Learn to Code

Team Collaboration and GitHub

Student Workbook #6c

Version 5.1 Y

Table of Contents

Module 1

Team Collaboration and GitHub

Section 1–1

Team Collaboration and GitHub

Team Collaboration on GitHub

- **GitHub is a service that allows you to make your Git repositories available to other developers**

- **It also provides a GitHub project board that helps teams collaborate by providing a centralized "board" for managing:**
  - backlog items
  - in-process items
  - completed items

- **It allows you to specify collaborators that can contribute to your code base**

- **It provides support for collaboration, including**
  - working in branches
  - creating pull requests (or merge requests)
  - merging branches

Working in a Team

- **On a development team, collaborating can quickly become a delicate process**
  - Without a strategy in place, one team member making contributions to the same project as another will result in them stepping on each other's toes

- **When working together, when one team member makes changes ahead of another member, there can be issues with their ability merge their code**

- **Using collaboration tools like branching, you can prevent some of these complications**

- **There are, however, many that can still appear without a defined branching strategy!**

Section 1–2

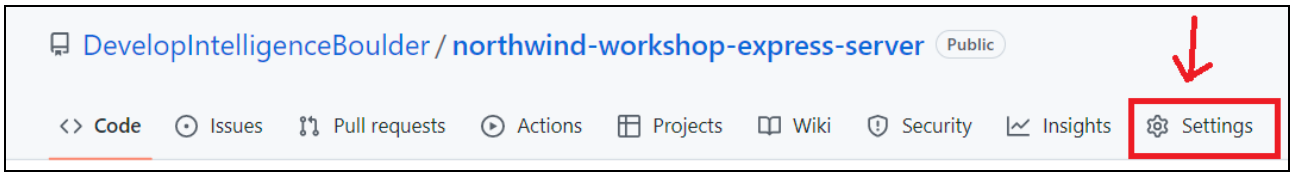
Setting Up Your Team

Setting Up Your Team

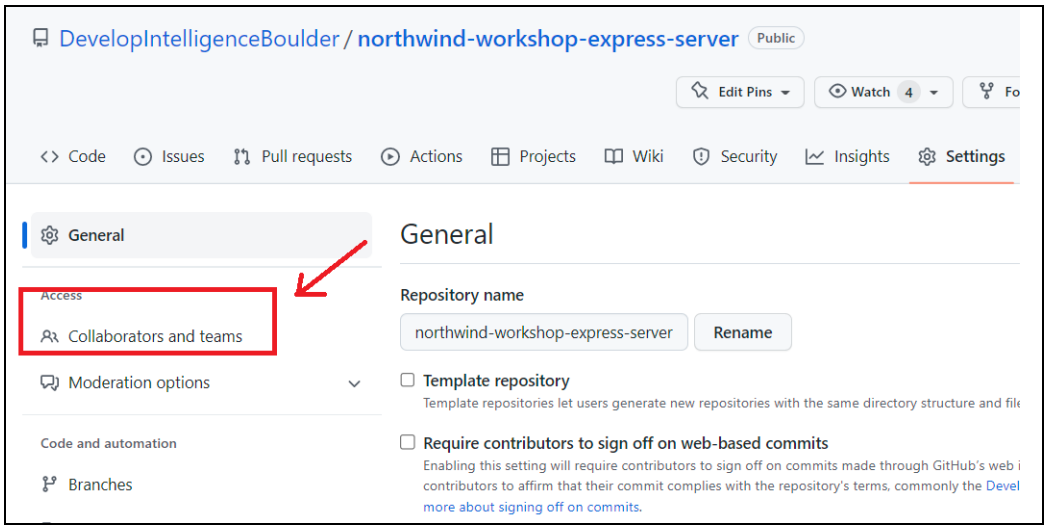
- **When you get ready to start on a team project, the first thing you must do is set up your team!**

- **One your GitHub repo is created, you must add your team members to the repo as collaborators**
  - Collaborators can push and merge to the repo, so be selective on who you add!

