# ACCESSING LINUX FILE SYSTEMS

## Objectives

- List the file system permissions on files and directories, and interpret the effect of those permissions on access by users and groups.
- Change the permissions and ownership of files using command-line tools.
- Control the default permissions of new files created by users, explain the effect of special permissions, and use special permissions and default permissions to set the group owner of files created in a particular directory.

# INTERPRETING LINUX FILE SYSTEMS PERMISSIONS

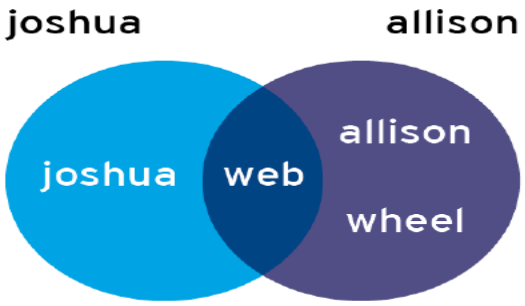
## 1. LINUX FILE SYSTEM PERMISSIONS

File permissions control access to files. The Linux file permissions system is simple but flexible, which makes it easy to understand and apply.

Files have three categories of user to which permissions apply. The file is owned by a user, normally the one who created the file. The file is also owned by a single group, usually the primary group of the user who created the file, but this can be changed. Different permissions can be set for the owning user, the owning group, and for all other users on the system that are not the user or a member of the owning group.

The most specific permissions take precedence. User permissions override group permissions, which override other permissions.

In Figure below, joshua is a member of the groups joshua and web, and allison is a member of allison, wheel, and web. When joshua and allison need to collaborate, the files should be associated with the group web and group permissions should allow the desired access.




Three categories of permissions apply: read, write, and execute. The following table explains how these permissions affect access to files and directories.

**Effects of Permissions on Files and Directories**

| PERMISSION | EFFECT ON FILES | EFFECT ON DIRECTORIES |
|---|---|---|
| **r(read)** | Content of the files can be read | Contents of the directory can be listed |
| **w(write)** | Content of the files can be changed | Any file in the directory can be created or deleted |
| **x (execute)** | Files can be executed as commands | Contents of the directory can be accessed. |

Note that users normally have both read and execute permissions on read-only directories so that they can list the directory and have full read-only access to its contents. If a user only has read access on a directory, the names of the files in it can be listed, but no other information, including permissions or time stamps, are available, nor can they be accessed. If a user only has execute access on a directory, they cannot list the names of the files in the directory, but if they already know the name of a file that they have permission to read, then they can access the contents of that file by explicitly specifying the file name.

A file may be removed by anyone who has write permission to the directory in which the file resides, regardless of the ownership or permissions on the file itself. This can be overridden with a special permission, the sticky bit.

## 2. VIEWING FILE AND DIRECTORY PERMISSIONS AND OWNERSHIP

The -l option of the ls command shows more detailed information about file permissions and ownership:

[student@localhost~]$ **ls -l test**

-rw-rw-r--. 1 student student 0 Feb 8 17:36 test

You can use the -d option to to show detailed information about a directory itself, and [student@locahost ~]$ **ls -ld /home**

drwxr-xr-x. 5 root root 4096 Jan 31 22:00 /homenot its contents.

The first character of the long listing is the file type. You interpret it like this:

- - is a regular file.
- d is a directory.
- l is a soft link.
- Other characters represent hardware devices (**b** and **c**) or other special-purpose files (**p** and **s**).

The next nine characters are the file permissions. These are in three sets of three characters: permissions that apply to the user that owns the file, the group that owns the file, and all other users. If the set shows **rwx**, that category has all three permissions, read, write, and execute. If a letter has been replaced by -, then that category does not have that permission.

After the link count, the first name specifies the user that owns the file, and the second name the group that owns the file.

So in the example above, the permissions for user student are specified by the first set of three characters. User student has read and write on **test**, but not execute.

Group student is specified by the second set of three characters: it also has read and write on **test**, but not execute.

