

15.1 Introduction

File ownership is critical for file security. Every file has a user owner and a group owner.

This chapter will focus on how to specify the user and group ownership of a file. In addition, the concept of file and directory permissions will be explored, including how to change the permissions on files and directories. Default permissions are the permissions given to files and directories when they are initially created.

15.2 Linux Essentials Exam Objectives

This chapter will cover the topics for the following Linux Essentials exam objectives:

Topic 5: Security and File Permissions (weight: 7)

- **5.3: Managing File Permissions and Ownership**
 - Weight: 2
 - Description: Understanding and manipulating file permissions and ownership settings.
 - Key Knowledge Areas:
 - File/directory permissions and owners
 - The following is a partial list of the used files, terms, and utilities:
 - ls -l
 - chmod, chown
 - Things that are nice to know
 - chgrp

15.3 File Ownership

By default, users will own the files that they create. While this ownership can be changed, this function requires administrative privileges. Although, most commands will usually show the user owner as a name, the operating system is actually associating the user ownership with the UID for that user name.

Every file also has a group owner. In the previous chapter on creating users and groups, the user's primary group was discussed. By default, the primary group of the user who creates the file will be the group owner of any new files. Users are allowed to change the group owner of a file to any group that they belong. Similar to user ownership, the association of a file with a group is not actually done internally by the operating system by name, but by the GID of the group.

Since ownership is actually determined by the UID and GID associated with a file, changing the UID of a user (or deleting the user) has the effect of making a file that was originally owned by that user have no real user owner. When there is no UID in the `/etc/passwd` file that matches the UID of the

owner of the file, then the UID (the number) will be displayed as the user owner of the file instead of the user name (which no longer exists). The same occurs for groups.

The `id` command can be useful for verifying which user account you are using and which groups you have available to use. By viewing the output of this command, you can see the user's identity information expressed both as a number and as a name.

The output of the `id` command displays the UID and user account name of the current user followed by the GID and group name of the primary group and the GIDs and group names of all group memberships:

```
[sysadmin@localhost ~]$ id
uid=500(sysadmin) gid=500(sysadmin) groups=500(sysadmin),10001(research),10002(development) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

The above example shows the user has a UID of 500 for the user account `sysadmin`. It shows the primary group for this user has a GID of 500 for the group `sysadmin`.

Because the user account and primary group account have the same numeric identifier and same name, this indicates that this user is in a User Private Group (UPG). In addition, the user belongs to two supplemental groups: the `research` group with a GID of 10001 and the `development` group with a GID of 10002.

When a file is created it will belong to the current user and their current primary group. If the user from the above example were to execute a command like `touch` to create a file and then view the file details, the output would look like the following:

```
[sysadmin@localhost ~]$ touch /tmp/filetest1
[sysadmin@localhost ~]$ ls -l /tmp/filetest1
-rw-rw-r--. 1 sysadmin sysadmin 0 Oct 21 10:18 /tmp/filetest1
```

The user owner of the file is `sysadmin` and the group owner `sysadmin`.

15.4 newgrp and groups Commands

If you know that the file you are about to create should belong to a group different from your current primary group, then you can use the `newgrp` command to change your current primary group.

As shown previously, the `id` command will list your identity information, including your group memberships. If you are only interested in knowing what groups you belong to, then you can execute the `groups` command:

```
[sysadmin@localhost ~]$ groups
sysadmin research development
```

The `groups` output may not be as detailed as the output of the `id` command, but if all you need to know is what groups you can switch to by using the `newgrp` command, then the `groups` command provides the information that you need. The `id` command output does show your current primary group, so it is useful for verifying that the `newgrp` command succeeded.

For example, if the "sysadmin" user was planning on having a file owned by the group "research", but that wasn't the user's primary group, then the user could use the `newgrp` command and then verify the correct primary group with the `id` command before creating the new file:

```
[sysadmin@localhost ~]$ id
uid=502(sysadmin) gid=503(sysadmin) groups=503(sysadmin),10001(research),10002(development) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[sysadmin@localhost ~]$ newgrp research
[sysadmin@localhost ~]$ id
uid=502(sysadmin) gid=10001(research) groups=503(sysadmin),10001(research),10002(development) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

From the output of the previous commands, you see initially the user's GID is 503 for the sysadmin user, then the `newgrp research` command is executed, and then the user's primary GID is 10001, the research group. After these commands were executed, if the user was to create another file and view its details, the new file would be owned by the research group:

```
[sysadmin@localhost ~]$ touch /tmp/filetest2
[sysadmin@localhost ~]$ ls -l /tmp/filetest2
-rw-r--r--. 1 sysadmin research 0 Oct 21 10:53 /tmp/filetest2
```

