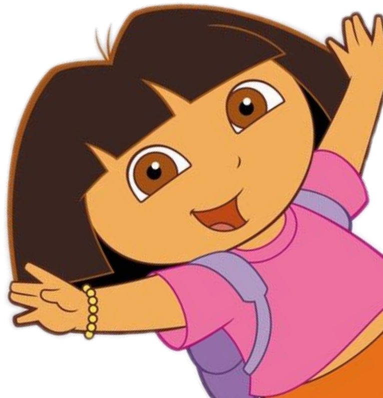
213/513 Linu-x/Git Bootcamp

Shivi, Di, Spoorthi

outline

1. ssh but also Windows ssh client especially
2. bash commands + navigating Linux
3. VIM and VS Code
4. Git



how to ssh

1. on OS X/Linux:

   ```
   $ ssh ANDREW-ID@shark.ics.cs.cmu.edu
   ```

   (don't type in the "$" this just means you're typing what follows into terminal)

2. type your password when prompted
3. if you see a warning about SSH host keys, click or enter "yes"

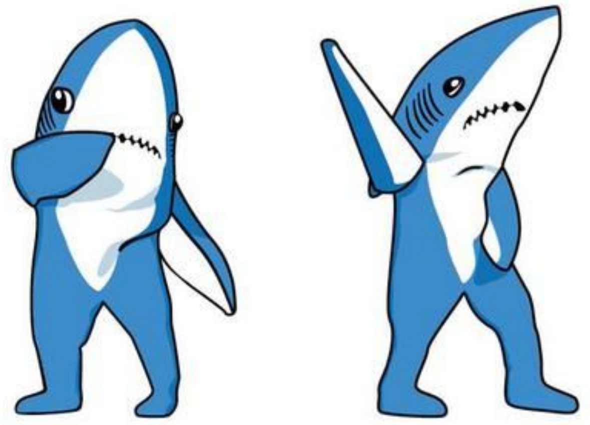
Windows computers???

- Use MobaXTerm for file transfer and ssh client!
- Instructions can be found here: http://www.cs.cmu.edu/~213/activities/linux-bootcamp/windows-setup.pdf

what are shark machines?

shark machines, linux.andrew.cmu.edu and unix.andrew.cmu.edu are all machines that access the same Andrew File System (AFS)

shark machines are explicitly set up for 213: they're standardized for benchmark tests and have correct versions of gcc, gdb and other tools



use the shark machines... otherwise your compiled code won't behave as expected!!!

navigating the shark machines

`$ ls`	list all files in folder. “-a” flag lists hidden files
`$ pwd`	print current file path
`$ cd PATH`	enter the folder PATH. “.” is current folder, “..” is parent
`$ mkdir NAME`	make a folder called NAME
`$ touch NAME`	make a file called NAME
`$ rm NAME`	remove file called NAME
`$ cat NAME`	output file NAME’s content to commandline
`$ mv FILE DEST`	move FILE to DEST folder
`$ cp FILE DEST`	move FILE to DEST folder
`$ scp FILE ANDREW-ID@shark.ics.cs.cmu.edu:DEST`	move FILE from local machine to DEST folder on shark machine
`$ tar OPT NAME`	compress to tar file or open tar file based on OPTs

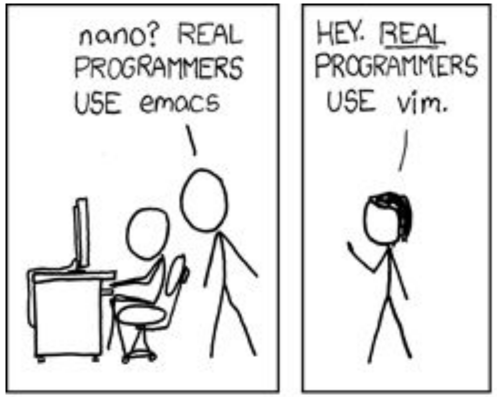
make files

- Makefile provided in the assignment handout
  - specifies source files and flags to compile with
- `$ make` compiles and links files; generates an executable
- `$ make clean` removes files created by running make

Editing files

Example 1: VIM

1. Can be run on pretty much any terminal, typically used in ssh and remote access
2. Highly customizable, in terms of plugins and scripting (with vimscript)
3. According to legend, if you learn all the keyboard shortcuts, the rate at which you code approaches lightspeed to the point of being potentially dangerous to those around you


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.

