The Official GitHub Training Manual

Intro to GitHub

Training Manual

Table of Contents

Preparing to Teach

GitHub for Developers is designed for new Git and GitHub users who are comfortable on the command line. Complete the following steps to prepare for class

1. Create a class repository in the `githubteacher` account.
2. Delete the githubteacher copy of the `github-games` repo.

Getting Ready for Class

Step 1: Set Up Your GitHub.com Account

For this class, we will use a public account on GitHub.com. We do this for a few reasons:

- We don’t want you to "practice" in repositories that contain real code.
- We are going to break some things so we can teach you how to fix them. (therefore, refer to #1 above)

If you already have a github.com account you can skip this step. Otherwise, you can set up your free account by following these steps:

1. Access GitHub.com and click Sign up.
2. Choose the free account.
3. You will receive a verification email at the address provided.
4. Click the link to complete the verification process.

Getting Started With Collaboration

We will start by introducing you to Git, GitHub, and the collaboration features we will use throughout the class. Even if you have used GitHub in the past, we hope this information will provide a baseline understanding of how to use it to build better software!

What is GitHub?

GitHub is a collaboration platform built on top of a distributed version control system called Git.

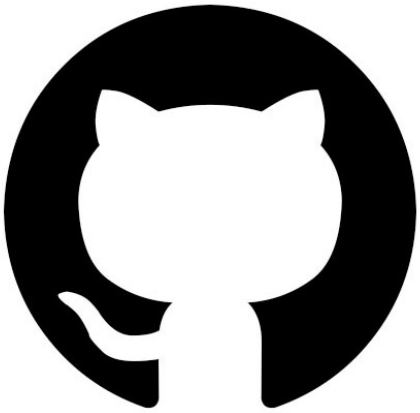

Figure 1. GitHub's beloved Octocat logo.

In addition to being a place to host and share your Git projects, GitHub provides a number of features to help you and your team collaborate more effectively. These features include:

- Issues
- Pull Requests
- Organizations and Teams

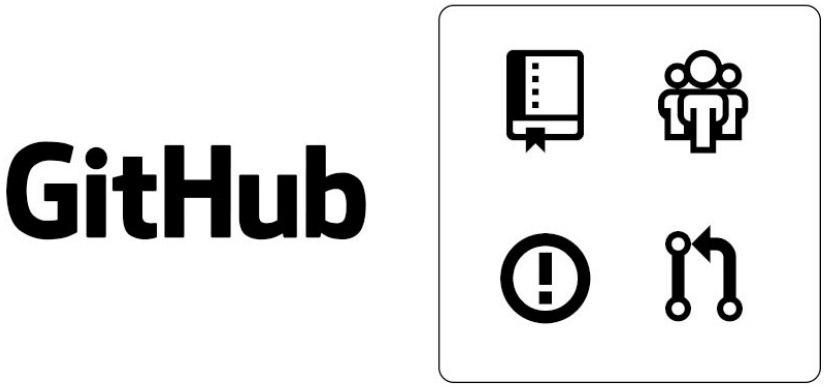

Figure 2. Key GitHub Features.

Rather than force you into a "one size fits all" ecosystem, GitHub strives to be the place that brings all of your favorite tools together. You may even find some new, indispensable tools like continuous integration and continuous deployment to help you and your team build software better, together.

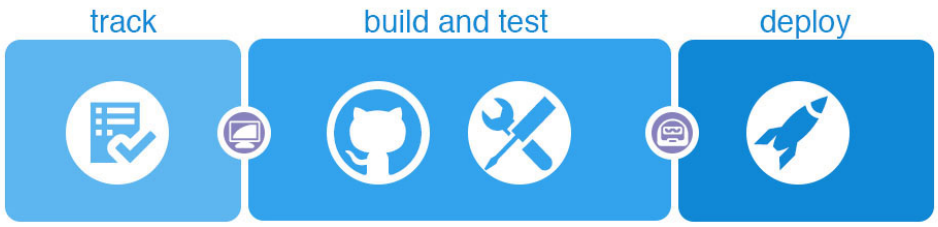

Figure 3. The GitHub Ecosystem.

What is Git?

Git is:

- a distributed version control system or DVCS.
- free and open source.
- designed to handle everything from small to very large projects with speed and efficiency.
- easy to learn and has a tiny footprint with lightning fast performance.

