

What are File Permissions in Linux?

File permissions control which actions can be performed by which users. Read, Write, and Execute are the three actions possible for every file.

Users are classified under three broad categories: Normal users, Groups, and Others. Linux allows users to set permissions at a very granular level. You can secure your file or directory in every possible location of a file system.

There are three important commands you'll use when managing file permissions:

1. `chmod` (Change mode)
2. `chown` (Change ownership)
3. `chgrp` (Change group)

Read and write are pretty self-explanatory – they determine whether a user can read or write to a file.

But, what's an executable file?

A file is said to be executable if it has a sequence of instructions to achieve something. A good example is scripting files (Shell Scripts).

What is the chmod Command?

chmod is a command that lets you change the permissions of a file or directory to all types of users.

Here's the syntax of the chmod command:

chmod <Operations> <File/Directory Name>

What are the operations you can perform?

The Operations in the above syntax are divided into 2 categories. Let's explore them below.

User Level permissions

These operations control permissions on the user level. Here's the commands you can use:

- u – Grant permission to a user
- g – Grant permission to a group (A Group of users)
- o – Grant permission to others (who do not come under either of the above).

File Level permissions

These control permissions on the file level.

- r – Grants read permission
- w – Grant write permission
- x – Grant execute permission

These operations need to be preceded with a '+' or '-' operator.

'+' indicates adding a new permission, and '-' indicates removing an existing permission.

Here's an example:

```
chmod +r sample.txt
```

Command to add read permission to a file

The above command adds read permission for the sample.txt file.

How to Make a File Executable in Linux

we can execute .sh files by just running them like this:

```
./install.sh
```

Let's see the code inside install.sh

```
echo "This is executable file"
```

He ran the same command but it did not work. This is because the file was not in executable format. So, I ran the magic command to make the file executable:

```
chmod +x install.sh
```

How to Remove Permissions from a File in Linux

Here we have a file named install.sh which has all permissions (Read, Write, Execute). Let's remove the execute permission for this script file.

```
chmod -x install.sh
```

```
chmod -r install.sh
```

Command to remove read permission from a file

Alright, with this command we've removed the read permission. Let's try to read that file using Nano (the file editor for the Linux terminal). You will be able to see the "Permission Denied" error at the bottom.

```
Inspiron-5515:~/permissions$ ls
Inspiron-5515:~/permissions$ ./install.sh
file 🐞
Inspiron-5515:~/permissions$ chmod -x install.sh
Inspiron-5515:~/permissions$ ./install.sh
: Permission denied
Inspiron-5515:~/permissions$ chmod -r install.sh
Inspiron-5515:~/permissions$ nano install.sh
--
[ Error reading install.sh: Permission denied ]
G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify   ^G Cur Pos  M-U Undo    M-A Mark Text M-T To Bracket M-Q Previous ^B Back      ^_ Prev Word  ^A Home
X Exit      ^R Read File  ^\ Replace  ^U Paste Text ^T To Spell  ^L Go To Line M-E Redo    M-G Copy Text M-W Where Was M-W Next     ^F Forward   ^N Next Word  ^E End
```

You can achieve all the above together using the below command:

chmod -rwx install.sh

How to Add or Remove Permissions for Directories (Folders) in Linux

If you work with Linux, you might have come across various directories such as /etc, /var, /opt, and others. But you may not be aware of why these directories exist. There's one thing in common for all these folders, though: that is, you'll not be able to create a file or folder inside them without root permission.

I created a directory named locked_directory and removed read permission from this directory. If I try to read the contents of the folder using the ls command, I'll end up seeing the "Permission Denied" error message.

```
chmod -r locked_directory/
```

But, did you know that I can create another directory inside locked_directory named dir1 and read the files and folders in dir1?

Then what's the purpose of the command we just ran before? Removing the read permission on the parent should remove the same on child directories too, right?

Well. That's the exact thing I told you earlier. Linux manages a very granular level of file permissions.

If you want to apply the permissions to the parent directory and all its child directories, you need to pass an exclusive flag with the chmod command.

That flag is -R. It basically means applying the same permissions recursively to all sub-directories (child directories). So this permission will apply to the end child of a file/directory.

```
sudo chmod -R <permission> <filename>
```

Remember that running the command to do a recursive operation needs root permission. So you need to add sudo at the beginning of this command. Here's what it looks like:

```
sudo chmod -R -r locked_directory
```

Another Way to Handle File Permissions in Linux

Alternatively, you can use Octal representation to control the file permissions.

