

GitHub for Mathematicians

GitHub for Mathematicians

Steven Clontz
Univeristy of South Alabama

January 3, 2024

Website: g4m.clontz.org¹

©2023–2024 Steven Clontz

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit [Creative Commons.org](http://creativecommons.org/licenses/by-sa/4.0)²

¹g4m.clontz.org

²creativecommons.org/licenses/by-sa/4.0

Abstract

Increasingly, the cyberinfrastructure of mathematics and mathematics education is built using GitHub to organize projects, courses, and their communities. The goal of this book is to help readers learn the basic features of GitHub available using only a web browser, and how to use these features to participate in GitHub-hosted mathematical projects with colleagues and/or students.

Acknowledgements

Thanks to the following people who've contributed to this handbook.

- The [American Institute of Mathematics](#)³, for funding my travel to JMM 2024 to run a professional enhancement program based upon this handbook.
- Jeremy Avigad, for adding Codespaces support to his book *Mathematics in Lean* in time for JMM 2024.
- Francesca Gandini, for co-organizing the JMM 2024 professional enrichment program that this book was written for originally.

³aimath.org

Contents

Abstract	iv
Acknowledgements	v
1 Intro to Git & GitHub	1
1.1 What Is Git?	1
1.2 What Is <i>GitHub</i> ?	2
1.3 G4M on <i>GitHub.com</i> .	2
2 Your First Repository	3
2.1 Making an Account	3
2.2 Creating the Repo	3
2.3 Editing README.md	3
2.4 Using <i>GitHub.dev</i> .	4
2.5 Committing Your Work	5
3 GitHub Pages	6
3.1 Creating a Simple Webpage	6
3.2 Using a Template	7
3.3 Customizing Your Site	8
4 Writing and Running Code	11
4.1 Codespaces	11
4.2 Python and Jupyter Notebooks	12
4.3 Previewing GitHub Pages	12
4.4 Managing Your Codespaces	13
5 Collaborating with Others	15
5.1 Live Share	15
5.2 Collaborators and Pull Requests	15
5.3 Forks	17
5.4 Handling Merge Conflicts	18

6	Math Projects Powered by GitHub	19
6.1	PreTeXt authoring system	19
6.2	π -Base Community Database of Topological Counterexamples	19
6.3	Lean Theorem Prover	20
6.4	code4math	20

Appendices

A	Additional Reading	21
B	Additional Topics	22
B.1	GitHub Desktop	22
B.2	VS Code Application	22
B.3	Using the Terminal	22
C	Definitions and Notes Quick Ref	23

Back Matter

Chapter 1

Intro to Git & GitHub

1.1 What Is Git?

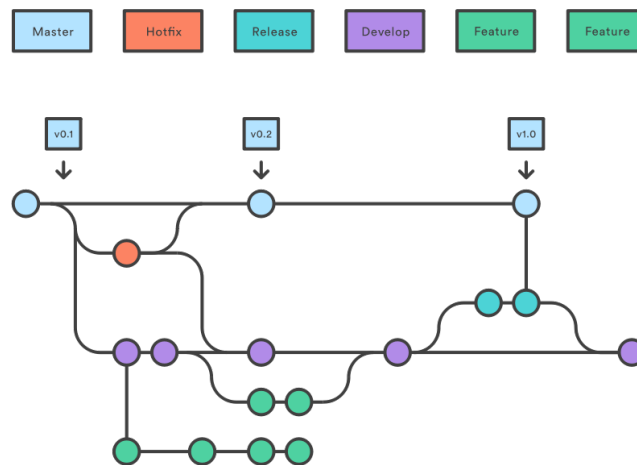


Figure 1.1.1 An illustration of a project’s history controlled by Git

Git is a distributed version control system that tracks changes in any set of computer files. This software was originally authored by Linus Torvalds in 2005 for development of the Linux kernel. Importantly, Git is free and open-source software, which means you have the legal and practical ability to use it however you want, and even modify it for your purposes if you wanted.

Two core concepts of Git are **commits** (illustrated in Figure 1.1.1 by circles) and **branches** (illustrated in Figure 1.1.1 by lines). A commit represents the state of of your project at a particular point in its history. Branches allow this history to not be linear: you can branch off to experiment on a particular new feature, then merge this “feature branch” back into the “main branch” when it’s complete. This is particularly useful when multiple people collaborate (Chapter 5) on a Git-managed project. Finally, a Git project is often called a **repository**, or **repo** for short.

Since you’re reading *GitHub for Mathematicians*, I’m obligated to describe Git history as either a finite partial order, or a loopless directed graph, depending on your preferred flavor of mathematical models. In particular, you might consider the normal history of a file to be a linear order or directed path: `article.tex`, then `article-dec-1.tex`, then `article-dec-1-fixed.tex`, and so on. But with Git, you don’t need to track your version history with filenames, you (and your colleagues) can branch your history into several time-

lines, you can merge them back together again, and look up the state of your project at any point where you committed your work.

1.2 What Is *GitHub*?

Another key feature of Git is the ability to share your project, along with its history, with other people. This is generally accomplished by hosting your repository on a service such as **GitHub**: [GitHub.com](https://github.com)¹. (Other such services include [BitBucket.org](https://bitbucket.org)² and [GitLab.com](https://gitlab.com)³.)