Any other user's permissions are specified by the third set of three characters: they only have read permission on **test**.

The most specific set of permissions apply. So if user student has different permissions than group student, and user student is also a member of that group, then the user permissions will be the ones that apply.

## 3. EXAMPLES OF PERMISSION EFFECTS

The following examples will help illustrate how file permissions interact. For these examples, we have four users with the following group memberships:

| USER | GROUP MEMBERSHIPS |
|---|---|
| operator1 | operator1,consultant1 |
| database1 | database1,counsultant1 |
| database2 | database2,operator2 |
| contractor1 | contractor1, |

Those users will be working with files in the dir directory. This is a long listing of the files in that directory:

[database1@locahost dir]$ **ls -la**

total 24

drwxrwxr-x. 2 database1 consultant1 4096 Apr 4 10:23

drwxr-xr-x. 10 root root 4096 Apr 1 17:34 ..

-rw-rw-r--. 1 operator1 operator1 1024 Apr 4 11:02 lfile1

-rw-r--rw-. 1 operator1 consultant1 3144 Apr 4 11:02 lfile2

-rw-rw-r--. 1 database1 consultant1 10234 Apr 4 10:14 rfile1

-rw-r-----. 1 database1 consultant1 2048 Apr 4 10:18 rfile2

The **-a** option shows the permissions of hidden files, including the special files used to represent the directory and its parent. In this example, . reflects the permissions of dir itself, and .. the permissions of its parent directory.

What are the permissions of **rfile1**? The user that owns the file (database1) has read and write but not execute. The group that owns the file (consultant1) has read and write but not execute. All other users have read but not write or execute.

The following table explores some of the effects of this set of permissions for these users:

| EFFECT | WHY IS THIS TRUE? |
|---|---|
| The user `operator1` can change the contents of **`rfile1`**. | User `operator1` is a member of the `consultant1` group, and that group has both read and write permissions on **`rfile1`**. |
| The user `database1` can view and modify the contents of **`rfile2`**. | User `database1` owns the file and has both read and write access to **`rfile2`**. |
| The user `operator1` can view but not modify the contents of **`rfile2`** (without deleting it and recreating it). | User `operator1` is a member of the `consultant1` group, and that group only has read access to **`rfile2`**. |
| The users `database2` and `contractor1` do not have any access to the contents of **`rfile2`**. | **`other`** permissions apply to users `database2` and `contractor1`, and those permissions do not include read or write permission. |
| `operator1` is the only user who can change the contents of **`lfile1`** (without deleting it and recreating it). | User and group `operator1` have write permission on the file, other users do not. But the only member of group `operator1` is user `operator1`. |
| The user `database2` can change the contents of **`lfile2`**. | User `database2` is not the user that owns the file and is not in group `consultant1`, so **`other`** permissions apply. Those grant write permission. |
| The user `database1` can view the contents of **`lfile2`**, but cannot modify the contents of **`lfile2`** (without deleting it and recreating it). | User `database1` is a member of the group `consultant1`, and that group only has read permissions on **`lfile2`**. Even though **`other`** has write permission, the group permissions take precedence. |
| The user `database1` can delete **`lfile1`** and **`lfile2`**. | User `database1` has write permissions on the directory containing both files (shown by `.`), and therefore can delete any file in that directory. This is true even if `database1` does not have write permission on the file itself. |

# MANAGING FILE SYSTEM PERMISSIONS FROM THE COMMAND LINE

## 1. CHANGING FILE AND DIRECTORY PERMISSIONS

The command used to change permissions from the command line is **chmod**, which means "change mode" (permissions are also called the mode of a file). The **chmod** command takes a permission instruction followed by a list of files or directories to change. The permission instruction can be issued either symbolically (the symbolic method) or numerically (the numeric method).

### Changing Permissions with the Symbolic Method

**chmod WhoWhatWhich file|directory**

- Who is u, g, o, a (for user, group, other, all)
- What is +,-,= (for add, remove, set exactly)
- Which is r,w,x (for read, write, execute)

The symbolic method of changing file permissions uses letters to represent the different groups of permissions: **u** for user, **g** for group, **o** for other, and **a** for all.

