

## MODULE 2: File System and Basic File attributes.

---

UNIX system has thousands of files. If you write a program, you add one more file to the system. When you compile it you add some more. Files grow rapidly, and if they are not organized properly, you will find it difficult to locate them. So UNIX has a file system (UFS) to manage or organizes its own files in directory.

### 1. UNIX FILES AND BASIC FILE TYPES/CATEGORIES

#### FILES SUPPORTED BY UNIX FILE SYSTEM

File is a collection of records. So, files are divided into three categories

**a. Ordinary file**

**b. Directory file**

**c. Device file**

The UNIX file system contains several different types of files:

#### **a. Ordinary Files or regular files**

It contains only data as a stream of characters.

An ordinary files itself divided into 2 types

**Text file:** contains only printable characters, and you can often view the contents and make sense out of them. All C and Java files are example of text file. A text file contains lines of characters where every line is terminated with the newline character, also known as **linefeed** (LF) when you press Enter while inserting text, the LF character is appended to every line. You won't see this character normally, but there is command (od) which can make it visible.

**Binary file:** it contains both printable and unprintable characters that cover the entire ASCII range(0 to 255).most UNIX commands are example of binary files.

#### **b. Directory files**

i. Contains no data, but keeps some details of the files and subdirectories that it contains.

ii. A directory file contains an entry for every file and sub directory that it houses. Each entry has two components

- The filename
- A unique Identification number for the file or directory( called the inode number)

iii. A directory contains the filename but not the contents of file.

iv. When you create or remove a file the kernel automatically updates its corresponding directory by adding or removing the enter i.e inode number associated with that file.

#### **c. Device files**

i. Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations

ii. Unix considers any device attached to the system to be a file - including your terminal:

iii. By default, a command treats your terminal as the standard input file (stdin) from which to read its input

iv. Your terminal is also treated as the standard output file (stdout) to which a command's output is sent.

v. stdin and stdout will be discussed in more detail later

vi. Two types of I/O: character and block

vii. Usually only found under directories named /dev

## 2. NAMING FILES OR WHAT'S IN A (FILE) NAME

a. UNIX permits file names to use most characters, but avoid spaces, tabs and characters that have a special meaning to the shell, such as:

**& ; ( ) | ? \ ' " ` [ ] { } < > \$ - ! /**

b. It is recommended that only the following characters be used in filenames.

c. Alphabetic characters and numerals

d. The period(.), the hyphen(-) and underscore(\_)

e. Case Sensitivity: uppercase and lowercase are not the same! These are three different files:

