File permissions and access control

In this section, you learn about:
• Users and groups
• Permissions on files and directories
• Changing permissions
• Access modes
• Default file creation modes

User and groups
By now, you know that Linux is a multi - user system and that each user belongs to one primary group
and possibly additional groups. It is also possible to log in as one user and become another user using
the su or sudo commands. Ownership of files in Linux is closely related to user ids and groups, so let's review some basic user and group information.

Who am I?
If you have not become another user, your id is still the one you used to log in. If you have become another user, your prompt may include your user id. If your prompt does not include your user id, then you can use the whoami command to check your current effective id. Listing 1 shows some
Examples:

Listing1. **Determining effective user id**

/home/ian$ whoami
tom
/home/ian$ exit
exit
$ whoami
ian

What groups am I in?
Similarly, you can find out what groups you are in by using the groups command.
You can find out both user and group information using the id command. Add a user id parameter to either groups or id to see information for that user id instead of the current user id. See Listing 2 for some examples.

Listing 2. **Determining group membership**

$ su tom
Password:
/home/ian$ groups
Testgroup

/home/ian$ id
uid=1001(tom) gid=1001(Testgroup) groups=1001(Testgroup)
/home/ian$ exit
$ groups
ian adm dialout cdrom floppy audio dip video plugdev lpadmin scanner admin Testgroup
$ id
uid=1000(ian) gid=1000(ian) groups=4(adm),20(dialout),24(cdrom),25(floppy),
29(audio),30(dip),44(video),46(plugdev),104(lpadmin),105(scanner),106(admin),
1000(ian),1001(Testgroup)
$ groups tom
tom : Testgroup

Creating groups and adding users to groups

Every group on a Linux system can have anywhere from no member s to as many member s as there are user account s on the system. Group member ship is controlled by the */etc /group* file. This is the file that contains a list of all the groups and member s of those groups on a Linux system.
Each user, when logging on to a Linux system, logs into their primary group. This is the group that specifies default group member ship and is set in the user's configuration file. When a user is logged into their primary group they can access files and run programs that are associated with that particular group to which they belong. If a user wants to access files or programs that are not in their primary group they can switch to the group with which the particular file or program is associated.
However, the user must be a member of that group in order to switch to that group. This is an excellent mean of controlling security on a Linux system. To change to a different group after logging into the system use the newgrp command. The syntax for this command is as follows:
$ newgrp group-name
For example: newgrp faculty-admins
In a Linux system, only the root account, superuser, has the power to create and manage groups. These tasks are performed using simple commands for creating, renaming, or deleting groups from the system. Most Linux users are assigned to a group during the account creation process. The following syntax is used to create a group on a Linux system:
$groupadd group-name
For example: *groupadd faculty-admins*
This command creates the faculty- admins group. After the group has been created, users can then be added to the group. This can be done with the following commands.
One way to add a user to a group is to do it when creating the user account. The following syntax is used to expand on the useradd command:
$useradd –g group username –c "real name"
For example: *useradd –g faculty-admins itckuser –c "ITCK User"*

To add a user (member) to a group, the following command is used:
$ gpasswd –M username group
The *gpasswd* command can be used to modify existing groups. Only the root user (system administrator) can use the –M (members) option.
In Debian based distributions adduser is also used to add existing users to groups. For example:
$ adduser user1 faculty-admins
The groups command can be used to display the current groups to which a user belongs.
$ groups

File ownership and permissions

Just as every user has an id and is a member of one primary group, so every file on a Linux system has one owner and one group associated with it.

Ordinary files

Use the ls - l command to display the owner and group.

