Lecture 18:

LIFE SKILL: CODE REUSE

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Outline

- Quick review of ”coding best practices”
- Some ethical questions
- Using online resources ethically
- How to attribute someone else’s code
- Version control: the great frustration mitigator ☺

Discussion

We've talked a lot about ideas like:

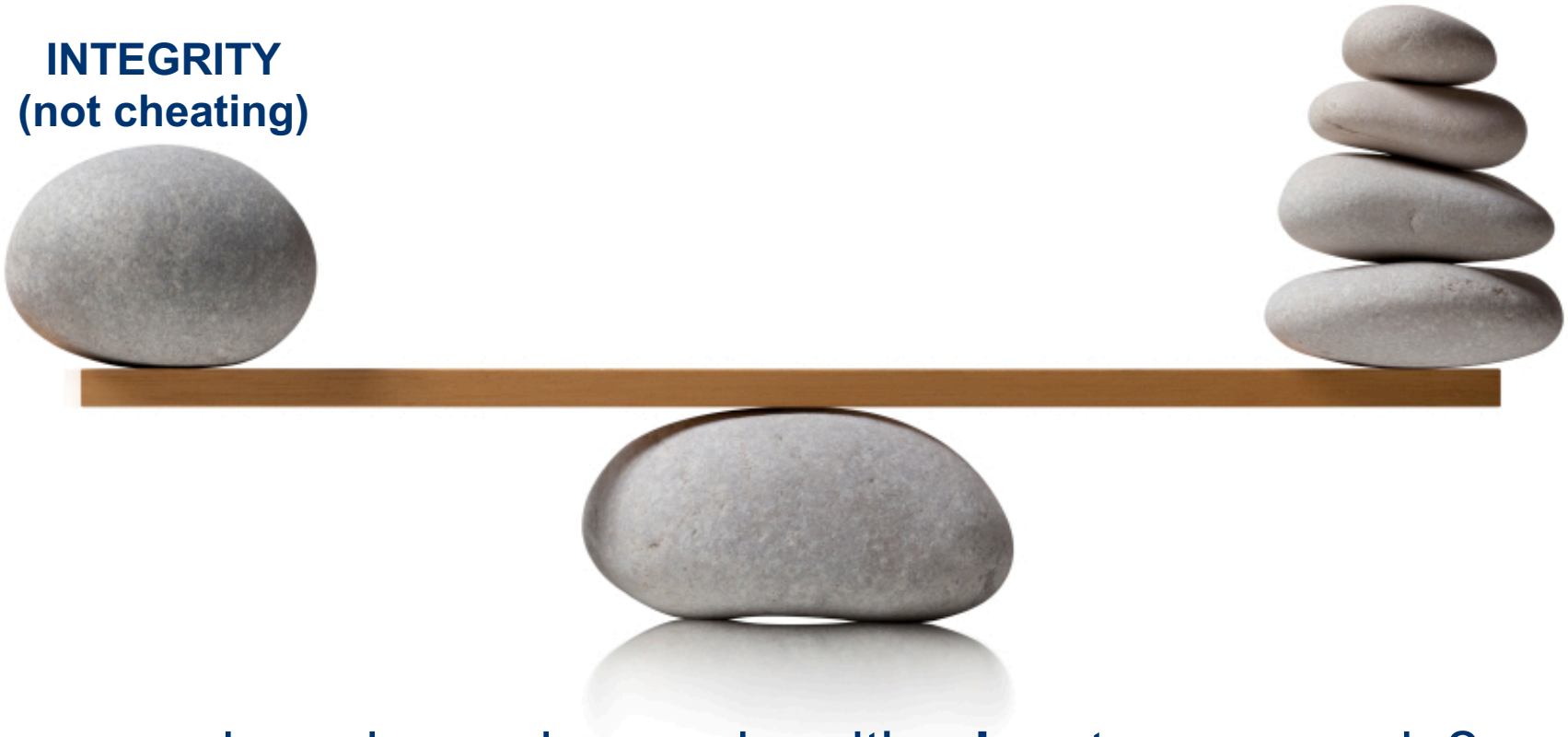
- Lecture 5: "S^4": start small | slow | simple
- Lecture 7: organizing your code so it's easy to reuse pieces
- Lecture 13: documenting your code so it's easy to come back to it
- Lecture 17: forking code from other people's respoitories

Can you think of any **ethical concerns** about this?

The balancing act…

EFFICIENCY
(not re-implementing)

INTEGRITY
(not cheating)



…so how do you know when it's **okay** to reuse code?

Let’s consider a more familiar case…

Same or different?

Dictionary

Enter a word, e.g. "pie"

pla·gia·rism

/ˈplājəˌrizəm/

noun

the practice of taking someone else's work or ideas and passing them off as one's own.
synonyms: copying, infringement of copyright, piracy, theft, stealing; *informal* cribbing
"accusations of plagiarism"

Translations, word origin, and more definitions

Scenario 0: self-reuse, not in a class

- You wrote a program that **solved a particular problem** as part of a project you're working on for fun
- Later on for **a different project** you're working on for fun, you need to solve the same problem
- Questions:
  - Can you reuse the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?

Scenario 1: self-reuse, in-class work

- You wrote a program that **solved a particular problem** for a previous assignment in a course
- Later on for **a project you're working on for fun**, you need to solve the same problem
- Questions:
  - Can you reuse the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?

Scenario 2: self-reuse, same course

- You wrote a program that **solved a particular problem** for a previous assignment in a course
- In a **later assignment for that same course**, you need to solve the same problem as part of a larger process
- Questions:
  - Can you reuse the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?
  - Does it matter if you copy/paste or **`import`** it?

Scenario 3: self-reuse, different course

- You wrote a program that **solved a particular problem** for a previous assignment in a course
- In **an assignment for a different course**, you need to solve the same problem
- Questions:
  - Can you reuse the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?
  - Does it matter if it's the **whole assignment**, or just one part?

Scenario 4: self-reuse, academic work →

- You wrote a program that **solved a particular problem** for an assignment in a course
- You later get a **job as a software engineer**, and you need to solve the same problem
- Questions:
  - Can you reuse the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?
  - Does it matter if it's the **whole assignment**, or just one part?

Scenario 5: professors and TAs

- You are trying to **solve a particular problem** for an assignment in a course, but you are stuck
- You ask the professor or TA for advice, they walk you through **how to implement one of the functions**
- Questions:
  - Can you use the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?

Scenario 6: peers

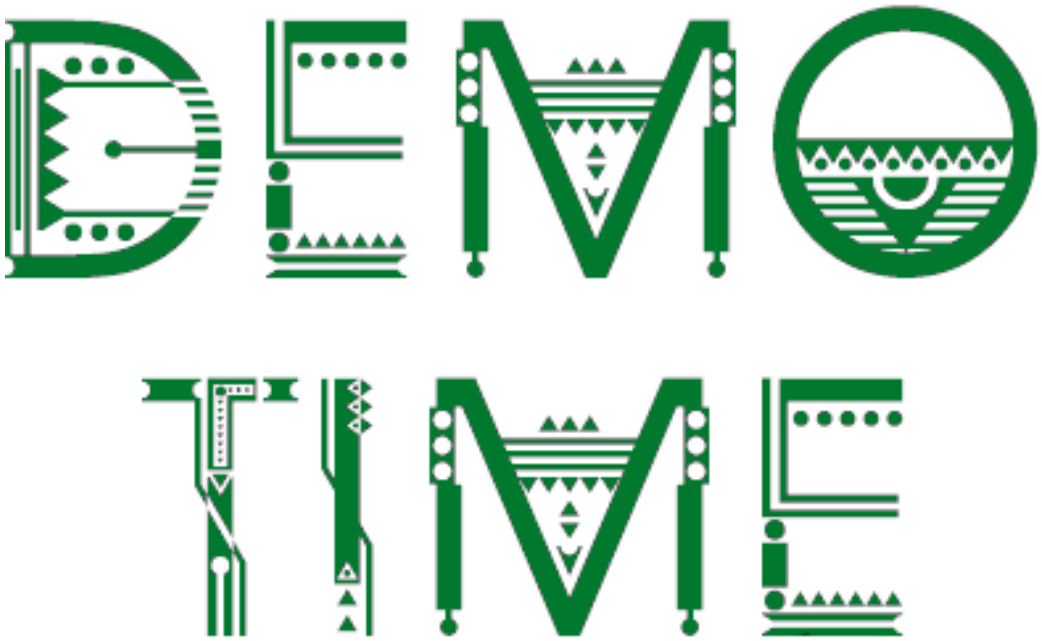
- You are trying to **solve a particular problem** for an assignment in a course, but you are stuck
- You ask a friend who took the class last year, they walk you through **how to implement one of the functions**
- Questions:
  - Can you use the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?

Scenario 7: online sources

- You are trying to **solve a particular problem** for an assignment in a course, but you are stuck
- You look online to try to understand a concept, someone walks through **how to implement one of the functions**
- Questions:
  - Can you use the code?
  - Does it matter what the code **does**?
  - Do you need to **attribute** the code?

