

Course Name: Project Management with Git

UNIT - 2

Term: October 2024 – January 2025

Faculty:
Brunda G
Assistant Professor

UNIT 2

Unit II (3 Lectures)

Setting Up a GitHub Website Repo- Turning a Project Repo into a Website, Setting Up a Personal Website Repo, Creating Issues for Your Website, Setting Up Your Local Environment.

Creating a Website with GitHub Pages - Jumping into an Existing GitHub Project, Preparing Your Contribution, Building Your Personal Website

Contributing to your first project: Forking GitHub Repositories- Introducing Forking, Cloning, Forking, and Duplicating, cloning a Repository, Forking a Repository. Writing and Committing Code- Creating a Repository, Writing Code, creating a Commit, writing a Good Commit Message, Committing Code with GitHub Desktop, Using GitHub Conventions in Commit Messages.

Setting Up a GitHub Website Repo-

What are GitHub Pages?

GitHub Pages is a feature provided by GitHub that allows you to create and host websites directly from your GitHub repository. It is commonly used for personal, project, or organizational websites and is particularly popular among developers due to its simplicity and integration with GitHub.

Turning a Project Repo into a Website

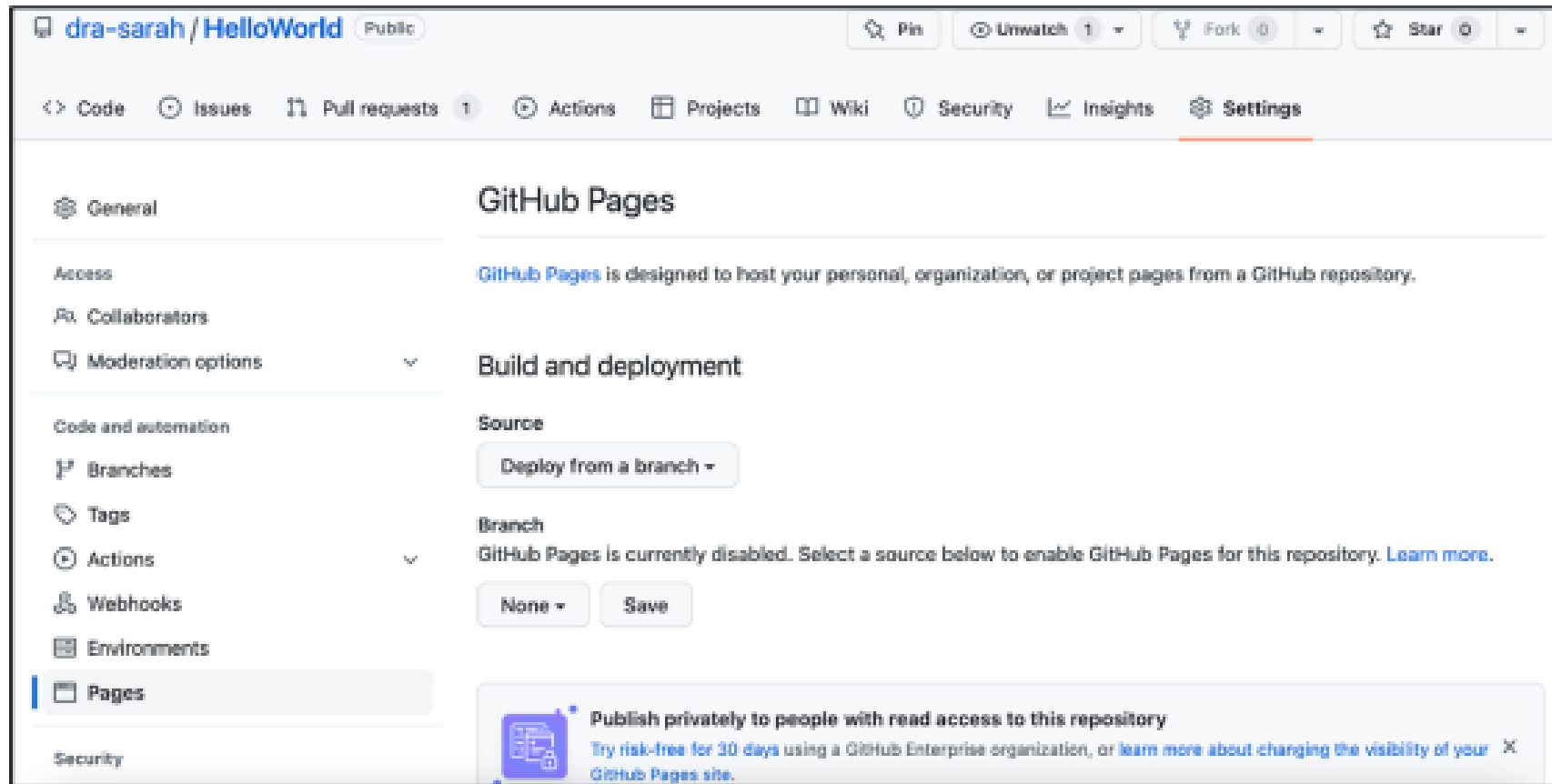
GitHub Pages is a great tool that is integrated into GitHub.com. GitHub Pages looks for a README.md file on your main branch and use it as the landing page for your website, meaning you don't have to do much to get it up and running! Just follow these steps:

1. Open a repository on GitHub.com.

For this example, I created a simple HelloWorld repo that has a basic README and a very simple solution for a sum function written in Python. You can see this repository at <https://github.com/dra-sarah/HelloWorld>.

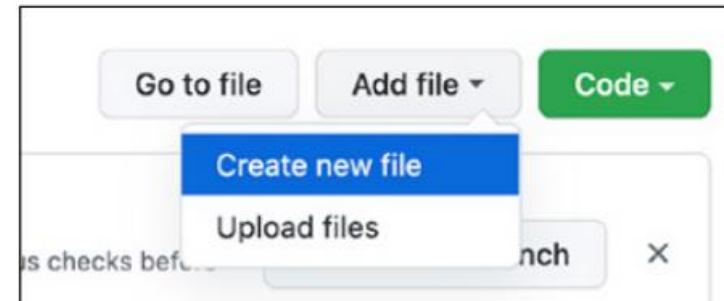
2. On the home page for the repo, click the Settings tab on the top right to open the Settings page.
3. Scroll down on the left menu and click the Pages option, shown in Figure 4-1

Turning a Project Repo into a Website



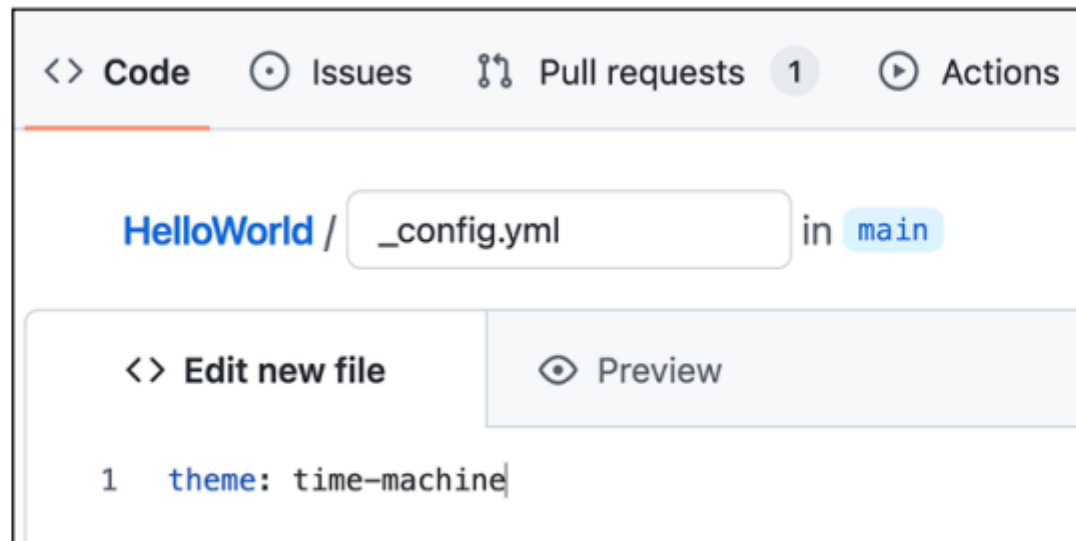
4. From the Source drop-down menu, change the source for GitHub Pages to Deploy from a Branch. From the Branch drop-down menu, change the branch to deploy from None to main and then click Save.
5. Choose a theme for your website. Under the specified deployment branch you should see a link to choose a theme, shown in Figure 4-1. The Jekyll theme docs page opens, where you can browse supported themes at <https://pages.github.com/themes/>.
6. After you choose a theme for your website, go back to the Code tab on your repository to add the config file. Click the Add File drop-down menu and select Create New File, as shown in Figure 4-2. Name the file `_config.yml`.

FIGURE 4-2:
The GitHub add a
new page option
in the repository.



- 7. Add the theme you chose to the config file, shown in Figure 4-3. Then commit the changes to the main branch by clicking the Commit Changes button at the bottom of the editor. The config file should really only have one line of code; in this example, I have chosen the theme time-machine: remote_theme: pages-themes/modernist@v0.2.0 plugins: - jekyll-remote-theme

FIGURE 4-3:
The single line
of code you have
to add to a
single file to
update the title.



8. On the Code tab, click the the Environments option on the left menu, as shown in Figure 4-4.
9. Click the active GitHub Pages environment, and then click the View Deployment button on the right-hand side. Now you see a web page that says Hello World, just like is written in the README file for your repository.

