Operating System:

An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a important component of the system software in a computer system. Application programs require an operating system to function.

Time-sharing operating systems use schedule tasks for efficient use of the system and correct use of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware. Operating systems can be found on almost any device that contains a computer—from cellular phones and video game consoles to supercomputers and web servers.

Functions of OS:

1). Resource Manager: The OS manages all the Resources that are attached to the system like memory and Processor and all the Input output Devices.

2). Storage Management: Operating System also controls all the Storage Operations which helps the data or files to be stored into the computers and how the files will be accessed by the users, creation of files, creation of directories and reading and writing the data of files and directories and also copy the contents of the files and the directories from one place to another place.

3). Process Management : The Operating System also controls the Process of the computer that is all the Processes those are given by the user or the Process those are System ‘s own Process are Handled by the Operating System . The Operating System create the Priorities for the user and also Start or Stops the Execution of the Process and Also Makes the Child Process after dividing the Large Processes into the Small Processes.

4). Memory Management: Operating System also manages the memory of the Computer System that is it provides the memory to the Process and Also de-allocate the memory from the process. It also defines that if a process gets completed then this will de-allocate the memory from the processes.

UNIX operating system (History):

The first version of UNIX was created in 1969 by Kenneth Thompson and Dennis Ritchie, system engineers at AT&T's Bell Labs. It went through many revisions and gained in popularity until 1977, when it was first made commercially available by Interactive Systems Corporation. The University of California at Berkeley was working to improve UNIX. In 1977 it released the first Berkeley Software Distribution, which became known as BSD. This time unix was known to have C- shell. The AT&T version was developing in different ways. The 1978 release of Version 7 included the Bourne Shell for the first time. By 1983 commercial interest was growing and Sun Microsystems produced a UNIX workstation called System V. It was directly taken from the original AT&T UNIX and it is most widely used version. There are two main versions of UNIX in use now-days System V and BSD. System V is the more popular of the two. The differences are in the structure of the file system or in behavior of the commands.

The version of UNIX like operating system now used is LINUX. Linux is a free open source UNIX OS for PCs that was originally developed in 1991 by Linus Torvalds. Linux is neither pure SYS-V nor pure BSD. It combines some features from each (e.g. SYSV-style startup files but BSD-style file system layout) and is based on a set of IEEE standards called POSIX (Portable Operating System Interface). To maximize code portability, it typically supports SYSV, BSD and POSIX system calls (e.g. poll, select, memset, memcpy, bzero and bcopy are all supported).

The open source nature of Linux means that the source code for the Linux kernel is freely available so that anyone can add features and correct deficiencies or errors. The open source approach has not just successfully been applied to kernel code, but also to application programs for Linux. As Linux has

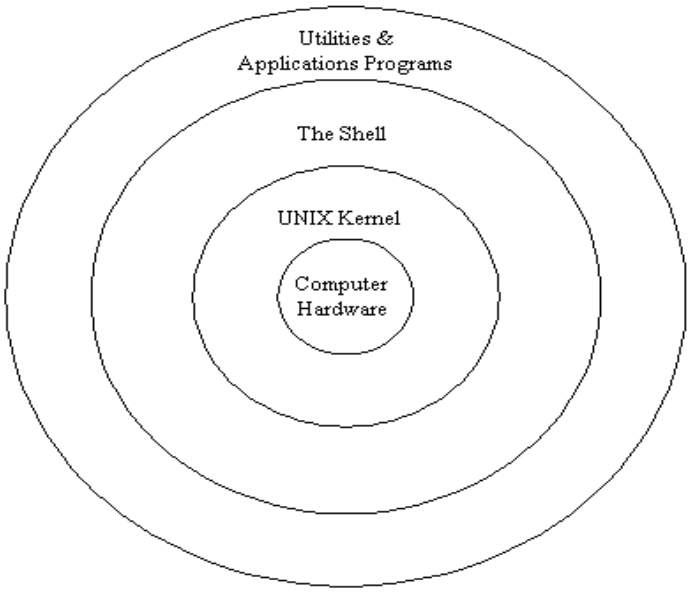
become more popular, several different development distributions have emerged, e.g. Redhat, Mandrake, Debian, and Caldera. A distribution comprises a prepackaged kernel, system utilities, GUI interfaces and application programs.
Redhat is the most popular distribution because it has been ported to a large number of hardware platforms (including Intel, Alpha, and SPARC), it is easy to use and install and it comes with a comprehensive set of utilities and applications including the X Windows graphics system, GNOME and KDE GUI environments, and the StarOffice suite (an open source MS-Office clone for Linux).

