An Introduction to Version Control with git and GitHub

Modern Data Structures
GR 5072

Dr. Anderson

Agenda

- What is version control software?
- git and GitHub
- Using git and Github Desktop
- End to end process for making updating project code

What is Version Control software?

The three aims of a Data Science project…

And how version control software supports these aims

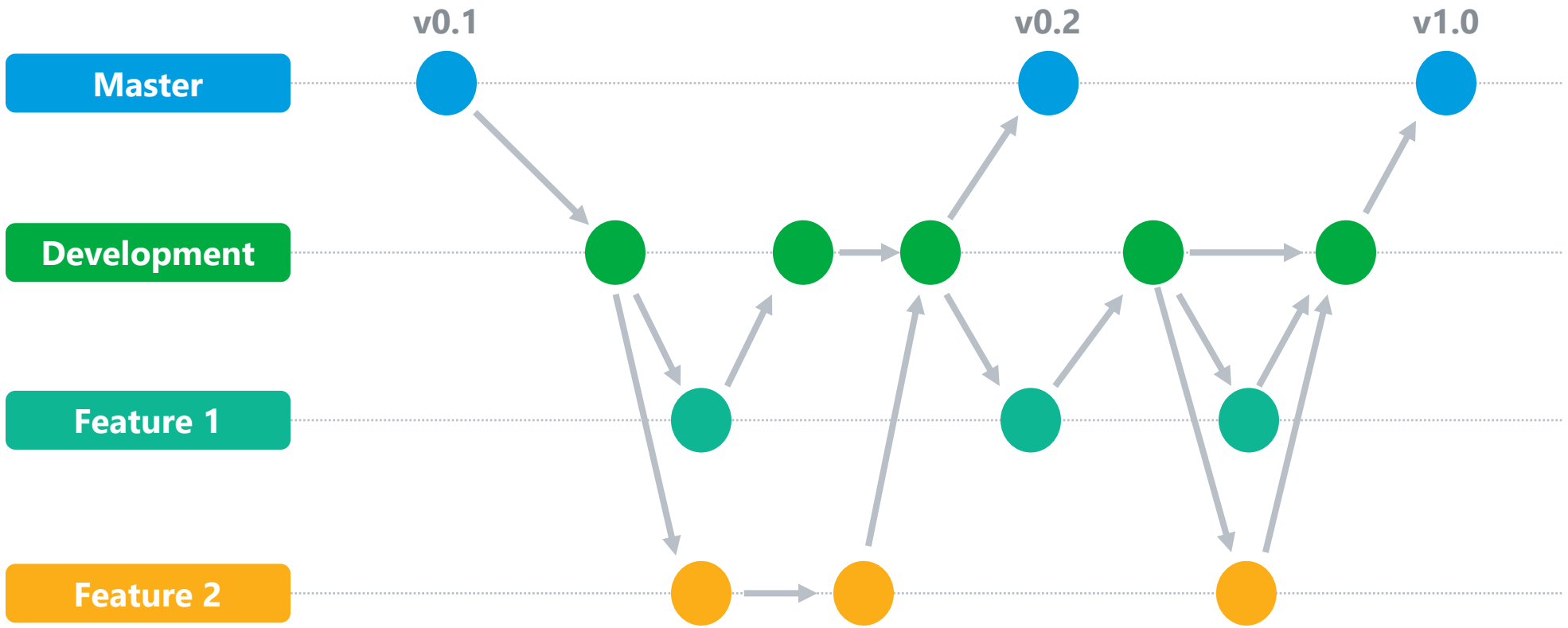
1. **Reproducibility:** Anyone should be able to arrive at your **same results**
2. **Portability:** Anyone should be able to **pick up where you left off** on any machine
3. **Scalability:** Your project should work for **larger data sets** and/or be on the path of **automation**

- These three aims are crucial for **collaborative work**.
- **Version control** software:
  - Directly supports the three goals above, by storing "**snapshots**" of your code over time
  - Makes it easy for you or your teammates to see how the code has evolved over time

What is version control software?

Version control is software to manage development. It allows simultaneous work, gives clear control over integration of changes, and makes reviewing easier, more efficient and more accurate

Sample version control development process



Version control software

GitHub and Bitbucket are the two main options and are used by almost every major company

GitHub

Userbase

37 million users and more than 100 million repositories (as of 2022)

Example Clients

BCG GAMMA, stripe, skyscanner, Spotify, SAP, SOCIETE GENERALE

Bitbucket

Userbase

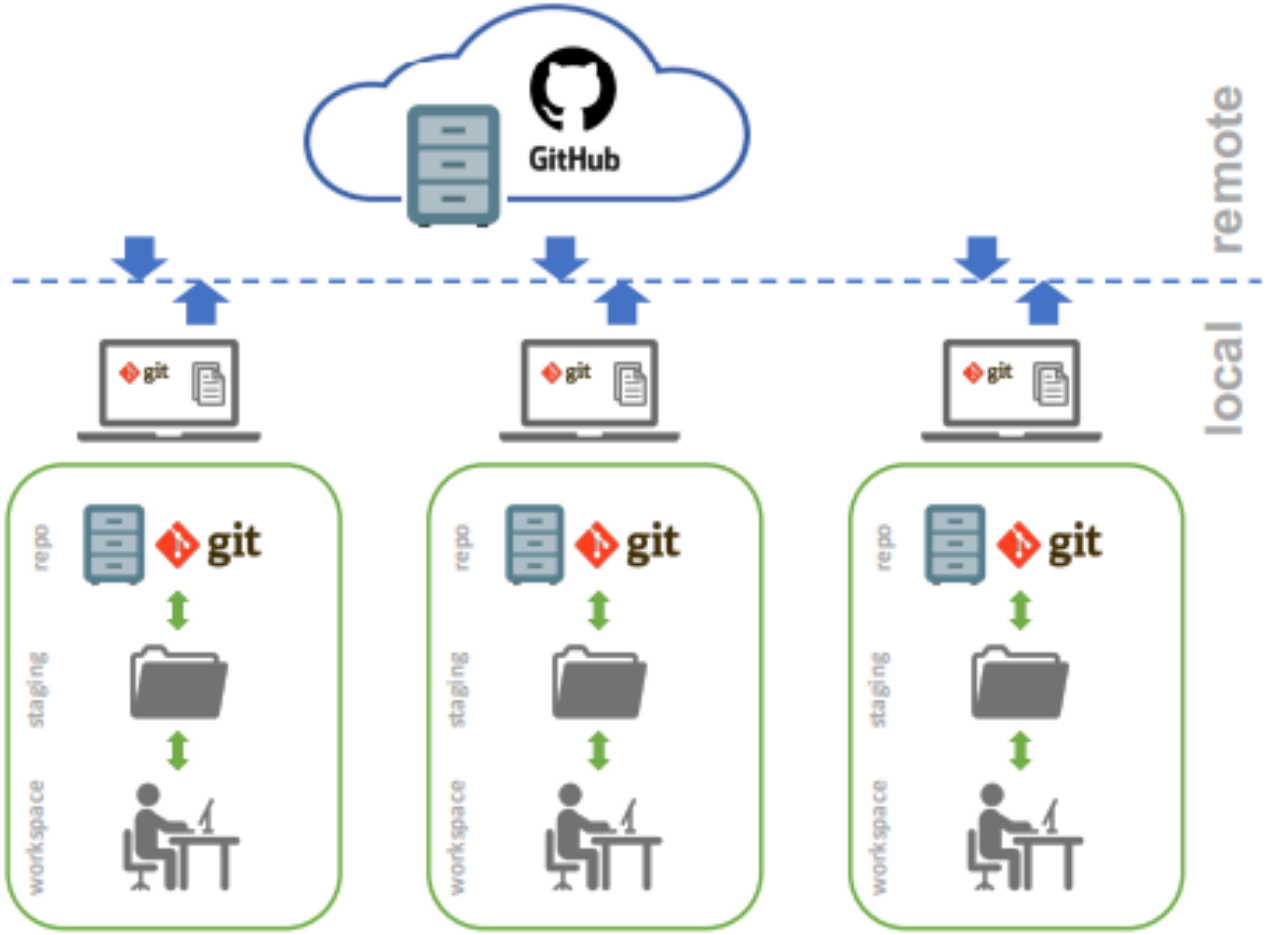
10 million registered users and over 28 million repositories (as of 2022)

Example Clients

PayPal, TESLA, Zillow, salesforce, STARBUCKS COFFEE

GitHub Enterprise is version control software used by corporations. While it is still managed by git / GitHub, data is stored in a different set of servers to keep each company's GitHub data siloed away

An ideal version control setup for collaboration



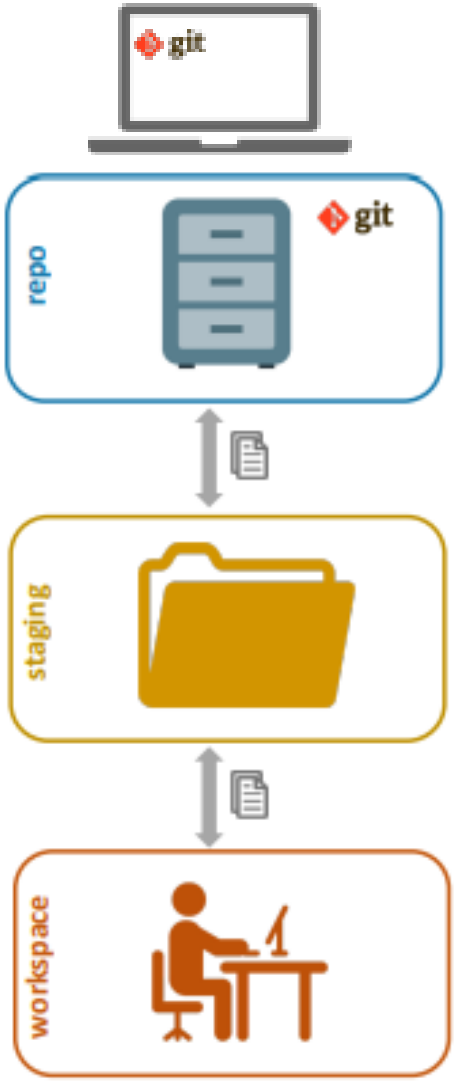
What is git and GitHub?

2

What is this git thing?