NOVEMBER    November    november

f. Length: can be up to 256 characters

g. Extensions: may be used to identify types of files

libc.a     - *archive, library file*

program.c - *C language source file*

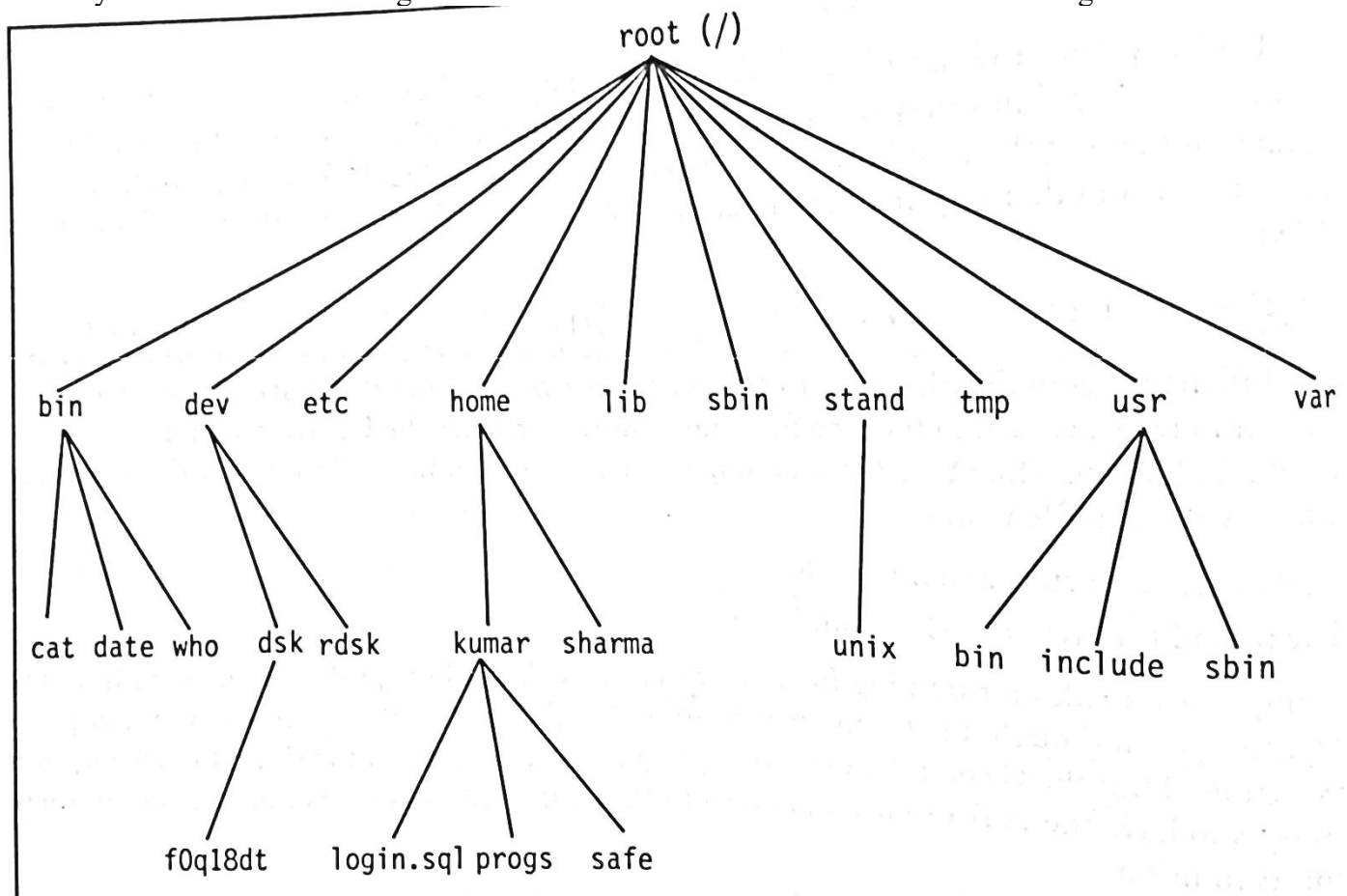
alpha2.f   - *Fortran source file*

xwd2ps.o   - *Object/executable code*

mygames.Z   - *Compressed file*

## 3. PARENT CHILD RELATIONSHIP/ UNIX FILE SYSTEM

All files in UNIX are related to one another. The file system in unix is a collection of all ordinary, directory and device files and organized in a hierarchical structure as shown in below fig.



The implicit feature of every UNIX file system is that there is a top which serves as reference point for all files. This top is called **root** & is represented by a /(front slash). Root is actually a directory. The root

directory has a number of sub directories under it. These sub directories in turn have more sub directories and others files under them.

For instance bin and usr are two directories directly under root, while a second bin and kumar are sub directories under usr.

Every file apart from root must have a parent. Thus the home directory is the parent of kumar, while / is the parent of home and grandparent of kumar. If you create a file login.sql under the kumar directory, then kumar will be the parent of this file.

The first group contains the files that are made available during system installation

- **/bin and /usr/bin:** these are the directories where all the commonly used UNIX commands are found.
- **/sbin and /usr/sbin:** If there's a command that you can't execute but the system administrator can execute, it would be probably in one of these directories.
- **/etc:** this directory contains the configuration files of the system. You can change a very important aspect of system functioning by editing a text file in this directory. Your login name and password are stored in files /etc/passwd and etc/shadow
- **/dev:** This directory contains all device files. These files don't occupy space on disk. there could be more sub directories like pts, disk and rdisk in this directory
- **/lib and /usr/lib:** Contains shared library files and sometimes other kernel-related files.
- **/usr and /include:** contains the standard header files used by C programs. The statement `#include<stdio.h>` used in most C programs refers to the file stdio.h in this directory.
- **/usr/share/man:** this is where the man pages are stored. There are separate subdirectories here (like man1, man2 etc) that contains the pages for each section. For instance, the man page of **ls** can be found in /usr/share/man/man1

User also work with their own files, they write programs, send and receive mail and also create temporary files. These files are available in the second group shown below

- **/tmp:** the directory where users are allowed to create temporary files. These files are wiped away regularly by the system
- **/var:** The variable part of the file system. Contains all your print jobs and your outgoing and incoming mail.
- **/home:** On many systems users are housed here. Kumar would have his home directory in /home/kumar

#### 4. THE HOME VARIABLE and THE PATH VARIABLE

**HOME DIRECTORY :** When log on to the system, UNIX automatically places you in a directory called the **home directory**.

