04 – The Find command, editing, and scripting

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano

January 30, 2019

Cornell University

Table of Contents

The **`find`** Command

If you Leave this Class with Anything...

- Quite possibly the most underrated tool for your terminal:
  - `find`: searching for files / directories by name or attributes.

Finding Yourself

Search for Files in a Directory Hierarchy

```
find [where to look] criteria [what to do]
```

- Used to locate files or directories.
- Search any set of directories for files that match a criteria.
- Search by name, owner, group, type, permissions, last modification date, and *more*.
    - Search is recursive (will search all subdirectories too).
        - Sometimes you may need to limit the depth.
- Comprehensive & flexible. Too many options for one slide.

Some Useful Find Options

- `-name`: name of file or directory to look for.
- `-maxdepth num`: search at most `num` levels of directories.
- `-mindepth num`: search at least `num` levels of directories.
- `-amin n`: file last access was `n` minutes ago.
- `-atime n`: file last access was `n` days ago.
- `-group name`: file belongs to group `name`.
- `-path pattern`: file name matches shell pattern `pattern`.
- `-perm mode`: file permission bits are set to `mode`.

Of course...a lot more in `man find`.

Some Details

- This command is extremely powerful...but can be a little verbose (both the output, and what you type to execute it). That's normal.
- Modifiers for `find` are evaluated in conjunction (a.k.a AND).
- Can condition your arguments with an OR using the `-o` flag.
  - Must be done *for each* modifier you want to be an OR.
- Can execute command on found files / directories by using the `-exec` modifier, and `find` will execute the command for you.
  - The variable name is `{}`.
  - You have to end the command with either a
    - Semicolon (`;`): execute command *on each* result as you *find* them.
    - Plus (`+`): *find* all results first, *then* execute command.
    - Warning: have to escape them, e.g. `\;` and `\+`
    - The `;` and `+` are shell expansion characters!

Basic Examples

Find all files accessed at most 10 minutes ago

```
find . -amin -10
```

Find all files accessed at least 10 minutes ago

```
find . -amin +10
```

Comparing AND vs OR behavior

```
find . -type f -readable -executable
```

- All files that are *readable* **and** *executable*.

```
find . -type f -readable -o -executable
```

- All files that are *readable* **or** *executable*.

Display all the contents of files accessed in the last 10 minutes

```
find . -amin -10 -exec cat {} \+
```

On a Mac and ended up with .DS_Store Everywhere?

```
find . -name ".DS_Store" -exec rm -f {} \;
```

- Could be `;` or `+` since `rm` allows multiple arguments.

Solve maze in one line

Maze in 2 seconds

```
find / -iname victory -exec handin maze {} \+
```

- imagine how much more complicated `maze` could get in the real world...

More Involved Example

- Your boss asks you to backup all the logs and send them along.
- Combining some of the things we have learned so far (also zip)

```
# Become `root` since `/var/log` is protected:
$ sudo su
<enter password for your user>
# Make a containment directory to copy things to
$ mkdir ~/log_bku
# `find` and copy the files over in one go
$ find /var/log -name "*.log" -exec cp {} ~/log_bku/ \;
# The `cp` executed as `root`, so we cannot read them.
$ chown -R mpm288 ~/log_bku # My netID is mpm288
# Give the folder to yourself.
$ mv ~/log_bku /home/mpm288/
# Become your user again
$ exit
# Zip it up and send to your boss
$ zip -r log_bku.zip ~/log_bku
```

More Involved Example: Analysis

- Don't *have* to be `root`: `sudo find` was too long for slides.
  1. Make the directory `<dir>` as normal user.
  2. `sudo find ... -exec cp {} <dir> \;`
  3. `sudo chown -R <you> <dir>`
  4. `zip -r <dir>.zip <dir>`
- **Cannot** use `\+` instead of `\;` in this scenario:
  - Suppose you found `/var/log/a.log` and `/var/log/b.log`.
  - Executing with `\;` (`-exec` as you `find`):
    1. `cp /var/log/a.log ~/log_bku/`
    2. `cp /var/log/b.log ~/log_bku/`
  - Executing with `\+` (`find` all first, then `-exec` once):
    - `cp /var/log/a.log /var/log/b.log ~/log_bku/`
    - `cp` gets mad: you gave three arguments

Scripting

What is a Script?