With the symbolic method, it is not necessary to set a complete new group of permissions. Instead, you can change one or more of the existing permissions. Use **+** or **-** to add or remove permissions, respectively, or use = to replace the entire set for a group of permissions.

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

**Examples:**

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

  [student@localhost ~]$ chmod go-rw file1

- Add execute permission for everyone on file2:

  [student@localhost ~]$ chmod a+x file2

## Changing Permissions with the Numeric Method

In the example below the # character represents a digit.

**chmod ### file|directory**

- Each digit represents permissions for an access level: user, group, other.
- The digit is calculated by adding together numbers for each permission you want to add, 4 for read, 2 for write, and 1 for execute.

Using the numeric method, permissions are represented by a 3-digit (or 4-digit, when setting advanced permissions) octal number. A single octal digit can represent any single value from 0-7.

In the 3-digit octal (numeric) representation of permissions, each digit stands for one access level, from left to right: user, group, and other. To determine each digit:

1. Start with 0.

2. If the read permission should be present for this access level, add 4.

3. If the write permission should be present, add 2.

4. If the execute permission should be present, add 1.

Examine the permissions **-rwxr-x---**. For the user, **rwx** is calculated as 4+2+1=7. For the group, **r-x** is calculated as 4+0+1=5, and for other users, --- is represented with 0. Putting these three together, the numeric representation of those permissions is 750.

This calculation can also be performed in the opposite direction. Look at the permissions 640. For the user permissions, 6 represents read (4) and write (2), which displays as rw-. For the group part, 4 only includes read (4) and displays as r--. The 0 for other provides no permissions (---) and the final set of symbolic permissions for this file is -rw-r-----.

Experienced administrators often use numeric permissions because they are shorter to type and pronounce, while still giving full control over all permissions.

**Examples**

- Set read and write permissions for user, read permission for group and other, on samplefile:

  [student@localhost ~]$ **chmod 644 samplefile**

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

  [student@localhost ~]$ **chmod 750 sampledir**

## 2. CHANGING FILE AND DIRECTORY USER OR GROUP OWNERSHIP

A newly created file is owned by the user who creates that file. By default, new files have a group ownership that is the primary group of the user creating the file. In Red Hat Enterprise Linux, a user's primary group is usually a private group with only that user as a member. To grant access to a file based on group membership, the group that owns the file may need to be changed.

Only root can change the user that owns a file. Group ownership, however, can be set by root or by the file's owner. root can grant file ownership to any group, but regular users can make a group the owner of a file only if they are a member of that group.

File ownership can be changed with the **chown** (change owner) command. For example, to grant ownership of the test_file file to the student user, use the following command:

[student@localhost ~]# **chown student test_file**

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

[student@localhost ~]# **chown -R student test_dir**

The **chown** command can also be used to change group ownership of a file by preceding the group name with a colon (:). For example, the following command changes the group test_dir to admins:

[student@locahost ~]# **chown :admins test_dir**

The chown command can also be used to change both owner and group at the same time by using the owner:group syntax. For example, to change the ownership of test_dir to visitor and the group to guests, use the following command:

[student@localhost ~]# **chown visitor:guests test_dir**

Instead of using chown, some users change the group ownership by using the chgrp command. This command works just like chown, except that it is only used to change group ownership and the colon (:) before the group name is not required.

# MANAGING DEFAULT PERMISSIONS AND FILE ACCESS

## 1. SPECIAL PERMISSIONS

Special permissions constitute a fourth permission type in addition to the basic user, group, and other types. As the name implies, these permissions provide additional access-related features over and above what the basic permission types allow. This section details the impact of special permissions, summarized in the table below.