Common online Q&A resources

Demo: using StackOverflow



How to attribute online code

```
# ------- START ATTRIBUTED CODE SECTION -------
# Code created with the help of Stack Overflow
# https://stackoverflow.com/questions/49581417
#
# Question by Alden:
# https://stackoverflow.com/users/9378177/alden
#
# Answer by CD Lane:
# https://stackoverflow.com/users/5771269/cdlane

# ...THE ACTUAL CODE GOES HERE...

# --------- END ATTRIBUTED CODE SECTION --------
```

Discussion

For educational purposes, does copyright matter?

Isn’t everything covered by “**fair use**”?

Rules for code reuse

- Always **attribute**
- Only use code you **actually understand**
- If it's for a course, **talk to the professor first**
- Understand **the license,** e.g. for StackOverflow

Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

Activity: code reuse

- With your group, pick a simple game (like
- Use **online resources** to find a working python implementation
- **One partner**: make a new repl containing the code you found, add your partners as **collaborators**
- Write up the **attribution**
- Make sure that you **understand each piece of the code** (add lots of comments!)

Show and tell

What did you **find**?

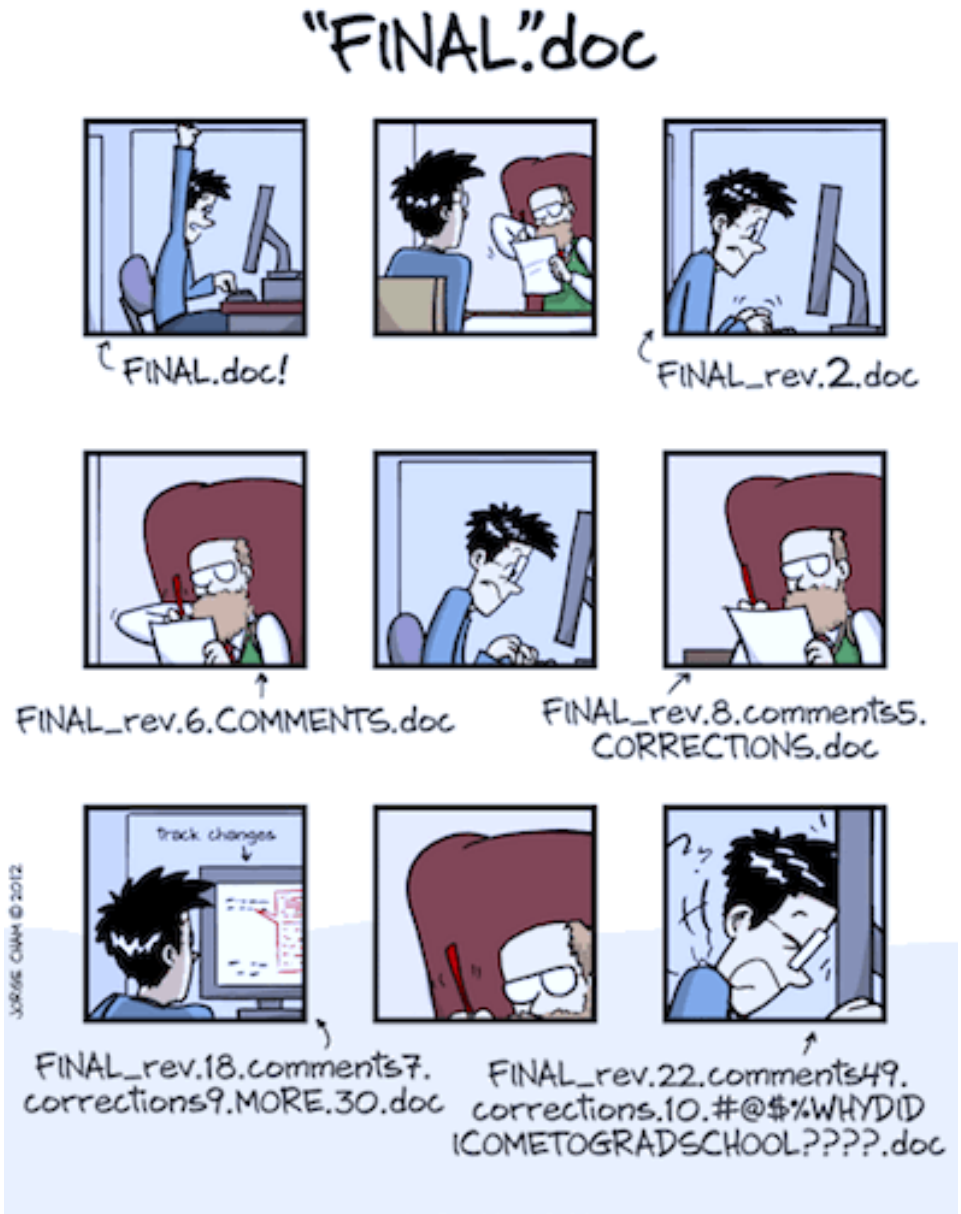
How does it **work**?

Version control

- A system that **records changes** to files over time
- Allows you to **restore** specific versions later if you need to

A common (terrible) pattern

(thanks to phdcomics.com for being hilarious and Ben Baumer for the following slides)

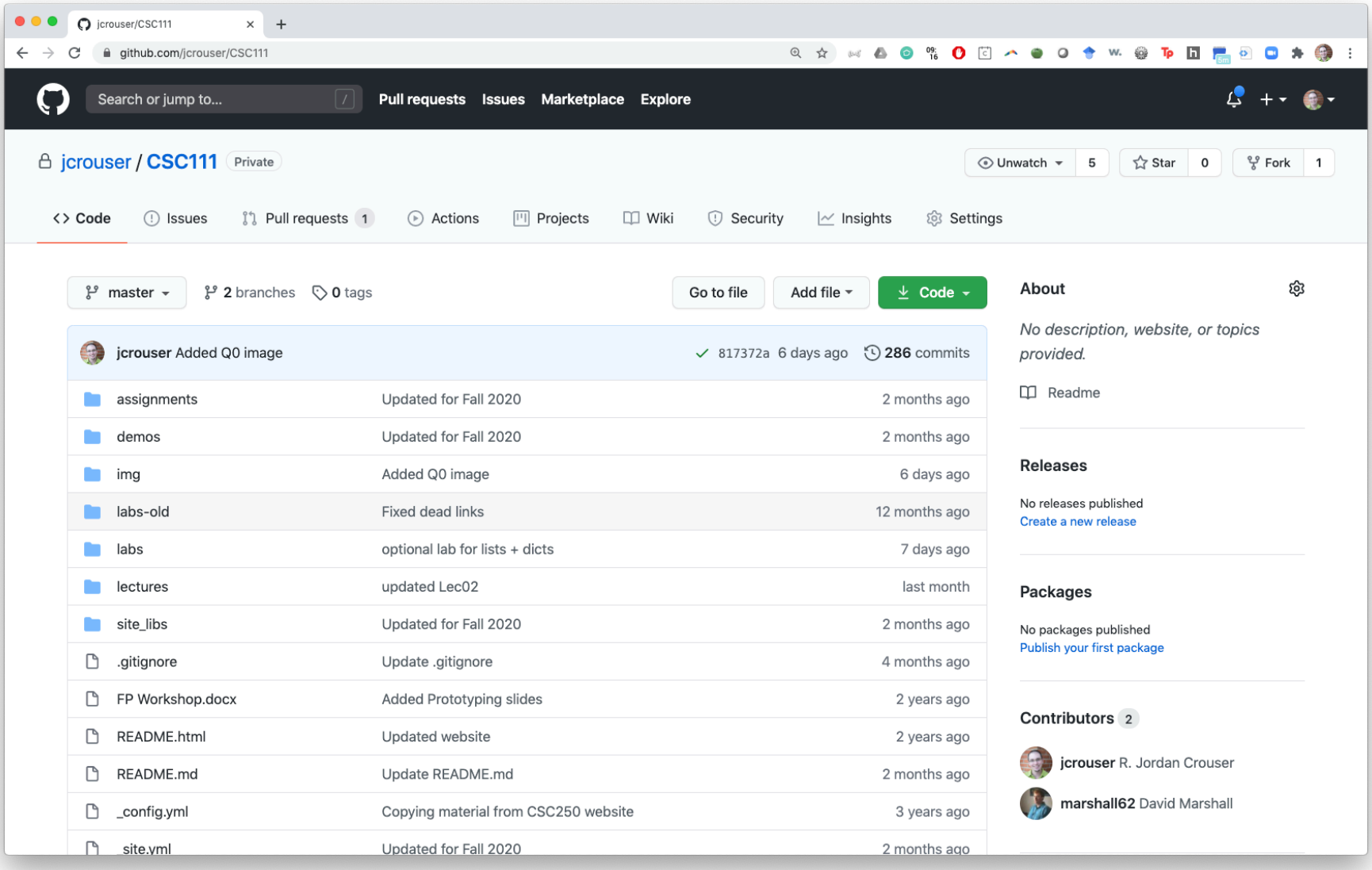


Slightly better

- `2019-07-25_1151am_hw5.py`
- `2019-07-25_0232pm_hw5.py`
- `2019-07-25_0603pm_hw5.py`
- `2019-07-25_0947pm_hw5.py`
- `2019-07-25_1153pm_hw5.py`
- `2019-07-26_0117am_hw5.py`

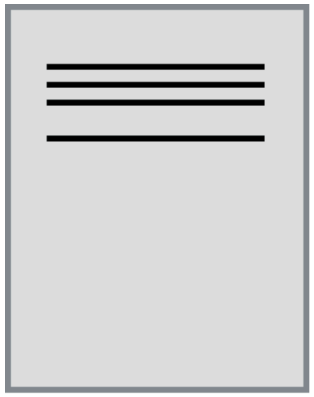
...but what if you have multiple files working together?

