File Security

Lock Down Your Data

Brian Reames
January 22, 2012

Table of Contents

Table of Contents

1.0 Basic Linux Permissions

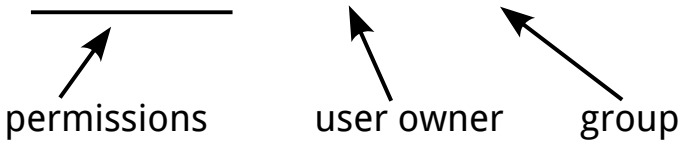
Permissions are Linux's method of protecting files and directories. Every file or directory is owned by a user and assigned to a group. The owner of a file has the right to set permissions in order to protect the file from being accessed, modified or destroyed.

1.1 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.

The 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
```



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.

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.

1.2 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.

1.3 Changing Permissions

Only the person who owns a file (and the root user) can change a 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.

1.3.1 Symbolic method

The symbolic method is useful for changing just one or two permissions. Following the **chmod** command, you specify three items: whose permissions 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)	= (set)	x (execute)
a (all three)		

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

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

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

1.3.2 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.

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

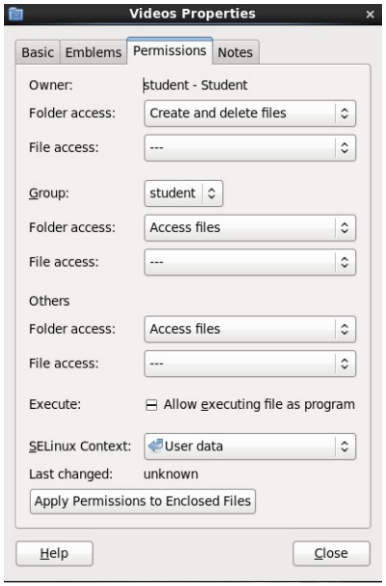
```
[steve@machine steve]$ cd
[steve@machine steve]# ls -l
-rw----r-- 1 steve   staff 512 Oct 11  10:43 myprofile
[steve@machine steve]# chmod 764 myprofile
[steve@machine steve]# ls -l
-rwxrw-r-- 1 steve staff  512 Oct 11  10:43 myprofile
```

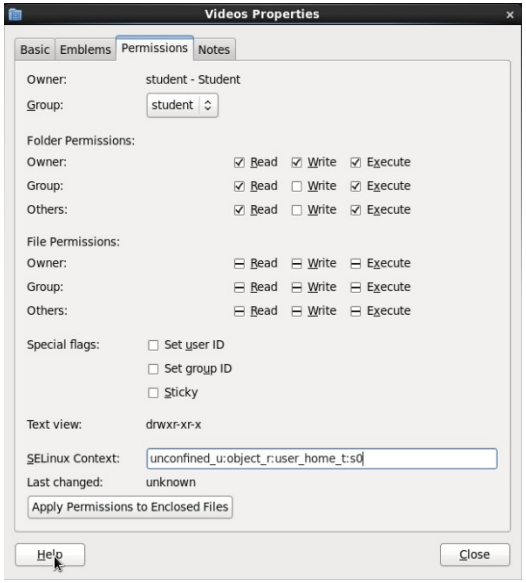
1.3.3 Changing Permissions Graphically

If you use the File Browser application (**nautilus**), you can set permissions for files or directories you own.

Navigate to where the file is located and then select it. Next, using the File menu, or the right-click menu, choose Properties. Then click on the Permissions tab.

If the file is a folder (directory), then there are two sets of permissions to set. The folder access permissions determine the access to the directory. The file access permissions will be applied to the existing files within the directory. Remember to click the button at the bottom of the dialog to "Apply Permissions to Enclosed files".





1.4 Examples of Unintentional Access

Example 1 Can Janet read or modify Bob's file, test.file?

```
[bob@machine example]$ ls -l
-rw-r--r-- 0 bob users 100 2011-01-01 11:00 test.file
```

Example 2 Suppose Janet and Bob both belong to the "users" Can Janet read or modify Bob's file, /example/test.file, now?