We can use numbers to represent file permissions (the method most commonly used to set permissions). When you change permissions using the Octal mode, you represent permissions for each triplet using a number (4, 2, 1, or combination of 4, 2, and 1).

Let's see the syntax for using octal mode:

```
chmod <user><group><others> install.sh
```

Syntax to use Octal Mode

Here's an example of octal mode:

```
chmod 777 install.sh
```

Command to grant all permission using Octal Mode. How can I remove permissions using Octal Mode?

We can use 0 to remove permissions from a file. Here's an example:

```
chmod 000 install.sh
```

Command to remove all permission using Octal Mode

Access	Symbolic Mode	Octal Mode
Read	r	4
Write	w	2
Execute	x	1

The table shows the Octal code for each file permission:

Access	Symbolic Mode Eg:u+rwX,g+rw,o+r	Octal Mode Eg:764 (User, Group, Others)
User	u	<first place>
Group	g	<middle place>
Others	o	<last place>

Let's consider a scenario.

You want to grant read, write, and execute permissions to users and read-only permission for groups and others to the install.sh file.
Let's see how to do that using the above two methods.

How to manage permissions in Symbolic Mode

```
chmod u+rw,go+r install.sh
```

Command to achieve above scenario using Symbolic Mode

Let's dismantle each part and try to understand them:

- **u+rw** represents adding read, write, and execute permissions for users
- **go+r** represents adding read permission for groups and others

How to manage permissions in Octal Mode

chmod 744 install.sh

Command to achieve above scenario using Octal Mode

Let's dismantle each of these numbers and try to understand them:

- **The first number (7) represents permission for a user: 7 = (4 (read) +2 (write) +1(execute))**
- **The second number (4) represents permissions for a group: 4 (read)**
- **The third number (4) represents permissions for others: 4 (read)**

Which Mode is Best?

It turns out that symbolic mode is more powerful than octal mode.

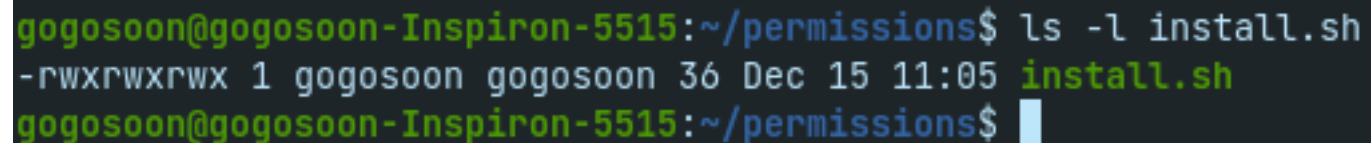
The reason is, in the symbolic mode we can mask out the permission bits we want to change. But in octal mode, permission modes are absolute and can't be used to change individual bits.

How to Find Permissions of a File

We can find the existing permissions of a file using ls command.

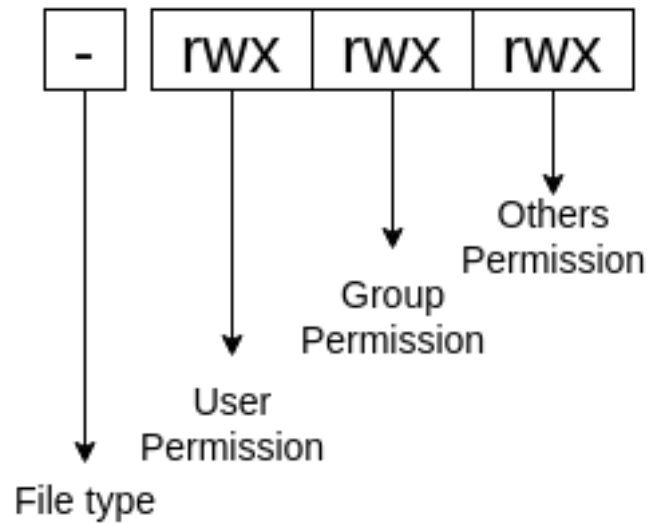
I hope you all know about ls command. Adding the -l flag and file name with the ls command shows some more info about the file, including the permissions.

ls -l install.sh



```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l install.sh
-rwxrwxrwx 1 gogosoon gogosoon 36 Dec 15 11:05 install.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$
```

Look at the first part of the output (-rwxrwxrwx) from the above screenshot. Let's explore what it means:



The first character indicates the type of input.

