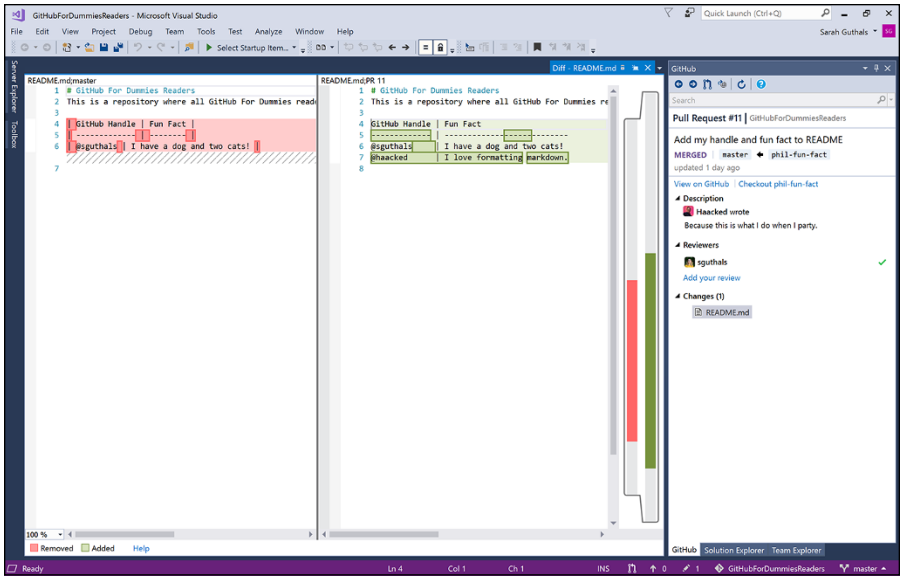

FIGURE 13-9: Interacting with a pull request in Visual Studio.

Following the GitHub for Visual Studio extension

The GitHub for Visual Studio extension is open source, so while it is highly recommended that you install and use the integrated Git experience for Visual Studio 2022, if you want to make improvements on the 2015, 2017, 2019 GitHub integration, you still can! Go to `https://github.com/github/visualstudio` to find documentation for how to use the GitHub for Visual Studio extension to improve your development workflow.

Using GitHub for XCode

Apple has developed an integration for GitHub in XCode. After you install XCode, you can sign in with GitHub by clicking the Clone from Existing Git Repository button, clicking the arrow next to the message to sign in to an account, and then opening the Accounts menu. Click the + button and choose GitHub, and click Continue, as shown in Figure 13-10. Follow the prompts to sign in to your GitHub account. If you have two-factor authentication set up, you are asked for your 2FA code.

TIP

You need to have a personal access token to sign in to GitHub. You can create one at `https://github.com/settings/tokens`.

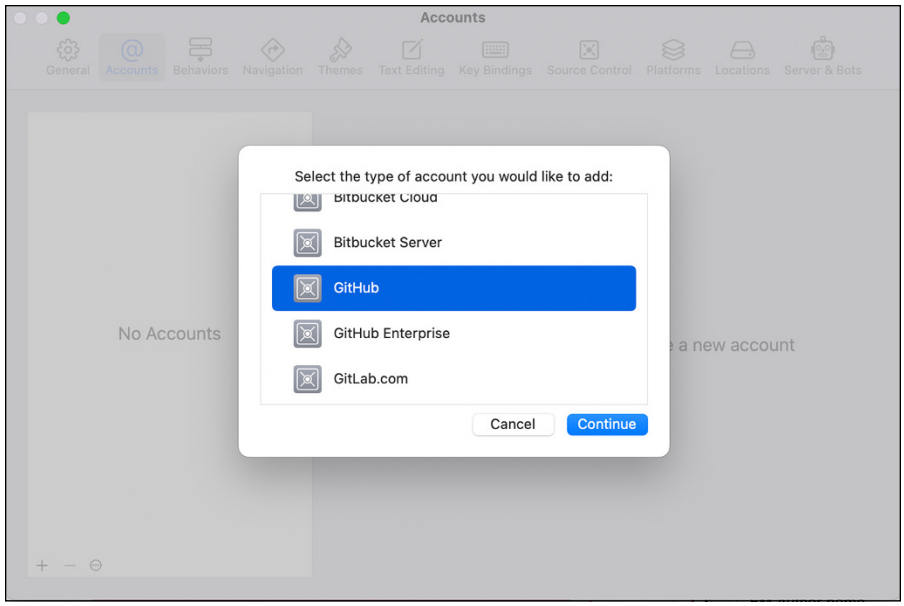

FIGURE 13-10: Signing in to GitHub inside of XCode.

Now when you choose Source Control➪Clone, your GitHub repos load below the box where you can insert a URL to clone. When you click one, you get information, such as the primary language used, the number of forks and starts for this repo, and a link to the README file, as shown in Figure 13-11.

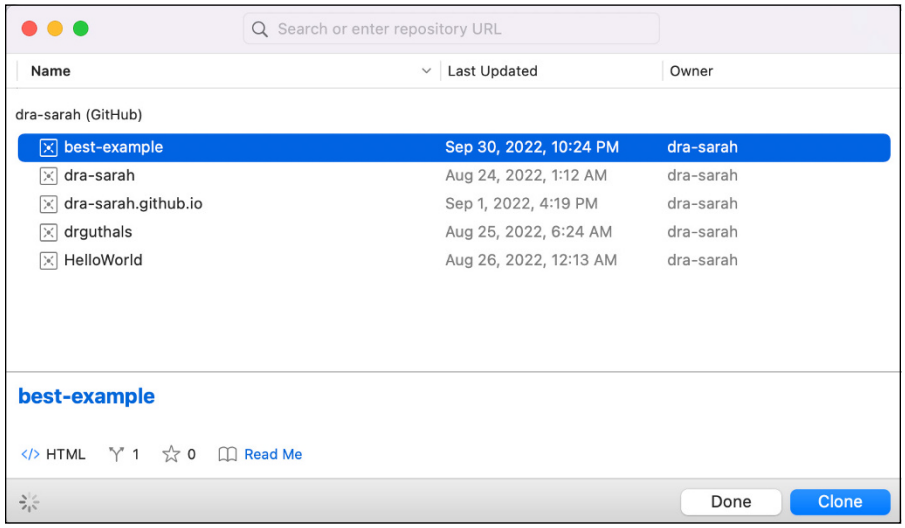

FIGURE 13-11: List of GitHub repositories you have access to clone from inside XCode.