And how version control software supports these aims

- **git** is a version control software
  - It is installed "**locally**" on your computer (or virtual machine)
  - Tracks snapshots of your code over time
  - Allows you to "time travel" back to previous states of your code
  - Keep collaboration organized when multiple people are working on the same project

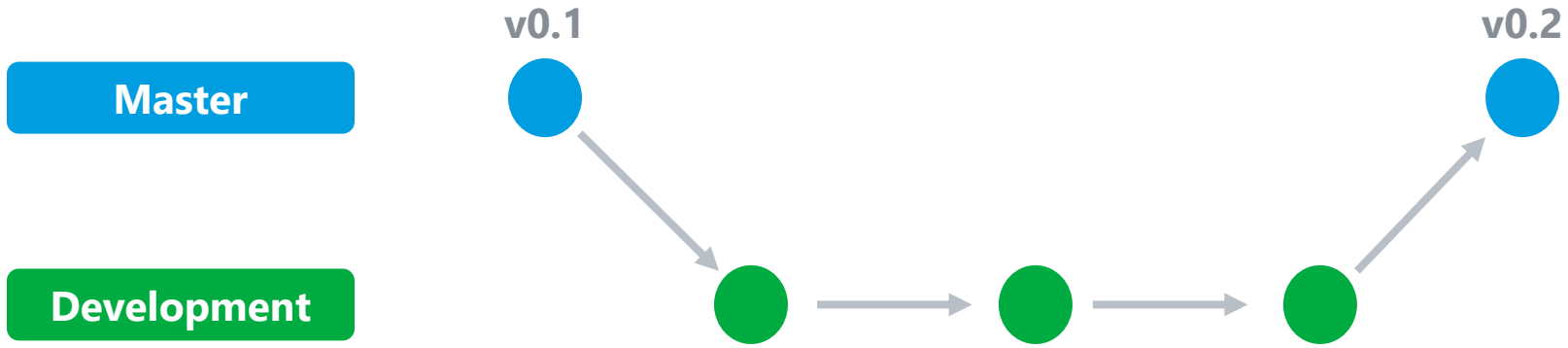


git: A mental model

How git works

The central idea of git is that it tracks changes made to files, allowing you to work in a separate versions (usually called a “Development” branch), while the main or master branch remains in a production-ready state

*Here the master, **production-ready** copy is kept separate from the development version until it is time to review and merge all updates*



How git works

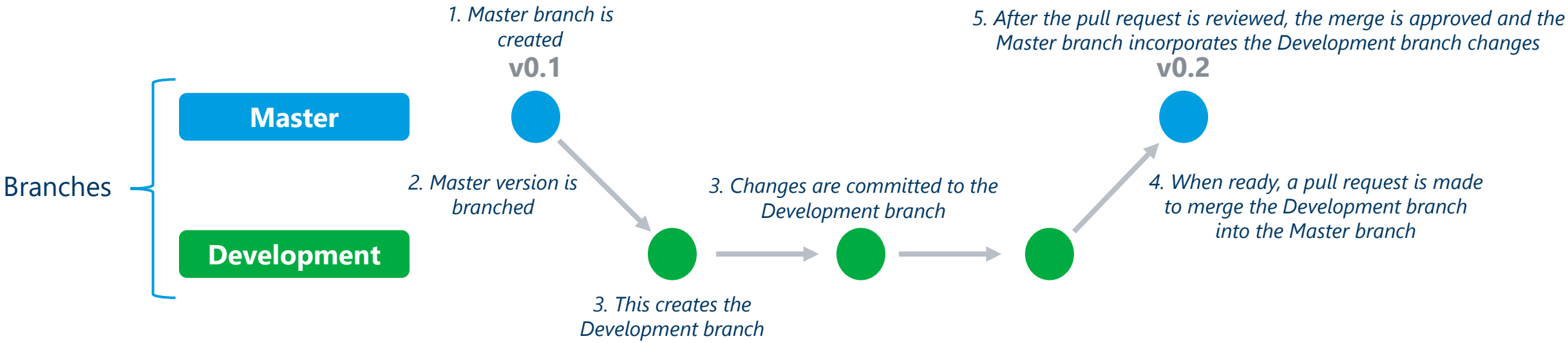
There is a well-defined workflow for making these separate versions and merging them together, which is important to understand

Making a copy of a version is called **branching** the version, this creates a new **branch**

Adding changes to a version is called **committing** changes, or **commits**

Commits are only saved on your local machine, until you **push** the updates to GitHub

Requesting that all these changes are merged back in is a **pull request**



commits: Tracking what's changed

Once you've committed the updates you've made, and pushed them to GitHub, you'll be able to see a side-by-side view of what's changed

Minor updates

Browse files

main

NickAnderson94 committed 3 days ago 1 parent dd3a5ba commit 83770ee0bcbab74961a674c3bd92b1408a83ad98

Showing **2 changed files** with **2 additions** and **16 deletions.**

Split | Unified

Filter changed files

- 2 Code
  - Checking [redacted] thematic co...
- ROPSO Analysis.Rmd

2 2 Code/Checking [redacted] thematic codes.R

```
@@ -222,7 +222,7 @@ rltsp_type2      <- ChiSqrTest(DV_df=DV_df[!is.na(dat1$rltsp_type2), ],    IV =
222 222  first_love      <- ChiSqrTest(DV_df=DV_df[!is.na(dat1$first_love), ],    IV =
         dat1$first_love[!is.na(dat1$first_love)])
223 223
224 224  # Combine results into a single table
225     - factor1 <- c( rep (rep( c(rownames(tests)[1:length(DV_df)]), each = 2), 3), rep( c(rownames(tests)
         [1:length(DV_df)]), times = 5), rep( c(rownames(tests)[1:length(DV_df)]), times = 3))
    225 + factor1 <- c( rep (rep( c(rownames(tests)[1:length(DV_df)]), each = 2), 3), rep( c(rownames(tests)
         [1:length(DV_df)]), each = 5), rep( c(rownames(tests)[1:length(DV_df)]), each = 3))
226 226  factor2 <- c( rep( c("gender", "sexuality", "rltshp_status"), each = c(16)), rep("rltshp_type", times = 40),
         rep("first_love", times = 24))
227 227  categorical_tab_results <- data.frame(cbind( factor1, factor2  , rbind(gender[[1]], sexuality2[[1]],
         curr.rltsp.status.single[[1]], rltsp_type2[[1]], first_love[[1]] )))
228 228  factor1 <- rep( rownames(tests)[1:length(DV_df)] , times = 5)
```

What is GitHub

GitHub is a cloud service that hosts git repositories, in other words, all you code is stored on GitHub servers

Github Servers

Code

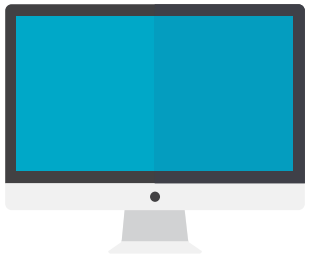
- Master branch
- Development branch
- All other branches that have been committed

Data

- Anything not included in the .gitignore file will be uploaded

---

Your Computer



Code

- Whichever branch you decide to load locally – loading a branch will replace all files in your directory with the files from that branch.

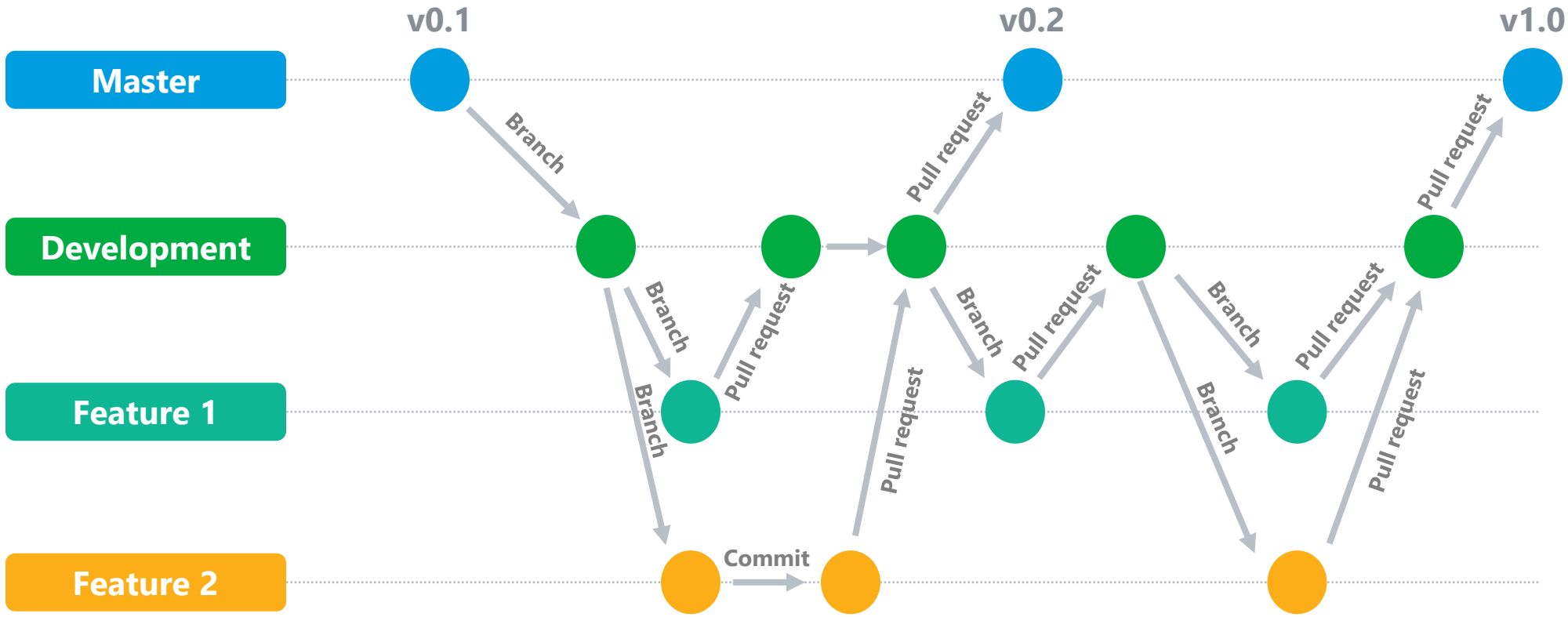
Data

- Anything included in the .gitignore file will exist only on your computer

How Github supports multiple branches

Here you can see a sample workflow, with multiple features worked on simultaneously

Changes are being made independently in branches "Feature 1" and "Feature 2", then pull-requested into Development. Occasionally the latest version of Development becomes the latest full release when merged into Master



Branching: An example

This shows four different branches, which would be typical for any production-ready application your team is deploying

Main

- The *Main* branch is the latest full release.
- Should be completely customer ready and thoroughly reviewed.

Development

- The *Development* branch is the latest in-progress version.
- No changes should be made in it directly – you should make a new branch based off it (e.g. ‘Feature 1’), make your changes in that, then make a pull request to merge that new branch into development.
- This way changes can be reviewed, and Development stays as a fully reviewed version.

Feature 1

Feature 2

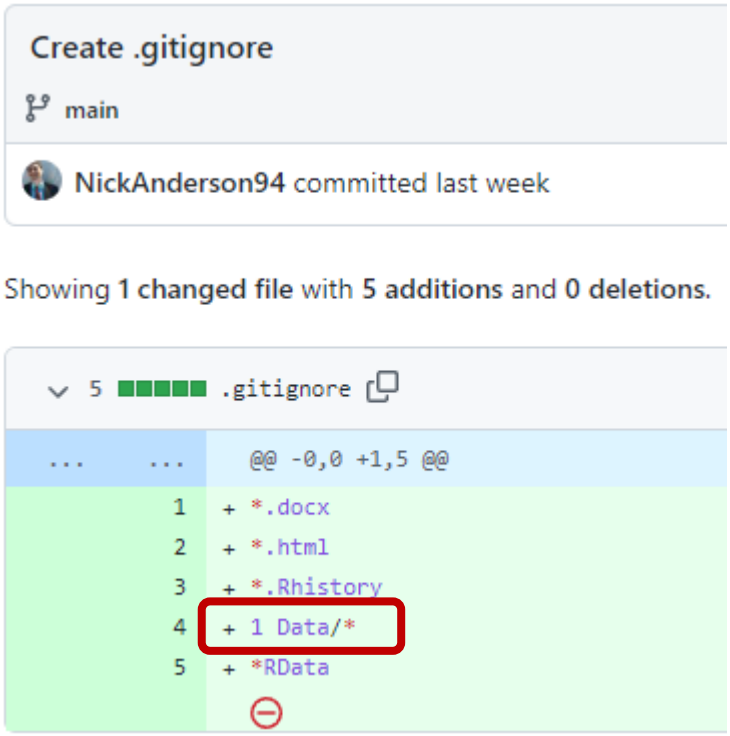
- Other branches are for new features, bug fixes, modifications etc.
- Create these by branching off development when you want to make any changes
- Name them something descriptive to what you’re working on in them
- Its best to do little and often – make your new branch, change what you need to change, then make the pull request. This way reviews are shorter and changes can be merged more easily

