Introduction

Completed 100 XP 2 minutes

GitHub is a development platform that enables you to host and review code, manage projects, and build software alongside 50 million developers.

Why is everyone building on GitHub? Because it provides the important DevOps features companies and organizations of all sizes need for their public and private projects. Whether it's planning features, fixing bugs, or collaborating on changes, GitHub is the place where the world's software developers gather to make things. And then make them better.

In this module, you learn to use key GitHub features, including issues, notifications, branches, commits, and pull requests.

Learning objectives

In this module, you will:

- Communicate with the project community in issues
- Manage notifications for project events
- Create branches to manage work in parallel
- Make commits to update project source
- Introduce changes with pull requests
- Deploy a web page to GitHub Pages
- Recognize the differences between Git and GitHub and the roles they play in the software development lifecycle
- Describe a repository fork and how it differs from a clone
- Explain the functionality of repository labels and where to apply them in issues and pull requests

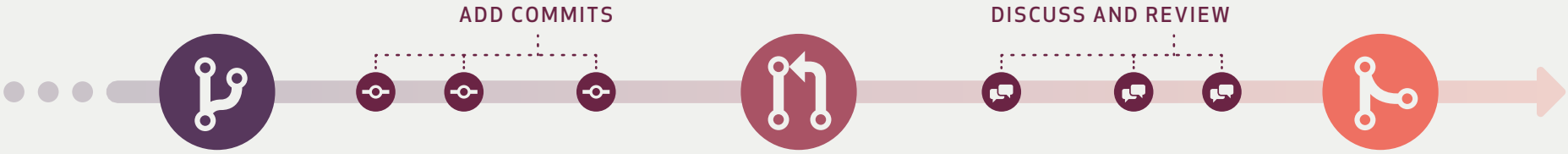
Prerequisites

A GitHub account

Next unit: What is GitHub?

WORK FAST
WORK SMART
THE GITHUB FLOW

The GitHub Flow is a lightweight, branch-based workflow that's great for teams and projects with regular deployments. Find this and other guides at **http://guides.github.com/**.



CREATE A BRANCH

Create a branch in your project where you can safely experiment and make changes.

OPEN A PULL REQUEST

Use a pull request to get feedback on your changes from people down the hall or ten time zones away.

MERGE AND DEPLOY

Merge your changes into your master branch and deploy your code.

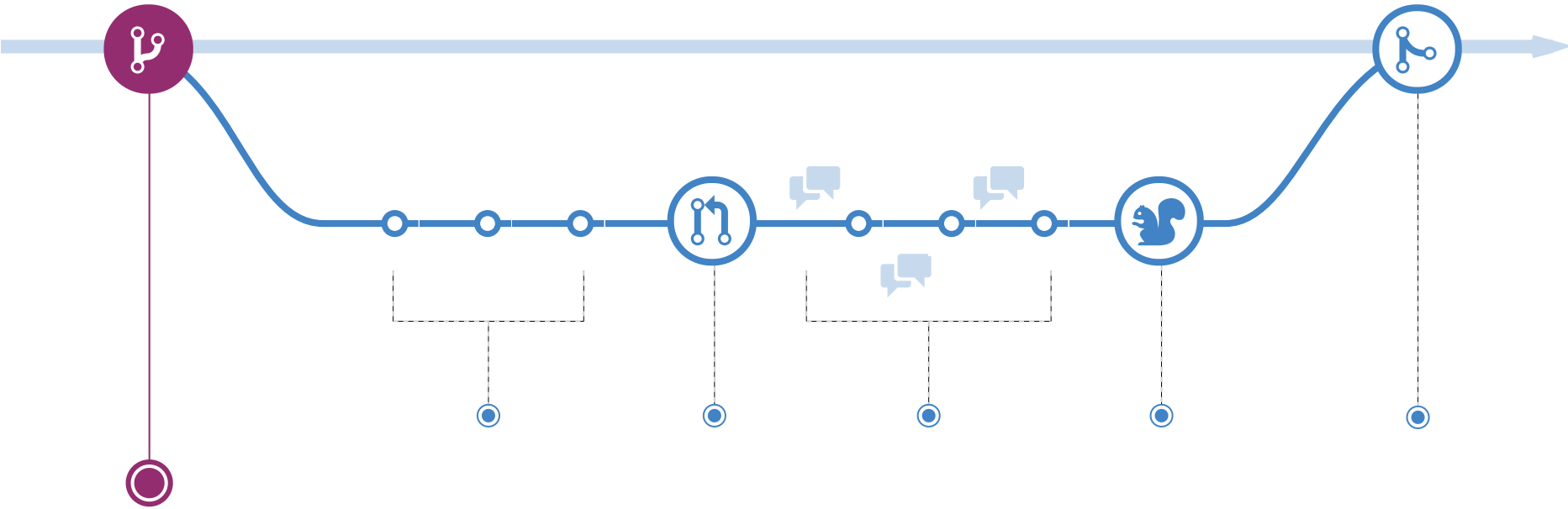
GitHub is the best way to build software together.

GitHub provides tools for easier collaboration and code sharing from any device. Start collaborating with millions of developers today!

Understanding the GitHub flow

5 minute read Download PDF version

GitHub flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. This guide explains how and why GitHub flow works.



Create a branch

When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the `main` branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

ProTip

Branching is a core concept in Git, and the entire GitHub flow is based upon it. There's only one rule: anything in the `main` branch is always deployable.

Because of this, it's extremely important that your new branch is created off of main when working on a feature or a fix. Your branch name should be descriptive (e.g., `refactor-authentication`, `user-content-cache-key`, `make-retina-avatars`), so that others can see what is being worked on.

Last updated July 24, 2020

What is GitHub?

The GitHub flow

In addition to providing a platform for collaborative software development, GitHub also offers a workflow designed to optimize use of its various features. While this unit offers a cursory overview of important platform components, it's recommended that you first review Understanding the GitHub flow.

Git and GitHub

As you work with **Git** and **GitHub**, you may wonder about the difference between the two.

Git is a distributed version control system (DVCS) that allows multiple developers or other contributors to work on a project. It provides a way to work with one or more local branches and push them to a remote repository. Git is responsible for everything GitHub-related that happens locally on your computer. Key features provided by Git include:

- Installed and used on your local machine
- Handles version control
- Supports branching

To learn more about **Git**, see Using common Git commands.

GitHub is a cloud platform that uses Git as its core technology. It simplifies the process of collaborating on projects and provides a website, command-line tools, and overall flow that allows developers and users to work together. GitHub acts as the "remote repository" mentioned previously in the **Git** section.

Key features provided by GitHub include:

- Issues
- Discussions
- Pull requests
- Notifications
- Labels
- Actions
- Forks
- Projects

To learn more about **GitHub**, see Getting started with GitHub.

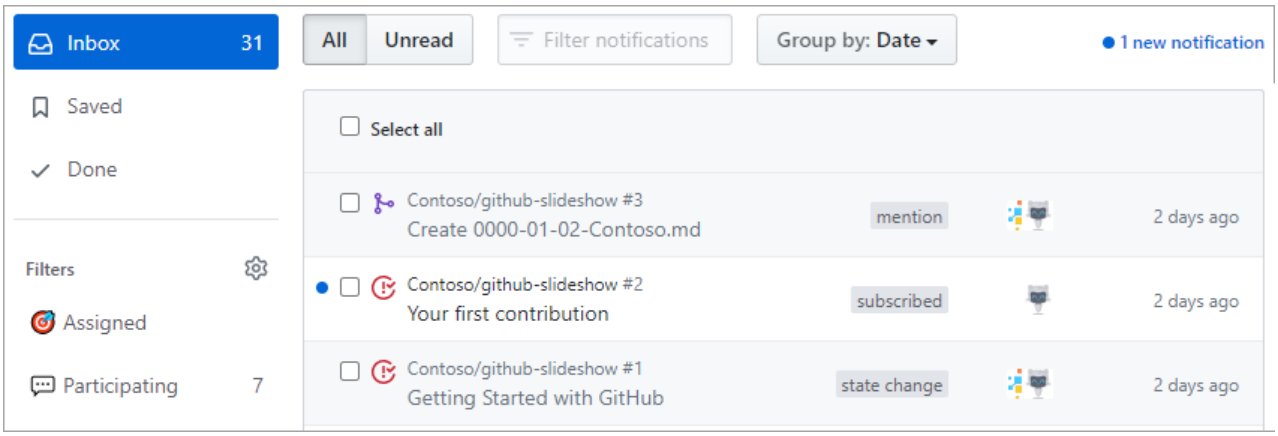
Issues

Issues are where most of the communication between a project's consumers and development team occurs. An *issue* can be created to discuss a broad set of topics, including bug reports, feature requests, documentation clarifications, and more. Once an issue has been created, it can be assigned to owners, labels, projects, and milestones. You can also associate issues with pull requests and other GitHub items to provide future traceability.

To learn more about GitHub Issues, see Mastering Issues.

Notifications

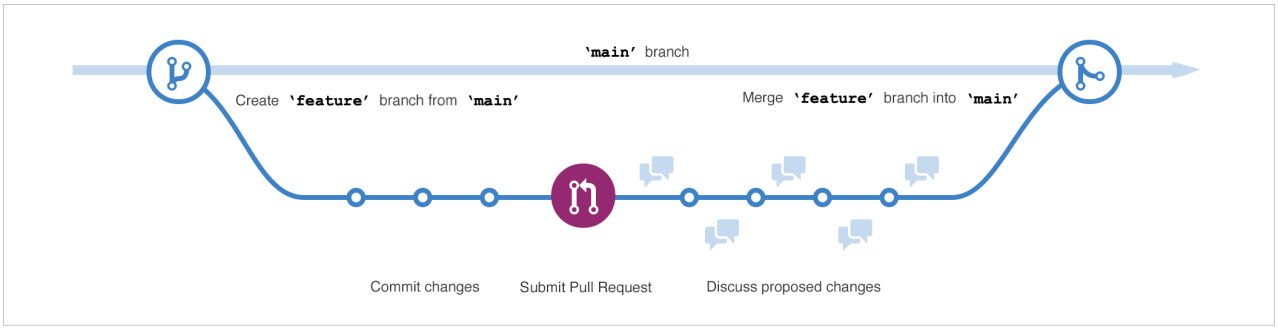
As a collaborative platform, GitHub offers **notifications** for virtually every event that takes place within a given workflow. These notifications can be finely tuned to meet your preferences. For example, you can subscribe to all issue creations and edits on a project, or you can just receive notifications for issues in which you are mentioned. You can also decide whether you receive notifications via email, web & mobile, or both. To keep track of all of your notifications across different projects, use the GitHub Notifications dashboard.



To learn more about GitHub notifications, see Configuring notifications.

Branches

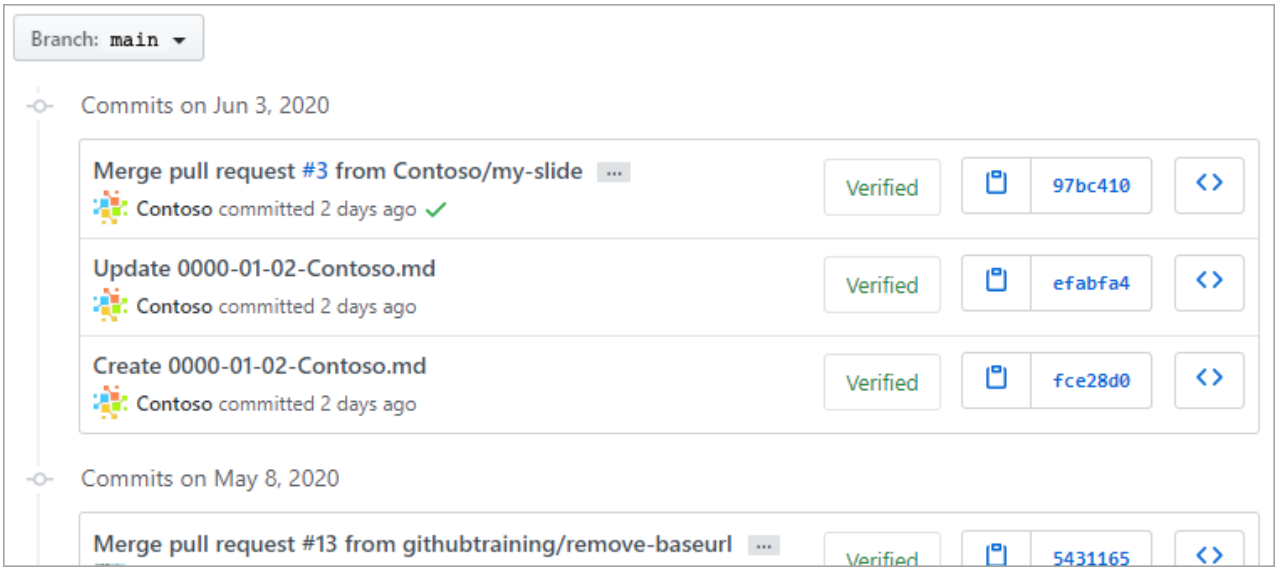
Branches are the preferred way to create changes in the GitHub flow. They provide isolation so that multiple people may simultaneously work on the same code in a controlled way. This model enables stability among critical branches, such as `main`, while allowing complete freedom for developers to commit any changes they need to meet their goals. Once the code from a branch is ready to become part of the `main` branch, it may be merged via pull request.



To learn more about GitHub branches, see About branches.

Commits

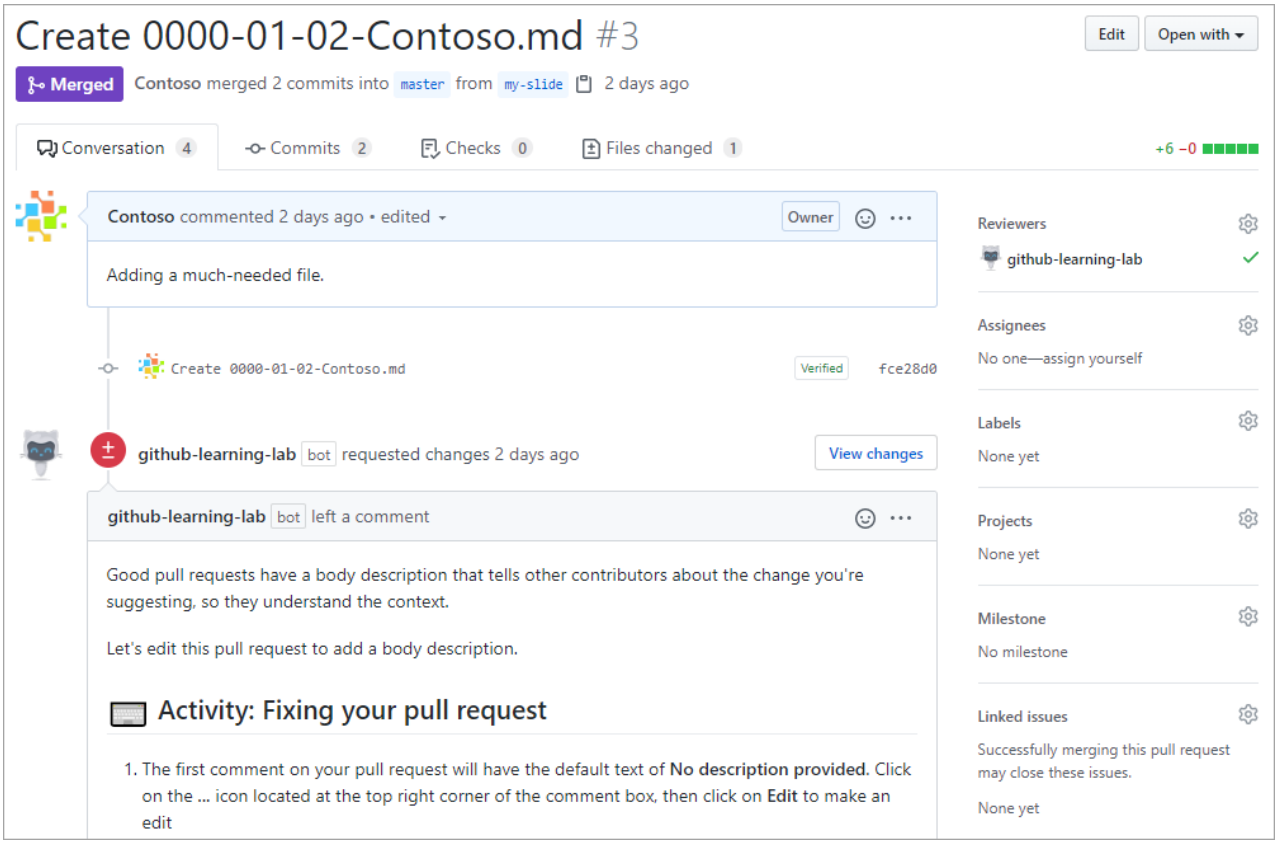
A **commit** is a change to one or more files on a branch. Every time a commit is created, it is assigned a unique ID and tracked, along with the time and contributor. This provides a clear audit trail for anyone reviewing the history of a file or linked item, such as an issue or pull request.



To learn more about GitHub commits, see Committing and reviewing changes to your project.

Pull Requests

A **pull request** is the mechanism used to signal that the commits from one branch are ready to be merged into another branch. The developer submitting the **pull request** will often request one or more reviewers to verify the code and approve the merge. These reviewers have the opportunity to comment on changes, add their own, or use the pull request itself for further discussion. Once the changes have been approved (if approval is required), the pull request's source branch (the compare branch) may be merged in to the base branch.



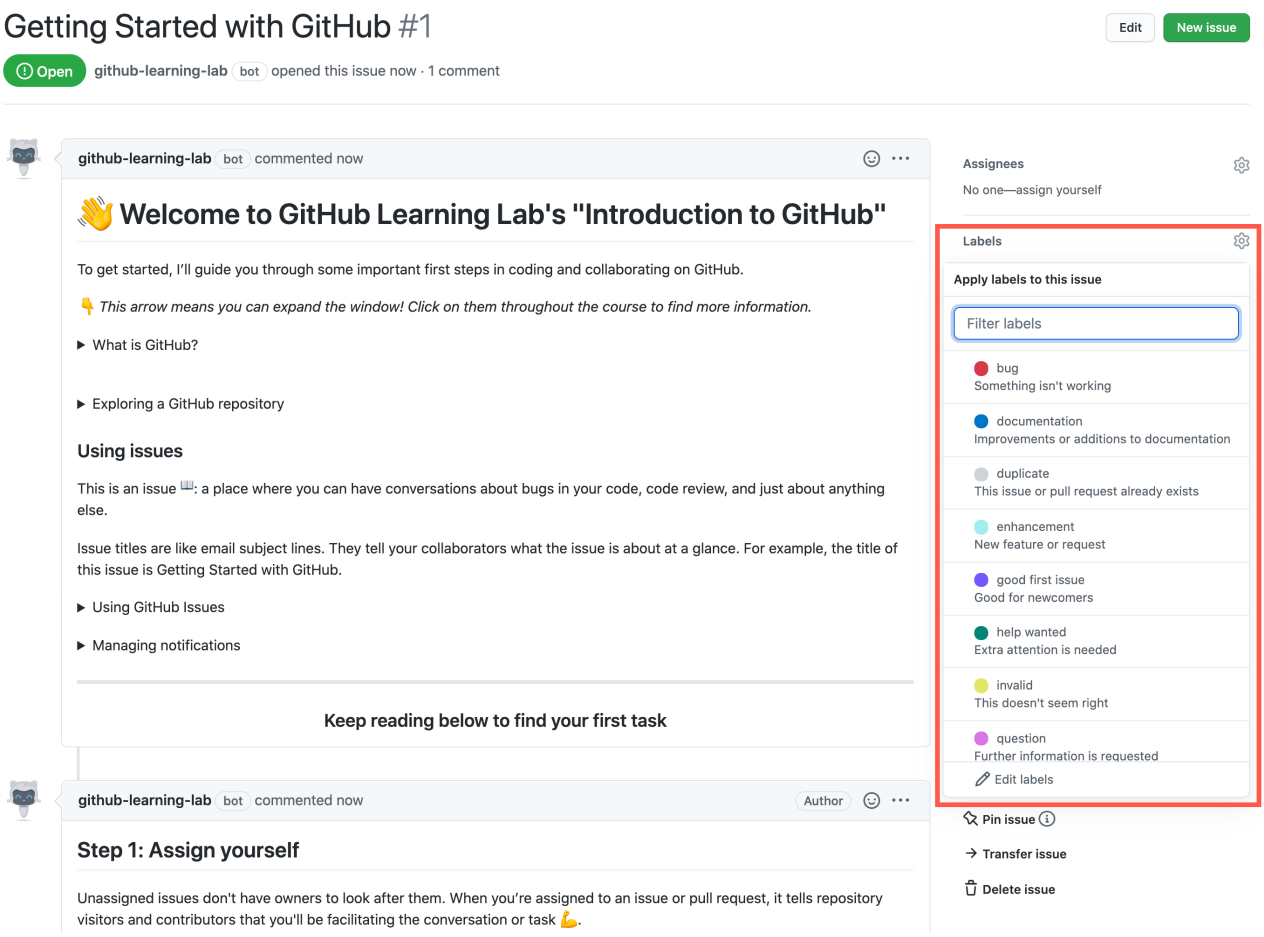
To learn more about GitHub pull requests, see About pull requests.

Labels

Labels provide a way to categorize and organize **issues** and **pull requests** in a repository. As you create a GitHub repository several labels will automatically be added for you and new ones can also be created.

Examples of Labels include:

- bug
- documentation
- duplicate
- help wanted
- enhancement
- question



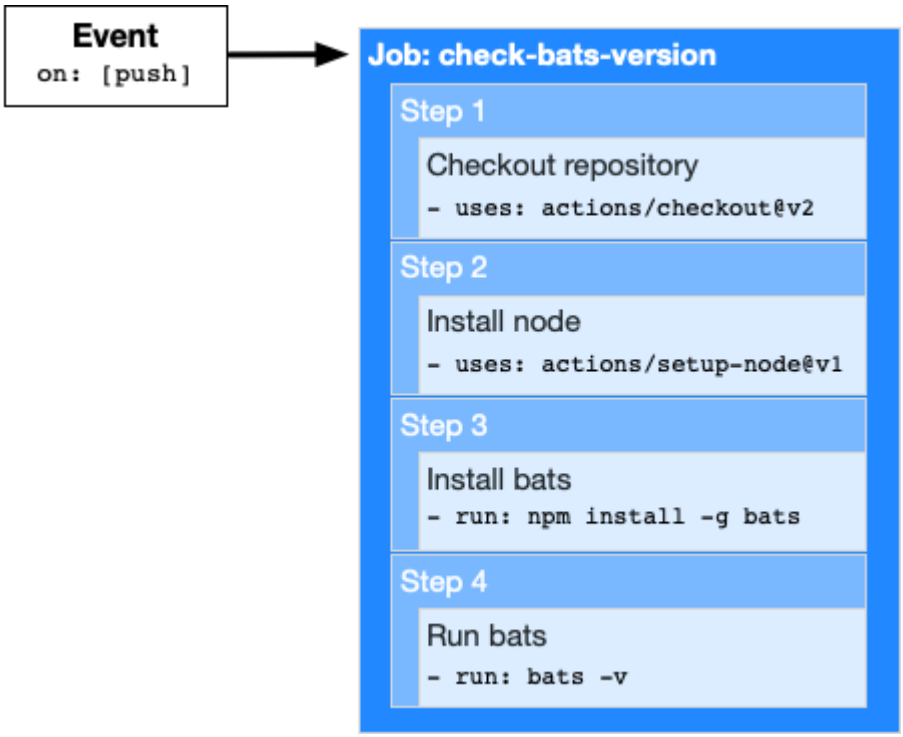
To learn more about GitHub labels see About labels.

Actions

GitHub actions provide task automation and workflow functionality in a repository. Actions can be used to streamline processes in your software development lifecycle and implement continuous integration and continuous deployment (CI/CD).

GitHub Actions are composed of the following components:

- **Workflows**: Automated processes added to your repository.
- **Events**: An activity that triggers a workflow.
- **Jobs**: A set of steps that execute on a runner.
- **Steps**: A task that can run one or more commands (actions).
- **Actions**: Standalone commands that can be combined into steps. Multiple steps can be combined to create a job.
- **Runners**: Server that has the GitHub Actions runner application installed.



To learn more about GitHub actions see Introduction to GitHub Actions.

Cloning and forking

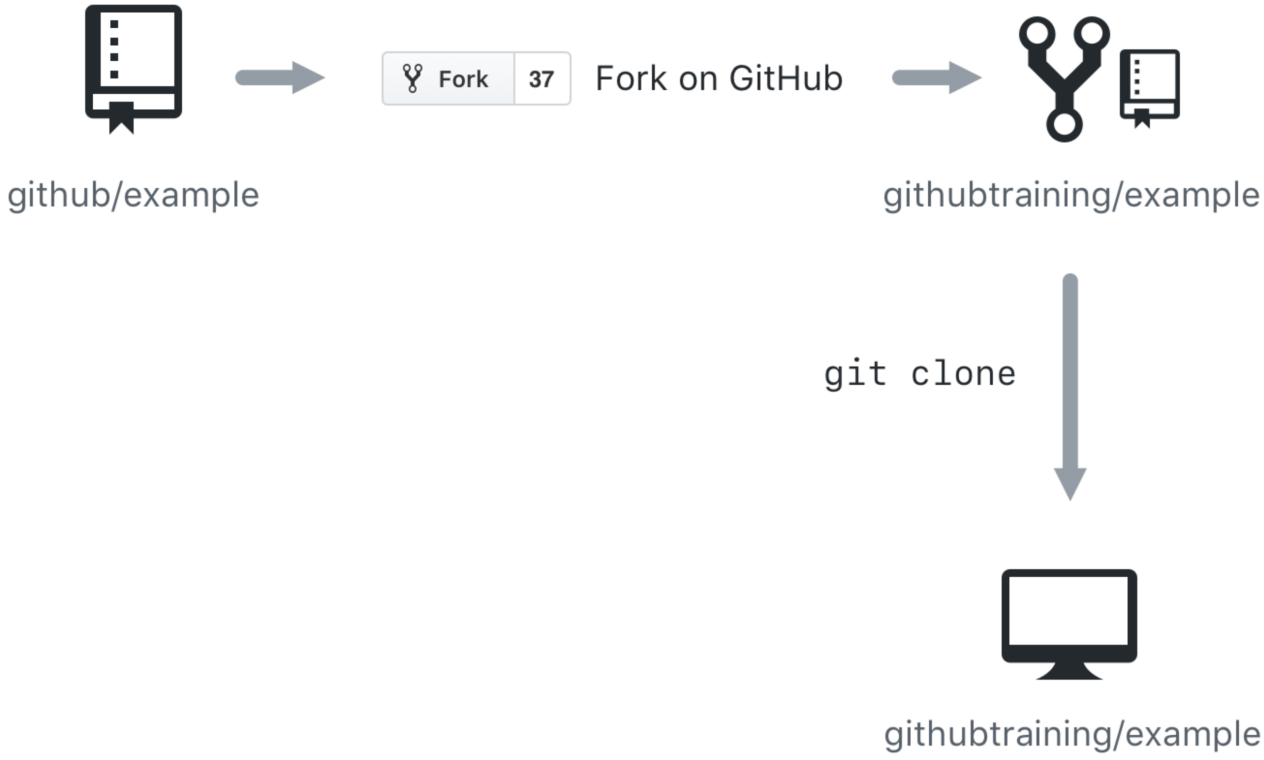
GitHub provides multiple ways to copy a repository so that you can work on it.

- **Cloning a Repository** - Cloning a repository will make a copy of the repository and its history on your local machine. If you have write access to the repository you can push changes from your local machine to the remote repository (called the **origin**) as they're completed. To clone a repository you can use the `git clone [url]` command or the GitHub CLI's `gh repo clone [url]` command.
- **Forking a Repository** - **Forking** a repository makes a copy of the repository in your GitHub account. The parent repository is referred to as the **upstream** while your forked copy is referred to as the **origin**. Once you've forked a repository into your GitHub account you can **clone** it to your local machine. Forking allows you to freely make changes to a project without affecting the original **upstream** repository. To contribute changes back to the **upstream** repository you create a **pull request** from your forked repository. You can also run `git` commands to ensure that your local copy stays synced with the **upstream** repository.

