ULI101: INTRODUCTION TO UNIX / LINUX AND THE INTERNET

WEEK 4: LESSON 2

FILE PERMISSIONS

LESSON 2 TOPICS

File Permissions

- Purpose
- **Directory** vs. **Regular File** Permissions
- Changing File Permissions (**chmod**)
- Setting File Permissions for Newly Created Directories and Regular Files (**umask**)
- Demonstration

Perform Week 4 Tutorial

- Investigation 2
- Review Questions (Questions 6 – 12)

FILE PERMISSIONS

The very first character is a hyphen "-" and this tells us the file is a regular file and not a socket, symlink, or another type of object.
The owner, group, and other permissions are listed in octal format.
The number of hard links pointing to this file. In this case, and in most cases, it will be one.
The file owner is murray.saul.
The group owner is users.
The file size is 6 bytes.
The file was last modified on 19th January 2019.
The file name is myregfile.

```
drwxr-xr-x 2 murray.saul users 6 Jan 19 14:06 mydir
-rw-r--r-- 1 murray.saul users 0 Jan 19 14:05 myregfile
```

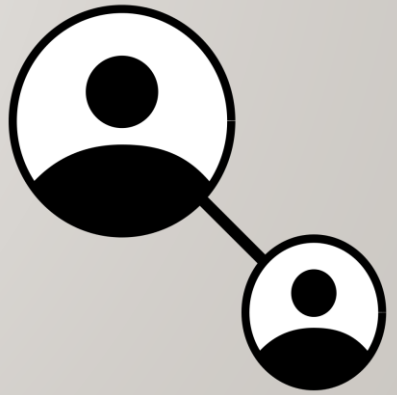
financial department vs human resource department

File Permissions

Since Unix / Linux operating systems allow for **multiple user accounts**, it is essential to have a system to **share** or **limit** access to directories and files contained in those file systems.

When **directories** and **regular files** are created, they are assigned to an **owner** (typically the username of the creator).

To *allow* or *limit* **access** to those files and directories, those files and directories are assigned to an initial **group** referred to as a "**primary group**".



Users that own those *directories* and *regular files* are referred to as **users**, users that belong within that **same primary group** are referred to as **same group members**, and those users are do NOT belong to a particular group are referred to as **other group members**.

The first position in the string is used to specify if the data object is a file (-) or a directory (d).
-rw-r--r-- (file) drwx------(directory)

FILE PERMISSIONS

Files (type):
No r:
cat, more, less, wc (DOESN't WORK)
ls, mv, rm (WORKS)
file: writable, regular file, no read permission (WORKS)

No w:
rm: need to confirm of deletion
ls, cat, more, less, file ,wc, mv

No x:
ls, rm cat, more, less, file, wc, mv (WORKS)

x for directory can allow you to pass through the directory, if you know the file locations

Directory (type):
No w:
mkdir(DOESN't WORK)

ls, cat, more, less, file, tree, mv, wc, rmdir (WORKS)

File permissions consist of **two-layers**:

- ***First***, the permissions relating to a **directory.**

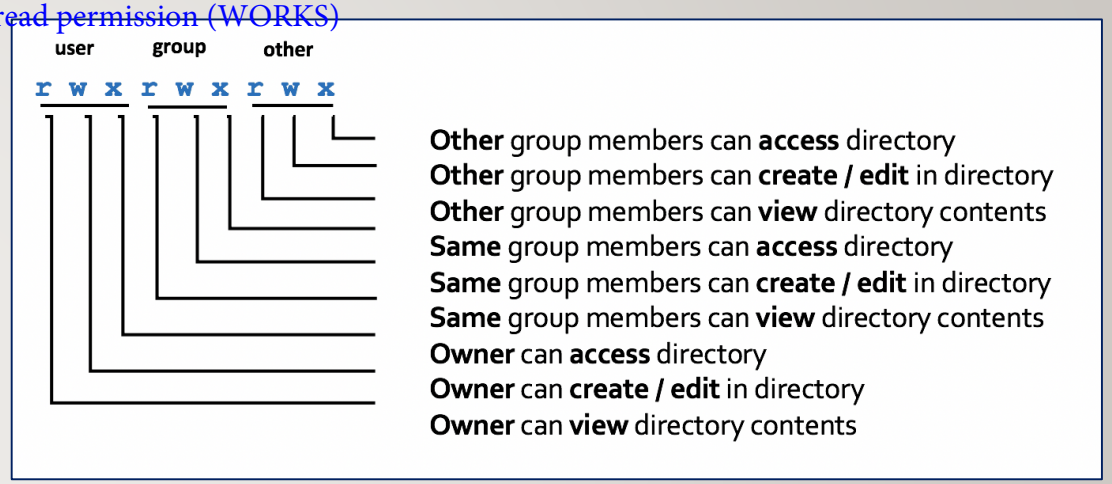
Directory (type):
No r:
ls, cat, tree, less, wc, rm (DOESN't WORK)
mkdir, file, more, mv (WORKS)

- ***Second***, the permissions relating to the **regular files** contained within a directory.

NOTE: *Permissions for **directories** have a different meaning than permissions for **regular files**.*

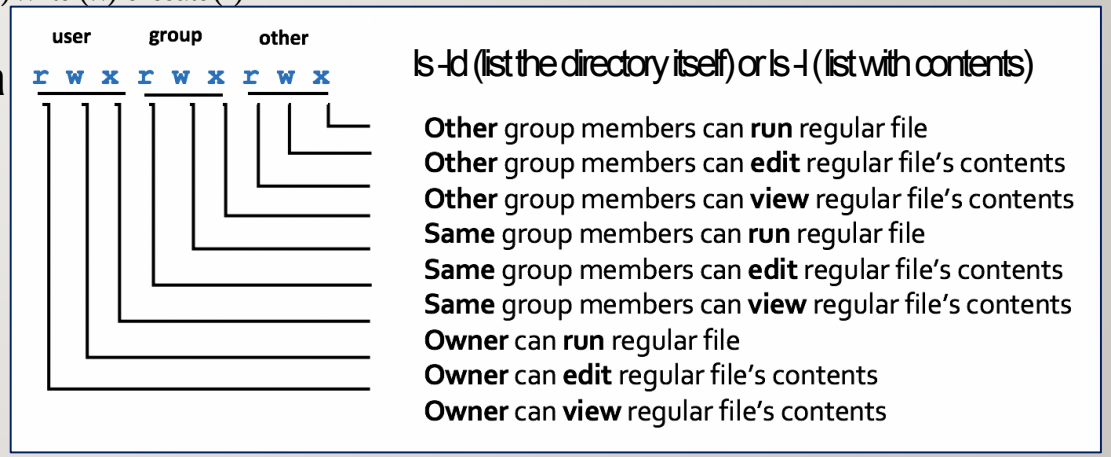
NOTE: A symbol *dash* **"-"** indicates that the permission is **NOT** granted.

Read permission (r):
The file can be opened, and its content viewed.
Write permission (w):
The file can be edited, modified, and deleted.
Execute permission (x):
If the file is a script or a program, it can be run (executed).



read (r) write (w) execute (x)

Octa



add / at the end : for file completion
/. = / = no /

Files need to manually add execution permission even though umask permit it

FILE PERMISSIONS

u: User, meaning the owner of the file.
g: Group, meaning members of the group the file belongs to.
o: Others, meaning people not governed by the u and g permissions.
a: All, meaning all of the above.

If none of these are used, chmod behaves as if "a" had been used.

–: Minus sign. Removes the permission.
+: Plus sign. The permission is added to the existing permissions. If you want to have this permission and only this permission set, use the = option, described below.
=: Equals sign. Set a permission and remove others.