If you have a file open in XCode, the Source Control menu displays additional actions you can perform, such as Commit, Push, Pull, and Fetch and Refresh Status.

This extension isn't open source, so it's best to keep up with the latest Apple developer news to know what new features may be available. You can read about all the source control integrations that Apple releases at `https://developer.apple.com/documentation/xcode/source-control-management`.

Using GitHub for IntelliJ

The IntelliJ IDE from JetBrains has GitHub pull request support for any GitHub repo you have open. After you install IntelliJ, you can choose to clone a project from the Start menu, as shown in Figure 13-12.

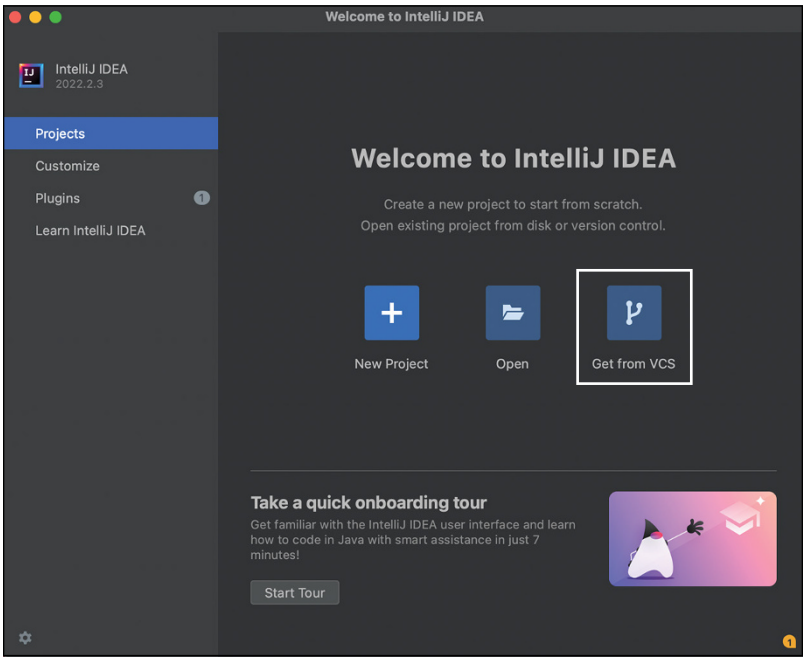

FIGURE 13-12: Clone from a version control system to get started integrating GitHub into IntelliJ.

A new window asks for a URL. On the left side panel of this window, click the GitHub button, and then click the Log In via GitHub button, as shown in Figure 13-13. You're asked to authorize GitHub in the browser, similar to signing in with Visual Studio Code. After you have successfully logged in, all the GitHub

repositories that you have access to appear in the clone list, as shown in Figure 13-14.

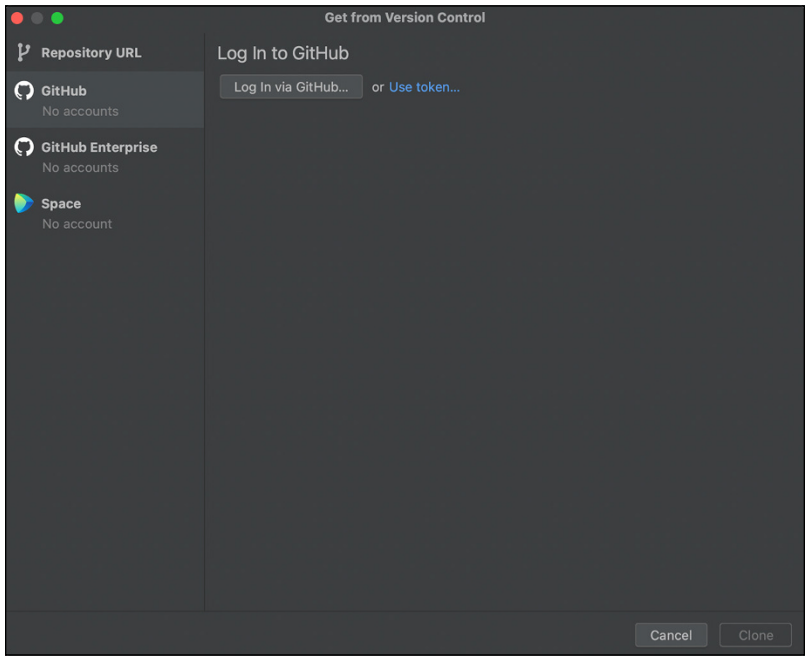

FIGURE 13-13: The GitHub log-in window inside IntelliJ.

After you choose a repository and click Clone, an IntelliJ project window opens with your project. When the project is open in IntelliJ, you can open the GitHub pull request preview by clicking the Pull Requests button on the left pane. A new section opens in the IntelliJ window with a list of the open pull requests, as shown in Figure 13-15. If you click one, the description and list of changed files opens. If you double-click one of the changed files, a diff of that file opens in a new window (see Figure 13-16).