GitHub benefits

- Persisting repository storage
  - No need to create folders on your computer for the same project files (e.g., “app final”, “app adding feature 1”, “app debugging issue 1”)
- Seamless transition between environments
- Essential for collaboration, and backing up code (you should use this on a daily basis)
- Synchronizes your work and minimizes the risk of people stepping on each other’s toes
- As we will see when we explore the GitHub website, it allows you to centrally track issues like bugs or product enhancements
  - You can categorize these issues based on the skills needed to address them
  - You can assign these issues to teammates

.gitignore

Use .gitignore files to designate which files / folders should not be uploaded to GitHub
Typically, this includes confidential data files and other miscellaneous project files created as you work



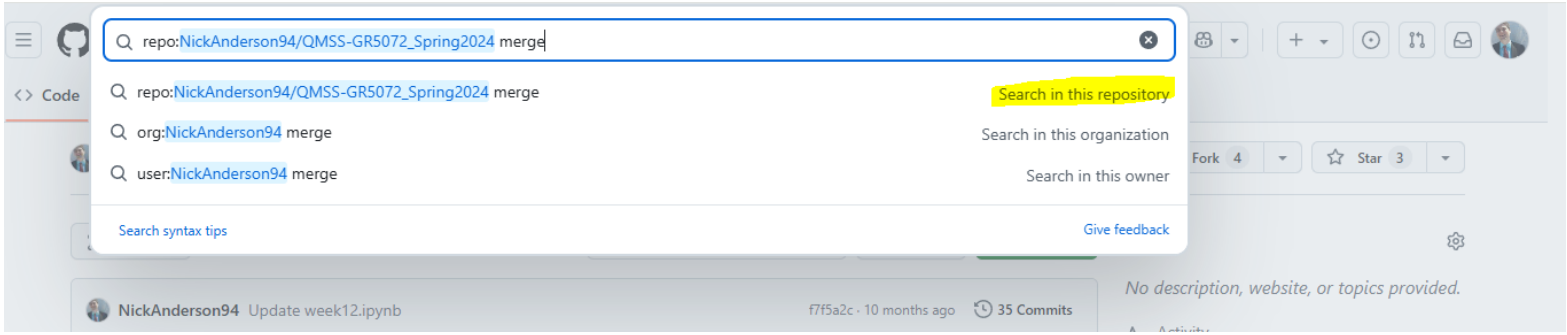
- In Python for example, we will be using Jupyter Notebook a lot this semester. So our **.gitignore** file will ignore Jupyter Checkpoint files:
  - We do this by adding ***.ipynb_checkpoints** to the .gitignore file
- See here for a guide to .gitignore syntax

- Note: You can use "#" for commenting out code

- To create a **.gitignore** file, use any text editor and follow these steps:
  1. Add the files/directories that you don't want git to track
  2. Click 'save as' name the file **.gitignore**
     - Note: This file **cannot** end in any other file extensions, like .txt
- A second option is copy and pasting other **.gitignore** files into a new directory
- A third option is to create them on GitHub when you create a new repo

Pro Tips to Using git

- Repositories will often have many code files. If you can't remember where something is, you can search for a word/phrase to find everywhere it occurs in the repo. For example, below I search for the word "merge" in the Spring 2024 repo.



- Never rename files locally, always use GitHub to do this. When renaming files locally, git sees new file names as you creating a new file (with no version history) while deleting the old file (thus losing version history for that file)
- When exploring commit history, you can see all previous versions of your code on GitHub. Reverting is not as easy on GitHub Desktop as it is for command line git. I've only ever had to do this once though, and I just manually downloaded the file with the correct version and overwrote it locally
- The first step to using git successfully is just understanding which files are on your local machine vs the GitHub cloud
  - git **push** and **'fetch' origin** are what you use to exchange information between the two
- **commit** small chunks of logically grouped changes
  - If you have modified separate files at once, it may make sense to **commit** these in separate batches
- **commit** with informative messages
- Currently, most people are using **main** as the default branch, used to be **master**

Review the key terms
How do we define the following terms?

- git
- GitHub
- Repository (aka, “repo”)
- commit
- branch
- pull request
- push request
- Challenge definitions for class:
  - “fetch” origin
  - fork

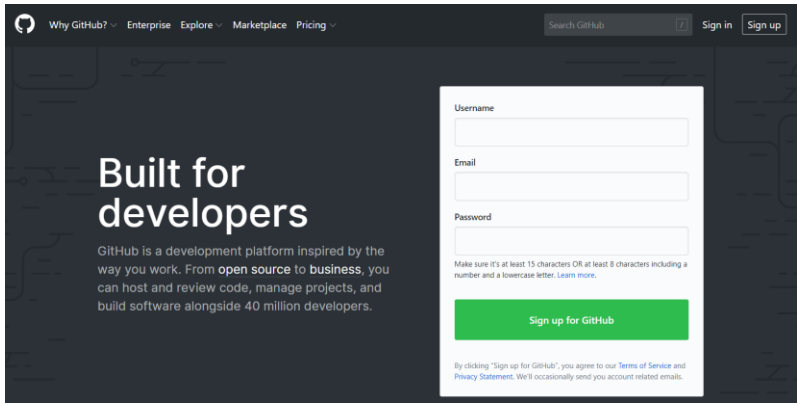
Using git and GitHub Desktop

Tools you will need

Github.com

https://github.com/

- The central source for all code and project management



Github Desktop

https://desktop.github.com/

- A desktop tool to easily handle creating branches, loading branches, pull requests etc

Optional: Use git through the command line

- For advanced users

Using GitHub Desktop and GitHub

Please follow along on your local PCs as we explore the following on GitHub

On GitHub web, we'll explore:

- Accessing your repositories
- Creating a new repo
- Within a repo:
  - Code
  - Issues
  - Pull Requests
  - Settings
    - How to add **collaborators**
    - How to **protect** a branch (so only an **owner** can modify it)

On GitHub Desktop:

- Making a **commit**
- Creating a new **branch**
- Creating a **pull request**
- **Pushing** to origin
- How to **"fetch" origin**
- How to create and use **.gitignore** files

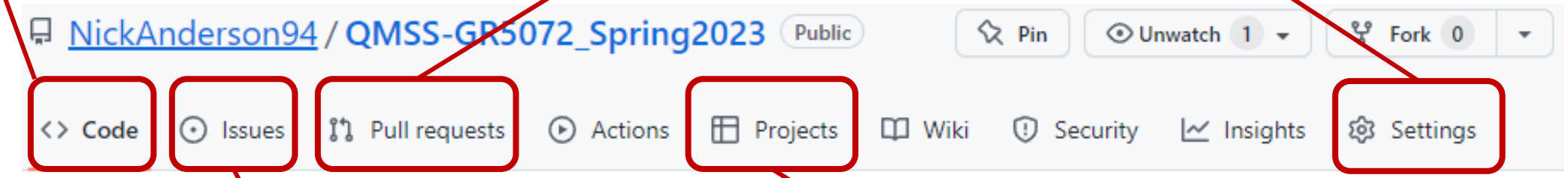
Using Github Web: Overall

When on a particular repository's page, you can navigate with the bar at the top

Code: The code itself

'Pull requests': a list of outstanding requests to merge code

Manage repo settings, like whether it is private or public. Invite collaborators if it is private

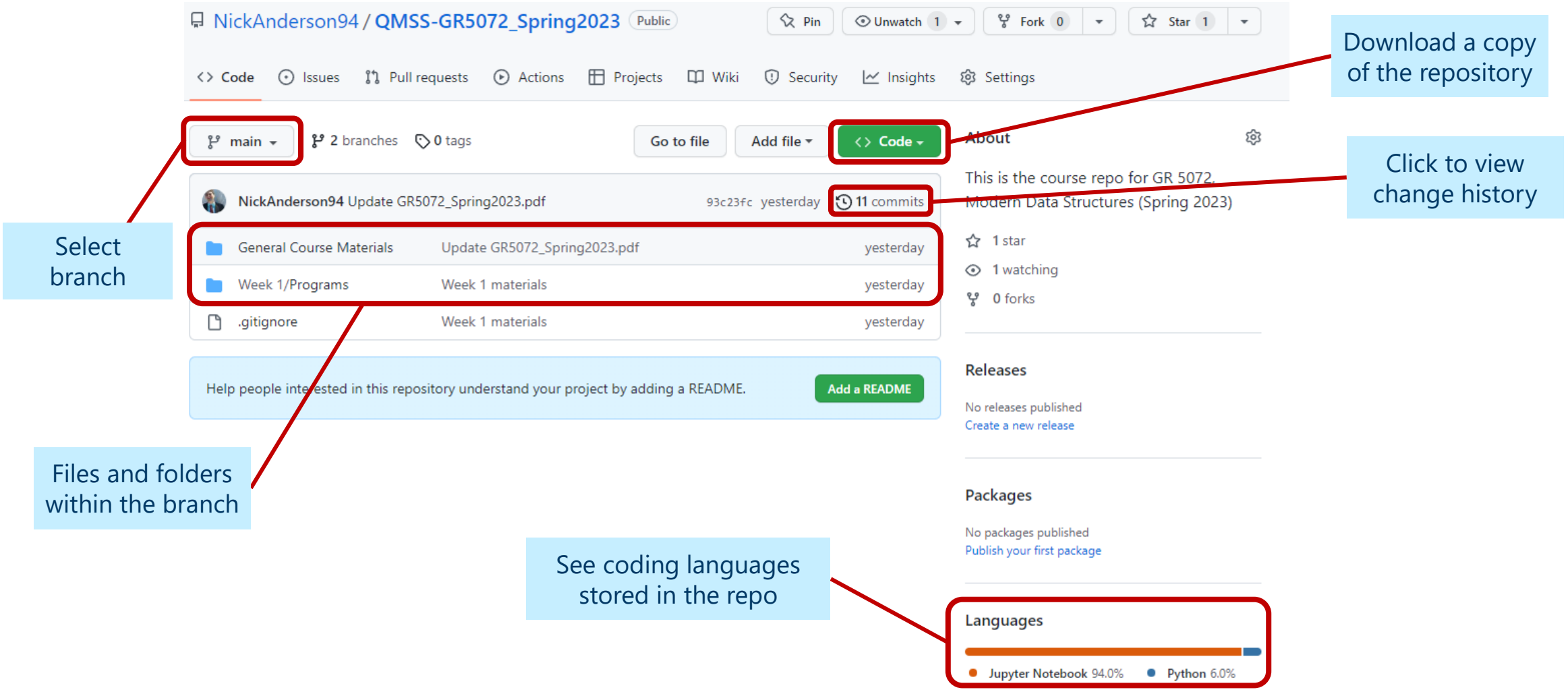


'Issues': a to-do list of outstanding issues

'Projects' – a tracker of issues and their progress for each area of the repository