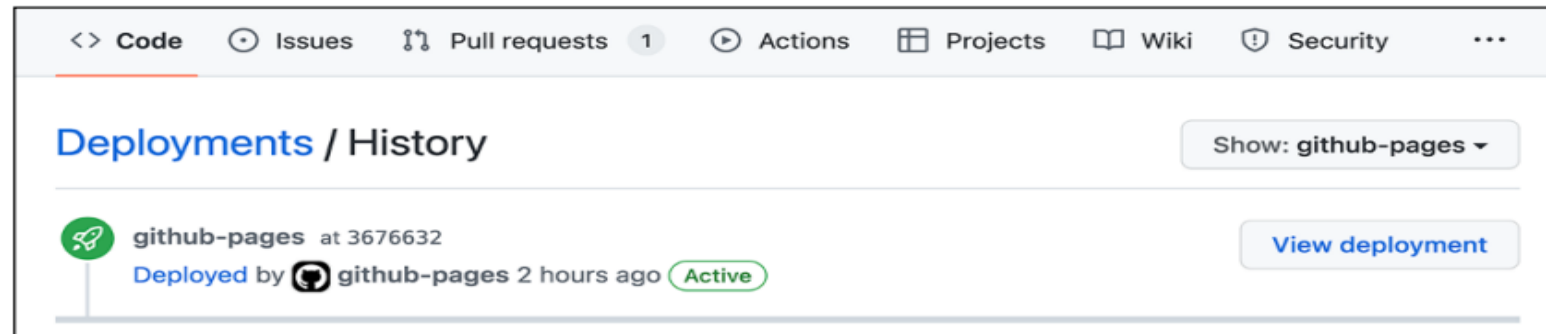


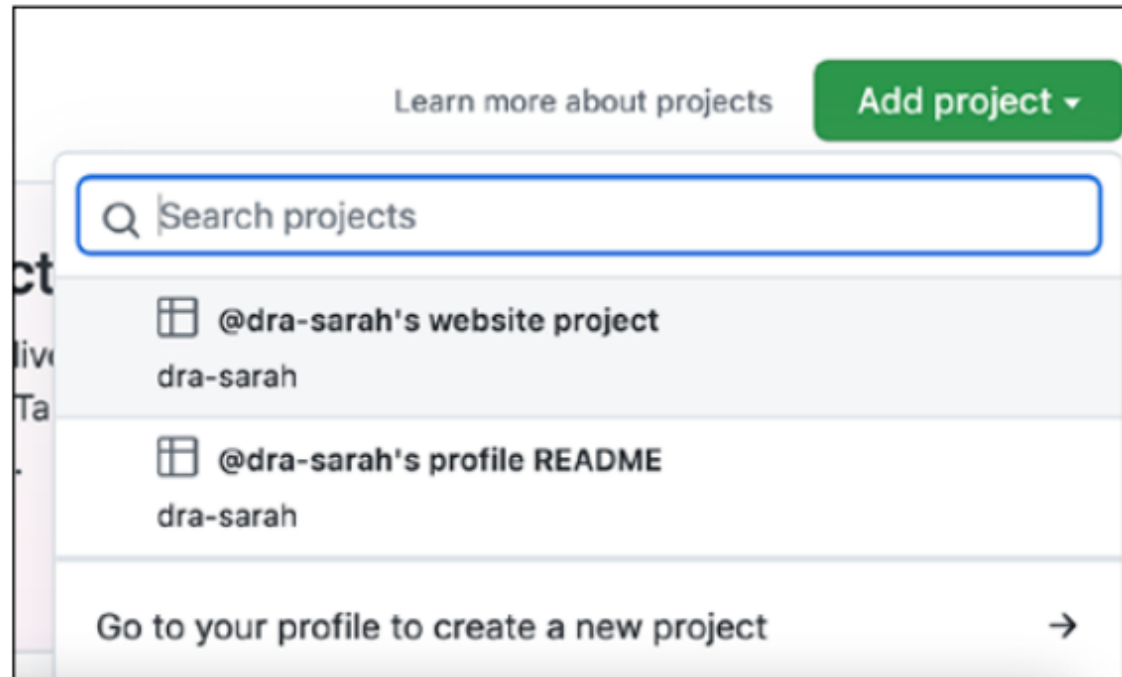
FIGURE 4-4:
The GitHub Pages
website theme
chooser.

Setting Up a Personal Website Repo

To create a new repo that houses your own personal website, you need to set up your repo:

1. Create a new repository and name it `username.github.io`, where `username` is replaced with your actual GitHub username. For example, the name of my repository is `dra-sarah.github.io`. If you're unsure how to create a new repository
2. Make the repository public, initialize it with a README, choose a license, if you want, and then click Create Repository.
3. Create a new project that is the board style.
4. Once you have created the project, head back to the project page of your repository, click the Add Project drop-down menu, and choose the project you just created,
5. At the top of your repo again, click the Settings tab and scroll down until you see the GitHub Pages section.

FIGURE 4-5:
Adding a project
to a GitHub
repository for
quick access via
the Projects tab.



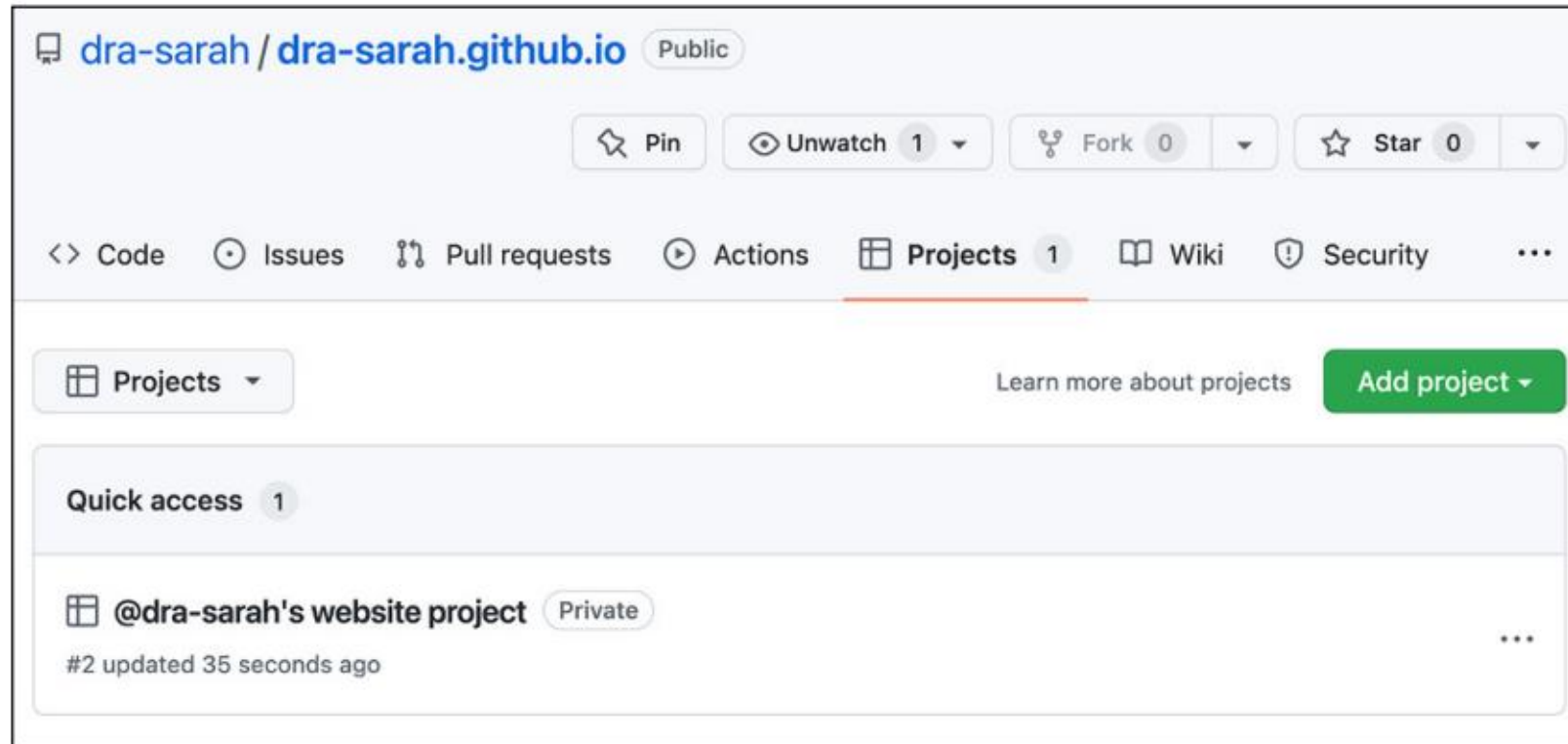


FIGURE 4-6:
A GitHub pages
project linked to a
specific
repository.

Notice that it says your site is ready to be published at a certain URL (mine is <https://dra-sarah.github.io/>). If you go to this URL, you see a simple barebones web page with the contents of your README.md file. If you get a 404, just wait a moment and refresh the page. It can take a few seconds for GitHub Pages to build your site.

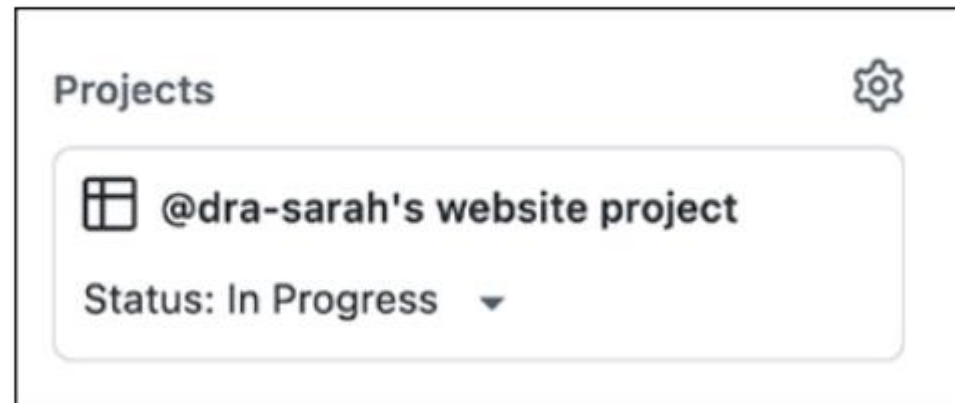
6. In the GitHub Pages section, click Choose a Theme and then click Select Theme. If you don't know how to do this, review the previous section in this chapter, "Turning a Project Repo into a Website."

7. Edit and commit your README.md file. Back on the Code tab of your repository, click the pencil icon in the top-right of your README.md file and add some information about you. Commit those changes to a new branch and open a pull request. The page refreshes with a new pull request ready to be created. You can merge your pull request. Then you can navigate back to your code, and you see the `_config.yml` file and the changes to your page.

8. Create a pull request with a more descriptive title. Change the title of the pull request to describe everything you want to do for this iteration of your website and add a description. For example, I added the following list: To create a basic website: - [] Add a short personal bio - [] Add links to my social media channels

9. Link this pull request to the project and update the status of this item. To ensure your project board automatically tracks the progress of your website, choose the project created in Step 3 on the right side of the pull request from the Projects section. Then update the status of this item to “In Progress,” shown in Figure 4-7

FIGURE 4-7:
A GitHub pull request linked to a specific project with a status of “In Progress.”



