# The Official GitHub Training Manual

# Table of Contents

## Getting Started



## GitHub Flow



## Pipeline

# Setup CI & CD



# End to End Test



# Connecting the Dots



# Real World Work



# Appendix

# Welcome to GitHub

Today you will embark on an exciting new adventure: learning how to use Git and GitHub.

As we move through today's materials, please keep in mind: this class is for you! Be sure to follow along, try the activities, and ask lots of questions!

## License

# Goals for this course

During our time together, we are going to discuss common workflows on GitHub with hands on walkthroughs of industry software development pipelines from start to finish. GitHub is the platform which connects all of the important tools in your software development lifecycle. We will:

- Discuss and dissect different workflows
- Explore software development past version control
- Hands on activities and Workflows
- Create a full pipeline, and design future pipelines
- Create, discuss, and problem solve with different workflow challenges
- Practice mapping workflows

By the end of this course, you'll see that GitHub goes far beyond version control.

# Prerequisites

This one-day class leads participants through common situations with GitHub, graphical clients, and file versioning. Participants will learn how to use version control without touching the command line and discover solutions in working with a variety workflows, tools, and user perspectives.

Be sure to follow the prerequisite(s) below before coming to the workshop to get started.

- 👉 **GitHub.com Account.** If you don't already have an account with GitHub, sign up and create a new account at https://github.com/join before the workshop.
- 👉 **Heroku Account.** If you don't already have an account with Heroku, sign up and create a new account at https://signup.heroku.com/login before the workshop.
- 👉 **CircleCI Account.** If you don't already have an account with CircleCI, sign up and create a new account at https://circleci.com/ before the workshop.
- 👉 **Internet Access**.
- 👉 Your favorite browser, but it has to be one of the following: Chrome, Firefox, Safari, Microsoft Edge or Internet Explorer 11.
- 👉 Download and install GitHub Desktop.
- 👉 Log in to both **GitHub** and **Heroku** just before the workshop begins.

# The Modern Software Organization

Software has become a critical component of every business. Software is embedded in the product, responsible for tasks like running the machines that make the product, managing logistics, or actually *being* the product. Reliability and shortened release cycles are a competitive advantage in a quickly changing technological environment.

Successful development teams have had to change their paradigms for how they work over time. The complexity of software is increasing, and teams are more reliant on one another to solve problems and evaluate potential solutions. Evolving organizational structures and international geography are making it more difficult to get the right people in the room. And, Open Source is changing how we think about IP.

Whether you're aware of it or not, source code management impacts your business. How you protect your source code matters. Does it facilitate collaborative workflows, or does it make it harder for developers to work? You have an opportunity and an responsibility to provide assurances of code quality and speed up release cycles, all of which is tightly related to your source code.

## The GitHub Ecosystem

Rather than force you into a "one size fits all" ecosystem, GitHub strives to be a place of reliability and discovery, bringing all of your favorite tools together alongside exciting new prospects. For more information on integrations, check out https://github.com/integrations. Find even more tools at https://github.com/works-with.

# Join our Class Repository

For the next exercises, navigate to the class repository. To be added as a **collaborator**, please find issue #1 and leave a comment.

---

## In the meantime, while you're navigating...

What's your name?

What team do you work on?

What types of software systems are you managing (e.g. embedded software, business systems, developer operations, SaaS, etc)?

How do you use GitHub now?

How do you want to improve on that by the time this course is over?

# Git and GitHub

We will start by introducing you to Git, GitHub, and the collaboration features you will use to get things done. Even if you have used GitHub in the past, this information will provide a better understanding of how to implement Git and GitHub within your organization or team to build better software together.

## What is GitHub?

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

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
- Projects
- Organizations and Teams

## 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 features 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!

## User Accounts vs. Organization Accounts

There are two account types in GitHub, user accounts and organization accounts. While there are many differences in these account types, one of the more notable differences is how you handle permissions.

## User Accounts

When you signed up for GitHub, you were automatically given a user account. Permissions for a user account are simple, you add people as collaborators to specific repositories to give them full read-write access to the project.

## Organization Accounts

Organization accounts provide more granular control over repository permissions. In an organization account you create teams of people and then give those teams access to specific repositories. Permissions can be assigned at the team level (e.g, read, write, or admin).

## 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 help you write better software by facilitating code review and showing the status of any automated tests.

## Projects

Projects allow you to visualize your work with Kanban style boards. Projects can be created at the repository or organization level.

## 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 & Graphs

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

## ISSUE_TEMPLATE.md and PULL_REQUEST_TEMPLATE

The ISSUE_TEMPLATE.md and the PULL_REQUEST_TEMPLATE.md are used to generate templated starter text for your project issues and pull requests. Any time someone opens an issue or pull request, the content in the template will be pre-populated in the body of the issue or pull request.

## Using GitHub Issues

In GitHub, you will use issues to record and discuss ideas, enhancements, tasks, and bugs. Issues make collaboration easier by:

- Replacing email for project discussions, ensuring everyone on the team has the complete story, both now and in the future.
- Allowing you to cross-link to related 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 with @ mentions and team mentions.

## Using Markdown

GitHub uses a syntax called **Markdown** to help you add basic text formatting to issues, pull requests, and files with the `.md` extension.

## Commonly Used Markdown Syntax

```
# Header
```

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

```
* List item
```

A single `*` or `-` followed by a space will create a bulleted list.

```
**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 `:`.

# Introduction to GitHub Pages

GitHub Pages enable you to host free, static web pages directly from your GitHub repositories. Several of the projects we use in class will use GitHub Pages as the deployment strategy. We will barely scratch the surface in this class, but there are a few things you need to know:

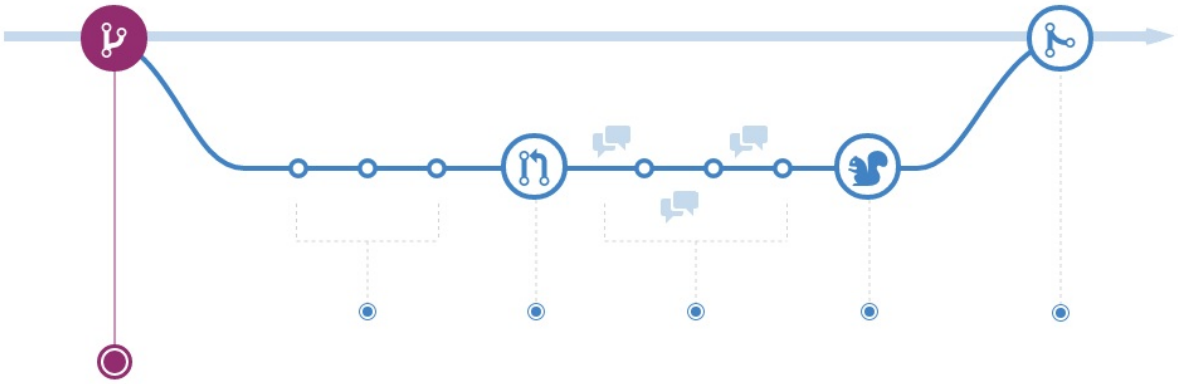
- You can create two types of websites, a user/organization site or a project site. We will be working with project websites.
- For a project site, GitHub will only serve the content on a specific branch. Depending on the settings for your repository, GitHub can serve your site from a `master` or `gh-pages` branch, or a `/docs` folder on the `master` branch.

- The rendered sites for our projects will appear at `githubschool.github.io/repo-name`.

# Understanding the GitHub flow

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

## The Essential 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 default, the "blessed" or "canonical" version of your project lives on a branch called `master`. This branch can actually be named anything, as we will see in a few minutes.

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.

## Exploring

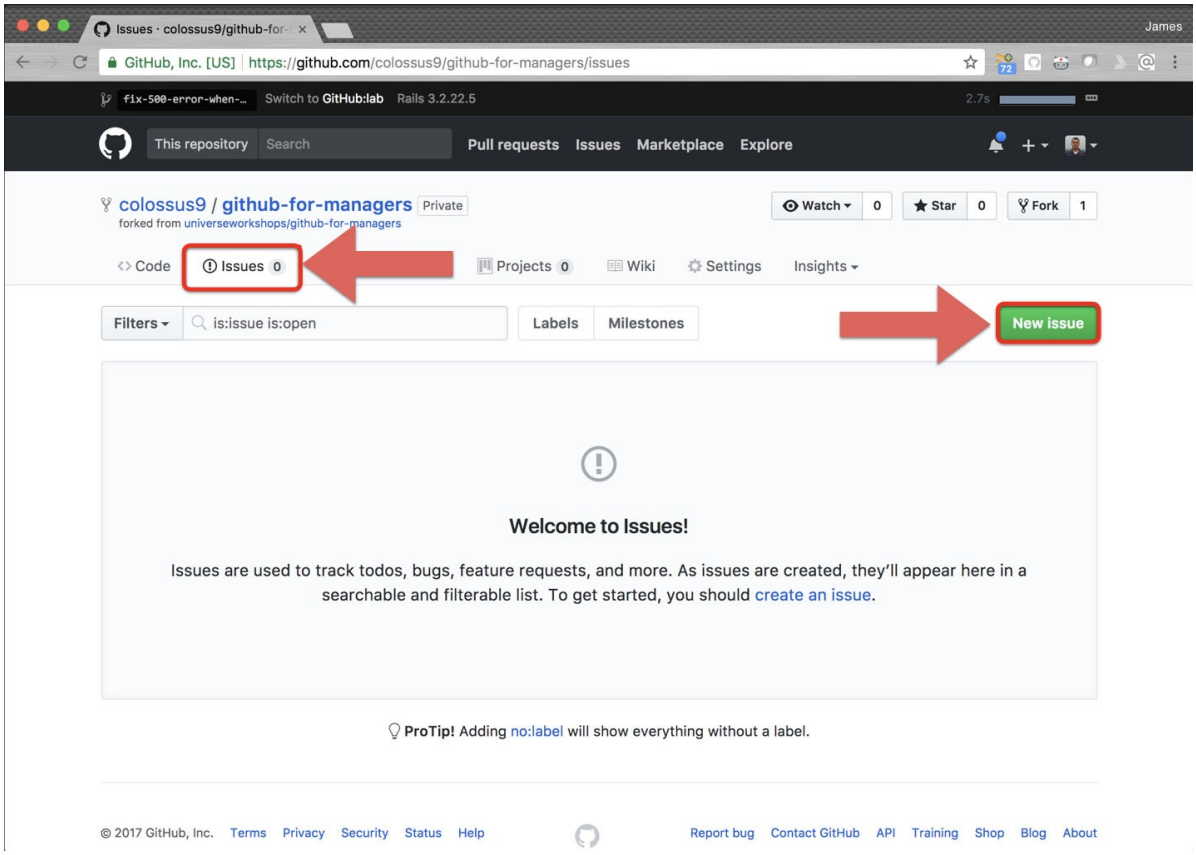
Here are some interesting things you can check out later:

- *guides.github.com/introduction/flow/* An interactive review of the GitHub Workflow.

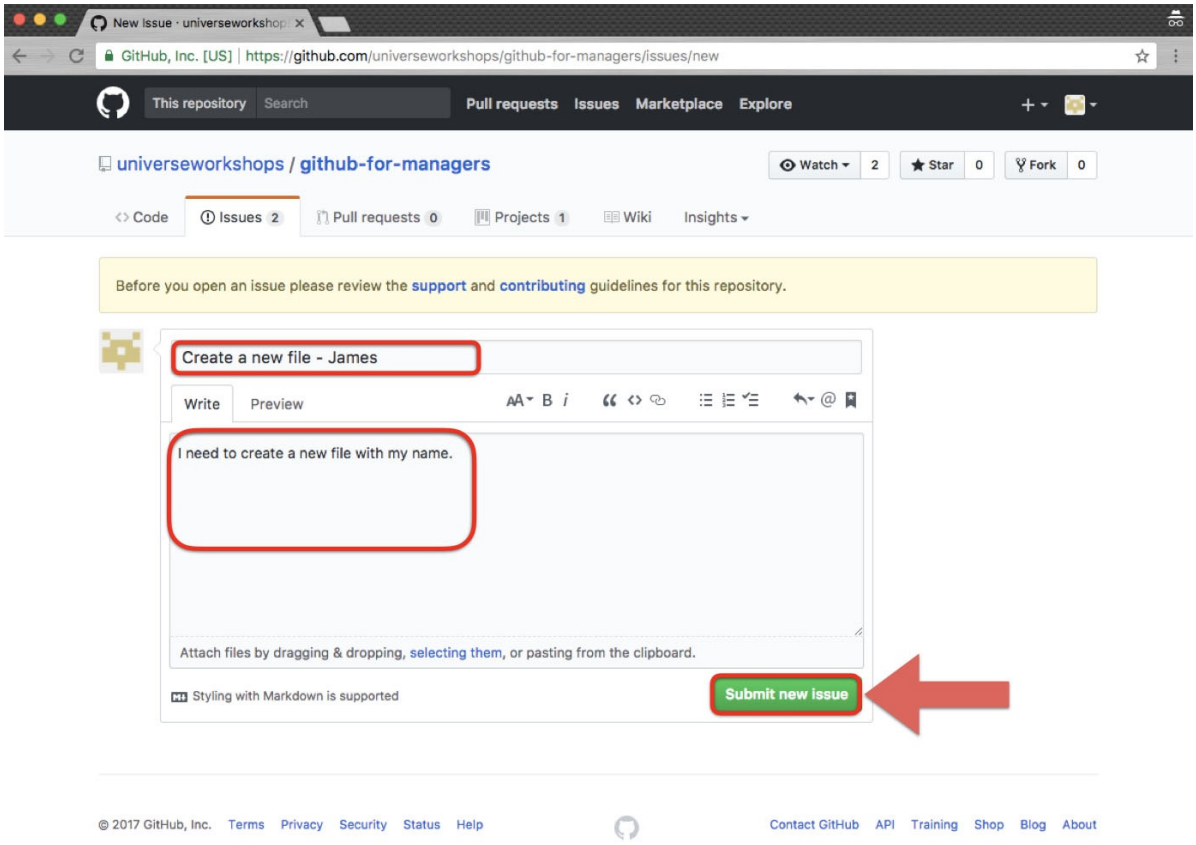
# Create an Issue

Let's create a tracking Issue to record our first item of work, which will be to create a file with your name.

1. In the repository's **Issues** tab, click **New issue**.



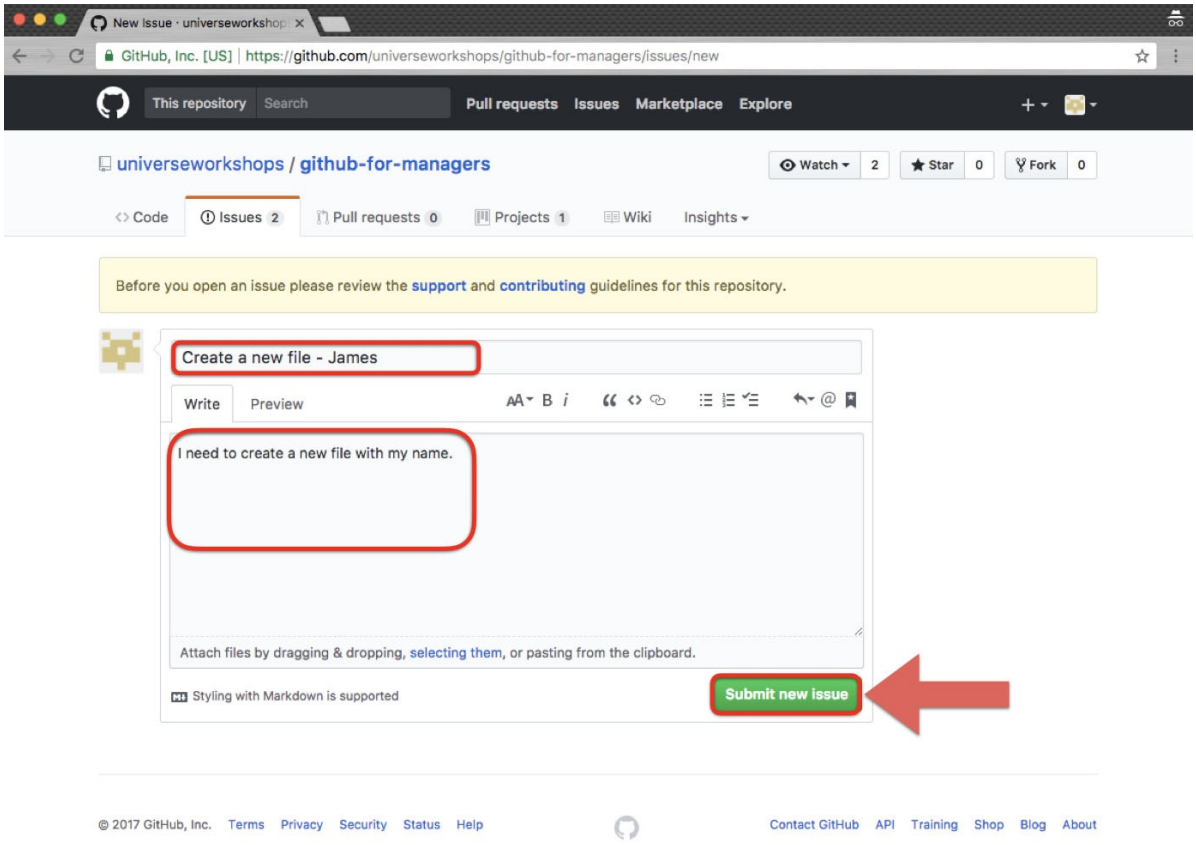

2. Enter a **Title** indicating you will create a file with your name on it.




. Example:

```
Create a new file - John
```

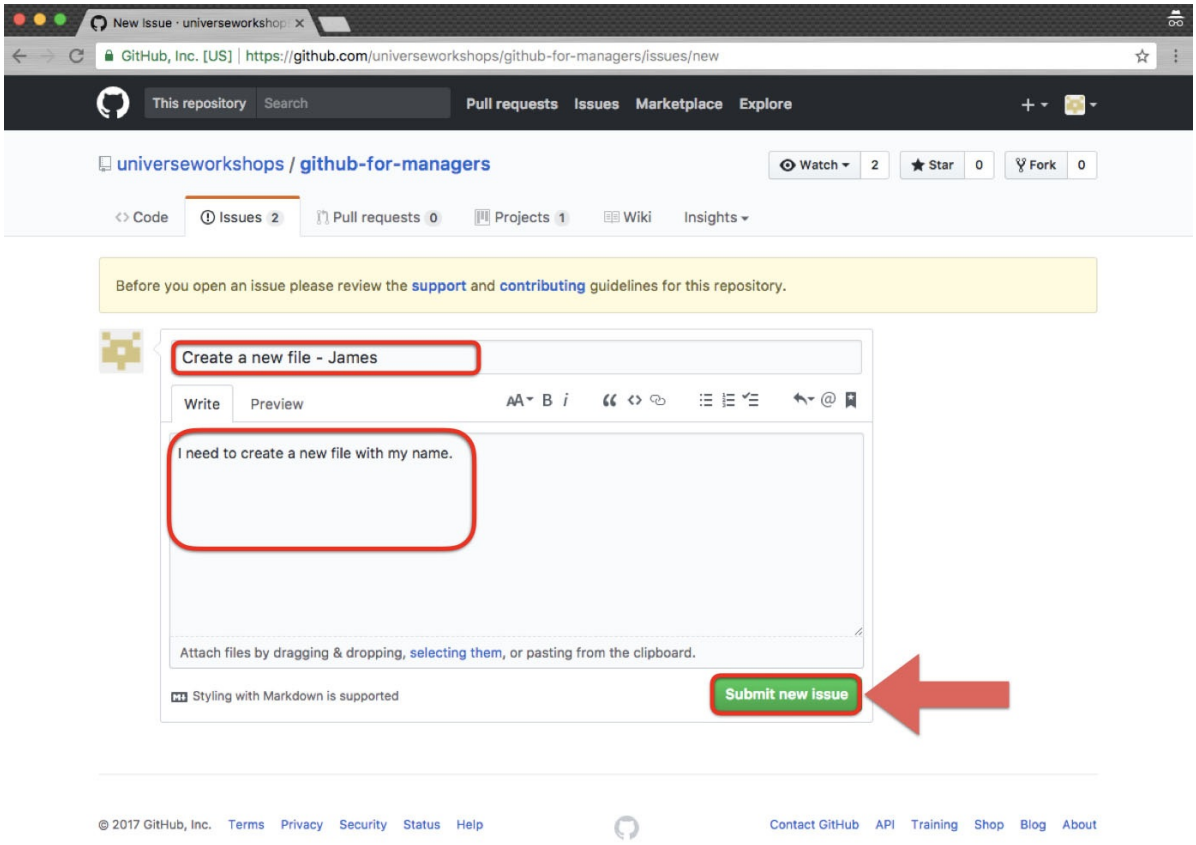
3. In the **Body** of the Issue, let's add a *Description* similar to the following




:

```
I need to create a new file with my name.
```

4. Click **Submit new issue**.


New Issue · universeworkshop
GitHub, Inc. [US] | https://github.com/universeworkshops/github-for-managers/issues/new
This repository
Search
Pull requests
Issues
Marketplace
Explore
universeworkshops / github-for-managers
Watch 2
Star 0
Fork 0
Code
Issues 2
Pull requests 0
Projects 1
Wiki
Insights
Before you open an issue please review the support and contributing guidelines for this repository.
Create a new file - James
Write
Preview
I need to create a new file with my name.
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.
Styling with Markdown is supported
Submit new issue
© 2017 GitHub, Inc.
Terms
Privacy
Security
Status
Help
Contact GitHub
API
Training
Shop
Blog
About

# GitHub Project Boards

Project boards are Kanban style organization tools.

Project boards are one tool, but you could also use many other options to fill this project management role.

1. Navigate to the repository's **Project** tab.
2. Add your issue as a card to the "In Progress" column.

We will come back to the project board throughout the process to track our changes.

# Activity: Creating A Branch with GitHub

Earlier you created an issue about the file you would like to edit. Let's create the branch you will use to edit your file.

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

> You will need to have collaborator access on the class repository before you can create a branch on GitHub.

1. Navigate to *Code* tab of the class repository.
2. Click the *branch dropdown*.
3. Enter the branch name 'github-username-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 default branch. If you want to continue working on a feature branch, you will need to reselect it using the branch dropdown.

# Create a File

1. In the repository's **Code** tab, navigate to the `hometown` folder.
2. While in the `hometown` folder, click **Create new file**.
3. On the **Create new file** screen, be sure to provide a new filename.
4. In the **Edit new file** box, type some info about your hometown, for example:

   ```
   The best restaurant in Olathe, KS is Joe's Kansas City BBQ.
   ```

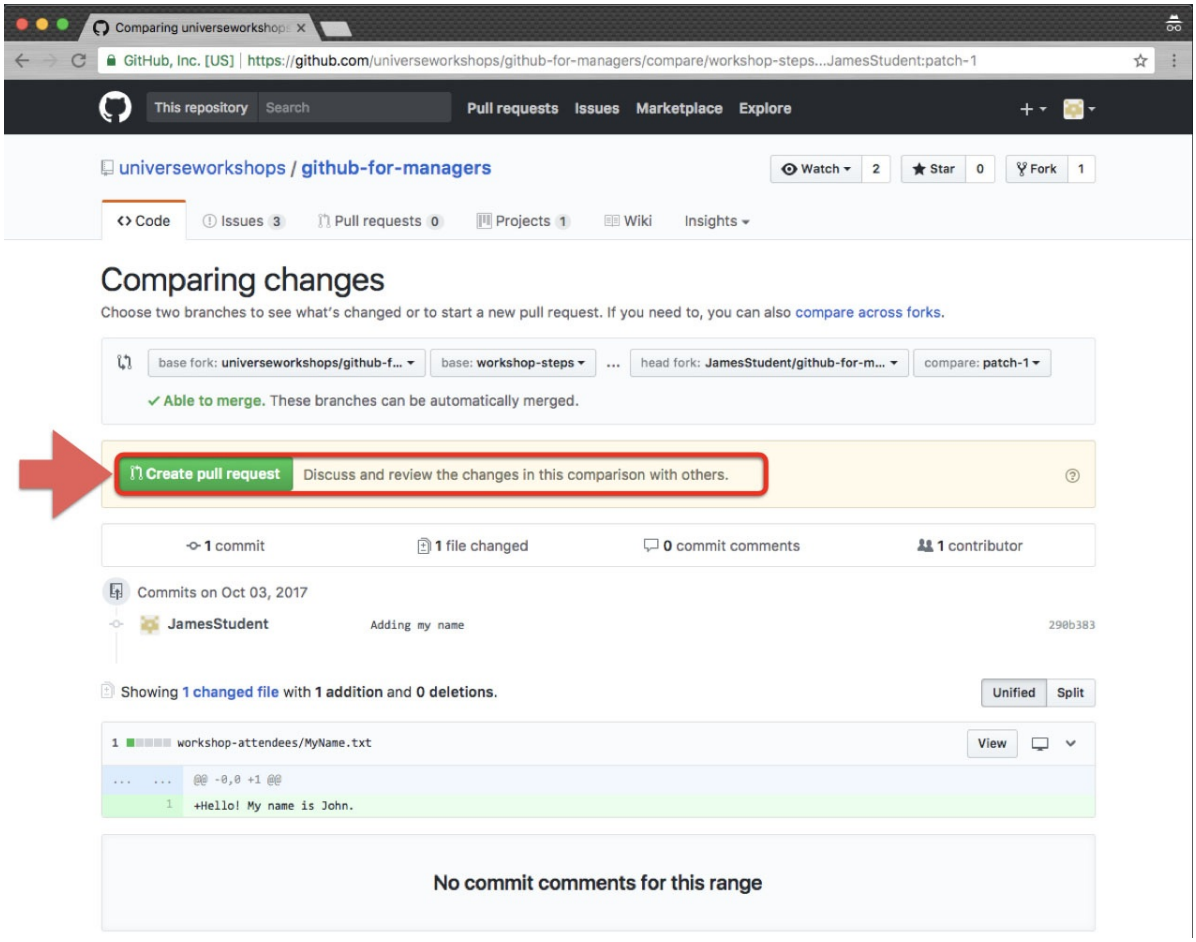
5. Scroll down to the **Propose new file** section and provide a commit message. Example:

   ```
   Adding my file
   ```

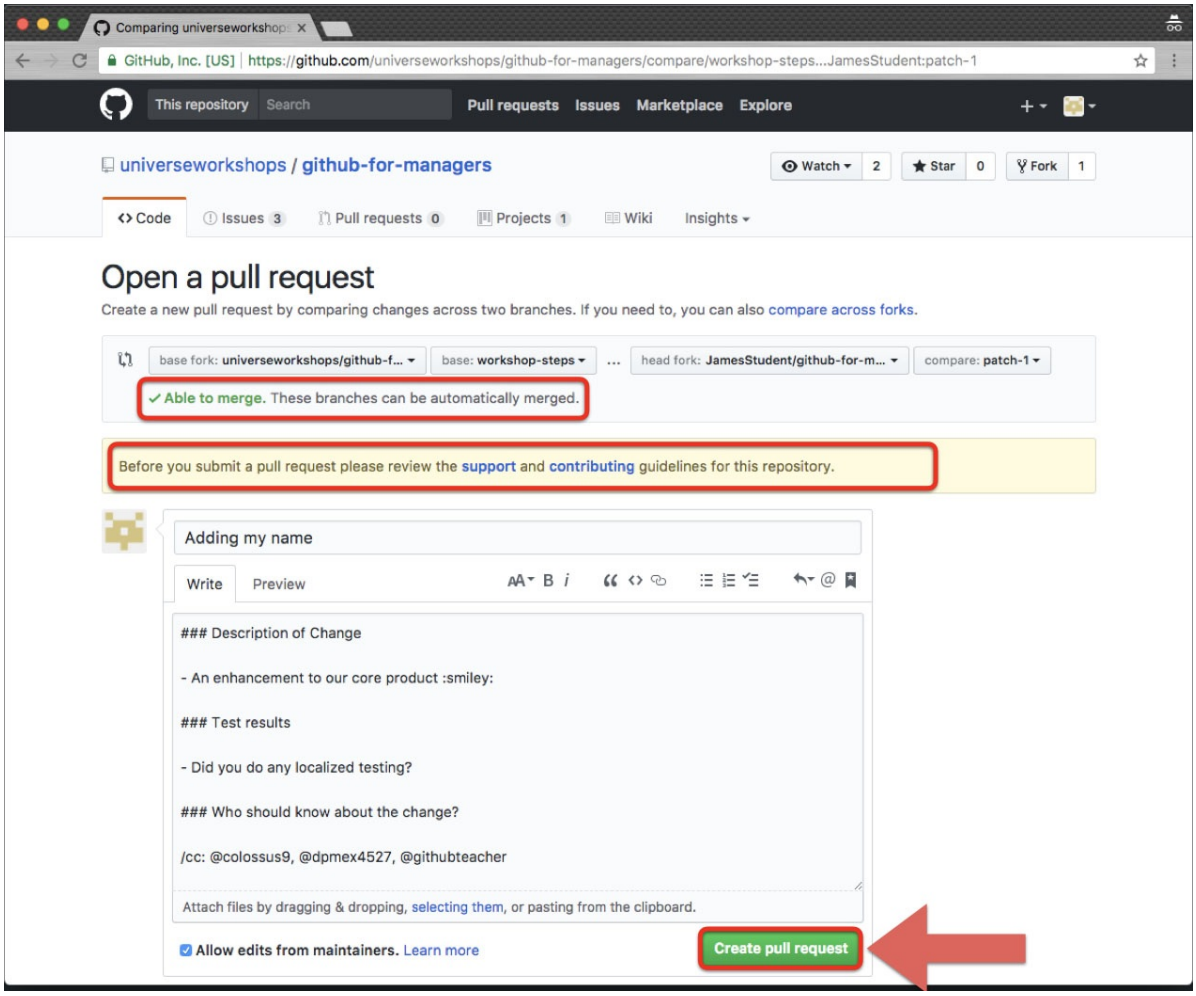
6. Click **Propose new file**.

# Open a Pull Request

1. The next screen will show the creation of the Pull Request, with some info about the specifics of your proposed change. Click **Create pull request**. A Pull Request will be opened for this proposed change.

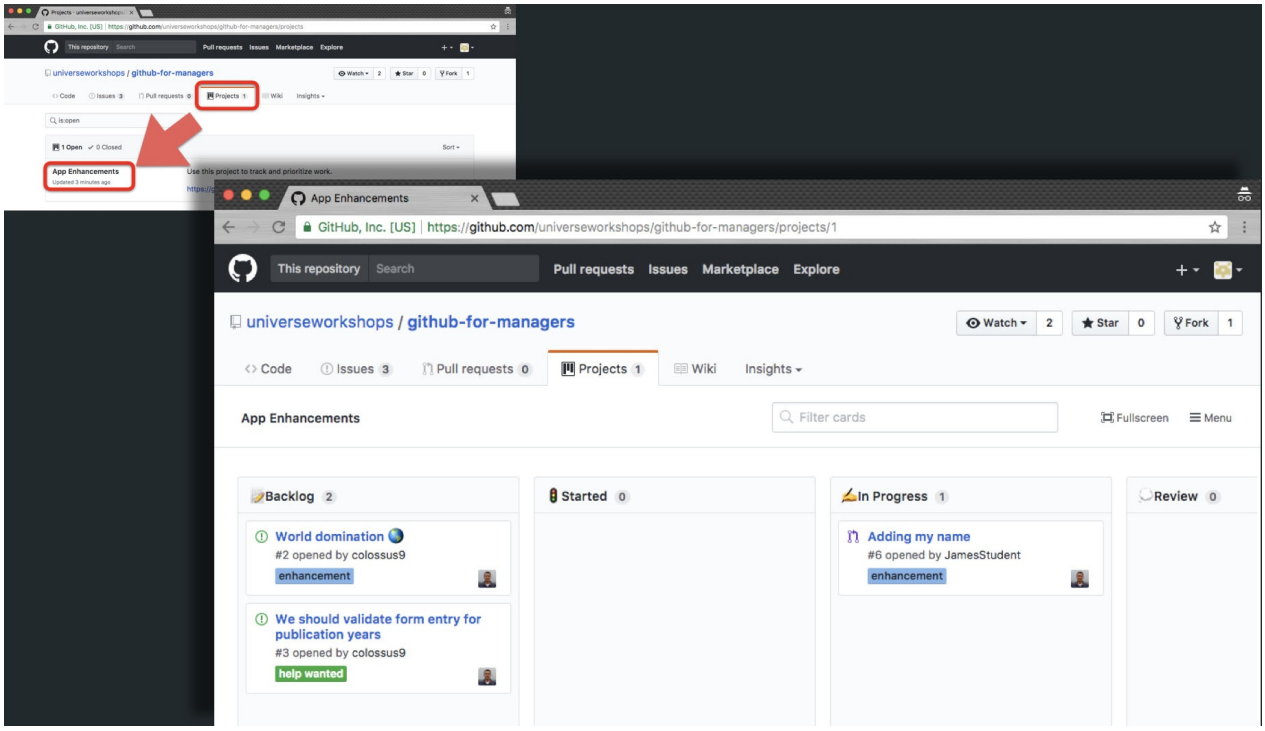



2. You may be presented with a body of text where you can add and remove as you see necessary. This is where you tell the repository owner a bit more about your change. If asked, click **Create pull request**. Here, make sure to write "closes issue" and include your issue number.




There are many important traceability and tracking aspects with Issues and Pull Requests, including:

```
- Linking to Issues
- Code Review
- Assignment
- Label(s)
- Milestone
- Project Visualization
```




1. Add your pull request to the **Recommendations** Project board.
2. Navigate back to the Project board, and add your pull request from triaging to **In Progress**.

# Collaborating on Pull Requests

## 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 commits made on the branch as well as assignments, labels and reviews that have been applied to the pull request.

## Commits view

The commits view contains more detailed information about who has made changes to the files. Clicking each commit ID will allow you to see the changes applied in that specific commit.

## Files changed view

The Files changed view allows you to see cumulative effect of all the changes made on the branch. We call this the `diff`. Our diff isn't very interesting yet, but as we make changes your diff will become very colorful.

## Code Review in Pull Requests

To provide feedback on proposed changes, GitHub offers three 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.

## Review

When you are making line comments, you can also choose to *Start a Review*. When you create a review, you can group many line comments together with a general message: Comments, Approve, or Request Changes. Reviews have special power in GitHub when used in conjunction with protected branches.

## Activity: Code Review

One of the best ways to ensure code quality is to make peer reviews a part of every Pull Request. Let's review your partner's code now:

1. Click the *Pull Request* tab.
2. Use the *Author* drop down to locate your partner's pull request.
3. Click the *Files Changed* tab.
4. Hover over a single line in the file to see the blue +. Click the + to add a line comment.
5. Comment on the line and click *Start review*.
6. Repeat these steps to add 2-3 comments on the file.
7. Click *Review* in the top right corner.
8. Choose whether to *Approve* or *Request changes*
9. Enter a general comment for the review.
10. Click *Submit review*
11. Click the *Conversation* view to check out your completed review.

# Key Collaboration Questions

- Is software essential to your business? Why?
- As software has proliferated every area of the business, how has that changed the way you must work as a development team?
- Given the importance of software to your success, how does source code management impact your business?

# 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 `gh-pages` branch.

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 to ensure at least one review has taken place.

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.

1. Navigate to your Pull Request (HINT: Use the Author or Assignee drop downs to find your Pull Request quickly)
2. Click *Conversation*
3. Scroll to the bottom of the Pull Request and click the *Merge pull request* button
4. Click *Confirm merge*
5. Click *Delete branch*
6. Click *Issues* and confirm your original issue has been closed.
7. Navigate to the Project board and confirm that your pull request and issue have been moved to the *Done* column.

> GitHub offers three different merge strategies for Pull Requests:

> - **Create a merge commit:** This is the traditional option that will perform a standard recursive merge. A new commit will be added that shows the point when the two branches were merged together.
> - **Squash and merge:** This option will take all of the commits on your branch and compress them into a single commit. The commit messages will be preserved in the extended commit message for the commit, but the individual commits will be lost.
> - **Rebase and merge:** This option will take all of the commits and replay them as if they just happened. This allows GitHub to perform a fast forward merge (and avoids the addition of the merge commit).

# Software Development Pipeline

GitHub concepts are well documented features, but it is important to understand how developers do their work. Even new users to GitHub can set up a CI/CD development pipeline with little effort.

A pipeline is a concept of how your tools fit together, so the development process is seamless. The goals are to:

- Provide quick feedback to developers
- Automate as much as possible
- Deploy with higher quality code

These three smaller goals all fit together to accomplish a much larger and more abstract goal: to develop faster and better.

