Basic File Attributes

The UNIX file system allows the user to access other files not belonging to them and without infringing on security. A file has a number of attributes (properties) that are stored in the inode. In this chapter, we discuss,

- ls –l to display file attributes (properties)
- Listing of a specific directory
- Ownership and group ownership
- Different file permissions

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:

- File type and Permissions
- Links
- Ownership
- Group ownership
- File size
- Last Modification date and time
- File name

The file type and its permissions are associated with each file. Links indicate the number of file names maintained by the system. This does not mean that there are so many copies of the file. File is created by the owner. Every user is attached to a group owner. File size in bytes is displayed. Last modification time is the next field. If you change only the permissions or ownership of the file, the modification time remains unchanged. In the last field, it displays the file name.

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   9  10:31     helpdir
```

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

Listing Directory Attributes

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. The significance of the attributes of a directory differs a good deal from an ordinary file. To see the attributes of a directory rather than the files contained in it, use ls –ld with the directory name. Note that simply using ls –d will not list all subdirectories in the current directory. Strange though it may seem, ls has no option to list only directories.

File Ownership

When you create a file, you become its owner. 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

File Permissions

UNIX follows a three-tiered file protection system that determines a file's access rights. It is displayed in the following format:

Filetype owner (rwx) groupowner (rwx) others (rwx)

For Example:

```
-rwxr-xr--  1   kumar   metal   20500  may 10 19:21  chap02

          r w x                      r - x                   r - -

         owner/user              group owner                    others
```

The first group has all three permissions. The file is readable, writable and executable by the owner of the file. The second group has a hyphen in the middle slot, which indicates the absence of write permission by the group owner of the file. The third group has the write and execute bits absent. This set of permissions is applicable to others.

You can set different permissions for the three categories of users – owner, group and others. It's important that you understand them because a little learning here can be a dangerous thing. Faulty file permission is a sure recipe for disaster

Changing File Permissions

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:

In a relative manner by specifying the changes to the current permissions
In an absolute manner by specifying the final permissions

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)

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

Let us discuss some examples:
Initially,

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

```
chmod u+x xstart
```

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

The command assigns (+) execute (x) permission to the user (u), other permissions remain unchanged.

```
chmod ugo+x xstart     or
chmod a+x xstart       or
chmod +x xstart
```

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

chmod accepts multiple file names in command line

```
chmod u+x note note1 note3
```

Let initially,

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

```
chmod go-r xstart
```

Then, it becomes

```
-rwx--x--x    1   kumar   metal   1906   sep 23:38    xstart
```

Absolute Permissions

Here, we need not to know the current file permissions. We can set all nine permissions explicitly. A string of three octal digits is used as an expression. The permission can be represented by one octal digit for each category. For each category, we add octal digits. If we represent the permissions of each category by one octal digit, this is how the permission can be represented:

- Read permission – 4 (octal 100)
- Write permission – 2 (octal 010)
- Execute permission – 1 (octal 001)

Octal	**Permissions**	**Significance**
0	- - -	no permissions
1	- - x	execute only
2	- w -	write only
3	- w x	write and execute
4	r - -	read only
5	r - x	read and execute
6	r w -	read and write
7	r w x	read, write and execute

We have three categories and three permissions for each category, so three octal digits can describe a file's permissions completely. The most significant digit represents user and the least one represents others. chmod can use this three-digit string as the expression.

Using relative permission, we have,

```
chmod a+rw xstart
```

Using absolute permission, we have,

```
chmod 666 xstart

chmod 644 xstart

chmod 761 xstart
```

will assign all permissions to the owner, read and write permissions for the group and only execute permission to the others.

777 signify all permissions for all categories, but still we can prevent a file from being deleted. 000 signifies absence of all permissions for all categories, but still we can delete a file. It is the directory permissions that determine whether a file can be deleted or not. Only owner can change the file permissions. User can not change other user's file's permissions. But the system administrator can do anything.

The Security Implications

Let the default permission for the file xstart is

```
        -rw-r--r--
chmod u-rw, go-r xstart                  or

chmod 000 xstart

----------
```

This is simply useless but still the user can delete this file
On the other hand,

```
chmod a+rwx xstart

chmod 777 xstart

        -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

We can use chmod Recursively.

```
chmod -R a+x shell_scripts
```

This makes all the files and subdirectories found in the shell_scripts directory, executable by all users. When you know the shell meta characters well, you will appreciate that the * doesn't match filenames beginning with a dot. The dot is generally a safer but note that both commands change the permissions of directories also.

