File Permissions in Unix

A presentation brought to you by the Blind Leading the Blind Foundation and Joe Shepherd

Hey, Joe, what are permissions?

Thanks for asking, Joe. First of all, this presentation is about Linux systems, so OSX folks pay attention.

Windows people, take a nap or visit

http://en.wikipedia.org/wiki/Cacls

I'm not sure Eliza likes your smartass attitude, Joe. Get Moving.

Good idea, Joe.

So, Every file on the system has a set of permissions that go with it. Permissions tell UNIX who can do what with a file. There are three things you can (or can't) do with a given file:

1) Read it

I'm not sure Eliza likes your smartass attitude, Joe. Get Moving.

Good idea, Joe.

So, Every file on the system has a set of permissions that go with it. Permissions tell UNIX who can do what with a file. There are three things you can (or can't) do with a given file:

1) Read it

2) Write it (modify it)

I'm not sure Eliza likes your smartass attitude, Joe. Get Moving.

Good idea, Joe.

So, Every file on the system has a set of permissions that go with it. Permissions tell UNIX who can do what with a file. There are three things you can (or can't) do with a given file:

1) Read it
2) Write it (modify it)
3) Execute it

Let's concentrate on the execute part. This is a lightning talk, not a molasses talk.

Now who's being a smartass?

Anyway, if you're in your terminal type

```
ls -l
```

in a directory and you'll see something like this:

```
-rw-r--r--  1  shepright  users   1462   Jul 10  18:30 NSS-lecture-notes.txt
```

```
-rw-r--r--  1  shepright  users   1462   Jul 10  18:30 NSS-lecture-notes.txt
```

These symbols:

-rw-r--r--

are the permissions.

The dashes - separate the permissions into three types:

1) The owner's (shepright's) permissions.

The dash - before the rw means that this is a normal file that contains any type of data. A directory, for example, would have a d instead of a dash.

The rw that follows means that I can read and write to (modify) my own file.

Well, duh.

```
-rw-r--r--  1  shepright  users   1462   Jul 10  18:30 NSS-lecture-notes.txt
```

These symbols:

-rw-r--r--

are the permissions.

The dashes - separate the permissions into three types:

2) Permissions for the group.

There are two dashes after the second r because there are no write permissions for the group.

```
-rw-r--r--  1  shepright  users   1462   Jul 10  18:30 NSS-lecture-notes.txt
```

These symbols:

-rw-r--r--

are the permissions.

The dashes - separate the permissions into three types:

3) Other, or global, user permissions.

Anyone who might have access to the computer from inside or outside (in the case of a network) can read this file. Like someone accessing a file with a browser.

Joe, are we at `chmod` yet? I'm bored.

ADD meds not kicked in yet, Joe?

`chmod` stands for **change mode**. It's how we change permissions. The command is written in this structure:

COMMAND : OWNER : GROUP : WORLD : PATH

An actual command looks like this:

Joe, are we at `chmod` yet? I'm bored.

ADD meds not kicked in yet, Joe?

`chmod` stands for **change mode**. It's how we change permissions. The command is written in this structure:

COMMAND : OWNER : GROUP : WORLD : PATH

An actual command looks like this:

`chmod 7 5 5 myDoc.txt`

Joe, are we at `chmod` yet? I'm bored.

ADD meds not kicked in yet, Joe?

`chmod` stands for **change mode**. It's how we change permissions. The command is written in this structure:

COMMAND : OWNER : GROUP : WORLD : PATH

An actual command looks like this:

`chmod 7 5 5 myDoc.txt`

Got it? Good, moving on...

WTF was that, Joe?

Just seeing if you were still awake, Joe.
OK -- about those numbers.

(COMMAND : OWNER : GROUP : WORLD : PATH)

`chmod 7 5 5 myDoc.txt`

This is called the **Octal Method**. Each permission has a number.

WTF was that, Joe?

Just seeing if you were still awake, Joe.
OK -- about those numbers.

(COMMAND : OWNER : GROUP : WORLD : PATH)

chmod 7 5 5 myDoc.txt

No permission = 0
Execute = 1
Write = 2
Read = 4

Combine them to apply multiple permissions to a single slot. In the ex. above, the owner can read, write & execute. 4+2+1= 7

The Octal Method is not a James Bond Movie

Nope, but it is a cool way to assign permissions.

With the combination we just looked at you type...

$chmod 755 <filename>

in the command line and get:

-rwxr-xr-x filename

in the directory list **(ls -l)**

So what about just using the r, w, x?

That's called the symbolic method. To get the same result from before (-rwxr-xr-x) you would write this:

```
$ chmod u=rwx,go=rx
```

u = user

g = group

o = other, or global

a = u,g,o

= **a**

Which leads us to...!

Which, leads us to...

Aw, you stole my line, Joe.

Sorry, Joe.

Which leads us to the command Eliza used on the *futureperfect.rb* file in the lecture yesterday:

```
$ chmod +x
```

It could say a+x, but you can leave the "a" out.

"+x" makes the file executable by everyone.

Yay.

That was fun, Joe!
Let's do `chown`, now.

But, Joe, it's 1:00 am and I still haven't done that table relationship chart that Eliza assigned me....you...us.

Screw that, Joe. She'll see all the hard work you did on this molass..I mean lightning talk and let that slide.

Somehow I doubt that, Joe. So suffice it to say that **`chown`** means "change owner". Use it to change the ownership rights of a file. It doesn't change read, write or execution permissions.

Joe, that was the worst ending since the *LOTR:The Two Towers* movie.

Bite me, Joe. You can just go look up the other 5 zillion things you can learn about file permissions yourself. Try these. They'll look familiar since I lifted all the important stuff for this talk directly from them:

1) **The chmod manual** `$ man chmod`

2) **Linux.org**

   `http://www.linux.org/article/view/file-permissions-chmod`

3) **Wikipedia**

   `http://en.wikipedia.org/wiki/Filesystem_permissions`

Say goodbye, Joe

Goodbye, Joe.