



VERSION CONTROL FOR SCIENTISTS

# AN INTRODUCTION TO GIT, GITHUB, AND ATOM

MAY 2019

STEVEN J. CLIPMAN, MSPH  
PHD CANDIDATE  
JOHNS HOPKINS UNIVERSITY



OCTOCAT!

# BEFORE WE GET STARTED

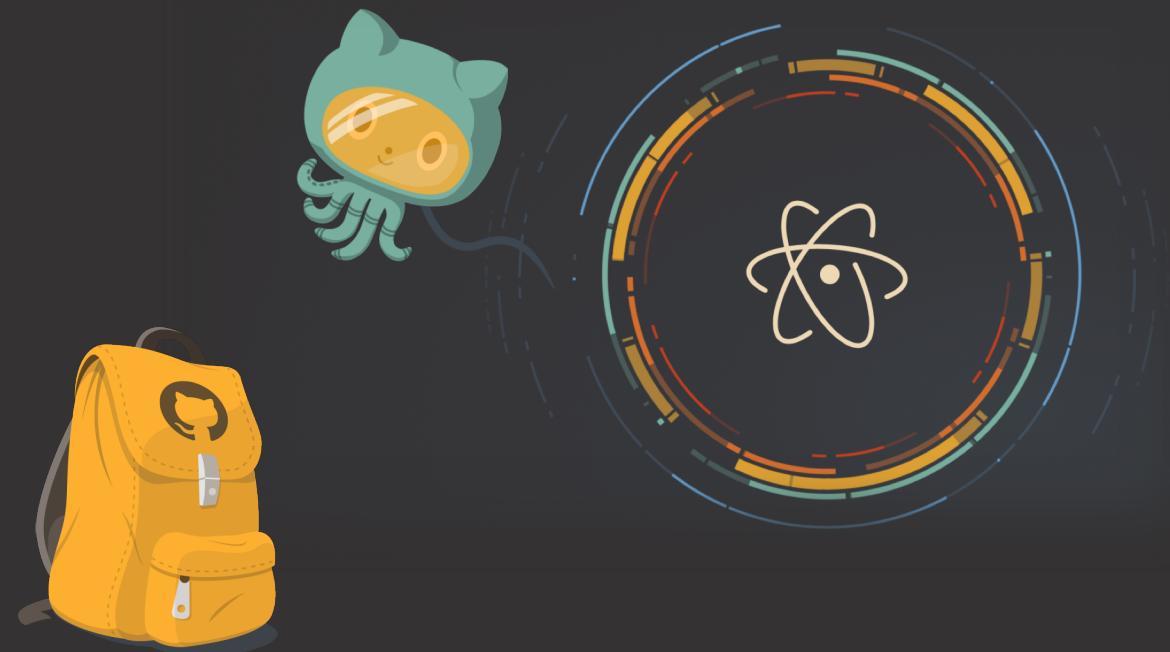


1. Sign-up for GitHub Pro with a student developer pack:

[education.github.com/pack](https://education.github.com/pack)

2. Download and install Atom

[atom.io](https://atom.io)



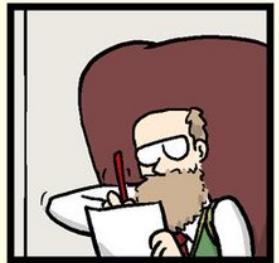
# "FINAL".doc



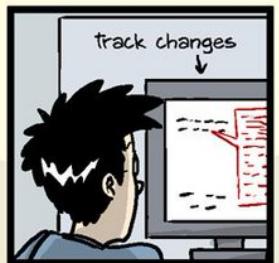
FINAL.doc!



FINAL\_rev.2.doc



FINAL\_rev.6.COMMENTS.doc      FINAL\_rev.8.comments5.CORRECTIONS.doc



JORGE CHAM © 2012

FINAL\_rev.18.comments7.corrections9.MORE.30.doc      FINAL\_rev.22.comments49.corrections.10.#@\$%WHYDID  
ICOMETOGRADSCHOOL?????.doc

## WHAT IS VERSION CONTROL AND WHY USE IT?

# VERSION CONTROL

- A system that manages and keeps records of changes made to files
- Allows for collaborative development
- Allows you to know who made what changes and when
- **Allows you to revert any changes and go back to a previous state**

