

Chapter 3 The Unix File System

Section 3: Permissions

6. File Attributes

- In the previous sections, you created directories, navigated the file system, and copied and removed files without any problem.
- In real life, however, you may have problems when handling a file or directory.
- Your file may be modified or even deleted by others.
- A restoration from a backup may be unable to write to your directory.
- You must know why these problems happen and how to prevent and rectify them.
- A file has a number of attributes which are changeable by certain well-defined rules.
- We will be using the **ls** command to display these attributes.

6.1 Listing Files

0In the previous section, we used the **ls** command to display some files in the current directory.

0When you use the **ls** command the list might be long. To list a particular file just use the **ls** command with the particular filename

```
$ ls note1
```

0To list the contents of a directory, just specify the path for the directory and use the path with the **ls** command:

```
$ ls /home/user1/progs
```

6.2 ls Options

Output in Multiple Columns: When you have several files, it is better to display the filenames in multiple columns. Use the **-x** option to produce an output with multiple columns.

```
$ ls -x
```

```
a      another  anotherfile  b  c  course  
Index.html nohup.out note1      note2  progs
```

Identifying Directories and Executables (-F)

○The output of **ls** that you have seen so far merely showed filenames.

○You didn't know how many of them, if any, were directory files. To identify directories and executable files, the **-F** option should be used.

○Combining this option with **-x** option produces a multi-columnar output.

```
$ ls -Fx
```

```
a      anotherfile*  c  Desktop/  index.html  another  b  
      course/
```

○The ***** indicates that the file contains executable code; the **/** refers to a directory.

Showing Hidden Files (-a):

◦ There are certain hidden files (those beginning with a dot) in every directory, especially in the home directory, that normally don't show up in the listing.

◦ The -a(all) option lists all hidden files.

```
$ls -axF
```

Listing Directory Contents:

◦ To list the contents of a directory, just specify the name of the directory with the **ls** command:

```
$ ls -x progs
```

Recursive Listing:

◦ The -R (recursive) option lists all files and subdirectories in a directory tree.

```
$ ls -xR
```

6.2 ls -l: Listing File Attributes

0The -l option of ls reveals most attributes of a file-like its permission, size and ownership details.

0The output in UNIX is often referred to as the **listing**:

```
$ ls -l
```

```
total 35
```

```
-rw-r-r--  1  kebede    users  19514  May 10 13:45  
          chap01
```

```
-rw-r-r   1  Kebede    users   4174   May 10 15:01  
          chap02
```

```
-rw-rw-rw  1  kebede    users    84  Feb 12 12:30  
          dept.lst
```

```
drwxr-xr-x  2  Ayele     users    64  May 9  10:31  
          helpdir
```

```
drwxr-xr-x  2  Ayele     users   320   May 9   9:57  
          progs
```

0The list is preceded by the words total 35; a total of 35 blocks are occupied by these files in the disk (Each block contains 512 or 1024 bytes).

Type and Permissions:

0The first column shows the type and **permissions** associated with each file.

0The first character in this column shows a - for the first three files, which indicates that the file is an ordinary one.

0For the directories however, there is a d at the beginning of this column.

0You then see a series of characters that can take the values of r, w, x and -.

0In the UNIX system, a file can have three types of permissions- read, write and execute.

Links: The second column indicates the number of **links** associated with the file.

○ This is actually the number of names maintained by the system of that file.

○ UNIX lets a file have as many names as you want it to have, even though there is a single file on disk

○ Using these names, a reference to the file can be made.

○ You can use a link to “notionally place” a file in a specific directory where many programs expect to find it.

○ Say, you have a number of programs which invoke **perl** using the absolute pathname `/usr/local/bin/perl`.

○ When you upgraded your system, you found **perl** has moved to `/usr/bin`.

○ You wouldn't change all your programs to point to this new path.

○ Just leave a link like this in `/usr/local/bin`:

In `/usr/bin/perl` `/usr/local/bin/perl`

Ownership and Group Ownership:

○When you create a file, you automatically become its **owner**. The third column shows kebede as owner of all files except the directories.

○The owner has full authority to tamper with the file's contents, permissions, and even ownership-a privilege not available with others except the root user.

○When opening a user account, the system administrator also assigns the user to some group.

○The fourth column stands for the **group owner** of the file.

○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.

Size: The fifth column shows the size of the file in bytes.

○ This is actually a measure of the character count and not the disk space occupied by the file.

○ The space used on disk is usually larger than this figure since files are written to disk in blocks of 1024 bytes (normally).

○ In other words, even though the file `dept.lst` contains 84 bytes, it occupies 1024 bytes on the disk.

○ The two directories show smaller file sizes.

○ A directory maintains a list of file names along with an identification number (the inode) for each file.

○ The size of the directory file depends on the size of this list – whatever the size of the files themselves.

