Files and Directories permissions

utrains.org

Permission Denied

Table of content

1

File/directory access

Who has access to a file and who don't?

File/directory access

- You may want to **cat** a file on the system, but you obtain a **Permission denied** after running the command
- **Let's practise that on an example:**
- Launch a centos server from your VS code and connect remotely using ssh
- Switch to the root user: **# su** (enter your root password)
- Create a new user on your server: **# useradd serge**
- Give the user a password: **# passwd serge**
- Enter and confirm the password **school1** for this user

File/directory access

- Still as the root user, Let's view the content of the file: **/etc/shadow** with the command: **# cat /etc/shadow**
- **Output:** At the end of the page, you will find the password encryption for user **student** and user **serge**

```
john1:!!:18983:0:99999:7:::
manolo:!!:18984:0:99999:7:::
serge:$1$nFLCM8en$e7D5sq78x.a31YLoV8ZXJ/:18988:0:99999:7:::
student:$1$MdV9mxZa$8bPPXWjE7xWmHOmOIf9wK/:18988:0:99999:7:::
```

- **NB:** The **/etc/shadow** file contains the **passwords encryption of the users on the server.**
- If you don't have a user **student** on your system, create it and give **school1** as password

File/directory access

- Now, let's switch to the user **serge** we previously created:
  - **# su serge**.
- Enter the password if asked: **school1**
- you can run **$ id** to check that you are user serge indeed
- Let's try to view the content of the file **/etc/shadow** with the command:
  - **$ cat /etc/shadow**

```
[serge@localhost vagrant]$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

- The **permission denied** here means **user serge can't access that file**

File/directory access

- There is another user on our system called **student**. Thus, in the **/home** directory, there is a folder called **student** for that user (check that with **$ ls /home**).
- Now run the following command: **$ cd student**
- **Output:**

```
[serge@localhost vagrant]$ cd student
bash: cd: student: Permission denied
```

- **There is a permission for access set on this folder. User serge can't access the student folder**

File/directory access

- Now you can better understand that on a system, **some people might have access to some resources while others won't**
- **The question now is: How is all this set up?**
- Let's switch to the root user and create some files on which we will practise this notion:
  - **$ su root**
  - Enter your root password
  - **# touch file2 review success hardwork**
  - **# mkdir prayer class dir**

File/directory access

- Now run **# ll** or **# ls -l** to list the content of the current directory (the one in which we previously created the files and directories)

```
drwxr-xr-x. 2 root root 6 Apr 26 20:44 class
drwxr-xr-x. 2 root root 6 Apr 26 20:44 dir
-rw-r--r--. 1 root root 0 Apr 26 20:44 file2
-rw-r--r--. 1 root root 0 Apr 26 20:44 hardwork
drwxr-xr-x. 2 root root 6 Apr 26 20:44 prayer
-rw-r--r--. 1 root root 0 Apr 26 20:44 review
-rw-r--r--. 1 root root 0 Apr 26 20:44 success
```

- Look at the first column of all the rows displayed
- **Let's pick one line and check this in details**

File/directory access

- Take the following line for example (informations on the file **file2**):

```
-rw-r--r--. 1 root root 0 Apr 26 20:44 file2
```

- **What are the various elements displayed and what do they represent?**

-	rw-r--r--	1	root	root	0	Apr 26 20:24	file2
Type of document (A file here)	Permissions on the file	Number of links	Owner	group	size	Date and time	Document's name

Now let’s go in details on files or directories permissions to better understand this notion

2

Files/directory permissions

What do these permissions represent?

File/directory Permissions

As stated in the previous table,

