A How-To Guide on Linux Permissions By:
Michael Wicherski

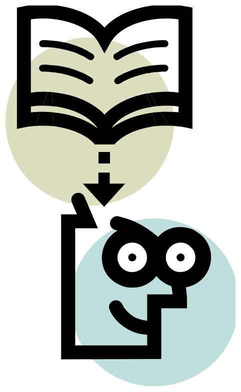
Table of Contents

Introduction

What you will need

A basic understanding of Linux and directory structure and a Linux interface, such as putty or Virtual Machine or an actual install.

The Basics



o.O The Basics; Begin DOWNLOAD of information… Er, I mean let's learn!

Permissions

What are the permissions?

Read means you can view the contents of a file/folder;

write means you can modify the file/folder;

and **execute** means you can execute the file/folder/program/etc (leave the x option alone (aka there) on folders because weird stuff will start to happen if you don't have execute permissions on a folder)

Format

- --- --- ---

Permissions are expressed in 4 sets. The first set is the type of file. It will always show as a " –, d, l " The other 3 sets are permissions.

The Three Parts

The remaining 3 sets are permissions for the "owner"(user) for the group and for others.

So the layout is:

- --- --- --- where each of the 3 bars is a permission setting r/w/x

Permission Codes

r = read

w = write

x = execute

Permissions seem intuitive enough. (See above if not)

Permission Code Table:

Permission	Its corresponding code
--x (execute)	1
-w- (write)	2
-wx (write and execute)	3
r-- (read only)	4
r-x (read and execute)	5
rw- (read and write)	6
rwx (read, write, and execute)	7
--- (none)	0

(To see how these codes are derived see the binary section under advanced of this tutorial)

Remember, you need 3 numbers to make a permission code; one for user(owner) one for group, and one for others (See permissions format)

Displaying Permissions

Two ways to do it.

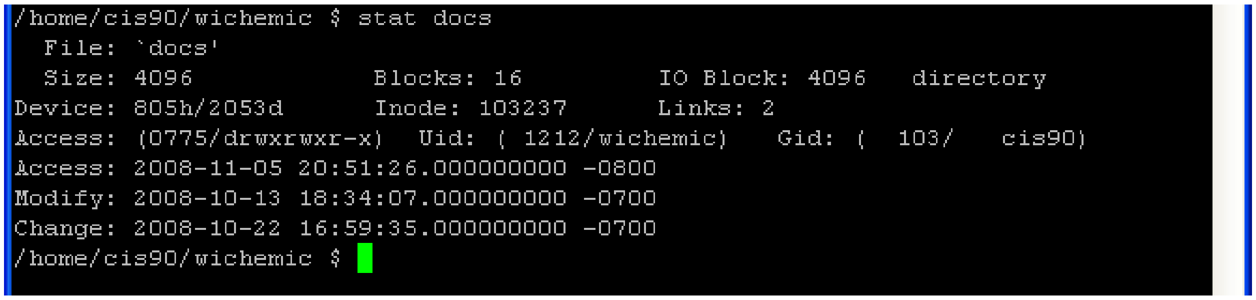
You can use what I hope by now is a familiar command   ls –l

Or you can use a brand new spiffy command called stat

Display Format

Depending on which command you use the display will be different

For stat this is what it looks like



```
/home/cis90/wichemic $ stat docs
  File: `docs'
  Size: 4096            Blocks: 16          IO Block: 4096   directory
Device: 805h/2053d      Inode: 103237       Links: 2
Access: (0775/drwxrwxr-x)  Uid: ( 1212/wichemic)   Gid: (  103/   cis90)
Access: 2008-11-05 20:51:26.000000000 -0800
Modify: 2008-10-13 18:34:07.000000000 -0700
Change: 2008-10-22 16:59:35.000000000 -0700
/home/cis90/wichemic $
```

You can see me check the permissions on the "docs" directory here.

stat tells you the file's name, the size, what type of file it is, in this case a directory; some mumbo jumbo you don't need for now, the inode number, the number of hard links, and finally the "access"

Access: (0775/drwxrwxr-x)

So, d means it is a directory, rwx rwx r-x = user has all; group has all; others have read and execute

(As for the numbers, see the binary section and then octal to understand what this means **note the leading 0 is for the sticky bit, but don't worry about that for now; that's addressed in more advanced classes)

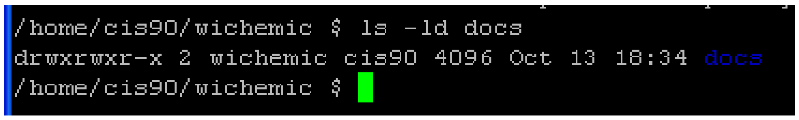
UID: tells you which user is considered the owner and therefore who the "user" permissions apply to

GID: tells you which group has access to this file and therefore who the "group" permissions apply to

And then it also gives you info about when the file was accessed, modified, etc.