- "-" indicates a file
- "d" indicates a directory
- "l" indicates a link (a symlink, which is a shortcut to a file/directory)

Linux is a multi-user OS which means that it supports multiple users at a time.

As many people can access the system simultaneously and some resources are shared, Linux controls access through ownership and permissions.

Linux file ownership

In Linux, there are three types of owners: user, group, and others .

Linux User

A user is the default owner and creator of the file. So this user is called owner as well.

Linux Group

A user-group is a collection of users. Users that belonging to a group will have the same Linux group permissions to access a file/ folder.

You can use groups to assign permissions in a bulk instead of assigning them individually. A user can belong to more than one group as well.

Other

Any users that are not part of the user or group classes belong to this class.

Linux File Permissions

File permissions fall in three categories: read, write, and execute.

Read permission

For regular files, read permissions allow users to open and read the file only. Users can't modify the file.

Similarly, for directories, read permissions allow the listing of directory content without any modification in the directory.

Write permission

When files have write permissions, the user can modify (edit, delete) the file and save it.

For folders, write permissions enable a user to modify its contents (create, delete, and rename the files inside it), and modify the contents of files that the user has write permissions to.

Execute permission

For files, execute permissions allows the user to run an executable script. For directories, the user can access them, and access details about files in the directory.

Below is the symbolic representation of permissions to user, group, and others.

rWX	rWX	rWX
user	group	other


Symbolic representation of permissions

Note that we can find permissions of files and folders using long listing (`ls -l`) on a Linux terminal.

```

zaira@Zaira:~/freeCodeCamp$ ls -l
total 3856
-rw-r--r-- 1 zaira zaira 89 Apr 5 20:46 CODE_OF_CONDUCT.md
-rw-r--r-- 1 zaira zaira 210 Apr 5 20:46 CONTRIBUTING.md
-rw-r--r-- 1 zaira zaira 1513 Apr 5 20:46 LICENSE.md
-rw-r--r-- 1 zaira zaira 19933 Apr 5 20:46 README.md
drwxr-xr-x 4 zaira zaira 4096 Apr 6 22:45 api-server
-rw-r--r-- 1 zaira zaira 67 Apr 5 20:46 babel.config.js
drwxr-xr-x 10 zaira zaira 4096 Apr 6 22:55 client
drwxr-xr-x 5 zaira zaira 4096 Apr 6 22:54 config

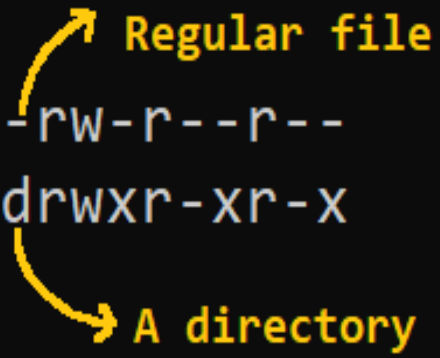
```



MODE OWNER GROUP SIZE MODIFICATION DATE FILE/FOLDER NAME

Output of long listing

In the output above, d represents a directory and - represents a regular file.



Regular file

-rw-r--r--

drwxr-xr-x

A directory

How to Change Permissions in Linux Using the chmod Command

Now that we know the basics of ownerships and permissions, let's see how we can modify permissions using the chmod command.

Syntax of chmod:

chmod permissions filename

Where,

- permissions can be read, write, execute or a combination of them.
- filename is the name of the file for which the permissions need to change. This parameter can also be a list of files to change permissions in bulk.

We can change permissions using two modes:

1. **Symbolic mode:** this method uses symbols like u, g, o to represent users, groups, and others. Permissions are represented as r, w, x for read write and execute, respectively. You can modify permissions using +, - and =.
2. **Absolute mode:** this method represents permissions as 3-digit octal numbers ranging from 0-7.

Now, let's see them in detail.

How to Change Permissions using Symbolic Mode

The table below summarize the user representation:

USER REPRESENTATION	DESCRIPTION
u	user/owner
g	group
o	other

We can use mathematical operators to add, remove, and assign permissions. The table below shows the summary:

OPERATOR	DESCRIPTION
+	Adds a permission to a file or directory
—	Removes the permission
=	Sets the permission if not present before. Also overrides the permissions if set earlier.

