

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?



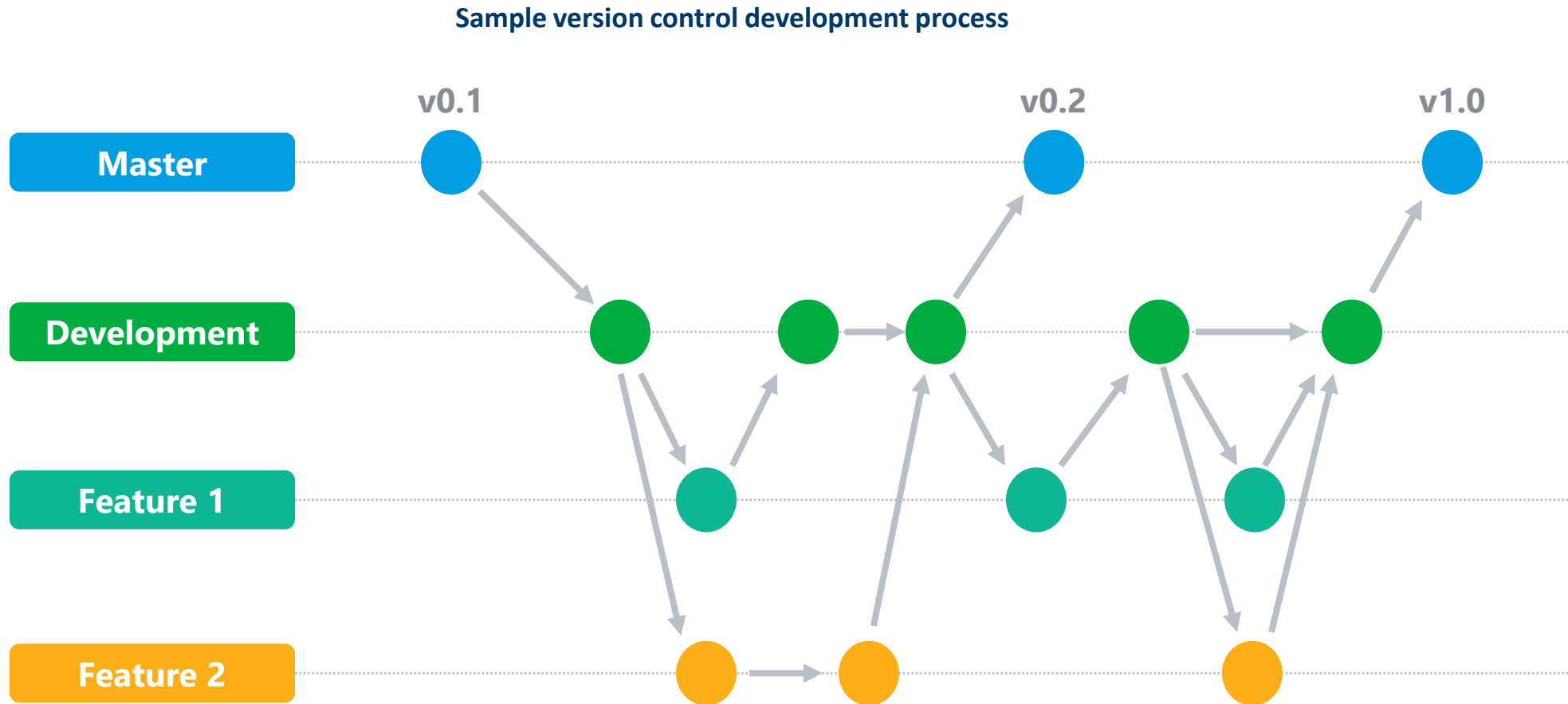
The three aims of a Data Science project...

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



Version control software

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



Userbase

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

Example Clients



Userbase

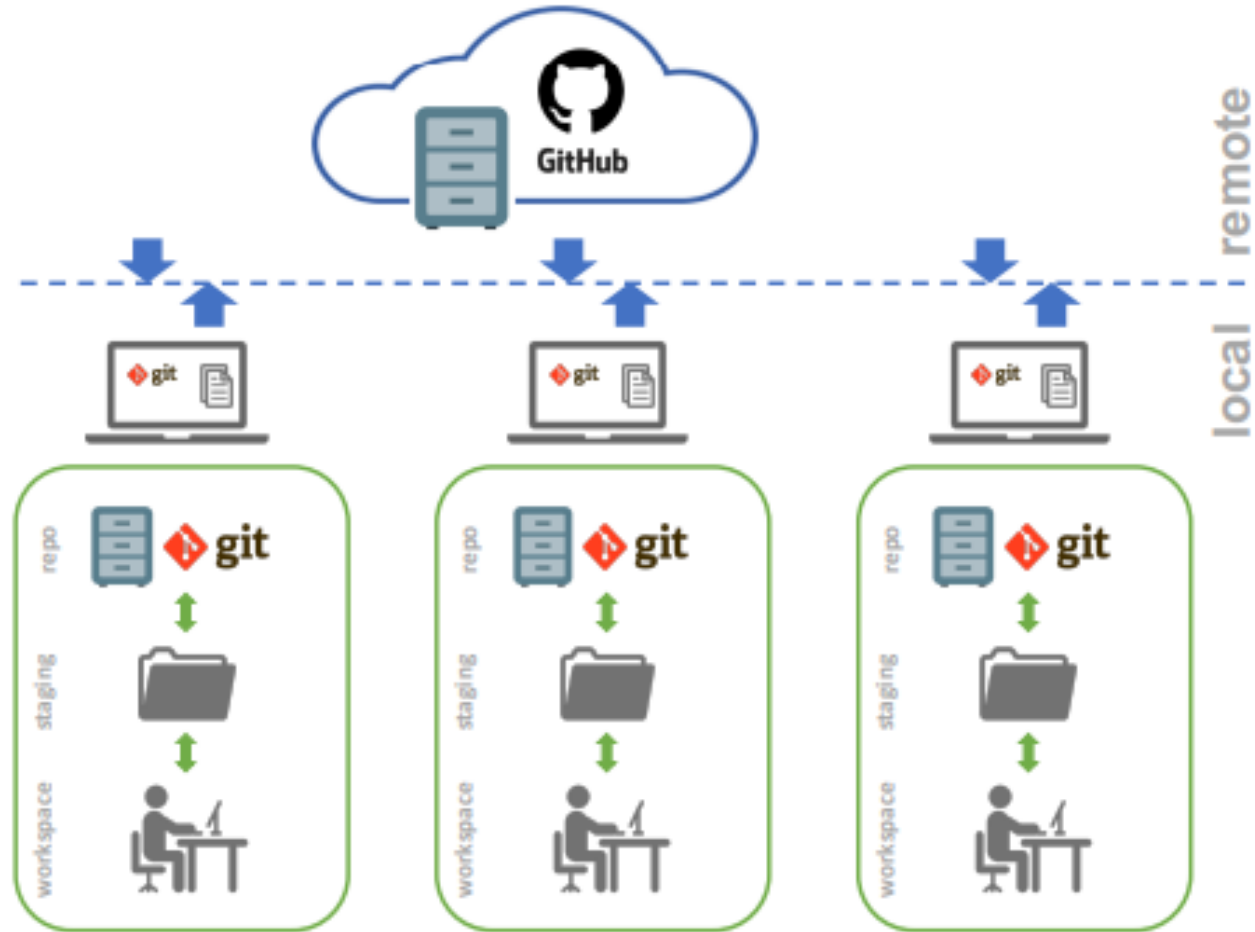
*10 million registered users and over 28 million repositories
(as of 2022)*