**Effects of Special Permissions on Files and Directories**

| SPECIAL PERMISSION | EFFECT ON FILES | EFFECT ON DIRECTORIES |
|---|---|---|
| **u+s** (suid) | File executes as the user that owns the file, not the user that ran the file. | No effect. |
| **g+s** (sgid) | File executes as the group that owns the file. | Files newly created in the directory have their group owner set to match the group owner of the directory. |
| **o+t** (sticky) | No effect. | Users with write access to the directory can only remove files that they own; they cannot remove or force saves to files owned by other users. |

The **setuid** permission on an executable file means that commands run as the user owning the file, not as the user that ran the command. One example is the passwd command:

[student@localhost ~]$ **ls -l /usr/bin/passwd**

-rwsr-xr-x. 1 root root 35504 Jul 16 2010 /usr/bin/passwd

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

The special permission **setgid** on a directory means that files created in the directory inherit their group ownership from the directory, rather than inheriting it from the creating user. This is commonly used on group collaborative directories to automatically change a file from

the default private group to the shared group, or if files in a directory should be always owned by a specific group. An example of this is the **/run/log/journal** directory:

[student@localhost ~]$ **ls -ld /run/log/journal**

drwxr-sr-x. 3 root systemd-journal 60 May 18 09:15 /run/log/journal

If setgid is set on an executable file, commands run as the group that owns that file, not as the user that ran the command, in a similar way to setuid works. One example is the locate command:

[student@localhost ~]$ **ls -ld /usr/bin/locate**

-rwx--s--x. 1 root slocate 47128 Aug 12 17:17 /usr/bin/locate

In a long listing, you can identify the setgid permissions by a lowercase **s** where you would normally expect the **x** (group execute permissions) to be. If the group does not have execute permissions, this is replaced by an uppercase **S**.

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

[student@localhost ~]$ **ls -ld /tmp**

drwxrwxrwt. 39 root root 4096 Feb 8 20:52 /tmp

In a long listing, you can identify the sticky permissions by a lowercase t where you would normally expect the **x** (other execute permissions) to be. If other does not have execute permissions, this is replaced by an uppercase **T**.

**Setting Special Permissions**

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

**Examples**

Add the setgid bit on directory:

[student@localhost ~]# **chmod g+s directory**

Set the setgid bit and add read/write/execute permissions for user and group, with no access for others, on directory:

[student@localhost ~]# **chmod 2770 directory**

## DEFAULT PERMISSIONS

When you create a new file or directory, it is assigned initial permissions. There are two things that affect these initial permissions. The first is whether you are creating a regular file or a directory. The second is the current umask.

If you create a new directory, the operating system starts by assigning it octal permissions 0777 (**drwxrwxrwx**). If you create a new regular file, the operating system assignes it octal permissions 0666 (**-rw-rw-rw-**). You always have to explicitly add execute permission to a regular file. This makes it harder for an attacker to compromise a network service so that it creates a new file and immediately executes it as a program.

However, the shell session will  also set a umask to further restrict the permissions that are initially set. This is an octal bitmask used to clear the permissions of new files and directories created by a process. If a bit is set in the umask, then the corresponding permission is cleared on new files. For example, the umask 0002 clears the write bit for other users. The leading zeros indicate the special, user, and group permissions are not cleared. A umask of 0077 clears all the group and other permissions of newly created files.

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

[student@localhost ~]$ **umask**

0002

Use the **umask** command with a single numeric argument to change the umask of the current shell. The numeric argument should be an octal value corresponding to the new umask value. You can omit any leading zeros in the umask.

The system's default umask values for Bash shell users are defined in the **/etc/profile** and **/etc/bashrc** files. Users can override the system defaults in the **.bash_profile** and **.bashrc** files in their home directories.

**umask Example**

The following example explains how the umask affects the permissions of files and directories. Look at the default umask permissions for both files and directories in the current shell. The owner and group both have read and write permission on files, and other is set to read. The owner and group both have read, write, and execute permissions on directories. The only permission for other is read.