Features of UNIX OS:

The main features of UNIX include multiuser, multitasking and portability. Multiple users can access the system by connecting to points known as terminals. Several users can run multiple programs or processes simultaneously on one system. UNIX uses a high-level language that is easy to modify and transfer to other machines, which means you can change language codes according to the requirements of new hardware on your computer & so, have the flexibility to choose any hardware, modify the UNIX codes accordingly and use UNIX across multiple architectures.

The hub of a UNIX operating system, the kernel manages the applications and peripherals on a system. Together, the kernel and the shell carry out your requests and commands. You communicate with your system through the UNIX shell, which translates to the kernel. The UNIX shell is a program that gives and displays your prompts and, in combination with the kernel, executes your commands. The shell even maintains a history of the commands you enter, allowing you to reuse a command by scrolling through your history of commands.

All the functions in UNIX involve either a file or a process. Processes are executions of programs. The files are collections of data created by you. Files may include a document, programming instructions for the system or a directory. UNIX uses a hierarchical file structure.
So the UNIX operating system has three important features; a kernel, the shell and a filesystem.



KERNEL: The kernel is at the core of each UNIX system and is loaded in whenever the system is started up or the system boot.
It manages the entire resources of the system. The functions performed by the kernel are:

- managing the machine's memory and allocating it to each process.
- scheduling the work done by the CPU so that the work of each user is carried out as efficiently as is possible.
- organising the transfer of data from one part of the machine to another.
- accepting instructions from the shell and carrying them out.
- enforcing the access permissions that are in force on the file system.

SHELL: Whenever we login to a UNIX system we are in contact with a program called the shell. It is represented as a prompt at the bottom left of the screen. To perform any task the user enter commands at this prompt.

The shell acts as a command interpreter; it takes each command and passes it to the operating system kernel which works on it. It then displays the results of this operation on the screen.

The shell provides you with one or more of the following features like : create an environment, write shell scripts, define command aliases, manipulate the command history, automatically complete the command line, edit the command line.

The different types of Shell available in UNIX OS are:

1). Bourne shell (sh): This is the original Unix shell written by Steve Bourne of Bell Labs. It is available on all UNIX systems. This shell does not have the interactive facilites provided by modern shells such as the C shell and Korn shell. The Bourne shell does provide an easy to use language with which you can write shell scripts.

2). C shell (csh): This shell was written at the University of California, Berkeley. It provides a C-like language with which to write shell scripts.

3). TC shell (tcsh): This shell is available in the public domain. It provides all the features of the C shell together with emacs style editing of the command line.

4). Korn shell (ksh): This shell was written by David Korn of Bell labs. It is now provided as the standard shell on Unix systems. It provides all the features of the C and TC shells together with a shell programming language similar to that of the original Bourne shell. It is the most efficient shell.

5). Bourne Again Shell (bash): This is a public domain shell written by the Free Software Foundation under their GNU initiative. It is a full implementation of the IEEE Posix Shell and Tools specification. This shell is widely used within the academic community. Bash provides all the interactive features of the C shell (csh) and the Korn shell (ksh). Its programming language is compatible with the Bourne shell (sh).

Entering shell commands

The basic form of a Unix command is: commandname [-options] [arguments]

The command name is the name of the program you want the shell to execute. The command options, indicated by a dash, allow you to perform different condition of the same command. The arguments are the names of files, directories, or programs that the command needs to access. The square brackets ([and]) signify optional parts of the command that may be not be necessary to type.

