

- » Setting up a repository for OSS
- » Managing OSS workflows
- » Ending an OSS project

GO TO LAST PAGE 20

## Chapter **10**

# Starting Your Own OSS

**T**here comes a point in many developers' lives when they have their own code to share with the world. Many different approaches and motivations may drive a person to share code. In some cases, a person may write some code they think others will find interesting and just “toss it over the wall” without any intent to take contributions back. On the other end is the case where someone wants to start a real movement with large numbers of contributors.

Whatever your motivation, the only requirement for open source software (OSS) is to choose a license that complies with the Open Source Definition (OSD). You can find the definition at <https://opensource.org/osd>. The Open Source Initiative (OSI) has a process for approving licenses, and its site lists a plethora of licenses.

But for the life of most open source software, choosing a license is just a starting point. A lot more goes with managing an open source project. In this chapter, I cover what it means to start an open source project on GitHub, maintain it, and if it comes to it, sunset it.

## Creating an Open Source Repository

When you create a repository with the intent to make it open source, it's common to make the repository public and select a license during the repository creation process.

Sometimes you want to defer those choices to later. Perhaps you want to set up your repository right before you make it public to the world. Or you may want to spend more time thinking about the license. The following sections focus on the scenario of turning an existing private repository into an open source one. I assume that you've already created a repository with a `README.md` file, but didn't make the repository public nor did you choose a license. (If you haven't, see Chapter 3.)

## Adding a license

The only requirement that software has to be considered open source is to have an open source license. GitHub makes adding a license to an existing project easy.

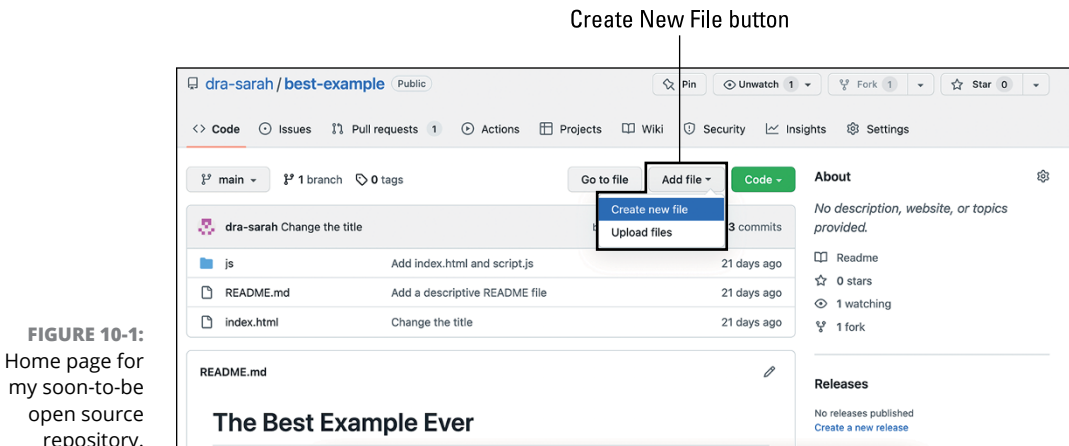
Figure 10-1 shows the home page for a private repository without a license. To add a license:

1. **Click the Create New File button.**

After you click the button, GitHub prompts you to name the file.

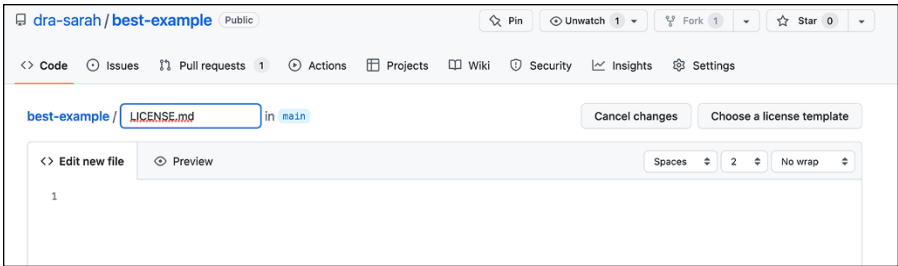
2. **Type your filename.**

For this example, I named mine `LICENSE`. GitHub notices you're attempting to create a license file and helpfully adds a button to the right labeled Choose a License Template, as shown in Figure 10-2.



**FIGURE 10-1:** Home page for my soon-to-be open source repository.

**FIGURE 10-2:**  
Choose a  
License Template  
button magically  
appears.



**3. Click the Choose a License Template button to see a list of license templates.**

The three most common licenses used on GitHub are listed first in bold, as shown in Figure 10-3. Chapter 3 provides a brief guide on choosing a license, but this page has a link to a more detailed guide to help you pick which license best fits your project.

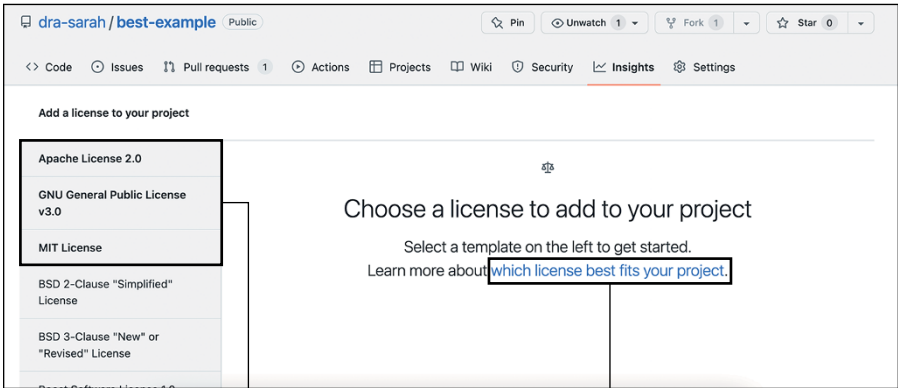
**4. After you know which license you want, click the license to view information about the license.**

This page includes a high-level summary with bullet points about the key traits of the license. This overview helps you gain an understanding of the license without having to wade through all the legalese.