Example:

Suppose, I have a script and I want to make it executable for owner of the file zaira. Current file permissions are as follows:

```
zaira@Zaira:~$  
zaira@Zaira:~$ ls -l | grep mymotd.sh  
-rw-r--r-- 1 zaira zaira 77 Mar 11 13:41 mymotd.sh
```

Let's split the permissions like this:

-	rW-	r--	r--
file type	user	group	other

To add execution rights (x) to owner (u) using symbolic mode, we can use the command below:

chmod u+x mymotd.sh

Output:

Now, we can see that the execution permissions have been added for owner zaira.

```
zaira@Zaira:~$ ls -lrt | grep mymotd.sh
-rwxr--r-- 1 zaira zaira 77 Mar 11 13:41 mymotd.sh
```

Permissions changed to execution "x"

Additional examples for changing permissions via symbolic method:

- Removing read and write permission for group and others: `chmod go-rw`.
- Removing read permissions for others: `chmod o-r`.
- Assigning write permission to group and overriding existing permission: `chmod g=w`.

How to Change Permissions using Absolute Mode

Absolute mode uses numbers to represent permissions and mathematical operators to modify them.

The below table shows how we can assign relevant permissions:

PERMISSION	PROVIDE PERMISSION
------------	--------------------

read	add 4
------	-------

write	add 2
-------	-------

execute	add 1
---------	-------

Permissions can be revoked using subtraction. The below table shows how you can remove relevant permissions.

PERMISSION	REVOKE PERMISSION
------------	-------------------

read	subtract 4
------	------------

write	subtract 2
-------	------------


execute	subtract 1
---------	------------

Example:

-
- Set read (add 4) for user, read (add 4) and execute (add 1) for group, and only execute (add 1) for others.

chmod 451 file-name


This is how we performed the calculation:

	read	write	execute	sum			
user	4	0	0	4		451	
group	4	0	1	5			
other	0	0	1	1			
					Final permission		

Note that this is the same as r--r-x--x.


- Remove execution rights from other and group.

To remove execution from other and group, subtract 1 from the execute part of last 2 octets.

	read	write	execute	sum			
user	4	0	0	4		440	
group	4	0	1-1 (subtract)	4			
other	0	0	1-1 (subtract)	0			
					Updated permission		

- Assign read, write and execute to user, read and execute to group and only read to others.

This would be the same as `rwxr-xr--`.

	read	write	execute	sum			
user	4	2	1	7			
group	4	0	1	5		754	
other	4	0	0	4		Final permission	

How to Use the chown and chgrp Commands

When it comes to large organizations, Users and Groups in Linux play an important role in helping keep systems secure and properly functioning.

There can be different levels of users in an organization with different roles and permissions. And you'll need a good understanding Linux permission to manage and/or understand them.

To protect files and directories in Linux from access by certain types of users, we can use the chown and chgrp commands. These commands let you manage which type of user can read, write, and execute a file.

What are Group and Users in Linux?

A user is a regular entity that can manipulate files, directories, and perform various types of actions in a system. We can create any number of users in Linux.

A group contains zero or more users in it. Users in a group share the same permissions. The group allows you to set permissions on the group level instead of having to set permissions for individual users.

Let's consider a scenario in software development where a machine gets used by various types of people like Administrators, Developers, and Testers.

Each person should have an individual level of access to the files in the system. Accordingly, there will be a common set of permissions for developers, testers and admins, in their respective groups.

Let's say there are 10 developers and 8 testers on your team and you're using 1 shared computer (each of you has a laptop too).

What are Group and Users in Linux?

A user is a regular entity that can manipulate files, directories, and perform various types of actions in a system. We can create any number of users in Linux.

A group contains zero or more users in it. Users in a group share the same permissions. The group allows you to set permissions on the group level instead of having to set permissions for individual users.

Let's consider a scenario in software development where a machine gets used by various types of people like Administrators, Developers, and Testers.

Each person should have an individual level of access to the files in the system. Accordingly, there will be a common set of permissions for developers, testers and admins, in their respective groups.

Let's say there are 10 developers and 8 testers on your team and you're using 1 shared computer (each of you has a laptop too).