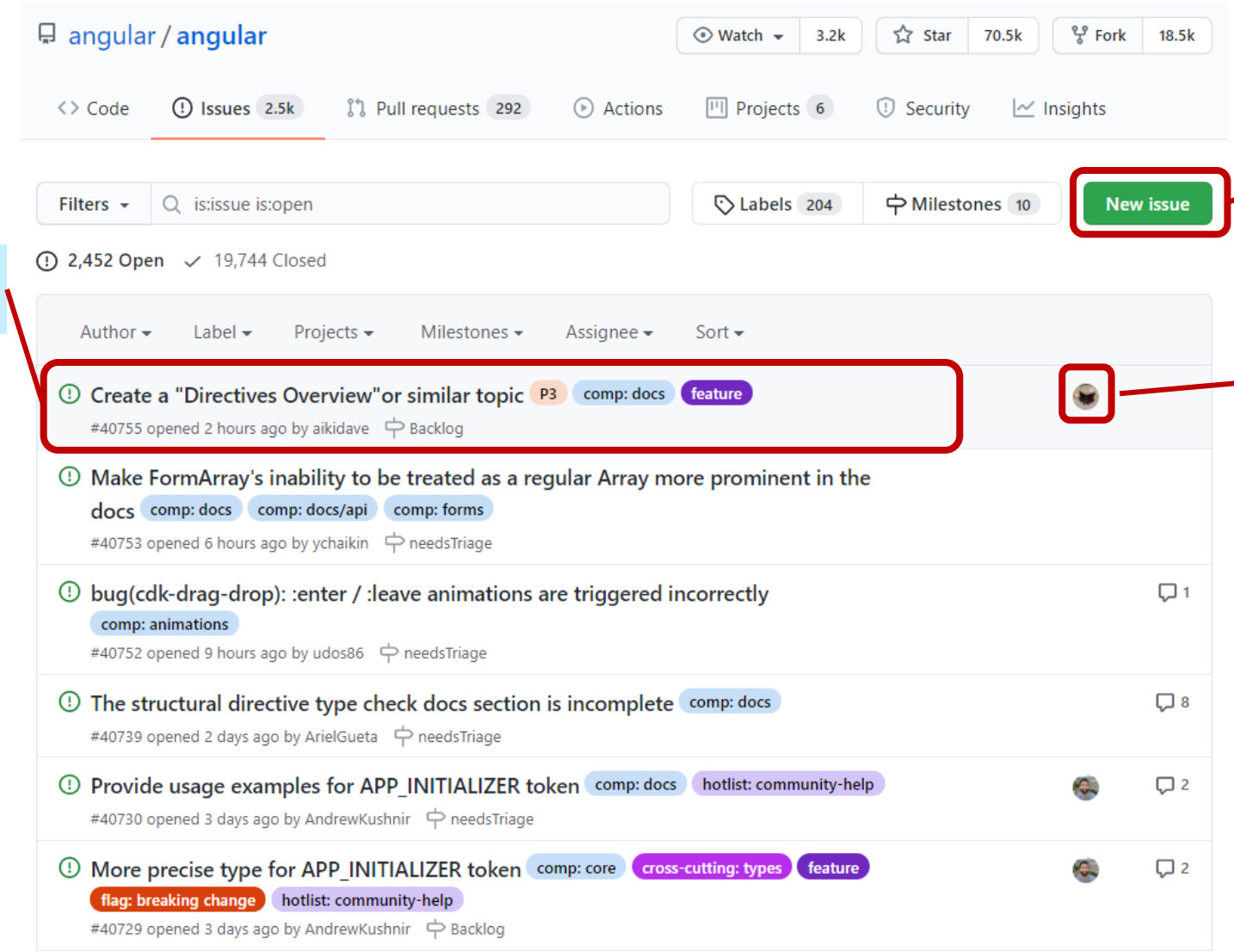
Using Github Web: Code

The Code tab is where all the code resides, and can be viewed by branch



Using Github Web: Issues

The issues tab is where issues are created and tracked – whether questions, feature requests, bugs etc. they all go here like a to-do list



Click on an issue to view it in more detail

Click to create a new issue

See who has been assigned to resolve an issue

Using Github Web: Issues

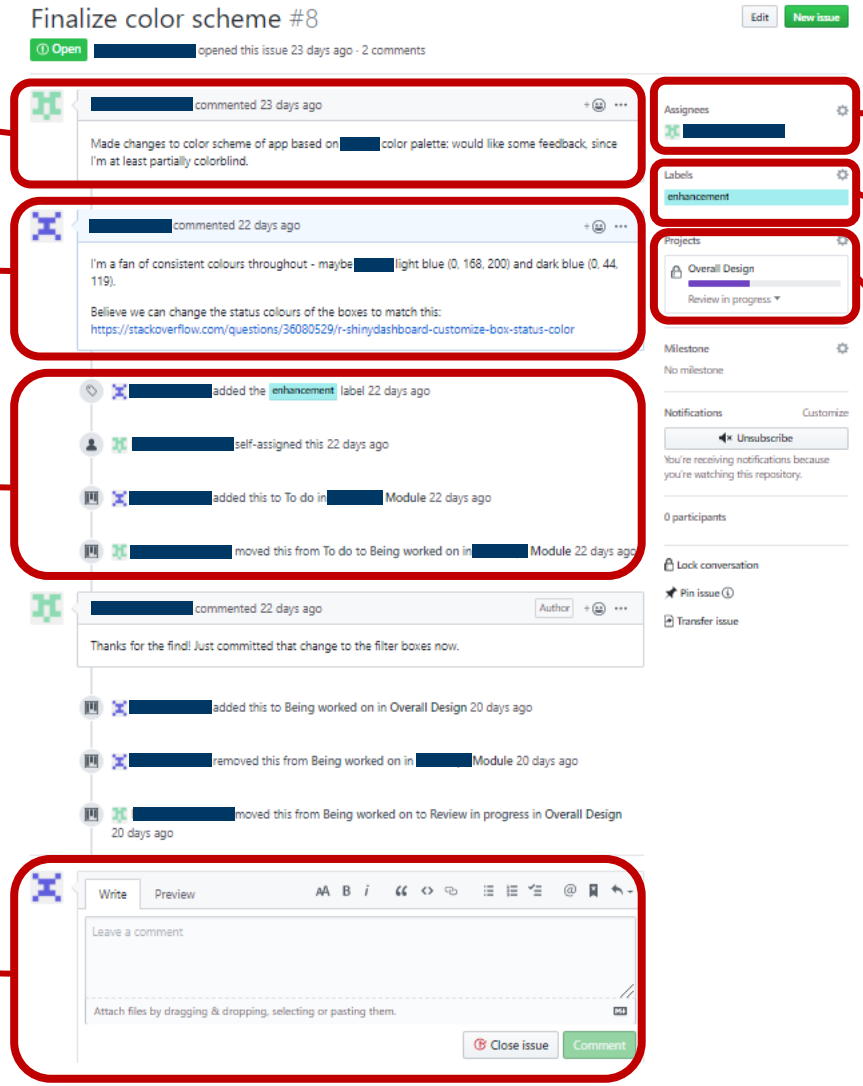
Click on an issue to view it in more detail

The initial detail of the issue

A comment on the issue

Details of changes to the issue's categorization

Place to enter a new comment on the issue

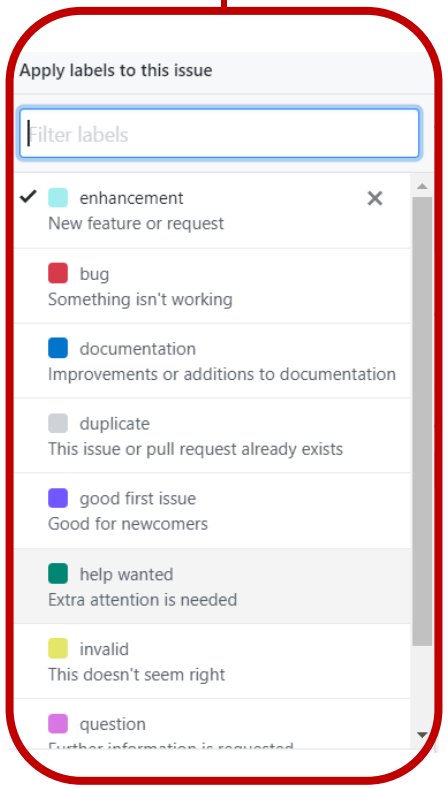


List of those assigned to the issue

Labels for the issue

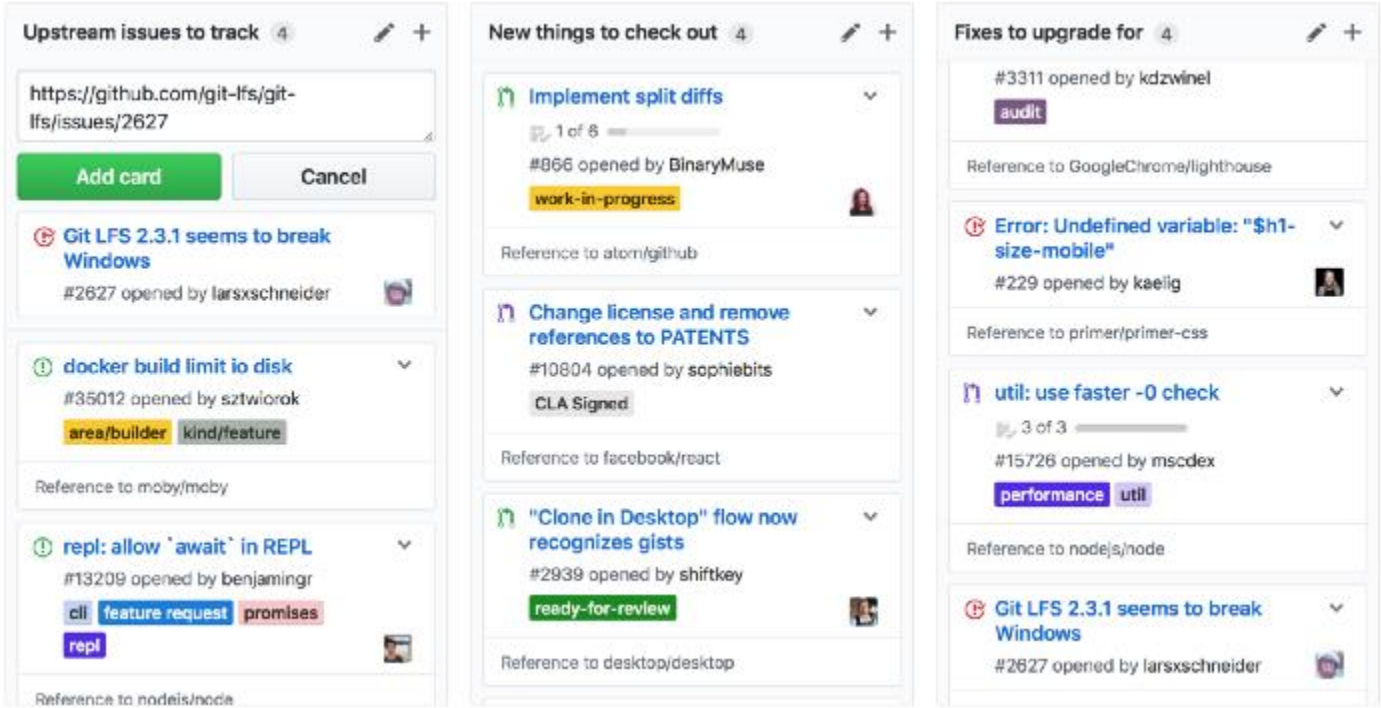
Projects the issue is tagged to, and its status within them

Example labels



Using Github Web: Projects

Issues are collected together in Projects – like categorizations of issue (by module, type etc.)



