



Managing File Permissions

Linux Fundamentals

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Welcome to Managing File Permissions.

What you will learn

At the core of the lesson

You will learn how to:

- View and change file permissions
- Compare symbolic and absolute representations of file permissions



You will learn how to:

- View and change file permissions
- Compare symbolic and absolute representations of file permissions



View and change file permissions

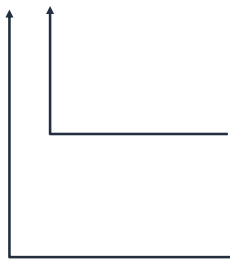
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In this section, you will review how to view and change file permissions.

Linux permission types

```
-rwxr-xr-x
```

-rwx



Read (r): This permission gives the user control to open and read a file.

Write (w): This permission gives the user control to change the file's content.

Execute (x): This permission gives the user the ability to run a program.

File type: The file type indicates the regular file in a directory.

In Linux, the term *permissions* refers to how someone can use a file or directory.

Files and directories in a Linux system have the following three permissions:

- **Read:** The read permission gives the control to open and read a file. For example, the read permission on a directory lets you print out the file's content.
- **Write:** The write permission gives the user control to change the file's content. For example, the write permission on a directory gives the control to add, remove, and rename files that are stored in the directory. A user might have write permission on a file that is stored in a directory but does not have write permission for the directory itself. In that case, the user can change the file contents. However, the user cannot rename or remove the file from the directory.
- **Execute:** In Linux, you must have this permission to run a program. If the permission is not set, a user can see or change the program code (if read and write permissions are set) but not run it.

Permission modes

```
]# chmod g+w file_1
```

An example of symbolic mode

```
]# chmod 764 file_2
```

An example of absolute mode

Symbolic mode

```
[root@server00 Documents]# chmod g+w file1
```

Absolute mode

```
[root@server00 Documents]# chmod 764 file2
```

Linux uses two modes to configure permissions. You must be able to interpret and use both modes, though you might prefer one over the other.

The two modes are **absolute mode** and **symbolic mode**.

- **Absolute mode:** Use numbers to represent file permissions. This mode is the most commonly used to set permissions.
- **Symbolic mode:** Use a combination of letters and symbols to add permissions or to remove any set permissions.

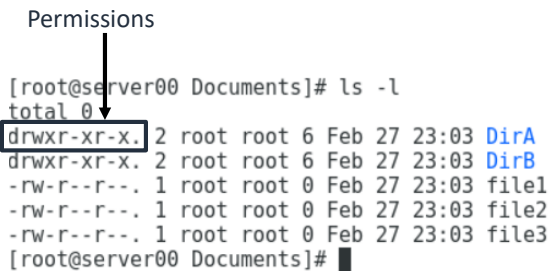
Note: With the `chmod` command, the user can change the permissions on a file. You must be a superuser or the owner of a file or directory to change its permissions.

Using the `ls -l` command to view permissions

```
]# ls -l
```

The `ls` command is used to list files and directories. Option `-l` shows the file or directory, size, modified date and time, file or folder name, and owner of file and its **permissions**.

Permissions

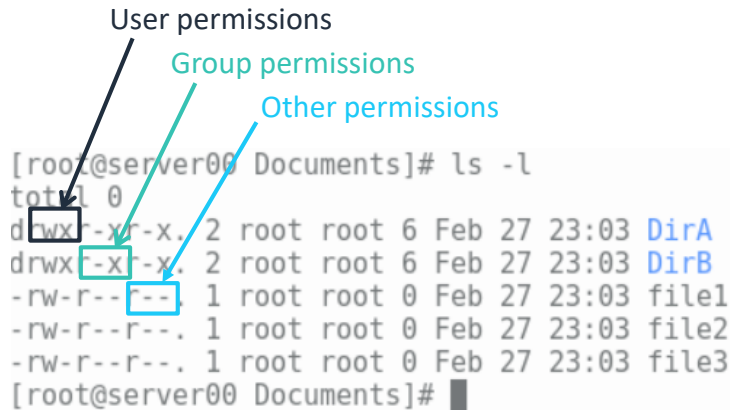


```
[root@server00 Documents]# ls -l
total 0
drwxr-xr-x. 2 root root 6 Feb 27 23:03 DirA
drwxr-xr-x. 2 root root 6 Feb 27 23:03 DirB
-rw-r--r--. 1 root root 0 Feb 27 23:03 file1
-rw-r--r--. 1 root root 0 Feb 27 23:03 file2
-rw-r--r--. 1 root root 0 Feb 27 23:03 file3
[root@server00 Documents]#
```