- It is created by the system when user account is opened.
- If you log in using the login name sharma, you will land up in a directory that could have the pathname  
**/home/sharma**

- The shell variable HOME known's yours home directory

**\$echo \$HOME**

**/home/sharma**

You will be doing much of your work in your home directory and subdirectories.

- Home variable: it is also called environment variables. **Environment variables** are a set of dynamic named values that can affect the way running processes will behave on a computer.

- Here **\$HOME** is an environment variable it indicates the home directory of the current user: the default argument for the cd built-in command.

## 5. PATH VARIABLE:

- The PATH environment variable is a colon-delimited list of directories that your [shell](#) searches through when you enter a command.
- Program files (executables) are kept in many different places on the [Unix](#) system. Your path tells the Unix shell where to look on the system when you request a particular program.
- To find out what your path is, at the Unix shell prompt `echo $PATH`
- Your path will look something like the following.

```
/usr2/username/bin:/usr/local/bin:/usr/bin:.
```

You will see your username in place of username. Using the above example path, if you enter the `ls` command, your shell will look for the appropriate executable file in the following order: first, it would look through the directory `/usr2/username/bin`, then `/usr/local/bin`, then `/usr/bin`, and finally the local directory, indicated by the `.` (a period).

## 6.DIRECTORY COMMANDS – PWD, CD, MKDIR, RMDIR COMMANDS

### 6.1 pwd (PRINT WORKING DIRECTORY) (checking your current directory)

As the name states, command '**pwd**' prints the current working directory or simply the directory user is, at present. It prints the current directory name with the complete path starting from root (`/`). This command is built in shell command and is available on most of the shell – bash, Bourne shell, ksh, zsh, etc.

#### Basic syntax

**\$pwd [option]**

Options	Description
<b>-L</b> (logical)	Use PWD from environment, even if it contains symbolic links
<b>-P</b> (physical)	Avoid all symbolic links
<b>-help</b>	Display this help and exit
<b>-version</b>	Output version information and exit

If both '**-L**' and '**-P**' options are used, option '**L**' is taken into priority. If no option is specified at the prompt, `pwd` will avoid all symlinks, i.e., take option '**-P**' into account.

---

### 6.2 cd: CHANGING THE CURRENT DIRECTORY

- The **cd** command, which stands for "change directory", changes the shell's current working directory.
- The **cd** command is one of the commands you will use the most at the command line in UNIX.
- It allows you to change your working directory. You use it to move around within the hierarchy of your file system.

Ex 6.2.1 When used with an argument it changes the current directory to the directory specified as argument for example assume `gmit` is a directory under user directory `Kumar`. To change from `Kumar` directory to `gmit` directory, issue the command as follows

```
$pwd
/home/kumar
$cd gmit
$pwd
/home/kumar/gmit
```

Ex 6.2.2: When `cd` used without arguments: `cd` when used without arguments reverts to home directory

```
$pwd
/home/kumar/gmit
$cd
```

cd without argument will change directory from gmit to its home directory Kumar

```
$pwd
/home/kumar
```

Ex 6.2.3: If your present working directory is /home/Kumar and you need to switch to /bin directory directly, use absolute pathname i.e /bin wd cd command

```
$pwd
/home/kumar
$cd /bin
$pwd
/bin
```

---

### 6.3 mkdir: "making directory".

- **mkdir** is used to create directories on a file system.
- If the specified *DIRECTORY* does not already exist, **mkdir** creates it.
- More than one *DIRECTORY* may be specified when calling **mkdir**.

**mkdir syntax**

**mkdir [OPTION ...] DIRECTORY ...**

Ex 6.3.1: To create a directory named gmit, issue the following command.

```
$mkdir gmit
```

gmit directory is created under present working directory.  
Assume that pwd is /home/kumar , then gmit directory is created under kumar directory.

Ex 6.3.2: To create three directories at a time, named patch, dbs, doc, pass directory names as arguments.

```
$mkdir patch dbs doc
```

Ex 6.3.3: To create a directory tree:

To create a directory named gmit and create two subdirectories named cse and ise under gmit, issue the command. gmit is a parent directory.

```
$mkdir parent directory sub-directories
$mkdir gmit gmit/cse gmit/ise
```

Ex 6.3.4: Error while creating a directory tree

```
$mkdir gmit/cse gmit/ise
```

mkdir: Failed to make a directory "gmit/cse"; no such file or directory  
mkdir: Failed to make a directory "gmit/ise"; no such file or directory

---

Error is due to the fact that the parent directory named gmit is not created before creating sub directories cse and ise.