Example Clients



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

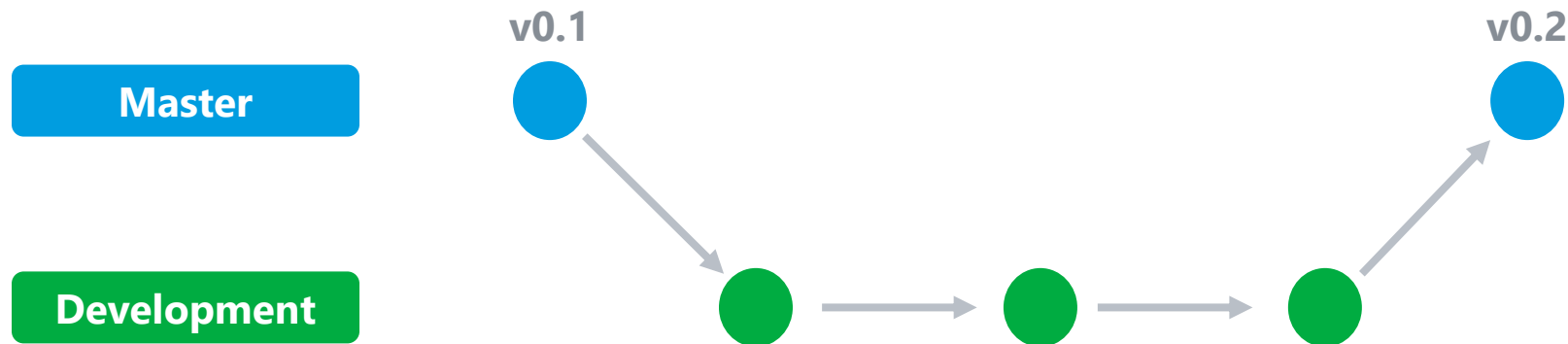


git: A mental model

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

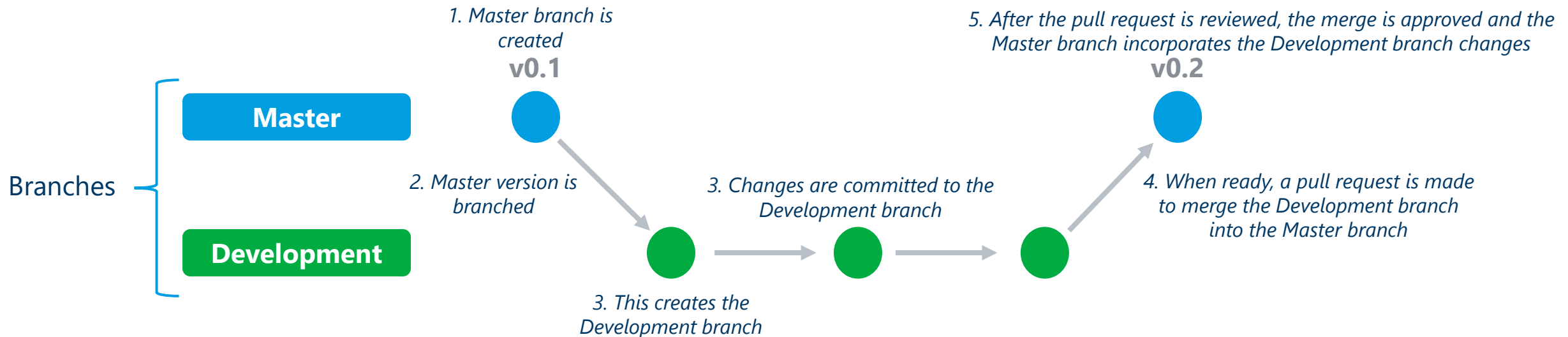
There is a well-defined workflow for making these separate versions and merging them together, which is important to understand

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 [thematic co...](#)