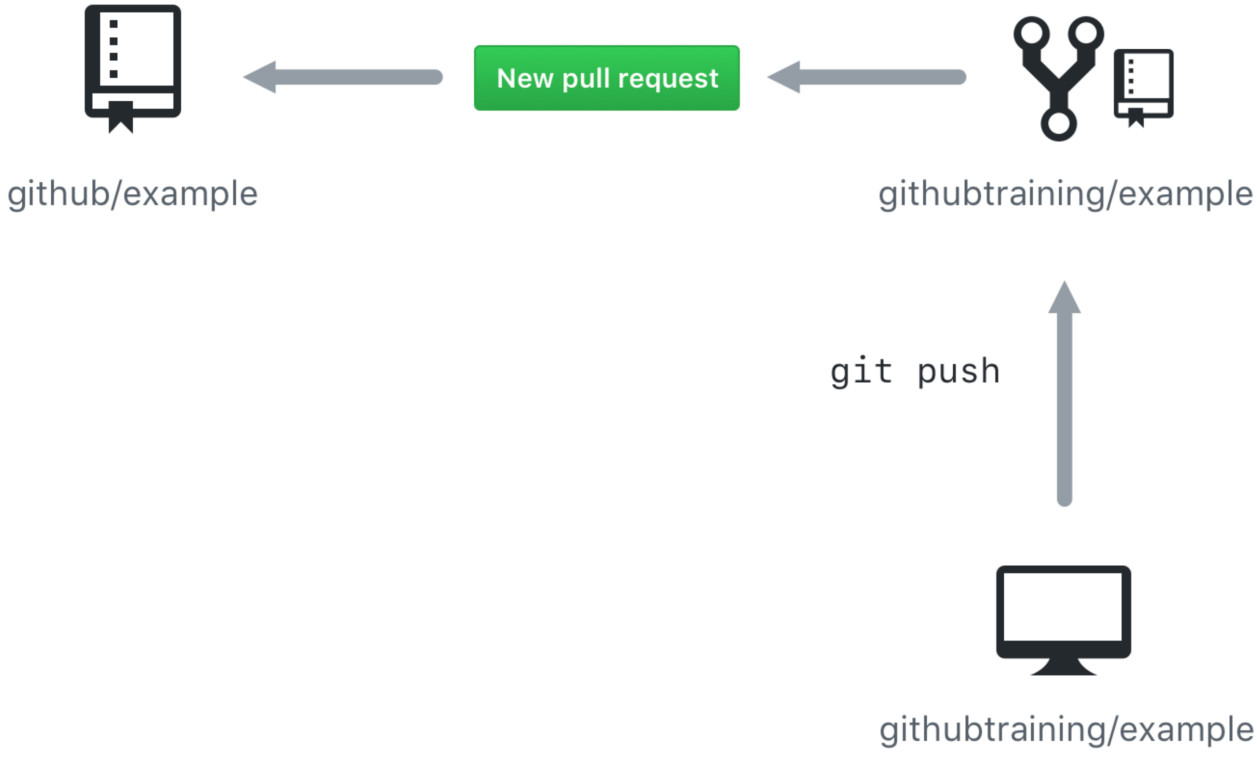
When would you clone a repository versus fork a repository? If you're working with a repository and have write access you can clone it to your local machine. From there you can make modifications and push your changes directly to the **origin** repository.

If you need to work with a repository created by another owner such as `github/example` and don't have write access, you can fork the repository into your GitHub account, and then clone the fork to your local machine. To see this visually, let's assume that your GitHub account is called `githubtraining` for this example. Using the

GitHub website you can fork `githubtraining` or any other examples into your account. From there you can clone the forked version of the repository to your local machine. These steps are shown in the following image.



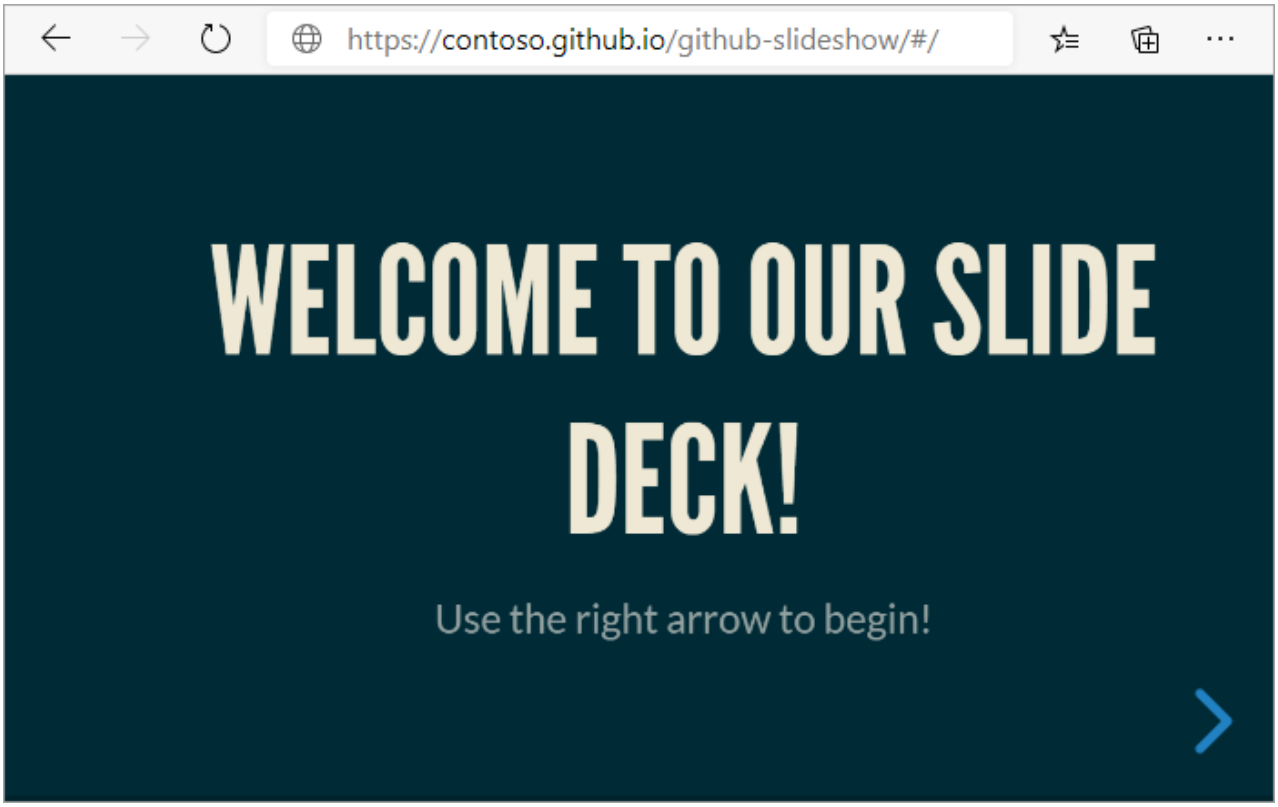
Changes can be made to your local copy of `githubtraining/example` and then pushed back to your remote **origin** repository (`githubtraining/example`). The changes can then be submitted to the `github/example` **upstream** repository using a **pull request** as shown next.



To learn more, see Fork a repo.

GitHub Pages

GitHub Pages is a hosting engine that's built right into your GitHub account. By following a few conventions, and enabling the feature, you can build your own static site generated from HTML and markdown code pulled directly from your repository.



To learn more, see GitHub Pages.

Next unit: Exercise - A guided tour of GitHub

Continue

Mastering Markdown

guides.github.com/features/mastering-markdown

Markdown is a lightweight and easy-to-use syntax for styling all forms of writing on the GitHub platform.

What you will learn:

- How the Markdown format makes styled collaborative editing easy
- How Markdown differs from traditional formatting approaches
- How to use Markdown to format text
- How to leverage GitHub's automatic Markdown rendering
- How to apply GitHub's unique Markdown extensions

What is Markdown?

Markdown is a way to style text on the web. You control the display of the document; formatting words as bold or italic, adding images, and creating lists are just a few of the things we can do with Markdown. Mostly, Markdown is just regular text with a few non-alphabetic characters thrown in, like `#` or `*`.

You can use Markdown most places around GitHub:

- Gists
- Comments in Issues and Pull Requests
- Files with the `.md` or `.markdown` extension

For more information, see "Writing on GitHub" in the *GitHub Help*.

Examples

```
It's very easy to make some words **bold** and other words *italic* with Markdown.
You can even [link to Google!](http://google.com)
```

It's very easy to make some words **bold** and other words *italic* with Markdown. You can even link to Google!

Syntax guide

Here's an overview of Markdown syntax that you can use anywhere on GitHub.com or in your own text files.

Headers

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

Emphasis

```
*This text will be italic*
_This will also be italic_

**This text will be bold**
__This will also be bold__

_You **can** combine them_
```

Lists

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
1. Item 2
1. Item 3
   1. Item 3a
   1. Item 3b
```

Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

Links

```
http://github.com - automatic!
[GitHub](http://github.com)
```

Blockquotes

```
As Kanye West said:

> We're living the future so
> the present is our past.
```

Inline code

```
I think you should use an
`<addr>` element here instead.
```

GitHub Flavored Markdown

GitHub.com uses its own version of the Markdown syntax that provides an additional set of useful features, many of which make it easier to work with content on GitHub.com.

Note that some features of GitHub Flavored Markdown are only available in the descriptions and comments of Issues and Pull Requests. These include @mentions as well as references to SHA-1 hashes, Issues, and Pull Requests. Task Lists are also available in Gist comments and in Gist Markdown files.

Syntax highlighting

Here's an example of how you can use syntax highlighting with GitHub Flavored Markdown:

```
```javascript
function fancyAlert(arg) {
  if(arg) {
    $.facebox({div:'#foo'})
  }
}
```
```

You can also simply indent your code by four spaces:

```
    function fancyAlert(arg) {
      if(arg) {
        $.facebox({div:'#foo'})
      }
    }
```

Here's an example of Python code without syntax highlighting:

```
def foo():
    if not bar:
        return True
```

Task Lists

```
- [x] @mentions, #refs, [links](), **formatting**, and <del>tags</del> supported
- [x] list syntax required (any unordered or ordered list supported)
- [x] this is a complete item
- [ ] this is an incomplete item
```

If you include a task list in the first comment of an Issue, you will get a handy progress indicator in your issue list. It also works in Pull Requests!

Tables

You can create tables by assembling a list of words and dividing them with hyphens `-` (for the first row), and then separating each column with a pipe `|`:

```
First Header | Second Header
------------ | -------------
Content from cell 1 | Content from cell 2
Content in the first column | Content in the second column
```

Would become:

First Header	Second Header
Content from cell 1	Content from cell 2
Content in the first column	Content in the second column

SHA references

Any reference to a commit's SHA-1 hash will be automatically converted into a link to that commit on GitHub.

```
16c999e8c71134401a78d4d46435517b2271d6ac
mojombo@16c999e8c71134401a78d4d46435517b2271d6ac
mojombo/github-flavored-markdown@16c999e8c71134401a78d4d46435517b2271d6ac
```

Issue references within a repository

Any number that refers to an Issue or Pull Request will be automatically converted into a link.