Article

## Excuse Me, Do You Have a Moment to Talk About Version Control?

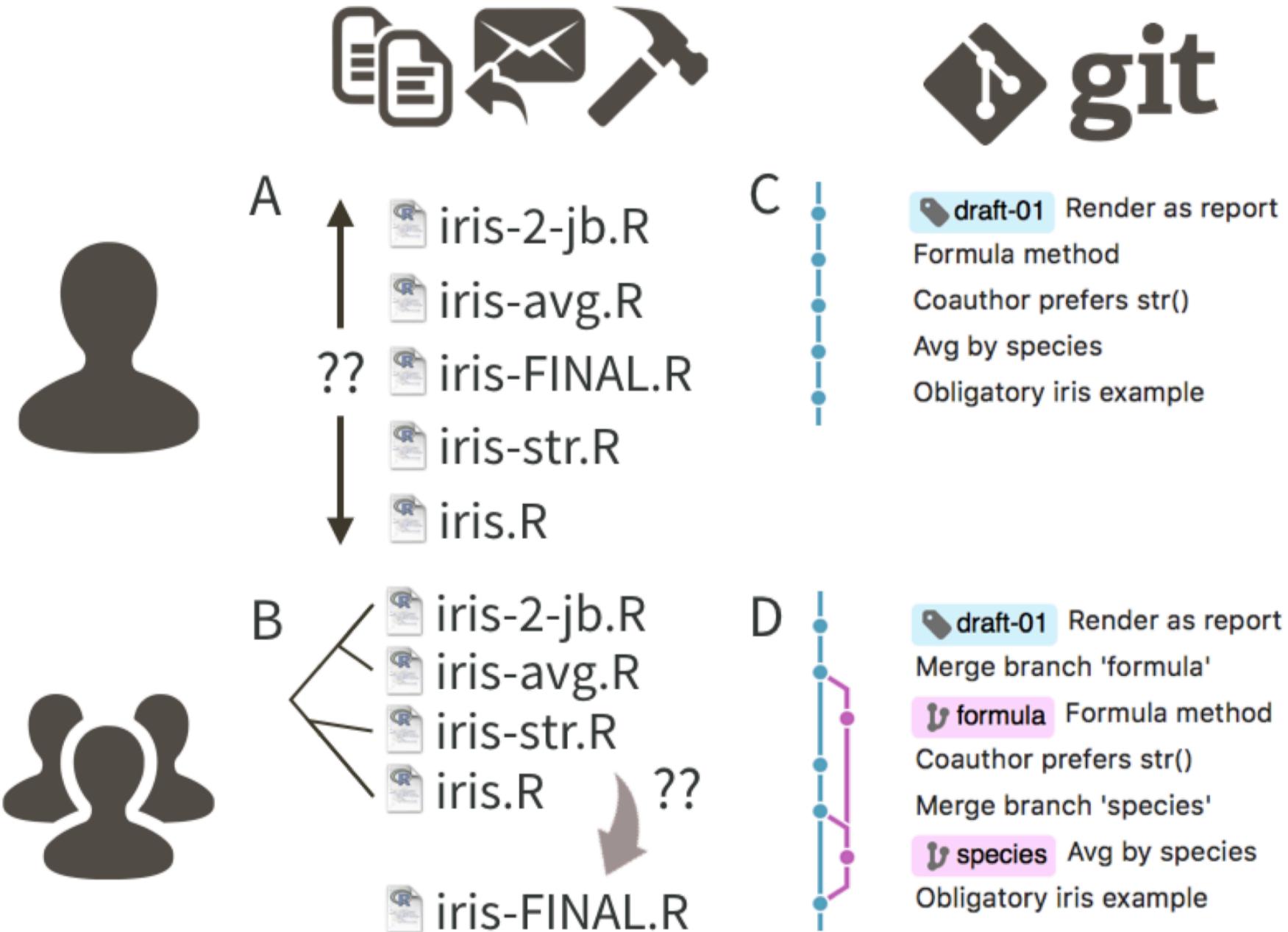
Jennifer Bryan

Pages 20-27 | Received 01 Jul 2017, Accepted author version posted online: 14 Nov 2017, Published online: 24 Apr 2018

 Download citation

 <https://doi.org/10.1080/00031305.2017.1399928>



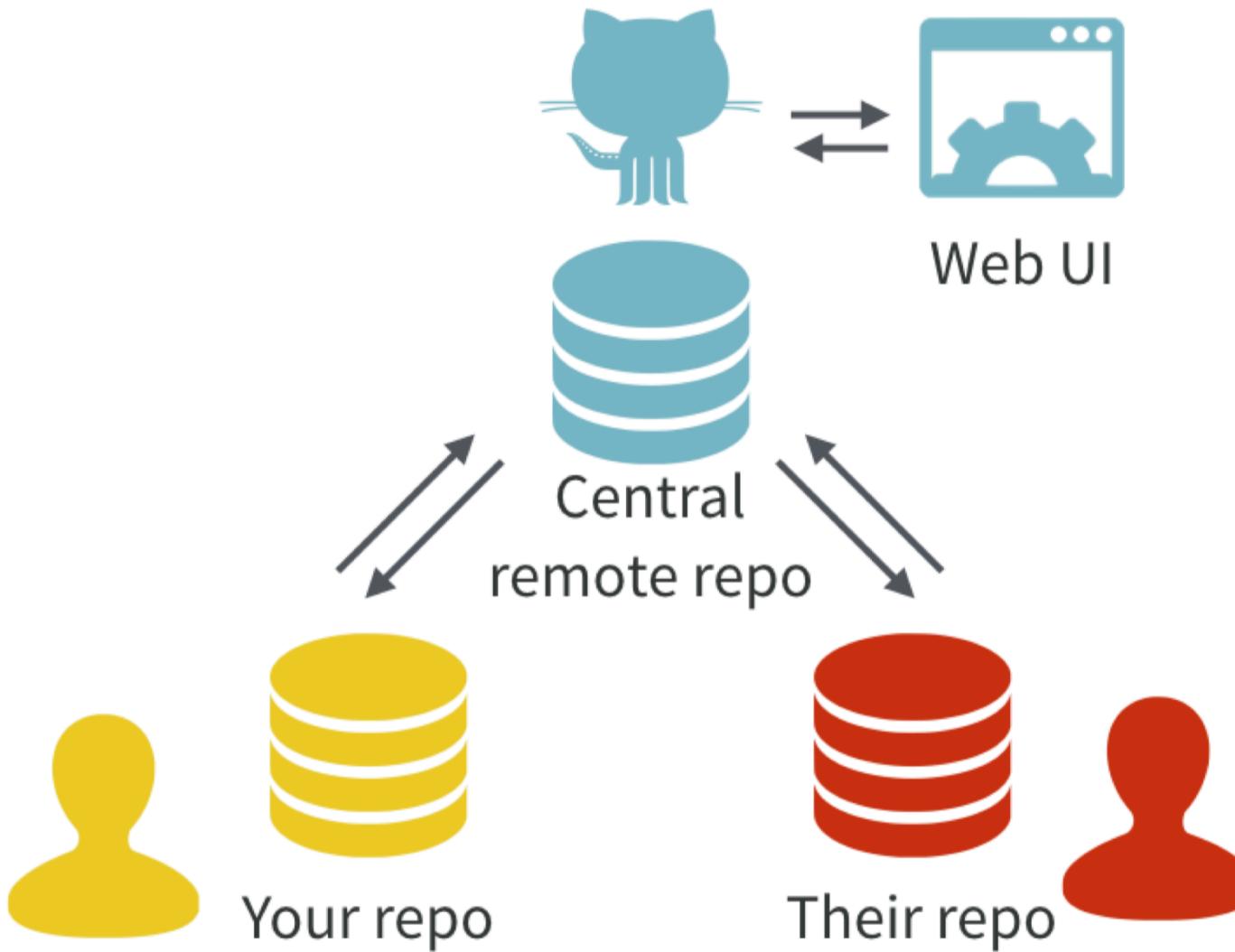


# WHAT ARE GIT AND GITHUB?



- Git and GitHub are **two different things**
- Git is a particular implementation of version control originally designed by Linus Torvalds in 2005 as a way of managing the Linux kernel. Git manages the evolution of a set of files – called a repository or repo
- Essentially, the language of version control
- GitHub is an online hub for hosting Git repositories and provides GUI software for using Git
- GitHub complements Git by providing a slick user interface and distribution mechanism for repositories. Git is the software you will use locally to record changes to a set of files. GitHub is a hosting service that provides a Git-aware home for such projects on the internet
- GitHub is like DropBox or Google Drive, but more structured, powerful, and programmatic





With Git, all contributors have a copy of the repo, with all files and the full history. It is typical to stay in sync through the use of a central remote repo, such as GitHub. Hosted remotes like GitHub also provide access to the repo through a web browser.