```
[bob@machine example]$ ls –ld
drwxrwxr-- 0 root users 100 2011-01-01 11:00 example
[bob@machine example]$ ls -l
-rw-r--r-- 0 bob  users 100 2011-01-01 11:01 test.file
```

Example 3 Suppose an administrator creates a subdirectory for bob to store his files in. Would Janet be able to view or modify /example/bob/test.file?

```
[bob@machine example]$ ls –ld
drwxrwxr-- 0 root users 100 2011-01-01 11:00 example
[bob@machine example]$ ls –ld bob/
drwxrwxr-- 0 bob  bob   100 2011-01-01 15:27 bob
[bob@machine example]$ ls –l
-rw-r--r-- 0 bob  users 100 2011-01-01 11:01 test.file
```

Answer 1 Maybe...Janet must also have read and execute permission on the parent directory in order to get to the file.

Answer 2 Yes; Janet will be able to access the information in the file. No; Janet should not be able to modify the file...but she can copy the contents to a new file, modify it, and delete the original. **Janet can delete Bob's test.file because she has "write" permission on the directory!** She may also be able to force changes to the file, depending on how she decides to modify it.

Answer 3 No; Janet will not be able to view the file because she does not have "execute" permission on its parent directory. No; she cannot modify the file because she does not have "write" permission on the file and she cannot "execute" into the directory. Janet cannot force changes to the file because she does not have "write" or "execute" permission on the parent directory. Furthermore, she cannot delete the /example/bob directory; **rmdir** will fail because the directory is not empty. **rm -r** will fail because she does not have "execute" permission on the directory.

As shown above permissions in shared directories can quickly become complex. Advanced permissions exist to deal with these and other logistical concerns. These permissions will be discussed later in this presentation.

2.0 Setting Default Permissions

When you create a file or directory, predefined permissions are set on that file or directory. The maximum 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 ---

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

```
[user@machine ~]# umask
027
[user@machine ~]# umask 026
026
```

The **umask** command will only alter the umask of the shell it is executed in; the new **umask** will be lost upon logging out or opening a new shell. In order for this new **umask** setting to be a permanently changed, you must place the **umask** command into your initialization file.

```
[user@machine ~]$ echo “umask 026” >> .bashrc
```

The above command will append the new umask rule to your initialization file. Any future bash shells opened by “user” will have a **umask** set automatically to 026.

2.1 Common umask Settings

Below are illustrated a few common umask settings, the default permissions they create, and some advantages and disadvantages of each scheme.

umask 002	**Files:**	664	-rw-rw-r--	Common to systems that use user-only groups (user, bob; group bob). Under this scheme, files will be readable by anyone on the system by default.
	Directories:	775	drwxrwxr-x	
umask 022	**Files:**	644	-rw-r--r--	Traditionally the default for Linux and Unix systems. More secure, but still flexible.
	Directories:	755	drwxr-xr-x	
umask 077	**Files:**	600	-rw-------	Very secure. However, this may cause access problems to common scripts. Absolutely no collaboration by default.
	Directories:	700	drwx------	

3.0 Advanced Linux Permissions

In most circumstances, basic Linux permissions will be enough to accommodate the security needs of individual users or organizations. However, when multiple users need to work routinely in the same directories and files, these permissions may not be enough. The special permissions **setuid, setgid** and the **sticky bit** are designed to address these concerns.

Even finer-grained control can be achieved with Access Control Lists (ACLs) where needed.

3.1 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: --- --- ---

So, when the typical user runs the **passwd** command and the system tries to access (modify) the /etc/shadow file, 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.

```
[bob@machine ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31768 Jan 28 2010 /usr/bin/passwd
[bob@machine ~]$ id
uid=10051(bob) gid=1001(other)
[bob@machine ~]$ passwd          (accesses files as root, not bob)
```

3.1.1 Seting the setuid permission

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

- To add the **setuid** permission symbolically, run: **chmod u+s <file>**
- To add the **setuid** permission numerically, add 4000 to the file's existing permissions: **chmod 4775 <file>**
- To remove the **setuid** permission symbolically, run: **chmod u-s <file>**
- To remove the **setuid** permission numerically, subtract 4000 from the file's existing permissions: **chmod 0775 <file>**

In the following example, adding the **setuid** permission to the **ls** command would allow user bob the ability to view the contents of a restricted directory (such as /etc/lvm).

