**MANAGING LOCAL LINUX USERS AND GROUPS**

## **PERMISSION GROUPS**

*--- --- ---*

*rwx rwx rwx*

*user group other*

***USER*** */* ***OWNER******u*** *A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.*

***GROUP******g*** *: The group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.*

***OTHERS******o*** *: This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else*

## **PERMISSION TYPES**

|  |  |  |
| --- | --- | --- |
| *PERMISSION* | *ON A FILE* | *ON A DIRECTORY* |
| *READ [r] [4]* | *Read file content* | *Read directory content* ***ls*** |
| *WRITE [w] [2]* | *Change file content* | *Create files in directory* ***touch*** |
| *EXECUTIVE [x] [1]* | *Execute the files* | *Enter the directory* ***cd*** |

*X*

*If the file is a regular file, then it will add the executable attribute to its permissions, if the file already has some execute bit set. If the file has no execute bit set, none will be added.*

## **Directory permission**

* If a user only has read access on a directry, the name if the files in it can be listed, but no other information including permisions or time stampls are available nor can they be accessed.
* If a user only has exec access on a directory, they cannot list the names of the files in the directory, but if they already know the name of a file which they have permission to read, then they can access the contentsof that fileby explicitly specifiying the file name.
* A file may be removed by anyone who has write permission to the directlry in which the file resides regardless of the ownership or permissions on the file itself. [This can overridden with a special permission, the sticky bit]
* Permissions on a directory are not inherited automatically by the subdirectories and files within in it.
* **The write permission on a directory has the ability to delete the files & subdirectories**. But they cant modilfy the files if don’t have write to that specific file inside the directory.
* Root user restricted by the systems selinux policy

## **File permission**

* Precedence
* USER >> GROUPS >>> OTHERS

## **CHMOD**

*SYNTAX $ chmod [options] [mode] [filename]*

*-v verbose*

*-R Recursive*

***Numeric Method***

*$ chmod ### file |directory r =4 w=2 o=1*

***Symbolic method***

*$ chmod whowhatwhich file | directory*

*Who u , g, o*

*What + , - , =*

*Which r, w, x*

*$ chmod ugo=rw file2*

*$ chmod a+x file2*

*$ chmod -R g+rwx demodir*

## **CHOWN**

**Changing File/Directory user or group ownership**

**Only root can change the ownership of a file. Group ownership however can be set by root or the file’s owner. ROOT can grant ownership to any group while non-root users can grant ownership only to groups they belong to.**

*SYNTAX $ chown owner:group file|directory*

*-R recursive*

*$ chown student filename*

*$ chown -R student Directory*

*$ chown :groupname directory*

*$chown visitors:guests directory*

## **CHGRP**

$ chgrp groupname FILE|DIRECTOORY

**-R recursive**

## **SPECIAL PERMISSIONS**

* + SET USER ID suid
  + SET GROUP ID sgid
  + STICKY BIT sticky

|  |  |  |
| --- | --- | --- |
| **Special Permission** | **Effect on files** | **Effect on Directories** |
| u+s (suid) | File executes as the user that owns the file not the user that the file. | No effect. |
| g+s (sgid) | File executes as the group that owns the file. | Files newly created in the directory have their group owner set to match group owner of the directory. |
| o+t | No effect | Users with WRITE on the directory can only remove files that they own. They cannot remove or force saves to files owned by other users. |

## **SUID (s) (4)**

The suid/setuid bit is represented by **s** (Octal 4).This special permission allows a user to access files and directories that are normally only available to the owner.

If a file was owned by the root user and has the setuid bit set, no matter who executed the file it would always run with root user privileges.

You must be the owner of the file or the root user to set the setuid bit

* $ chmod u+s filename
* $ chmod 4664 filename

-rwSrw-r-- 1 user1 user1 0 2017-10-29 21:41 file1

**capital S >>> no executive permission**

**lower s >>> user has executive permssion**

## **GUID (g) (2)**

When the Set Group ID bit is set, the executable is run with the authority of the group. For example, if a file was owned by the users’ group, no matter who executed that file it would always run with the authority of the user’s group.

* $ chmod u+s filename
* $ chmod 2664 filename

When the setgid bit is set on a directory, all files created within said directory inherit the group ownership of that directory

**Note**: Both the setuid and setgid bits are set using the s symbol. The setgid is represented the same as the setuid bit, except in the group section of the permissions.

## **STICKY BIT (a)**

When the sticky bit is set on a directory, only the root user, the owner of the directory, and the owner of a file can remove files within said directory.

An example of the sticky bit is the /tmp directory. Use the ls -ld /tmp command to view the permissions

* # ls -ld /tmp
* drwxrwxrwt 24 root root 4096 2017-10-30 22:00 tmp

The **t** at the end symbolizes that the sticky bit is set. A file created in the /tmp directory can only be removed by its owner, or the root user

* **$ chmod a+t DIRECTORY**
* **$ chmod 1777 DIRECTORY**

**Default Permissions UMASK**

* The user file-creation mode mask (umask) is use to determine the file permission for newly created files. It can be used to control the default file permission for new files. It is a four-digit octal number.
* A umask can be set or expressed using: Symbolic values ||| Octal values
* The default permissions are files are set by the processes that create them.
* Every process on the system has a umask, which octal bitmask that is used to clear the permissions of new files and directories that are created by the process.
* The system default umask values for bash shell users are defined in the /etc/profile and /etc/bashrc files. Users can override the system defaults in their .bash\_profile and .bashrc files.
* In Linux, the default permissions value is 666 for a regular file, and 777 for a directory. When creating a new file or directory, the [kernel](https://www.computerhope.com/jargon/k/kernel.htm) takes this default value, "subtracts" the umask value, and gives the new files the resulting permissions.
* If it is less than 3 digits than leading zeros are assumed.
* Default file permission : 666
* Default Directory Permission : 777
* Default UMASK permission for regular user: 0002
* Default UMASK permission for Root user: 0022
* **Syntax**
* Umask [options] [mode]

-S Symbolic form

*To view the Current UMASK value*

*$ umask*

*To view the current umask value in symbolic   
 $ umask -S*

*How to set umask value*

*$ umask 007 #660*

*$ umask 0 #666*

* Modify /etc/bashrc and /etc/profile to change the default umask for bash shell users. Since the default umask for unprivelledg users is 0002, look for the umask command in these files that sets umask to that value.Change then to set the umask to 007

**$ less /etc/bashrc**