

## 4.11. Ownership and Permissions

As a regular user, try to enter root's home directory by entering the command `cd /root/`. Note the error message:

```
-bash: cd: /root/: Permission denied
```

That was one demonstration of Linux's security features. Linux, like UNIX, is a multi-user system and file permissions are one way the system protects against malicious tampering.

One way to gain entry when you are denied permission is to enter the command `su -`. This is because whoever knows the root password has complete access.

However, switching to the superuser is not always convenient or recommended, since it is easy to make mistakes and alter important configuration files as the superuser.

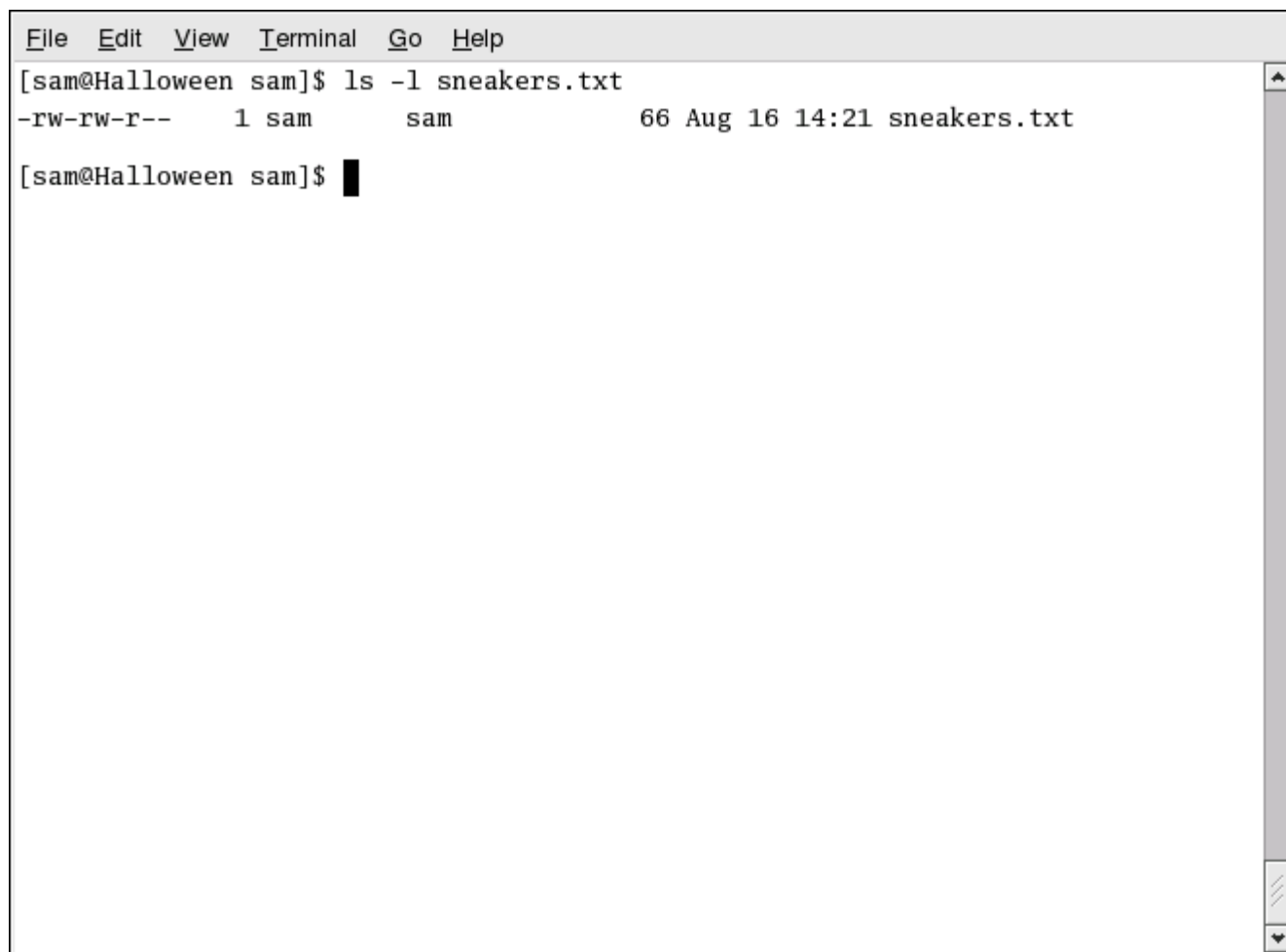
All files and directories are "owned" by the person who created them. You created the file `foo.txt` (refer to [Section 4.9.2 Using Redirection](#)) in your login directory, so `foo.txt` belongs to you.

That means you can specify who is allowed to read the file, write to the file, or (if it is an application instead of a text file) who can execute the file.