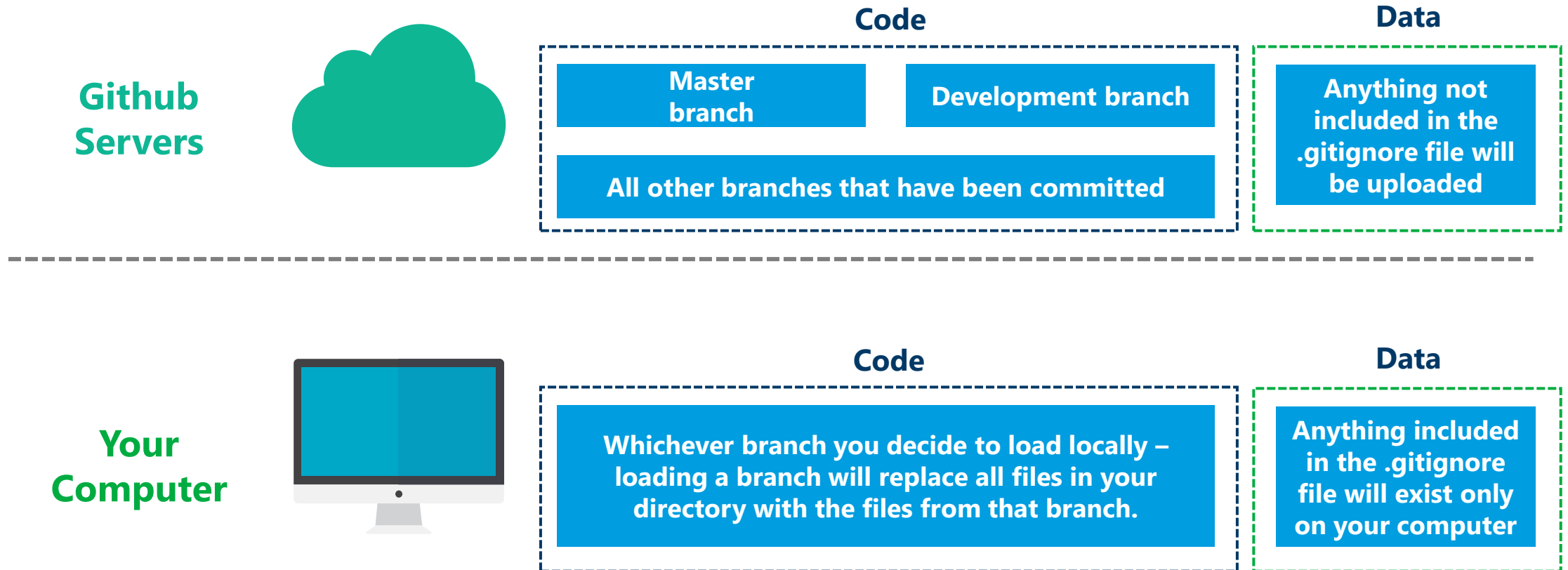
ROPSO Analysis.Rmd

2 Code/Checking [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

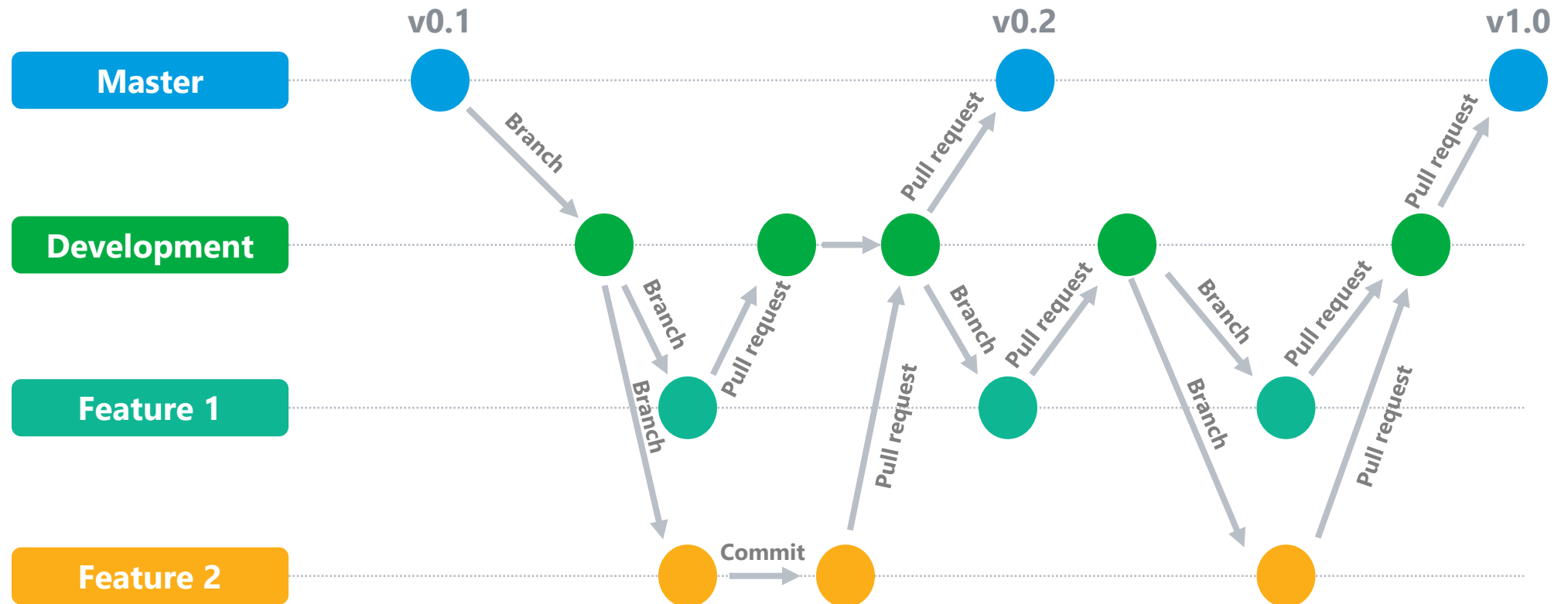
GitHub is a cloud service that hosts git repositories, in other words, all your code is stored on GitHub servers



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

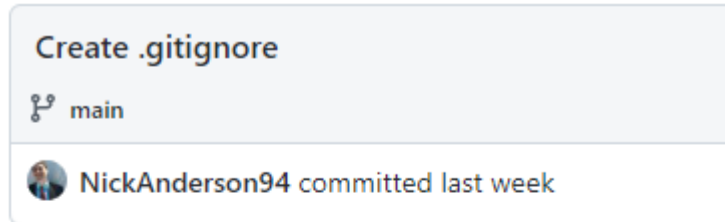
GitHub benefits

- 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



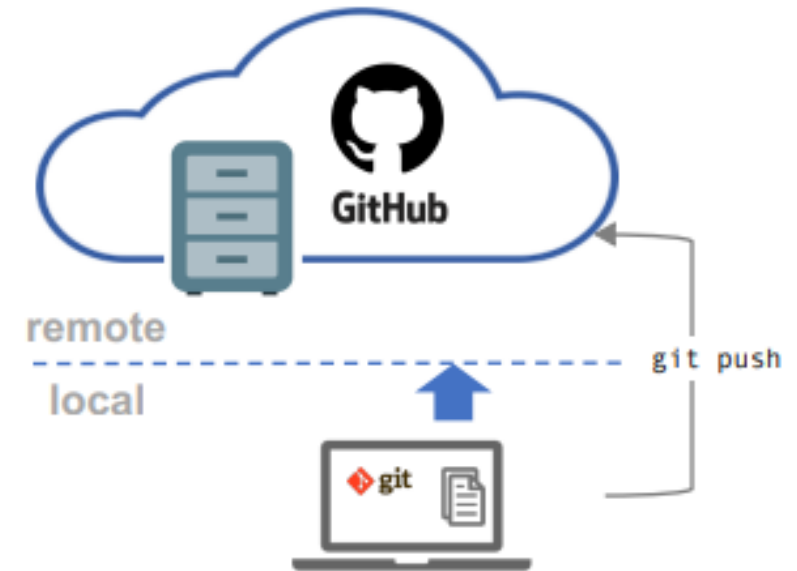
Showing 1 changed file with 5 additions and 0 deletions.



- 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

- 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

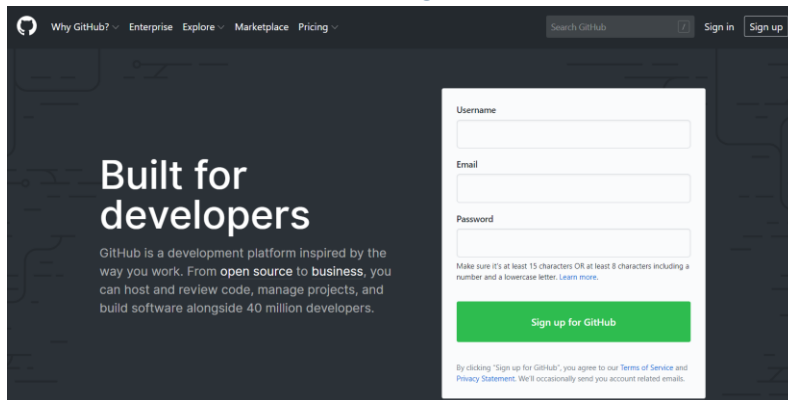
3

Tools you will need

Github.com

<https://github.com/>

- The central source for all code and project management



Optional: Use git through the command line

- For advanced users

Github Desktop

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

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



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:

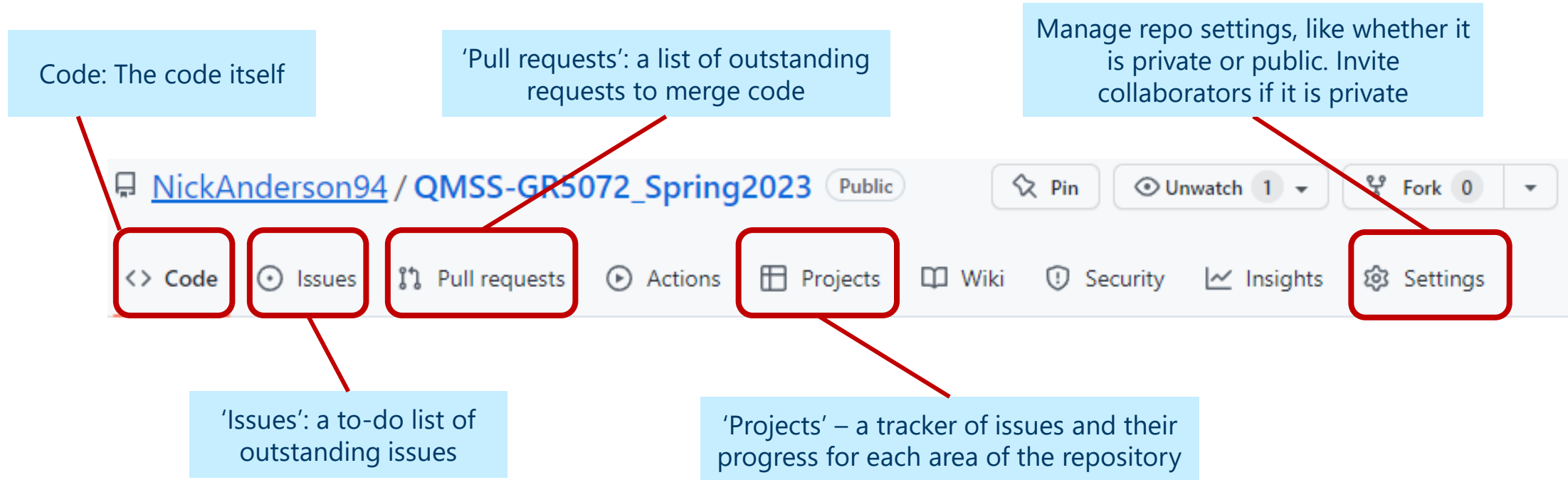
- Accessing your repositories
- Creating a new repo
- Within a repo:
 - Code
 - Issues
 - Pull Requests
 - Settings
 - How to add **collaborators**
 - How to **protect** a branch (so only an **owner** can modify it)

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



Using Github Web: Code

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

The screenshot shows the GitHub web interface for the repository 'NickAnderson94 / QMSS-GR5072_Spring2023'. The 'Code' tab is selected, showing a list of files and folders. Annotations with red boxes and lines point to various features:

- Select branch:** Points to the 'main' branch dropdown menu.
- Files and folders within the branch:** Points to the file list showing 'General Course Materials', 'Week 1/Programs', and '.gitignore'.
- See coding languages stored in the repo:** Points to the 'Languages' section at the bottom, which shows a bar chart for 'Jupyter Notebook 94.0%' and 'Python 6.0%'.
- Download a copy of the repository:** Points to the green 'Code' button with a download icon.
- Click to view change history:** Points to the '11 commits' link.

The repository details on the right include: 'About' (This is the course repo for GR 5072. Modern Data Structures (Spring 2023)), '1 star', '1 watching', '0 forks', 'Releases' (No releases published), and 'Packages' (No packages published).

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

The screenshot shows the Github interface for the angular/angular repository. At the top, there are tabs for Code, Issues (2.5k), Pull requests (292), Actions, Projects (6), Security, and Insights. The Issues tab is selected. Below the tabs, there are filters (is:issue is:open), labels (204), and milestones (10). A green 'New issue' button is highlighted with a red box. Below the filters, there are statistics: 2,452 Open and 19,744 Closed. A table of issues is displayed with columns for Author, Label, Projects, Milestones, Assignee, and Sort. The first issue, 'Create a "Directives Overview" or similar topic', is highlighted with a red box. To its right, the assignee's profile picture is also highlighted with a red box. Below the first issue, there are four more issues listed, each with its title, labels, and status.

angular / angular

Watch 3.2k Star 70.5k Fork 18.5k

Code Issues 2.5k Pull requests 292 Actions Projects 6 Security Insights

Filters is:issue is:open Labels 204 Milestones 10 New issue

2,452 Open 19,744 Closed

Author Label Projects Milestones Assignee Sort

Create a "Directives Overview" or similar topic P3 comp: docs feature #40755 opened 2 hours ago by aikidave Backlog

Make FormArray's inability to be treated as a regular Array more prominent in the docs comp: docs comp: docs/api comp: forms #40753 opened 6 hours ago by ychaikin needsTriage

bug(cdk-drag-drop): :enter / :leave animations are triggered incorrectly comp: animations #40752 opened 9 hours ago by udos86 needsTriage

The structural directive type check docs section is incomplete comp: docs #40739 opened 2 days ago by ArielGueta needsTriage

Provide usage examples for APP_INITIALIZER token comp: docs hotlist: community-help #40730 opened 3 days ago by AndrewKushnir needsTriage

More precise type for APP_INITIALIZER token comp: core cross-cutting: types feature flag: breaking change hotlist: community-help #40729 opened 3 days ago by AndrewKushnir Backlog

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 screenshot shows a GitHub issue titled "Finalize color scheme #8". The page is annotated with red boxes and lines pointing to various elements, which are explained in text boxes on the left and right.

Annotations on the left:

- The initial detail of the issue:** Points to the top section of the issue, including the title, status (Open), and the user who opened it.
- A comment on the issue:** Points to one of the comments in the list.
- Details of changes to the issue's categorization:** Points to the "Labels" and "Projects" sections on the right side of the issue.
- Place to enter a new comment on the issue:** Points to the "Write" section at the bottom of the page.

Annotations on the right:

- List of those assigned to the issue:** Points to the "Assignees" section.
- Labels for the issue:** Points to the "Labels" section.
- Projects the issue is tagged to, and its status within them:** Points to the "Projects" section.
- Example labels:** Points to a separate panel on the far right showing a list of labels with their descriptions.