Importantly, GitHub is *not* itself open-source software, but is a service owned and operated by Microsoft. However, Microsoft makes GitHub available for use at no cost to the public, with additional “pro” features available for free to instructors and researchers.

We’ll use GitHub not only as a host for our repositories, but also to take advantage of all the tools it provides to author content using only a web browser. If you’ve looked into using Git in the past, you may have hesitated due to the apparent need for software developer experience to get started. However, using GitHub’s web applications, there will be no need for complicated installations or memorizing command line incantations like `git commit -m "foobar"` to type into a terminal. (Of course, you still *can* choose to use such tools to get as much control over your Git project as you want, should the need ever arise: see [Appendix B](#))

Another reason to use GitHub: community! GitHub is often marketed as a “social coding platform”, because it not only provides tools to create and deliver digital content, but it also provides social features such as Following users, Starring repositories, participation in Discussions and Issues, and more. Particular in open-source, we like to work together and support each other, and GitHub provides much of the social cyberinfrastructure necessary to do so efficiently.

1.3 G4M on GitHub.com

An example of a project using Git and GitHub is the document you’re reading right now! This book is open-sourced and shared at <https://github.com/StevenClontz/github-for-mathematicians>, and was authored completely in a web browser using only the GitHub features we will explore together in this handbook.

¹github.com

²bitbucket.org

³gitlab.com

Chapter 2

Your First Repository

2.1 Making an Account

All the features of GitHub we'll be using are available using a free GitHub account.


Note 2.1.1 Anyone can create a free GitHub account by visiting <https://github.com/signup>.

For additional free resources, you may also be eligible for a waiver.

Note 2.1.2 Many students and faculty of schools, colleges, and universities are eligible to request an educator discount, providing “Pro” features at no cost, by visiting <https://education.github.com/> while logged into their GitHub account.

You do not need to wait for approval of your educator discount before continuing to the next section.

2.2 Creating the Repo

Once logged in, a new repository can be created by pressing the  button in the toolbar, or visiting <https://github.com/new>.

The repository will need a name, which can be something like `my-first-repo` for this tutorial. (GitHub will also suggest a cute random name like `ubiquitous-space-tribble` if you have writer's block.)

Repositories can be **public** to everyone on the internet or **private** to only people you approved. I encourage you to work publicly, to make it easier to collaborate with the open-source community – I can personally attest to publishing many garbage repositories on GitHub (along with my hopefully-useful ones), and no one has called me out for it yet!

The last option we'll make sure to select is to “Initialize this repository with: Add a README file”. Then click “Create repository”.

2.3 Editing README.md

While logged into [GitHub.com](https://github.com), you have the ability to edit individual files on your repositories. (If your repository is public, others can see those files, but cannot edit them unless you make them a collaborator, see [Chapter 5](#).)

An easy way to edit an individual file is just to click the pencil icon such as the one that appears on your README. This file is written in **Markdown**, a markup language that takes plain text like `*this*` and renders it “like *this*”.

Try to edit your file to say something like “I’m learning how to use GitHub!”, perhaps adding a link back to this document using [this markup](https://g4m.clontz.org). You can click the Preview tab to see what your README will look like, and visit sites like <https://www.MarkdownTutorial.com> or <https://www.MarkdownGuide.org> to learn more. GitHub also provides a panel of several formatting options you can click on.

When you’re happy with your updated README, click the “Commit changes” button. This will create a new **commit**, representing a new moment in your project’s history. You should write a useful commit message summarizing the work you’ve done since your last commit (or perhaps keep the default “Update README.md”) Doing this will update the README visible on your repository homepage on GitHub.com.

Finally, you might be interested in visiting the “Insights” tab for your repository, and specifically the “Network” page. It should reveal a graphic similar to [Figure 1.1.1](#) visualizing the history of you project across all GitHub collaborators. Right now you don’t have any collaborators and just a couple commits, but keeping in mind this model for your project history will be useful as we juggle various commits and pushes and syncs and so on down the line.

2.4 Using GitHub.dev


Using the GitHub.com interface to author or edit just one file can be useful (I do this all the time to make quick typo fixes on my blog), but you will likely be using GitHub to manage projects that involve editing multiple files at the same time, and likely you will have non-text files (such as images) that you need to include in your work as well.

One way to quickly be able to manage several files at once is to use the [GitHub.dev](#)¹ service offered by GitHub. Try clicking that link - you should have a fully-functional VS Code text editor right inside your web browser.

Note 2.4.1 It’s best to use an updated version of Chrome, Edge, or Firefox when using GitHub. In particular, Safari tends to show off its rough edges when using advanced web applications like GitHub.dev, so it’s best to choose an alternative.

You can create files, edit them, upload images, and do whatever you like at GitHub.dev. But this isn’t your repository - it’s just an example. So, we’ll need a way to tell GitHub.dev we want to work on the repository we just made instead.

Note 2.4.2 There are two very easy ways to access the GitHub.dev service. The first is to just change the address of your repository from GitHub.com to GitHub.dev in your browser. For example, if your repository lives at <https://github.com/YourUserName/YourGreatRepo>, you should visit <https://github.dev/YourUserName/YourGreatRepo>.

The other trick is even fancier. When you are visiting <https://github.com/YourUserName/YourGreatRepo> in your web browser and not writing in a text box, press the period () key.