# GIT CAN BE COMPLICATED AT FIRST BUT IS EASY TO LEARN



Here are some great places to go to learn Git:

- [Udemy - Git complete step-by-step guide](#)
- [TeamTreehouse - Git Basics](#)
- [Codecademy - Learn Git](#)
- [Lynda.com - Git Essential Training](#)
- [atlassian.com - Getting Git Right](#)

Nice work Steven

Show that you know this skill by adding your badge to your profile

Git  Passed: LinkedIn Assessment

Show on profile

## GLOSSARY:

- `cd` - change directory
- directory - the same thing as a folder
- `ls` - list the files and folders in a folder
- `touch` - create an empty file
- repository - the saved history of a folder and files, used by git.
- `init` - start or initialize a git repository
- `add` - put a file into the staging area, so that git starts tracking it
- staging/index area - where files are stored before going into the history
- `commit` - send files in the staging/index area into the history (the git repository)
- `status` - check the status of the folder and the git repository
- `diff` - compare a file to the a file in the history
- `log` - view the commit history in the git repository

# MANY GIT COMMANDS CAN BE EXECUTED WITH GITHUB & ATOM

```
cd ~/Desktop ## Move to your desktop  
mkdir playing ## Create a folder (aka directory)  
cd playing  
git init ## Create the repository (init = initialize)
```

```
touch bio.txt ## Command to create a file called bio.txt  
ls ## Check that you created the file, ls = list files  
git add bio.txt ## Track the file  
## Save the file to the history with a message (-m)  
git commit -m "Initial commit"
```

```
git status ## Check the activity  
git diff bio.txt ## Compare to the one in the history  
git add bio.txt ## This sends it to the staging area  
git commit -m "Added my bio" ## This sends it to the history
```

```
git remote add origin https://github.com/yourusername/playing.git  
git push origin master  
git pull
```

# HOW DOES GIT WORK: KEY CONCEPTS

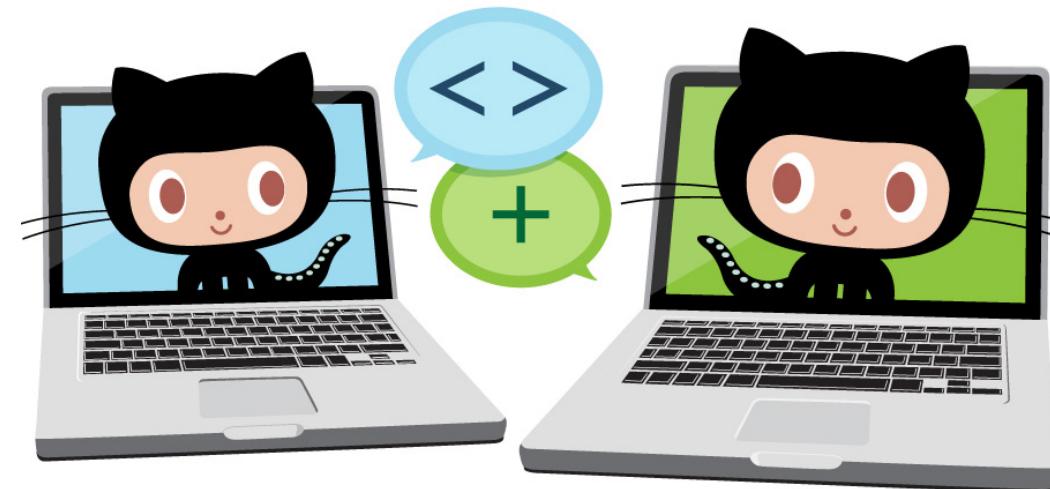
- Git keeps track of your files using “snapshots” that record what the files in your repository look like at any given point in time
- You decide when to take a snapshot and of what files, this is known as a **commit**
  - Can be used as noun or verb
  - *“I committed code”*
  - *“I just made a new commit”*
- Have the ability to go back and visit any commit
- A project is made up of a bunch of commits



# KEY CONCEPTS: COMMIT

Commits contain 3 pieces of information:

1. Information about how the files have changed from the previous commit
2. A reference to the commit that came before it, known as the **parent commit**
3. An SHA-1 hash code (e.g. fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e)



# WHAT IS AN SHA-1 HASH?



SHA-1 is an algorithm and what it does is:

It takes some data as input and generates a unique 40 character string from it.

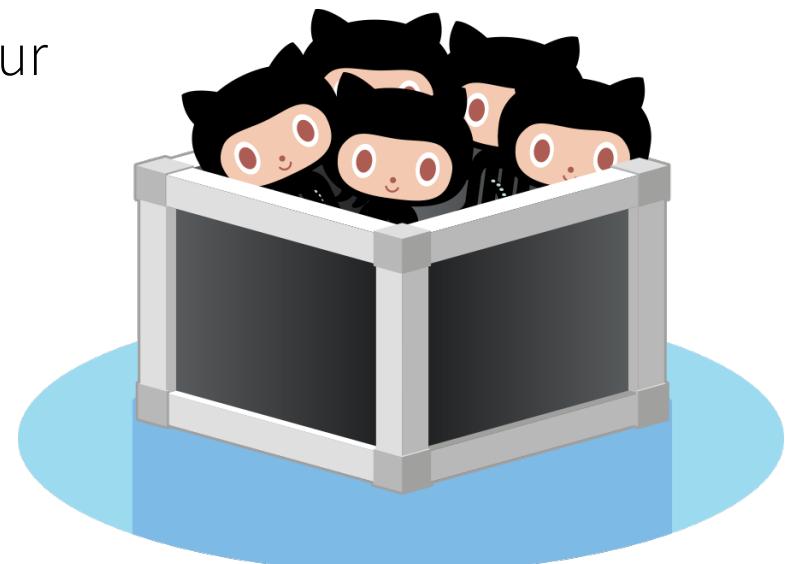
What does unique mean in this context? Unique means that no other input data should ever produce the same hash. The same input data however should always produce exactly the same hash.