[student@localhost ~]$ **umask**

0002

[student@localhost ~]$ touch default

[student@localhost ~]$ ls -l default.txt

-rw-rw-r--. 1 user user 0 May 9 01:54 default.txt

[student@localhost ~]$ mkdir default

[student@localhost ~]$ ls -ld default

drwxrwxr-x. 2 user user 0 May 9 01:54 default

By setting the umask value to 0, the file permissions for other change from read to read and write. The directory permissions for other changes from read and execute to read, write, and execute.

[student@localhost ~]$ **umask 0**

[student@localhost ~]$ **touch zero.txt**

[student@localhost ~]$ **ls -l zero.txt**

-rw-rw-rw-. 1 user user 0 May 9 01:54 zero.txt

[student@localhost ~]$ **mkdir zero**

[student@localhost ~]$ **ls -ld zero**

drwxrwxrwx. 2 user user 0 May 9 01:54 zero

To mask all file and directory permissions for other, set the umask value to 007.

[user@host ~]$ **umask 007**

[user@host ~]$ **touch seven.txt**

[user@host ~]$ **ls -l seven.txt**

-rw-rw----. 1 user user 0 May 9 01:55 seven.txt

[user@host ~]$ **mkdir seven**

[user@host ~]$ **ls -ld seven**

drwxrwx---. 2 user user 0 May 9 01:54 seven

A umask of 027 ensures that new files have read and write permissions for user and read permission for group. New directories have read and write access for group and no permissions for

other.

[student@localhost ~]$ **umask 027**

[student@localhost ~]$ **touch two-seven.txt**

[student@localhost ~]$ **ls -l two-seven.txt**

-rw-r-----. 1 user user 0 May 9 01:55 two-seven.txt

[student@localhost ~]$ **mkdir two-seven**

[student@localhost ~]$ **ls -ld two-seven**

drwxr-x---. 2 user user 0 May 9 01:54 two-seven

The default umask for users is set by the shell startup scripts. By default, if your account's UID is 200 or more and your username and primary group name are the same, you will be assigned a umask of 002. Otherwise, your umask will be 022.

As root, you can change this by adding a shell startup script named **/etc/profile.d/localumask.sh** that looks something like the output in this example:

[student@localhost ~]# **cat /etc/profile.d/local-umask.sh**

```
# Overrides default umask configuration
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 007
else
    umask 022
fi
```

The preceding example will set the umask to 007 for users with a UID greater than 199 and with a username and primary group name that match, and to 022 for everyone else. If you just wanted to set the umask for everyone to 022, you could create that file with just the following content:

```
# Overrides default umask configuration
umask 022
```

To ensure that global umask changes take effect you must log out of the shell and log back in. Until that time the umask configured in the current shell is still in effect.

# Tasks: Managing File System Permissions from the Command Line

1. use the **ssh** command to log in to localhost as the student user.
2. Switch to the root user using redhat as the password.

   [student@localhost ~]$ su -

   Password: redhat

   [root@localhost ~]#
3. Use the mkdir command to create the /home/consultants directory.

   [root@localhost ~]# **mkdir /home/consultants**
4. Use the **chown** command to change the group ownership of the consultants directory to consultants.

   [root@localhost ~]# **chown :consultants /home/consultants**
5. Ensure that the permissions of the group allow group members to create and delete files. The permissions should forbid others from accessing the files.

   a. Use the ls command to confirm that the permissions of the group allow group members to create and delete files in the /home/consultants directory.

      [root@localhost ~]# **ls -ld /home/consultants**

      drwxr-xr-x. 2 root consultants 6 Feb 1 12:08 /home/consultants

   Note that the consultants group currently does not have write permission.

   b. Use the **chmod** command to add write permission to the consultants group.

      [root@localhost ~]# **chmod g+w /home/consultants**

      [root@localhost ~]# **ls -ld /home/consultants**

      drwxrwxr-x. 2 root consultants 6 Feb 1 13:21 /home/consultants

   c. Use the **chmod** command to forbid others from accessing files in the /home/consultants directory.