## What tools do you use?

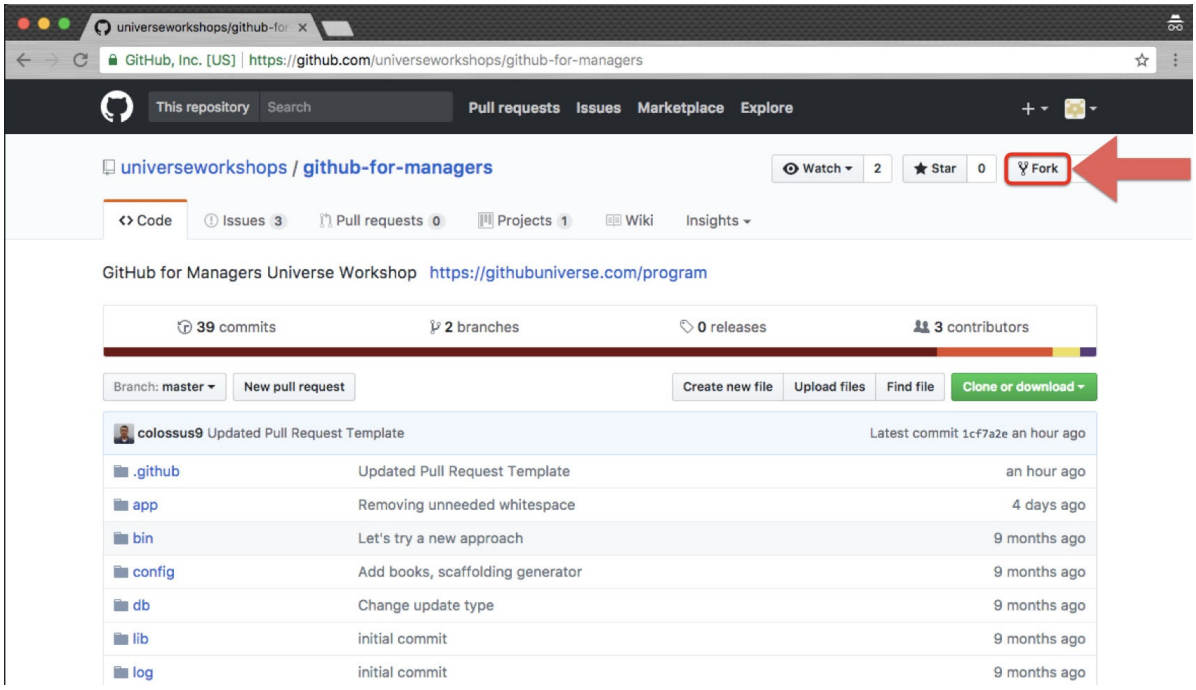
What tools are already in use in your organization? What tools have been used in the past, or may be used in the future? GitHub provides a centralized "Hub" where all of these tools can be connected so developers can see all of the important puzzle pieces connected where they're working.

# Fork the Class Repository

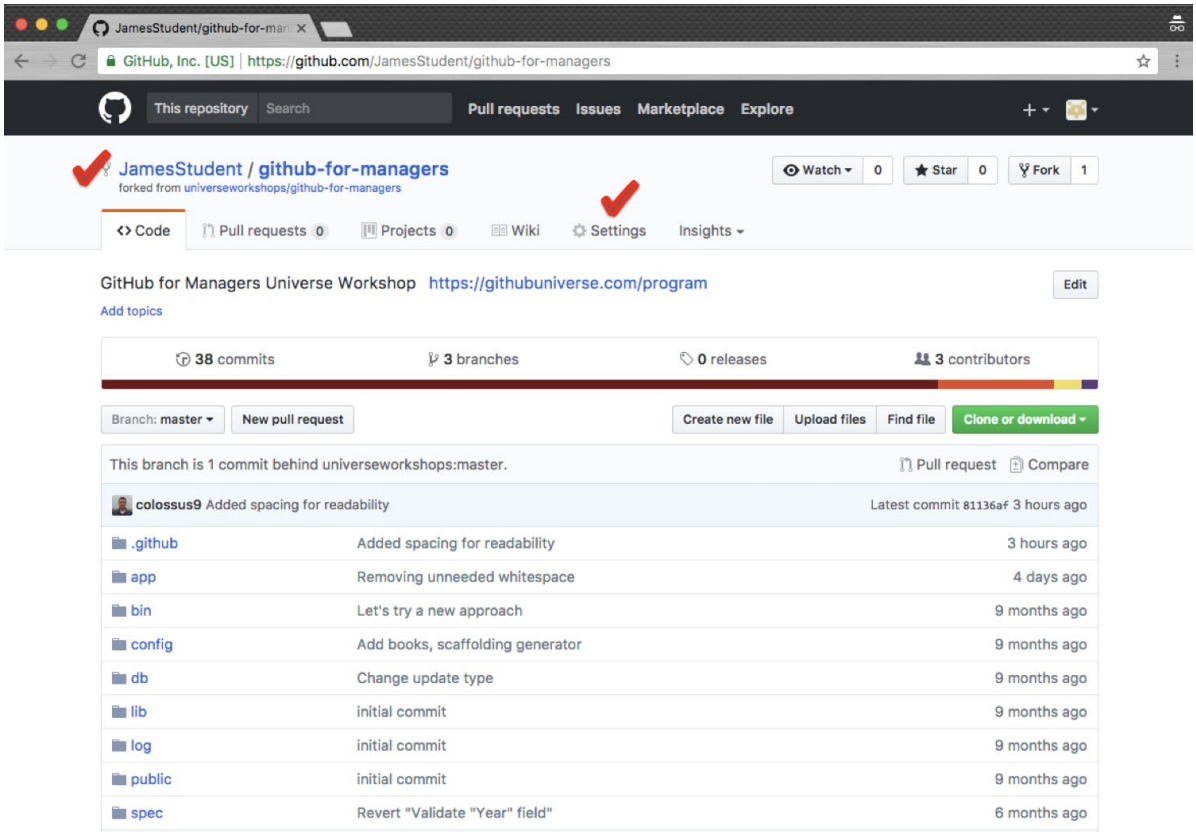
To practice creating a pipeline, let's set up our own personal workspace. To create our workspace we will fork the class repository. A **fork** is effectively a copy of a project you can use to freely experiment without affecting the original project.

Although some organizations are still using the fork method to ensure security about what gets merged into the master branch, many organizations are beginning to use "protected branches" to minimize risk while merging pull requests.

1. Navigate to the class repository..
2. In the top-right-hand corner, click **Fork**.



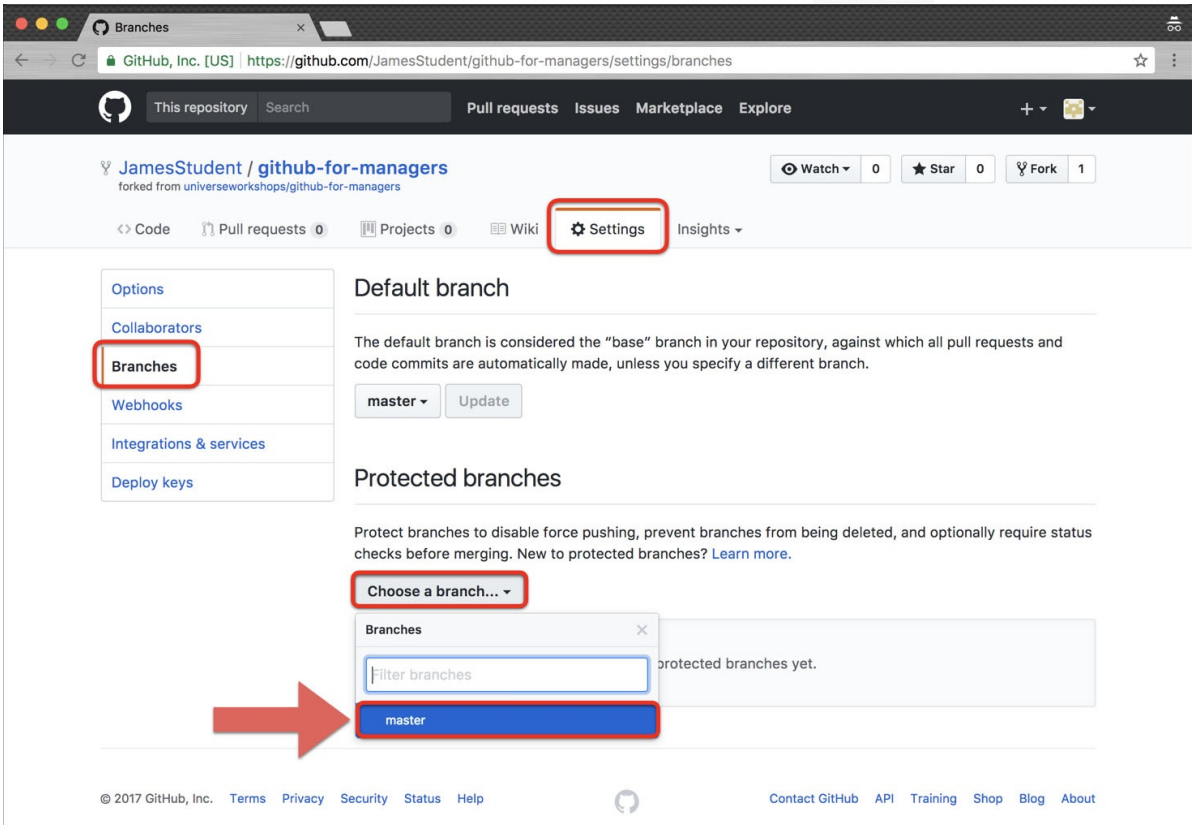

   If asked, fork to your personal GitHub account.
3. You will then be taken to your forked repository where you can experiment. Notice the `forked from...` message in the top-left, and notice you can now see a **Settings** tab because you are the admin here.


JamesStudent/github-for-man
GitHub, Inc. [US] | https://github.com/JamesStudent/github-for-managers
This repository
Search
Pull requests
Issues
Marketplace
Explore
JamesStudent / github-for-managers
forked from universeworkshops/github-for-managers
Watch 0
Star 0
Fork 1
Code
Pull requests 0
Projects 0
Wiki
Settings
Insights
GitHub for Managers Universe Workshop https://githubuniverse.com/program
Edit
Add topics
38 commits
3 branches
0 releases
3 contributors
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
This branch is 1 commit behind universeworkshops:master.
Pull request
Compare
colossus9 Added spacing for readability
Latest commit 81136af 3 hours ago
.github Added spacing for readability 3 hours ago
app Removing unneeded whitespace 4 days ago
bin Let's try a new approach 9 months ago
config Add books, scaffolding generator 9 months ago
db Change update type 9 months ago
lib initial commit 9 months ago
log initial commit 9 months ago
public initial commit 9 months ago
spec Revert "Validate "Year" field" 6 months ago

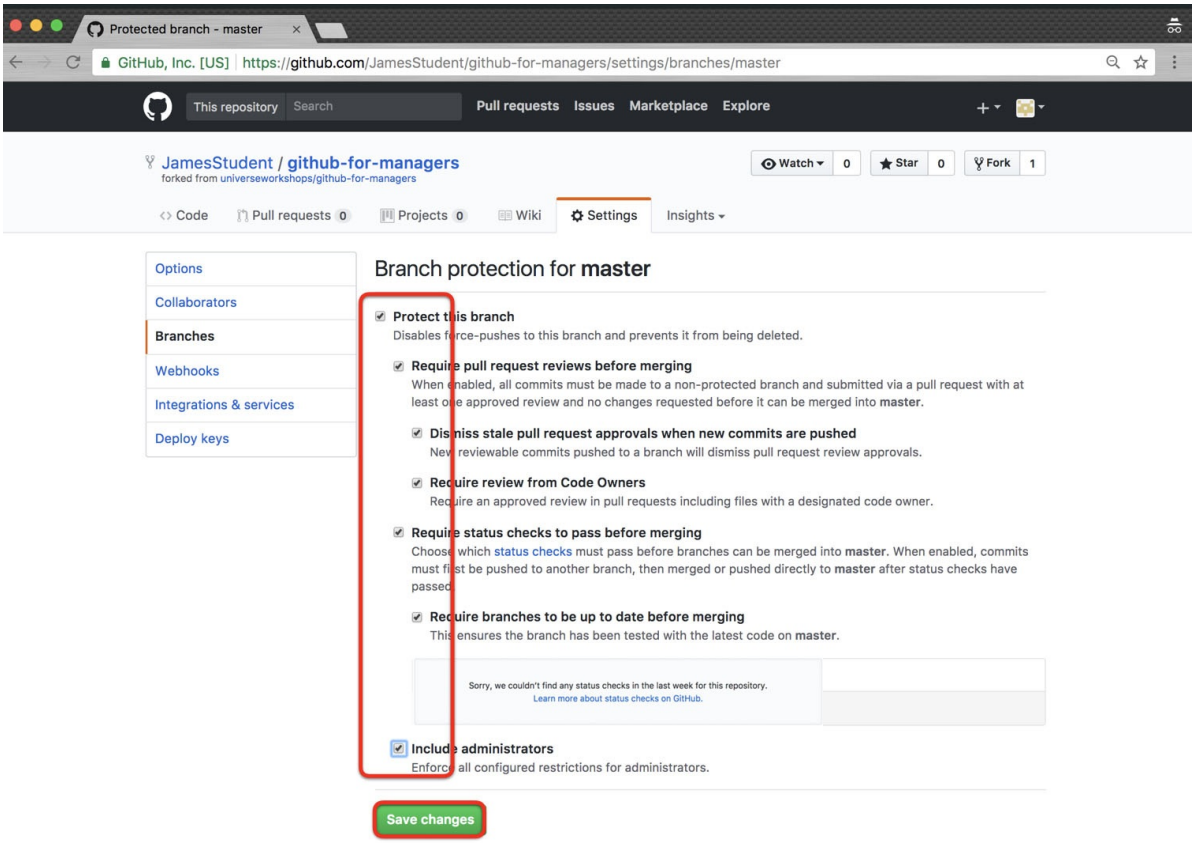
# Administration Settings

Every repository contains administrative settings for you to manage your project effectively, including (but not limited to) visibility, collaboration, and quality checking. Check out each of the settings to learn how you can fine tune your environment.

1. Let's protect the `master` branch. Click the repository's **Settings** tab, then **Branches**. Under **Protected branches** click the dropdown and select `master`.




2. In the **Branch protection for `master`** page, check all available settings.


Protected branch - master
GitHub, Inc. [US] https://github.com/JamesStudent/github-for-managers/settings/branches/master
This repository Search
Pull requests Issues Marketplace Explore
JamesStudent / github-for-managers
forked from universeworkshops/github-for-managers
Watch 0 Star 0 Fork 1
Code Pull requests 0 Projects 0 Wiki Settings Insights
Options
Collaborators
Branches
Webhooks
Integrations & services
Deploy keys
Branch protection for master
Protect this branch
Disables force-pushes to this branch and prevents it from being deleted.
Require pull request reviews before merging
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with at least one approved review and no changes requested before it can be merged into master.
Dismiss stale pull request approvals when new commits are pushed
New reviewable commits pushed to a branch will dismiss pull request review approvals.
Require review from Code Owners
Require an approved review in pull requests including files with a designated code owner.
Require status checks to pass before merging
Choose which status checks must pass before branches can be merged into master. When enabled, commits must first be pushed to another branch, then merged or pushed directly to master after status checks have passed.
Require branches to be up to date before merging
This ensures the branch has been tested with the latest code on master.
Sorry, we couldn't find any status checks in the last week for this repository.
Learn more about status checks on GitHub.