Reading, writing, and executing are the three main settings in permissions. Since users are placed into a group when their accounts are created, you can also specify whether certain groups can read, write to, or execute a file.

Take a closer look at `foo.txt` with the `ls` command using the `-l` option (refer to [Figure 4-11](#)).

A lot of detail is provided here. You can see who can read (r) and write to (w) the file, as well as who created the file (user), and to which group the owner belongs (user). (By default, the name of your group is the same as your login name.)

A terminal window with a menu bar (File, Edit, View, Terminal, Go, Help) and a scrollbar on the right. The command `ls -l sneakers.txt` has been executed, showing the output: `-rw-rw-r-- 1 sam sam 66 Aug 16 14:21 sneakers.txt`. The prompt `[sam@Halloween sam]$` is followed by a cursor.

```
File Edit View Terminal Go Help
[sam@Halloween sam]$ ls -l sneakers.txt
-rw-rw-r-- 1 sam sam 66 Aug 16 14:21 sneakers.txt
[sam@Halloween sam]$
```

**Figure 4-11. Permissions for foo.txt**

Other information to the right of the group includes file size, date and time of file creation, and file name.

The first column shows current permissions; it has ten slots. The first slot represents the type of file. The remaining nine slots are actually three sets of permissions for three different categories of users.

For example:

```
-rw-rw-r--
```

Those three sets are the owner of the file, the group in which the file belongs, and "others," meaning other users on the system.

```
-      (rw-)      (rw-)      (r--) 1 user user
```

The first item, which specifies the file type, will probably be one of the following:

- `d` — a directory
- `-` (dash) — a regular file (rather than directory or link)
- `l` — a symbolic link to another program or file elsewhere on the system

Others are possible, but are beyond the scope of this manual. Refer to the Red Hat Enterprise Linux System Administration Guide for more information.

Beyond the first item, in each of the following three sets, you may see one of the following:

- `r` — file can be read
- `w` — file can be written to
- `x` — file can be executed (if it is a program)
- `-` (dash) — specific permission has not been assigned

When you see a dash in owner, group, or others, it means that particular permission has not been granted. Look again at the first column of `foo.txt` and identify its permissions.

```
ls -l foo.txt
-rw-rw-r-- 1 user user 150 Mar 19 08:08 foo.txt
```

The file's owner (in this case, `user`) has permission to read and write to the file. The group, `user`, has permission to read and write to `foo.txt`, as well. It is not a program, so neither the owner or the group has permission to execute it.

### 4.11.1. The `chmod` Command

Use the `chmod` command to change permissions. This example shows how to change the permissions on `foo.txt` with the `chmod` command.

The original file looks like this, with its initial permissions settings:

```
-rw-rw-r-- 1 user user 150 Mar 19 08:08 foo.txt
```

If you are the owner of the file or are logged into the root account, you can change any permissions for the owner, group, and others.

Right now, the owner and group can read and write to the file. Anyone outside of the group can only read the file (`r--`).



#### Caution

Remember that file permissions are a security feature. Whenever you allow anyone else to read, write to, and execute files, you are increasing the risk of files being tampered with, altered, or deleted. As a rule, you should only grant read and write permissions to those who truly need them.

In the following example, you want to allow everyone to write to the file, so they can read it, write notes in it, and save it. That means you must change the "others" section of the file permissions.

Take a look at the file first. At the shell prompt, type:

```
ls -l foo.txt
```

The previous command displays this file information:

```
-rw-rw-r-- 1 user user 150 Mar 19 08:08 foo.txt
```

Now, type the following:

```
chmod o+w foo.txt
```

The `o+w` command tells the system you want to give others write permission to the file `foo.txt`. To check the results, list the file's details again. Now, the file looks like this:

```
-rw-rw-rw- 1 user user 150 Mar 19 08:08 foo.txt
```

Now, everyone can read and write to the file.

To remove read and write permissions from `foo.txt` use the `chmod` command to take away both the read and write permissions.

```
chmod go-rw foo.txt
```

By typing `go-rw`, you are telling the system to remove read and write permissions for the group and for others from the file `foo.txt`.

The result looks like this:

```
-rw----- 1 user user 150 Mar 19 08:08 foo.txt
```

Think of these settings as a kind of shorthand when you want to change permissions with `chmod`, because all you really have to do is remember a few symbols and letters with the `chmod` command.

