

RHSA (Admin 1) Red Hat Enterprise Linux 8 Essentials

Prepared by Eng. Taghreed Elsayed

Lecture 5

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

Controlling Access to Files with Linux File System Permissions



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

LINUX FILE SYSTEM PERMISSIONS

- ❑ Access to files by users are controlled by file permissions. The Linux file permissions system is simple but flexible, which makes it easy to understand and apply, yet able to handle most normal permission cases easily.
- ❑ Files have just three categories of user to which permissions apply. The file is owned by a user, normally the one who created the file.
- ❑ The file is also owned by a single group, usually the primary group of the user who created the file, but this can be changed. Different permissions can be set for the owning user, the owning group, and for all other users on the system that are not the user or a member of the owning group.
- ❑ The most specific permissions apply. So, user permissions override group permissions, which override other permissions.

- ❑ In the graphic that follows, joshua is a member of the groups joshua and web, while allison is a member of allison, wheel, and web.
- ❑ When joshua and allison have the need to collaborate, the files should be associated with the group web and the group permissions should allow the desired access.



There are also just three categories of permissions which apply: read, write, and execute. These permissions affect access to files and directories as follows:

Effects of permissions on files and directories

PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
r (read)	Contents of the file can be read.	Contents of the directory (file names) can be listed.
w (write)	Contents of the file can be changed.	Any file in the directory may be created or deleted.

PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
x (exec)	Files can be executed as commands.	Contents of the directory can be accessed (dependent on the permissions of the files in the directory).

- ❖ Read and exec on read-only directories, so that they can list the directory and access its contents.
- ❖ If a user only has read access on a directory, the names of the files in it can be listed, but no other information, including permissions or time stamps, 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 contents of that file by explicitly specifying the file name.
- ❖ A file may be removed by anyone who has write permission to the directory in which the file resides, regardless of the ownership or permissions on the file itself. (This can be overridden with a special permission, the sticky bit, which will be discussed at the end of the unit.)

VIEWING FILE/DIRECTORY PERMISSIONS AND OWNERSHIP

- ❑ The `-l` option of the `ls` command will expand the file listing to include both the permissions of a file and the ownership:

```
[student@desktopX ~]$ ls -l test
-rw-rw-r--. 1 student student 0 Feb  8 17:36 test
```

- ❑ The command `ls -l directory name` will show the expanded listing of all of the files that reside inside the directory.
- ❑ To prevent the descent into the directory and see the expanded listing of the directory itself, add the `-d` option to `ls`:



```
[student@desktopX ~]$ ls -ld /home
drwxr-xr-x. 5 root root 4096 Jan 31 22:00 /home
```

EXAMPLES: LINUX USER, GROUP, OTHER CONCEPTS

Users and their groups:

lucy	lucy,ricardo
ricky	ricky,ricardo
ethel	ethel,mertz
fred	fred,mertz

File attributes (permissions, user & group ownership, name):

drwxrwxr-x	ricky	ricardo	dir (which contains the following files)
-rw-rw-r--	lucy	lucy	lfile1
-rw-r--rw-	lucy	ricardo	lfile2
-rw-rw-r--	ricky	ricardo	rfile1
-rw-r-----	ricky	ricardo	rfile2

ALLOWED/DENIED BEHAVIOR	CONTROLLING PERMISSIONS
lucy is the only person who can change the contents of lfile1.	lucy has write permissions on the file lfile1 as the owner. No one is listed as a member of the group lucy . The permissions for <i>other</i> do not include write permissions.
ricky can view the contents of lfile2, but cannot modify the contents of lfile2.	ricky is a member of the group ricardo , and that group has read-only permissions on lfile2 . Even though <i>other</i> has write permissions, group permissions take precedence.
ricky can delete lfile1 and lfile2.	ricky has write permissions on the directory containing both files, and as such, he can delete any file in that directory.
ethel can change the contents of lfile2.	Since ethel is not lucy , and is not a member of the ricardo group, <i>other</i> permissions apply to her, and those include write permission.
lucy can change the contents of rfile1.	lucy is a member of the ricardo group, and that group has both read and write permissions on rfile1 .
ricky can view and modify the contents of rfile2.	ricky owns the file and has both read and write access to rfile2 .
lucy can view but not modify the contents of rfile2.	lucy is a member of the ricardo group, and that group has read-only access to rfile2 .
ethel and fred do not have any access to the contents of rfile2.	<i>other</i> permissions apply to ethel and fred , and those permissions do not include read or write permission.

MANAGING FILE SYSTEM PERMISSIONS FROM THE COMMAND LINE

- ❑ **CHANGING FILE/DIRECTORY PERMISSIONS** The command used to change permissions from the command line is `chmod`, short for "change mode" (permissions are also called the mode of a file).
- ❑ The `chmod` command takes a permission instruction followed by a list of files or directories to change. The permission instruction can be issued either symbolically (the symbolic method) or numerically (the numeric method).