Include administrators
Enforce all configured restrictions for administrators.
Save changes

# What is CI/CD?

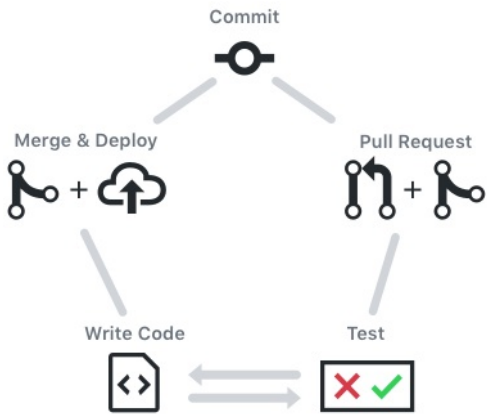
Continuous Integration and Continuous Deployment tools take manual tasks and do them automatically.

## Why run tests? Why use CI?

- Test code automatically
- Add structure to development process
- Test driven development (TDD) isn't just the addition of random tests, but rather about building tests as software requirements
- Builds are done on separate servers by the CI services
- Other benefits include consistent configurations, battery of tests, reduces "works on my machine"

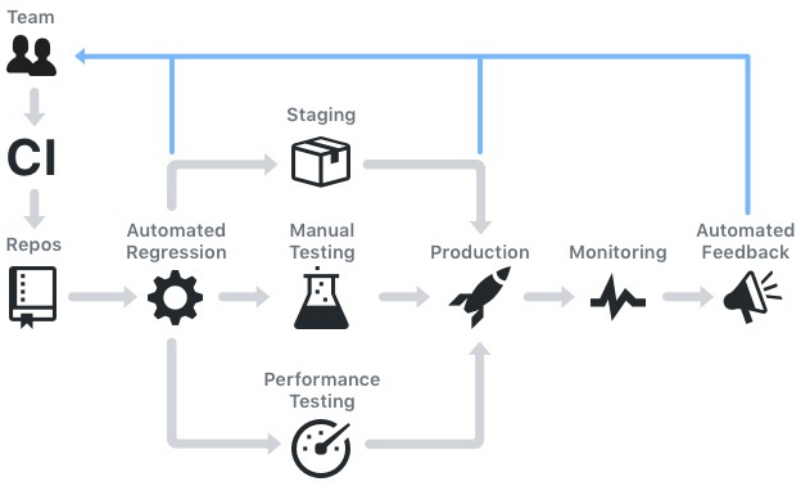
## How does CI/CD work with GitHub?

- When used with GitHub, tests can be run automatically on branches and pull requests every time there is a new commit, and return a status through GitHub's API
- Tests are run in consistent environment based on your software's production environment
- Protected Branches can serve as a gate, keeping pull requests without passing tests from being merged



# What is the difference between CI and CD?

- **CI**: Continuous integration
  - Continuous Integration is the practice of automatically kicking off tests with each push, rather than ad-hoc testing.
- **CD**: Continuous deployment
  - Code pushed to deployment automatically based on custom circumstances
  - Continuous Deployment is the practice of continuously deploying to production servers. In conjunction with CI, companies can move to a CD model by giving developers the freedom to deploy several times a day upon successful builds. Merging in and of itself isn't considered CD, unless it's tied to some deployment strategy (ex: Heroku automatically deploy anything that gets merged into master).
  - How do you deploy before merge? This can help us perform a better code review, and we can see our actual application deployment before merging to `master`.
- Though the concepts are different, CI and CD are frequently discussed and implemented together.
  - Typically CI and CD are implemented at the same time because many of the tasks are already defined. For example, a project may already have the tests written, and the steps for deployment. Once a CI/CD tool is introduced, it's simple to have them be done together.

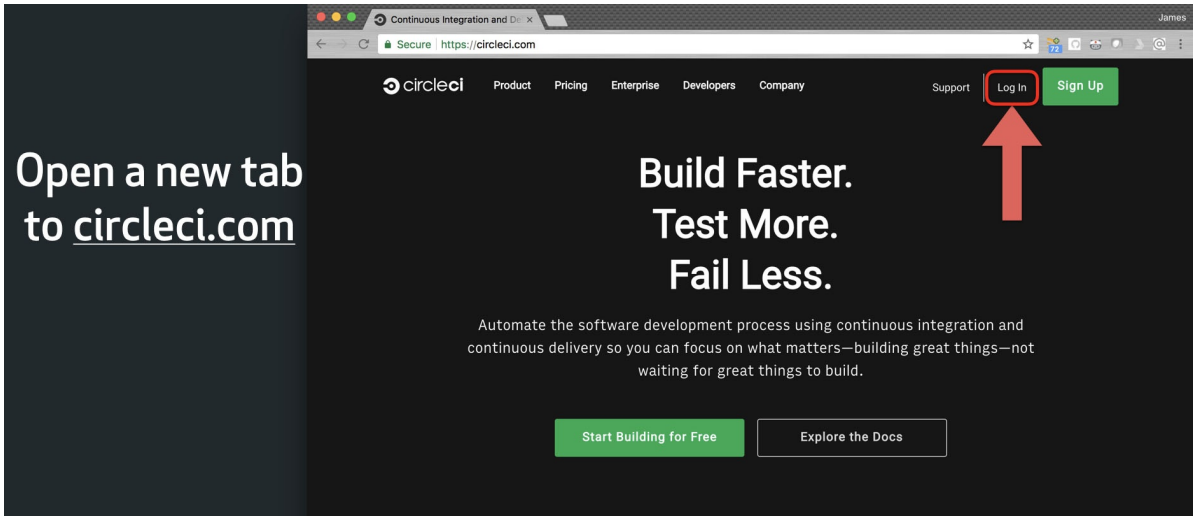


# Other Things to Think About

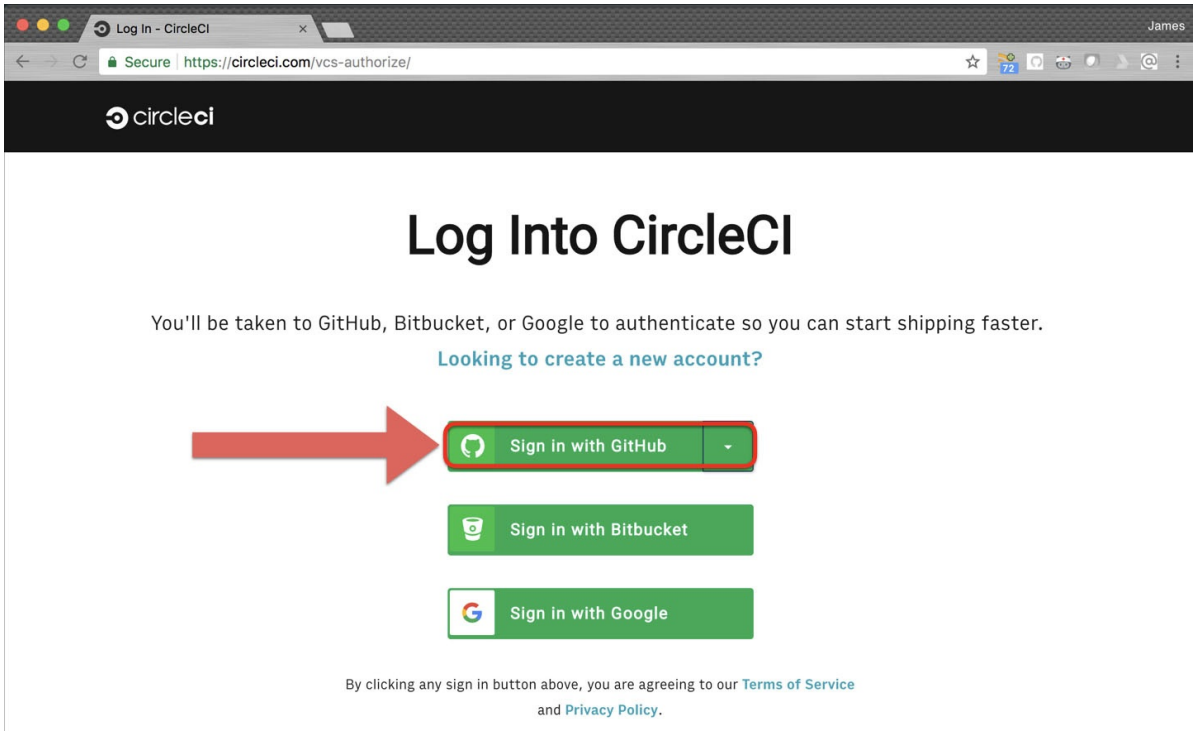
- Protected branches
  - On GitHub, you may want to set up protected branches when you set up CI/CD.
  - Protected branches block code from being merged in to the master branch on the remote before it passes a set of configurable requirements, like passing CI/CD tests and builds, or having approved reviews.
- Automatic deploys
  - With CI/CD, you can also set up deploys to happen automatically when code passes the tests on the master branch.
  - Much like many integrations work with GitHub, many deployment options work with CI/CD services.
  - One example is Heroku. You could configure a project on Heroku with a CI/CD service, and whenever a build passes on the master branch, deployment to Heroku would be streamlined and taken care of automatically.

# Configure CI

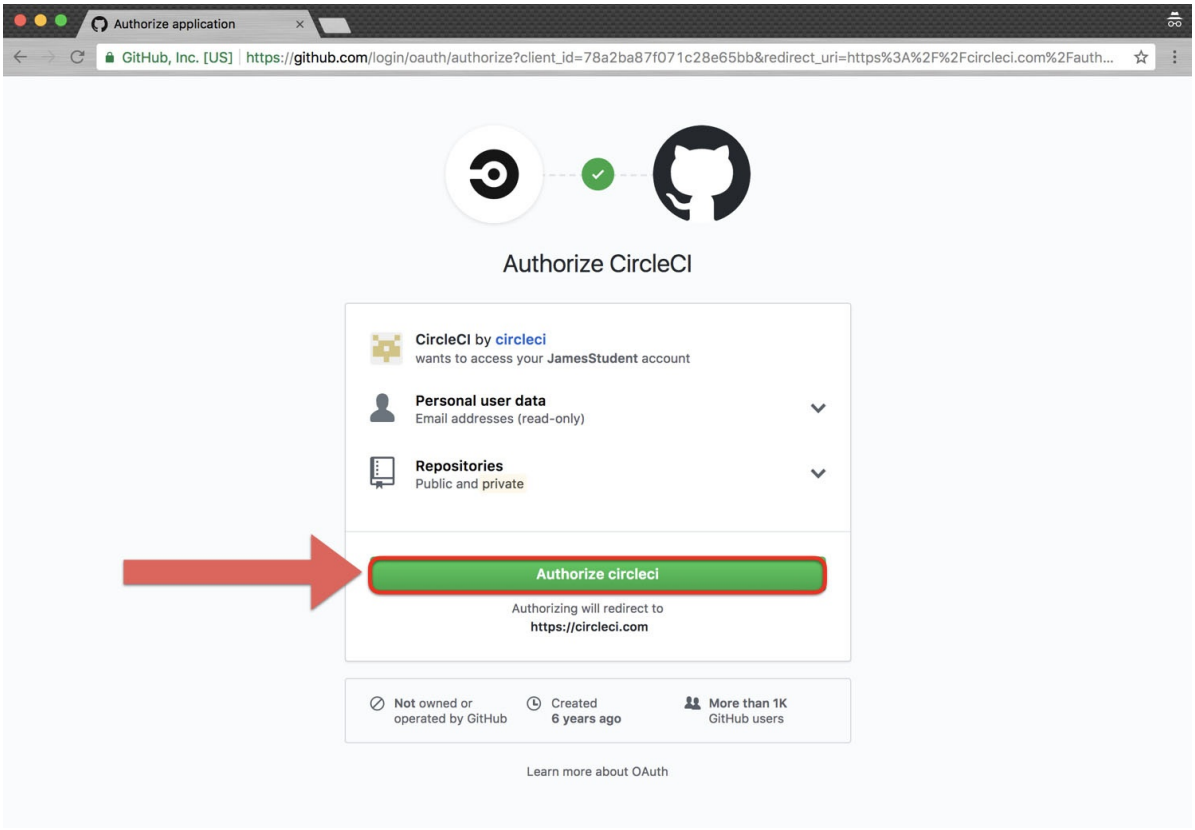
1. Open a new tab, go to https://circleci.com and click **Log In**.




2. Click **Sign in with GitHub**.

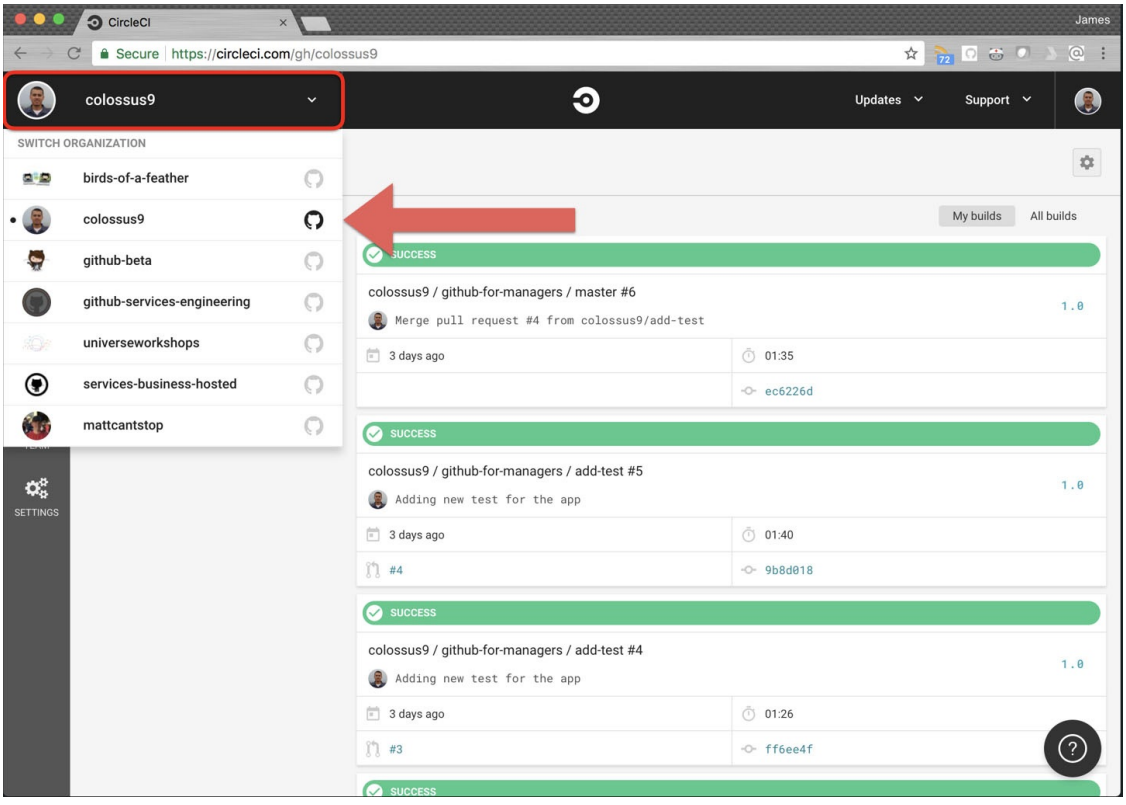



3. Click **Authorize circleci**.

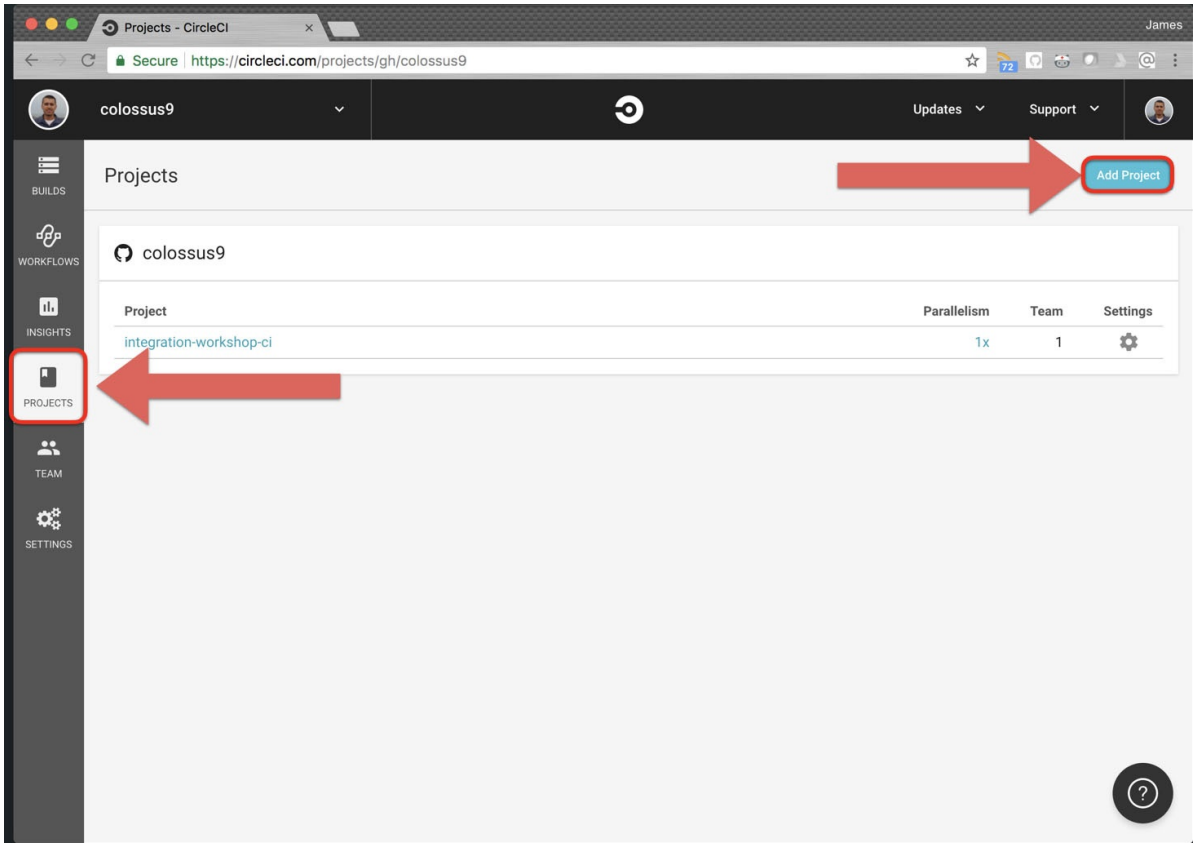



.

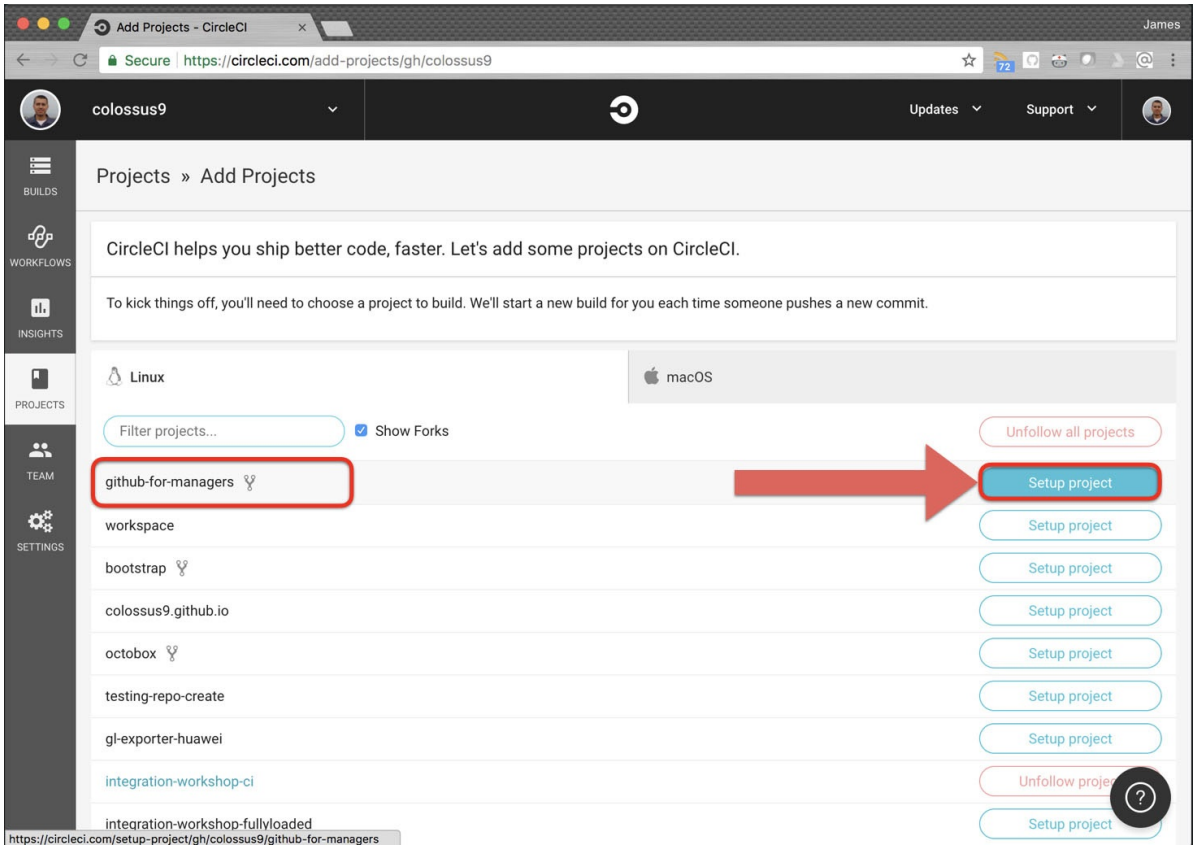
4. After authorization, you can access the CircleCI app at https://circleci.com/dashboard.
    - **Note:** If you are already a member of multiple GitHub orgs, you need to select your username in the dropdown at the top of the page.




5. On the left side, click **Projects** and then click **Add Project**.



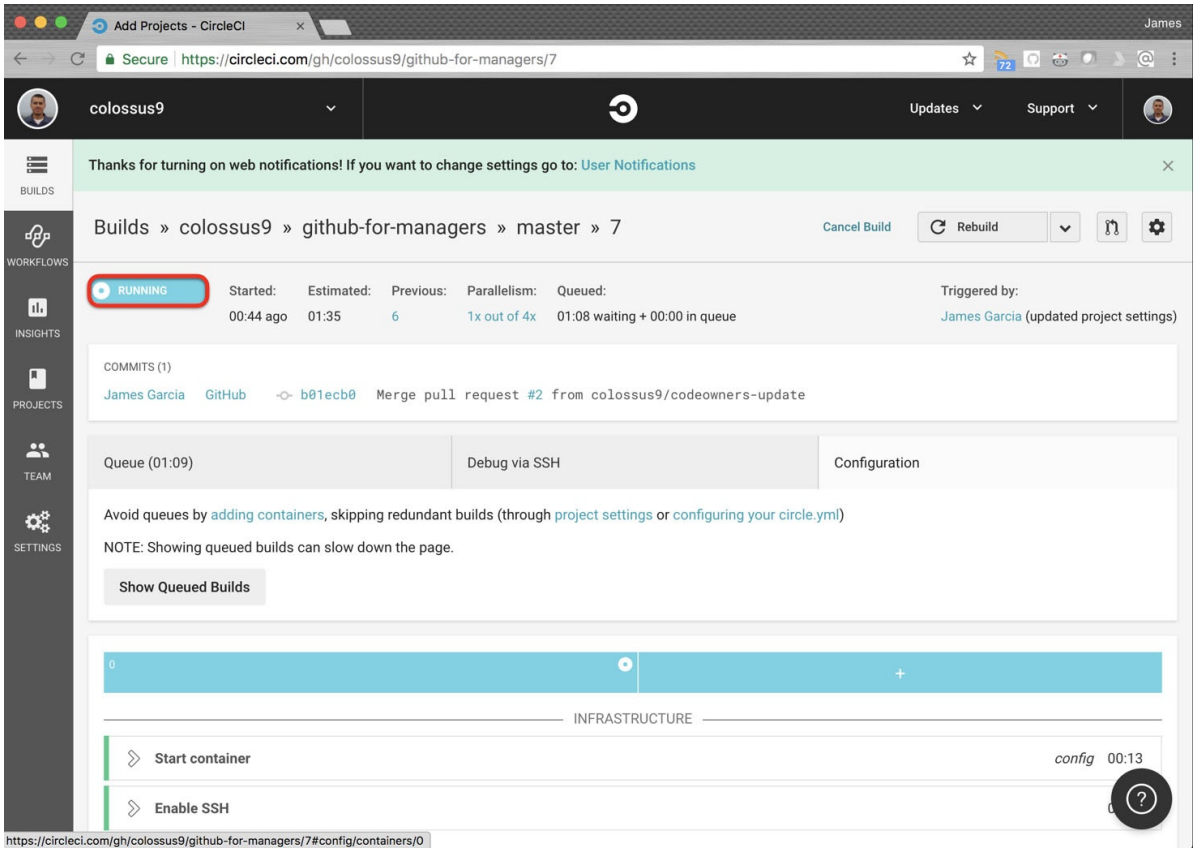

6. Next to **github-for-managers**, click **Setup project**.




7. Make the selections for setting up the project.
    - For **Operating System**, select **Linux**
    - For **Platform**, select **2.0**

- For **Language**, select **Ruby**
- Click **Start building**

8. After up to four minutes, your CircleCI tests should pass 🎉🎉.

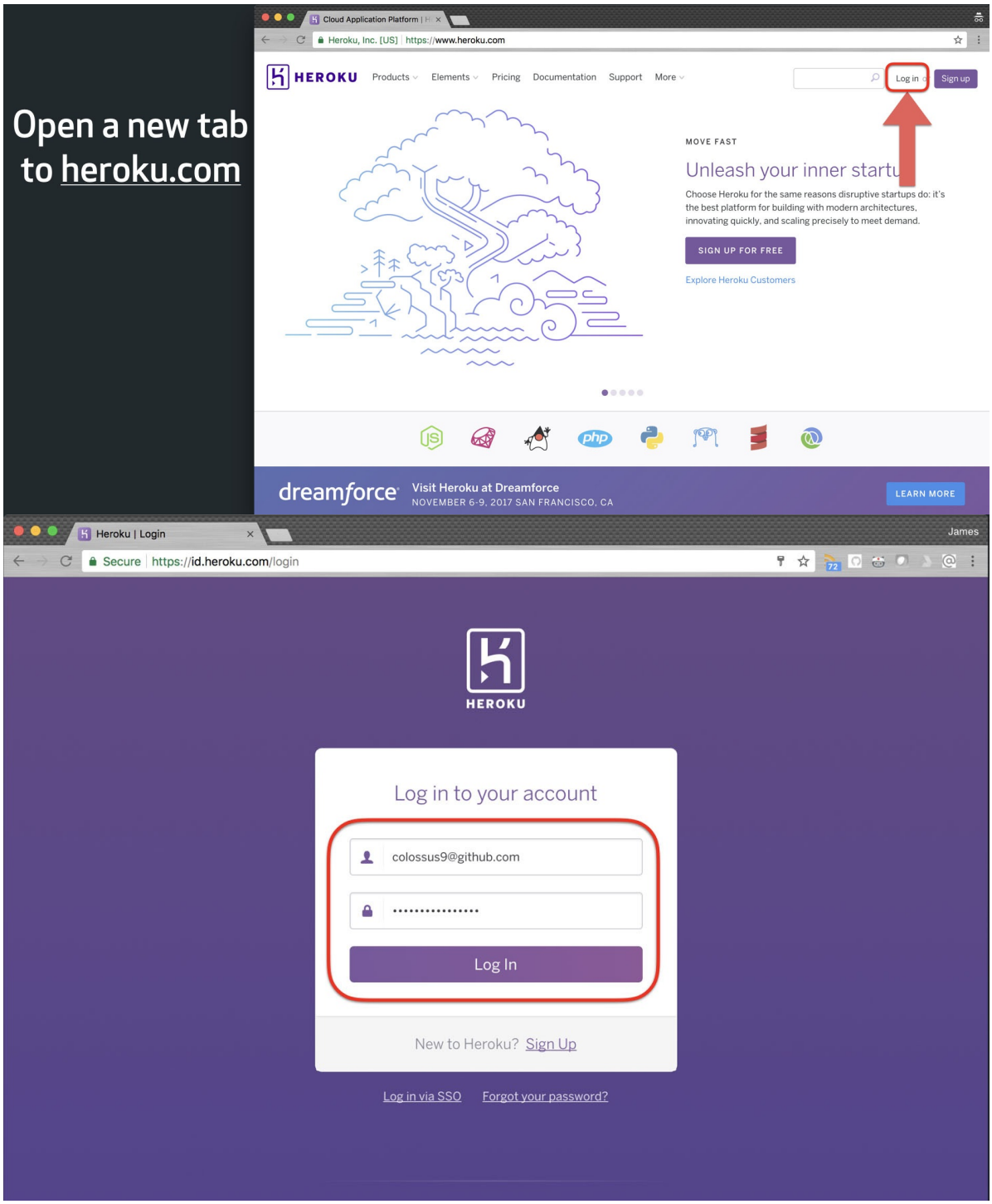


# Configure CD

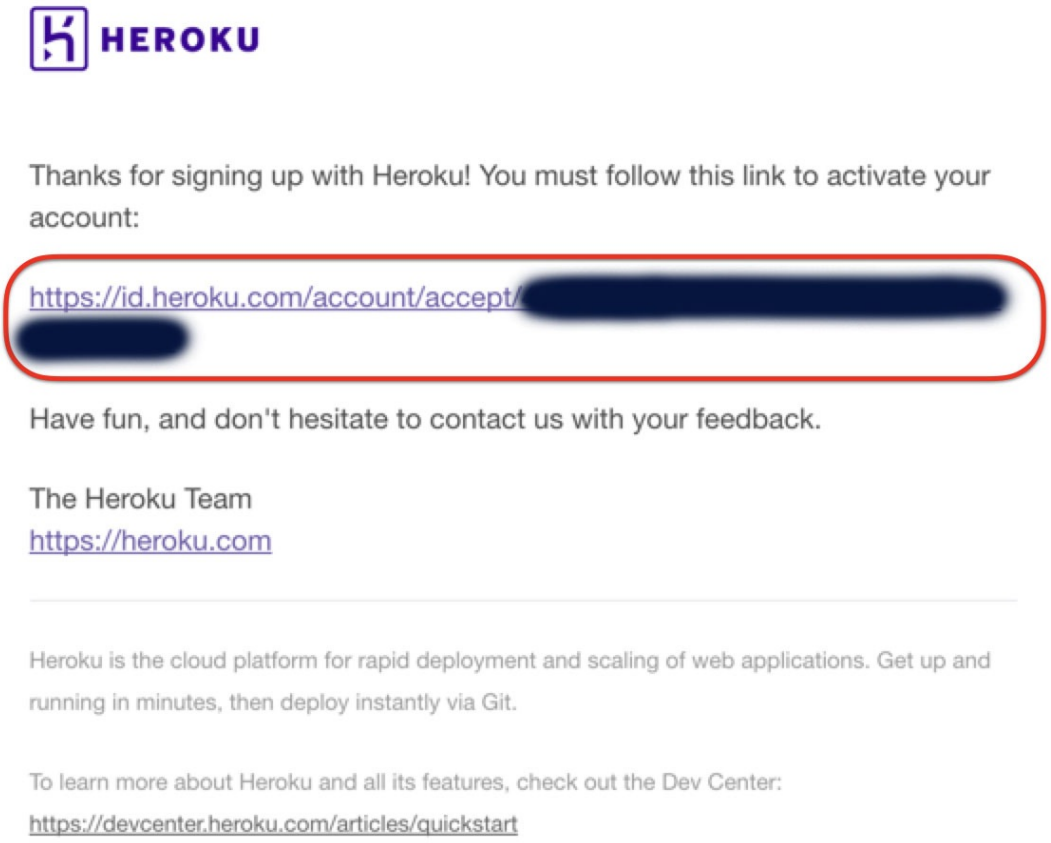
## Configuring Heroku and GitHub

First, let's connect Heroku to GitHub.

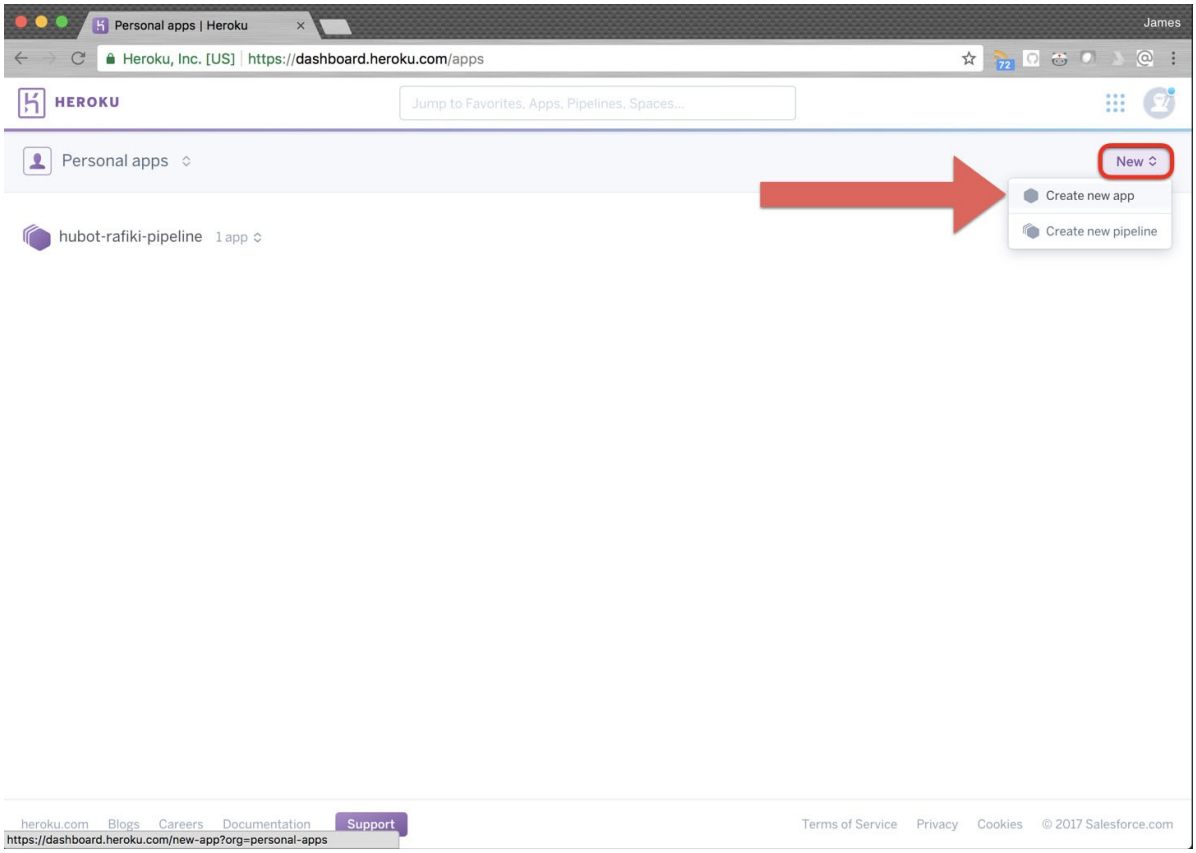