```
#1
mojombo#1
mojombo/github-flavored-markdown#1
```

Username @mentions

Typing an `@` symbol, followed by a username, will notify that person to come and view the comment. This is called an "@mention", because you're *mentioning* the individual. You can also @mention teams within an organization.

Automatic linking for URLs

Any URL (like `http://www.github.com/`) will be automatically converted into a clickable link.

Strikethrough

Any word wrapped with two tildes (like `~~this~~`) will appear crossed out.

Emoji

GitHub supports emoji!

To see a list of every image we support, check out the Emoji Cheat Sheet.

Last updated Jan 15, 2014

nodejs / node-v0.x-archive Public archive

github.com/nodejs/node-v0.x-archive/issues

nodejs

This repository has been archived by the owner. It is now read-only.

Linking with fcgi_stdio.h ?

#25909 opened on Aug 29, 2015 by kevmuret

2

Provide a portable prebuilt Windows zip or tarball download

#25907 opened on Aug 29, 2015 by pombredanne

3

vm: node.js v0.12: Mangled exceptions generated in the evalmachine

#25904 opened on Aug 28, 2015 by n-riesco

1

child_process.spawn fails on Windows given a space in both the command and an argument

#25895 opened on Aug 25, 2015 by smrq

10

Cannot compile with GCC v4.9.2

build

#25890 opened on Aug 24, 2015 by Sydney-09

3

child process cannot resolve binaries via PATH with tilde character

#25882 opened on Aug 19, 2015 by h2non

https server dynamic secure context

#25880 opened on Aug 19, 2015 by petkaantonov

1

Include iowait and steal in os.cpus()

#25874 opened on Aug 18, 2015 by synologic

ECONNRESET using https to TLSv1.2 site

#25872 opened on Aug 18, 2015 by lmarkus

node does not propagate the module.paths to the require('module')

#25861 opened on Aug 15, 2015 by wyattbiker

1

Error: connect EADDRINUSE

#25836 opened on Aug 11, 2015 by jbfm21

1

Incorrect loop->active_handles count when nested setTimeout calls have same value

timer v0.12

#25831 opened on Aug 10, 2015 by abbr

1

3

Can't install on Ubuntu 14.04, no valid OpenPGP data found.

#25828 opened on Aug 9, 2015 by santosh

2

TLS: OCSPRequest - certificate and issuer are null when pfx used

#25820 opened on Aug 7, 2015 by Codelica

Linux APT contains obsolete version

build

#25817 opened on Aug 6, 2015 by inf3rno

2

Redirecting stdout to another TTY has no effect

#25809 opened on Aug 6, 2015 by wleslie

make install doesn't expand ~ on Linux

#25806 opened on Aug 5, 2015 by simonbcn

1

Assertion `parser->current_buffer_.IsEmpty()' failed.

#25802 opened on Aug 3, 2015 by fmsouza

3

Make ECDH optional

crypto defer-to-converged feature-request

#25798 opened on Aug 2, 2015 by blshkv

[timers] setImmediate executes after setTimeout

timer v0.10 v0.12

#25788 opened on Jul 31, 2015 by a0viedo

6

TLS feature request: Allow trusting different CAs than the default bundle

defer-to-converged feature-request tls

#25785 opened on Jul 31, 2015 by kelseyfrancis

2

net::Server.unref() failed on cluster mode

cluster iojs-backport net S-confirmed-bug

#25782 opened on Jul 30, 2015 by kyriosli

2

Signal handlers are not removed/reverted with process.removeAllListeners([signal])

events iojs-backport process S-confirmed-bug

#25781 opened on Jul 30, 2015 by chjj

1

SIGTRAP when using exec

#25771 opened on Jul 27, 2015 by martlin2cz

2

Update docs for os.platform() to enumerate the return values and conditions

defer-to-converged doc feature-request

#25769 opened on Jul 26, 2015 by tomprogers

12

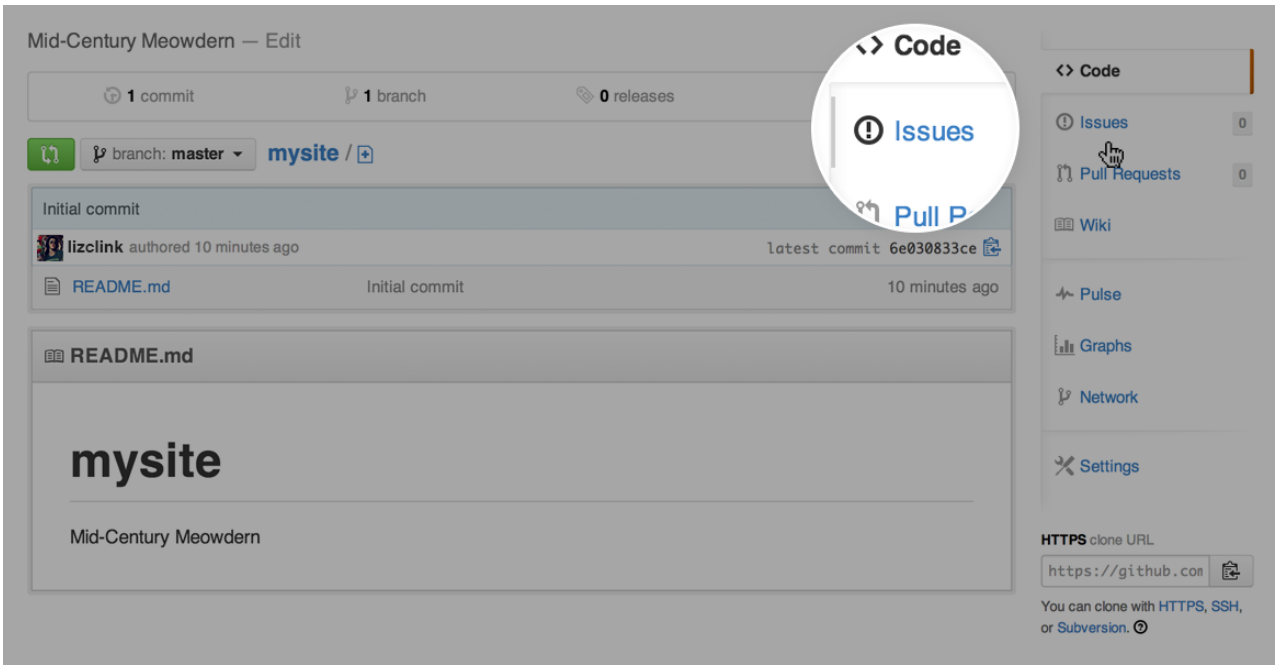
ProTip! What's not been updated in a month: updated:<2021-09-10.

Mastering Issues

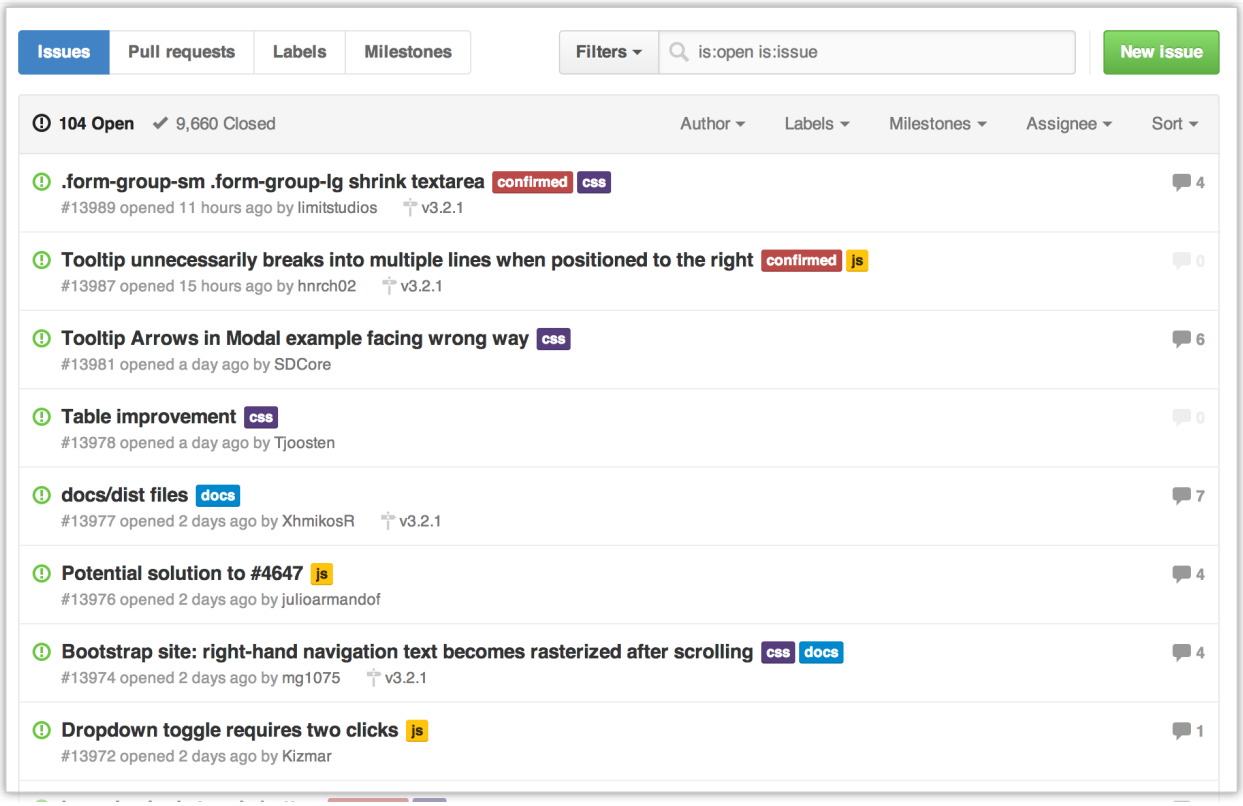
guides.github.com/features/issues

10 minute read

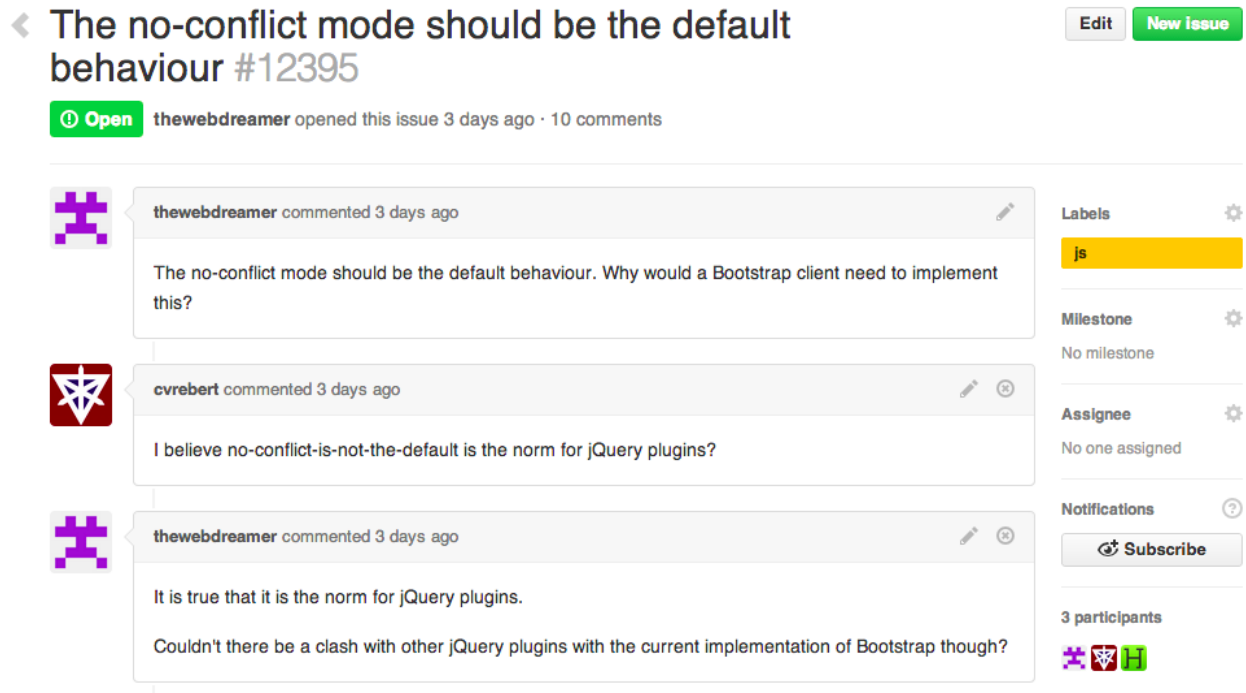
Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. They're kind of like email—except they can be shared and discussed with the rest of your team. Most software projects have a bug tracker of some kind. GitHub's tracker is called **Issues**, and has its own section in every repository.



For example, let's take a look at Bootstrap's Issues section:



GitHub's issue tracking is special because of our focus on collaboration, references, and excellent text formatting. A typical issue on GitHub looks a bit like this:



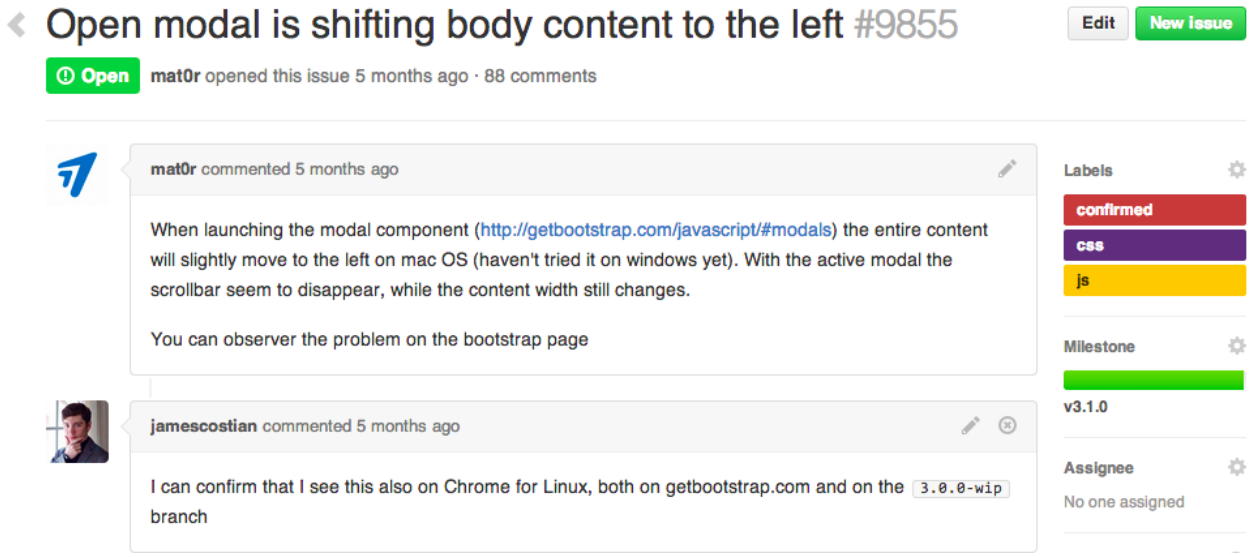
- A **title** and **description** describe what the issue is all about.
- Color-coded **labels** help you categorize and filter your issues (just like labels in email).

- A **milestone** acts like a container for issues. This is useful for associating issues with specific features or project phases (e.g. *Weekly Sprint 9/5-9/16* or *Shipping 1.0*).
- One **assignee** is responsible for working on the issue at any given time.
- **Comments** allow anyone with access to the repository to provide feedback.

Milestones, Labels, and Assignees

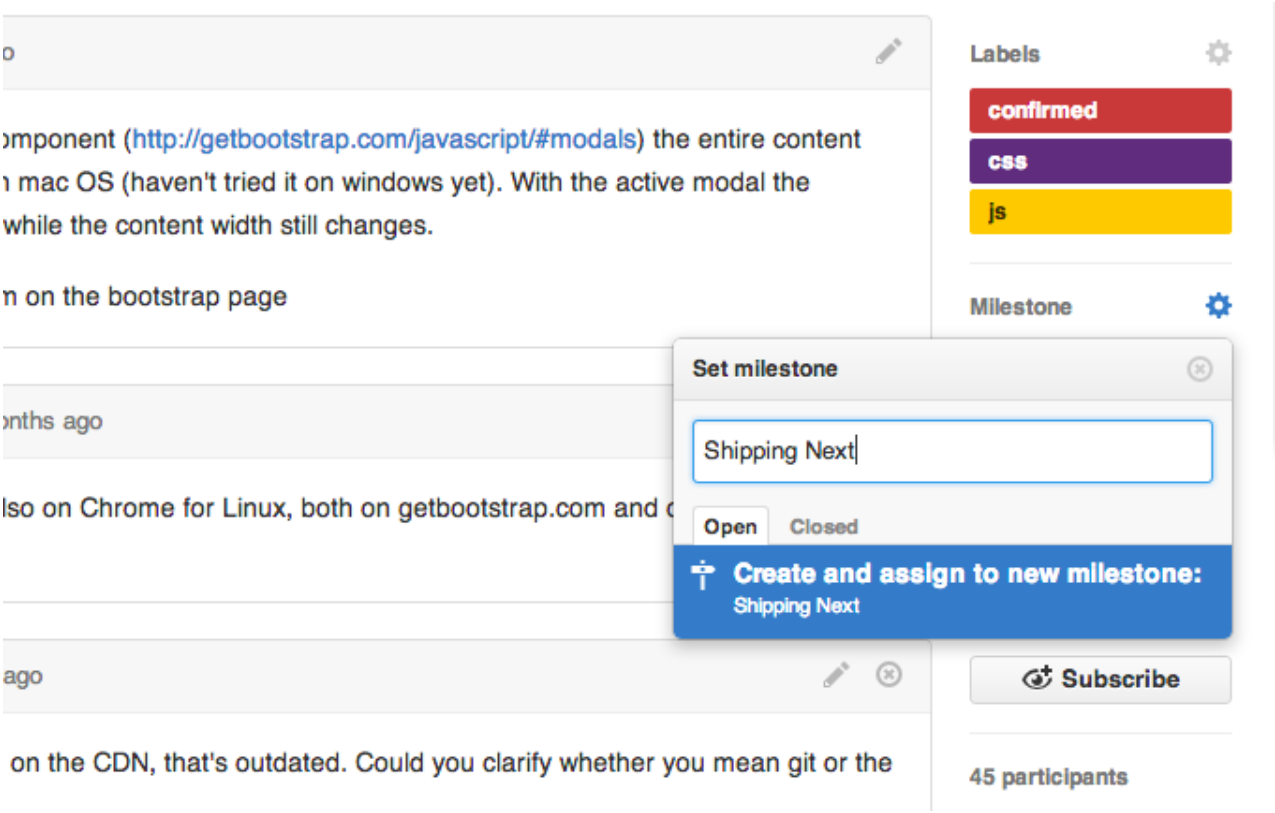
Once you've collected a lot of issues, you may find it hard to find the ones you care about. **Milestones**, **labels**, and **assignees** are great features to filter and categorize issues.

You can change or add a milestone, an assignee, and labels by clicking their corresponding gears in the sidebar on the right.



If you don't see edit buttons, that's because you don't have permission to edit the issue. You can ask the repository owner to add you as a collaborator to get access.

Milestones

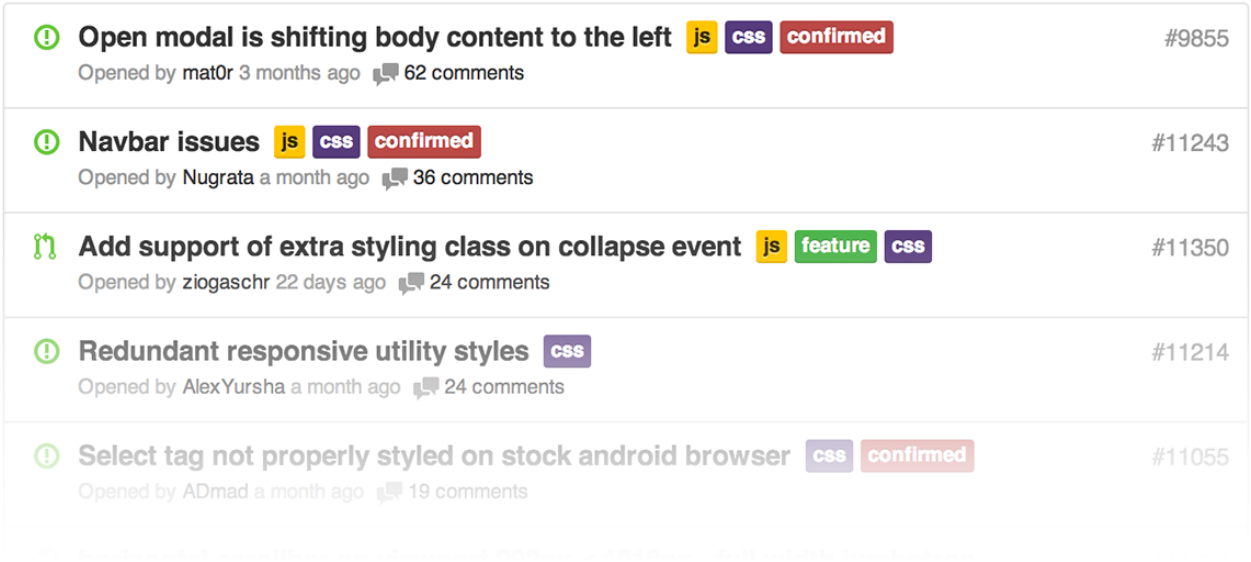


Milestones are groups of issues that correspond to a project, feature, or time period. People use them in many different ways in software development. Some examples of milestones on GitHub include:

- **Beta Launch** — File bugs that you need to fix before you can launch the beta of your project. It's a great way to make sure you aren't missing anything.
- **October Sprint** — File issues that you'd like to work on in October. A great way to focus your efforts when there's a lot to do.
- **Redesign** — File issues related to redesigning your project. A great way to collect ideas on what to work on.

Labels

Labels are a great way to organize different types of issues. Issues can have as many labels as you want, and you can filter by one or many labels at once.

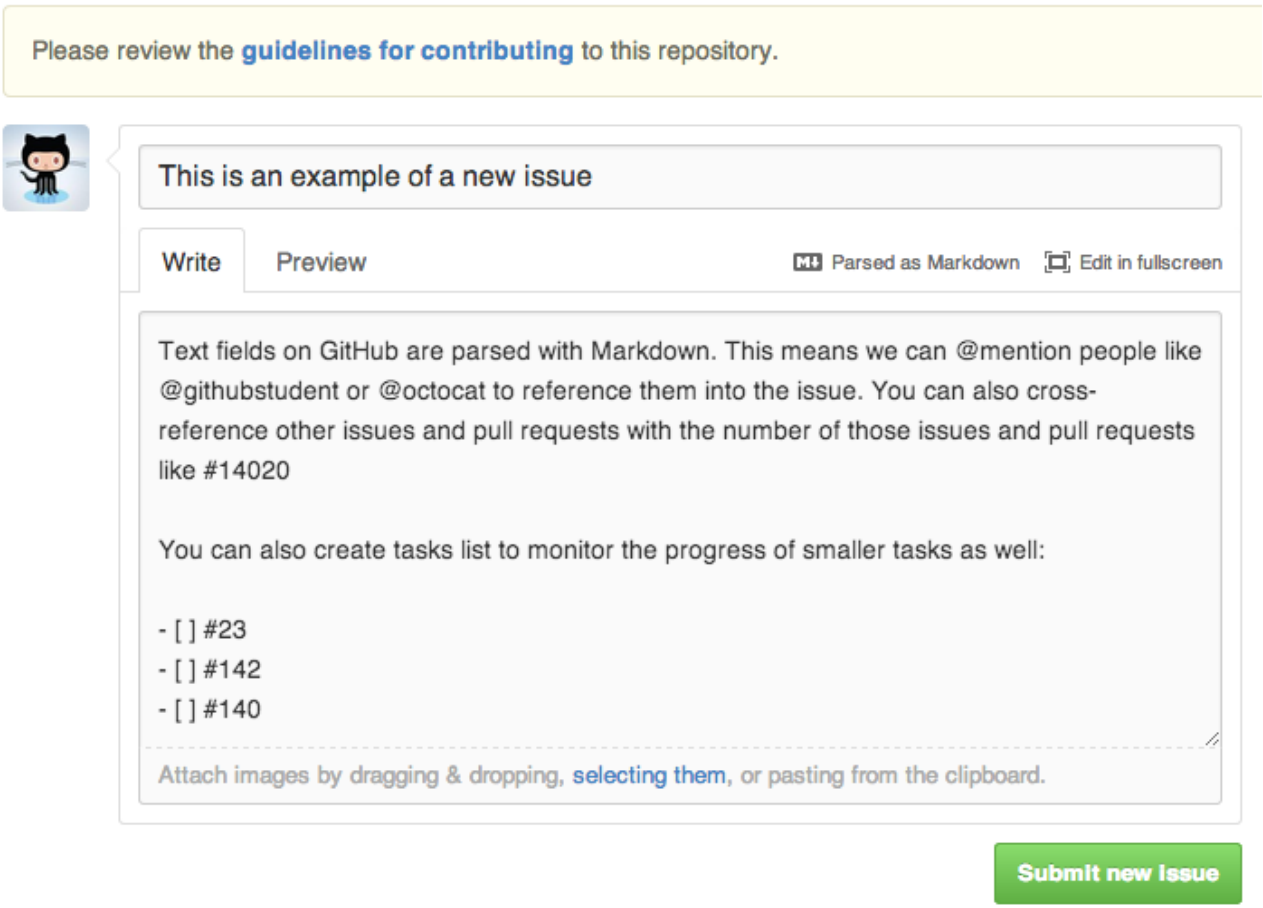


Assignees

Each issue can have an assignee — one person that's responsible for moving the issue forward. Assignees are selected the same way milestones are, through the grey bar at the top of the issue.

Notifications, @mentions, and References

By using @mentions and references inside of Issues, you can notify other GitHub users & teams, and cross-connect issues to each other. These provide a flexible way to get the right people involved to resolve issues effectively, and are easy to learn and use. They work across all text fields on GitHub — they're a part of our text formatting syntax called GitHub Flavored Markdown.

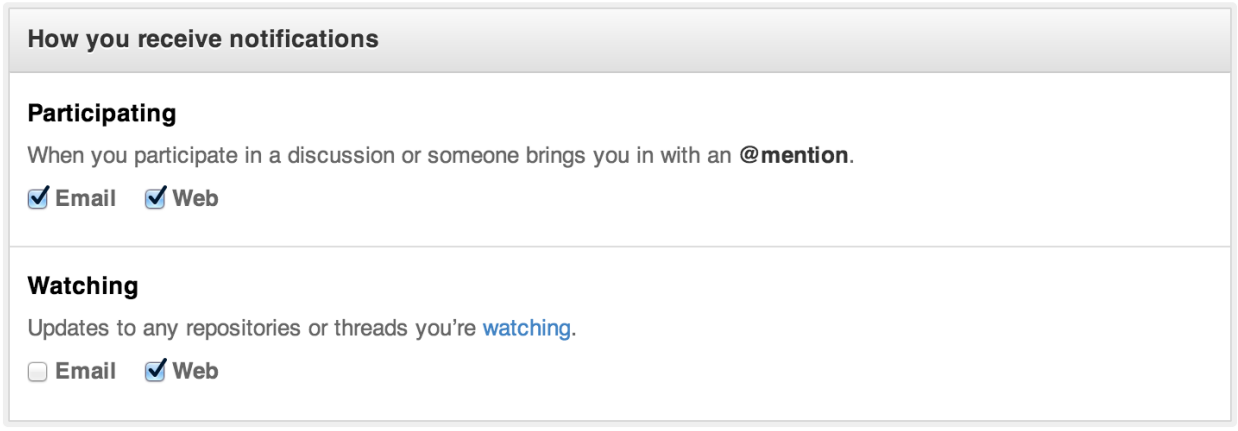


If you'd like to learn more, have a look at **Mastering Markdown**.

Notifications

Notifications are GitHub's way to keep up to date with your Issues. You can use them to find out about new issues on repositories, or just to know when someone needs your input to move forward on an issue.

There are two ways to receive notifications: via email, and via the web. You can configure how you receive notifications in your settings. If you plan on receiving a lot of notifications, we like to recommend that you receive web + email notifications for **Participating** and web notifications for **Watching**.



With these settings, you receive emails when people specifically mention you, then visit the web-based interface to keep up to date with repositories you're interested in.

You can access your notifications through the notifications screen. This screen is nice for scanning many notifications at once and marking them as read or muting the thread. Try using keyboard shortcuts to speed up your workflow here — press `?` on the page to see which shortcuts are available.

Muted threads won't show up as unread again until you are specifically @mentioned again. This makes muting a great strategy for threads that you have little interest in (perhaps a sub-system that you aren't familiar with). If you mark an issue as read, it will stay that way until someone comments on the thread again.

GitHub also syncs read/unread status for email notifications — if you read a notification in your email client, it will be marked as read in the web-based interface (make sure you allow your email client to display images if you'd like this functionality).

@mentions

@mentions are the way that we reference other GitHub users inside of GitHub Issues. Inside of the description or any comment of the issue, include the @username of another GitHub user to send them a notification. This works very similar to how Twitter uses @mentions.

We like to use the `/cc` syntax (an abbreviation for carbon copy) to include people in issues:

> It looks like the new widget form is broken on Safari. When I try and create the widget, Safari crashes. This is reproducible on 10.8, but not 10.9. Maybe a browser bug?
>
> /cc @kneath @jresig

This works great if you know the specific users to include, but many times we're working across teams and don't really know who might be able to help us. @mentions also work for Teams within your GitHub organization. If you create a Team called *browser-bugs* under the @acmeinc organization, you can reference the team with @mentions:

> /cc @acmeinc/browser-bugs

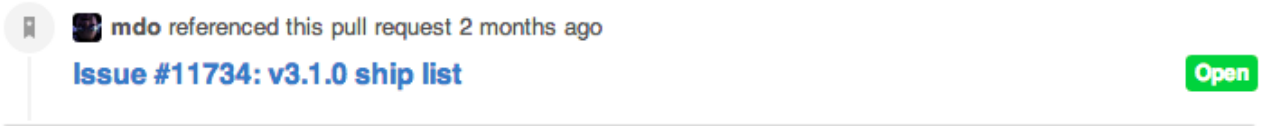
This will send notifications to every member of the browser-bugs team.

References

Often times issues are dependent on other issues, or at least relate to them and you'd like to connect the two. You can reference issues by typing in a hashtag plus the issue number.

> Hey @kneath, I think the problem started in #42

When you do this, we'll create an event inside of issue #42 that looks something like this:



Issue in another repository? Just include the repository before the name like `kneath/example-project#42`.

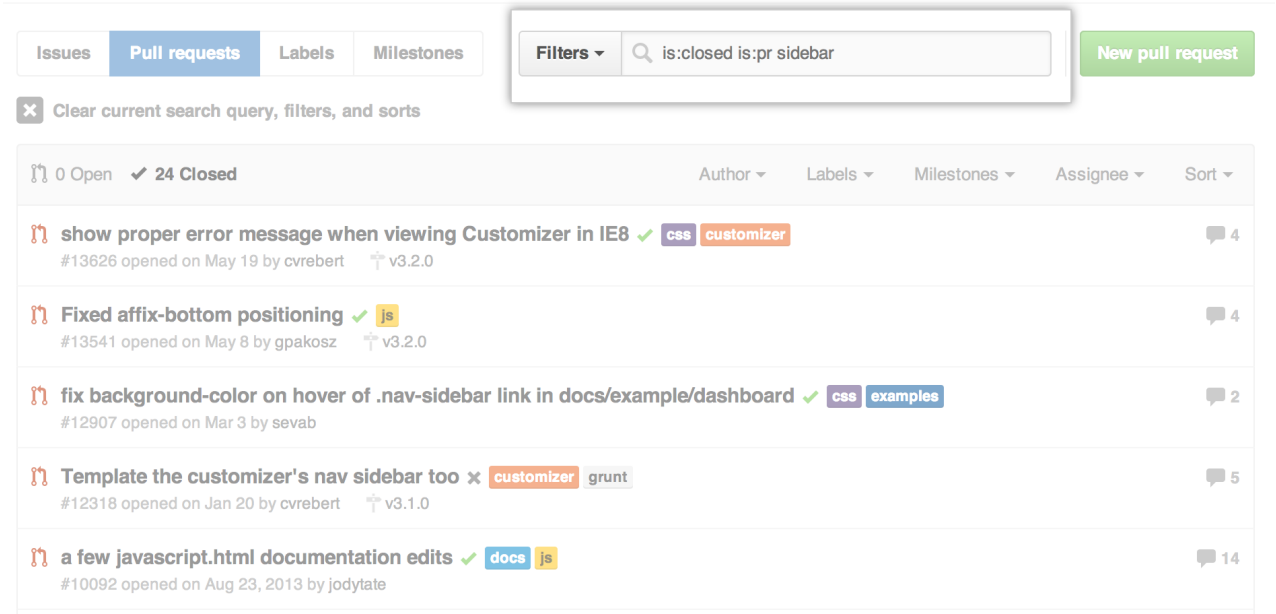
One of the more interesting ways to use GitHub Issues is to reference issues directly from commits. Include the issue number inside of the commit message.

By prefacing your commits with "Fixes", "Fixed", "Fix", "Closes", "Closed", or "Close" when the commit is merged into main, it will also automatically close the issue.

References make it possible to deeply connect the work being done with the bug being tracked, and are a great way to add visibility into the history of your project.

Search

At the very top of each page is a search box that lets you search through issues.



You can scope search results by:

- Keyword, such as, all issues mentioning the sidebar
- State, such as, all issues mentioning the sidebar that are closed
- Assignee, such as, all issues mentioning the sidebar that were assigned to @mdo

Our Help article on searching Issues can show you other ways to search: using created/updated dates, labels, authors, comment counts, by repository owner, and more.

Overviews & Reports

Outside of the Issues section, there are two other pages that help summarize what's going on with Issues across your repository and across all of your repositories.

The Issue Dashboard

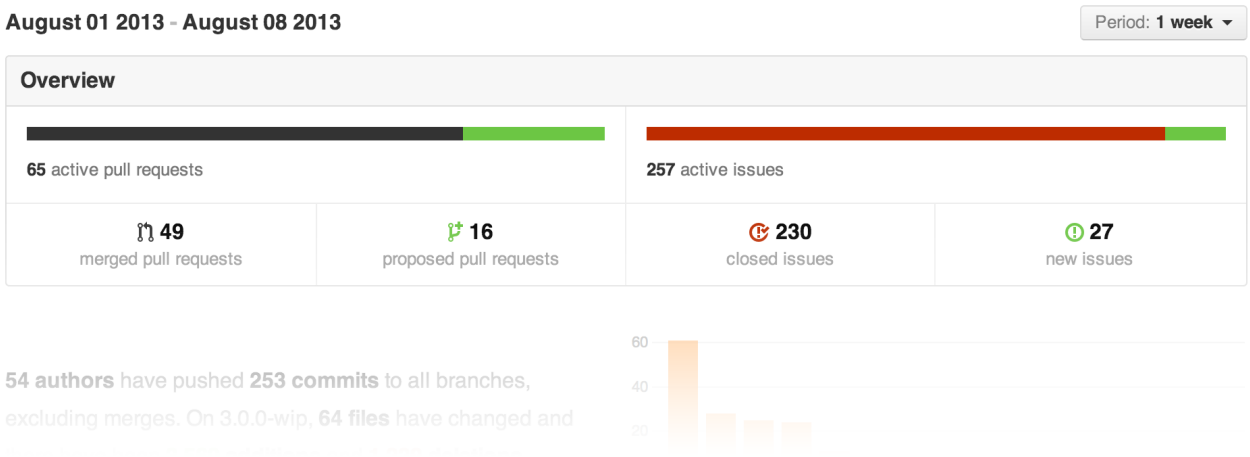
If you're looking for a broader listing of all of your issues across many projects, the Issues Dashboard can be a great tool. The dashboard works very similar to the issues section, but collects issues differently:

- All issues in repositories you own and collaborate on
- Issues assigned to you
- Issues you've created

If you use organizations, each one has its own Issues dashboard that separates out Issues within the organization.

Pulse

Underneath each repository is a section called **Pulse** — Pulse is a snapshot of everything that's happened in the repository in the past week (or day, or past 3 months, etc).



It's a great way to catch up with repositories when you've been away and don't want the granularity notifications offer when watching a repository.

Other Uses for Issues

Issues are great for tracking all kinds of things — and GitHub is a great place to easily share and collaborate on your issues. Here's some of our favorites:

- Bug tracker for your open source projects
- Request for recipes (maybe you have a good **gluten-free pizza dough recipe?**)

Fin

Now congratulate yourself — that was a lot to read! Issue management is one of the most powerful tools at any developer's disposal. I guess all that's left is to actually fix the bugs now.

Last updated July 24, 2020

Managing labels

You can classify issues, pull requests, and discussions by creating, editing, applying, and deleting labels.

Members of an enterprise with managed users can only make changes in repositories that are part of their enterprise.

About labels

You can manage your work on GitHub by creating labels to categorize issues, pull requests, and discussions. You can apply labels in the repository the label was created in. Once a label exists, you can use the label on any issue, pull request, or discussion within that repository.

About default labels

GitHub provides default labels in every new repository. You can use these default labels to help create a standard workflow in a repository.

Label	Description
`bug`	Indicates an unexpected problem or unintended behavior
`documentation`	Indicates a need for improvements or additions to documentation
`duplicate`	Indicates similar issues, pull requests, or discussions
`enhancement`	Indicates new feature requests
`good first issue`	Indicates a good issue for first-time contributors
`help wanted`	Indicates that a maintainer wants help on an issue or pull request
`invalid`	Indicates that an issue, pull request, or discussion is no longer relevant
`question`	Indicates that an issue, pull request, or discussion needs more information
`wontfix`	Indicates that work won't continue on an issue, pull request, or discussion

Default labels are included in every new repository when the repository is created, but you can edit or delete the labels later.

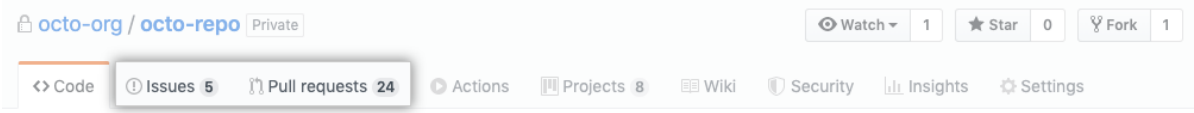
Issues with the `good first issue` label are used to populate the repository's `contribute` page. For an example of a `contribute` page, see github/docs/contribute.

Organization owners can customize the default labels for repositories in their organization. For more information, see "Managing default labels for repositories in your organization."