Symbolic method keywords:

```
chmod whowhatwhich file|directory
```



- Who is u, g, o, a (for user, group, other, all)
 - What is +, -, = (for add, remove, set exactly)
 - Which is r, w, x (for read, write, execute)
- The symbolic method of changing file permissions uses letters to represent the different groups of permissions: u for user, g for group, o for other, and a for all.
- ❑ With the symbolic method, it is not necessary to set a complete new group of permissions.
 - ❑ Instead, it is possible to change one or more of the existing permissions. In order to accomplish this, use three symbols:
- ❑ + to add permissions to a set
 - ❑ - to remove permissions from a set
 - ❑ = to replace the entire set for a group of permissions.

- ❑ The permissions themselves are represented by a single letter: r for read, w for write, and x for execute
- ❑ When **using chmod** to change permissions with the symbolic method, using a **capital X** as the permission flag will add execute permission only if the file is a directory or already has execute set for user, group, or other.

❑ Numeric method:

```
chmod ### file|directory
```



- Each digit represents an access level: user, group, other. • # is sum of r=4, w=2, and x=1. Using the numeric method, permissions are represented by a three-digit (or four, when setting advanced permissions) octal number. A single octal digit can represent the numbers 0-7, exactly the number of possibilities for a three-bit number.
- ☐ To convert between **symbolic and numeric** representation of permissions, we need to know how the mapping is done.
- ☐ In the [three-digit octal \(numeric\) representation](#), each digit stands for one group of permissions, from left to right: user, group, and other. In each of these groups, start with 0. If the read permission is present, add 4. Add 2 if write is present, and 1 for execute.



- ❑ Numeric permissions are often used by advanced administrators since they are shorter to type and pronounce, while still giving full control over all permissions.
- ❑ Examine the permissions `-rwxr-x---`. For the user, `rwx` is calculated as $4+2+1=7$. For the group, `r-x` is calculated as $4+0+1=5$, and for other users, `---` is represented with 0.
- ❑ Putting these three together, the numeric representation of those permissions is 750.
- ❑ This calculation can also be performed in the opposite direction. Look at the permissions 640.
- ❑ For the user permissions, **6 represents read (4) and write (2), which displays as rw-**. For the group part, 4 only includes read (4) and displays as `r--`. The 0 for other provides no permissions (`---`) and the final set of symbolic permissions for this file is `-rw-r-----`.



Examples :

- Remove read and write permission for group and other on file1:

```
[student@desktopX ~]$  
chmod go-rw file1
```

- Add execute permission for everyone on file2:

```
[student@desktopX ~]$  
chmod a+x file2
```



- Set read, write, and execute permission for user, read, and execute for group, and no permission for other on **sampledir**:

```
[student@desktopX ~]$  
chmod 750 sampledir
```

- ❖ **The chmod command** supports the **-R option** to **recursively set permissions on the files in an entire directory tree**.
- ❖ When using the -R option, it can be useful to set permissions symbolically using the X flag.

- ❖ This will allow the execute (search) permission to be set on directories so that their contents can be accessed, without changing permissions on most files.
- ❖ But be cautious. If a file has any execute permission set, X will set the specified execute permission on that file as well.
- ❖ **For example**, the following command will recursively set read and write access on **demodir** and all its children for their group owner, but will only apply group execute permissions to directories and files which already have execute set for user, group, and/or other.

```
[student@desktopX ~]#  
chmod -R g+rwX demodir
```



CHANGING FILE/DIRECTORY USER OR GROUP OWNERSHIP

- ❑ File ownership can be changed with the `chown` command (change owner). For example, to grant ownership of the file `foofile` to `student`, the following command could be used:

```
[root@desktopX ~]#  
chown student foofile
```

- ❑ **chown** can be used with **the -R** option to recursively change the ownership of an entire directory tree. The following command would grant ownership of **foodir** and all files and subdirectories within it to `student`:



```
[root@desktopX ~]#  
chown -R student foodir
```



NOTE

Instead of using **chown**, some users change the group ownership by using the **chgrp** command; this command works exactly the same as changing ownership with **chown**, including the use of **-R** to affect entire directory trees.

MANAGING DEFAULT PERMISSIONS AND FILE ACCESS

- ❑ SPECIAL PERMISSIONS The setuid (or setgid) permission on an executable file means that the command will run as the user (or group) of the file, not as the user that ran the command.
- ❑ One example is the passwd command:

```
[student@desktopX ~]$ ls -l /usr/bin/passwd  
-rwsr-xr-x. 1 root root 35504 Jul 16  2010 /usr/bin/passwd
```

- ❑ In a long listing, you can spot the setuid permissions by a lowercase s where you would normally expect the x (owner execute permissions) to be.
- ❑ If the owner does not have execute permissions, this will be replaced by an uppercase S.