- The high-level story is: nothing special.
  - Just a sequence of operations being performed.
  - Runs from top to bottom.
- Common practice:
  - Executable filetype.
  - Shebang.

Bash Scripting at a Glance

```
#!/bin/bash
echo "hello world!"
echo "There are two commands here!"
```

```
#!/usr/bin/python3

print('hello there friend');
```

- The *shebang* `#!/bin/bash` is the interpreter
- Run a command or two!
- Always test your scripts!

```
#!/bin/bash
#this is a comment. Maze solution script!
find / -iname victory -exec handin maze {} \+
```

Some execution details

- Run your scripts by providing a *qualified path* to them.
  - path must start with a folder
  - Current directory? use `./scriptname`
  - somewhere else? specify the path to your script
- Scripts execute from top to bottom.
- This is just like Python, for those of you who know it already.
- Bad code? you may only realize it when (and if) the script reaches that line
- The script starts at the top of the file.
- Execution continues down until the bottom (or `exit` called).
  - Broken statement? It still keeps executing the subsequent lines.

Text Editors

Nano, and VIM vs Emacs

- There is a great and ancient war among the *NIXfolk ... long has it raged, and ever shall it burn.
- To use VIM, or to use emacs?
- I will (try to) teach both.
    - But the easiest editor is nano
- **NANO:** the OG notepad
- **VIM:** *mode*-based editor
- **EMACS:** *hotkey*-based editor