10. Update the pull request when appropriate. The description of a pull request is not static. When you first create a pull request, you may have a list of things you want to do before merging the code into the main branch. You may also end up making changes throughout. Make sure to always revisit the description and make sure it is accurate. For example, if you're following these steps exactly, you have already added some information about yourself and maybe even links to your other social media channels, so you can likely check off those two boxes.

11. Verify the project board automation. Go back to the project board by clicking the Projects tab at the top of your repo and notice that a new card is in the In progress column. If you click into the card, you're redirected to the pull request.

12. Switch to the pull request branch. Go back to the Code tab of your repo and switch to the branch that is associated with your pull request. Mine is readme-init.

13. Create an index.md file. On the top right of your code file list, click Create New File. Name the file index.md and add a header that says "Hello World!": # Hello World! Commit that file to the same branch you've been working on, with a title and commit message.

14. Update and merge the pull request. Go back to the pull request and add a new item to your checklist “create an index.md landing page,” and then check it off. Because everything that you wanted to do for this pull request has been completed, the pull request can be merged. Click Merge Pull Request, Confirm Merge, and then Delete Branch.

15. Verify the project board automation. The pull request shows in the conversation that it has been moved from the In Progress column to the Done column. You can also go back to the project board. Notice that the card has moved to the Done column, and both checklist items have been completed.

16. Verify the website was published. Go back to your URL (mine is <https://dra-sarah.github.io/>). You have a working website with the theme you chose.

Creating Issues for Your Website

Creating issues for everything you want to add or change about your website can help you plan and remember all the little things you want to change. Say that someone gives you a great suggestion.

You don't want to pull out your computer and make the change right then and there, but you can quickly jump on [GitHub.com](https://github.com) and create an issue to remind you to add it later.

Creating an issue can also be useful if you are working on your website, and you've found something that is bothering you that you want to change. Instead of derailing what you're already working on with a new task, you can just make a quick issue and get to it later. To get started with this planning phase, go to the Issues tab on your repo and click New Issue. Create an issue for all the things you want to add to your website.

» **Change the title and tagline.** The title and tagline of your website is currently something auto-generated. You probably want to change it to your name and some tagline that represents you. Create an issue, assigning yourself to it and linking it to your project board:

Issue Title: Change the title and tagline

Issue Description: Make the title and tagline something unique to me.

» **Add sections to the website.** Without even having to leave an issue you created, you can click New Issue and create another issue to add in sections to your website, assign yourself to the issue, and link it to your project board. In my example, I've chosen to add three sections:

Issue Title: Add a couple of section]s to the website

Issue Description: Add three sections to the website:

- [] About Me
- [] Contact Information
- [] Current Focus

Two issues now appear under the Issues tab of my website repo, as shown in Figure 4-8. They also appear on my project board in the Todo column (see Figure 4-9).

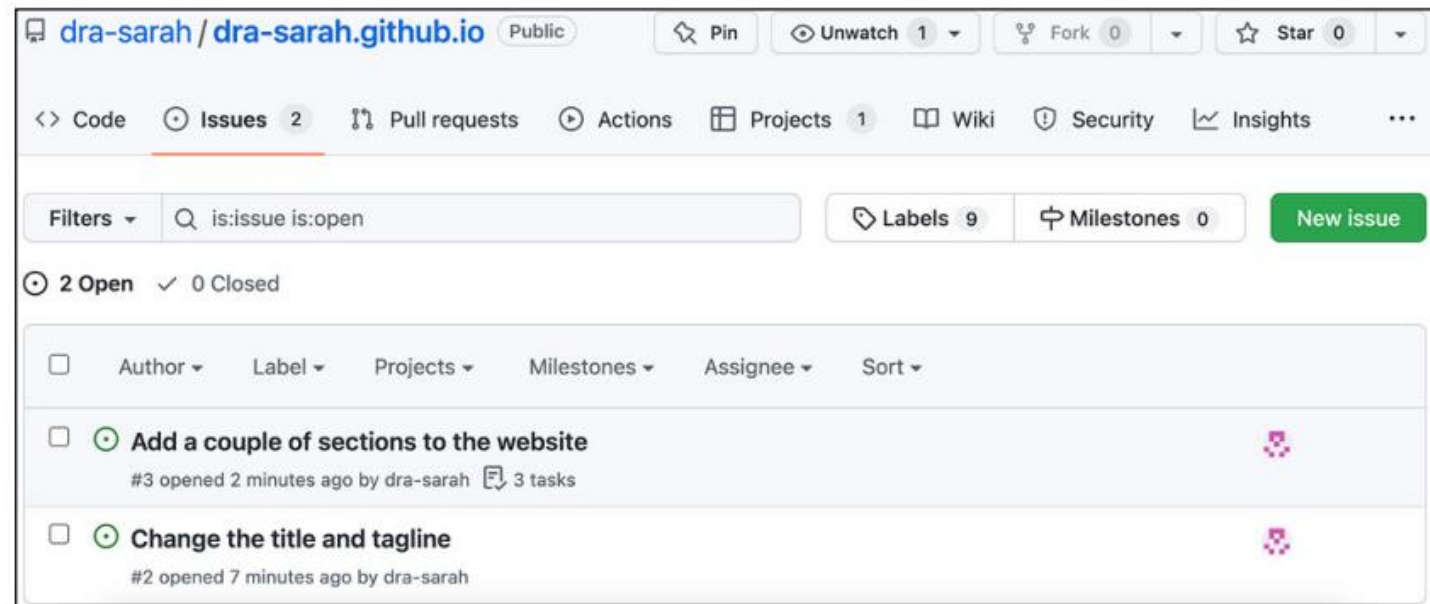


FIGURE 4-8:
The issue list for
the website
repository.

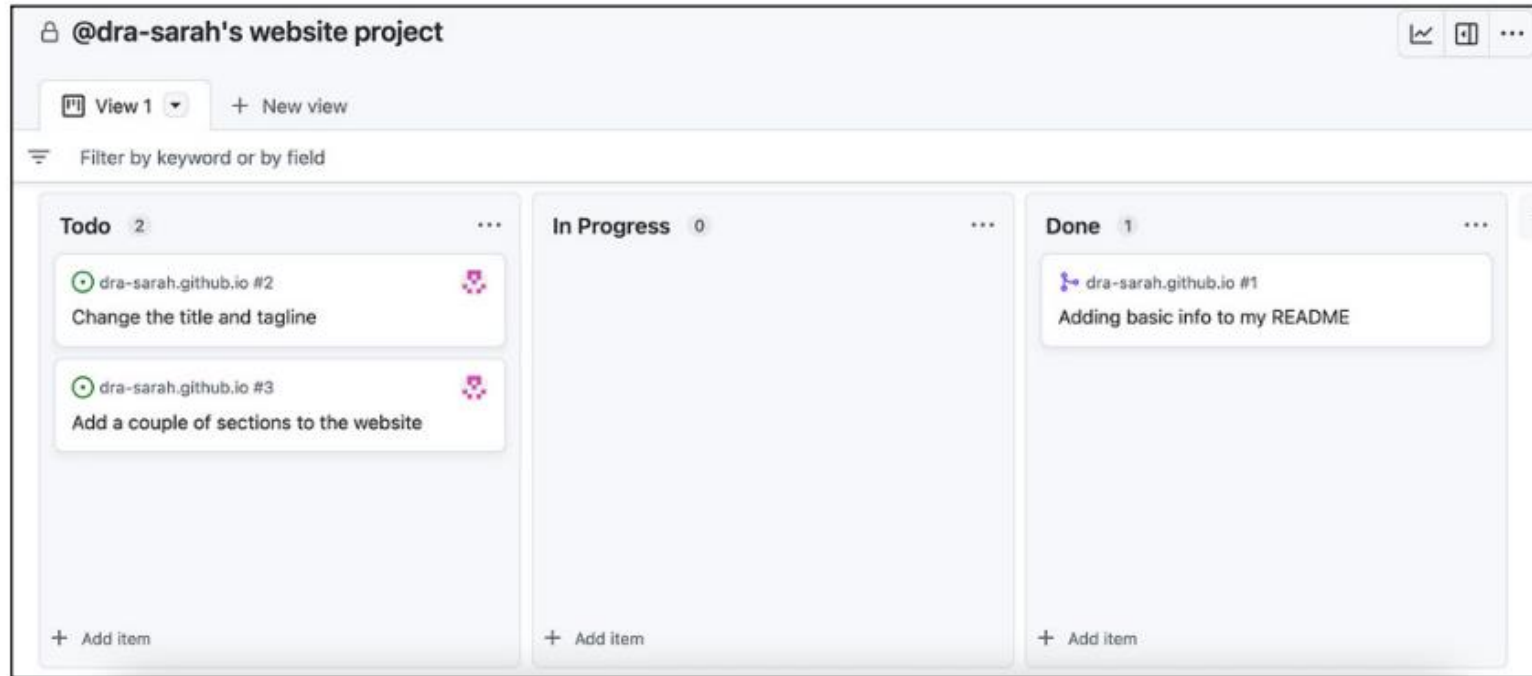


FIGURE 4-9:
The project board
for the website
repository.

Setting Up Your Local Environment

This section assumes you already set up GitHub Desktop and Visual Studio Code.

Cloning a repo in GitHub Desktop

The first step in modifying files on your local computer is to get your website repo onto your computer: 1. Open GitHub Desktop and choose File ⇨ Clone a Repository on the menu bar. You see a dialog box with three tabs: GitHub.com, Enterprise, and URL. A nice alternative approach to cloning a repository when you have GitHub Desktop is to click the Clone or Download button on the home page of every repository. When you click the button, you see a flyout menu that includes an Open in Desktop button. Click that button to launch GitHub Desktop (if it's not already running) and clone the repository to your local machine.

2. On the GitHub.com tab of the Clone dialog box, your repositories autofill for you from GitHub.com.

If your GitHub.com repositories don't autofill in the Clone dialog box, it probably means you're no longer signed in to GitHub.com in GitHub Desktop. You can log in by choosing GitHub Desktop ⇨ Preferences from the top menu bar. Click the Accounts tab and sign in. If it appears you're logged in but your repos are still not showing up, try signing out and signing back in.

3. Choose your personal website repository and choose where you want it to be stored on your local machine. I chose the default path as the place to store the repository on my local machine.