Git outclasses many other SCM tools with features like cheap local

branching, convenient staging areas, and multiple workflows.

As we begin to discuss Git (and what makes it special) it would be helpful if you could forget everything you know about other version control systems (VCSs) for just a moment. Git stores and thinks about information very differently than other VCSs.

We will learn more about how Git stores your code as we go through this class, but the first thing you will need to understand is how Git works with your content.

Snapshots, not Deltas

One of the first ideas you will need understand is that Git does not store your information as series of changes. Instead Git takes a snapshot of your repository at a given point in time. This snapshot is called a commit.

Optimized for Local Operations

Git is optimized for local operation. When you clone a copy of a repository to your local machine, you receive a copy of the entire repository and its history. This means you can work on the plane, on the train, or anywhere else your adventures find you!

Branches are Lightweight and Cheap

Branches are an essential concept in Git.

When you create a new branch in Git, you are actually just creating a pointer that corresponds to the most recent snapshot in a line of work. Git keeps the snapshots for each branch separate until you explicitly tell it to merge those snapshots into the main line of work.

Git is Explicit

Which brings us to our final point for now; Git is very explicit. It does not do anything until you tell it to. No auto-saves or auto-syncing with the remote, Git waits for you to tell it when to take a snapshot and when to send that snapshot to the remote.

Exploring a GitHub Repository

A repository is the most basic element of GitHub. It is easiest to imagine as a project's folder. However, unlike an ordinary folder on your laptop, a GitHub repository offers simple yet powerful tools for collaborating with others. A repository contains all of the project files (including documentation), and stores each file's revision history. Whether you are just curious or you are a major contributor, knowing your way around a repository is essential!

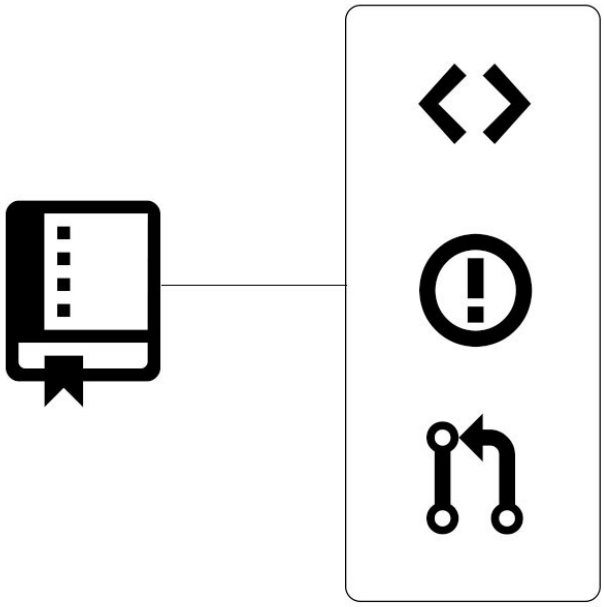

Figure 4. GitHub Repositories.

Repository Navigation

Code

> The code view is where you will find the files included in the repository. These files may contain the project code, documentation, and other important files. We also call this view the root of the project. Any changes to these files will be tracked via Git version control.

Issues

> Issues are used to track bugs and feature requests. Issues can be assigned to specific team members and are designed to encourage

discussion and collaboration.

Pull Requests

: A Pull Request represents a change, such as adding, modifying, or deleting files, which the author would like to make to the repository. Pull Requests are used to resolve Issues.

Wiki

: Wikis in GitHub can be used to communicate project details, display user documentation, or almost anything your heart desires. And of course, GitHub helps you keep track of the edits to your Wiki!

Pulse

: Pulse is your project's dash board. It contains information on the work that has been completed and the work in progress.

Graphs

: Graphs provide a more granular view into the repository activity, including who has contributed, when the work is being done, and who has forked the repository.

README.md

: The README.md is a special file that we recommend all repositories contain. GitHub looks for this file and helpfully displays it below the repository. The README should explain the project and point readers to helpful information within the project.

CONTRIBUTING.md

: The CONTRIBUTING.md is another special file that is used to describe the process for collaborating on the repository. The link to the CONTRIBUTING.md file is shown when a user attempts to create a new issue or pull request.

Using GitHub Issues

Use GitHub issues to record and discuss ideas, enhancements, tasks, and bugs. They make collaboration easier in a variety of ways, by:

- Replacing email for project discussions, ensuring everyone on the team has the complete story.