Of course, the page also presents the full text of the license for those who relish legalese. On the right are fields required by the license.

**5. Enter your information in the fields to customize the license to your situation.**

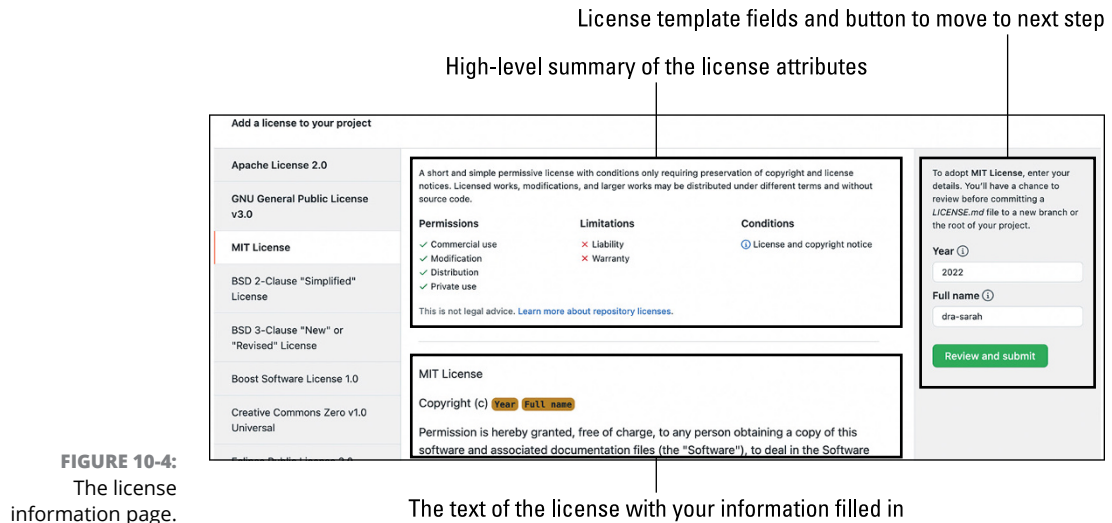
You can see an example of this page in Figure 10-4.



**FIGURE 10-3:**  
The license  
chooser page.

The three most popular licenses on GitHub

Link to more information on choosing a license



6. After you're satisfied with everything, click the **Review and Submit** button.

You return to the file creation page with the text of the license filled in.

7. Scroll down to enter your commit message and click **Commit New File** to create the license file.

## Adding contributor guidelines

Another useful document to include in your repository is a file named `CONTRIBUTING.md`. This file provides information on how to contribute to your project. It should include information such as where people should ask questions and where to provide feedback.

Chapter 9 provides more details on what a contributor can expect to find in a `CONTRIBUTING.md` document. That chapter is a good place to start for your own project's contribution guidelines.



REMEMBER

A robust `CONTRIBUTING.md` document saves you and your future contributors a lot of headache and time. Be sure to revisit it from time to time to make sure it's up to date.

## Adding a code of conduct

A code of conduct makes behavioral expectations clear for everyone who participates with your project, whether they contribute code, comment on an issue, or

otherwise interact with others on your repository. A code of conduct sets the tone for the type of open and welcoming environment you want to foster for your project.

To add a code of conduct, follow the same process as you do when adding a license, but name the file `CODE_OF_CONDUCT.md`. (See the “Adding a license” section, earlier in this chapter.) This process is covered in Chapter 9 as well.

## Making a Repository Public

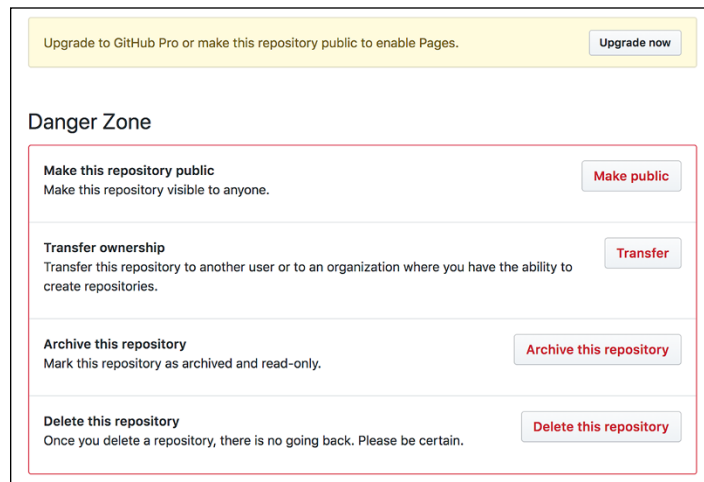
At this point, your repository has the basic items it needs to go public. To change a repository from private to public:

1. Visit the **Settings** page for your repository.
2. Scroll to the bottom of the Settings page to reach the **Danger Zone**, as shown in Figure 10-5.
3. Click the **Make Public** button to initiate making a repository public.



WARNING

This step is a potentially dangerous operation because you may inadvertently expose information to the world you’d rather keep private. So make sure you really are ready to make this repository public. If you created it with the intention of making it public from the beginning, the repository shouldn’t contain any secrets.