#### Ex 6.3.5: **\$mkdir test**

mkdir: Failed to make directory “test”; Permission denied.

This can happen due to:

- a. The directory named test may already exist
- b. There may be an ordinary file by the same name in the current directory.
- c. The permissions set for the current directory do not permit the creation of files and directories by the user.

---

## 6.4 rmdir: REMOVING DIRECTORIES

The **rmdir** utility removes the directory entry specified by each directory [argument](#), provided the directory is empty.

### Ex 6.4.1: **\$rmdir progs**

removes the directory named progs

Arguments are processed in the order given. To remove both a [parent](#) directory and a [subdirectory](#) of that parent, the subdirectory must be specified first, so the parent directory is empty when **rmdir** tries to remove it.

The reverse logic of mkdir is applied.

<b>\$rmdir</b>	<b>subdirectories</b>	<b>parent directory</b>
<b>\$rmdir</b>	<b>gmit/cse gmit/ise</b>	<b>gmit</b>

- You cant delete a directory with rmdir unless it is empty. In this example gmit directory cannot be removed until the sub directories cse and ise are removed.
- You cant remove a sub directory unless you are place in a directory which is hierarchically above the one you have chosen to remove.

## 6.5 ABSOLUTE PATHNAMES:

- If the first character of a pathname is / the files location must be determined with respect to root(/) . Such a pathname is called absolute pathname.  

cat /home/kumar
- When you have more than one / in a pathname for such / you have to descend one level in the file system. Thus Kumar is one level below home and two levels below root.
- When you specify a file y using frontslashes to demarcate the various levels, you have a mechanism of identifying a file uniquely. No two files in a UNIX system can have same absolute pathnames.
- When you specify the date command, the system has to locate the file date from a list of directories specified in the PATH variable and then execute it.
- However if you know the location of a command in prior, for example date is usually located in /bin or /usr/bin . Use absolute pathname i.e precede its name with complete path  

\$/bin/date

For example if you need to execute program **less** residing in /usr/local/bin you need to enter the absolute pathname

\$/usr/local/bin/less

## 6.6 RELATIVE PATHNAMES

- Pathnames that don't begin with / specify locations relative to your current working directory.
- Uses either the current or parent directory as reference and specifies path relative to it.
- A relative pathname uses one of these cryptic symbols.
  - . (a single dot) → this represents the current directory.
  - .. (two dots) → this represents the parent directory

Command	Function
<code>cd</code>	Returns you to your login directory
<code>cd ~</code>	Also returns you to your login directory
<code>cd /</code>	Takes you to the entire system's root directory
<code>cd /root</code>	Takes you to the home directory of the root or superuser, account created at installation, you must be root user to access this directory.
<code>cd /home</code>	Takes you to the home directory where user login directories are usually stored
<code>cd ..</code>	Moves you up one directory
<code>cd ~otheruser</code>	Takes you to the otheruser's login directory
<code>cd /dir/subdirfoo</code>	Regardless of which directory you are in, the absolute path takes you directly to subdirfoo, a subdirectory of dir.

Ex 6.6.1: Assume the current directory is `/home/kumar/progs/data/text`, using `cd ..` will move one level up

```
$pwd
/home/kumar/progs/data/text
$ cd ..
$pwd
/home/kumar/progs/data
```

Ex 6.6.2 : To move two levels up

```
$pwd
/home/kumar/progs
$ cd ../../
$pwd
/home
```

**Ex 6.6.3:** My present location is `/etc/samba` and now I want to change directory to `/etc`.

Using relative path: `$ cd ..`

Using absolute path: `$cd /etc`

**Ex 6.6.4:** My present location is `/var/ftp/` and I want to change the location to `/var/log`

Using relative path: `cd ../log`

Using absolute path: `cd /var/log`

**Ex 6.6.5:** My present location is `/etc/lvm` and I want to change my location to `/opt/oradba`

Using relative path: `cd ../../opt/oradba`

Using absolute path: `cd /opt/oradba`

## 7. ls : listing directory contents:

To obtain a list of all filenames in the current directory.