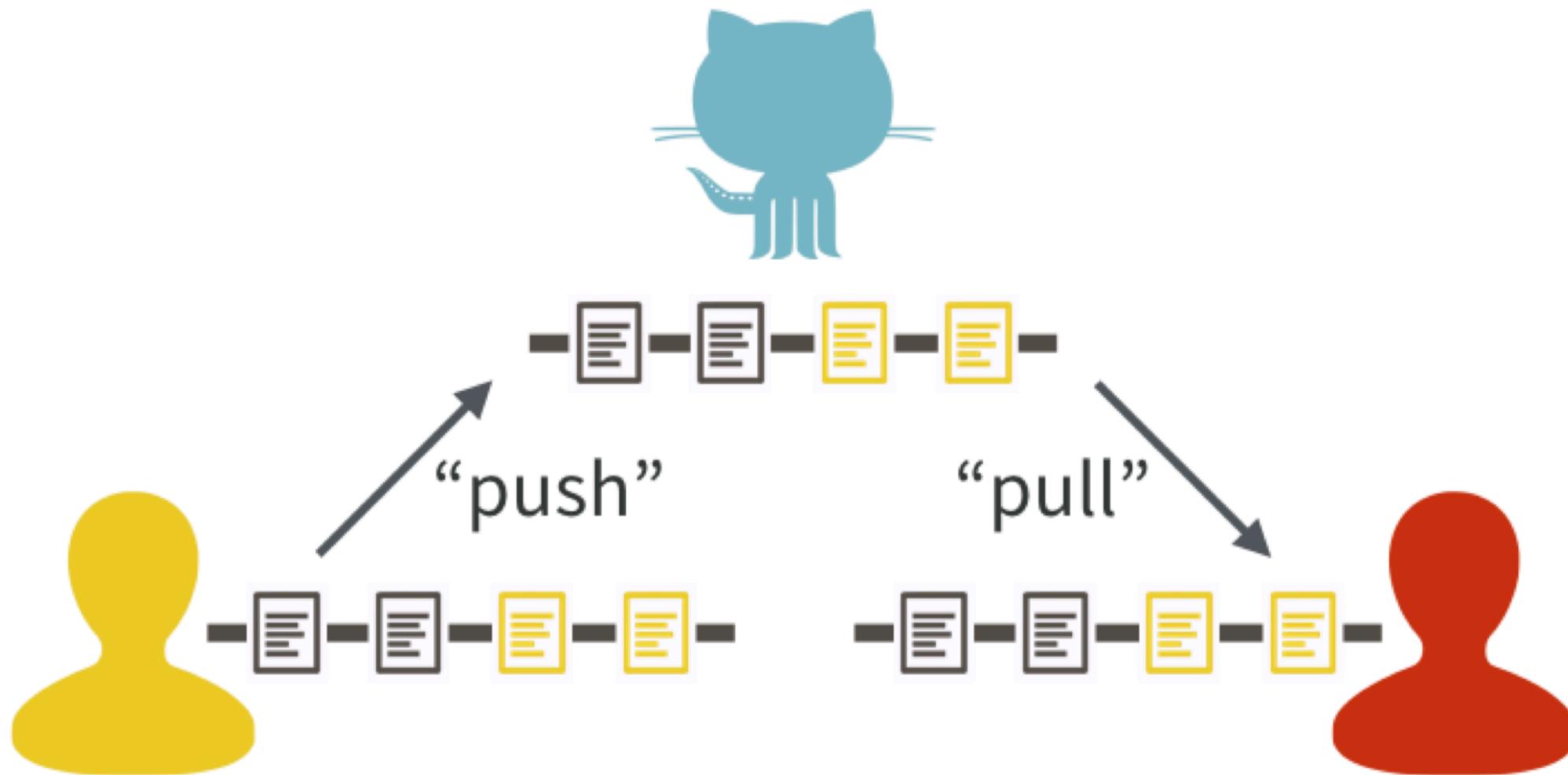
If you and I both look at `revision f4f78b319c308600eab015a5d6529add21660dc1` on our machines and Git tells us that we have a clean working directory, **we can be 100% certain that we are looking at exactly the same files.**

The hash ensures there is no way someone could manipulate a single bit without Git knowing about it.

# KEY CONCEPTS: REPOSITORIES

- Often referred to as a **repo**, this a collection of all your files and the history of those files (i.e. commits)
- Can live on a local machine or on a remote server (e.g. GitHub)
- The act of copying a repo from a remote server is called cloning
- Cloning from a remote server allows teams to work together
- The process of downloading commits that don't exist on your machine from a remote server is called **pulling** changes
- The process of adding your local changes to a remote repo is called **pushing** changes