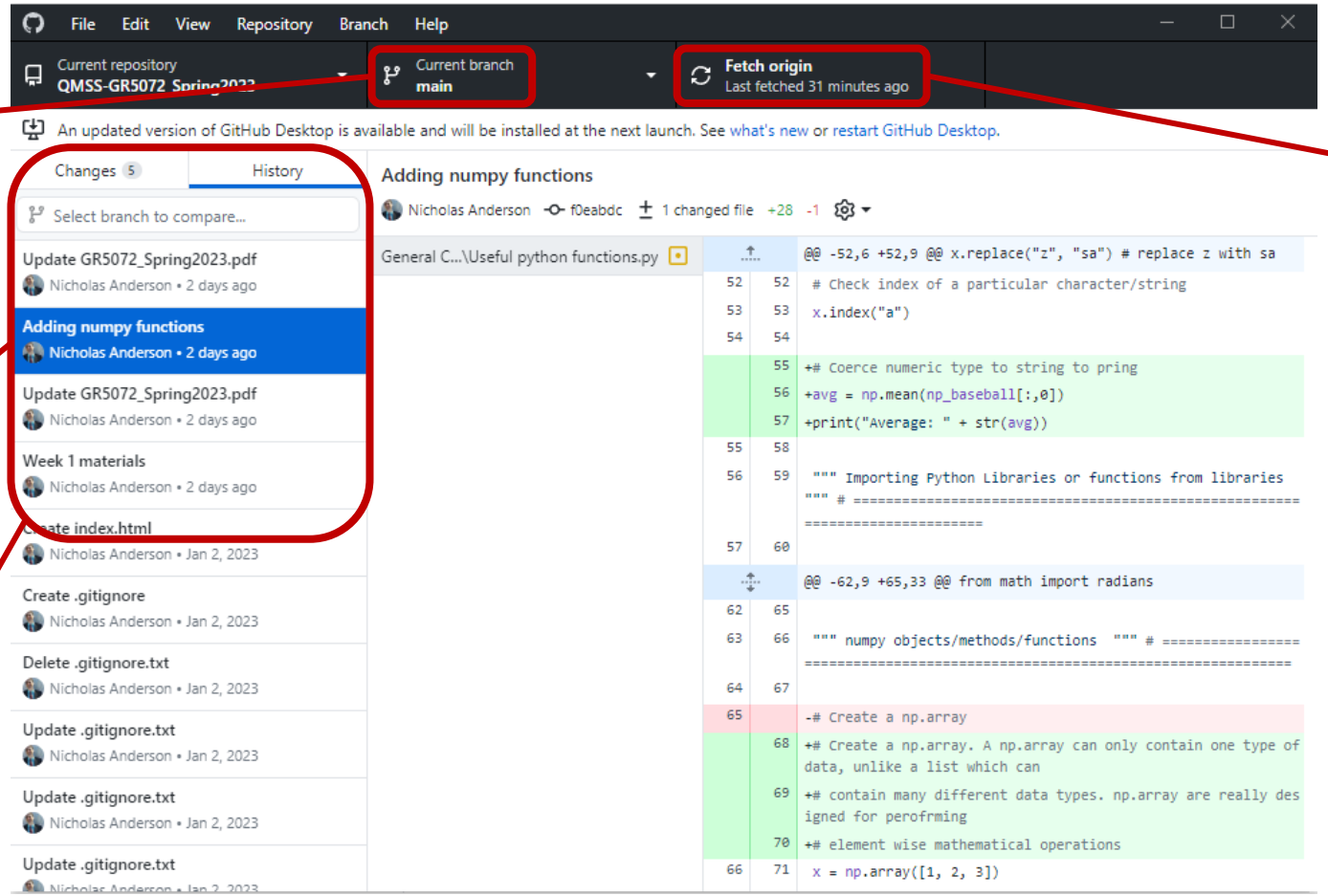
Using GitHub Desktop

You should use Github desktop to manage making adjustments to the code (except for renaming files, which should be done in github.com)

Click to change branch – this will change all the files in your Github directory to be those from the selected branch

If you select 'history' you can see all of the commits to that branch, and see exactly what changed in each one

If you select 'changes' you can see all of the changes you have made to that branch, and commit them with a message



Click to pull the latest version of that branch from Github – you should always do this so you have the latest version

End to end process for making a change

Making a change

Let's look at an example of the process for making a change – the following slides show screenshots of each step

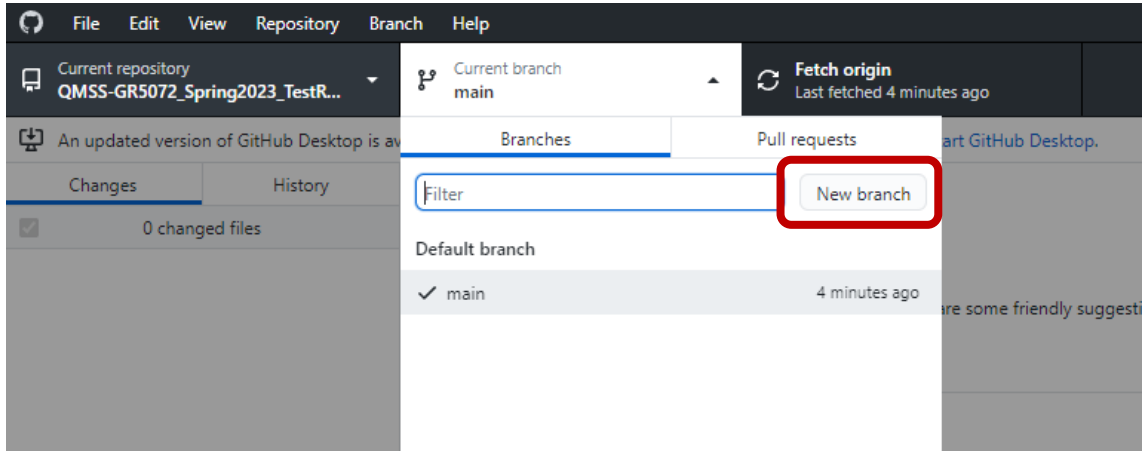
1. Raise an issue if one isn't already created, and assign it to yourself
2. Move it from "To do" to "Being worked on" on the project board
3. Go to Github desktop, load the development branch, and click "Fetch Origin"

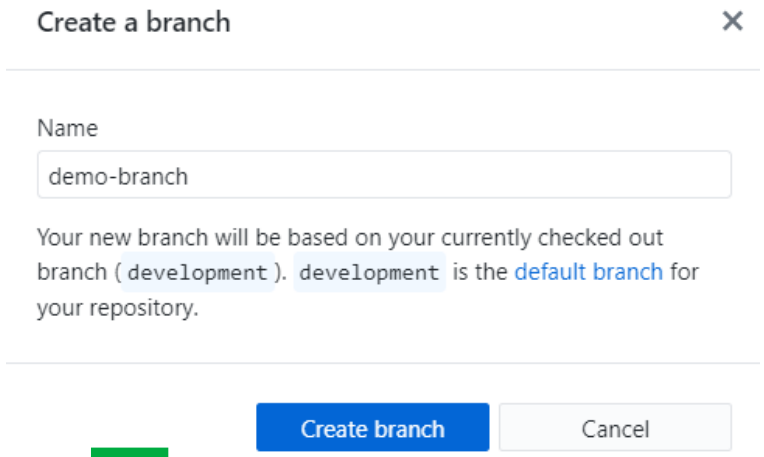
Note: These first three steps are beyond the scope of this class, but are typical steps if you are working on a large team. I therefore don't include walkthroughs on these in the subsequent slides

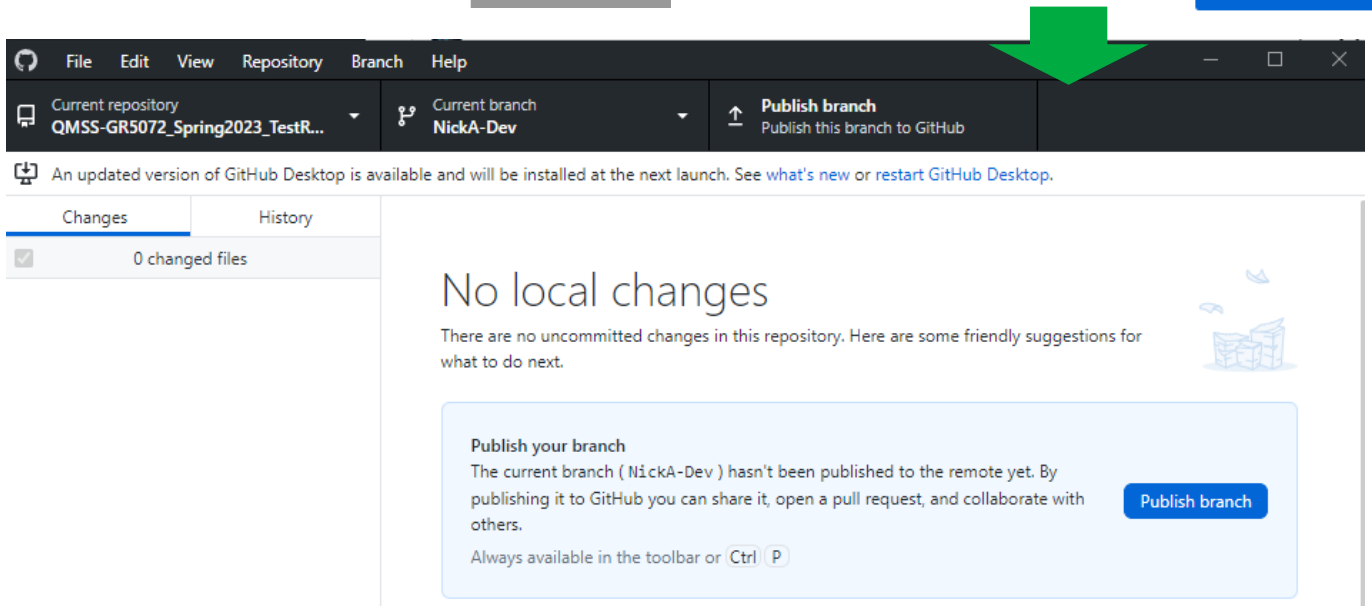
4. Create a new branch for your changes and load it
5. Make any changes you need to the code
6. Commit the changes locally in Github desktop
7. Publish your local branch to Github
8. Create a pull request
9. Reviewer reviews the changes, and approves (or makes comments to be addressed)
10. Once the merge is approved, the code is updated and the branch can be deleted

Making a change (4)

