CHAPTER-7 CONTROL ACCESS TO FILES

Interpret Linux File System Permissions

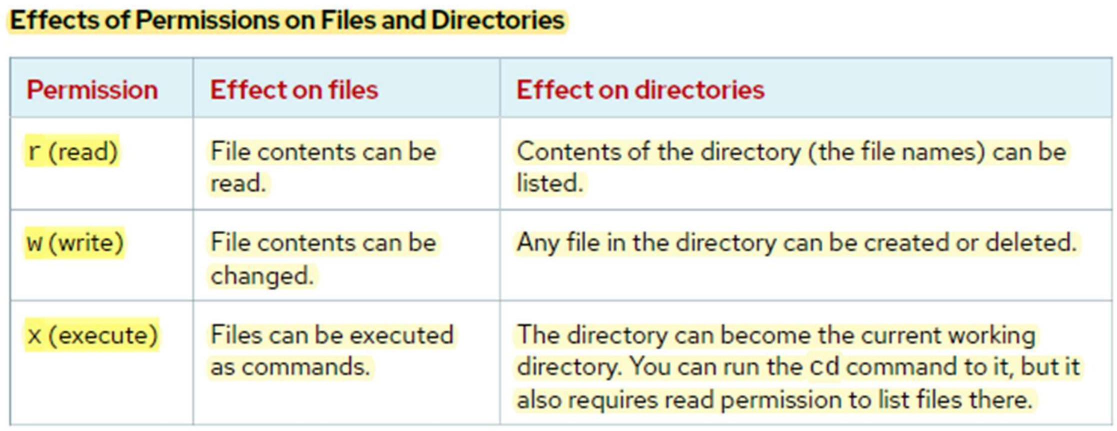
File permissions control access to files. Linux file permissions are simple but flexible and able to handle most normal permission cases.

Files have three user categories that permissions apply to. The file is owned by a user, normally the file creator. The file is also owned by a single group, which is usually the primary group of the user who created the file, but you can change it.

You can set different permissions for the owning user (user permissions), the owning group (group permissions), and for all other users on the system that are not the user or a member of the owning group (other permissions).

The most specific permissions take precedence. User permissions override group permissions, which override other permissions.

Three permission categories apply: **read, write, and execute**. The following table explains how these permissions affect access to files and directories.



Effects of Permissions on Files and Directories

Permission	Effect on files	Effect on directories
r (read)	File contents can be read.	Contents of the directory (the file names) can be listed.
w (write)	File contents can be changed.	Any file in the directory can be created or deleted.
x (execute)	Files can be executed as commands.	The directory can become the current working directory. You can run the cd command to it, but it also requires read permission to list files there.

View File and Directory Permissions and Ownership

The ls command -l option shows detailed information about permissions and ownership:

```
[user@host ~]$ ls -l test
-rw-rw-r--. 1 student student 0 Mar 8 17:36 test
```

Use the ls command -d option to show detailed information about a directory itself, and not its contents.

```
[user@host ~]$ ls -ld /home
drwxr-xr-x. 5 root root 4096 Feb 31 22:00 /home
```

The first character of the long listing is the file type, and is interpreted as follows:

-	is a regular file.
d	is a directory.
l	is a symbolic link.
c	is a character device file.
b	is a block device file.
p	is a named pipe file.
s	is a local socket file.

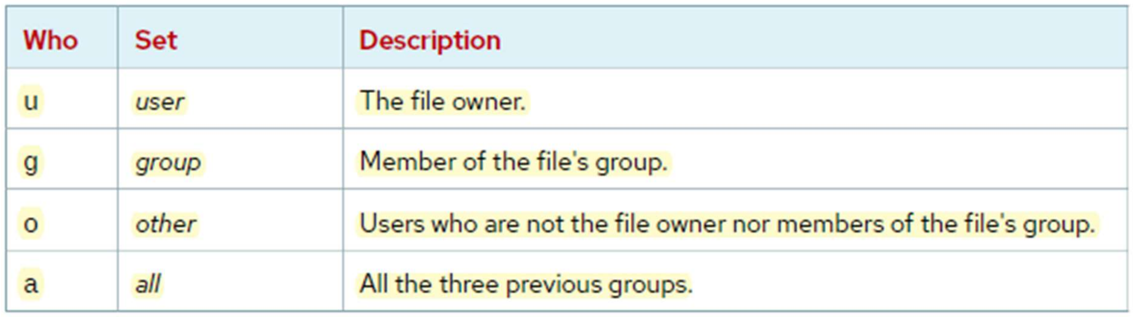
The **next nine characters represent** the **file permissions**. These characters are interpreted as **three sets of three characters**: the **first set** are **permissions** that apply to the **file owner**, the **second set** are for the **file's group owner**, and the **last set** applies to **all other** (world) users. A letter is replaced by a **-** dash character, then that set does not have that permission.