As for ls –l

(since I am using docs which is a directory ls –ld)



```
/home/cis90/wichemic $ ls -ld docs
drwxrwxr-x 2 wichemic cis90 4096 Oct 13 18:34 docs
/home/cis90/wichemic $
```

It is much simpler.

drwxrwxr-x   2   wichemic   cis90   4096   oct 13 18.34   docs

d, it is a directory and then user group others links owner group size date created name

Permissions for New Files

When you make a new file or folder, they get default permissions assigned to them after the umask is applied.

The Defaults

The default permission sets are:

-rw-rw-rw- or 666 for files

drwxrwxrwx or 777 for folders (**Note – unless you want funky stuff happening, always leave execute permission on a directory**)

umask

umask is an “offset” of the defaults. It is the number you subtract from the default to get the actual permissions set on newly created files.

Note umask is ONLY taken into account when files are CREATED

What is the Point of umask?

You don’t always want the default set of permissions, which essentially give access to your files/folders to everyone. Setting a umask modifies files at creation so you don’t have to manually change them later.

How umask Does Its Thing

Let’s say the umask is 002 (Which is the default and what it gets set to everytime you login).

What would a newly created file’s permissions be?

Recall that umask is the number you subtract from the default

```
 666
-002
 664
```

So -rw-rw-r-- (See the reference table for permission # codes or octal explanation)

For a folder

```
 777
-002
 775
```

So drwxrwxr-x (Again, see the reference or octal)

Setting the umask

Setting the umask is simple.

Just type umask #

To set it to 002

```
umask 002
```

to set it to 777

```
umask 777
```

etc

Setting and Changing Permissions

We already know that when you first create a file it gets permissions set to it with the umask offset.

But now you have a file and want to change its permissions. How?

Simple, the command chmod takes all the pain away.

The chmod Command

All you need to remember is the following command to change permissions on folders/files:

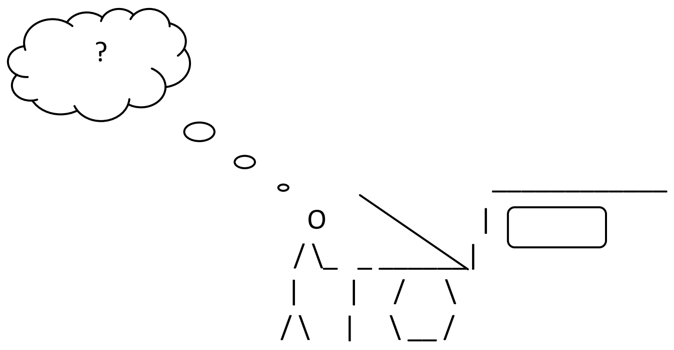
```
chmod <permission code #> <file>
```

Advanced

< _ > Help meeeeeeeeeee!

This is the section you do not really need to understand, BUT it will be VERY VERY VERY helpful to you if you do.

Under the Hood: How Things Work



Numbers! It’s all about the Numbers!

Number Systems

There are several number systems in the world, most important to you for the purpose of what this tutorial is about are decimal, binary, and octal, but I have also thrown in hexadecimal because it is really easy once you get octal ~.^

Links to subsections:

Decimal

This is the number system you should be most familiar and comfortable with since you have been using it since the dawn of your time. 0-9 is all you have here. Decimal, or in computer nerd terms base 10, (because you have 10 possibilities: 0,1,2,3,4,5,6,7,8,9).

Binary

The computer's decimal. Not only is this the computer's native number system, it is also its native language. Binary, computer nerds call it by its name! But it can also be referred to as base2, (because you have two possibilities 0 and 1). Think of 0 as false, 1 as true; 0 as non-existent, 1 as existent.