4. Click Clone. GitHub Desktop refreshes with your repo information included.

Touring GitHub Desktop

GitHub Desktop offers a variety of features to help you with your development and interactions with GitHub.com. You can check out the GitHub Desktop User Guides at <https://help.github.com/desktop> if you need additional support beyond this book. Figure 4-10 highlights the top six features:

- » **Repository list:** As you clone more repositories to your local computer, clicking the Current Repository drop-down menu reveals all the repositories that you have on your local computer, enables you to quickly switch between them, and gives you a button to quickly add a new one.
- » **Branch list:** The branch list gives you a quick overview of all the branches that you have checked out on your local computer, as well as a button to quickly create a new branch.
- » **Pull request list:** One the same drop-down list as the branch list, you see a second tab that lists all the pull requests that are open on this repo.
- » **Sync Project button:** As you start to make changes and/or changes are made on the repo outside of what you're doing on your local machine, you need to sync. Because Desktop hasn't detected any changes made on GitHub.com or your local computer, the option presents itself as a fetch to start. If you start to make changes on your local machine, you can choose to push your local changes to GitHub.com. If you start to make changes on GitHub.com, you can choose to pull those changes to your local machine. If you create a repository on your local machine and it isn't on GitHub.com yet, you can choose to publish your project to GitHub.com. If you do not push your changes to GitHub.com, they won't be available for other people and if your computer were to crash, you would lose all your work. I highly recommend that you push your code often.

» **Changes list:** As you start to make changes to your code, the files that you've added, deleted, or modified show up in this changes list. You can click each file to see the diff to the right. When you're ready to commit to those changes, you can add a Summary and Description and click the Commit to Main button. At that point, you can push your changes to the branch that you're on clicking the Sync Project button. You should always double check which branch you're on before you commit and push your changes. You can undo commits and pushes, but avoiding it is best because the process can get hairy really quickly.

» **History list:** Next to the changes you find the history of this repo. The history includes activity from your local machine that has been synced with GitHub. com and activity from GitHub.com that you may have never done on your local machine. When you click one of the events, you see a list of activity that happened.

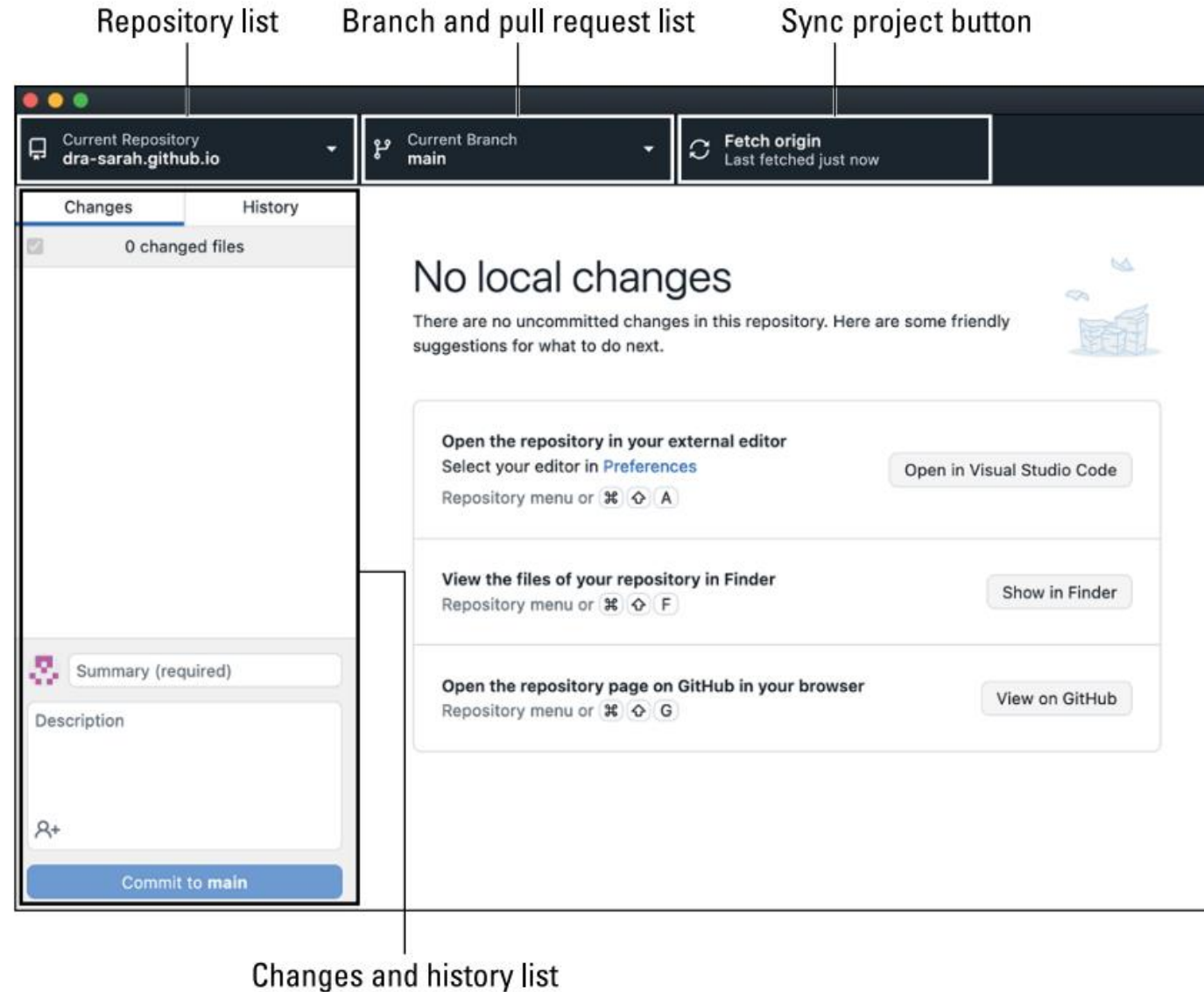


FIGURE 4-10:
An overview of
GitHub Desktop.

Opening your repo in Visual Studio Code

To edit your files on your local computer, you can use a number of applications, including Visual Studio Code or even TextEdit. In this book, I use Visual Studio Code (VS Code).

To open your repo in VS Code:

1. Open VS Code.

You see a blank window.

2. Choose File ⇨ Open from the top menu bar.

A file finder dialog box appears.

3. From the file chooser, open the folder for your repo and click Open.

Your project is now open in VS Code.

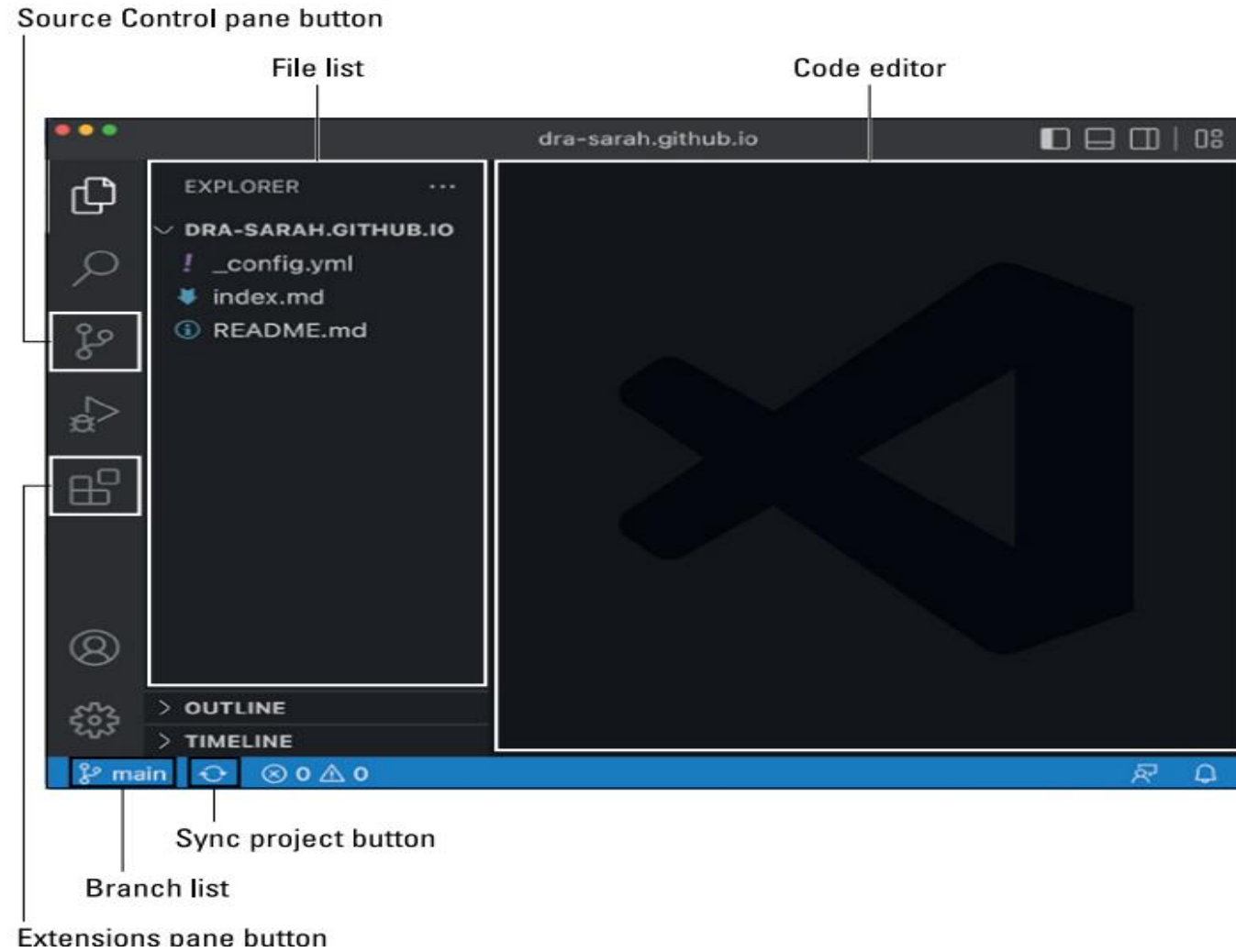


FIGURE 4-11:
An overview of
Visual Studio
Code

Creating a Website with GitHub Pages

Jumping into an Existing GitHub Project

Whether you're revisiting a project that you started yesterday, one you worked on last year, or finding a new project that you've never worked on, there are quick and easy ways to get oriented with a GitHub project.

To get started, make sure that you've opened your browser to [GitHub.com](https://github.com) and have signed in.

Accessing the GitHub.com repo

On the left side of the GitHub.com home page is the list of repositories that you have recently opened, contributed to, or created. Directly above the list is a search bar for searching repositories. This search bar becomes more useful as you interact with more GitHub repositories because the list of repositories can grow large, especially if you belong to a big organization.