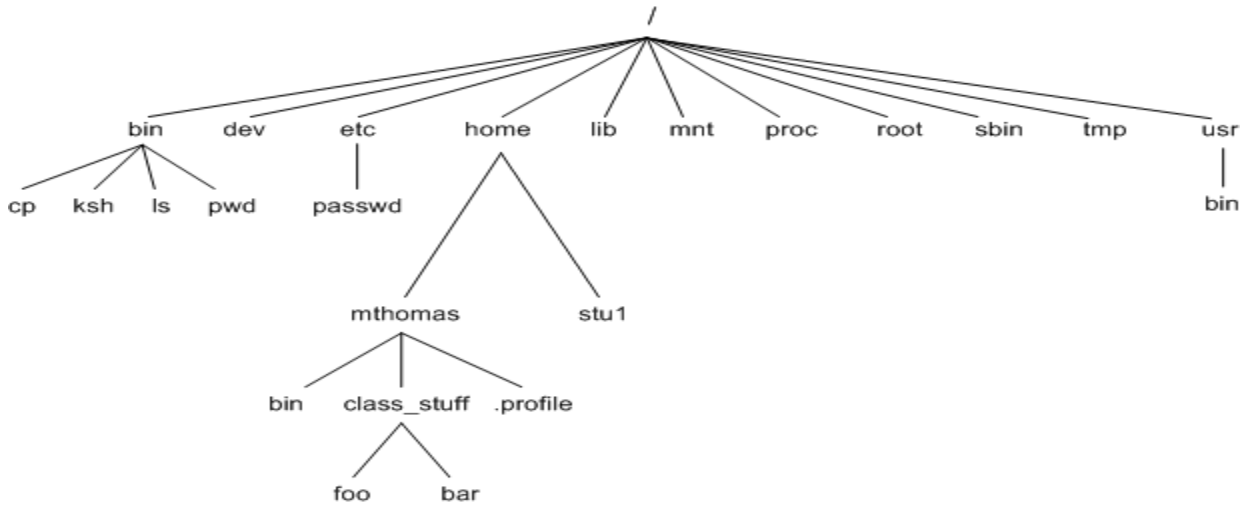
UNIX File System:

A file system is a logical method for organising and storing large amounts of information in a way which makes it easy to manage. The file is the smallest unit in which information is stored. The UNIX file system has several important features.

Structure of UNIX File System:

The Unix file system has a hierarchical (or tree-like) structure with its highest level directory called root (denoted by /, *slash*). Just below the root level directory are several subdirectories, most of which contain system files. Below this different system files, application files, and/or user data files are present. It follows the concept of the process parent-child relationship, all files on a Unix system are related to one another. That is, files also have a parent-child existence. Thus, all files (except one) share a common parental link, the top-most file (i.e. /) do not have any parent link.

The diagram shows a "typical" Unix file system. The the top-most directory is / (slash), with the directories directly below are the system directories. The files can vary of different vendors of UNIX but the basic structure remains same.



The following directories are present in most UNIX file Systems:

- *bin* - short for binaries, this is the directory where many commonly used executable commands reside /are present.
- *dev* - contains device specific files
- *etc* - contains system configuration files
- *home* - contains user directories and files
- *lib* - contains all library files
- *mnt* - contains device files related to mounted devices
- *proc* - contains files related to system processes
- *root* - the root users' home directory (note this is different than /)
- *sbin* - system binary files reside here. If there is no sbin directory on your system, these files most likely reside in etc
- *tmp* - storage for temporary files which are periodically removed from the filesystem
- *usr* - also contains executable commands

The UNIX filesystem contains several types of file.

1). Ordinary files: This type of file is used to store your information, such as some text or image. The user is the owner of the file created by him & the user can set access permissions to control which other users can have access to them. Any file is always contained in a directory.

2). Directories: A directory is a file that holds other files and other directories. Directories can be created in the home directory to hold files and other sub-directories. It gives a specified location or place to the user to work or to perform task. The user is the OWNER of the directories created by him & can set access permissions to control which other users can have access to the information they contain.

3). Special files (Device files): This type of file is used to represent a real physical device such as a printer, tape drive or terminal. To use any device like a file allows you to send the output of a command to a device in the same way as you send it to a file. For example:

```
cat scream.au > /dev/audio
```

This sends the contents of the sound file scream.au to the file /dev/audio which represents the audio device attached to the system. The directory /dev contain the special files which are used to represent devices on a UNIX system.

4). Pipes: UNIX allows you to link commands together using a *pipe*. The pipe acts as a temporary file which only exists to hold data from one command until it is read by another.

File Names

- UNIX permits file names which have characters, but spaces, tabs and characters that have a special meaning to the shell should not be used. Such as:
  & ; ( ) | ? \ ' " ` [ ] { } < > $ - ! /
- UNIX filenames/directory names are Case Sensitive. That is uppercase/lowercase are not same.
- Length: can be up to 256 characters
- Extensions: may be used to identify types of files
  libc.a - *archive, library file*
  program.c - *C language source file*
  alpha2.f - *FORTRAN source file*
  xwd2ps.o - *Object/executable code*
  mygames.Z - *Compressed file*

Hidden Files have names that begin with a dot (.) For example:

.cshrc .login .mailrc .mwmrc

No two files with the same parent directory can have the same name. Files located in separate directories can have identical names.