Creating a label

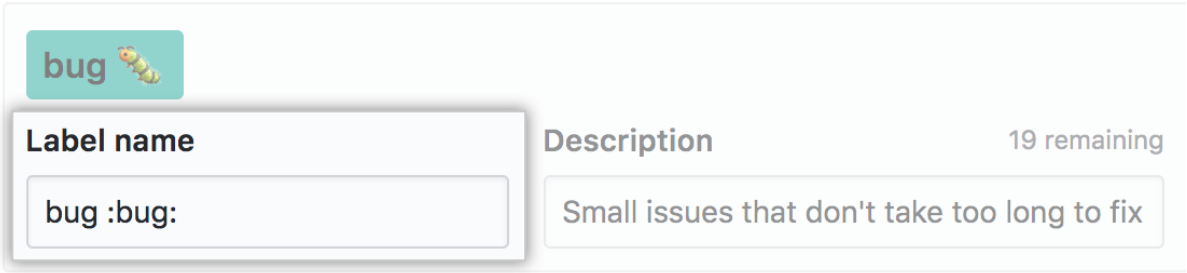
Anyone with write access to a repository can create a label.

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Issues** or **Pull requests**.

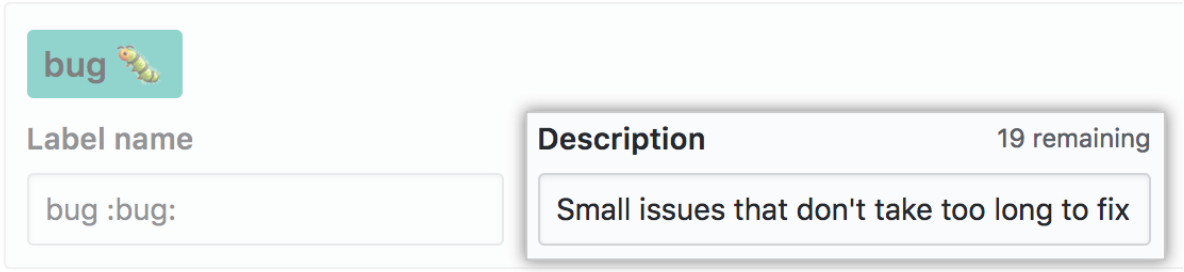


3. Above the list of issues or pull requests, click **Labels**.

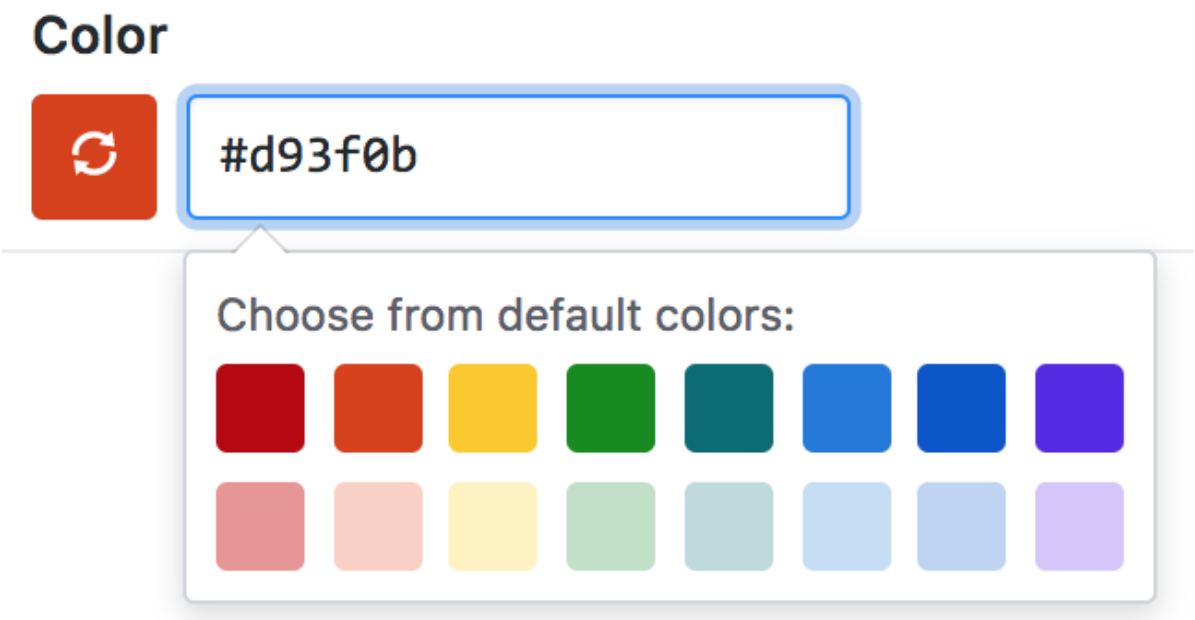
4. To the right of the search field, click **New label**.
5. Under "Label name", type a name for your label.



6. Under "Description", type a description to help others understand and use your label.



7. Optionally, to customize the color of your label, edit the hexadecimal number, or click the refresh button for another random selection.



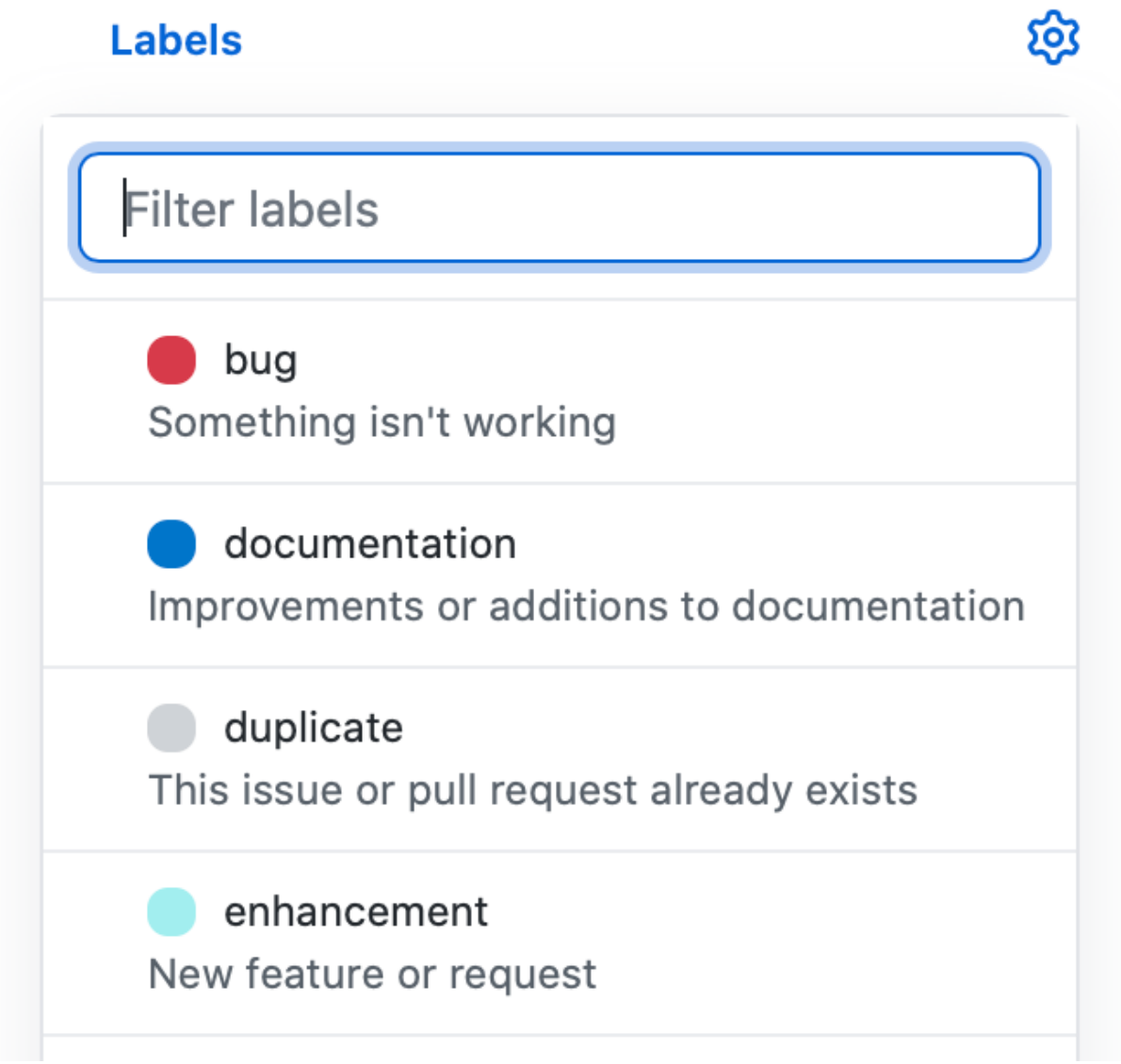
8. To save the new label, click **Create label**.

Applying a label

Anyone with triage access to a repository can apply and dismiss labels.

1. Navigate to the issue, pull request, or discussion.

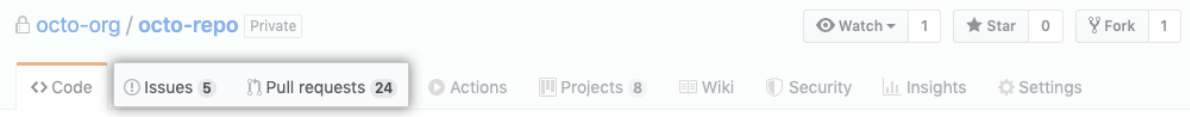
2. In the right sidebar, to the right of "Labels", click , then click a label.



Editing a label

Anyone with write access to a repository can edit existing labels.

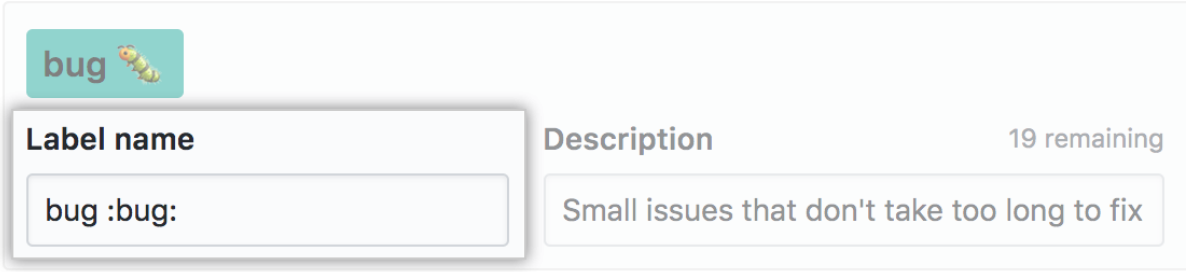
1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Issues** or **Pull requests**.



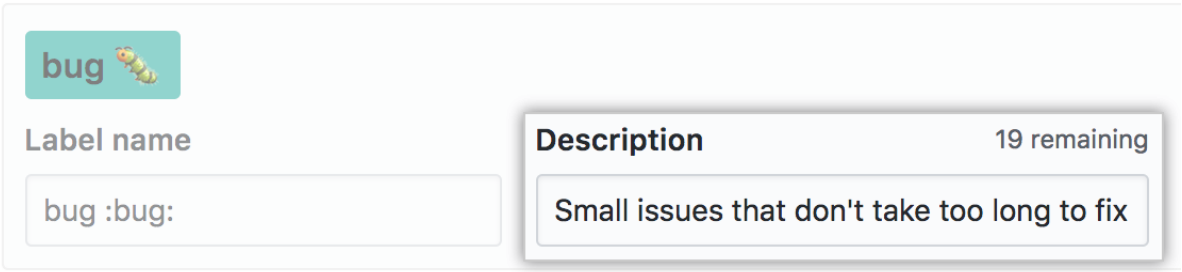
3. Above the list of issues or pull requests, click **Labels**.

4. In the labels list, to the right of the label you want to edit, click **Edit**.

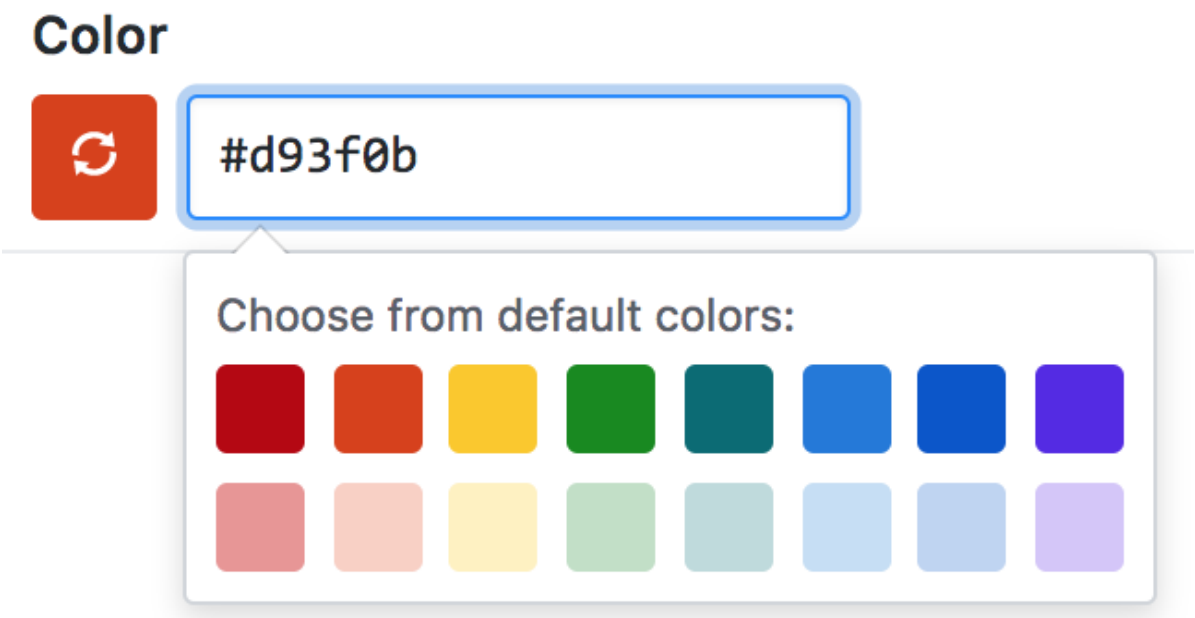
5. Under "Label name", type a name for your label.



6. Under "Description", type a description to help others understand and use your label.



7. Optionally, to customize the color of your label, edit the hexadecimal number, or click the refresh button for another random selection.



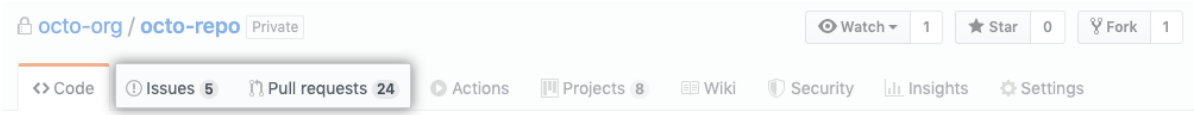
8. Click **Save changes**.

Deleting a label

Anyone with write access to a repository can delete existing labels.

Deleting a label will remove the label from issues and pull requests.

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Issues** or **Pull requests**.



3. Above the list of issues or pull requests, click **Labels**.

4. In the labels list, to the right of the label you want to delete, click **Delete**.

Further reading

About pull requests

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

> **Note:** When working with pull requests, keep the following in mind:
>
> - If you're working in the shared repository model, we recommend that you use a topic branch for your pull request. While you can send pull requests from any branch or commit, with a topic branch you can push follow-up commits if you need to update your proposed changes.
> - When pushing commits to a pull request, don't force push. Force pushing can corrupt your pull request.

You can create pull requests on GitHub.com, with GitHub Desktop, in Codespaces, on GitHub for mobile, and when using GitHub CLI.

After initializing a pull request, you'll see a review page that shows a high-level overview of the changes between your branch (the compare branch) and the repository's base branch. You can add a summary of the proposed changes, review the changes made by commits, add labels, milestones, and assignees, and @mention individual contributors or teams. For more information, see "Creating a pull request."

Once you've created a pull request, you can push commits from your topic branch to add them to your existing pull request. These commits will appear in chronological order within your pull request and the changes will be visible in the "Files changed" tab.

Other contributors can review your proposed changes, add review comments, contribute to the pull request discussion, and even add commits to the pull request.

You can see information about the branch's current deployment status and past deployment activity on the "Conversation" tab. For more information, see "Viewing deployment activity for a repository."

After you're happy with the proposed changes, you can merge the pull request. If you're working in a shared repository model, you create a pull request and you, or someone else, will merge your changes from your feature branch into the base branch you specify in your pull request. For more information, see "Merging a pull request."

If status checks are required for a repository, the required status checks must pass before you can merge your branch into the protected branch. For more information, see "About protected branches."

You can link a pull request to an issue to show that a fix is in progress and to automatically close the issue when someone merges the pull request. For more information, see "Linking a pull request to an issue."

Tips:

- To toggle between collapsing and expanding all outdated review comments in a pull request, hold down `Alt` and click **Show outdated** or **Hide outdated**. For more shortcuts, see "Keyboard shortcuts."
- You can squash commits when merging a pull request to gain a more streamlined view of changes. For more information, see "About pull request merges."

You can visit your dashboard to quickly find links to recently updated pull requests you're working on or subscribed to. For more information, see "About your personal dashboard."

Draft pull requests

Draft pull requests are available in public repositories with GitHub Free for organizations and legacy per-repository billing plans, and in public and private repositories with GitHub Team, GitHub Enterprise Server 2.17+, and GitHub Enterprise Cloud. For more information, see "GitHub's products."

When you create a pull request, you can choose to create a pull request that is ready for review or a draft pull request. Draft pull requests cannot be merged, and code owners are not automatically requested to review draft pull requests. For more information about creating a draft pull request, see "Creating a pull request" and "Creating a pull request from a fork."

When you're ready to get feedback on your pull request, you can mark your draft pull request as ready for review. Marking a pull request as ready for review will request reviews from any code owners. You can convert a pull request to a draft at any time. For more information, see "Changing the stage of a pull request."

Differences between commits on compare and pull request pages

The compare and pull request pages use different methods to calculate the diff for changed files:

- Compare pages show the diff between the tip of the head ref and the current common ancestor (that is, the merge base) of the head and base ref.

- Pull request pages show the diff between the tip of the head ref and the common ancestor of the head and base ref at the time when the pull request was created. Consequently, the merge base used for the comparison might be different.

Further reading

Committing and reviewing changes to your project

GitHub Desktop tracks all changes to all files as you edit them. You can decide how to group the changes to create meaningful commits.

About commits

Similar to saving a file that's been edited, a commit records changes to one or more files in your branch. Git assigns each commit a unique ID, called a SHA or hash, that identifies:

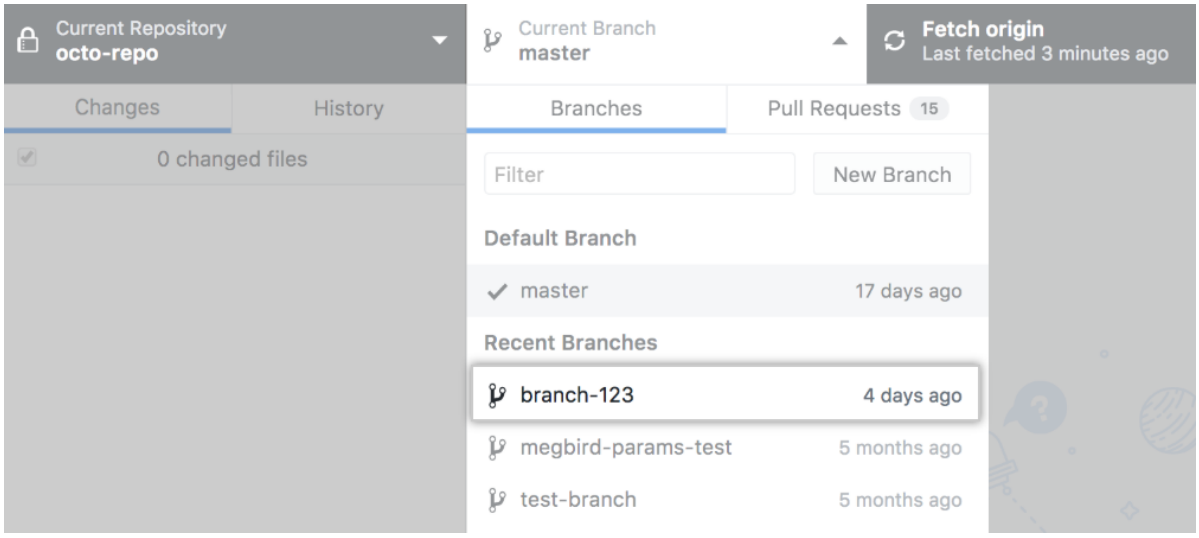
- The specific changes
- When the changes were made
- Who created the changes

When you make a commit, you must include a commit message that briefly describes the changes. You can also add a co-author on any commits you collaborate on.

If the commits you make in GitHub Desktop are associated with the wrong account on GitHub, update the email address in your Git configuration using GitHub Desktop. For more information, see "Configuring Git for GitHub Desktop."

Choosing a branch and making changes

1. Create a new branch, or select an existing branch by clicking **Current Branch** on the toolbar and selecting the branch from the list.



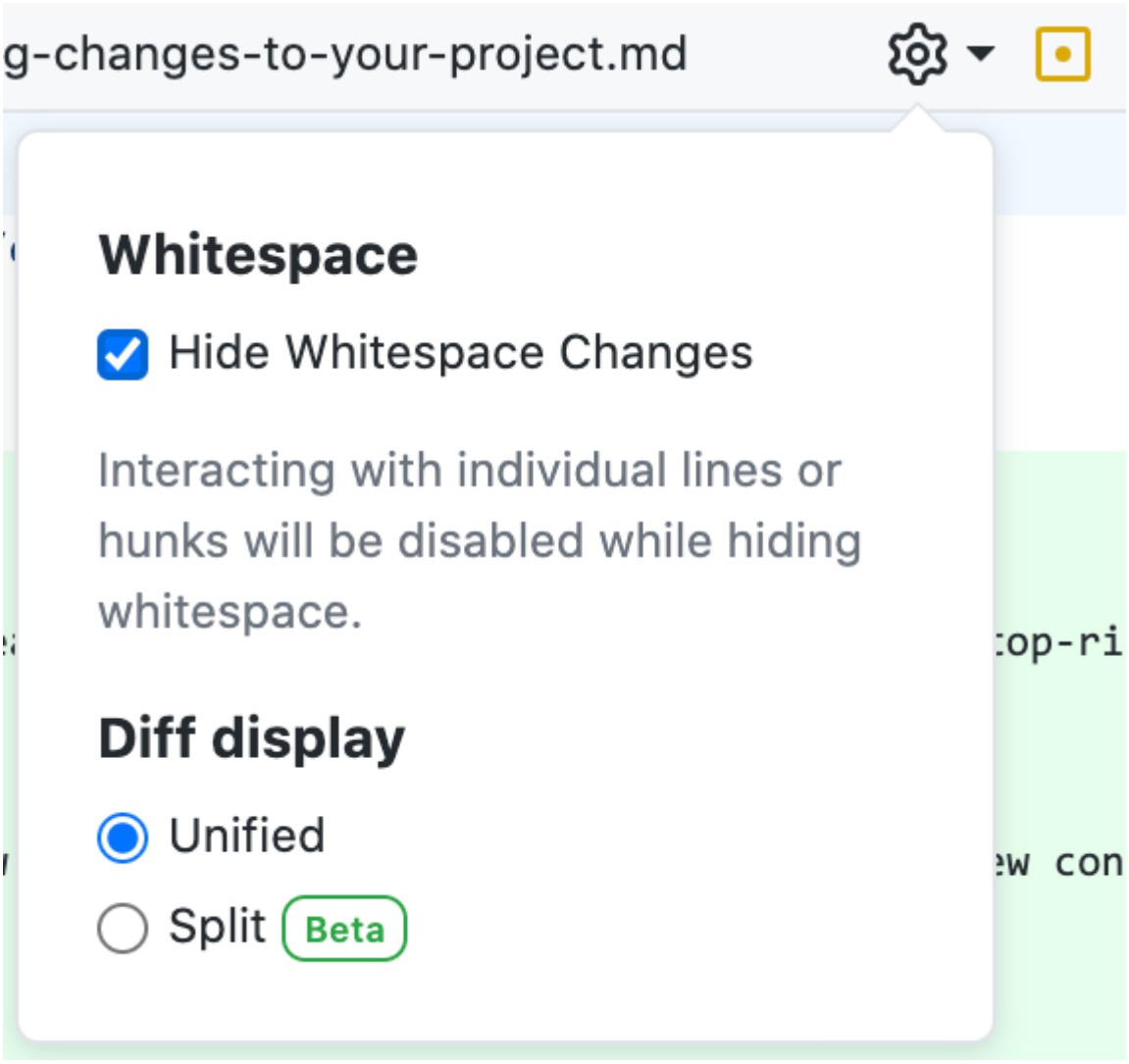
2. Using your favorite text editor, such as Atom, make the necessary changes to files in your project.

Choosing how to display diffs

You can change the way diffs are displayed in GitHub Desktop to suit your reviewing needs.

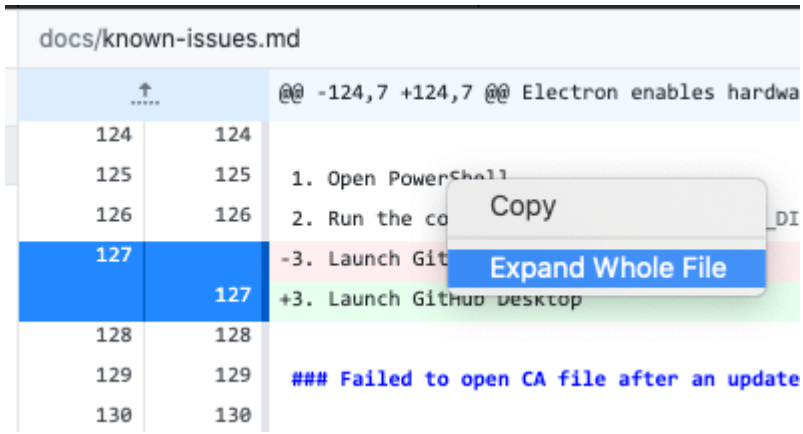
To change how you view diffs, in the top-right corner of the diff view, click ⚙.

- To change how the entire diff is displayed, under "Diff display", select **Unified** or **Split**. The Unified view shows changes linearly, while the Split view shows old content on the left side and new content on the right side.
- To hide whitespace changes so you can focus on more substantive changes, select **Hide Whitespace Changes**.



If you need to see more of the file than GitHub Desktop shows by default, you can expand the diff.

- To see the next few lines above or below the highlighted changes, click the arrow above or below the line numbers.
- To see the entire file, right-click in the diff view and click **Expand Whole File**.

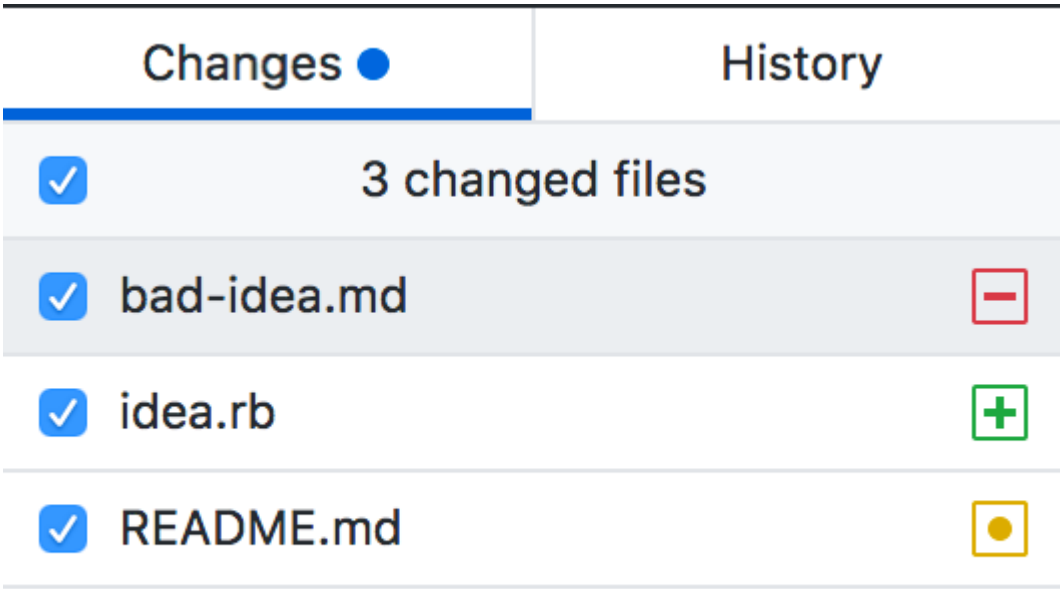


Selecting changes to include in a commit

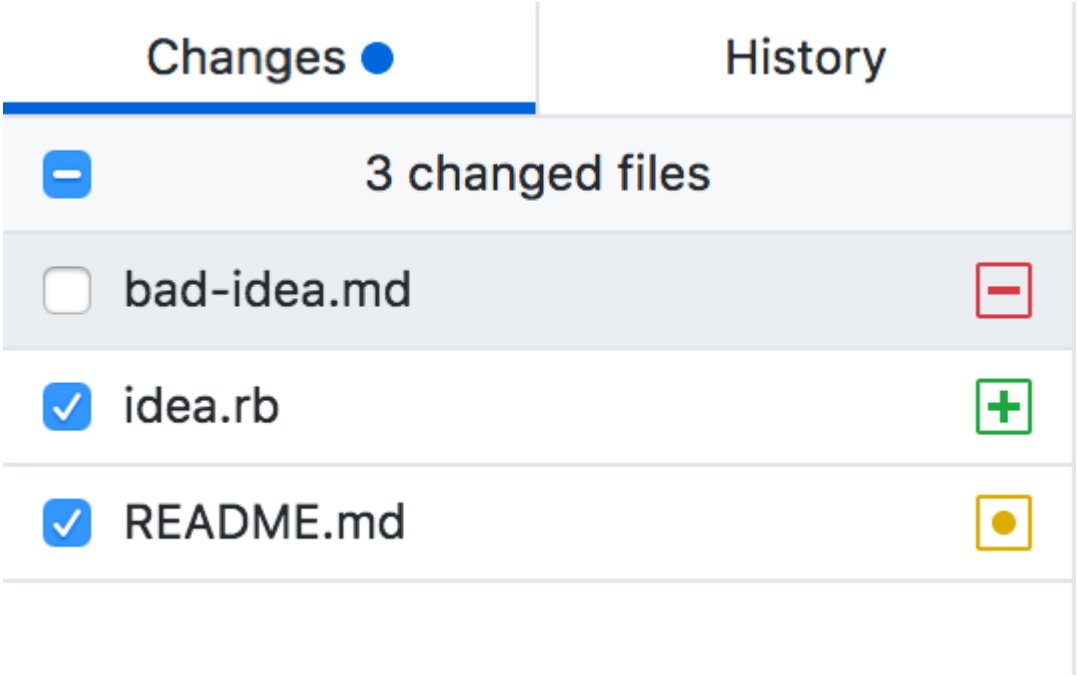
As you make changes to files in your text editor and save them locally, you will also see the changes in GitHub Desktop.

- The red ⊟ icon indicates removed files.
- The yellow ⊡ icon indicates modified files.
- The green ⊞ icon indicates added files.
- To access stashed changes, click **Stashed Changes**.

- To add **all changes in all files** to a single commit, keep the checkbox at the top of the list selected.



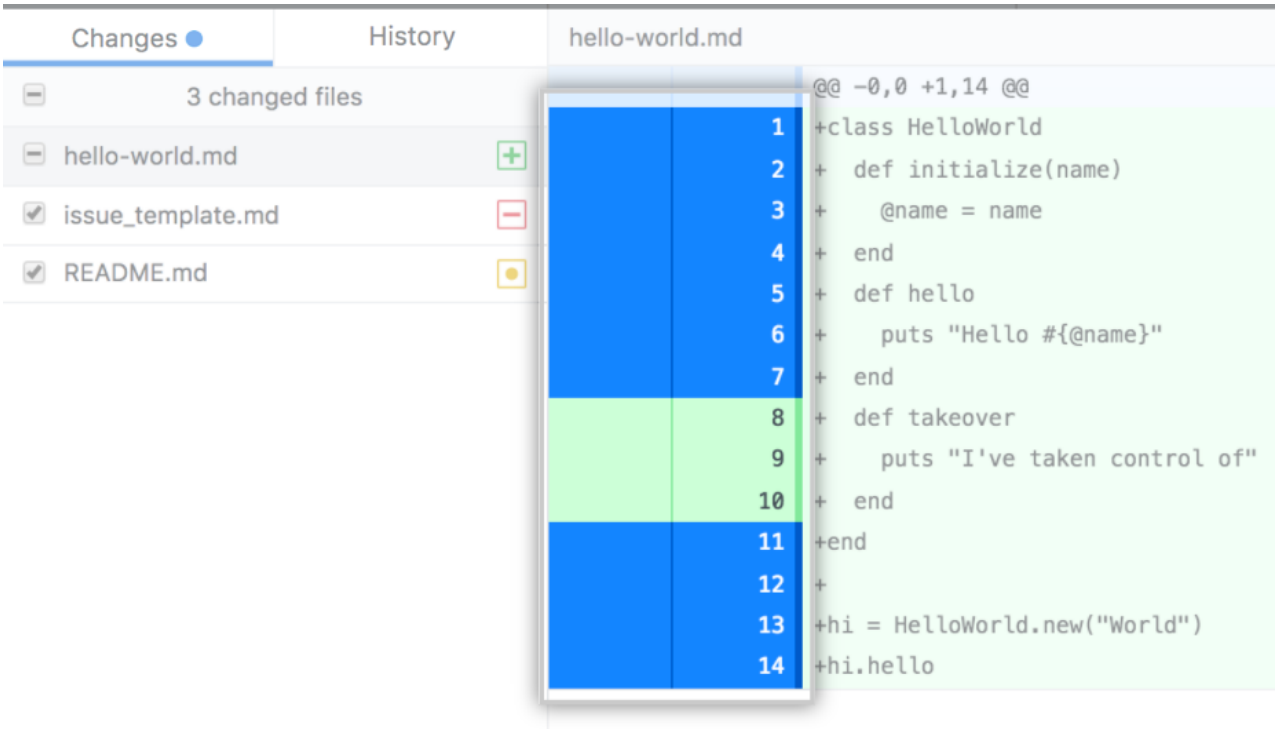
- To add **all changes in one or more files** to a single commit, unselect the checkboxes next to the files you don't want included, leaving only the files you want in the commit. You can toggle the checkbox with the `Spacebar` or `Enter` keys after selecting a file.



Creating a partial commit

If one file contains multiple changes, but you only want some of those changes to be included in a commit, you can create a partial commit. The rest of your changes will remain intact, so that you can make additional modifications and commits. This allows you to make separate, meaningful commits, such as keeping line break changes in a commit separate from code or prose changes.

To exclude changed lines from your commit, click one or more changed lines so the blue disappears. The lines that are still highlighted in blue will be included in the commit.



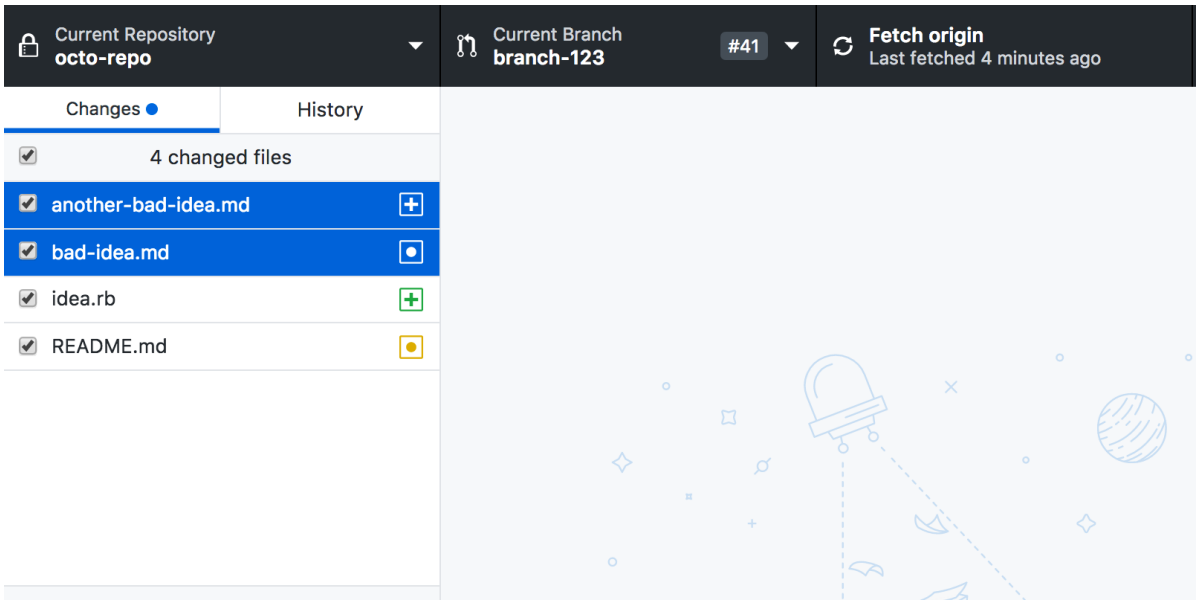
Discarding changes

If you have uncommitted changes that you don't want to keep, you can discard the changes. This will remove the changes from the files on your computer. You can discard all uncommitted changes in one or more files, or you can discard specific lines you added.

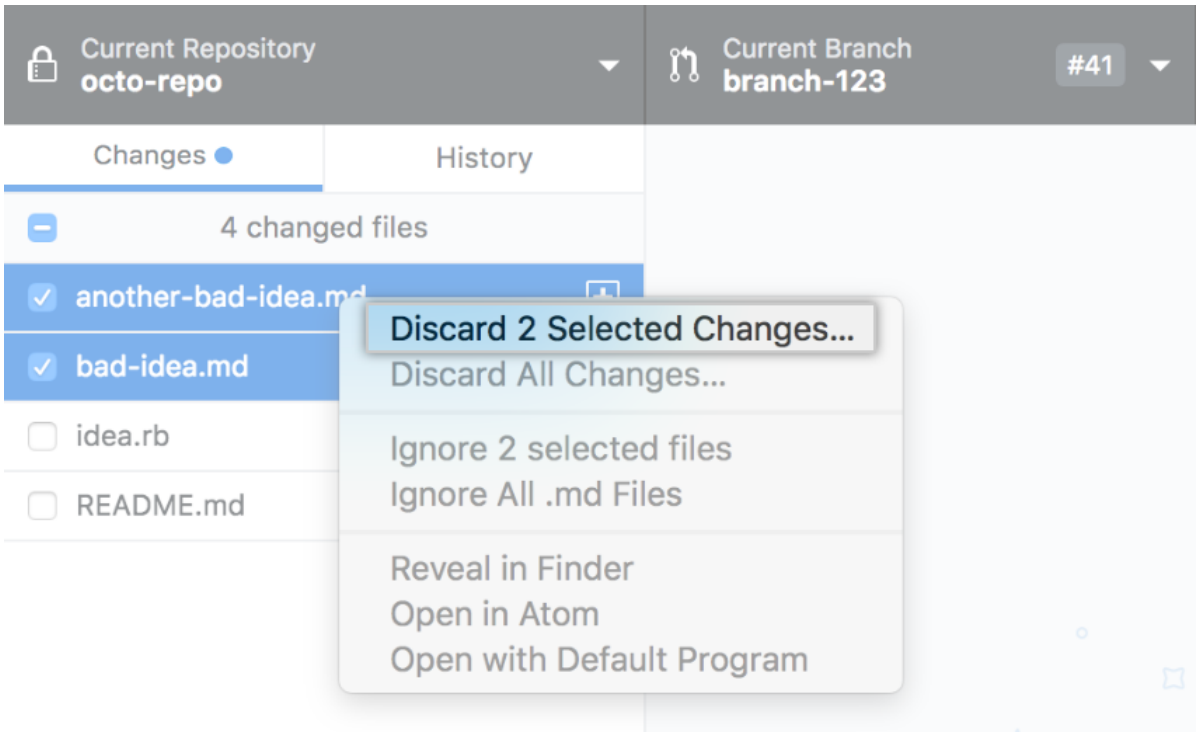
Discarded changes are saved in a dated file in the Trash. You can recover discarded changes until the Trash is emptied.

Discarding changes in one or more files

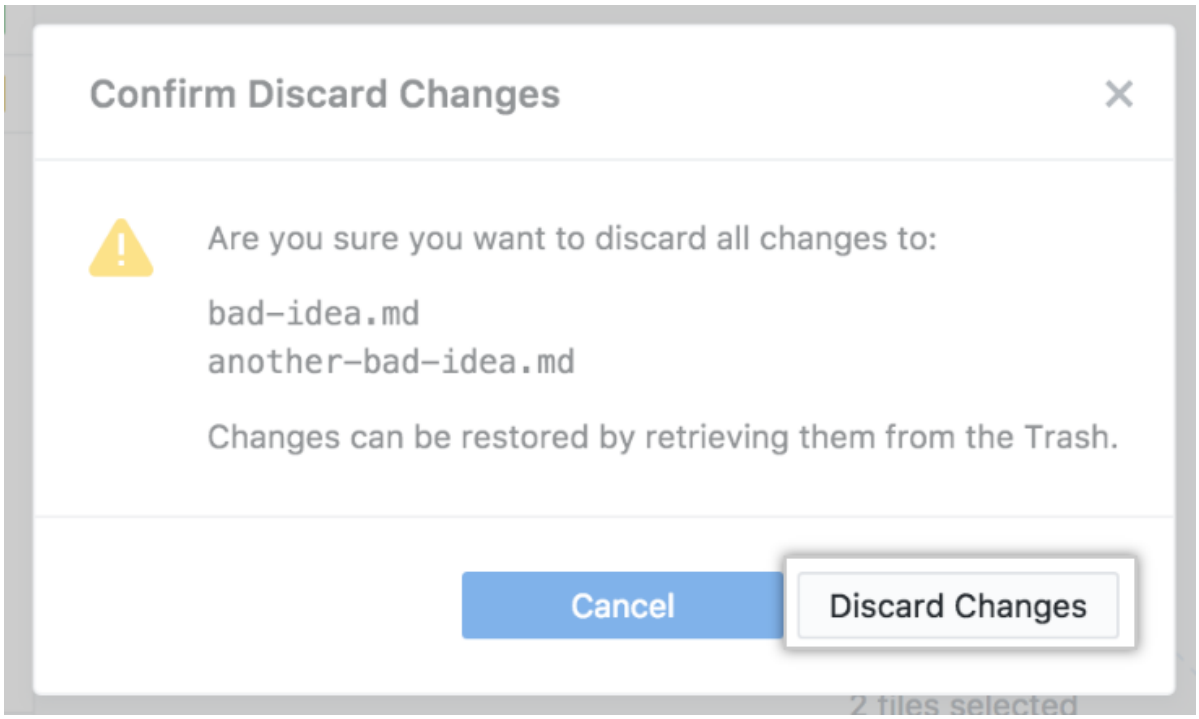
1. In the list of changed files, select the files where you want to discard the changes since the last commit. To select multiple files, click `shift` and click on the range of files you want to discard changes from.



2. Click **Discard Changes** or **Discard Selected Changes** to discard changes to one or more files, or **Discard All Changes** to discard changes to all files since the last commit.



3. To confirm the changes, review the files affected and click **Discard Changes**.

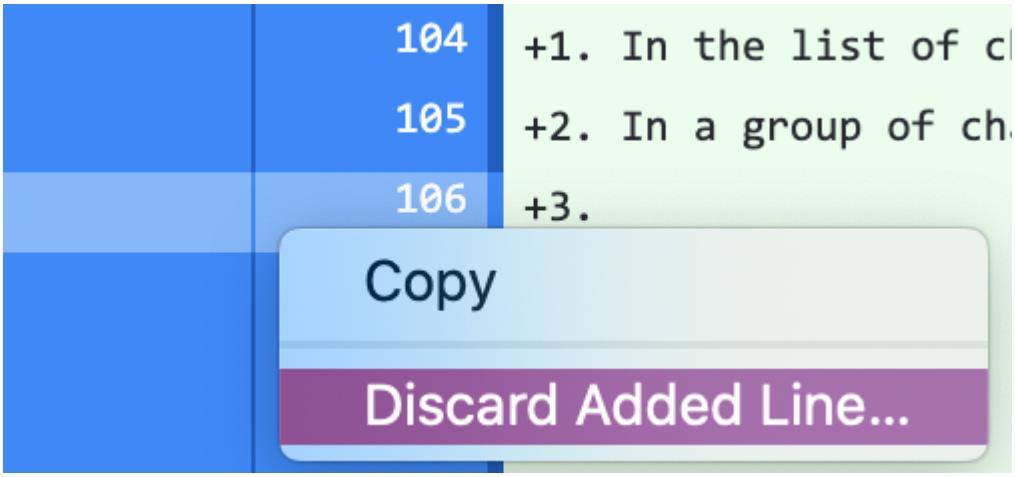


Discarding changes in one or more lines

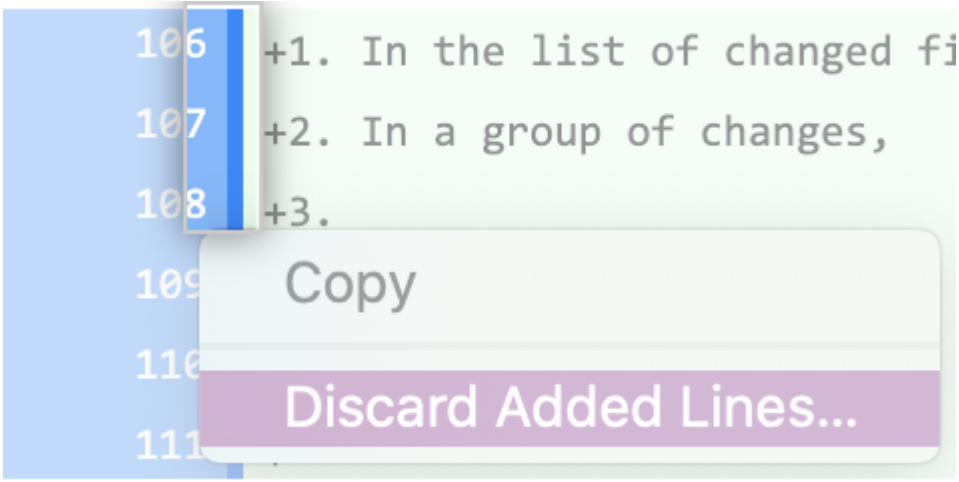
You can discard one or more changed lines that are uncommitted.

> **Note:** Discarding single lines is disabled in a group of changes that adds and removes lines.

To discard one added line, in the list of changed lines, right click on the line you want to discard and select **Discard added line**.



To discard a group of changed lines, right click the vertical bar to the right of the line numbers for the lines you want to discard, then select **Discard added lines**.



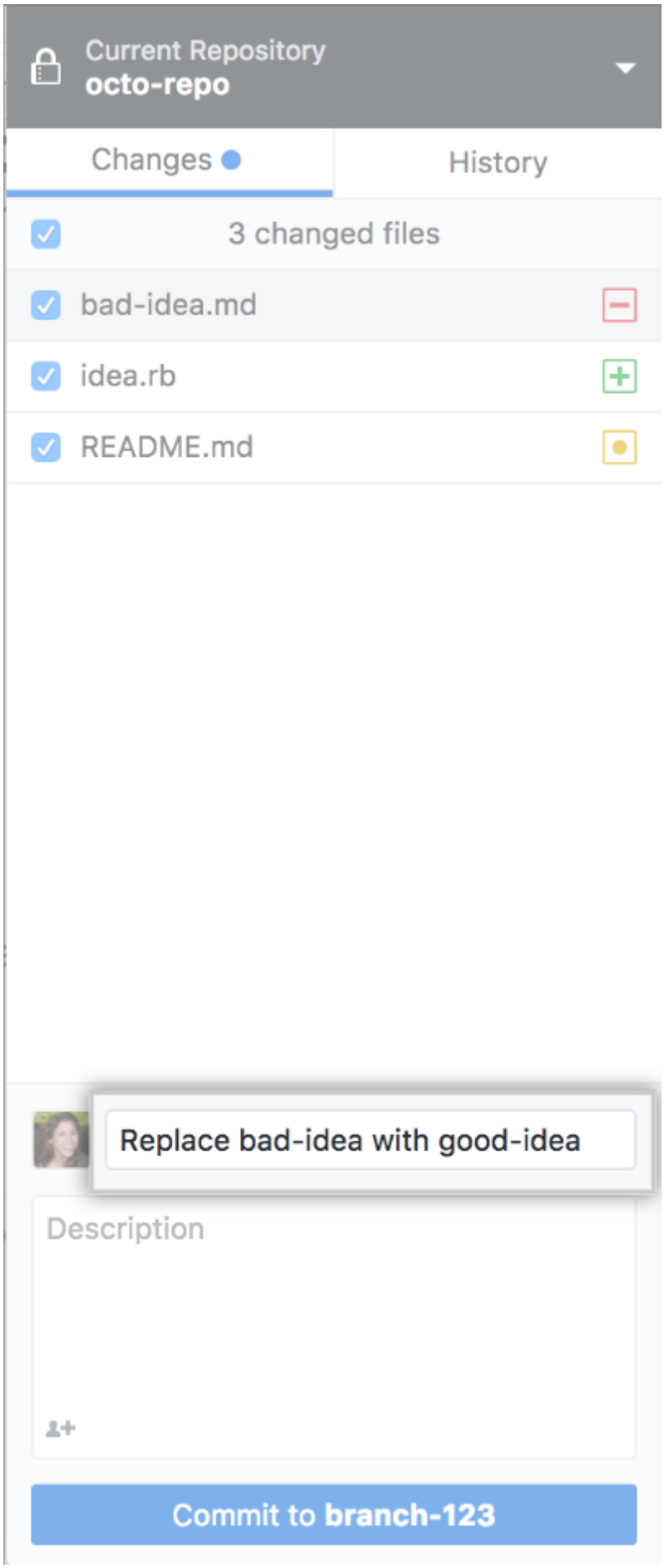
Write a commit message and push your changes

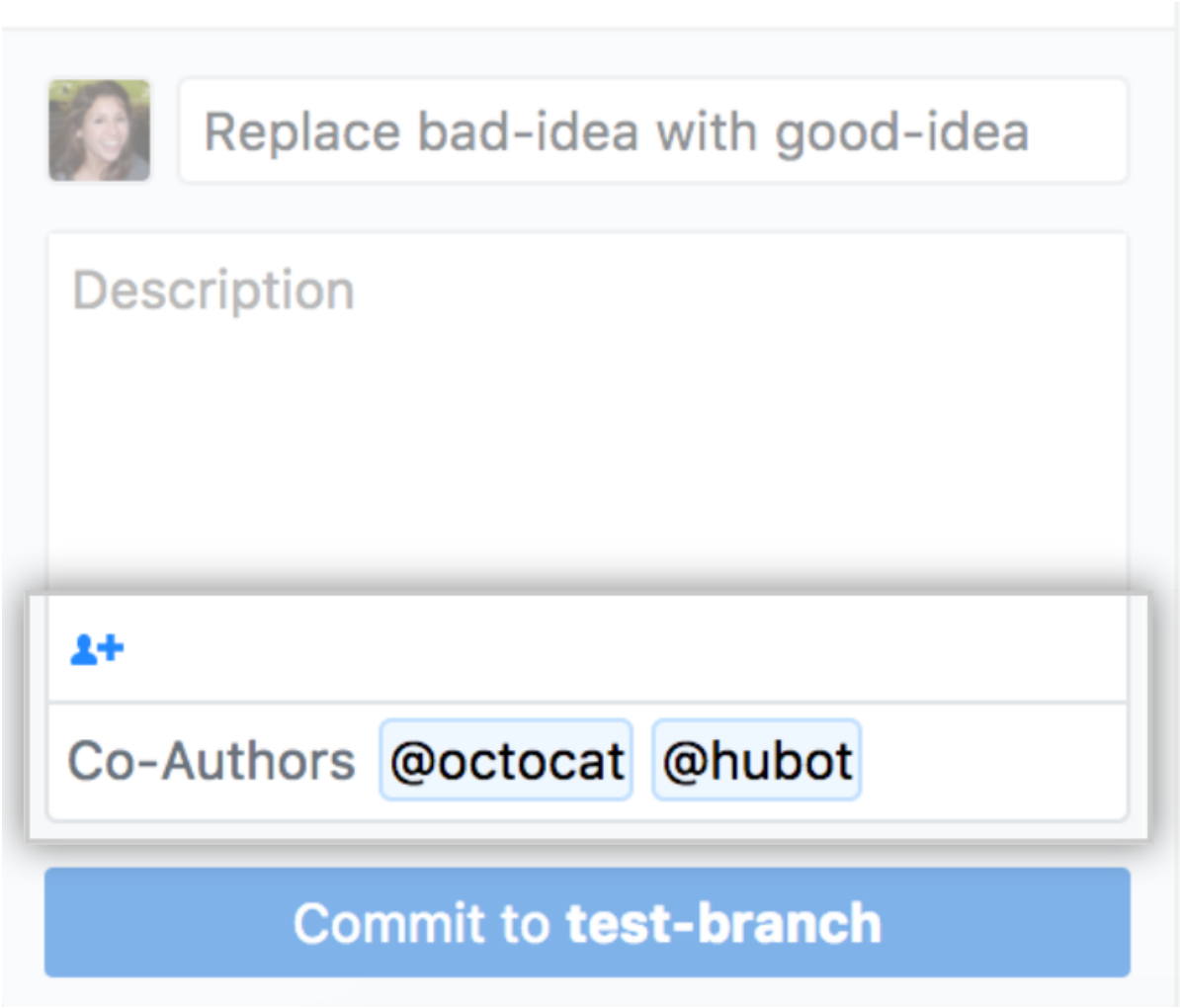
Once you're satisfied with the changes you've chosen to include in your commit, write your commit message and push your changes. If you've collaborated on a commit, you can also attribute a commit to more than one author.

> **Note**: By default, GitHub Desktop will push the tag that you create to your repository with the associated commit. For more information, see "Managing tags."

1. At the bottom of the list of changes, in the Summary field, type a short, meaningful commit message. Optionally, you can add more information about the change in the Description field.

2. Optionally, to attribute a commit to another author, click the add co-authors icon and type the username(s) you want to include.

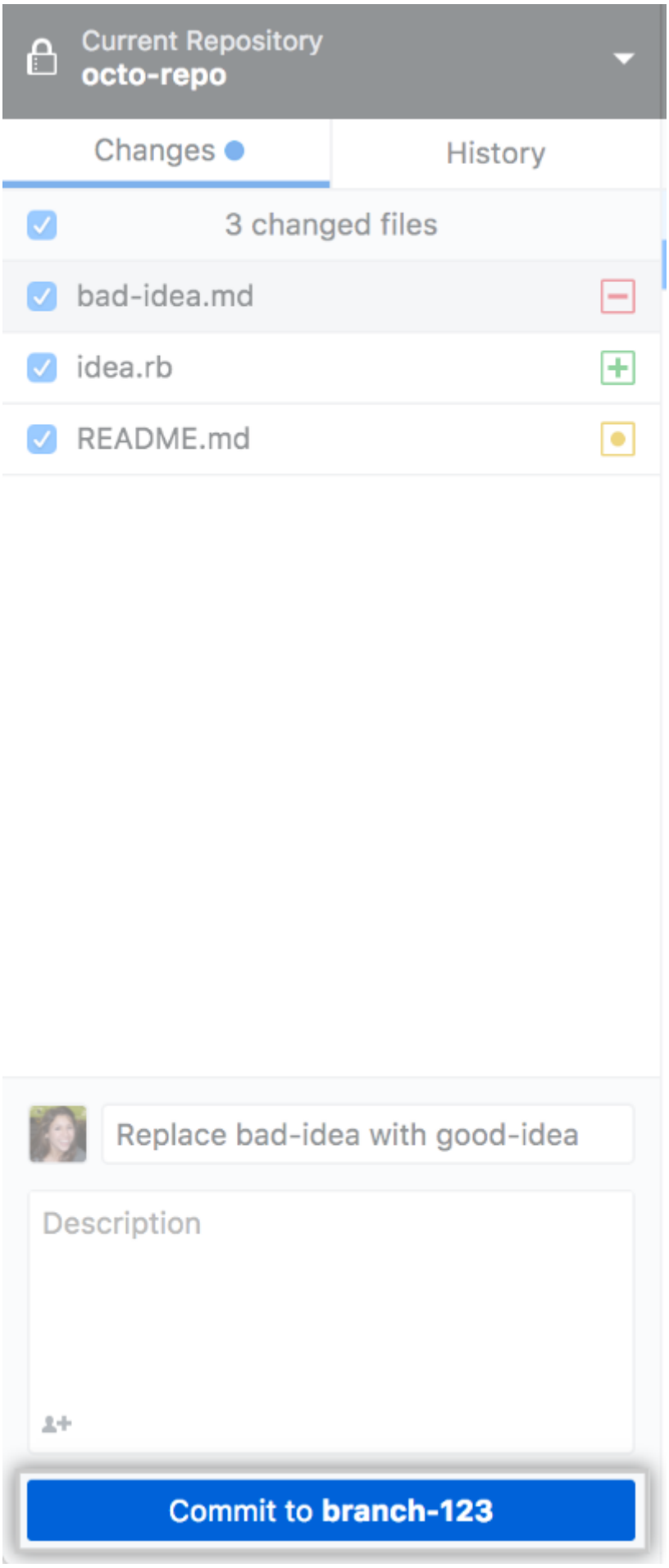


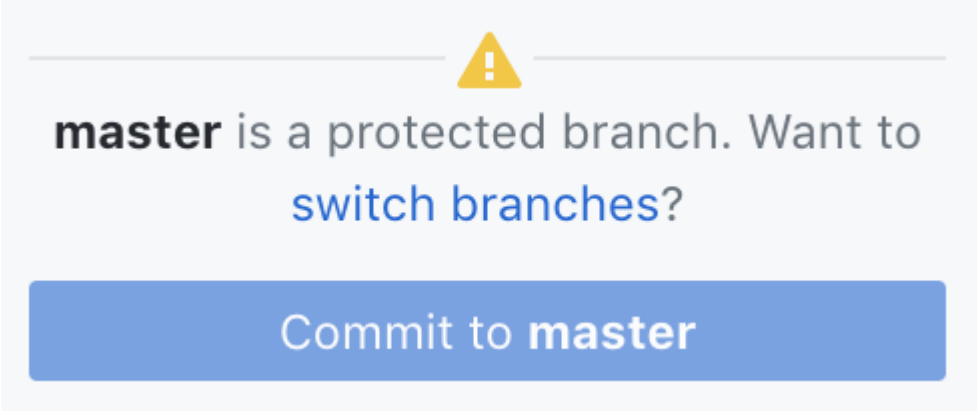


3. Under the Description field, click **Commit to *BRANCH***.

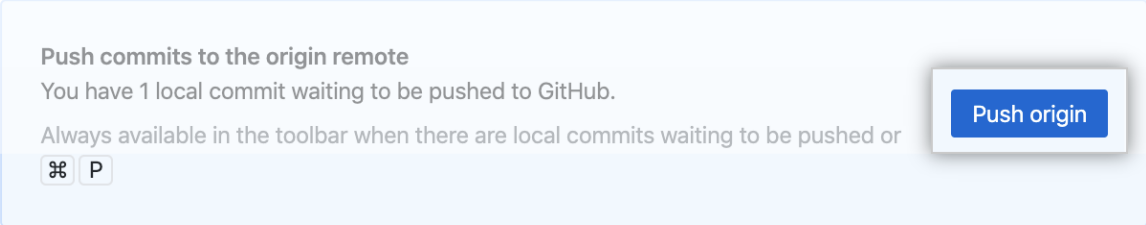
4. If the branch you're trying to commit to is protected, Desktop will warn you.
    - To move your changes, click **switch branches**.
    - To commit your changes to the protected branch, click **Commit to *BRANCH***.

    For more information about protected branches, see "About protected branches".





5. Click **Push origin** to push your local changes to the remote repository.



About branches

Use a branch to isolate development work without affecting other branches in the repository. Each repository has one default branch, and can have multiple other branches. You can merge a branch into another branch using a pull request.

Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.

You always create a branch from an existing branch. Typically, you might create a new branch from the default branch of your repository. You can then work on this new branch in isolation from changes that other people are making to the repository. A branch you create to build a feature is commonly referred to as a feature branch or topic branch. For more information, see "Creating and deleting branches within your repository."

You can also use a branch to publish a GitHub Pages site. For more information, see "About GitHub Pages."

You must have write access to a repository to create a branch, open a pull request, or delete and restore branches in a pull request. For more information, see "Access permissions on GitHub."