Directory Permissions

It is possible that a file cannot be accessed even though it has read permission, and can be removed even when it is write protected. The default permissions of a directory are,

```
rwxr-xr-x (755)
```

A directory must never be writable by group and others

Example:

```
mkdir c_progs

ls –ld c_progs

drwxr-xr-x      2  kumar  metal  512  may  9  09:57  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.

Changing File Ownership

Usually, on BSD and AT&T systems, there are two commands meant to change the ownership of a file or directory. Let kumar be the owner and metal be the group owner. If sharma copies a file of kumar, then sharma will become its owner and he can manipulate the attributes

chown changing file owner and **chgrp** changing group owner

On BSD, only system administrator can use chown
On other systems, only the owner can change both

chown

Changing ownership requires superuser permission, so use **su** command

```
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 can not 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.

chgrp

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

              chgrp dba dept.lst; ls –l dept.lst

-rw-r--r--    1  kumar  dba  139  jun  8  16:43  dept.lst
```

In this chapter we considered two important file attributes – permissions and ownership. After we complete the first round of discussions related to files, we will take up the other file attributes.

The vi Editor

To write and edit some programs and scripts, we require editors. UNIX provides vi editor for BSD system – created by Bill Joy. Bram Moolenaar improved vi editor and called it as vim (vi improved) on Linux OS.

vi Basics

To add some text to a file, we invoke,

vi <*filename*>

In all probability, the file doesn't exist, and vi presents you a full screen with the filename shown at the bottom with the qualifier. The cursor is positioned at the top and all remaining lines of the screen show a ~. They are non-existent lines. The last line is reserved for commands that you can enter to act on text. This line is also used by the system to display messages. This is the command mode. This is the mode where you can pass commands to act on text, using most of the keys of the keyboard. This is the default mode of the editor where every key pressed is interpreted as a command to run on text. You will have to be in this mode to copy and delete text

For, text editing, vi uses 24 out of 25 lines that are normally available in the terminal. To enter text, you must switch to the input mode. First press the key i, and you are in this mode ready to input text. Subsequent key depressions will then show up on the screen as text input.

After text entry is complete, the cursor is positioned on the last character of the last line. This is known as current line and the character where the cursor is stationed is the current cursor position. This mode is used to handle files and perform substitution. After the command is run, you are back to the default command mode. If a word has been misspelled, use ctrl-w to erase the entire word.

Now press esc key to revert to command mode. Press it again and you will hear a beep. A beep in vi indicates that a key has been pressed unnecessarily. Actually, the text entered has not been saved on disk but exists in some temporary storage called a buffer. To save the entered text, you must switch to the execute mode (the last line mode). Invoke the execute mode from the command mode by entering a: which shows up in the last line.

The Repeat Factor

vi provides repeat factor in command and input mode commands. Command mode command k moves the cursor one line up. 10k moves cursor 10 lines up.
To undo whenever you make a mistake, press

Esc u

To clear the screen in command mode, press

```
ctrl-l
```

Don’t use (caps lock) - vi commands are case-sensitive
Avoid using the PC navigation keys

Input Mode – Entering and Replacing Text

It is possible to display the mode in which is user is in by typing,