*Numerals first*

*Uppercase next*

*Lowercase Then*

**\$ls**

**Output:**08\_packets.html

```
calendar
dept.lst
emp.lst
helpdir
usksk06
```

### ls options:

- Output in multiple columns(-x):

**\$ls -x**

```
08_packets.html  calendar  dept.lst  emp.lst
helpdir          progs     usksk07   usksk07
```

- Identifying directories and executables(-F)

**\$ ls -Fx**

```
08_packets.html  calendar*  cptodos.sh*  dept.lst
emp.lst          helpdir /   progs /       usksk07
```

The \* indicates the file contains the executable code and / refers to directory

- Showing Hidden files also(-a):

**\$ls -axF**

```
.profile          .exrc      .kshrc       .xinitrc
08_packets.html  calendar*  cptodos.sh*  dept.lst
emp.lst          helpdir /   progs /       usksk07
```

The hidden files are indicated by . (dot) displayed before filename.

- Listing directory contents:

**\$ls -x helpdir progs**

helpdir:

```
forms.obd  graphics.obd
```

progs:

```
array.pl  n2words.pl
```



If we specify two directories named helpdir and progs , the contents of the directory i,e filenames are listed out.

- Recursive listing(-R)

The recursive option lists all sub-directories and files in a directory tree structure.

**\$ls -xR**

```
08_packets.html    calendar    cptodos.sh  dept.lst
emp.lst            helpdir    progs       usdsk07
./helpdir
forms.hlp    graphics.hlp
./progs
arrays.pl    n2words.pl
```

### 8.Cat command : Displaying and creating files

cat is one of the most well known commands of UNIX system.

Cat is useful for creating a file .

Its mainly used to display the contents of a small file on the terminal.

- **Using cat to create a file:**  
Enter the command cat, followed by >(right chevron) character and the filename.  
Example: take a filename named foo  
\$ cat > foo  
> Symbol following command means that the output goes to filename following it.  
[ctrl+d]            /\* to terminate or to signify end of the input.  
\$
- **Using cat to display a file**  
Enter the cat command followed by filename  
\$ cat foo  
Symbol following command means that the output goes to filename following it.

#### Cat options (-v and -n)

##### Displaying Non printing characters(-v)

cat is normally used for displaying text files only. If you have non-printing ASCII characters in your input , you can see cat with -v option to display these characters

##### Numbering lines(-n)

The -n option numbers lines.

Cat with more than one filename as arguments:

**cat filename1 filename2 ....**

**cat chap01 chap02**

The contents of second file are displayed immediately after the first file without any header information.

---

### 9. cp: copying a file

- cp command copies a file or a group of files.it creates an exact image of the file on the disk with the different name.
- The syntax requires atleast two filenames to be specified in the command line.
- When both are ordinary files, the first is copied to second file.  
cp source file destination file

### **cp chap01 unit1**

if destination file i.e unit1 does not exist, first it will be created before copying. if not it will be simply overwritten without any warning.

- Copying a file to another directory

ex: assume there is a file named chap01 and it has to be copied to progs directory

#### **cp chap01 progs**

output: chap01 is now copied to directory named progs with the same name chap01.

- Copying a file to another directory with different name

ex: assume there is a file named chap01 and it has to be copied to progs directory with chap01 file renamed as unit1

#### **cp chap01 progs/unit1**

output: chap01 is now copied to directory named progs with the same name unit1

- Copy more than one file with a single command.

#### **cp chap01 chap02 chap03 progs**

chap01, chap02, chap03 files are copied to directory named progs.

- Copy all files beginning with chap

#### **cp chap\* progs**

## **9.1 cp options:**

**Interactive copying (-i):** the -i option warns the user before overwriting the destination file.

Ex: **\$ cp -i chap01 unit1**

cp: overwrite unit1(yes/no)? y

A y at this prompt will overwrite the file.

**Copying directory structure(-R) :** the -R command behaves recursively to copy an entire directory structure say progs to newprogs.

Ex say progs directory contains three files kernel, bash, korn. To copy all three files under progs to newprogs directory

**\$ cp -R progs newprogs**

---

## **10. rm : deleting files**

The rm command deletes one or more files.

Ex 1: The following command deletes three files chap01, chap02, chap03.

