Linux Special Permissions

Student Manual

One Course Source
Your One Source for All Training

www.OneCourseSource.com

One Course Source
Your One Source for All Training

Advantages of OCS instructor-led online classes

Feature	Classroom training	Other online classes	OCS online classes
OCS Office Hours - On select Fridays each month, your instructor is available online to answer questions and assist students. This after-class support is unique in the industry.	☒	☒	✓
Small Class - OCS class sizes are limited to 15 students, providing you with more personal interaction with the instructor.	☒	☒	✓
Recorded lecture - OCS class lecture are recorded and participants have access to the recorded lecture for up to 45 days after class ends.	☒	☒	✓
First Day Free guarantee - If after the first day of class you are not satisfied, notify your instructor and receive a full refund.	☒	☒	✓
100% Guaranteed to Run Classes - Sign up for an OCS online class and be assured that it will run. Classes are not canceled due to low enrollment.	☒	☒	✓
Choose your device - Take classes on the device of your choice: Windows, Mac, Linux, Android, ipad, iphone, etc.	☒	☒	✓
Virtual Machine - Don't want to reinstall your own system? Your OCS class comes with a free VM (Virtual Machine).	☒	☒	✓
End of Class project - When class is over, your learning does not end. Take advantage of OCS's unique End of Class projects to help you affirm your new found knowledge.	☒	☒	✓
Digital Course materials - Course content is provided in searchable PDF format (except when third-party courseware is required).	☒	☒	✓
Courseware updates - If the courseware used in your class changes within 1 calendar year, you are provided with a new free digital copy of the course materials (exception: third-party materials).	☒	☒	✓
Price - OCS instructor-led online classes are the most affordable in the industry.	☒	☒	✓
Linux cert flashcards for Linux+/LPIC - Free with your OCS Linux+/LPIC training, these online flashcard help you prepare for your certification exams.	Varies	Varies	✓
Learn From Home - Learn from the comfort of your home or office.	☒	✓	✓
Real-time Instructor-Led - All OCS classes are lead by highly qualified instructors.	✓	Varies	✓
Interact with other students - Enhance your learning experience by interacting with students online during your class.	✓	Varies	✓
Lectures and hands on-labs - All classes include instructor-led lectures and hands-on labs.	✓	✓	✓

One Course Source
Your One Source for All Training

Upcoming Schedule

Course	Dates	~~Cost~~	SCALE discount
Linux+ Prep (for LX0-101)	March 25-29	~~$1,900~~	$1,235
Beginning Perl	April 2-4	~~$995~~	$647
Linux+ Prep (for LX0-102)	April 8-12	~~$1,900~~	$1,235
Intermediate Perl	April 16-18	~~$995~~	$647
Linux Essentials[1]	April 22-26	~~$1,900~~	$1,235
Beginning Tcl/TK	April 30-May 2	~~$995~~	$647
Linux System Administration I[1]	May 6-10	~~$1,900~~	$1,235
Beginning Perl	May 21-23	~~$995~~	$647
Linux System Administration II[2]	May 27-31	~~$1,900~~	$1,235
Intermediate Perl	June 4-6	~~$995~~	$647
Linux+ Prep (for LX0-101)	June 10-14	~~$1,900~~	$1,235
Advanced Perl	June 18-20	~~$995~~	$647
Linux+ Prep (for LX0-102)	June 24-28	~~$1,900~~	$1,235

[1]Prepares for RHCSA and RHCE [2]Prepares for RHCE

SCALE Discounts

SCALE attendees can lock in a huge **35%** off discount by signing up for classes during the SCALE event. Just provide your contact information and you will be given a **35%** discount when you register for class. You are not obligated if you later decide that you don't wish to attend the training. To receive this discount, you must register for your class within 45 days.

If you don't sign up for a class during SCALE, you can still receive a 20% SCALE discount by entering the code **SCALE-2013** during checkout. To receive this discount, you must register for your class within 30 days.

One Course Source
Your One Source for All Training

Unit One
Linux Special Permission

Unit topics: Page

Notes:

1.1 Special Permission: setuid

When a user runs a command that accesses files, the system checks the user's permissions for the files. In some cases, this may cause problems.