```
[bob@machine ~]$ ls -l /bin/ls
-rwxr-xr-x. 1 root root 111744 Jun 14 2010 /bin/ls
[bob@machine ~]$ ls /etc/lvm/
ls: cannot open directory /etc/lvm/: Permission denied
[bob@machine ~]$ ls -ld /etc/lvm/
drwx------. 5 root root 4096 May 6 15:40 /etc/lvm/
[bob@machine ~]$ exit
logout
[root@machine ~]# chmod u+s /bin/ls
[root@machine ~]# ls -l /bin/ls
-rwsr-xr-x. 1 root root 111744 Jun 14 2010 /bin/ls
[root@machine ~]# su - bob
[bob@machine ~]$ ls /etc/lvm/
archive  backup  cache  lvm.conf
[bob@machine ~]$ exit
logout
[root@machine ~]# chmod a-s /bin/ls
[root@machine ~]# ls -l /bin/ls
-rwxr-xr-x. 1 root root 111744 Jun 14  2010 /bin/ls
```

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

3.1.2 Caution Regarding setuid

setuid files present a security risk on the system (especially files that are owned by root). Be careful of creating files with **setuid** and make sure you are aware of which files have **setuid** on your system.

*Note: the **setuid** permission is not honored on regular user scripts, such as bash scripts.*

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

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

3.2 Special Permission: setgid

Setgid is similar to **setuid**, but uses the group owner permissions. There are actually two forms of **setgid** permissions: **setgid** on a file and **setgid** on a directory. How **setgid** works depends on if it is set on a file or directory, and may not always be obvious.

3.2.1 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 any group the person is a part of, the system pretends the person is a member of the group that owns the file.

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

```
[root@machine ~]#  find / -type f -perm -2000 -ls
{output omitted}
```

3.2.2 setgid on a directory

Files created in a directory with **setgid** are automatically group-owned by the directories group, not the creators primary group.

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

User	Primary Group	Supplementary Groups
bob	bob	**beta**, staff
steve	steve	accounting, **beta**, staff
sue	sue	**beta**, payroll
nick	nick	admin, **beta**

The company policy is for all users to have the umask 027. With this umask, new files will have initial permissions of 640 and new directories will have initial permissions of 750.

In an attempt at collaboration, a new directory is created with the following characteristics. It is group owned by beta (a group common to all four users) and the group has read, write, and execute permissions on the directory. All users will store the files for this project in a directory called /home/beta_prog_a

```
[root@machine ~]# mkdir /home/beta_prog_a
[root@machine ~]# chgrp beta /home/beta_prog_a
[root@machine ~]# chmod g+w /home/beta_prog_a
[root@machine ~]# ls -ld /home/beta_prog_a
drwxrwx---. 2 root beta 4096 May 24 08:28 /home/beta_prog_a
```

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

```
[root@ocs ~]# ls -l /home/beta_prog_a/
total 6
-rw-r-----. 1 bob       bob   124 May 24 08:31 2011_data
-rw-r-----. 1 nick  nick  575 May 24 08:32 hr_data
-rw-r-----. 1 sue       sue   560 May 24 08:32 salaries
-rw-r-----. 1 steve steve 560 May 24 08:32 tax_table
```

Based upon the above information, you will note that there is a problem here. While each user can store files in the /home/beta_prog_a directory, no user can see another user's work. In its current state, this directory is not truly collaborative.

We can correct this problem with a few easy steps:

1. Add the **setgid** permission to the directory.
2. Recursively change group ownership to any existing files in the directory.

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