Last Modification Time:

- The next set of three columns indicates the last **modification time** of the file.
- A file is considered to be modified if its contents have changed in any way.
- The year is displayed if the file is more than a year old since its last modification time.

Filename:

- The last column displays the filenames arranged alphabetically.

Listing Directory Attributes (ls -d)

- ls** lists files in the directory when used with a directory name. If you want to list the directory attributes rather than its contents, you have to use the **-d(directory)** option.

\$ ls -ld helpdir progs

```
drwxr-xr-x 2  Ayele      users  64  May 9 10:31
             helpdir
drwxr-xr-x 2  Ayele      users 320   May 9  9:57
             progs
```

◦The -l option has been combined here to show that the listing for a directory is somewhat different from an ordinary file.

◦Directories are easily identified in the listing by the first character of the first column which is always d.

◦For ordinary files, this place always shows a - (hyphen), and for device file, either a b or a c.

File Permissions:

◦UNIX has a simple, but well defined system of assigning **permissions** to files.

```
ls -l chap02 dept.lst dateval.sh
```

```
-rwxr-xr-- 1 kebede users 20500 May 10 19:21
           chap02
-rwxr-xr-x 1 kebede users 890   Jan 29 23:17
           dateval.sh
-rw-rw-rw 1 kebede users 84 Feb 12 12:30
           dept.lst
```

Observe the first column that represents the file permissions. These permissions are different for the three files.

UNIX follows a three-tiered file protection system that determines the access rights that you have for the file.

Each tier represents a **category**, and comprises of a string of rs, ws, and xs to represents the three types of permissions.

Or indicates read permission which means **cat** will display the file.

Ow indicates write permission-you can edit the file with a text editor.

Ox indicates execute permission. The file can be executed as a program.

OIn the previous sample output representing the permission of the file chap02, the first group (rwx) has all three permissions.

OThe file is readable, writable and executable by the **owner** of the file.

OThe owner of the file is Kebede which is indicated in the third column.

OYou have to log in as Kebede for these permissions to apply to you.

OThe second group (r-x) has a hyphen in the middle slot and applies to the **group owner** of the file.

- This group owner is **users** and all users belonging to the users group have read and executable permission only.
- The third group (r--) has the write and executable bit absent.
- This set of permissions is applicable to **others**-those who are neither the owner Kebede nor belong to the users group.
- The **group** and **others** categories are often collectively referred to as the **world**.
- If you are told that a file is “world-writable,” it means that there is a w in that part of the permissions string that applies to group and others (like r—rw-rw-).

6.3 chmod: Changing File Permissions

- Let us create a shell script to see the default file permission.
- Create a file called test using vi text editor:
\$ vi test

Then type in the following commands:

```
echo "Enter the year";
```

```
read y;
```

```
cal $y;
```

Save the file and quit.

See the read-write permission of the file with `ls -l` command.

```
$ ls -l test
```

```
-rw-rw-r--  user1  users  80  Sep 24 13:31  test
```

Notice that a file doesn't have executable permission by default – not even for the owner. When you try to run the file, it informs you that you are denied of permission.

```
$ ./test
```

```
bash: ./test: Permission Denied
```

○ To assign executable permission, just change the permission of the file with the **chmod (change mode)** command.

○ Only the owner of the file can use this command.

○ **chmod** sets a file's permissions (read, write and execute) for all three categories of users (owner, group and others) and uses the following syntax:

```
chmod category operation permission file(s)
```

○ The structure of a **chmod** command with its arguments is :

- Category of user(owner, group owner or others)
- Operation to be performed (assign or remove a permission)
- Permission type (read, write or execute)

Let us assign executable permission to the file test(for the owner of the file).

```
$ chmod u+x test
```

```
$ ls -l test
```

```
-rwxr--r-- 1 kebede users 10 May 10 20:30  
test
```

○The command assigns (+) executable permission (x) to the user (u).

○User in **chmod** is to be understood as owner.

○You can now execute the file.

```
$ ./test
```

Enter the year

[you should enter the year you want the calendar of and the calendar for that year will be displayed]

○To enable execution by all, you should use more than one character for the category.

```
$ chmod ugo+x test; ls -l test
```

```
-rwxr-xr-x  1  kebede    users  10  May 10 20:30  
            small
```

u stands for users.

g for group, and

o for others

There is even a shorthand notation to specify all categories:

```
chmod a +x test
```

a implies ugo

```
chmod +x test
```

By default, a is implied

Permissions are removed with - operator.

To remove the read permission from both group and other, use the expression go-r:

```
$ chmod ugo-x test
```

```
$ chmod go-r test; ls -l test
```

```
-rwx-----  1  kebede    users  10  May 10 20:30
```

```
test
```

To remove the rest of the permission from the owner, you have to use:

```
$ chmod u-rwx test; ls -l test
```

```
----- 1  kebede    users  10 May 10 20:30  
                test
```

Now if you try the following commands it will give you error:

```
$ cat test
```

```
cat: test: Permission Denied
```

```
$ echo "This is the last Unix class" > test
```

```
bash: test: Permission Denied
```

```
$ ./test
```

```
Bash: ./test: Permission Denied
```

6.3.1 Using Multiple Expressions

Using multiple expressions and the comma as delimiter, **chmod** can set permissions in any manner you want.