Lastly, you can create a pull request from inside of IntelliJ as well. If you're on a new branch and have already made some changes and committed them to your branch, you can click the + above the list of pull requests, from the same Pull Request button on the left side of your project window shown in Figure 13-15. In the panel, shown in Figure 13-17, specify the title and description of your pull request, and add reviewers, assignees, or labels if you want to.

This GitHub pull request feature is embedded in the IntelliJ IDE, so it's best to follow the IntelliJ blog and documentation for up-to-date information on its development.

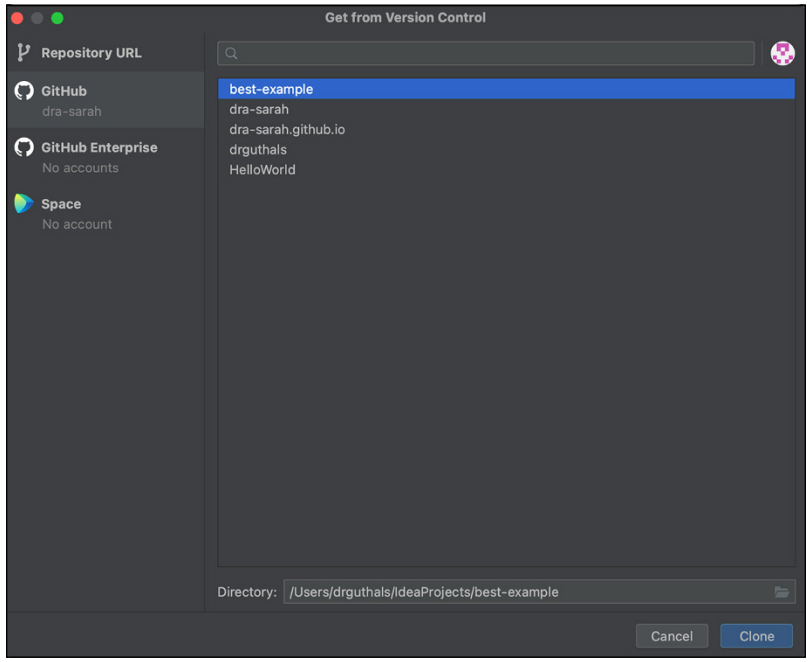

FIGURE 13-14: List of GitHub repositories you have access to clone from inside IntelliJ.

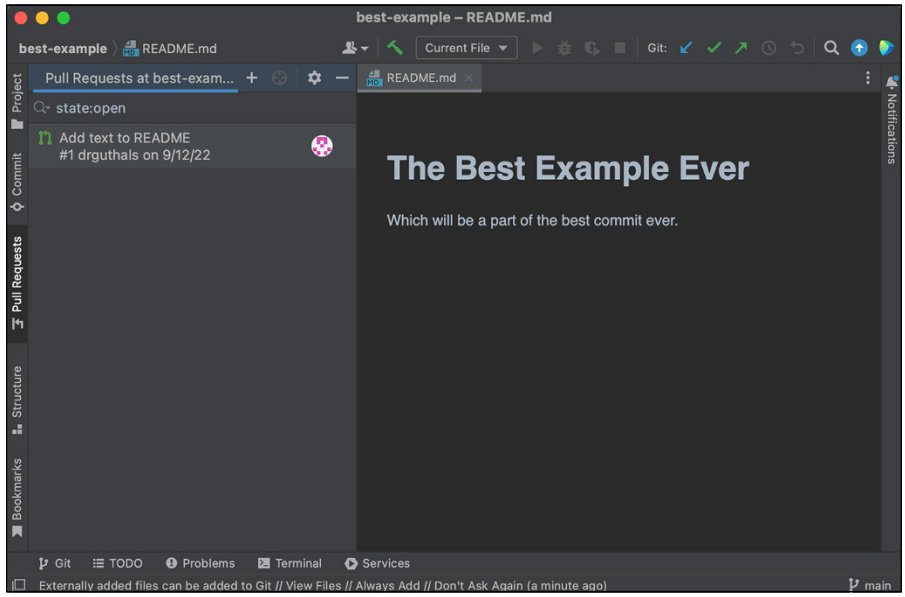

FIGURE 13-15: A list of open pull requests from inside of IntelliJ.

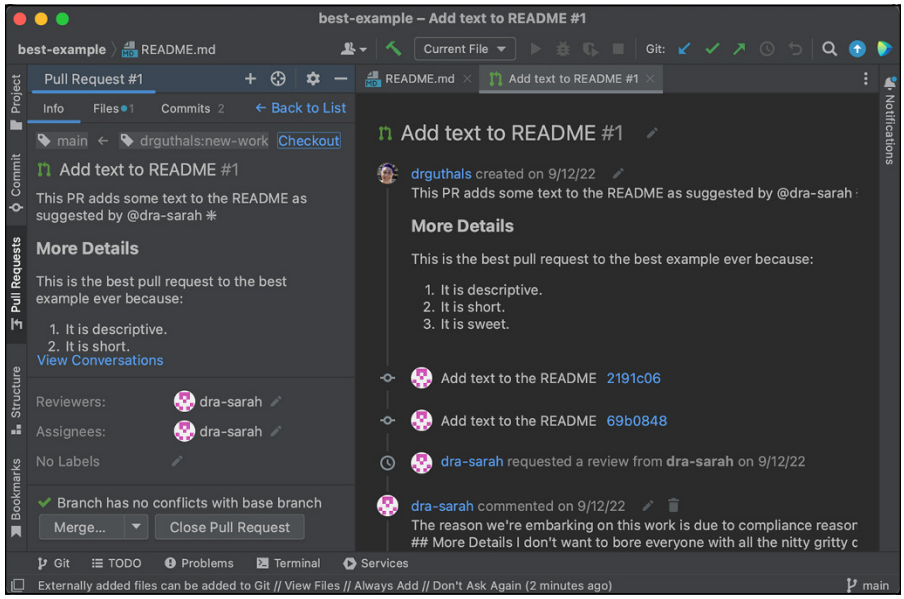

FIGURE 13-16: Viewing a pull request from inside of IntelliJ.