Best: version control tools like **`git`**



Version control 101

- Start with a base version
- Instead of saving lots of copies, just save just the **changes** (and note where they belong)



Think of it like a tape: can rewind & play back each change

Image source: software-carpentry.org

Collaboration with version control

- **Multiple people** can contribute changes
- “Playing back” different sets of changes = different **versions** of the document

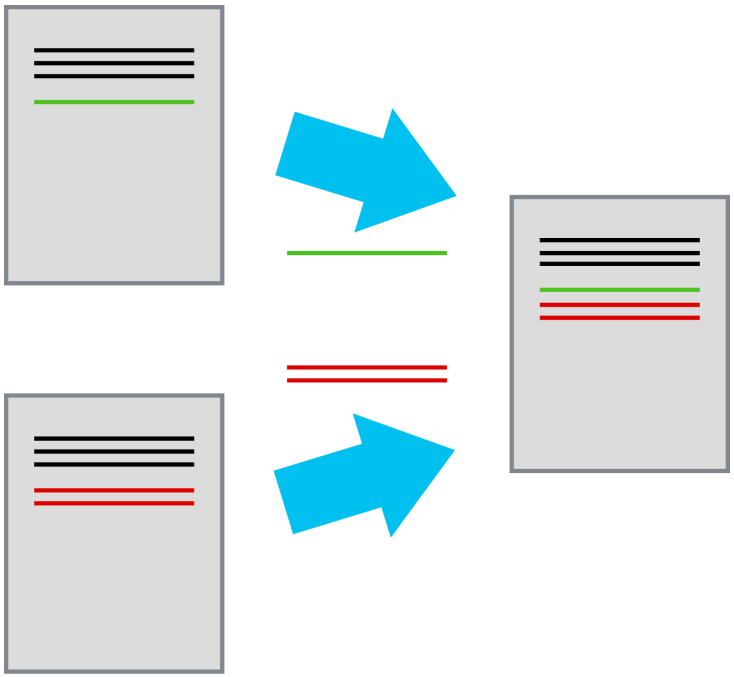


Image source: software-carpentry.org

Git and GitHub

`git`

- Command-line interface (CLI) that does the work*
- OS-specific installation
- Needed to use GitHub
- `git pull`, `git push`, etc.

- hosting for git projects
- strong community
- additional tools to enhance collaboration
- a common location to share your work

* you also have a CLI for python!

Commits

Commits on Jul 25, 2019

Fixed dead links

jcrouser committed 3 days ago ✓
01f96c3

Added Lab 16

jcrouser committed 4 days ago ✓
7c7feae

Adding Lab 12

jcrouser committed 4 days ago ✓
131090e

Commits on Jul 24, 2019

Fixed lab date

jcrouser committed 4 days ago ✓
135e881

Added detail on exam scores

jcrouser committed 5 days ago ✓
72ee277

Diff (“difference”)

Fixed dead links

master

Browse files

jcrouser committed 3 days ago — 1 parent 7c7feae — commit 01f96c315aa6c2d05f9bc59df258057d6af165a2

Showing 9 changed files with **39 additions** and **8 deletions**.

Unified | Split

4 README.html

```
@@ -339,8 +339,10 @@
339       <li>
340         <a href="labs/lab-11-classes.html">11. Slow Pokémon</a>
341       </li>
342 +     <li>
343 +       <a href="labs/lab-12-recursion.html">12. Fractals</a>
344 +     </li>
342       <li class="dropdown-header">12. Sorting</li>
343 -     <li class="dropdown-header">13. Fractals</li>
344       <li class="dropdown-header">14. Making a Scene</li>
345       <li class="dropdown-header">15. Fish Tank</li>
346       <li class="dropdown-header">16. Interactive Fish Tank</li>
```

This seems complicated… why bother?

- Automatic tracking of changes
- No download/upload/emailing of files
- Easy collaboration, with ability to resolve merge conflicts
- Searchability
- Can track issues, releases, and milestones
- Visibility:
  - GitHub repositories are **public**
  - Can request free **private** repositories for education or research

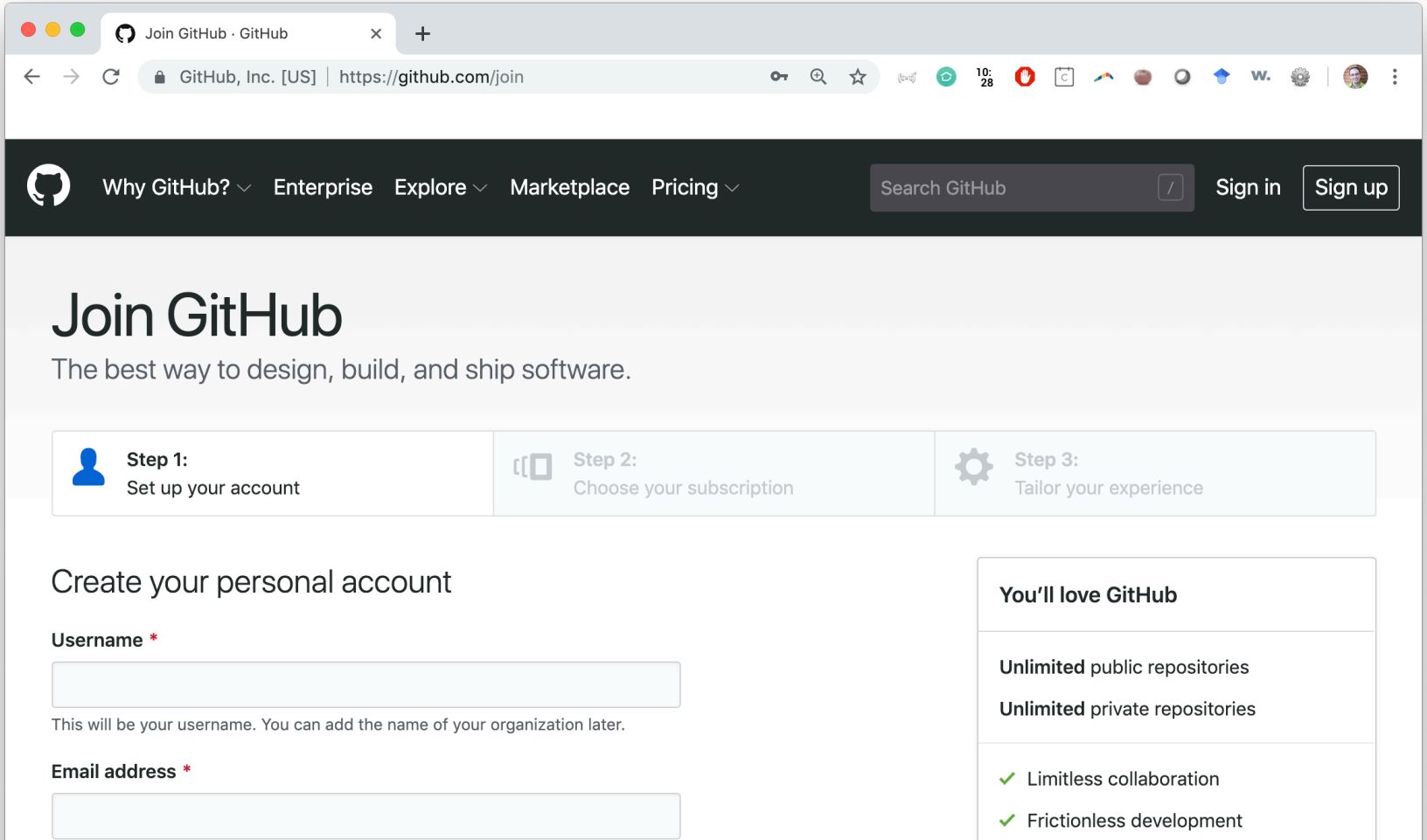
Alternatives

- **Pros**: easy, seamless
- **Cons**: no simultaneous editing, limited version control (no branches)

- **Pros**: easy, simultaneous editing
- **Cons**: no version control, not actually a `.py` file (can't run it)