About the default branch

When you create a repository with content on GitHub, GitHub creates the repository with a single branch. This first branch in the repository is the default branch. The default branch is the branch that GitHub displays when anyone visits your repository. The default branch is also the initial branch that Git checks out locally when someone clones the repository. Unless you specify a different branch, the default branch in a repository is the base branch for new pull requests and code commits.

By default, GitHub names the default branch `main` in any new repository.

You can change the default branch for an existing repository. For more information, see "Changing the default branch."

You can set the name of the default branch for new repositories. For more information, see "Managing the default branch for your repositories," "Managing the default branch name for repositories in your organization," and "Enforcing repository management policies in your enterprise account."

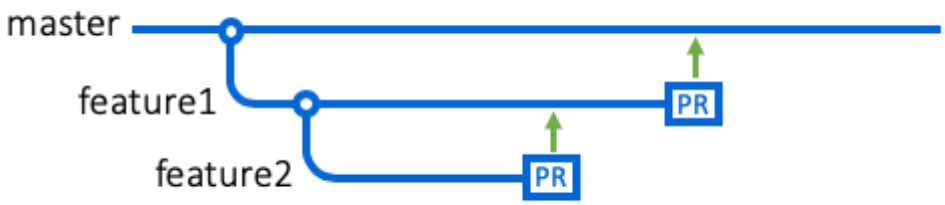
Working with branches

Once you're satisfied with your work, you can open a pull request to merge the changes in the current branch (the *head* branch) into another branch (the *base* branch). For more information, see "About pull requests."

After a pull request has been merged, or closed, you can delete the head branch as this is no longer needed. You must have write access in the repository to delete branches. You can't delete branches that are directly associated with open pull requests. For more information, see "Deleting and restoring branches in a pull request"

If you delete a head branch after its pull request has been merged, GitHub checks for any open pull requests in the same repository that specify the deleted branch as their base branch. GitHub automatically updates any such pull requests, changing their base branch to the merged pull request's base branch. The following diagrams illustrate this.

Here someone has created a branch called `feature1` from the `master` branch, and you've then created a branch called `feature2` from `feature1`. There are open pull requests for both branches. The arrows indicate the current base branch for each pull request. At this point, `feature1` is the base branch for `feature2`. If the pull request for `feature2` is merged now, the `feature2` branch will be merged into `feature1`.



In the next diagram, someone has merged the pull request for `feature1` into the `master` branch, and they have deleted the `feature1` branch. As a result, GitHub has automatically retargeted the pull request for `feature2` so that its base branch is now `master`.

Now when you merge the `feature2` pull request, it'll be merged into the `master` branch.

Working with protected branches

Repository administrators can enable protections on a branch. If you're working on a branch that's protected, you won't be able to delete or force push to the branch. Repository administrators can additionally enable several other protected branch settings to enforce various workflows before a branch can be merged.

> **Note:** If you're a repository administrator, you can merge pull requests on branches with branch protections enabled even if the pull request does not meet the requirements, unless branch protections have been set to "Include administrators."

To see if your pull request can be merged, look in the merge box at the bottom of the pull request's **Conversation** tab. For more information, see "About protected branches."

When a branch is protected:

- You won't be able to delete or force push to the branch.
- If required status checks are enabled on the branch, you won't be able to merge changes into the branch until all of the required CI tests pass. For more information, see "About status checks."
- If required pull request reviews are enabled on the branch, you won't be able to merge changes into the branch until all requirements in the pull request review policy have been met. For more information, see "Merging a pull request."
- If required review from a code owner is enabled on a branch, and a pull request modifies code that has an owner, a code owner must approve the pull request before it can be merged. For more information, see "About code owners."
- If required commit signing is enabled on a branch, you won't be able to push any commits to the branch that are not signed and verified. For more information, see "About commit signature verification" and "About protected branches."
- If you use GitHub's conflict editor to fix conflicts for a pull request that you created from a protected branch, GitHub helps you to create an alternative branch for the pull request, so that your resolution of the conflicts can be merged. For more information, see "Resolving a merge conflict on GitHub."

Further reading

Configuring notifications

Choose the type of activity on GitHub that you want to receive notifications for and how you want these updates delivered.

Notification delivery options

You can receive notifications for activity on GitHub in the following locations.

- The notifications inbox in the GitHub web interface
- The notifications inbox on GitHub for mobile, which syncs with the inbox on GitHub
- An email client that uses a verified email address, which can also sync with the notifications inbox on GitHub and GitHub for mobile

To use the notifications inbox on GitHub and GitHub for mobile, you must enable web and mobile notifications in your notification settings. For more information, see "Choosing your notification settings."

> **Tip:** If you receive both web and email notifications, you can automatically sync the read or unread status of the notification so that web notifications are automatically marked as read once you've read the corresponding email notification. To enable this sync, your email client must be able to view images from `notifications@github.com`.

Benefits of the notifications inbox

The notifications inbox on GitHub and GitHub for mobile includes triaging options designed specifically for your GitHub notifications flow, including options to:

- Triage multiple notifications at once.
- Mark completed notifications as **Done** and remove them from your inbox. To view all of your notifications marked as **Done**, use the `is:done` query.
- Save a notification to review later. Saved notifications are flagged in your inbox and kept indefinitely. To view all of your saved notifications, use the `is:saved` query.
- Unsubscribe and remove a notification from your inbox.
- Preview the issue, pull request, or team discussion where the notification originates on GitHub from within the notifications inbox.
- See one of the latest reasons you're receiving a notification from your inbox with a `reasons` label.
- Create custom filters to focus on different notifications when you want.

- Group notifications in your inbox by repository or date to get a quick overview with less context switching

In addition, you can receive and triage notifications on your mobile device with GitHub for mobile. For more information, see "Managing your notification settings with GitHub for mobile" or "GitHub for mobile."

Benefits of using an email client for notifications

One benefit of using an email client is that all of your notifications can be kept indefinitely depending on your email client's storage capacity. Your inbox notifications are only kept for 5 months on GitHub unless you've marked them as **Saved**. **Saved** notifications are kept indefinitely. For more information about your inbox's retention policy, see "About notifications."

Sending notifications to your email client also allows you to customize your inbox according to your email client's settings, which can include custom or color-coded labels.

Email notifications also allow flexibility with the types of notifications you receive and allow you to choose different email addresses for updates. For example, you can send certain notifications for a repository to a verified personal email address. For more information, about your email customization options, see "Customizing your email notifications."

About participating and watching notifications

When you watch a repository, you're subscribing to updates for activity in that repository. Similarly, when you watch a specific team's discussions, you're subscribing to all conversation updates on that team's page. For more information, see "About team discussions."

To see repositories that you're watching, go to your watching page. For more information, see "Managing subscriptions and notifications on GitHub."

You can configure notifications for a repository on the repository page, or on your watching page.

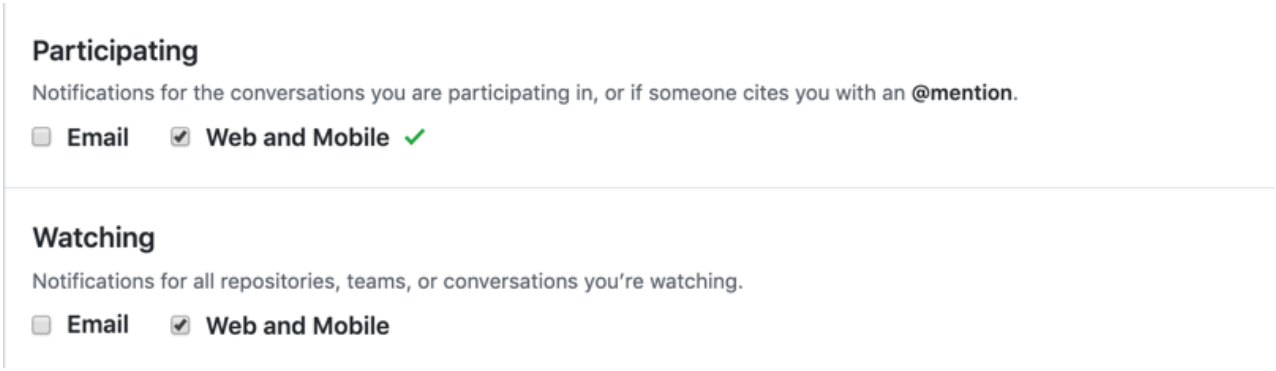
About custom notifications

You can customize notifications for a repository. For example, you can choose to only be notified when updates to one or more types of events (issues, pulls requests, releases, security alerts, or discussions) happen within a repository, or ignore all notifications for a repository. For more information, see "Configuring your watch settings for an individual repository" below.

Participating in conversations

Anytime you comment in a conversation or when someone @mentions your username, you are *participating* in a conversation. By default, you are automatically subscribed to a conversation when you participate in it. You can unsubscribe from a conversation you've participated in manually by clicking **Unsubscribe** on the issue or pull request or through the **Unsubscribe** option in the notifications inbox.

For conversations you're watching or participating in, you can choose whether you want to receive notifications by email or through the notifications inbox on GitHub and GitHub for mobile.



For example:

- If you don't want notifications to be sent to your email, unselect **email** for participating and watching notifications.
- If you want to receive notifications by email when you've participated in a conversation, then you can select **email** under "Participating".

If you do not enable watching or participating notifications for web and mobile, then your notifications inbox will not have any updates.

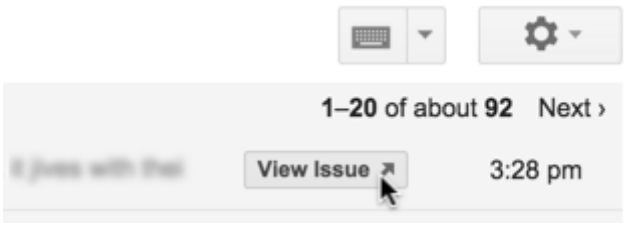
Customizing your email notifications

After enabling email notifications, GitHub will send notifications to you as multipart emails that contain both HTML and plain text copies of the content. Email notification content includes any Markdown, @mentions, emojis, hash-links, and more, that appear in the original content on GitHub. If you only want to see the text in the email, you can configure your email client to display the plain text copy only.

> **Tip:** If you receive both web and email notifications, you can automatically sync the read or unread status of the notification so that web notifications are automatically marked as read once you've read the corresponding email notification. To enable this sync, your email client must be able to view images from `notifications@github.com`.

If you're using Gmail, you can click a button beside the notification email to visit the original issue or pull request that generated the notification.

Choose a default email address where you want to send updates for conversations you're participating in or watching. You can also specify which activity on GitHub you want to receive updates for using your default email address. For example, choose whether you want updates to your default email from:



- Comments on issues and pull requests.
- Pull request reviews.
- Pull request pushes.
- Your own updates, such as when you open, comment on, or close an issue or pull request.

Depending on the organization that owns the repository, you can also send notifications to different email addresses. Your organization may require the email address to be verified for a specific domain. For more information, see "Choosing where your organization's email notifications are sent."

You can also send notifications for a specific repository to an email address. For more information, see "About email notifications for pushes to your repository."

You'll only receive notification emails if you've chosen to receive email notifications in your notification settings.

If an organization you're a member of restricts email notifications to an approved email domain, you'll need to verify an email address in that domain to receive email notifications about activity in the organization. For more information, see "Restricting email notifications to an approved domain."

Filtering email notifications

Each email notification that GitHub sends contains header information. The header information in every email is consistent, so you can use it in your email client to filter or forward all GitHub notifications, or certain types of GitHub notifications.

If you believe you're receiving notifications that don't belong to you, examine the `X-GitHub-Recipient` and `X-GitHub-Recipient-Address` headers. These headers show who the intended recipient is. Depending on your email setup, you may receive notifications intended for another user.

Email notifications from GitHub contain the following header information:

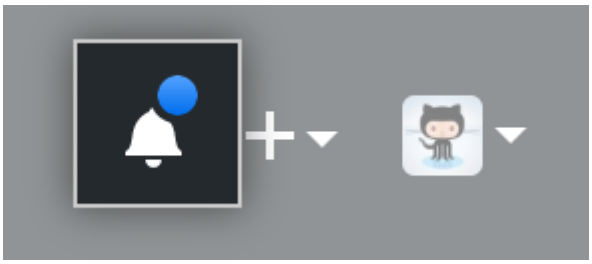
Header	Information
`From` address	This address will always be '`notifications@github.com`'.

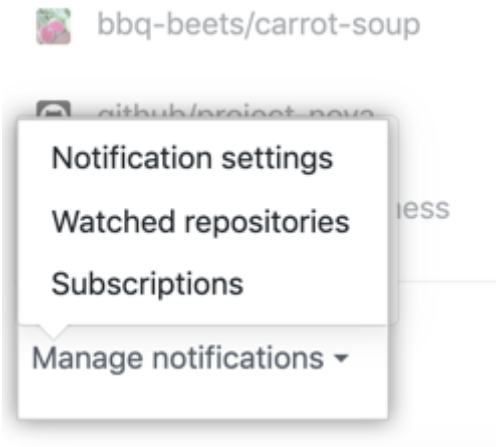
Header	Information
`To` field	This field connects directly to the thread. If you reply to the email, you'll add a new comment to the conversation.
`Cc` address	GitHub will `Cc` you if you're subscribed to a conversation. The second `Cc` email address matches the notification reason. The suffix for these notification reasons is `@noreply.github.com`. The possible notification reasons are: • `assign` : You were assigned to an issue or pull request. • `author` : You created an issue or pull request. • `ci_activity` : A GitHub Actions workflow run that you triggered was completed. • `comment` : You commented on an issue or pull request. • `manual` : There was an update to an issue or pull request you manually subscribed to. • `mention` : You were mentioned on an issue or pull request. • `push` : Someone committed to a pull request you're subscribed to. • `review_requested` : You or a team you're a member of was requested to review a pull request. • `security_alert` : GitHub detected a vulnerability in a repository you receive alerts for. • `state_change` : An issue or pull request you're subscribed to was either closed or opened. • `subscribed` : There was an update in a repository you're watching. • `team_mention` : A team you belong to was mentioned on an issue or pull request. • `your_activity` : You opened, commented on, or closed an issue or pull request.