Listing 3. Determining file ownership

gretchen@pinguino:~$ ls -l /bin/bash .bashrc
-rw-r--r-- 1 gretchen Testgroup 2227 Dec 20 10:06 .bashrc
-rwxr-xr-x 1 root root 645140 Oct 5 08:16 /bin/bash

In this particular example, user gretchen's .bashrc file is owned by her and is in the Testgroup group, which is her primary group. Similarly, /bin /ba sh is owned by user root and is in the group root. User names and groups names come from separate namespaces, so a group name may be the same as a user name. In fact, many distributions default to creating a matching group for each new user.
The Linux permissions model has three types of permission for each filesystem object. The permissions are read (r), write (w), and execute (x). Write permission includes the ability to alter or delete an object. In addition, these permissions are specified separately for the file's owner, member s of the file's group, and everyone else.
Referring back to the first column of Listing 3, notice that it contains a ten- character string. The first character describes the type of object (- for an ordinary file in this example) and the remaining nine characters represent three groups of three character s. The first group indicates the read, write, and execute permissions for the file's owner. A - Indicates that the corresponding permission is not granted. So user gretchen can read and write the .bashrc file, but not execute it; while root can read, write, and execute the /bin / ba sh file. The second group indicates the read, write, and execute permissions for the file's group. Members of the Testgroup group can read gretchen's .bashrc file, but not write it, as can everyone else. Similarly, member s of the root group and everyone else can read or execute the /bin / ba sh file.

Octal digit	Text equivalent	Binary	Meaning
0	---	000	All types of access are denied
1	--x	001	Execute access is allowed only
2	-w-	010	Write access is allowed only
3	-wx	011	Write and execute access are allowed
4	r-	100	Read access is allowed only
5	r-x	101	Read and execute access are allowed
6	rw-	110	Read and write access are allowed
7	rwx	111	Everything is allowed

Directories

Directories use the same permissions flags as regular files but they are interpreted differently. Read permission for a directory allows a user with that permission to list the content s of the directory. Write permission means a user with that permission can create or delete files in the directory. Execute permission allows the user to enter the directory and access any subdirectories. Without execute permission, the filesystem objects inside a directory are not accessible. Without read permission, the filesystem objects inside a directory are not viewable, but these objects can still be accessed as long as you know the full path to the object on disk. Listing 4 is a somewhat artificial example that illustrates these points.

Listing 4. Permissions and directories

```
ian@pinguino:~$ ls -l /home
total 8
drwxr-x--- 2 greg Testgroup 60 2005-12-20 11:37 greg
drwx------ 13 gretchen Testgroup 4096 2005-12-21 12:22 gretchen
drwxr-xr-x 15 ian ian 4096 2005-12-21 10:25 ian
d-wx--x--x 2 tom Testgroup 75 2005-12-21 11:05 tom
ian@pinguino:~$ ls -a ~greg
. .. .bash_history .bash_profile .bashrc
ian@pinguino:~$ ls -a ~gretchen
ls: /home/gretchen: Permission denied
ian@pinguino:~$ ls -a ~tom
ls: /home/tom: Permission denied
```

ian@pinguino:~$ head -n 3 ~tom/.bashrc
 # ~/.bashrc: executed by bash(1) for non-login shells.
 # see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
for examples
The first character of a long listing describes the type of object (d for a directory). User greg's home directory has read and execute permission for member s of the Testgroup group, so user ian can list the directory. User gretchen's home directory has neither read nor execute permis sion, so user ian cannot access it.User tom's home has execute but not read permission, so user ian cannot list the content s, but can access objects within the directory if he knows they exist.

Changing permissions

Adding permissions

Suppose you create a "Hello world" shell script. When you first create the script, it will usually not be executable. Use the chmod command with the +x option to add the execute permissions as shown in Listing 5.

Listing 5. Creating an executable shell script

ian@pinguino:~$ echo 'echo "Hello world!"'>hello.sh
ian@pinguino:~$ ls -l hello.sh
-rw-r--r-- 1 ian ian 20 2005-12-22 12:57 hello.sh
ian@pinguino:~$ ./hello.sh
-bash: ./hello.sh: Permission denied
ian@pinguino:~$ chmod +x hello.sh
ian@pinguino:~$ ./hello.sh
Hello world!
ian@pinguino:~$ ls -l hello.sh
-rwxr-xr-x 1 ian ian 20 2005-12-22 12:57 hello.sh

You can use r to set the read permissions, and w to set the write permissions in a similar manner. In fact, you can use any combination of r, w, and x together. For example, using chmod +rwx would set all the read, write, and execute permis sions for a file. This form of chmod adds permissions that are not already set.