- Reserved Filenames:
  / - *the root directory (slash)*
  . - *current directory (period)*
  .. - *parent directory (double period)*
  ~ - *your home directory (tilde)*

Home Directory

A UNIX system can have many users on it at a time. All users are given a home directory in which the user is placed or resides when the user Login into the system. User's home directories are grouped together under a system directory such as /home.

Pathnames

Every file and directory in the file system can be identified by a complete list of the names of the directories that are on the route or path or way from the root directory to that file or directory.

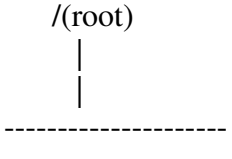
Absolute Pathnames

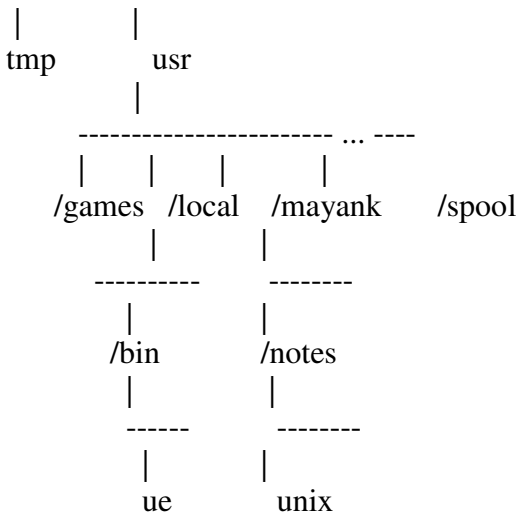
Each directory name on the route is separated by a / (forward slash). For ex:

/usr/local/bin/ue

/usr/mayank/notes/unix

This gives the full pathname or absolute pathname starting at the root directory and going down through the directories usr, local and bin to the file ue .





Relative pathnames

The Relative pathname is opposite to Absolute pathname. A relative pathname does not begin with a / (slash). Any user can define a file or directory by its location in relation to his/her current directory. The pathname is given as / (slash) separated list of the directories on the route to the file (or directory) from your current directory. A .. (dot dot) is used to represent the directory immediately above the current directory(one level up).

In all shells except the Bourne shell, the ~ (tilde) character can be used for the full pathname to the home directory.

Examples of using relative pathnames

1). To give a relative path to a file below your current directory:

```
# pwd (prints the current directory)
# /home/BCA/mayank  (the full pathname to the current directory)
# ls -l  notes/docs/unix.txt (lists information  on the file)
```

The above command defines the file unix.txt in the directory docs which is in the directory notes which itself is in the current directory specified by pwd command.

2). To define a file that is in another directory:

```
# pwd (prints the current directory)
# /home/BCA/mayank/notes/docs     (the full pathname)
# cd ../DBMS  (use relative pathname)
# pwd (check the current directory)
  /home/BCA/mayank/notes/DBMS
```

The user mayank uses the relative pathname ../DBMS to move up to the directory notes and then down into the directory DBMS.

Login Unix: When you first connect to a UNIX system, you see a prompt:

login:

After entering the userid, the next prompt is as follows:

password:

After providing password, the user gets the command prompt assigned by the shell.

The various command prompts(shell prompts) are:

a). bourne shell ➔ $
b). C- shell ➔ %
c). tcsh shell ➔ %
d). korn shell ➔ $
e). bash shell ➔ $
The root who is superuser or administrator for the unix system has the shell prompt as # character.
a). $pwd (print working directory)
This command prints or displays the current working directory
Directory & File Handling Commands:
1). cal
The cal(calendar) command is used to display the calendar for the current or present month.
Syntax: $ cal or $ cal -1
2). whoami
This command displays the username or userid of the currently logged-in user.
Syntax: $ whoami
3). who
This command displays the details of all the users logged-in into the unix system at the same time. The same function can be performed by users or w commands.
Syntax: $ who or $ users or $ w
4). ls
The ls(list) command list or displays all the files & directories in the current directory.
Syntax: $ ls [options] [directories]
$ls –l (displays long list with complete information of files & directories that is mode, links, owner, group, size, time of last modification, and name)
$ ls –a (display all the entries including hidden files starting with .)
$ ls –i (displays inode numbers)
$ ls –F (displays directory with / sign behind directory name)
5). cat
The cat (concatenate) command is used to view the contents of the files. If used with redirection operator it can take different forms.
Syntax:
$ cat >FILENAME (creates a file)
$ cat FILENAME (view contents of the file).
$ cat >>FILENAME (used to append contents to a file)
$ cat FILE1 FILE2 >FILE3 (used to combine the contents of file1 & file2 into file3. If file3 not exist it creates & combines the contents).
6). mkdir
The command mkdir(make directory) is used to create a single or multiple directories.
Syntax:
$ mkdir [options] [directories]
$ mkdir DIRECTORYNAME
$ mkdir DIR1 DIR2 DIR3 (creates multiple directories)
$mkdir –p dir1/dir2 (used to create parent directories if it does not exist)
7). cd or chdir
This command cd (change directoiry) is used to change the current working directory.