Changing File Permissions with `chmod` command - *Symbolic Method:*

The **chmod** command can use **symbols** to **add**, **remove**, and **set `rwx`** permissions for **user**, **same group members**, **other group members** or **ALL** categories:

chmod -R o-x fileName

include only the subdirectory inside the directory you typed

NOTE: You can use the **-R** option to set permissions for directory, subdirectory and directory contents **recursively**.

Command	Description
`chmod ugo+x script.bash`	Add execute permissions to the file **script.bash** so it can be run.
`chmod u=rwx,go=x ~`	Set "**pass-thru**" permissions of your **home** directory for same group members and other group members to navigate to other subdirectories (that may have access / view permissions).
`chmod go-w ~/shared`	Remove write permissions for same group members and other group members for the directory **~/shared**
`chmod a=rx myfile.txt`	Set read and execute permissions for the directory **myfile.txt**

+: dont change anything if I dont tell u

In order to have access to directory contents, at least the "x" permission is necessary.This is called the "pass-through" permission. The pass-through permission is the key to grant access to only selected directories and/or files.

gcc -o hello hello.c : compile the code, which will add execute permission by default, as the complier knows that users gonna run it later

FILE PERMISSIONS

Instructor Demonstration

Your instructor will now demonstrate how to **add**, **remove** and **set** permissions with the **chmod** command the *Symbolic* method

FILE PERMISSIONS

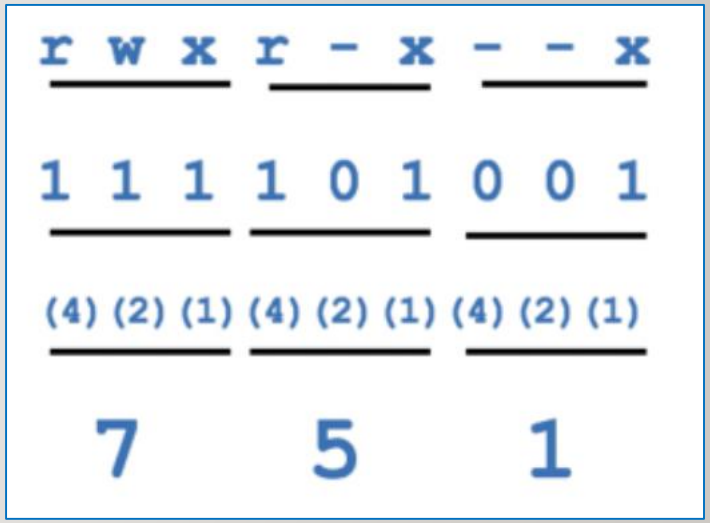
256 128 64 32 16 8 4 2 1

Changing File Permissions with `chmod` command -
Absolute (Octal) Method:

You can also use **octal numbers** to **set** permissions.

This method is a shortcut and may require less typing than using the *symbolic* method.

- **First**, write **permissions** for user, group and others that you want to set**. If permission is granted, write 1 and if not granted, write 0**.
- **Second**, perform a **binary to octal conversion**, for each group of three (user, group, other) and then issue the **chmod** command using the absolute (octal) method.



need to change all the permission for user,group,other at once and cannot change only one of them, give 3 number every time

You can only use this method to **set** file permissions (as opposed to *adding* or *removing* permissions).

FILE PERMISSIONS

Changing File Permissions with `chmod` command: *Absolute (Octal) Method*

Below is a table that displays common **chmod** commands (using the Absolute / Octal method) for common purposes.

Command	Description
`chmod 500 script.bash`	Set read and execute permissions for only the **user** for the file **script.bash** so it can be run.
`chmod 711 ~`	Set "**pass-thru**" permissions of your **home** directory. at least grant execute permission
`chmod 750 ~/shared`	Set full permissions for user, read and access permissions for some group members and no permissions for other group members for the directory **~/shared**
`chmod 555 myfile.txt`	Set read and execute permissions for the directory **myfile.txt**

FILE PERMISSIONS

Instructor Demonstration

Your instructor will now demonstrate how to **set** permissions with the **chmod** command using the *Absolute* / *Octal* method.