Your friend Nano

Edit files like it's 1989

`nano file`

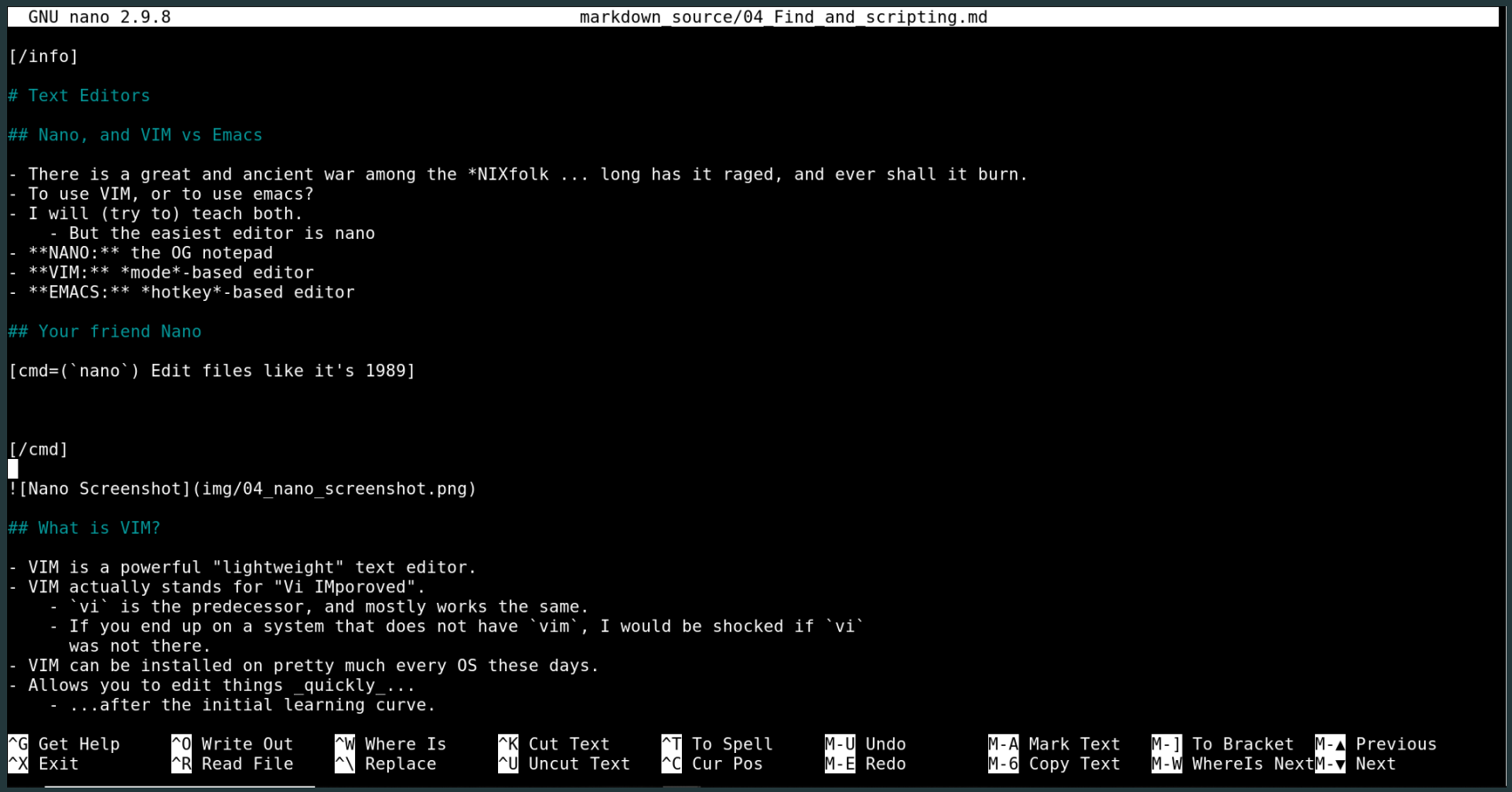

Figure 1: Nano Screenshot

What is VIM?

Edit files like it's 1976. or 1991.

```
vim file
```

- VIM is a powerful "lightweight" text editor.
- VIM actually stands for "Vi IMporoved".
  - `vi` is the predecessor, and mostly works the same.
  - If you end up on a system that does not have `vim`, try vi.
    - if no vi, try nano
- VIM can be installed on pretty much every OS these days.
- Allows you to edit things *quickly*...
  - ...after the initial learning curve.

The 3 Main Modes of VIM

- Normal Mode:
  - Launching pad to issue commands or go into other modes.
  - Can view the text, but not edit it directly (only through commands).
  - Return to normal mode *from other modes*: press **ESCAPE**
- Visual Mode:
  - Used to highlight text and perform block operations.
  - Enter visual mode *from normal mode*: press `v`
    - Visual Line: `shift+v`
    - Visual Block: `ctrl+v`
    - Explanation: try them out, move your cursor around...you'll see it.
- Insert Mode:
  - Used to type text into the buffer (file).
  - Like any regular text-editor you've seen before.
  - Enter *from normal mode*: press `i`

Moving Around VIM

- Most of the time, you can scroll with your mouse / trackpad.
- You can also use your arrow keys.
- VIM shortcuts exist to avoid moving your hands at all. Use
  - `h` to go left.
  - `j` to go down.
  - `k` to go up.
  - `l` to go right.
- Hardcore VIM folk usually map left caps-lock to be **ESCAPE**.
  - Goal: avoid moving your wrists at all costs. Arrows are so far!
  - I don't do this. I also don't use VIM.

Useful Commands

`:help`	help menu, e.g. specify `:help v`
`:u`	undo
`:q`	exit
`:q!`	exit without saving
`:e [filename]`	open a different file
`:syntax [on/off]`	enable / disable syntax highlighting
`:set number`	turn line numbering on
`:set nonumber`	turn numbering off (e.g. to copy paste)
`:set spell`	turn spell checking on
`:set nospell`	turn spell checking off
`:sp`	split screen horizontally
`:vsp`	split screen vertically
`<ctrl+w> <w>`	rotate between split regions
`:w`	save file
`:wq`	save file and exit
`<shift>+<z><z>`	alias for `:wq` (hold shift and hit `z` twice)

WOW How about no. let's see Emacs

- Basic editing works like notepad (except no mouse)
- No switching between modes to edit/search/save/etc.
- Emacs can also be installed on pretty much every OS.
- Allows you to edit things *moderately* quickly...
    - ...and keeps getting faster as you learn it

Emacs modes

An editor, also from 1976.

```
emacs file
```

- Based on file and action type
  - Java file detected? IDE mode engaged!
  - Plain file detected? Basic edit mode engaged!
  - LaTeX file detected? TeXstudio mode!
- Shortcuts and actions *mostly* independent of mode
  - But modes hide a lot of power...
  - Sometimes accused of being a whole OS.