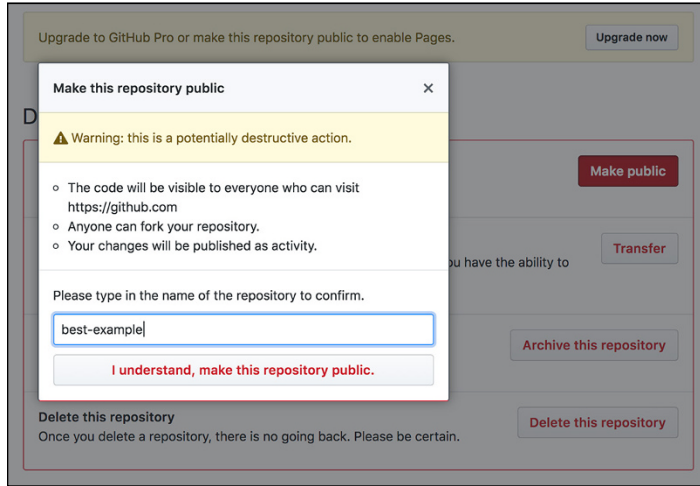


**FIGURE 10-5:**  
Danger zone:  
where you can  
make a private  
repo public.

GitHub displays a confirmation dialog after you click the Make Public button asking you to type the name of the repository to confirm this change, as shown in Figure 10-6.

**4. Type the repository name and then click the I Understand, Make This Repository Public button to make the repository public.**

The Settings page now has a section that comes in handy as you manage your new open source project.



**FIGURE 10-6:**  
Confirmation to  
convert a private  
repo to public.

## Enforcing a Code of Conduct

A *code of conduct* is a great way to communicate behavioral expectations for a community. But a code of conduct isn't enough to create a healthy community on its own; you have to enforce it.



**WARNING**

It would be wonderful if you never had to enforce your code of conduct. Most people who participate in your project will be helpful and kind. However, as your project grows and more people get involved, it's almost inevitable that you'll have to step in.

## Responding with kindness

Keep in mind that even when someone acts in a matter that doesn't fit in the spirit of your code of conduct, they may not necessarily be malicious or a bad actor. Maybe they're having a bad day, and they took it out on your repository. That doesn't excuse their behavior, but it is relatable. We've all done it.



TIP

If someone makes an off color or angry comment in an issue in your repository that goes against the type of community you want to foster, it's often disarming to reply back with a kind response that expresses empathy for their situation, but is clear that the comment doesn't meet community guidelines. Be specific in how it does not meet the guidelines and if you can, suggest an alternative approach they could have taken.

For example, if someone comments on the repository that the maintainer must hate other people because they haven't fixed a particular bug, you can respond with something like "Personal attacks are not welcome in this repository. We are all volunteers trying to work on this in our spare time. I understand that you are frustrated. Instead of a personal attack, perhaps write about how the bug affects you and why that is frustrating to you."



REMEMBER

But always remember, you don't owe anybody anything. You're not there to be a punching bag for people. If they're abusive or if you're simply tired of a thousand paper cuts of negative comments, rather than respond while angry, take a break. Walk away. Ask for help. Your own mental well-being comes first.

## Leveraging the ban hammer

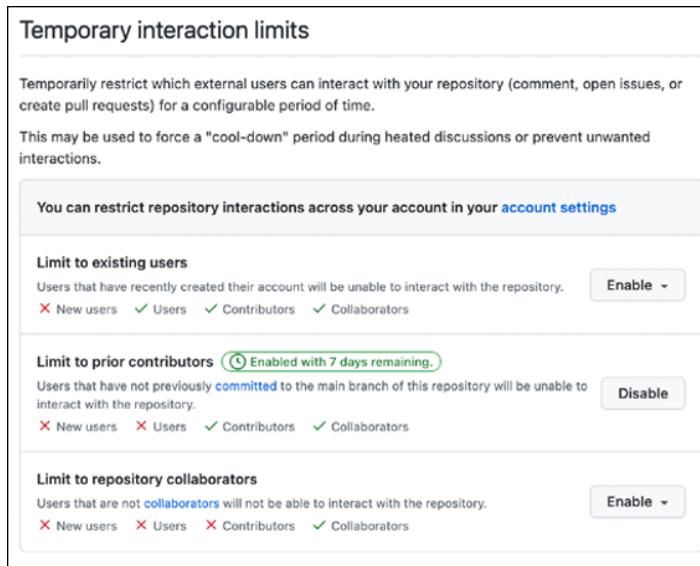
It would be nice if a kind admonishment worked in every situation (see preceding section), but sometimes tempers flare. We're all human, and sometimes we lose our temper. So will people who participate in your repository. This situation is where temporary interaction limits come in handy.

At the very bottom of the Settings page for your public repository (this setting doesn't exist for a private repository) is a moderation section with a link to Interaction Limits. Click this link to see the set of limits available. Figure 10-7 shows this page after enabling the Limit to Prior Contributors option for 1 week.



REMEMBER

These limits are not permanent bans. They are cool-down periods. While interaction limits across your repository remain in effect until you disable them, an individual restriction is intended to be a limit and not a ban. This tool is great to use when a discussion or an issue gets a bit too heated. It gives everyone time to cool down and prevents the discussion from running even further out of control. If a user who is not authorized to make changes to the repository attempts to make a change, they are met with a message. For example, if someone who hasn't contributed to a repository yet tries to open a pull request on a repository that has a limit to prior contributors, they see a message that states: "An owner of this repository has limited the ability to open a pull request to users who have contributed to this repository in the past."



**FIGURE 10-7:**  
The set of  
interaction limits.

## Blocking users

Every once in a while, you may encounter a truly abusive person in your project. You've tried to kill them with kindness. You've tried the 24-hour cool-down periods, but they persist in being rude or antagonistic. It's time to block or report the user.

To block or report the user:

- 1. Click the user's name next to their comment to visit their profile page.**

Underneath their profile picture is a link to block or report the user, as shown in Figure 10-8.

