Learning Objectives

Learners will be able to…

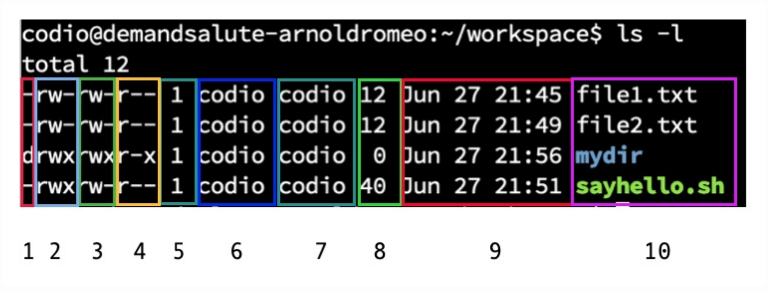
- Create and delete user and group accounts
- Perform account management tasks
- Identify and modify the correct configuration files for account management
- Execute commands as another user
- Analyze and troubleshoot user access and file permissions

Introduction to Permissions

Permissions allow you to control access to files. For example, you might have a document that you want everyone to view but not allow anyone but yourself to edit it. In another case you might want to grant a group of people edit and viewing access but not allow anyone else access. There are many permutations you might want to allow for with file access, permissions allow you to set those up.

The `ls -l` command

We have used the `ls` command many times in this course. Now we will dive a little deeper into the information that is revealed when you type `ls -l` and how you can modify the different attributes that are displayed. The `-l` stands for “long format”.



```
codio@demandsalute-arnoldromeo:~/workspace$ ls -l
total 12
-rw-rw-r-- 1 codio codio 12 Jun 27 21:45 file1.txt
-rw-rw-r-- 1 codio codio 12 Jun 27 21:49 file2.txt
drwxrwxr-x 1 codio codio  0 Jun 27 21:56 mydir
-rwxrw-r-- 1 codio codio 40 Jun 27 21:51 sayhello.sh

1 2  3  4  5   6     7    8      9          10
```

An ls -l directory listing with highlighted sections

Fields explained:
1. Blank for file, **d** for directory and **l** for link
2. Permissions for the owner (**r**ead, **w**rite, e**x**ecute)
3. Permissions for the group (**r**ead, **w**rite, e**x**ecute)
4. Permissions for everyone (**r**ead, **w**rite, e**x**ecute)
5. Number of links or directories inside (if it’s a directory)
6. User who owns the file or directory
7. Group owner of the file or directory
8. Size in bytes
9. Date of last modification
10. Name of the file or directory

Read, write and execute permissions

The following table shows what read, write and execute permissions mean for ordinary files and for directories:

	File	Directory

Read	Can read the file	Can list files in the directory
Write	Can edit the file	Can create and delete files in the directory
Execute	Can run the file as a program	Can change to the directory

Users

With **sudo** (super user do) privileges you can add users, delete users and modify their settings and passwords.

### Add a user:

```
sudo useradd testname
```

You can see that the new user has been added to the bottom of this user configuration file:

```
cat /etc/passwd
```

Files that are copied over to a new user's home directory are stored in `/etc/skel` (for skeleton). They are copied over the `/etc/profile` directory for the new user.

```
ls -lha /etc/skel
```

The file .bash_profile is read and executed when Bash is invoked as an interactive login shell. The .bashrc file is executed for an interactive non-login shell.

### Setup a password:

You can set up a password for the user:

```
sudo passwd testname
```

Encrypted passwords are stored in the file `/etc/shadow`. Account expiration information is also stored in that file.

You can switch to that user (`su` - stands for switch user):

```
su testname
```

Ask who you are to see the switch:

```
whoami
```

Switch back to the Codio user (password is codio):

```
su codio
```

Ask who you are to see the switch:

```
whoami
```

Changing user password expiration date `chage`

View the manual for this command

```
man chage
```

info

The `adduser` command

The `adduser` command wraps the `useradd` functionality with a script and adds other useful features you would want when adding a user such as prompting you for a password and other details and creating a directory for the user in the home directory.