Moving around and basic editing:

- move by character? Use the arrow keys!
- move by word? Hold control and use the left/right arrow keys!
- move by paragraph? Hold control and use the up/down arrow keys!
- Saving: hold CTRL, press X then S (all while holding control
- Closing: hold CTRL, press X then C (all while holding control)
- Convention: C-x means “hold control, press x”
    - C-x C-s means “press x and s, all while holding control”
- These editors predate “normal” shortcuts!

Useful Shortcuts

C-x C-f	Open a file for editing
C-x C-s	Save the current file
C-x C-c	exit
C-x b	change to a different open file
C-space (arrow key)	Start highlighting (marking) a region
C-w	Cut the code in the highlighted region
Alt-w	Copy the code in the highlighted region
C-g	Quit (cancel command, “escape”)
C-y	paste
C-s	search (find)
Escape-x	Enter a command by name (C-g to quit)
C-x k	close a file (it will ask) (emas stays open)
Escape-$	spellcheck the word under the cursor
Escape-x ispell	spellcheck the highlighted region
Escape-x help	Get just a lot of help information
Escape-x `<tab>`	List ALL THINGS EMACS CAN DO

What editor to choose?

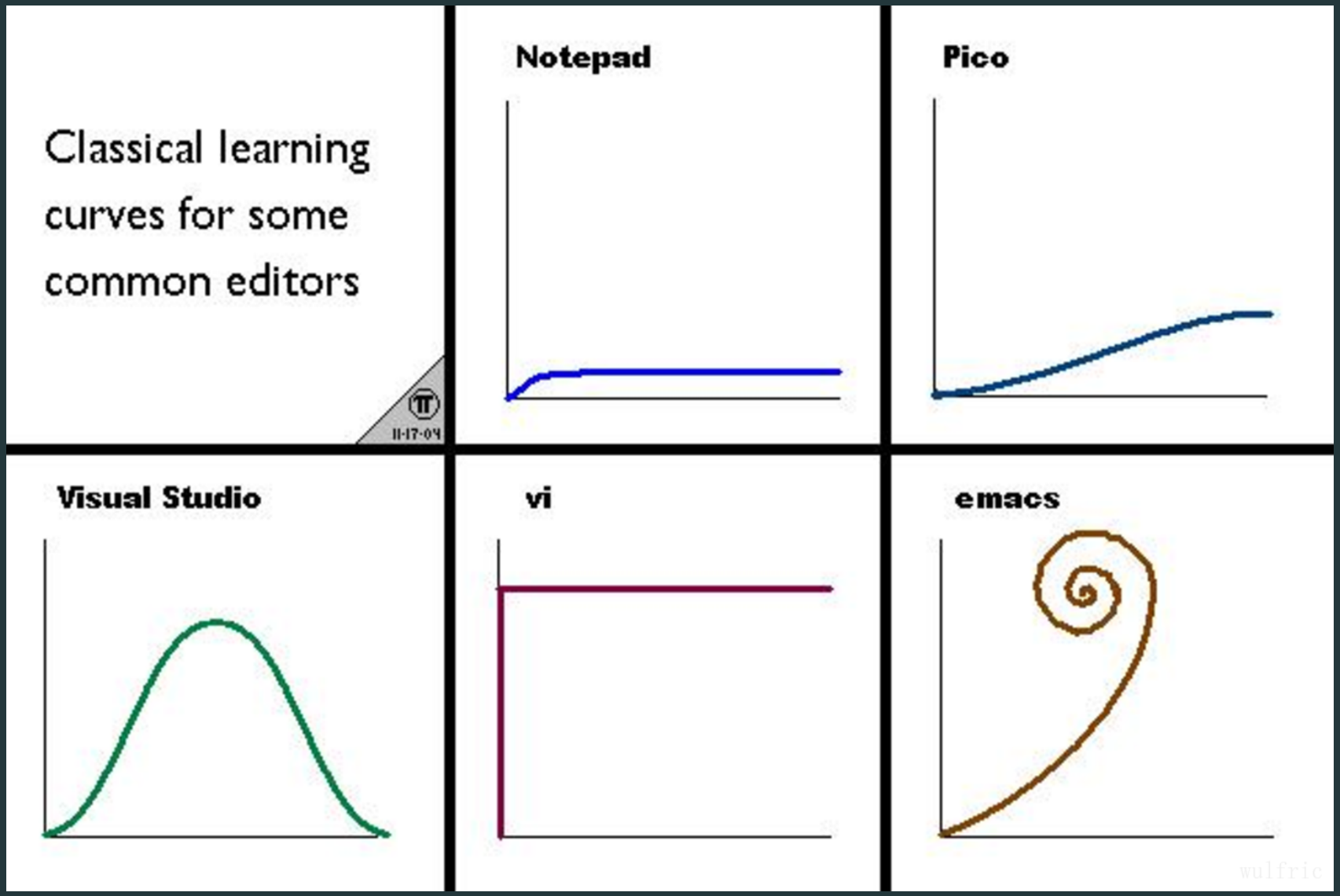

Figure 2: Editor Learning Curves

Let’s Git Started

What is `git`?

- `git` is a *decentralized* version control system.
- Like “historic versions” for DropBox/OneDrive
- Except far more advanced, and more streamlined
- It enables you to save changes as you go to your code.
  - As you make these changes, if at any point in time you discover your code is “broken”, you can *revert* back in time!
  - Of course, if you haven’t been “saving” frequently, you have less to work with.
  - Mantra: *commit* early and often.
- Can also *share* your code with friends!!
  - Can work on same version, or...
  - can “go back in time” to latest working one!
  - You will have trouble – we all do.

The Official Man Entry

The Stupid Content Tracker

