Object permissions

Cedar Rapids Area Homeschools’ 2023 Cyberdefense Team

# This Week’s Plan

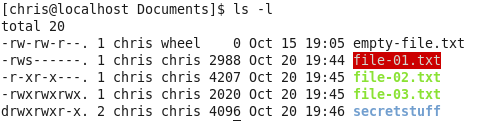
This week we are going to take a closer look at object permissions

# A Primer on Object Permissions

If you have time, read through the **Linux File Permissions** tutorial at <https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-permissions>. This page, provided by a GoDaddy competitor (see how fair I am?) is an abbreviated version of what we will talk about tonight. It’s really clear, although not nearly as informative as these notes (if I’m fair, I get to be honest, right?).

There are several other places on the Web that summarize how to work with Linux permissions. One is <https://dougvitale.wordpress.com/2013/02/16/linux-file-permissions-and-chmod/>, which is good and quite detailed / informative.

For what it’s worth, I would like to write my own summary of Linux object permissions here. Behold the output of the humble **ls -l** command:



For each entry listed in the output, there are ten characters at the beginning of the line that summarize the object type and permissions. Here is a summary of what those ten characters mean:

**Position 1:** This character shows the object type. For the files we look at the most, this is usually a hyphen, the letter d, or (less often) a lower-case “ell.” The possible values for this position are:

|  |  |
| --- | --- |
| **Character** | **Meaning** |
| - (hyphen) | The object is a regular file. |
| d | The object is a directory. |
| l (lower-case “ell”) | The object is a link, which is a pointer to another object. |
| b | The object is a block device, which is a device that the system uses to transmit data in blocks. Examples include devices used to represent hard disks or disk partitions (such as /dev/sda\*). |
| c | The object is a character device, which is a device that the system uses to transmit data one character at a time. Examples include device files for terminals (/dev/tt\*). |
| p | The device is a FIFO (first-in, first-out) queue. Examples on my CentOS include queues used by the postfix mail system (found in /var/spool/postfix/public). |
| s | The device is a local socket (local IP network connection). Postfix uses these also (they are found in /var/spool/postfix/public and /var/spool/postfix/private). |

**Using object type with the find command**

The usefulness of this comment might be more apparent to our returners than to our first-time learners. That’s OK. It’s worth mentioning that the find command takes a parameter called -type that allows you to find files that are one of these specific types. For example, find . –type f will find only regular files that are in the current directory or one of its subdirectories. You can search for more than one file type at a time, but you have to use a little more funky find syntax, specifically the -o operator which just means “or.” So, for example, the following command searches the whole file system (starting at the filesystem root, also known as **/** ) for files that are either FIFO queues or sockets:

find / -type p -o -type s

That’s enough of that for now. Let’s get back to those permissions.

**Positions 2 through 4:** These three characters show the permissions of the object’s owner.

|  |  |  |  |
| --- | --- | --- | --- |
| **Position** | **Character Value** | **Numeric Value** | **Meaning** |
| 2 | r | 0400 | The object’s owner can read the object. |
| 2 | - | 0000 | The object’s owner is not granted permission to read the object. |
| 3 | w | 0200 | The object’s owner can write the object. |
| 3 | - | 0000 | The object’s owner is not granted permission to write the object. |
| 4 | x | 0100 | The objects’s owner can execute the object. |
| 4 | - | 0000 | The object’s owner is not granted permission to execute the object. |
| 4 | s | 4000 | The SETUID bit has been set for the object. This means that when the object is executed, it will behave as though its owner executed it. The fact that the “s” is in lower-case means that the x permission is set for the object’s owner – in other words, the owner has permission to execute the object. |
| 4 | S | 4000 | The SETUID bit has been set for the object, just like in the previous entry. However, the “S” is upper case, which means that the x permission is not set for the object’s owner – in other words, the owner does not have permission to execute the object. |

**What does “execute” mean, exactly?**

Executing an object means different things for different object types. For example, executing a file means running it as a script or program, but executing a directory means making it your current working directory (perhaps by using the **cd** command).

**Positions 5 through 7:** These three characters show the permissions of the object’s group.

This is very (extremely) similar to positions 2 through 4, except that the focus is on the object’s group instead of its owner.

|  |  |  |  |
| --- | --- | --- | --- |
| **Position** | **Character Value** | **Numeric Value** | **Meaning** |
| 5 | r | 0040 | The object’s group can read the object. |
| 5 | - | 0000 | The object’s group is not granted permission to read the object. |
| 6 | w | 0020 | The object’s group can write the object. |
| 6 | - | 0000 | The object’s group is not granted permission to write the object. |
| 7 | x | 0010 | The objects’s group can execute the object. |
| 7 | - | 0000 | The object’s group is not granted permission to execute the object. |
| 7 | s | 2000 | The SETGID bit has been set for the object, as well as the “x” permission. This means that when the object is executed, it will behave as though a member of the object’s group executed it. The fact that the “s” is in lower case means that the x permission is set for the object’s group. |
| 7 | S | 2000 | The SETGID bit has been set, as explained above. However, the fact that the “S” is in upper case means that the “x” permission is not set for the object’s group. |

**Positions 8 through 10:** These three characters show the permissions of all users.

This is very similar to positions 2 through 4, except that the focus is on other users (or all users) instead of the object owner.