Example labels panel:

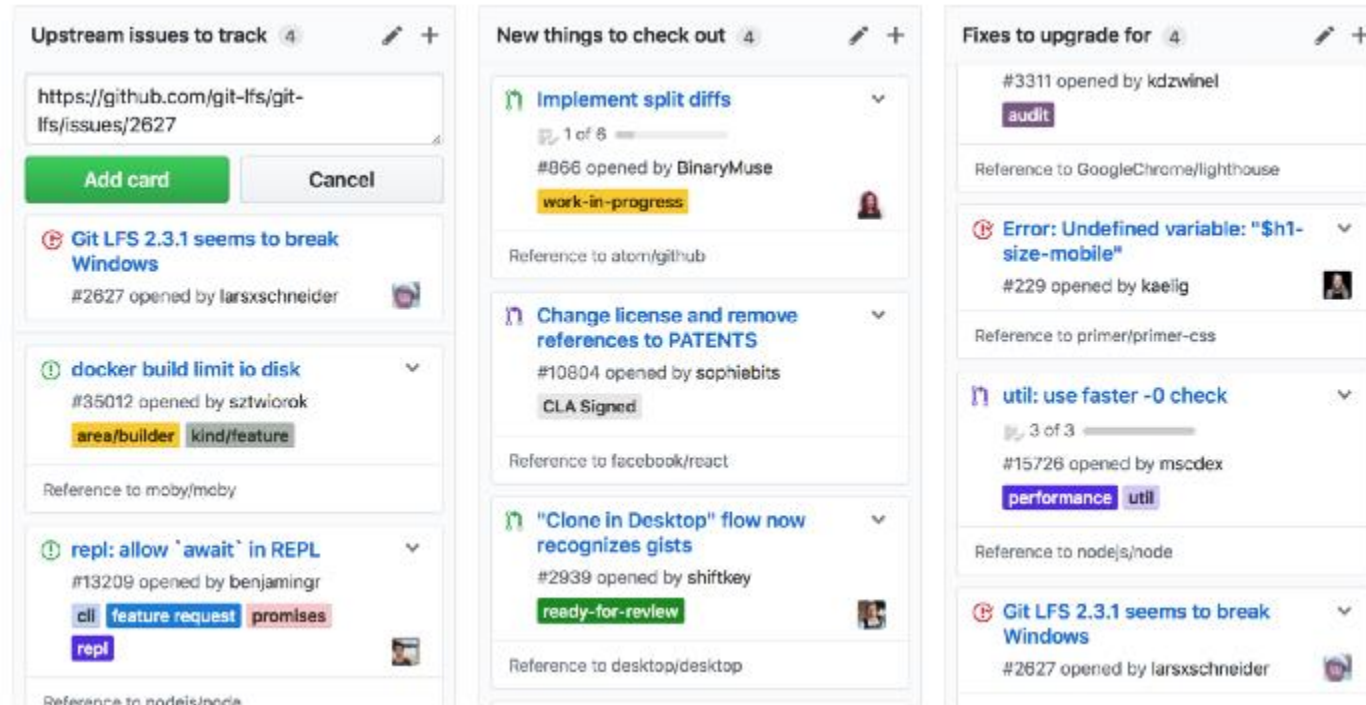
Apply labels to this issue

Filter labels

- ☒ enhancement
New feature or request
- ☐ bug
Something isn't working
- ☐ documentation
Improvements or additions to documentation
- ☐ duplicate
This issue or pull request already exists
- ☐ good first issue
Good for newcomers
- ☐ help wanted
Extra attention is needed
- ☐ invalid
This doesn't seem right
- ☐ question
Further information is requested

Using Github Web: Projects

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



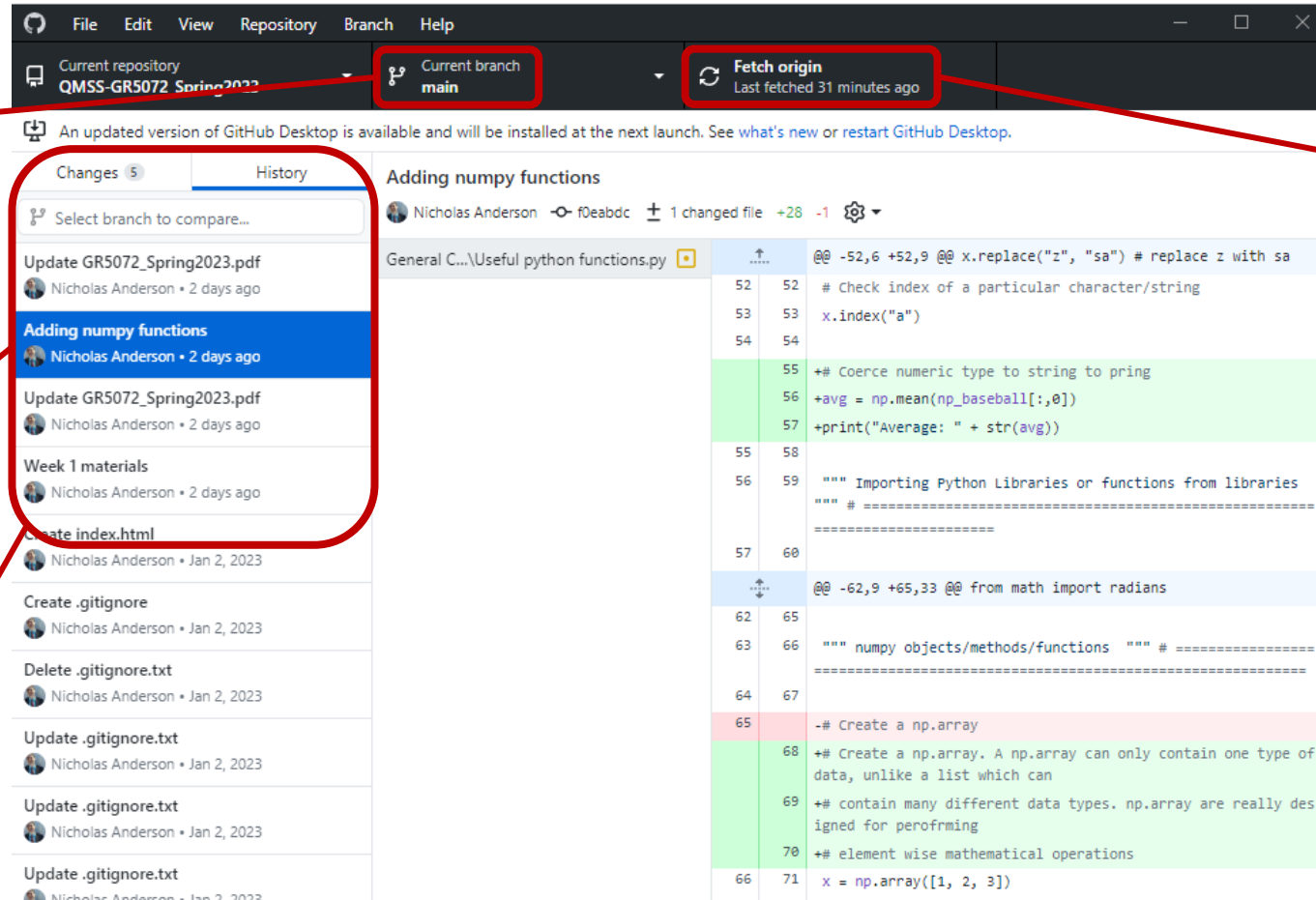
Using GitHub Desktop

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

4

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”