At the top of the home page is another search bar that you can use to search for repositories, project boards, and teams (a feature of organizations beyond the scope of this book). By default, this top search bar is scoped to your current context. If you're on GitHub's home page, it searches all of GitHub. If you navigate to a repository on GitHub.com, the top search bar searches within that repository.

The search bar always gives you the option to search all of GitHub.com no matter where you are. Figure 5-1 shows the three ways to find a specific repository you may be looking for and two places to find new repositories

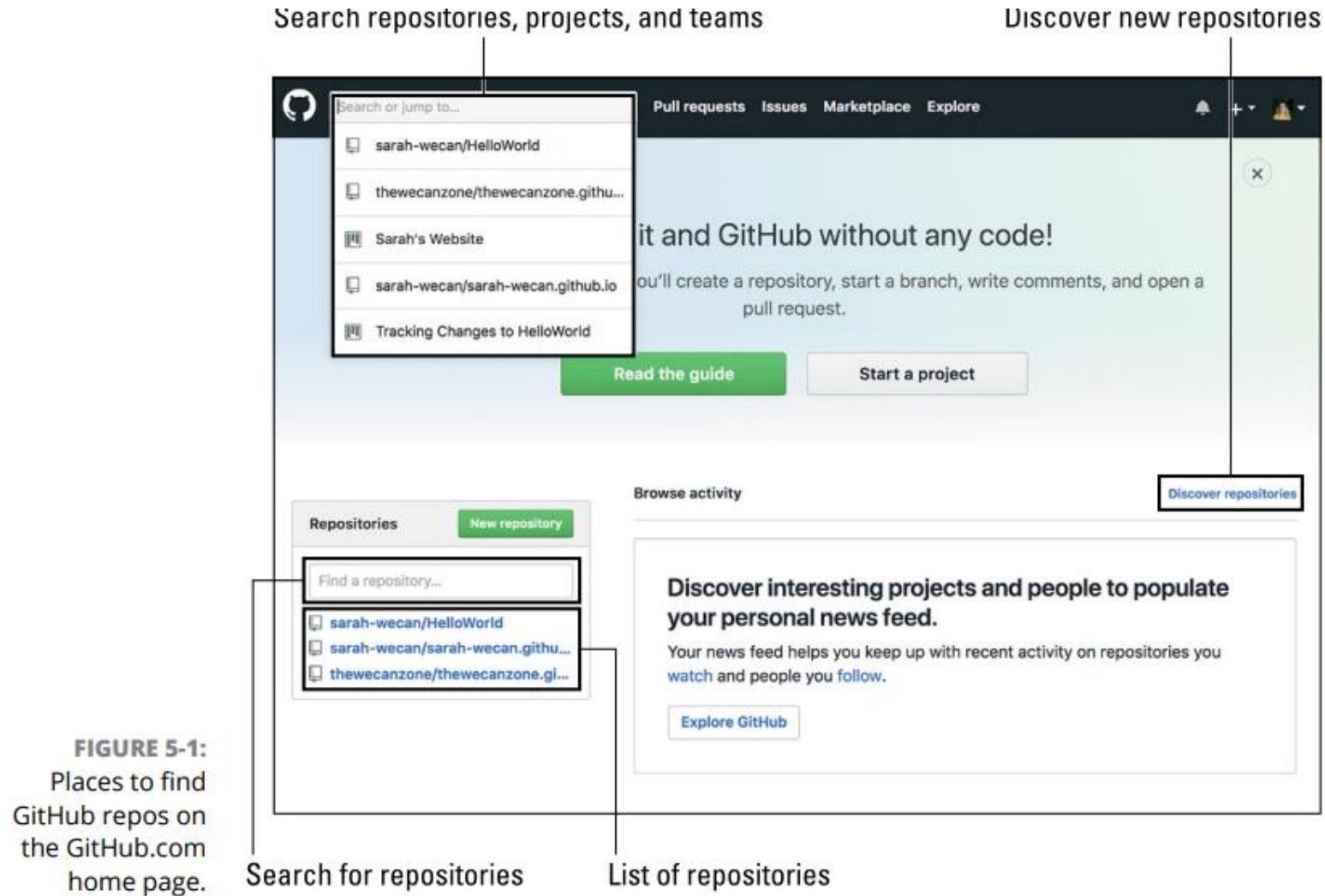


FIGURE 5-1:
Places to find
GitHub repos on
the GitHub.com
home page.

After you find the repository you want to start collaborating with, click it. The repository's home page appears. This chapter gives specific examples about contributing to the GitHub Pages website repo that you're the owner of, so if you want to follow specifically for that, choose the repo titled your-username.github.io. The repo I use in this example is dra-sarah.github.io.

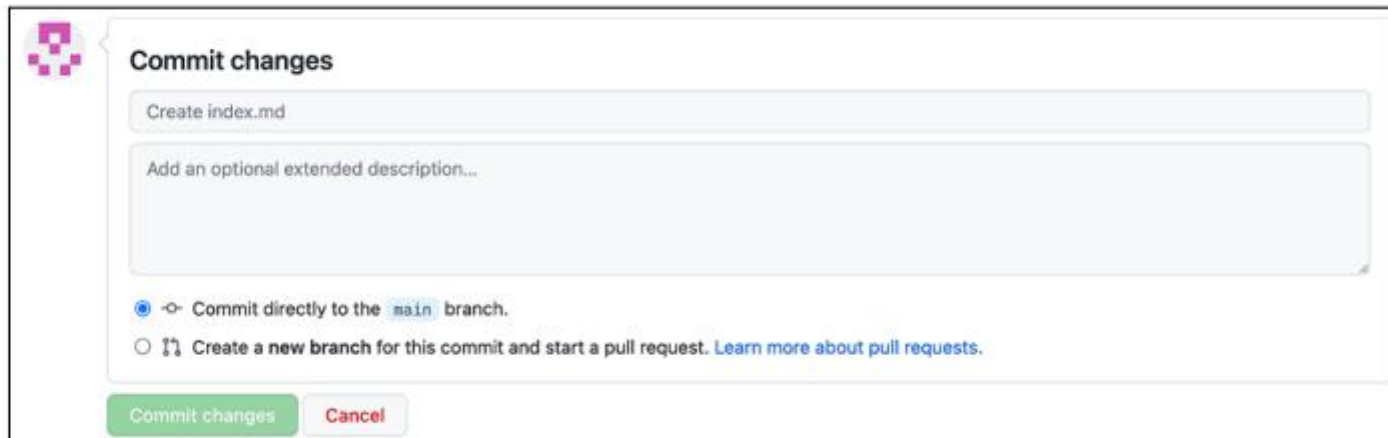
Verifying your permissions for the repo

If you're the owner or admin of a repository, you see a Settings tab at the top of the repository's home page. You have complete control over the repository, including the ability to

- » Invite collaborators.
- » Change the visibility of a repository from public to private or from private to public.
- » Delete the repository.
- » Archive the repository.

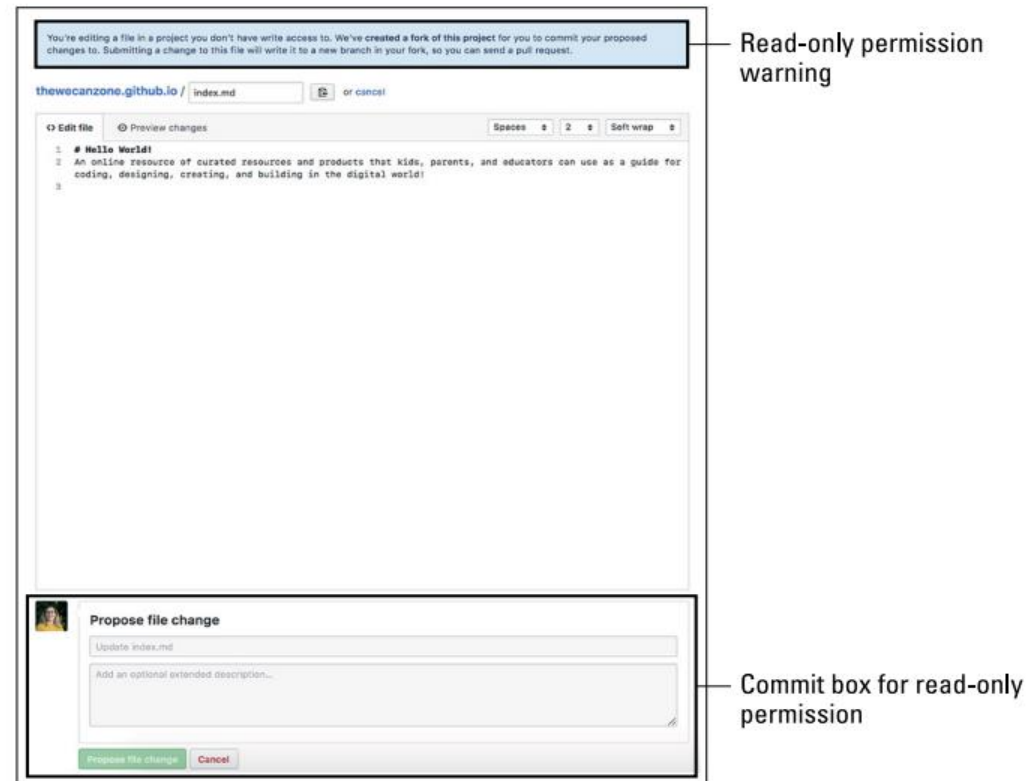
If you don't see the Settings tab, you're not the owner/admin, but you may have write permissions. To determine whether you do, you can attempt to make a change to a file by navigating to the file and clicking the pencil icon. If you're able to make a change and are presented with a Commit Changes box similar to the one shown in Figure 5-2, you have write permissions and were added as a collaborator for the project.

FIGURE 5-2:
The Commit
Changes box
when you have
write permissions
on a repo.



If you attempt to make a change but see the warning and commit box shown in Figure 5-3, you do not have write permissions and are not a collaborator. You can still create issues and propose file changes to the repo, but you have to get approval from someone with more permissions than you.

FIGURE 5-3:
Warning and
commit box
when you have
read-only
permissions
on a repo.



Preparing Your Contribution

After you orient yourself with your project, you need to decide what you're going to work on for your contribution. For the example in this book, I chose issue #2 in the Todo column of the project board: Change the title and tagline. Other good candidates are any issues that were opened with the label bug, help wanted, or good first issue because these issues are typically urgent for owners or good entry points for contributors. This section assumes you already have a repo cloned onto your machine. The examples in this section use GitHub Desktop and VS Code to resolve the issues.