Either way, you should now have a GitHub.dev window where you can manage all the files of your project. Using the **Explorer** sidebar ([Figure 2.4.3](#)), you can create new files, rename files, move files, upload files, and more. Selecting a file opens it, and lets you edit it as needed. Your changes are saved automatically in GitHub.dev, but they won’t show up at GitHub.com just yet.

¹github.dev

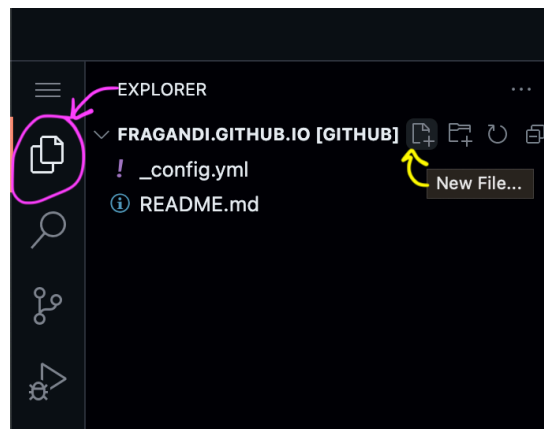


Figure 2.4.3 Explorer sidebar and New File button

2.5 Committing Your Work

After you’ve tried creating/editing/uploading a few files, now it’s time to *commit* those changes to your repository. The easiest way to do this is to use the **Source Control** sidebar. You may have noticed that a numerical badge appeared by the Source Control icon as you created, edited, or deleted files. This number represents the number of files that have been changed in some way since the previous commit. By opening the Source Control panel, you’ll see a list of these files.

Clicking these file names not only lets you open the file and edit it further, but you are shown a **diff** - a summary of the lines that have been altered since the previous commit. (This is a good reason to not write in a long continuous line, but to break lines every 80ish characters or so. That way you can easily see where exactly a change is made between each commit.)

The idea is this: edit as you see fit, knowing that your files are being saved at `GitHub.dev` and won’t be lost if you accidentally refresh your web browser. However, you’ll need to eventually commit those changes to the repository in order to share your work with anyone else, and to ensure that the work is preserved in the long term. The Source Control panel provides a place to write a **commit message**, a short phrase or sentence that summarizes the work you’ve done. (Writer’s block? For now just type “learning `GitHub.dev`”.) Then once you click the “Commit and Push” button, your work will be logged as a permanent commit to the repository.

This is a good point to review your commit history again. You probably have three commits: the initial commit made when you created the repository, the `README.md` update you made using `GitHub.com`’s editing interface, and this more elaborate `GitHub.dev` commit involving possibly several files. To visualize this history, you can go to the Insights/Network page described earlier, or click on the “3 commits” link from your `GitHub.com` repository homepage to see a linearization of this history. From there you can click on each commit to see exactly what has changed from the previous commit across all files.

Chapter 3

GitHub Pages

3.1 Creating a Simple Webpage

Having made your first repository in [Chapter 2](#), and committed a few changes, you are now ready to share your work with the public. One way is to share a link to your repository at [GitHub.com](#); as long as you made it a public repository, anyone can see your files.

Another option is to use **GitHub Pages** to host a customizable website with your work. This can be done with any existing repository by manually authoring HTML files, so let's try it out with our existing example first. (Or, you can skip ahead to [Section 3.2](#) to create a portfolio website without using HTML.)

Use `GitHub.dev` ([Note 2.4.2](#)) to create three files in the root of your project.

- Create a file named `.nojekyll` (note the period at the start) but don't add any text to it. Simply creating this file will disable some advanced features of GitHub Pages we don't need right now.
- Create an `index.html` file. This book won't discuss in depth how to author HTML, as we'll learn how to author our website content in Markdown in the next section, but for now add the following content:

```
<!-- index.html -->
<html>
  <head>
    <title>My website!</title>
  </head>
  <body>
    <p>Thanks for visiting!</p>
    
  </body>
</html>
```

- Download [git-branches.png](#)¹ (used for [Figure 1.1.1](#)) and upload it to `GitHub.dev`.

You can alternatively use this [ZIP file](#)² which has all three files created for you (be sure to unzip it first!).

Commit this update to your repository using Source Control, and confirm you see the two new files on your [GitHub.com](#) repository webpage.

¹<https://g4m.clontz.org/external/git-branches.png>

²<https://g4m.clontz.org/external/website-example.zip>

Note 3.1.1 To enable GitHub Pages, go to your repository Settings, and choose Pages from the sidebar. From there you can select to “Deploy from a branch”, using the `main` branch and the `/ (root)` directory, and after a few moments your site will be enabled.

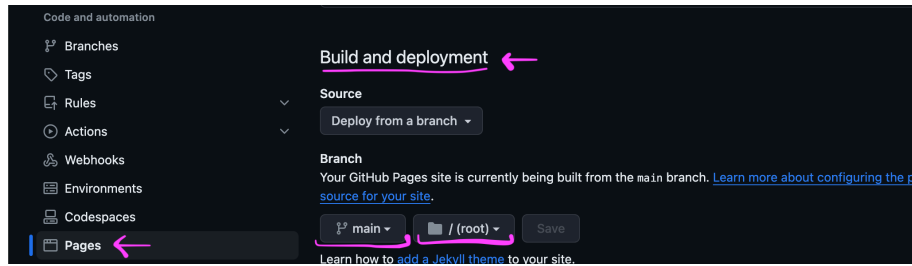


Figure 3.1.2 Settings to deploy a web page.

