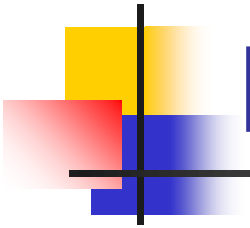
File Permissions



File Permissions

As you recall from our previous notes, that Unix/Linux recognizes everything as a file:

- Regular files to store data, programs, etc…
- Directory files to store regular files and subdirectories
- Special Device files which represent hardware such as hard disk drives, printers, etc…

You may ask, "Since I can navigate throughout the Unix/Linux file system – what prevents someone from removing important files on purpose or by accident?"

Answer: Ownership of the file, and file permissions

File Permissions

In previous classes, you only noted a few items from a detailed listing such as type of file, file size and date of creation/modification.

Let's look at the following detailed listing of a device (a hard-disk partition) located in the /dev (devices) directory and explore more items:

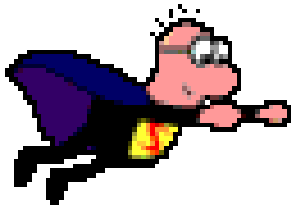
```
[username] ls -l /dev/hda
brw-r----- 1 root disk 3,0 2003-03-14 08:07 /dev/hda
```

Let's explore the results of this detailed listing in the next slide

File Permissions

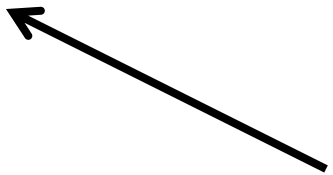
```
brw-r----- 1 root disk 3,0 2003-03-14 08:07 /dev/hda
```

This indicates the user who “owns” the file. In this case, the superuser or “root” probably created the file…



File Permissions

```
brw-r----- 1 root disk 3,0 2003-03-14 08:07 /dev/hda
```



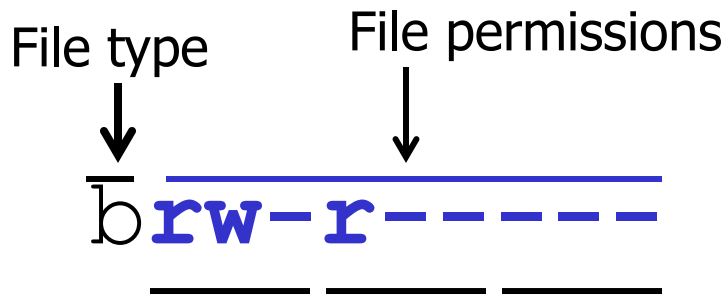
This indicates:

1. File Type (i.e. "**b**" or "**c**" for device file,
"**-**" for regular files, "**d**" for directory file)

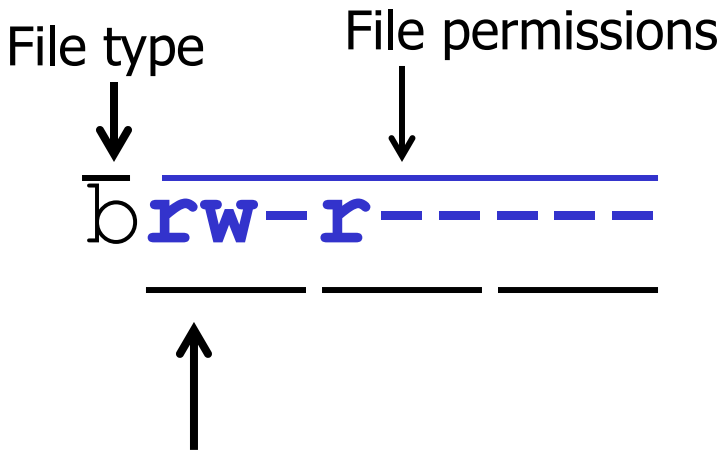
2. File Permissions (i.e. what permissions are granted by the owner regarding file access, file modification, and/or file execution)

Let's look at these permissions in more detail in the next slide…

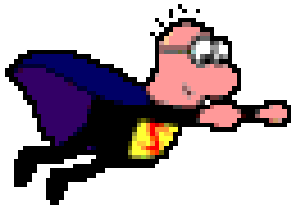
File Permissions



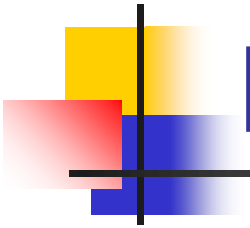
File Permissions



File owner permissions:



In this case, the owner (in this case root) can access (read) the file, the owner can modify (write) the file, but a dash instead of an "x" means that the owner cannot run (execute) the file like a program….

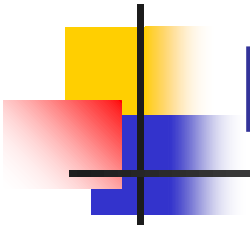


File Permissions

OK, I can now see that only the owner (root) is the only user that has permissions to make changes (write) to the file /dev/hda, so no other user can damage or edit and save changes to that file.

But what if an owner of a file wanted other users to view or write to their file? Can the owner of the file allow access to some users, and not to others?

Answer: That is what the other 2 sets of permissions are for. Look at the next slide…

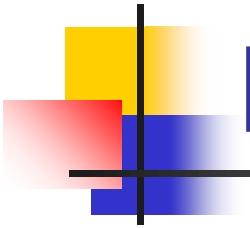


File Permissions

Let's look at the detailed listing for a regular file owned by someone else:

```
[murray.saul] ls -l ~/work_together

-rw-rw---- 1 murray.saul users 0 2006-02-02 10:47 ~/work_together
```



File Permissions

Let's look at the detailed listing for a regular file owned by someone else:

```
[murray.saul] ls -l ~/work_together

-rw-rw---- 1 murray.saul users 0 2006-02-02 10:47 ~/work_together
```

This indicates the user "murray.saul" owns the file "work_together". The owner "murray.saul" can read and write to that file.