Syntax:

$ cd [options] [dirname or path]

$ cd dirname (changes the location to specified directory, in the current directory).

$ cd ../dirname (changes the location to specified path which is not in current directory. The .. specifies the parent directory)

$ cd ➔ (changes the location to the home directory)

$ cd / (changes the location to root's directory that is /(slash)).

$ cd .. (changes the location up by one level i.e to the parent directory of the current directory)

8). rmdir

This command is used to remove or delete directories but the directories must be empty. It can also remove multiple directories.

Syntax: $ rmdir [options] directoryname

$ rmdir directoryname (removes or deletes only empty directory)

$ rmdir –p DIR1/DIR2/DIR3 (removes nested directories that is directory within directory)

$ rm –rf DIR1 (removes directories with files or sub-directories in it).

9). rm

This command is used to remove or delete files or directories

Syntax: $rm [options] filename

$ rm FILENAME

$ rm –i FILENAME (Interactive: it prompts the user for confirmation before deleting a file).

$ rm –f FILENAME (forcefully: it delete the file without confirmation)

$ rm –r FILENAME (recursive: it delete non-empty directories i.e sub-directories & files).

10). cp

This command is used to copy a file or directory from one location to another location.

Syntax: $cp [options] source file destination file

$ cp FILE1 FILE2 (copies contents of FILE1 to FILE2, if FILE2 already has contents it would be overwritten)

$ cp fig2 part2/figure2 (**To copy a file from your current working directory to a subdirectory. This copies the file fig2 from your current working directory to the file figure2 in the subdirectory part2.)**

$ cp –i FILE1 FILE2 (Interactive mode: It prompts before overwriting)

$ cp –v FILE1 FILE2 (Verbose: Explains what is being done)

$ cp –R DIR1 DIR2 (copies DIR1 directory to DIR2directory)

$cp *.doc DIR1 (copies all the files with extensions .doc to directory DIR1)

11). mv

This command is used to move a file or directory from one location to another or it renames a file if in current directory.

Syntax: $ mv [options] source filename destination filename

Options:

f: Do not prompt before overwriting a file.

i : Prompts for the user input before overwriting a file.

$ mv oldname newname (renames the oldname by newname in the current directory)

$ mv olddirname newdirname (renames the old directory name to new directory name)

$ mv –i oldname newname (if newname already exist then it is overwritten by the file oldname, so to avoid overwriting –i option is used)

$ mv dir1 /dir2/ (moves the dir1 directory into dir2 directory)

Unix File & Directory Permissions(Access control in unix)

Unix file and directory permission is in the form of a 3×3 structure. i.e Three permissions (read, write and execute) available for three types of users (owner, groups and others).
In the output of ls -l command, the 9 characters from 2nd to 10th position represents the permissions for the 3 types of users.

Three file permissions:

- read: permitted to read the contents of file.
- write: permitted to write to the file.
- execute: permitted to execute the file as a program/script.

Three directory permissions:

- read: permitted to read the contents of directory (view files and sub-directories in that directory).
- write: permitted to write in to the directory. (create files and sub-directories in that directory)
- execute: permitted to enter into that directory.

Numeric values for the read, write and execute permissions:

- read 4
- write 2
- execute 1

To have combination of permissions, add required numbers. For example, for read and write permission, it is 4+2 = 6.

Changing File and Directory Permissions Using Chmod Command:

The octal representation or symbolic representation is used to change the permission of a file or directory.