Once enabled, GitHub will provide a link to your public GitHub Pages website, hosted at `GitHub.io`. Click it and you’ll see the content of your `index.html` file, which displays the image downloaded as `git-branches.png`. (It should look like [this](#)³.)

Note 3.1.3 It’s good to remember how to distinguish the three GitHub domains:

- `GitHub.com` is where your repository lives. It can be public or private.
- `GitHub.dev` is where you can make changes to your repository through your web browser. This is private to you and you must commit and push your changes to the `GitHub.com` repository every so often. (See also [Section 4.3](#).)
- `GitHub.io` is your public GitHub Pages website, which you can edit by updating your repository files.

I recommend you add a link to your `GitHub.io` website from your `GitHub.com` repository page.

Note 3.1.4 On your repository page, you can edit the “About” sidebar to add useful information about your project. In particular, there’s a checkbox to automatically display your `GitHub.io` link to make it easy for others (and yourself!) to find your GitHub Pages site.

3.2 Using a Template

While you can create a custom website by authoring HTML, it’d be great to not have to! There are several “templates” available for GitHub Pages that allow you to author your content in Markdown, as well as providing nice themes, automatic linking between different sections of your website, and so on.

Definition 3.2.1 A **template** repository on GitHub provides other GitHub users the ability to easily obtain a shallow copy of the latest commit to the template, created as a new repository they control. ◇

This is meant for situations like a GitHub Pages website, where you probably don’t care about every single change that was made to create the template you’re using, and you don’t plan on contributing any of your changes back to the original repository. Instead, you just want the latest working files so you can insert your own content and get it online.

³<https://g4m.clontz.org/external/website-example.html>

Visit [this page](#)¹ and click “Use this template”, and “Create a new repository”. This creates a new repository you own on GitHub.com, and you can follow the instructions in [Note 3.1.1](#) to enable GitHub Pages.

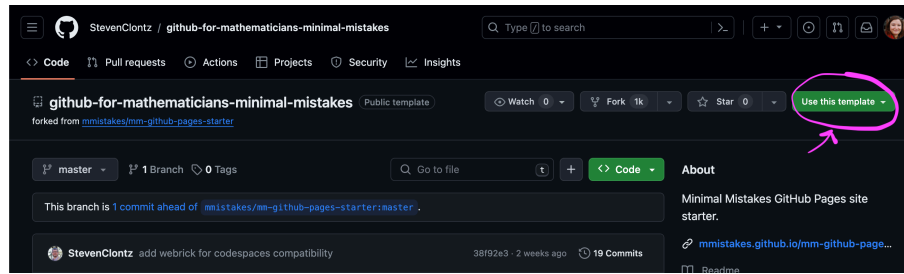


Figure 3.2.2 Use this template button.

Once that’s done, visit your new GitHub.io website to see the placeholder content of your new website (don’t forget to add a link to your “About” sidebar, see [Note 3.1.4](#)).

Note 3.2.3 Deploying to GitHub Pages can take some time, so visiting the “Actions” tab on your repository page will let you see how this process is progressing. You can also see the status of this process by looking for the following icon next to your commit message: an orange dot (in progress), a green checkmark (deployed), or a red X (failure).

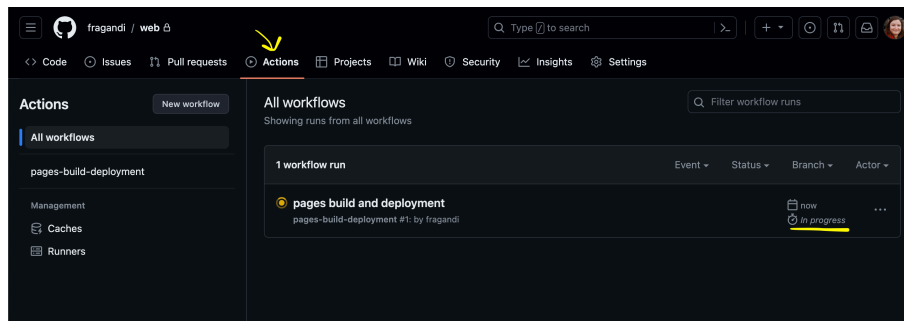


Figure 3.2.4 Actions tab on Github.com.

3.3 Customizing Your Site

Now that you have the template website hosted by GitHub Pages, you of course will want to customize it to yourself. For this book, I’ll get you started by handling a few of the obvious first steps, assuming you’re using the GitHub.dev service ([Note 2.4.2](#)).

3.3.1 Configuration

First things first, let’s configure some basic elements of your site. These settings are found in `/_config.yml`. There are several pieces of this file you likely aren’t interested in editing (nor do you need to know at this point what they do), but you should at least find the `title:` and `description:` lines and edit them with your own information. The same goes for the `author: name:` and `author: bio:` entries as well.