      [root@localhost ~]# **chmod 770 /home/consultants**

      [root@localhost ~]# **ls -ld /home/consultants**

      drwxrwx---. 2 root consultants 6 Feb 1 12:08 /home/consultants/
6. Exit the root shell and switch to the consultant1 user. The password is redhat.

   [root@localhost ~]# exit

   logout

[student@localhost ~]$

[student@localhost ~]$ su - consultant1

Password: redhat

7. Navigate to the /home/consultants directory and create a file called consultant1.txt.
    a. Use the cd command to change to the /home/consultants directory.
       [consultant1@localhost ~]$ cd /home/consultants
    b. Use the touch command to create an empty file called consultant1.txt.
       [consultant1@localhost consultants]$ touch consultant1.txt
8. Use the ls -l command to list the default user and group ownership of the new file and its permissions.

   [consultant1@localhost consultants]$ ls -l consultant1.txt

   -rw-rw-r--. 1 consultant1 consultant1 0 Feb 1 12:53 consultant1.txt
9. Ensure that all members of the consultants group can edit the consultant1.txt file. Change the group ownership of the consultant1.txt file to consultants.
    a. Use the **chown** command to change the group ownership of the consultant1.txt file to consultants.
       [consultant1@localhost consultants]$ chown :consultants consultant1.txt
    b. Use the ls command with the -l option to list the new ownership of the consultant1.txt file.
       [consultant1@servera consultants]$ ls -l consultant1.txt
       -rw-rw-r--. 1 consultant1 consultants 0 Feb 1 12:53 consultant1.txt
10. Exit the shell and switch to the consultant2 user. The password is redhat.

    [consultant1@localhost consultants]$ exit

    logout

    [student@localhost ~]$ su - consultant2

    Password: redhat

    [consultant2@localhost ~]$
11. Navigate to the /home/consultants directory. Ensure that the consultant2 user can add content to the consultant1.txt file. Exit from the shell.
    a. Use the cd command to change to the /home/consultants directory. Use the

echo command to add text to the consultant1.txt file.

[consultant2@localhost ~]$ cd /home/consultants/

[consultant2@ localhost consultants]$ echo "text" >> consultant1.txt

[consultant2@ localhost consultants]$

b. Use the cat command to verify that the text was added to the consultant1.txt file.

[consultant2@localhost consultants]$ cat consultant1.txt text

[consultant2@ localhost consultants]$

c. Exit the shell.

[consultant2@localhost consultants]$ exit

logout

[student@ localhost ~]$

**12.** Log off from servera.

[student@ localhost ~]$ exit

logout

## Tasks: Managing Default Permissions and File Access

1. Use the **ssh** command to log in to localhost as the student user.

   [student@localhost ~]$ **ssh student@localhost**

   [student@localhost ~]$

2. Use the **su** command to switch to the operator1 user using redhat as the password.

   [student@localhost ~]$ **su - operator1**

   Password: redhat

   [operator1@localhost ~]$

3. Use the **umask** command to list the operator1 user's default umask value.

   [operator1@localhost ~]$ **umask**

   0002

4. Create a new directory named **/tmp/shared**. In the **/tmp/shared** directory, create a file named defaults. Look at the default permissions.

   a. Use the **mkdir** command to create the **/tmp/shared** directory. Use the **ls -ld** command to list the permissions of the new directory.

      [operator1@localhost ~]$ **mkdir /tmp/shared**

      [operator1@localhost ~]$ **ls -ld /tmp/shared**

      drwxrwxr-x. 2 operator1 operator1 6 Feb 4 14:06 /tmp/shared

   b. Use the **touch** command to create a file named defaults in the **/tmp/shared** directory.

      [operator1@localhost ~]$ **touch /tmp/shared/defaults**

   c. Use the **ls -l** command to list the permissions of the new file.

      [operator1@localhost ~]$ **ls -l /tmp/shared/defaults**

      -rw-rw-r--. 1 operator1 operator1 0 Feb 4 14:09 /tmp/shared/defaults

5. Change the group ownership of **/tmp/shared** to operators. Confirm the new ownership and permissions.

   a. Use the **chown** command to change the group ownership of the **/tmp/shared** directory to operators.