```
[root@ocs ~]# chmod g+s /home/beta_prog_a/
[root@ocs ~]# chgrp -R beta /home/beta_prog_a/
[root@ocs ~]# ls -ld /home/beta_prog_a/
drwxrws---. 2 root beta 4096 May 24 08:32 /home/beta_prog_a/
[root@ocs ~]# ls -l /home/beta_prog_a/
total 6
-rw-r-----. 1 bob       beta 124 May 24 08:31 2011_data
-rw-r-----. 1 nick  beta 575 May 24 08:32 hr_data
-rw-r-----. 1 sue       beta 560 May 24 08:32 salaries
-rw-r-----. 1 steve beta 560 May 24 08:32 tax_table
```

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 are set. If the “S” is upper case, it means only **setgid** (not execute) is set.

Note: While members of the beta group can read each other’s files, they cannot easily modify the files. This could be corrected by either changing their **umask** values or setting default ACLs on the directory (covered later in this presentation).

3.2.3 Caution Regarding setgid

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

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

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

The **setgid** permission on directories is typically used for collaboration and is not much of a security risk. You can use the **find** command to find which directories on the system have the **setgid** permission set:

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

3.3 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 set the permissions to 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 gives 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

3.3.1 Seting sticky bit

To set the sticky bit permission symbolically, run: **chmod o+t *DIRECTORY***
To set the sticky bit permission numerically, add *1000* to the directory's existing permissions.

```
[root@ocs ~]# chmod 1777 /export/home/pub
[root@ocs ~]# ls -ld /export/home/pub
drwxrwxrwt. 2 root root 4096 May 24 09:10 /export/home/pub
```

Notice the "t" character in the place where the "x" should be for others. If the "t" is lower case, it means both the **sticky bit** and execute permission are set. If the "T" is upper case, it means only the sticky bit (not execute) is set.

4.0 Access Control Lists

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. Bob also changes the file to be group owned by the payroll group.

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.
2. The CEO now has access to any file that is group owned by payroll.

The ext3 and ext4 filesystems include a feature called Access Control Lists. ACLs allow you to specify permissions for individual users or groups.

4.1 Enabling ACLs

While ext3 and ext4 filesystems are capable of allowing ACLs, ext3 and ext4 filesystems created after the system was installed don't normally have this feature enabled by default. To enable ACLs, you may need to have the filesystem mounted with the "acl" option or you need to add acl as a default mount option to the filesystem's superblock. Note that any ext3 or ext4 filesystem created during installation (including the /boot and / filesystems) *should* already have this feature enabled by default.

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
```

To enable ACLs by default at boot, modify the "options" column for the filesystem in the /etc/fstab file to include "acl"

```
/dev/sdb  /deep_storage  ext4  acl  0 0
```

4.2 Setting ACLs

To create a new ACL for a file or a directory, use the **setfacl** command with the **-m** option. The following are some typical examples of using the **setfacl** command. Additional examples can be found at the bottom of the man page for **setfacl**.

```
setfacl -m user:bob:6 sample.txt
setfacl -m group:games:rw sample.txt
```

The **setfacl** command is fairly flexible allowing the use of full words or abbreviations as well as symbolic or octal permissions. When using symbolic permissions, dashes can be omitted (except for 0 where one dash is required).

Abbreviations		Symbolic & Octal Permissions			
user	= u	rwx	= 7	-wx	= 3
group	= g	rw-	= 6	-w-	= 2
other	= o	r-x	= 5	--x	= 1
mask	= m	r--	= 4	---	= 0
default	= d				

Consider how to give the saleries.txt file the permissions of...

Owner:	rw-
Group:	r--
Others:	r--
Mask:	rw-
ceo	rw-
games	rw-

```
[root@machine payroll]# ls -l saleries.txt
-rw-r--r-- bob payroll 100 2012-01-11 01:12 saleries.txt
```

The regular file permissions for the owner, the group, and others already match, so there is no need to set specific ACLs for them. You can set the ACL permissions for user bob and group games with either one command or two in either the short format of the long format:

```
[root@machine payroll]# setfacl -m u:ceo:6,g:games:6 saleries.txt
```

...or the commands:

```
[root@machine payroll]# setfacl -m user:ceo:rw saleries.txt
[root@machine payroll]# setfacl -m group:games:rw saleries.txt
```

Note: UID and GID numbers can be used instead of user or group names.