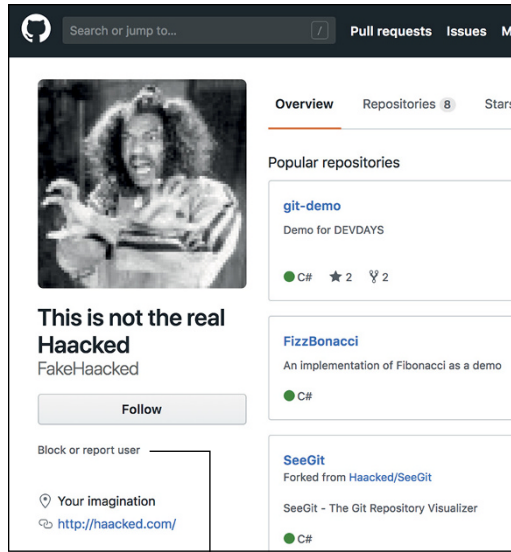
- 2. Click the link to block or report the user.**

A dialog box with two options appears: Block User or Report Abuse, as shown in Figure 10-9.

- 3. Click Block User to immediately block the user.**

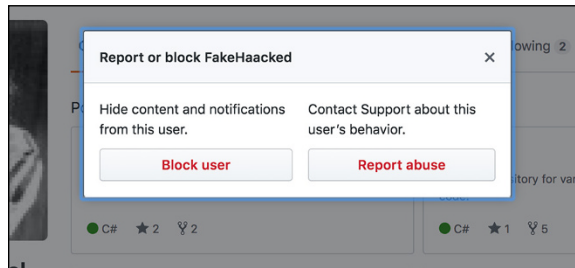
Blocking a user denies a user from accessing your activity and repositories. It also prevents them from sending you notifications, such as when they @ mention your username. They pretty much no longer exist as far as you are concerned, and vice versa. For more details on what happens when you block a user, visit <https://docs.github.com/communities/maintaining-your-safety-on-github/blocking-a-user-from-your-personal-account>.





**FIGURE 10-8:** Profile page with link to block or report user.

Link to block or report user



**FIGURE 10-9:** Dialog box with Block User and Report Abuse buttons.



You can also block users, see a list of users you currently have blocked, and unblock users from your account Settings page by going to [https://github.com/settings/blocked\\_users](https://github.com/settings/blocked_users).

## Writing a README.md File

The `README.md` file holds a special significance on GitHub. When you visit a repository's home page, GitHub displays the contents of the `README.md` rendered as Markdown on the home page. The `README.md` contents is what visitors to your repository expect to see to learn more about your project.

Chapter 3 includes information about what should be included in a `README.md` file. It should also include links to your `CONTRIBUTING.md` and your `CODE_OF_CONDUCT.md` files. After reading this file, a person should understand what your project does, how to get involved, and where to go to find out more information.

## Writing Good Documentation

One of the most neglected aspects of open source is documentation. Writing good docs is a great way to distinguish your project from the rest. Also, documentation provides a nice way for others to dip their toes into open source.



TIP

GitHub supports a wiki feature, but I recommend using a relatively unknown feature, serving a documentation site from the `docs` folder in the `main` branch of your repository. This feature has many benefits, but one of the big ones is the ability to version your documentation along with the code that it documents.

Chapter 4 walks through building a GitHub Pages site. Serving your documentation from the `docs` folder is very similar to that process. The following steps assume that you have a repository that is not a GitHub Pages website. Chapter 3 walks through creating a regular repository if you need a refresher.

Make sure to commit a file named `index.html` to the `docs` folder in the `main` branch of your repository. This feature requires that there's already a `docs` folder before you can enable it. If you're unclear about how to do that, Chapter 7 covers writing and committing code in detail.

Navigate to your repository settings and then scroll down to the GitHub Pages section. By default, the Source is set to None. Click the button and select the `main` branch and then specify the `/docs` folder as shown in Figure 10-10 and then click the Save button.

You can now visit your documentation site at `https://{USERNAME}.github.io/{REPO-NAME}/`. To see an example of this in action, visit `https://dra-sarah.github.io/best-example/`.

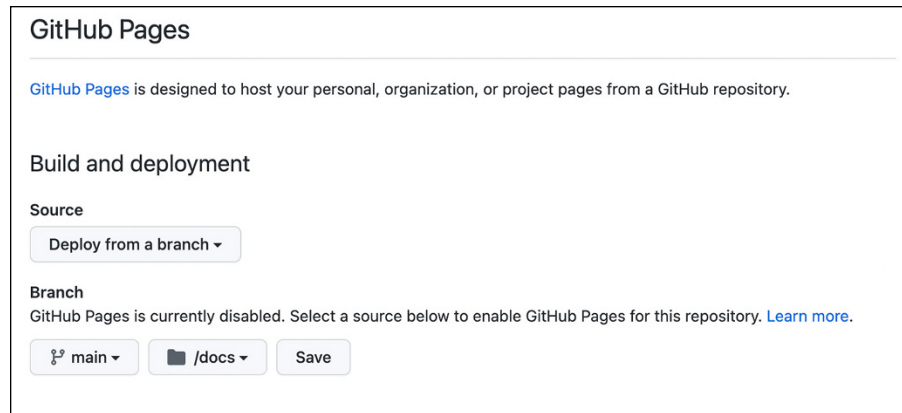
As I cover in Chapter 4, you can select a theme and add a custom domain name if needed.



REMEMBER

Don't forget to add a link to the docs site to your `README` file so that people can find it.

**FIGURE 10-10:**  
Serving a GitHub  
Pages site from  
the docs folder.



Alternatively, you can deploy GitHub pages using a GitHub Action. Select GitHub Actions instead of Branch under Source, and choose one of the templates for Jekyll or static HTML.