`mailing list` field	This field identifies the name of the repository and its owner. The format of this address is always `<repository name>.<repository owner>.github.com`.
`X-GitHub-Severity` field	Email notifications for Dependabot alerts that affect one or more repositories include the `X-GitHub-Severity` header field. You can use the value of the `X-GitHub-Severity` header field to filter email notifications for Dependabot alerts. The possible severity levels are: • `low` • `moderate` • `high` • `critical` For more information, see "About alerts for vulnerable dependencies."

Choosing your notification settings

1. In the upper-right corner of any page, click 🔔.
2. In the left sidebar, under the list of repositories, use the "Manage notifications" drop-down to click **Notification settings**.

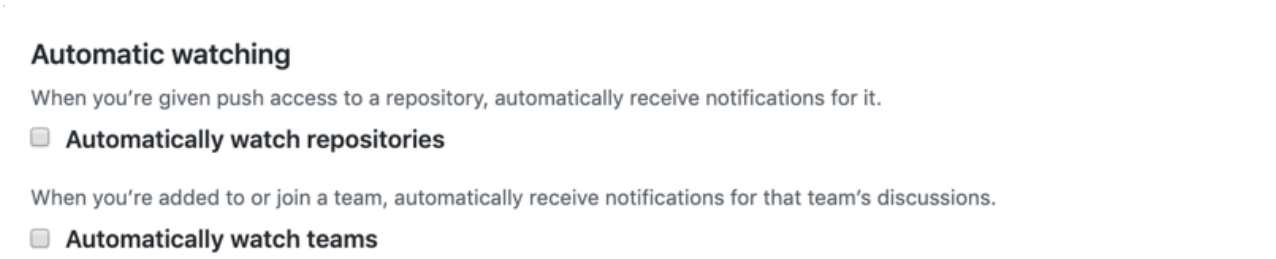
3. On the notifications settings page, choose how you receive notifications when:
    - There are updates in repositories or team discussions you're watching or in a conversation you're participating in. For more information, see "About participating and watching notifications."
    - You gain access to a new repository or you've joined a new team. For more information, see "Automatic watching."
    - There are new Dependabot alerts in your repository. For more information, see "Dependabot alerts notification options."
    - There are workflow runs updates on repositories set up with GitHub Actions. For more information, see "GitHub Actions notification options."





Automatic watching

By default, anytime you gain access to a new repository, you will automatically begin watching that repository. Anytime you join a new team, you will automatically be subscribed to updates and receive notifications when that team is @mentioned. If you don't want to automatically be subscribed, you can unselect the automatic watching options.



If "Automatically watch repositories" is disabled, then you will not automatically watch your own repositories. You must navigate to your repository page and choose the watch option.

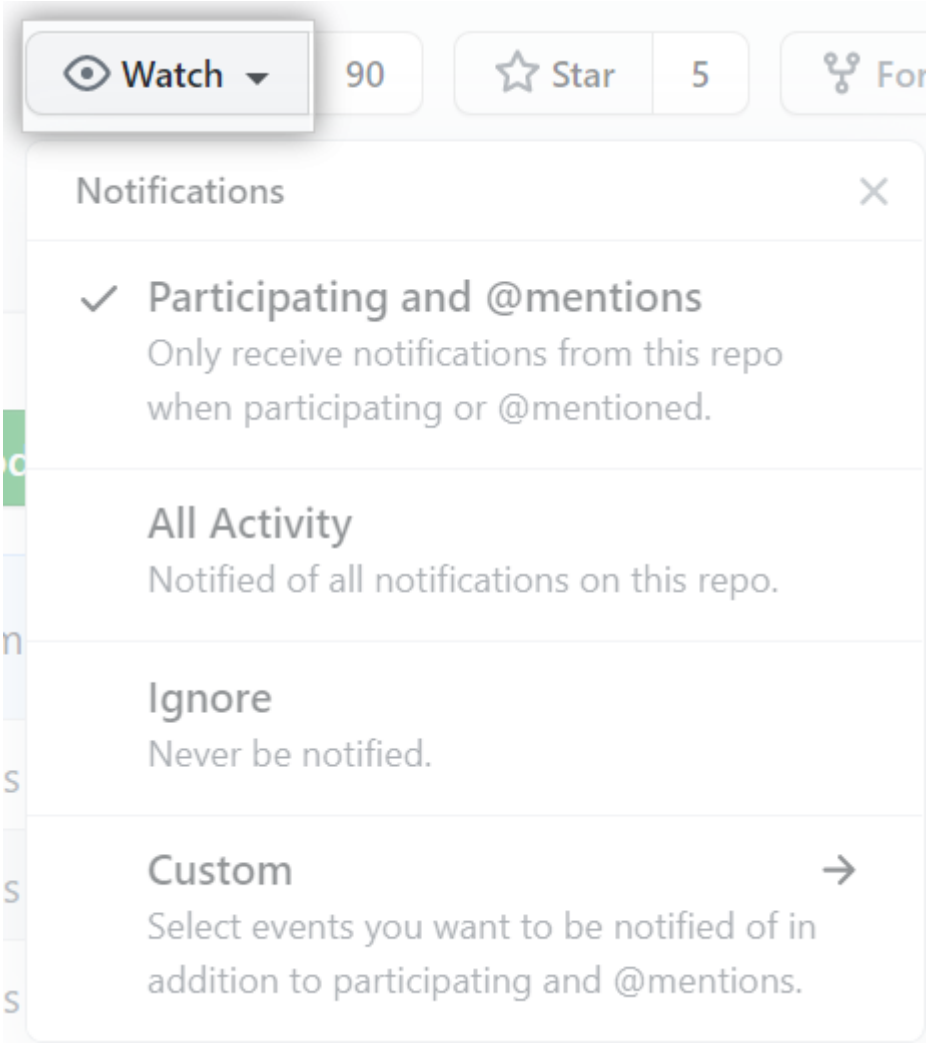
Configuring your watch settings for an individual repository

You can choose whether to watch or unwatch an individual repository. You can also choose to only be notified of certain event types such as issues, pulls requests, releases, security alerts, or discussions (if enabled for the repository) , or completely ignore an

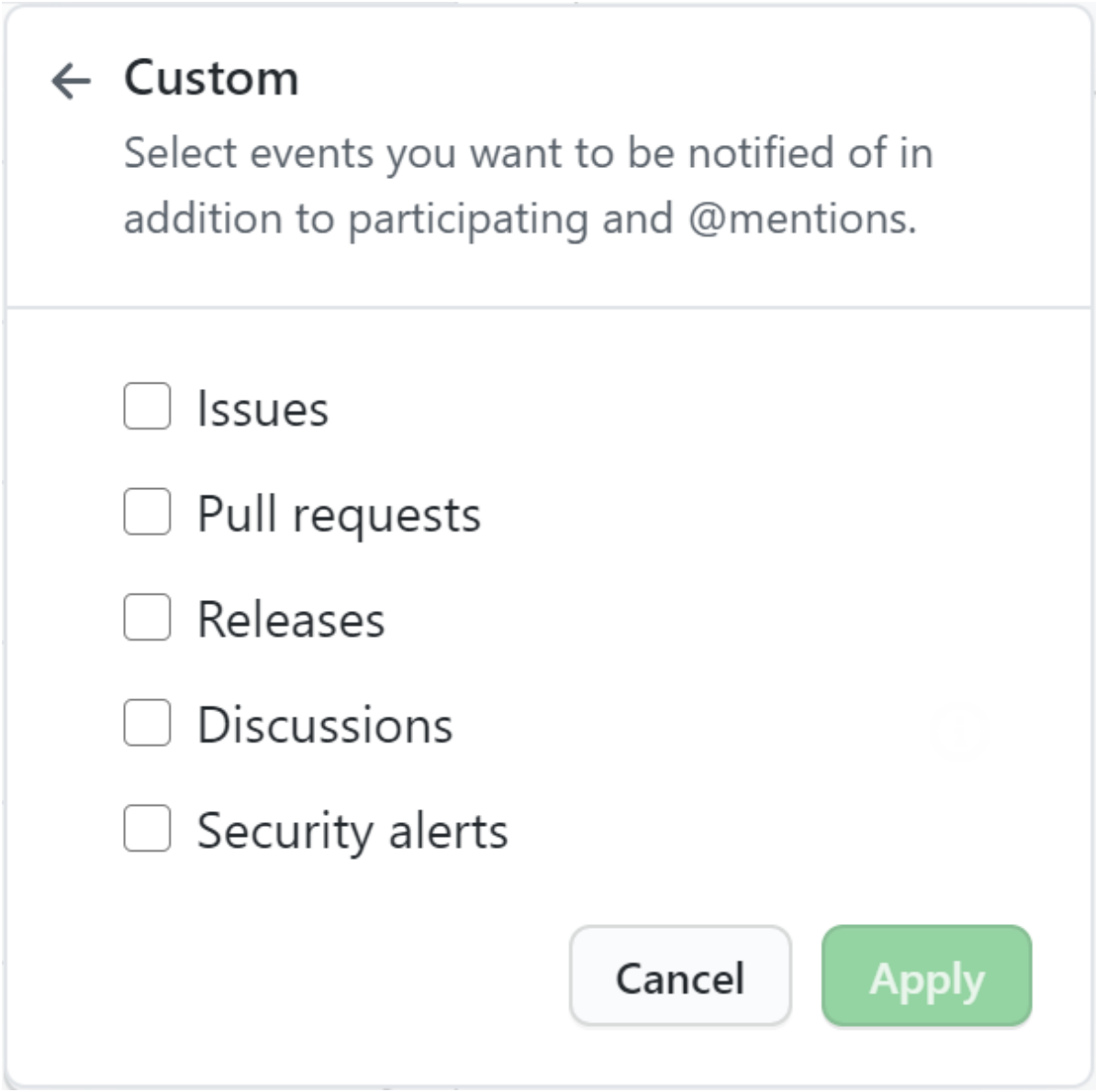
individual repository.

1. On GitHub, navigate to the main page of the repository.

2. In the upper-right corner, click the "Watch" drop-down menu to select a watch option.



The **Custom** option allows you to further customize notifications so that you're only notified when specific events happen in the repository, in addition to participating and @mentions.



If you select "Issues", you will be notified about, and subscribed to, updates on every issue (including those that existed prior to you selecting this option) in the repository. If you're @mentioned in a pull request in this repository, you'll receive notifications for that too, and you'll be subscribed to updates on that specific pull request, in addition to being notified about issues.

Choosing where your organization's email notifications are sent

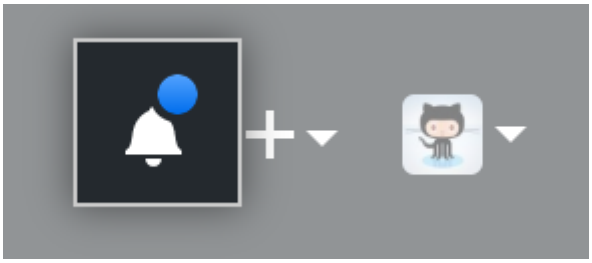
If you belong to an organization, you can choose the email account you want notifications for organization activity sent to. For example, if you belong to an organization for work, you may want your notifications sent to your work email address, rather than your personal address.

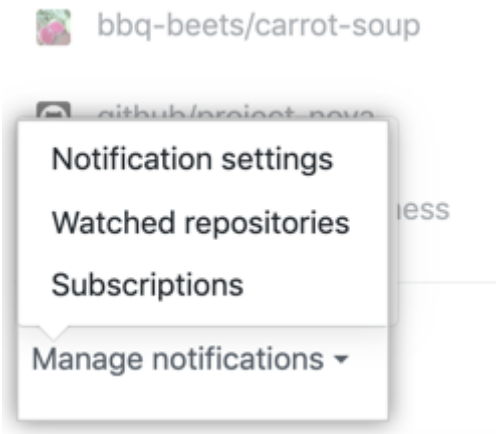
You'll only receive notification emails if you've chosen to receive email notifications in your notification settings.

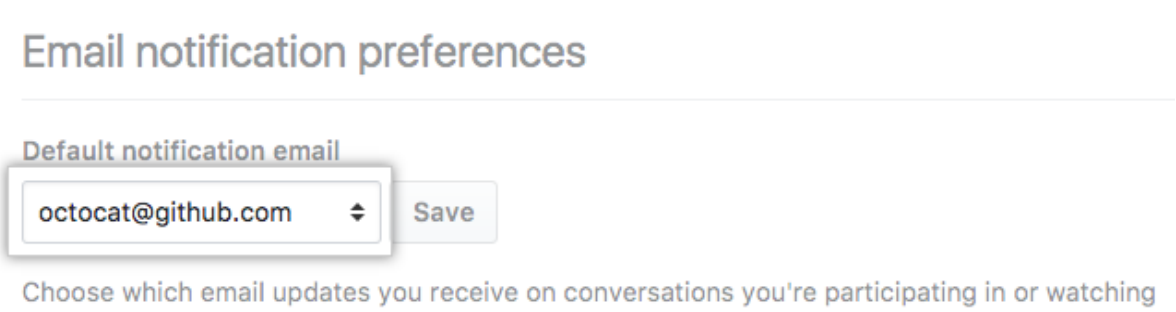
If an organization you're a member of restricts email notifications to an approved email domain, you'll need to verify an email address in that domain to receive email notifications about activity in the organization. For more information, see "Restricting

email notifications to an approved domain."

1. In the upper-right corner of any page, click .
2. In the left sidebar, under the list of repositories, use the "Manage notifications" drop-down to click **Notification settings**.
3. Under "Default notification email", select the email address you'd like notifications sent to.







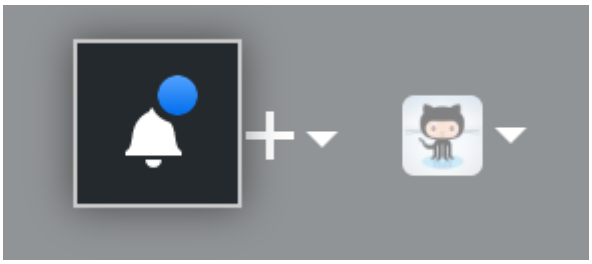
4. Click **Save**.

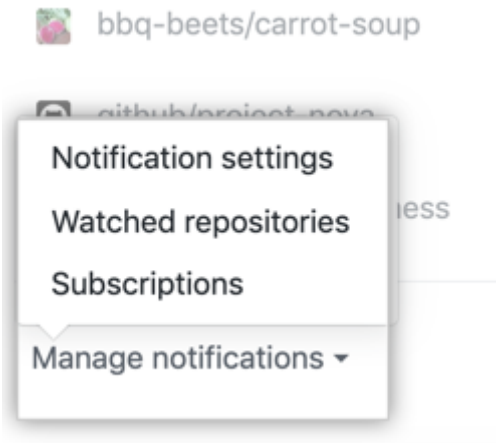
Customizing email routes per organization

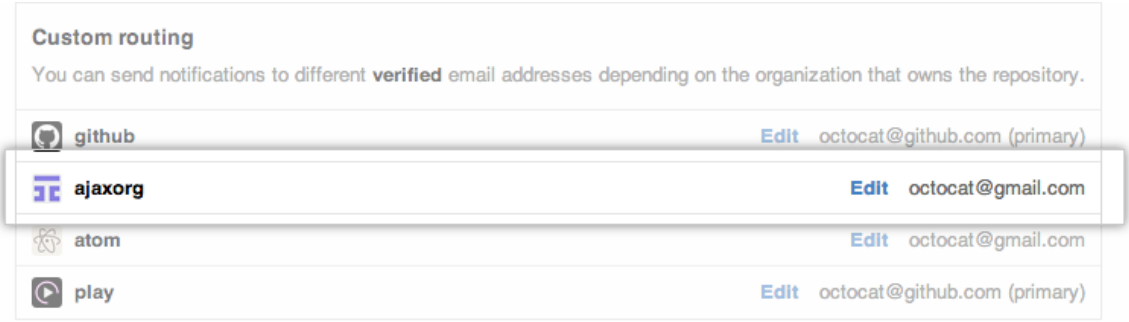
If you are a member of more than one organization, you can configure each one to send notifications to any of your verified email addresses. For more information, see "Verifying your email address."

1. In the upper-right corner of any page, click .
2. In the left sidebar, under the list of repositories, use the "Manage notifications" drop-down to click **Notification settings**.

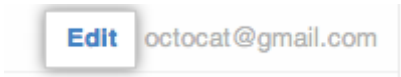
3. Under "Custom routing," find your organization's name in the list.







4. Click **Edit** next to the email address you want to change.
5. Select one of your verified email addresses, then click **Save**.



Dependabot alerts notification options

To receive notifications about Dependabot alerts on repositories, you need to watch these repositories, and subscribe to receive "All Activity" notifications or configure custom settings to include "Security alerts." For more information, see "Configuring your watch settings for an individual repository."

You can choose the delivery method for notifications, as well as the frequency at which the notifications are sent to you.

By default, you will receive notifications:

- by email, an email is sent when Dependabot is enabled for a repository, when a new manifest file is committed to the repository, and when a new vulnerability with a critical or high severity is found (**Email each time a vulnerability is found** option).
- in the user interface, a warning is shown in your repository's file and code views if there are any vulnerable dependencies (**UI alerts** option).
- on the command line, warnings are displayed as callbacks when you push to repositories with any vulnerable dependencies (**Command Line** option).
- in your inbox, as web notifications. A web notification is sent when Dependabot is enabled for a repository, when a new manifest file is committed to the repository, and when a new vulnerability with a critical or high severity is found (**Web** option).
- on GitHub for mobile, as web notifications. For more information, see "Enabling push notifications with GitHub for mobile."

Note: The email and web/GitHub for mobile notifications are:

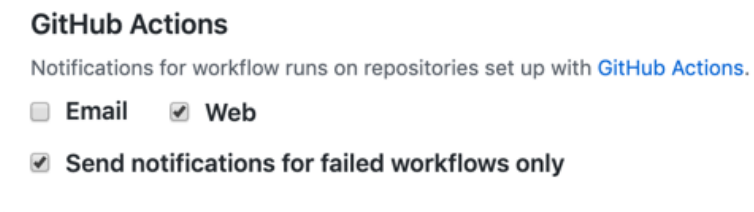
- *per repository* when Dependabot is enabled on the repository, or when a new manifest file is committed to the repository.
- *per organization* when a new vulnerability is discovered.

You can customize the way you are notified about Dependabot alerts. For example, you can receive a weekly digest email summarizing alerts for up to 10 of your repositories using the **Email a digest summary of vulnerabilities** and **Weekly security email digest** options.

For more information about the notification delivery methods available to you, and advice on optimizing your notifications for Dependabot alerts, see "Configuring notifications for vulnerable dependencies."

GitHub Actions notification options

Choose how you want to receive workflow run updates for repositories that you are watching that are set up with GitHub Actions. You can also choose to only receive notifications for failed workflow runs.



Managing your notification settings with GitHub for mobile

When you install GitHub for mobile, you will automatically be opted into web notifications. Within the app, you can enable push notifications for the following events.

- Direct mentions
- Assignments to issues or pull requests
- Requests to review a pull request
- Requests to approve a deployment

You can also schedule when GitHub for mobile will send push notifications to your mobile device.

GitHub Enterprise Server uses background fetch to support push notifications without sending your information to a third-party service, so you may experience a delay in receiving push notifications.

Managing your notification settings with GitHub for iOS

1. In the bottom menu, tap **Profile**.
2. To view your settings, tap ⚙.
3. To update your notification settings, tap **Notifications** and then use the toggles to enable or disable your preferred types of push notifications.
4. Optionally, to schedule when GitHub for mobile will send push notifications to your mobile device, tap **Working Hours**, use the **Custom working hours** toggle, and then choose when you would like to receive push notifications.

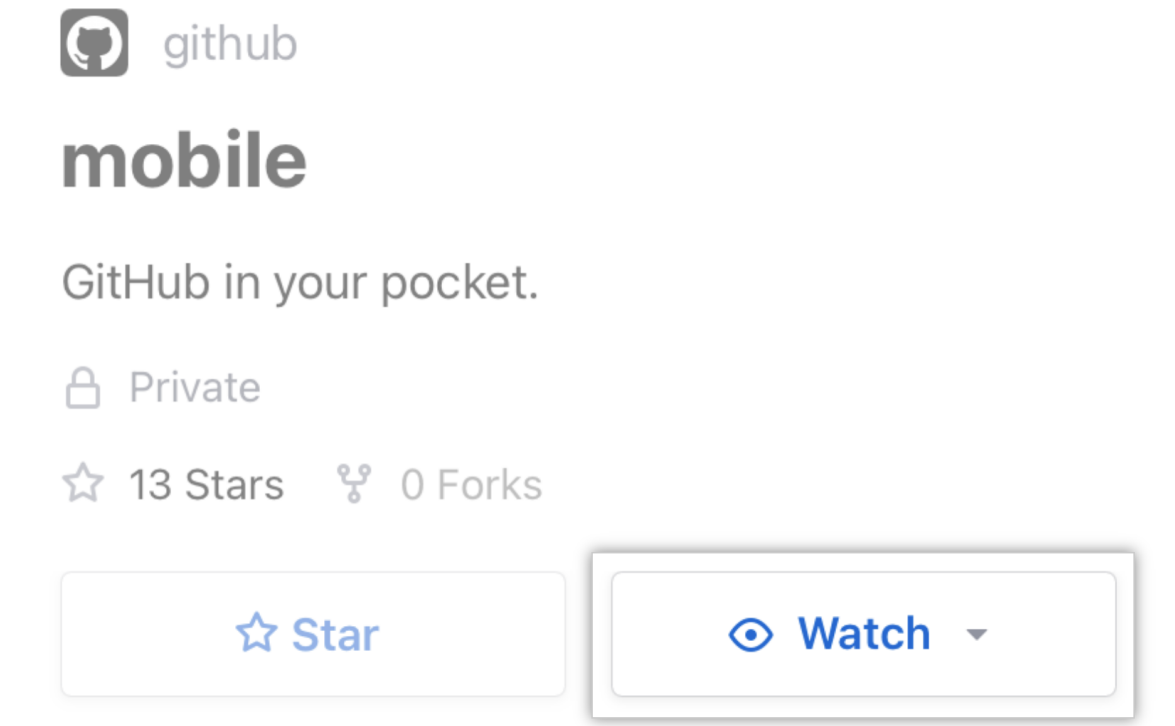
Managing your notification settings with GitHub for Android

1. In the bottom menu, tap **Profile**.
2. To view your settings, tap ⚙.
3. To update your notification settings, tap **Configure Notifications** and then use the toggles to enable or disable your preferred types of push notifications.
4. Optionally, to schedule when GitHub for mobile will send push notifications to your mobile device, tap **Working Hours**, use the **Custom working hours** toggle, and then choose when you would like to receive push notifications.

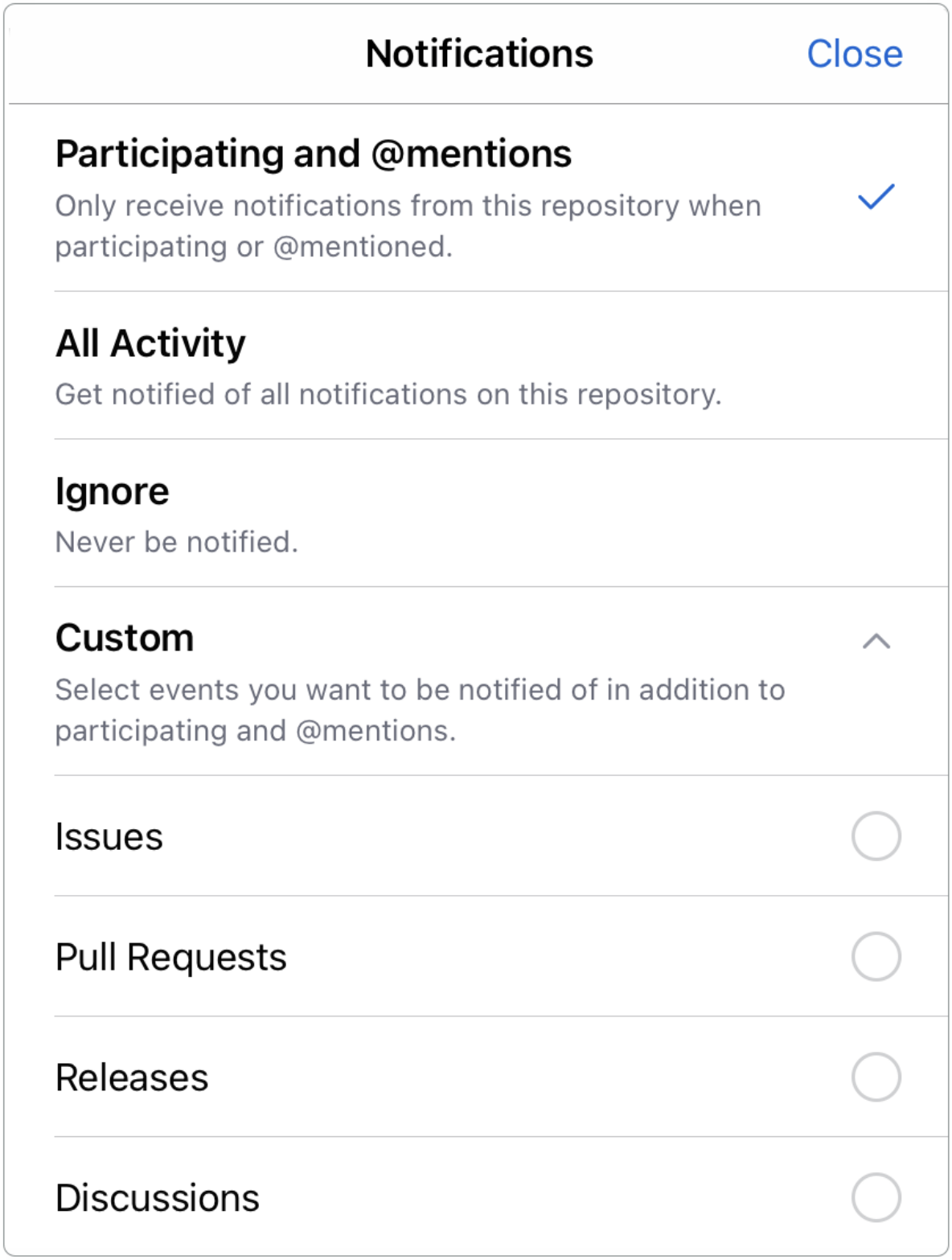
Configuring your watch settings for an individual repository with GitHub for mobile

You can choose whether to watch or unwatch an individual repository. You can also choose to only be notified of certain event types such as issues, pull requests, discussions (if enabled for the repository) and new releases, or completely ignore an individual repository.

1. On GitHub for mobile, navigate to the main page of the repository.
2. Tap **Watch**.



3. To choose what activities you receive notifications for, tap your preferred watch settings.



Understanding GitHub Actions

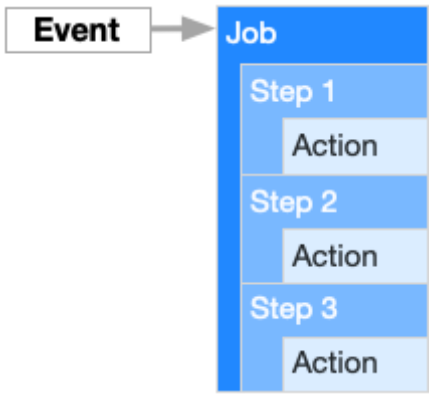
docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Learn the basics of GitHub Actions, including core concepts and essential terminology.

Overview

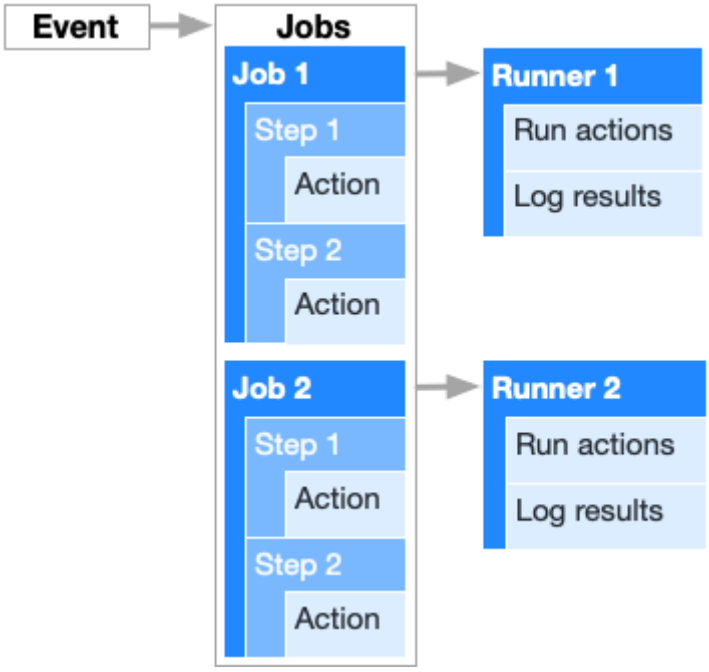
GitHub Actions help you automate tasks within your software development life cycle. GitHub Actions are event-driven, meaning that you can run a series of commands after a specified event has occurred. For example, every time someone creates a pull request for a repository, you can automatically run a command that executes a software testing script.

This diagram demonstrates how you can use GitHub Actions to automatically run your software testing scripts. An event automatically triggers the *workflow*, which contains a *job*. The job then uses *steps* to control the order in which *actions* are run. These actions are the commands that automate your software testing.



The components of GitHub Actions

Below is a list of the multiple GitHub Actions components that work together to run jobs. You can see how these components interact with each other.



Workflows

The workflow is an automated procedure that you add to your repository. Workflows are made up of one or more jobs and can be scheduled or triggered by an event. The workflow can be used to build, test, package, release, or deploy a project on GitHub. You can reference a workflow within another workflow, see "Reusing workflows."

Events

An event is a specific activity that triggers a workflow. For example, activity can originate from GitHub when someone pushes a commit to a repository or when an issue or pull request is created. You can also use the repository dispatch webhook to trigger a workflow when an external event occurs. For a complete list of events that can be used to trigger workflows, see Events that trigger workflows.

Jobs

A job is a set of steps that execute on the same runner. By default, a workflow with multiple jobs will run those jobs in parallel. You can also configure a workflow to run jobs sequentially. For example, a workflow can have two sequential jobs that build and test code, where the test job is dependent on the status of the build job. If the build job fails, the test job will not run.

Steps

A step is an individual task that can run commands in a job. A step can be either an *action* or a shell command. Each step in a job executes on the same runner, allowing the actions in that job to share data with each other.

Actions

Actions are standalone commands that are combined into *steps* to create a *job*. Actions are the smallest portable building block of a workflow. You can create your own actions, or use actions created by the GitHub community. To use an action in a workflow, you must include it as a step.

Runners

A runner is a server that has the GitHub Actions runner application installed. You can use a runner hosted by GitHub, or you can host your own. A runner listens for available jobs, runs one job at a time, and reports the progress, logs, and results back to GitHub. GitHub-hosted runners are based on Ubuntu Linux, Microsoft Windows, and macOS, and each job in a workflow runs in a fresh virtual environment. For information on GitHub-hosted runners, see "About GitHub-hosted runners." If you need a different operating system or require a specific hardware configuration, you can host your own runners. For information on self-hosted runners, see "Hosting your own runners."

Create an example workflow

GitHub Actions uses YAML syntax to define the events, jobs, and steps. These YAML files are stored in your code repository, in a directory called `.github/workflows`.

You can create an example workflow in your repository that automatically triggers a series of commands whenever code is pushed. In this workflow, GitHub Actions checks out the pushed code, installs the software dependencies, and runs `bats -v`.

1. In your repository, create the `.github/workflows/` directory to store your workflow files.
2. In the `.github/workflows/` directory, create a new file called `learn-github-actions.yml` and add the following code.