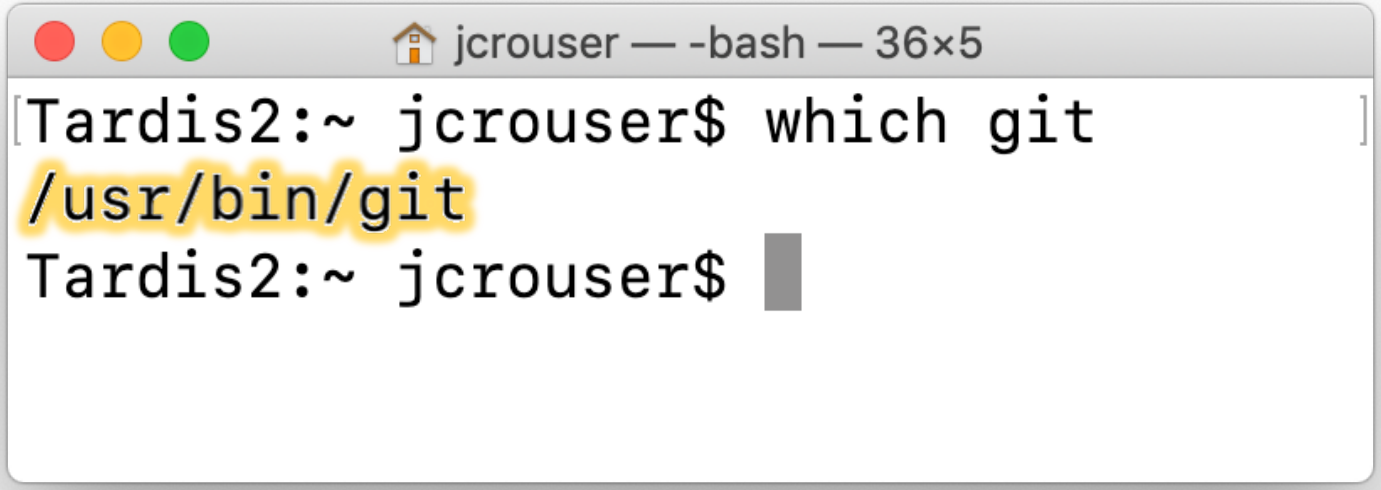
Getting started with `git` (a walkthrough)

- **Step 1**: go to `github.com/join` and create an account



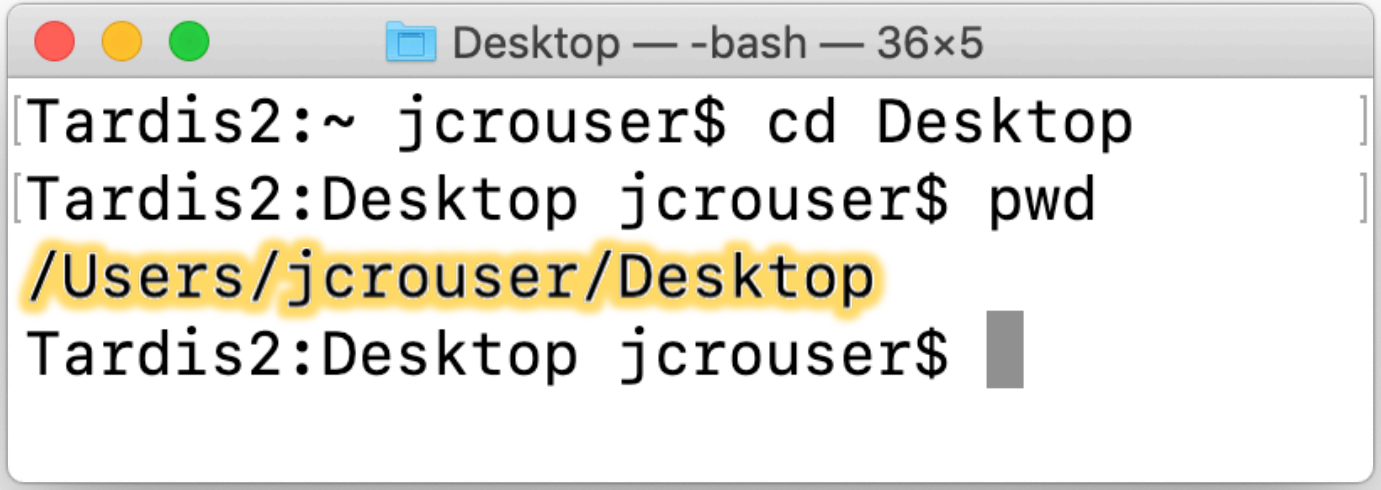
Getting started with **`git`** (a walkthrough)

- **Step 2:** make sure `git` is installed (it should be)
  - open the **Terminal app** (not IDLE)
  - type **`which git`** at the prompt and hit return



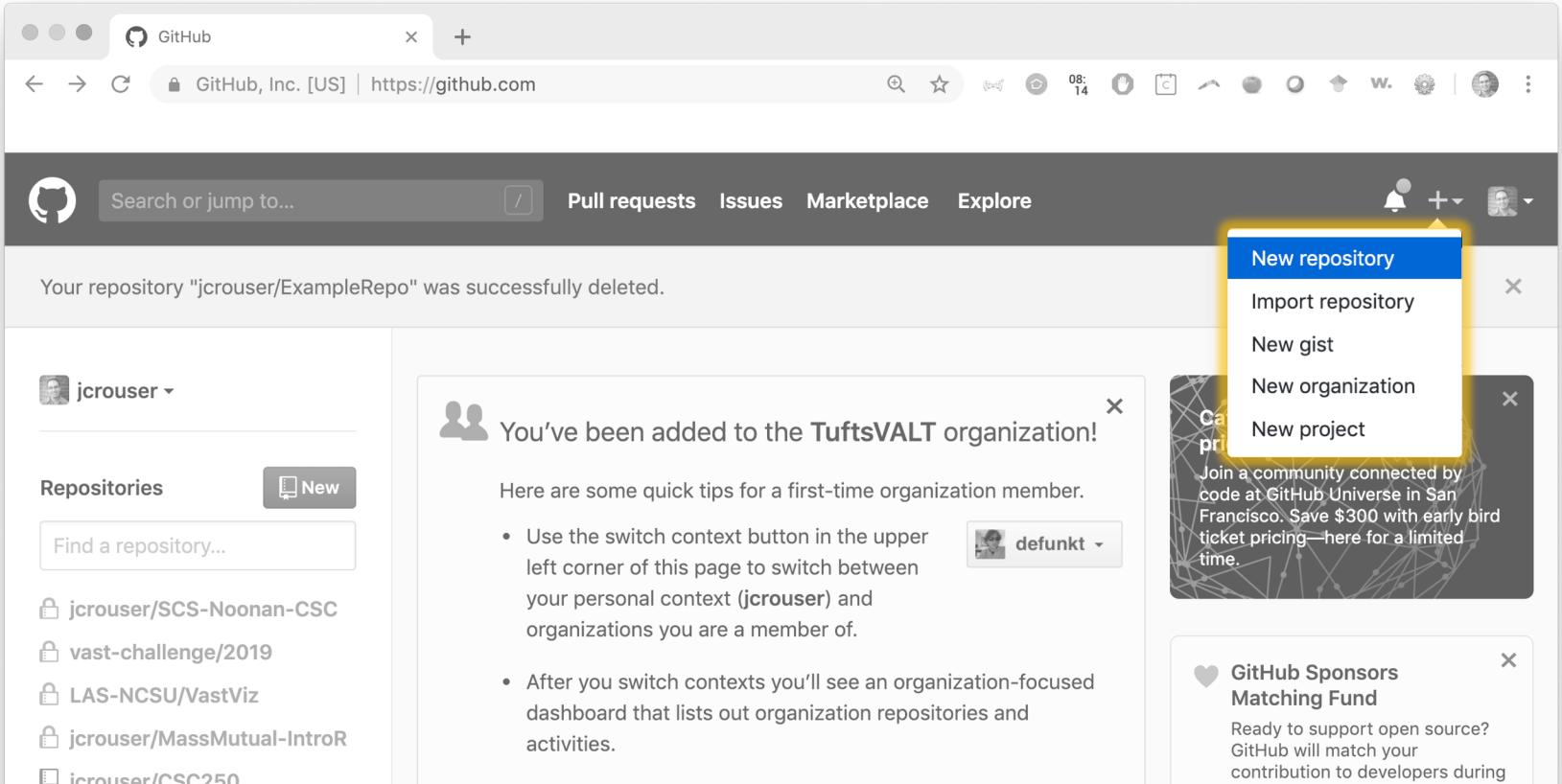
Getting started with **`git`** (a walkthrough)

- **Step 3:** navigate to someplace useful, like the Desktop
  - type **`cd Desktop`** (“change directory”) at the prompt and hit return
  - If you want, type **`pwd`** (“print working directory”) to see the path



