CHAPTER

1

Unix Getting Started

The UNIX operating system is capable of handling activities from multiple users at the same time.

What is Unix ?

The UNIX operating system is a set of programs that act as a link between the computer and the user.

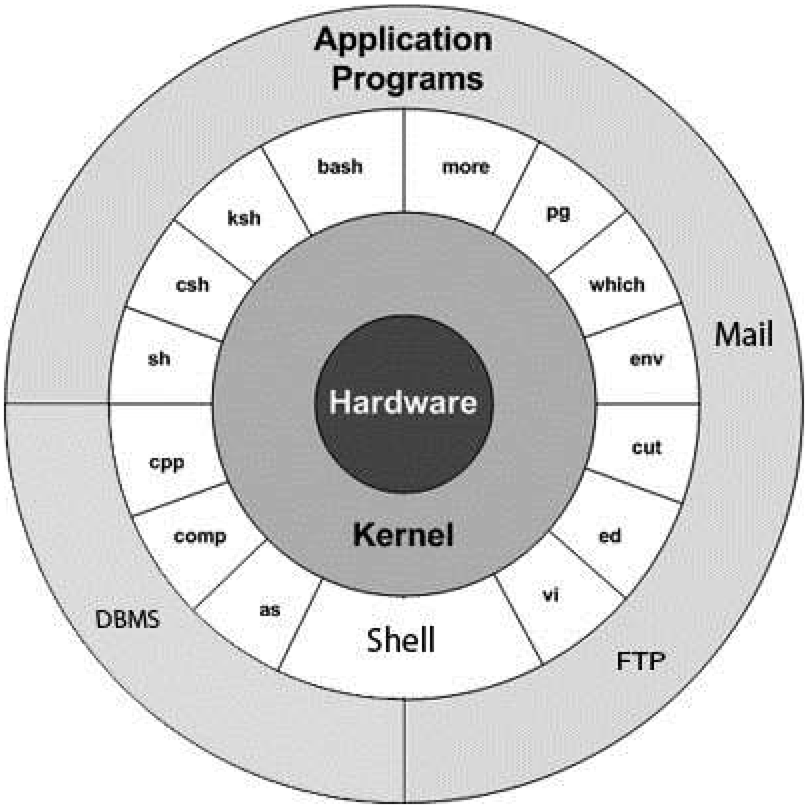
The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the operating system or kernel.

Users communicate with the kernel through a program known as the shell. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

- Unix was originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are few examples. Linux is also a flavor of Unix which is freely available.
- Several people can use a UNIX computer at the same time; hence UNIX is called a multiuser system.
- A user can also run multiple programs at the same time; hence UNIX is called multitasking.

Unix Architecture:

Here is a basic block diagram of a UNIX system:



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

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

System Bootup:

If you have a computer which has UNIX operating system installed on it, then you simply need to turn on its power to make it live.

As soon as you turn on the power, system starts booting up and finally it prompts you to log into the system, which is an activity to log into the system and use it for your day to day activities.

Login Unix:

When you first connect to a UNIX system, you usually see a prompt such as the following:

```
login:
```

To log in:

1. Have your userid (user identification) and password ready. Contact your system administrator if you don't have these yet.

2. Type your userid at the login prompt, then press ENTER. Your userid is case-sensitive, so be sure you type it exactly as your system administrator instructed.

3. Type your password at the password prompt, then press ENTER. Your password is also case-sensitive.

4. If you provided correct userid and password then you would be allowed to enter into the system. Read the information and messages that come up on the screen something as below.

```
login : amrood
amrood's password:
Last login: Sun Jun 14 09:32:32 2009 from 62.61.164.73
$
```

You would be provided with a command prompt (sometime called $ prompt) where you would type your all the commands. For example to check calendar you need to type **cal** command as follows:

```
$ cal
     June 2009
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

$
```

Change Password:

All Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and crackers. Here are the steps to change your password:

1. To start, type **passwd** at command prompt as shown below.

2. Enter your old password the one you're currently using.

3. Type in your new password. Always keep your password complex enough so that no body can guess it. But make sure, you remember it.