Creating a branch for your contribution

Before you modify or add code to any project, whether a private solo project you own or a large open source project, I highly recommend creating a specific branch to contain all the code you write. Branching is a Git function that essentially copies code (each branch is a copy of the code), allowing you to make changes on a specific copy, and then merging your changes back into the main branch.

When you create a branch, Git doesn't actually copy all the code, which would be time consuming and inefficient. Git does something way smarter, but the specifics of what it does aren't important to day-to-day usage of Git, which is why I say Git creates a copy of the code. It's not technically correct, but it's conceptually correct. It's a useful mental model for branches.

Creating a branch for your changes gives you a safe space to try out solutions and make mistakes without threatening your project's integrity. If your solution is taking you down a wrong path, you can simply delete the branch, create a new one, and start over.

Creating a branch helps your main branch remain in an always working state because you only merge your code into the main branch when you're confident it doesn't introduce any new problems and/or accurately solves the problem you were targeting

To create the branch for your contribution, follow these steps:

1. Verify the repo you cloned.

Open the GitHub Desktop application and verify that the top left drop-down menu has your website repo selected. Figure 5-5 shows the repo correctly selected.

2. Start a new branch.

Click the branch drop-down list in the top center of GitHub Desktop and click the New Branch button (see Figure 5-5). When you click the New Branch button, a dialog box appears.

3. Give your branch a name.

Type a name for your branch. I use new-title-and-tagline because that is the change I plan to make.

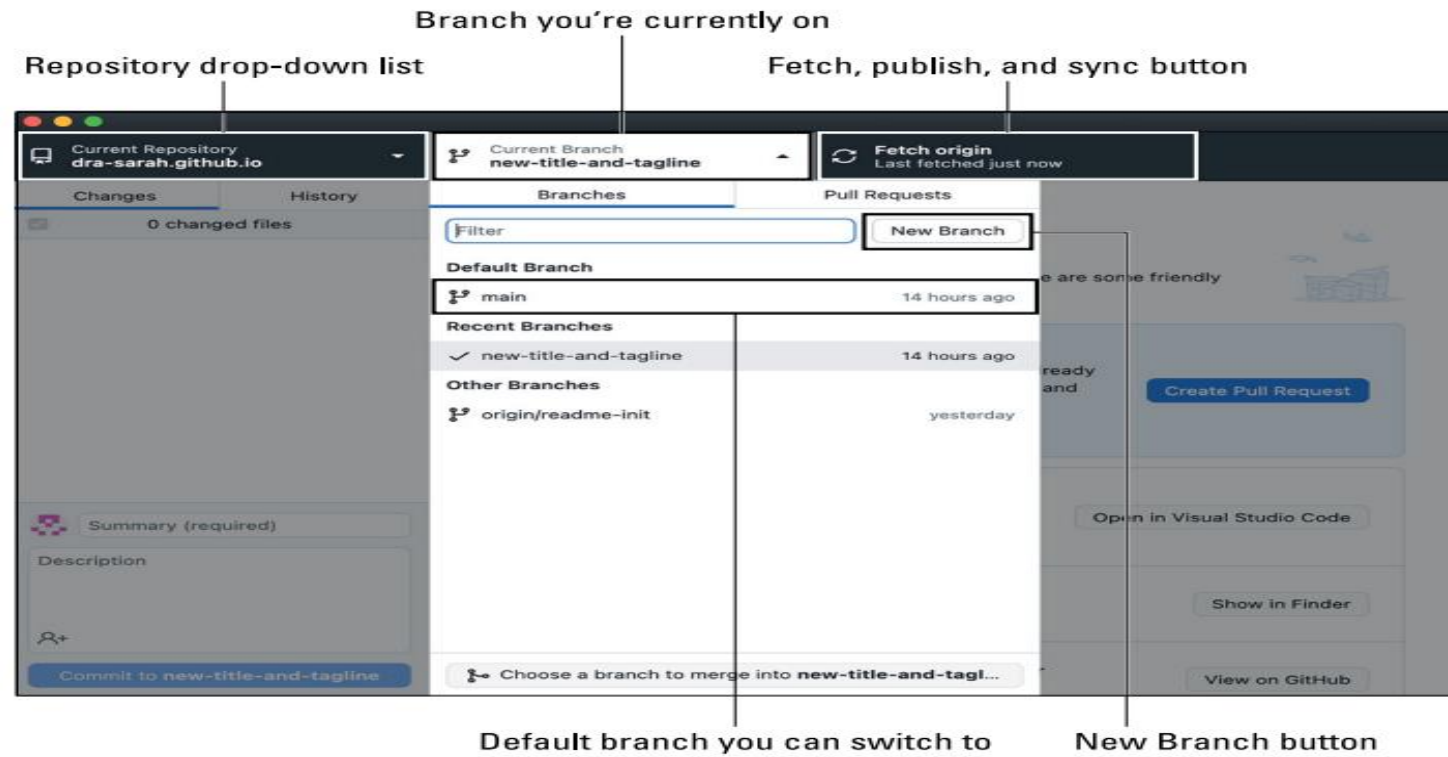


FIGURE 5-5:
The button to
create a new
branch in GitHub
Desktop for a
specific repo.

-
4. Click Create Branch. The dialog box closes, and you switch to the new branch. Figure 5-6 shows the current branch as new-title-and-tagline, but the main branch is still available within the drop-down menu. The button at the top right of GitHub Desktop also changes from Fetch Origin to Publish Branch.
 5. Publish your branch. Before making any changes to your branch, I recommend publishing your branch to GitHub.com. That way, as you start to modify or add code and push it to your branch, you won't lose any of your work. Click the Publish Branch button on the top right of GitHub Desktop. When the branch has been successfully published, this button goes back to the Fetch Origin button.
 6. Open your project in VS Code. Open the VS Code application and verify that your project folder is open. A tree-view of your project appears in the lefthand Explorer pane. Notice that the branch selector at the bottom of the VS Code application window now shows the branch name. Figure 5-6 shows the correct branch checked out in VS Code

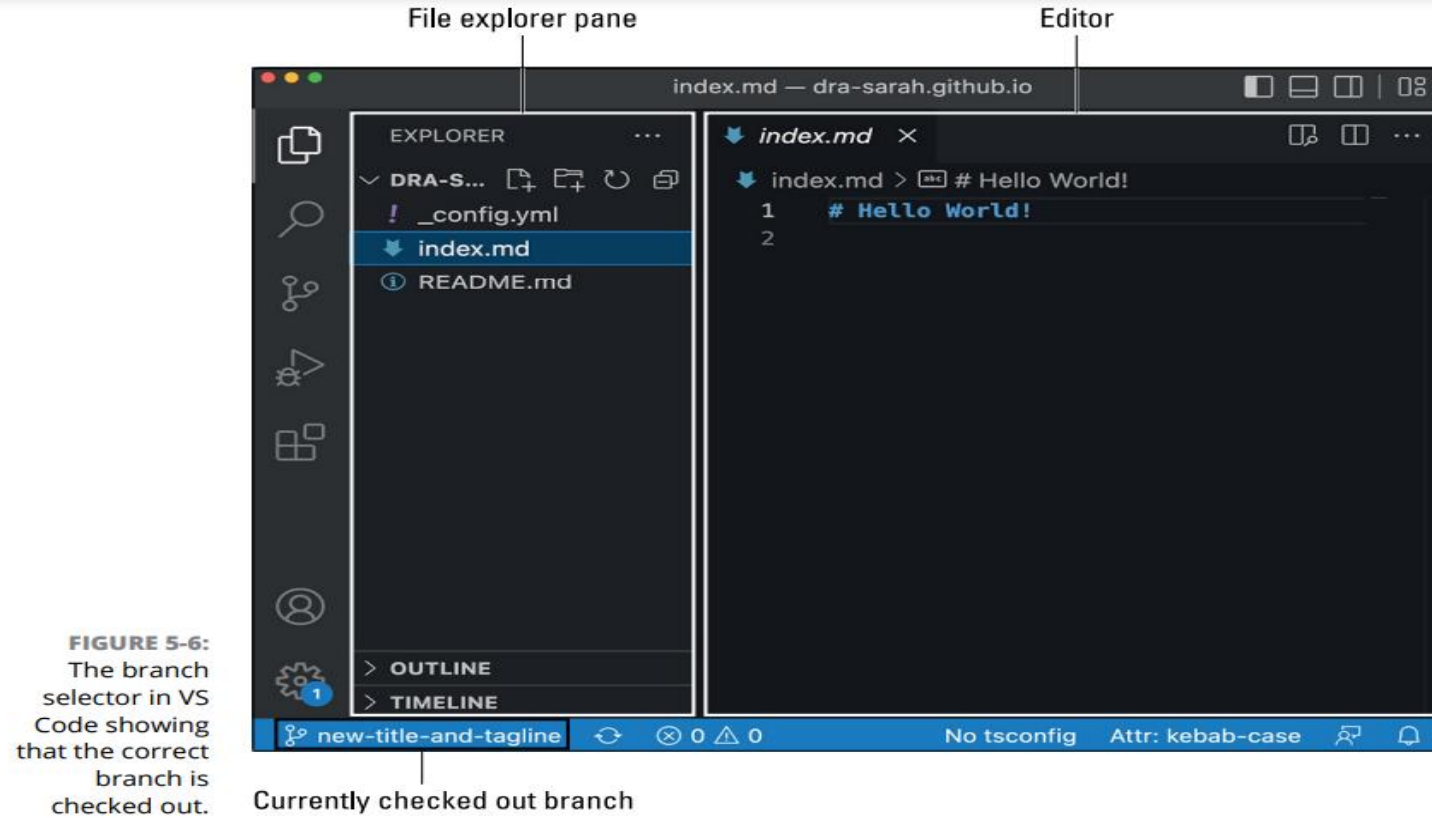


FIGURE 5-6:
The branch selector in VS Code showing that the correct branch is checked out.

Confirming your branch is published

Before writing code that you want to keep, confirm that you correctly published your branch and are able to push to the branch. If you're working on the same project, on the same computer, then you most likely are still properly set up. But if you're contributing to a project for the first time or you have just set up a new computer, you should consider confirming you're able to contribute

To get started, follow these six steps: 1. Click a file in the file tree in VS Code. The file opens in the editor. Figure 5-6 shows index.md open. 2. Modify the file. Don't make a lot of changes at this stage because the goal is to confirm you're able to push changes to the branch that you published. 3. Save the file. Choose File ⇨ Save to save your changes. When you click Save, the color of the file in the file tree changes, and the Source Control button changes to indicate that a change occurred (see Figure 5-7).

