An introduction to Linux Access Control Lists (ACLs)

Linux Access Control Lists, or ACLs, can take some getting used to, but they're invaluable for getting a finer-grained control of your Linux filesystem permissions.

Posted: February 6, 2020 | Glen Newell (Sudoer)

*Photo by **Pixabay** from **Pexels***

Among the challenges of administering Linux in the modern business environment is the expectation that we can and should manage who has access to what information. Once upon a time, the only folks who needed access to Linux filesystems could be categorized in a general way: through Linux filesystem permissions.

Reviewing the basics

More Linux resources

- Advanced Linux Commands Cheat Sheet for Developers

- Get Started with Red Hat Insights
- Download Now: Basic Linux Commands Cheat Sheet
- Linux System Administration Skills Assessment

The Linux filesystem gives us three types of permissions. Here is a simplified review:

- **U**ser (or user owner)
- **G**roup (or owner group)
- **O**ther (everyone else)

With these permissions, we can grant three (actually five, but we'll get to that in a minute) types of access:

- **R**ead
- **W**rite
- e**X**ecute

These levels of access are often adequate in many cases. Say that you have a directory where files from the accounting department live. You might set these permissions to:

```
drwxrwxr-x  2 accounting accounting  6 Jan  8 15:13
```

The accounting service user (the user owner) can read and write to the directory, and members of the `accounting`group (or owner group) can read and write, but no one else can. Note that with these permissions, others can *see*what's in there, which some might think is a bad idea.

[Also popular: Linux sysadmin basics: User account management]

So, we might change the permissions to this:

```
drwxrwx---  2 accounting accounting  6 Jan  8 15:13 .
```

Note: You can also use something called sticky bits to control settings, like who actually owns the files created in that directory, and whether members of the group can edit each others' files. However, that's outside the scope of this discussion.

Viewing the current ACL

What if you have an intern (Kenny) who needs to be able to read certain files (or even just the files owned by Fred)? Or maybe people in the sales department also need access to the `accounting` owner's files to create invoices for that Fred's team in order to bill customers, but you don't want them to see the other reports that his team generates. This situation can be tricky because, with regular permissions, each file and directory can have only one user and group owner at a time. This type of situation is what Linux Access Control Lists (ACLs) were intended to resolve.

ACLs allow us to apply a more specific set of permissions to a file or directory without (necessarily) changing the base ownership and permissions. They let us "tack on" access for other users or groups.

We can view the current ACL using the `getfacl` command:

```
[root@lab1 accounting]# getfacl /accounting
getfacl: Removing leading '/' from absolute path names
# file: accounting
# owner: accounting
# group: accounting
user::rwx
group::rwx
other::---
```

We can see that right now, there are no ACLs on this directory. In this case, we expected this result, since I just created this directory in the lab and haven't done anything other than assigning ownership. So, let's start by adding a default ACL.

Setting an ACL

The syntax for setting an ACL looks like this:

```
setfacl [option] [action/specification] file
```

The 'action' would be `-m` (modify) or `-x` (remove), and the specification would be the user or group followed by the permissions we want to set. In this case, we would use the option `-d` (defaults). So, to set the default ACL for this directory, we would execute:

```
[root@lab1 accounting]# setfacl -d -m accounting:rwx /accounting
```

After which we can now see the default ACL info for that directory:

```
[root@lab1 accounting]# getfacl /accounting
[root@lab1 accounting]# getfacl: Removing leading '/' from absolute path names
# file: accounting
# owner: accounting
# group: accounting
user::rwx
group::rwx
other::---
default:user::rwx
default:user:accounting:rwx
default:group::rwx
default:mask::rwx
default:other::---
```

Now, what if I created a file in that directory as Fred?

```
[fred@lab1 accounting]$touch test.
[fred@lab1 accounting]$ ls -la
drwxrwx---+  2 accounting accounting  18 Jan  8 17:51 .
dr-xr-xr-x. 18 root  root  262 Jan  8 15:13 ..
-rw-rw----+  1 fred  accounting  0 Jan  8 17:51 test
[fred@lab1 accounting]$ getfacl test
# file: test
# owner: fred
# group: accounting
user::rw-
user:accounting:rwx  #effective:rw-
group::rwx  #effective:rw-
```

Now, what happens if we try to create a file logged in as user kenny? We can already guess that because kenny is not in the accounting group, he won't have permission. But we want Kenny to have a good experience working with us, so we need to give him the ability to see what files are in the accounting directory, and we want him to be able to create new files:

```
[root@lab1 accounting]setfacl -m kenny:rwx /accounting
[root@lab1 accounting]getfacl ./
# file: .
# owner: accounting
# group: accounting
user::rwx
user:kenny:rwx
```

Okay, so far so good. But what if we don't want this user to create files in the accounting directory? Instead, we only want to let him read the files there, and he can create new files in his own folder.

