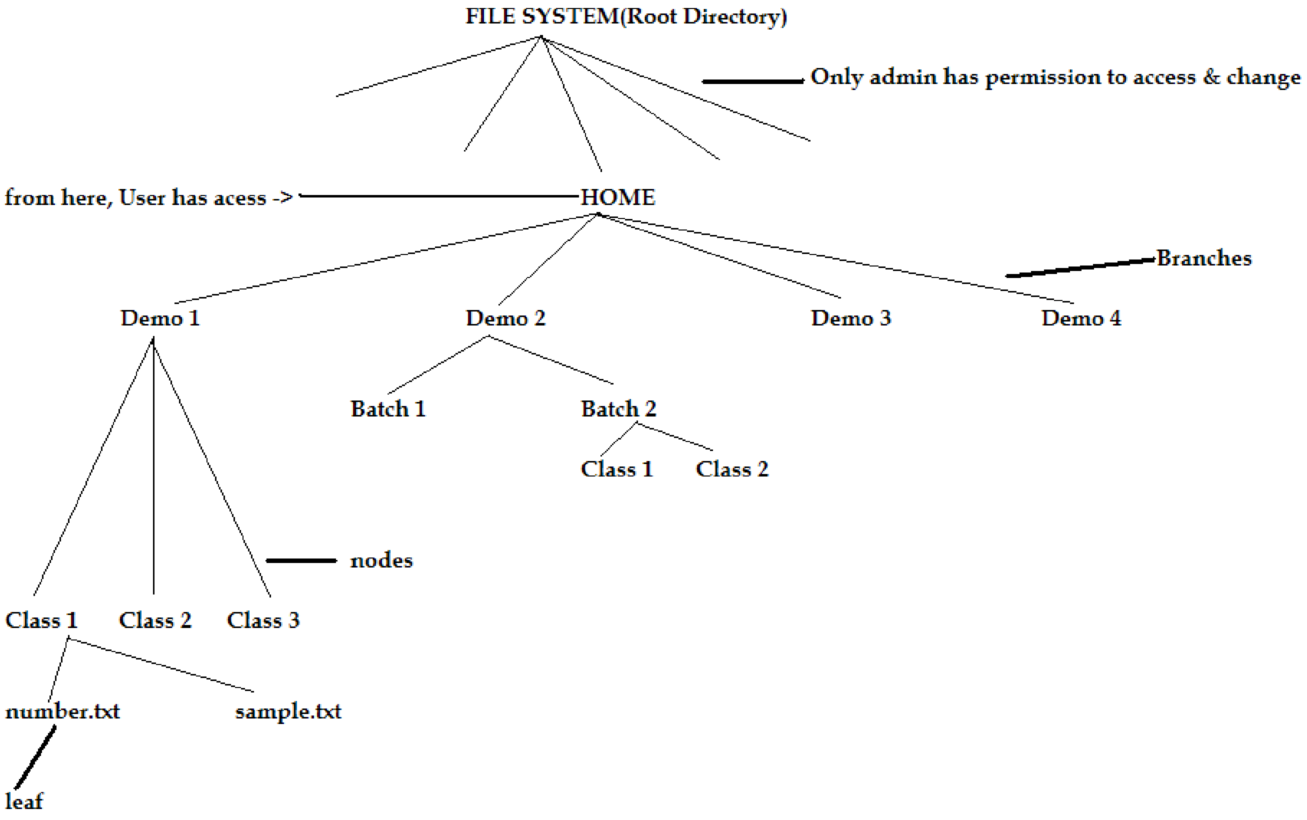
Lost & found – same as 'restore'. Lost files or directories are stored here. When we are working on Unix and there is a power shutdown. If we wish to recover the documents we were earlier working on, then they can be found in lost&found directory.

Mnt – stands for 'mount'. Used for mounting external devices. It is also used for connecting external storage devices like – HDD(hard disk drive), USB, DVD, C etc.