4. You would need to verify the password by typing it again.

```
$ passwd
Changing password for amrood
(current) Unix password:******
```

```
New UNIX password:*******
Retype new UNIX password:*******
passwd: all authentication tokens updated  successfully

$
```

Note: I have put stars (*) just to show you the location where you would need to enter the current and new passwords otherwise at your system, it would not show you any character when you would type.

Listing Directories and Files:

All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

You can use **ls** command to list out all the files or directories available in a directory. Following is the example of using **ls** command with **-l** option.

```
$ ls -l
total 19621
drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf

$
```

Here enteries starting with **d.....** represent directories. For example uml, univ and urlspedia are directories and rest of the enteries are files.

Who Are You?

While you're logged in to the system, you might be willing to know : **Who am I**?

The easiest way to find out "who you are" is to enter the **whoami** command:

```
$ whoami
 amrood

$
```

Try it on your system. This command lists the account name associated with the current login. You can try **who am i** command as well to get information about yourself.

Who is Logged In?

Sometime you might be interested to know who is logged in to the computer at the same time.

There are three commands are available to get you this information, based on how much you'd like to learn about the other users: **users, who,** and **w**.

```
$ users
 amrood bablu qadir
```

```
$ who
amrood ttyp0 Oct 8 14:10 (limbo)
bablu  ttyp2 Oct 4 09:08 (calliope)
qadir  ttyp4 Oct 8 12:09 (dent)

$
```

Try **w** command on your system to check the output. This would list down few more information associated with the users logged in the system.

Logging Out:

When you finish your session, you need to log out of the system to ensure that nobody else accesses your files while masquerading as you.

To log out:

1. Just type **logout** command at command prompt, and the system will clean up everything and break the connection

System Shutdown:

The most consistent way to shut down a Unix system properly via the command line is to use one of the following commands:

Command	**Description**
halt	Brings the system down immediately.
init 0	Powers off the system using predefined scripts to synchronize and clean up the system prior to shutdown
init 6	Reboots the system by shutting it down completely and then bringing it completely back up
poweroff	Shuts down the system by powering off.
reboot	Reboots the system.
shutdown	Shuts down the system.

You typically need to be the superuser or root (the most privileged account on a Unix system) to shut down the system, but on some standalone or personally owned Unix boxes, an administrative user and sometimes regular users can do so.

CHAPTER

2

Unix File Management

All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

When you work with UNIX, one way or another you spend most of your time working with files. This tutorial would teach you how to create and remove files, copy and rename them, create links to them etc.

In UNIX there are three basic types of files:

1. **Ordinary Files:** An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.

2. **Directories:** Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.

3. **Special Files:** Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Listing Files:

To list the files and directories stored in the current directory. Use the following command:

```
$ls
```

Here is the sample output of the above command:

```
$ls

bin        hosts  lib     res.03
ch07       hw1    pub     test_results
ch07.bak   hw2    res.01  users
docs       hw3    res.02  work
```

The command **ls** supports the **-1** option which would help you to get more information about the listed files:

```
$ls -l
total 1962188

drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
```

```
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
drwxr-xr-x  2    200    300      4096 Nov 25  2007 webthumb-1.01
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf
drwxr-xr-x 11 amrood amrood      4096 May 29  2007 zlib-1.2.3
$
```

Here is the information about all the listed columns:

1. First Column: represents file type and permission given on the file. Below is the description of all type of files.
2. Second Column: represents the number of memory blocks taken by the file or directory.
3. Third Column: represents owner of the file. This is the Unix user who created this file.
4. Fourth Column: represents group of the owner. Every Unix user would have an associated group.
5. Fifth Column: represents file size in bytes.
6. Sixth Column: represents date and time when this file was created or modified last time.
7. Seventh Column: represents file or directory name.

In the ls -l listing example, every file line began with a d, -, or l. These characters indicate the type of file that's listed.

Prefix	Description
-	Regular file, such as an ASCII text file, binary executable, or hard link.
b	Block special file. Block input/output device file such as a physical hard drive.
c	Character special file. Raw input/output device file such as a physical hard drive
d	Directory file that contains a listing of other files and directories.
l	Symbolic link file. Links on any regular file.
p	Named pipe. A mechanism for interprocess communications
s	Socket used for interprocess communication.