Example 1: VIM

1. Vim has a command grammar, and most fancy functionality originate from a coherent verb-modifier-noun structure of commands, just like simplified English. Some examples: (<> to denote the key combination, in normal mode unless specified)
   a. verbs: <i> (insert), <a> (append after cursor), <Shift-a> (append to end of the line), <c> (change), <d> (delete), <y> (yank/copy), <h><j><k><l> (move one char left, down, up, right resp.)
   b. modifiers: f (find and jumps to char), / (search..find a string/regex)
   c. text objects: w (word), s (sentence), p (paragraph), b (block/parentheses)
2. Three big modes
   a. Normal mode: <esc> from anywhere (terminates all pending commands), default mode
   b. -- INSERT -- mode: in normal mode, <i> to insert (prior to the cursor)
   c. -- VISUAL -- mode: in normal mode, <v> to highlight in the traditional sense

Example 1: VIM

1. Let's start by SSH'ing into the shark machines!

```
$ ssh ANDREW-ID@shark.ics.cs.cmu.edu
```

2. From here, let's make VIM "spicy" by running the following (to initialize your own custom vim configuration file!):

```
$ vim ~/.vimrc
```

Example 1: VIM

3. Press <i> and make sure you see "-- INSERT --" at the bottom. Then type that into the text buffer → → →
4. When done, press <esc> and then type in <:w> to save
5. Type in <:q> to quit VIM. (This can be combined into <:wq> to save and quit in one command :-0)

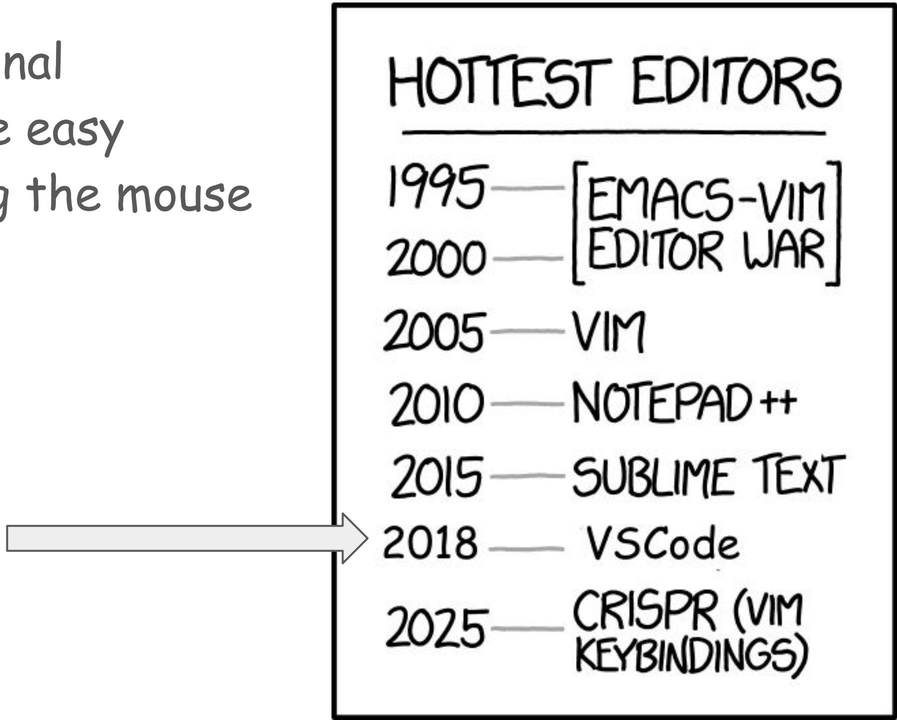
```
colorscheme desert
set mouse=a
set number
set cursorline
set colorcolumn=81
set tabstop=2
set shiftwidth=2
set softtabstop=2
set expandtab
set smartindent
```