TIP

If you want to still deploy only your docs subfolder, make sure you change the GitHub Action to point to `docs`` as the the path instead of `..`

## Managing Issues

As the word gets out about your open source project, people will start to try it out. And at some point, people will open issues. Congratulations! When an issue is opened, it means someone out there cared enough to report a bug, ask a question, or make a feature request. This interest is a good thing for an open source project.

But if your project becomes very successful, the influx of new issues can start to get overwhelming. In the following sections, I cover some tips on managing incoming issues.

### Labeling issues

GitHub provides a default set of issue labels when you create a website.

- » bug
- » duplicate
- » enhancement

- » good first issue
- » help wanted
- » invalid
- » question
- » ready for review
- » won't fix

You can manage these labels by going to the Issues tab and clicking the Labels button. Make sure the set of labels makes sense for your project. Assigning labels helps you manage issues as they come in.

## Triaging issues

As new issues are created, it's important to triage issues. The word *triage* comes from the medical field. Merriam-Webster defines triage as “the sorting of patients (as in an emergency room) according to the urgency of their need for care.”

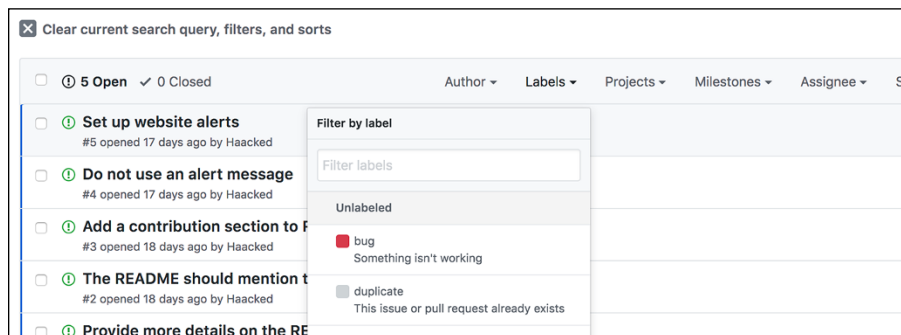
Triaging issues is similar to triaging in a hospital. *Triage*, when applied to issues, is the process of reviewing new issues and assigning labels to indicate what type of issue are they (bug report, feature request, and so on), determining their priority, and assigning issues to people. In some cases, you may close issues you don't plan to address during triage.



TIP

Every triaged issue should have some label applied. If necessary, you can even create a triaged label for this purpose. Assigning a triaged label to issues that you've triaged makes it possible to see all issues that haven't yet been triaged by using the unlabeled filter, as shown in Figure 10-11.

**FIGURE 10-11:**  
Unlabeled  
issues filter.



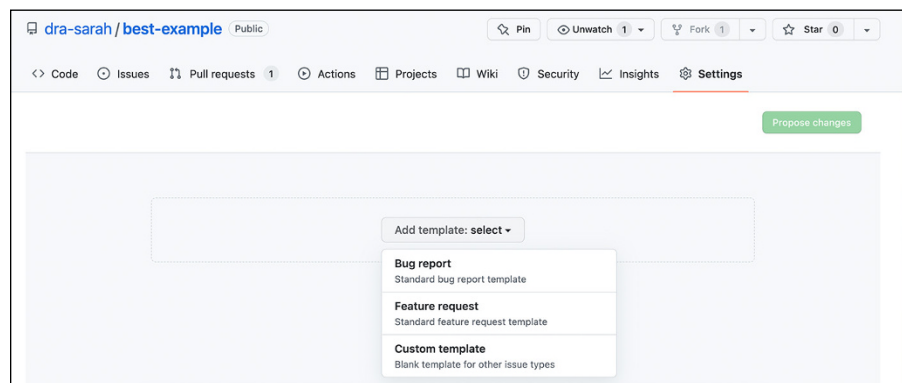
To maintain a sense of sanity, it often makes sense to schedule triage on a regular cadence as opposed to doing it all the time as people create new issues.

## Issue templates

It would be great if everyone who submits an issue understood how to write a proper bug report or if they understood what makes for a good feature request. A bug report that just says “Your stuff is broken” is not too helpful to the repository maintainer.

To help foster good issues, you can set up issue templates that guide people filing issues on your repository to supply the information you need.

In the Settings page for your repository, in the Issues section in the main pane, is a big green button labeled Set Up Templates. Clicking that button takes you to a templates setup page where you can choose from a set of pre-existing issue templates or create your own, as shown in Figure 10-12.



**FIGURE 10-12:**  
Choosing an issue  
template.

From the templates setup page, follow these steps:

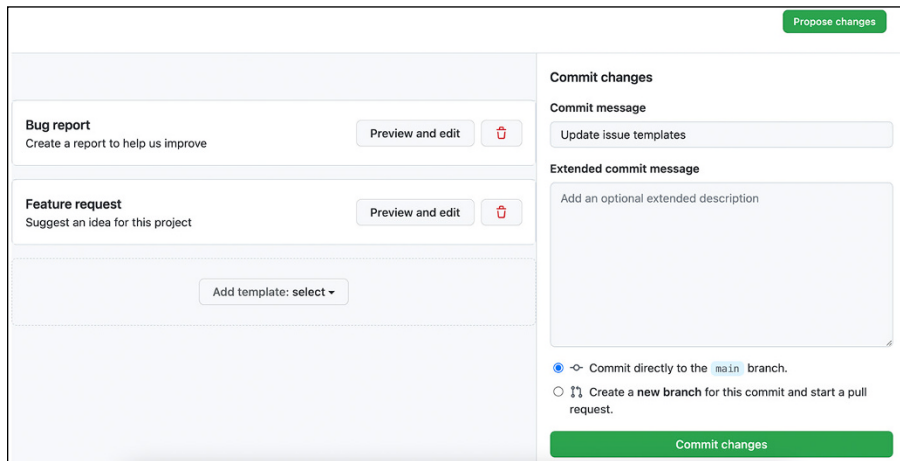
- 1. Select both the Bug Report template and then select the Feature Request template.**

You should see them listed on the page. Note they're not yet active. They still need to be committed to the repository. You can click the Preview and Edit button for each template if you want to change anything.