FILE PERMISSIONS

stat fileName:
Modify: The modification timestamp. This is the time when file's contents were last modified. (As luck would have it, the contents of this file were last changed four years ago to the day.)
Change: The change timestamp. This is the time the file's attributes or contents were last changed. If you modify a file by setting new file permissions, the change timestamp will be updated (because the file attributes have changed), but the modified timestamp will not be updated (because the file contents were not changed).

default permission

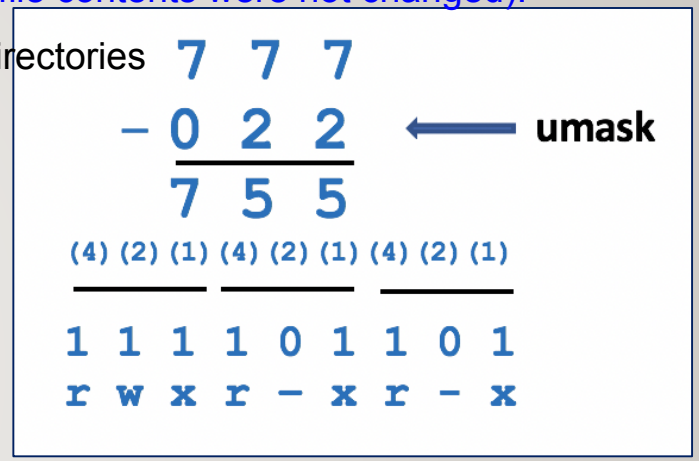
Setting Permissions for Newly-Created Directories and Regular Files (umask): user mask

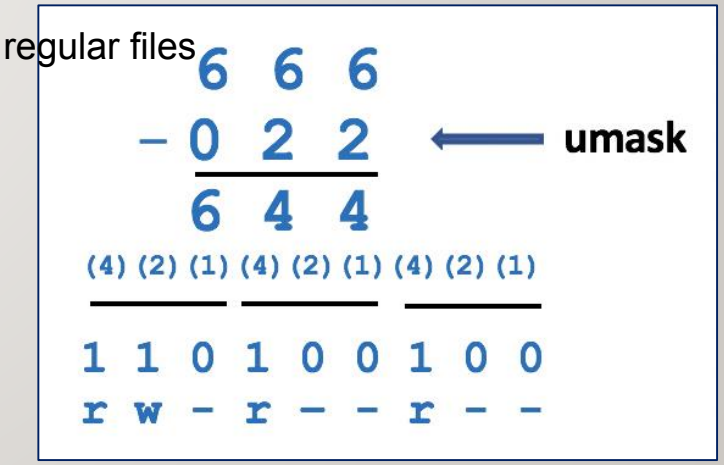
The `umask` command is used to set the permissions of **newly-created directories and regular files**. Issuing the `umask` command without arguments will display the current umask value.

The diagram on the above right shows how to calculate permissions for newly-created **directories** using the **umask** command.

The diagram on the below right shows how to calculate permissions for newly-created **regular files** using the **umask** command.

Setting the **umask** value works only in the current shell session unless the umask command is contained in a start-up file (e.g. **.profile**, **.bash_profile**, or **.bashrc**). Start-up files are discussed at the end of this course.





no x by default when create file or directories

FILE PERMISSIONS

Instructor Demonstration

Your instructor will now demonstrate how to **set / confirm** permissions of newly-created directories and regular files using the **umask** command.

HOMEWORK

Getting Practice

Perform the online tutorial **Tutorial 4: Unix / Linux File Management (Due: Friday Week 5 @ midnight for a 2% grade)**:

- INVESTIGATION 2: FILE PERMISSIONS
- LINUX PRACTICE QUESTIONS (Questions 6 – 12)