```
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

3. Commit these changes and push them to your GitHub repository.

Your new GitHub Actions workflow file is now installed in your repository and will run automatically each time someone pushes a change to the repository. For details about a job's execution history, see "Viewing the workflow's activity."

Understanding the workflow file

To help you understand how YAML syntax is used to create a workflow file, this section explains each line of the introduction's example:

`name: learn-github-actions`	*Optional* - The name of the workflow as it will appear in the Actions tab of the GitHub repository.
`on: [push]`	Specify the event that automatically triggers the workflow file. This example uses the `push` event, so that the jobs run every time someone pushes a change to the repository. You can set up the workflow to only run on certain branches, paths, or tags. For syntax examples including or excluding branches, paths, or tags, see "Workflow syntax for GitHub Actions."
`jobs:`	Groups together all the jobs that run in the `learn-github-actions` workflow file.
`check-bats-version:`	Defines the name of the `check-bats-version` job stored within the `jobs` section.
`  runs-on: ubuntu-latest`	Configures the job to run on an Ubuntu Linux runner. This means that the job will execute on a fresh virtual machine hosted by GitHub. For syntax examples using other runners, see "Workflow syntax for GitHub Actions."
`  steps:`	Groups together all the steps that run in the `check-bats-version` job. Each item nested under this section is a separate action or shell command.
`    - uses: actions/checkout@v2`	The `uses` keyword tells the job to retrieve `v2` of the community action named `actions/checkout@v2`. This is an action that checks out your repository and downloads it to the runner, allowing you to run actions against your code (such as testing tools). You must use the checkout action any time your workflow will run against the repository's code or you are using an action defined in the repository.

`    - uses: actions/setup-node@v2`
`      with:`
`        node-version: '14'`
	This step uses the `actions/setup-node@v2` action to install the specified version of the `node` software package on the runner, which gives you access to the `npm` command.

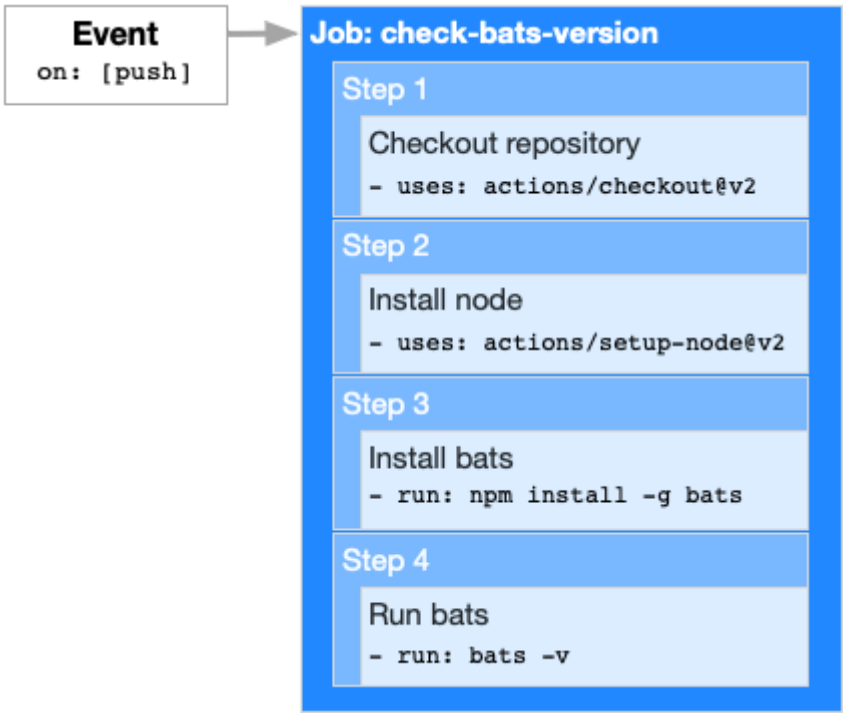
`    - run: npm install -g bats`	The `run` keyword tells the job to execute a command on the runner. In this case, you are using `npm` to install the `bats` software testing package.

```
    - run:
bats -v
```

Finally, you'll run the `bats` command with a parameter that outputs the software version.

Visualizing the workflow file

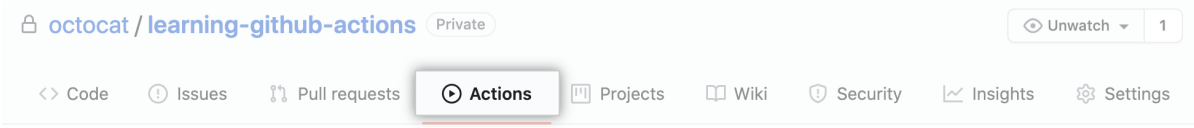
In this diagram, you can see the workflow file you just created and how the GitHub Actions components are organized in a hierarchy. Each step executes a single action or shell command. Steps 1 and 2 use prebuilt community actions. Steps 3 and 4 run shell commands directly on the runner. To find more prebuilt actions for your workflows, see "Finding and customizing actions."



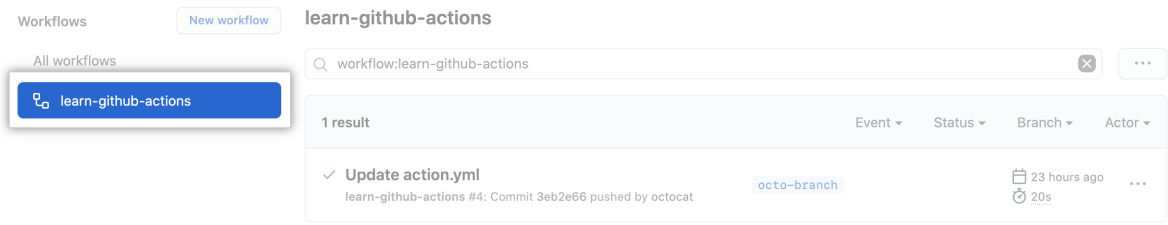
Viewing the job's activity

Once your job has started running, you can see a visualization graph of the run's progress and view each step's activity on GitHub.

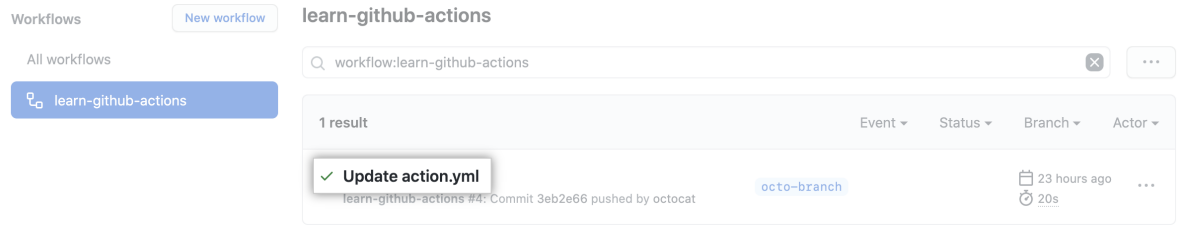
1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.



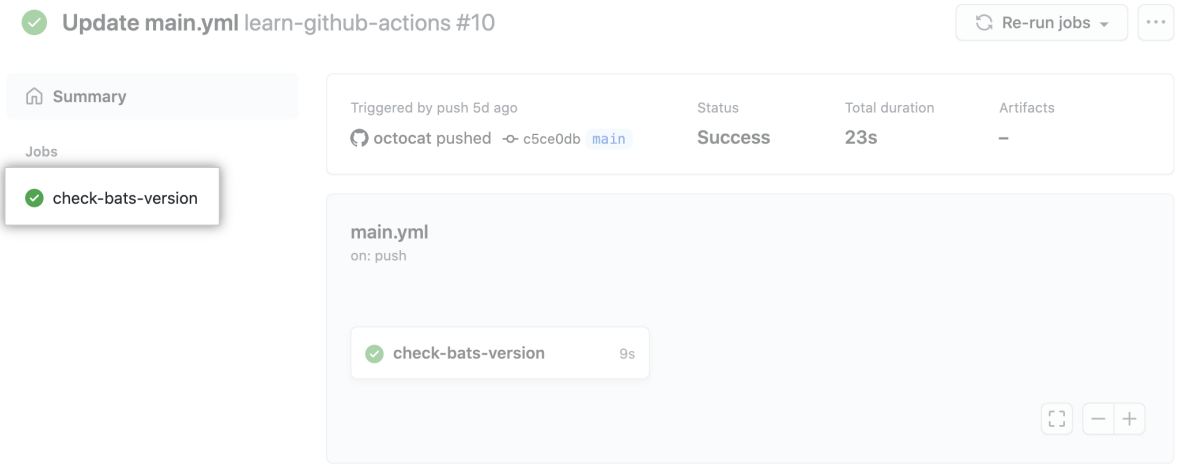
3. In the left sidebar, click the workflow you want to see.



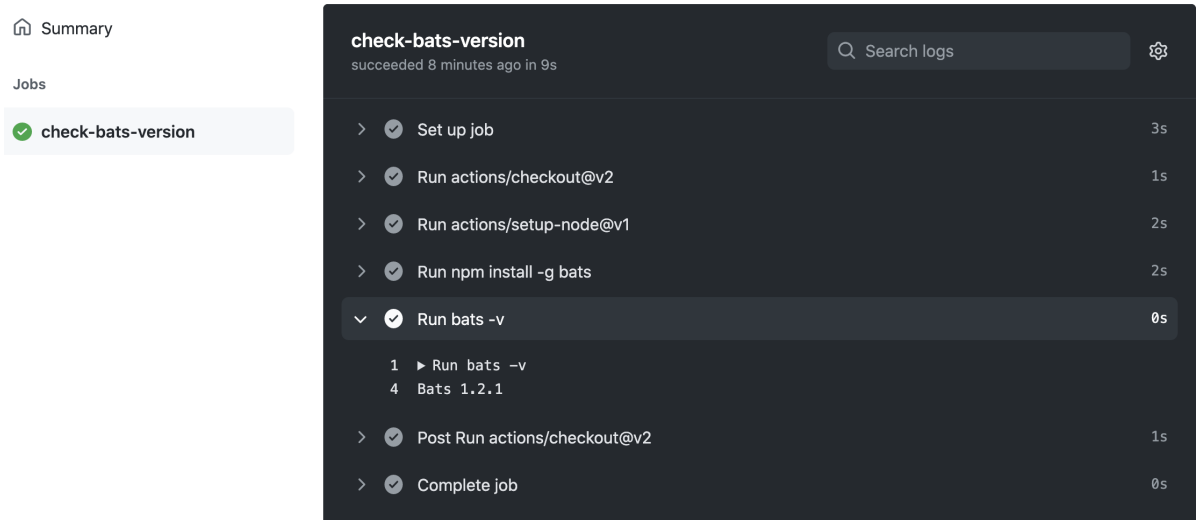
4. Under "Workflow runs", click the name of the run you want to see.



5. Under **Jobs** or in the visualization graph, click the job you want to see.



6. View the results of each step.



Next steps

To continue learning about GitHub Actions, see "Finding and customizing actions."

To understand how billing works for GitHub Actions, see "About billing for GitHub Actions".

Contacting support

If you need help with anything related to workflow configuration, such as syntax, GitHub-hosted runners, or building actions, look for an existing topic or start a new one in the GitHub Community Support's GitHub Actions category.

If you have feedback or feature requests for GitHub Actions, share those in the Feedback form for GitHub Actions.

Contact GitHub Support for any of the following, whether your use or intended use falls into the usage limit categories:

- If you believe your account has been incorrectly restricted
- If you encounter an unexpected error when executing one of your Actions, for example: a unique ID
- If you encounter a situation where existing behavior contradicts expected, but not always documented, behavior

Getting changes from a remote repository

docs.github.com/en/get-started/using-git/getting-changes-from-a-remote-repository

You can use common Git commands to access remote repositories.

Options for getting changes

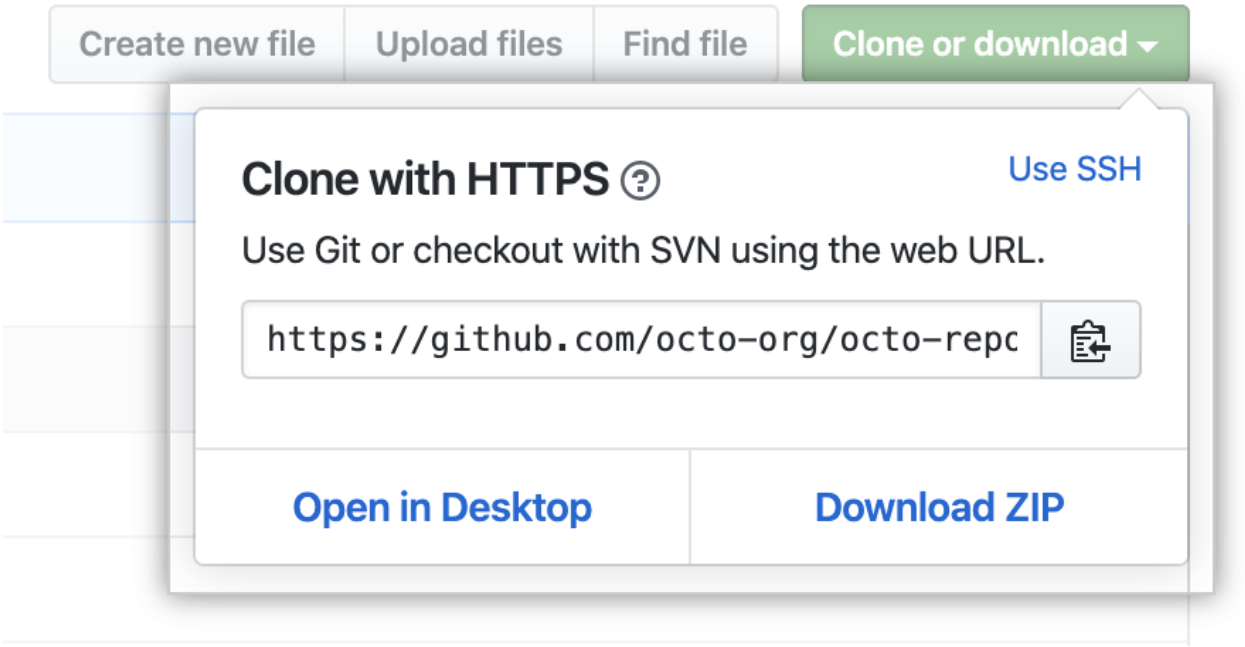
These commands are very useful when interacting with a remote repository. `clone` and `fetch` download remote code from a repository's remote URL to your local computer, `merge` is used to merge different people's work together with yours, and `pull` is a combination of `fetch` and `merge`.

Cloning a repository

To grab a complete copy of another user's repository, use `git clone` like this:

```
$ git clone https://github.com/USERNAME/REPOSITORY.git
# Clones a repository to your computer
```

You can choose from several different URLs when cloning a repository. While logged in to GitHub, these URLs are available below the repository details:



When you run `git clone`, the following actions occur:

- A new folder called `repo` is made
- It is initialized as a Git repository
- A remote named `origin` is created, pointing to the URL you cloned from
- All of the repository's files and commits are downloaded there
- The default branch is checked out

For every branch `foo` in the remote repository, a corresponding remote-tracking branch `refs/remotes/origin/foo` is created in your local repository. You can usually abbreviate such remote-tracking branch names to `origin/foo`.

Fetching changes from a remote repository

Use `git fetch` to retrieve new work done by other people. Fetching from a repository grabs all the new remote-tracking branches and tags *without* merging those changes into your own branches.

If you already have a local repository with a remote URL set up for the desired project, you can grab all the new information by using `git fetch *remotename*` in the terminal:

```
$ git fetch remotename
# Fetches updates made to a remote repository
```

Otherwise, you can always add a new remote and then fetch. For more information, see "Managing remote repositories."

Merging changes into your local branch

Merging combines your local changes with changes made by others.

Typically, you'd merge a remote-tracking branch (i.e., a branch fetched from a remote repository) with your local branch:

```
$ git merge remotename/branchname
# Merges updates made online with your local work
```

Pulling changes from a remote repository

`git pull` is a convenient shortcut for completing both `git fetch` and `git merge` in the same command:

```
$ git pull remotename branchname
# Grabs online updates and merges them with your local work
```

Because `pull` performs a merge on the retrieved changes, you should ensure that your local work is committed before running the `pull` command. If you run into a merge conflict you cannot resolve, or if you decide to quit the merge, you can use `git merge --abort` to take the branch back to where it was in before you pulled.

Further reading

gh repo clone

Clone a repository locally

Synopsis

Clone a GitHub repository locally.

If the "OWNER/" portion of the "OWNER/REPO" repository argument is omitted, it defaults to the name of the authenticating user.

Pass additional 'git clone' flags by listing them after '--'.

```
gh repo clone <repository> [<directory>] [-- <gitflags>...]
```

Options inherited from parent commands

```
--help   Show help for command
```

In use

Using OWNER/REPO syntax

You can clone any repository using OWNER/REPO syntax.

CLI manual menu

```
~/Projects$ cd cli
~/Projects/cli$
```

Using other selectors

You can also use GitHub URLs to clone repositories.

```
# Cloning a repository
~/Projects/my-project$ gh repo clone https://github.com/cli/cli
Cloning into 'cli'...
remote: Enumerating objects: 99, done.
remote: Counting objects: 100% (99/99), done.
remote: Compressing objects: 100% (76/76), done.
remote: Total 21160 (delta 49), reused 35 (delta 18), pack-reused 21061
Receiving objects: 100% (21160/21160), 57.93 MiB | 10.82 MiB/s, done.
Resolving deltas: 100% (16051/16051), done.

~/Projects/my-project$
```


CLI manual menu

CLI manual menu

Fork a repo

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

MacWindowsLinux

About forks

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. You can fork a repository to create a copy of the repository and make changes without affecting the upstream repository. For more information, see "Working with forks."

Propose changes to someone else's project

For example, you can use forks to propose changes related to fixing a bug. Rather than logging an issue for a bug you've found, you can:

- Fork the repository.
- Make the fix.
- Submit a pull request to the project owner.

Use someone else's project as a starting point for your own idea.

Open source software is based on the idea that by sharing code, we can make better, more reliable software. For more information, see the "About the Open Source Initiative" on the Open Source Initiative.

For more information about applying open source principles to your organization's development work on GitHub, see GitHub's white paper "An introduction to innersource."

When creating your public repository from a fork of someone's project, make sure to include a license file that determines how you want your project to be shared with others. For more information, see "Choose an open source license" at choosealicense.com.

For more information on open source, specifically how to create and grow an open source project, we've created Open Source Guides that will help you foster a healthy open source community by recommending best practices for creating and maintaining repositories for your open source project. You can also take a free GitHub Learning Lab course on maintaining open source communities.

Prerequisites

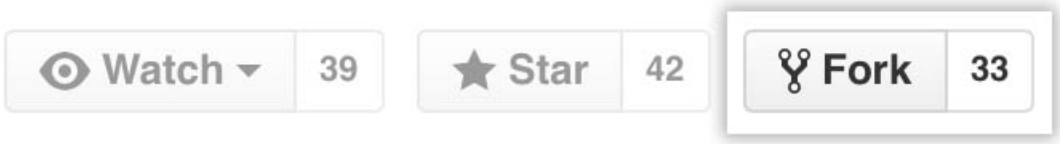
If you haven't yet, you should first set up Git. Don't forget to set up authentication to GitHub from Git as well.

Forking a repository

GitHub.com GitHub CLI Desktop

You might fork a project to propose changes to the upstream, or original, repository. In this case, it's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line. You can practice setting the upstream repository using the same octocat/Spoon-Knife repository you just forked.