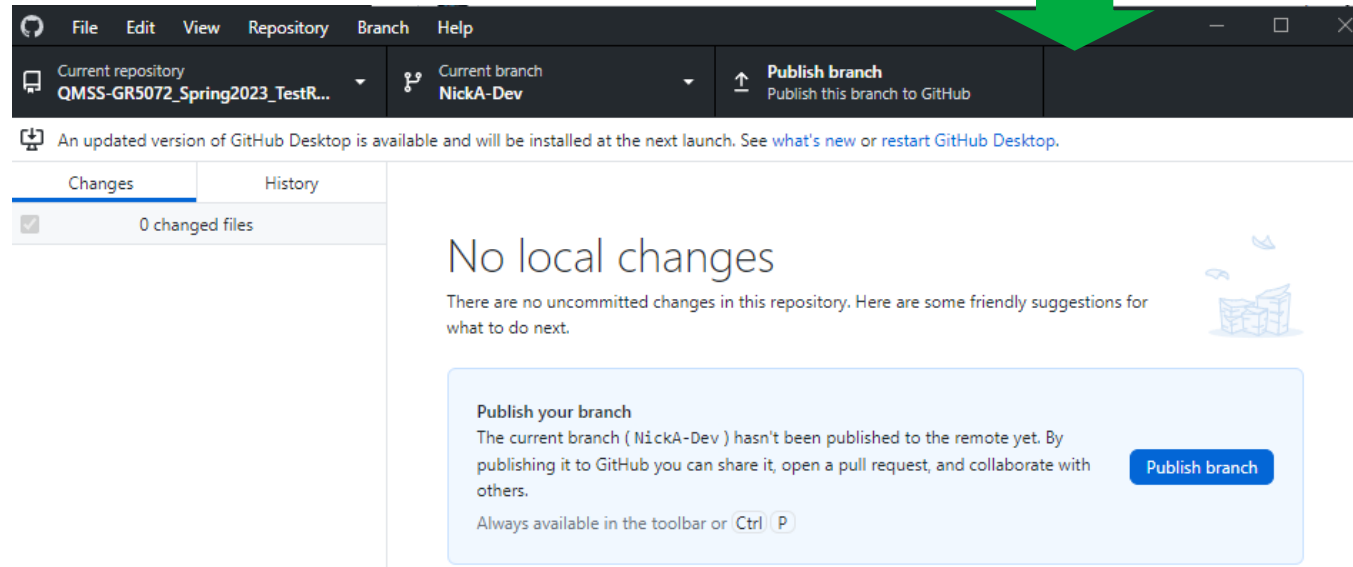
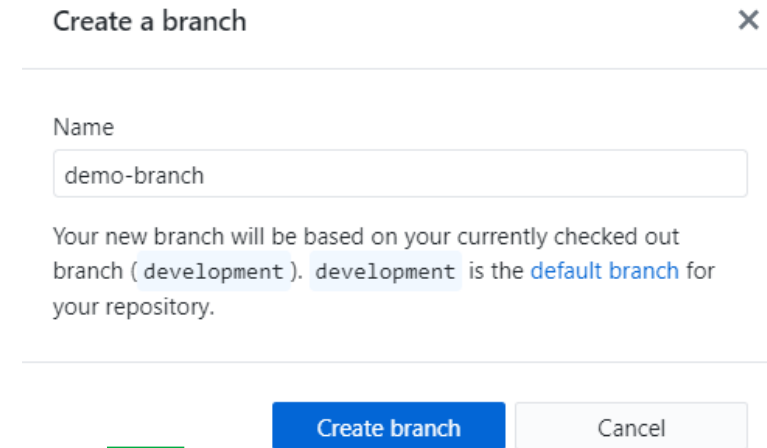
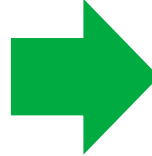
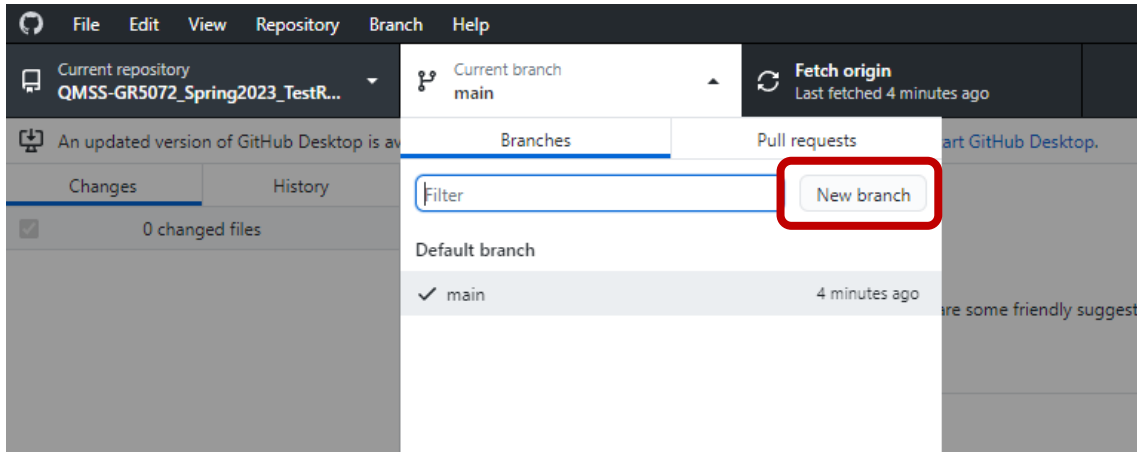
Note: These first three steps are beyond the scope of this class, but are typical steps if you are working on a large team. I therefore don't include walkthroughs on these in the subsequent slides

- 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

The screenshot shows the Github Desktop interface with three main sections: 'Changes', 'History', and a file diff view for 'README.md'. The 'Changes' section on the left shows '1 changed file' and 'README.md'. The diff view on the right shows the changes to the README.md file, with a new line added (line 10) highlighted in green. The commit dialog at the bottom is open, showing the commit message 'Readme updated with test line' and a 'Commit to demo-branch' button. Three callout boxes provide additional context: 'List of changes that have been made' points to the 'Changes' section, 'Detail of changes' points to the diff view, and 'Add each commit with details of what has changed. This will commit it to this new branch' points to the commit dialog.