Being selective

You may have noticed in the above example that execute permission was set for the owner, group, and others. To be more selective, you may prefix the mode expression with u to set the permission for users, g to set it for groups, and o to set it for others. Listing 6 shows how to add user and group write and execute permissions to another copy of the shell script.

Listing 6. Selectively adding permissions

```
ian@pinguino:~$ echo 'echo "Hello world!"'>hello2.sh
ian@pinguino:~$ chmod ug+xw hello2.sh
ian@pinguino:~$ ls -l hello2.sh
-rwxrwxr-- 1 ian ian 20 2005-12-22 13:17 hello2.sh
```

Removing permissions

Sometimes you need to remove permissions rather than add them. Simply change the + to a - , and you remove any of the specified permissions that are set. Listing 7 shows how to remove all permissions for other users on the two shell scripts.

Listing 7. Removing permissions

```
ian@pinguino:~$ ls -l hello*
-rwxrwxr-- 1 ian ian 20 2005-12-22 13:17 hello2.sh
-rwxr-xr-x 1 ian ian 20 2005-12-22 12:57 hello.sh
ian@pinguino:~$ chmod o-xrw hello*
ian@pinguino:~$ ls -l hello*
-rwxrwx--- 1 ian ian 20 2005-12-22 13:17 hello2.sh
-rwxr-x--- 1 ian ian 20 2005-12-22 12:57 hello.sh
```

Note that you can change permissions on more than one file at a time. As with some other commands you can even use the - R (or - - recursive) option to operate recursively on directories and files.

Setting permissions

Now that you can add or remove permis sions, you may wonder how to set just a specific set of permissions. Do this using = instead of + or - . To set the permis sions on the above scripts so that other users have no access rights, you could use chmod o=0 hello*, instead of the command we used to remove permissions.

If you want to set different permissions for user, group, or other, you can separate different expressions by commas; for example, ug=rwx,o=rx, or you can use numeric permissions, which are described next.

Octal permissions

So far you have used symbols (ugoa and rxw) to specify permissions. There are three possible permissions in each group. You can also set permissions using octal number s instead of symbols. Permissions set in this way use up to four octal digits.

We will look at the first digit when we discuss attributes. The second digit defines user permissions, the third group permissions and the fourth other permis sions.

Each of these three digits is constructed by adding the desired permissions settings: read (4), write (2), and execute (1). In the example for hello.sh in Listing 45, the script was created with permissions - rw- r- - r- - , corresponding to octal 644. Setting execute permission for everyone changed the mode to 755.
Using numeric permis sions is very handy when you want to set all the permissions at once without giving the same permissions to each of the groups.

The umask

When a new file is created, the creation process specifies the permissions that the new file should have. Often, the mode requested is 0666, which makes the file readable and writable by anyone.
However, this permissive creation is affected by a umask value, which specifies what permissions a user does not want to grant automatically to newly created files or directories. The system uses the umask value to reduce the originally requested permissions. You can view your umask setting with the umask command, as shown below
Listing 13. Displaying octal umask

```
ian@pinguino:~$ umask
0022
```

Remember that the umask specifies which permissions should not be granted. On Linux systems, the umask normally defaults to 0022, which removes group and other write permission from new files.
Use the - S option to display the umask symbolically, in a form that shows which are the permissions that are allowed.
You can use the umask command to set a umask as well as display one. So, if you would like to keep your files more private and disallow all group or other access to newly created files, you would use a
umask value of 0077. Or set it symbolically using umask u=rwx,g=,o=, as illustrated below
Listing 14. Setting the umask

```
ian@pinguino:~$ umask
0022
ian@pinguino:~$ umask - S
u=rwx,g=rx,o=rx
ian@pinguino:~$ umask u=rwx,g=,o=
ian@pinguino:~$ umask
0077
ian@pinguino:~$ touch newfile
ian@pinguino:~$ ls - l newfile
- rw- - - - - - - 1 ian ian 0 2005- 12- 26 12:49 newfile
```

The next section shows you how to change the owner and group of an existing filesystem object.