      [operator1@localhost ~]$ **chown :operators /tmp/shared**

**b.** Use the **ls -ld** command to list the permissions of the **/tmp/shared directory**.

[operator1@localhost ~]$ **ls -ld /tmp/shared**

drwxrwxr-x. 2 operator1 operators 22 Feb 4 14:09 /tmp/shared

**c.** Use the touch command to create a file named group in the /tmp/shared directory. Use the ls -l command to list the file permissions.

[operator1@localhost ~]$ **touch /tmp/shared/group**

[operator1@localhost ~]$ **ls -l /tmp/shared/group**

-rw-rw-r--. 1 operator1 operator1 0 Feb 4 17:00 /tmp/shared/group

**6.** Ensure that files created in the /tmp/shared directory are owned by the operators group.

a. Use the **chmod** command to set the group ID to the operators group for the **/tmp/shared** directory.

[operator1@localhost ~]$ **chmod g+s /tmp/shared**

b. Use the **touch** command to create a new file named operations_database.txt in the **/tmp/shared** directory.

[operator1@localhost ~]$ **touch /tmp/shared/operations_database.txt**

c. Use the ls -l command to verify that the operators group is the group owner for the new file.

[operator1@localhost ~]$ **ls -l /tmp/shared/operations_database.txt**

-rw-rw-r--. 1 operator1 operators 0 Feb 4 16:11 /tmp/shared/ operations_database.txt

**7.** Create a new file in the /tmp/shared directory named operations_network.txt. Record the ownership and permissions. Change the umask for operator1. Create a new file called operations_production.txt. Record the ownership and permissions of the operations_production.txt file.

a. Use the **echo** command to create a file called operations_network.txt in the / tmp/shared directory.

[operator1@localhost ~]$ echo text >> /tmp/shared/operations_network.txt

b. Use the **ls -l** command to list the permissions of the operations_network.txt file.

[operator1@localhost ~]$ **ls -l /tmp/shared/operations_network.txt**

-rw-rw-r--. 1 operator1 operators 5 Feb 4 15:43 /tmp/shared/operations_network.txt

c. Use the **umask** command to change the umask for the operator1 user to 027. Use the umask command to confirm the change.

[operator1@localhost ~]$ **umask 027**

[operator1@localhost ~]$ **umask**

0027

d. Use the touch command to create a new file named operations_production.txt in the **/tmp/shared/** directory. Use the **ls –l** command to ensure that newly created files are created with read-only access for the operators group and no access for other users.

[operator1@localhost ~]$ **touch /tmp/shared/operations_production.txt**

[operator1@localhost ~]$ **ls -l /tmp/shared/operations_production.txt**

-rw-r-----. 1 operator1 operators 0 Feb 4 15:56 /tmp/shared/operations_production.txt

**8.** Open a new terminal window and log in to localhost as operator1.

[student@localhost ~]$ ssh operator1@localhost

[operator1@localhost ~]$

**9.** List the umask value for operator1.

[operator1@localhost ~]$ umask

0002

**10.** Change the default umask for the operator1 user. The new umask prohibits all access for users not in their group. Confirm that the umask has been changed.

a. Use the echo command to change the default umask for the operator1 user to 007.

[operator1@localhost ~]$ **echo "umask 007" >> ~/.bashrc**

[operator1@localhost ~]$ **cat ~/.bashrc**

# .bashrc

# Source global definitions

if [ -f /etc/bashrc ]; then

. /etc/bashrc

fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:

# export SYSTEMD_PAGER=

# User specific aliases and functions

umask 007

b. Log out and log in again as the operator1 user. Use the umask command to confirm that the change is permanent.

[operator1@localhost ~]$ exit

logout

Connection to servera closed.

[student@localhost ~]$ ssh operator1@servera

[operator1@localhost ~]$ umask

0007