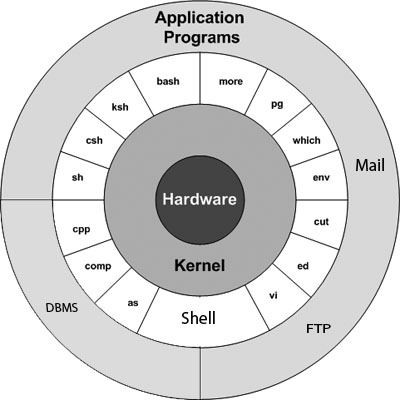
UNIX Architecture:

Here is a basic block diagram of a UNIX system:



The main concept that unites all versions of UNIX is the following four basics:

* **Kernel:**The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, scheduling and file management.
* **Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are most famous shells which are available with most of the UNIX variants.
* **Commands and Utilities:** There are various command and utilities which you would use in your day to day activities. **cp, mv, cat** and **grep** etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.
* **Files and Directories:** All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system

## To log in:

1. Have your userid (user identification) and password ready.
2. Type your userid at the login prompt, and then press ENTER. Your userid is case-sensitive, so be sure you type it exactly.
3. Type your password at the password prompt, and then press ENTER. Your password is also case-sensitive.
4. If you provided correct userid and password then you would be allowed to enter into the system.

You would be provided with a command prompt (sometime called $ prompt) where you would type commands.

#prompt -> when logged in as Administrator

$prompt -> when logged in as a user

1. ***logname*** -> displays current user name
2. ***hostname*** -> displays host name
   1. -i to display ip address (io) (/sbin/ifconfig -> to display internet ip address)
3. ***who*** -> displays all logged users; tells you who's logged on and login time
4. ***whoami*** -> displays current user; returns user name
5. ***uname*** -> uname prints system information

**-a** -> all information (-s -> kernel name, -r -> kernel release, -n -> node name, -v -> kernel version, -m -> machine hardware name, -p -> processor type or unknown, -o -> operating system, -I -> hardware platform)

1. ***exit*** -> logs out from current user
2. ***clear*** -> clear the terminal screen
3. ***pwd*** -> print name of current/working directory
4. ***passwd*** -> lets you change your password
5. ***date*** -> Displays current date time

Format: Mon Mar 21 15:32:20 EDT 2011

*date +%m -> displays month from the date*

%d -> day of months (1-31)

%m -> month in two digits

%y -> last two digits of year

%Y -> four digits year

%H -> hours (0..23) 24 hour format

%M -> Minutes

%S -> Seconds

%D -> Date; same as %m/%d/%y

%T -> only time part from current date

1. ***cal*** ->displays calendar for current month

eg: cal 01 2010 <displays jan-2010 calendar>

cal 2011 - displays 2011 year calendar

cal -3 -> displays calendar for previous, current and next months

1. ***printenv***

prints environment variable values

$ printenv EDITOR USERNAME ….. -> prints the values of environment variables listed

If no variables are specified, printenv prints the value of every environment vairable.

1. ***echo*** -> echo prints given string;

echo is also use to print the value of environment variable when specified with $;

eg: printenv EDITOR = echo $EDITOR

some of the environment variables are: HOME, USER, PATH, TERM, MAIL, EDITOR

$ echo My present working directory is `pwd` -> outputs

( ` -> Acute/Back quote symbol helps to concatenate the command output with text )

My present working directory is /home/hari

1. ***man*** -> $man echo -> provides help on unix command echo (displays manual)

* . files are hidden files and usually contain important program configuration information

1. ***ls*** -> displays files and directories order by name

ls -x -> width wise

ls -l -> displays long list of files and directories order by name

ls -r -> reverse order of file name

ls -t -> order by timestamp (latest first)

ls -lrt -> order by timestamp (oldest -> latest)

ls -a -> displays hidden files also

ls -R -> displays list of files and directories recursively (including sub directories contents)

ls \*.\* -> displays all files with any extensions

ls p\* -> displays only files starting with character ‘p’

ls a\* -> lists all the files starting with character 'a'

ls \*e -> lists files ending with character 'e'

ls -d \*/ -> displays only directories from present working directory

***Wild Card Characters***

-----------------------------

\* -> combination of zero or more characters

? - represents single character

[..] -> any character from the list

[!..] -> other than characters specified in the list

- -> used to specify range

1. ***chmod*** ->The chmod command (abbreviated from change mode) lets user change the permissions of files and directories.