1. On GitHub, navigate to the octocat/Spoon-Knife repository.
2. In the top-right corner of the page, click **Fork**.

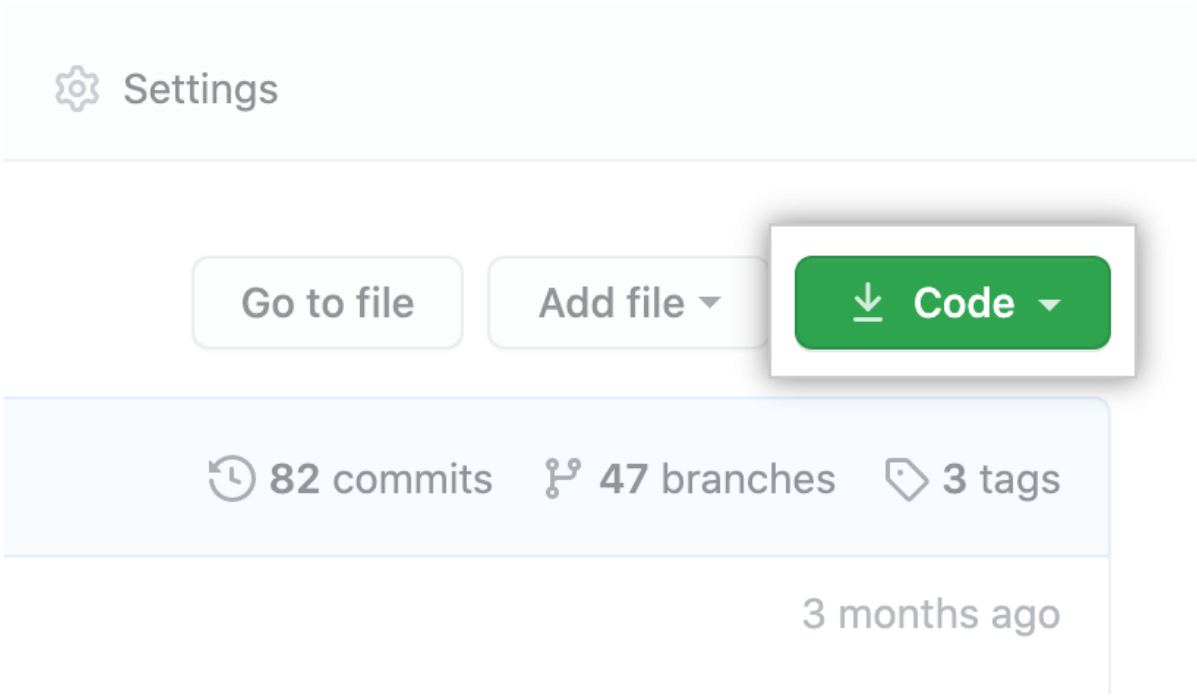


Cloning your forked repository

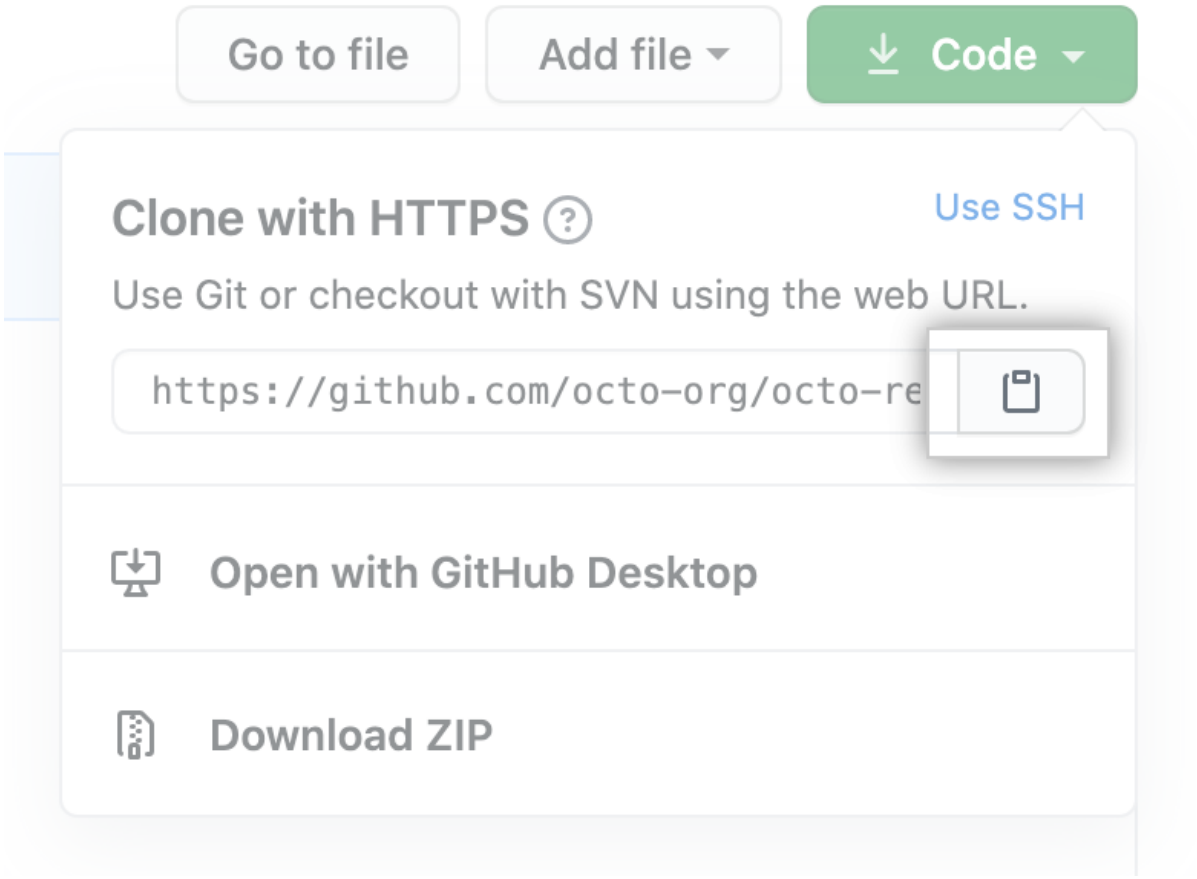
Right now, you have a fork of the Spoon-Knife repository, but you don't have the files in that repository locally on your computer.

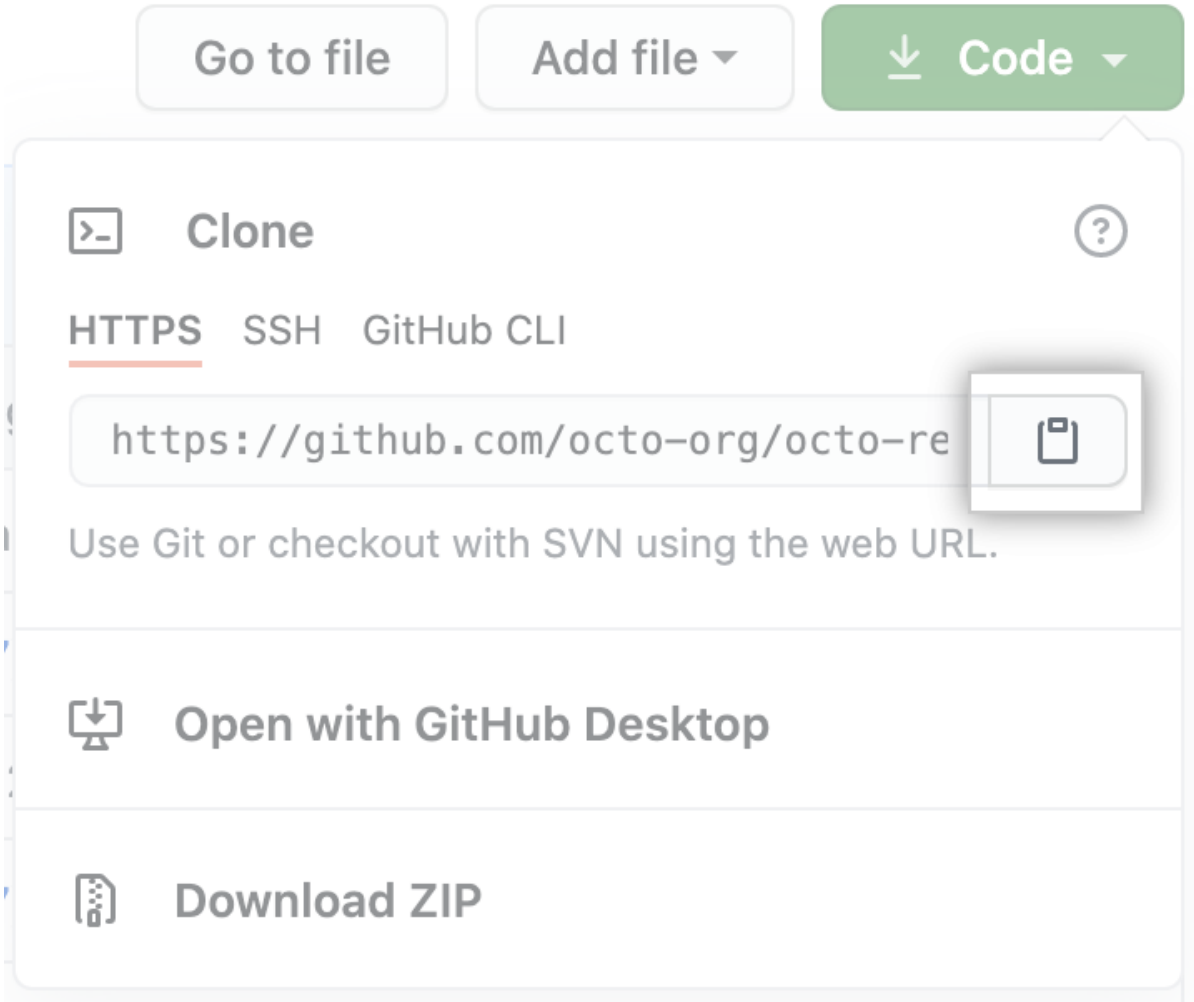
GitHub.com GitHub CLI Desktop

1. On GitHub, navigate to **your fork** of the Spoon-Knife repository.
2. Above the list of files, click **Code**.



3. To clone the repository using HTTPS, under "Clone with HTTPS", click . To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH**, then click . To clone a repository using GitHub CLI, click **Use GitHub CLI**, then click .





4. Open Git Bash.
5. Change the current working directory to the location where you want the cloned directory.
6. Type `git clone`, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of `YOUR-USERNAME`:

   ```
   $ git clone https://github.com/YOUR-USERNAME/Spoon-Knife
   ```

7. Press **Enter**. Your local clone will be created.

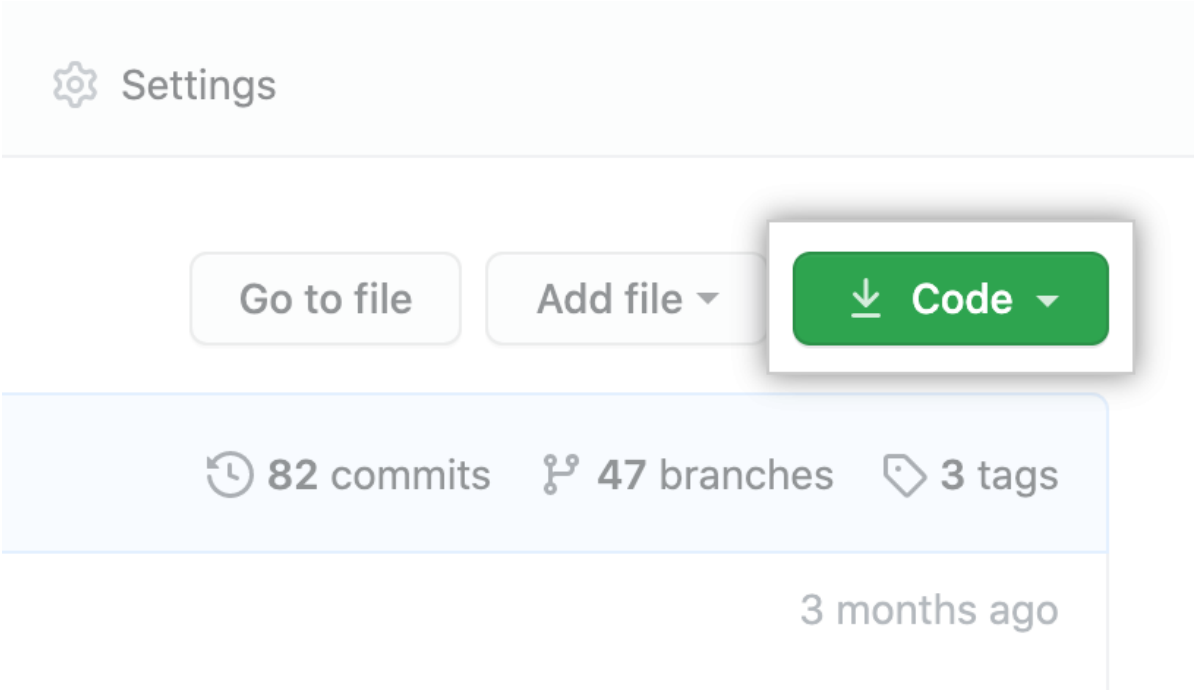
   ```
   $ git clone https://github.com/YOUR-USERNAME/Spoon-Knife
   > Cloning into `Spoon-Knife`...
   > remote: Counting objects: 10, done.
   > remote: Compressing objects: 100% (8/8), done.
   > remove: Total 10 (delta 1), reused 10 (delta 1)
   > Unpacking objects: 100% (10/10), done.
   ```

Configuring Git to sync your fork with the original repository

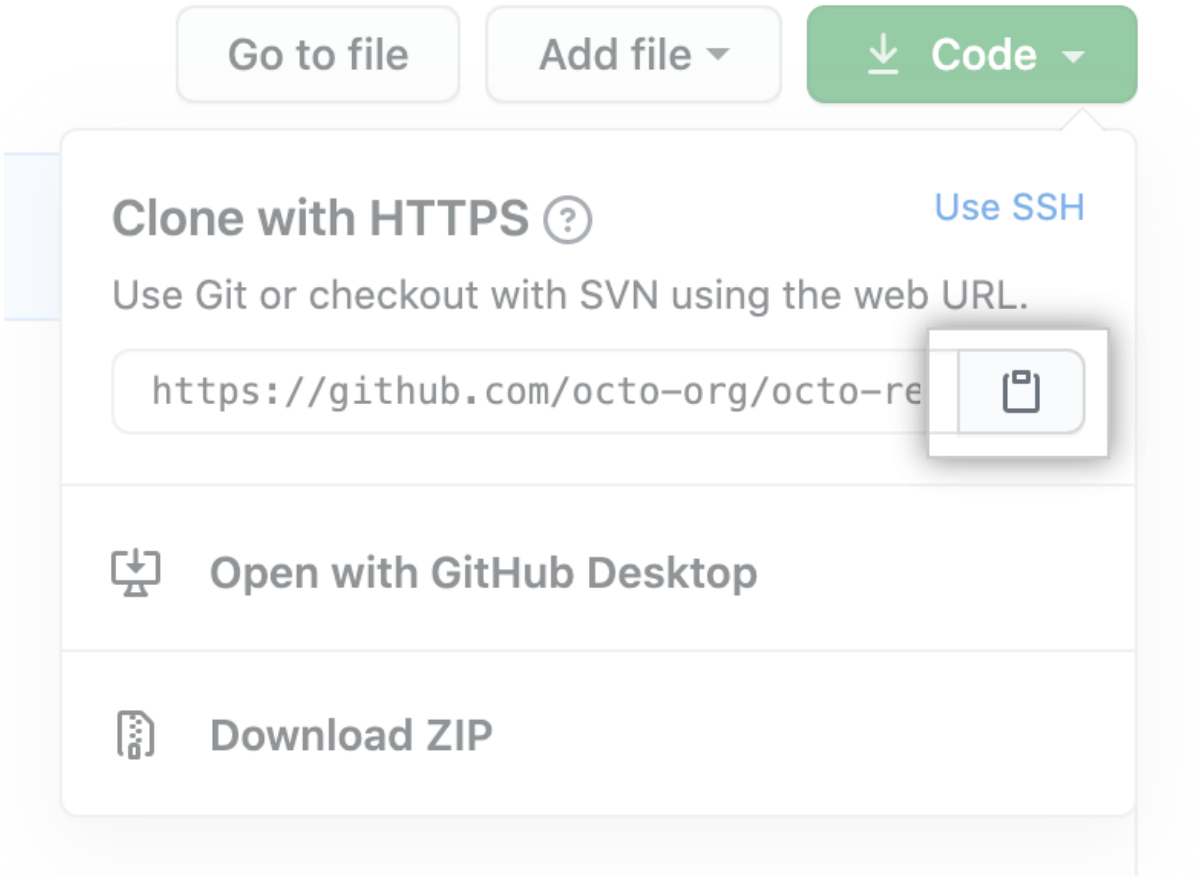
When you fork a project in order to propose changes to the original repository, you can configure Git to pull changes from the original, or upstream, repository into the local clone of your fork.

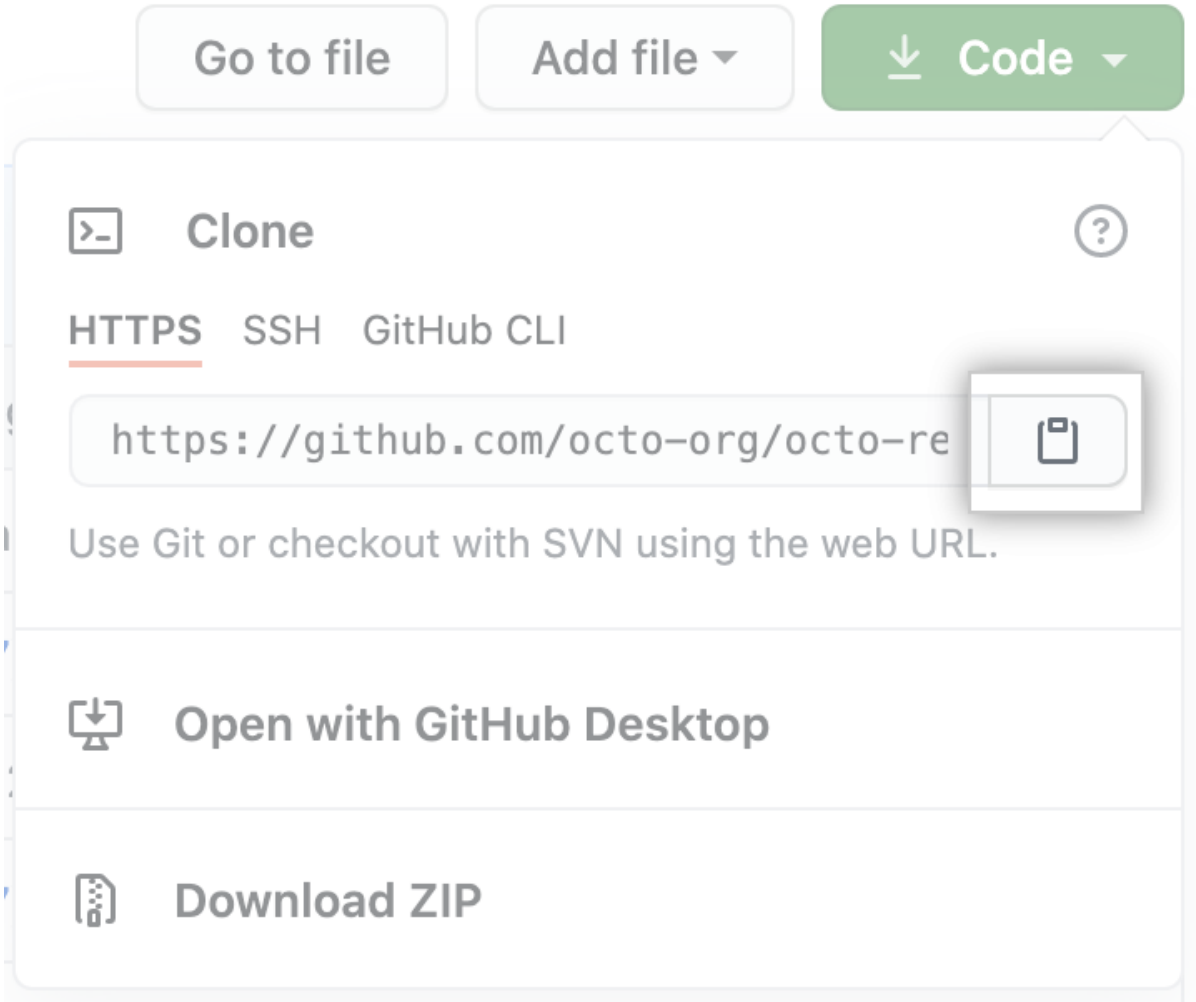
GitHub.com GitHub CLI Desktop

1. On GitHub, navigate to the octocat/Spoon-Knife repository.
2. Above the list of files, click **Code**.



3. To clone the repository using HTTPS, under "Clone with HTTPS", click . To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH**, then click . To clone a repository using GitHub CLI, click **Use GitHub CLI**, then click .





4. Open Git Bash.
5. Change directories to the location of the fork you cloned.
    - To go to your home directory, type just `cd` with no other text.
    - To list the files and folders in your current directory, type `ls`.
    - To go into one of your listed directories, type `cd your_listed_directory`.
    - To go up one directory, type `cd ..`.
6. Type `git remote -v` and press **Enter**. You'll see the current configured remote repository for your fork.

```
$ git remote -v
> origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
```

7. Type `git remote add upstream`, and then paste the URL you copied in Step 2 and press **Enter**. It will look like this:

```
$ git remote add upstream https://github.com/octocat/Spoon-Knife.git
```

8. To verify the new upstream repository you've specified for your fork, type `git remote -v` again. You should see the URL for your fork as `origin`, and the URL for the original repository as `upstream`.

```
$ git remote -v
> origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
> upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
> upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

Now, you can keep your fork synced with the upstream repository with a few Git commands. For more information, see "Syncing a fork."

Next steps

You can make any changes to a fork, including:

- **Creating branches:** *Branches* allow you to build new features or test out ideas without putting your main project at risk.
- **Opening pull requests:** If you are hoping to contribute back to the original repository, you can send a request to the original author to pull your fork into their repository by submitting a pull request.

Find another repository to fork

Fork a repository to start contributing to a project. You can fork a repository to your user account or any organization where you have repository creation permissions. For more information, see "Permission levels for an organization."

If you have access to a private repository and the owner permits forking, you can fork the repository to your user account or any organization on GitHub Team where you have repository creation permissions. You cannot fork a private repository to an organization using GitHub Free. For more information, see "GitHub's products."

You can browse Explore to find projects and start contributing to open source repositories. For more information, see "Finding ways to contribute to open source on GitHub."

Celebrate

You have now forked a repository, practiced cloning your fork, and configured an upstream repository. For more information about cloning the fork and syncing the changes in a forked repository from your computer see "Set up Git."

You can also create a new repository where you can put all your projects and share the code on GitHub. For more information see, "Create a repository."

Each repository in GitHub is owned by a person or an organization. You can interact with the people, repositories, and organizations by connecting and following them on GitHub. For more information see "Be social."

GitHub has a great support community where you can ask for help and talk to people from around the world. Join the conversation on Github Support Community.

Exercise - A guided tour of GitHub

In this exercise you use GitHub Learning Lab to learn about key GitHub features, including issues, notifications, branches, commits, and pull requests.

GitHub Learning Lab is an integrated experience that's easy to use. You get feedback and instructions throughout the lab as you work in your GitHub repository.

Here are a few suggestions to make the Learning Lab exercise more enjoyable.

- GitHub Learning Lab is installed on your account in the first step of this lab. If you're asked, be sure to *install it on all repositories*. This won't affect the organizations that you're a member of, just the personal repositories that the lab creates for you.
- After the install, you may be returned to the main page. To get back to your lab, just use the button on the bottom of this page.
- GitHub will create a repository for you to use. Give permissions to GitHub Learning Lab.
- GitHub Learning Lab will set itself as a reviewer on your pull requests so that it can give you the next steps just in time. Sometimes reviewing your pull request will take a few minutes.
- When you're given a link for creating or editing a file or told to open a tab, **be sure to open it in another tab in your browser**. This way you can come back to the instructions without leaving the file.
- Comments and instructions will continue on your pull request or in an issue on your repository.

When you've finished the exercise in GitHub, return here for:

- A quick knowledge check
- A summary of what you've learned
- To earn a badge for completing this module

Start the learning lab on GitHub

Next unit: Knowledge check

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue.

200 XP

Knowledge check

3 minutes

Choose the best response for each question. Then select "Check your answers."

Check your knowledge

1. What is the best way to report a bug to a GitHub project?

- ○ Send an email to a project owner.
- ○ I don't bother reporting software bugs because there's no transparency and they never get fixed anyway.
- ◉ Search for the bug in the project's existing issues and create a new one if it hasn't been reported yet. ✓

A project's issues are visible to anyone who has access to the project, so you may find a resolution is already planned or available. Otherwise, you can create and track the issue yourself.

2. Suppose you have created a bug fix on a new branch and want it to become part of the next production build generated from the `main` branch. What should you do next?

- ○ Copy your branch changes and commit them directly to the `main` branch.
- ◉ Create a pull request to merge your new branch into the `main` branch. ✓

Pull requests are the correct way to communicate that commits are ready for review and ultimate inclusion on the `main` branch.

- ○ On second thought, maybe I won't share this fix. I'll just put it in my own private version of the source code.

3. Suppose you'd like to work with a project on GitHub but you don't have write access to the project. What can you do to contribute?

- Fork the project's repository to your GitHub account, clone the forked ✓

- [x] repository to your local machine, push changes to your repository, and submit a pull request to the target (upstream) repository.

GitHub provides forking functionality designed to allow you to work with projects where you aren't an owner or don't have write access. Forking makes a remote copy of the project in your repository that you can then clone locally. To submit updates to the target repository (upstream repository) you can submit a pull request.

- [ ] Clone the project to your local machine and push updates directly to the project repository.
- [ ] Use git commands to make a copy of the project so that you can work locally. Submit an issue to get your changes into the target repository.

Next unit: Summary

Continue >

Summary

Completed 100 XP

In this module, you learned about the key features of GitHub, including issues, commits, and pull requests. You also used GitHub Pages to publish a public site based on the contents of your project.

You learned about:

- Communicating with the project community in issues
- Managing notifications for project events
- Creating branches to manage work in parallel
- Making commits to update project source
- Introducing changes with pull requests
- Deploying a web page to GitHub Pages
- Differences between Git and GitHub and the roles they play in the software development lifecycle
- How a repository fork differs from a clone
- Repository labels and where to apply them in issues and pull requests

Now that you're familiar with the basics of GitHub, learn to Upload your project by using GitHub best practices.

Learn more

Here are some links to more information on the topics we discussed in this module.

- Setting up and managing organizations and teams
- Committing changes to your project
- Collaborating with issues and pull requests
- About the role of labels
- GitHub Actions
- Fork a repo
- Working with GitHub Pages

Module complete:

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue.

Introduction to GitHub

A quick reference guide to complete the steps in the course.
https://lab.github.com/courses/introduction-to-github/

1. Assign yourself

On the right side of the issue screen, under the "Assignees" section, click the gear icon and select yourself

2. GitHub Pages

2.1. Click on the **Settings** tab in this repository
2.2. Scroll down to the "GitHub Pages" section
2.3. From the "Source" drop down, select the `master` branch
2.4. Click **Save**

3. Close an issue

Click the **Close issue** button at the bottom of the issue

4. Create a branch

4.1. Navigate to the **Code** tab
4.2. Click **Branch: master** in the drop-down
4.3. In the field, enter a name for your branch
4.4. Click **Create branch: <name>** or press the “Enter” key to create your branch

5. Commit a file

5.1. While on the branch you just created, click **Create new file** on the "Code" tab.
5.2. In the **file name** field, type `_posts/0000-01-02-YOUR-USERNAME.md`
5.3. When you're done naming the file, add the following content to your file:

```
---
layout: slide
title: "Welcome to our second slide!"
---
Your test
Use the left arrow to go back!
```

5.4. After adding the text, you can commit the change by entering a commit message in the text-entry field below the file edit view.
5.5. When you've entered a commit message, click **Commit new file**.

6. Open a pull request

6.1. On the **Code** tab, click **New pull request**
6.2. In the **base:** drop-down menu, make sure the `master` branch is selected
6.3. In the **compare:** drop-down menu, select the branch you recently made your commit on
6.4. When you've selected your branch, enter a title for your pull request, for example `Add username's file`
6.5. The next field helps you provide a description of the changes you made. Feel free to add a description of what you've accomplished so far. As a reminder, you have: `created a branch, created a file and made a commit, and opened a pull request`
6.6. Click **Create pull request**

7. Respond to a review

7.1. Click the **Files Changed** tab in this pull request
7.2. Click on the pencil (✏) icon found on the right side of the screen
7.3. Replace line 5 with something new
7.4. Scroll to the bottom and click **Commit Changes**.

8. Merge your pull request

8.1. Click **Merge pull request**
8.2. Click **Confirm merge**
8.3. Once your branch has been merged, you don't need it anymore. Click **Delete branch**.