Then create a new branch with an appropriate title, and load it up



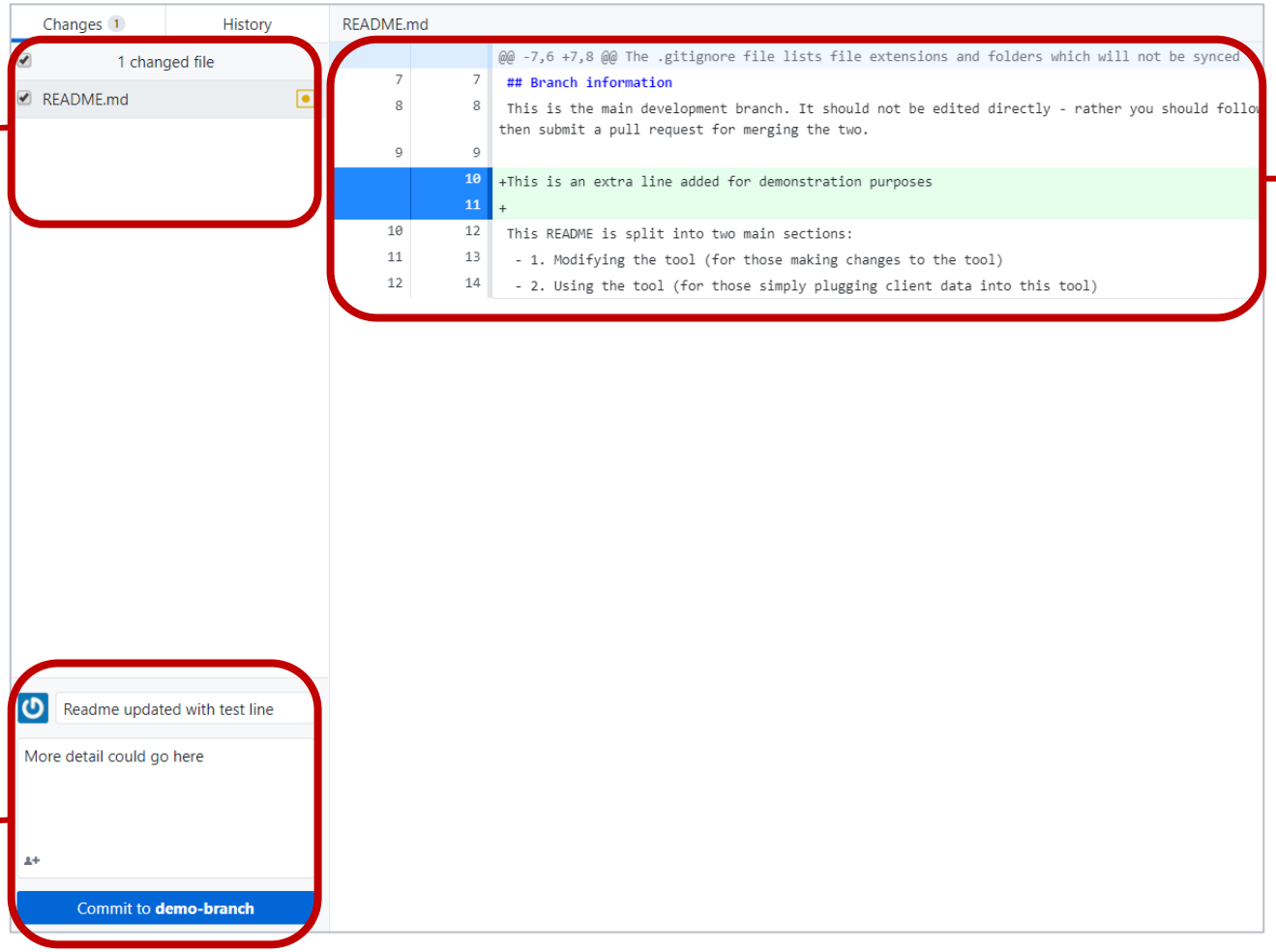




Making a change (6)

Once you've made and saved changes to a file tracked by git, go to Github Desktop and you will see the changes you've made. Click commit to add them to the branch locally, with a description

List of changes that have been made

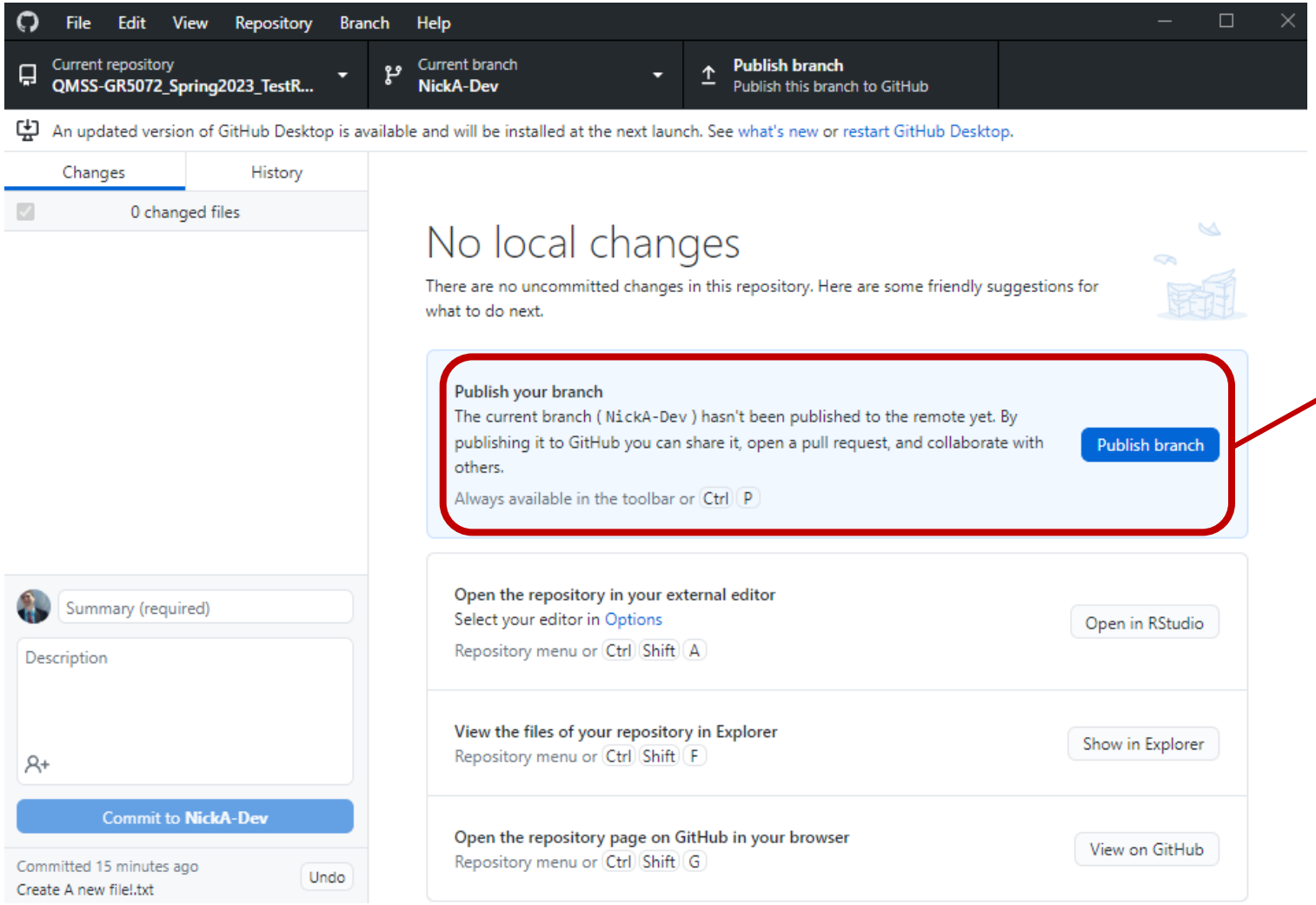


Detail of changes

Add each commit with details of what has changed. This will commit it to this new branch

Making a change (7)

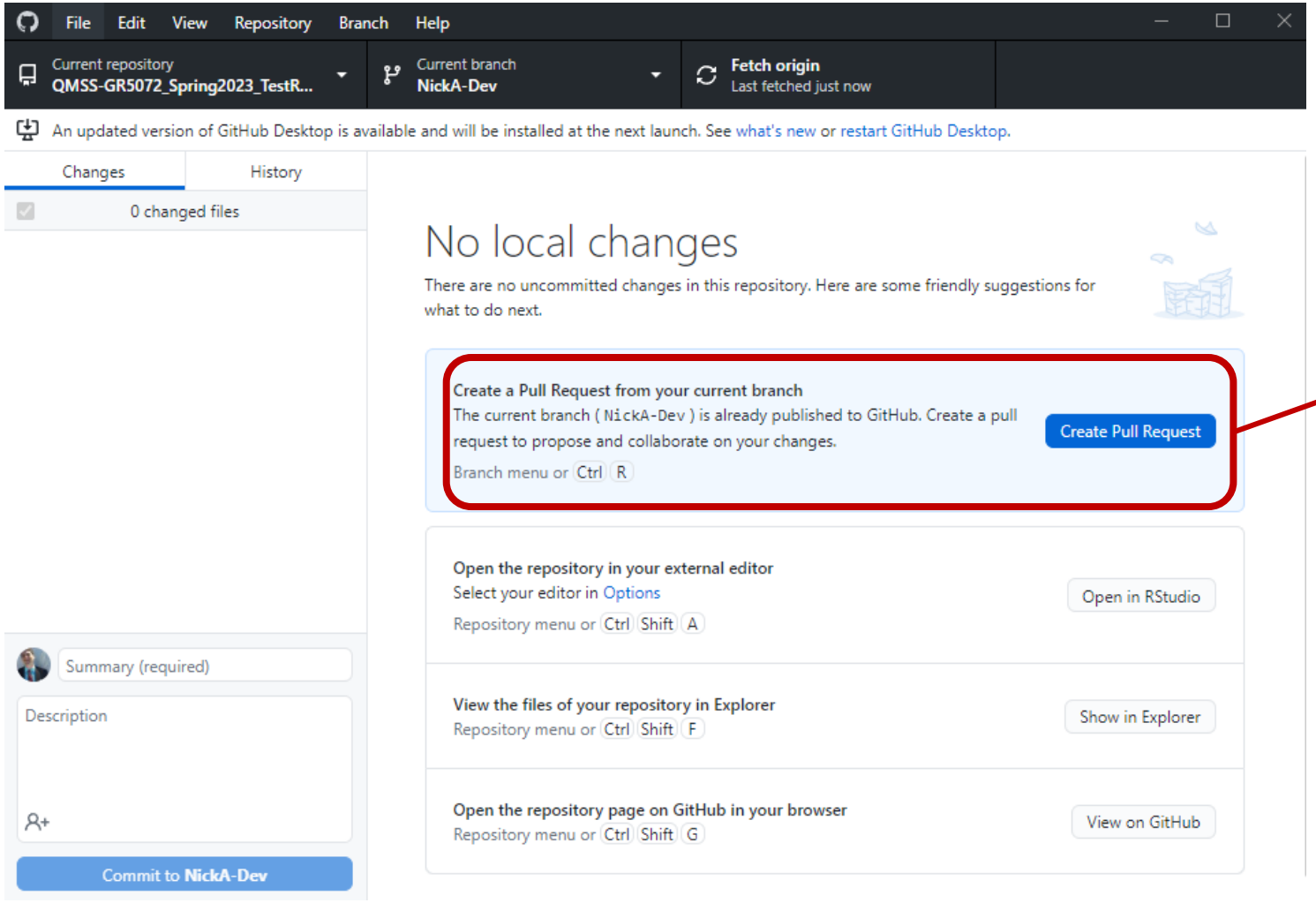
When your changes have been committed to your local branch, you may need to publish that branch to GitHub, effectively uploading it to the central repository, if it doesn't already exist on GitHub



Publish branch to Github

Making a change (8)

