Make a linux file executable, chmod, PATH

Introduction

When a file is created, it is not executable; its "mode" does not allow it to be run. The mode can be changed with the "chmod" command.

After I have an executable file, to run the program I have to specify the program's location. Does that make sense? Yes and no. How can I use "cp" and "rm" commands without specifying the location of these programs? The answer to this question is in the user's PATH.

Make a shell script executable

To start, I use the "touch" command to create script "zorro.sh".

```
$ touch zorro.sh
```

```
$ ls -lF
```

```
-rw-rw-r-- 1 df df      0 Nov 13 14:44 zorro.sh
```

Above, we see empty file "zorro.sh" has permissions:

	user (6)	**group** (6)	**other** (4)
(4) **r**ead	yes	yes	yes
(2) **w**rite	yes	yes	no
(1) e**x**ecute	no	no	no

If I want to e**x**ecute (run) this file, I have to change its permissions. The following changes the execute mode for all three categories (user, group, other):

```
$ chmod +x zorro.sh
```

or

```
$ chmod 775 zorro.sh
```

```
-rwxrwxr-x 1 df df      0 Nov 13 14:44 zorro.sh*
```

File zorro.sh can now be run by user, group, and other:

	user (7)	**group** (7)	**other** (5)
(4) **r**ead	yes	yes	yes
(2) **w**rite	yes	yes	no
(1) e**x**ecute	yes	yes	yes

A suggested reference for file and directory permissions is:
https://www.guru99.com/file-permissions.html

> Linux does not care about the file suffix, but I do because it tells me the type of file. Here, "zorro.sh" has the .sh suffix so I am reminded that I created this file as a shell script. But, Linux will let me run this file because it has execute permission, regardless of the file suffix (even without a suffix).

Execute (run) a script

I am going to put some content into the script using "gedit," my favorite, quick linux editor. Then I will explain how to run the script.

```
$ gedit zorro.sh &
```

```
#!/bin/bash                          <--  specify the shell
echo "  -->  whoami = $(whoami)"     <--  print my user name
echo "  -->  CWPROOT = $CWPROOT"     <--  print the value my CWPROOT
echo ""                              <--  print a blank line
df -H                                <--  print disk usage report
exit 0                               <--  end the script politely
```

Save, Exit

As I wrote in the Introduction, to run a script, specify the script's location. When I am in the same directory as the script, I use "`./`" (*here*) and the name of the script. No spaces!

```
$ ./zorro.sh
```

When I am not in the same directory as the script, I have to tell the system how to get to the script. For example, I am in my `/home/df` directory and my script is in my `work` subdirectory.

```
$ ./work/zorro.sh
```

The leading dot-slash ("./") means "here", then the name of the subdirectory, then the rest points to the script; that is, from *here* ("./"), go to my `work` subdirectory, then execute my `zorro.sh` script.

Or, I can supply the full path in either of these two ways:

```
$ /home/df/work/zorro.sh
```

```
$ ~/work/zorro.sh
```

PATH

This section is to understand PATH. There is probably nothing here that you will ever do.

In the previous section, I showed that to execute (run) a script that I create, I have to specify the location of the script. But, I also know I do not have to specify the location of system commands like "cp" (copy) and "rm" (remove) to run them. Why is there a difference? The answer is, programs in directories that are in my PATH are universally available to me.

To learn which directories are in my PATH, I enter the following command:

```
$ echo $PATH
```

My output is:

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/game
s:/usr/local/games:/snap/bin:/usr/su44r19/bin
```

Each colon separates a path. (Notice that my Seismic Unix "bin" directory is the last path.)

When I enter a command, like "cp", the system tries to find the command in my PATH. If a command is in my PATH, I can ask the system where it is. For example, where is the "cp" (copy) command?

```
$ which cp
```

My output is:

```
/usr/bin/cp
```

If a command is not in my PATH, the system returns, "command not found". Suppose I try to run the script I created earlier without specifying its location:

```
$ zorro.sh
```

My output is:

```
zorro.sh: command not found
```

I can copy (or move) the file to one of the "bin" directories in my PATH, then run it without specifying its location.

```
$ sudo cp ~/work/zorro.sh /usr/local/bin/.
```

Now when I enter

```
$ zorro.sh
```

the script runs!

The preceding also means I can have two (or more!) versions of executable files with the same name, one in a PATH directory and any others in directories not in my PATH. Each is "run" in a different way. The one in a PATH directory is run without specifying its location, the others are run by specifying their location in the run command.

Saying this another way, if I do not specify a path when I run a program, the system expects to find the program in a PATH directory.