4.3 The Mask setting

The mask setting enforces a “maximum” permission for all users and groups (except the owner) of the file. The mask is automatically calculated when a new ACL is placed on a file or directory. When calculated automatically, it equals the maximum combined permissions of any user or group (except the owner). When set explicitly, it can modify a user or group’s effective permissions. Therefore, in the previous example, if we also ran (**setfacl -m m:4 sample.txt**) the effective permissions for both user bob and group games would be just read (not read/write). We will examine this setting in more detail after looking at how to display ACLs.

The mask setting is intended to provide you a method of avoiding accidentally setting permissions 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@machine payroll]# setfacl -m m:4 saleries.txt
[root@machine payroll]# getfacl saleries.txt
# file: saleries.txt
# owner: root
# group: root
user::rw-
user:bob:rw-            #effective:r--
group::r--
group:games:rw-         #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.

4.4 Order of Precedence for Permissions

Setting ACLs on a file or a directory does not negate user, group, and other permissions. The following evaluations take place. Note that with the exception of groups, access is granted based on the first match.

1. Are you the user owner of the file? If so you get the regular user owner's permissions.
2. Is there a specific ACL for your user? If so, you get the permissions specified by the ACL for your user.
3. Do you belong to the group that owns the file or is there a specific ACL for a group to which you belong? If so, you get the maximum combined regular and ACL permissions for all of your groups.
4. If none of the above match, you get the permissions designated for others.

4.5 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@machine payroll]# ls -l saleries.txt
-rw-rw-r--+ 1 bob payroll 100 2012-01-11 01:12 saleries.txt
```

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

```
[root@machine payroll]# getfacl saleries.txt
# file: saleries.txt
# owner: root
# group: root
user::rw-
user:ceo:rw-
group::r--
group:games:rw-
mask::rw-
other::r--
```

4.6 Removing ACLs

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

```
[root@machine payroll]# setfacl -x u:ceo saleries.txt
[root@machine payroll]# getfacl saleries.txt
# file: saleries.txt
# owner: bob
# group: payroll
user::rw-
group::r--
group:games:rw-
mask::rw-
other::r--
```

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)

4.7 Default ACLs

If you apply a default ACL to a directory, that ACL will be applied automatically to all new files and subdirectories created within that directory. This can be done by adding either “d” or “default” to the **setfacl** command.

```
[root@machine home]# mkdir acl_dir
[root@machine home]# setfacl -m d:u:bob:7 acl_dir
```

Notice the location of default ACL permissions when running **getfacl**:

```
[root@machien home]# 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::r-x
default:mask::rwx
default:other::r-x
```

Note: you may want to add a regular ACL to give bob write permission to the directory. Without this, bob will not be able to create new files in the directory.

```
[root@machine home]# setfacl -m u:bob:7 acl_dir
```

4.7.1 Creating Files in an 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@machine home]# cd acl_dir
[root@machine acl_dir]# touch acl.txt
[root@machine acl_dir]# ls -l acl.txt
-rw-rw-r--+ 1 root root 0 May 24 11:49 acl.txt
[root@ocs acl_dir]# getfacl acl.txt
# file: acl.txt
# owner: root
# group: root
user::rw-
user:bob:rwx    #effective:rw-
group::r-x      #effective:r--
mask::rw-
other::r--
```

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

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

Appendix A: Summary of Commands

Command	Description
chmod	Changes file and directory permissions
getfacl	Displays ACL permissions of files and directories
groups	Displays what groups the current user is a member of
setfacl	Sets ACL permissions on files and directories
umask	Sets the default permissions for all new files or directories

Appendix B: Additional Resources

Books

Cert Guide
By Damian Tomasino
Publisher: Pearson
ISBN: 978-0-321-76795-0
Pgs. 131-141

Running Linux
By Matt Welsh, Matthias Kalle Dalheimer, Terry Dawson, Lar Kaufman
Publisher: O'Reilly
ISBN: 0596007604
Chapter #4

Web sites

www.OneCourseSource.com/pdf/ResourceGuide - Sections: 4.1.10, 4.4.7; 4.4.8

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

Man pages

Chmod	groups	umask
getfacl	setfacl	