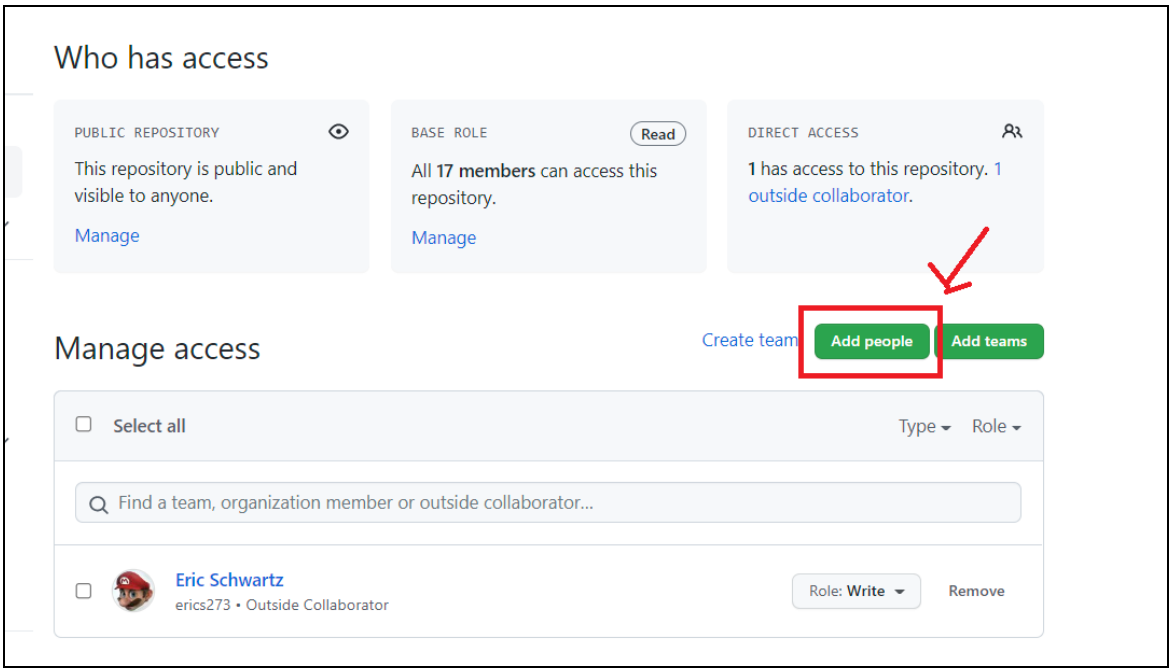
- **To add collaborators, go to the "Settings" tab of your repo**



- **Then click on "Collaborators and teams"**



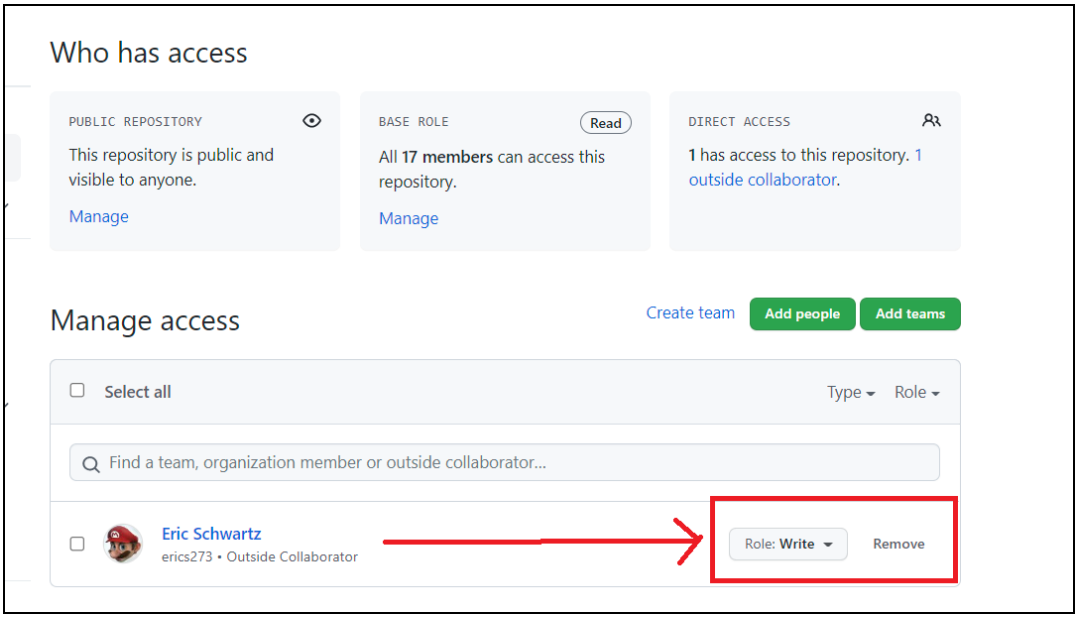
- **Click the "App people" button and specify the email or GitHub username of the collaborator you want to add**
  - They will receive an email letting them know they have been added as a collaborator to the repo