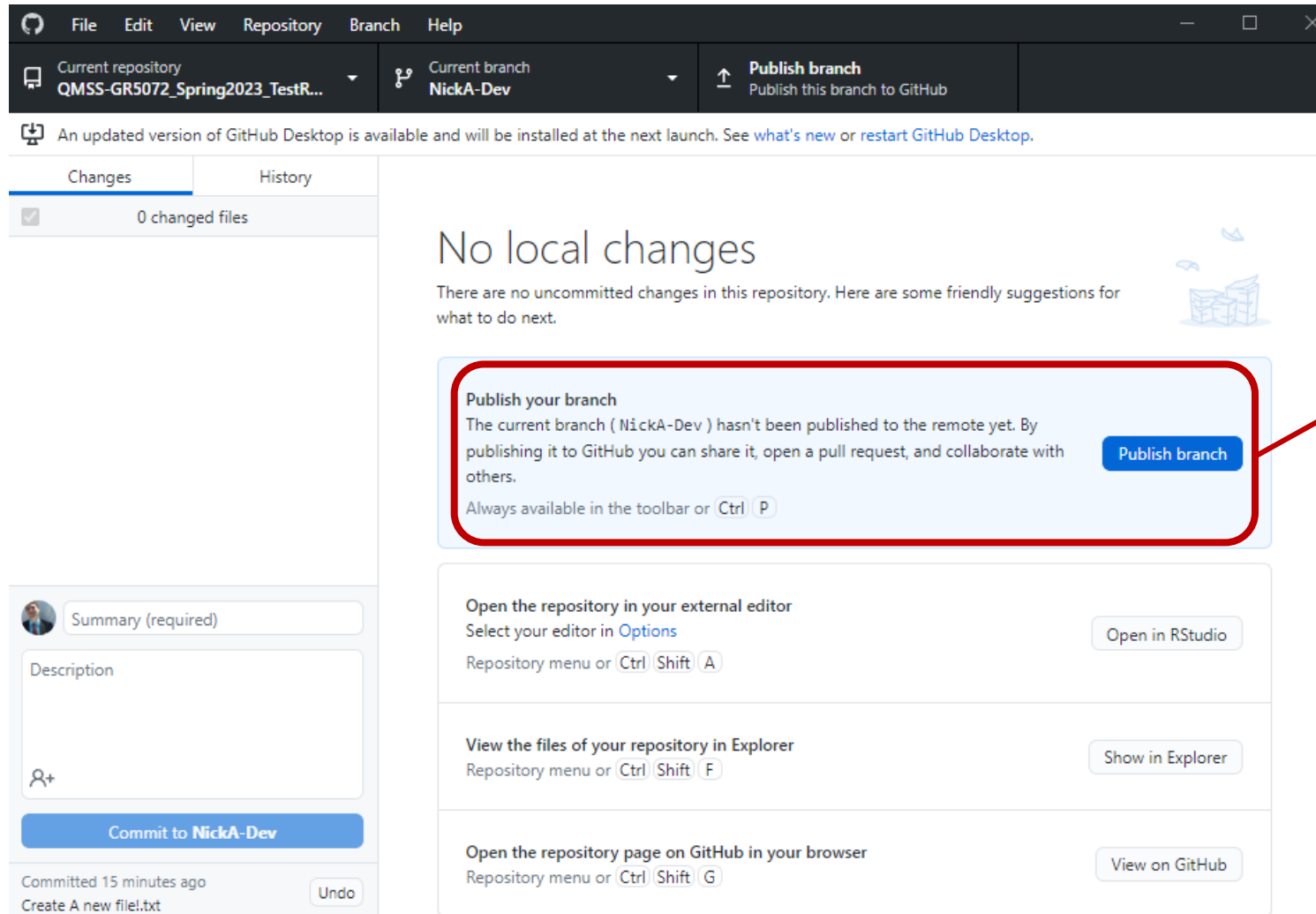
List of changes that have been made

Detail of changes

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

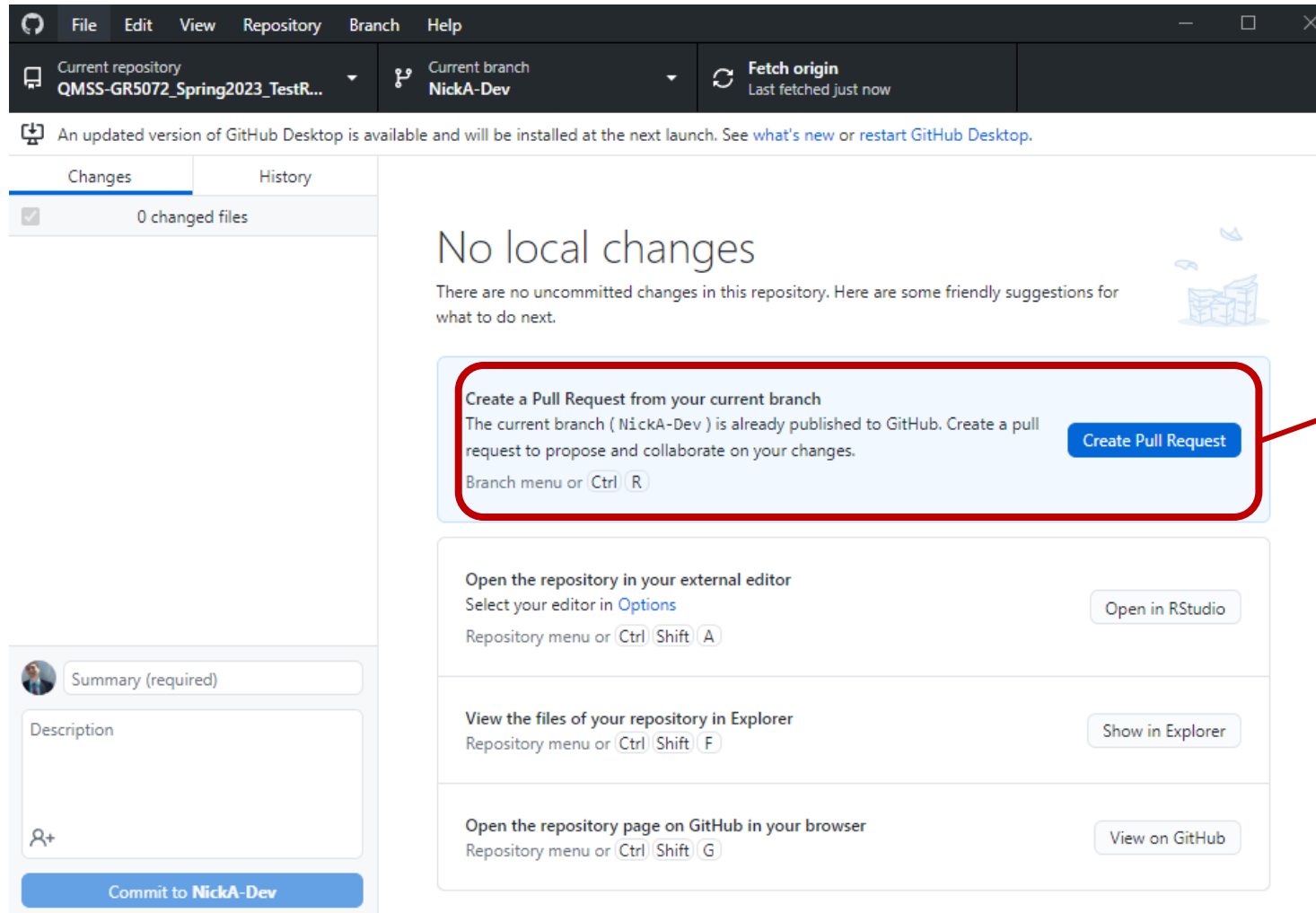
Making a change (7)

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



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



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

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main

compare: NickA-Dev

✓ Able to merge. These branches can be automatically merged.



