02 - The Unix File System, First Glimpse at Git

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

Stephen McDowell

January 29th, 2016

Cornell University

Table of contents

Some Logistics

- HW0: You need GitHub. And a Unix environment.

Some Logistics

- HW0: You need GitHub. And a Unix environment.
- Getting Started page updated with some videos, 32bit option available.

Some Logistics

- HW0: You need GitHub. And a Unix environment.
- Getting Started page updated with some videos, 32bit option available.
- (Poll) OH scheduling. Thanks Jerome!

Some Logistics

- HW0: You need GitHub. And **MUST HAVE** a Unix environment.
- Getting Started page updated with some videos, 32bit option available.
- (Poll) OH scheduling. Thanks Jerome!

Notation

Commands will be shown on slides using `teletype text`.

Introducing new commands

New commands will be introduced in block boxes like this one. A summary of calling the command will be shown listing the command name and potential optional arguments / flags.

some-command [opt1] [opt2]

To execute a command, just type its name into the shell and press return / enter.

Unix Filesystem Overview

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`
- Directories are separated by `/` instead of `\`

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`
- Directories are separated by `/` instead of `\`
  - UNIX: `/home/sven/lemurs`

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`
- Directories are separated by `/` instead of `\`
  - UNIX: `/home/sven/lemurs`
  - Windows: `E:\Documents\lemurs`

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`
- Directories are separated by `/` instead of `\`
  - UNIX: `/home/sven/lemurs`
  - Windows: `E:\Documents\lemurs`
- Hidden files and folders begin with a `"."`

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`
- Directories are separated by `/` instead of `\`
  - UNIX: `/home/sven/lemurs`
  - Windows: `E:\Documents\lemurs`
- Hidden files and folders begin with a "`.`"
  - e.g. `.git/` (a hidden directory)

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume)
  - The root directory is just `/`
- All files and directories are case sensitive
  - `hello.txt != hElLo.TxT`
- Directories are separated by `/` instead of `\`
  - UNIX: `/home/sven/lemurs`
  - Windows: `E:\Documents\lemurs`
- Hidden files and folders begin with a "`.`"
  - e.g. `.git/` (a hidden directory)
- Example: my home directory

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives
- `/usr`: Mostly user-installed programs and amenities

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives
- `/usr`: Mostly user-installed programs and amenities
- `/etc`: System-wide settings

What's Where: Programs Edition

Programs are usually installed in one of the "binaries" directories:

- `/bin`: System programs

What's Where: Programs Edition

Programs are usually installed in one of the "binaries" directories:

- `/bin`: System programs
- `/usr/bin`: Most user programs

What's Where: Programs Edition

Programs are usually installed in one of the "binaries" directories:

- `/bin`: System programs
- `/usr/bin`: Most user programs
- `/usr/local/bin`: A few other user programs

Personal Files

Your personal files are in your home directory (and its subdirectories), which is *usually*[*] located at

Linux	Mac
`/home/username`	`/Users/username`

There is also a built-in alias for it: `~`

For example, the Desktop for the user **`sven`** is located at

Linux	Mac
`/home/sven/Desktop`	`/Users/sven/Desktop`
`~/Desktop`	`~/Desktop`

Basic Navigational Commands

Where am I?

Most shells default to using the current path in their prompt. If not

Print working directory

`pwd`

- prints the "full" path of the current directory
- handy on minimalist systems when you get lost
- can be used in scripts

What's here?

Knowing where you are is useful, but understanding what (or who?) else is there is too...

The list command

`ls`

- lists directory contents (including subdirectories)
- works like the dir command in Windows
- the `-l` flag lists detailed file / directory information (we'll learn more about flags later).
- use `-a` to list hidden files

Ok lets go!

Moving around is as easy as

Changing directories

`cd [directory name]`

- changes directory to `[directory name]`
- if not given a destination defaults to the user's home directory
- you can specify both absolute and relative paths

  - Absolute paths start at `/`
    - e.g. `cd /home/sven/Desktop`
  - Relative paths start at the current directory
    - e.g. `cd Desktop`, if you were already at `/home/sven`

Relative Path Shortcuts

Shortcuts

`~`	current user's home directory
`.`	the current directory (this is actually useful...)
`..`	the parent directory of the current directory
`-`	for `cd` command, return to previous working directory

An example: starting in `/usr/local/src`

```
> cd    # now at /home/sven
> cd -  # now at /usr/local/src
> cd .. # now at /usr/local
```

File and Folder Manipulation

Creating a new File

The easiest way to create an empty file is using

touch

`touch [flags] <file>`

- adjusts the timestamp of the specified file
- with no flags uses the current date and time
- if the file does not exist, `touch` creates it

File extensions (`.txt`, `.c`, `.py`, etc) often **don't** matter in Unix. Using `touch` to create a file results in a blank plain-text file (so you don't necessarily have to hadd `.txt` to it).

Creating a new Directory

No magic here...

Make directory

`mkdir [flags] <directory1> <directory2> <...>`

- can use relative or absolute paths
  - a.k.a. you are not restricted to making directorys in the current directory only
- need to specify at least one directory name
- can specify multiple, separated by spaces

File Deletion

Warning: once you delete a file (from the command line) there is no easy way to recover the file.

Remove File

`rm [flags] <filename>`

- removes the file `<filename>`
- using wildcards (more on this later) you can remove multiple files
  - `rm *` - removes every file in the current directory
  - `rm *.jpg` - removes every `.jpg` file in the current directory
- `rm -i <filename>` - prompt before deletion

Deleting Directories

By default, `rm` cannot remove directories. Instead we use...