What are Primary and Secondary Groups in Linux?

As the name implies, a primary group is a group that a user belongs to by default.

For example, let's assume your username is arun, and you create a group called admin. Then you will belong to the group admin by default.

A secondary group is a group to which you can add any number of users.

How to Create a User

You can create a user by using the `useradd` command. Each user in a Linux system has a unique user id.

```
useradd [OPTIONS] <user_name>
```

Let's create a new user named developer:

```
useradd developer
```

How to Create a Group

Groups are created by using the `groupadd` command. Similar to users, each group in a Linux system has a unique group id.

```
groupadd [OPTIONS] <group_name>
```

Terminal command syntax to add a group

Let's create a new group named `developers_group`:

```
groupadd developers_group
```

Terminal command to create a user called `developer_group`

How to Add a User to a Group

So, we created a user and a group. Let's add the user (developer) to the group (developers_group). The command to add a user to a group is usermod -aG.

```
sudo usermod -aG <group_name> <user_name>
```

Here's the actual command to add the user developer to developers_group group:

```
sudo usermod -aG developers_group developer
```

How to List Out Groups

You might wonder how you can verify if the created group exists, and how to verify if the user has been added to the group. The list of groups and the users who have permissions in the group are stored in a file called group. It will be located under the /etc directory.

We can see the available groups by reading that file using the cat command like this:

```
cat /etc/group
```

How to Find the Current Owner and Group Ownership of a File

There's a powerful – and likely familiar – command in Linux which shows the permissions involved in a file/directory. This is the ls -l command:

`ls -l test.sh`

```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l test.sh
-rw-rw-r-- 1 gogosoon gogosoon 29 Jan  5 10:22 test.sh
```

Ownership of test.sh file

Let's go over the output separated by spaces and understand each part of it:

- -rw-rw-r-- 1 – Permission for file test.sh
- 1st occurrence of gogosoon – Owner of the file
- 2nd occurrence of gogosoon – Group ownership of the file

How to Change the Owner of a File or Directory

You can use the `chown` command to change the ownership of a file. The `chown` command is abbreviated from "change owner".

From our previous example, we have seen the file `test.sh` owned by the user named `gogosoon`.

`chown <user_name> <file_name>`

Terminal command syntax to change ownership of a file/directory

Let's change the ownership of the file to the user admin using the chown command. We can do that like this:

```
sudo chown admin test.sh
```

```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l test.sh
-rw-rw-r-- 1 gogosoon gogosoon 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ sudo chown admin test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l test.sh
-rw-rw-r-- 1 admin gogosoon 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$
```

How to Change Ownership using the chown Command

You can change the ownership of a file or folder using the chown command. In some cases, changing ownership requires sudo permissions.

Syntax of chown:

chown user filename

How to change user ownership with chown

Let's transfer the ownership from user zaira to user news.

chown news mymotd.sh

```
zaira@Zaira:~$ ls -lrt | grep motd
-rwx-w-r-- 1 zaira zaira 77 Mar 11 13:41 mymotd.sh
```

Current owner is zaira

Command to change ownership: sudo chown news mymotd.sh

Output:

```
zaira@Zaira:~$ ls -lrt | grep motd
-rwx-w-r-- 1 news zaira 77 Mar 11 13:41 mymotd.sh
```

Owner changed to 'news'

How to change user and group ownership simultaneously

We can also use chown to change user and group simultaneously.