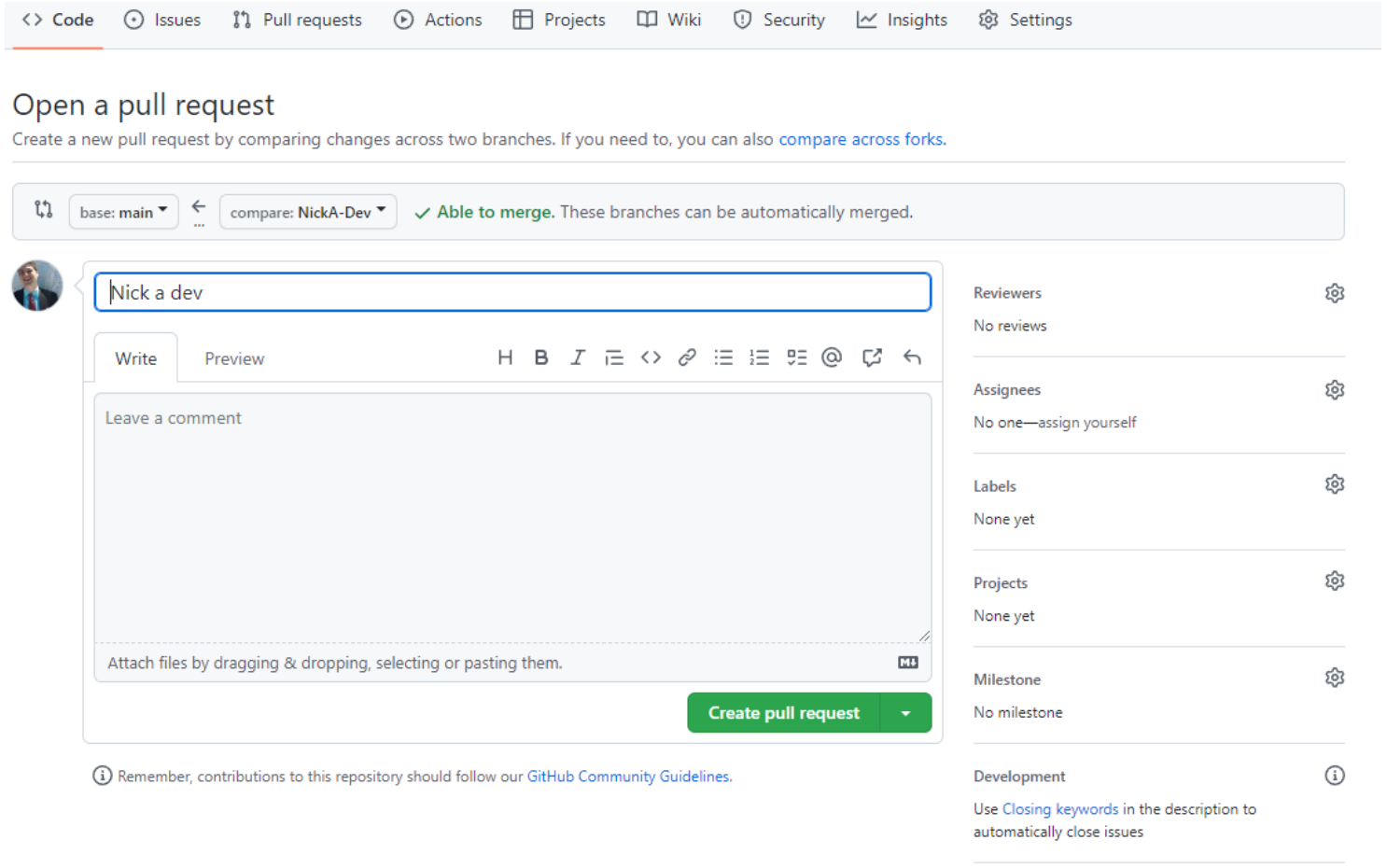
Now the branch has been published to GitHub, you can make a pull request, i.e., make a request for the branch to be merged into the main Development branch



Create pull request

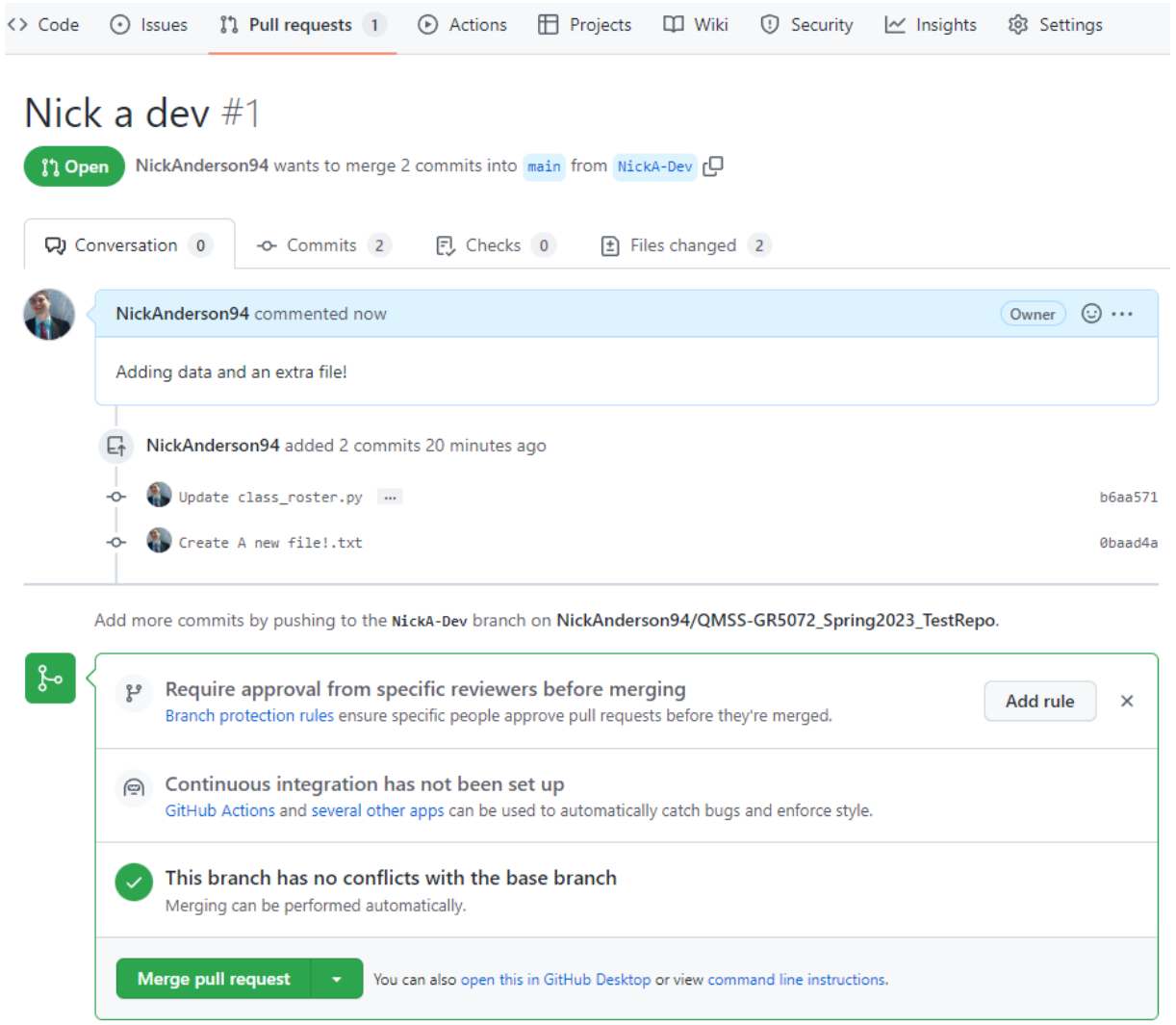
Making a change (8)

This will open up GitHub web, where you can add a title and detail, as well as a reviewer and the relevant project it is for



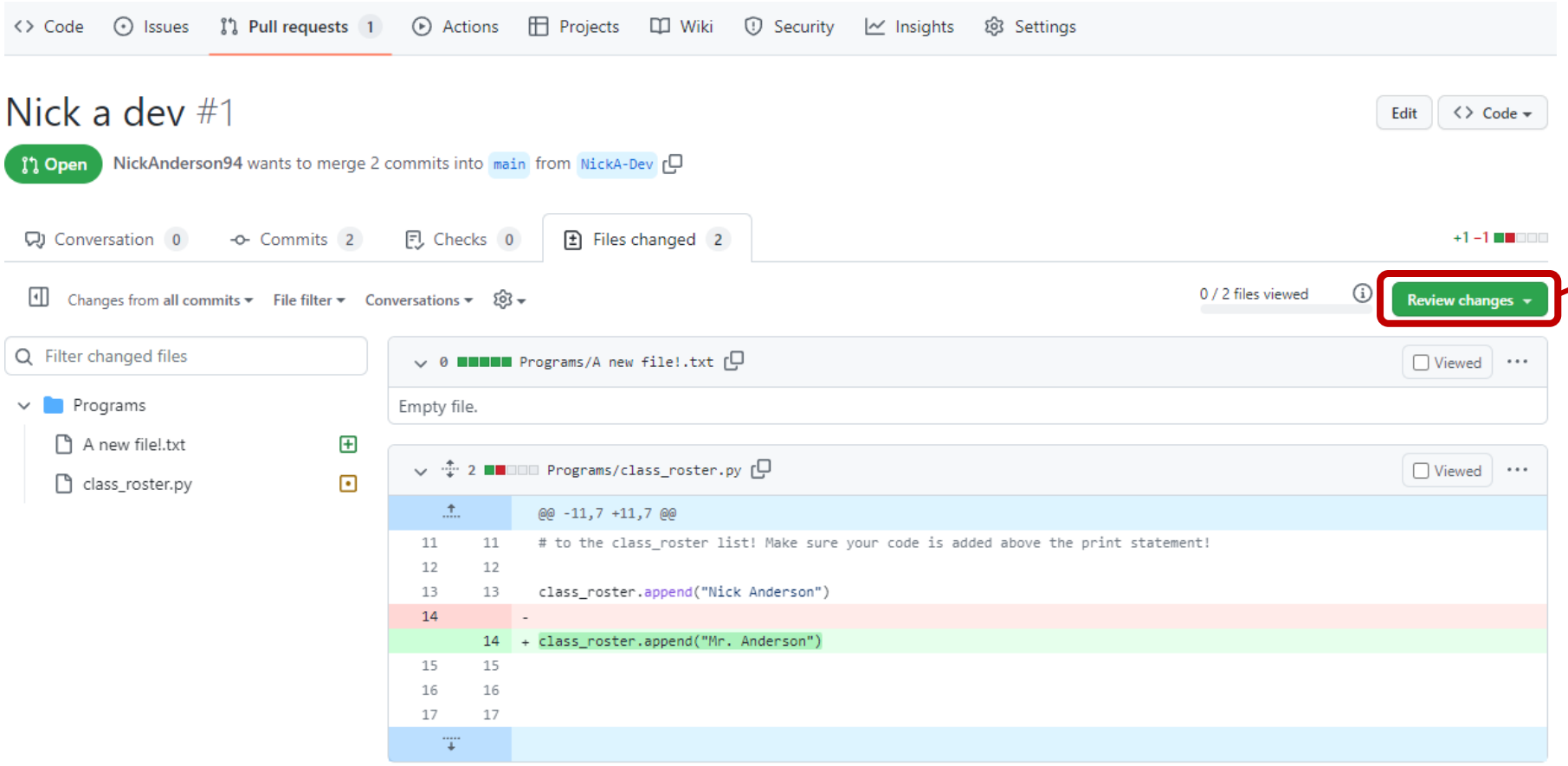
Making a change (9)

The reviewer can then go into the pull requests and select yours



Making a change (9)

