ATMS 305 WEEK 1: INTRODUCTION TO UNIX/LINUX

Lecture 2: File permissions and basic security

Adapted from: http://ryanstutorials.net/linuxtutorial/permissions.php

INTRODUCTION TO FILE PERMISSIONS

Last class, we learned about chmod, which is the command to change file permissions. In this section we'll learn about how to set Linux permissions on files and directories.

Permissions specify what a particular person may or may not do with respect to a file or directory. As such, permissions are important in creating a secure environment. For instance you don't want other people to be changing your files and you also want system files to be safe from damage (either accidental or deliberate).

Luckily, permissions in a Linux system are quite easy to work with if you learn a few basic concepts.

LINUX PERMISSIONS DICTATE 3 THINGS YOU MAY DO WITH A FILE, READ, WRITE AND EXECUTE. THEY ARE REFERRED TO IN LINUX BY A SINGLE LETTER EACH.

r read - you may view the contents of the file.

w write - you may change the contents of the file.

x execute - you may execute or run the file if it is a program or script.

FOR EVERY FILE WE DEFINE 3 SETS OF PEOPLE FOR WHOM WE MAY SPECIFY PERMISSIONS.

owner - a single person who owns the file. (typically the person who created the file but ownership may be granted to some one else by certain users)

group - every file belongs to a single group.

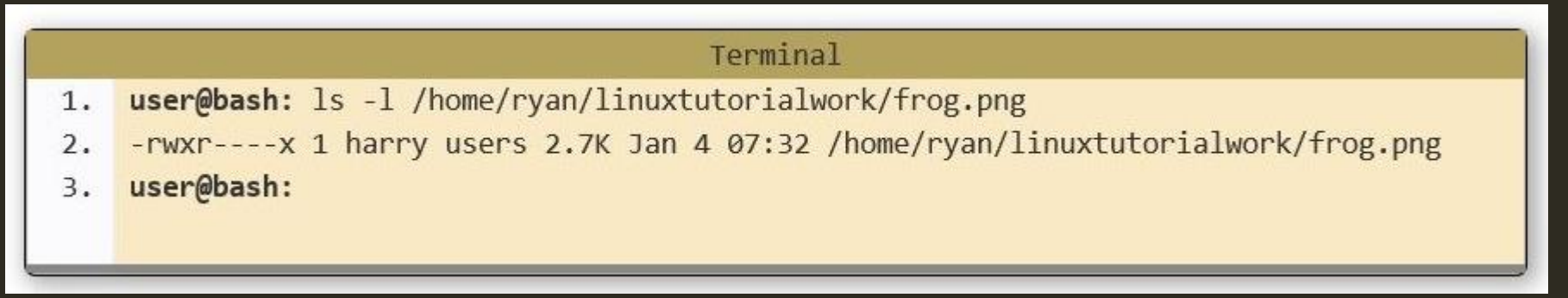
others - everyone else who is not in the group or the owner.

HOW TO VIEW PERMISSIONS

```
ls -l [path]
```

Terminal

```
1. user@bash: ls -l /home/ryan/linuxtutorialwork/frog.png
2. -rwxr----x 1 harry users 2.7K Jan 4 07:32 /home/ryan/linuxtutorialwork/frog.png
3. user@bash:
```



In the above example the first 10 characters of the output are what we look at to identify permissions.

- The first character identifies the file type. If it is a dash (-) then it is a normal file. If it is a d then it is a directory.
- The following 3 characters represent the permissions for the owner. A letter represents the presence of a permission and a dash (-) represents the absence of a permission. In this example the owner has all permissions (read, write and execute).
- The following 3 characters represent the permissions for the group. In this example the group has the ability to read but not write or execute. Note that the order of permissions is always read, then write then execute.
- Finally the last 3 characters represent the permissions for others (or everyone else). In this example they have the execute permission and nothing else.

CHANGING PERMISSIONS

To change permissions on a file or directory we use a command called **chmod** It stands for change file mode bits which is a bit of a mouthfull but think of the mode bits as the permission indicators.

```
chmod [permissions] [path]
```

chmod has permission arguments that are made up of 3 components

Who are we changing the permission for? [ugoa] - user (or owner), group, others, all

Are we granting or revoking the permission - indicated with either a plus (+) or minus (-)

Which permission are we setting? - read (r), write (w) or execute (x)

The following examples will make their usage clearer.