1. Open a new tab, go to the Heroku dashboard at https://dashboard.heroku.com and Log In.



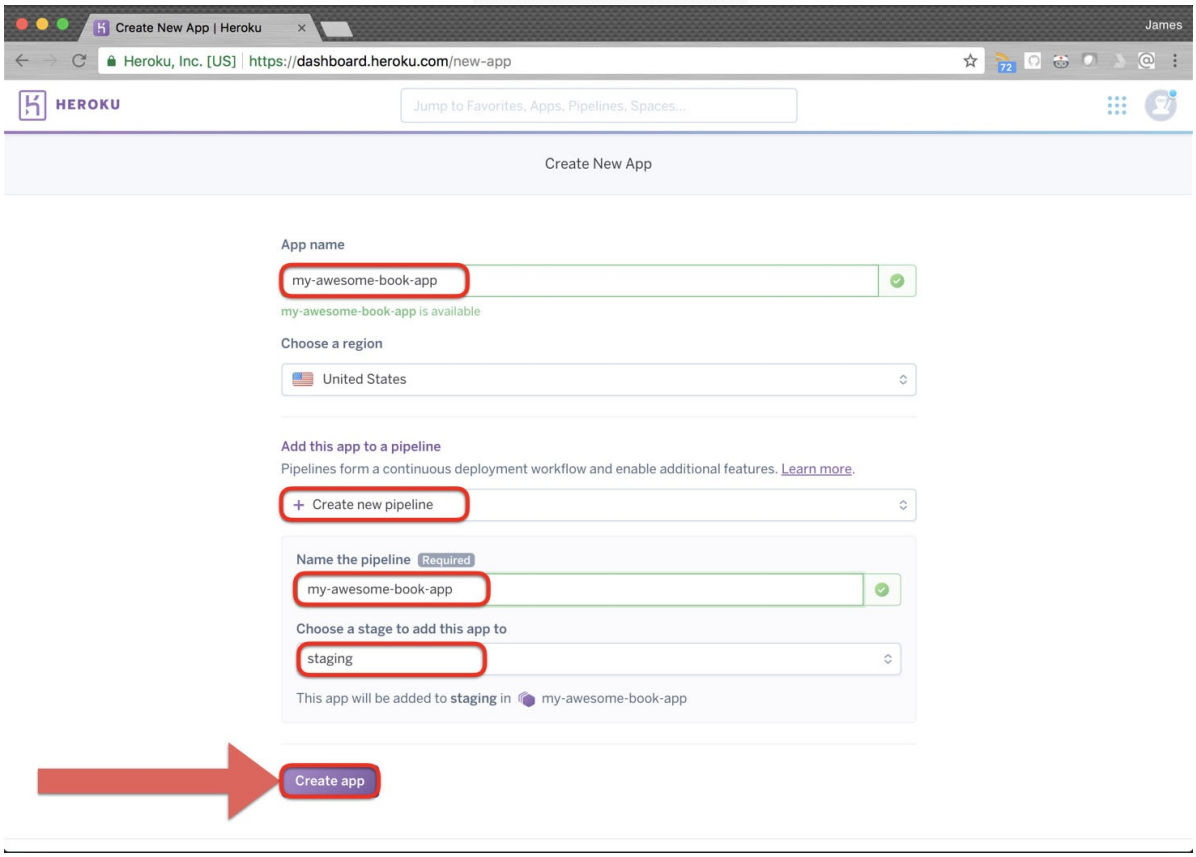
- If you haven't yet, register for a new Heroku account. You will be sent a confirmation email.



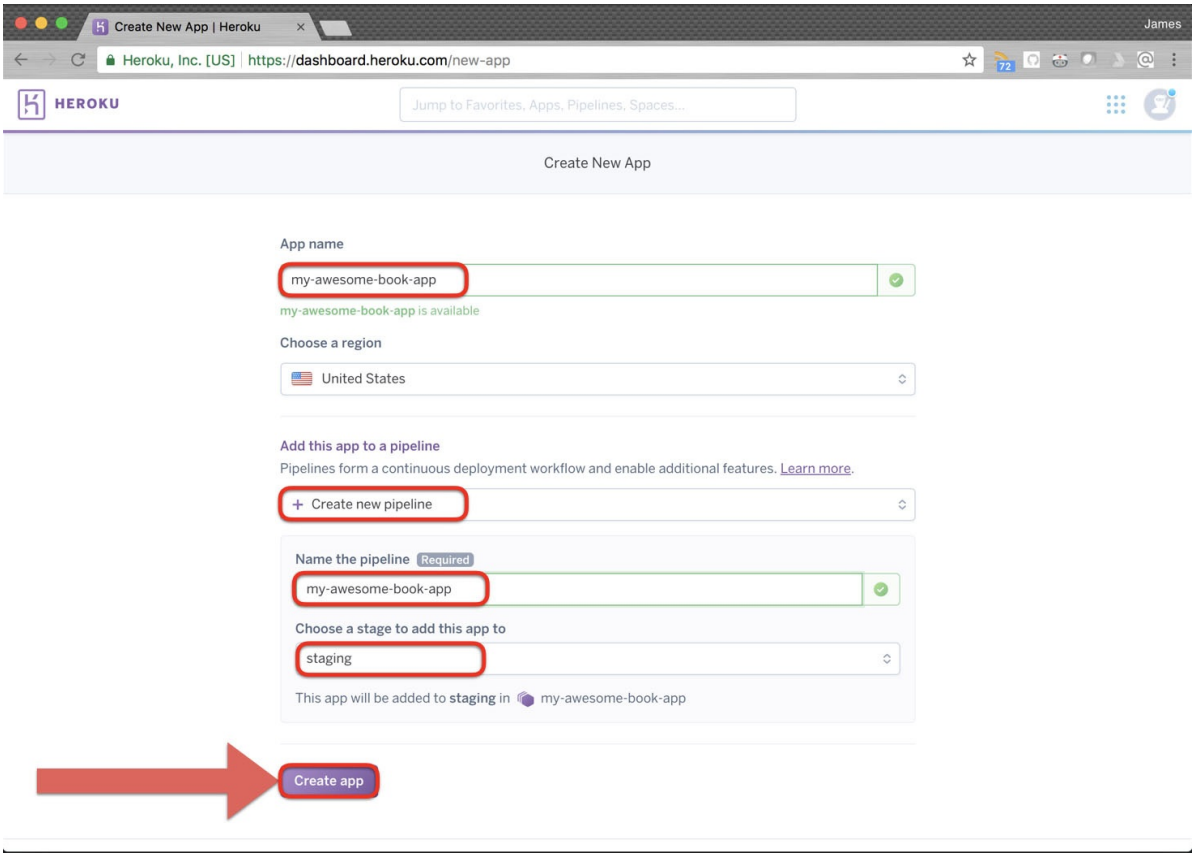

2. Click **New** > **Create new app**.



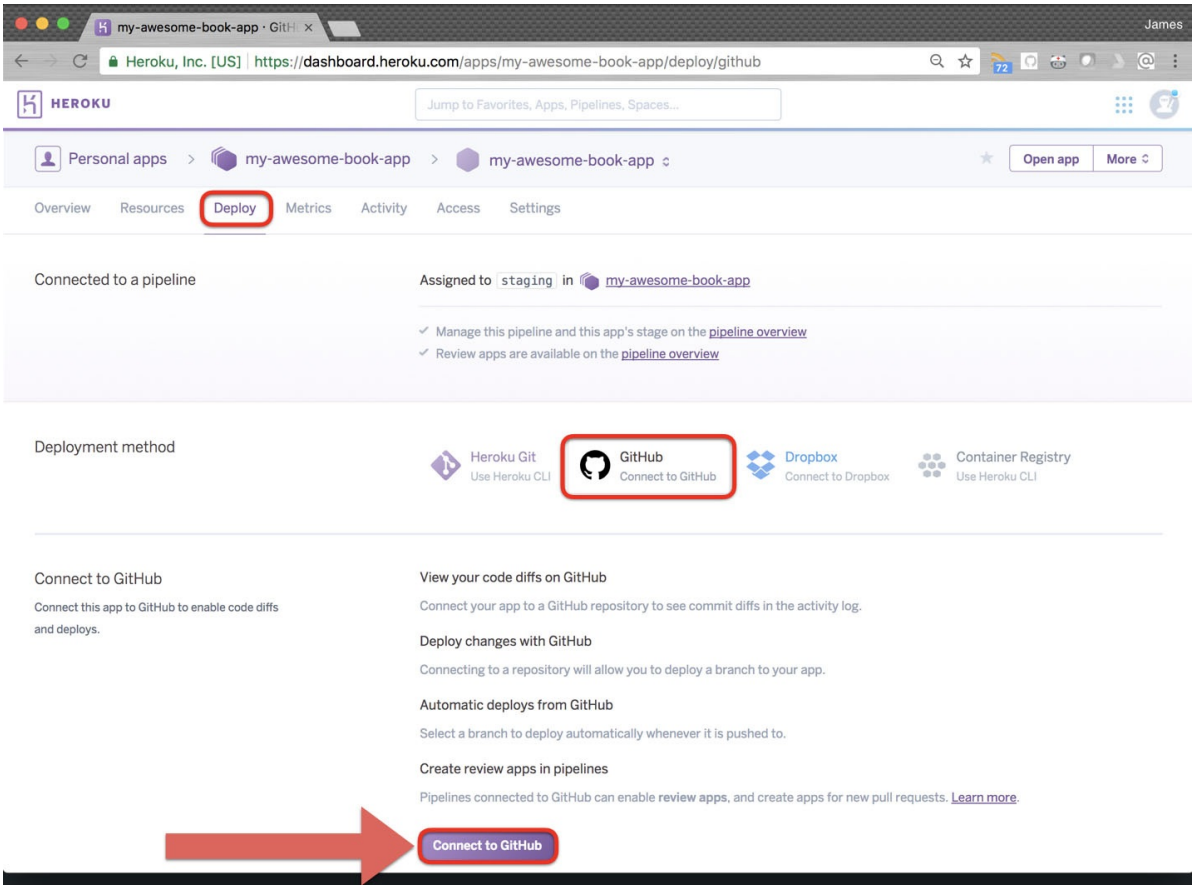

3. For the **App name**, enter `my-awesome-book-app-GITHUB_USERNAME`. Heroku app names must be unique so append `-GITHUB_USERNAME` to the name.




4. Click **Add to pipeline**, then select **Create new pipeline**. Leave the defaults and click **Create app**.

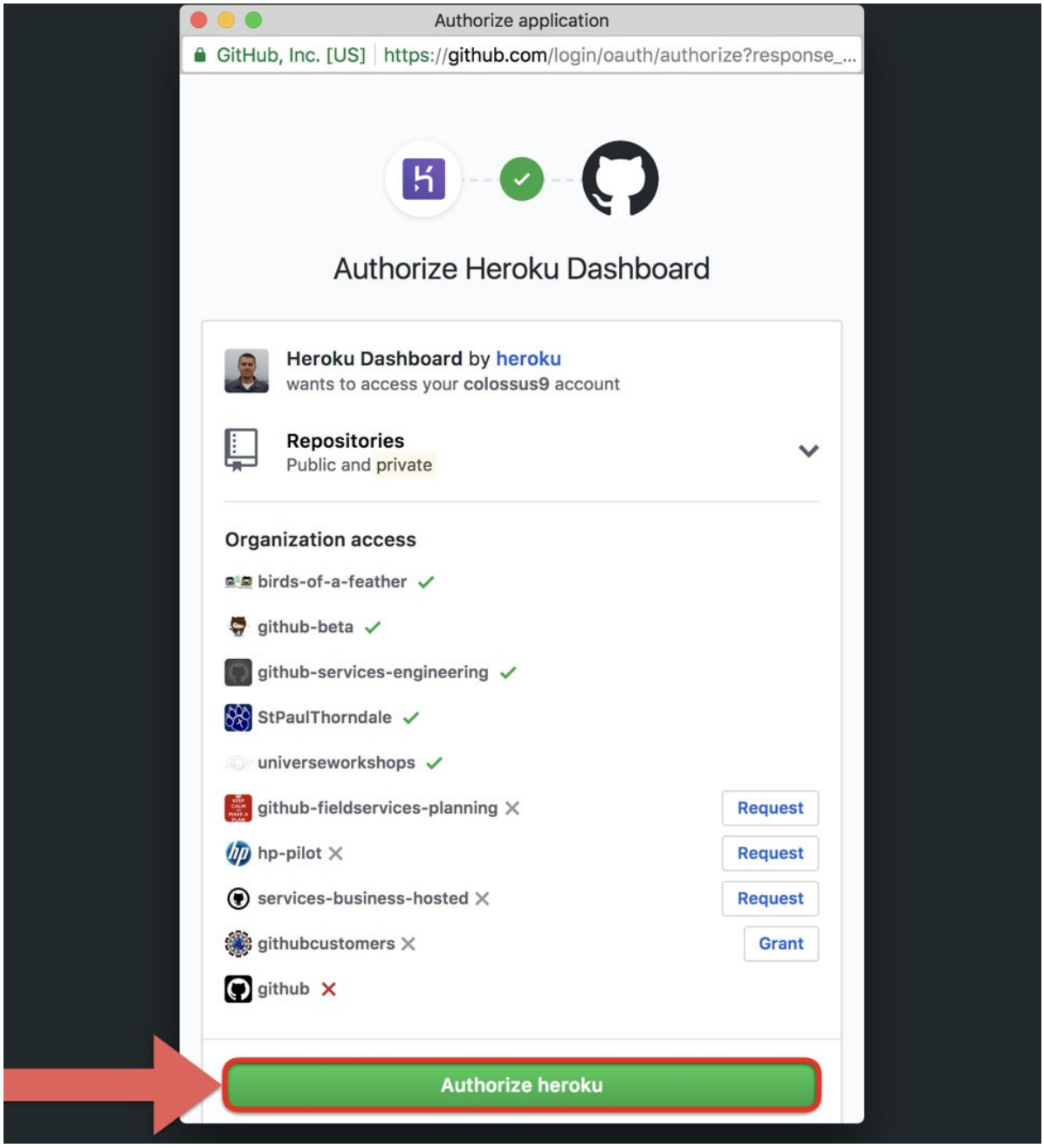



5. In the Heroku **Deploy** tab, for the **Deployment method** select **GitHub** and **Connect to GitHub**.

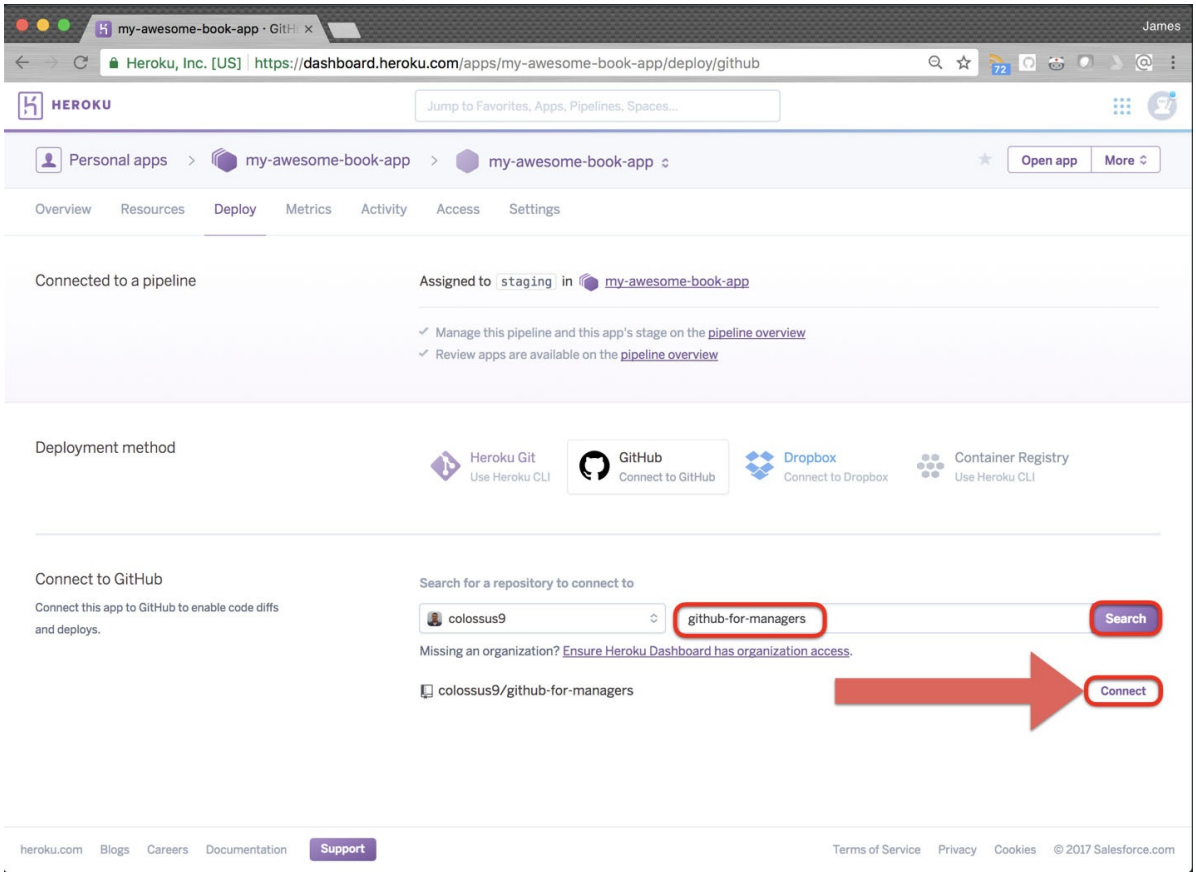



6. Log in if prompted, but it should take your already logged-in session, where you can

click **Authorize heroku**.



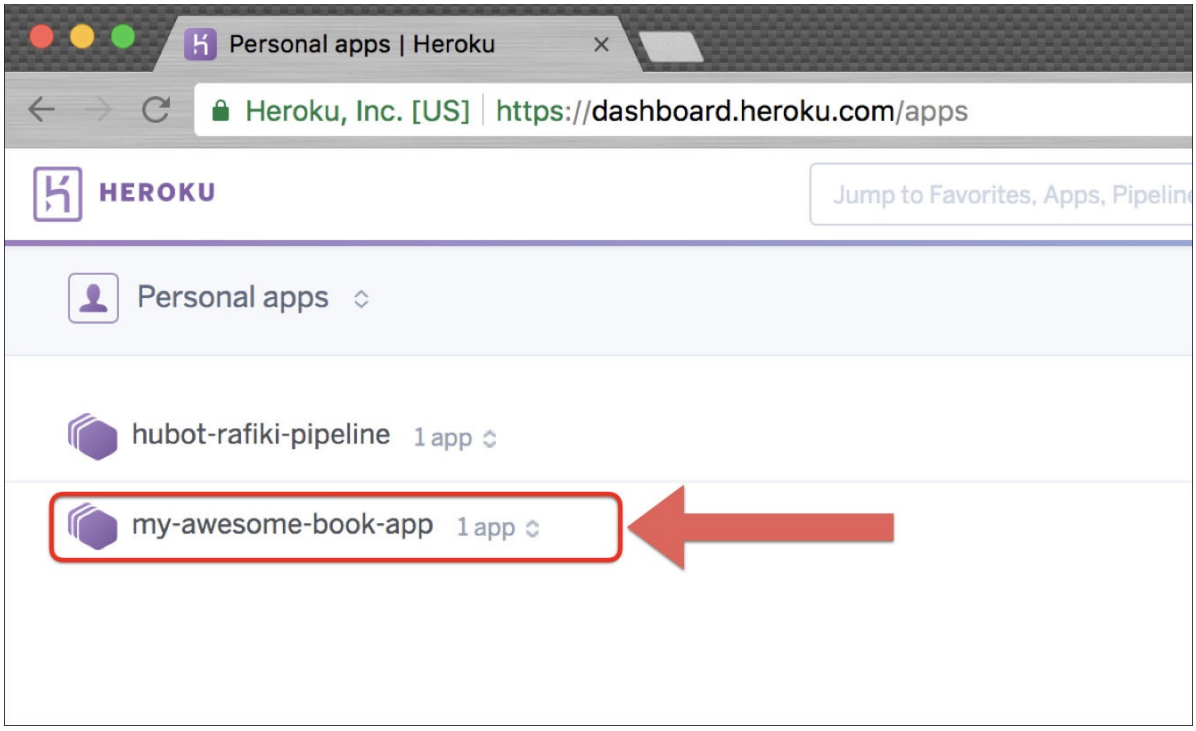

7. After authenticating Heroku to GitHub, type `github-for-managers` in the text box and click **Search**. Our repository should be displayed, then click **Connect**.



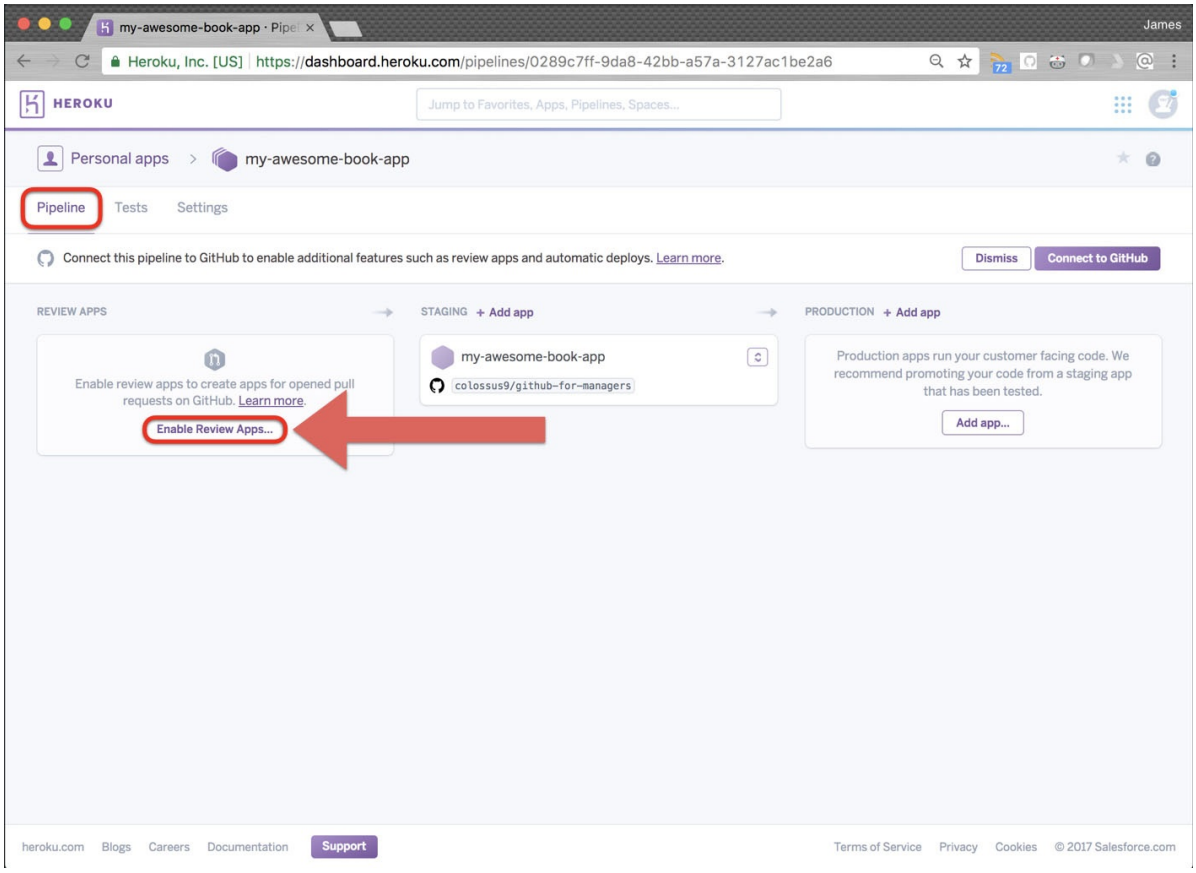

# Configuring Heroku Review Apps

Now, let's enable Heroku Review Apps. When a Pull Request is opened, each new commit can trigger automatic test deployments in an isolated environment, called Review Apps in Heroku.

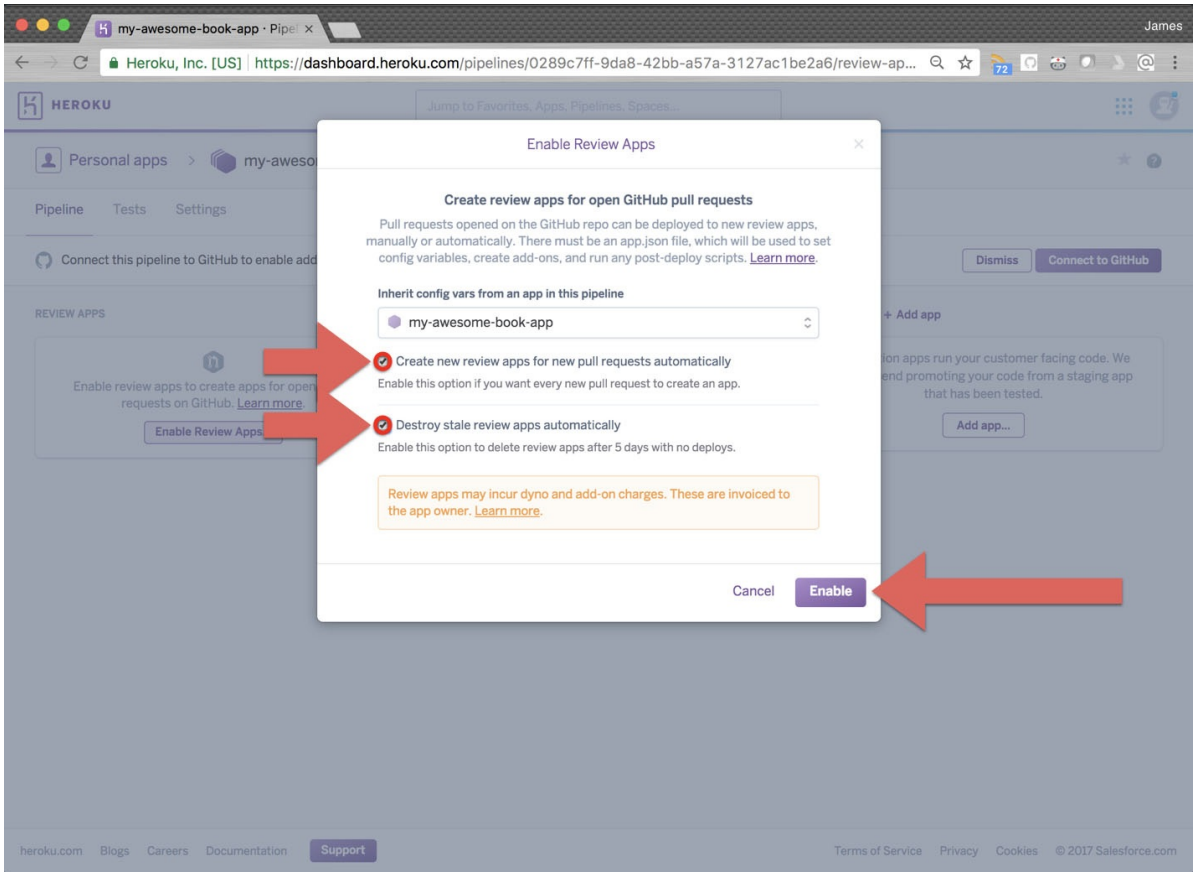
1. Go to the Heroku dashboard at https://dashboard.heroku.com. Select **my-awesome-book-app-GITHUB_USERNAME**.



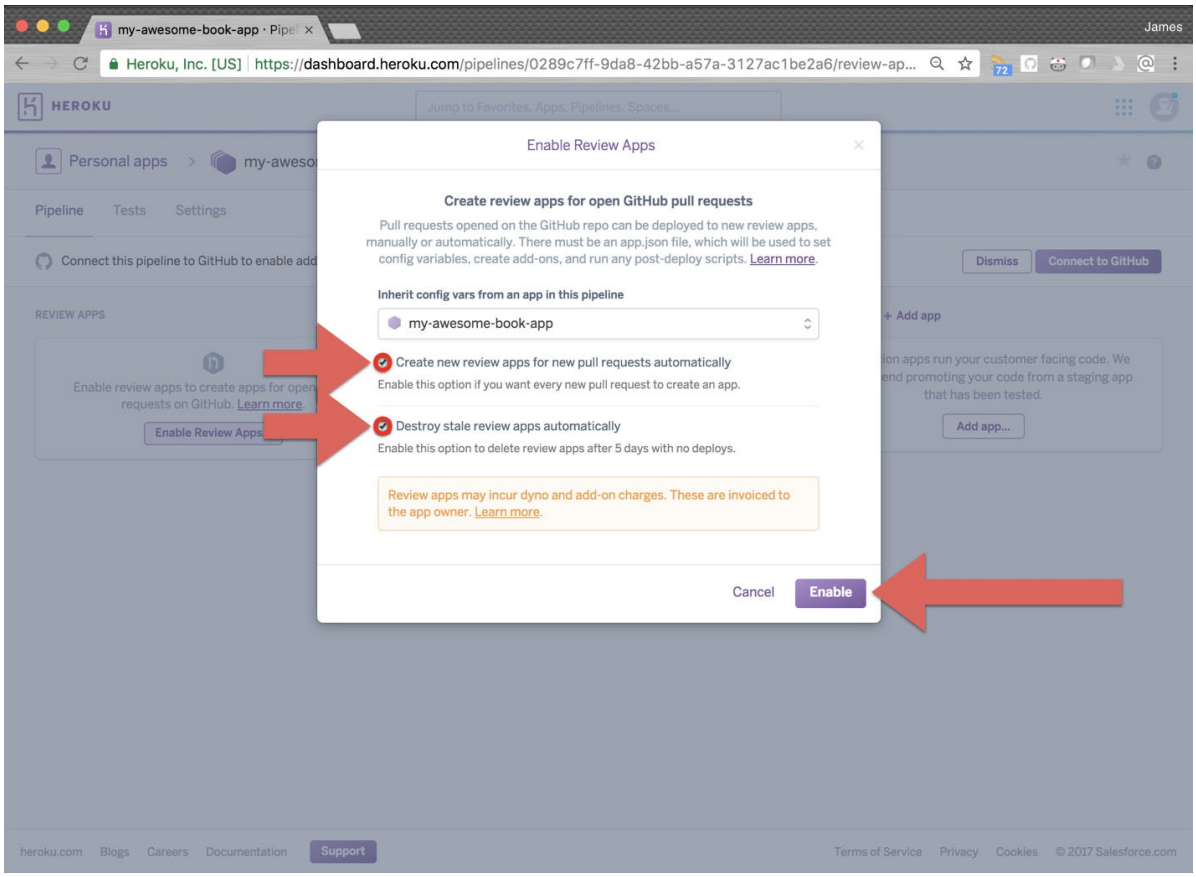

2. Click **Enable Review Apps**



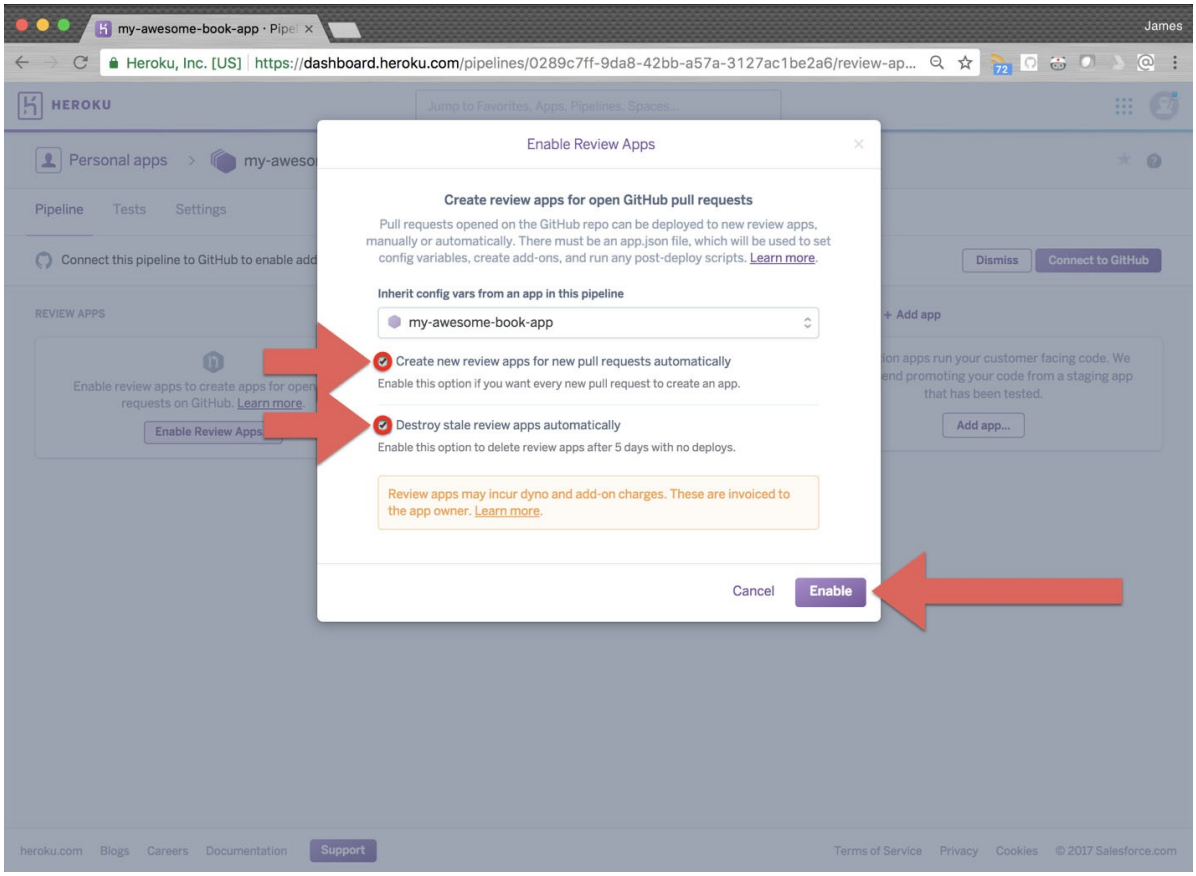

3. In the box that pops onto your screen, select both options.
4. ☑ Select Create new review apps for new pull requests automatically.




5. ☑ Select Destroy stale review apps automatically.



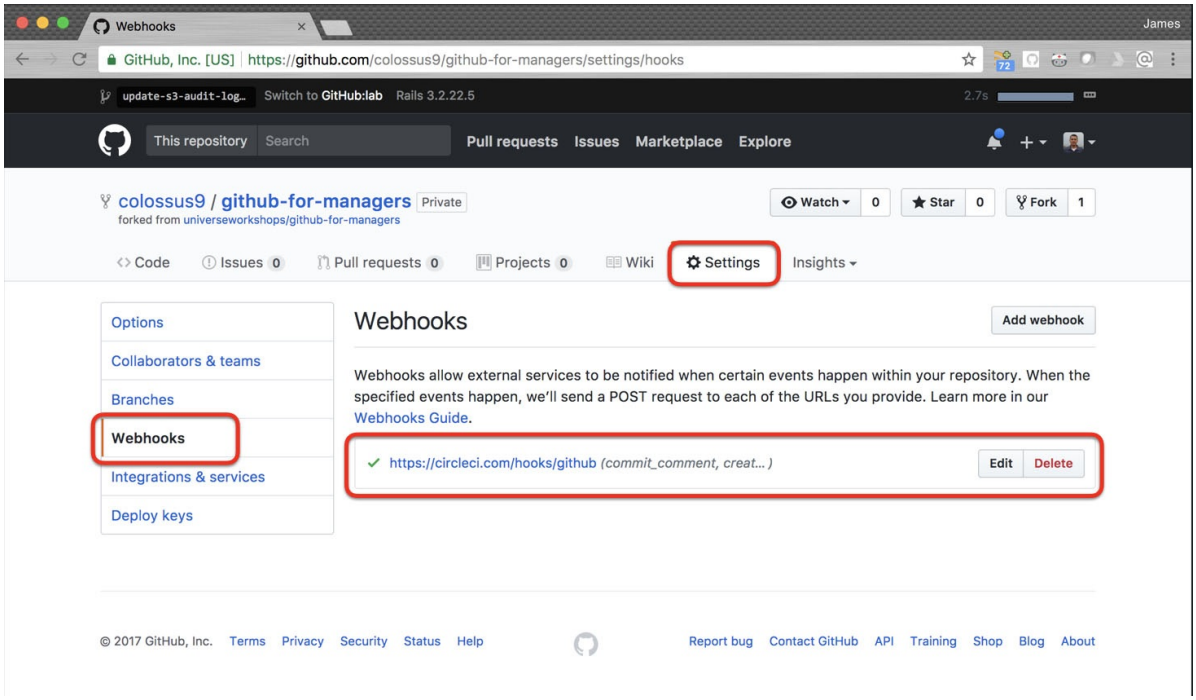

6. Click **Enable**.


my-awesome-book-app · Pipel
James
Heroku, Inc. [US] | https://dashboard.heroku.com/pipelines/0289c7ff-9da8-42bb-a57a-3127ac1be2a6/review-ap...
HEROKU
Jump to Favorites, Apps, Pipelines, Spaces...
Personal apps
Pipeline
Tests
Settings
Connect this pipeline to GitHub to enable add
Dismiss
Connect to GitHub
REVIEW APPS
+ Add app
Enable review apps to create apps for open
requests on GitHub. Learn more.
Enable Review Apps
Add app...
Enable Review Apps
Create review apps for open GitHub pull requests
Pull requests opened on the GitHub repo can be deployed to new review apps, manually or automatically. There must be an app.json file, which will be used to set config variables, create add-ons, and run any post-deploy scripts. Learn more.
Inherit config vars from an app in this pipeline
my-awesome-book-app
Create new review apps for new pull requests automatically
Enable this option if you want every new pull request to create an app.
Destroy stale review apps automatically
Enable this option to delete review apps after 5 days with no deploys.
Review apps may incur dyno and add-on charges. These are invoiced to the app owner. Learn more.
Cancel
Enable
heroku.com
Blogs
Careers
Documentation
Support
Terms of Service
Privacy
Cookies
© 2017 Salesforce.com

# Webhooks on GitHub

Check out the new Webhook created in your GitHub repository. Webhooks are a powerful tool for streamlining your development pipeline, and allow you to build or set up GitHub Apps which subscribe to certain events on GitHub.com. GitHub Apps and integrations use Webhooks, but individual developers and teams can also use Webhooks to customize their workflow.

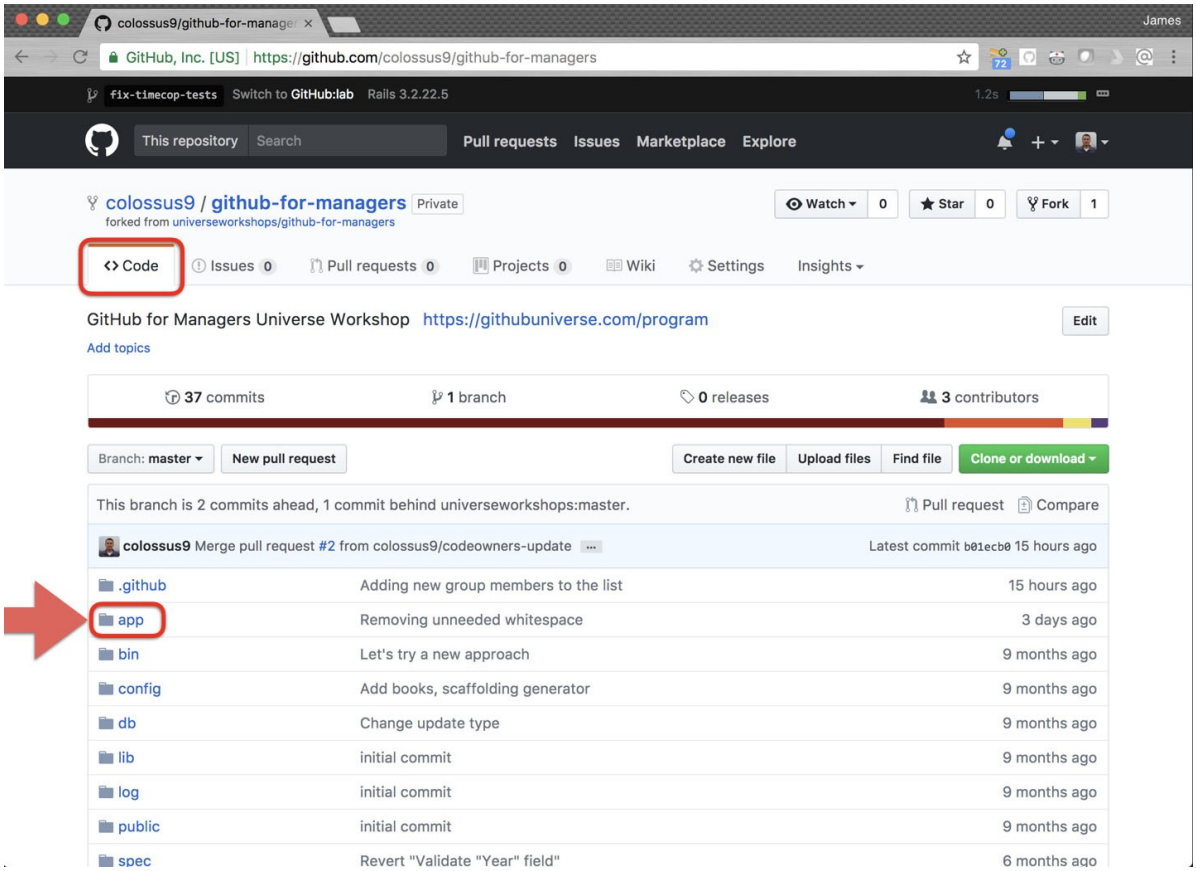
1. Go to the repository's **Settings** tab, then select **Webhooks** to see that it was automatically created when you authorized it.



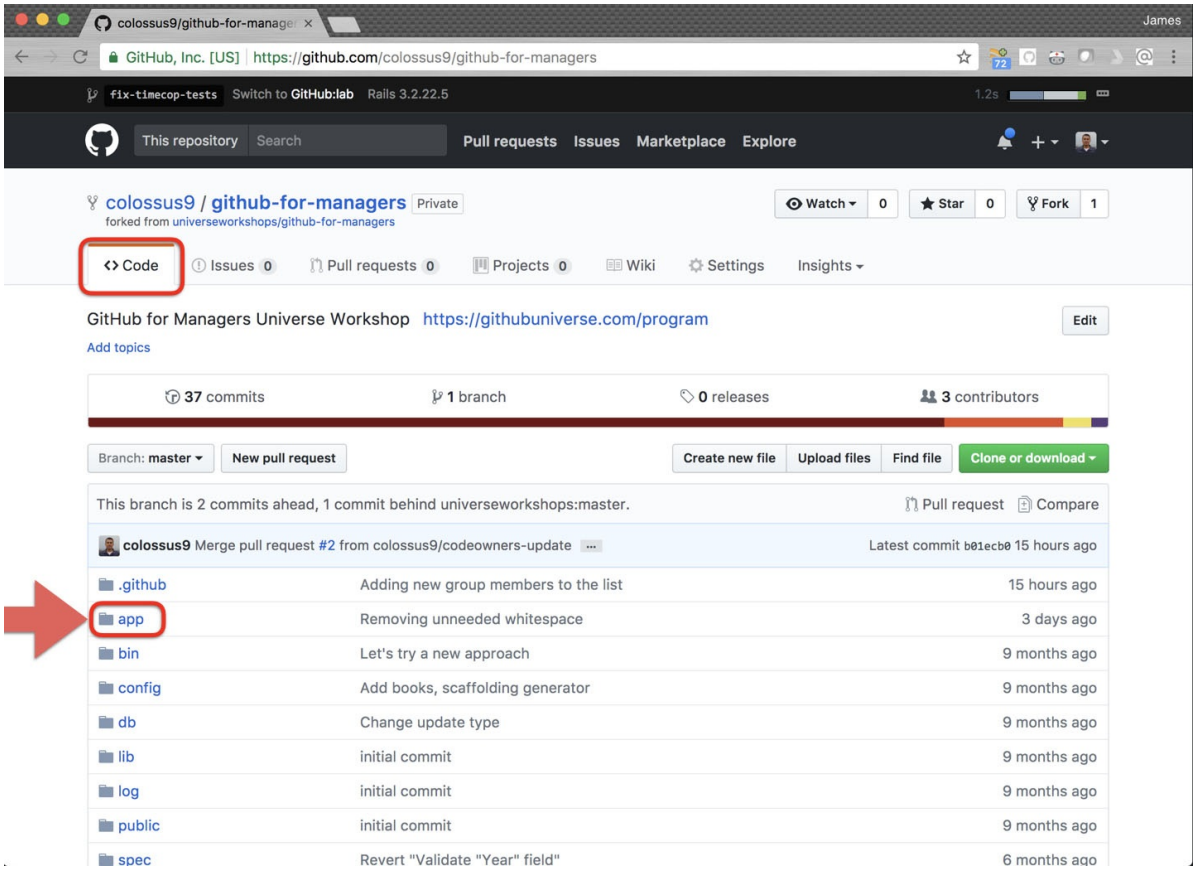
# End-to-End Test

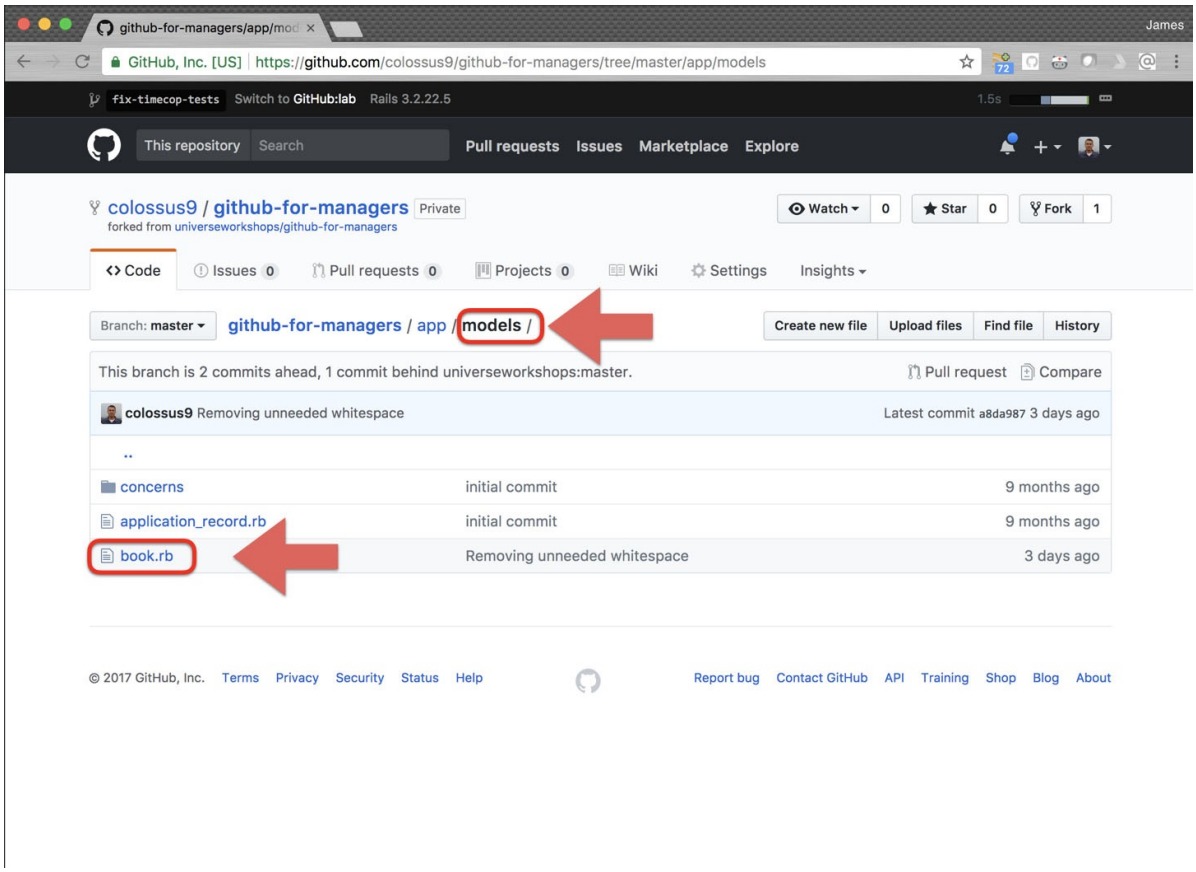
Now we have CI/CD setup for our app. Let's try it out!

1. Navigate back to your GitHub repository **fork** and select the **Code** tab.

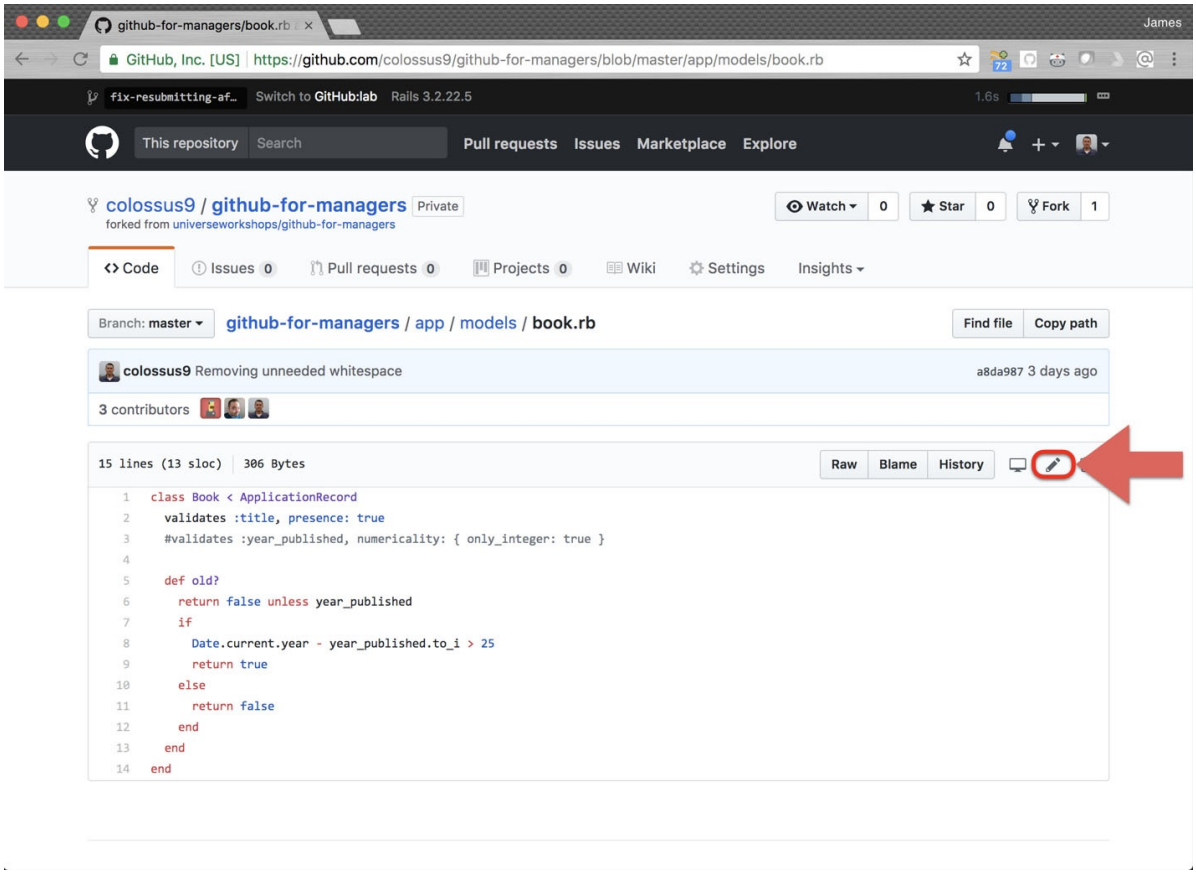


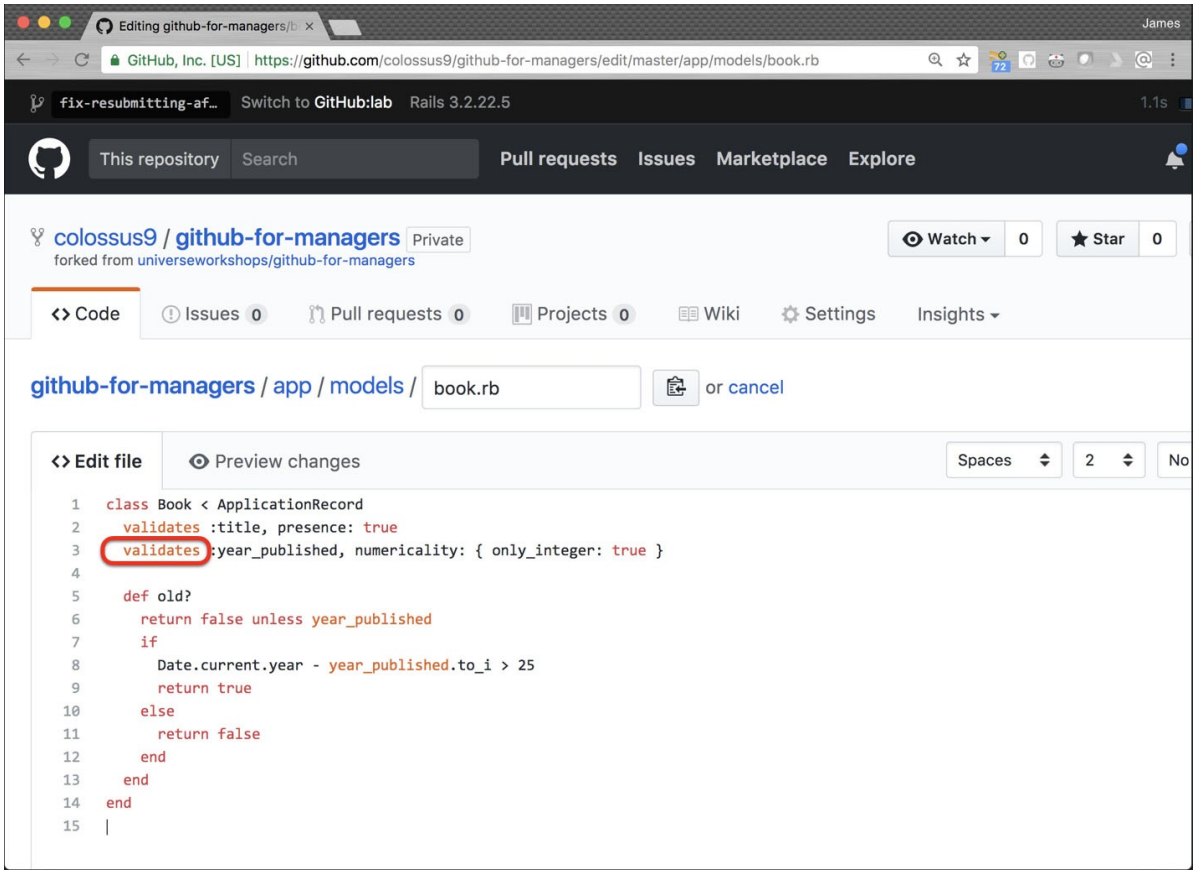

2. Go to the file `app/models/book.rb` and click the ✏️ pencil icon to edit it.




3. On Line 3, uncomment the line by deleting the `#` character at the beginning.

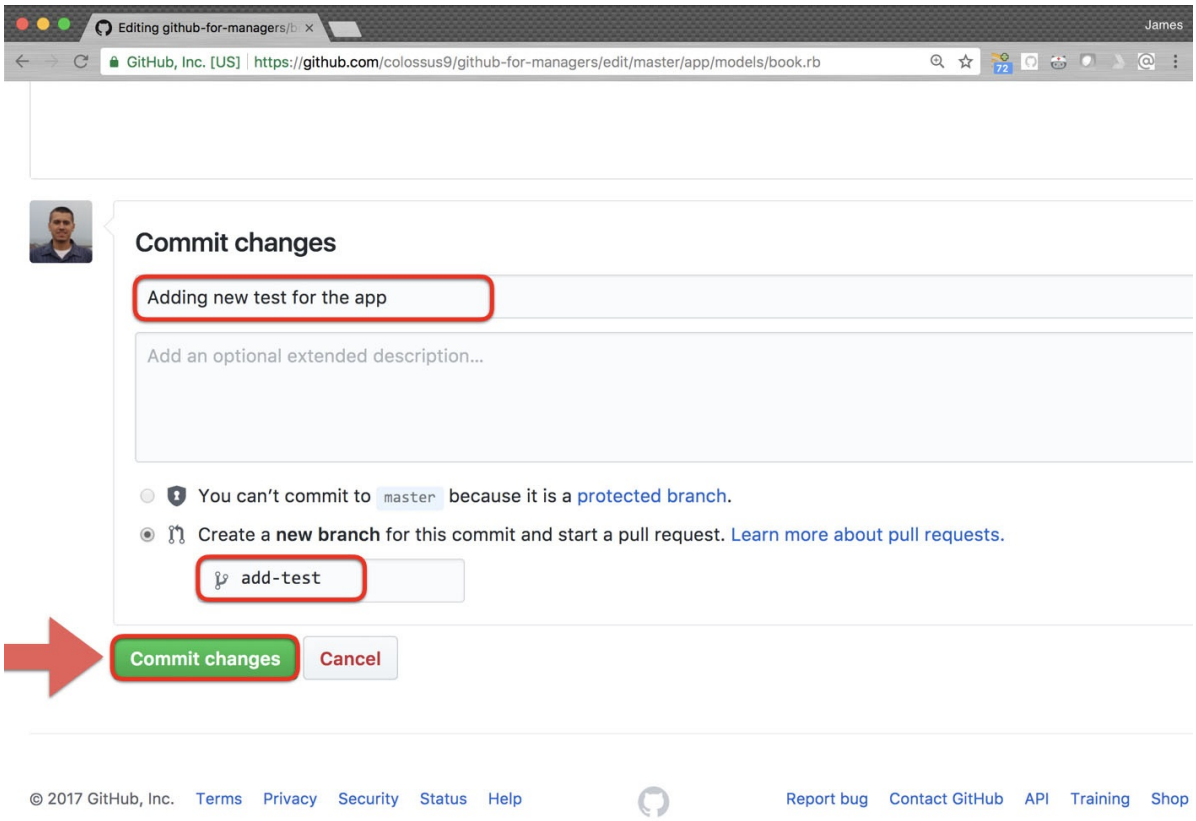



4. Scroll down to the **Commit changes** section. Add a commit message, using the following as an example:

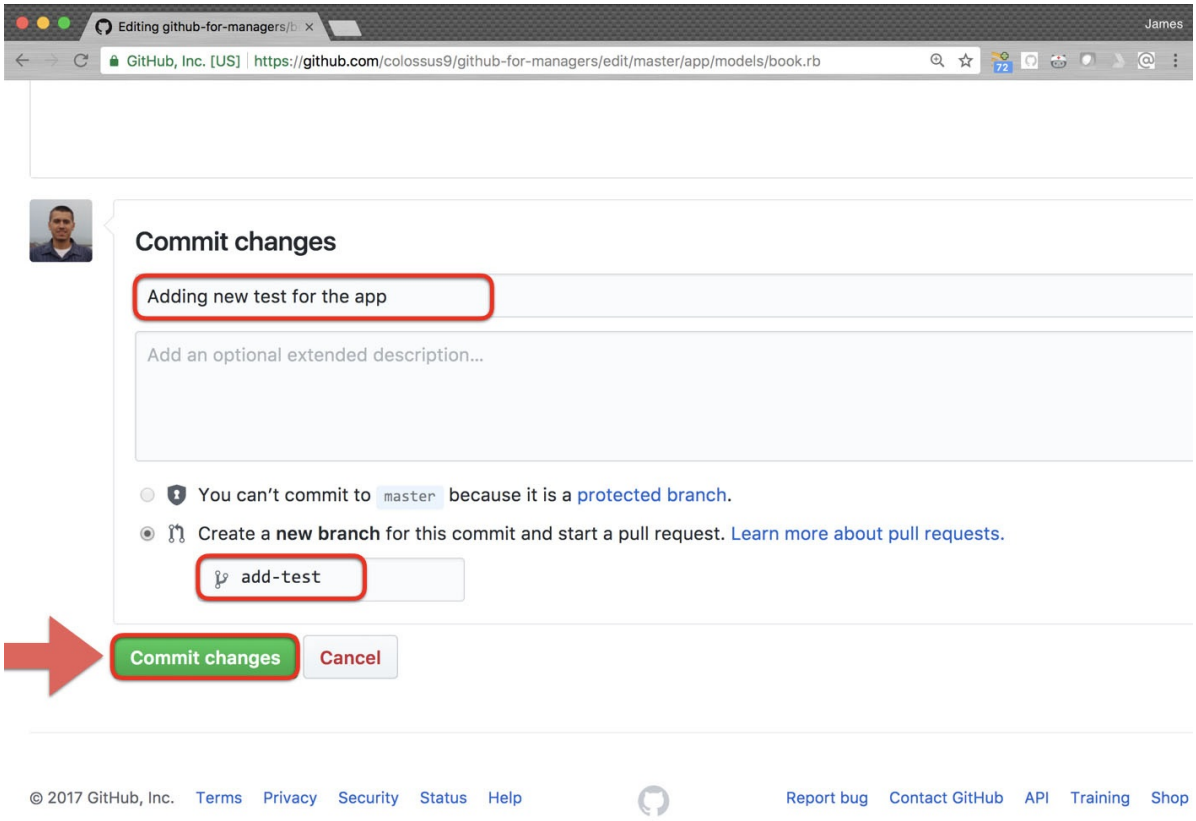
```
```