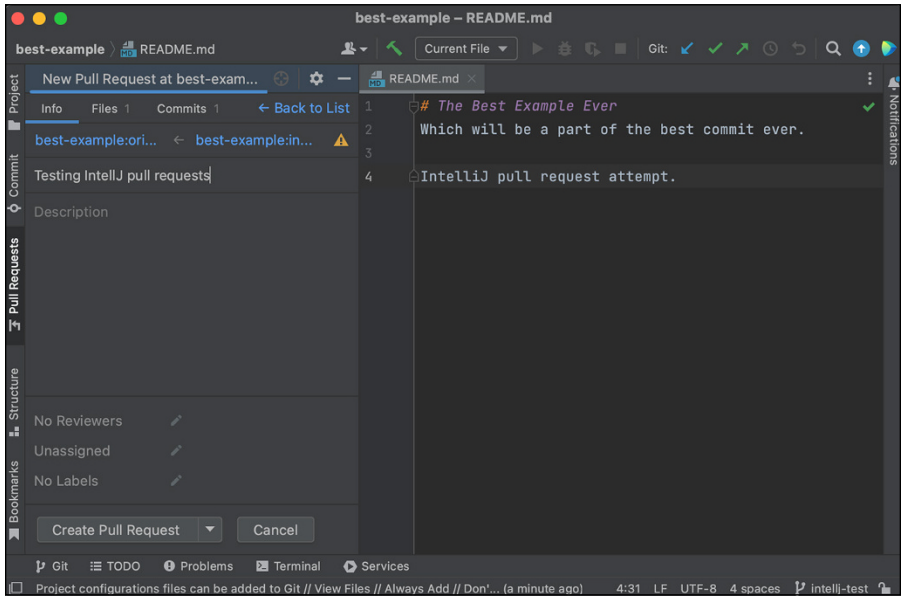

FIGURE 13-17: Creating a pull request from inside of IntelliJ.

IN THIS CHAPTER

- Using Chrome Extensions to extend GitHub
- Setting up Probot to personalize GitHub
- Using GitHub Actions to customize GitHub

Chapter 14
Personalizing GitHub

Developers have a lot of opinions about how they work and invest a lot of time personalizing their tools to work just the way they want. GitHub offers an extensive API that lets developers write tools to interact with GitHub data in a myriad of ways. Also, because GitHub.com runs in a browser, anyone can build browser extensions to customize the experience of using GitHub.

In this chapter, I look at some of the available ways to personalize your use of GitHub.

Using Browser Extensions

Browser extensions can completely customize the experience of using a browser. You can find extensions for every possible scenario you can think of.

In this section, I look at a few useful extensions that work with GitHub. Some of these extensions are available for multiple browsers, but I focus on Google Chrome extensions for brevity.

For a more comprehensive list of browser extensions that work with GitHub, check out this list of awesome browser extensions for GitHub repository made by Stefan Buck at `https://github.com/stefanbuck/awesome-browser-extensions-for-github`.

Refining GitHub

The Refined GitHub extension is an open source extension that simplifies the GitHub interface and adds some useful features to GitHub.com. The extension is available for the Chrome, Firefox, and Opera browsers.

You can find the source code at `https://github.com/sindresorhus/refined-github`. The README has a link to install the extension. Note that when you install the extension, you grant it the ability to read and change your data on `api.github.com`, `gist.github.com`, and `github.com`. It can also modify data you copy and paste.

After you install the extension, you should see an Octocat icon to the right of the address bar. This icon provides some light customization options. One option lets you define some custom CSS specific to GitHub.com. In Figure 14-1, you can see I added some custom CSS to make the repository name larger and dark red. Note that after you change the CSS, you have to refresh the page to see your changes in effect.

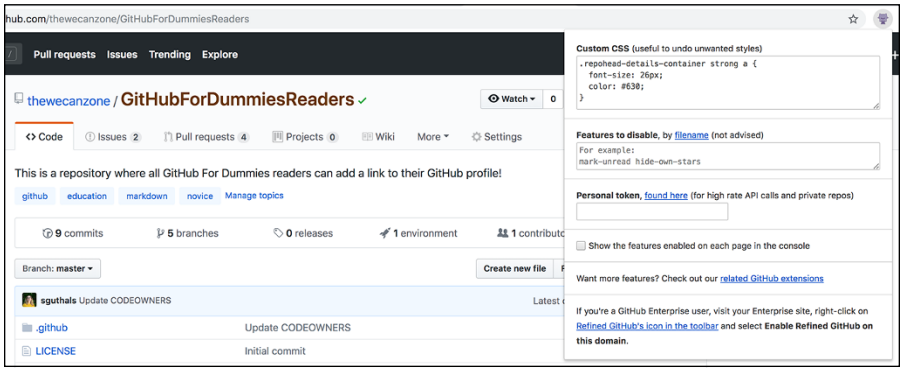

FIGURE 14-1: Configuring the Refined GitHub extension.

Changing the CSS is just a parlor trick compared to the many other enhancements Refined GitHub brings with it.

WARNING

Browser extensions work by manipulating the generated HTML of a website. It often looks for known *landmarks* in the pages (such as an HTML element with a known ID) and then adds its own UI elements, removes elements, or changes elements. However, it does this all outside of the code running on the server that actually generates the HTML. What this means is that if a website such as GitHub.com changes its HTML markup, an extension feature could stop working temporarily until the authors update their extension to adapt to the new change. So if any of these features stop working, try again later after the extension is updated.