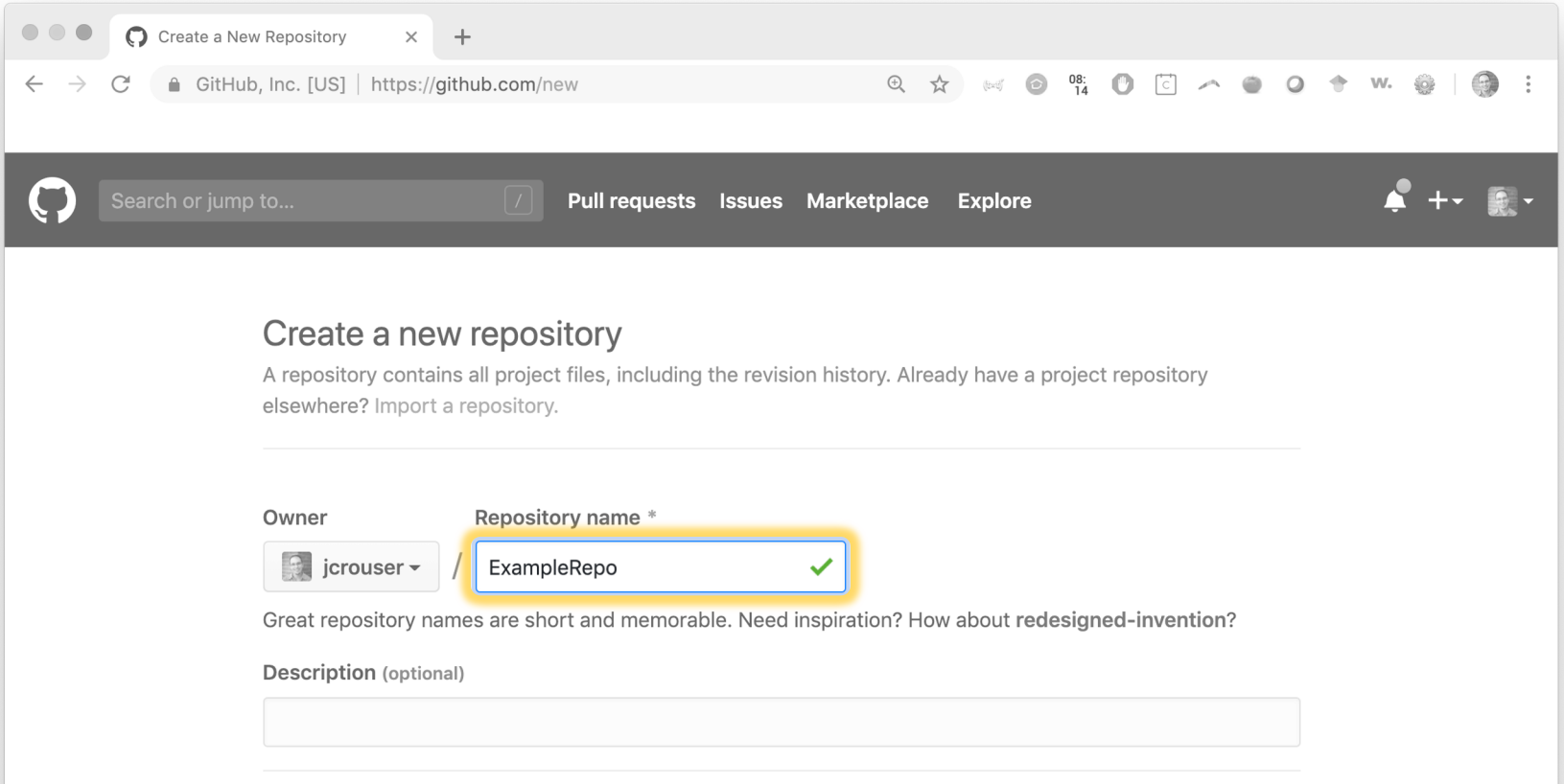
Getting started with `git` (a walkthrough)

- **Step 4a:** to make a **new** repository
  - Click the “+” icon in the upper right corner of the page
  - Select “New repository” from the menu



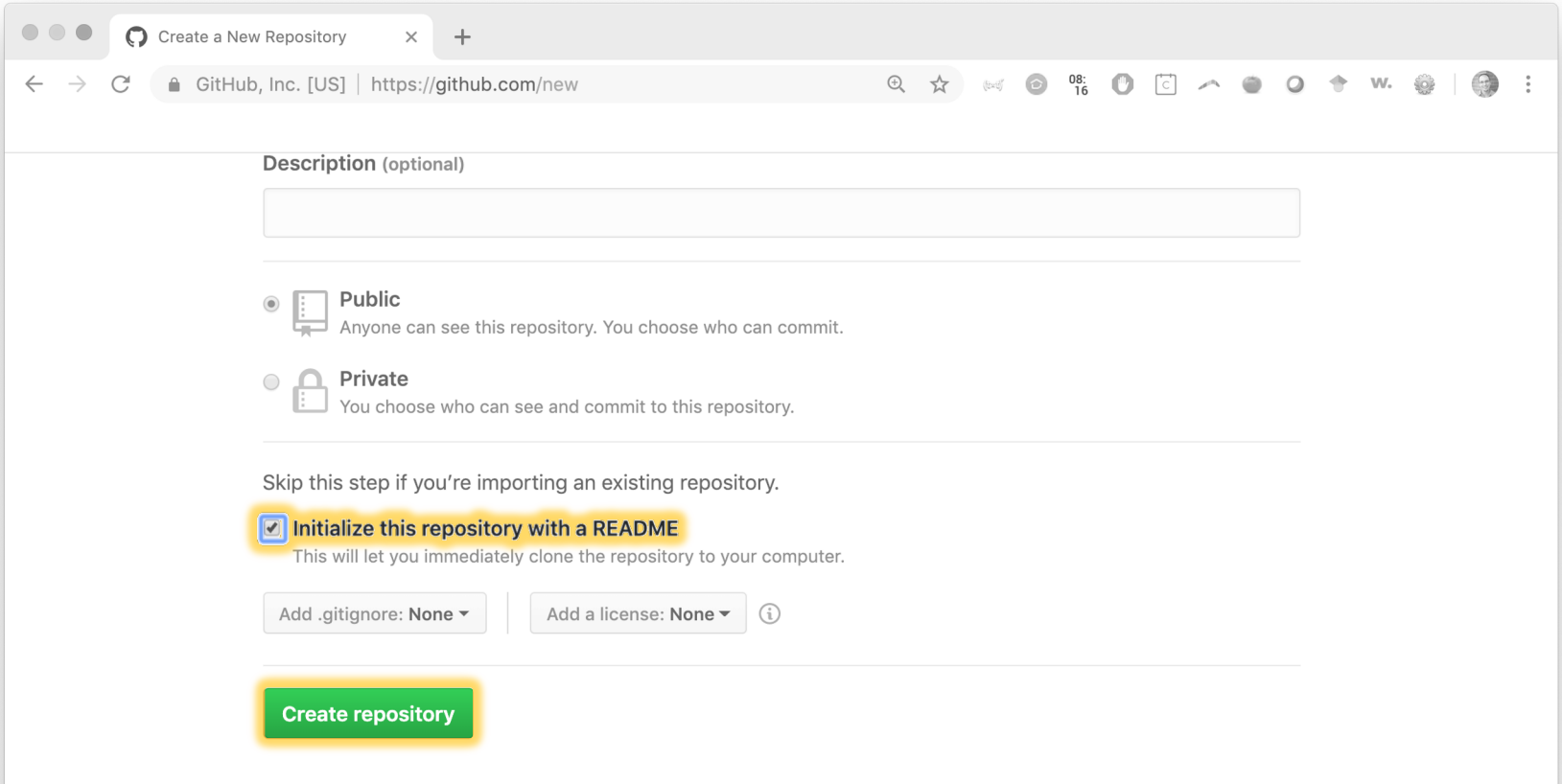
Getting started with `git` (a walkthrough)

- **Step 4b:** set up your **new** repository
  - Give your repository a name (like **`ExampleRepo`**)



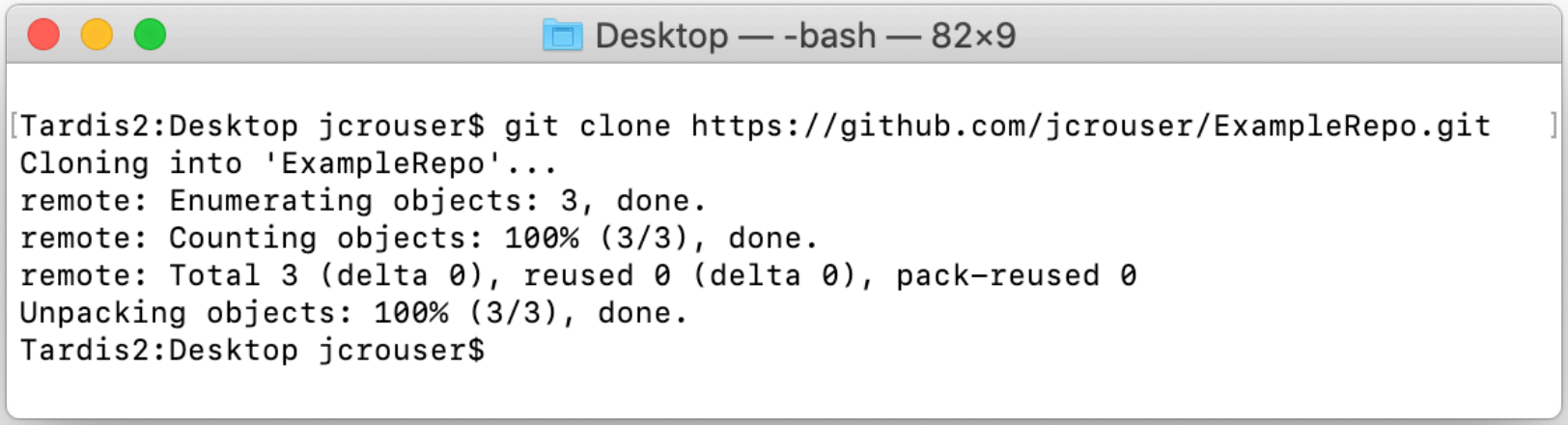
Getting started with `git` (a walkthrough)