```
:set showmode
```

Messages like INSERT MODE, REPLACE MODE, CHANGE MODE, etc will appear in the last line.
Pressing ‘i’ changes the mode from command to input mode. To append text to the right of the cursor position, we use *a, text.* I and A behave same as i and a, but at line extremes I inserts text at the beginning of line. A appends text at end of line. o opens a new line below the current line

- *r<letter>* replacing a single character
- *s<text/word>* replacing text with s
- *R<text/word>* replacing text with R
- Press esc key to switch to command mode after you have keyed in text

Some of the input mode commands are:

COMMAND	**FUNCTION**
i	inserts text
a	appends text
I	inserts at beginning of line
A	appends text at end of line
o	opens line below
O	opens line above
r	replaces a single character
s	replaces with a text
S	replaces entire line

Saving Text and Quitting – The ex Mode

When you edit a file using vi, the original file is not distributed as such, but only a copy of it that is placed in a buffer. From time to time, you should save your work by writing the buffer contents to disk to keep the disk file current. When we talk of saving a file, we actually mean saving this buffer. You may also need to quit vi after or without saving the buffer. Some of the save and exit commands of the ex mode is:

Command	**Action**
:W	saves file and remains in editing mode
:x	saves and quits editing mode
:wq	saves and quits editing mode
:w <*filename*>	save as
:w! <*filename*>	save as, but overwrites existing file
:q	quits editing mode
:q!	quits editing mode by rejecting changes made
:sh	escapes to UNIX shell
:recover	recovers file from a crash

Navigation

A command mode command doesn't show up on screen but simply performs a function. To move the cursor in four directions,

k	moves cursor up
j	moves cursor down
h	moves cursor left
l	moves cursor right

Word Navigation

Moving by one character is not always enough. You will often need to move faster along a line. vi understands a word as a navigation unit which can be defined in two ways, depending on the key pressed. If your cursor is a number of words away from your desired position, you can use the word-navigation commands to go there directly. There are three basic commands:

b	moves back to beginning of word
e	moves forward to end of word
w	moves forward to beginning word

Example,

5b takes the cursor 5 words back
3w takes the cursor 3 words forward

Moving to Line Extremes

Moving to the beginning or end of a line is a common requirement.

To move to the first character of a line

0 or |

30| moves cursor to column 30
$ moves to the end of the current line
The use of these commands along with b, e, and w is allowed

Scrolling

Faster movement can be achieved by scrolling text in the window using the control keys. The two commands for scrolling a page at a time are

ctrl-f	scrolls forward
ctrl-b	scrolls backward

10ctrl-f	scroll 10 pages and navigate faster

ctrl-d	scrolls half page forward
ctrl-u	scrolls half page backward

The repeat factor can also be used here.

Absolute Movement

The editor displays the total number of lines in the last line

Ctrl-g	to know the current line number
40G	goes to line number 40
1G	goes to line number 1
G	goes to end of file

Editing Text

The editing facilitates in vi are very elaborate and invoke the use of operators. They use operators, such as,

d	delete
y	yank (copy)

Deleting Text

x	deletes a single character
dd	delete entire line
yy	copy entire line

6dd	deletes the current line and five lines below

Moving Text

Moving text (p) puts the text at the new location.
p and P place text on right and left only when you delete parts of lines. But the same keys get associated with “below” and “above” when you delete complete lines

Copying Text

Copying text (y and p) is achieved as,

yy	copies current line
10yy	copies current line & 9 lines below

Joining Lines

J	to join the current line and the line following it
4J	joins following 3 lines with current line

Undoing Last Editing Instructions

In command mode, to undo the last change made, we use	u
To discard all changes made to the current line, we use	U

vim (LINUX) lets you undo and redo multiple editing instructions. u behaves differently here; repeated use of this key progressively undoes your previous actions. You could even have the original file in front of you. Further 10u reverses your last 10 editing actions. The function of U remains the same.

You may overshoot the desired mark when you keep u pressed, in which case use ctrl-r to redo your undone actions. Further, undoing with 10u can be completely reversed with 10ctrl-r. The undoing limit is set by the execute mode command: set undolevels=n, where n is set to 1000 by default.

Repeating the Last Command

The . (dot) command is used for repeating the last instruction in both editing and command mode commands
For example:

2dd deletes 2 lines from current line and to repeat this operation, type. **(dot)**

Searching for a Pattern

/ search forward
? search backward

/printf
The search begins forward to position the cursor on the first instance of the word

?pattern
Searches backward for the most previous instance of the pattern

Repeating the Last Pattern Search

n repeats search in same direction of original search

n doesn't necessarily repeat a search in the forward direction. The direction depends on the search command used. If you used? printf to search in the reverse direction in the first place, then n also follows the same direction. In that case, N will repeat the search in the forward direction, and not n.

Search and repeat commands

Command	**Function**
/pat	searches forward for pattern pat
?pat	searches backward for pattern pat
n	repeats search in same direction along which previous search was made
N	repeats search in direction opposite to that along which previous search was made

Substitution – search and replace

We can perform search and replace in execute mode using :s. Its syntax is,

```
:address/source_pattern/target_pattern/flags
```

:1,$s/director/member/g	can also use % instead of 1,$
:1,50s/unsigned//g	deletes unsigned everywhere in lines 1 to 50
:3,10s/director/member/g	substitute lines 3 through 10
:.s/director/member/g	only the current line
:$s/director/member/g	only the last line

Interactive substitution: sometimes you may like to selectively replace a string. In that case, add the c parameter as the flag at the end:

```
:1,$s/director/member/gc
```

Each line is selected in turn, followed by a sequence of carets in the next line, just below the pattern that requires substitution. The cursor is positioned at the end of this caret sequence, waiting for your response.

The ex mode is also used for substitution. Both search and replace operations also use regular expressions for matching multiple patterns.

The features of vi editor that have been highlighted so far are good enough for a beginner who should not proceed any further before mastering most of them. There are many more functions that make vi a very powerful editor. Can you copy three words or even the entire file using simple keystrokes? Can you copy or move multiple sections of text from one file to another in a single file switch? How do you compile your C and Java programs without leaving the editor? vi can do all this.

- Source: Sumitabha Das, “UNIX – Concepts and Applications”, 4th edition, Tata McGraw Hill, 2006