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

Some Logistics

- Last day to add is Wednesday 2/3.

Some Logistics

- Last day to add is Wednesday 2/3.
- HW0: Due today at 5pm

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

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 the usage of `sudo`

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 the usage of `sudo`
  - I strongly advise taking a *snapshot* of your VM

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 the usage of `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

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

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- Each user has absolute control over any files they own, which can only be superseded by `root`

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- Each user has absolute control over any files they own, which can only be superseded by `root`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`

```
> ls -l Makefile
-rw-rw-r--. 1 sven users 4.9K Jan 31 04:42 Makefile
              sven       # the user
                   users # the group
```

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

What is this RWX Nonsense?

R = read, W = write, X = execute

-rwxrwxrwx

- User permissions

What is this RWX Nonsense?

R = read, W = write, X = execute

-rwxrwxrwx

- User permissions
- Group permissions

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)

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?

An example

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

- It is a file.

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

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

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)
  - Whether adding or removing permissions; add with `+`, remove with `-`
  - Which permissions are being modified - 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

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>`

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

Change Ownership

`chown user:group <file>`

- Changes the ownership of `<file>`
- `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 need
- `Uid` is the user, `Gid` is the group
- Can be useful if you want to mimic file permissions you don't know
  - `--format=%A`: human readable, e.g. `-rw-rw-r--`
  - `--format=%a`: octal (great for `chmod`), e.g. `664`

Platform Notes I

Convenience flag for `chown` and `chmod` on non-BSD Unix

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

It will set the permissions of `dest` to the permissions of `src`!
Mac users: sorry :/

The `stat` on BSD: the `--format` does not exist, it is just `-f`.
The options seem to be the same, but read the man page.

Platform Notes II

The `stat` command performs a little differently on OSX by default. For example, on the `Makefile` it produces this giant wall (on one line, continued for presentation purposes):

```
> stat Makefile
> 16777218 6517959 -rw-r--r-- 1 sven staff 0 4945
    "Feb  1 11:48:14 2016" "Jan 31 07:02:42 2016"
    "Jan 31 08:28:22 2016" "Jan 31 07:02:42 2016"
    4096 16 0 Makefile
```

To get more useful output for the intended purpose of `stat` in how I am presenting it, you need to do `stat -x Makefile`. This will print out the `Uid` and `Gid` for you.

Types of Files and Usages

Plain Files

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

- Documentation

Plain Files

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

- Documentation
- Application settings

Plain Files

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

- Documentation
- Application settings
- Source code

Plain Files

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

- Documentation
- Application settings
- Source code
- Logs

Plain Files

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

- Documentation
- Application settings
- Source code
- Logs
- 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

Binary Files

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

- Executables
- Libraries

Binary Files

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

- Executables
- Libraries
- Media files

Binary Files

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

- Executables
- Libraries
- Media files
- Archives (.zip, etc)

Reading Files Without Opening

Print a file to the screen

`cat <filename>`

- Prints the contents of the file to the terminal window

`cat <file1> <file2>`

- Prints `file1` first, then `file2`.

more

`more <filename>`

- Scroll through one page at a time

less

`less <filename>`

- Scroll by pages or lines (mouse wheel, space bar, and arrows)

Beginning and End

Long files can be a pain with the previous tools.

Head and Tail

```
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

Let's Git Started

Another Brief Git Demo

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

```
> git clone <url>
> git status
> git add <file(s)>
> git commit
> git push
```

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.
Linux and unix chmod command help and examples.
`http://www.computerhope.com/unix/uchmod.htm`, 2016.