UNIX OS supports multiple users at any given point of time. The users of UNIX will be assigned a separate directory known as home directory. Each user of UNIX OS has their own username & password and separate home directory.
When a user is logged into UNIX OS, the default working directory will be user home directory.
A user directory is used for installation of a s/w which can be used by all the users of the UNIX OS.

Which directory will be there when we log into the system ?
Answer is *user home directory*.

A **user directory** is used for installation of a software which can be used by all the users of the UNIX operating system.



How to write **pathname** – say upto **demo 1 – class 1**,
In UNIX -> **/ home / Demo2 / Batch1 / Class1**
In Windows -> **C: \ > home \ Demo 2 \ Batch 1 \ Class 1**

How to open a shell

Right click on screen -> Click on **Open Terminal**

Shell types

- Bourne shell
- C – shell
- Korn shell
- BASH – Bourne Again Shell

We are using BASH
BASH prompt is **#** or **$**

# cd /
Changes the directory from the current directory to the required directory

# ls
Lists all the directories

# ls etc
Lists all the files in **etc** directory

# clear
Clears the screen

# ls dev
Lists all the files in **dev** directory

# ls bin
Lists all the files in **bin** directory

Syntax of UNIX commands

Starts with **$** or **#**

```
$ commandname    - options    arguments
```

All commands are lowercase & no space between the commands in command name.

Commandname – the operation / task to be executed. Ex – cd, mkdir, ls

- options -> it controls the way commands execute & its output. Always options must start with '-' (hyphen). Options are case sensitive & denoted by alphabets.

Arguments – it is nothing but data passed to the commands.

Ex - $ mkdir directory1 directory2

COMMANDS

Name : **UName**
Description : Unix name -> displays the name of the OS

Syntax: **$ uname**
Output: Name of the OS

Examples,
1) **$ uname –v**
Displays when the OS was released to the market
-v -> displays release date of OS

2) **$ uname –r**
Displays the version of the OS

3) **$ uname –r –v**
Displays both the release date & the version of the OS

Can also be re-written as,
$ uname -vr OR **$ uname –rv**

Name : **who**
Description : displays user names of all logged users

Syntax : **$ who**
Output : list of users who logged into Unix Os

```
Login Name          Time
```

Examples,
1) For 3 users, output will be

```
Login Name      Time
User 1          9:05
User 2          9:14
User 3          9:25
```

Q) ***How to find how many users are currently working on the Unix OS?***
Ans) use **who** command

Name : **whoami**
Description : displays user name of the user who is executing the command
Syntax : **$ whoami**
Output : displays the user who is executing the command

Name : **finger**
Description : similar to **who** command. Displays all logged users, but detailed information of all logged users.

Syntax : **$ finger**
Output : user details of all the logged users in the format shown below,