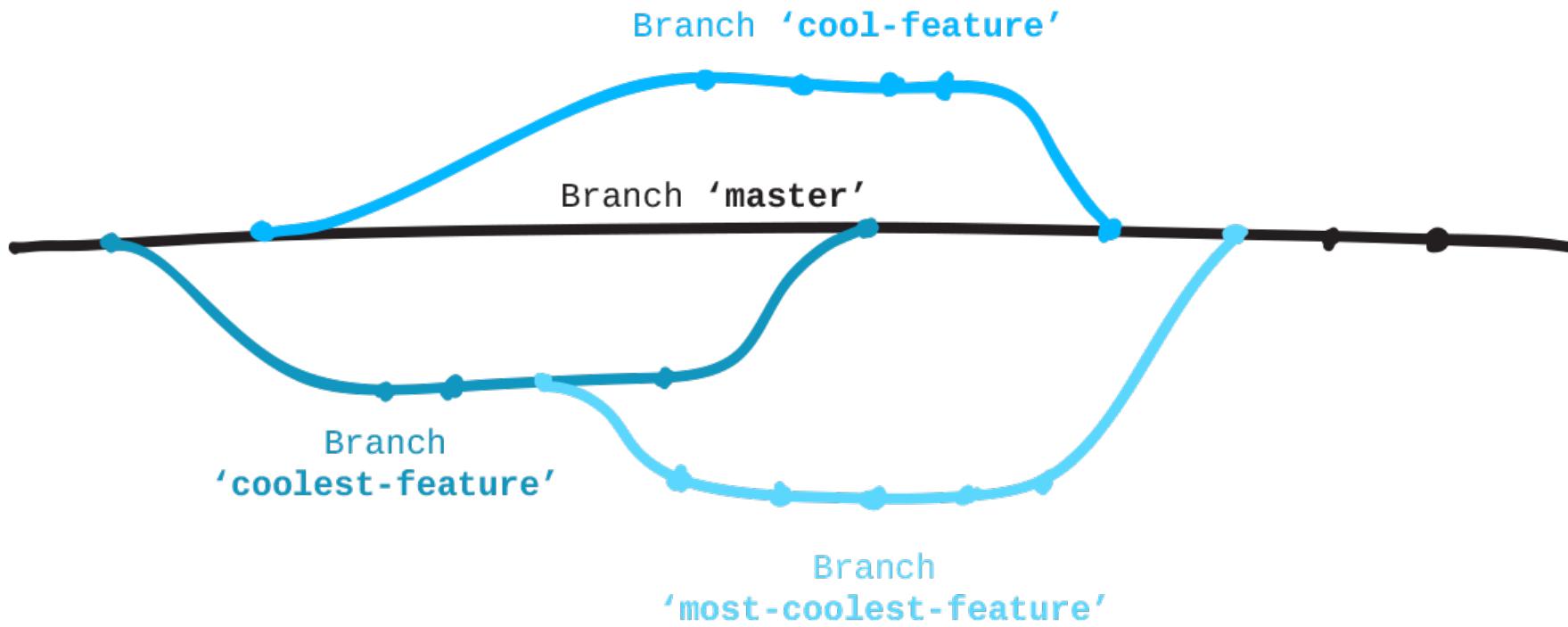


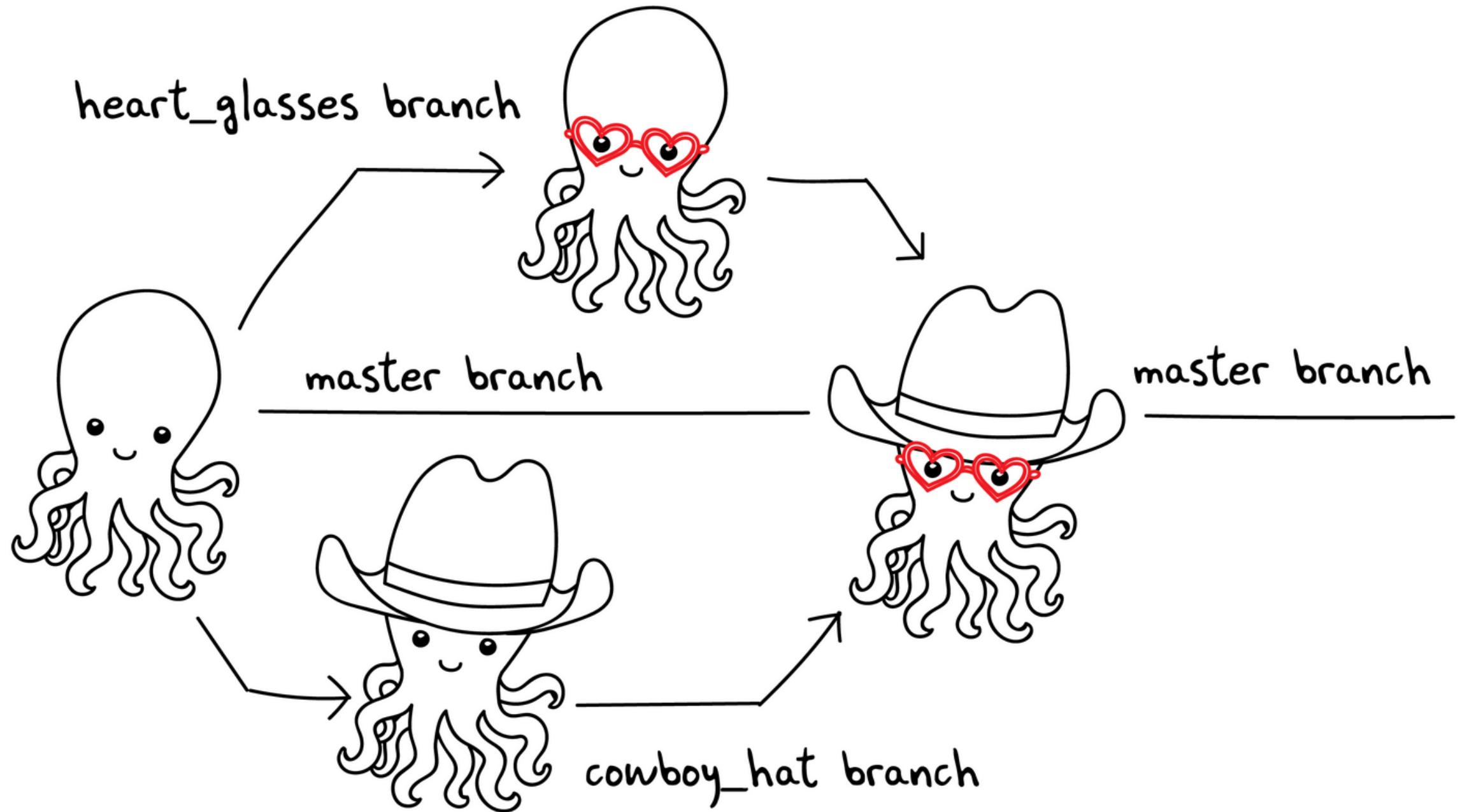


One contributor has made two new commits and updates the master copy on GitHub with a push.  
Another contributor stays up-to-date with a pull from GitHub.

# KEY CONCEPTS: BRANCHES

- All commits live on a branch
- There can be many branches
- The project's main branch is called the **master**





# KEY CONCEPTS: README.MD

- **Collaborate responsibly!** Every repository should be initialized with a **README.MD** file
- This is where you detail all the information about the project so that the work can be understood and recreated by another student or collaborator
- Arguably the most important file in your repo
- Use **markdown** syntax to make your file clear and easy to read



# MARKDOWN

```
319
320 ## Analysis 2: HIV Phylogenetics
321
322 1. Run `/Code/CombineFASTAs.py` with arg1 = directory of .fasta files arg2 = output
323 `python3 ~/github/Dissertation-Aim1/Code/CombineFASTAs.py ~/github/Dissertation-A
324
325 | HIV | Total | AM | DH | KA | IM | AZ |
326 | --- | --- | --- | --- | --- | --- | --- |
327 | env | 669 | 131 | 287 | 54 | 20 | 177 |
328
329 *See `/Sequences/sample_list_env.txt` for listing of all env samples*
330 *See `Sequence_Availability.xlsx` for full listing of samples with sequence by region*
331
332 2. Remove AZ samples from combined fasta file using grep in TextWrangler.
333
334 - Find: `>AZ(?:s).*\n` Replace: `>`
335 - Save as: `env_combined_sansAZ.fasta`
336
337 3. Determine subtype distribution using: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/
338 Env Results: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/job/343559231/
339 Pol Results: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/job/1773838861/
340
341 | HIV Subtype | Number of sequences | Percentage |
342 | --- | --- | --- |
343 | C | 467 | 94.92%
344 | A | 2 | 0.41%
345 | Recombinant C,A1 | 8 | 1.63%
346 | Recombinant C,B | 10 | 2.03%
347 | Unassigned | 5 | 1.02%
348 | Total | 492 | 100% |
349
350 4. Align sequences in combined fasta file using MUSCLE in MEGA7 with default parameters as necessary.
351 - Gap Open: -400
352 - Gap Extend: 0
353 - Max Iterations: 8
354 - Clustering Method (Iteration 1, 2 and others): UPGMB
355 - Min Diag Length (lambda): 24
```



## Analysis 2: HIV Phylogenetics

1. Run `/Code/CombineFASTAs.py` with arg1 = directory of .fasta files arg2 = output filename python3  
~/github/Dissertation-Aim1/Code/CombineFASTAs.py ~/github/Dissertation-Aim1/Sequences/hiv\_env  
env\_combined.fasta

HIV	Total	AM	DH	KA	IM	AZ
env	669	131	287	54	20	177

See `/Sequences/sample\_list\_env.txt` for listing of all env samples See `Sequence\_Availability.xlsx` for full listing of samples with sequence by region

2. Remove AZ samples from combined fasta file using grep in TextWrangler.
  - Find: `>AZ(?:s).\*\n` Replace: `>`
  - Save as: `env\_combined\_sansAZ.fasta`
3. Determine subtype distribution using: <http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/> Env Results: <http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/job/343559231/> Pol Results: <http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/job/1773838861/>

HIV Subtype	Number of sequences	Percentage
C	467	94.92%
A	2	0.41%
Recombinant C,A1	8	1.63%
Recombinant C,B	10	2.03%
Unassigned	5	1.02%
Total	492	100%

4. Align sequences in combined fasta file using MUSCLE in MEGA7 with default parameters as necessary.
  - Gap Open: -400
  - Gap Extend: 0
  - Max Iterations: 8
  - Clustering Method (Iteration 1, 2 and others): UPGMB
  - Min Diag Length (lambda): 24

# MARKDOWN

## Further reading:

[github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet/](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet/)

### Headers

```
# H1  
## H2  
### H3  
#### H4  
##### H5  
##### H6
```

Alternatively, for H1 and H2, an underline-ish style:

```
Alt-H1  
=====
```

```
Alt-H2  
=====
```

## H1

## H2

## H3

## H4

## H5

## H6

### Blockquotes

```
> Blockquotes are very handy in email to emulate reply text.  
> This line is part of the same quote.
```

Quote break.

```
> This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can put
```

Blockquotes are very handy in email to emulate reply text. This line is part of the same quote.

