Princeton University
COS 217: Introduction to Programming Systems
A Linux File Sharing Trick

This document describes Linux file and directory permissions, and a common trick to use them to share files with another user on a Linux system. We encourage you to use the trick to share Assignment 5 files with your partner.

Linux File/Directory User Classes

On a Linux system, each file or directory has three classes of users: *owner*, *group*, and *others*:

- A file/directory has an **owner**. A file/directory's owner is the user who created it. A file/directory's owner can issue `chmod` commands to change its permissions.
- A file/directory has a **group**. A group is a set of users. The owner of the file/directory can issue `chgrp` commands to change a file/directory's group to any other group of which the owner is a member.
- **Others** are everyone else!

You can issue the `id` command to determine the groups to which you belong. For example, consider this command:

```
courselab01:~/demo/linux$ id
uid=42579(rdondero) gid=33 groups=33
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

The output indicates that the user's id is `42579`, which has the name `rdondero`. The user `rdondero` belongs to one group whose id is `33`.

The groups to which you belong are determined by the Linux system administrators. An ordinary user cannot change the groups to which he/she belongs.

Linux File Permissions

On a Linux system each file has *read*, *write*, and/or *execute* permissions.

- If a file has **read** permissions for its owner, its group, or others, then its owner, its group, or others can *examine* the contents of a file (via `cat`, `more`, `less`, `xxd`, `emacs`, etc.).
- If a file has **write** permissions for its owner, its group, or others, then its owner, its group, or others can *change* the contents of that file (via `emacs`, etc.).
- If a file has **execute** permissions for its owner, its group, or others, then its owner, its group, or others can *execute* that file as a command. It makes sense to give a file execute permissions if and only if it contains executable code: executable binary code, a Bash shell script, a Python script, etc.

You can issue the `ls -al` command (aliased to `ll`) to determine the owner, group, and permissions of your files. For example, consider this command:

```
courselab01:~/demo/linux$ ll
total 147
drwx------. 2 rdondero 33   76 Sep 13 00:56 .
drwx------. 9 rdondero 33  165 Sep 13 00:55 ..
-rw-------. 1 rdondero 33   71 Sep 13 01:02 mydata.txt
-rwx------. 1 rdondero 33 6412 Sep 13 01:02 mypgm
-rw-------. 1 rdondero 33   80 Sep 13 01:02 mypgm.c
```

The first boldfaced line of the output indicates that:

- The working directory contains a file/directory named `mydata.txt`.
- The owner of `mydata.txt` is `rdondero`.
- The group of `mydata.txt` is `33`.
- The `mydata.txt` file has permissions that are indicated by the `-rw-------` permission string.

The `-rw-------` permission string is interpreted as follows:

- The first character (`-`) indicates that `mydata.txt` is a file, not a directory.
- The next three characters (`rw-`) indicate that owner `rdondero` has read and write permissions, but not execute permissions.
- The next three characters (`---`) indicate that the group `33` has no permissions.
- The next three characters (`---`) indicate that others have no permissions.

The second boldfaced line of the output indicates that:

- The working directory contains a file/directory named `mypgm`.
- The owner of `mypgm` is `rdondero`.
- The group of `mypgm` is `33`.
- The `mypgm` file has permissions that are indicated by the `-rwx------` permission string.

The `-rwx------` permission string is interpreted as follows:

- The first character (`-`) indicates that `mypgm` is a file, not a directory.
- The next three characters (`rwx`) indicate that owner `rdondero` has read, write, and execute permissions.
- The next three characters (`---`) indicate that the group `33` has no permissions.
- The next three characters (`---`) indicate that others have no permissions.

You can issue the `chmod` command to change file permissions. For example, consider this command:

```
chmod 644 mydata.txt
```

To understand that command, think of `644` as an octal (that is, a base 8) number. Then convert it to binary, yielding this result:

```
110100100
```

Then convert that binary number to a permission string using this approach: by position, consider each 1 to indicate the presence of a permission, and each 0 to indicate the absence of a permission. This is the result:

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

So that `chmod` command gives `rw-r--r--` permissions to the `mydata.txt` file. A subsequent `ll` command confirms that:

```
courselab01:~/demo/linux$ ll
total 147
drwx------. 2 rdondero 33   76 Sep 13 00:56 .
drwx------. 9 rdondero 33  165 Sep 13 00:55 ..
-rw-r--r--. 1 rdondero 33   71 Sep 13 01:02 mydata.txt
-rwx------. 1 rdondero 33 6412 Sep 13 01:02 mypgm
-rw-------. 1 rdondero 33   80 Sep 13 01:02 mypgm.c
```

Issuing the command:

```
chmod 600 mydata.txt
```

changes the permissions of `mydata.txt` back to `rw-------`.

Linux Directory Permissions

On a Linux system each directory, like each file, has *read*, *write*, and/or *execute* permissions. The key is to think of a directory as a table of file and directory names:

- If a directory has **read** permissions for its owner, its group, or others, then its owner, its group, or others can *examine* the table, that is, can find out what files are in the directory by issuing a `ls` command.
- If a directory has **write** permissions for its owner, its group, or others, then its owner, its group, or others can *change* the table, that is, can create new files/directories in the directory, remove files/directories from the directory, or rename files/directories in the directory.
- If a directory has **execute** permissions for its owner, its group, or others, then its owner, its group, or others can *visit* the table, that is, can `cd` to that directory. If a directory also has **read** permissions for its owner, its group, or others, then its owner, its group, or others can *copy* files from that directory.

You can issue the `ls -al` command (aliased to `ll`) to determine the owner, group, and permissions of your directories. For example, consider this command:

```
courselab01:~/demo/linux$ ll
total 147
drwx------. 2 rdondero 33   76 Sep 13 00:56 .
drwx------. 9 rdondero 33  165 Sep 13 00:55 ..
-rw-------. 1 rdondero 33   71 Sep 13 01:02 mydata.txt
-rwx------. 1 rdondero 33 6412 Sep 13 01:02 mypgm
-rw-------. 1 rdondero 33   80 Sep 13 01:02 mypgm.c
```

The boldfaced line indicates that the working directory ("."):

- Is owned by `rdondero`.
- Has group `33`.
- Has permissions that are indicated by the permission string `drwx------`

The `drwx------` permission string is interpreted as follows:

- The first character (`d`) indicates that "." is a directory, not a file.
- The next three characters (`rwx`) indicate that owner `rdondero` has read, write, and execute permissions.
- The next three characters (`---`) indicate that the group `33` has no permissions.
- The next three characters (`---`) indicate that others have no permissions.

You can issue the `chmod` command to change directory permissions. For example, consider this command:

```
chmod 711 .
```

Convert the octal number 711 to binary:

```
111001001
```

And then convert that binary number to a permission string:

```
rwx--x--x
```

So that `chmod` command gives `rwx--x--x` permissions to the working directory. A subsequent `ll` command confirms that:

```
courselab01:~/demo/linux$ ll
total 147
drwx--x--x. 2 rdondero 33   76 Sep 13 00:56 .
drwx------. 9 rdondero 33  165 Sep 13 00:55 ..
-rw-------. 1 rdondero 33   71 Sep 13 01:02 mydata.txt
-rwx------. 1 rdondero 33 6412 Sep 13 01:02 mypgm
-rw-------. 1 rdondero 33   80 Sep 13 01:02 mypgm.c
```

The command:

```
chmod 700 .
```

gives `rwx------` permissions to the working directory, thus changing its permissions back to their original values. A `ll` command confirms that:

```
courselab01:~/demo/linux$ ll
total 147
drwx------. 2 rdondero 33   76 Sep 13 00:56 .
drwx------. 9 rdondero 33  165 Sep 13 00:55 ..
-rw-------. 1 rdondero 33   71 Sep 13 01:02 mydata.txt
-rwx------. 1 rdondero 33 6412 Sep 13 01:02 mypgm
-rw-------. 1 rdondero 33   80 Sep 13 01:02 mypgm.c
```

Interaction of File and Directory Permissions

File/directory permissions are subject to permissions on the parent directories. For example, consider the file `/u/rdondero/demo/linux/mydata.txt`. If I wanted to allow others to read that file, then I would give `644` permissions to the `mydata.txt` file, so others have read permission on that file. But that would not be enough. I also would need to give:

- `711` permissions to the `linux` directory so others have execute permission and thereby can visit it.
- `711` permissions to the `demo` directory so others have execute permission and thereby can visit it.
- `711` permissions to the `rdondero` directory so others have execute permission and thereby can visit it.

The `u` directory already has `755` permissions, and the `/` directory already has `555` permissions.

Sharing Files

Suppose Alice (userid `alice`) and Bob (userid `bob`) are Assignment 5 partners and want to share some files on CourseLab. How should they proceed? There are (at least) three approaches:

- Alice asks the CourseLab system administrators to create a Linux group containing Alice's userid and Bob's userid. Alice and Bob use group-level permissions to control access. That's a good approach. However, it requires intervention of CourseLab system administrators. It would not be reasonable for them to handle many requests for new groups.

- Alice and Bob use a multi-programmer version control system, such as Git, Subversion, CVS, or Mercurial. That's a great approach, but it's beyond the scope of COS 217.

  If you do use a multi-programmer version control system, then make sure that its repository is not public. That is, make sure that its repository is readable by you and your partner only. If your repository is public

and another COS 217 student (in the current semester or some future one) copies your files, then you will be accused of abetting plagiarism, will be required to appear before the Committee on Discipline, and probably will be placed on academic probation. Please be careful.

- Alice and Bob use the trick described in the next section.

A File Sharing Trick

Alice and Bob perform these steps:

(1) Alice issues a `cd` command to make her home directory her working directory. She issues a `ll` command to examine the permissions of the files and directories in her home directory. She repeatedly issues `chmod 600 fileName` commands so each non-executable file in her home directory has 600 permissions, and `chmod 700 fileOrDirName` commands so each executable file and directory in her home directory has 700 permissions.

(2) Alice issues a `chmod 711 .` command to change the permissions of her home directory such that she has read. write, and execute permissions, and her group and others have only execute permissions. So her group and others can `cd` to her home directory, but cannot `ls` it.

(3) Alice issues a `head -c4 /dev/random | xxd -g4 | cut -b10-17` command to generate a random character sequence. She issues a `mkdir shareCRYPTIC` command (where *CRYPTIC* is that random character sequence) to create a directory whose name is `share` followed by that random character sequence.

(4) Alice issues a `chmod 777 shareCRYPTIC` command to change the permissions of the `shareCRYPTIC` directory so it has read, write, and execute permissions for everyone.

(5) Alice issues a `cd shareCRYPTIC` command to make `shareCRYPTIC` her working directory. She creates files in her `shareCRYPTIC` directory, and issues `chmod 666 filename` commands so each non-executable file has read and write permissions for everyone, and `chmod 777 filename` commands so each executable file has read, write, and execute permissions for everyone.

(6) Alice tells Bob (but nobody else!) the name of her `shareCRYPTIC` directory.

(7) Bob issues a `cd /u/alice/shareCRYPTIC` command to make the `shareCRYPTIC` directory his working directory. (Note that anyone who knows the name of the directory can do that, but only Bob and Alice know that name.) He creates and edits files in the `shareCRYPTIC` directory, and issues `chmod 666 fileName` commands so each non-executable file has read and write permissions for everyone, and `chmod 777 fileName` commands so each executable file has read, write, and execute permissions for everyone.

IMPORTANT: After completing the assignment...

(8) Bob issues a `cd` command, a `mkdir asgt5` command, a `cd asgt5` command, and a `cp /u/alice/shareCRYPTIC/* .` command to copy all files from the `shareCRYPTIC` directory to his `asgt5` directory. Thereby he has copies of the files for future reference.

(9) Alice issues a `cd` command and a `chmod 700 .` command to change the permissions of her home directory such that she has read, write, and execute permissions, and her group and others have no permissions.