**\$ rm chap01 chap02 chap03**

Ex 2: to delete files named chap01 and chap02 under progs directory

**\$ rm progs/chap01 progs/chap02**

Ex 3: to remove all file

**\$ rm \***

## **10.1 rm options:**

**Interactive deletion (-i):** the -i option makes the command ask the user for confirmation before removing each file.

**\$rm -i chap01 chap02 chap03**

rm: remove chap01(yes/no)?y

rm: remove chap01(yes/no)?y

rm: remove chap01(yes/no)?y

**Recursive deletion(-r or -R)** deletes all subdirectories and files recursively. Rm wont normally

remove directories but when used with -r or -R option it will.

```
$ rm -r *
```

Forcing removal: rm prompts for removal, if a file is write protected. The -f option overrides this minor protection and forces removal.

```
$ rm -rf * /*(deletes everything in the current directory and below)
```

---

## 11. mv: RENAMING FILES.

The mv command renames or moves files. It has two distinct functions:

- a. It renames a file or directory
- b. it moves a group of files to a different directory

**To rename a file chap01 to man01**

```
$ mv chap01 man01
```

mv replace the filename in the existing directory entry with the new name.

No additional space is consumed on disk during renaming.

**To rename a directory:**

```
$ mv pts perdir
```

pts directory is renamed as perdir

**To move group of files to a directory**

```
mv chap01 chap02 chap03 progs
```

to move three files chap01, chap02, chap03 to directory named progs

## 12.wc command: COUNTING LINES,WORDS,CHARACTERS

wc command takes one or more filenames as arguments and displays four columnar output.

First we will create a file named infile

```
$ cat > infile
```

I am the wc command

I count characters,words and lines

[ctrl+D]

```
$wc infile
```

```
2      10     55    infile
```

wc counts lines in first column ,words in second column,characters in third column and filename in fourth column..

A line is any group of characters not containing a newline

A word is group of characters not containing a space tab or newline.

A character is the smallest unit of information and includes a space, tab and newline

wc options:

```
$ wc -l infile
```

```
2
```

```
$wc -w infile
```

```
10
```

```
$wc -c  infile
```

```
55
```

- When two filenames are passed as wc argument

```
[vizion@localhost ~]$ cat > chap01
unix is a multitasking os
its is a multiuser os
[vizion@localhost ~]$ cat > chap02
who cal date
ls rm mv
[vizion@localhost ~]$ wc chap01 chap02
 2 10 48 chap01
 2  6 22 chap02
 4 16 70 total
```

First line : number of lines, words and characters of chap01

Second line: number of lines, words and characters of chap02

Third line: Total number of lines, words and characters of both.

---

### 13. od Command: DISPLAYING DATA IN OCTAL.

\$ cat odfile

White space includes a

The ^G character rings a bell

#### \$ od -b odfile

The -b option displays the octal values for each character.

```
000000 127 150 151 164 145 040 163 160 141 143 145 040 151 156 143 154
```

```
000000 165 144 145 163 040 141 040 011 012 124 150 145 040 007 040 143
```

Each line displays 16 bytes of data in octal , preceded by the offset in the file of the first byte in the line.

#### \$ od -bc odfile

The -b and -c option combined

Each line is now replaced with two.

The octal values are shown in first line and printable characters and escape sequences are shown in second line

```
000000 127 150 151 164 145 040 163 160 141 143 145 040 151
      W  h  i  t  e          s  p  a  c  e          i
      156 143 154
      n  c  l
000000 165 144 145 163 040 141 040 011 012 124 150 145 040
      u  d  e  s          a      \t  \n  T  h  e
      007 040 143
      007      c
```

The octal equivalent of characters are displayed ex for W- 127, i-151, \t (tab)-011, \n(newline)-012  
^G(Bell character)- 007

## BASIC FILE ATTRIBUTES

### 1. ls -l: LISTING FILE ATTRIBUTES

ls command is used to obtain a list of all filenames in the current directory. The output in UNIX lingo is often referred to as the listing. Sometimes we combine this option with other options for displaying other attributes, or ordering the list in a different sequence. ls look up the file's inode to fetch its attributes. It lists seven attributes of all files in the current directory and they are:

#### 1.1 File type and Permissions

- 1.2 Links
- 1.3 Ownership
- 1.4 Group ownership
- 1.5 File size
- 1.6 Last Modification date and time
- 1.7 File name

**1.1 The file type and its permissions:** The first column shows the type and permissions associated with each file. The first character in this column is mostly a – which indicates that the file is an ordinary one. In unix, file system has three types of permissions- read, write and execute.

**1.2 Links:** The second column indicates the number of links associated with the file. This is actually the number of filenames maintained by the system of that file.

**1.3 Ownership:** The third column shows the owner of files. The owner has full authority to tamper with files content and permissions. Similarly, you can create, modify or remove files in a directory if you are the owner of the directory.

**1.4 Group ownership:** The fourth column represents the group owner of the file. When opening a user account, the system admin also assigns the user to some group. The concept of a group of users also owning a file has acquired importance today as group members often need to work on the same file.