Example 1: VIM

- Normal mode: <esc>
    - -- INSERT -- mode: <i> key
        - Type and stuff :-0
    - -- VISUAL -- mode: <v> key
        - Use any movement verb (command that move your cursor) to highlight a selection
        - “Copy and paste”:
            - Highlight text, press <y> to yank (copy) and <p> to paste (within VIM)
            - Similarly, pressing <dd> will delete the selection, which also makes it available to paste with <p>
    - Save: <:w>
    - Quit: <:q>
- With “set mouse=a” in .vimrc, you can also scroll and click with the mouse
- Highly recommend `$ vimtutor` for a canonical introduction into VIM
- Some useful links: https://devhints.io/vim

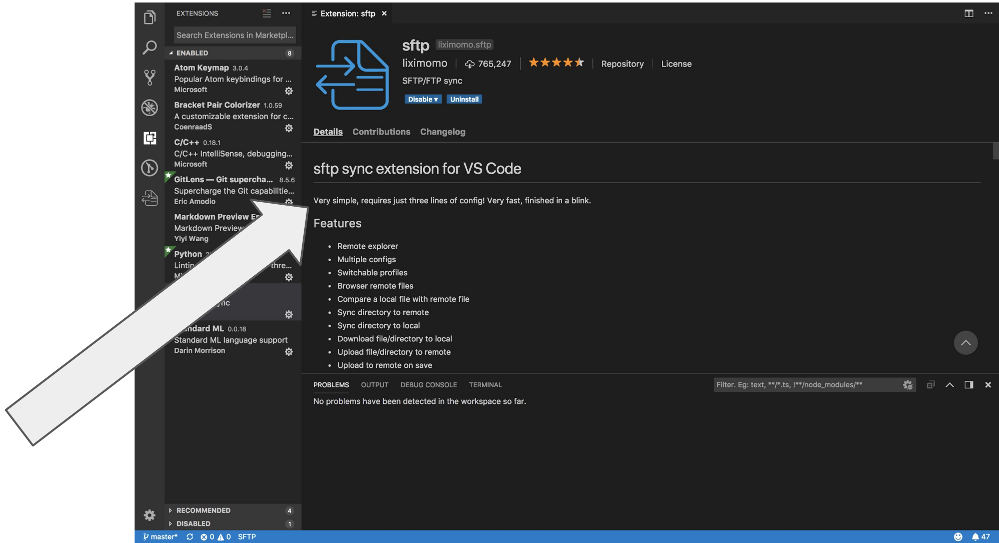
Example 2: VSCode + SFTP

1. Visually appealing text editor with lots of cool keyboard shortcuts and functionality
2. Tabs, easy window split, built-in terminal
3. Cool plugins to make code pretty + life easy
4. People won't make fun of you for using the mouse



Example 2: VSCode + SFTP

- Download here: https://code.visualstudio.com/download
- You can check out some of the other extensions (Microsoft C plugin?, tabnine?!???) but absolutely download liximomo's sftp plugin because that's how we're gonna be writing code



Atom Keymap 3.0.4
Popular Atom keybindings for ...
Microsoft

Bracket Pair Colorizer 1.0.59
A customizable extension for c...
CoenraadS

C/C++ 0.18.1
C/C++ IntelliSense, debugging...
Microsoft

GitLens — Git supercha... 8.5.6
Supercharge the Git capabilitie...
Eric Amodio

Markdown Preview E
Markdown Preview
Yiyi Wang

Python
Lintin thre...
M

nc

ndard ML 0.0.18
Standard ML language support
Darin Morrison

sftp liximomo.sftp
liximomo | 765,247 | ★★★★★ | Repository | License
SFTP/FTP sync

sftp sync extension for VS Code

Very simple, requires just three lines of config! Very fast, finished in a blink.