```
-rw-r--r--. 1 root root 0 Apr 26 20:44 file2
```

- The **rw-r--r--** represents the permissions on file2 (the first **-** is just the identify that this is a file, for directories, it will display letter **d**)
- You can also notice that the **owner** of the file is the **root** user
- The **group to which the file belongs** is also the **root**

Types of permissions

There are 3 types of permissions on a file/directory and theses permissions are applied to 3 groups of users on the system.

File/directory Permissions

- We have 3 types of permissions:

Permission	Letter	Number	Description
Read	r	4	Enables a user to display or visualize the content of the file (cat, more, less ...)
Write	w	2	Enables a user to modify and save the file/directory (vi, gedit ...)
Execute	x	1	Enables a user to execute the file/directory (specially for scripts)

- **NB:** To cd into a folder, a user must have the permission to execute it.

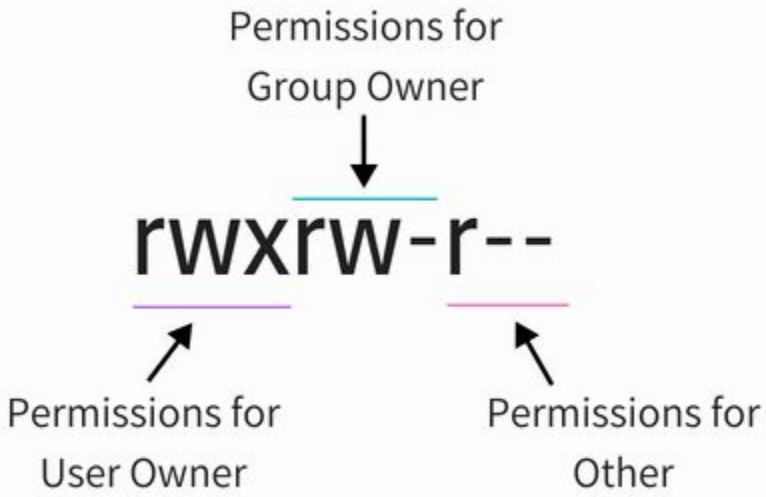
File/directory Permissions

- The permissions are applied on **3 categories of users**

Category	Corresponding letter	Description
The Owner	u	The owner or the one who created the file
The group	g	The group to which the file belongs
The World/others	o	The other users on the system

- Thus, we can subdivide the permissions into 3 parts:
  - **User's permissions** (owner's permission),
  - **group permissions**,
  - **others' permissions**

File/directory Permissions



r = read
w = write
x = execute
- = No Permissions Set

File/directory Permissions

Let's check that on the directory **class** we previously created

```
drwxr-xr-x. 2 root root 6 Apr 26 20:44 class
```

- The **owner (root) has all permissions** (**rwx**)
- The **group (root) can read and execute** (**r-x**)
- The **others can read and execute** (**r-x**)

You can check the permissions on other files or directories in our current directory.

The big question now is:

Can we modify the permissions set on a file/directory?

3

Modify file/directory permissions

How do we modify a file/directory permissions?

Modify permissions

- To modify the permissions on a file or directory, we use the command
  - **# chmod add/removePermission fileName**
- There are two ways to do that:
  - **Using letters (r, w, x for permissions and u, g, o for users)**
  - **Using numbers (r=4, w=2, x=1)**
- While using letters, we use **+** to **add a permission** and **-** to **remove a permission.**

Let's see that on examples.

Modify permissions

Method 1: Using letters

- **Example 1**: To add all permissions to the owner of file2, we run the command: # chmod u+rwx file2
- **Example 2**: To remove the permission to execute file2 from the owner, we run the command: # chmod u-x file2
- **Example 3:** To add the permission to write to the group, we will run the command: # chmod g+w file2

NB: In companies, we mostly use numbers to modify permissions on files

Modify permissions

Method 2: Using numbers

- Here, the command **chmod** is followed by **3 numbers** and the name of the file/directory.
  - The first number represents the permissions for the **Owner**
  - The second number represents the permissions for the **Group**
  - The third number represents the permissions for **Others**

Example: # chmod 755 file2

How do we know which number to put and how does this work?

Modify permissions

Method 2: Using numbers

To explain that, let's consider the permissions rwxr-xr-x.

We know that r=4, w=2 and x=1

rwx	r-x	r-x
↕	↕	↕
4+2+1	4+0+1	4+0+1
=	=	=
7	5	5

Modify permissions

Method 2: Using numbers

Looking at this, we can say:

rwx = 4 + 2 + 1 = 7	-wx = 0 + 2 + 1 = 3
rw- = 4 + 2 + 0 = 6	-w- = 0 + 2 + 0 = 2
r-x = 4 + 0 + 1 = 5	--x = 0 + 0 + 1 = 1
r-- = 4 + 0 + 0 = 4	--- = 0 + 0 + 0 = 0

777 = full permission

000 = No permission

Modify permissions

Method 2: Using numbers

Example 1: Let's remove the execution permission on directory **class** to the **Others**. (use # ll to see if your modifications have been applied)

- The permissions now on that directory are (**rwx r-x r-x = 755**). Others have the permission to read and execute (**r-x = 5**)