**1.5 File size:** The fifth column shows the size of the file in bytes. The important thing to remember here is that it only a character count of the file and not a measure of the disk space that it occupies.

**1.6 Last modification time:** The sixth, seventh and eighth columns indicate the last modification time of the file, which is stored to the nearest second. A file is said to be modified only if its content have changed in any way. If the file is less than a year old since its last modification time, the year won't be displayed.

**1.7 Filename:** The last column displays the filename arranged in ASCII collating sequence.

**For example, \$ ls -l**

```
total 72
-rw-r--r--  1 kumar  metal  19514  may  10 13:45  chap01
-rw-r--r--  1 kumar  metal   4174  may  10 15:01  chap02
-rw-rw-rw-  1 kumar  metal    84    feb  12 12:30  dept.lst
-rw-r--r--  1 kumar  metal   9156  mar   12 1999  genie.sh
drwxr-xr-x  2 kumar  metal    512  may   09 10:31  helpdir
drwxr-xr-x  2 kumar  metal    512  may   09 09:57  progs
```

## 2. LISTING DIRECTORY ATTRIBUTES (-d option)

- **ls -d** will not list all subdirectories in the current directory

**For example,**

**\$ls -ld helpdir progs**

```
drwxr-xr-x  2 kumar  metal   512  may  9 10:31  helpdir
drwxr-xr-x  2 kumar  metal   512  may  9 09:57  progs
```

- Directories are easily identified in the listing by the first character of the first column, which here shows a **d**.

- To see the attributes of a directory rather than the files contained in it, use **ls -ld** with the directory name.

### 3. FILE OWNERSHIP

- When you create a file, you become its **owner** and it shows up in the third column of the files listing.
- Group name is seen in the fourth column; your group is the **group owner** of the file.
- Every owner is attached to a group owner. Several users may belong to a single group, but the privileges of the group are set by the owner of the file and not by the group members.
- When the system administrator creates a user account, he has to assign these parameters to the user:

The **user-id (UID)** – both its name and numeric representation

The **group-id (GID)** – both its name and numeric representation

### 4. FILE PERMISSIONS

- UNIX has a simple and well defined system of assigning permissions to files.
- Lets issue the **ls -l** command once again to view the permissions of a few lines .

```
$ls -l chap02 dept.lst dateval.sh
-rwxr-xr-- 1 kumar metal 25000 May 10 19:21 chap02
-rwxr-xr-x 1 kumar metal 890 Jan 10 23:17 dept.lst
-rw-rw-rw- 1 kumar metal 84 Feb 18 12:20 dateval.sh
```

Consider the first column.

- rwx r-x r--

Each group here represents the category and contain three slots representing the read, write and execute permissions of the file.

**r** indicates the read permission; **w** indicates write permission; **x** indicates execute permission

- (hyphen) indicates the absence of the corresponding permission.

**In the above example, the file permissions of chap02 file is**

File	owner/user	group	others
-	rwx	r-x	r--

**First group(rwx)** has all the three permissions.

- The file is readable, writable and executable by the **owner** of the file, kumar.
- The third column shows the owner of the file.
- The first permissions group applies to kumar.
- You have to login with the name kumar for the privileges to apply to you.

**Second group(r-x):**

- has a hyphen in the middle slot, which indicates the absence of write permissions by the **group** owner of the file.
- The group owner is metal and all users belonging to group metal has only read and execute permissions.

**Third group(r--):**

- has the write and execute bits absent.
- This set is applicable to **others** i.e those who are neither the owner nor group.
- This category is referred to as the world.

## 5. CHANGING FILE PERMISSIONS- **chmod** command.

A file or a directory is created with a default set of permissions, which can be determined by **umask**. Let us assume that the file permission for the created file is **-rw-r--r--**. Using **chmod** command, we can change the file permissions and allow the owner to execute his file. The command can be used in two ways:

**5.1 In a relative manner by specifying the changes to the current permissions**

**5.2 In an absolute manner by specifying the final permissions**

### 5.1 RELATIVE PERMISSIONS

**chmod** only changes the permissions specified in the command line and leaves the other permissions unchanged.

Its syntax is:

**chmod        category        operation        permission        filename(s)**

**chmod** takes an expression as its argument which contains:

user category (user, group, others)

operation to be performed (assign or remove a permission)

type of permission (read, write, execute)

The below shows the abbreviations used by **chmod** command

<b>Category</b>	<b>operation</b>	<b>permission</b>
u - user	+ assign	r - read
g – group	- remove	w - write
o - others	= absolute	x - execute
a - all (ugo)		

Ex 1:

```
$ls -l xstart
-rw-r--r-- 1 kumar metal 1906 sep 23:38 xstart
```

Here user is having the only read and execute permission .