Consider a command like **passwd**. When this command runs, it edits the `/etc/shadow` file. If you look at the permissions of the `/etc/shadow` file, you will see that the permissions are: r-- --- ---

So, when the typical user runs the **passwd** command and the system tries to access (modify) the `/etc/shadow` file, if it will deny the user access…except…

The **passwd** command has a special permission set on it called setuid. When the **passwd** command is run and the command accesses files, the system pretends that the person accessing the file is the owner of the **passwd** command, not the person who is running the command.

```
[root@ocs root]# ls -l /bin/passwd
-r-sr-sr-x   3 root     sys        96796 Jul 15  1997 /bin/passwd
[root@ocs root]#  id
uid=10051(bob) gid=1(other)
# passwd                          (accesses files as root, not bob)
```

Notes:

To set setuid permission

The setuid permission can be set using either octal or symbolic methods:

```
[root@ocs root]#  ls -l /usr/bin/ls
-r-xr-xr-x   1 bin      bin        17440 Jul 15  1997 /usr/bin/ls
[root@ocs root]#  chmod a+s /usr/bin/ls
[root@ocs root]#  ls -l /usr/bin/ls
-r-sr-xr-x   1 bin      bin        17440 Jul 15  1997 /usr/bin/ls
[root@ocs root]#  chmod a-s /usr/bin/ls
[root@ocs root]#  ls -l /usr/bin/ls
-r-xr-xr-x   1 bin      bin        17440 Jul 15  1997 /usr/bin/ls
[root@ocs root]#
[root@ocs root]#  chmod 4555 /usr/bin/ls
[root@ocs root]#  ls -l /usr/bin/ls
-r-sr-xr-x   1 bin      bin        17440 Jul 15  1997 /usr/bin/ls
[root@ocs root]#  chmod 0555 /usr/bin/ls
[root@ocs root]#  ls -l /usr/bin/ls
-r-xr-xr-x   1 bin      bin        17440 Jul 15  1997 /usr/bin/ls
```

Notice the “s” character located in the owner’s permissions. This indicates that the setuid permissions is set. If the “s” is lower case, it means both setuid and the execute permission is set. If the “S” is upper case, it means only setuid (not execute) is set.

Notes:

Be careful of setuid

setuid files present a security risk on the system (especially files that are owned by root). Be careful of when you create setuid files and make sure you are aware of what setuid files are on your system

You can use the **find** command to find which programs on the system have the setuid permission set:

```
[root@ocs root]#  find / -perm -4000 –ls
{output omitted}
```

Notes:

1.2 Special Permission: setgid

There are actually two forms of setgid permissions: setgid on a file and setgid on a directory.

setgid on a file

This essentially means the same thing as setuid on a file. When someone runs the command, instead of accessing files as the group the person is a part of, the system pretends the person is a member of the group the file is owned by.

Notes:

setgid on a directory

Consider the following situation: Four people from different groups in a company are working on a common project. The four users and the groups to which they belong are:

User	**Groups**
bob	staff
steve	accounting, staff
sue	payroll
nick	admin

The company policy is for all users to have the umask 027.

All users store the files for this project in a directory called `/home/beta_prog_a`

Notes:

After a few of the users store some files in this directory, a listing of that directory looks like this:

```
[root@ocs root]#  ls -l /home/beta_prog_a
total 6
-rw-r-----   1 bob    staff       124  Mar  4  1998  1999_data
-rw-r-----   1 steve  accounting  575  Jul 15  1997  tax_table
-rw-r-----   1 sue    payroll     560  Jul 15  1997  salaries
-rw-r-----   1 nick   admin       560  Jul 15  1997  hr_data
```

Based upon the above information, you will note that there is problem here. While each user can store files in the `/home/beta_prog_a` directory, no user can see another user's work.

Notes:

To avoid this problem we can take four steps:

1. Create a new group (called beta in this case).
2. Place all user in the new group .
3. Give group ownership of the directory to the new group.
4. Set the setgid permission on the directory.

After taking these steps, any new file in the directory `home/beta_prog_a` will be group owned by the new group. Example:

```
[root@ocs root]#  mkdir /home/beta_prog_a
[root@ocs root]#  groupadd -g 133 beta
[root@ocs root]#  vi /etc/group
{add each user to the new group with the usermod command}
[root@ocs root]#  chgrp beta /home/beta_prog_a
[root@ocs root]#  chmod g+s /home/beta_prog_a
{no output for any command}
```

Notice the “s” character located in the group’s permissions. This indicates that the setgid permissions is set. If the “s” is lower case, it means both setgid and the execute permission is set. If the “S” is upper case, it means only setgid (not execute) is set.

Notes:

Be careful of setgid

setgid files present a security risk on the system (especially files that are owned by system groups). Be careful of when you create setgid files and make sure you are aware of what setguid files are on your system.