Features

- Remote explorer
- Multiple configs
- Switchable profiles
- Browser remote files
- Compare a local file with remote file
- Sync directory to remote
- Sync directory to local
- Download file/directory to local
- Upload file/directory to remote
- Upload to remote on save

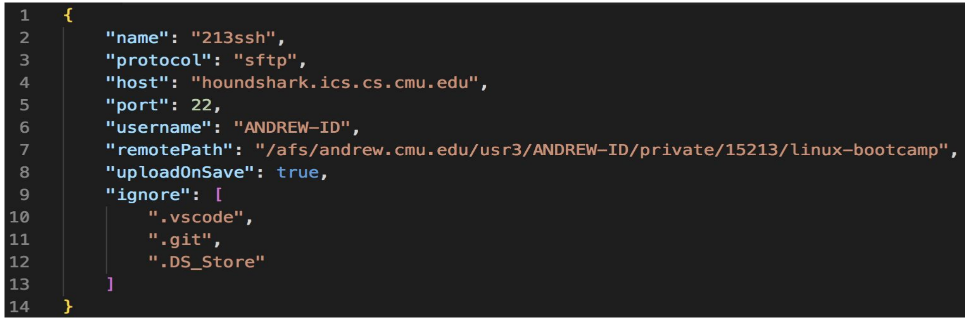
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Filter. Eg: text, **/*.ts, !**/node_modules/**

No problems have been detected in the workspace so far.

Example 2: VSCode + SFTP

- Go to your 213/513 folder on your local machine and create a folder called “linux-bootcamp.” Open it in VSCode
- Ctrl + Shift + P (Windows) or Cmd + Shift + P (Mac) to open up Command Palette:
- Type in “SFTP: Config”
- This should open “sftp.json”
- Type in the following info → → →

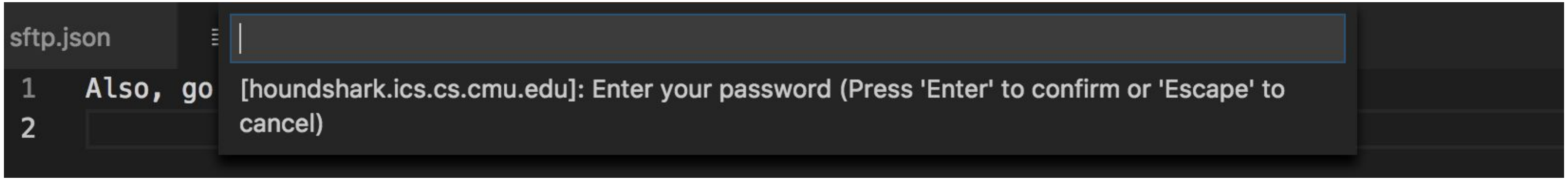


```json
{
    "name": "213ssh",
    "protocol": "sftp",
    "host": "houndshark.ics.cs.cmu.edu",
    "port": 22,
    "username": "ANDREW-ID",
    "remotePath": "/afs/andrew.cmu.edu/usr3/ANDREW-ID/private/15213/linux-bootcamp",
    "uploadOnSave": true,
    "ignore": [
        ".vscode",
        ".git",
        ".DS_Store"
    ]
}
```

- Visit https://github.com/liximomo/vscode-sftp/wiki/config for extra config options

Example 2: VSCode + SFTP

- Create a file called “example.txt” and type whatever you want into it
- When you save, this should prompt a popup to type in your ssh password
- Now if you ssh into a shark machine and navigate to the same file path, you should see “example.txt” inside!



sftp.json

```
Also, go
```

[houndshark.ics.cs.cmu.edu]: Enter your password (Press 'Enter' to confirm or 'Escape' to cancel)

Example 2: VSCode + SFTP

REMINDERS:

1. SFTP means you're downloading code from AFS onto your local machine, so take extra precaution to make sure that code is secure and no one steals it!
2. Any time you run `$ make`, please do so on the shark machines!!