Meta Characters:

Meta characters have special meaning in Unix. For example ***** and **?** are metacharacters. We use ***** to match 0 or more characters, a question mark **?** matches with single character.

For Example:

```
$ls ch*.doc
```

Displays all the files whose name start with ch and ends with .doc:

```
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc ch02-1.doc c
```

Here * works as meta character which matches with any character. If you want to display all the files ending with just .doc then you can use following command:

```
$ls *.doc
```

Hidden Files:

An invisible file is one whose first character is the dot or period character (.). UNIX programs (including the shell) use most of these files to store configuration information.

Some common examples of hidden files include the files:

- **.profile:** the Bourne shell (sh) initialization script
- **.kshrc:** the Korn shell (ksh) initialization script
- **.cshrc:** the C shell (csh) initialization script
- **.rhosts:** the remote shell configuration file

To list invisible files, specify the -a option to ls:

```
$ ls -a

.         .profile       docs     lib     test_results
..        .rhosts        hosts    pub     users
.emacs    bin            hw1      res.01  work
.exrc     ch07           hw2      res.02
.kshrc    ch07.bak       hw3      res.03
$
```

- Single dot **.**: This represents current directory.
- Double dot **..**: This represents parent directory.

Note: I have put stars (*) just to show you the location where you would need to enter the current and new passwords otherwise at your system, it would not show you any character when you would type.

Creating Files:

You can use **vi** editor to create ordinary files on any Unix system. You simply need to give following command:

```
$ vi filename
```

Above command would open a file with the given filename. You would need to press key **i** to come into edit mode. Once you are in edit mode you can start writing your content in the file as below:

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
```

Once you are done, do the following steps:

- Press key **esc** to come out of edit mode.
- Press two keys **Shift + ZZ** together to come out of the file completely.

Now you would have a file created with **filemame** in the current directory.

```
$ vi filename
$
```

Editing Files:

You can edit an existing file using **vi** editor. We would cover this in detail in a separate tutorial. But in short, you can open existing file as follows:

```
$ vi filename
```

Once file is opened, you can come in edit mode by pressing key **i** and then you can edit file as you like. If you want to move here and there inside a file then first you need to come out of edit mode by pressing key **esc** and then you can use following keys to move inside a file:

- **l** key to move to the right side.
- **h** key to move to the left side.
- **k** key to move up side in the file.
- **j** key to move down side in the file.

So using above keys you can position your cursor where ever you want to edit. Once you are positioned then you can use **i** key to come in edit mode. Edit the file, once you are done press **esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

Display Content of a File:

You can use **cat** command to see the content of a file. Following is the simple example to see the content of above created file:

```
$ cat filename
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
$
```

You can display line numbers by using **-b** option along with **cat** command as follows:

```
$ cat filename -b
1   This is unix file....I created it for the first time.....
2   I'm going to save this content in this file.
$
```

Counting Words in a File:

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is the simple example to see the information about above created file:

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns:

1. First Column: represents total number of lines in the file.
2. Second Column: represents total number of words in the file.
3. Third Column: represents total number of bytes in the file. This is actual size of the file.

4. Fourth Column: represents file name.

You can give multiple files at a time to get the information about those file. Here is simple syntax:

```
$ wc filename1 filename2 filename3
```

Copying Files:

To make a copy of a file use the **cp** command. The basic syntax of the command is:

```
$ cp source_file destination_file
```

Following is the example to create a copy of existing file **filename**.

```
$ cp filename copyfile
$
```

Now you would find one more file **copyfile** in your current directory. This file would be exactly same as original file **filename**.

Renaming Files:

To change the name of a file use the **mv** command. Its basic syntax is:

```
$ mv old_file new_file
```

Following is the example which would rename existing file **filename** to **newfile**:

```
$ mv filename newfile
$
```

The **mv** command would move existing file completely into new file. So in this case you would fine only **newfile** in your current directory.

Deleting Files:

To delete an existing file use the **rm** command. Its basic syntax is:

```
$ rm filename
```

Caution: It may be dangerous to delete a file because it may contain useful information. So be careful while using this command. It is recommended to use **-i** option along with **rm** command.

Following is the example which would completely remove existing file **filename**:

```
$ rm filename
$
```

You can remove multiple files at a tile as follows:

```
$ rm filename1 filename2 filename3
$
```

Standard Unix Streams:

Under normal circumstances every Unix program has three streams (files) opened for it when it starts up:

1. **stdin :** This is referred to as *standard input* and associated file descriptor is 0. This is also represented as STDIN. Unix program would read default input from STDIN.

2. **stdout :** This is referred to as *standard output* and associated file descriptor is 1. This is also represented as STDOUT. Unix program would write default output at STDOUT

3. **stderr :** This is referred to as *standard error* and associated file descriptor is 2. This is also represented as STDERR. Unix program would write all the error message at STDERR.

CHAPTER

3

Unix Directories

A directory is a file whose sole job is to store file names and related information. All files whether ordinary, special, or directory, are contained in directories.

UNIX uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree . The tree has a single root node, the slash character (/), and all other directories are contained below it.

Home Directory:

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command:

```
$cd ~
$
```

Here **~** indicates home directory. If you want to go in any other user's home directory then use the following command:

```
$cd ~username
$
```

To go in your last directory you can use following command:

```
$cd -
$
```

Absolute/Relative Pathnames:

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute if it is described in relation to root, so absolute pathnames always begin with a /.

These are some example of absolute filenames.

```
/etc/passwd
/users/sjones/chem/notes
/dev/rdsk/Os3
```

A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood' home directory, some pathnames might look like this:

```
chem/notes
personal/res
```

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory:

```
$pwd
/user0/home/amrood

$
```

Listing Directories:

To list the files in a directory you can use the following syntax:

```
$ls dirname
```

Following is the example to list all the files contained in /usr/local directory:

```
$ls /usr/local

X11       bin          gimp       jikes       sbin
ace       doc          include    lib         share
atalk     etc          info       man         ami
```

Creating Directories:

Directories are created by the following command:

```
$mkdir dirname
```

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command:

```
$mkdir mydir
$
```

Creates the directory mydir in the current directory. Here is another example:

```
$mkdir /tmp/test-dir
$
```

This command creates the directory test-dir in the /tmp directory. The **mkdir** command produces no output if it successfully creates the requested directory.

If you give more than one directory on the command line, mkdir creates each of the directories. For example:

```
$mkdir docs pub
$
```

Creates the directories docs and pub under the current directory.

Creating Parent Directories:

Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, mkdir issues an error message as follows:

```
$mkdir /tmp/amrood/test
mkdir: Failed to make directory "/tmp/amrood/test";
No such file or directory
$
```

In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example:

```
$mkdir -p /tmp/amrood/test
$
```

Above command creates all the required parent directories.

Removing Directories:

Directories can be deleted using the **rmdir** command as follows:

```
$rmdir dirname
$
```

Note: To remove a directory make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can create multiple directories at a time as follows:

```
$rmdir dirname1 dirname2 dirname3
$
```

Above command removes the directories dirname1, dirname2, and dirname2 if they are empty. The rmdir command produces no output if it is successful.

Changing Directories:

You can use the **cd** command to do more than change to a home directory: You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as follows:

```
$cd dirname
$
```

Here, dirname is the name of the directory that you want to change to. For example, the command:

```
$cd /usr/local/bin
$
```

Changes to the directory /usr/local/bin. From this directory you can cd to the directory /usr/home/amrood using the following relative path:

```
$cd ../../home/amrood
$
```

Renaming Directories:

The mv (move) command can also be used to rename a directory. The syntax is as follows:

```
$mv olddir newdir
$
```

You can rename a directory **mydir** to **yourdir** as follows:

```
$mv mydir yourdir
$
```

The directories . (dot) and .. (dot dot)

The filename . (dot) represents the current working directory; and the filename .. (dot dot) represent the directory one level above the current working directory, often referred to as the parent directory.

If we enter the command to show a listing of the current working directories files and use the -a option to list all the files and the -l option provides the long listing, this is the result.