LoginName	**ActualName**	**LoginTime**	**Status**	**EmailId**	**PhoneNo**

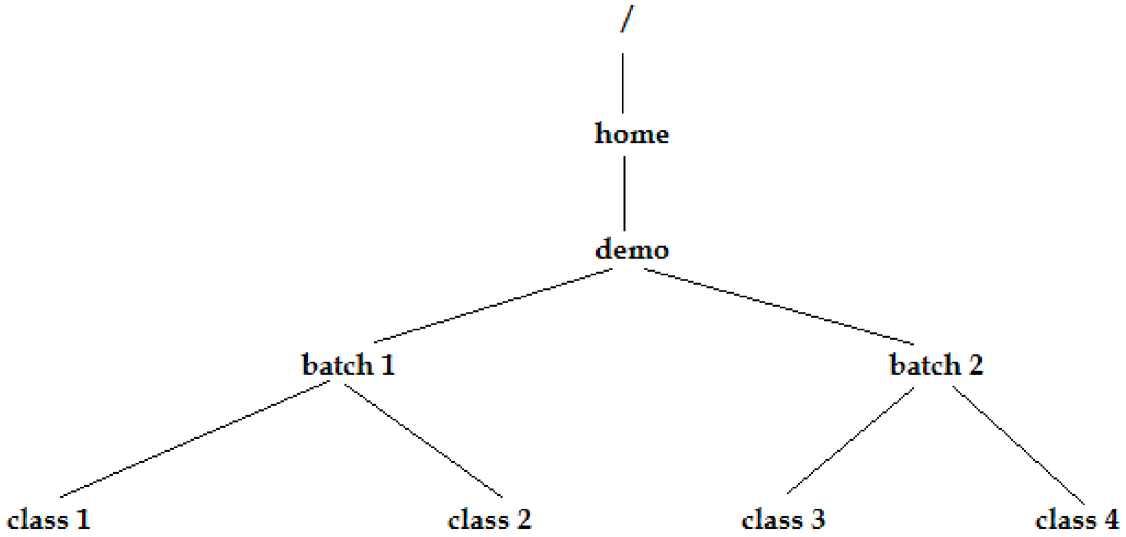
Q) ***What is the advantage of the 'Finger' command?***
Ans) we can communicate with other users by their email id & phone numbers

---

Name : **pwd**
Description : stands for Print Working Directory. Gives the current working directory

Syntax : **$ pwd**
Output : the current working directory path should be displayed.

Example



1) $ pwd
/home / demo / batch 1 / class 1

This is how we represent the path from root to class 1

Name : **cd**

Description : stands for '**change directory**'. It changes current working directory to another directory.

Syntax : **$ cd** path of directory

Examples,

1) $ cd batch1

This changes from current working directory to batch 1. We can check if it has changed to batch1 by using the command **$pwd**

2) To change from child class to parent class

i.e, to change from class 1 to batch 1

$ cd . .

The 2 dots represented above changes to parent directory from the current working directory.

$ cd . . / . .

This changes from class 1 to demo

3) to change from class 1 to class 3

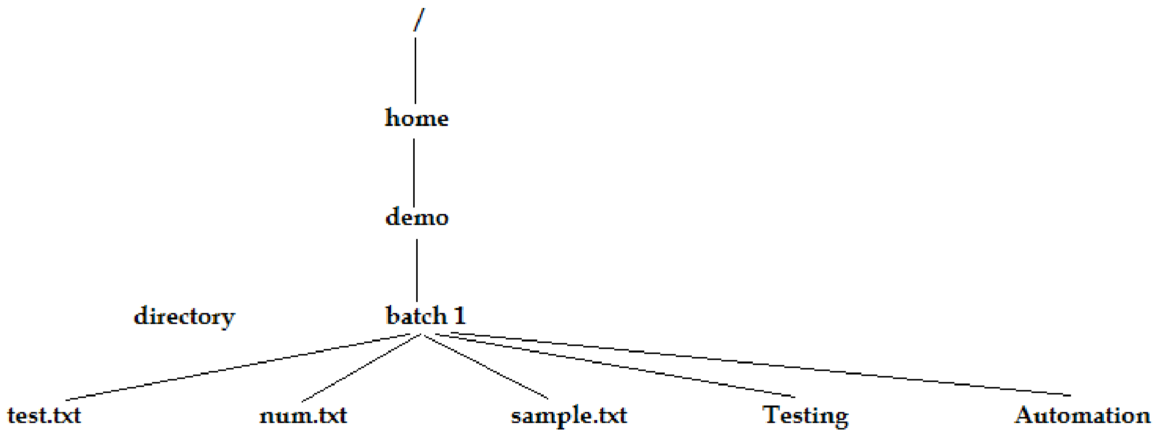
$ cd . . / . . / batch 2 / class 3

A path in file system can be specified in 2 ways,

- **Absolute path** – the path is entirely mentioned from the starting point in the file system & always the path contains parent to child navigation.
- **Relative path** – in this, the path is specified w.r.to current working directory. The path contains navigation from child to parent & parent to child.