There are many ways by which UNIX permission schemes are represented. The most common form is symbolic notation

ls -l prints symbolic notation for files and directories

The first character indicates the file type:

- denotes a regular file

d denotes a directory

b denotes a block special file

c denotes a character special file etc.,

Each class of permissions is represented by three characters

- The first set of characters represents the ***user class***

- The second set represents the ***group class***

- The third and final set of three characters represents the ***others class***

Each of the three characters represents the **read**, **write**, and **execute** permissions respectively

r if the read bit is set, **-** if it is not

w if the write bit is set, **-** if it is not

x if the execute bit is set, **-** if it is not

The following are some examples of symbolic notation:

-rwxr-xr-x for a regular file whose user class has full permissions and whose group and others classes have only the read and execute permissions

crw-rw-r-- for a character special file whose user and group classes have the read and write permissions and whose others class has only the read permission

dr-x------ for a directory whose user class has read and execute permissions and whose group and others classes have no permissions

u represents user

g represents group

o represents others

a represents all

r represents read

w represents write

x represents execute

+ -> to add a permission

- -> to remove permission

= -> to assign permission

chmod g=rwx sample.txt

chmod g=r\_x sample.txt

chmod ugo=rwx sample.txt

chmod g+rwx,u+rwx,o+rwx ***sample.txt***

Access rights on directories

r -> allows to list files in the directory

w -> user may delete files or move files into it

x -> right to access the files; If you have 'x' for the directory than you can read a file in it

r -> 4; w -> 2; x -> 1

octal digit values:

0 --- no permission

1 --x execute

2 -w- write

3 -wx write and execute

4 r-- read

5 r-x read and execute

6 rw- read and write

7 rwx read, write and execute

chmod 537 ***sample.txt*** to give read/execute to user, write/execute to group, read/write/execute to others

Usually UNIX administrator takes care of changing owner and group of a file

1. ***touch*** -> used to create new 0 byte file(s)

$touch file1.txt file2.txt -> creates two 0 byte files with names file1.txt and file2.txt

-t option in touch command is used to change the timestamp of an existing file or to create a new file with different timestamp

$ touch -t 201201081145 file1.txt -> file1.txt timestamp will be changed to Jan 8 2012 11:45am

$ touch –t 201202191345 newfile.txt -> newfile.txt created with Feb 19 2012 1:45pm timestamp

1. ***cat*** -> to display contents of an existing file or to create a new file

**Existing:**

$cat type1.txt -> displays contents of type1.txt

$cat type1.txt type2.txt type3.txt -> displays contents of type1.txt followed by contents of type2.txt followed by contents of type3.txt

**New:**

$cat > type3.txt

SCD type3 stores partial history

<Ctrl+d>

$cat type1.txt type2.txt type3.txt > type.txt -> contents of type1,type2,type3 will be stored in type.txt

$cat type1.txt >> abc.txt -> appends the contents of type1.txt to abc.txt

1. ***more*** -> displays page full of file content

enter -> next line; space -> next page; q -> quit;

1. ***mkdir*** -> to create new directory/directories

$mkdir dir1 dir2…. dirn

1. ***cd*** -> command used to change the directory

cd <dir> change directory

cd .. Change to parent directory

cd\ change to root directory

cd - will change prompt to previous path

tab -> will give full name of file if starts typing;

eg: type sr and then press tab -> displays all the files starting with sr

1. ***cp*** -> copies files and directories

cp <f1> <f2> -> copies f2 with same content as f1

cp -i file1 file2 -> if file2 already exists, then asks for confirmation to overwrite

cp -p -> to copy with same timestamp

1. ***rmdir*** -> command to remove an empty directory
2. ***rm*** -> removes files or directories

rm -i -> asks for confirmation before deleting

rm -r -> to remove directory with contents

rm\*.\* -> removes all files with extensions

1. ***mv*** -> to rename a file/directory

mv f1 f2 -> to rename a file f1 to f2

$mv sample .sample -> makes sample as hidden file

***>*** -> Output redirection

***>>*** -> append redirection output

***<*** -> input redirection

***;*** -> is a command separator – it helps to execute more than one commands separated by ;

**|** -> redirects the output to next command

1. ***head*** -> command used to display top ***n*** lines from an existing file

$ head sample.txt -> displays top 10 lines from file sample.txt

$ head -5 sample.txt -> displays top 5 lines/rows from file sample.txt

1. ***tail*** -> command used to display bottom n lines/rows from an existing file

tail student -> displays last 10 lines/rows from file student

tail -5 student -> displays last 5 lines/rows from file student

head -15 student | tail -5 -> displays from 11th to 15th records from file student

tail -f rel355load.log -> (-f follow) output appended data as file grows)