Now That You Are A Collaborator…

- **Once you get your email, go to the GitHub repo page and:**
  - Clone the project to your local computer
  - cd into the folder
  - Open it in VS Code
- **Always remember: you and your teammates want to collaborate using the same GitHub repo**

GitHub Repo Owner Can Also Manage Collaborators



- **You can limit the abilities of a collaborator by changing their role in the dropdown to the left of their name from "Write" to "Read"**

- **You can remove a collaborator by clicking "Remove" in the dropdown to the left of their name**

Section 1–3

Branching, Pull Requests, and Merging

Branching

- **Recall that a branch can hold code that isn't part of the main branch**
  - In a collaborative environment, team members code in a branch dedicated to the feature they are coding
  - Sometimes programmers create throw-away branches to test ideas without affecting the main code base
- **A branch is a divergence from a repository's main history**
  - It allows us to create new changes and commit them to a shared repository without having to affect the main branch
- **Once a divergent branch has a feature that is ready, it can be merged back into the main branch**
  - Using branching in a Git repository, and learning branching strategies, we can effectively execute as a team.

Working with Branches

- **Below are steps that any developer or individual contributor can follow when working on a task**
  - Create a branch for you to do your work in for the task you have selected
    - To keep things simple, use a descriptive name of the task as the branch name (ex: `add_details_page`)

Example

```
git checkout -b new_feature
```

  - Make sure your branch is tracked by the remote repository

Example

```
git push -u origin new_feature
```

  - Work on your task! Add and commit your changes as often as possible.

Example

```
git add . (instead of add . you can list out individual files)
git commit -m "description of change I made"
git push
```

Merging Code Back into the "Main" Branch When You Are the Only Developer

- **You can merge any new changes from main into the feature branch**

  Example

  ```
  git pull origin main
  ```

- **If there are any conflicts, fix, add, and commit the updates**

- **Now, we need to get our changes into the main branch and pushed to the remote repo (origin)**

  – Make sure you switch back to the branch you want to merge the changes into before using `git merge`

  Example

  ```
  git checkout main
  git merge new_feature
  git push
  ```

- **But in a team environment, there are more issues!**

The Collaborative Quirk and Pull Requests

- **As you will soon discover, team members can work intensively over the course of several hours or even a day on separate features in their own branches**

- **But at some point, they complete their work and want to merge them back into the main branch**

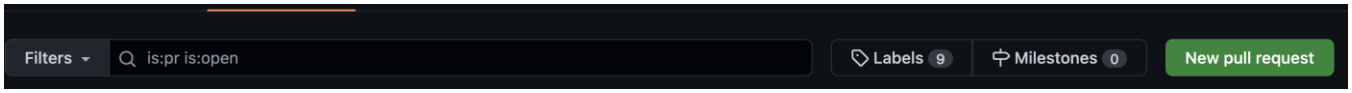
- **This is a dangerous time!**
  - If each team member is working on a code base that doesn't overlap, they will be in good shape
  - But, for example, what if each member is editing, adding, and deleting CSS rules in `styles.css`???
  - What happens during the merge?
  - CONFLICTS!