- **Step 4c:** set up your **new** repository
  - Check the box “Initialize this repository with a README”
  - Click the green “**Create Repository**” button



Getting started with **`git`** (a walkthrough)

- **Step 5:** clone your repository to your **computer**
  - go back to the **Terminal** (command line)
  - **`git clone https://github.com/yourname/ExampleRepo.git`**



  - look on the Desktop and you'll see a new folder called **`ExampleRepo`**
  - take a look inside!

Getting started with **`git`** (a walkthrough)

- **Step 6:** adding a new file to the repository
  - make a file called **`demo.py`**
  - save it in **`ExampleRepo`**

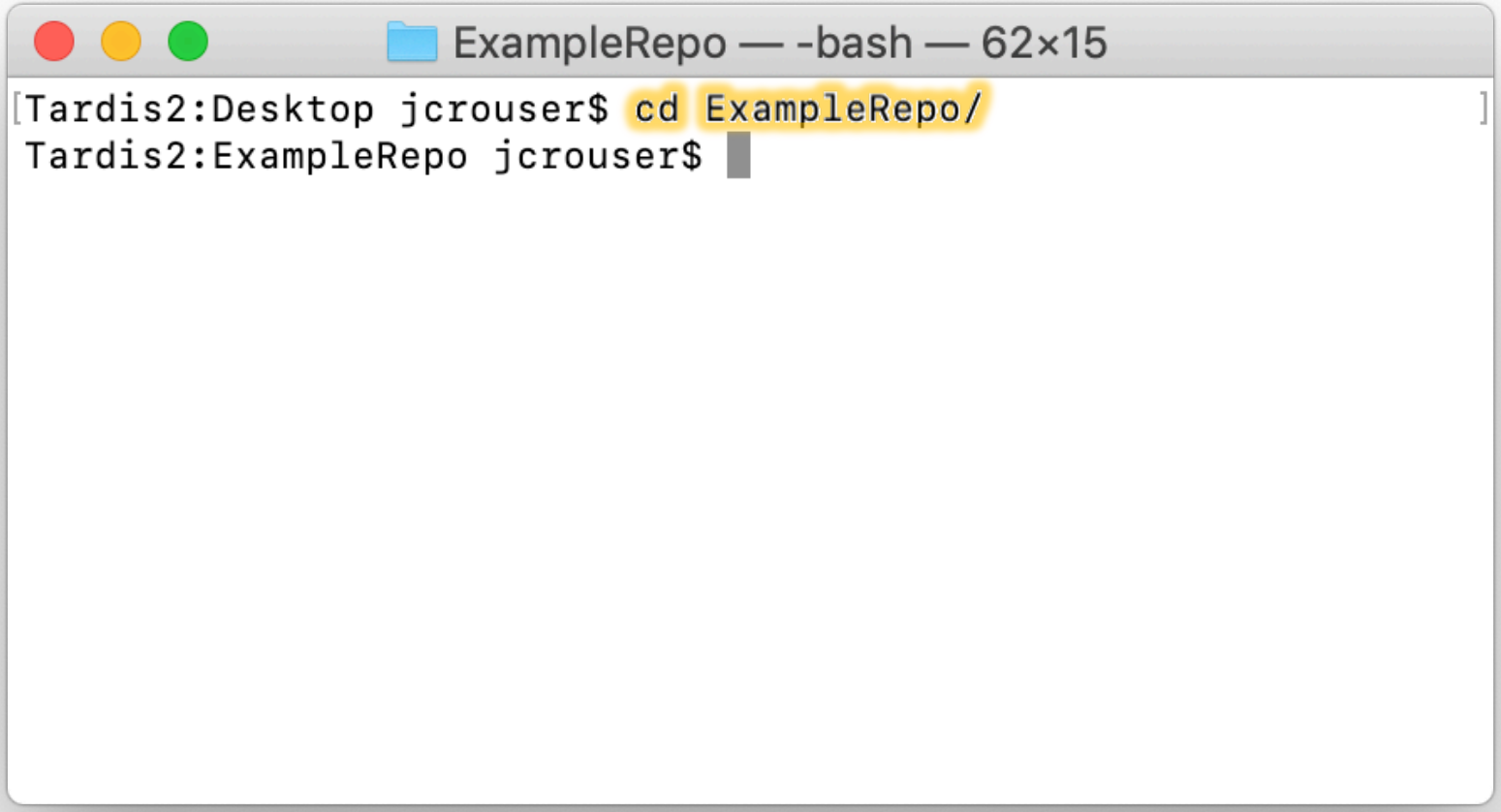
demo.py - /Users/jcrouser/Desktop/ExampleRepo/demo.py (3.7.4)

```
#----------------------------------------------------------
#     Name(s): R. Jordan Crouser
#        Date: 28 July 2019
#
# Description: This is a sample python file,
#              including a header and a main function
#----------------------------------------------------------

def main():

    name = "Jordan"
    print("Hello, my name is", name)

if __name__ == "__main__":
    main()
```

Ln: 16 Col: 0

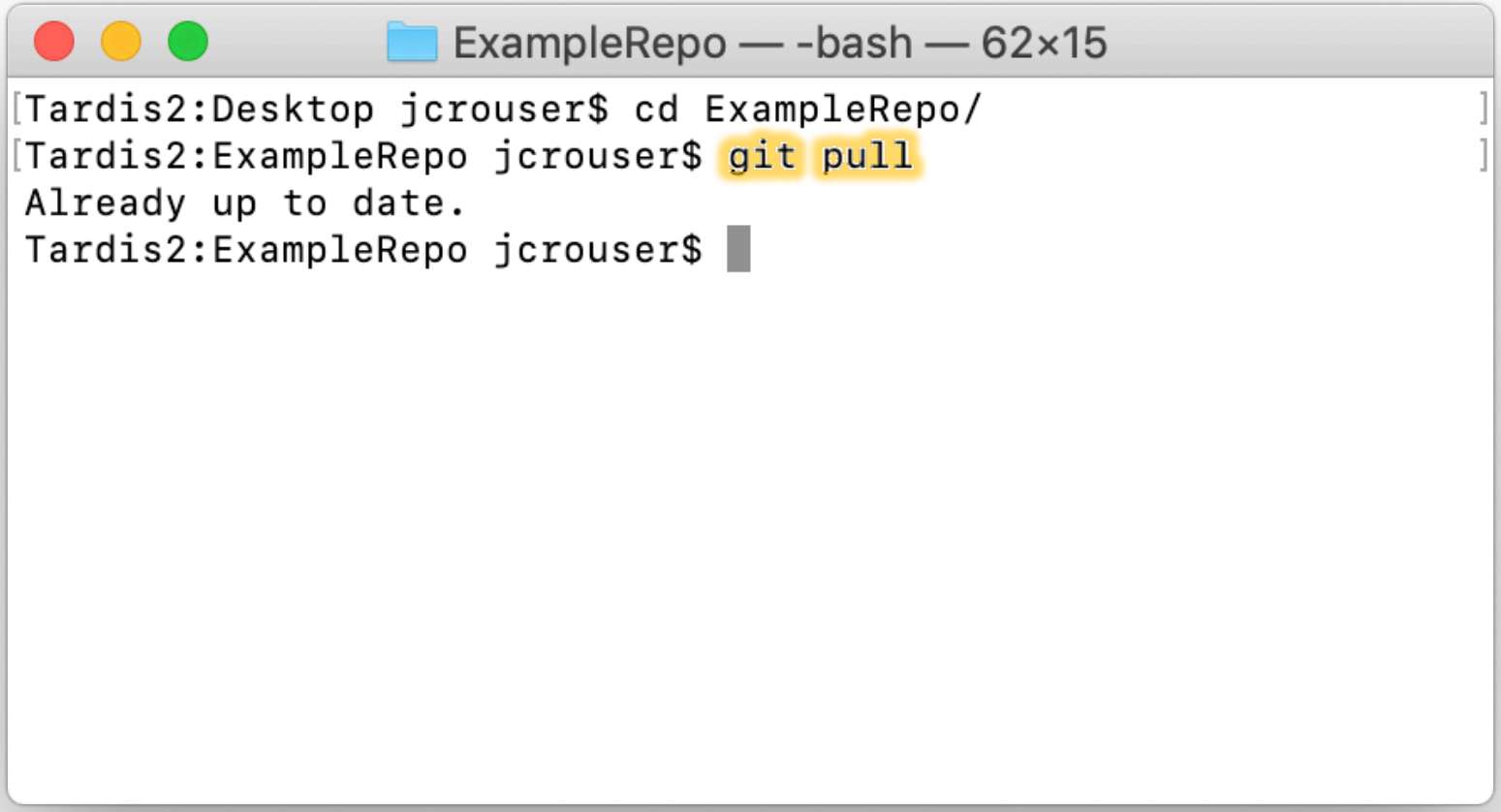
Getting started with **`git`** (a walkthrough)