Source control button showing changes ready to be staged/committed

A list of modified, added, or removed files

The editor showing a modified file

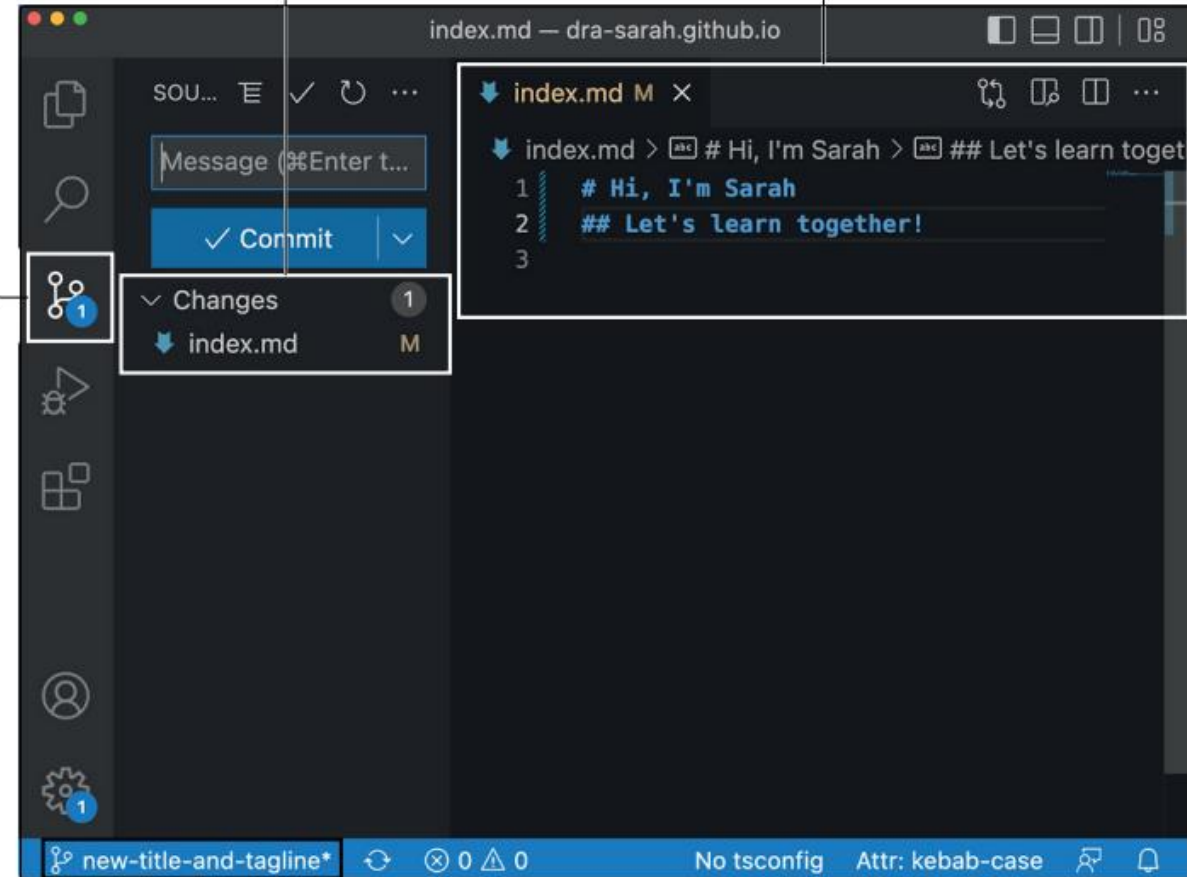


FIGURE 5-7:
VS Code after a modification has been made to the index.md file, but the change hasn't been committed.

The branch with an * showing there are changes that have not been committed or pushed yet

-
4. Open the Source Control pane by clicking the Source Control button. You see the changes in the changes list. The first time you're using VS Code to stage, commit, and push changes to a GitHub repo, if you try to commit changes that haven't been staged yet, you're asked whether you want to set all unstaged changes to be staged when you click the Commit button. For more information, you can check out the VS Code docs on Git support at [https://code.visualstudio.com/docs/editor/ versioncontrol#_git-support](https://code.visualstudio.com/docs/editor/versioncontrol#_git-support).
 5. Stage and commit the changes. Write a commit message. Then click Commit. Figure 5-8 shows how VS Code represents code that has been committed, but not yet pushed.

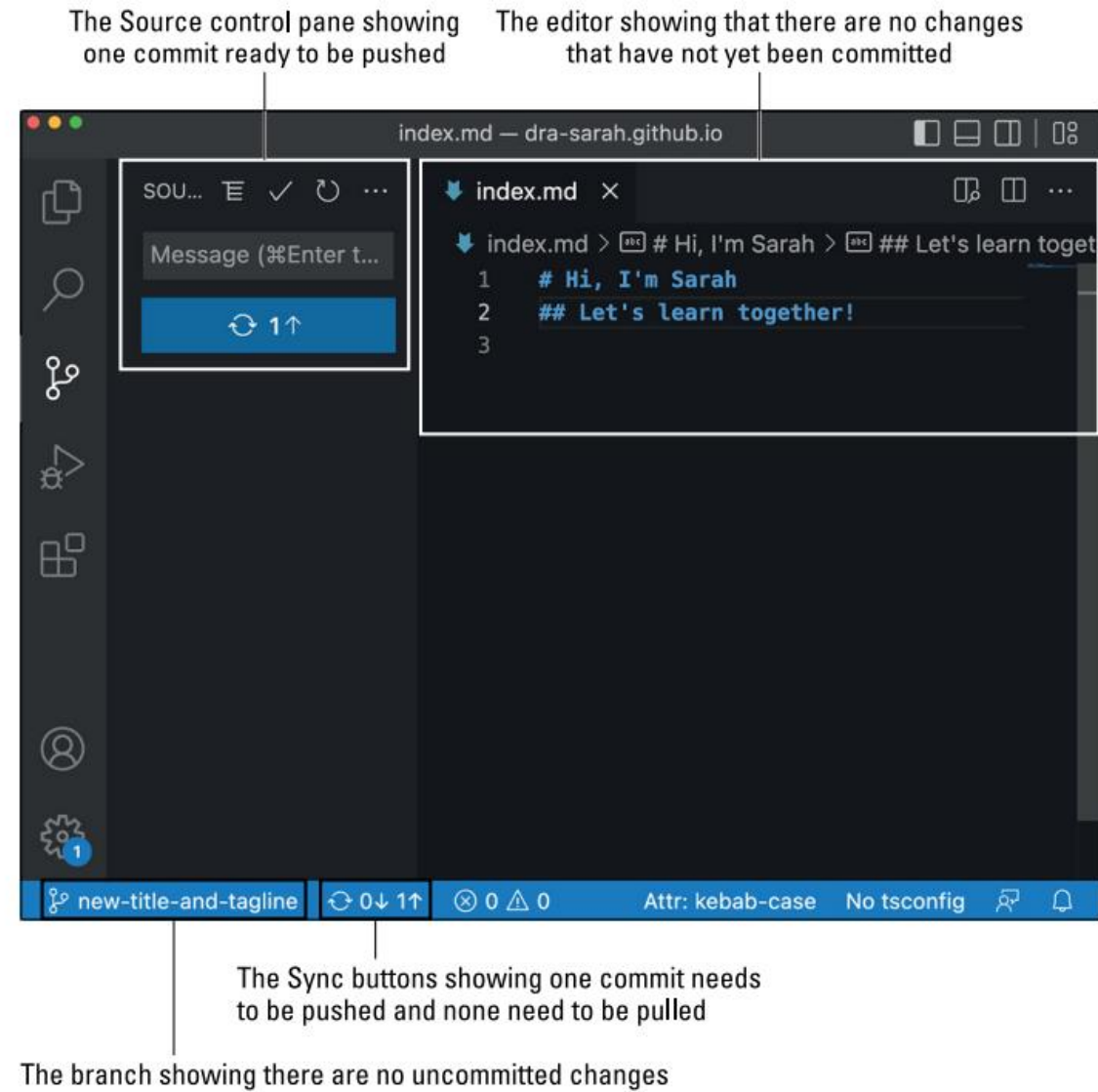


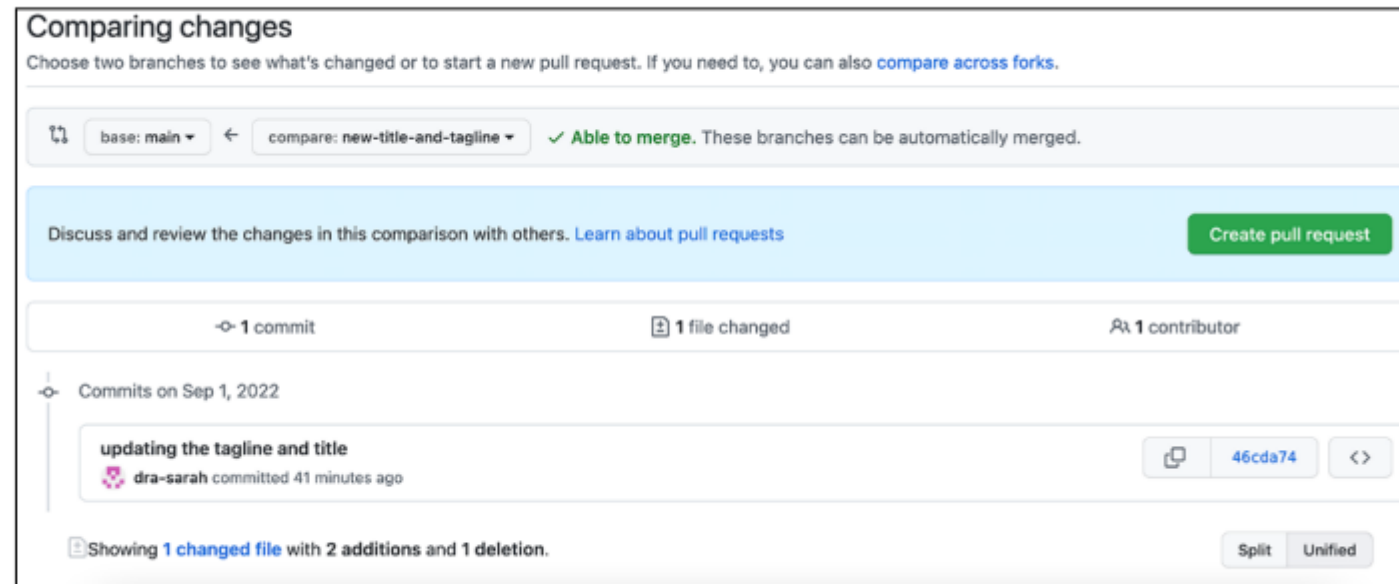
FIGURE 5-8:
VS Code after a
commit has
been made and
before the local
repository has
been pushed.