EXAMPLE 1

Grant the execute permission to the group. Then remove the write permission for the owner.

Terminal

```
1.  user@bash: ls -l frog.png
2.  -rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png
3.  user@bash:
4.  user@bash: chmod g+x frog.png
5.  user@bash: ls -l frog.png
6.  -rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
7.  user@bash:
8.  user@bash: chmod u-w frog.png
9.  user@bash: ls -l frog.png
10. -r-xr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```

EXAMPLE 2

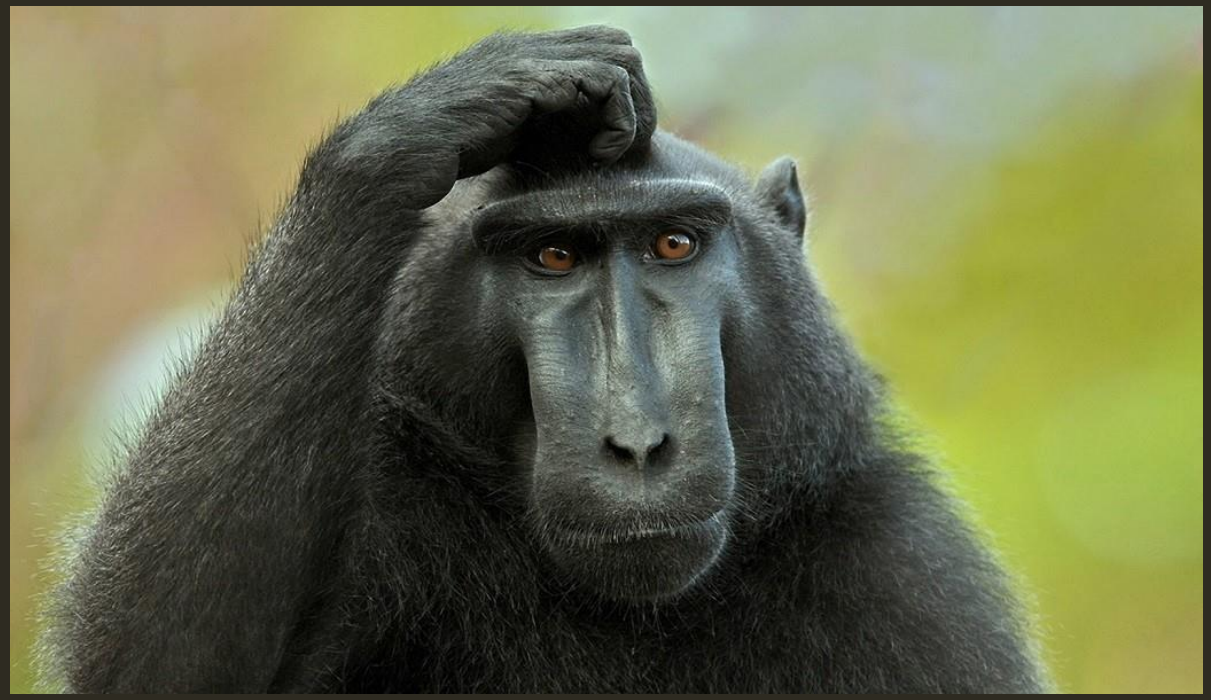
Don't want to assign permissions individually? We can assign multiple permissions at once.

Terminal

```
1.  user@bash: ls -l frog.png
2.  -rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png
3.  user@bash:
4.  user@bash: chmod g+wx frog.png
5.  user@bash: ls -l frog.png
6.  -rwxrwx--x 1 harry users 2.7K Jan 4 07:32 frog.png
7.  user@bash:
8.  user@bash: chmod go-x frog.png
9.  user@bash: ls -l frog.png
10. -rwxrw---- 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```

WHY ARE WE DOING THIS AGAIN?

