

# Unix Permissions

Damien Suttle (dsuttle), Michelle Engel (mengel) et al.

October 2003

To view:

`gv /pro/consult/Minicourses/perms/perms.ps`

# ls -l

Sample ls output (from anon's home dir):

```
[cslab9c] /u/anon $ ls -l
total 64
-rw-----    1 anon    ugrad      209 May 17 07:23 TODO
-rw-r-----    1 anon    consult   8033 Sep 15 02:03 perms.tex
drwxr-x---     6 anon    ugrad    20480 Sep 15 14:31 bin
drwx-----   10 anon    ugrad   16384 Sep 15 16:20 doc
drwxr-xr-x     7 anon    ugrad    4096 Sep 14 18:27 pub
drwxr-x---     5 anon    cs017ta  4096 Sep 12 09:40 cs017
[cslab9c] /u/anon $
```

The command **ls** displays the files you have in your directory. Sometimes you want more information than just file names. **ls -l** gives you a “long” listing, as you can see above.

The first thing you see in each line is something like

```
-rw-r-----
```

What this tells you is the **permissions** that are set on the file – that is, who can do what to the file.

# A Bit About Unix

## Why do we need permissions in the first place?

- multi-user system
- need a method of protecting one user from another
  - securing a file so only the creator can view it (for example, your homework files..)
  - not allowing someone to delete someone else's file

## Entities in Unix

- user
  - unique user name
  - member of at least one group
- group
  - unique group name
  - 0 or more user members

Type **groups** or **groups [username]** in a shell to see which groups you or someone else is in.

# More About Unix

## Files and Directories

- every file and directory has one user owner and one group owner
- the user owner, group owner and everyone else (called other or world) can have different permissions to a file or directory

Example: Say I want to set permissions on my cs017 directory. I (user owner) want to be able to view and modify all the files. I'd like the cs017 ta's (group owner) to be able to view the files. But I do not want anyone else (other) to be able to view or modify the files.

Each entity (user owner, group owner, other) can have any or none of the following permissions on a file or directory:

read, write, execute

Weird thing is, read, write and execute mean slightly different things for files and directories.

# What Does It All Mean?

## Files

- read - view the file's contents
- write - modify the file's contents
- execute - execute the code stored in the file

## Directories

- read - view list of directory's files, their permissions, file size, etc (ie, you can 'ls' that dir)
- write - delete directory itself, create files in directory, delete files in directory
- execute - change current directory to that directory (ie, you can 'cd' to it)

Some examples forthcoming, first let's review 'ls -l's output.

# ls -l

Here's that sample output again.

```
[cslab9c] /u/anon $ ls -l
total 64
-rw-----    1 anon    ugrad      209 May 17 07:23 TODO
-rw-r-----    1 anon    consult    8033 Sep 15 02:03 perms.tex
drwxr-x---     6 anon    ugrad     20480 Sep 15 14:31 bin
drwx-----   10 anon    ugrad     16384 Sep 15 16:20 doc
drwxr-xr-x     7 anon    ugrad      4096 Sep 14 18:27 pub
drwxr-x---     5 anon    cs017ta    4096 Sep 12 09:40 cs017
[cslab9c] /u/anon $
```

Looking at the line with TODO on it, let's see what the important parts mean:

- 1st column (-rw-----) - the permissions for the file or directory. We'll talk about this in a second.
- 3rd column (anon) - the user owner of the file
- 4th column (ugrad) - the group owner of the file
- 7th column (TODO) - the name of the file

# Permissions in ls -l

```
drwxr-x---    6 anon    ugrad    20480 Sep 15 14:31 bin
```

Break it up:

```
d  rwx  r-x  ---  
|  |    |    permissions for Other  
|  |    permissions for Group owner  
|    permissions for User owner  
file type (is this a directory or not)
```

**r** - read, **w** - write, **x** - execute

How to interpret:

- Each entity (user owner, group owner, other) has 3 types of permissions (read, write, execute)
- If they have the permission, you'll see the character abbreviation, otherwise a '-' will be in its spot

Thus, in the example above:

- user owner: "rwx" can read, write, execute
- group owner: "r-x" can read and execute, but not write
- other: "---" cannot read, write or execute

# Examples

```
[cslab9c] /u/anon $ ls -l
total 64
-rw-----      1 anon      ugrad          209 May 17 07:23 TODO
-rw-r-----      1 anon      consult       8033 Sep 15 02:03 perms.tex
drwxr-x---       6 anon      ugrad        20480 Sep 15 14:31 bin
drwx-----     10 anon      ugrad        16384 Sep 15 16:20 doc
drwxr-xr-x       7 anon      ugrad         4096 Sep 14 18:27 pub
drwxr-x---       5 anon      cs017ta       4096 Sep 12 09:40 cs017
[cslab9c] /u/anon $
```

## So in short...

- TODO : anon can read and write, no one else has any access
- perms.tex : anon can read and write, users in group consult can read, everyone else has no access
- bin : anon can read, write and execute, users in group ugrad (all of you) can read and execute, everyone else has no access
- doc : anon can read and execute, everyone else has no access
- etc, etc...

# Changing Perms

## **chmod**

- Only works if you are the user owner of the file or directory
- To use: `chmod <new perms> <file or directory name>`

There are two ways to use chmod. One involves numbers, and one involves symbols. We'll show you numbers first.

## Chmod 666 [filename]

Suppose anon has finished making this excellent permissions tutorial. anon's file is called perms.tex, but it doesn't do anyone any good if they can't read it. As it is, only consultants can read this file, since they're in group **consult**:

```
-rw-r----- 1 anon    consult    8033 Sep 15 02:03 perms.tex
```

Instead, what we want is for anon to be able to read and write to the file, and for everyone else to be able to read it.