[Related article: [Linux sysadmin basics: User account management with UIDs and GIDs](#) **]**

We can set Kenny's access on the accounting folder like this:

```
[root@lab1 accounting]# setfacl -m kenny:r-x /accounting
[root@lab1 accounting]# getfacl ./
# file: .
# owner: accounting
# group: accounting
user::rwx
User:kenny:r-x
```

Now we make Kenny his own folder, give him ownership, and then make the accounting group the group owner so that other people in the accounting group can see what's in there:

```
[root@lab1 accounting]# mkdir ./kenny
[root@lab1 accounting]# chown kenny:accounting ./kenny
[root@lab1 accounting]# getfacl ./kenny
# file: kenny
# owner: kenny
# group: accounting
user::rwx
user:accounting:rwx
group::rwx
```

So, we've created a folder within the accounting group that is owned by user kenny. He should now be able to see the accounting folder, but only create files in his own folder:

```
[root@lab1 accounting]# su kenny
[kenny@lab1 accounting]$ touch test
touch: cannot touch ‘test’: Permission denied
[kenny@lab1 accounting]$ cd ./kenny
[kenny@lab1 kenny]$ touch test
[kenny@lab1 kenny]$ ls
test
```

Note that because the folder is owned by the `accounting` group, anyone in that group can put files there. Because we're dealing with an intern, this factor is probably fine. However, what if we give Kenny a promotion to chief auditor and want to keep his work a secret from Fred?

```
[root@lab1 accounting]# setfacl -m fred:- ./kenny
[root@lab1 accounting]# getfacl ./kenny
# file: kenny
# owner: kenny
# group: accounting
user::rwx
user:accounting:---
user:fred:---
```

What if we didn't want *anyone* to see what Kenny is working on?

```
[root@lab1 accounting]# setfacl -m g:accounting:- ./kenny
```

Note: When we want to set a *group* ACL, we need to specify this by putting `g:` in front of the group's name. For users, just change the `g` to a `u`, but `setfacl` will assume we are talking about a user if you don't put anything in that spot.

We still have to remove the base permissions for the group owner so that the rest of the accounting team can't snoop into Kenny's reports:

```
[root@lab1 accounting]# chmod g-rwx ./kenny
[root@lab1 accounting]# ls -al
total 0
drwxrwx-wx+  3 accounting accounting  44 Jan  9 16:38 .
dr-xr-xr-x. 18 root       root       262 Jan  8 15:13 ..
drwx------+  2 kenny      accounting  18 Jan  9 17:07 kenny
-rw-rw----+  1 root       root         0 Jan  9 16:33 test
-rw-rw----+  1 kenny      accounting   0 Jan  9 16:27 test2
[root@lab1 accounting]# getfacl ./kenny
# file: kenny
# owner: kenny
# group: accounting
user::rwx
user:accounting:---
user:fred:---
group::rwx  #effective:---
[root@lab1 accounting]# su jan
[jan@lab1 accounting]$ touch ./kenny/test
touch: cannot touch ‘./kenny/test’: Permission denied
```

Now we can manage who else can see or write to Kenny's folder without changing the ownership. Let's give the CEO (Lisa, who is not a member of the accounting team, and won't have access to the rest of the folder) access to Kenny's stuff:

```
[root@lab1 accounting]# useradd lisa
[root@lab1 accounting]# setfacl -m u:lisa:rwx ./kenny
[root@lab1 accounting]# su lisa
[lisa@lab1 accounting]$ touch ./kenny/lisa
[lisa@lab1 accounting]$ ls ./kenny
lisa  test
[lisa@lab1 accounting]$ touch test
touch: cannot touch ‘test’: Permission denied
[root@lab1 accounting]# getfacl ./kenny
# file: kenny
# owner: kenny
# group: accounting
user::rwx
user:accounting:---
user:fred:---
user:lisa:rwx
group::rwx
group:accounting:---
```

Note again that the group owner permissions remain wide open, but the accounting group (which is still the owner), no longer has access to that folder. So, who owns it?

```
drwxrwx---+  2 kenny  accounting  30 Jan  9 17:16 kenny
```

This part is tricky. It’s useful to know that we can take away the owner’s permissions without changing ownership, but you might want to consider whether this is the result you want.

Conclusion

So these are the basics. ACLs can be confusing, so I encourage you to give the man pages for `setfacl` and `getfacl` a good read. There are many more interesting and useful things you can do with these tools, but hopefully, you now understand enough to get you started.