- 2. Click the Propose Changes button to enter a commit message for these templates, as shown in Figure 10-13.**

### 3. Click the **Commit Changes** button to commit these templates to the repository.

They are stored in a special `.github/ISSUE_TEMPLATE` folder. If you navigate to that folder, you should see two files, `bug_report.md` and `feature_request.md`. Take a look at the contents of those files to understand the structure of an issue template.

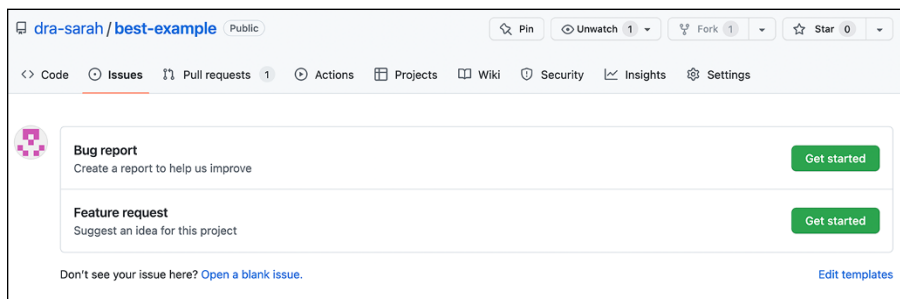


The screenshot shows the GitHub 'Propose changes' interface. On the left, there are two templates: 'Bug report' (Create a report to help us improve) and 'Feature request' (Suggest an idea for this project). Each has a 'Preview and edit' button and a trash icon. Below them is an 'Add template: select' dropdown. On the right, the 'Commit changes' section shows the commit message 'Update issue templates' and an 'Extended commit message' field. At the bottom, there are two radio buttons: 'Commit directly to the main branch.' (selected) and 'Create a new branch for this commit and start a pull request.' A green 'Commit changes' button is at the bottom right.

**FIGURE 10-13:**  
Choosing an issue  
template.

You can create new templates by going through the previous steps or by manually adding a file to the `.github/ISSUE_TEMPLATE` directory with the same basic structure as the existing issue template files.

Now, when someone creates an issue on your repository, they first see a set of issue templates, as shown in Figure 10-14.



The screenshot shows the GitHub 'Issues' page for a repository named 'dra-sarah / best-example'. The page has a header with repository name, 'Public' status, and buttons for Pin, Unwatch (1), Fork (1), and Star (0). Below the header is a navigation bar with links for Code, Issues (selected), Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. The main content area shows two templates: 'Bug report' (Create a report to help us improve) and 'Feature request' (Suggest an idea for this project). Each has a 'Get started' button. At the bottom, there is a link 'Don't see your issue here? Open a blank issue.' and an 'Edit templates' link.

**FIGURE 10-14:**  
Choose an issue  
template to  
create an issue.

Clicking the Get Started button on one of the templates then displays the issue creation form, but with the contents prepopulated with the template information. Figure 10-15 shows an issue prepopulated with the default Bug Report template.

**FIGURE 10-15:**  
A new issue with  
the Bug Report  
template filled in.

**Issue: Bug report**

Create a report to help us improve. If this doesn't look right, [choose a different type](#).

Title

Write Preview

**\*\*Describe the bug\*\***  
A clear and concise description of what the bug is.

**\*\*To Reproduce\*\***  
Steps to reproduce the behavior:

1. Go to '...'
2. Click on '...'
3. Scroll down to '...'
4. See error

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

Styling with Markdown is supported

Submit new issue

The contents of the issue walks the issue creator through all the information expected of them. Of course, the person creating the issue can always choose to ignore the template and put anything they want in the issue.

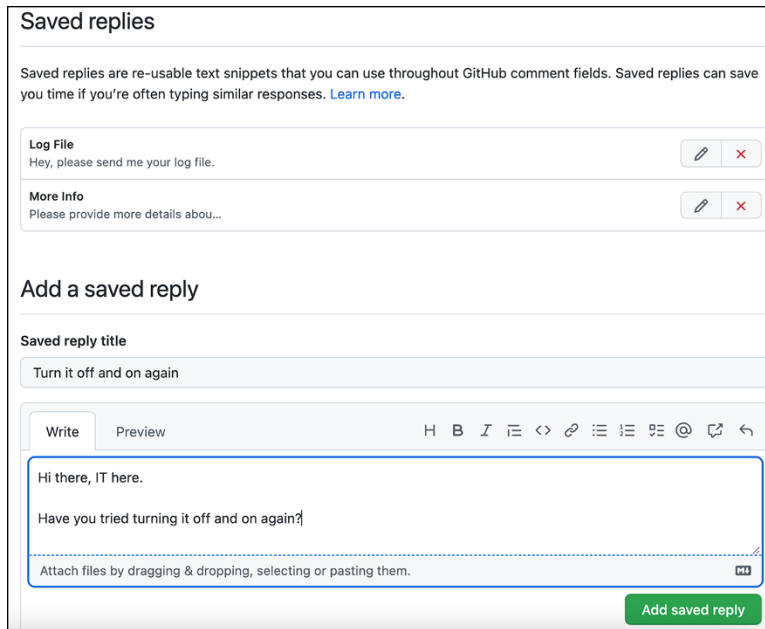
## Saved replies

Often, as your repository grows in popularity and the issues start to flow in, you'll find yourself repeating responses over and over again. For example, if someone reports a bug, you may need them to supply a log file. If every bug report should include a log file, then you should mention that in an issue template.