- Allowing you to cross-link to other issues and pull requests.
- Creating a single, comprehensive record of how and why you made certain decisions.
- Allowing you to easily pull the right people into a conversation.

Activity: Creating A GitHub Issue

Follow these steps to create an issue in the class repository:

Activity Instructions

1. Click the Issues tab.
2. Click **New Issue**.
3. Type a subject line for the issue.
4. A `-` followed by a space and `[ ]` will create a handy checklist in your issue or pull request.
5. When you @mention someone in an issue, they will receive a notification - even if they are not currently subscribed to the issue or watching the repository.
6. A # followed by the number of an issue or pull request (without a space) in the same repository will create a cross-link.
7. Tone is easily lost in written communication. To help, GitHub allows you to drop emoji into your comments. Simply surround the emoji id with `:`.

Using Markdown

GitHub uses a syntax called Markdown to help you add basic text formatting to issues.

Commonly Used Markdown Syntax

`# Header`

: The `#` indicates a Header. # = Header 1, ## = Header 2, etc.

`* List item`

: A single `*` followed by a space will create a bulleted list. You can also use a `-`.

`Bold item`

Two asterix `**` on either side of a string will make that text bold.

`- [ ] Checklist`

A `-` followed by a space and `[ ]` will create a handy checklist in your issue or pull request.

`@mention`

When you @mention someone in an issue, they will receive a notification - even if they are not currently subscribed to the issue or watching the repository.

`#975`

A # followed by the number of an issue or pull request (without a space) in the same repository will create a cross-link.

`:smiley:`

Tone is easily lost in written communication. To help, GitHub allows you to drop emoji into your comments. Simply surround the emoji id with `:`.

Understanding the GitHub Flow

In this section, we will discuss the collaborative workflow enabled by GitHub.

The Essential GitHub Workflow

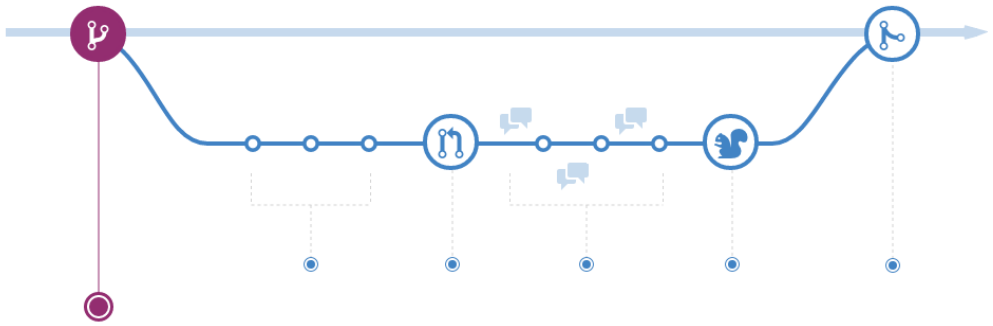

Figure 5. GitHub Workflow.

The GitHub flow is a lightweight workflow that allows you to experiment with new ideas safely, without fear of compromising a project.

Branching is a key concept you will need to understand. Everything in GitHub lives on a branch. By convention, the "blessed" or "canonical" version of your project lives on a branch called `master`.

When you are ready to experiment with a new feature or fix an issue, you create a new branch of the project. The branch will look exactly like `master` at first, but any changes you make will only be reflected in your branch. Such a new branch is often called a "feature" branch.

As you make changes to the files within the project, you will commit your changes to the feature branch.

When you are ready to start a discussion about your changes, you will open a pull request. A pull request doesn't need to be a perfect work of art - it is meant to be a starting point that will be further refined and polished through the efforts of the project team.

When the changes contained in the pull request are approved, the feature branch is merged onto the master branch. In the next section, you will learn how to put this GitHub workflow into practice.

Branching with Git

The first step in the GitHub Workflow is to create a branch. This will allow us to separate our work from the master branch.

