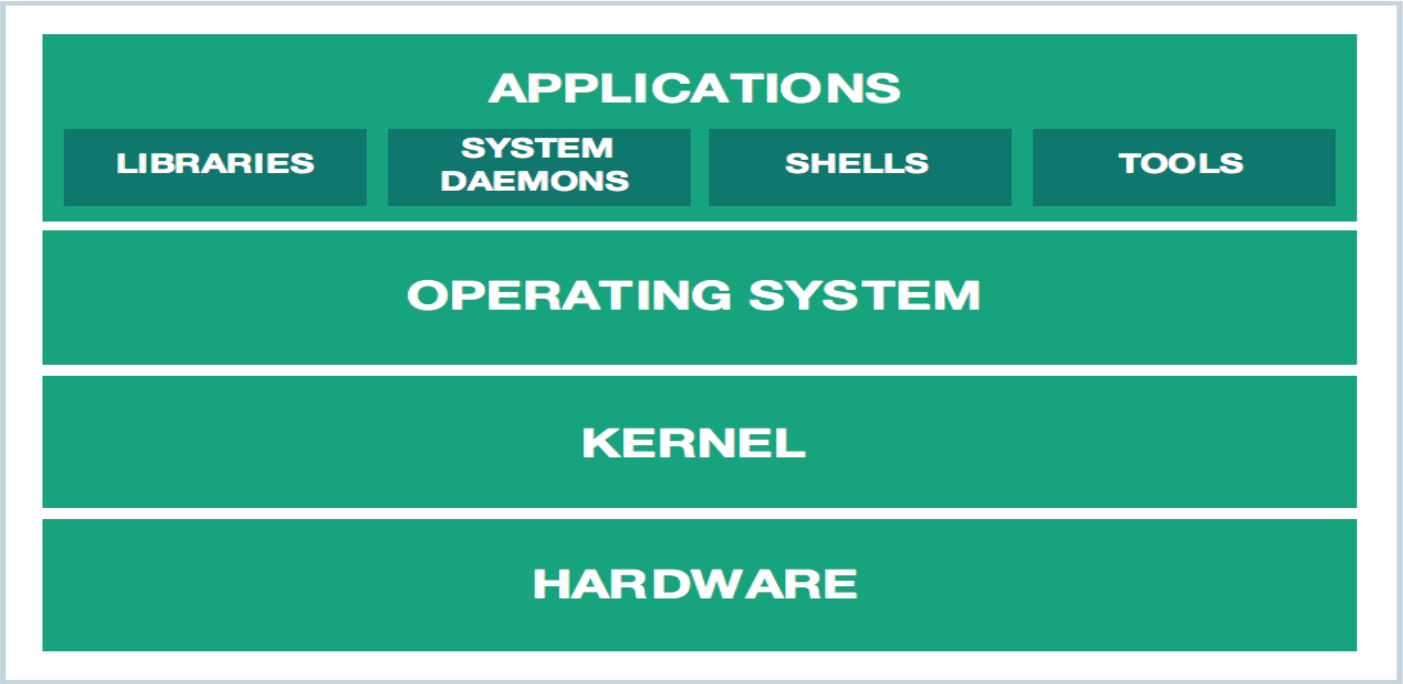
UNIX

Unix Architecture:



Command	Description	Flags
man	Displays information about any Linux command, including the commands used to start GroupWise programs.	
Users	List users in your machine	
whoami	Displays who you are logged in as.	
uname	Prints system info	a/r/m/v : all info/kernel-release/kernel-machine/version
Logout	To logout	
Cd	Change directory	~/- / ~username
Date	To display current date	
Cal	To display calendar	
Clear	It is used to clear the screen	
Ls	List files	-l / -a / -s / h / -r / -t
		*.doc
Cat *filename*	Displays content of file	
Wc *filename*	Word count file name	
Mkdir *dir_name*	Make directory	-p
Rmdir *dir_name*	Remove directory	
Cp *file1 file2*	Copy files	-r /-f
Mv file1 location	Move files	-r / -f

rm *filename*	Remove file	
Pwd	Current working directory	
Echo	Prints content on screen	
File	Finds a file in current working directory	

Going to home directory
Cd ~

Access modes in unix:

Read -> **r**
Write-> **w**
Execute-> **x**

Permissions in unix:

First 3 chars(2-4): permission for file owner
Second group of 3 chars(5-7): permission for group of users
Third group of 3 chars(8-10): permission for others

Types of permission:

U -> User
G -> Group
O -> Other

<u>Changing the file permissions:</u>

1	**+** Adds the designated permission(s) to a file or directory.
2	**-** Removes the designated permission(s) from a file or directory.

Chmod u+rwx filename
Chmod u-rwx filename
Chmod u+r filename

Chmod g+rx filename
Chmod o-x filename

```
chmod o+wx,u-x,g+rx filename
```

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

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

```
chmod 755 testfile

chmod 743 testfile

chmod 043 testfile
```

To check commands history

```
history

history -d line_number
```

Environment Variables

An important Unix concept is the **environment**, which is defined by environment variables. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

```
TEST="Unix Programming"
$echo $TEST
```

It produces the following result.

```
Unix Programming
```

Note that the environment variables are set without using the **$** sign but while accessing them we use the $ sign as prefix. These variables retain their values until we come out of the shell.