```
git [--version] [--help] [-C <path>] [-c <name>=<value>]
    [--exec-path[=<path>]] [--html-path] [--man-path]
    [--info-path] [-p|--paginate|--no-pager]
    [--no-replace-objects] [--bare] [--git-dir=<path>]
    [--work-tree=<path>] [--namespace=<name>]
    <command> [<args>]
```

- Do **not** expect to learn `git` once and be done.
- You will learn it steadily, over time. The sooner you start, the better off you will be in your deveolpment career.
- **Git is not just for CS Majors**.
    - It is for *anybody* working with **any** code.

`git` Terminology

- The tracked folder is called a *repository* (*repo*)
- You `git init .` to create repository “here”
- To *track* a file in a repository, you `git add <filename>`
- The act of “saving” is *commit*, and needs a message
  - to commit all tracked files,
    `git commit -a -m 'your message here'`
- To copy a repository, you `git clone` it
- To work with friends, you need to
  - `git clone` their (or a common) repository
  - `git pull /other/repo/path` their changes
- if you edited the same file, you get a *conflict*
  - if you have uncommitted changes, you can’t pull

Teaser: Example Scenario

- Suppose you (`A`), and your best friend `B` are working in the same repo.
- You `init` the repository and make a `commit`; your friend then `clones` from you
- `A` and `B` both edit the same file and `commit` the edits
- `A` `pull`s, and discovers the conflict! You resolve it, but..
- `B` `pull`s, and discovers another one!
- Basically, `git` can get complicated quickly. Nothing replaces actual communication!

Demo Time! Everybody!

```
git clone /course/cs2043/demos/git-demo cd git-demo
```

```
git pull /course/cs2043/demos/git-demo
```

```
nano demo-file
```

```
git commit -a -m ‘mucking with the demo’
```

```
git pull /course/cs2043/demos/git-demo
```

References

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