It may seem odd that as the owner of a file we can remove our ability to read, write and execute that file but there are valid reasons we may wish to do this. Maybe we have a file with data in it we wish not to accidentally change for instance. While we may remove these permissions, we may not remove our ability to set those permissions and as such we always have control over every file under our ownership.



SETTING PERMISSIONS SHORTHAND

The method outlined above is not too hard for setting permissions but it can be a little tedious if we have a specific set of permissions we sould like to apply regularly to certain files. Luckily, there is a shorthand way to specify permissions that makes this easy.

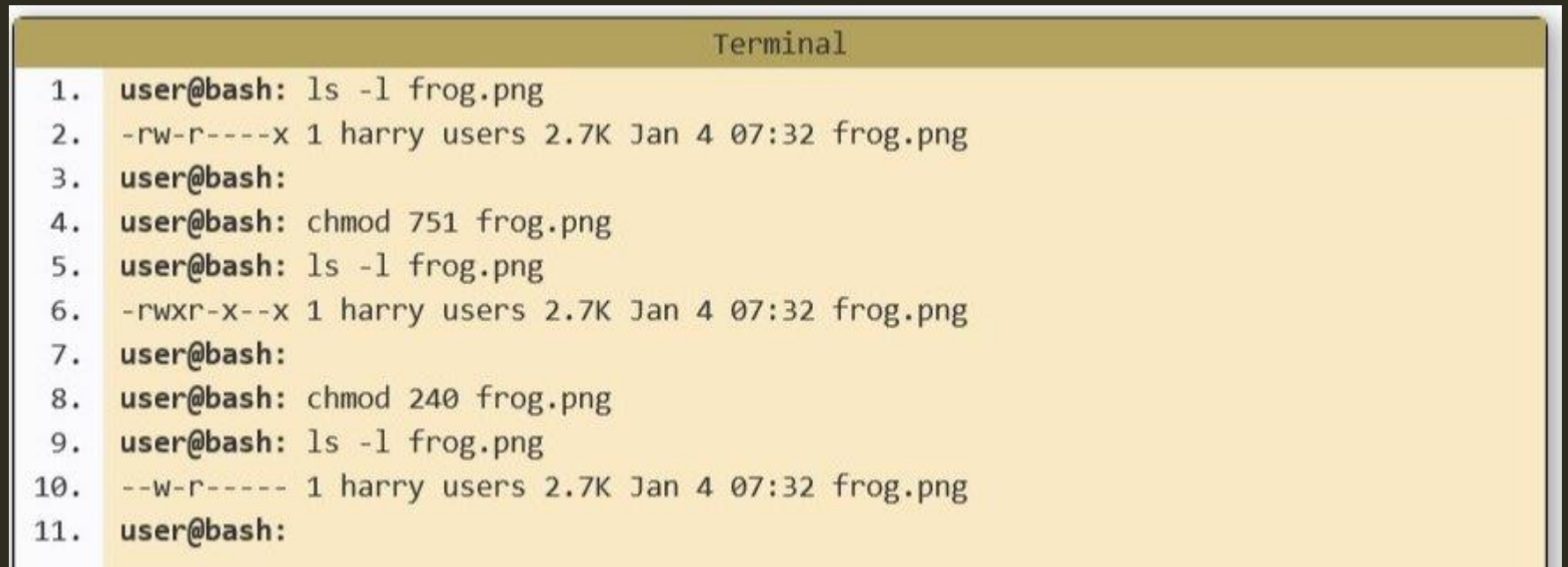
USING BINARY NUMBERS TO SET PERMISSIONS

To understand how this shorthand method works we first need a little background in number systems. Our typical number system is decimal. It is a base 10 number system and as such has 10 symbols (0 - 9) used. Another number system is octal which is base 8 (0-7). Now it just so happens that with 3 permissions and each being on or off, we have 8 possible combinations (2^3). Now we can also represent our numbers using binary which only has 2 symbols (0 and 1). The mapping of octal to binary is in the table at right.

Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

BINARY PERMISSIONS EXAMPLE

Now the interesting point to note is that we may represent all 8 octal values with 3 binary bits and that every possible combination of 1 and 0 is included in it. So we have 3 bits and we also have 3 permissions. If you think of 1 as representing on and 0 as off then a single octal number may be used to represent a set of permissions for a set of people. Three numbers and we can specify permissions for the user, group and others. For example:



ONE TO REMEMBER!