GIT

What is git?

- Version control system
  - Better than:
    - copy pasting code
    - emailing the code to yourself
    - taking a picture of your code and texting it to yourself
    - zipping the code and messaging it to yourself on facebook
- git ≠ github
- **using git this semester is mandatory!!!** ~*style*~ point deductions if you don't use it

Important commands

`$ git init`	make a new repository
`$ git clone`	initialize a repository locally from a remote server
`$ git status`	MOST IMPORTANT COMMAND
`$ git log`	show commit history. Can use --decorate --graph --all to make it pretty
`$ git add`	stages files to be committed. Flags: --a, -u
`$ git commit -m`	commit the changes in the staged files (use good messages!)
`$ git push`	push changes to a remote server (--set-upstream origin branchname)
`$ git pull`	pull changes from a server
`$ git branch`	make a new branch
`$ git checkout`	switch to a different branch. Can use -b to make a new branch
`$ git merge name`	merge "name" branch into your current branch
`$ git reset HEAD`	Used to unstage files
`$ git reset --hard + hash`	Used to reset to an old commit (with a commit hash)

Example

https://github.com/eyluo/linux-bootcamp

if that link is too long, try:

https://tinyurl.com/goKnicks213

Configuring git

```
$ git config --global user.name "<Your Name>"

$ git config --global user.email “<Your Email>”

$ git config --global push.default simple
```

(Make sure the email is your Andrew ID, and make sure to add that email to your GitHub account!)

Cloning the repository

1. Go to to link in previous slide and click "fork" in the top right corner to copy the repository to your Github account
2. Make sure you are in your account, and click the green "clone or download" on the right
3. Copy the link
4. Open up a terminal window (or xterm for windows users) and ssh into a shark machine
   a. `$ ssh ANDREW-ID@shark.ics.cs.cmu.edu`
   b. navigate to a folder where you want to do this example
5. `$ git clone` + the link you copied
   a. This will initialize the git repository on your computer, with GitHub as the remote server
6. `$ cd` switch into the repository

Committing, pushing, pulling

1. `$ ls` we have 4 files here
2. `$ git status` branch is up to date with the server, nothing to commit
3. `$ git log --graph --decorate --all`
    i. Shows a pretty graph of the commit history.
4. `$ vim example.txt` lets make some changes to example.txt
5. `$ git status` now shows that we have unstaged files
6. `$ git add example.txt` stages the file to be committed
7. `$ git reset HEAD example.txt` unstages the file (to show you how to do that)
8. `$ git add example.txt` to restage the file
9. `$ git commit -m “insert a relevant commit message here”`
10. `$ git status` shows you are 1 commit ahead of “origin” = remote server
11. `$ git push` this updates the remote server
12. `$ git log --graph --decorate --all` now we can see the new commit on top of all the old ones

Merging

1. `$ git log --graph --decorate --all`note the other branch "realistic ending" that branches away from master
2. `$ git checkout realistic_ending`switch to the other branch
3. `$ git branch` shows all of our branches
4. `$ ls` note that there are different files here
5. `$ vim example.txt` we can see the story is different than in the master branch-finish it!
6. Add and commit the file, push to the server.
7. `$ git checkout master` switch back to the master branch
8. `$ git merge realistic_ending`will attempt to merge the two branches, but there's a conflict
   a. `$ git status` shows that the conflict is in example.txt
   b. `$ vim example.txt` fix the story
   c. `$ git add example.txt`
   d. `$ git commit -m "appropriate message for a merge"`now the merge is complete
9. `$ git log -- decorate --graph --all`shows that now you still have 2 branches, but they've been merged and point to the same files

Resetting, Branching

1. `$ git log --decorate --graph --all` copy the commit hash of a past commit (first 6ish characters usually fine)
2. `$ git branch newbranchname` make a new branch
3. `$ git checkout newbranchname` switch to the new branch
4. `$ git reset --HARD + hash` from old commit
5. `$ git log --decorate --graph --all` note that now HEAD is at the old commit, master is still at the merge commit from last slide
6. `$ ls` the files are different now
7. `$ vim example.txt` the story is different too. Add a line or two to it
8. Add and commit
9. `$ git log --decorate --graph --all` now we can see how it has separated from the rest of the tree
    a. This is how you would test out new feature. If you decide you like it, you can later merge it into the master branch. If not, you can just leave it and switch back to master.