- **Step 7a:** your first **commit**
  - type **`cd ExampleRepo`** to move into the new directory



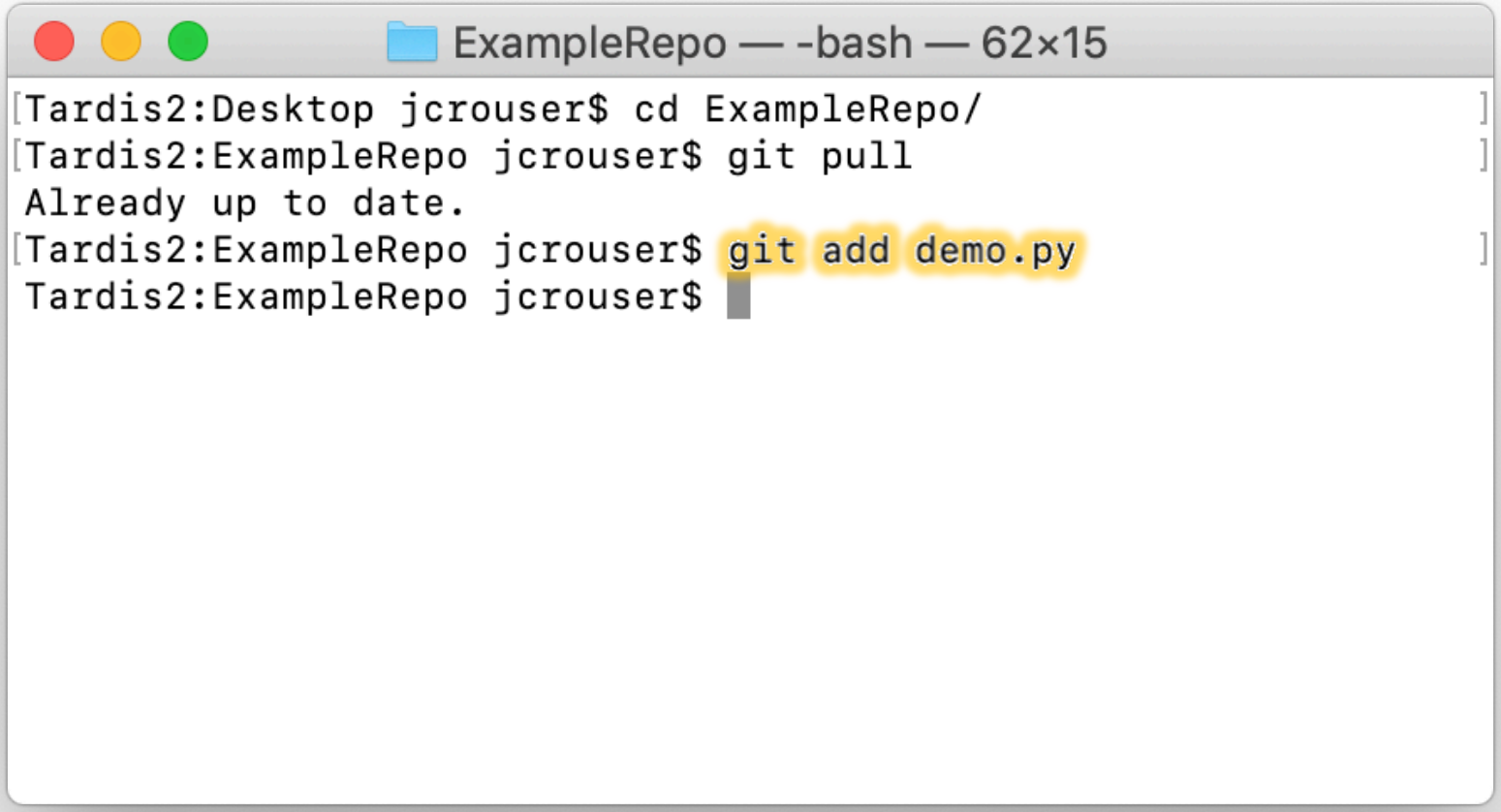
Getting started with **`git`** (a walkthrough)

- **Step 7b:** your first **commit**
  - type **`git pull`** to make sure your local version is up-to-date



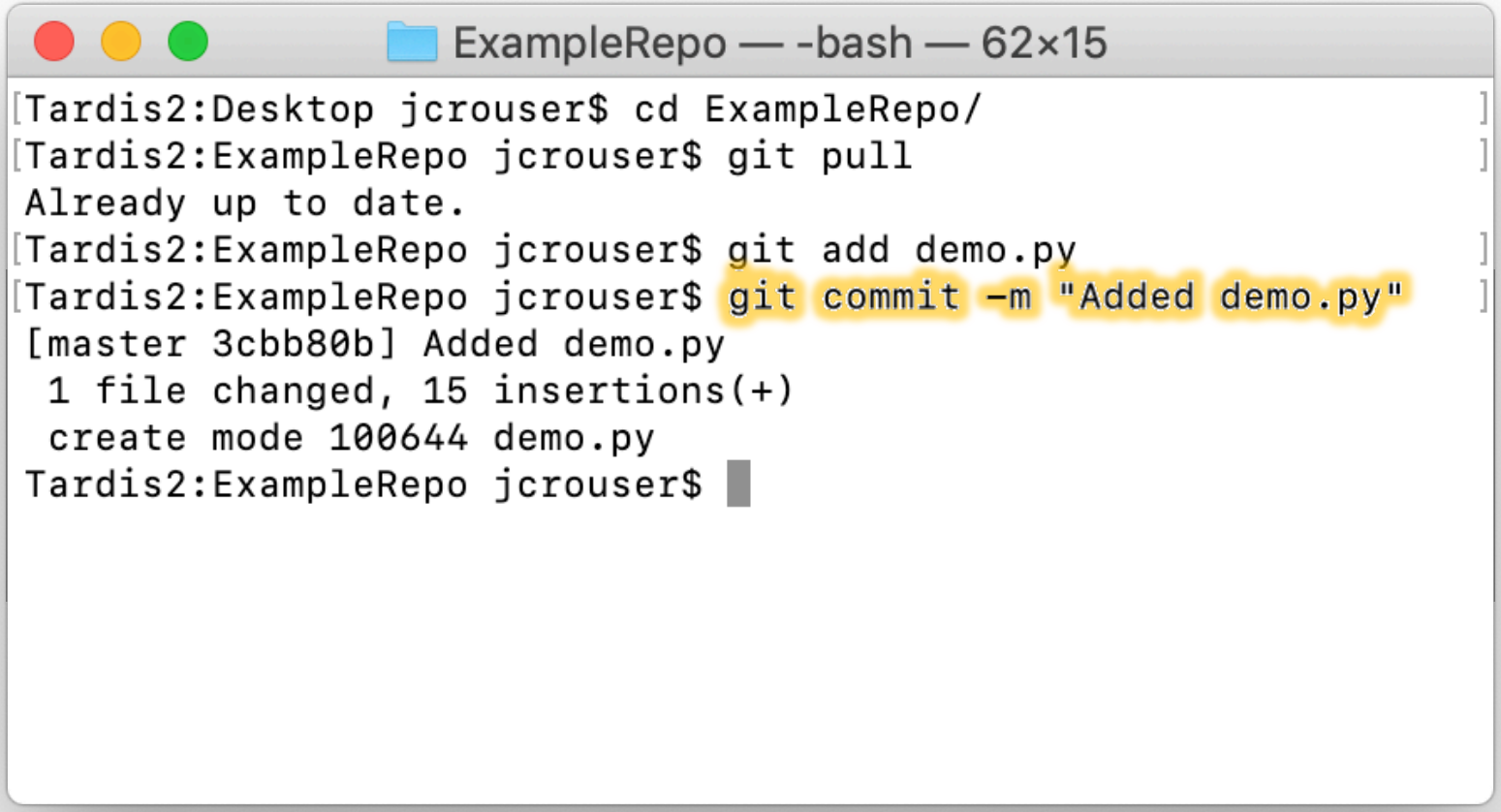
Getting started with **`git`** (a walkthrough)