Branching Defined

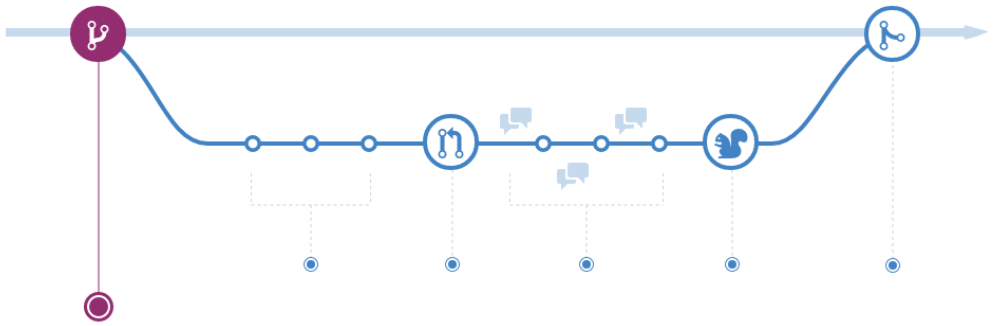

Figure 6. GitHub Workflow.

When you create a branch, you are essentially creating an identical copy of the project at that point in time that is completely separate from the master branch.

This keeps your the code on your master branch safe while you experiment and fix issues.

Let's learn how you can create a new branch.

Activity: Creating A Branch with GitHub

Earlier you created an issue about a change you would like to introduce into the project. Let's create a branch that you will use to add your file.

Follow these steps to create a new branch in the class repository:

Activity Instructions

1. Navigate to the `class repository`.
2. Click the branch dropdown.
3. Enter the branch name 'firstname-lastname-hometown'.

4. Press `Enter`.

When you create a new branch on GitHub, you are automatically switched to your branch. Now, any changes you make to the files in the repository will be applied to this new branch.

A word of caution. When you return to the repository or click the top level repository link, notice that GitHub automatically assumes you want to see the items on the master branch. If you want to continue working on your branch, you will need to reselect it using the branch dropdown.

Adding a New File on GitHub

Now that we have a branch, let's add a new file to our class repository using GitHub.com.

Activity: Creating a New File on GitHub.com

Activity Instructions

1. Make sure you are on your branch using the branch dropdown on the upper left side
2. Click **New File**.
3. Create a file named after your home town with the `.md` extension
4. Add the file to a directory named after your GitHub username
5. Enter a short and descriptive commit message
6. Select **commit directly to the `<your-new-branch>` branch**
7. Click **Commit New File**

Collaborating on Your Code

Now that you have made some changes on your branch, let's create a Pull Request.

Activity: Creating a Pull Request

Pull Requests are used to propose changes to the project files. A pull request introduces an action that addresses an Issue. A Pull Request is considered a "work in progress" until it is merged into the project.

Now that you have created a file, you will open a pull request to discuss the file with your team mates. Follow these steps to create a Pull Request in the class repository:

Activity Instructions

1. Click the **Pull Request** icon.
2. Click **New Pull Request**.
3. In the **base** dropdown, choose `master`
4. In the **compare** dropdown, choose your branch.
5. Type a subject line and enter a comment.
6. Use markdown formatting to add a header and a checklist to your Pull Request.
7. @mention the teacher.
8. Use one of the keywords `closes`, `fixes`, or `resolves` to note which Issue the Pull Request resolves.
9. Use **Preview** to see how your Pull Request will look.
10. Assign the Pull Request to yourself.
11. Click **Create Pull Request**.

When you navigate to the class repository, you should see a banner at the top of the page indicating you have recently pushed branches, along with a button that reads **Compare & pull request**. This helpful button will automatically start the pull request process between your branch and the repository's default branch.

Exploring a Pull Request

Now that we have created a Pull Request, let's explore a few of the features that make Pull Requests the center of collaboration:

Conversation view

: Similar to the discussion thread on an Issue, a Pull Request contains a discussion about the changes being made to the repository. This discussion is found in the Conversation tab and also includes a record of all of the commit that have been made on the branch.

Commits view

: The commits view contains information about who has made changes to the files. Each commit is an updated view of the repository, allowing us to see how changes have happened from commit to commit.

Files changed view

: The Files changed view allows you to see the change that is being proposed. We call this the `diff`. Notice that some of the text is highlighted in red. This is what has been removed. The green text is what has been added.

Code Review in Pull Requests

To provide feedback on proposed changes, GitHub offers two levels of commenting:

General Conversation

: You can provide general comments on the Pull Request within the **Conversation** tab.

Line Comments

In the files changed view, you can hover over a line to see a blue + icon. Clicking this icon will allow you to enter a comment on a specific line. These line level comments are a great way to give additional context on recommended changes. They will also be displayed in the conversation view.