You can use the **find** command to find which programs on the system have the setgid permission set:

```
[root@ocs root]#  find / -perm -2000 –ls
{output omitted}
```

Notes:

1.3 Special Permission: sticky bit

Consider the following situation: You have a directory in which users can post announcements called `/export/home/pub`. In order for all users to be able to post (create files in) this directory, you need to give the permissions 777.

Unfortunately, these permissions also allow any user to remove any file from the pub directory. What if a user decides to run the command **rm -r *** on that directory?

The sticky bit permission give you the ability to allow anyone to add to a directory, but limits who can delete files in that directory. The only users who can delete files in a sticky bit directory are:

1. root
2. The owner of the directory
3. The owner of the file

Notes:

<u>To set sticky bit</u>

To set the sticky bit permission, use the **chmod** command:

```
[root@ocs root]# chmod 1777 /export/home/pub
[root@ocs root]# ls -ld /export/home/pub
drwxrwxrwt   2 root     other        512 Feb 18 18:11 /export/home/pub
```

Notice the “t” character in the place where the “x” should be for others. This “t” tell you that the sticky bit has been set on this directory.

Notes:

1.4 Access Control Lists

Access Control Lists Essentials

Consider the following situation: There are 500 user accounts on a system. The group “payroll” has 15 users assigned to it. Bob, who is a member of the payroll group, creates the file “salaries” and gives it the permissions 660.

In this scenario, Bob and all the members of the payroll group have the ability to read and modify the salaries file. Nobody else can do anything with this file.

The CEO of the company, who is not in the payroll group, requests to have read access to this file. There are two methods of giving the CEO access to the file:

1. Add the CEO to the payroll group.
2. Give read permission to everyone.

Obviously, the second method is a very bad idea. The first method might be ok; however, there are a couple of disadvantages: #1. Each user can only be assigned to 16 groups and #2. The CEO now has access to any file that has group ownership.

The ext3 filesystem includes a feature called Access Control Lists. ACL’s allow you to specify permissions for individual users or groups.

Notes:

Enable ACLs

While ext3 filesystems are capable of allowing ACLs, they don't have this feature enabled by default. To enable ACLs, you need to have the filesystem mounted with the "acl" option.

The mounting process will be discussed in greater detail in a future Unit. For now use the following command to enable ACLs on a filesystem:

```
mount -o remount,acl /mount_point
```

Notes:

Setting ACL’s

To create a new ACL for a file, use the **setfacl** command with the **-m** option. The syntax of the **setfacl** command when using the **-m** option is:

```
setfacl --set user::perm,group::perm,other:perm,mask:perm,[user:UID:perm],[group:GID:perm] filename
```

Note: “user”, “group”, “other” and “mask” can be abbreviated to “u”, “g”, “o” and “m”.

The “perm” can be give either in octal format or symbolic:

Symbolic Permission	Octal Permission
rwx	7
rw-	6
r-x	5
r--	4
-wx	3
-w-	2
--x	1
---	0

Notes:

To give the sample.txt file the permissions of…

Owner:	rwx
Group:	r-x
Others:	r--
Mask:	r-x
bob	r-x
games	r-x

....use the command:

```
[root@ocs boot]# setfacl -m u::7,g::5,o:4,m:5,u:bob:5,g:games:5 sample.txt
```

…or the command:

```
[root@ocs boot]# setfacl -m u::rwx,g::r-x,o:r--,m:r-x,u:bob:r-x,g:games:r-x sample.txt
```

Note: You can also specify either a user's name or UID number.

The Mask setting

The mask setting enforces a "maximum" permission for all users and groups (except the owner) on the file. Therefore, in the previous example, bob's effective permissions are just read (not read/write). We will this setting in more detail after looking at how to display ACLs.

Notes:

Displaying ACLs

When a file has an ACL, a "+" character will be displayed next to the permissions of the file when you run the **ls -l** command:

```
[root@ocs boot]# ls -l sample.txt
-rwxr-xr--+   1 root     root           0 Jan 22 09:33 sample.txt
```

To display ACLs, use the command **getfacl**:

```
[root@ocs boot]# getfacl sample.txt
# file: sample.txt
# owner: root
# group: root
user::rwx
user:bob:r-x
group::r-x
group:games:r-x
mask::r-x
other::r--
```

Notes:

More details regarding the mask setting

The mask setting is intended to provide you a method of avoiding accidentally providing permissions to a file that give undesired access to the file.