Manage File System Permissions from the Command Line

The chmod command changes file and directory permissions from the command line. The chmod command can be interpreted as "change mode", because the mode of a file is another name for file permissions. The chmod command takes a permission instruction followed by a list of files or directories to change. You can set the permission instruction either symbolically or in octal (numeric) notation.

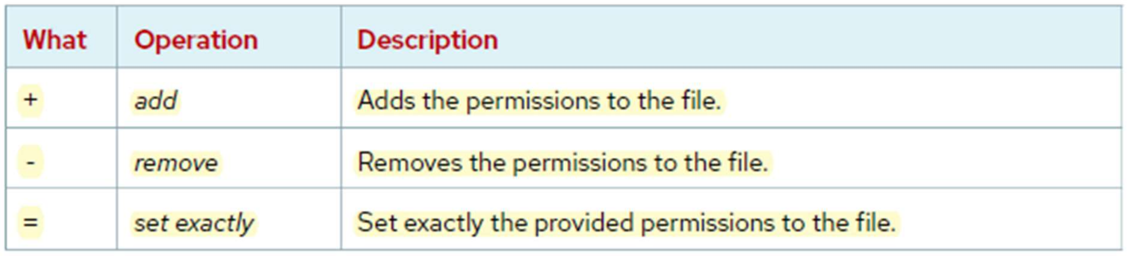
Change Permissions with the Symbolic Method

Who is the class of user, as in the following table. If you do not provide a class of user, then the chmod command uses the all group as default.



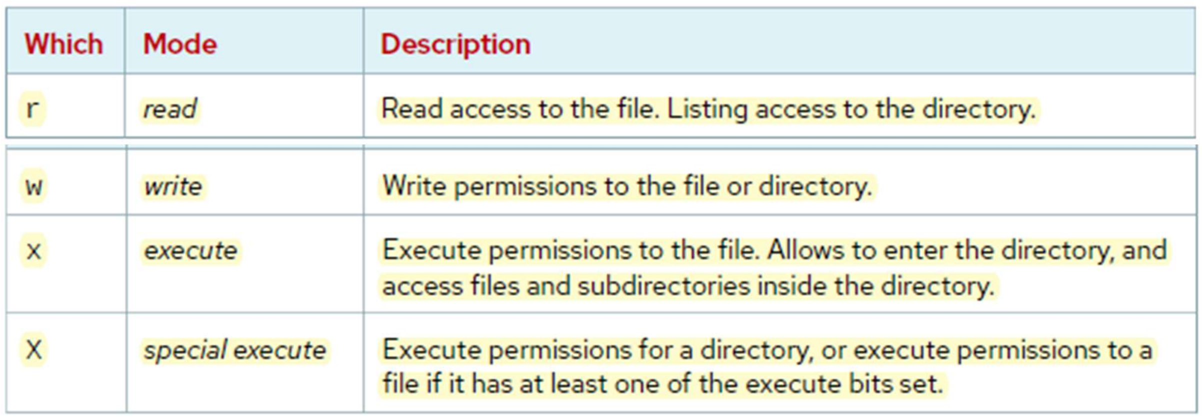
Who	Set	Description
u	*user*	The file owner.
g	*group*	Member of the file's group.
o	*other*	Users who are not the file owner nor members of the file's group.
a	*all*	All the three previous groups.

What is the operator that modifies the which, as in the table below.



What	Operation	Description
+	*add*	Adds the permissions to the file.
-	*remove*	Removes the permissions to the file.
=	*set exactly*	Set exactly the provided permissions to the file.

Which is the mode, and specifies the permissions to the files or directories, as in the table below.



Which	Mode	Description
r	*read*	Read access to the file. Listing access to the directory.
w	*write*	Write permissions to the file or directory.
x	*execute*	Execute permissions to the file. Allows to enter the directory, and access files and subdirectories inside the directory.
X	*special execute*	Execute permissions for a directory, or execute permissions to a file if it has at least one of the execute bits set.