```
$ls -la
drwxrwxr-x    4    teacher   class   2048  Jul 16 17.56 .
drwxr-xr-x    60   root              1536  Jul 13 14:18 ..
----------    1    teacher   class   4210  May 1 08:27 .profile
-rwxr-xr-x    1    teacher   class   1948  May 12 13:42 memo
$
```

CHAPTER

4

Unix File Permission Setup

File ownership is an important component of UNIX that provides a secure method for storing files. Every file in UNIX has the following attributes:

- **Owner permissions:** The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions:** The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions:** The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators:

While using **ls -l** command it displays various information related to file permission as follows:

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

Here first column represents different access mode ie. permission associated with a file or directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x):

- The first three characters (2-4) represent the permissions for the file's owner. For example -rwxr-x**r--** represents that onwer has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example -rwxr-x**r--** represents that group has read (r) and execute (x) permission but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example -rwxr-x**r--** represents that other world has read (r) only permission.

File Access Modes:

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which are described below:

1. Read:

Grants the capability to read ie. view the contents of the file.

2. Write:

Grants the capability to modify, or remove the content of the file.

3. Execute:

User with execute permissions can run a file as a program.

Directory Access Modes:

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned:

1. Read:

Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

2. Write:

Access means that the user can add or delete files to the contents of the directory.

3. Execute:

Executing a directory doesn't really make a lot of sense so think of this as a traverse permission.

A user must have execute access to the **bin** directory in order to execute ls or cd command.

Changing Permissions:

To change file or directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod: symbolic mode and absolute mode.

Using chmod in Symbolic Mode:

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

Chmod operator	**Description**
+	Adds the designated permission(s) to a file or directory.
-	Removes the designated permission(s) from a file or directory.
=	Sets the designated permission(s).

Here's an example using testfile. Running ls -1 on testfile shows that the file's permissions are as follows:

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example chmod command from the preceding table is run on testfile, followed by ls -l so you can see the permission changes:

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
```

```
$ls -l testfile
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g=r-x testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Here's how you could combine these commands on a single line:

```
$chmod o+wx,u-x,g=r-x testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Using chmod with Absolute Permissions:

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Here's an example using testfile. Running ls -1 on testfile shows that the file's permissions are as follows:

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example chmod command from the preceding table is run on testfile, followed by ls -l so you can see the permission changes:

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Changing Owners and Groups:

While creating an account on Unix, it assigns a owner ID and a group ID to each user. All the permissions mentioned above are also assigned based on Owner and Groups.

Two commands are available to change the owner and the group of files:

1. **chown:** The chown command stands for "change owner" and is used to change the owner of a file.
2. **chgrp:** The chgrp command stands for "change group" and is used to change the group of a file.

Changing Ownership:

The chown command changes the ownership of a file. The basic syntax is as follows:

```
$ chown user filelist
```

The value of user can be either the name of a user on the system or the user id (uid) of a user on the system.

Following example:

```
$ chown amrood testfile
$
```

Changes the owner of the given file to the user **amrood**.

NOTE: The super user, root, has the unrestricted capability to change the ownership of a any file but normal users can change only the owner of files they own.

Changing Group Ownership:

The chrgp command changes the group ownership of a file. The basic syntax is as follows:

```
$ chgrp group filelist
```

The value of group can be the name of a group on the system or the group ID (GID) of a group on the system.

Following example:

```
$ chgrp special testfile
$
```

Changes the group of the given file to **special** group.

SUID and SGID File Permission:

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file /etc/shadow.
As a regular user, you do not have read or write access to this file for security reasons, but when you change your password, you need to have write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file /etc/shadow.

Additional permissions are given to programs via a mechanism known as the Set User ID (SUID) and Set Group ID (SGID) bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is true for SGID as well. Normally programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter "s" if the permission is available. The SUID "s" bit will be located in the permission bits where the owners execute permission would normally reside. For example, the command

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x  1   root   bin  19031 Feb 7 13:47  /usr/bin/passwd*
$
```

Which shows that the SUID bit is set and that the command is owned by the root. A capital letter S in the execute position instead of a lowercase s indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users:

- The owner of the sticky directory
- The owner of the file being removed
- The super user, root

To set the SUID and SGID bits for any directory try the following:

```
$ chmod ug+s dirname
$ ls -l
drwsr-sr-x 2 root root  4096 Jun 19 06:45 dirname
$
```