- **Developers can push their feature branch to GitHub**

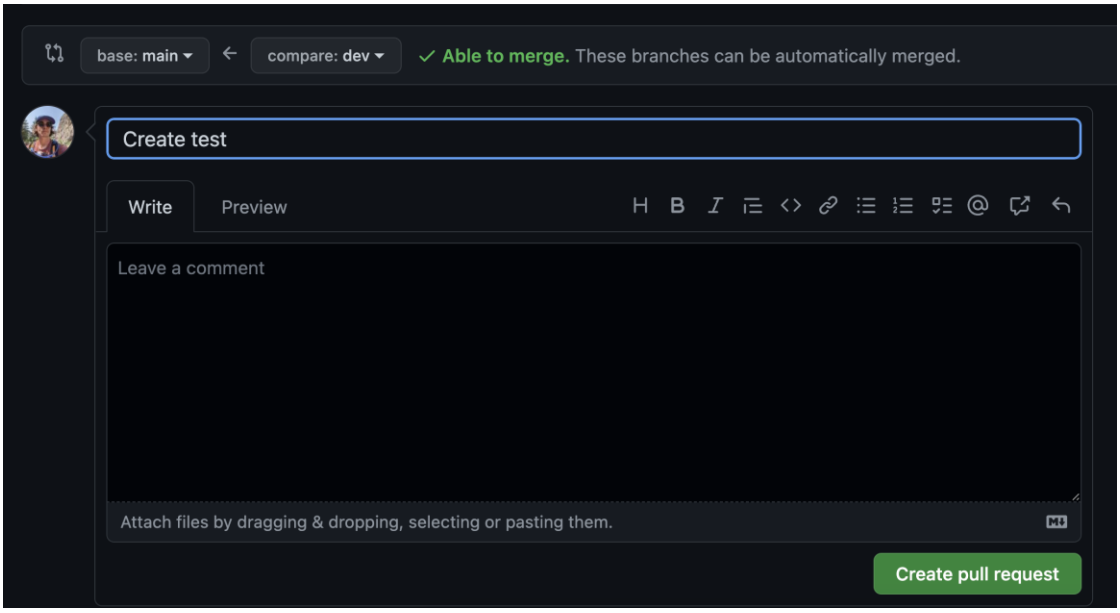
- **Then, on GitHub, they can create a *pull request* (or merge request) to ask for changes in a branch to be merged into the main branch (or any other branch)**
  - When a pull request is opened, the collaborator can discuss and review the branch changes with their teammates
  - The owner of the project will then review the changes and merge them if all is good

Creating a Pull Request

- **When you go to the “Open a pull request” page, you can click "New Pull Request"**

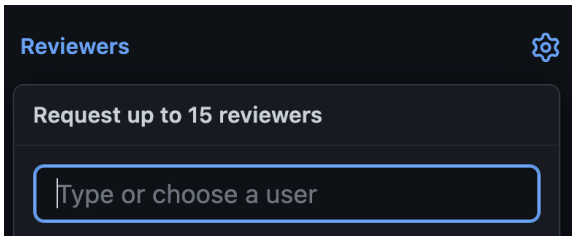


  - You should enter a brief description of what you changed in your branch



- **You can then select a reviewer, and that would be whoever your team decided would be the “pull request owner”**