The `newgrp` command opens a new shell; as long as the user stays in that shell, the primary group won't change. To switch the primary group back to the original, the user could leave the new shell by running the `exit` command. For example:

```
[sysadmin@localhost ~]$ id
uid=502(sysadmin) gid=10001(research) groups=503(sysadmin),10001(research),10002(development) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[sysadmin@localhost ~]$ exit
exit
[sysadmin@localhost ~]$ id
uid=502(sysadmin) gid=503(sysadmin) groups=503(sysadmin),10001(research),10002(development) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Administrative privileges are required in order to permanently change the primary group of the user. The root user would execute the `usermod -g groupname username` command.

15.5 chgrp and stat Commands

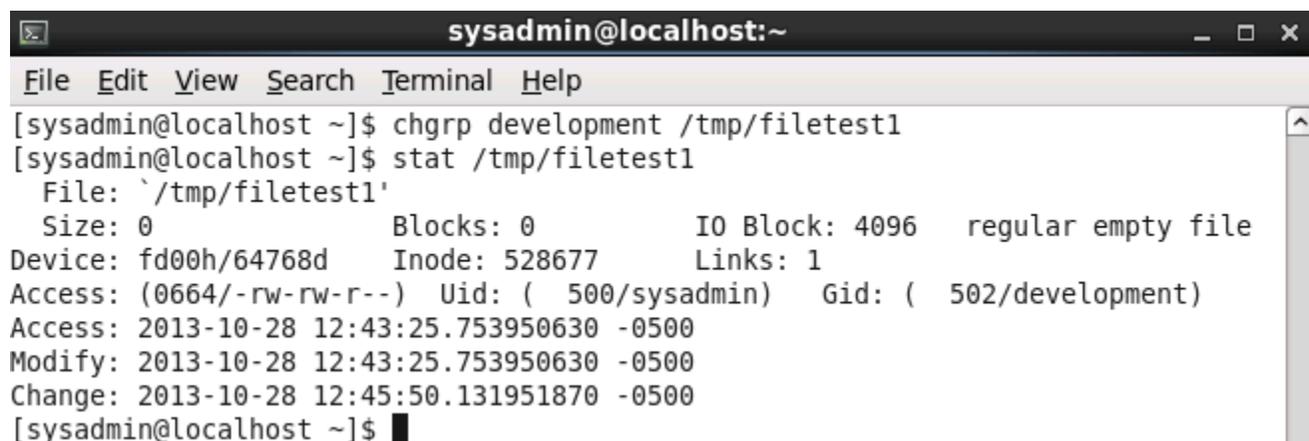
If you need to change the group ownership of an existing file, then you can use the `chgrp` command. As a user without administrative privileges, the `chgrp` command can only be used to change the group owner of the file to a group that the user is already a member. As the root user, the `chgrp` command can be used to change the group owner of any file to any group.

While you can view the ownership of a file with the `-l` option to the `ls` command, the system provides another command that is useful when viewing ownership and file permissions: the `stat` command. The `stat` command displays more detailed information about a file, including providing the group ownership both by group name and GID number:

```
[sysadmin@localhost ~]$ stat /tmp/filetest1
File: `/tmp/filetest1'
Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d  Inode: 31477       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 502/sysadmin)  Gid: ( 503/sysadmin)
Access: 2013-10-21 10:18:02.809118163 -0700
Modify: 2013-10-21 10:18:02.809118163 -0700
Change: 2013-10-21 10:18:02.809118163 -0700
```

The `stat` command will also be useful later in this chapter when file permissions are covered, as it provides more detail than the `ls -l` command provides.

The following graphic illustrates the `sysadmin` user changing the group ownership of a file that the user owns:



```
sysadmin@localhost:~
File Edit View Search Terminal Help
[sysadmin@localhost ~]$ chgrp development /tmp/filetest1
[sysadmin@localhost ~]$ stat /tmp/filetest1
File: `/tmp/filetest1'
Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d  Inode: 528677     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 500/sysadmin)  Gid: ( 502/development)
Access: 2013-10-28 12:43:25.753950630 -0500
Modify: 2013-10-28 12:43:25.753950630 -0500
Change: 2013-10-28 12:45:50.131951870 -0500
[sysadmin@localhost ~]$
```

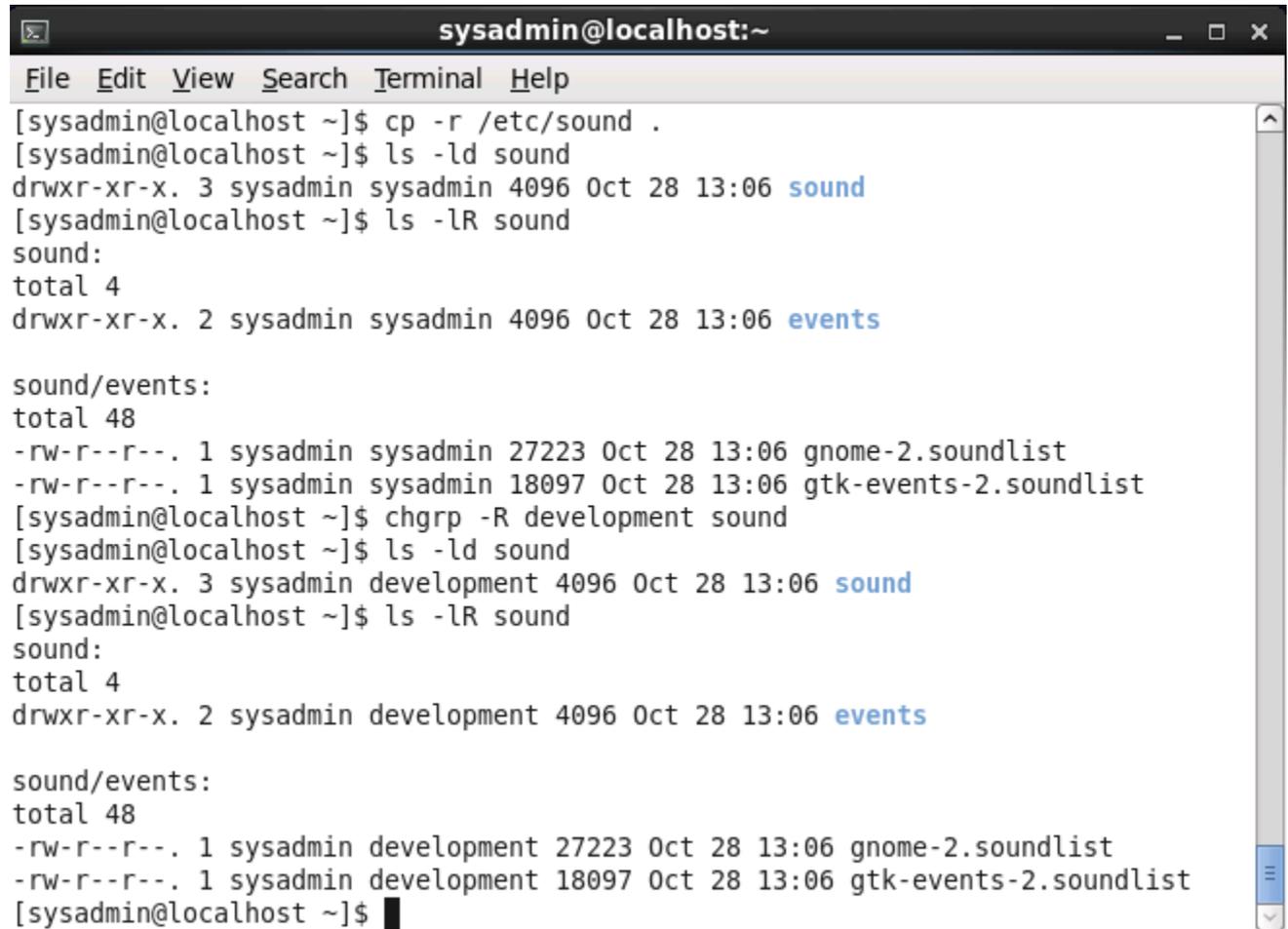
If a user attempts to modify the group ownership of a file that the user doesn't own, they will receive an error message:

```
[sysadmin@localhost ~]$ chgrp development /etc/passwd
```

```
chgrp: changing group of '/etc/passwd': Operation not permitted
```

Sometimes you want to be able to not only change the files in your current directory, but also the files in sub-directories. When executed with the `-R` (recursive) option, the `chgrp` command will operate not only on the current directory, but all directories that may be nested underneath the specified directory. The operation will also affect all files in the subdirectories, not just the directories themselves.

The following graphic illustrates the use of the `-R` option:



```
sysadmin@localhost:~  
File Edit View Search Terminal Help  
[sysadmin@localhost ~]$ cp -r /etc/sound .  
[sysadmin@localhost ~]$ ls -ld sound  
drwxr-xr-x. 3 sysadmin sysadmin 4096 Oct 28 13:06 sound  
[sysadmin@localhost ~]$ ls -lR sound  
sound:  
total 4  
drwxr-xr-x. 2 sysadmin sysadmin 4096 Oct 28 13:06 events  
  
sound/events:  
total 48  
-rw-r--r--. 1 sysadmin sysadmin 27223 Oct 28 13:06 gnome-2.soundlist  
-rw-r--r--. 1 sysadmin sysadmin 18097 Oct 28 13:06 gtk-events-2.soundlist  
[sysadmin@localhost ~]$ chgrp -R development sound  
[sysadmin@localhost ~]$ ls -ld sound  
drwxr-xr-x. 3 sysadmin development 4096 Oct 28 13:06 sound  
[sysadmin@localhost ~]$ ls -lR sound  
sound:  
total 4  
drwxr-xr-x. 2 sysadmin development 4096 Oct 28 13:06 events  
  
sound/events:  
total 48  
-rw-r--r--. 1 sysadmin development 27223 Oct 28 13:06 gnome-2.soundlist  
-rw-r--r--. 1 sysadmin development 18097 Oct 28 13:06 gtk-events-2.soundlist  
[sysadmin@localhost ~]$
```

15.6 chown Command

The `chown` command allows the root user to change the user ownership of files and directories. A regular user cannot use this command to change the user owner of a file, even to give the ownership of one of their own files to another user. However, the `chown` command also permits changing group ownership, which can be accomplished by either root or the owner of the file.

There are three different ways the `chown` command can be executed. The first method is used to change just the user owner of the file. For example, if the root user wanted to change the user ownership of the `abc.txt` file to the user "ted", then the following command could be executed:

```
[root@localhost ~]# chown ted abc.txt
```

The second method is to change both the user and the group; this also requires root privileges. To accomplish this, you separate the user and group by either a colon or a period character. For example:

```
[root@localhost ~]# chown user:group /path/to/file
[root@localhost ~]# chown user.group /path/to/file
```

If a user doesn't have root privileges, they can use the third method to change the group owner of a file just like the `chgrp` command. To use `chown` to only change the group ownership of the file, use a colon or a period as a prefix to the group name:

```
[root@localhost ~]# chown :group /path/to/file
[root@localhost ~]# chown .group /path/to/file
```

15.7 Permissions

When you execute the `ls -l` command, the resulting output displays ten characters at the beginning of each line, which indicate the type of file and the permissions of the file:

- The first character indicates the file type.
- Characters 2-4 indicate the permissions for the user that owns the file.
- Characters 5-7 indicate the permissions for the group that owns the file.
- Characters 8-10 indicate the permissions for "others" or what is sometimes referred to as the world's permissions. This would include all users who are not the file owner or a member of the file's group.

For example, consider the output of the following command:

```
[root@localhost ~]# ls -l /etc/passwd
-rw-r--r--. 1 root root 4135 May 27 21:08 /etc/passwd
```

Based on the previous command output, the first ten characters could be described by the following table:

File	User owner			Group owner			Other or world		
type	read	write	execute	read	write	execute	read	write	execute
-	r	w	-	r	-	-	r	-	-

The following table describes the possible values for the file type. Recall that these types were covered in a previous chapter:

Character	Type of the File
-	A regular file which may be empty, contain text or binary data.
d	A directory file which contains the names of other files and links to them.
l	A symbolic link is a file name that refers (points) to another file.
b	A block file is one that relates to a block hardware device where data is read in blocks of data.
c	A character file is one that relates to a character hardware device where data is read one byte at a time.
p	A pipe file works similar to the pipe symbol, allowing for the output of one process to communicate to another process through the pipe file, where the output of the one process is used as input for the other process.
s	A socket file allows two processes to communicate, where both processes are allowed to either send or receive data.

Although all the file types are listed in the table above, most likely you will never encounter anything but regular, directory and link files unless you explore the `/dev` directory.

The characters in the permissions part of the output have the following meanings:

- **r** stands for **read** permission
- **w** stands for **write** permission
- **x** stands for **execute** permission

The permissions set on these files determine the level of access that a user will have on the file. When a user runs a program and the program accesses a file, then the permissions are checked to determine if the user has the correct access rights to the file.

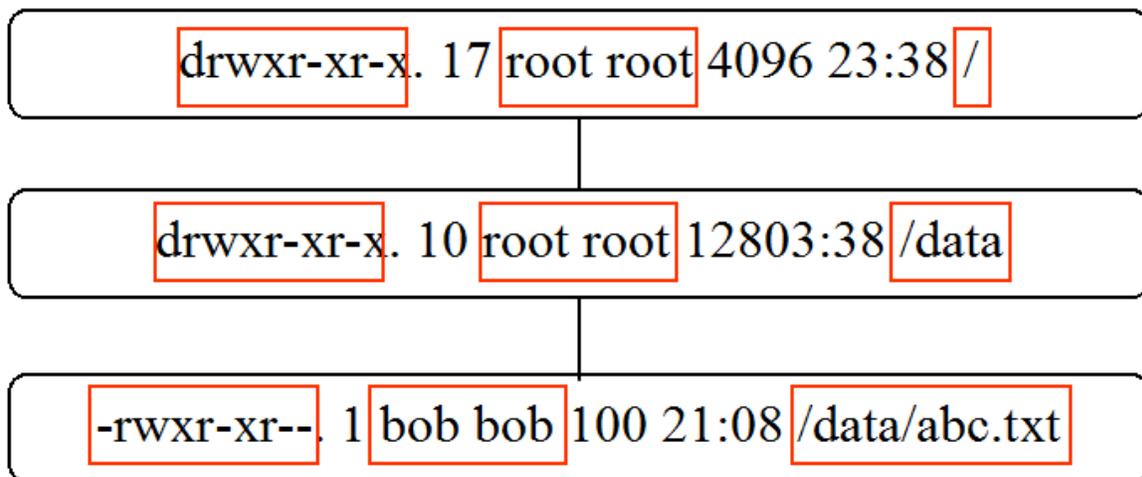
The permissions themselves are deceptively simple and have a different meaning depending on whether they are applied to a file or a directory:

Permission	Meaning on a file	Meaning on a directory
r	The process can read the contents of the file, meaning the contents can be viewed and copied.	File names in the directory can be listed, but other details are not available.
w	The file can be written to by the process, so changes to a file can be saved. Note that w permission really requires r permission on the file to work correctly.	Files can be added to or removed from the directory. Note that w permission requires x permission on the directory to work correctly.
x	The file can be executed or run as a process.	The user can use the <code>cd</code> command to "get into" the directory and use the directory in a pathname to access files and, potentially, subdirectories under this directory.

15.7.1 Understanding Permissions

While the chart on the previous page can be a handy reference, by itself it doesn't provide a clear description of how permissions work. To better understand how permissions work, consider the following scenarios.

To understand these scenarios, you should first understand the following diagram:



The important information is highlighted. The first box represents the `/` directory, with a user owner of "root", a group owner of "root" and permissions of `drwxr-xr-x`. The second box represents the `/data` directory, a directory that is under the `/` directory. The third box represents the `abc.txt` file, which is stored in the `/data` directory.

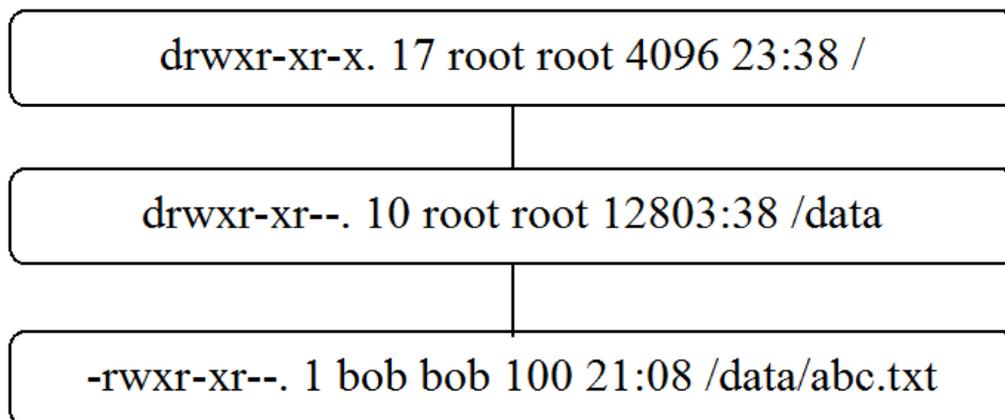
You should also understand that if you are the owner of the file/directory, then only the user owner permissions are used to determine access to that file/directory.