For example, to restore the original permissions (rw-rw—r--) to the file test (currently, -----), you have to assign read permission to all and write permission to the owner.

\$ chmod a+r, u+w, g+w test: ls -l test

```
-rw-rw—r--    1  kebede    users  10 May 10 20:30
                  test
```

6.3.2 Absolute Assignment

Absolute Assignment by **chmod** is done with the = operator.

Unlike the + or - operator, it assigns only those permissions that are specified along with it and removes other permissions.

Hence, if you want to assign only read permission to all three categories and remove all other permissions from the file test, you can do it using either of the following commands:

```
chmod ugo =r test  
chmod a=r test  
chmod =r test
```

◦ Sometimes, you don't need to know what a file's current permissions are but want to set all the time permission bits explicitly.

◦ The absolute assignment method is then preferable and easier to use.

6.3.3 The Octal Notation

◦ There is a shorthand notation available for representing file permissions.

◦ **chmod** also takes a numeric argument that describes both the category and the permission.

○The notation uses octal numbers (numbers using the base 8 contrasted with the standard decimal system which uses the base 10).

○Each type of permission is assigned a number as shown:

- Read Permission – 4
- Write Permission – 2
- Execute permission – 1

○When one category has multiple permissions, the respective numbers are added.

○For example, if the owner has read and write permission, the permissions for this category are represented by the number 6 (4 +2).

○When this exercise is repeated for the other categories, you have a three-character octal number following this sequence: user, group and others.

○Like the = operator, the assignment is absolute

○ You can use this method to assign read and write permission to all three categories:

```
$ chmod 666 test; ls -l test
```

```
-rw-rw-rw- 1 kebede users 10 May 10 20:30  
test
```

○ The 6 indicates read and write permission (4 + 2).

○ To restore the original permissions to the file, you need to remove the write permission from group and others:

```
$ chmod 644 test; ls -l test
```

```
-rw-r--r-- 1 kebede users 10 May 10 20:30  
test
```

○ The highest possible number for each category is 7 (4 + 2 + 1) and the lowest is 0.

○ So 777 signifies all permissions for all categories, while 000 indicates absence of all permissions for all categories.

6.3.4 Recursive Operation (-R)

○ **chmod** works with multiple files also.

○ You can assign the same set of permissions to a group of files using a single **chmod** command:

```
chmod u+x note1 note2
```

○ It is now possible to apply the **chmod** command recursively to all files and subdirectories.

○ This is done with the **-R** (recursive) option and needs only the directory name or the metacharacter ***** as argument:

```
chmod -R a+x progs      Acts recursively on all files in progs  
                        directory
```

```
chmod -R a+x *          Acts recursively on all files in the  
                        current directory
```

6.4 Directory Permissions

◦ **Read Permission:** Read permission for a directory means the **ls** can read the list of filenames stored in that directory.

◦ If you remove its read permission, **ls** won't work, and standard utilities won't be able to read the directory information:

```
$ chmod -r progs; ls -ld progs
```

```
d-wx—x—x    2  kebede    128    Jun 18 22:41 progs
```

```
$ ls -l progs
```

```
ls: progs: Permission denied
```

◦ However if you remember the filenames, you can still read the files separately. It is just that **ls** won't display their names-a useful security feature.

6.5 umask: Default File Permissions

○ When you create files and directories, the default permissions that are assigned to them depend on the system's default setting.

○ The default permissions are inherited by files and directories created by all users:

rw-rw-rw- (octal 666) for regular files

rw-rw-rw- (octal 777) for directories

○ However, these are not the permissions you see when you create a file or a directory.

○ Actually, this default is transformed by subtracting the **user mask** from it to remove one or more permissions.

○ This mask is evaluated by using **umask** without arguments

\$ umask

0002

○ This is an octal number, and subtracting this value from the file default yield $666-0002=664$.

○ This represents the default permission that you normally see when you create a regular file (rw-rw—r--).

○ Similarly, the default directory permissions are also rwxrw-x-r-x ($777-0002 = 775$)

○ The umask setting can be changed

umask 666 *All permission are off except
executable permission for directories*

umask 000 *All read-write permission are on*

6.7 chgrp: Changing File Ownership

○ The **chgrp** (change group) command changes group owner of a file. In the following example user1 changes group ownership to ateferi.

\$ chgrp ateferi note1: ls -l note1

```
-rw-r--r-- 1 user1 ateferi 139 Jun 8 16:43 note1
```

To change the owner of the file, only the super user and the creator of the user accounts is allowed to do so. And he has to use the **chown** command:

```
$ chown user2 note1;
```