```
Adding new test for the app
```

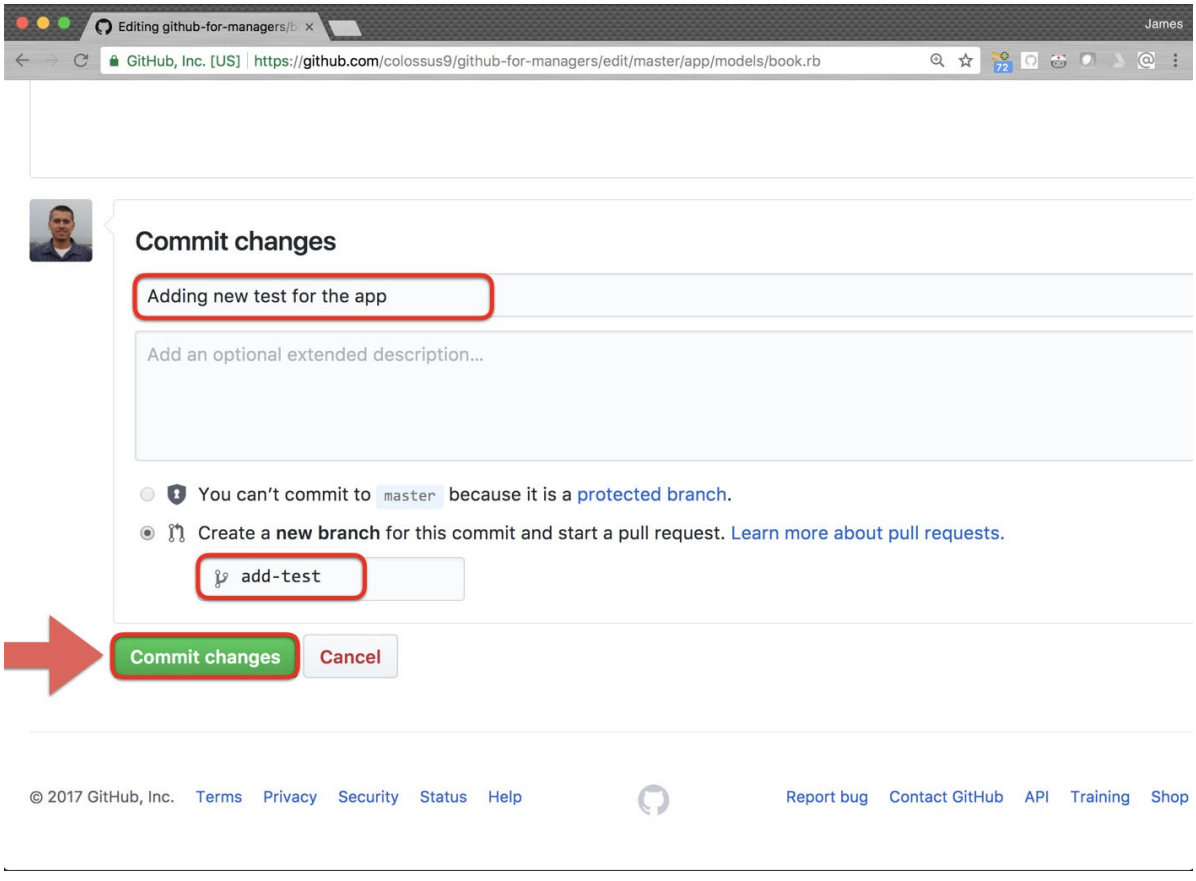



5. In the **Create a new branch...** option, enter a new branch name, using the following as an example:

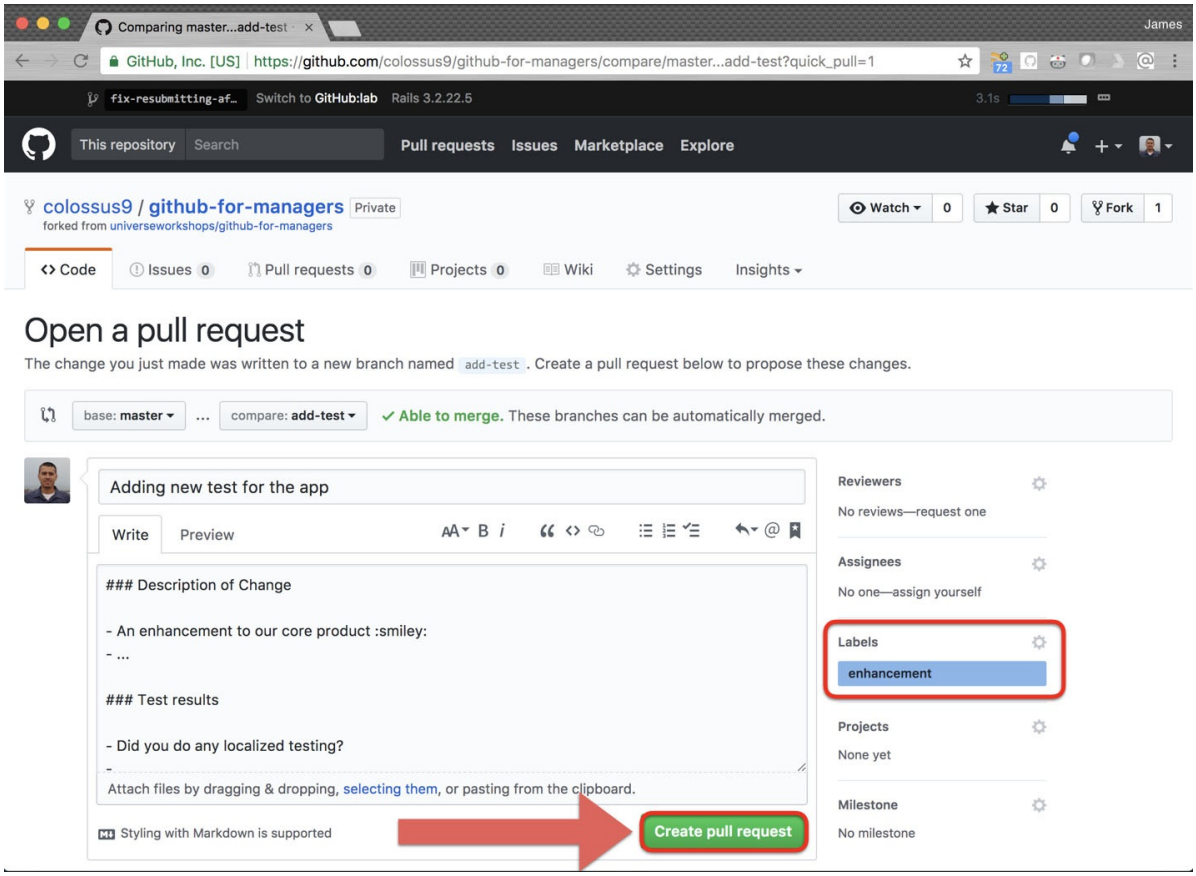
```
add-test
```



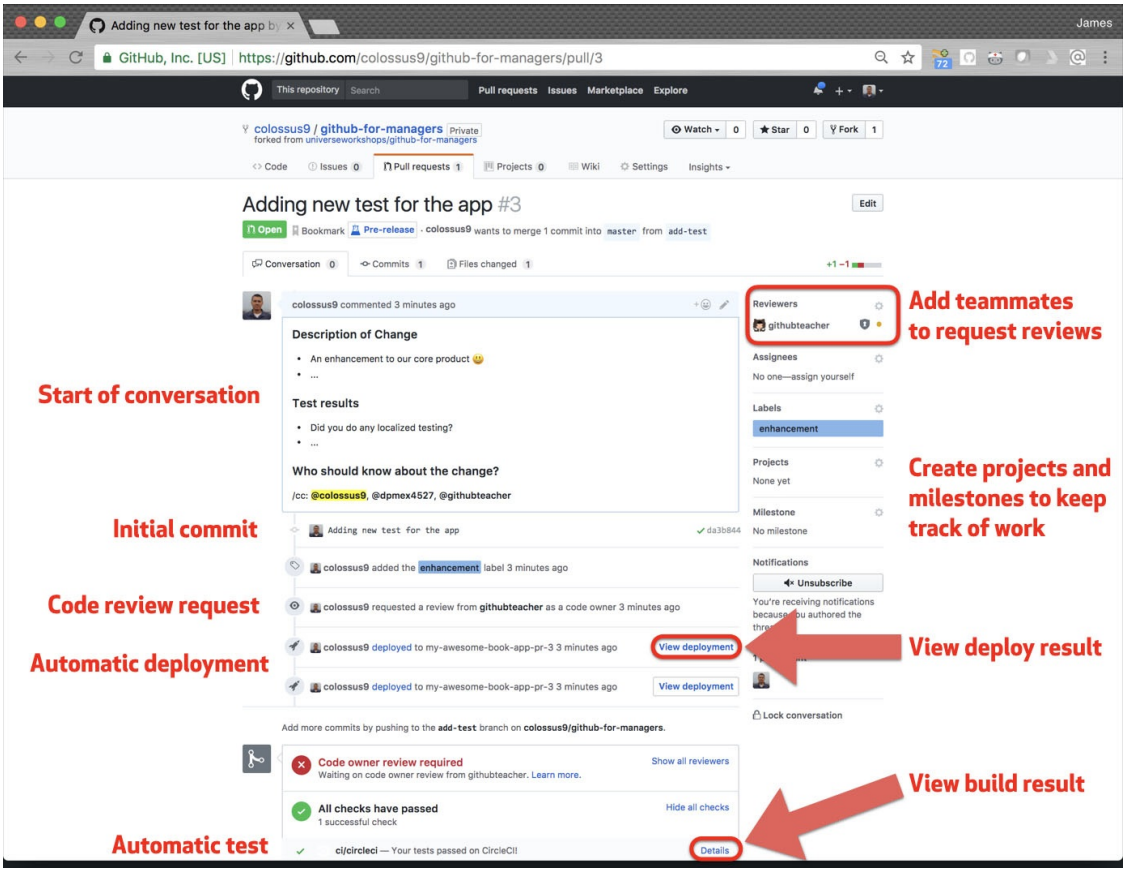

6. Click **Commit changes**. A Pull Request will automatically be opened for this proposed change.




7. Add the `enhancement` label and click **Create pull request**.

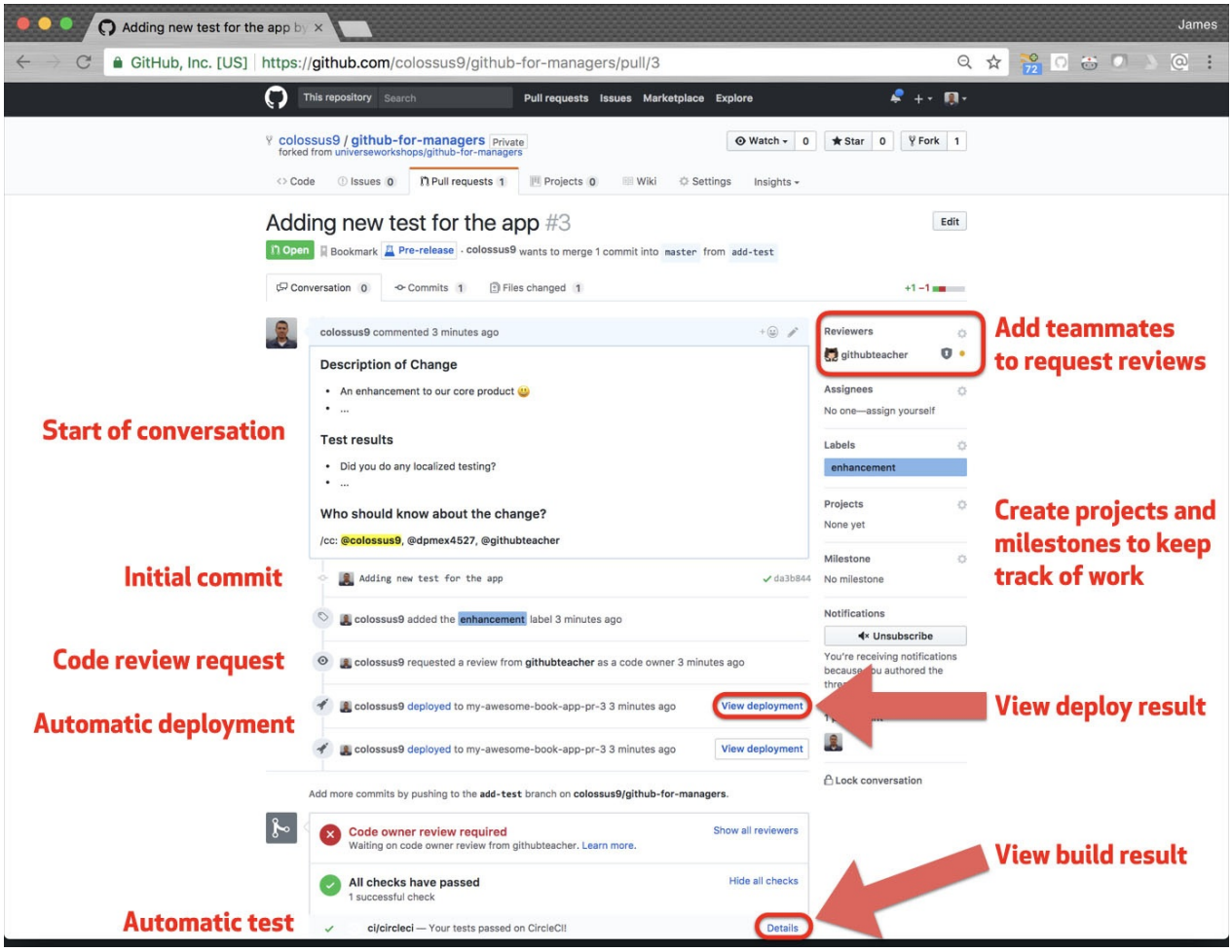



8. The Pull Request will now show the creation of a conversation about the change as well as the integrated dev tools you can follow throughout the requested change.
    - Code Owner enforcement with automatic Code Review
    - Conversation
    - Initial and subsequent commits to this branch
    - Automatic build/test with link
    - Automatic staging deployment with link

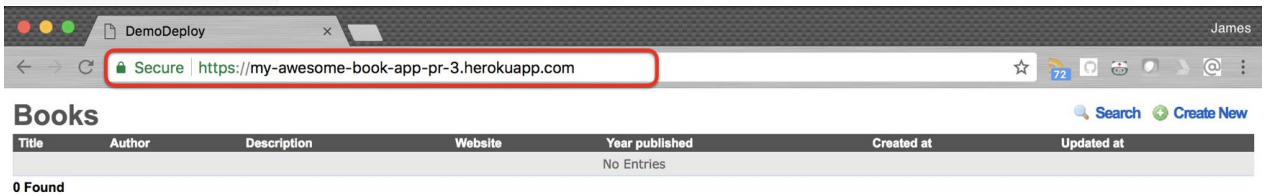

Start of conversation
Initial commit
Code review request
Automatic deployment
Automatic test
Add teammates to request reviews
Create projects and milestones to keep track of work
View deploy result
View build result
Adding new test for the app #3
Description of Change
Test results
Who should know about the change?
Reviewers
Assignees
Labels
Projects
Milestone
Notifications
View deployment
Code owner review required
All checks have passed
Details

# Interacting with GitHub Apps

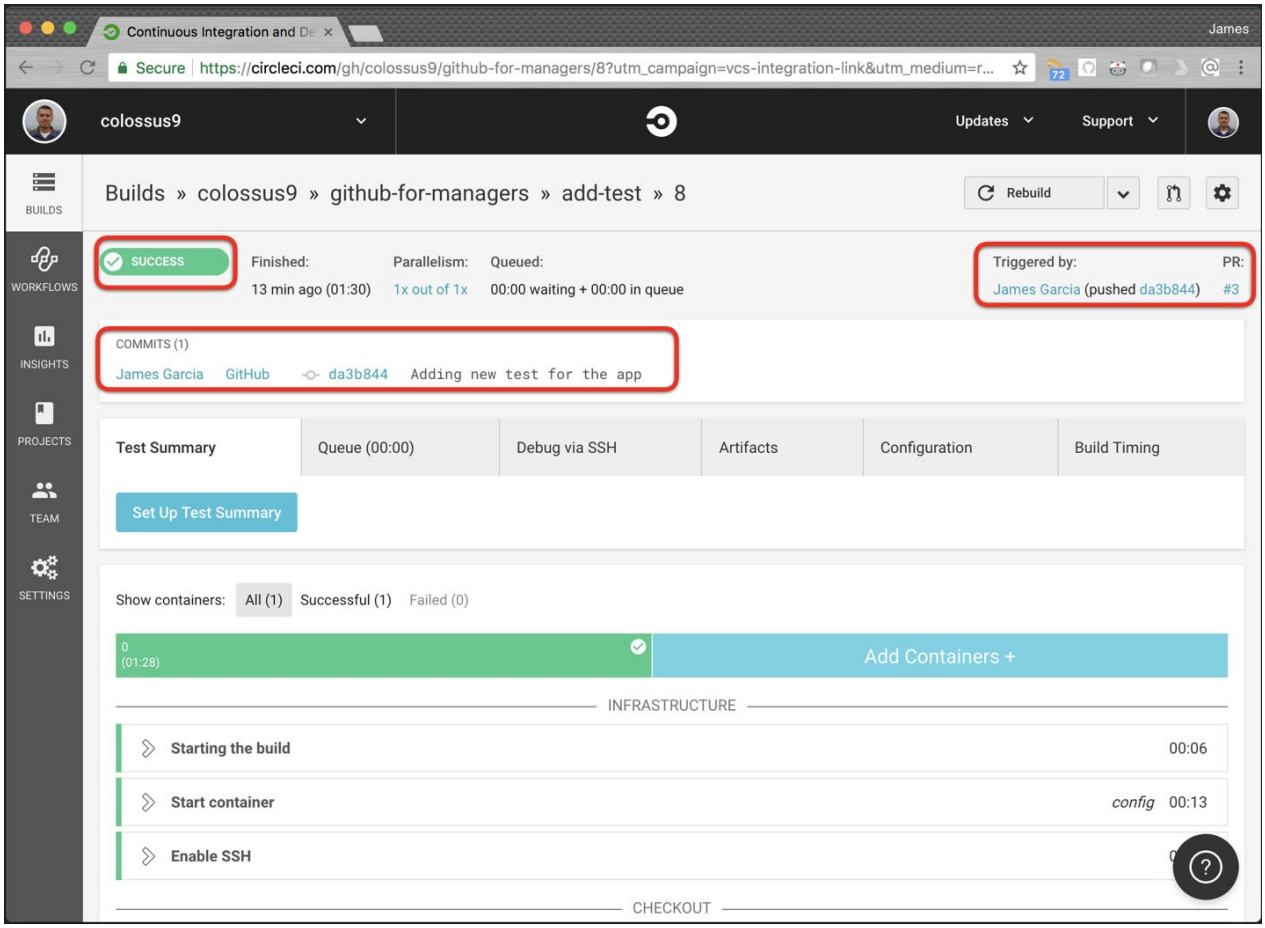
Browse the links in the Pull Request to see additional detail about your change and the checks it triggers.




One of the deployment links in the thread will link to an actual deployment of the `MyAwesomeBookApp` application.



Another link to show you the status of the CircleCI build/testing.



# Reviewing the Pipeline

Let's take some time to discuss what we've covered so far.

## Software development pipeline

We discussed the GitHub flow, and worked together on a project from start to finish. This means the project went from planning stages, through creating code, into having tests run (and pass), and finally deploying to a live site.

## Avoiding context switching

Developers drive progress, and switching between many tools takes away valuable time and focus from the real work. Integrating all parts of the pipeline into one platform allows developers to avoid switching context and keep their focus centered.

## Automation when possible

Many of the pieces of the pipeline are automated - as many as possible. Though this requires up-front work to set up the automation, it saves an abundance of time and minimizes mistakes that could easily be made if done manually.

# More Possibilities

So far, we've gone through one pre-defined workflow. Hopefully at this point you're seeing how this general concept can be applied to your own teams and projects to automate more to reduce repetitive tasks and human error.

Let's explore some additional features and examples of how you can fully take advantage of a software development pipeline with GitHub.

## The GitHub API

GraphQL is a powerful way to interact with GitHub. If you find that you want something to integrate a certain way, but the tool doesn't exist, you can create custom tooling using GitHub's API and webhooks. Find more documentation here.

## Pulse/Graphs

We covered these briefly earlier, but let's talk a bit more about Pulse and Graphs.

When you want to know more about what's going on in any given repository, the "Insights" tab is the right place to look. Here you can see all sorts of valuable information about the code and the developers working on it.

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

## Customize your pipeline

With integrations, configurable GitHub settings, and direct access to Git and GitHub data via the API, the possibilities of how your toolchain could look are endless.

# Documentation with GitHub Pages

Projects can use GitHub Pages to provide end user documentation, testing instructions, and more. With GitHub Pages, you can maintain a project's documentation right from the repository inside a special folder called the `docs` folder.

# Create Structure for Documentation

Before we enable GitHub Pages on our repository, we should create a `docs` folder in our repository so we can select it as our the source for our documentation.

1. Create a new branch
2. After creating a new branch, click **Add a file**
3. In the field, enter `docs/README.md`