Modify information:

You can learn more about the `usermod` command by looking at the manual pages.

```
man usermod
```

Delete a user:

```
sudo userdel testname
```

Other user information utilities

The `who` command displays a list of currently logged in users and time of login

```
who
```

The `w` command displays information about the users currently on a machine, and the processes they are running.

```
w
```

The `id` command displays user and group ids.

```
id
```

Groups

Groups make it easy to give sets of people permissions.

First we'll add a few users so that we have test cases for the group.

```
sudo useradd test1
sudo useradd test2
sudo useradd test3
sudo useradd test4
```

Create a new Group:

```
sudo groupadd mygroup
```

Add users to a group:

```
sudo adduser test1 mygroup
sudo adduser test2 mygroup
sudo adduser test3 mygroup
```

View the group information:

Group information is stored in the `/etc/group` file. Newest additions are at the bottom.

```
cat /etc/group
```

Delete a user from a group:

```
sudo deluser test3 mygroup
```

Modify group settings:

Learn more about the settings you can modify by looking at the manual:

```
man groupmod
```

Change a group’s name:

```
sudo groupmod -n mynewgroup mygroup
```

Delete a group:

```
sudo groupdel mynewgroup
```

File Permissions

Take a look at the files in the current directory by typing `ls -l` to get more information about them. You will notice that the file `sayhello.sh` is different from the others, it has an `x` in the user section of the permissions. This means the file is executable. Type the command below to execute it.

```
./sayhello.sh
```

▼ **Why do you need the `./` before the file name?**
The `./` is needed because the current directory is not included in the $PATH < /strong >$
$environmentvariable.WhenyourunacommandonthecommandlineLinuxsearchesthedirecto$
$strong >$**PATH** to find the executable. If we don't provide the path, Linux will not find the file.

To see the value of the `$PATH`, type `echo $PATH` on the command line.

Remove the execution permission with the following command and list the directory so we can see the change.
- The `u-x` part of the command below means: from **u**ser **-** (subtract) e**x**ecute.

```
chmod u-x sayhello.sh
ls -l
```

Try executing sayhello.sh by typing `./sayhello.sh`. That no longer works, but you can still use `bash` run the file:

```
bash sayhello.sh
ls -l
```

To give everyone execution privileges for the file you would use the following command:
- ugo stands for **u**ser, **g**roup and **o**ther.

```
chmod ugo+x sayhello.sh
ls -l
```

Default Permissions - `umask`

The `umask` command sets the default file permissions for newly created files in your current session.

1. Create a new file and view the permissions

```
touch test1.txt
ls -l test1.txt
```

2. Now change the umask value, create a file and view the settings for it

```
umask u=rw,g=r,o=r
touch test2.txt
ls -l test2.txt
```

3. To view the `umask` value in **symbolic** form (as opposed to octal) type:

```
umask -S
```

4. You can get more information about the `umask` command by typing `umask --help`.

### Giving different privileges to user, group, other

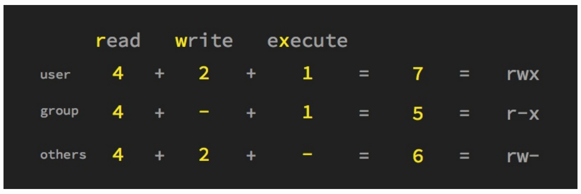
You can use commas to delineate different privileges for different constituencies:

```
chmod g-w,o-x sayhello.sh
```

You can also just set permissions using `=` rather than add or subtract them:

```
chmod u=rwx,g=r,o=r sayhello.sh
```

Permissions can be reduced to a 3 digit number



	read		write		execute				
user	4	+	2	+	1	=	7	=	rwx
group	4	+	-	+	1	=	5	=	r-x
others	4	+	2	+	-	=	6	=	rw-

Shows the numeric values for each privilege

Take a look at the **user** row :
The **total value of** 7 equals the `rwx` permissions for the **user** type and is the sum of the **read**, **write** and **execute** permission permanent values.

The **read** permission has a value of `4`, the **write** a value of `2` and **execute** a value of `1` for a total of 7.