Here is a list of what the shorthand represents:

### Identities

- u — the user who owns the file (that is, the owner)
- g — the group to which the user belongs
- o — others (not the owner or the owner's group)
- a — everyone or all (u, g, and o)

### Permissions

- r — read access
- w — write access
- x — execute access

### Actions

- + — adds the permission

- — removes the permission
- = — makes it the only permission

Want to test your permissions skills? Remove all permissions from `foo.txt` — for everyone.

```
chmod a-rwx foo.txt
```

Now, see if you can read the file with the command `cat foo.txt`, which should return the following:

```
cat: foo.txt: Permission denied
```

Removing all permissions, including your own, successfully locked the file. But since the file belongs to you, you can always change its permissions back with the following command:

```
chmod u+rw foo.txt
```

Use the command `cat foo.txt` to verify that you, the file owner, can read the file again.

Here are some common examples of settings that can be used with `chmod`:

- `g+w` — adds write access for the group
- `o-rwx` — removes all permissions for others
- `u+x` — allows the file owner to execute the file
- `a+rw` — allows everyone to read and write to the file
- `ug+r` — allows the owner and group to read the file
- `g=rx` — allows only the group to read and execute (not write)

By adding the `-R` option, you can change permissions for entire directory trees.

Because you can not really "execute" a directory as you would an application, when you add (or remove) the execute permission for a directory, you are really allowing (or denying) permission to search through that directory.

Examine the `dir1/` directory you created in section `FIXME` by listing all of the files in your home directory.

```
ls -l /home/>user</
```

The permissions on this directory are:

```
drwxrwxr-x  2 mgoldin mgoldin 4096 Jan  6 15:05 dir1
```

If you do not allow others to have execute permission on the `dir1/` directory, it does not matter who has read or write access. No one can access the directory unless they know the exact file name.

For example, type

```
chmod a-x dir1/
```

to remove everyone's execute permissions.

Here is what happens when you try to change directories using the `cd dir1/` command after removing everyone's execute permissions:

```
bash: dir1/: Permission denied
```

Next, restore your own and your group's access:

```
chmod ug+x dir1/
```

If you check your work with `ls -l`, you can see that only others are denied access to the `/dir1/` directory.

### 4.11.2. Changing Permissions With Numbers

Another way to change permissions uses numeric representations.

Go back to the original permissions for `foo.txt`:

```
-rw-rw-r--    1 user user    150 Mar 19 08:08 foo.txt
```

Each permission setting can be represented by a numerical value:

- `r = 4`
- `w = 2`
- `x = 1`
- `- = 0`

When these values are added together, the total is used to set specific permissions. For example, if you want read and write permissions, you would have a value of 6; 4 (read) + 2 (write) = 6.

For `foo.txt`, here are the numerical permissions settings:

```
-  (rw-)  (rw-)  (r--)
```

The total for the user is six(4+2+0), the total for the group is six(4+2+0), and the total for others is four(4+0+0). The permissions setting is read as `664`.

If you want to change `foo.txt` so those in your group do not have write access, but can still read the file, remove the access by subtracting two (2) from that set of numbers.

The numerical values then become six, four, and four (644).

To implement these new settings, type:

```
chmod 644 foo.txt
```

Now verify the changes by listing the file. Type:

```
ls -l foo.txt
```

The output should be:

```
-rw-r--r--    1 user user    150 Mar 19 08:08 foo.txt
```

Now, neither the group nor others have write permission to `foo.txt`. To return the group's write access for the file, add the value of `w` (2) to the second set of permissions.

```
chmod 664 foo.txt
```



### Warning

Setting permissions to 666 allows everyone to read and write to a file or directory. Setting permissions to 777 allows everyone read, write, and execute permission. These permissions could allow tampering with sensitive files, so in general, it is not a good idea to use these settings.

Here is a list of some common settings, numerical values and their meanings:

Setting	Numerical	Meaning
<code>-rw-----</code>	(600)	Only the owner has read and write permissions.
<code>-rw-r--r--</code>	(644)	Only the owner has read and write permissions; the group and others have read only.
<code>-rwx-----</code>	(700)	Only the owner has read, write, and execute permissions.
<code>-rwxr-xr-x</code>	(755)	The owner has read, write, and execute permissions; the group and others have only read and execute.
<code>-rwx--x--x</code>	(711)	The owner has read, write, and execute permissions; the group and others have only execute.
<code>-rw-rw-rw-</code>	(666)	Everyone can read and write to the file. (Be careful with these permissions.)
<code>-rwxrwxrwx</code>	(777)	Everyone can read, write, and execute. (Again, this permissions setting can be hazardous.)

**Table 4-3. File permissions settings, numerical values, and their meanings**

Here are some common settings for directories:

Setting	Numerical	Meaning
drwx-----	(700)	Only the user can read, write in this directory.
drwxr-xr-x	(755)	Everyone can read the directory; users and groups have read and execute permissions.

**Table 4-4. Directory permissions settings, numerical values, and their meanings**[Prev](#)

Using Multiple Commands

[Home](#)[Up](#)[Next](#)

Using Your System