Adding your new branch to the remote server

1. `$ git status` note that it says nothing about the origin remote server
2. `$ git push` doesn't work, there is no "upstream branch" (nothing on the server)
3. `$ git push --set-upstream origin newbranchname`
    a. This creates a new branch on the origin server, and sets it as the "upstream" of your current branch. In the future when you push, you can just do git push and it will work.
4. `$ git status` now branch is up to date with origin/newbranchname
5. `$ git checkout master`
6. `$ git status` we're far ahead of the remote server
7. `$ git push`

$ .gitignore files

- Make one in each of your projects
  - Can use touch, emacs, vim, whatever you want
- *.o will ignore all .o files, or object files (* matches any substring, and .o will match exactly)
- Useful because when you add a lot of new files with $ git add -a you want git to ignore certain files

Appendix

VIM

Cursor movement

h - move left
j - move down
k - move up
l - move right
w - jump by start of words (punctuation considered words)
W - jump by words (spaces separate words)
e - jump to end of words (punctuation considered words)
E - jump to end of words (no punctuation)
b - jump backward by words (punctuation considered words)
B - jump backward by words (no punctuation)
0 - (zero) start of line
^ - first non-blank character of line
$ - end of line
G - Go To command (prefix with number - 5G goes to line 5)
Note: Prefix a cursor movement command with a number to repeat it. For example, 4j moves down 4 lines.

Insert Mode - Inserting/Appending text

i - start insert mode at cursor
I - insert at the beginning of the line
a - append after the cursor
A - append at the end of the line
o - open (append) blank line below current line (no need to press return)
O - open blank line above current line
ea - append at end of word
Esc - exit insert mode

Editing

r - replace a single character (does not use insert mode)
J - join line below to the current one
cc - change (replace) an entire line
cw - change (replace) to the end of word
c$ - change (replace) to the end of line
s - delete character at cursor and subsitute text
S - delete line at cursor and substitute text (same as cc)
xp - transpose two letters (delete and paste, technically)
u - undo
. - repeat last command

Marking text (visual mode)

v - start visual mode, mark lines, then do command (such as y-yank)
V - start Linewise visual mode
o - move to other end of marked area
Ctrl+v - start visual block mode
O - move to Other corner of block
aw - mark a word
ab - a () block (with braces)
aB - a {} block (with brackets)
ib - inner () block
iB - inner {} block
Esc - exit visual mode

Visual commands

> - shift right
< - shift left
y - yank (copy) marked text
d - delete marked text
~ - switch case

Cut and Paste

yy - yank (copy) a line
2yy - yank 2 lines
yw - yank word
y$ - yank to end of line
p - put (paste) the clipboard after cursor
P - put (paste) before cursor
dd - delete (cut) a line
dw - delete (cut) the current word
x - delete (cut) current character