Quote break.

This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can put Markdown into a blockquote.

### Tables

Tables aren't part of the core Markdown spec, but they are part of GFM and *Markdown Here* supports them. They are an easy way of adding tables to your email -- a task that would otherwise require copy-pasting from another application.

Colons can be used to align columns.

Tables	Are	Cool
col 3 is	right-aligned	\$1600
col 2 is	centered	\$12
zebra stripes	are neat	\$1

There must be at least 3 dashes separating each header cell. The outer pipes (||) are optional, and you don't need to make the raw Markdown line up prettily. You can also use inline Markdown.

Markdown	Less	Pretty
\*Still\* | `renders` | \*\*nicely\*\*  
1 | 2 | 3

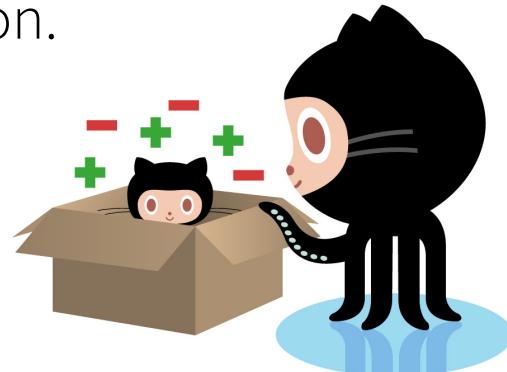
Colons can be used to align columns.

Tables	Are	Cool
col 3 is	right-aligned	\$1600
col 2 is	centered	\$12
zebra stripes	are neat	\$1

# KEY CONCEPS: MAKING A COMMIT

The process:

- Make some changes to a file
- Use the `git add` command or Atom to add files to the **staging area**
- Type a commit message (that shortly describes the changes you made since the last commit) into the commit message box, and click the Commit button.



# WRITING A GOOD COMMIT MESSAGE

The 7 rules of a great commit message:

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and *why* vs. *how*

	COMMENT	DATE
O	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
O	ENABLED CONFIG FILE PARSING	9 HOURS AGO
O	MISC BUGFIXES	5 HOURS AGO
O	CODE ADDITIONS/EDITS	4 HOURS AGO
O	MORE CODE	4 HOURS AGO
O	HERE HAVE CODE	4 HOURS AGO
O	AAAAAAA	3 HOURS AGO
O	ADKFJSLKDFJSDFKLJ	3 HOURS AGO
O	MY HANDS ARE TYPING WORDS	2 HOURS AGO
O	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

# WRITING A GOOD COMMIT MESSAGE

Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72 characters or so. In some contexts, the first line is treated as the subject of the commit and the rest of the text as the body. The blank line separating the summary from the body is critical (unless you omit the body entirely); various tools like `log`, `shortlog` and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you are making this change as opposed to how (the code explains that). Are there side effects or other unintuitive consequences of this change? Here's the place to explain them.

Further paragraphs come after blank lines.

- Bullet points are okay, too
- Typically a hyphen or asterisk is used for the bullet, preceded by a single space, with blank lines in between, but conventions vary here

If you use an issue tracker, put references to them at the bottom, like this:

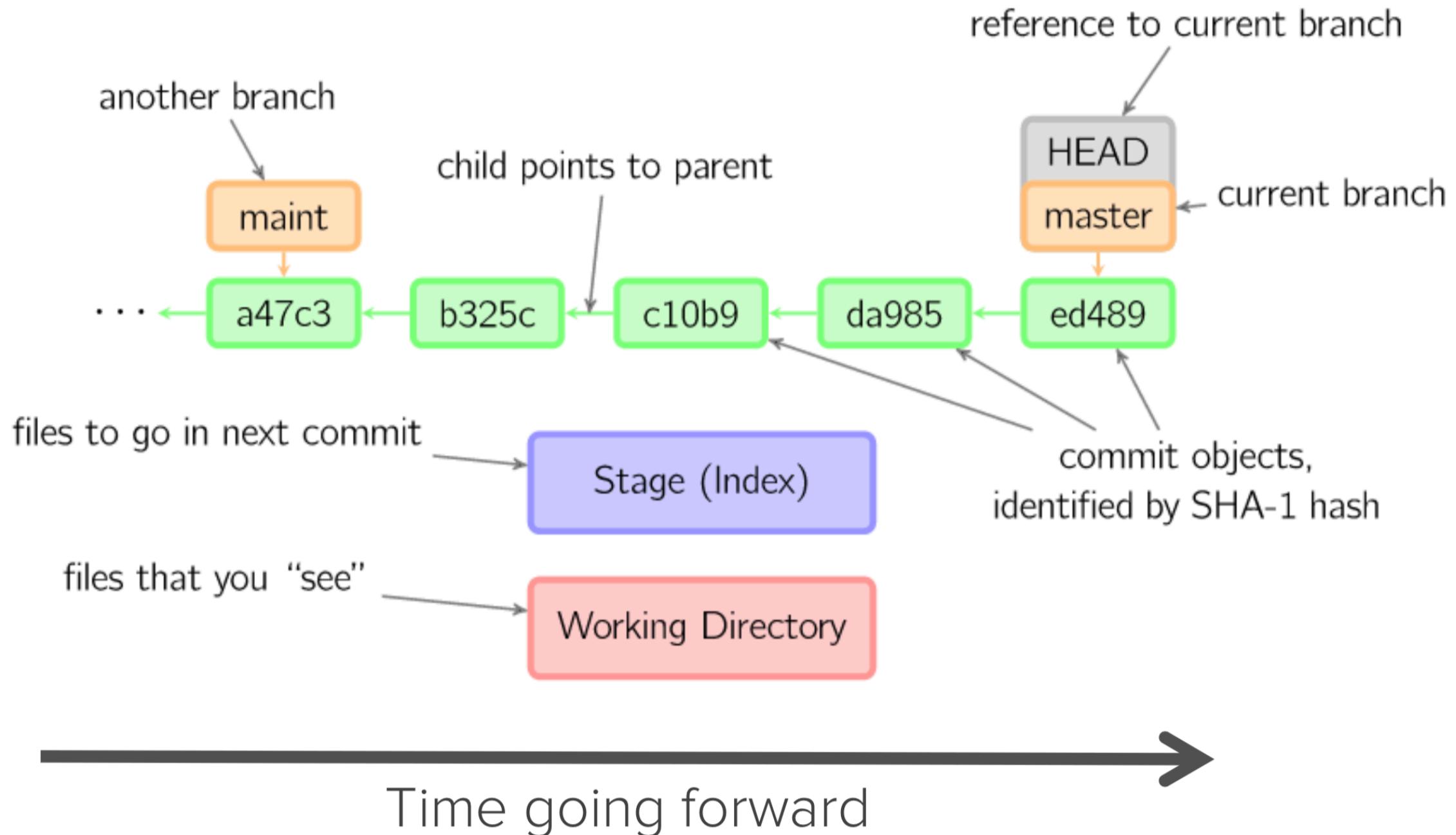
Resolves: #123

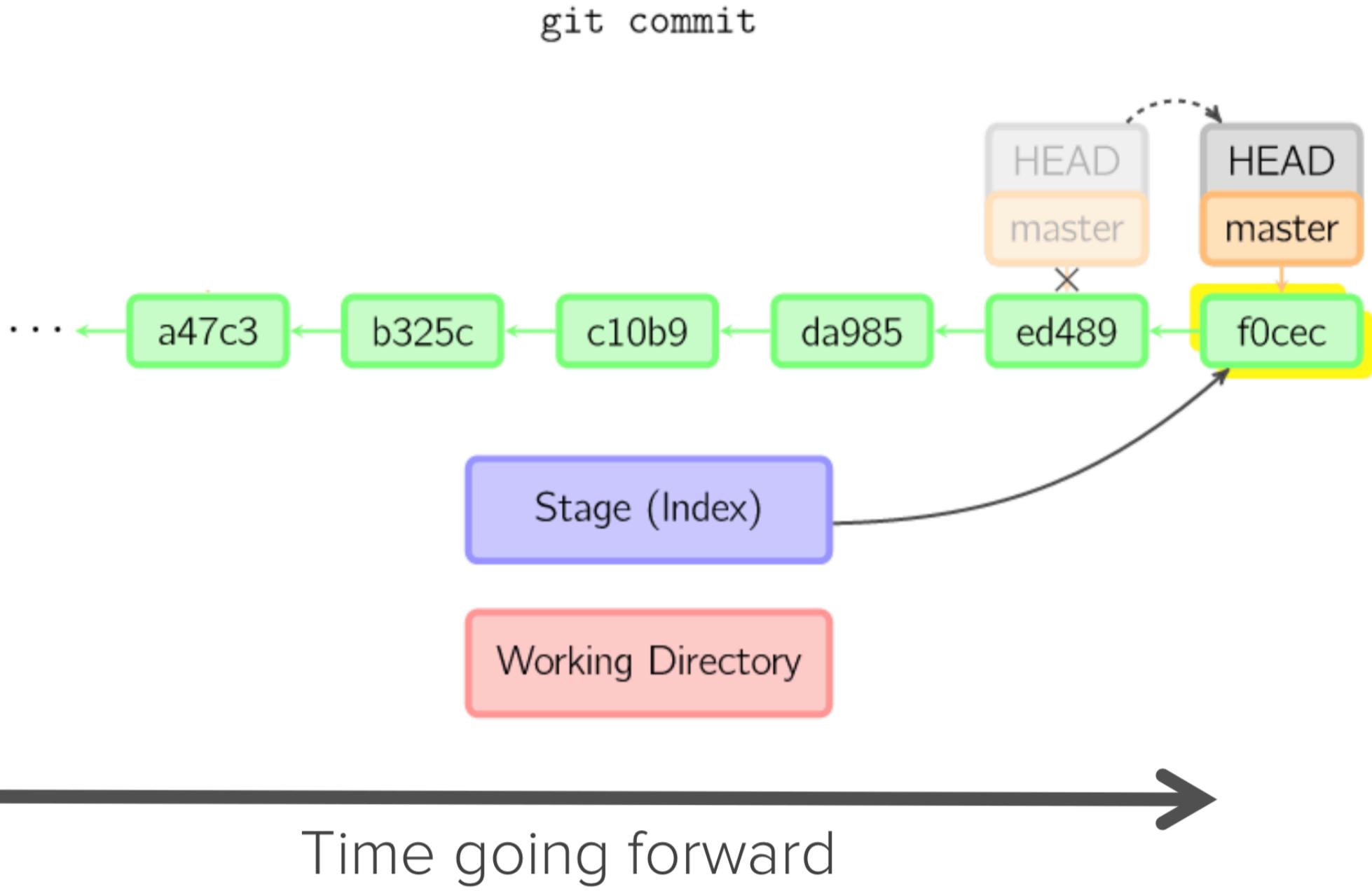
See also: #456, #789

## Further reading:

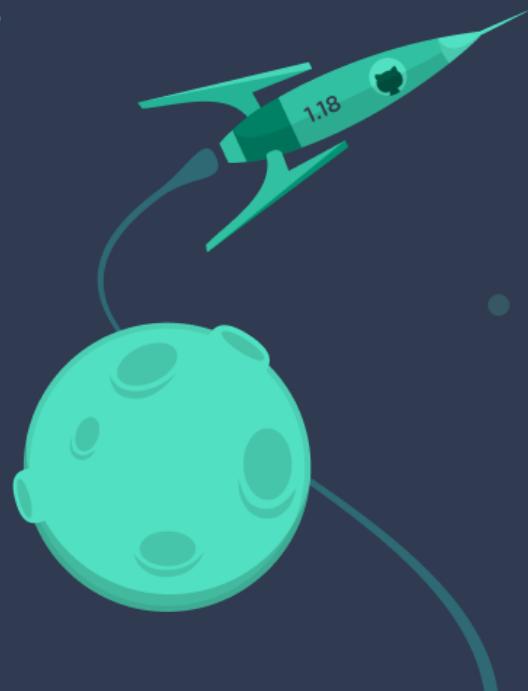
[chris.beams.io/posts/git-commit/](http://chris.beams.io/posts/git-commit/)