- ❑ The sticky bit for a directory sets a special restriction on deletion of files: Only the owner of the file (and root) can delete files within the directory. An example is /tmp:

```
[student@desktopX ~]$ ls -ld /tmp  
drwxrwxrwt. 39 root root 4096 Feb  8 20:52 /tmp
```

Effects of special permissions on files and directories

SPECIAL PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
u+s (suid)	File executes as the user that owns the file, not the user that ran the file.	No effect.



SPECIAL PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
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 the group owner of the directory.
o+t (sticky)	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.

Setting special permissions

- Symbolically: setuid = u+s; setgid = g+s; sticky = o+t
- Numerically (fourth preceding digit): setuid = 4; setgid = 2; sticky = 1



Examples

- Add the setgid bit on **directory**:

```
[root@desktopX ~]#  
chmod g+s directory
```

- Set the setgid bit, and read/write/execute for user and group on **directory**:

```
[root@desktopX ~]#  
chmod 2770 directory
```



❏ → DEFAULT FILE PERMISSIONS

- The default permissions for files are set by the processes that create them.
- For example, text editors create files so they are readable and writeable, but not executable, by everyone. The same goes for shell redirection.
- Additionally, binary executables are created executable by the compilers that create them.
- The **mkdir** command creates new directories with all permissions **set—read, write, and execute**.
- These permissions are not typically set when new files and directories are created. This is because some of the permissions are cleared by the **umask** of the shell process.

The **umask command** without arguments will display the current value of the shell's **umask**:

```
[student@desktopX ~]$ umask  
0002
```

- ✓ The leading zeros indicate the special, user, and group permissions are not cleared. A **umask** of **077** clears all the group and other permissions of newly created files.
- ✓ **umask** → Octal bitmask that is used to clear the perms of new files and directories created by that process.
- ✓ Use the **umask** command with a single numeric argument to change the **umask** of the current shell.
- ✓ The numeric argument should be an octal value corresponding to the new **umask value**. If it is less than 3 digits, leading zeros are assumed.

1. Create a new file and directory to see how the default umask affects permissions.

```
[student@desktopX ~]$ touch newfile1[student@desktopX ~]$ ls -l newfile1
-rw-rw-r--. 1 student student 0 May  9 01:54 newfile1
[student@desktopX ~]$ mkdir newdir1
[student@desktopX ~]$ ls -ld newdir1
drwxrwxr-x. 2 student student 0 May  9 01:54 newdir1
```

2. Set the **umask** value to 0. This setting will not mask any of the permissions of new files. Create a new file and directory to see how this new umask affects permissions.

```
[student@desktopX ~]$ umask 0[student@desktopX ~]$ touch newfile2[student@desktopX ~]$ ls -l newfile2
-rw-rw-rw-. 1 student student 0 May  9 01:54 newfile2
[student@desktopX ~]$ mkdir newdir2
[student@desktopX ~]$ ls -ld newdir2
drwxrwxrwx. 2 student student 0 May  9 01:54 newdir2
```



3. Set the umask value to 007. This setting will mask all of the “other” permissions of new files.

```
[student@desktopX ~]$ umask 007[student@desktopX ~]$ touch  
newfile3[student@desktopX ~]$ ls -l newfile3  
-rw-rw----. 1 student student 0 May  9 01:55 newfile3  
[student@desktopX ~]$ mkdir newdir3  
[student@desktopX ~]$ ls -ld newdir3  
drwxrwx---. 2 student student 0 May  9 01:54 newdir3
```

4. Set the umask value to 027. This setting will mask write access for group members and all of the “other” permissions of new files.

```
[student@desktopX ~]$ umask 027[student@desktopX ~]$ touch  
newfile4[student@desktopX ~]$ ls -l newfile4  
-rw-r-----. 1 student student 0 May  9 01:55 newfile4  
[student@desktopX ~]$ mkdir newdir4  
[student@desktopX ~]$ ls -ld newdir4  
drwxr-x---. 2 student student 0 May  9 01:54 newdir4
```

5. Log in as root to change the default umask for unprivileged users to prohibit all access for users not in their group.