When you log in to the system, the shell undergoes a phase called **initialization** to set up the environment. This is usually a two-step process that involves the shell reading the following files −

- /etc/profile
- profile

The .profile File

The file **/etc/profile** is maintained by the system administrator of your Unix machine and contains shell initialization information required by all users on a system.

The file **.profile** is under your control. You can add as much shell customization information as you want to this file.

Setting the PATH

When you type any command on the command prompt, the shell has to locate the command before it can be executed.

The PATH variable specifies the locations in which the shell should look for commands. Usually the Path variable is set as follows −

```
$PATH=/bin:/usr/bin
```

Environment Variables set by System

HOME

Indicates the home directory of the current user: the default argument for the cd **built-in** command.

PATH

Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands.

PWD

Indicates the current working directory as set by the cd command.

RANDOM

Generates a random integer between 0 and 32,767 each time it is referenced.

UID

Expands to the numeric user ID of the current user, initialized at the shell startup.

Piping: Connecting one or more commands together

You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, put a **vertical bar (|)** on the command line between two commands.

GREP COMMAND

Used for searching a pattern, text, word ,etc.

Grep pattern filename

```
ls -l | grep "Aug"
```

Examples

grep ***ate*** file1.txt

- ➔ searches all occurrences of ***ate*** in file1.txt

grep ‘hello world’ file1.txt

- ➔ searches string the two words hello world in exact representation.

```
grep -n ate file1.txt
```

➔ searches all occurrences along with the line number where it is found.

GREP command is always case-sensitive. In order to make it search word or pattern in-case sensitive, make use of flag **-i**

```
grep -i ate  file1.txt
```

Displays all the matching occurrences in incase-sensitive manner.

```
Grep -v ate file1.txt
```

Display everything which doesn't contain pattern "**ate"**

```
Ls -l | grep -l "ate"
```

Displays all the filenames in which ate is found.

```
Grep -c "ate" file1.txt
```

Prints the count of matching lines.

Regular Expression or RegEX

- **^** – search at the beginning of the line.
- **$** – search at the end of the line.
- **.** – Search any character.

```
Grep "tech" file.txt
```

Returns all occurrences with tech present in it.

```
Grep ^tech file.txt
```

Returns all occurrences that begins with tech keyword.

```
Grep x$ file.txt
```

Returns all occurrences that ends with x .

```
Grep .n file.txt
```

Searches any word that has "n" in it.

<u>Sort command</u>

Usage:

Sort file1.txt

Displays content in sorted order.

Sort -r

Displays sorting in reverse order.

What are Programs ??

Program is basically a piece of code which contains a set of instruction written by a developer.

Program is meant for achieving a particular objective. For eg. Calculator in windows is a program, it is meant for performing calculation, etc.

What is a Process ???

A program under execution is called a process. For eg. When you tried out the **ls** command to list the directory contents, you started a process. LS is basically a program which contains instructions to list files and directories.

When a process is created it possess something called as PID(Process Identifier). A PID is always unique . No two process can have same PIDs.

Two nature of process

- ➔ Foreground
- ➔ Background

Process that directs the output directly on the screen is called a foreground process.

No other command is executable during foreground process.

Process that doesn't display output directly on the screen and process output in background is called a background process.

Can be done using nohup cmd &

<u>Listing running proces</u>

Ps

List running process

```
PID       TTY      TIME        CMD
18358     ttyp3    00:00:00    sh
18361     ttyp3    00:01:31    abiword
```

```
18789     ttyp3    00:00:00    ps
```

Ps -f

```
UID       PID  PPID C STIME    TTY   TIME CMD
root   6738 3662 0 10:23:03 pts/6 0:00 first_one
root   6739 3662 0 10:22:54 pts/6 0:00 second_one
root   6892 3662 4 10:51:50 pts/6 0:00 ps -f
```

1	**UID** User ID that this process belongs to (the person running it)
2	**PID** Process ID
3	**PPID** Parent process ID (the ID of the process that started it)
4	**C** CPU utilization of process
5	**STIME** Process start time
6	**TTY** Terminal type associated with the process
7	**TIME** CPU time taken by the process
8	**CMD** The command that started this process

Flags with PS

E -> Extended info (listing along with sys processes)

A -> Info about all users

F -> Full information represented with extra columns

Working with tar

```
tar -cf archive.tar foo bar  # Create archive.tar from files foo and bar.
tar -tvf archive.tar         # List all files in archive.tar verbosely.
tar -xf archive.tar          # Extract all files from archive.tar.
```

Directory structure.

1	**/** This is the root directory which should contain only the directories needed at the top level of the file structure
2	**/bin** This is where the executable files are located. These files are available to all users
3	**/dev** These are device drivers
4	**/etc** Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages
5	**/lib** Contains shared library files and sometimes other kernel-related files
6	**/boot** Contains files for booting the system
7	**/home**

	Contains the home directory for users and other accounts
8	**/mnt** Used to mount other temporary file systems, such as **cdrom** and **floppy** for the **CD-ROM** drive and **floppy diskette drive**, respectively
9	**/proc** Contains all processes marked as a file by **process number** or other information that is dynamic to the system
10	**/tmp** Holds temporary files used between system boots
11	**/usr** Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others
12	**/var** Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
13	**/sbin** Contains binary (executable) files, usually for system administration. For example, ***fdisk*** and ***ifconfig*** utlities
14	**/kernel** Contains kernel files