The following is a small sampling of enhancements that are on as a default when you have this extension installed:

- **Mark issues and pull requests as unread.** This enhancement adds a Mark as Unread button to the Notifications section of an issue or pull request. Click it to mark the issue or pull request unread, which puts it back in your notifications list.
- **Stop the page jumps from recently pushed branches.** Normally, when you push a new branch, the home page of a repository displays a list of recently pushed branches in a yellow bar above the list of code files. The sudden appearance of this list can cause the rest of the page to jump down to make room for the bar. Refined GitHub displays the list of recently pushed branches in the top right overlaying the location where the Fork button may be. By displaying the branches in an overlay, Refined GitHub ensures that the page doesn't jump down.
- **Adds option to wait for successful checks.** If you have continuous integration (CI) set up for a repository, it can take a while after someone pushes a pull request before all the checks are completed. The CI might be running static analysis or a linter, unit tests, integration tests, and so on. It can be annoying to wait for all those processes to complete after you've reviewed some code and are ready to merge the pull request. Refined GitHub adds a check box that lets you indicate that it should go ahead and merge the pull request after the checks are complete and successful.
- **Reaction avatars show who reacted to a comment.** Typically a reaction comment shows only the reaction and the count for that reaction. With Refined GitHub, you can see who all gave a specific reaction.

Refined GitHub contains many more enhancements big and small. The list here is just the tip of the iceberg. In Figure 14-1, notice that there is also a place to disable features that are a part of the extension, as well as a place to put your GitHub personal access token so that the extension can work on private repos as well.

Taking a GitHub selfie

As an open source project maintainer, I am ecstatic when someone comes along and submits a pull request for a project. I'm happy when someone opens an issue that identifies a problem I didn't know about. I feel gratitude for the folks who take time out of their day to help out my project.

And sure, I could use words to communicate my gratitude, but they always seem to fall short of my true feelings. If the old saying that a picture is worth a thousand words is true, how many words is an animated gif worth?

GitHub Selfie is an open source browser extension (`https://github.com/thieman/github-selfies`) that adds a Selfie button to the comment field that lets you take a selfie using your computer's camera. You can choose to take a still picture, but where's the fun in that? It also provides an option to take an animated gif.

Figure 14-2 shows a ridiculous self-portrait.

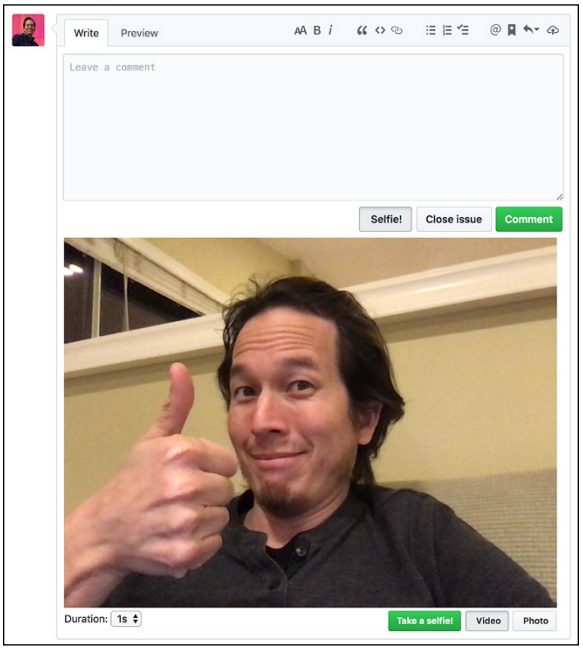

FIGURE 14-2: Taking a short video with GitHub Selfie.

Adding a selfie to express your gratitude is a small detail, but adds a nice warm personal touch when you're working with people from all over the world.

GitHub Apps and Probot

Chapter 12 covers a few integrations that connect GitHub to other applications, such as Slack and Trello. What those integrations have in common is they were implemented as GitHub apps.

Apps on GitHub let you extend GitHub in powerful ways. GitHub apps are web applications that can respond to events on GitHub. These event subscriptions are called *web hooks.* When an event occurs on GitHub that the app is interested in,

GitHub makes an HTTP request to the app with information about the event. The app can then respond to that event in some manner, often resulting in a call back to GitHub via the GitHub API.

In this section, I walk you through building a simple GitHub app that brings a bit of levity to your issue discussions. There's an old meme in the form of an animated gif with a little girl who asks the question, "Why don't we have both?" The typical application of this meme is in response to a question that presents a false dichotomy. In other words, when someone presents a question with two choices, someone might respond with this image.

In this section, you create a GitHub application that will automatically do this as a fun exercise.

Introducing Probot

GitHub apps are web applications that need to listen to HTTP requests. You have a lot of important details to get just right when building an HTTP request, such as what is the format of the data posted to the app? All these details can be confusing and time consuming to get correct when building a GitHub app from scratch. Knowing where to start is difficult.

GitHub's Probot framework comes in handy when getting started with a GitHub app. Probot handles much of the boilerplate and nitpicky details of building a GitHub app. It's a framework for building GitHub apps using Node.js. It provides many convenience methods for listening to GitHub events and for calling into the GitHub API.

Probot makes it easy to build a GitHub app, but it doesn't solve the problem of where to host the app.

Hosting the app

A GitHub app can take many forms. It could be a Node.js app running in Heroku, a serverless function in GitHub Actions, or any other cloud provider — it doesn't matter. It just needs to be persistent and available via the public Internet so that GitHub can reach it with event payloads.

Setting all that up can be time consuming, so for my purposes, I developed the simple Probot app locally using the command line interface (CLI) and then used Glitch to deploy.

Introducing Glitch

Glitch (`https://glitch.com`) is a hosting platform for web applications that removes a lot of the friction with getting a web app up and running. Any app you create in Glitch is live on the web from the beginning. You don't have to think about how you plan to deploy the code because any change you make is auto-saved and automatically deployed.