Unfortunately, this often means that the permissions that you specify are not the permissions that you end up getting:

```
[root@ocs boot]# setfacl -m m:4 sample.txt
[root@ocs boot]# getfacl sample.txt
# file: sample.txt
# owner: root
# group: root
user::rwx
user:bo:r-x             #effective:r--
group::r-x              #effective:r--
group:games:r-x         #effective:r--
mask::r--
other::r--
```

In this example, the user bob only gets read permission on the file even though our original **setfacl** command requested both read and write permissions.

Note: If you change a user or group ACL, the mask setting may be changed as well to allow the specified permissions.

Notes:

Removing ACLs

To remove an ACL, use the **-x** option:

```
[root@ocs boot]# setfacl -x u:bo sample.txt
[root@ocs boot]# ls -l sample.txt
-rwxr-xr--    1 root     root           0 Jan 22 09:33 sample.txt
```

Note: If the ACL permission is the last one in the ACL table, the ACL table will be removed and the "+" character next to the permissions will no longer be displayed.

Other useful **setfacl** options

Option	**Description**
-b	Remove all ACLs (owner, group & other permissions still apply)
-R	Apply ACLs to directory and all contents (recursive)

Notes:

Default ACLs

If you apply an ACL to a directory, that ACL will be applied automatically to all new files and subdirectories created within that directory. When you initially create an ACL for a directory, you must specify an ACL permission for the user owner, group owner, others and mask. You also need to specify that these are default ACLs by placing a “d” character in front of each permission set:

```
[root@ocs boot]# mkdir acl_dir
[root@ocs boot]# setfacl -m d:u::7,d:g::7,d:o:5,d:m:7,d:u:bob:7 acl_dir
```

Default permission sets show up in a different location than regular ACL entries when you use the **getfacl** command:

```
[root@ocs boot]# getfacl acl_dir
# file: acl_dir
# owner: root
# group: root
user::rwx
group::r-x
other::r-x
default:user::rwx
default:user:bob:rwx
default:group::rwx
default:mask::rwx
default:other::r-x
```

Notes:

Creating files in a ACL directory

When you create a new file in a directory that has a default ACL set on it, the directory's ACL is applied to the new file **after** it has been "filtered" by the umask setting:

```
[root@ocs boot]# cd acl_dir
[root@ocs acl_dir]# touch acl.txt
[root@ocs acl_dir]# ls -l acl.txt
-rw-rw-r--+   1 root     root           0 Jan 22 09:53 acl.txt
[root@ocs acl_dir]# getfacl acl.txt
# file: acl.txt
# owner: root
# group: root
user::rw-
user:bo:rwx                    #effective:rw-
group::rwx                     #effective:rw-
mask::rw-
other::r--
```

If the permissions specified by the ACL are higher than the umask setting then the umask setting "wins out".

Notes:

Creating a subdirectory in an ACL directory

When you create a directory in an ACL directory, the umask setting is not used. The ACL permissions, including the default permissions, are passed from the parent directory to the subdirectory:

```
[root@ocs acl_dir]# mkdir new_acl
[root@ocs acl_dir]# getfacl new_acl
# file: new_acl
# owner: root
# group: root
user::rwx
user:bo:rwx
group::rwx
mask::rwx
other::r-x
default:user::rwx
default:user:bo:rwx
default:group::rwx
default:mask::rwx
default:other::r-x
```

Notes:

1.5 Summary of Commands and Files

Command	Description
chmod	Changes file and directory permissions
getfacl	Displays ACL permissions of files and directories
setfacl	Sets ACL permissions on files and directories

File	Description
None	

Notes:

1.6 Additional Resources

Books

None

Web sites

http://www.tldp.org/HOWTO/Security-HOWTO/index.html - Chapter #5: Files and Filesystem Security

Man pages

chmod
getfacl
setfacl

One Course Source
Your One Source for All Training

Appendix
Basic File Security

Unit topics: Page

Notes:

2.1 Basic Linux Permissions

Permissions are Linux's method of protecting files and directories. Every file or directory is owned by a user and assigned to a group. The owner of a file has the right to set permissions in order to protect the file from being accessed, modified or destroyed.

Determining permissions

To determine the permissions of a file, use the **ls -l** command. The first character of the output of the **ls -l** command specifies the file type. The next nine characters represent the permissions set on the file. There are three types of permissions: r (read), w (write), and x (execute). These permissions have different meanings for files and directories.

First three permissions are for the **user owner**, second three are for people in the **group**, and last three are for everyone else (**others**).

```
-rw-r--r--   1     steve      staff    512     Oct 11 10:43      tmp
    ↑              ↑            ↑
permissions   user owner    group
```

Therefore, in the preceding example, the owner of the file (steve) has read and write permissions, the members of the group (staff) have read permission and everyone else has read permission.