- ❖ Modify /etc/bashrc and /etc/profile to change the default umask for Bash shell users. Since the default umask for unprivileged users is 0002, look for the umask command in these files that sets the umask to that value. Change them to set the umask to 007.

```
[root@desktopX ~]# less /etc/bashrc
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 002
else
    umask 022
fi

# Only display echos from profile.d scripts if we are no login shell
[root@desktopX ~]# vim /etc/bashrc
[root@desktopX ~]# less /etc/bashrc
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 007
else
    umask 022
fi
```



```
# Only display echos from profile.d scripts if we are no login shell
[root@desktopX ~]# less /etc/profile
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 002
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do
[root@desktopX ~]# vim /etc/profile
[root@desktopX ~]# less /etc/profile
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 007
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do
```



6. Log back in as **student** and confirm that the umask changes you made are persistent.

```
[student@desktopX ~]$ umask  
0007
```



EXAMPLE

Your serverX machine has three accounts, curly, larry, and moe, who are members of a group called stooges. The password for each account is password.

1. Open a terminal window and become root on serverX.

```
[student@serverX ~]$ su -  
Password: redhat  
[root@serverX ~]#
```

2. Create the **/home/stooges** directory.

```
[root@serverX ~]# mkdir /home/stooges
```

3. Change group permissions on the **/home/stooges** directory so it belongs to the stooges group.

```
[root@serverX ~]# chown :stooges /home/stooges
```

4. Set permissions on the **/home/stooges** directory so it is a set GID bit directory (2), the owner (7) and group (7) have full read/write/execute permissions, and other users have no permission (0) to the directory.

```
[root@serverX ~]# chmod 2770 /home/stooges
```



5. Check that the permissions were set properly.

```
[root@serverX ~]# ls -ld /home/stooges
drwxrws---. 2 root stooges 1024 Dec 9 1:38 /home/stooges
```

6. Modify the global login scripts so that normal users have a umask setting which prevents others from viewing or modifying new files and directories.

```
[root@serverX ~]# vim /etc/bashrc
[root@serverX ~]# vim /etc/profile
[root@serverX ~]# less /etc/bashrc
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 007
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do
```

7. When you finish, open a terminal window on serverX and run **lab permissions grade** to confirm you have done everything correctly.

```
[student@serverX ~]$ lab permissions grade
```



Chapter 06

Controlling Access to Files with Linux File System Permissions



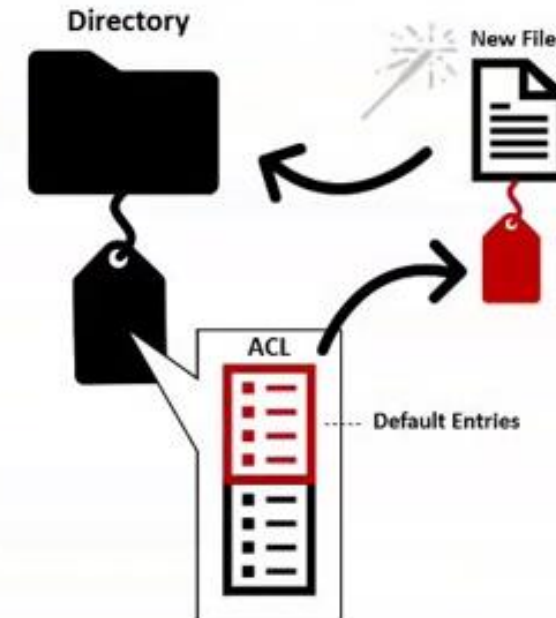
Setting ACL using Command Line Interface (CLI)

Typical ACL for a file or a directory can be represented as a following table:

Type	Name	Right	Flags
USER	jsmith	read, write, execute	Default
GROUP	jsmith	read, execute	Default
OTHERS		none	Default
USER	adoe	read, write	
GROUP	posixusers	read	

ACL entry with **Default** flag can be **only** created for a **directory** and, instead of setting **subject** and **access rights** for it, rather sets **default subject** and its **rights** for **any new file or directory** created **within**. Example:

```
[sergiy@cedarl ~]$ cd ~/
[sergiy@cedarl ~]$ mkdir webinar
[sergiy@cedarl ~]$ setfacl -d -m g:wg_staff:rx webinar
[sergiy@cedarl ~]$ setfacl -m g:wg_staff:rx webinar
[sergiy@cedarl ~]$ touch webinar/sample-0027.file
```



❑ Chmod symbolic mode

Chmod: user $\rightarrow$ u+ or - or =

Group $\rightarrow$ g+ or - or =

Others $\rightarrow$ o+ or - or =

all $\rightarrow$ a+ or - or =

❑ Chmod Octal mode

➤ R w x $\rightarrow 2*2*2=8 \rightarrow 0/1 \rightarrow$ binary



0	0	0	→0	----
0	0	1	→1	--x
0	1	0	→2	-w-
0	1	1	→3	-wx
1	0	0	→4	r--
1	0	1	→5	r-x
1	1	0	→6	rw-
1	1	1	→7	rwX



❑ 7 → 0 0 7
User group others

❑ 7 0 0
User group others

Chmod	7	5	4	file40
	Owner user	owner group	others	
	Rwx	r-x	r--	



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