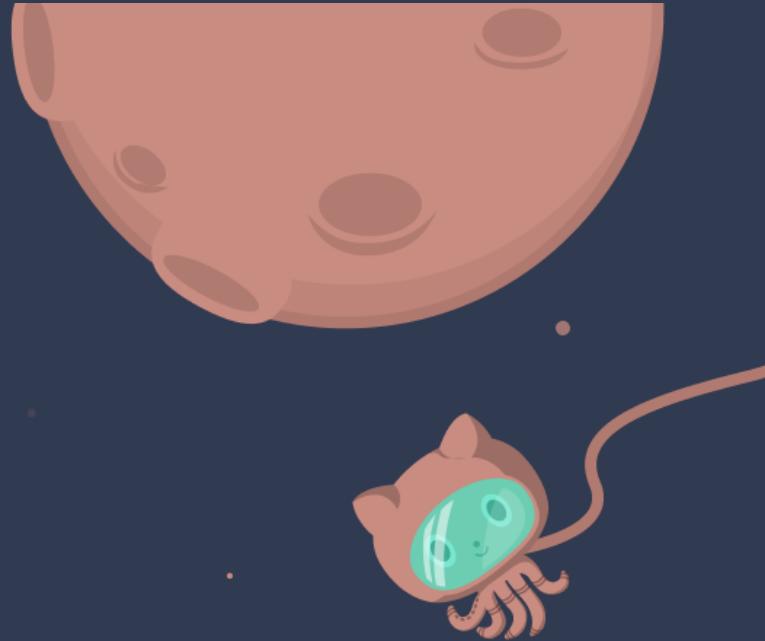




# USING GITHUB WITH ATOM



GitHub for Atom

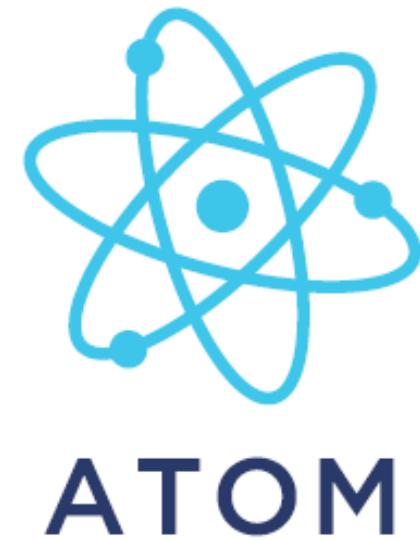


**Further reading:**

<https://www.hongkiat.com/blog/manage-git-github-atom/>

# WHAT IS ATOM?

- At it's core Atom is a plaintext editor that provides fully customizable package support
- Known as an **IDE** or Integrated Development Environment in computer science
- Offers integration with GitHub to conveniently manage your repos and commits in a user-friendly GUI format



# A FEW SETUP TIPS

- Follow steps to create an SSH key on your machine and associate it with your GitHub account:  
<https://help.github.com/en/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>
- Open the Git panel in Atom (CTRL+SHFT+9) and log in
- Execute commands by opening the command pallet (SHFT+CMD+P)
  - Type clone into the command pallet and paste a link from GitHub to setup a local copy of that repository



# A FEW SETUP TIPS

Make sure your email address is associated with Git so that commits are linked to the same GitHub account

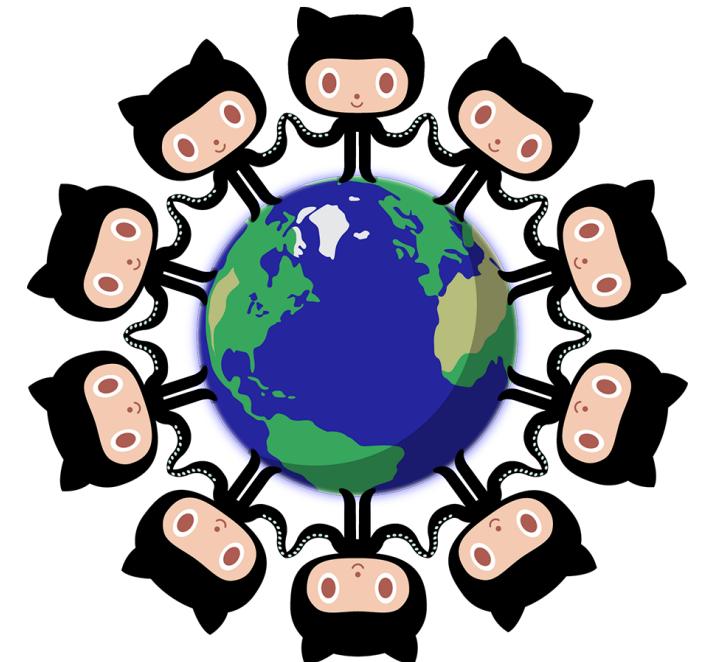
## Setting your email address for *every* repository on your computer

- 1 Open Terminal.
- 2 Set an email address in Git. You can use your GitHub-provided no-reply email address or any email address.

```
$ git config --global user.email "email@example.com"
```

- 3 Confirm that you have set the email address correctly in Git:

```
$ git config --global user.email  
email@example.com
```



# ADDITIONAL RESOURCES

**Official git site and tutorial:**

<https://git-scm.com/>

**GitHub guides:**

<https://guides.github.com/>

**Command cheatsheet:**

<https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf>

**Markdown Cheatsheet:**

<https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet>

**Interactive git tutorial:**

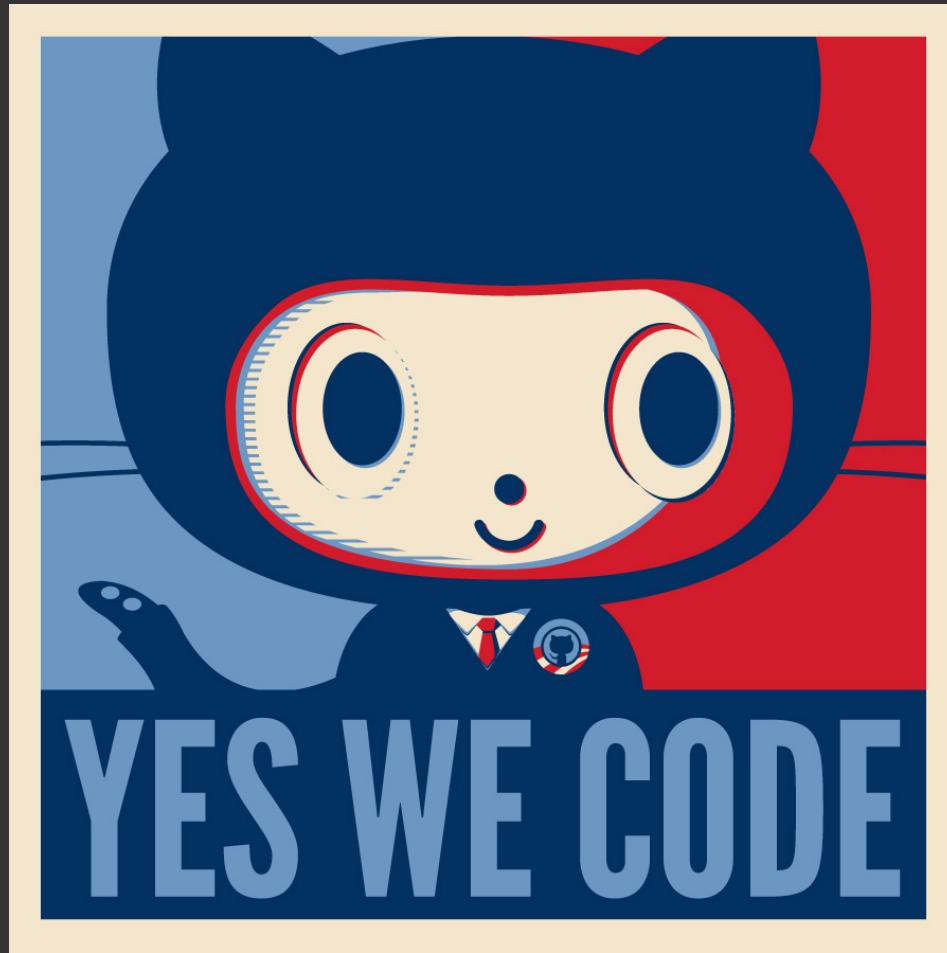
<https://try.github.io/levels/1/challenges/1>

**Visual/interactive cheatsheet:**

<http://ndpssoftware.com/git-cheatsheet.html>



# DEMO



Repurposing version control software for science  
represents a transparent, replicable, and streamlined  
process that fosters responsible collaboration