Octal representation for permissions:

- First number is for user
- Second number is for group
- Third number is for others

$ chmod 644 filename (gives read, write (4+2 = 6) to user and read (4) to group and others).
$ chmod 540 filename (gives read, execute (4 + 1 = 5) to user and read (4) to group, and nothing (0) to others).
$ chmod 604 filename (gives read, write (4 + 2 = 6) to user and nothing (0) to group, and read (4) to others).

Umask :

Umask is responsible for the default permission of a file. The user file-creation mode mask (umask) is used to determine the file permission for newly created files. It can be used to control the default file permission for new files. It is a four-digit octal number. A umask can be set or expressed using:

- Symbolic values
- Octal values

The default umask value is 0022, which decides the default permission for a new file or directory. Default permission for a directory is 0777, for files the permissions are 0666 from which the default umask value 0022 is deducted to get the newly created files or directory permission.
Final default permission for a file is calculated as shown below:

- Default file permission: 666
- Default umask : 022
- Final default file permission: 644

Final default permission for a directory is calculated as shown below:

- Default directory permission: 777
- Default umask: 022
- Final default directory permission: 755

You can change the umask value to correct or as needed value based upon the above calculation. For example, if you don't want anybody other than the user (owner) to do anything on the file or directory then you can give umask as 0077.

$ umask 0077

After this, if you create a file or directory, it will have permissions only for the user as shown below:

$ ls -l testfile

-rw------- 1 root root 2 Mar 17 08:23 testfile

Symbolic Representation: The symbolic representation of three different roles/account:

- u is for user,
- g is for group,
- o is for others.

Following are the symbolic representation of three different permissions:

- r is for read permission,
- w is for write permission,
- x is for execute permission.

Examples to use the symbolic representation on chmod.

1. Add single permission to a file/directory: "+" symbol means adding permission. To give execute permission for the user:

$ chmod u+x filename

2. Add multiple permission to a file/directory: Use comma to separate the multiple permission sets

$ chmod u+r, g+x filename

3. Remove permission from a file/directory: Removes read and write permission for the user.

$ chmod u-rx filename

4. Change permission for all roles on a file/directory: Assigns execute permission to user, group and others (anyone can execute this file).

$ chmod a+x filename

5. Apply the permission to all the files under a directory recursively: Option -R to change the permission recursively.

$ chmod -R 755 directory-name/

6. Change execute permission only on the directories (files are not affected)

On a particular directory if you have multiple sub-directories and files, the following command will assign execute permission only to all the sub-directories in the current directory (not the files in the current directory). If the files has execute permission already for either the group or others, the above command will assign the execute permission to the user.

$ chmod u+X *

Examples

1. To add a type of permission to several files:

$ chmod g+w chap1 chap2 (This adds write permission for group members to the files chap1 and chap2).

2. To make several permission changes at once:

$ chmod go-w+x mydir (This denies group members and others the permission to create or delete files in mydir (go-w) and allows group members and others to search mydir or use it in a path name (go+x)).
This command can also be mentioned as:
chmod g-w mydir
chmod o-w mydir
chmod g+x mydir
chmod o+x mydir

3. To permit only the owner to use a shell procedure as a command:

chmod u=rwx,go= cmd
This gives read, write, and execute permission to the user who owns the file (u=rwx). It also denies the group and others the permission to access cmd in any way (go=). If you have permission to execute the cmd shell command file, then you can run it by entering:

cmd

4. To use the absolute mode form of the chmod command:

$ chmod 644 text (This sets read and write permission for the owner, and it sets read-only mode for the group and others.)
5). To recursively descend directories and change file and directory permissions given the tree structure:
./dir1/dir2/file1

./dir1/dir2/file2
./dir1/file1
enter this command sequence:
chmod -R 777 f*
which will change permissions on ./dir1/file1.
But given the tree structure of:
./dir1/fdir2/file1
./dir1/fdir2/file2
./dir1/file3
the command sequence:
chmod -R 777 f*
will change permissions on:
./dir1/fdir2
./dir1/fdir2/file1
./dir1/fdir2/file2
./dir1/file3