```
drwxr-xr-x. 2 root root 6 Apr 26 20:44 class
```

- Removing the execution permission to **Others** means henceforth, they will only have the permission to read (**r-- = 4**).
- So, we need to run **# chmod 754 class** (**7 5 4 ⇔ rwx r-x r--**)

```
drwxr-xr--. 2 root root 6 Apr 26 20:44 class
```

Modify permissions

Method 2: Using numbers

Example 2: Let's remove the execution permission on **file2** to the **Owner**.

- The permissions now on that directory are (**rwx r-x r-- = 764**). The **Owner** has all the permissions (**rwx = 7**)

```
-rwxrw-r--. 1 root root 0 Apr 26 20:44 file2
```

- Removing the execution permission to the **Owner** means henceforth, he will only have the permission to read and to write (**rw- = 6**).
- So, we need to run **# chmod 664 file2** (6 6 4 ⇔ rw- rw- r--)

```
-rw-rw-r--. 1 root root 0 Apr 26 20:44 file2
```

4

Change file/directory Ownership

How do we change the group Owner or the Owner of a file/directory?

Change group owner of a file/directory: chgrp

Change Group ownership

- If a file/directory belongs to a group, then **that file is accessible to all the users of that group**
- To change the group ownership of a file/directory, we use the command
  - **#chgrp groupName fileName**
- To put that into practice, let's create a new group called **linux** on our system:
  - **# groupadd linux**

Change Group ownership

- Now, let's **change the group ownership on the directory class**.

```
drwxr-xr--. 2 root root 6 Apr 26 20:44 class
```

- Class here belongs to the group root
- **# chgrp linux class** (**class now belongs to group linux**)

```
drwxr-xr--. 2 root linux 6 Apr 26 20:44 class
```

You can check that by running the **# ll** command

Change Group ownership

- Henceforth, each member of the group **linux** will be able to read and execute (r-x) the directory **class**

```
drwxr-xr--. 2 root linux 6 Apr 26 20:44 class
```

- To give them the permission to **write**, we can run:
  - **# chmod g+w class or # chmod 774 class**
- The output of **# ll** is now:

```
drwxrwxr--. 2 root linux 6 Apr 26 20:44 class
```

Change the Owner of a file/directory: chown

Change file/directory Owner

- To change the owner of a file, you use the command:

  # chown userName fileName

Example: To make user **serge** the new owner of the directory **class**, we run:

 # chown serge class

```
drwxrwxr--. 2 serge linux 6 Apr 26 20:44 class
```

NB: Whenever a user belonging to a specific group creates a file or a directory, that file will automatically belong to that group.

Change file/directory Owner

Example: Switch to user serge and create a new file:

- **# su - serge**
- Enter the password
- **$ touch learning**
- **$ ll**

```
-rw-rw-r--. 1 serge serge 0 Apr 26 21:11 learning
```

The file was created by user **serge** and it belongs automatically to group **serge**

Change the Owner and group of a file/directory: chown

Change the Owner & the group

- You can use the **chown** command **to change the owner and the group** of a file/directory at the same time.
- To do that, we run **# chown newOwner:newGroup fileName**

Example: Go back to the root user. Let's change the ownership of the file **learning** from user **serge group serge** to **user student and group linux**:

- **# su**
- **# chown student:linux learning**
- **# ll**