$ cd – remains in the same directory

$ cd ~ -> goes to user home directory. ~ -> special variable which stores the path of user home directory.



Name : **ls**
Description : list directory contents. It list all the contents of directory.

Syntax : **$ ls**

Examples,
1) $ ls
Output is –

```
Automation    num.txt       sample.txt    test.txt       testing
```

We can see that all the contents are displayed in alphabetical order.

2) $ ls –F
List the contents of a directory with a differentiator i.e, it identifies the directories & executable files.

Q) ***How to differentiate if a file is a file or directory?***
Ans) we use the command **ls –F**

When the various files are listed using the **ls** command, the following color codes are used for the files which helps us in differentiating between them,

- ≈ Black content – file
- ≈ Blue content – directory
- ≈ Green content – executable files
- ≈ Brown content – zipped / compressed files

3) $ ls –l
Stands for long list. It gives the details of each & every directory.

It gives the details in 7 columns as shown below,

Permission	**Links**	**Owner**	**Group**	**Size**	**Time**	**Content name**
				In terms of bytes	Created date & time	

Permission has 10 bits / characters. The 1st bit represents whether it is a file / directory. If directory, it indicates 'd'. If it is a file, it indicates '-'.

```
-  |  - - -   |  - - -   |  - - -
   |  r w x   |  r w x   |  r w x
   |  OWNER   |  GROUP   |  OTHER(s)
```

User is classified into 3 categories,

- → Owner
- → Group
- → Others

R – stands for read. W – stands for write. X – stands for execute

The owner has all 3 permissions – to read, write & execute the files & directories.
The group – has only read & execute permissions.
Other(s) – has only read permission.

Any user who creates the file is the owner of the file.

4) $ ls –rl
It displays the reverse sorting order

5) $ ls –tl
It sorts with respect to time of creation or modification (recently created / modified)

6) $ ls –rtl
Reverse sort the content w.r.to time

7) $ ls –Sl
Sort w.r. to size of content (highest to lowest)

8) $ ls –rsl
Reverse sort w.r. to size of contents

9) $ ls –R
Lists the directories as well as sub directories recursively

Name : **man**
Description: opens manual pages

Syntax : **$ man** command-name

Example,
1) $ man pwd
Displays full description of pwd(print working directory)

Press **enter** to go to the next page in the description.
Press **Q** to come out of the document.

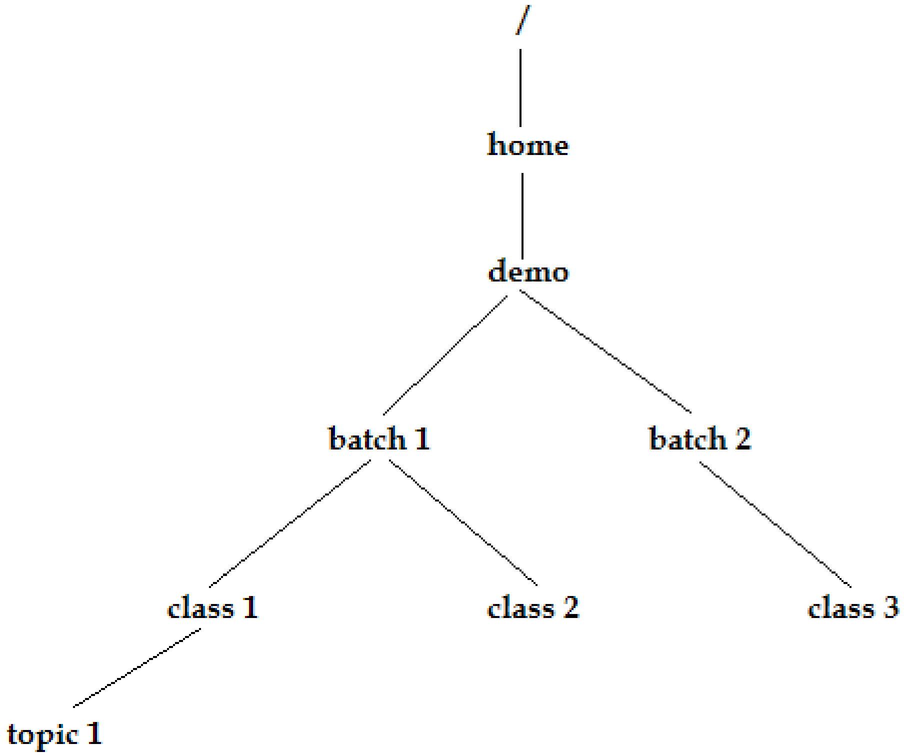
Name: **mkdir**
Description : Directories are created with mkdir command. This command is followed by the names of the directories to be created