¹github.com/StevenClontz/github-for-mathematicians-minimal-mistakes

To see that this worked, use Source Control to Commit and Push your edits. After a while (Note 3.2.3) you should be able to refresh your website and see your updated title, name, etc. (In Section 4.3, we will learn how to preview our edits more quickly, and without needing to push them to a live website, but at the expense of a more complicated editing environment.) You can repeat this process after each of the edits described below to see your results reflected on the live website.

3.3.2 Photo

Next, let's add your photo. A placeholder is available at `/assets/images/bio-photo.jpg`. You can drag your own JPG-format photo onto it in the File Explorer. Then you can delete the placeholder `bio-photo.jpg` and rename your photo to `bio-photo.jpg`.

3.3.3 Pages

By default you have five files in your `/_pages/` directory. The `404.md` file describes what visitors see when a page isn't found, and the three `*-archive.md` files can be used to customize pages that display certain blog posts.

The `about.md` file describes the content of your About page. The top few lines (Listing 3.3.1) describe some metadata about the page. You can edit the permalink to change the web address that will be used for this page, and the title to change the title shown in the browser tab for this page.

```
---
permalink: /about/
title: "About"
---
```

Listing 3.3.1 About page metadata

Below the metadata is Markdown source that can be edited to include whatever content you'd like to appear within the page.

To create additional pages, copy-paste `about.md` to create new files in the `/_pages/` directory, making sure to assign each page its own permalink. If you want these pages to appear in the navigation bar on top of your site, edit the `/_data/navigation.yml` configuration file to point to each permalink.

You can preview a compiled version of your markdown files (`.md`) on the web, without having to run any commands. It is enough to open a markdown file and split the screen.

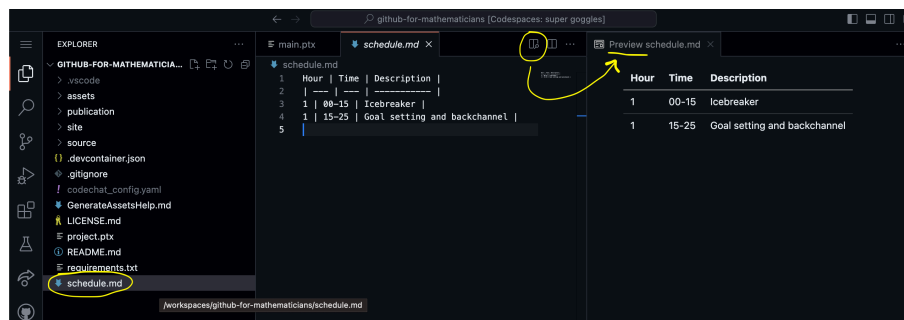


Figure 3.3.2 Split screen to preview markdown file.

3.3.4 Posts

Posts are similar to pages, and live in the `/_posts/` directory. To create a new post, copy-paste any of the existing post files and rename it into the form `YYYY-MM-DD-my-new-post.md` (where `YYYY-MM-DD` is the date you want associated with the post).

The content of your post is just Markdown, as with pages. However, you have slightly different metadata to edit ([Listing 3.3.3](#)). In the `date` you can set the specific time of day you want your post to be associated with. You can also choose to assign each post `categories` and `tags`, which allow your posts to be sorted into appropriate category and tag pages, which are generated automatically for you.

```
---
title: "Welcome to Jekyll!"
date: 2019-04-18T15:34:30-04:00
categories:
  - blog
tags:
  - Jekyll
  - update
---
```

Listing 3.3.3 Post metadata

Chapter 4

Writing and Running Code

4.1 Codespaces

While the quick `GitHub.dev` interface is great for quick edits, many projects require the ability to run applications and execute code as you would on your personal computer. Fortunately, GitHub offers a service to run such programs on their servers.

Definition 4.1.1 A **Codespace** is a “dev environment” service offered by [GitHub](#)¹. Each codespace is essentially a personal virtual computer that runs in the cloud, that you access through your web browser.

Full documentation on Codespaces is available on [docs.github.com](#)². ◇

Note 4.1.2 To create a Codespace on any repository you own, use the green “Code” button, select the Codespaces tab, and press the green “Create codespace on [branchname]” button.

After the Codespace boots up, you’ll have an interface similar to the `GitHub.dev` environment you learned about in [Section 2.4](#).

Note 4.1.3 Here are a few key differences between `GitHub.dev` and Codespaces.

1. A `GitHub.dev` URL looks like `github.dev/username/reponame`, while a Codespace URL looks like `random-word-123randomcharacters789.github.dev`.
2. `GitHub.dev` is quicker to load than a Codespace.
3. `GitHub.dev` has a much more limited selection of VS Code extensions to use.
4. You can only install applications and execute code on a Codespace.

One similarity between `GitHub.dev` and Codespaces (besides the obviously similar VS Code user interfaces), is that your work is still private to you and can only be shared with the public (and retained in the long term) by committing and pushing your progress every so often to your `GitHub.com` repository. The Source Control tool works slightly different in a Codespace, however.

Note 4.1.4 One quick way to commit and push your changes from a Codespace is to use “Source Control” from the left toolbar.

- Enter a short commit message describing your changes as a note to your-

¹[github.com/features/codespaces](#)

²[docs.github.com/en/codespaces](#)

self. (This is required and can be a pain to fix if you forget to do so!)