Glitch focuses on the community aspect of building apps. Every file can be edited by multiple people in real-time, in the same way you might edit a document in Google Docs. And every project can be remixed by clicking a button. This encourages a lot of sharing of code and learning from each other, which comes in handy when we build our own GitHub app.

Before you continue, make sure to create an account on Glitch if you don't have one already.

Creating a Probot app

Follow these steps to get started:

1. **Create a local Probot app by opening a command line and typing:**

```
npx create-probot-app why-not-both
```

2. **Type** `y` **to create the Probot app locally, and then answer the questions in the command line.**

   For example:

```
? App name: why-not-both
? Description of app: A probot app to reply to issues
? Author's full name: drguthals
? Which template would you like to use? basic-js => Comment
  on new issues
```

3. **Once the files are created, open the folder in Visual Studio Code (or your editor of choice) and edit the index.js file with the following code:**

```
/**
 * This is the main entrypoint to your Probot app
 * @param {import('probot').Probot} app
 */
module.exports = (app) => {
  // Your code here
  app.log.info("Yay, the app was loaded!");
```

```
  app.on("issues.opened", async (context) => {
    const message = context.payload.issue.body;
    if (message.indexOf(' or ') > -1) {
      const issueComment = context.issue({
        body: "![The why not both girl](https://media3.
  giphy.com/media/3o85xIO33l7RlmLR4I/giphy.gif)",
      });
      return context.octokit.issues.
  createComment(issueComment);
    }
  });

  // For more information on building apps:
  // https://probot.github.io/docs/

  // To get your app running against GitHub, see:
  // https://probot.github.io/docs/development/
};
```

This code listens to new issue comments, looks for the word *or* surrounded by spaces, and if it finds it, creates a new comment with a markdown image.


TIP

This approach is not very smart. You can find a slightly better approach at `https://git.io/fhHST`. It would be even better if I could employ artificial intelligence (AI) in the form of natural language processing (NLP). But that's beyond my skillset and out of the scope for this book.

4. **Save the file, and back in your terminal, start the server with the following command:**

```
npm run start
```

5. **Once the server starts, open a browser and point it to localhost:3000.**

   The Register GitHub App button is now part of the interface, as shown in Figure 14-3.

6. **Click the button and follow the instructions to register the app to your GitHub account and whatever repositories you want to associate it with.**

7. **After registering, shut down the local server by pressing ⌘-C or Control+C.**

8. **Open the `.env` file and notice that GitHub set critical environment variables:**

   - WEBHOOK_PROXY_URL
   - APP_ID

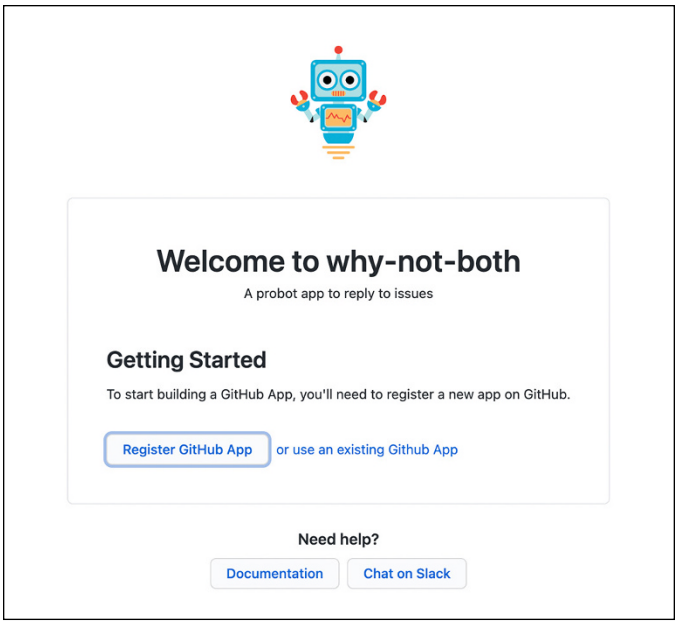

FIGURE 14-3: The Probot app with a button to register the app with GitHub.

- PRIVATE_KEY
- WEBHOOK_SECRET
- GITHUB_CLIENT_ID
- GITHUB_CLIENT_SECRET

9. **Test your Probot app by restarting the npm server with the following command:**

```
npm run start
```

10. **Go to the one of the repositories that you authorized and create a new comment with the word *or* in it.**

    Watch the Probot app automatically responds with a gif of the "Why not both" meme, as shown in Figure 14-4.

Make sure you shut down your npm server by pressing ⌘-C or Control+C in your terminal.

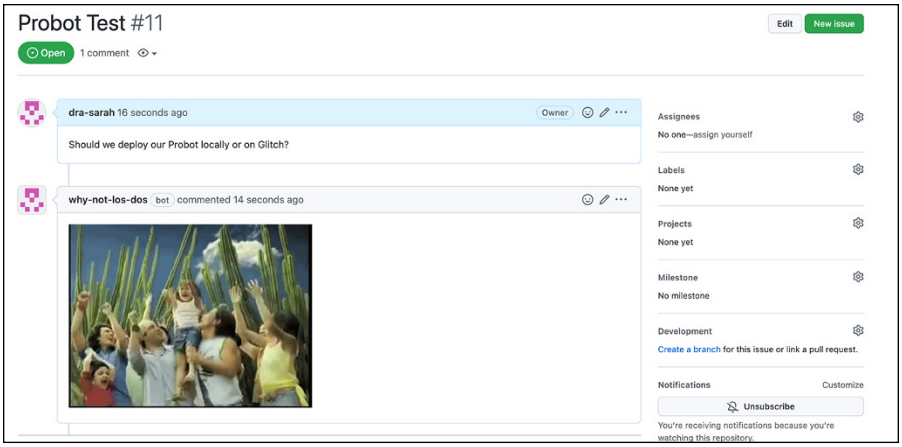

FIGURE 14-4: The Probot app responding to a new issue that has the word *or* in it.