Next look at the group row:
The **total value of 5** equals the `r-x`, `4` for **read**, `-` is `0`, 1 for **execute**.

Finally look at others row:
The **total value of 6** equals the `rw-`, `4` for **read**, `2` for **write** and `0` for **execute**.

To change the permission as represented above the command would be `chmod 756`.

Try it on the file `sayhello.sh` and see that it matches the image.

```
chmod 756 sayhello.sh
ls -l
```

File Attributes

The `chattr` command

The command `chattr` is short for **change attribute** and is mostly used to keep files secure and prevent them from being deleted accidentally or overwritten. The `chattr` command can only be run by **super users** and therefore must be preceded by the command `sudo`. These file attributes are stored in a file's metadata properties.

A few of the attributes for the `chattr` command

Flag	What it does
`a`	File can only be opened in append mode
`i`	Make a file immutable (uneditable)
`S`	File will be synchronously updated on the disk
`u`	Makes it possible to undelete the file.

Use a `+` to add to the existing attributes of the files, a `-` to remove an attribute and an `=` to overwrite the existing attributes.

More information about attributes here.

Example of setting the immutable attribute (Note: this command does not work in this virtual environment):

```
sudo chattr +i myfile
```

The `lsattr` command

The `lsattr` command lets you view the attributes of a file.

You can learn more about the `lsattr` command by typing:

```
more lsattr
```

The Access Control List

The Access Control List (ACL)

An ACL allows you to apply specific privileges to files and directories without needing to change owners or groups. It is a way to specify privileges for people/groups who are not the owner.

Utilities

Name	Description
`setfacl`	set file access control list
`getfacl`	get file access control list

1. Set up users and a group for the exercises on this page

```
sudo useradd janedoc
sudo useradd joedoc
sudo groupadd documentation
sudo adduser janedoc documentation
sudo adduser joedoc documentation
```

2. View the groups we set up

```
cat /etc/group
```

3. Look at the ACL entries for `file1.txt` and `file1.txt`, they do not have any

```
getfacl file1.txt
```

4. Give the group `documentation` all access to `file1.txt`

```
setfacl -m "g:documentation:rwx" file1.txt
```

5. Look at it after setting up the permissions for the group. Notice the `+` sign next to the priviliges for the file, that indicates there is an ACL in use.

```
getfacl file1.txt
```

6. Remove the ACL settings you just made for documentation group

```
setfacl -b file1.txt
```

7. Learn more about the things you can do with the `setfacl` command

```
setfacl -h
```

Special Permissions

Special permissions are a level above the user, group, anyone permissions we covered earlier. There are three special type permissions for executable files and directories.

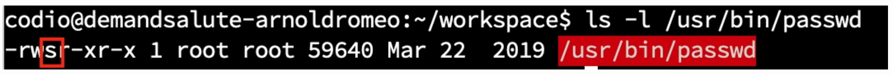
SUID

SUID stands for **set owner user id**. A file with an SUID permission will always execute as if run by the **user** who owns the file.

You can see an example of this:

```
ls -l /usr/bin/passwd
```

This particular file has an SUID permission set by default. Notice the `s` in place of an `x` in the execute permission of the user and the file name has a red background. This is important because users need to be able to change their own passwords, and with this bit set they don't have to be the root user to do so.



```
codio@demandsalute-arnoldromeo:~/workspace$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Mar 22  2019 /usr/bin/passwd
```

Set the SUID for the file `sayhello.sh`:

```
chmod u+s sayhello.sh
ls -l
```

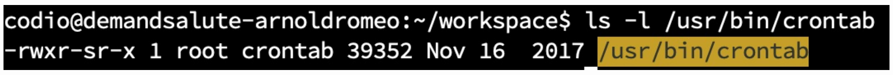
With the SUID bit set, anyone can run `sayhello.sh` as if they were the owner.

SGID

This is similar to the SUID - A file with an **SGID** permission will always execute as if run by the **group** who owns the file. In some versions of UNIX this setting is called GUID. If this permission is set on a directory, any files created in the directory will have their group ownership set to that of the owner of the directory.