  - You can have more than one team member designated for this role



- **When you’re done, click “Create pull request”**

Workflow and Pull Requests

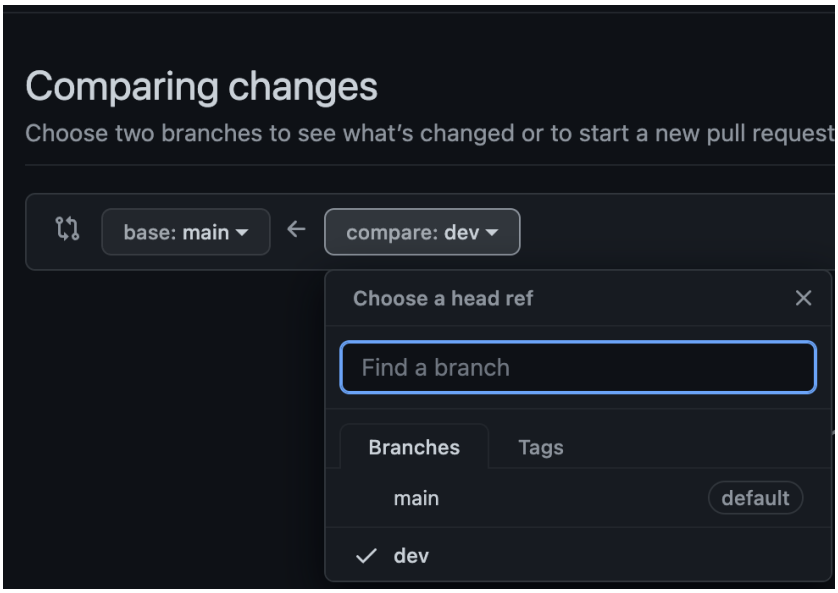
- **Before submitting a pull request to the *`main`* branch:**
  - Pull the latest version of the ***`main`*** branch to your local repo
  - Merge it into your feature branch
  - Resolve any merge conflicts which may arise. (If you are nervous, ask your team for assistance!)
  - Push your working branch to GitHub
  - Submit a pull request from your working branch to ***`main`***

Comparing Changes

- **GitHub has a visual tool that allows you to compare the changes between two branches**

- **You can select the branch in the drop-down "Branch:" menu and select the branch you just pushed up**