- Select “Commit & Sync” from the menu next to the green Commit button.
- In the dialogs that follow, I suggest choosing to “Always” stage all your changes and commit them directly, then “OK, Don’t Show Again” when told this action will pull and push commits from and to `origin/main`, and finally “Yes” that you would like your Codespace to periodically run `git fetch`.

4.2 Python and Jupyter Notebooks

[Python](#)¹ is an popular open-source all-purpose programming language, and a convenient way to write, execute, and share the results of Python code is a [Jupyter notebook](#)².

To get started, create a Codespace ([Note 4.1.2](#)) on either an existing or new repository ([Section 2.2](#)). You can then create a Jupyter notebook file named `notebook.ipynb`.

Note 4.2.1 In a Codespace, any file with the extension `*.ipynb` (short for “*IP*ython *NO*te*B*ook”, Jupyter’s original name) will be treated as a Jupyter notebook. When opening this file, you’ll see a notebook interface, and be prompted to “install the recommended ‘Python’ extension” if it’s not already enabled - do this.

Then in your notebook file, click the “Select Kernel” button, then “Install/Enable suggested extensions” for Python+Jupyter. You should then have the option to select a “Python environment” such as `Python 3.*.*`.

If successful, you should be able to enter `import sys; print(sys.version)` into the displayed text box, and see the result of executing it with `Shift+Enter`.

There are plenty of existing tutorials on the internet to help you get acquainted with Python and Jupyter now that you have them available to you in your Codespace. But to get you started, I’ve provided one [sample notebook](#)³ that you can upload to your Codespace to break the ice.

4.3 Previewing GitHub Pages

Another useful application of Codespaces is the ability to preview your GitHub Pages site created in [Section 3.2](#). Return to that repository on [GitHub.com](#) and create a new Codespace ([Note 4.1.2](#)).

To spin up your live preview, open a terminal by using the `Ctrl/Cmd+Shift+`` keys. To make sure the necessary software has been installed, type `bundle` and hit `Enter`. Then, you can enter `ekyll serve` to start the preview server.

You’ll see some output, and eventually `Server address: http://127.0.0.1:4000`.

At that time an alert will appear that says “Your application on port 4000 is available”. You can use its “Open in browser” button, or hover over the `http://127.0.0.1:4000` link to be given the same option.

This should open a URL such as `random-words-123randomletters789-4000.app.github.dev`, which will show a live preview of your GitHub Pages site in a new tab. As soon as you make edits in your Codespace tab, you can return to this tab to

¹[python.org](#)

²[jupyter.org/](#)

³<https://g4m.clontz.org/external/sample-notebook.ipynb>

(within a second or two) see how your edits will update your live site. Note that this URL is private to you, and your public site won't be updated until you Commit & Sync your changes (Note 4.1.4).

Personally, I use `GitHub.dev` (or even just the `GitHub.com` edit button) rather than a full Codespace when adding a quick post or making a quick edit on many of my GitHub Pages websites. But the Codespace option is very handy for when bigger changes are necessary, and you want to make sure everything looks just right before pushing it live to the public.

4.4 Managing Your Codespaces

GitHub users are provided with a limited amount of free Codespace hours and storage each month, with additional resources available to Pro users, including those with the free GitHub Education benefit (Note 2.1.2). If needed, there is the option to pay for additional resources.

As a mathematician who almost exclusively uses GitHub Codespaces for doing the kinds of work described in this handbook, I've found that I often approach the quota provided for Pro users, but have not yet exceeded it. I accomplish this by being sure to not run my Codespaces when I'm not actively working on them.

Note 4.4.1 To manage your Codespace resources, visit <https://github.com/codespaces>. You can stop a Codespace temporarily to preserve your hourly quota, and delete a Codespace you don't plan to use for a while to save on your storage quota. While actively working in a Codespace, you can stop it by pressing `[Ctrl/Cmd]+[Shift]+[P]`, typing `stop current codespace`, and confirming.

In any case, a stopped Codespace can be restarted later when you want to resume work, even if you haven't committed and pushed your changes. (But be warned: a stopped Codespace and its uncommitted changes may be deleted by GitHub after a few days of inactivity, so don't leave it alone for long.)

A deleted Codespace can always be recreated later based upon your most recent commit.

Putting it all together, we have seen three ways to access files on your repo. Going from the least easy to edit to the most editable we have: `GitHub.com`, `GitHub.dev`, `random-codespace-string.github.dev`. Here is what these environments look like.