If you are not the owner, but are a member of the group that owns the file/directory, then only group owner permissions are used to determine access to that file/directory.

If you aren't the owner and not a member of the file/directory group, then your permissions would be "others".

15.7.1.1 Scenario #1 - The importance of Directory Access

Question: Based on the following diagram, what access would the user "bob" have on the file `abc.txt`?



Answer: None

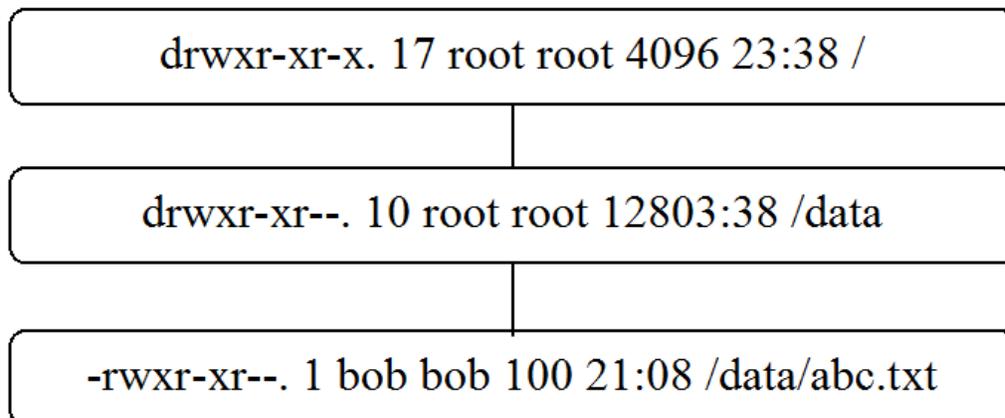
Explanation: Initially it would appear that the user "bob" can view the contents of the `abc.txt` file as well as copy the file, modify its contents and run it like a program. This erroneous conclusion would be the result of looking solely at the file's permissions (`rwX` for the user "bob" in this case).

However, in order to do anything with the file, the user must first "get into" the `/data` directory. The permissions for "bob" for the `/data` directory are the permissions for "others" (`r--`), which means "bob" can't even use the `cd` command to get into the directory. If the execute permission (`--x`) was set for the directory, then the user "bob" would be able to "get into" the directory, meaning the permissions of the file itself would apply.

Lesson learned: The permissions of all parent directories must be considered before considering the permissions on a specific file.

15.7.1.2 Scenario #2 - Viewing Directory Contents

Question: Based on the following diagram, who can use the `ls` command to display the contents of the `/data` directory (`ls /data`)?



Answer: All users

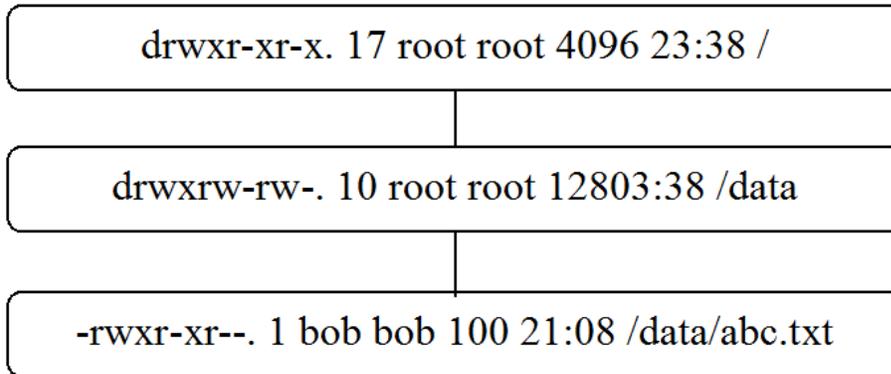
Explanation: All that is required to be able to view a directory's contents is `r` permission on the directory (and the ability to access the parent directories). The `x` permission for all users in the `0 /` directory means all users can use `/` as part of a path, so everyone is able to get through the `/` directory to get to the `/data` directory. The `r` permission for all users in the `/data` directory means all users can use the `ls` command to view the contents.

However, note that in order to see file details (`ls -l`) would also require `x` permission on the directory. So while the root user and members of the root group have this access on the `/data` directory, no other users would be able to execute `ls -l /data`.

Lesson learned: The `r` permission allows a user to view a listing of the directory.

15.7.1.3 Scenario #3 - Deleting Directory Contents

Question: Based on the following diagram, who can delete the `/data/abc.txt` file?



Answer: Only the root user

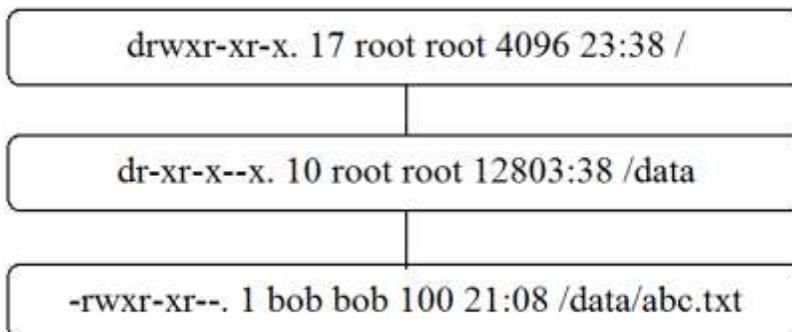
Explanation: A user needs no permissions at all on the file itself to delete a file. The **w** permission on the directory that the file is stored in is required to delete a file in a directory. Based on that, it would seem that all users could delete the `/data/abc.txt` file, since everyone has **w** permission on the file.

However, to delete a file, you must also be able to "get into" the directory. Since only the root user has **x** permission on the `/data` directory, only root can "get into" that directory in order to delete files in this directory.

Lesson learned: The **w** permission allows a user to delete files from a directory, but only if the user also has **x** permission on the directory.

15.7.1.4 Scenario #4 - Accessing the Contents of a Directory

Question: True or false: Based on the following diagram, the user "bob" can successfully execute the following command: `more /data/abc.txt`?



Answer: True

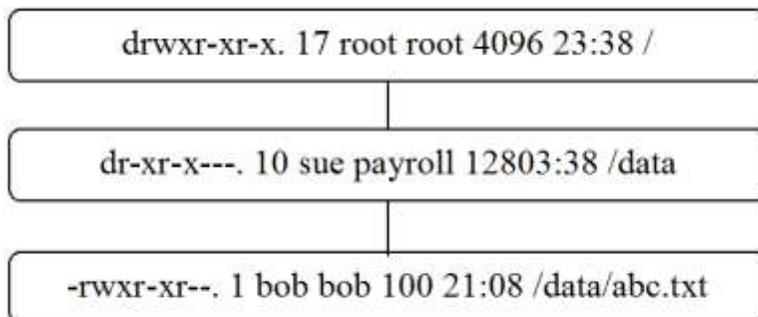
Explanation: As previously mentioned, in order to access a file, the user must have access to the directory. The access to the directory only requires **x** permission; even though **r** permission would be useful to list files in a directory, it isn't required to "get into" the directory and access files within the directory.