Base2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Equivalent	128	64	32	16	8	4	2	1

Binary numbers are derived from powers of 2.

Notice I made an 8 cell table, this is the most popular form of binary called an 8 bit binary number(also referred to as a byte). Every position of binary, that is each of the columns of this 8 column table, is called a bit. Every bit has the option of 0 and 1.

Trivia! - Have you ever heard the term byte? (megabyte?) A byte is 8 bits (megabyte is (about) 1000 kilobytes, kilobyte is (about) 1000 bytes.

Anyway, a question that would be good to address now is: How do you represent a decimal number (what humans like) in binary (what computers like)?

Recall that 0 = doesn't exist or not present; 1 = exists or present

For example, if you want to express 128 in binary - You would write 10000000

10000000 is 128 because there is 1 "unit" of 128 present in the number 128

01000000 is 64 because there is 1 "unit" of 64 present in the number 64

00000101 is 5 because there is 1 "unit" of 4 and 1 "unit" of 1 present in 5 (4+1)

00000111 is 7 because there is 1 "unit" of 4, 1 "unit" of 2, and 1 "unit" of 1 in 7 (4+2+1)

Note 00000111 is the same as 111 because in binary LEADING 0s do not matter, just like in decimal you can express 1 dollar as 1.00 or 01.00*

See the table below to see how that works if you feel confused, remember a 1 in binary means present or true; and 0 means nothing.

Base2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Equivalent	128	64	32	16	8	4	2	1
128 in Binary	1	0	0	0	0	0	0	0
64 in binary	0	1	0	0	0	0	0	0
5 in binary	0	0	0	0	0	1	0	1
7 in binary	0	0	0	0	0	1	1	1

Octal

Oct- ; we know the prefix means 8. That is exactly what this is. Octal is base 8. Which means every “column” can go between 0 and 7, base 8 = 8 options (0,1,2,3,4,5,6,7)

Now, going from binary to octal is very easy, recall that 2^3 is 8! That means that you group off a binary number into sets of 3 and just use their decimal equivalents. Let’s try with 7 for now.

Decimal 7 = 00000111_{base2} = 000 000 111 = 007_{base8}

(Notice we needed an extra 0 in front, which is ok because leading 0s do not affect binary)

Decimal 23 = 00010111_{base2} = 000 010 111 = 047_{base8}

See the relationship between the octal number system and permissions format? user group others

So let’s look at this relationship closer.

Decimal 7 = 00000111_{base2} = 000 000 111 = 007_{base8}

--- --- rwx = 007

Decimal 23 = 00010111_{base2} = 000 010 111 = 047_{base8}

--- -w- rwx = 027

So as far as permissions are considered, you only need to know enough of binary to get up to 8 combinations using decimal numbers so up to column of 2^2.

Notice how the 0s and 1s are in the same spots as permissions! 1 means true; 0 means false!

(Continue to next page)

Ask yourself the question: “Can I <permission> <file/folder>?”

For example: “Can I(owner) read testfile?”

Testfile permissions:

Octal = 664

Binary = 110110100

Letter Equivalent: rw-rw-r--

Yes I can read; true!

Summary! :

Binary	Decimal equivalent	Permission
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw-
111	7	rwx

And now you just need to be able to do it for 3 groups to get your octal equivalent of the letters and there you have it! – Your permission code number! (which you can now use with chmod !)

Hexadecimal

Hex… 16. Right so, base16: 16 options (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

LETTERS!?!?!?!?!?! … Yes, it is the only way to get between 0 and 15 in one slot.

Unlike binary and base8 when you write a hex number, you don't write a subscript to show it is hex.

Instead you write the number as 0x _ _ _ _ where the _ is a value 0-F

Now recall how octal was 2^3 ; hex is 2^4 . Any ideas? You might have guessed that to get a hexadecimal number from binary you simply group the binary off into 4 parts (adding leading 0s where needed) and just like octal, use the decimal equivalents for each slot

Decimal 15 = 00001111_{base2} = 0000 1111 = **0x**0F

Decimal 255 = 11111111_{base2} = 1111 1111 = **0x**FF

Decimal 154 = 10011010_{base2} =1001 1010 = **0x**9A

In the end, computers always gets down and dirty with the binary