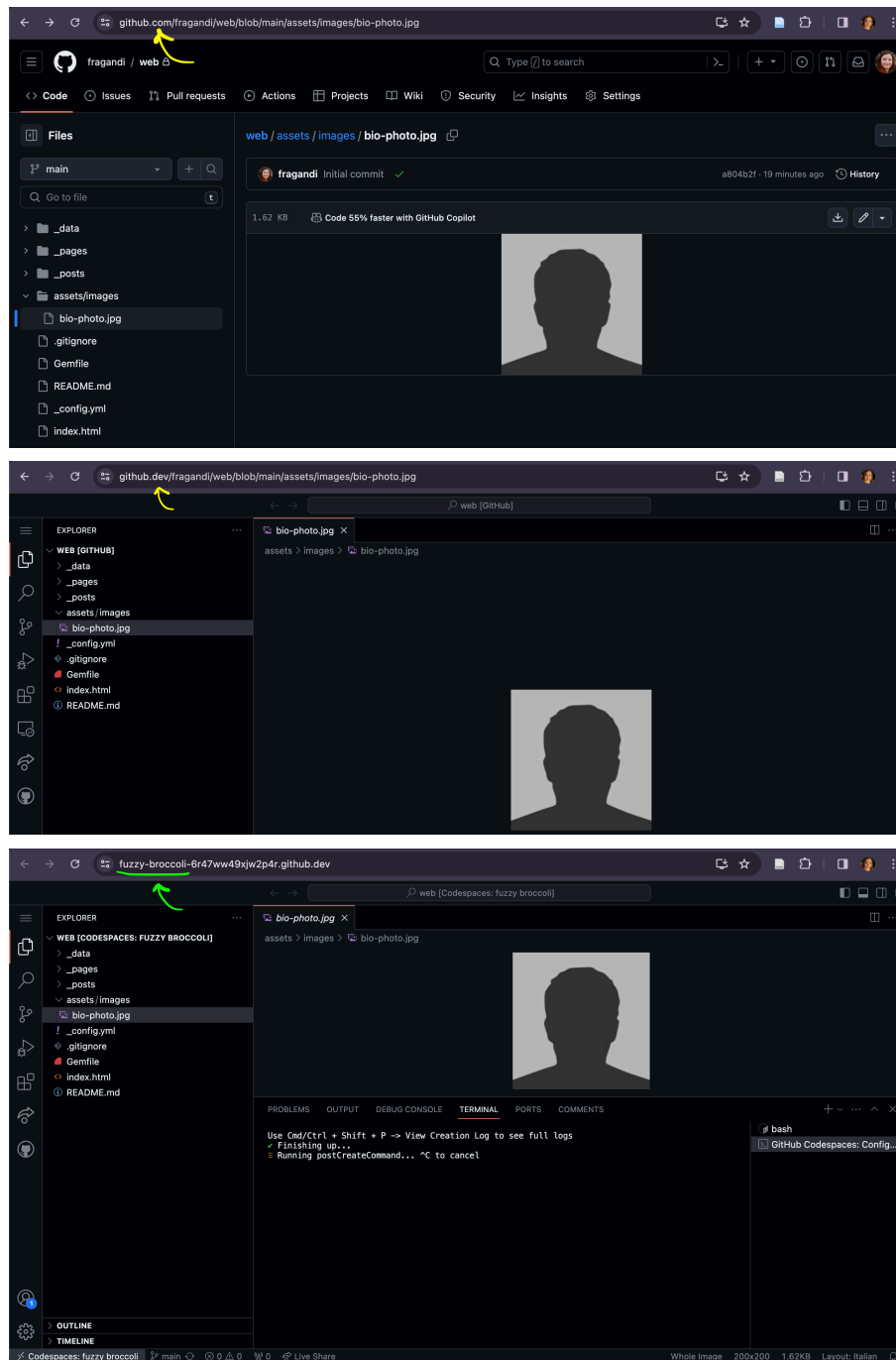


Figure 4.4.2 Repo from different points of view.

Chapter 5

Collaborating with Others

5.1 Live Share

A nice feature of collaborative authoring tools such as Google Docs and Overleaf is the ability for several collaborators to edit the same file synchronously, with a feature set similar to `GitHub.dev`.

Using the Extensions sidebar, search for and install Microsoft’s “Live Share” extension. Once installed, you will have a “Live Share” option in your bottom toolbar. Clicking it will automatically copy a URL ending in `https://.../join?[randomStringOfCharacters]`. If you share this URL with a colleague (or colleagues) you trust, they will be able to collaboratively edit your repository’s text files with you via their web browser. When you commit your changes, they will be listed as co-authors of the commit.

5.2 Collaborators and Pull Requests

A particular downside of Live Share ([Section 5.1](#)) is that it requires the repository owner to create the Codespace and provision the Live Share session. As a result, there’s no way for a collaborator to make a contribution via Live Share to a GitHub project without the active involvement of the repository owner.

To address this, one solution is for the repository owner to add other GitHub users as collaborators.

Definition 5.2.1 A **collaborator** for a GitHub repository has the ability to commit and sync changes to the project, as well as adjust certain settings of the repository. [GitHub documentation](#)¹ provides some details on the different permissions/abilities that owners have in comparison to collaborators. ◇

Collaborators are added by going to your repository’s Settings tab, using the Collaborators link in the sidebar. Each collaborator will need their own GitHub account, and must accept the invitation to collaborate before gaining access.

Once they have access, a collaborator can either use `GitHub.dev` ([Note 2.4.2](#)) or create their own Codespace ([Definition 4.1.1](#)).

Warning 5.2.2 If two collaborators on the same repository make commits on the same branch, they will desynchronize your project’s history: person A’s history will think that commit X is followed by Y on branch `main`, but person

¹docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-user-account-settings/permission-levels-for-a-personal-account-repository

B’s history will think that commit X is followed by Z on branch `main`.

As seen in [Figure 1.1.1](#), Git is meant to support non-linear history. However, to support this, contributors must name their distinct branches.

Note 5.2.3 One workflow to support multiple collaborators on the same repository is to *never directly commit to the main branch*, even if you’re the owner.