chown user:group filename

How to change directory ownership

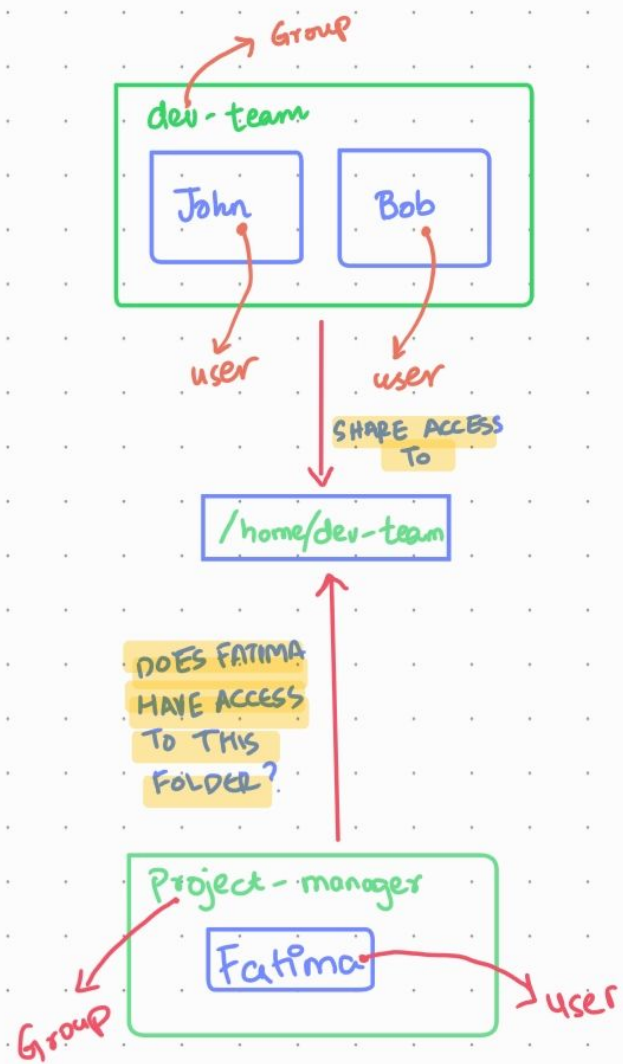
You can change ownership recursively for contents in a directory. The example below changes the ownership of the /opt/script folder to allow user admin.

chown -R admin /opt/script

How to change group ownership

In case we only need to change the group owner, we can use chown by preceding the group name by a colon:

chown :admins /opt/script



How to Copy Ownership from One File to Another

I have faced this scenario once in my career. We use a common system in some rare use cases. Here's what was going on:

One day I was working on that system to complete a POC which required me to create hundreds of files with a different user ownership. A file was created with default permissions (owned by me) whenever it was created.

But I want the file to be owned by another user. I was too lazy to change the ownership for each file manually. If I changed the ownership for one file, I wanted to be able to copy the same ownership for other files. I was sure that there must be some command that allowed me to do this.

```
chown --reference=<source_file_name> <destination_file_name>
```

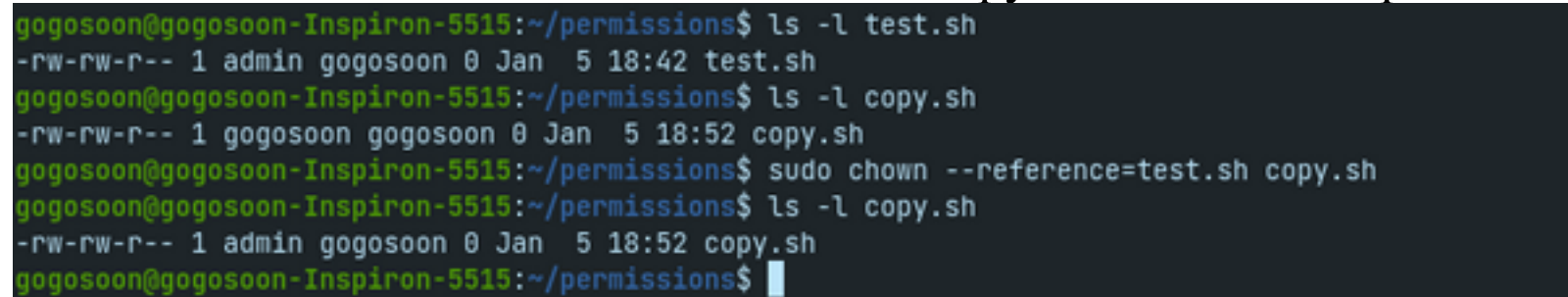
Terminal command syntax to copy the ownership of one file to another
Let's explore this with an example:

Let's create a new file named copy.sh with my user account gogosoon.

The owner of the test.sh file is the admin user (from our previous example). I want the ownership of test.sh file to be copied to the newly created copy.sh file which is owned by the gogosoon user.

```
sudo chown --reference=test.sh copy.sh
```

Terminal command to copy ownership of test.sh file



```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l test.sh
-rw-rw-r-- 1 admin gogosoon 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l copy.sh
-rw-rw-r-- 1 gogosoon gogosoon 0 Jan  5 18:52 copy.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ sudo chown --reference=test.sh copy.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l copy.sh
-rw-rw-r-- 1 admin gogosoon 0 Jan  5 18:52 copy.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$
```

Ownership of test.sh file copied to copy.sh file

From the above screenshot, you can see that the first command describes the ownership of the test.sh file, which is owned by the admin user.