Setting file owner and group

In this section, you learn about:
• Changing a file's group
• Default group for new files
• Changing the owner of a file
In the previous section you learned how every filesystem object has an owner and a group. In this section you learn how to change the owner or group of an existing file and how the default group for new files can be set.

File group

To change the group of a file, use the chgrp command with a group name and one or more filenames.
You may also use the group number if you prefer. An ordinary user must be a member of the group to which the file's group is being changed. The root user may change files to any group. Listing 15 shows an example.

Listing 15. Changing group ownership

```
ian@pinguino:~$ touch file1 file2
ian@pinguino:~$ ls - l file*
- rw- r- - r- - 1 ian ian 0 2005- 12- 26 14:09 file1
- rw- r- - r- - 1 ian ian 0 2005- 12- 26 14:09 file2
ian@pinguino:~$ chgrp Testgroup file1
ian@pinguino:~$ chgrp 1001 file2
ian@pinguino:~$ ls - l file*
- rw- r- - r- - 1 ian Testgroup 0 2005- 12- 26 14:09 file1
- rw- r- - r- - 1 ian Testgroup 0 2005- 12- 26 14:09 file2
```

As with many of the commands covered in this tutorial, chgrp has a - R option to allow changes to be applied recursively to all selected files and subdirectories.

Default group

In the previous section you learned how setting the sgid mode on a directory causes new files created in that directory to belong to the group of the directory rather than to the group of the user creating the file.
You may also use the newgrp command to temporarily change your primary group to another group of which you are a member. A new shell will be created, and when you

exit the shell, your previous group will be reinstated, as shown in Listing 16.

Listing 16. Using newgrp to temporarily change default group

```
ian@pinguino:~$ newgrp Testgroup
ian@pinguino:~$ groups
Testgroup adm dialout cdrom floppy audio dip video plugdev lpadmin scanner admin ian
ian@pinguino:~$ touch file3
ian@pinguino:~$ ls - l file3
- rw- r- - r- - 1 ian Testgroup 0 2005- 12- 26 14:34 file3
ian@pinguino:~$ exit
ian@pinguino:~$ groups
ian adm dialout cdrom floppy audio dip video plugdev lpadmin scanner admin Testgroup
```

File owner

The root user can change the ownership of a file using the chown command. In its simplest form, the syntax is like the chgrp command, except that a user name or numeric id is used instead of a group name or id. The file's group may be changed at the same time by adding a colon and a group name or id right after the user name or id. If only a colon is given, then the user's default group is used.

Naturally, the –R option will apply the change recursively. Listing 17 shows an example.

Listing 17. Using newgrp to temporarily change default group

```
root@pinguino:~# ls - l ~ian/ file4
- rw- r- - r- - 1 ian ian 0 2005- 12- 26 14:44 /home / ian / f ile4
root@pinguino:~# chown greg ~ian / f ile4
root@pinguino:~# ls - l ~ian/ file4
    -      rw- r- - r- - 1 greg ian 0 2005- 12- 26 14:44 /home / ian / f ile4
root@pinguino:~# chown tom: ~ian/ file4
root@pinguino:~# ls - l ~ian/ file4
- rw- r- - r- - 1 tom Testgroup 0 2005- 12- 26 14:44 /home / ian / f ile4
```

An older form of specifying both user and group used a dot instead of a colon. This is no longer recommended as it may cause confusion when names include a dot.