The symbolic method of changing file permissions uses letters to represent the different groups of permissions: **u for user, g for group, o for other,** and **a for all**.

DESCRIPTION	**COMMANDS / OPTIONS**
To Add remove permission file:	**Syntax:** chmod [Options] [permission] files/dir **Example:** To Remove read and write permission for group and other on the file1.txt file: [root@host ~]# **chmod go-rw file1.txt** **Example:** Add execute permission for everyone on the script.sh file: [root@host ~]# **chmod a+x script.sh** **-R** To recursively set permissions on the files in an entire directory tree. **-X** To allows you to set the execute (search) permission on directories so that their contents can be accessed, without changing permissions on most files.

Change Permissions with the Octal Method

DESCRIPTION	COMMANDS / OPTIONS
Read	4
Write	2
Execute	1
DESCRIPTION	**COMMANDS / OPTIONS**
To Add remove permission file:	**Syntax:** chmod [Options] [permission] files/dir **Example:** To Set read and write permissions for user, and read permission for group and other, on the File1.txt file: [root@host ~]# **chmod 644 file1.txt** **Example:** Set read, write, and execute permissions for user, read and execute permissions for group, and no permission for other on the script directory: [root@host ~]# **chmod 750 script** **-R** To recursively set permissions on the files in an entire directory tree. **-X** To allows you to set the execute (search) permission on directories so that their contents can be accessed, without changing permissions on most files.

Change File and Directory User or Group Ownership

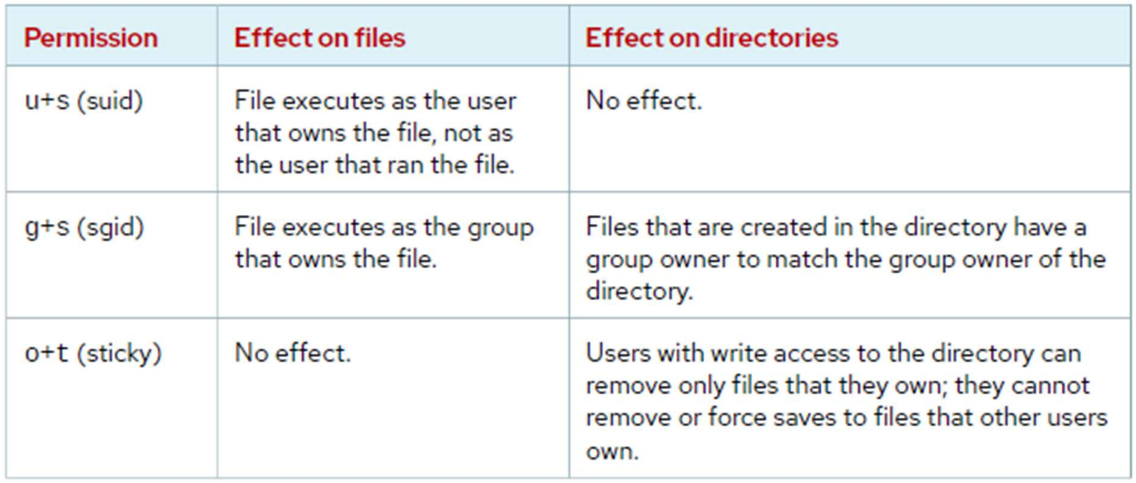
To grant access to a file based on group membership, you might need to change the group that owns the file.

DESCRIPTION	COMMANDS / OPTIONS
To change ownership of user or group	**Syntax:** chown[Options] [user] files/dir **Example:** To grant ownership of the dhcp.conf file to the student user, use the following command: [root@host ~]# **chown student dhcp.conf** **-R** recursively changes the ownership of an entire directory tree **Example:** To change group ownership of a file by preceding the **group name** with a **colon (:)** For example, the following command changes the

	group ownership of the Video directory to admins: [root@host ~]# **chown :admins Video** **To change both owner and group at the same time** by using the **owner:group** syntax **Example:** To change the ownership of the Video directory to the visitor user and the group to guests [root@host ~]# **chown visitor:guests Pictures**

Manage Default Permissions and File Access

Special permissions are a **fourth permission** type in addition to the basic user, group, and other types. As the name implies, special permissions provide additional access-related features beyond what the basic permission types allow. This section describes the impact of special permissions, which are summarized in the table below.