The `ls -l` command is used to print out (view) the file's or directory's permissions details. Option `-l` shows the file or directory, size, modified date and time, file or folder name, and owner of a file and its *permissions*.

When the command is used, you see a list of different essential file attributes that you need to know about the file permissions.

Understanding the differences



The image shows a terminal window with the command `ls -l` executed in the `Documents` directory. The output lists five items: `DirA`, `DirB`, `file1`, `file2`, and `file3`. Annotations with colored arrows point to specific parts of the permission strings:

- User permissions:** Points to the first three characters of the permission string for `DirA` (`dwx`).
- Group permissions:** Points to the next three characters of the permission string for `DirA` (`-xr-x`).
- Other permissions:** Points to the last three characters of the permission string for `DirA` (`-x`).

```
[root@server00 Documents]# ls -l
total 0
dwx-xr-x. 2 root root 6 Feb 27 23:03 DirA
drwxr-x-x. 2 root root 6 Feb 27 23:03 DirB
-rw-r--r--. 1 root root 0 Feb 27 23:03 file1
-rw-r--r--. 1 root root 0 Feb 27 23:03 file2
-rw-r--r--. 1 root root 0 Feb 27 23:03 file3
[root@server00 Documents]#
```

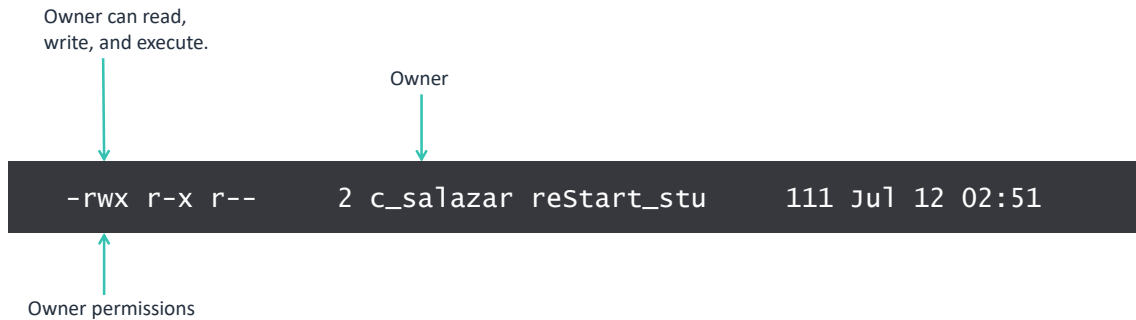
Files and directories in a Linux system assign three types of ownership. The three ownership types are as follows.

Note: A *user* is the owner of the file. Every file in the system belongs to precisely one user name, which is also called the *file owner*.

- **User:** A *user* can create a new file or directory. The file's ownership is set to the user ID of the user who created the file.
- **Group:** A user *group* can contain multiple users. Users who belong to that group will have the same Linux group permissions to access the file. Every file is also associated with one group name, which is called the *group owner*.
- **Other:** *Other* ownership means that the user did not create the file and does not belong to a user group that could own the file.

In the following few slides, you will look at permissions in Linux.