A command is part of the shell itself.
A Utility is an executable program that is installed (usually in a system directory such as /bin) which the shell executes. UNIX utilities are commands that, generally, perform a single task. It may be as simple as printing the date and time, or a complex as finding files that match many criteria throughout a directory hierarchy.
Commands can be combined through pipelines that connect the standard output of one utility to the standard input of another.
Some utilities are built into the shell. Many UNIX utilities are sh – shell, cp – copy, ln – link one file to another, ls – list files or directory contents, mv – move or rename one or more files, rm – remove files, pr – print files with pagination or in columns, tr – translate characters, wc – word count, lp – line printer, du – disk usage, ftp – file transfer protocol, mt – magnetic tape, ed – edit file, vi – visual editor, sed – stream editor, grep – global regular expression and print, bash – Bourne again shell, tar – tape archive, find – search a file hierarchy for files that match various criteria, date – set or print the system date, mail – read and/or send electronic mail, more – file page wise, tail – display the last lines (tail) of file

Utility commands(text processing) commands in UNIX:
1). more: The *more* command writes the file onto the screen one page at a time. After each page, it stops and waits for you to tell it what to do. It may also print the name of the file and the percent of the file it has read, or maybe it will display the prompt *more*.
On pressing the *[space]* bar another *page* (screen) is displayed & so on, until the file's end. To scroll through the file one line at a time, we can press the *[enter]* key in place of the *[space]* bar. To quit before reaching the end of the file, we can press the *q* key.
That is we can say that more command displays text one screen at a time.
Syntax: $ more FILENAME
$ more ~/profile
2). less: The less command is same as more command, but less allows both forward and backward movements. This command doesn't require to load the whole file before viewing. The screen navigation commands while viewing large log files.

- CTRL+F – forward one window
- CTRL+B – backward one window
- CTRL+D – forward half window
- CTRL+U – backward half window

Syntax: $ less FILENAME
$ less install.log (will display a screen at a time)
3). file: The file command displays the contents of a given file, whether it is a text (Ascii) or binary file.
Syntax: $ file FILENAME
4). wc: (word count): This command prints the number of lines, words and characters (bytes) in the given input.
Syntax: $ wc FILENAME (displays lines words characters)
Options: -c output character count only, -l output line count only, -L output length of longest line, -w output word count only

5). cmp: This command is used to compare any two files. The cmp utility compares two files of any type and writes the results to the standard output. By default, cmp is does not return any value if the files are the same. If they are different, then the byte and line number at which the first difference is present is displayed at the output. Bytes and lines are numbered beginning with one.
Syntax: cmp [options] FILE1 FILE2
6). diff: The diff command will compare the two files and print out the differences between.
Syntax: $ diff FILE1 FILE2
For Ex:
Contents of file1 are
This is first file
this is second line
this is third line
this is different as;lkdjf
this is not different
Contents of file2 contains
This is first file
this is second line
this is third line
this is different xxxxxxxas;lkdjf
this is not different
$ diff file1 file2
4c4
< this is different as;lkdjf

> this is different xxxxxxxas;lkdjf
7). lp: Used to print the file
$ lp FILENAME
8). banner: Used to print any text written with the command in the form of ascii art poster. It doesn't work in linux(bash) shell.
Syntax: $ banner TEXT
9). tty: The tty command will display your terminal.
Syntax: $ tty
10). ln: This command does not copy a file but make links to existing files.
Syntax: $ ln FILENAME
If you want to create a link to a file called coolfile in /usr/local/bin directory then you can enter this command.
$ ln mycoolfile /usr/local/bin/coolfile
11). comm:
Selects or rejects lines common to two sorted files.
Syntax
comm [-1 -2 -3] File1 File2
The comm command reads the File1 and File2 parameters and writes, by default, a three-column output to standard output. The columns consist of:

Lines that are only in File1
Lines that are only in File2
Lines that are in both File1 and File2.
Both File1 and File2 should be sorted
-1 Suppresses the display of the first column (lines in File1).
-2 Suppresses the display of the second column (lines in File2).
-3 Suppresses the display of the third column (lines common to File1 and File2).

The Bourne Shell

The Bourne shell is a type of Unix shells (C shell, Bash shell etc.). It is both a command language and a programming language. A shell is frequently called a command interpreter. As a command language it provides a user interface to Unix/Linux. It executes commands entered by the user or from a file.

Files containing commands allow users to create commands according to need. Such files are called: shell scripts, shell programs, or command files. The Bourne shell was the default UNIX shell of Unix Version 7. Most Unix-like systems continue to have /bin/sh which is the Bourne shell. It was developed by Stephen Bourne at Bell Labs. It was released in 1977 in the Version 7 Unix release.

