02 – The Unix File System

CS 2043: Unix Tools and Scripting, Spring 2017 [1]

Stephen McDowell

January 27, 2017

Cornell University

Table of Contents

1. Unix Filesystem Overview
2. Basic Navigational Commands
3. File and Folder Manipulation
4. Flags & Command Clarification

Notation

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

Introducing New Commands

```
some-command [opt1] [opt2] <arg1> [arg2]
```

- New commands will be introduced in block boxes like this one
- `[brackets]` indicate *optional* items (flags / arguments)
- `<arg1>`: `arg1` is required
- `[arg2]`: command supports multiple arguments

- To execute `some-command`, just type its name into the shell and press return / enter.
- `$` in code-blocks indicate a new command being entered.

```
$ some-command
output of some-command (where applicable)
```

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 root directory is just /
- All files and directories are case sensitive.
    - `hello.txt != hElLo.TxT`
- Directories are separated by / in Unix instead of \ in Windows.
    - 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.
- `/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 *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, you can find out where you are with

Print Working Directory

`pwd`

- Prints the “full” path of the current directory.
    - The `-P` flag is needed when *symbolic* links are present.
- 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 else is there is too...

List Directory Contents

```
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

Change Directories

`cd [directory name]`

- Changes directory to `[directory name]`.
- If not given a destination defaults to the user's home directory.
  - The home directory is ~
- You can specify both absolute and relative paths.

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

Relative Path Shortcuts

- Relative path shortcuts worth remembering:

Shortcut	Expands To
`~`	current user's home directory
`.`	the current directory
`..`	the parent directory of the current directory
`-`	for `cd`, return to previous working directory

- An example:
  - `/usr/local/src` arbitrary choice, nothing special about it.
  - After each `cd` command, execute `pwd` to confirm.

```
$ cd /usr/local/src # go to starting location
$ 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

Change File Timestamps

```
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.
- "But I swear I haven't changed the file, look at the timestamp."
    - ... timestamps prove nothing.

- 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.
    - You don't have to add `.txt` if you don't want to.

Creating a new Directory

- No magic here...

Make Directories

```
mkdir [flags] <dir1> <dir2> <...> <dirN>
```

- Can use relative or absolute paths.
    - Not restricted to making directories in the current directory only.
- Need to specify at least one directory name.
- Can specify multiple, separated by spaces.
- The `-p` flag is commonly used in scripts:
    - Makes all parent directories if they do not exist.
    - Convenient because if the directory exists, `mkdir` will not fail.

File Deletion

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

Remove Files or Directories

```
rm [flags] <filename>
```

- Removes the file `<filename>`.
- Remove multiple files with wildcards (more on this later).
  - Remove every file in the current directory: `rm *`
  - Remove every `.jpg` file in the current directory: `rm *.jpg`
- Prompt before deletion: `rm -i <filename>`

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.
- You are encouraged to use this command: failing on non-empty can and will save you!

- To delete a directory and all its subdirectories, we pass `rm` the flag `-r` (for recursive)
    - `rm -r /home/sven/oldstuff`
    - THIS IS DANGEROUS!

Copy That!

Copy

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

- Copies from one location to another.
- To copy multiple files, use wildcards (such as `*`).
    - Globs / patterns can only be used for `<src>`.
    - `<dest>` must be explicit and singularly defined.
    - Completely reasonable...how would it know what to do if there is ambiguity in where to send the file(s)?
- To copy a complete directory: `cp -r <src> <dest>`

Move it!

- Unlike the `cp` command, the move command automatically recurses for directories.
  - Think of the implication of if it did not...

Move (or Rename) Files and Directories

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

- Moves a file or directory from one place to another.
- Also used for renaming, rename `<oldname>` to `<newname>`.
  - `mv badFolderName correctName`

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 Clarification

Flags and Options: A bad Analogy

- Think of a **command** as a computer. Then the **flags** could be thought of as the hardware installed.
    - Everything is already there: motherboard, hard drives, cpu, etc.
    - Let’s consider the *hard drive* “flag”.
- Say you have Windows installed on the hard drive.
    - When you boot the computer, you passed the “Windows” flag.
- Swap original hard drive for one with Fedora installed.
    - When you boot your computer, you passed the “Fedora” flag.
- None of the other components changed:
    - At the root: it’s just a bunch of electricity being routed around!
    - Same processor, motherboard, etc.
    - We only changed the “Operating System flag”

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, keep in mind:
    - Flags modify the behavior of the command / how it executes.
    - Some flags take precedence over others, and some flags you specify can implicitly pass additional flags to the command.
- There is no absolute rule here: research the command.

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.
- We generally use “flag” and “switch” interchangeably:
    - “flag” the command, telling it that “action X” should occur
    - specify to the command to “switch on/off action X”

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.
- One option:
  - `ls -a`
  - `ls --all`
- Two options:
  - `ls -l -Q`
  - `ls -lQ`
- *Usually* applied from left to right in terms of operator precedence, but not always:
  - This is up to the developer of the tool.
  - Prompts: `rm -fi <file>`
  - Does **not** prompt: `rm -if <file>`

Flags and Options: Argument Specifiers

- 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 argument to `hide`
- The `--argument value` format (space after the `argument`).
    - Quote rules same as above.
    - `ls --hide "Desktop" ~/`
    - `ls --hide Desktop ~/`
- Usually, `--argument value` and `--argument=value` are interchangeable.
    - Not always!

Flags and Options: Conventions, Warnings

- Generally, always specify the flags before the arguments.
- `ls -l ~/Desktop/` and `ls ~/Desktop/ -l` both work.
  - Sometimes flags after arguments **get ignored**.
  - Depends both on the command, and the flag(s).
- The special sequence `--` signals the end of the options.
  - Executes as expected: `ls -l -a ~/Desktop/`
  - Only uses `-l`: `ls -l -- -a ~/Desktop/`
    - `"ls: cannot access -a: No such file or directory`
    - The `-a` was treated as an **argument**, and there is no `-a` directory (for me)
- In this example:
  - `-l` and `-a` are the **flags**.
  - `~/Desktop/` is the **argument**.

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 google, manpages are `system-specific`.
- Usually very comprehensive. Sometimes *too* comprehensive.
- Type `/keyword` to search for `keyword`, and hit `<enter>`.
- The `n` key jumps to the next search result.

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

Man oh man

- The `man` command is really useful!

```
$ man man #  you now have the manual loaded
$ /useful #  type /useful, then hit enter
########### [[[ first result highlighted ]]]
$ n       #  followed by enter
########### [[[ next result highlighted ]]]
# The default 'pager' is `less`, type `q`
# without backticks to exit.
```

- Subtle differences depending on distribution, e.g. `ls -B`
- BSD/OSX: Force printing of non-printable characters in file names as `\xxx`.
  - `xxx` is the numeric value of the character in **octal**.
- GNU (Fedora, Ubuntu): don't list implied entries ending with ~
  - Files ending with ~ are *temporary* backup files that certain programs generate (e.g. some text-editors, your OS).

References

[1] Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years. “Previous Cornell CS 2043 Course Slides”.