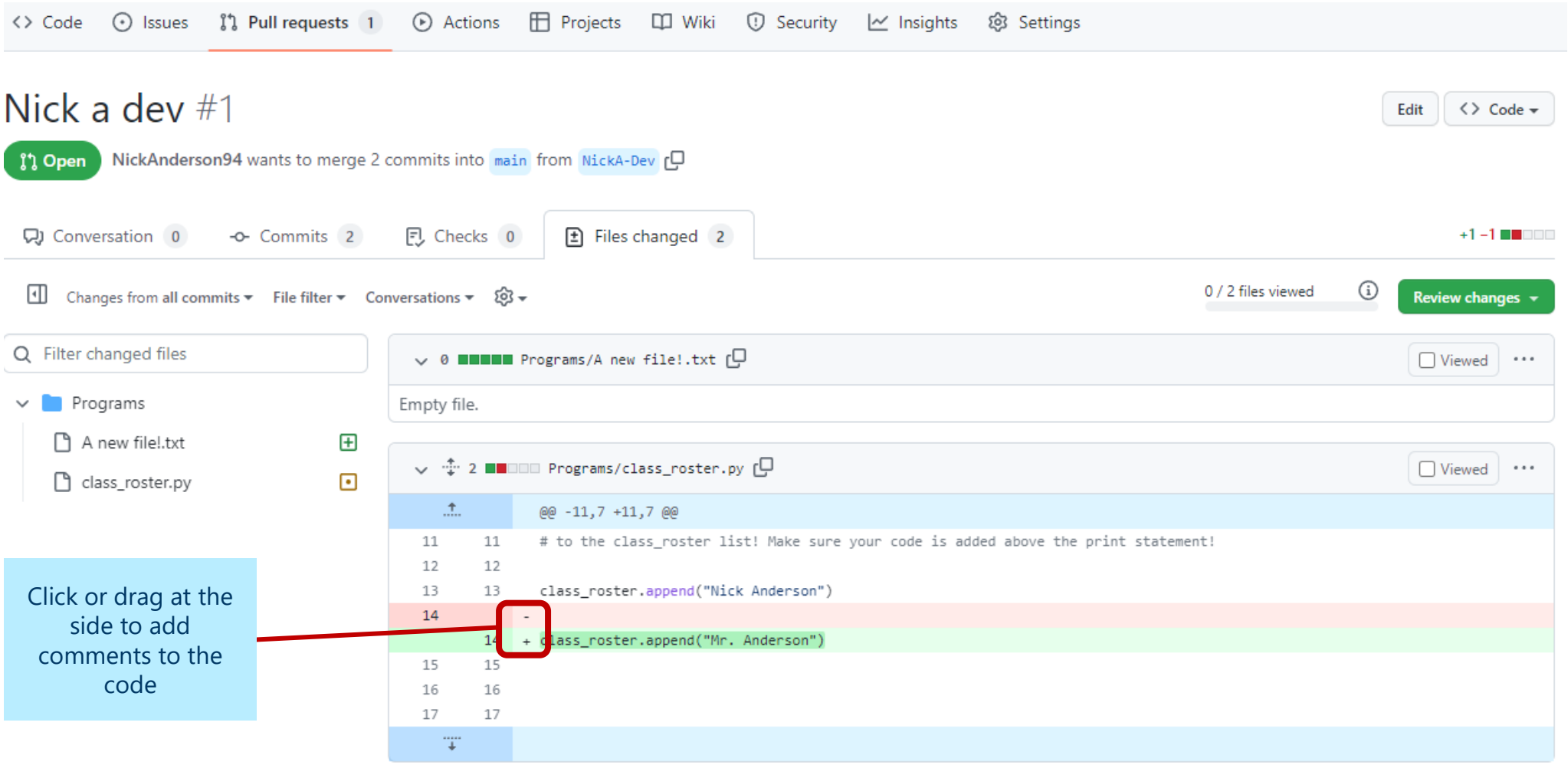
Who ever is managing the higher-level branch, can review the code and send feedback to who ever created the pull request



Review changes here

Making a change (9)

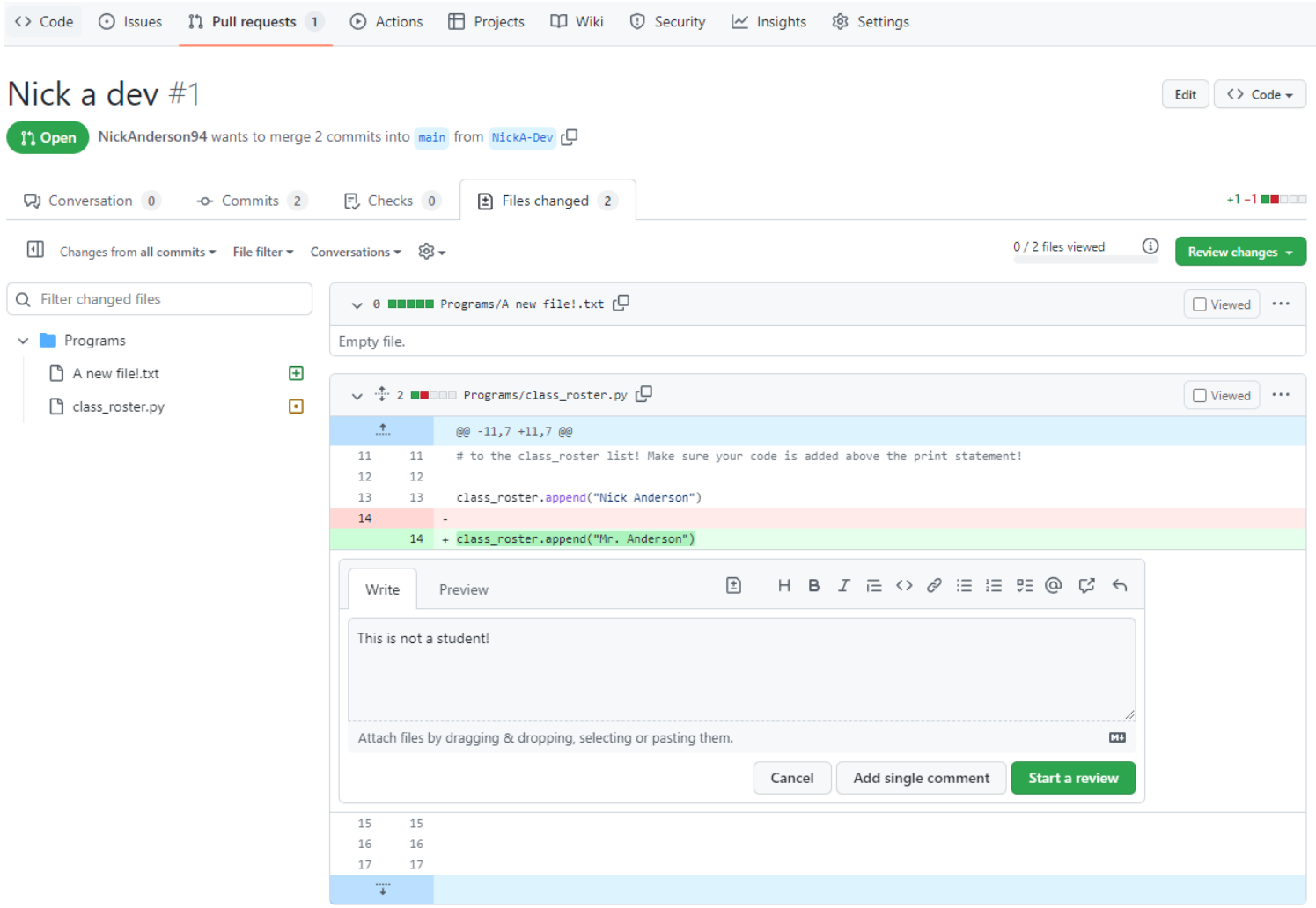
They can then see all the changes, and add any comments they want

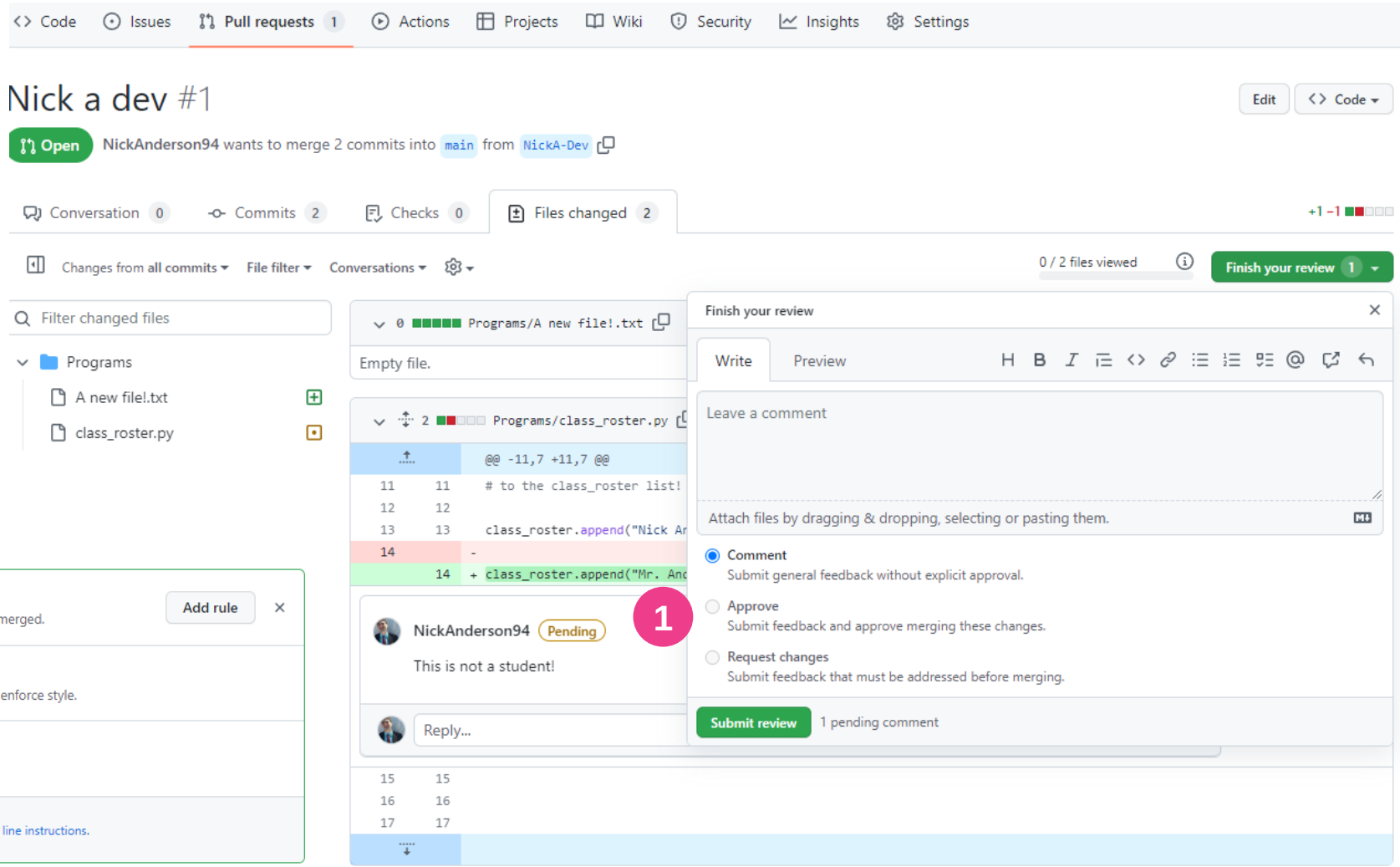


Click or drag at the side to add comments to the code

Making a change (9)

They can then submit their review with any comments – either submitting it with feedback to be addressed, or approving it all and merging it



Making a change (9)

This will update the pull request with the review comments, and the pull request can be merged or sent back for revision