Syntax : **$ mkdir** directoryname_1 directoryname_2

Examples,

1)
```
$ mkdir batch1 batch2
$ cd batch1
$ mkdir class1 class2
$ cd class1
$ mkdir batch1 / class1 / topic1
```

The above commands will create the following tree structure,



When a directory is created, the default size is 4KB.

Q) ***How to see the entire structure from root directory?***
Ans) **$ ls –R**
R stands for Recursive

Name: **touch**
Description: it touches the contents. It creates an empty file.

Syntax: **$ touch** filename_1 filename_2

Any **touch** command will create a file with empty content. This is the main drawback of **touch** command.

Examples,
1) $ touch test1.txt test2.txt
Once it **touch**, it updates the time to current system time.

2) $ touch test3.txt – If it's not there, it will create it.

3) $ touch test1.txt – it just touches the content & updates the time w.r.to system time.

A **touch** command is used to touch the contents. If the content is already present, it updates the time of the content to the current system date & time. If the content is not existing, it creates an empty file with the name of a content specified in the command syntax.

The **touch** command is a useful UNIX command for specific troubleshooting situations where it is unclear if a specified file is actually executing.

A command takes a maximum of 9 arguments.

The vi Editor

Vi stands for 'Visual Improved'

The **vi editor** has 2 modes,

- ➢ Insert mode – for editing
- ➢ Command mode – run commands of Vi editor

1) Open editor by using command **vi**
2) Go to **insert** mode, by pressing letter '**i**' on keyboard
3) Edit the file
4) Exit **insert** mode – press **esc** key on the keyboard
5) Go to **command** mode – press '**:**' key

- ✓ W – write(save)
- ✓ Q – quit (without save)
- ✓ X – exit(save & exit)

Until we give '**!**' character, it will not quit without save. Thus, we must give – **q!** to quit without save.

Vi is a powerful editing tool.
Running is in normal mode. It will not work in command mode & insert mode.

1) Copy a word
Yw – copies a word – from where the cursor is placed till the end of the word
P – paste

2yw – copies 2 words.
Y stands for **yank** (Greek word) which means copy

2) Copy a line
Yy – copies entire line
2yy – copies 2 entire lines

3) Delete word
Dw – deletes a word

4) Delete a line
Dd – deletes a line
2dd – deletes 2 entire lines

For navigation purposes in **normal** mode, we use keywords,
K – move cursor up
J – move cursor down
H – move cursor left
L – move cursor right

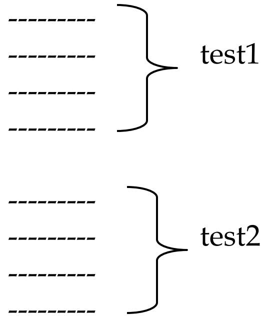
To copy a word, the cursor should be at the starting letter of the word. If the cursor is not at the starting character, the **yw** command copies the entire characters from the cursor location till the end of the word.
To copy an entire line, the cursor can be anywhere in the line.