The second command describes the ownership of the copy.sh file which is owned by the gogosoon user.

The third command copies the ownership of the test.sh to the copy.sh file.

The last command again describes the ownership of the `copy.sh` file which is now owned by admin user.

You may wonder that at the beginning I told that I created hundreds of files – but how did I change the ownership of all the files at once?

That's a different story. But here's a quick answer: I wrote a script that looped over all the files and changed the ownership by referencing a single master file.

How to Change Ownership of Multiple Files with a Single Command

You can do this by passing multiple file names to the `chown` command with one user name. This sets the ownership of all the given files to that particular user.

```
sudo chown <user_name> file1 file2 ...
```

Terminal command syntax to change ownership of multiple files with a single command
Here's an example where I want to set the ownership of the files `copy.sh` and `test.sh` to the admin user:

sudo chown admin copy.sh test.sh

Set ownership of copy.sh and test.sh files

```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l copy.sh test.sh
-rw-rw-r-- 1 gogosoon gogosoon 0 Jan  5 18:52 copy.sh
-rw-rw-r-- 1 gogosoon gogosoon 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ sudo chown admin copy.sh test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l copy.sh test.sh
-rw-rw-r-- 1 admin gogosoon 0 Jan  5 18:52 copy.sh
-rw-rw-r-- 1 admin gogosoon 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$
```

Set ownership of copy.sh and test.sh files to admin user

How to Change the Group Ownership of a File

Almost all the operations related to groups can be achieved with chgrp command (an abbreviation of "change group"). It's pretty similar to the chown command.

Syntax of the chgrp command:

```
sudo chgrp <group_name> <file/dir_name>
```

Terminal command syntax to change group ownership of a file/directory

I have already created a group called admin . I do not belong to this group. Let's change the group ownership of the test.sh file from gogosoon to the admin group.

sudo chgrp admin test.sh

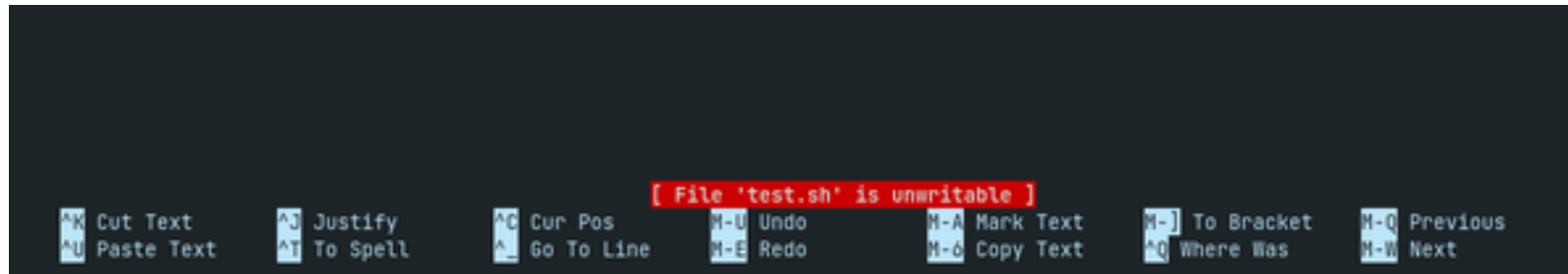
Change group ownership of test.sh file

```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l test.sh
-rw-rw-r-- 1 admin gogosoon 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ sudo chgrp admin test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l test.sh
-rw-rw-r-- 1 admin admin 0 Jan  5 18:42 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$
```

Change group ownership of test.sh file to admin

From the above screenshot, you can see that I changed group ownership of the test.sh file from gogosoon to admin. Since I do not belong to this group, I will not have write access to the file.