1. ***wc*** -> print the number of newlines, words, and bytes in files

wc -l filename -> print the newline count in file

wc -w filename -> displays word count in file

wc -c filename -> print the byte count in file

1. ***grep*** -> grep stands for -> globally search regular expression and print

grep 'abc' sample.txt -> displays all lines having abc from file sample.txt

-w -> searches for 'abc' as whole word and not part of another word

-v -> displays non-matching lines

-i -> ignores case when searching

-r -> searches in all files under folder recursively

-n -> prints line numbers as well

-c -> to print only the total count of matched lines

-o -> Show only the part of a matching line that matches search pattern

-l -> list files having search string

-L -> list files not having search string

^ -> to specify line starting with specific pattern

grep '^abc' sample.txt displays lines that starts with abc in file sample.txt

$ -> to specify line ending with specific pattern

grep 'abc$' sample.txt displays lines that ends with abc

grep '^abc$' sample.txt displays lines that contains only abc

grep '^$' sample.txt displays blank lines

grep '^..$' sample.txt displays lines with only two characters

ls -l | grep '^d' -> displays directories from present working directory

ls -l | grep '^-' -> displays regular files from present working directory

1. ***find*** -> The find command searches through the contents of one or more directories, including all of their sub directories. You have to tell find in which directory to start its search;

find . -name abc.txt -> searches for files with name abc.txt under current directory including sub directories

$find -newer file1 -> displays files newer than file1

$find . -name 'h\*' -> displays files starting with h in current directory including sub-directories

$ find . -name 'D\*' -type f -> displays ***only fi***

***les*** starting with D

$ find /home/hari/ITA -type f -name "h\*" -> searches for **only files** starting with h in /home/hari/ITA including sub directories

$ find /home/hari -path /home/hari/ITA1 -prune -o -type f -name hello -> displays all the files with name hello found in /home/hari **excluding** /home/hari/ITA1

$ find /home/hari -type d -name "IT\*" -> displays only directories starting with IT in /home/hari and sub directories

$ find . \( -name "\*.txt" -o -name "\*.dat" \) -type f -ls -> searches for files with extension **.txt** or **.dat** and displays as long list

find . -mmin -4 -print -> displays all the files/directories modified within last 4minutes

find . -mmin +4 -print -> displays all the files/directories modified atleast 4minutes back

-mmin -> modified -amin -> accessed

find . -mtime +30 -ls -> lists all the files/directories modified atleast 30days back

find . -mtime +30 -prints -> prints files/directories modified atleast 30days back

$find . -size 0 -> displays 0 byte files along with path starting current folder

$find . -empty -type f -print -> to print only empty files

$find . -empty -type d -print -> to print only empty directories

$find . -empty -print -> to print empty files and directories

$find . -size +500c -print -> prints all the files with more than 500 characters in size

1. ***cut*** -> Print selected parts of lines from each FILE to standard output. Popular usage is to get required columns from a delimited file.

-c -> select only these characters

-d -> to specify delimiter

-f -> to specify field list

-s -> do not print lines not containing delimiters

$ -> prints 2nd, 4th to 5th characters from file student to standard output

$cut -d ';' -f 1,3 filename -> cuts columns 1 and 3 from file (delimiter is semi colon) - which is specified using -d; if you do not specify any delimiter - unix assumes delimiter is tab

1. ***sort*** -> sort command prints the content of a text file in sorted order. 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.

-r -> outputs in reverse order

-n -> option makes the program sort according to numerical value (eg: if we have 1001, 110 normal sort will display 1001 first and then 110, but when used -n, 110 is displayed first and then 1001)

-t -> to specify delimiter when sorting a delimited text file

-k -> to specify a key (or range of keys) to sort

-f -> to ignore case

-c -> check whether input is sorted; do not sort (output not displayed when input is sorted; else print the location of disorder)

$sort file1 > file2 -> sorts the contents of file1 and redirects the output into file2

sort -k4 <filename> -> sorts the contents of file with respect to 4th column

sort -k4 -k2 <filename> -> sorts the contents of file by 4th column and then by 2nd column

sort file1 –o sfile -> sorts files file1 and redirects the output to sfile instead of standard output

$ sort -t'|' -k2 <filename> -> to sort a pipe-delimited file based on second column

1. ***uniq*** -> uniq is a Unix utility which, when fed a text file, outputs the file with adjacent identical lines collapsed to one. It is a kind of filter program. Typically it is used after sort. It can also output only the duplicate lines, or add the number of occurrences of each line etc.,

\*provide sorted input\*

uniq filename -> outputs file content with adjacent identical lines collapsed to one.

uniq -c filename -> also adds number of occurences of each line (-d -u are ignored when used -c)

uniq -u filename -> displays only non-duplicate rows

uniq -d filename -> displays only duplicated rows

-i -> ignore case when comparing lines

eg: sort filename | uniq -ui

-f ignores number of fields (A field is a run of whitespace, then non-whitespace characters.

sort -k4 uniq\_demo1 | uniq -f3 -> sorts file based on 4th column and redirects the output to uniq -f3 command (uniq -f3 ignores first 3 columns and displays the output considering only data after 3 cols)

-s -> skips number of characters in a line

-w -> compare no more than specified characters in each line.