Name: **cat**
Description: stands for **concatenate**. Displays the file contents on the screen

Syntax: **$ cat** filename

Examples,
1) $ cat test1.txt
Displays all the contents of test1

2) $ cat test1.txt test2.txt



We cannot make out which content belongs to which file.

3) $ cat –n test1.txt
Displays file content along with line numbers

A UNIX terminal is capable of displaying a maximum of 25 lines when a command is executed.

Differences between MORE & LESS command

MORE	**LESS**
1) Displays file contents **2)** Syntax **$ more** filename **3)** The **output** displays file line-by-line. Next lines are viewed by pressing **ENTER** key. Arrow keys will not work **4)** No backward navigation	**1)** Displays file contents **2)** Syntax **$ less** filename **3)** The next lines are viewed by using arrow keys or by **ENTER** key **4)** The file contents can be navigated forward & backward

Differences between HEAD & TAIL command

HEAD	**TAIL**
1) Displays topmost lines **2)** Syntax **$ head –n** filename n > = 1 (default 'n' value is 10) The output displays 'n' top-most lines **3)** Ex, **$ head -15 test1.txt** Displays top 15 lines	**1)** Displays bottom-most lines **2)** Syntax **$ tail –n** filename n >= 1 (default 'n' value is 10) The output displays 'n' bottommost lines **3)** Ex, **$ tail -15 test1.txt** Displays bottom 15 lines

---

PIPES

Pipes is an utility / mechanism used to run multiple commands at a time.

The syntax to use a pipe is given below,
$ cmd1 | cmd2 | cmd3

Here, an output of previous command will act as an input for next command
The final output of the syntax depends on the last command in the syntax.

To display between 50 & 60 lines in the page

```
        $ head -60 test1.txt | tail -11

                (OR)

        $ tail -51 test1.txt | head -11
```

We should never give the filename after the pipe symbol because the concept of pipes where the output of 1st command is input to the next command.
For ex,
$ head -60 nums.txt | tail -11 nums.txt -> this is an error.

Name: **rm**
Description: removes / deletes a file

Syntax: **$ rm** filename1 filename2

Example,
1) $rm test1.txt
Removes test1

2) $ rm test1.txt test2.txt
Removes test1 & test2

Name: **rmdir**
Description: removes a directory

Syntax: **$ rmdir** directory1 directory2

Examples,
1) $ rmdir class1
2) $ rmdir class1 class2

We cannot remove a directory which has some contents in it. We should 1st remove the contents before removing the directory.

$ rm –f test2.txt
It removes the file without confirmation message.
f – stands for **force remove**

Name: **cp**
Description: copies a file

Syntax: **$ cp source_file destination_file**

Examples,
1) $cp test1.txt test2.txt
If the destination file doesn't exist, it creates it – then copies the contents of source file to destination file.

2) $ cp test1.txt test2.txt
If test1 has 4 lines & test2 has 3 lines – then the contents of test1 will over-write over the contents of test2.

3) $ cp –b test1.txt test2.txt
It creates a back-up of test2 before over-writing it with test1 contents.
The back-up copy of test2 will look like this in the directory,
Test2.txt ~

Name: **mv**
Description: moves a file

Syntax: **$ mv source_file destination_file**

Examples,
1) $ mv test1.txt test2.txt
If destination file is not there, it creates it.
It moves the contents from source file to destination file –test1 to test2
It removes source file.

2) $ mv test1.txt test2.txt
If test2 already exists, it over-writes test2
Test1 is removed

3) $ mv test1.txt . . /class2 / test2.txt
It moves from one class to another.

4) $ mov test1.txt . . / class2 /
If no test1 is there in class2, it copies test1 from class1 to class2

A **mov** command acts as a rename command if the files are moved within the directory.
The same command acts as **mov** as well **rename** when the files are copied across the directory.