Nick a dev

Write

Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Use [Closing keywords](#) in the description to automatically close issues

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Making a change (9)

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

The screenshot shows a GitHub Pull Request page for a repository named 'Nick a dev #1'. The top navigation bar includes links for Code, Issues, Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. The Pull requests tab is active, showing a pull request titled 'Nick a dev #1' by NickAnderson94. The pull request description states: 'NickAnderson94 wants to merge 2 commits into main from NickA-Dev'. Below the title, there are tabs for Conversation (0), Commits (2), Checks (0), and Files changed (2). The Conversation tab is selected, showing a comment by NickAnderson94: 'Adding data and an extra file!'. Below the comment, a commit history is shown with two commits: 'Update class_roster.py' (b6aa571) and 'Create A new file!.txt' (0baad4a). At the bottom, there is a section for 'Add more commits by pushing to the NickA-Dev branch on NickAnderson94/QMSS-GR5072_Spring2023_TestRepo.' This section contains three items: 1. 'Require approval from specific reviewers before merging' (Branch protection rules ensure specific people approve pull requests before they're merged.) with an 'Add rule' button. 2. 'Continuous integration has not been set up' (GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.) 3. 'This branch has no conflicts with the base branch' (Merging can be performed automatically.) At the bottom of this section is a green 'Merge pull request' button and a link to 'You can also open this in GitHub Desktop or view command line instructions.'

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

The screenshot shows a GitHub Pull Request page. At the top, the navigation bar includes links for Code, Issues, Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. The main heading is 'Nick a dev #1'. Below it, a green 'Open' button is followed by the text 'NickAnderson94 wants to merge 2 commits into main from NickA-Dev'. A secondary bar shows 'Conversation 0', 'Commits 2', 'Checks 0', and 'Files changed 2'. On the right, there is a '+1 -1' status and a 'Review changes' button, which is highlighted with a red rectangle. A red arrow points from a blue callout box to this button. The left sidebar shows a search bar 'Filter changed files' and a file tree with 'Programs' containing 'A new file!.txt' and 'class_roster.py'. The main content area shows two files: 'Programs/A new file!.txt' (empty) and 'Programs/class_roster.py'. The diff for 'class_roster.py' shows a change on line 14, where a new line is added: 'class_roster.append("Mr. Anderson")'.

<> Code Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

Nick a dev #1

Edit <> Code

Open NickAnderson94 wants to merge 2 commits into main from NickA-Dev

Conversation 0 Commits 2 Checks 0 Files changed 2 +1 -1

Changes from all commits File filter Conversations 0 / 2 files viewed Review changes

Filter changed files

- Programs
 - A new file!.txt
 - class_roster.py

Programs/A new file!.txt

Empty file.

Programs/class_roster.py

```
@@ -11,7 +11,7 @@
11 11 # to the class_roster list! Make sure your code is added above the print statement!
12 12
13 13 class_roster.append("Nick Anderson")
14 -
14 + class_roster.append("Mr. Anderson")
15 15
16 16
17 17
```

Review changes here

Making a change (9)

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

The screenshot shows a GitHub Pull Request titled "Nick a dev #1" by user NickAnderson94. The interface includes a top navigation bar with links for Code, Issues, Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. Below the title, there's a green "Open" button and a summary: "NickAnderson94 wants to merge 2 commits into main from NickA-Dev". A progress bar shows 0 conversations, 2 commits, 0 checks, and 2 files changed. A "Review changes" button is visible. On the left, a file filter shows "Programs" with files "A new file!.txt" and "class_roster.py". The main area displays a diff for "Programs/class_roster.py". The diff shows a new line added at line 14: `class_roster.append("Mr. Anderson")`. A red box highlights the "+" sign next to the new line. A blue callout box with a red arrow points to this box, containing the text: "Click or drag at the side to add comments to the code".

Code Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

Nick a dev #1

Edit Code

Open NickAnderson94 wants to merge 2 commits into main from NickA-Dev

Conversation 0 Commits 2 Checks 0 Files changed 2 +1 -1

Changes from all commits File filter Conversations 0 / 2 files viewed Review changes

Filter changed files

Programs

- A new file!.txt
- class_roster.py

Programs/A new file!.txt

Empty file.

Programs/class_roster.py

```
@@ -11,7 +11,7 @@
11 11 # to the class_roster list! Make sure your code is added above the print statement!
12 12
13 13 class_roster.append("Nick Anderson")
14 -
14 + class_roster.append("Mr. Anderson")
15 15
16 16
17 17
```

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

The screenshot displays a GitHub Pull Request (PR) for the repository 'Nick a dev #1'. The PR is titled 'NickAnderson94 wants to merge 2 commits into main from NickA-Dev'. The interface shows the 'Files changed' tab with two files: 'Programs/A new file!.txt' and 'Programs/class_roster.py'. The 'class_roster.py' file is open, showing a diff with two changes: a new line of code and a modification to an existing line. The diff shows line 14 being changed from a single hyphen to a line that appends 'Mr. Anderson' to the 'class_roster' list. Below the diff, there is a review box with a 'Write' tab and a 'Preview' tab. The 'Write' tab is active, showing a text area with the comment 'This is not a student!'. Below the text area, there are buttons for 'Cancel', 'Add single comment', and 'Start a review'. The 'Start a review' button is highlighted in green. The interface also includes a sidebar with a search bar and a list of files, and a top navigation bar with links to Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

<> Code Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

Nick a dev #1

Edit <> Code

Open NickAnderson94 wants to merge 2 commits into main from NickA-Dev

Conversation 0 Commits 2 Checks 0 Files changed 2 +1 -1

Changes from all commits File filter Conversations 0 0 / 2 files viewed Review changes

Filter changed files

Programs

- A new file!.txt
- class_roster.py

Programs/A new file!.txt

Empty file.

Programs/class_roster.py

```
@@ -11,7 +11,7 @@
11 11 # to the class_roster list! Make sure your code is added above the print statement!
12 12
13 13 class_roster.append("Nick Anderson")
14 -
14 + class_roster.append("Mr. Anderson")
```

Write Preview

This is not a student!

Attach files by dragging & dropping, selecting or pasting them.

Cancel Add single comment Start a review

15 15
16 16
17 17

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

The screenshot shows a GitHub Pull Request (PR) titled "Nick a dev #1" by user NickAnderson94. The PR is open and shows 2 commits merged into the main branch from the NickA-Dev branch. The interface includes a navigation bar with links to Code, Issues, Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. Below the title, there are buttons for "Open", "Edit", and "Code". The PR description indicates that NickAnderson94 wants to merge 2 commits into main from NickA-Dev. The "Files changed" section shows two files: "A new file!.txt" and "class_roster.py". The "class_roster.py" file is highlighted, showing a diff view with line numbers 11 to 17. A comment from NickAnderson94 is visible, stating "This is not a student!". A "Finish your review" modal is open, showing options to "Comment", "Approve", or "Request changes". A "Submit review" button is at the bottom of the modal. On the left side, a sidebar shows a list of files changed, including "A new file!.txt" and "class_roster.py". At the bottom left, a green box contains a "Merge pull request" button and a note: "You can also open this in GitHub Desktop or view command line instructions." A pink circle with the number "2" is next to the "Merge pull request" button. A pink circle with the number "1" is next to the "Submit review" button in the modal.

2 Merge pull request You can also open this in GitHub Desktop or view command line instructions.

1 Submit review 1 pending comment