1. ***cmp*** -> compares both the files and gives no output in case both the files are equal. It returns the location of first difference when files are unequal
2. ***diff*** -> this command displays difference between two files ; If there are no differences between the files, you will see no output, but if two text files are indeed different, all the text mismatches will be highlighted using the standard diff output:

d (deleted) -> denotes record is not present in file2

c (changed) -> denotes record in file1 is different from file2

a (added) -> denotes record is present in file2

< denotes content from file1

> denotes content from file2

< will be displayed in case of d

< and > will be displayed in case of c

> will be displayed in case of a

1. ***comm*** -> compare two sorted files line by line and produces three-column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files.

-1 suppress lines unique to FILE1

-2 suppress lines unique to FILE2

-3 suppress lines that appear in both files

1. ***gzip*** -> reduces the size of the named files and each file is replaced by one with the extension .gz, while keeping the same ownership modes, access and modification times, gzip will only attempt to compress regular files.

$ gzip sample -> produces file sample.gz

1. ***gunzip*** -> gunzip takes a list of files on its command line and replaces each file whose name ends with .gz

$ gunzip sample.gz -> uncompresses sample.gz and produces file sample

1. ***zcat*** -> Once file has been compressed, we cannot view it using the normal cat command. Unix provides a utility zcat to view the contents of a compressed/zipped file.

$ zcat sample.gz -> displays file contents in readable format

1. ***sleep*** -> sleep helps to delay for a specified amount of time

sleep 20 -> delays for 20 seconds which is same as sleep 20s

also we can use m or h m-> minutes, h-> hours

eg: sleep 3m -> delays for 3 minutes

1. ***ps*** -> ps selects all processes with the same effective user ID as the current user and associated with the same terminal as the invoker. It displays the process ID (pid=PID), the terminal associated with the process (tname=TTY), the cumulated CPU time in [dd-]hh:mm:ss format (time=TIME), and the executable name (ucmd=CMD). Output is unsorted by default.

ps -e -> to display all processes

ps -u <username> -> lists processes related to specific user

1. ***kill*** -> to terminate a process

kill PID -> kills process with specified PID

kill -9 <jobid> -> force kill the process

1. ***fg*** -> command helps to get the back ground process to foreground

$ fg % processid -> gets process to foreground

1. ***bg*** -> Resume each suspended job in the background

command helps you to change the foreground process to background process, once we start any process in foreground we can press ctrl+z to stop it and once you see $ prompt we can type bg which resumes that process as background process

1. nohup -> nohup is a command to ignore the HUP (hangup) signal, enabling the command to keep running after the user who issues the command has logged out. nohup is most often used to run commands in the background. Output that would normally go to the terminal goes to a file called nohup.out if it has not already been redirected. This command is very helpful when there is a need to run numerous batch jobs which are inter-dependent.

nohup is a low-level utility simply configuring a command to ignore a signal

**eg:**

$ vi a.sh

#!/bin/bash

echo " Starting Script"

sleep 15

echo " in middle of script"

sleep 15

echo "done"

a.sh takes atleast 30 seconds to complete

$ nohup sh a.sh& (executing a.sh as background process)

exit (immediately)

Output of the above job is appended to existing nohup.out file or will create a new file nohup.out and redirect the output of process/job.

Once you exit from the terminal and then re-login and check nohup.out - you can see that process got completed though we logged out without waiting for job completion.