|  |  |  |  |
| --- | --- | --- | --- |
| **Position** | **Character Value** | **Numeric Value** | **Meaning** |
| 8 | r | 0004 | All users can read the object. |
| 8 | - | 0000 | All users are not granted permission to read the object. |
| 9 | w | 0002 | All users can write the object. |
| 9 | - | 0000 | All users group are not granted permission to write the object. |
| 10 | x | 0001 | All users can execute the object. |
| 10 | - | 0000 | All users are not granted permission to execute the object. |
| 10 | t | 1000 | The “sticky bit” has been set for the object, as well as the “x” permission. The meaning of the sticky bit is system dependent, but in Linux it means that only the object’s owner can rename or delete the object.   Typically, the sticky bit is set for the /tmp directory. The fact that the “t” is in lower case means that the x permission is set for all users. It’s called the “sticky” bit because on some older systems, it was used to cause a copy of the contents of a file to be held in the swap filespace, so that it could be loaded faster. |
| 10 | T | 1000 | The “sticky bit” has been set for the object, as explained above. However, the fact that the “T” is in upper case means that the “x” permission is not set for all users. |

**Object owner and group:** After the ten permission characters, the next two columns of output for **ls -l** are the object’s owner and group, respectively. In the sample **ls -l** output above, the file **empty-file.txt** is owned by the user **chris** and associated with the group that is also named **chris**.

# Managing Permissions with chmod and chgrp

## Using chmod with numeric permission modes

The overall security settings of an object are called its mode. The command **chmod** stands for change mode and can be used to change the security settings of an object. You do this by adding up by adding up the (non-zero) numeric values from the tables above and using the resulting number as a 3- or 4-digit mode. For example:

# allows read-only access to myfile for myfile’s owner only:  
chmod 400 myfile   
  
# allows read and execute access to myfile’s owner and read access to myfile’s group  
# and to all other users:  
chmod 544 myfile  
  
# allows full (read / write / execute) access to myfile for any user, and also sets   
# the SETUID bit:  
chmod 4777 myfile

## Using chmod with symbolic permission modes

The **chmod** command can also be used with symbolic modes. This is more flexible, but also more verbose. I will show you some examples below. If you want to know more about using chmod symbolically, please study the article <http://en.wikipedia.org/wiki/Chmod> on Wikipedia.

First you need to know these symbolic abbreviations:

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| u | Owner of the object (user) |
| g | Group of the object |
| o | All users (others) |
| r | Read permission |
| w | Write permission |
| x | Execute permission |
| X | Special execute permission (see the Wiki article) |
| s | SETUID or SETGID (depends on usage) |
| T | Sticky bit |
| + | Add the specified permissions |
| - | Remove the specified permissions |

Here are the same examples we looked at before, but using symbolic modes instead of numeric modes:

# allows read-only access to myfile for myfile’s owner only:  
# the go= part is important because it sets the group and other parts of the mode to 0:  
chmod u=r,go= myfile   
  
# allows read and execute access to myfile’s owner and read access to myfile’s group  
# and to all other users:  
chmod u=rx,go=r myfile  
  
# allows full (read / write / execute) access to myfile for any user, and also sets   
# the SETUID bit:  
chmod u=srwx,go=rwx myfile

OK, those examples are pretty complicated actually, and would be at least as easy to do in numeric mode. However, consider the following examples. In each case symbolic mode lets you make a change to the object’s permissions without checking the existing permissions first.

# add read access to myfile for myfile’s owner:   
chmod u+r myfile   
  
# remove execute permission for myfile from the “all users” permissions:  
chmod o-x myfile

## Managing an object’s owner and group with chown and chgrp

To manage the owner and group of an object, use the commands **chown** (change owner) and **chgrp** (change group).

The **chown** command can be used to change an object’s owner, group, or both. For example:

# change myfile’s owner to be santa:  
chown santa myfile   
  
# change myfile’s owner to be santa, and its group to be elves:  
chown santa:elves myfile

# change myfile’s group to be reindeer:  
chown :reindeer myfile

The **chgrp** command changes an object’s group, but cannot change its owner. For example:

# change myfile’s group to be grinches:  
chgrp grinches myfile

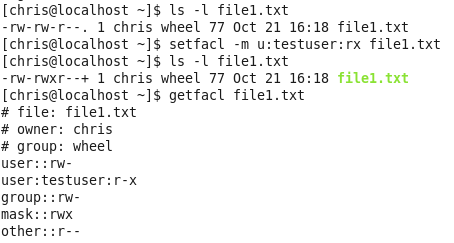
# Wait – what’s that 11th character?

If you were very observant, you might notice that there are not just 10 characters listed by **ls -l** in the permissions area, but actually 11. So far, every entry we’ve seen has had a dot in that position. This will usually be the case, and when the 11th character is a dot, it means “nothing to see here – move on.”

However, sometimes that dot (“.”) will be replaced by a plus sign (“+”). For example, in the following ls output, the object file1.txt has a dot in position 11. So the only permissions that apply to this file are read and write for the user chris, read and write for the group wheel, and read-only for all others.



However, we can use the **setfacl** command to add additional permissions for specific users and groups. For example, the following session shows the user adding read and execute permission for the user named games, and then that access is displayed using the **getfacl** command.



Among other surprising things, we see that the **ls -l** output shows the file as executable by its group! This isn’t actually true. The way I read **–rw-rwxr--+** is “oh wow, there’s a plus sign at end, so I need to run **getfacl** to see what’s really going on. And the **x** in the group part of the line kind of means “somebody can execute this who is not the owner, but not everyone can execute it, so the best place I have to stick the x is in between someplace – which means I put it in for the group.”