Notes:

Group accounts

Groups were invented to provide more flexibility when issuing permissions. Every user is a member of at least one group (a primary group) and may be a member of additional (secondary) groups. To see the groups you belong to, type the **groups** command.

File permissions vs. Directory permissions

Permissions have different meaning on files and directories. The following chart illustrates the differences:

Permission	**Symbol**	**Meaning for Files**	**Meaning for Directories**
Read	r	Can view or copy file	Can list with **ls**
Write	w	Can modify file	Can add or delete files in the directory (if execute permission is also set)
Execute	x	Can run file like a program	Can cd to that directory. Can also use that directory in a path.

Notes:

Changing Permissions

Only the person who owns the file (and the root user) can change the file's permissions.

There are two methods of changing the permissions on a file: symbolic and octal. The **chmod** command is used in both cases.

Symbolic method

The symbolic method is useful for changing just one or two permissions. Following the **chmod** command, you specify three items: whose permission you wish to change, whether you want to add or remove the permission, and the permission itself. The following chart illustrates the possibilities:

Who	**Operand:**	**Permission:**
u (user/owner)	- (remove)	r (read)
g (group)	+ (add)	w(write)
o (other)		x (execute)
a (all three)		

Notes:

For example, the following removes read permission for the group for the file `myprofile`:

```
[student@ocs1 student]#  ls -l
-rw-r--r--    1      steve       staff     512       Oct 11 10:43       myprofile
[student@ocs1 student]#  chmod g-r myprofile
[student@ocs1 student]#  ls -l
-rw----r--    1      steve       staff     512       Oct 11 10:43       myprofile
```

You can specify multiple permissions to change; the following example will add execute permission for the user owner and remove read permission for others for the file `myprofile`:

```
[student@ocs1 student]#  ls -l
-rw----r--    1      steve       staff     512       Oct 11 10:43       myprofile
[student@ocs1 student]#  chmod u+x,o-r myprofile
[student@ocs1 student]#  ls -l
-rwx------    1      steve       staff     512       Oct 11 10:43       myprofile
```

Notes:

Octal Method

The octal method is useful when you have to change many permissions on a file. It is based on the octal numbering system:

4=read
2=write
1=execute

By using a combination of numbers from 0 to 7, any possible combination of read, write and execute permissions can be specified. The following chart illustrates all of the possible combinations:

Value	**Meaning**
7	r w x
6	r w -
5	r - x
4	r - -
3	- w x
2	- w -
1	- - x
0	- - -

When the octal method is used to change permissions, all nine permissions must be specified. Because of this, the symbolic method is generally easier for changing a few permissions while the octal method is better for changes that are more drastic.

Notes:

To change the permission for the myprofile file to the permissions rwxrw-r-- use the following command:

```
[student@ocs1 student]#  cd
[student@ocs1 student]#  ls -l
-rw----r--    1      steve       staff     512       Oct 11 10:43       myprofile
[student@ocs1 student]#  chmod 764 myprofile
[student@ocs1 student]#  ls -l
- rwxrw-r--  1      steve       staff     512       Oct 11 10:43       myprofile
```

Notes:

2.2 Setting Default Permissions

When you create a file or directory, predefined permission are set on that file or directory. The current default permissions are:

files	**rw-rw-rw-**
Directories	**rwxrwxrwx**

To change the default permissions, you must change the **umask** setting.

The following text and chart can be used to determine an umask entry:

1. Know the MAXimum possible permissions for files and directories.
2. Determine what you want your permissions to be when you create either a new file or directory. You will need to pick one (either file or directory) and understand that the umask command affects both.
3. Determine what permissions of the MAXimum permissions need to be taken out (MASKed out).
4. Compute the values (in octal notation) of what needs to be MASKed out for the owner, group and other. The resulting three numbers is what you will use to set your umask.

Step		**File**	**Directory**
1	MAX	rw- rw- rw-	rwx rwx rwx
3	MASK	--- -m- mmm	--- -m- mmm
4	umask=027	0 2 7	0 2 7
2	desire	rw- r-- ---	rwx r-x ---

Notes:

The **umask** command displays and changes default permissions:

```
[student@ocs1 student]#  umask
027
[student@ocs1 student]#  umask 026
026
```

After setting this **umask**, if you were to log out (or open a new shell) you would lose the new **umask** setting. In order for this new **umask** setting to be a permanently changed, you must place the **umask** command into your initialization file (A later Unit will cover initialization files).