*Using relative file permission need to add the execute permission to user*

**chmod    category        operation(+,-)        permission        filename.**

```
$chmod        u                +                x                xstart
```

**\$chmod u+x xstart**

**\$ ls -l xstart**

```
-rwxr--r-- 1 kumar metal 1906 sep 23:38 xstart
```

After executing the **chmod** command, the command assigns (+) execute (x) permission to the user (u), other permissions remain unchanged.

Ex 2: To remove execute permission from all and assign read permission to group and others

**\$chmod a-x, go+r xstart**    /\*to remove execute permission from all(a)ie user, group, others  
                                      /\*to assign read permission to group and others (go+r)

Ex 3: To assign write and execute permissions for others.

**\$chmod o+wx xstart**





**\$chmod u-rw, go-r xstart or**

**\$chmod 000 xstart**

After Executing above any one command the output will be removes the all permission from all categories as shown below

-----

This is simply useless but still the user can delete this file

On the other hand,

**chmod a+rw xstart or**

**chmod 777 xstart**

After Executing either of the one command it adds the all permission to all categories as shown below

**-rwxrwxrwx**

The UNIX system by default, never allows this situation as you can never have a secure system. Hence, directory permissions also play a very vital role here.

#### 5.4 use chmod Recursively.(-R)

- It possible to make **chmod** descend directory hierarchy and apply the expression to every file and sub directory it finds. This is done by using -R option

**\$chmod -R a+x shell\_scripts**

This makes all the files and subdirectories found in the shell\_scripts directory, executable by all users.

### 6. DIRECTORY PERMISSIONS

- Directories also have their own permissions and the significance of these permissions differ from those of ordinary files.
- The default permissions of a directory are,  
**rwxr-xr-x (755)**
- A directory must never be writable by group and others

Ex1:

**\$mkdir c\_progs ; ls -ld c\_progs**

drwxr-xr-x 2 kumar metal 512 may 9 09:57 c\_progs

Here the c\_progs directory is created (mkdir c\_progs) and then the attributes of directory is listed out(ls -ld c\_progs)

If a directory has write permission for group and others also, be assured that every user can remove every file in the directory. As a rule, you must not make directories universally writable unless you have definite reasons to do so.

### 7. CHANGING FILE OWNERSHIP

There are two commands meant to change the ownership of a file or directory.

**chown** changing file owner and

**chgrp** changing group owner

## 7.1 chown : Changing file owner

- The syntax is  
**chown options owner [:group] file(s)**
- For changing ownership requires super user permission. So let's first change our status to that of super user with the **su** command:  
\$su  
Password: \*\*\*\*\*  
#\_
- After the password is successfully entered, **su** returns a # prompt, the prompt used by root. **su** lets us acquire super user status if we know the root password.
- Consider the file note owned by kumar. To now renounce the ownership of the file note to Sharma, use **chown** in the following way:  
**# ls -l note**  
-rwxr---x 1 kumar metal 347 may 10 20:30 note

```
#chown sharma note; ls -l note
-rwxr---x 1 sharma metal 347 may 10 20:30 note
```

Once ownership of the file has been given away to sharma, the user file permissions that previously applied to Kumar now apply to sharma. Thus, Kumar can no longer edit *note* since there is no write privilege for group and others. He cannot get back the ownership either. But he can copy the file to his own directory, in which case he becomes the owner of the copy.

## 7.2 chgrp : Changing Group Owner

- This command changes the file's group owner. No superuser permission is required.

```
$ ls -l dept.lst
-rw-r--r-- 1 kumar metal 139 jun 8 16:43 dept.lst
Here the group owner of dept.lst is metal
```

```
$chgrp dba dept.lst; ls -l dept.lst
-rw-r--r-- 1 kumar dba 139 jun 8 16:43 dept.lst
```

The group owner of the file dept.lst is changed from metal to dba by issuing the command  
\$chgrp dba dept.lst

The file attributes of dept.lst is listed by issuing the command  
\$ls -l dept.lst

- Using chown to change both file owner and group :**  
The syntax requires two arguments to be separated by :  
chown sharma:dba dept.lst  
Here the ownership of dept.lst is changed to sharma and group to dba