- **Step 7c:** your first **commit**
  - type **`git add demo.py`** to "stage" your changes



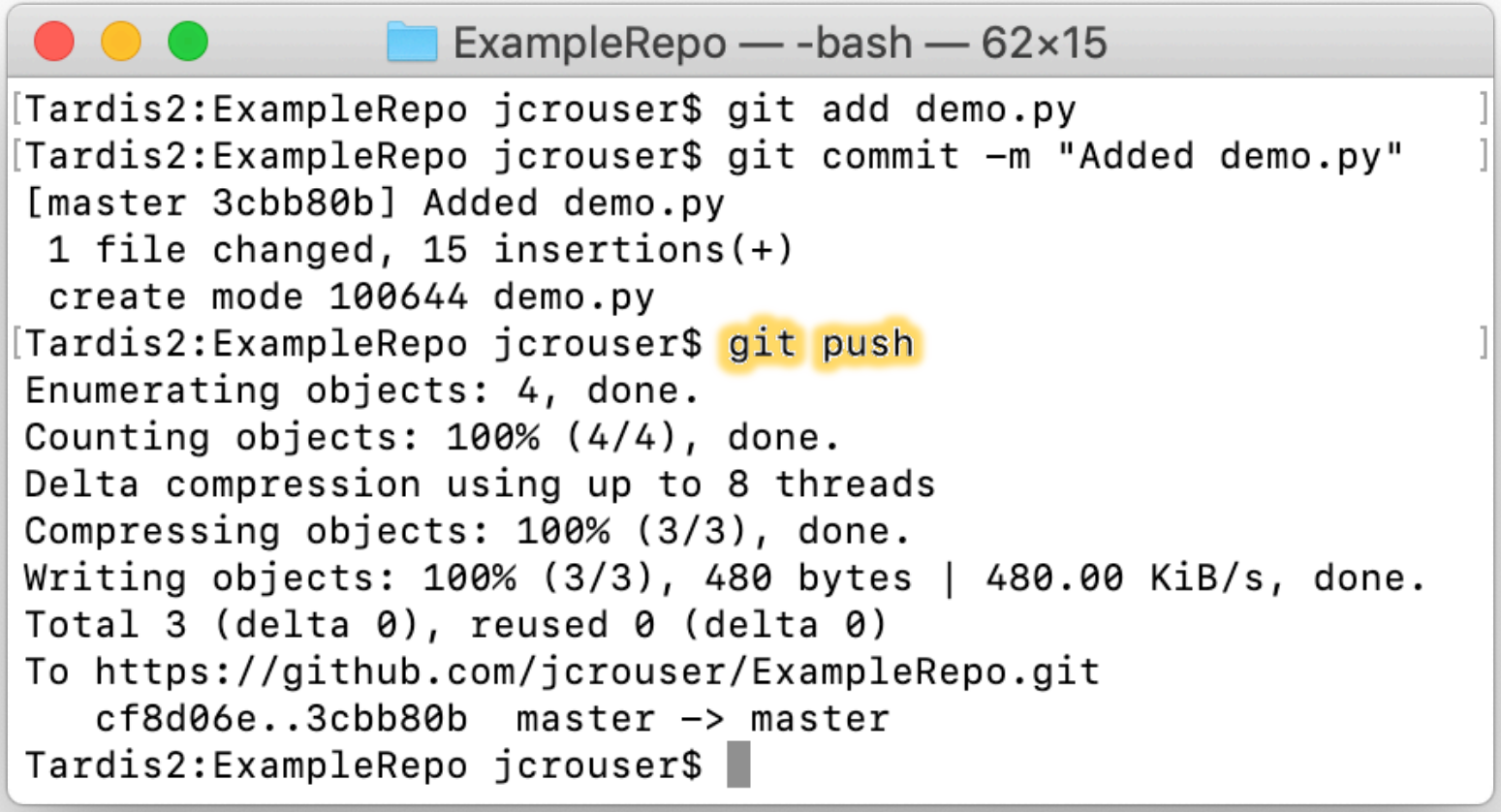
Getting started with **`git`** (a walkthrough)

- **Step 7d:** your first **commit**
  - type **`git commit –m “Added demo.py”`** to commit the changes, including a message about what you did



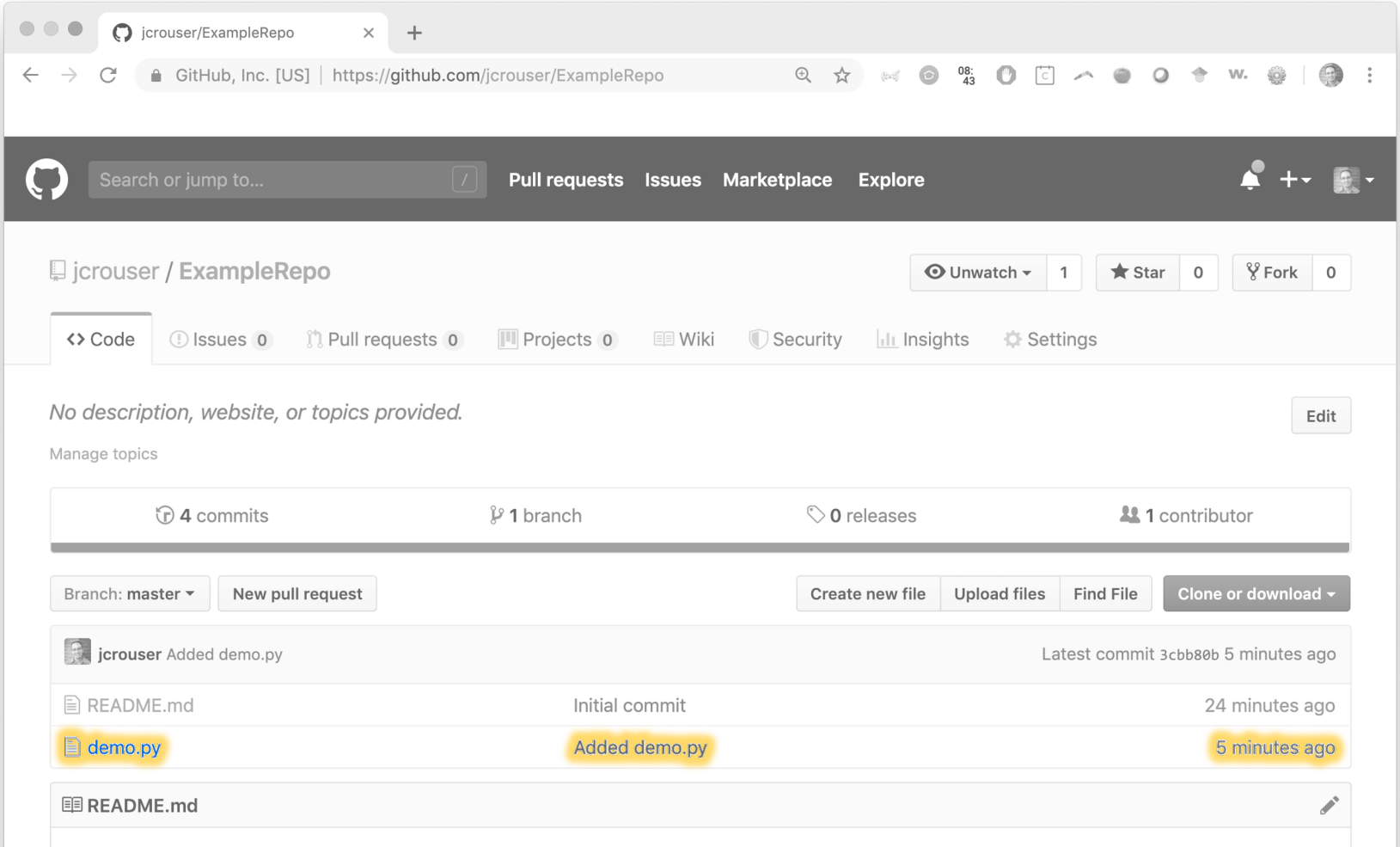
Getting started with **`git`** (a walkthrough)

- **Step 7e:** your first **commit**
  - type **`git push`** to push the changes to the repository



Getting started with `git` (a walkthrough)

- **Step 8:** go take a look at your repository!



Basic git procedure

1. **pull** any recent changes (`git pull`)
2. **make edits** on your computer
3. **commit** your local changes (`git add` / `git commit`)
   - this step might be repeated many times
4. **pull** again to avoid merge conflicts (`git pull`)
5. **push** your commit(s) to GitHub (`git push`)

GitHub advice

- This is all you really need to know for now
- Make small, frequent commits!
- Always **pull** before you **push**
- Slack message us if you get into trouble
- Cheatsheets / more advanced stuff (like pull requests):
  - https://education.github.com/git-cheat-sheet-education.pdf
  - https://www.git-tower.com/blog/git-cheat-sheet/

Up next

- ✓Monday: Life skill #4 - code reuse
- Wednesday: **Recursion pt. 2**
- Friday: Classes pt. 1 – Attributes and Methods
- **Midterm due Friday at 9am**