The `crontab` executable is an example of a file with the **SGID** permission set:

```
ls -l /usr/bin/crontab
```



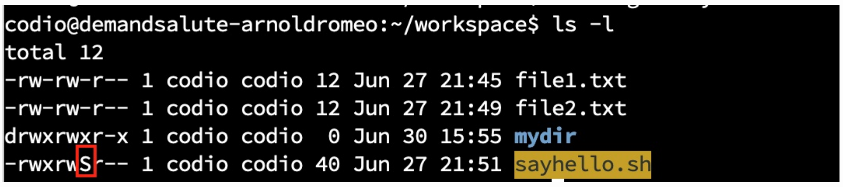
```
codio@demandsalute-arnoldromeo:~/workspace$ ls -l /usr/bin/crontab
-rwxr-sr-x 1 root crontab 39352 Nov 16  2017 /usr/bin/crontab
```

You can add the **SGID** permissions by running the commands below:

```
chmod g+s sayhello.sh
ls -l
```

important

Capitol S in privileges



```
codio@demandsalute-arnoldromeo:~/workspace$ ls -l
total 12
-rw-rw-r-- 1 codio codio 12 Jun 27 21:45 file1.txt
-rw-rw-r-- 1 codio codio 12 Jun 27 21:49 file2.txt
drwxrwxr-x 1 codio codio  0 Jun 30 15:55 mydir
-rwxrwSr-- 1 codio codio 40 Jun 27 21:51 sayhello.sh
```

There is a capitol `S` in the place of the execution privileges for the group. This is because we set the **SGID** and there were no execution privileges for the group in place.

▼ **How do you think you can fix the SGID for `sayhello.sh`?**

Running the command `chmod g+x sayhello.sh` will set this right.

Sticky bits

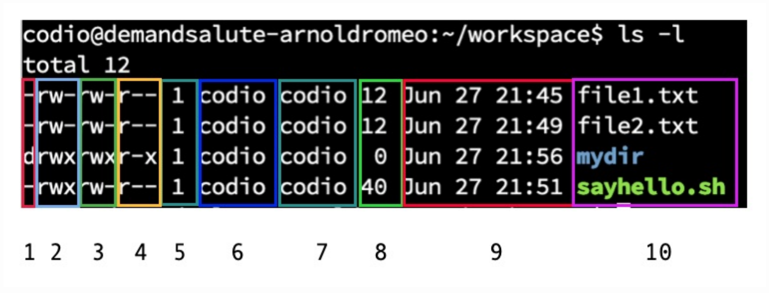
The **sticky bit** permission applies to directories only. When it is set, only the owner of a file and the root user can delete a file in that directory. This setting is particularly useful for shared directories.

You can see that setting on the `/tmp` directory (the `d` specifies you want information about the directory).

```
ls -ld /tmp
```

Owners

Information about a file or directory owner is also displayed when you run the command `ls -l`. In the image below, the user and group are both `codio`.



```
codio@demandsalute-arnoldromeo:~/workspace$ ls -l
total 12
-rw-rw-r-- 1 codio codio 12 Jun 27 21:45 file1.txt
-rw-rw-r-- 1 codio codio 12 Jun 27 21:49 file2.txt
drwxrwxr-x 1 codio codio  0 Jun 27 21:56 mydir
-rwxrw-r-- 1 codio codio 40 Jun 27 21:51 sayhello.sh
```

An ls -l directory listing with highlighted sections

Change the owner of a file or folder:

1. Add a couple of users and groups using the commands below

```
sudo useradd user1
sudo useradd user2
sudo groupadd group1
sudo groupadd group2
```

2. Change the owner of one of the files

```
sudo chown user1 file1.txt
ls -l
```

This command can also work recursively:

1. Add a couple of files to the `mydir` directory

```
touch mydir/myfile1.txt mydir/myfile2.txt
```

2. List them out, change the ownership and list them out again.

```
ls -l mydir
sudo chown -R user2 mydir
ls -l mydir
```

Change the group owner of a file or folder:

```
sudo chgrp group1 file1.txt
ls -l
```