Let's verify the same by opening the file in write mode using nano test.sh:



You can see that I do not have write permission for this file

How to Change the Group Ownership of a Directory

The same syntax for files is applicable to directories also. Here's a quick example:

```
sudo chgrp test group_test/
```

Change group ownership of group_test directory
to test group

```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ mkdir group_test
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l
total 8
drwxrwxr-x 2 gogosoon gogosoon 4096 Jan  5 10:29 group_test
-rw-rw-r-- 1 gogosoon test      29 Jan  5 10:22 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$ sudo chgrp test group_test/
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l
total 8
drwxrwxr-x 2 gogosoon test 4096 Jan  5 10:29 group_test
-rw-rw-r-- 1 gogosoon test  29 Jan  5 10:22 test.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions$
```

Change group ownership of group_test directory to test group

But remember that the above command changes the group ownership of only the files in that directory. To recursively change the group permissions of all the directories inside that directory, we have to add the -R flag to it like this:

```
sudo chgrp -R admin group_test/
```

Change group ownership of group_test directory to test group

```
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l group_test/
total 4
-rw-rw-r-- 1 admin gogosoon  0 Jan  5 22:23 hello1.sh
-rw-rw-r-- 1 admin gogosoon  0 Jan  5 22:23 hello2.sh
drwxrwxr-x 2 admin gogosoon 4096 Jan  5 22:23 rec
gogosoon@gogosoon-Inspiron-5515:~/permissions$ sudo chgrp -R admin group_test/
gogosoon@gogosoon-Inspiron-5515:~/permissions$ ls -l group_test/
total 4
-rw-rw-r-- 1 admin admin  0 Jan  5 22:23 hello1.sh
-rw-rw-r-- 1 admin admin  0 Jan  5 22:23 hello2.sh
drwxrwxr-x 2 admin admin 4096 Jan  5 22:23 rec
```

Now the group ownership for all the files and directories inside group_test have been changed from gogosoon to admin.

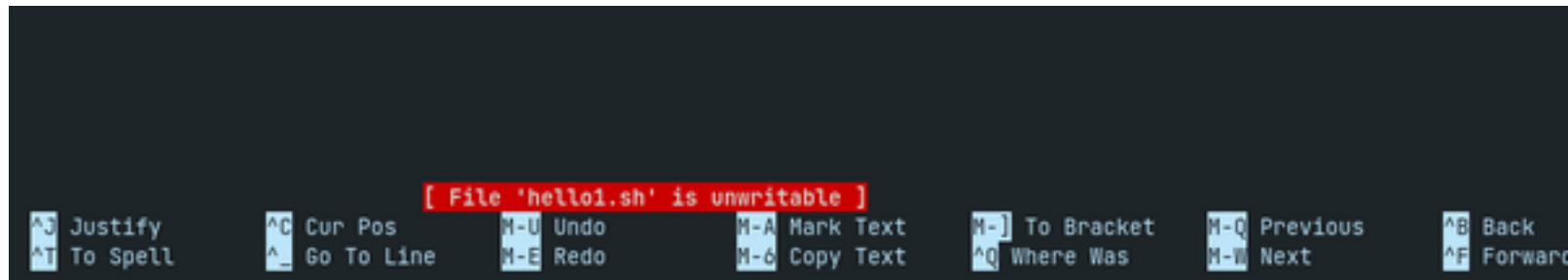
Let's verify the output by trying to write a file from the directory group_test as the gogosoon user:

```

gogosoon@gogosoon-Inspiron-5515:~/permissions$ cd group_test/
gogosoon@gogosoon-Inspiron-5515:~/permissions/group_test$ ls
hello1.sh  hello2.sh  rec
gogosoon@gogosoon-Inspiron-5515:~/permissions/group_test$ nano hello1.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions/group_test$ cd rec/
gogosoon@gogosoon-Inspiron-5515:~/permissions/group_test/rec$ ls
hello3.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions/group_test/rec$ nano hello3.sh
gogosoon@gogosoon-Inspiron-5515:~/permissions/group_test/rec$ █

```

Trying to edit the files hello1.sh and hello3.sh



```

      [ File 'hello1.sh' is unwritable ]
^J Justify      ^C Cur Pos      M-U Undo      M-A Mark Text  M-] To Bracket M-Q Previous  ^B Back
^T To Spell    ^_ Go To Line  M-E Redo      M-d Copy Text  M-^ Where Was  M-W Next     ^F Forward

```

Error showing hello1.sh file not

[File 'hello3.sh' is unwritable]

^J Justify
^T To Spell

^C Cur Pos
^_ Go To Line

M-U Undo
M-E Redo

M-A Mark Text
M-6 Copy Text

M-] To Bracket
^Q Where Was