

## Types of File Access

By Mr. Varun kumar

The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Thus, we could provide complete protection by prohibiting access. Alternatively, we could provide free access with no protection. Both approaches are too extreme for general use. What is needed is controlled access.

Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

- **Read.** Read from the file.
- **Write.** Write or rewrite the file.
- **Execute.** Load the file into memory and execute it.
- **Append.** Write new information at the end of the file.
- **Delete.** Delete the file and free its space for possible reuse.
- **List.** List the name and attributes of the file.

Other operations, such as renaming, copying, and editing the file, may also be controlled. For many systems, however, these higher-level functions may be implemented by a system program that makes lower-level system calls. Protection is provided at only the lower level. For instance, copying a file may be implemented simply by a sequence of read requests. In this case, a user with read access can also cause the file to be copied, printed, and so on.

Many protection mechanisms have been proposed. Each has advantages and disadvantages and must be appropriate for its intended application. A small computer system that is used by only a few members of a research group, for example, may not need the same types of protection as a large corporate computer that is used for research, finance, and personnel operations.

### Access Control

The most common approach to the protection problem is to make access dependent on the identity of the user. Different users may need different types of access to a file or directory. The most general scheme to implement identity dependent access is to associate with each file and directory an **access-control list (ACL)** specifying user names and the types of access allowed for each user.

When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

This approach has the advantage of enabling complex access methodologies.

The main problem with access lists is their length. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

- Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
- The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

These problems can be resolved by use of a condensed version of the access list.

To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file:

- **Owner.** The user who created the file is the owner.
- **Group.** A set of users who are sharing the file and need similar access is a group, or work group.
- **Universe.** All other users in the system constitute the universe.

## 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, the first column represents different access modes, i.e., the permission associated with a file or a 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-xr--** represents that the owner 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-xr--** represents that the 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-xr--** represents that there is **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 have been described below –

**Read:** Grants the capability to read, i.e., view the contents of the file.

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

**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 –

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

**Write :** Access means that the user can add or delete files from the directory.

**Execute:** Executing a directory doesn't really make sense, so think of this as a traverse permission.

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

## Changing Permissions

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

Sr.No	Chmod operator & Description
1	+ Adds the designated permission(s) to a file or directory.
2	- Removes the designated permission(s) from a file or directory.
3	= Sets the designated permission(s).

### 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

## 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 the Owner and the Groups.

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

- **chown** – The **chown** command stands for "**change owner**" and is used to change the owner of a file.
- **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 the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept –

```
$ chown amrood testfile
$
```

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

## Changing Group Ownership

The **chgrp** 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 helps you understand the concept –

```
$ chgrp special testfile
```

```
$
```

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

links to the file, the owner's name, the group's name, the size of the file in bytes, the date of last modification, and finally the file's name (with optional extension).