Exiting

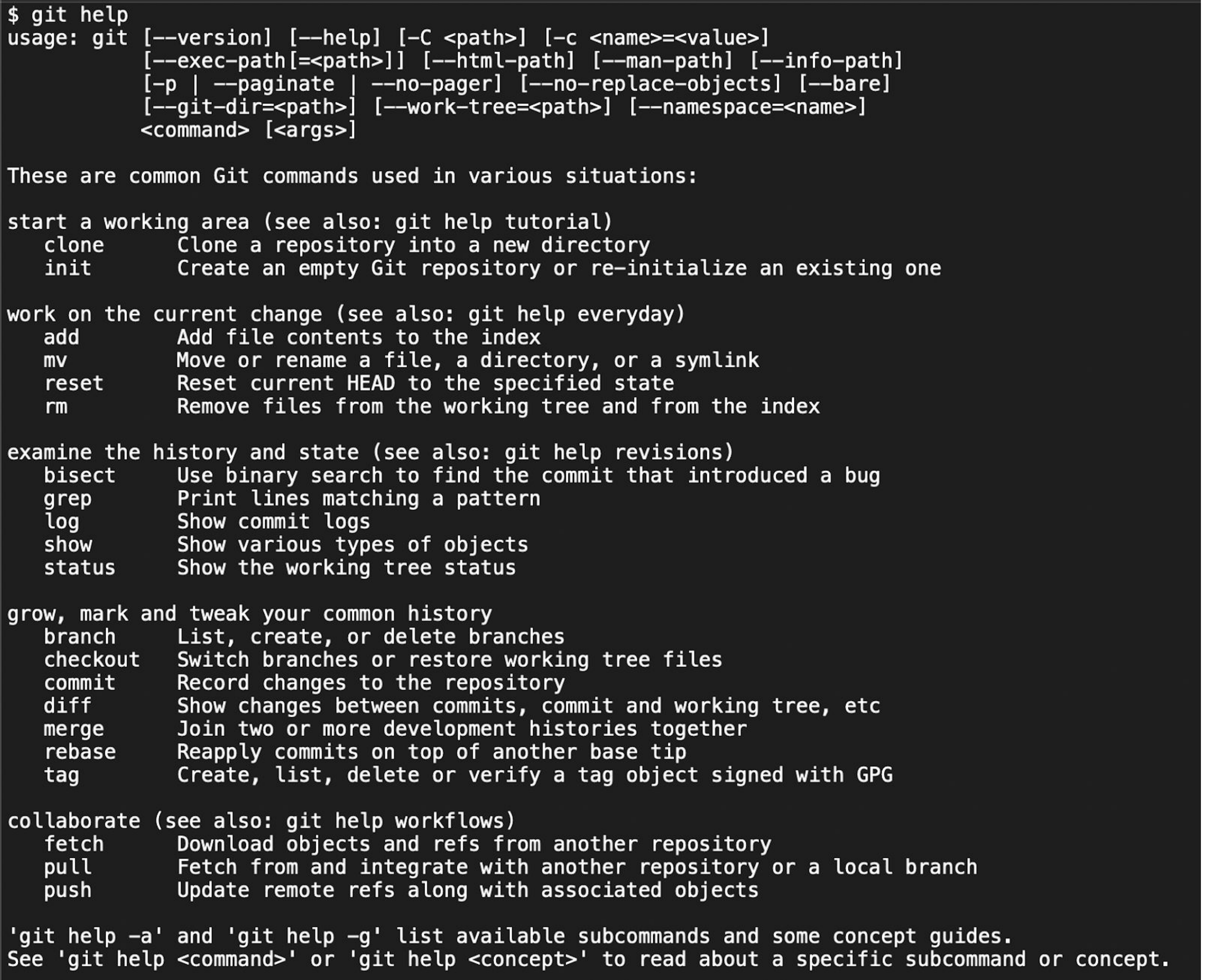
:w - write (save) the file, but don't exit
:wq - write (save) and quit
:q - quit (fails if anything has changed)
:q! - quit and throw away changes

Search/Replace

/pattern - search for pattern
?pattern - search backward for pattern
n - repeat search in same direction
N - repeat search in opposite direction
:%s/old/new/g - replace all old with new throughout file
:%s/old/new/gc - replace all old with new throughout file with confirmations

Working with multiple files

:e filename - Edit a file in a new buffer
:bnext (or :bn) - go to next buffer
:bprev (of :bp) - go to previous buffer
:bd - delete a buffer (close a file)
:sp filename - Open a file in a new buffer and split window
ctrl+ws - Split windows
ctrl+ww - switch between windows
ctrl+wq - Quit a window
ctrl+wv - Split windows vertically



```
$ git help
usage: 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>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or re-initialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some concept guides.
See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.
```

Courtesy of git help