Regular Expression

Regular expression = Normal Character + Meta Character

Normal characters – include alpha-numeric characters

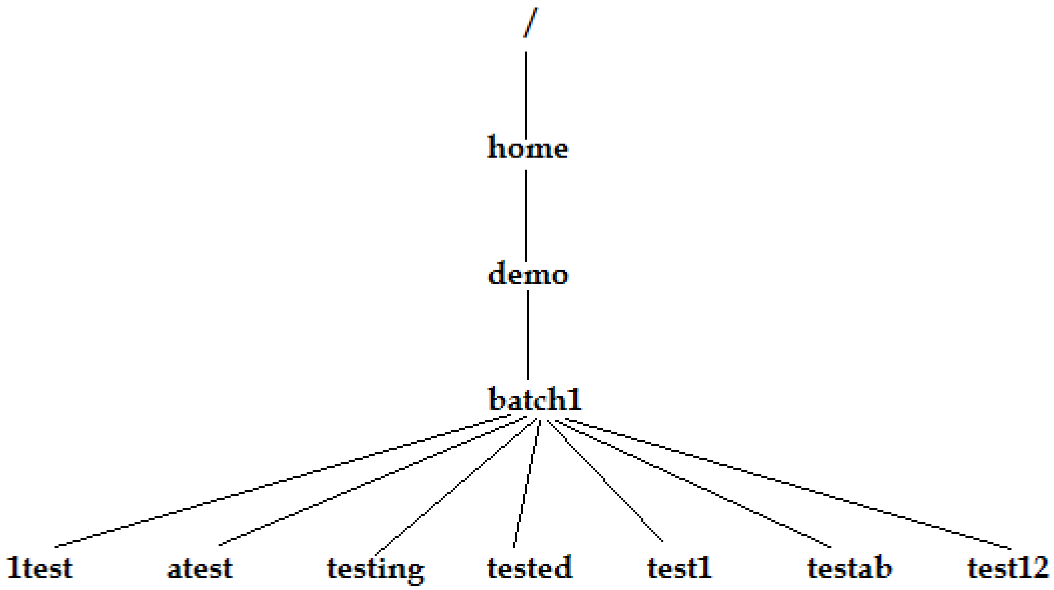
Meta characters – special characters like -> *, ?, . , \ , !

? – matches single character

*** - matches multiple characters (0 or maximum)**

[ ] – matches range or set of values. Ex – [a-z] or [0-9]

\ - escape characters



```
1) $ ls ?test
1test    attest
```

```
2) $ ls test?
Test1
```

```
3) $ ls test??
Testab          test12          tested
```

```
4) $ ls test[0 – 9]
Test1
```

```
5) $ ls test [5 -9]
No output
```

```
6) $ ls test*
Tested          test12          testab          testing          test1
```

```
7) $ ls *test* -> no output
```

Grep

Stands for **Global Regular Expression**

Grep scans its input for a pattern & can display the selected pattern, the line numbers or the filenames where the pattern occurs.

Syntax: **$ grep** pattern filename

Ex - $ grep flow test.txt

< - matches the characters at the staring of the word
> - matches a character at the end of the word

To search for the word Unix,
$ grep “\<unix\>” test.txt

1) –i (Ignoring case) -> when you look for a name, but are not sure of the case, **grep** offers the –i option which ignores the case for pattern matching.

2) –v (Deleting lines) -> this option selects all lines except the lines containing the pattern.

3) –n (Displaying line numbers) -> this option displays the line numbers containing the pattern along with the lines

4) –c (Counting lines containing patterns) -> this counts the number of lines containing the pattern.

5) –l (Displaying filenames) -> this option displays only the names of files containing the patterns.

Every UNIX command accepts an input through standard input device like keyboard & displays the output on a standard output device like terminal.

The process of changing the input as well as output of the commands is known as **redirection**

The process of making a command to display the output other than standard output device is known as **output redirection**. Denoted by >

The process of making a command to display input other than standard input device is known as **input redirection**. Denoted by <

PROCESS

A process is simply an instance of a running program.