6. Push and merge the changes.

The Sync button that replaces the Commit button in Step 5, or the bottom bar to the right of the branch name updates to show one commit is ready to be pushed to the remote branch. Click the Sync button to push your changes to GitHub. Then open GitHub.com and go to the repository. In this example, the repository is <https://github.com/dra-sarah/dra-sarah.github.io>. You now see a suggestion to open a new pull request (see Figure 5-9). If you don't, you can click the Pull Request tab, click Create New Pull Request, and create the pull request.

Create and merge the pull request

FIGURE 5-9:
GitHub.com
indicates when a
branch is ready to
be merged.



Building Your Personal Website

Modifying the title and tagline To modify the title and tagline of your website, open the `_config.yml` file in VS Code. Add two lines above the only line that is in the file, indicating the theme. Your code should look like this:

```
title: <Your Name>
description: <Your Description>
theme: jekyll-theme-cayman
```

Adding sections to your website Adding sections to your website is made easy with Jekyll and GitHub Pages because you can use Markdown and HTML. First, include some social media usernames. Open your `_config.yml` file and add as many social media usernames as you want. In this example, I add a Twitter and GitHub username:

```
title: Your Name
description: Your Description
twitter_username: Your Twitter username
github_username: Your GitHub username
theme: jekyll-theme-cayman
```

Then, open the `index.md` file and change the code to include sections. For example, my code looks like this:

```
# My Projects
Here is a list of projects that I am working on:
# My Interests
I'm interested in teaching novice coders about computer science!
# My Blog
```

```
I'm really excited to blog my journey on GitHub.com.
# Get in Touch
<ul>
<li><a href="https://twitter.com/{{ site.twitter_username
  }}">Twitter</a></li>
<li><a href="https://github.com/{{ site.github_username
  }}">GitHub</a></li>
</ul>
```

Save, stage, commit, and push your changes to your branch and then create and merge the pull request into the main branch. After a couple of minutes, refresh the website page to see your changes

Creating a blog

Having a blog on your site is a great way for you to share your GitHub journey with others. As you start to discover and create, you can share what you learn and build with a community of people with similar interests. This section guides you through creating the blog posts and linking them from your index.md file.

First, create a new folder called `_layouts` and create a file within that folder called `post.html`. The `_layout/post.html` file should contain the following code to create a blog-style:

```
layout: default
<h1>{{ page.title }}</h1><p>{{ page.date | date_to_string }} -
  {{ page.author }}</p>
{{ content }}
```

Make sure that you correctly name the folder `_layouts`. Jekyll searches for the `_layouts` folder for any custom layouts. Otherwise, it uses the defaults for that theme. You can change the name of the specific layout — for example, `post.md` — but it should match the layout metadata, as shown in the following code snippet.

Using layouts and specific naming, Jekyll can extrapolate the title, date, and author to display that both on the blog post page and on the home page.

Then, create a new folder called `_posts` and a file inside with the date and a title. Typically, blog posts are made with `YEAR-MONTH-DAY-TITLE.md`. For example, this code is in a file called `_posts/2019-01-01-new-year.md`:

```
---  
layout: post  
  
author: sguthals  
---  
Write your blog post here.
```

Finally, in your `index.md` file, add the following code below the My Blog section:

```
<ul>  
{% for post in site.posts %}  
<li>  
<a href="{{ post.url }}">{{ post.title }}</a>  
</li>  
{% endfor %}  
</ul>
```

Save, stage, commit, and push your changes to your branch and then create and merge the pull request into the `main` branch. After a couple of minutes, refresh the website page to see your changes.

Linking project repos

You can link GitHub project repos to your website in the same way you link social media, described in the section “Adding sections to your website,” earlier in this chapter. Putting a link directly to a repo can be efficient.

Open the index.md file and add the following code, replacing the URLs with links to projects you’re the author of. This code shows linking to a project repo web page and directly to a project repo:

```
<ul>
<li><a href="https://sarah-wecan.github.io/HelloWorld/">Hello
  World Project</a></li>
<li><a href="https://github.com/thewecanzone/GitHubForDummies
  Readers">GitHub For Dummies Repo</a></li>
</ul>
```

Contributing to Your First

Forking GitHub Repositories

A fork of a repository is essentially just a copy of the repository. In the spirit of open source, forking is a way to share with and learn from other developers. Developers can have many motivations for forking a repository, but three of the most common reasons are to

- » Contribute to someone else's project
- » Use someone else's project as a starting point
- » Experiment with someone else's code without making changes to their project

Cloning, Forking, and Duplicating

When you clone a GitHub repo, you're creating a local copy of the project on your computer. Forking a GitHub repo creates a copy of the repo on your GitHub.com account, and from there, you can clone the repo. A link between the original repo and the one you forked remains, allowing you to pull changes made on the original repo into your copy and push changes that you make on your copy to the original copy.

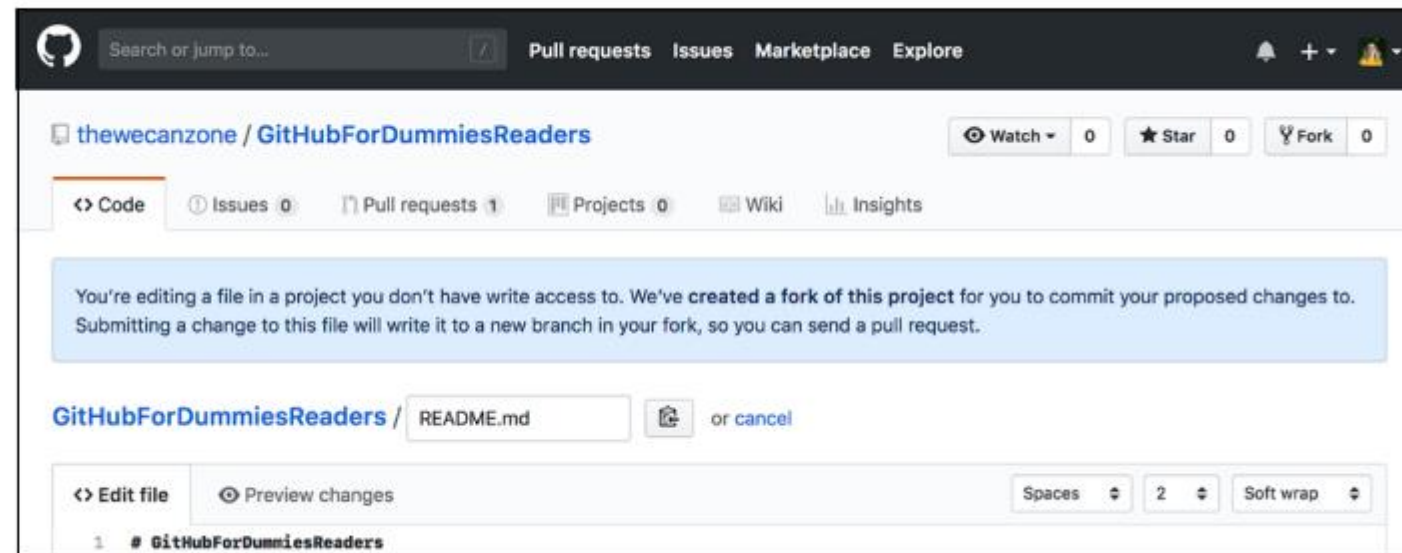
Duplicating a repo is when you make a copy that no longer has a link to the original copy. Duplication isn't a usual part of an open source workflow because it makes it more difficult to push changes back into the original repository. Even so, duplicating a repo can be useful sometimes, such as when the original project is no longer active and you plan to keep the project alive with your fork.

Cloning a Repository

Any public repo can be cloned, and you can run the code on your computer and make changes to the code. But you won't be able to push those changes back to the remote repo if you don't have push permissions to the repo.

Before you clone a repository, you should verify that you're able to push changes to it. The easiest way to verify this ability is to go to the repository home page. If you see a Settings tab on the right side of the home page, then you likely have push rights. If you don't, you likely have to fork the repo first. Alternatively, you can try to edit a file on GitHub.com, and if you get the message shown in Figure 6-1, then you don't have permission to directly contribute to the project.

FIGURE 6-1:
GitHub.com error
message if you
don't have edit
permissions
on a repo.



If you do end up cloning a repository where you don't have permission to contribute to it, you can end up making changes and not being able to push them. The section "Getting unstuck when cloning without forking," later in this chapter, gives steps for how to get out of this state.

After you have a repository cloned on your local computer, you can view and modify metadata about it in your terminal. Open the terminal and go to a directory where you have a GitHub repository. If you need an example, clone `https://github.com/thewecanzone/GitHubForDummiesReaders` by typing

```
$ git clone https://github.com/thewecanzone/  
  GitHubForDummiesReaders  
Cloning into 'GitHubForDummiesReaders'...  
remote: Enumerating objects: 15, done.  
remote: Counting objects: 100% (15/15), done.
```

```
remote: Compressing objects: 100% (15/15), done.  
remote: Total 15 (delta 4), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (15/15), done.  
$ cd GitHubForDummiesReaders
```

You can verify where the remote/target repo is with the following command:

```
$ git remote -v  
originhttps://github.com/thewecanzone/GitHubForDummiesReaders.  
  git (fetch)  
originhttps://github.com/thewecanzone/GitHubForDummiesReaders.  
  git (push)
```

If you cloned the same repo as I did, you see the exact same origin URLs for fetch and push. You should see that the remote repo is one owned by thewecanzone and not dra-sarah. Alternatively, if you run the same command on a repo that you own, you should see your username. For example, if I run the command in the directory where I cloned my website repo that I created in Chapter 4, I would see

```
$ git remote -v  
originhttps://github.com/dra-sarah/dra-sarah.github.io.git (fetch)  
originhttps://github.com/dra-sarah/dra-sarah.github.io.git (push)
```

If you try using the command on a Git repo that doesn't have a remote origin (meaning it isn't hosted on GitHub.com or any other remote place), I created a simple Git repo called git-practice. Running the command in that directory gives you nothing back:

```
$ git remote -v
```