1. ***mail*** -> mail is a command line email client for Unix and Unix-like operating systems

-s -> lets you give subject (eg: -s "Hello")

-c -> lets you give email address to be cc'ed (if more, separate by comma)

-b -> lets you give email address to be bcc'ed (if more, separate by comma)

-a -> lets you send file as an attachment

$ mail -s "Test Email" -c carboncopy@test.com -b blindcc@test.com recepient@test.com abc.txt

above command will

send an email to recepient@test.com

cc to carboncopy@test.com

bcc to blindcc@test.com

subject -> Test Email

body of the email -> contents of file abc.txt

to send abc.txt as attachment -> please specify -a abc.txt

1. ***split*** -> helps to output fixed-size multiple files of a given input file

Default size is 1000 lines and default PREFIX is 'x'

Assume sample has 2384 lines

$ split sample -> generates 3 files

xaa (with 1000 lines); xab (with 1000 lines); xac (with 384 lines)

$ split -d sample -> generates 3 files

x00 (with 1000 lines); x01 (with 1000 lines); x02 (with 384 lines)

$ split sample samp-> generates 3 files

sampaa (with 1000 lines); sampab (with 1000 lines); sampac (with 384 lines)

-b1000 -> outputs files each with 1000 bytes size

-l500 -> l stands for lines -> outputs files each with 500 lines

-a -> use suffixes of length N (default 2)

1. ***ftp*** -> FTP is the user interface to standard File Transfer Protocol. The program allows a user to transfer files to and from a remote host/network site.
2. ***lcd*** -> Change the working directory on the local machine
3. ***lpwd / !pwd*** -> display current working directory on local machine
4. ***put*** -> Store a local file on the remote machine
5. ***get*** -> Retrieve the remote-file and store it on the local machine
6. ***bye*** -> Terminate the FTP session with the remote server and exit ftp
7. ***sed*** -> stream editor

sed -n 10p numbers.txt -> prints 10th line

sed -n 10,20p numbers.txt -> prints line 10 to 20 in numbers.txt

sed 5d numbers.txt -> prints file numbers.txt except line 5

sed s/abc/def/ abc.txt -> replaces abc with def in file abc.txt (but only first occurance in each line - replaces in all lines but only first occurance)

sed s/is/was/g today -> replaces every occurance of 'is' with 'was' globally in file today and displays as output - it will not replace the contents of the original file

sed "s/A/a/g;s/N/n/g" dept -> A is replaced with a and N is replaced with n in a file dept

Please use '>' and new file name to redirect the output to a new file - as original file content is not changed using sed

s -> stands for substitute

g -> stands for global

/ -> delimiting symbol

I can create a file called changes.txt with below content

s/a/A/g

s/b/B/g

s/is/was/g

and use $ sed -f changes.txt today

displays today file contents with all the changes - but 'a' in 'was' is not displayed as capital letter

sed '/^$/d' today -> deletes empty lines from file today and displays - no changes made to actual file

sed '/ /d' today -> deletes any line with space and displays

sed '/e$/d' today -> deletes line(s) ending with e and displays

sed '/^ \*$/d' today -> blank lines or lines with spaces only are deleted in display

sed '/is\*/d' today -> deletes lines with occurance of is

$echo day | sed 's/d/(&)/' -> outputs (d)ay; & represents search string

***vi editor commands***

esc -> command mode

i -> enter into insert mode - can start inserting characters

shift + i -> move to beginning of the line and enters into insert mode

a -> enter into insert mode - can start inserting characters from next character to cursor

A -> start appending at end of the line

x -> to delete a character under cursor

X -> removes character prior to cursor (which works like backspace)

o -> insert an empty line below the current line

O (shift + o) -> insert an empty line above the current line

cc -> copy and clear text in current line without deleting the line

yy -> copy current line

y$ -> copy from cursor till end of the line (including the character under cursor)

y0 (zero) -> copy from beginning of the line till cursor (including the character under cursor)

yw -> copy from cursor till beginning of next word (including the character under cursor)

yb -> copy from beginning of the word till cursor (excluding the character under cursor)

p -> paste copied line below the current line

P (shift + p) -> paste copied line above the current line

dd -> delete current line (5dd deletes five lines including current line and following 4 lines)

dw -> deletes current word in forward direction

db -> delete current word on backward direction

d$ -> deletes current line from cursor position till end of the line

d^ -> deletes current line from beginning of the line till character before cursor(current) position

h -> moves cursor left

j -> moves cursor down

k -> moves cursor up

l -> moves cursor right

:w -> to save a file

:q -> to quit

:q! -> force quit (quit without saving)

:wq -> save and quit

:= -> total number of lines in file displayed at bottom the screen

:.= -> current line number

:set nu -> sets the line numbers for a file

:set nonu -> removes line numbers

H -> moves to top of the screen

M -> moves to middle of the screen

L -> moves to bottom of the screen

G -> Moves to last line in the current file (end of file)

:w file1 -> save as file1 -> to save current file as file1 -> without removing original file

/string -> searches for a string in forward direction

?string -> searches for a string in reverse direction

n -> next occurrence in search direction

N -> previous occurrence in search direction