As a programming language, each shell provides I/O variables, conditionals loops. Each shell (Bourne, C, Bash etc) has its own syntax. Shell programs are typically used to develop "User-friendly" commands, System administration utilities & Application utilities. System administrators can build shell programs to add new users, to shutdown the system etc.

The Bourne shell is an interactive command interpreter and command programming language. The shell carries out commands specified at the terminal or from a file. Your default shell is executed automatically upon login by the login command. The /etc/passwd file contains the default shell for each user. Initially the shell invoked by the login command executes commands found in the /etc/profile file and then in your $HOME/.profile file if one exists. The shell then accepts commands from either the command line or a file.

Executing Commands

When a command is issued through the shell sh, the command is evaluated and all substitutions (variables and aliases) are made. If the evaluated command matches a shell special command or a defined function, it is executed. If the command matches a name of an executable (binary) file, the shell (the parent) spawns a new (child) process which runs the binary program. If the command matches the name of a text file marked executable, the shell assumes that it is a shell procedure. To execute this procedure, the shell spawns a sub-shell that executes the commands specified in the file.

By default, the shell will search for external commands based on the value of the PATH environment variable. If the command can not be found in any of the directories specified by the path, it searches the current directory. If you specify a path with the command, the shell will not search the path for the specified command. Instead, it only searches the path specified on the command line.

Shell Commands

The Bourne shell is a programmable shell with several forms of structured commands.

a). The for command lets you execute a specified list of commands. The syntax for this command is:

for identifier [in word ...] do list done

where identifier is a variable assigned the value(s) specified by word ... and executes the commands specified by list. If the word option is not specified, the shell executes the list of commands for each positional parameter that is set.

b). The case command can be used to execute commands based on a particular setting of another variable. The syntax for this command is:
case word in pattern [| pattern] ...) list;; [pattern [|pattern]...) list;;] esac
where word is the variable to match with one of the specified patterns. When a matching pattern is found, the commands specified by list are executed. The vertical bar is used to denote an "or" operation.
c). if construct to specify conditions in the script. The syntax for this command is:
if List then list [elif List] ... [else list] fi
where the commands specified by list are executed if the last command executed by List has a return value of zero. The elif construct represents an else if phrase and the corresponding list commands are executed if the previous List returns a non zero value and the last command executed by the List following the elif phrase returns a value of zero. The else clause is executed only if all other conditions are return non-zero values.
d). while clause can be used to execute a list of commands while a certain condition holds true. The syntax for the while command is:
while List do list done
where the commands specified by list are executed after the last command specified by the List variable returns a zero value. The shell will continue to execute the commands until the last command specified in List returns a non-zero value.
e). until command works like the while command except the return values are reversed. This means the commands in list are executed as long as the last command specified by List returns a non-zero value. This repeats until the last command in List returns a zero value.
Commands executed within parentheses are run within a subshell. To execute commands within the current shell, enclose them in braces ('{' and '}').
Special Commands
The following table shows a listing of the special commands to the Bourne shell. These commands are built into the Bourne shell and are executed in the shell process.

Command	Description
:	The null command returns a zero exit value
. file	Executes commands specified in file using the search path and not starting a subshell
break [n]	Exits the nth iteration from the for, while, or until loop
continue [n]	Resumes the nth iteration of the for, while, or until loop
cd [directory]	Changes to the specified directory, or to $HOME if none specified
echo [string]	Writes string to standard output
eval [list]	Executes the commands specified in list
exec [list]	Executes the commands in list in place of the current shell
exit [n]	Causes the shell to exit with a return code of n
export [name]	Marks the specified name for export to the environment of subsequent commands.
hash [-r][command]	Finds and remembers the path of each specified command or if -r is used, forget location
pwd	Displays the current working directory
read [name ...]	Reads one line from standard input and assigns to name
readonly [name...]	Marks name to be read-only

return [n] Causes a function to return a value of n
set [flag[argument] See the section on Setting Flags
shift [n] Shifts the command line arguments to the left n places ($0 is never shifted)
test Expression Evaluates the expression
times Displays the accumulated user and system times for processes running from the shell
trap [command][n] Runs command when the shell receives a signal n
type [name...] For each name specified, indicates how the shell would interpret it as a command
ulimit [flag[var]] Sets or queries process size limits using a specified flag
umask [nnn] Sets the access mode mask to be used with newly created files.
unset [name...] Removes the specified variable, name from the environment
wait [n] Waits for the child process whose process number is n or all if none specified