Permission	Effect on files	Effect on directories
u+s (suid)	File executes as the user that owns the file, not as the user that ran the file.	No effect.
g+s (sgid)	File executes as the group that owns the file.	Files that are created in the directory have a group owner to match the group owner of the directory.
o+t (sticky)	No effect.	Users with write access to the directory can remove only files that they own; they cannot remove or force saves to files that other users own.

DESCRIPTION	**COMMANDS / OPTIONS**
Setting Special Permissions	**Symbolic :** setuid = u+s; setgid = g+s; sticky = o+t **Add the setgid bit** on the example directory by using the symbolic method: [user@host ~]# **chmod g+s example** **Remove the setuid bit** on the example directory by using the symbolic method: [user@host ~]# **chmod u-s example**

Set the setgid bit and add read, write, and execute permissions for user and group, with no
access for others, on the example directory by using the octal method:

```
[user@host ~]# chmod 2770 example
```

Remove the setgid bit and add read, write, and execute permissions for user and group, with no
access for others:

```
[user@host ~]# chmod 0770 example
```

Set-user-ID (SUID)

chmod u+s FileName

s identify the setuid permissions by **a lowercase s** character in the place where you would normally expect the **x** character

S If the owner does not have execute permissions, then this character is replaced by an **uppercase S** character.

remove SUID

chmod u-s FileName

Set-group-ID bit

chmod g+s FileName

remove SGID

chmod g-s FileName

The Sticky Bit

chmod +t FileName

t identify the sticky permissions by **a lowercase** t character in the place where you would normally expect the **x** character

T If other does not have execute permissions, then this character is replaced by **an uppercase** T character.

remove Sticky Bit

chmod -t FileName

Default File Permissions

On creation, a file is assigned initial permissions. Two things affect these initial permissions. The first is whether you are creating a regular file or a directory. The second is the current umask, which stands for user file-creation mask.

If you **create a directory**, then **its initial octal permissions are 0777** (drwxrwxrwx). If you **create a regular file**, then its **initial octal permissions are 0666** (-rw-rw-rw-).

The shell session **sets a umask** to further **restrict the initial permissions** of a file. The umask is an **octal bitmask** that **clears the permissions** of **new files and directories** that a process creates. If a bit is set in the umask, then the corresponding permission is cleared on new files.

Example: The umask 0002 clears the write bit for other users. The leading zeros indicate that the special, user, and group permissions are not cleared. A umask of 0077 clears all the group and other permissions of newly created files.

The umask command without arguments displays the current value of the shell's umask:
[user@host ~]$ **umask**

The system's default umask values for Bash shell users are defined in the **/etc/login.defs** file, and the **/etc/bashrc** file. Users can override the system defaults in the **.bash_profile** or **.bashrc** files in their home directories.

DESCRIPTION	COMMANDS / OPTIONS
To change the group ownership of a file or directory	**Syntax:** chgrp [OPTION] GROUP FILE **-R** To recursively change the group ownership **-c** The action for each File whose group actually changes **-f** To suppress error messages
ACCESS CONTROL LIST (ACL)	**To get details about permission** **Syntax:** getfacl[file/dir] **To set permission** **Syntax:** setfacl[options][ugo]:Name : [Permission]File/Dir **-m** (modify) **-x** (remove) **-d** (defaults) **-b** To remove all entries
	To list the attributes of files or directories

Attributes of a file in a directory	**Syntax:** lsattr [options] [files/directories] **-a:** Lists all files and directories, including hidden **-d:** If the argument is a directory, list the attributes of the directory itself rather than its contents. **-R:** Recursively lists the attributes of directories and their contents. **To changing the attributes of a file in a directory** **Syntax:** chattr [-RVf] [-v version] [mode] files... **-R:** It is used to display the list attributes of directories and their contents recursively. **-V:** It will display the version of the program. **-a:** Used to list all the files of a directory which also includes the whose name starts with a Period('.'). **-d:** This option will list the directories as regular files instead of listing their contents. **-v:** Used to display the file's version/generation number etc. **'+'** : Adding selected attributes to the existing attributes of the files. **'–' :** Causes selected attributes to be removed. **'=' :** Causes selected attributes to be the only attributes that the files have.