Pushing the Probot app to GitHub

Now that you have verified the Probot app works, it's time to get it hosted in the cloud. To get started, you need to push your code to GitHub. In your Probot root folder, initialize a Git repository with the following command:

```
git init –b main
```

Next, stage your initial commits with the following command:

```
git add . && git commit –m "initial commit"
```

Finally, push your repository to GitHub. To do this, type the following command into your terminal:

```
gh repo create
```

This kicks off the GitHub repository creation flow:

1. **Using your arrow keys, choose Push an Existing Local Repository to GitHub.**
2. **Press Enter to choose . as the location for the repository (this means the current folder).**
3. **Press Enter to choose the name of the folder as your repository name, or type a new repository name.**
4. **Add a short description, such as "A simple Probot app".**

5. **Using your arrow keys, choose Public, Private, or Internal for the repository visibility.**
6. **Type `Y` to create a remote to the GitHub repository with your local repository.**
7. **Press Enter to choose origin as the name of the remote.**
8. **Press Enter to choose Yes to commit your local changes to the remote GitHub repository.**

You can now navigate to your repository on GitHub and your code should be there. You can see mine at `https://github.com/drguthals/why-not-both`.

Hosting your Probot app on Glitch

Now that you have confirmed your Probot app works when it is locally hosted and you have published your code to GitHub, you can easily use Glitch to host your Probot app. Go to `https://glitch.com`, sign in, and choose New Project➪ Import from GitHub.

Type your repository name to the text field, making sure to include `.git` at the end. For example, for my Probot app I would type `https://github.com/drguthals/why-not-both.git`.

Once Glitch loads your code in the online Glitch editor, you should see all the files from your repo have been imported into Glitch and the `README.md` file is open in the code editor.

You now have to register this new Probot app with GitHub. To do this, follow these steps:

1. **Click Preview, found at the bottom of the Glitch editor.**
2. **Choose Preview in a New Window.**
3. **Click the Register GitHub App button.**
4. **Name your GitHub app.**
5. **Choose which organizations, accounts, and repositories to install your app on.**
6. **Click Install.**

Head to your repository and make sure the Glitch-hosted Probot app works by opening a new issue. You should get a response, as shown in Figure 14-5.

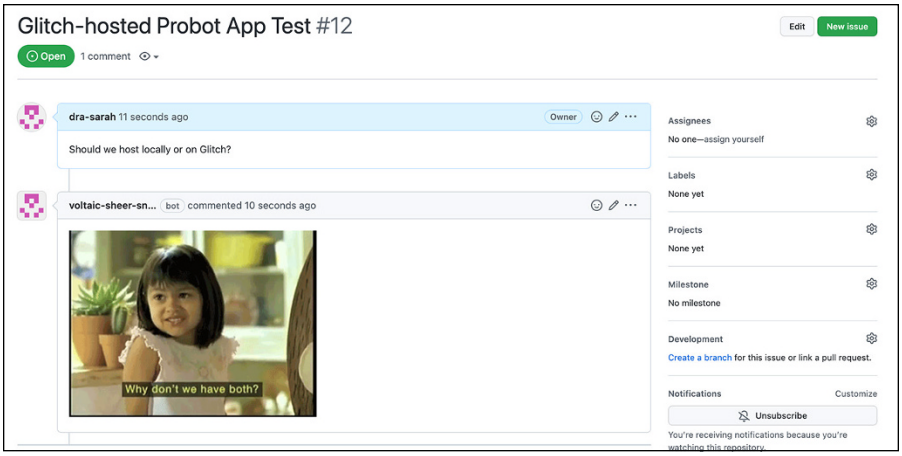

FIGURE 14-5: The same Probot app in action on a GitHub issue, but this one is hosted on Glitch.

Taking Action with GitHub Actions

In the previous section, I walk through personalizing GitHub by creating a GitHub app. This required that you host your app outside of GitHub. It turns out GitHub has a feature that removes the need to host your app outside of GitHub, which can reduce the number of moving parts when extending GitHub. This feature is called GitHub Actions.

GitHub Actions makes it possible to create custom workflows on GitHub. It lets you implement custom logic to respond to events on GitHub. In the previous section, you wrote a GitHub app to do that. With GitHub Actions, you don't need to build a custom app. You can build workflows using existing actions that others have written, or you can write your own actions that run in a Docker container.

EXPLORING GitHub ACTIONS

There are pre-written GitHub Actions — some really useful and complex — for almost every part of your developer workflow on GitHub. You can find a list of them at `https://github.com/marketplace?type=actions`. You can also follow some tech folks who write and open source more specific GitHub Actions.

You can also write custom Actions. How to do that is beyond the scope of this book, but you can look at the Actions docs at `https://docs.github.com/actions`.

To demonstrate GitHub Actions, consider the following scenario. You're maintaining an open source repository, but aren't able to be respond to every single new issue notification from new contributors. You can create a very similar GitHub Action to the Probot you created earlier in this chapter using the prewritten First Interaction Action at `https://github.com/marketplace/actions/first-interaction`.

The following steps guide you through setting up a GitHub Action:

1. **Click Actions at the top of your repository.**
2. **Click the New Workflow button.**

   You're taken to a page to choose a workflow or create one from scratch, as shown in Figure 14-6.

3. **Search for the Greetings Action and click Configure.**

   This creates a `greetings.yml` file in your `.github/workflow` folder, as shown in Figure 14-7.