Exploring

Here are some additional resources you may find helpful:

- **guides.github.com/features/mastering-markdown/** An interactive guide on using Markdown on GitHub.

Editing Files on GitHub

Since you created the pull request, you will be notified when someone adds a comment. In this case, the comment may ask you to make a change to the file you just created. Let's see how GitHub makes this easy.

Editing a File on GitHub

To edit a pull request file, you will need to access the **Files Changed** view.

Click the `edit` icon to access the GitHub file editor.

Committing Changes on GitHub

Once you have made some changes to your file, you will need to create a new commit.

Remember, a good commit message should describe what was changed in the present tense. For example, Add favorite color.

Since we accessed our file through the pull request, GitHub helpfully directs us to commit our changes to the same branch. Choose the option to **Commit directly to your branch** and click **Commit changes**.

If you want to see what was changed in a specific commit, you can go to the Commits tab and click on the Commit ID you would like to view.

Merging Pull Requests

Now that you have made the requested changes, your Pull Request should be ready to merge.

Merge Explained

When you merge your branch, you are taking the content and history from your feature branch and adding it to the content and history of the `master` branch.

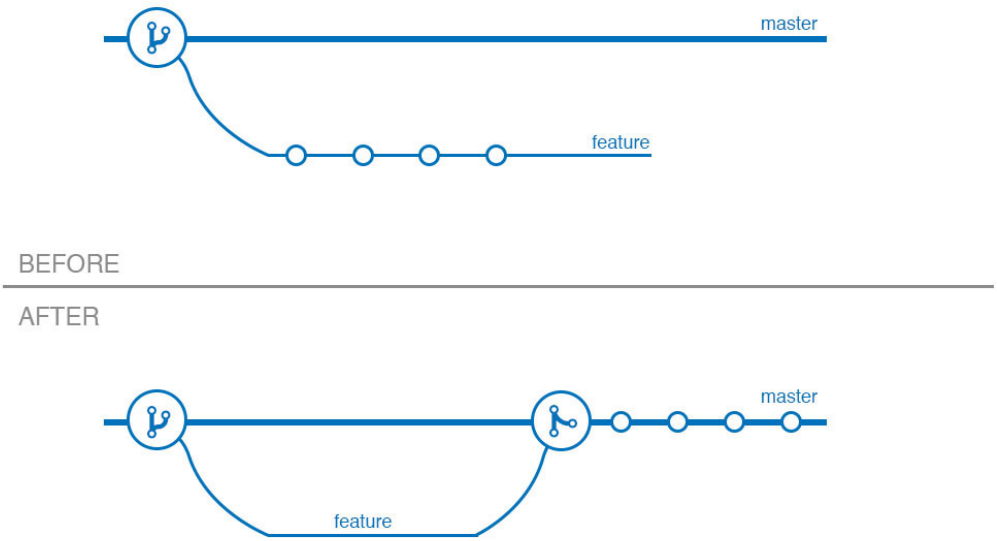

Figure 7. Merge visual.

Many project teams have established rules about who should merge a pull request. Some say it should be the person who created the pull request since they will be the ones to deal with any issues resulting from the merge. Others say it should be a single person within the project team to ensure consistency. Still others say it can be anyone other than the person who created the pull request. This is a discussion you should have with the other members of your team.

Merging Your Pull Request

Let's take a look at how you can merge the pull request.

First, let's use some emoji to indicate we have checked over the Pull Request and it looks good to merge. We like to use :+1: :ship: or :shipit:.

Activity Instructions

1. Open the Pull Request to be merged
2. Open the **Conversation** view
3. Scroll to the bottom of the Pull Request and click the **Merge pull request** button
4. Type your merge commit message
5. Click **Confirm merge**
6. Click **Delete branch**
7. Click **Issues** and confirm your original issue has been closed

Appendix A: Talking About Workflows

Discussion Guide: Team Workflows

Here are some topics you will want to discuss with your team as you establish your ideal process:

1. Which branching strategy will we use?
2. Which branch will serve as our "master" or deployed code?
3. Will we use naming conventions for our branches?
4. How will we use labels and assignees?
5. Will we use milestones?
6. Will we have required elements of Issues or Pull Requests (e.g. shipping checklists)?
7. How will we indicate sign-off on Pull Requests?
8. Who will merge pull requests?