By the way, you can change the ownership of files (assuming you own them by the chown command)

File Permissions

Let's look at the detailed listing for a regular file owned by someone else:

```
[murray.saul] ls -l ~/work_together

-rw-rw---- 1 murray.saul users 0 2006-02-02 10:47 ~/work_together
```

This indicates a group name (called "user") that is assigned to that file "work_together".

Group names can be set up (eg. by "root") and files can be assigned to those groups.
(for interest only, do a man on group, chgrp)

File Permissions

Let's look at the detailed listing for a regular file owned by someone else:

```
[murray.saul] ls -l ~/work_together

-rw-rw---- 1 murray.saul users 0 2006-02-02 10:47 ~/work_together
```

In this case the user "murray.saul" has given permission to other users that belong to the "users" group to read and write to the file "work_together".

Root can assign users to various groups … neat!

File Permissions

Let's look at the detailed listing for a regular file owned by someone else:

```
[murray.saul] ls -l ~/work_together

-rw-rw---- 1 murray.saul users 0 2006-02-02 10:47 ~/work_together
```

What does this last set of permissions refer to?

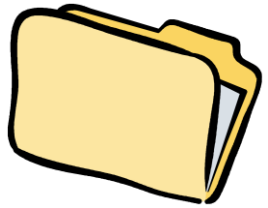
Answer: all "other" group names. In other words, users that DO NOT belong to the "users" group.

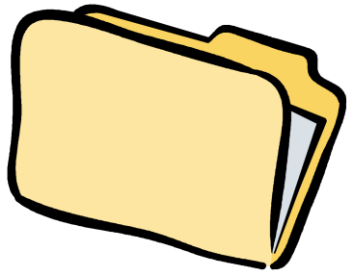
This allows the owner to be exclusive in file sharing!

File Permissions

OK, you can set permissions for the owner, the same group members or for "other" groups to read, modify (write) or run regular files, but how about permissions to REMOVE files or create new directories ?

Answer: You would need to set your directory permissions.





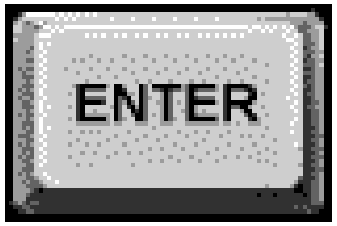
File Permissions

Directory Permissions

- Same concept remains for owner, group, and others
- r – Allows access to the directory
- r & x – Allows to access and view within directory
- r & w & x – You can do everything – eg. create subdirectories and remove files within directories.

(perhaps it is not a good idea to give ALL directory permissions to group or other since they can create files and directories in YOUR directory, but THEY would own that created file or directory, and may change permissions to deny YOU access to them, even if they are contained in your own directory!)

File Permissions



Home Directory Pass-Thru permissions

`rwx--x--x`

- Process of allowing group members and/or “other” group members to access your home directory.

- In this way, people can move to other directories that you have which may allow read and execute permission to view as well. In this way, you can tell friends the pathname to your shared information

File Permissions

The Internet

- Permissions for your home directory:
  `rwx--x--x`

- Permissions for your public_html directory:
  `rwxr-xr-x`

- Permissions for index.html file for access outside of the web-server location (i.e. “the world”)
  `rwx---r--`

File Permissions

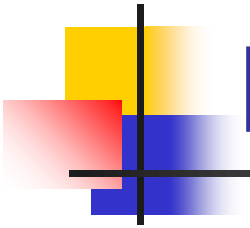
The Internet

- Permissions for index.html file for access inside and outside of the web-server location (i.e. everyone including "the world")

```
rwxr--r--
```

- Permissions for index.html file for access just inside the web-server location (commonly referred to as an INTRANET)

```
rwxr-----
```

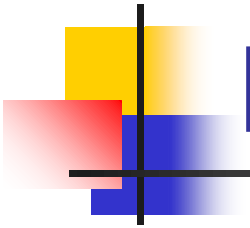


File Permissions

Changing Permissions via **chmod** command

chmod [option] [who] [operation] [permission] file

- Can be used change permissions for **directories** and **regular files**.
- There are two ways to set [who][operation][permission]:
  - Symbolic Method (using characters)
  - Absolute Method (using Octal Numbers)



File Permissions

Symbolic Method

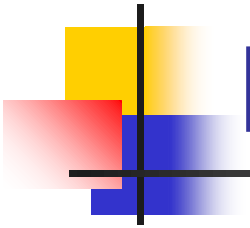
- who relates to user (u), group (g), others (o), or all (a)
- operation relates to adding (+), removing (-) or setting (=) permissions
- permissions are read (r), write (w) and execute (x)

Examples:

Add Permission: `chmod g+rw file_name`

Remove Permission: `chmod g-w file_name`

Set Permission: `chmod o=rx file_name`



File Permissions

Absolute Method

You can use the chmod command with octal number to represent (in binary) a permission (1) or removal of a permission (0) for the file or directory. It is considered to be a very fast and efficient method to set permissions (assuming you know binary to octal conversions)…

chmod 777 filename	**-rwxrwxrwx**
chmod 755 filename	**-rwxr-xr-x**
chmod 711 filename	**-rwx--x--x**
chmod 644 filename	**-rw-r--r--**