A process is said to be born when the program starts execution & remains alive as long as the program is active. After execution is completed, the process is said to die.

Process ID (PID) -> each process is uniquely identified by a unique integer called **process id(PID)** that is allotted by the kernel when the process is born.

Name : **ps**
Description: displays the processes associated with a user at the terminal.

Syntax: **$ ps**
Each line shows the PID, the terminal with which the process is associated, the cumulative processor time that has been consumed since the process has started & the process name.

1) $ ps –l -> displays the detail of process

2) $ ps –lp -> displays all process

The 1st process which gets started is **init** & the PID is 1.

Kill -> kills the process. The **kill** command sends a signal usually with the intention of killing the process. **Kill** is an internal shell command in most shells.
The external **/bin/kill** is executed only when the shell lacks the kill capability.

$ kill 105 -> terminates the job having PID 105

Syntax of **kill** command: **$ kill –signal PID**
$ kill –l -> lists all signal numbers

<u>Killing the last background job</u>
For most shells, the system variable **$!** Stores the PID of the last background job. So we can kill the last background process without using the **ps** command,
S kill $!

<u>CHMOD</u>
The **chmod** (change mode) command is used to set the permissions of one or more files for all three categories of users (owner, group, others). It can be run only by the user (owner) & the superuser.
This command can be used in two ways,

- ❖ In a **relative** manner by specifying the changes to the current position
- ❖ In a **absolute** manner by specifying the final permissions

---

<u>Relative permissions</u>
Here, chmod only changes the permission specified in the command line & leaves other permissions unchanged.

The abbreviations used by **chmod**,

- ≈ u – user
- ≈ g – group
- ≈ o – others
- ≈ a – all (user, group, others)
- ≈ + - assigns permission
- ≈ - -> removes permission
- ≈ = -> assigns absolute permission

- ≈ r – read permission
- ≈ w – write permission
- ≈ x – execute permission

The syntax of **relative permissions** are,

Chmod category operation permission filename(s)

Where,

Category -> user, group, others, all
Operation -> +, -, =
Permission -> r, w, x

Examples,

1) $ chmod u+x xstart

It assigns **execute** permission to the user (owner) , but other permissions remain unchanged.

2) $ chmod ugo+x xstart

It assigns **execute** permission to all categories.

Absolute permission

The expression used by chmod here is a string of 3 octal numbers.

Each type of permission is assigned a number as shown below,

- ✓ Read permission – 4
- ✓ Write permission – 2
- ✓ Execute permission – 1

For ex, 6 represents read & write permissions. 7 represents all permissions.

Examples,

1) $ chmod 666 xstart

It assigns read & write permission to user, group & others

2) $ chmod 644 xstart

It assigns read&write permission to the user, read permission to group, read permission to others.

3) $ chmod 761 xstart

It assigns read, write, execute permission to user. Read & write permission to group. Execute permission to others.

EXTRAS

Date command

It displays both date & time.
$ date.
We cannot change the date & time displayed by this command.

Calendar command

It can be used to see the calendar of any specific month or a complete year. With **cal**, we can produce the calendar for any month or year between the years 1 and 9999.
$ cal 4 2003
It displays the calendar of April 2003.

Counting the number of lines in a file

$ wc list

```
o/p :   6         6         42        list
```

it says that the file **list** contains 6lines, 6 words & 42 characters.

Understanding a MAN page

A man page is divided into a number of compulsory & optional sections. Every command doesn't have all sections, but the 1st three – NAME, SYNOPSIS, DESCRIPTION are generally seen in all man pages.
NAME presents a 1 line introduction to the command
SYNOPSIS shows the syntax used by the command
DESCRIPTION provides a detailed description

f / spacebar -> advances the display by 1 screen of text at a time in MAN page
b – moves back 1 screen in MAN page

If the **logout** command fails to exit UNIX, then we use **exit** command or also **CTRL + d**.

Author:
Sachin Agarwal
Senior DevOps Engineer