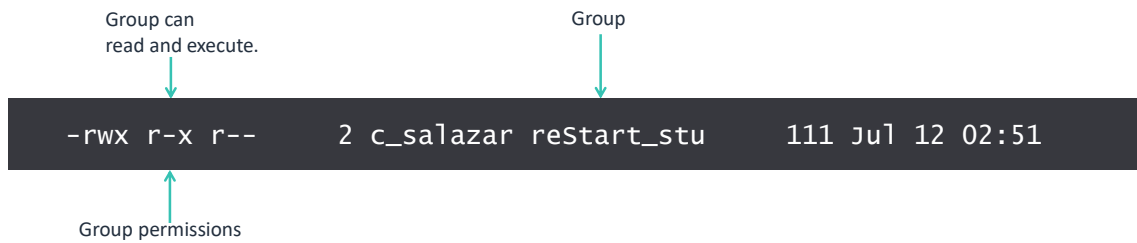
User or owner identity



The file owner controls permissions, which apply to the user identity or name.

The owner of the file is displayed to the right of where the permissions are displayed. The file owner controls permissions, and the permissions are set for the owner and apply to that user identity or name.

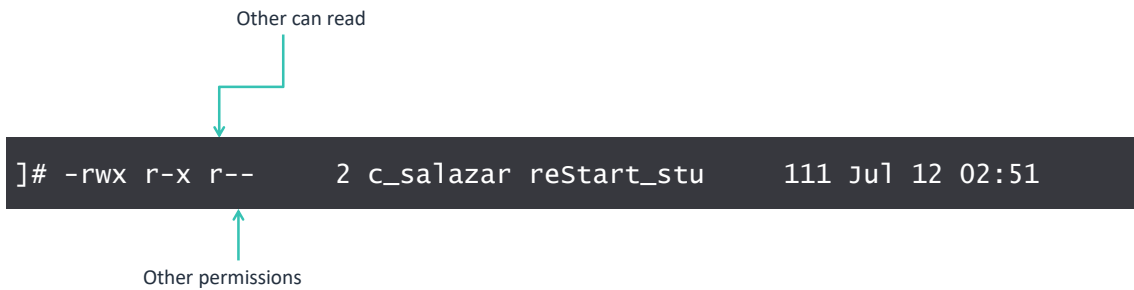
Group identity



Group members are granted permissions of the group to a file or directory.

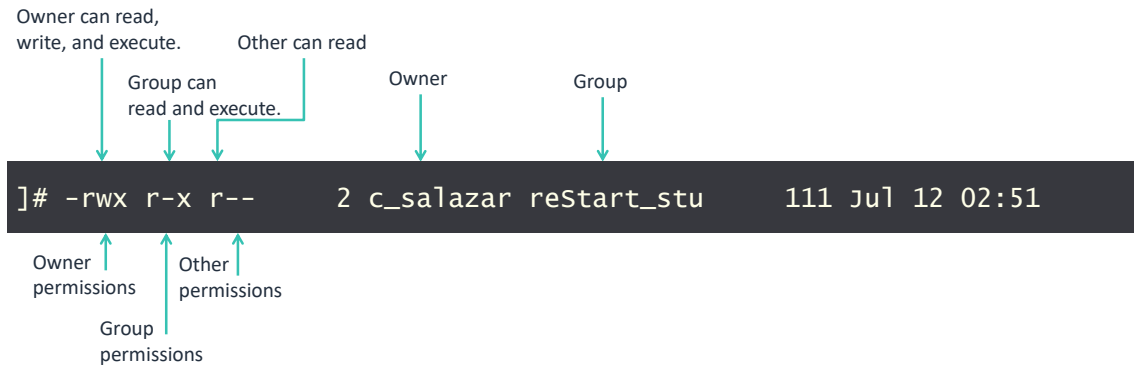
Group permissions can be used to simplify administrative tasks.

Other identity



Any user who is not the owner and not a member of an associated group is part of *other* for purposes of permissions.

Output for a full executable using the `ls -l` command



The example shows what is generated when the `ls -l` command is used.