```
> Notice anything cool when you add the `/`?
```

4. Commit the new file addition on your branch
5. Create a new pull request, comparing the `base: master` to your new branch.
6. Merge your new pull request.

# Enable GitHub Pages

You won't be able to see your site on the web until you enable GitHub Pages on this repository. All we're doing is asking GitHub to take what's on the master branch and publish a website based on its contents.

1. Click on the **Settings** tab.
2. Scroll down to the GitHub Pages section.
3. Select the **None** dropdown.
4. Click on `master/docs` branch.
5. Click **Save**.

Now that we have enabled GitHub Pages, lets take a look at the site that is currently being generated by the README.md file that we added to our `docs` folder.

# Selecting a Theme

1. Go back to the **Settings** tab for your repository
2. Click **Choose a theme**
3. Select a theme from those provided and click **Select theme**

Your GitHub Pages site is now using a theme.

# Designing and Guiding Workflows

Developers should know how to use Git and GitHub on a day to day basis. However, before any team gets started, it's critical to have a defined workflow. Later, we'll shape our own workflows as practice.

## Scenarios

Before we map out our own new ideas or problems with workflows, let's take a look at some existing workflow challenges. Consider the following scenarios and how you might create a workflow and choose tools to set the team up for success.

## Project 1: Tests and Releases

We have our code set up and we want to have the tests automated. The tests are already existent in the file that is native, so a Circle file wouldn't be necessary. We want all tests to be passed before any branch is merged. We don't have anything to deploy at this point in time, but we do want to have a recommended strategy for creating releases and how we will use release branches. We need to communicate these strategies to the team.

## Project 2: Repository planning organization

The majority of our work happens in issues and markdown documents. We want to get more organized with our issues with labels, crosslinks, project boards... whatever it takes. Things get passed by the wayside now. It's unclear to others how to get things done, so we also need help organizing templates and the information on the README so other teams know what's going on. We should also have some kind of recommended review process to encourage newer or less technical members from committing directly on master.

## Project 3: Migrated from CVCS

We just came from Subversion, and we did a clean cutover. That's all well and good, but our branching techniques haven't really changed much. We need some guidance on general workflow best practices. We have Circle ready to be connected but we aren't using it

properly on pull requests.

## Project 4: Simple deployment pipeline

All of the code is there, but it's a pain to manually deploy. We want anything that goes to `master` to automatically be deployed to development.

## Project 5: More complex deployment pipeline

We want automatic deploys to production from master, but more importantly, we want a deployment pipeline. We want review apps to be spun up automatically from each pull request, so we can immediately get review from everyone. Then, once it's merged into the `development` branch, we want it to automatically go to the development server, and when it's in `master`, to go to production.

# Real World Team Work

Now, we're going to divide into small teams of 2-3 people. Each team will be create a workflow scenario, then shape a possible pipeline to help the team while fitting into the team's given restrictions.

Each team might want to use some of the scenarios from the previous section, but please feel encouraged to shape a story that maps most closely to how you work on a daily basis.

## Team Activity Part 1: Create a scenario

1. Write out the problem in an issue in the class repository. This can include general statements, specific technical limitations or needs, imagined user testimonies, and whatever else you think will paint a detailed picture. Be as specific as possible, noting existing technologies, team size, and end goals.
2. Discuss (and document within the issue) the following questions as they pertain to your chosen example. These should start from a "what we DID have" perspective.
    - Do you have examples of the types of repositories you have?
    - What tooling is used outside of version control?
    - What is the regulation or auditing process like?
    - How many repositories and developers are currently involved?
    - How many developers are on each team?
    - How closely do development teams work with other teams, like infrastructure, dev-ops, QA, etc?

## Team Activity Part 2: Brainstorm solutions

1. Reflect on the process, and prepare to advise. Based on the first part of the activity, brainstorm possible solutions that could address the concerns of the group. Some questions that might help with concrete decisions include:
    - How many organization accounts should you have? How many do you have?
    - How do you decide when to create a new repository?
    - What GitHub features would you want to heavily utilize?
    - What administrative settings could be arranged to be helpful?
    - How is this change a part of a bigger picture in process and platform

    improvement?
    - How will culture be affected by software changes?
2. What GitHub settings will make sense for this group? Consider if they have the ability to change multiple tools, or if they only have the ability to make small changes at a time. Some potential GitHub tools to consider are
    - GitHub Pages
    - Git Flow vs GitHub Flow
    - Choosing tools in the development pipeline
    - Project Boards & Milestones
    - Templates
    - Avoiding context changing
    - Using Pulse and Graphs
    - Protected branches
    - Avoiding duplicated code and work
3. Keep in mind which tooling would make the most sense for the given scenario. Document the broad picture for a workflow for a team, including their needs for project management, continuous integration, and deployment.

# Team Activity Part 3: Map it all out

Use markdown in pull requests as well as drawing on whiteboards or large paper to map out the workflow solution your team has created. Create this resource in a way that it could be used to reference by other team leads and developers, and shared within the company to answer questions. It's OK if it's not a full and perfect solution, but acknowledging what questions aren't answered within the context of the larger workflow is generally a good idea. Include branching diagrams and deployment/release strategies.

We will share these completed scenarios from start to finish as a group.

# More Resources

- Developers are key to innovation and growth
- Why do we use Source Code Management (SCM)?
- Git + GitHub
    - Lightweight Issue Tracking
    - Social Coding
    - Webhooks
    - Teams and ACLs
    - Web-based file rendering
    - GitHub Pages
    - Contributions
    - Pull Requests
        - GitHub Flow
    - Project Boards
    - Templates
- Integrations and the GitHub Marketplace ✨