When the command `more /data/abc.txt` is executed, the following permissions are checked: **x** permission on the `/` directory, **x** permission on the `data` directory and **r** permission on the `abc.txt` file. Since the user "bob" has all of these permissions, the command executes successfully.

Lesson learned: The **x** permission is required to "get into" a directory, but the **r** permission on the directory is not necessary unless you want to list the directory's contents.

15.7.1.5 Scenario #5 - the Complexity of Users and Groups

Question: True or false: Based on the following diagram, the user "bob" can successfully execute the following command: `more /data/abc.txt` (note that the `/data` directory has different user and group owners than previous examples)?



Answer: Not enough information to determine.

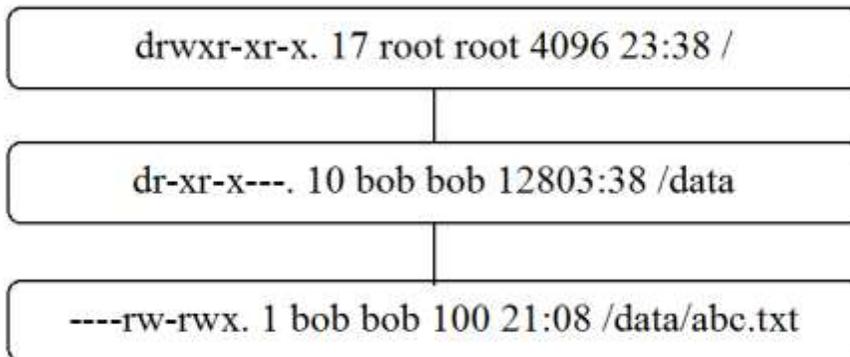
Explanation: In order to access the `/data/abc.txt` file, the user "bob" needs to be able to "get into" the `/data` directory. This requires **x** permission, which "bob" may or may not have, depending on if he is a member of the "payroll" group.

If "bob" is a member of the "payroll" group, then his permissions on the `/data` directory are **r-x** and the command `more /data/abc.txt` will execute successfully ("bob" also needs **x** on `/` and **r** on `abc.txt`, which he already has). If he isn't a member of the "payroll" group, his permissions on the `/data` directory are **---** and the `more` command would fail.

Lesson learned: You must look at each file and directory permissions separately and be aware of which groups the user account belongs to.

15.7.1.6 Scenario #6 - Permission Priority

Question: True or false: Based on the following diagram, the user "bob" can successfully execute the following command: `more /data/abc.txt` (note that the `/data` directory has different user and group owners than previous example)?



Answer: False

Explanation: Recall that if you are the owner of a file, then the only permissions that are checked are the user owner's permissions. In this case, that would be `---` for "bob" on the `/data/abc.txt` file.

In this case, members of the "bob" group and "others" have more permissions on the file than "bob" has.

Lesson learned: Don't provide permissions to the group owner and "others" without applying at least the same level of access as the owner of the file.

15.7.2 Using the `chmod` Command - Symbolic Method

The `chmod` (change mode) command is used to change permissions on a directory. There are two techniques that can be used with this command: symbolic and numeric. Both techniques use the following basic syntax: `chmod new_permission file_name`.

To change a file's permissions, you either need to own the file or log in as the root user.

If you want to modify some of the current permissions, the symbolic method will likely be easier to use. With this method, you specify which permissions you want to change on the file and the other permissions remain as they are.

When specifying the "new_permission", you start by using one of the following characters to indicate which permission set you want to change:

- `u` = change user owner's permissions
- `g` = change group owner's permissions
- `o` = change "others" permissions

- a = apply changes to all permission sets (user owner, group owner and "others")

Then you specify a **+** to add a permission or a **-** to remove a permission. Lastly, you specify **r** for "read", **w** for "write" and **x** for "execute".

For example, to give the user owner read permission on a file named `abc.txt`, you could use the following command:

```
[root@localhost ~]# chmod u+r abc.txt
```

Only the user owner's permission was changed. All other permissions remained as they were prior to the execution of the `chmod` command.

You can combine values to make multiple changes to the file's permissions. For example, consider the following command which will add read permission to the user owner and group owner while removing write permission for "others":

```
[root@localhost ~]# chmod ug+r,o-w abc.txt
```

Lastly, you could use the **=** character instead of **-** or **+** to specify exactly the permissions you want for a permission set:

```
[root@localhost ~]# chmod u=r-x abc.txt
```

15.7.3 Using the chmod Command - Numeric Method

The numeric method (also called octal method) is useful when you want to change many permissions on a file. It is based on the octal numbering system in which each permission type is assigned a numeric value:

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 for a single permission set. For example:

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

When the numeric 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 numeric method is better for changes that are more drastic.

For example, to set the permissions of a file named `abc.txt` to be **rwxr-xr--** you could use the following command:

```
[root@localhost ~]# chmod 754 abc.txt
```

15.8 Revisiting the stat Command

Recall the `stat` command discussed earlier in this chapter. This command provided more detailed information than the `ls -l` command provided.

Because of this, you may consider using the `stat` command instead of the `ls -l` command when viewing permissions on a file. One big advantage of the `stat` command is that it shows permissions both symbolically and by numeric method, as demonstrated below:

```
[sysadmin@localhost ~]$ stat /tmp/filetest1
File: `/tmp/filetest1'
```

```
Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d  Inode: 31477       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 502/sysadmin)  Gid: ( 503/sysadmin)
Access: 2013-10-21 10:18:02.809118163 -0700
Modify: 2013-10-21 10:18:02.809118163 -0700
Change: 2013-10-21 10:18:02.809118163 -0700
```

15.9 umask

The `umask` command is a feature that is used to determine default permissions that are set when a file or directory is created. Default permissions are determined when the *umask value* is subtracted from the maximum allowable default permissions. The maximum default permissions are different for files and directories:

file	rw-rw-rw-
directories	rwxrwxrwx

The permissions that are initially set on a file when it is created cannot exceed **rw-rw-rw-**. To have the execute permission set on a file, you first need to create the file and then change the permissions.

The `umask` command can be used to display the current umask value:

```
[sysadmin@localhost ~]$ umask
0002
```

A breakdown of the output:

- The first 0 indicates that the umask is given as an octal number.
- The second 0 indicates which permissions to subtract from the default user owner's permissions.
- The third 0 indicates which permissions to subtract from the default group owner's permissions.
- The last number (2) indicates which permissions to subtract from the default other's permissions.

Note that different users may have different umasks. Typically the root user will have a more restrictive umask than normal user accounts:

```
[root@localhost ~]# umask
0022
```

15.9.1 How umask Works

To understand how umask works, assume that the umask is set to 027 and consider the following :

File Default	666
Umask	-027
Result	640

The 027 umask means that, by default, new files would receive **640** or **rw-r-----** permissions, as demonstrated below:

```
[sysadmin@localhost ~]$ umask 027
[sysadmin@localhost ~]$ touch sample
[sysadmin@localhost ~]$ ls -l sample
-rw-r-----. 1 sysadmin sysadmin 0 Oct 28 20:14 sample
```

Because the default permissions for directories are different than for files, a umask of 027 would result in different initial permissions on new directories:

File Default	777
Umask	-027
Result	750

The 027 umask means that, by default directories files would receive **750** or **rwxr-x---** permissions, as demonstrated below:

```
[sysadmin@localhost ~]$ umask 027
[sysadmin@localhost ~]$ mkdir test-dir
[sysadmin@localhost ~]$ ls -ld test-dir
```

```
drwxr-x---. 1 sysadmin sysadmin 4096 Oct 28 20:25 test-dir
```

The new umask will only be applied to file and directories created during that session. When a new shell is started the default umask will again be in effect.

Permanently changing a user's umask requires modifying the `.bashrc` file located in that user's home directory.