Default permissions

When a standard user creates a file or directory by using the (non-root) `userA@`, or `username@`:

```
~]$ sudo command
```

When the root user creates a file or directory by using the `root@`:

```
]# command
```

The root account has superuser privileges and can run commands. Non-root users can perform or run commands that are similar to root users who use the **sudo** command. The user can run a one-off command by using the **sudo** command. The user will be prompted to enter the account user's password after entering the **sudo** command.

Note: Default permissions might vary by distribution.

chown command

```
[root@hostname ~]# chown [options] user[:group] file(s)
```

chown new-owner:new-group file1

↑ ↑ ↑ ↑
command user group file

chown new-owner file1

↑ ↑ ↑
command user file

chown :new-group file1

↑ ↑ ↑
command group file

```
[root@ip-10-0-4-100 projects]# chown jstiles file2
[root@ip-10-0-4-100 projects]# chown arosalez file3
[root@ip-10-0-4-100 projects]# chown :sales file1
[root@ip-10-0-4-100 projects]# █
```

The user (owner) and associated group for a file or directory can be changed. The **chown** command should be used to change the ownership of a file. The following list further explains how the **chown** command is used.

- **[options]** – The **chown** command can be used with or without options.
- **[user]** – This information indicates the user name or user ID of the new owner of the file or directory that is being altered.
- **[:]** – The colon is used when changing a group of the file or directory.
- **[group]** – Changing the ownership of the associated group is optional.
- **[file(s)]** – This is the source file or directory that will be altered.

Symbolic and absolute representations of file permissions

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In this next section, you will see a comparison of symbolic and absolute representations of file permissions.

chmod command

```
chmod [permissions] file1.txt
```

The command that is used to change permissions is the `chmod` command. The *change mode* or `chmod` command is used to set permissions on files and directories. The `chmod` syntax could be also be called the file's *mode*.

chmod command in symbolic mode

| Identity | Permission | Operator |
|--------------------------|--------------------|--|
| u (user or owner) | r (read) | + Grants a permission |
| g (group) | w (write) | - Removes a permission |
| o (other) | x (execute) | = Removes a permission and sets a new one |

```
]$ chmod u+x process.sh  
]$ chmod g=x process.sh  
]$ chmod g-rw Roseter.csv  
]$ chmod o-r process.sh
```

u+x – User granted the execute permission

g=x – Group execute permission removed

g-rw – Group read and write permission removed

```
-rwx--x--- 1 labsuser labsuser 0 Mar 29 07:48 process.sh  
-rw----r-- 1 labsuser labsuser 0 Mar 29 07:47 Roseter.csv
```

The slide shows some examples of how to change permissions in the symbolic mode. If the user is not a superuser, the user might not have the ability to change ownership of the file or directory. Only a superuser or root account can use the **chmod** command to change file or directory permissions.

chmod command in absolute mode

| Permission | Value |
|-----------------|-------|
| Read | 4 |
| Write | 2 |
| Execute | 1 |
| All permissions | 7 |

```
[root@server00 Documents]# chmod 764 file1
[root@server00 Documents]# chmod 740 file2
[root@server00 Documents]# chmod 777 file3
[root@server00 Documents]# ls -l
total 0
drwxr-xr-x. 2 root root 6 Feb 27 23:03 DirA
drwxr-xr-x. 2 root root 6 Feb 27 23:03 DirB
-rwxrw-r--. 1 root root 0 Feb 27 23:03 file1
-rwxr-----. 1 root root 0 Feb 27 23:03 file2
-rwxrwxrwx. 1 root root 0 Feb 27 23:03 file3
[root@server00 Documents]#
```

With the absolute mode, file permissions can be changed by using the octal number that represents the file permission.

```
]# chmod 400 file_1
```

The output would be the following:

```
]# - r-----. 2 c_salazar reStart_stu 111 Jul 12 02:51 file_1
```

On the slide, the examples show how to change permissions in the absolute mode. When specifying permissions in this mode, you use a three-digit octal number to specify the owner, group, and other permissions. For example, consider what happens if you write the following command:

```
chmod 400 file_1
```

With this output, the user can only read the file.

Consider the following question: Why should the command `chmod 777` be avoided?

