Scripts and Pipes

Creating Scripts

Creating scripts for commonly run sequences of commands can come in handy. You can save the commands as a text file and then run the sequence as a single step. The steps required for creating a script from a bird's eye view are as follows:

1. Create a text file containing the commands.
2. Make the top line of the file a shebang (explained later).
3. Save file.
4. Make the file executable using permissions (explained later).
5. Run the command ./myScript.sh.

Below is an example of a simple script called name.sh:

```
#!/usr/bin/env bash
echo First name: $1
echo Last name: $2
```

This script takes two arguments, one for first name and one for last. These arguments are represented in the code with $1 and $2. It would be executed by running

```
./name.sh Philip Kirkbride
```

When run two lines will output, the first line being "First name: Philip" and the second line "Last name: Kirkbride". Unless of course you swap the input for your own name in which case the names will be swapped out.

File Permissions

As mentioned previously, the fourth step in making an executable script is changing the permissions on the file to allow execution. The short and simple way of doing this is to run

```
chmod +x name.sh
```

This simply adds the execution permission to the file for our current user. After running the command, you'll be able to make use of it simply by running the following (assuming you're in the same directory as the file):

./name.sh

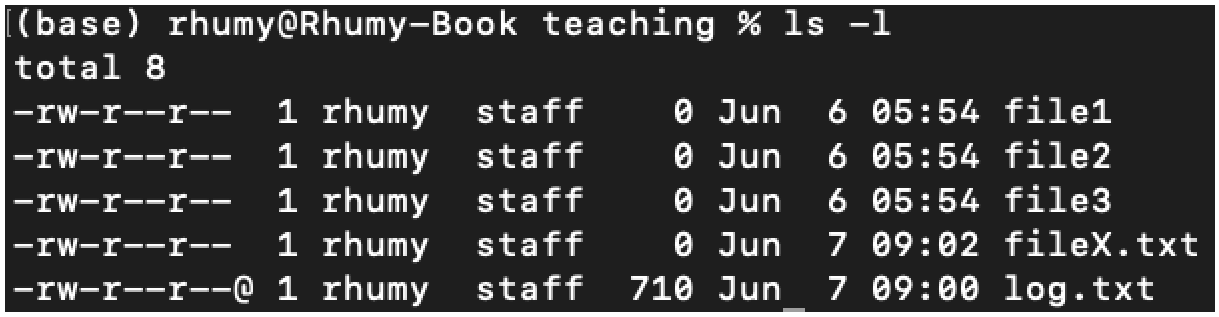
It's worth understanding the concept of permissions on Linux as it's a crucial aspect of the operating system. Every file has three different types of permissions:

- Read
- Write
- Execute

Each of these three permissions can be set separately for three groups:

- User
- Group
- Others

When using `ls -l`, you can see the set permissions for each file expressed on the left-hand side, as shown below:

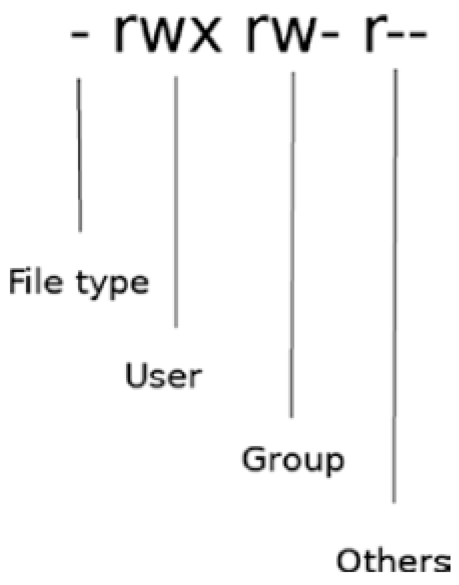


```
[(base) rhumy@Rhumy-Book teaching % ls -l
total 8
-rw-r--r--  1 rhumy  staff    0 Jun  6 05:54 file1
-rw-r--r--  1 rhumy  staff    0 Jun  6 05:54 file2
-rw-r--r--  1 rhumy  staff    0 Jun  6 05:54 file3
-rw-r--r--  1 rhumy  staff    0 Jun  7 09:02 fileX.txt
-rw-r--r--@ 1 rhumy  staff  710 Jun  7 09:00 log.txt
```

Fig 1: Permissions for files shown in the first column

Note: The first letter in this ten-letter sequence is used to indicate special file types. The possible values are d=directory, c=character device, s=symlink, p=named pipe, s=socket, b=block device, and D=door. We don't have to deal with these special types, but it's worth knowing what the first letter is.

After the first letter which indicates special file types, there are nine more letters. We can break these nine letters into three sets of three, as shown in Figure 2 – the first being file permissions for file owner, the second for user group, and the third for all other users.



For the three sections, we have three different letters which if present indicate that groups has said permission:

- r = read
- w = write
- x = execute

In the example given, we have a file with all permissions for the owner, read and write for the user group, and only the ability to read for all others who can access the file.
Permission data can also be displayed as a set of three numeric symbols where a single number represents the combination of the permissions. Each of the permission types is given a numeric value:

- 4 = read
- 2 = write
- 1 = execute

For any group we add up the permissions to get a number representing allowed permissions. For example, read and write would be 6 (2 + 4), write and execute would be 3 (1 + 2), and no permissions would be 0.

Using this notation, we would express: `rxw rw- r--` as 764. Either of these notations can be used when changing permissions for a file. For example, we can run

```
chmod 777 numbers.sh
```