```
-rw-rw-r--. 1 student linux 0 Apr 26 21:11 learning
```

Do many examples to better understand the notions. Rewatch the video if necessary

That said and done, let's move on to another notion: Special permissions

5

Special permissions

SUID, SGID, Sticky bit

Special permissions

There are **special permissions** in Linux:

- The **Set User ID (SUID)**
- The **Set Group ID (SGID)**
- The **Sticky bit**

SUID: Set User ID

Special permissions: SUID

- When the **SUID** is set on a file, **anybody on the system can execute that file like the owner of the file.**
- Simply put, if the file is owned by the **root**, when somebody else will try to execute the file, **the system will consider that it is the user root that is executing it.**
- The letter corresponding to the **SUID** is **s** and the number is **4**

To set the **SUID** on a file, we use:

 # chmod u+s fileName or **# chmod 4currentPermission fileName**

Special permissions: SUID

- **Example:** To set the **SUID** on **file2**, in **/home/vagrant** we use:
  - **# cd /root/Desktop**
  - **# ll** `-rw-rw-r--. 1 root root 0 Apr 26 20:44 file2`
- Now, let's set the SUID on that file. The permissions on the file here is 664
  - **# chmod u+s file2 or # chmod 4664 file2**
  - **# ll** `-rwSrw-r--. 1 root root 0 Apr 26 20:44 file2`
- The **S** has replace the **x** on the owner permissions of the file **file2**

How is this useful?

You may have a file on the system that can be accessed only by the root. For a particular reason, you may need regular users to access that file (to run it).

Instead of giving root privileges to all the users, you can just set the SUID on the file. After that, users will be able the run the file as the root ie the system will consider it's the root that is executing the file.

Special permissions: SUID

- To remove the **suid** on file2, you run:
  - **# chmod u-s file2** or **# chmod 0664 file2**
  - **# ll**

```
-rw-rw-r--. 1 root  root  0 Apr 26 20:44 file2
```

SGID: Set Group ID

Special permissions: SGID

- When the **SGID** is set on a directory, anything created under that directory will **inherit the group to which the directory belongs**.

Example: Consider a directory belonging to the group **linux** we previously created on the system. Suppose we have set the **SGID** on that directory. If somebody **cd** into that directory and create a file or a subdirectory, **the created file or subdirectory will automatically belong to the group linux**.

- The letter corresponding to the **SGID** is **X** and the number is **2**
- To set this up, we run the command:

  # chmod g+s dirName or **# chmod 2currentPermissions dirName**

Apply this on the directory class

The Sticky bit

Special permissions: Sticky bit

- When the **sticky bit** is set on a directory, all the users can delete what they created under that directory.
- **But, a user cannot delete a document that is owned by another user**
- The letter corresponding to the **sticky bit** is **T** and the number is **1**
- This is applied on **others**
- To set this up, we use:

  # chmod o+t dirName or **# chmod 1currentPermissions dirName**

Special permissions: Sticky bit

- **Example:** There is a special directory on our system (in the root directory /) that has the sticky bit set on it (tmp).
  - **# ll /**
- `drwxrwxrwt.  14 root root 4096 Apr 26 21:17 tmp`
- Everybody can create a content in the directory **tmp** but a user can only delete what he created.

Quick summary

ID	File	Directory
SUID	Run program as owner of the file	-
SGID	Assign authority to run program as owner of the file	Inherit group ownership of all of the item created beneath that directory
Sticky Bit	-	Only owner of the file can delete the file e.g. /tmp

You may have some difficulties understanding this concepts. I advise you to make more research to better understand permissions on files/directories

If you have questions, always feel free to post in the group or ask during the class. I will be glad to help you out!

Don't try to memorize all this at once! Just go through the various steps many times and make sure you understand the process.

See you guys in the next lesson!

Thanks!

Any questions?

You can find us at:

website: http://utrains.org/

Phone: +1 (302) 689 3440

Email: contact@utrains.org

u train

Click on the link below to contact the support team for any issue!

utrains.org/support/

Create a ticket for your problem and we will get back to you soon!