Remove directory

`rmdir [flags] <directory>`

- removes an **empty** directory
- throws an error if the directory is not empty

To delete a directory and all its subdirectories, we pass `rm` the flag `-r` (for recursive), e.g. `rm -r /home/sven/oldstuff`

Copy That!

Copy

`cp [flags] <file> <destination>`

- copies from one location to another
- to copy multiple files, use wildcards (such as `*`)
- to copy a complete directory use `cp -r <src> <dest>`

Move it!

Unlike the `cp` command, the move command automatically recurses for directories.

Move

`mv [flags] <source> <destination>`

- moves a file or directory from one place to another
- also used for renaming, just move from `<oldname>` to `<newname>`

Recap

- `ls` - list directory contents
- `cd` - change directory
- `pwd` - print working directory
- `rm` - remove file
- `rmdir` - remove directory
- `cp` - copy file
- `mv` - move file

Flags & Command Clarifaction

Flags and Options

Most commands take flags and optional arguments. These come in two general forms: switches (no argument required), and argument specifiers for lack of a better name.

When specifying flags for a given command, these will modify the behavior of the command and how it executes. Some flags take precedence over others, and some flags you specify can implicitly pass additional flags to the command.

Flags and Options: A bad Analogy

If you think of a command as a computer, you could think of the flags as the different hardware components installed. Let's say that in this case a hard drive is a flag.

The computer shipped to you with a CPU, motherboard, hard drive, etc and installed on that hard drive was the original operating system (say Windows). When you start it, the computer was executed with the Windows flag.

Now, you remove the original hard drive and insert another hard drive that has a different OS installed (say Fedora). Then you boot your computer, only this time you ended up passing the Fedora flag.

Nothing about the other components of the computer changed (it's just a bunch of electricity being routed around), but the behavior changed because of the flag you passed.

Flags and Options: Formats

A flag that is

- One letter is specified with a single dash (`-a`)
- More than one letter is specified with two dashes (`--all`)

The reason is because of how switches can be combined (next page).

Flags and Options: Switches

Switches take no arguments, and can be specified in a couple of different ways. Switches are usually one letter, and multiple letter switches usually have a one letter alias (`--all` has the `-a` alias).

- One option:
  - `ls -a`
  - `ls --all`
- Two options:
  - `ls -l -Q`
- Two options:
  - `ls -lQ`
- Applied from left to right:
  - `rm -fi <file>` $\Rightarrow$ prompts
  - `rm -if <file>` $\Rightarrow$ does *not* prompt

Flags and Options: Argument Specifiers

These flags expect an input, and you will encounter two general kinds.

- The `--argument="value"` format, where the `=` and quotes are needed if `value` is more than one word.
  - Yes: `ls --hide="Desktop" ~/`
  - Yes: `ls --hide=Desktop ~/`
    - one word, no quotes necessary
  - No: `ls --hide = "Desktop" ~/`
    - spaces by the `=` will be misinterpreted (it used `=` as the `hide` value...)
- The `--argument value` format, with a space after the `argument`. Quote rules same as above.
  - `ls --hide "Desktop" ~/`
  - `ls --hide Desktop ~/`

Note: The example I gave you was using the same `--hide` in both formats, but not *all* commands will accept both.

Advise `--argument="value"` format for higher success rates.

Flags and Options: Conventions, Warnings

Generally, you should always specify the flags before the arguments. In this example, the flag is `-l` and `~/Desktop/` is the argument.

- `ls -l ~/Desktop/` and `ls ~/Desktop/ -l` both work
- there exist scenarios in which flags after arguments do **not** get processed

There is a special sequence `--` that signals the end of the options. I will use another flag to demonstrate:

- `ls -l -a ~/Desktop/` ⇒ executes as expected
- `ls -l -- -a ~/Desktop/` ⇒ only used `-l`
  - `"ls: cannot access -a: No such file or directory"`
  - `-a` was treated as an *argument*, and there is no `-a` directory (for me)

Flags and Options: Conventions, Warnings (cont)

The special sequence `--` that signals the end of the options is often most useful if you need to do something special. Suppose I wanted to make the folder `-a` on my Desktop.

```
> cd ~/Desktop # for demonstration purpose
> mkdir -a     # fails: invalid option -- 'a'
> mkdir -- -a  # success! (ls to confirm)
> rmdir -a     # fails: invalid option -- 'a'
> rmdir -- -a  # success! (ls to confirm)
```

This trick can be useful in *many* scenarios, and generally arises when you need to work with special characters of some sort.

Your new best friend

How do I know what the flags / options for all of these commands are?

The manual command

`man <command_name>`

- Loads the manual (manpage) for the specified command
- Unlike ggogle, manpages are **system-specific**
- Usually very comprehensive. Sometimes *too* comprehensive
- Type `/<keyword>`
- The **n** key jumps through the search results

Search example on next page if that was confusing. Intended for side-by-side follow-along.

Man oh man

```
> man man  #  you now have the manual loaded
> /useful  #  type /useful, then hit enter
############ [first result highlighted]
> n        #  followed by enter
############ [next result highlighted]
```

Note that there are subtle differences between options on different systems. For example, `ls -B`:

- BSD/OSX: Force printing of non-printable characters in file names as `\xxx`, where `xxx` is the numeric value of the character in *octal*.
- Fedora, Ubuntu: do not list implied entries ending with ~
  - In these OS's, files ending with ~ are *temporary* backup files that certain programs generate

References I

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.
Previous cornell cs 2043 course slides.