People often remember commonly used number sequences for different types of files and find this method quite convenient.

For example **755** or **750** are commonly used for scripts.

chmod 755 myscript.sh

THE SAME SERIES OF PERMISSIONS MAY BE USED FOR DIRECTORIES BUT THEY HAVE A SLIGHTLY DIFFERENT BEHAVIOR.

r - you have the ability to read the contents of the directory (ie do an ls)

w - you have the ability to write into the directory (ie create files and directories)

x - you have the ability to enter that directory (ie cd)

EXAMPLE 1

Note, on lines 5 and 14 when we ran ls I included the -d option which stands for directory. Normally if we give ls an argument which is a directory it will list the contents of that directory. In this case however we are interested in the permissions of the directory directly and the -d option allows us to obtain that.

Terminal

```
1.  user@bash: ls testdir
2.  file1 file2 file3
3.  user@bash:
4.  user@bash: chmod 400 testdir
5.  user@bash: ls -ld testdir
6.  -r-------- 1 ryan users 2.7K Jan 4 07:32 testdir
7.  user@bash:
8.  user@bash: cd testdir
9.  cd: testdir: Permission denied
10. user@bash: ls testdir
11. file1 file2 file3
12. user@bash:
13. user@bash: chmod 100 testdir
14. user@bash: ls -ld testdir
15. ---x------ 1 ryan users 2.7K Jan 4 07:32 testdir
16. user@bash:
17. user@bash: cd testdir
18. user@bash: ls testdir
19. ls: cannot open directory testdir/: Permission denied
20. user@bash:
```

REMEMBER

These permissions can seem a little confusing at first. What we need to remember is **that these permissions are for the directory itself, not the files within.**

So, for example, you may have a directory which you don't have the read permission for. It may have files within it which you do have the read permission for.

As long as you know the file exists and it's name you can still read the file.

EXAMPLE 2

Terminal

```
1.  user@bash: ls -ld testdir
2.  --x------- 1 ryan users 2.7K Jan 4 07:32 testdir
3.  user@bash:
4.  user@bash: cd testdir
5.  user@bash:
6.  user@bash: ls testdir
7.  ls: cannot open directory .: Permission denied
8.  user@bash:
9.  user@bash: cat samplefile.txt
10. Kyle 20
11. Stan 11
12. Kenny 37
13. user@bash:
```

THE (G)ROOT USER

THE ROOT USER

On a Linux system there are only 2 people usually who may change the permissions of a file or directory. (1) The owner of the file or directory and (2) the root user.

The root user is a superuser who is allowed to do anything and everything on the system. Typically the administrators of a system would be the only ones who have access to the root account and would use it to maintain the system.

Typically normal users would mostly only have access to files and directories in their home directory and maybe a few others for the purposes of sharing and collaborating on work and this helps to maintain the security and stability of the system.

BASIC SECURITY

Your home directory is your own personal space on the system. You should make sure that it stays that way.

Most users would give themselves full read, write and execute permissions for their home directory and no permissions for the group or others however some people for various reasons may have a slightly different set up.

Stuff We Learnt

chmod
Change permissions on a file or directory.

ls -ld
View the permissions for a specific directory.

Important Concepts

Security
Correct permissions are important for the security of a system.

Usage
Setting the right permissions is important in the smooth running of certain tasks on Linux. (we will see an example of this in Section 13 on scripting)

IN-CLASS ACTIVITIES TO TRY

First off, take a look at the permissions of your home directory, then have a look at the permissions of various files in there.

Now let's go into your /data/atms305/a/netID/ATMS-305 directory and change the permissions of some of the files in there. Make sure you use both the shorthand and longhand form for setting permissions and that you also use a variety of absolute and relative paths. Try removing the read permission from a file then reading it. Or removing the write permission and then opening it in pico.

Let's play with directories now. Create a directory and put some files into it. Now play about with removing various permissions from yourself on that directory and see what you can and can't do.

Finally, have an explore around the system and see what the general permissions are for files in other system directories such as /etc and /bin.