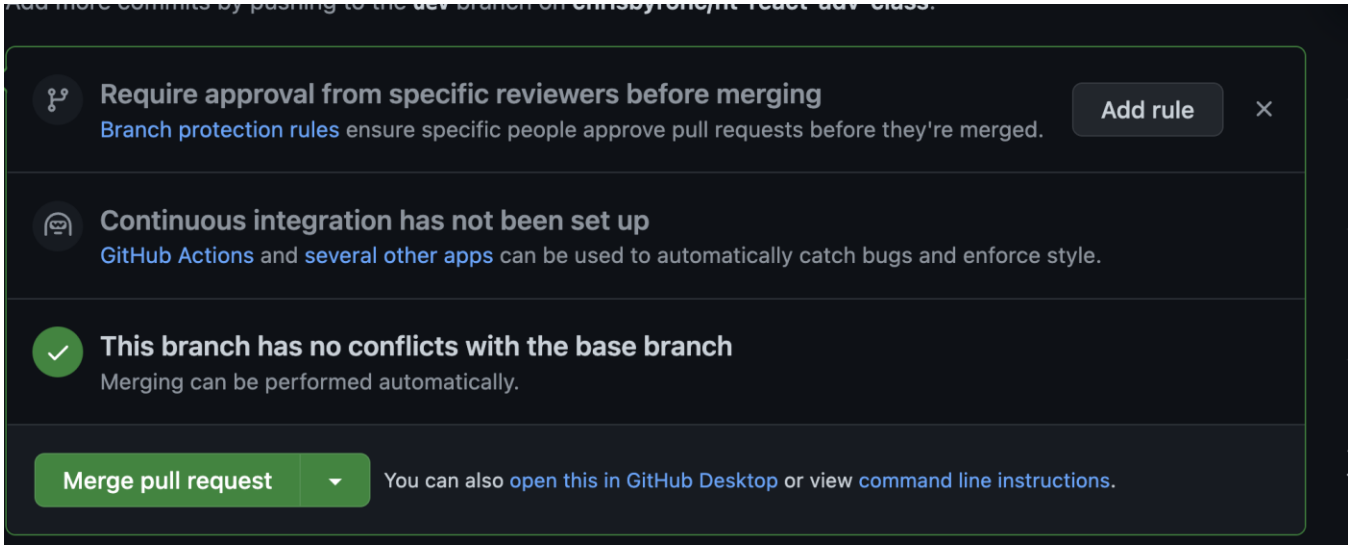
- **You will have a compare option**



Merging Pull Requests

- **The pull request owner will review and merge your code into main**
  - As best practice, the team member who merges the code should never be the same person who wrote it
  - Let fresh eyes understand conflicts

- **After reviewing the pull request, you can merge it**



Section 1–4

Strategies and Branching

Collaborating, Feature Branches and Right-Sizing Features

- **When you are working collaboratively, you want to make sure the master branch is always deployable**

- **You should create a branch for a new feature**
  - Complete it there
  - Commit and push often

- **When you are ready for the new feature to go be deployed, merge it into the main branch**

- **But there is a second, very important consideration of team collaboration and that is…**

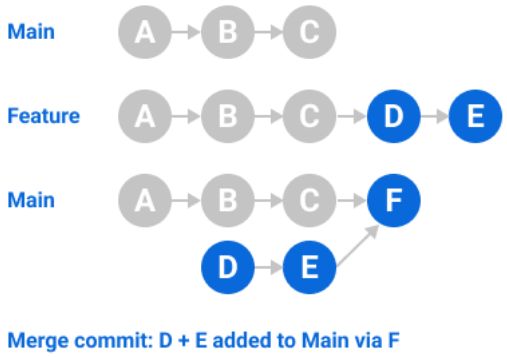
  DON'T DEFINE A FEATURE IN A WAY THAT HAS OVERLAPPING CODE WITH ANOTHER FEATURE!

Branching Strategy: Merging

- **Establishing a good branching strategy, is as simple as dedicating certain teams and versions to a branch.**

- **Let's explore the `main` branch**
  - It is the primary and latest stable working version of a project
  - The `main` branch is where all new features are eventually merged into

- **Any feature branches, before merging into `main`, will need to pull `main` into that feature branch first**
  - Why? Before merging, the feature branch is required to have all history of the branch it is merging into



  - This means that when directly merging, or creating a pull request, we will need to pull the *`main`* or the intended branch to hold the new features changes into the branch with the new feature

Common Collaborating Branches

- **Large development teams often have several branches in addition to their feature branches**
  - These branches are dedicated to the environment to which that branch would be deployed to
- **`main` : This branch maintains the primary and most stable version of a project or application**
- **`dev` : This branch can typically have the rough draft versions of developing features**
  - Developers push their local changes here to create a version that can be deployed to an environment closer to their production environment
    - It is often where code is created to confirm a new feature or bug fix prior to moving it to the main branch
- **`staging`: This branch is dedicated to the staging environment and allows for testing in an environment that is the same as production**
  - Having a staging branch is not as common
- **`qa` : This branch is used with QA teams and is built specifically for the testing of features**

Exercises

EXERCISE 1

In this exercise, we are going to practice a simple creation of a feature branch, creating a pull request, and merging that pull request into the `*main*` branch.

First, you can use an existing repository, or create a new one.

If creating a new project, be sure to load that project with some existing files. Possibly a previously created html file, or a whole website.

1. Start by creating a new feature branch. You can call this "new_feature"

2. Be sure to push the new feature branch so that it can be tracked by the remote repository

3. Update the html file, or other files with new changes

4. Check the status, confirm the changes

5. After confirming the changes, add, commit, and push those changes to the new branch

6. Create a pull request that sets up a merge from the "new_feature" branch, into the "main" branch

7. After creating this PR, you can merge the changes.

EXERCISE 2

Find someone to pair with! Create a new project and then create and add some changes to a branch and create a PR for the other member to approve and merge them!