The `chmod 777` command should be avoided because it grants read, write, and execute permissions to every user.

Demonstration: Identifying and configuring file permissions



In this demonstration, a file will be created and the default permissions will be examined.

Open the terminal window:

1. Create a new file by using the `touch` command.

```
[root@hostname ~]# touch file_1
```

2. Run the `ls -l` command and answer the following questions:

- Who owns the file?
- Which group owns the file?

```
[root@hostname ~]# ls -l
```

3. Make the new file read-only by using `chmod 400`.
4. Verify the permission change by running `ls -l`.

Demonstration: Identifying and configuring file permissions

Purpose: In this demonstration, a file will be created and the default permissions will be examined.

How to complete this demonstration:

1. Create a new file by using the `touch` command.
2. Run the `ls -l` command. Follow up by asking learners to answer the following questions:
 - Who owns the file?
 - Which group owns the file?
3. Make the new file read-only by using `chmod 400`.
4. Verify the permission change by running `ls -l`.

Best practices for managing file permissions

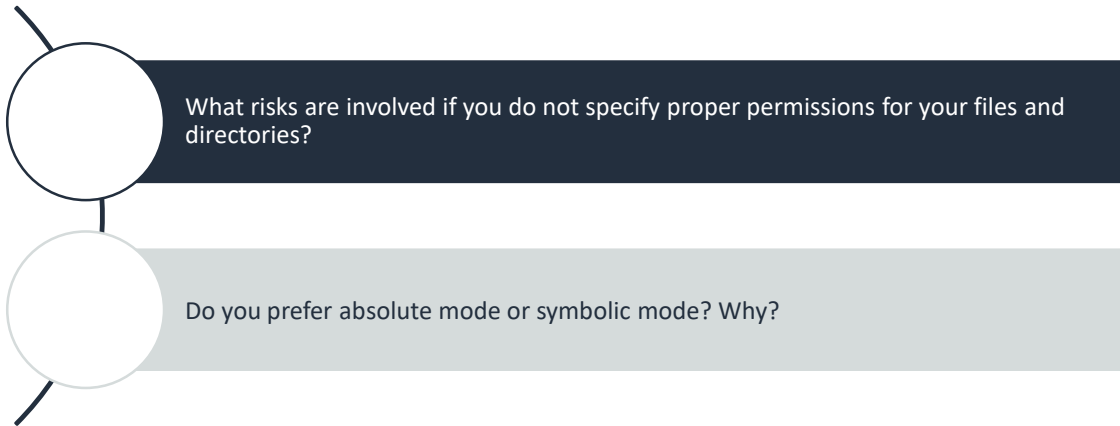


Here are some best practices when managing permissions in Linux.

These are some best practices when managing permissions in Linux:

- Do not use `chmod 777`. It grants read, write, and execute permissions to every user.
- Follow the principle of least permissions. Give the least number of users the least amount of file access at first, and grant more access only when the user has a need.
- Limit file names to alphanumeric characters, dots, and dashes.
- Remember that some characters have special functions.

Checkpoint questions



Q1: You might provide access to users who shouldn't have access, thus threatening the security of your data. Always remember the principle of least privilege.

Q2: When considering this question, remember the following:

- **Absolute mode:** Use numbers to represent file permissions. This mode is the most commonly used to set permissions.
- **Symbolic mode:** Use a combination of letters and symbols to add permissions or to remove any set permissions.

Key takeaways



- Permissions are set on files in Linux by using the `chmod` command in either absolute or symbolic mode.
- `ls -l` is used to view permissions.
- The `chown` command is used to change ownership.

Some key takeaways from this lesson include the following:

- Permissions are set on files in Linux by using the `chmod` command in either absolute or symbolic mode.
- `ls -l` is used to view permissions.
- The `chown` command is used to change ownership.



Thank you

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. Corrections, feedback, or other questions? Contact us at <https://support.aws.amazon.com/#/contacts/aws-training>. All trademarks are the property of their owners.



Thank you.