To do this, we use

```
chmod 644 perms.tex
```

Where does this 644 come from?

Notice that 644 has exactly 3 digits. This is nice, because permissions have three parts: permissions for user, permissions for group, and permissions for everyone else. Furthermore, if you look at the output from `ls -l`, you'll see that these permission are arranged from left to right:

```
- rw-(user) r--(group) r--(other)
```

In the number 644, we have

```
6(user) 4(group) 4(other)
```

The way we assign numbers goes like this:

- 4 corresponds to **r**, that is, read permission
- 2 corresponds to **w**, that is, write permission
- 1 corresponds to **x**, that is, execute permission

We want anon to have both read and write permission, so we give him the great sum of  $4 + 2 = 6$ . We want the group to have just read permission, so we give it 4. Same goes for other.

Suppose we wanted consultants to be able to write to the file too. Then we'd

```
chmod 664 perms.tex
```

because the middle digit applies to the group consult.

What would this do?

- `chmod 755 myFile`
- `chmod 610 myFile`

# Spelling lesson

There's another way to change permissions, which you might find more mnemonic. You use

```
chmod <new perms> <filename>
```

where the format of <new perms> is

- who the permission changes apply to
  - user owner (u) and/or
  - group owner (g) and/or
  - other (o) and/or
  - all (make it apply to user, group and other) (a)
- how to apply those changes
  - grant new permissions (+) OR
  - remove existing permissions (-) OR
  - set permissions to those specified (=)
- the permissions themselves
  - read (r) and/or
  - write (w) and/or
  - execute (x)

Confused? Let's see it in action.

# chmod Examples

```
-rw----- 1 anon ugrad 209 May 17 07:23 TODO
```

Say I want to make this file readable by the ugrad group. I can do:  
`chmod g+r TODO`

Now we have:

```
-rw-r----- 1 anon ugrad 209 May 17 07:23 TODO
```

Let's break it down:

- `g` : apply these changes to the group
- `+` : grant new permissions
- `r` : specify the read permission

Now I want to make the file writable by no one. I can do:  
`chmod a-w TODO`

Now we have:

```
-r--r----- 1 anon ugrad 209 May 17 07:23 TODO
```

Alternatively, since we knew that only the user owner had write permission, we could have done:  
`chmod u-w TODO`

# More chmod Examples

## Correct:

- `chmod go+rx foo` - grant the group owner and other read and execute permission on foo
- `chmod u=rwx bar foobar` - set the user owner's permissions to read, write and execute for bar and foobar
- `chmod g+w-x bar` - grant the group owner write permission and remove the execute permission for the group owner on bar

## Incorrect:

- `chmod g+ro-x foo` - the u, g, o and a must appear to the left of the first +, - or =

In the above case, you can do two things to make it work:

- simply make it two separate `chmod`'s
- use the comma separator, ie `chmod g+r,o-x foo`

If you want more info on `chmod`, you can type the following in a shell:  
`man chmod`

# Changing group owners

## chgrp

- Only works if you are the user owner of the file or directory AND you are a member of the group
- To use: `chgrp <group> <file or directory name>`

Say we have:

```
-rw-r----- 1 anon ugrad 209 May 17 07:23 TODO
```

I want to change the group owner of this file to consult. To do so, (assuming that I am a member of the consult group), I do:

```
chgrp consult TODO
```

Now we have:

```
-rw-r----- 1 anon consult 209 May 17 07:23 TODO
```

Remember, you can find out what groups you're in by typing **groups** in a shell.

# When to do this in the first place

You should always make sure you have your permissions set. There are stories about the kid who left a file in his home directory unprotected and found himself over quota on the next day because of an evil character who duly noted this fact... More to the point, you can be in violation of many classes' collaboration policies if your homework is world-readable. Sometimes you need to make things writable to a given group, such as your TAs. Besides, using `chmod` is incredible fun.

Other less obvious examples:

- You just copied a file from someone else's directory. Make sure the permissions are what you want, not what the original owner set them to be.
- You just untarred a nice tarball. Be aware that the permissions on the files inside the tarball are not necessarily the same as the permissions on the actual `.tar` file...
- On the flip side, if you're tarring a directory, want to share it with some friends, and make the `.tar` file world readable, you should be aware that the files inside the tarball might still not be readable at all
- Your friend wants you to edit some file, but has thoughtlessly forgotten to give you write permission for it. You do have read permission, though. Just copy the file and change the permissions on the copy.
- Watch out for the `.snapshot` directory. If you have some really sensitive file, and realize that for the past few weeks it's been world-readable, you should definitely change the permissions on that. Note however that in

the backups of that file, found in the .snapshot directory, the permissions stay the same... (moral: make sure you have the right permissions set at the time you create a file.)

# Practice Makes Perfect

Start by making a directory in your home directory.

```
mkdir ~/tempdir
```

Check the permissions on the directory by typing `ls -l` from your home directory. Most likely the default permissions are `drwxr-xr-x`. If they aren't, change the permissions to be so.

```
touch tempdir/tempfile to create a new file in tempdir
```

Now set the permissions of that file to be `-rw-rw----`. Try setting the permissions to be `-r--r--r--` and try deleting the file. Then set the permissions to be `--w-rw-rw-`. You should not be able to open the file, but you should be able to delete it.

```
touch tempfile again if you deleted it so you can use it later.
```

Now back up to your home directory again and set `tempdir`'s permissions to be `-rw-----`. Try to `cd tempdir`.

Change `tempdir`'s permissions again to be `dr-xr-x---`. `cd tempdir` and try to `touch newfile` and `rm tempfile`.

Now that you know how to change permissions and what different permissions can do, your `course/` directory, `.plan`, and `INBOX` should be safe from the rest of the CS department!