03 - Manipulating Files and Using Git

CS 2043: Unix Tools and Scripting, Spring 2016 [1]

Stephen McDowell February 1st, 2016

Cornell University

Table of contents

- 1. Working with Files
- 2. Types of Files and Usages
- 3. Let's Git Started
- 4. Demo Time!

Some Logistics

- · Last day to add is Wednesday 2/3.
- · HW0: Due today at 5pm.
- · My OH are Tuesdays 6:00pm 7:00pm, Gates G19.
- On moving forward independently, and using sudo.
 - I strongly advise taking a *snapshot* of your VM.
- · A note about HW1...

Working with Files

Users and Groups

Like most OS's, Unix allows multiple people to use the same machine at once. The question: who has access to what?

- · Access to files depends on the users' account.
- All accounts are presided over by the Superuser, or root account.
- Each user has absolute control over any files they own, which can only be superseded by root.
- Files can also be owned by a **group**, allowing more users to have access.

File Ownership

You can discern who owns a file many ways, the most immediate being ls -l

Permissions with **ls**

The third column is the *user*, and the fourth column is the *group*.

What is this RWX Nonsense?

- R = read, W = write, X = execute.
- rwxrwxrwx
 - · User permissions.
 - · Group permissions.
 - Other permissions (a.k.a. neither the owner, nor a member of the group).
- Directory permissions begin with a **d** instead of a -.

An example

What would the permissions - rwxr---- mean?

- · It is a file.
- User can read and write to the file, as well as execute it.
- Group members are allowed to read the file, but cannot write to or execute.
- Other cannot do anything with it.

Changing Permissions

Change Mode

chmod <mode> <file>

- Changes file / directory permissions to <mode>.
- The format of <mode> is a combination of three fields:
 - Who is affected: a combination of **u**, **g**, **o**, or **a** (all).
 - Use a + to add permissions, and a to remove.
 - Specify type of permission: any combination of r, w, x.
- Or you can specify mode in octal: user, then group, then other.
 - e.g. 777 means user=7, group=7, other=7 permissions.

The octal version can be confusing, but will save you time. Excellent resource in [2].

Changing Ownership

Changing the group

Change Group

chgrp group <file>

- Changes the group ownership of <file> to group.

As the super user, you can change who owns a file:

Change Ownership

chown user:group <file>

- Changes the ownership of **<file>**.
- The **group** is optional.
- The -R flag is useful for recursively modifying everything in a directory.

File Ownership, Alternate

If you are like me, you often forget which column is which in ls -l...

Status of a file or filesystem

stat [opts] <filename>

- Gives you a wealth of information, generally more than you will every actually need.
- **Uid** is the user, **Gid** is the group.
 - BSD/OSX: use **stat** -**x** for standard display of this command.
- Can be useful if you want to mimic file permissions you don't know.
 - Human readable: --format=%A, e.g. -rw-rw-r--
 - BSD/OSX: -f %Sp is used instead.
 - Octal: --format=%a (great for chmod), e.g. 664
 - BSD/OSX: -f %A is used instead.

Platform Notes

· Convenience flag for **chown** and **chmod** on non-BSD Unix:

```
>>> chmod --reference=<src> <dest>
```

- Set the permissions of dest to the permissions of src!
- BSD/OSX users: --reference does not exist, you will have to execute two commands.

```
>>> chmod `stat -f %A <src>` <dest>
```

- The stat command inside of the `backticks` gets evaluated before chmod does.
- The **stat** command performs a little differently on BSD/OSX by default. Read the **man** page.

Types of Files and Usages

Plain Files

Plain text files are human-readable, and are usually used for things like:

- · Documentation,
- Application settings,
- Source code,
- · Logs, and
- Anything you may want to read via the terminal (e.g. README.txt).

Binary Files

Binary files are not human-readable. They are written in the language your computer prefers.

- · Executables,
- · Libraries,
- · Media files,
- · Archives (.zip, etc), and many more.

Reading Files Without Opening

Concatenate

cat <filename>

- Prints the contents of the file to the terminal window

- Prints file1 first, then file2.

more

more <filename>

- Scroll through one page at a time.
- Program exits when end is reached.

less

less <filename>

- Scroll pages or lines (mouse wheel, space bar, and arrows).
- Program does not exit when end is reached.

Beginning and End

Long files can be a pain with the previous tools.

Head and **Tail** of Input

```
head -[numlines] <filename>
tail -[numlines] <filename>
```

- Prints the first / last numlines of the file.
- Default is 10 lines.

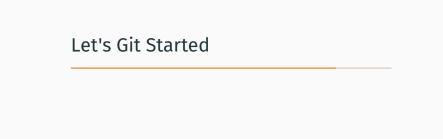
Not Really a File...YET

You can talk to yourself in the terminal too!

Echo

echo <text>

- Prints the input string to the standard output (the terminal).
- We will soon learn how to use **echo** to put things into files, append to files, etc.



Another Brief Git Demo

If you are not at lecture, don't worry about this slide not making any sense.

```
>>> git clone <url>  # get a local copy
>>> git status  # informs you of changes
>>> git add <file(s)> # if you need it online
>>> git commit  # saves this version
>>> git push  # puts the commit online
```

Demo Time!

Our first in class demo

Instructions are here:

https://github.com/cs2043-sp16/lecture-demos/tree/master/lec03

References I

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.

Previous cornell cs 2043 course slides.

[2] C. Hope.

2016.

Linux and unix chmod command help and examples. http://www.computerhope.com/unix/uchmod.htm,

23