But even if you do, many folks will forget to include it. This situation is where a saved reply can come in handy. A *saved reply* is a canned response you can use over and over again.

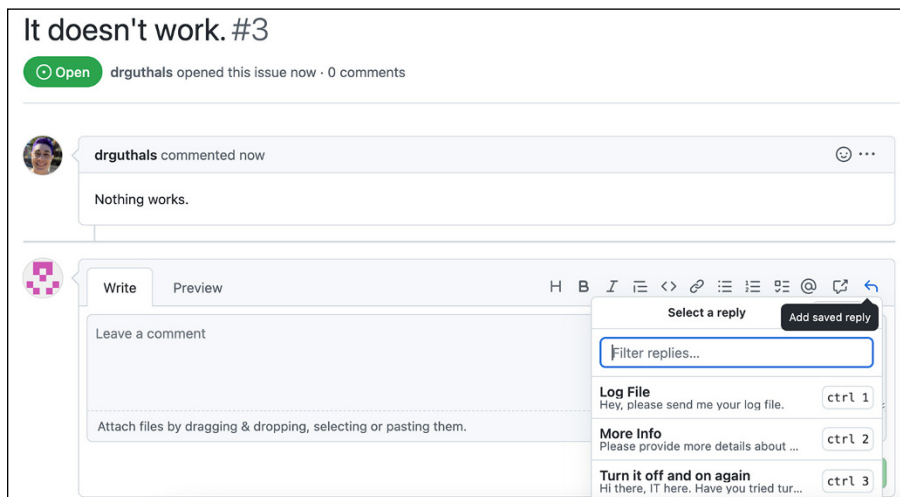
As of this writing, canned responses apply only to a user and are not specific to a repository. So you can create your own saved replies, but they aren't immediately available to other members of your repository.

Go to <https://github.com/settings/replies> to manage your list of saved replies. To create a new saved reply, fill in the form at the bottom, as shown in Figure 10-16.



**FIGURE 10-16:**  
Creating a new  
saved reply.

To access the saved replies when responding to an issue or a pull request comment, click the arrow in the top-right corner of the comment box, as shown in Figure 10-17. That brings up a list of your saved replies. Click the one you want to use to fill the comment box with that reply.



**FIGURE 10-17:**  
Selecting a saved  
reply in a  
comment box.



# Ending Your Project

At some point in the life of a repository, you may want to step away from it. This break may happen for many reasons. Perhaps the project is no longer useful, having been supplanted by other better projects. Perhaps the project is a runaway success, but you don't have the time to give it the attention it deserves.

Whatever the reason, it's important to handle the end of your involvement with the repository with the same care you showed in starting it.

## Archiving a project

Archiving a project is a good option when a project is no longer all that useful to others. Or even if it's still useful, but mostly complete, archiving could be a good option. *Archiving* a project indicates that the project is no longer actively maintained. The code is still available to the world, and people can fork and star your project, but nobody can create new issues, pull requests, or push code to an archived repository.

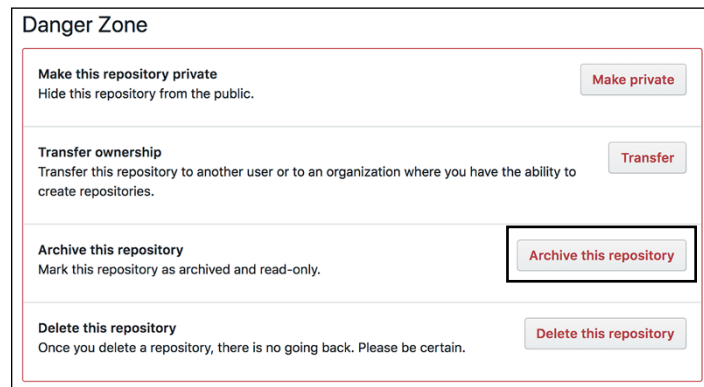
To archive a repository, go to the repository settings page:

1. **Scroll all the way down to the Danger Zone and click Archive This Repository, as shown in Figure 10-18.**

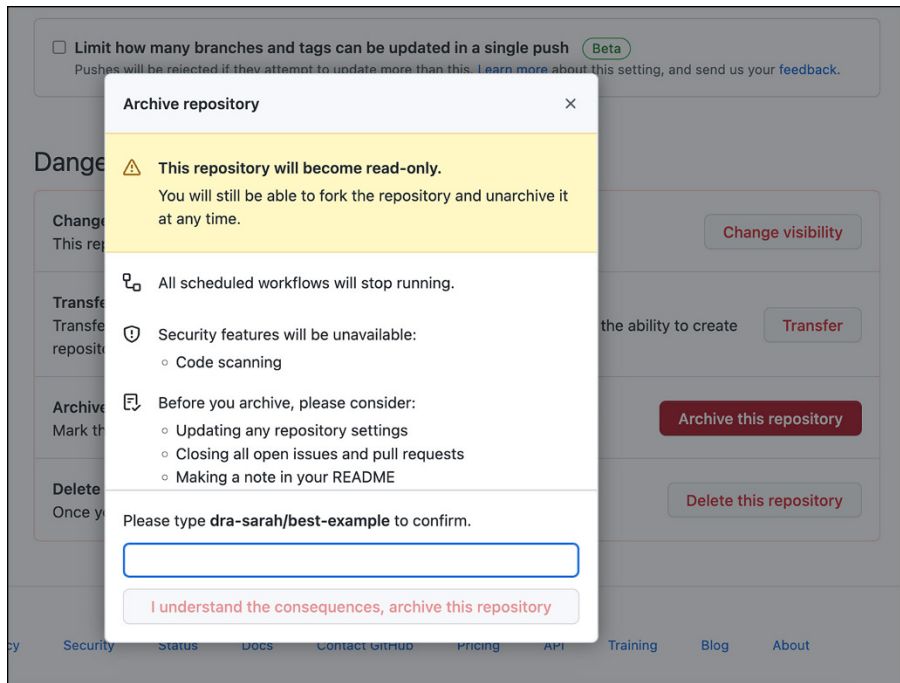
Clicking this button displays a detailed confirmation box that describes what will happen if you archive the repository, as shown in Figure 10-19.