4. **Click Start Commit and commit the new `greetings.yml` file to the `main` branch.**
5. **Invite a friend to open a new issue on your repository for the first time.**

   Right when the issue is created, if you head back to the Actions tab you can see the workflow running, as shown in Figure 14-8. When it completes, a comment is added to the new issue, as shown in Figure 14-9.

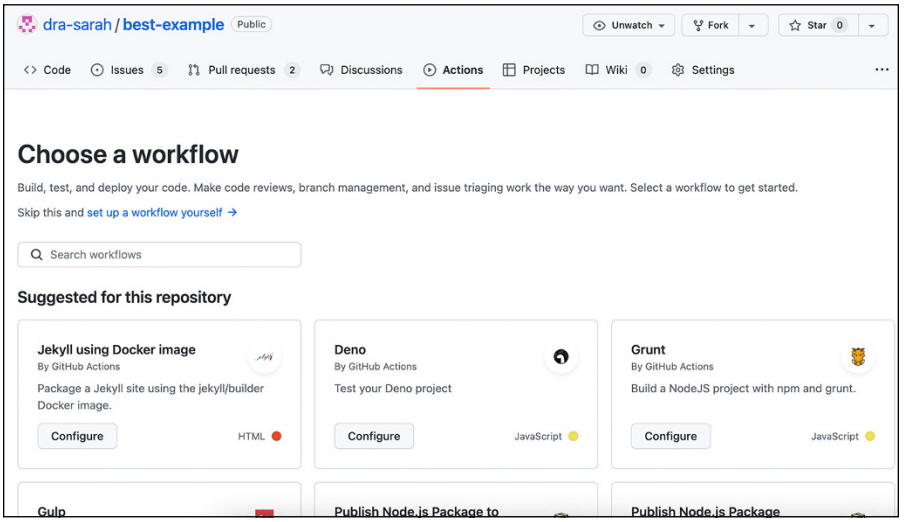

FIGURE 14-6: Options for choosing a workflow.

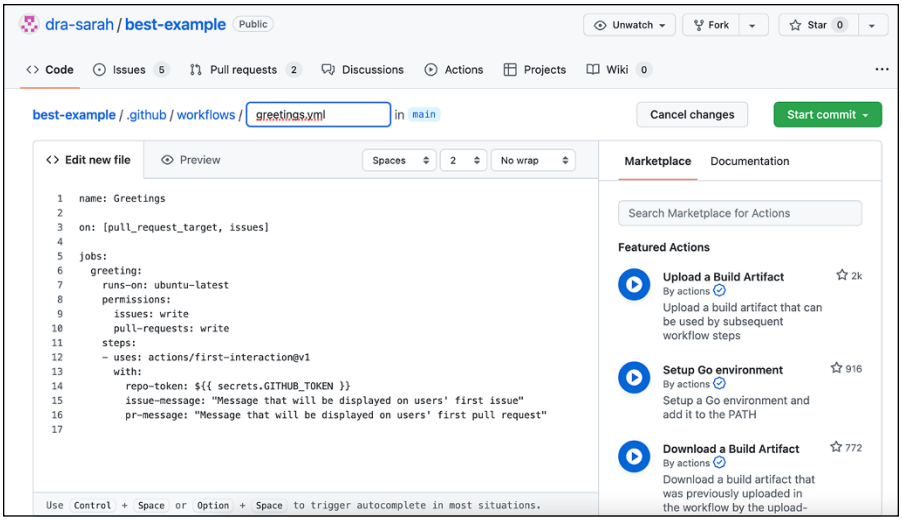

FIGURE 14-7: Create a greetings.yml file to install the GitHub Action.

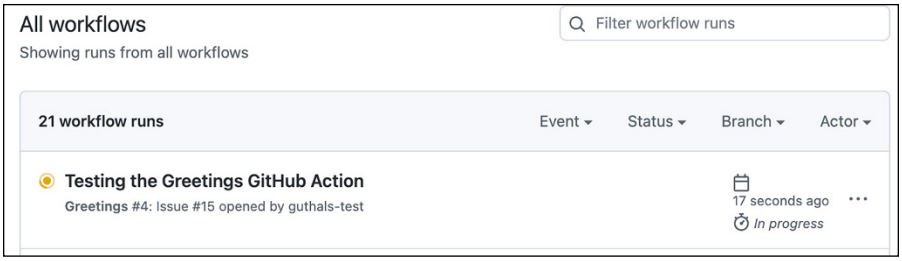

FIGURE 14-8: The GitHub Action running.

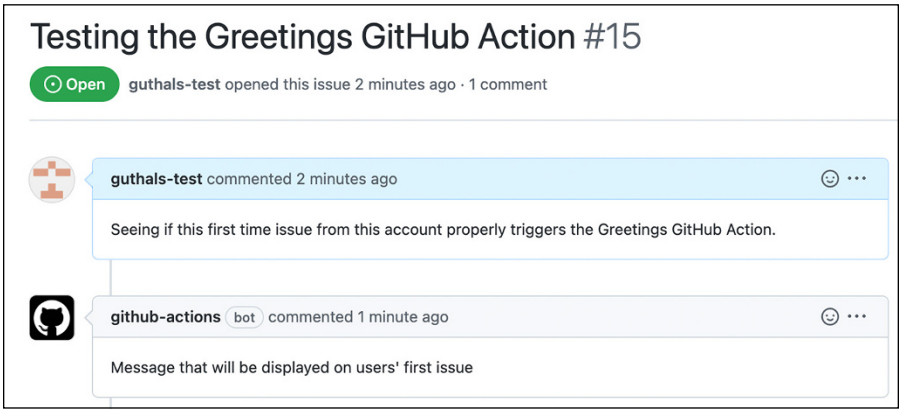

FIGURE 14-9: The comment added to a new issue by the GitHub Action.