2. **Type the name of the repository and click I Understand the Consequences, Archive This Repository to archive it.**

Your repository is archived.



**FIGURE 10-18:**  
The button to  
archive a  
repository.



**FIGURE 10-19:**  
Finish archiving a  
repository.

## Transferring ownership

If your project is popular with a robust set of maintainers, you may want to transfer ownership to another account rather than archive the project. Transferring ownerships lets the new owner have full rights to the repository and continue to maintain it.

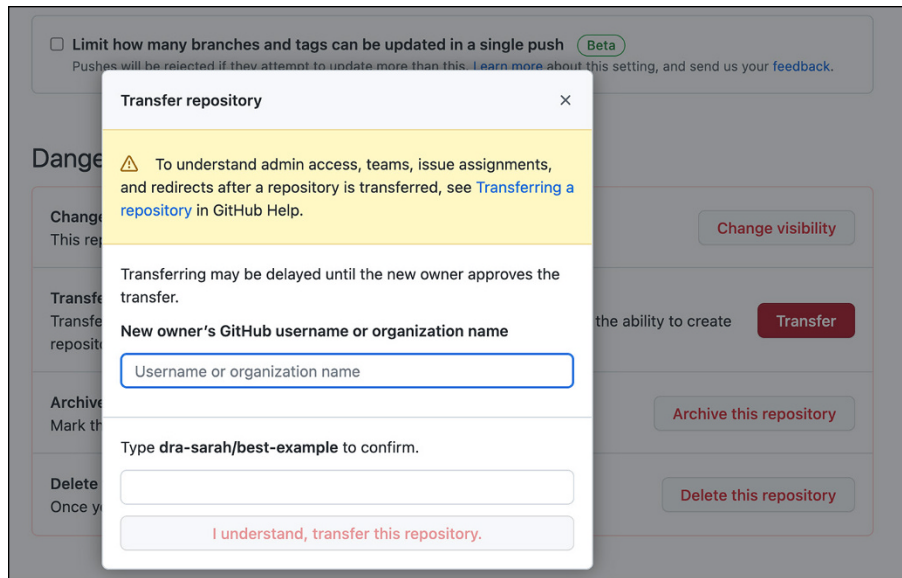
To transfer ownership:

1. **Go to your repository's Settings page and scroll down to the Danger Zone.**
2. **Click the Transfer button to bring up the transfer dialog box, shown in Figure 10-20.**
3. **Confirm the current repository's name and type the new owner's username or organization name.**

The URL to the repository changes, but GitHub redirects the old URL to the new URL when anyone tries to visit the repository.

4. **Type the name of the repository and click I Understand, Transfer This Repository.**

**FIGURE 10-20:**  
The transfer  
repository  
dialog box.



Once you transfer the repository, you see a message at the top of the repository that says “Repository transfer to `<username>` requested,” where `<username>` is the GitHub user or organization that you transferred the repository to.

The user you want to transfer the repository to then gets an email and can complete the transfer by clicking a link; if they do not accept the transfer in one day, the request expires. After clicking the link, they’re taken to GitHub where they find a message at the top of the GitHub home page that says “Moving repository to `<repository-name>`. This may take a few minutes.” Once the transfer is complete, the repository is removed from your account and added to the account of the person you transferred it to.

## Starting Your Own OSS

### Setting Up a Repository for OSS

**Creating an Open Source Repository:** When creating a repository with the intent to make it open source, it's common to make the repository public and select a license during the creation process. If you want to defer these choices, you can set up your repository privately first and then make it public later.

**Adding a License:** The only requirement for software to be considered open source is to have an open source license. GitHub makes it easy to add a license to an existing project by creating a LICENSE file and choosing a license template.

**Adding Contributor Guidelines:** Including a CONTRIBUTING.md file in your repository provides information on how to contribute to your project, saving time and headaches for both you and your contributors.

**Adding a Code of Conduct:** A code of conduct sets the tone for the type of open and welcoming environment you want to foster for your project. This can be added similarly to how you add a license.

### Managing OSS Workflows

**Making a Repository Public:** To make a repository public, visit the Settings page, scroll to the Danger Zone, and click the Make Public button. Confirm the change by typing the repository name.

**Enforcing a Code of Conduct:** It's important to enforce the code of conduct to maintain a healthy community. Respond with kindness to off-color comments and use temporary interaction limits if necessary.

**Blocking Users:** If someone is persistently abusive, you can block or report the user from their profile page.

### Ending an OSS Project

**Archiving a Project:** If a project is no longer useful or mostly complete, archiving it indicates that it is no longer actively maintained. This prevents new issues, pull requests, or code pushes.

**Transferring Ownership:** If your project is popular and has a robust set of maintainers, you may want to transfer ownership to another account rather than archive it. This allows the new owner to continue maintaining the project.

### Additional Tips

**Writing a README.md File:** The README.md file is crucial as it provides visitors with information about your project, how to get involved, and where to find more information.

**Writing Good Documentation:** Good documentation distinguishes your project and provides a way for others to dip their toes into open source. GitHub supports serving documentation from the docs folder in the main branch of your repository.

**Managing Issues:** As your project gains attention, managing issues becomes important. Labeling and triaging issues, setting up issue templates, and using saved replies can help manage the influx of new issues.