To commit to an alternative branch in `GitHub.dev` or `Codespaces`, select `main` in the bottom toolbar, then type the name of your new branch, and select “Create new branch”. The name of this branch could be topical, e.g. `add-derivative-chapter`, but it’s also fine to pick some other unique identifier, e.g. `lastname-YYYY-MM-DD`.

Once a collaborator is working on a branch, they are free to edit as they wish, and can (and should) commit and push/sync with `GitHub` to persist their contributions to the team’s repository.

To facilitate communication among collaborators working on different branches, it’s good practice to open a draft pull request once a new branch is created.

Definition 5.2.4 A **pull request** (or **PR** for short) is a discussion thread for a branch that proposes changes to a different (often, the `main`) branch. When the branch’s changes are ready to be merged, this can be accomplished by pressing a button on the pull request webpage.

A PR can be marked as a **draft** or **ready to review**. ◇

Note 5.2.5 Depending on whether the collaborator is using `GitHub.dev` or `Codespaces`, they may be prompted to create a pull request when first pushing/syncing changes. If not, a pull request can be created by navigating to the repository page on `GitHub.com`.

Recent pushes to a branch will reveal a prompt to create the pull request immediately. Otherwise, the PR can be created by using the Pull Requests tab of the page.

Unless the PR is for a single commit that’s immediately “ready for review”, a new PR should be created as a draft.

With a draft pull request created, the contributor can continue to commit and push/sync to the branch until it is ready for review. The discussion features of `GitHub` can allow contributors to discuss the proposed changes, whether they are in draft or review-ready status.

Note 5.2.6 A draft pull request has a large button near the end of the discussion thread to mark the pull request as ready for review.

A ready for review pull request can be converted to a draft by using a small link on the right sidebar.

Depending on the complexity or maturity of the project, you may wish to develop a review process with your collaborators, or simply use the PR workflow to clearly communicate when changes are being made to the main branch, and ensure no two collaborators make incompatible changes to the same branch. Whatever you choose, you’ll eventually want to incorporate these branched changes into your `main` branch.

Note 5.2.7 There are several options for merging a branch’s pull request. I recommend the “Squash and Merge” option, which converts all the branch’s commits/changes into a single new commit extending the target branch.

As long as contributions are made using appropriate branches and pull requests, you will have minimal problems with conflicting changes made between different collaborators, with `GitHub` handling the merging process automatically, even if two collaborators edit the same file. (But not always, see [Section 5.4](#).)

But a common error that I frequently make myself: what if you forget to create a branch with your work, and you accidentally commit to `main` directly? The first safety rail I recommend is to set up a policy on your repository that will prevent this accidental commit to be pushed to GitHub.

Note 5.2.8 To prevent unintended changes to your `main` branch, follow the instructions at [GitHub’s documentation](#)², using `main` as your “Branch name pattern”, and enabling required pull requests.

Finally, there’s no button to push that will fix a commit made to the local copy of `main` accidentally, but there is a quick-enough fix nonetheless.

Note 5.2.9 With branch protection [Note 5.2.8](#) enabled, if you accidentally make updates directly to your personal `main` branch, attempting to push these from a Codespace will result in the error message `Can't push refs to remote`. Try running `"Pull" first to integrate your changes`.

To fix this, open the Terminal and type [Listing 5.2.10](#), changing `my-new-branch` to the branch name you want to create. Use `[Enter]` to execute the command.

Then, copy-paste [Listing 5.2.11](#) and your local `main` branch will match the official repository, and any changes you’ve made will be reflected on the `my-new-branch` branch instead.

```
| BRANCH=my-new-branch
```

Listing 5.2.10 Defining the name for a new branch

```
git stash
git branch $BRANCH
git reset --hard origin/main
git checkout $BRANCH
git stash pop
```

Listing 5.2.11 Moving local changes to `main` to a branch

5.3 Forks

One great thing about working with open source on GitHub is that not only can you collaborate with your trusted colleagues, but you can also work with collaborators who do not have write access to your repository.

Definition 5.3.1 A **fork** for a public repository is a copy of the project’s entire history, made either for the `main` branch or for all publicly shared branches.

◇

Managing contributions from forked repositories is done using the same workflow as I recommend for collaborating with trusted colleagues that you’ve given write access to your repository ([Section 5.2](#)). The only difference is that an outside collaborator is creating branches and making commits on their forked copy of your project, not a branch of your original repository. But GitHub still gives essentially the same options for the outside collaborator to create a pull request to your project, without given them access to any data you aren’t already sharing with the public.

Note 5.3.2 To create a fork of a public repository, press the “Fork” button on its GitHub.com homepage. You can name this fork whatever you like, it will be tracked on GitHub as a fork of the original project, with the ability to make

²docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/managing-a-branch-protection-rule#creating-a-branch-protection-rule

“upstream” contributions by way of pull requests.

Those of us who work in open source typically *love* getting pull requests from random collaborators. For example, if you find a typo in this book, you can fix it by creating a fork at <https://github.com/StevenClontz/github-for-mathematicians/fork>, editing the appropriate `source/*.ptx` file to fix the word, and open a pull request.

5.4 Handling Merge Conflicts

To come in a future revision of this handbook...

Chapter 6

Math Projects Powered by GitHub

6.1 PreTeXt authoring system

[PreTeXt](#)¹ is an authoring and publishing system for scholarly documents, especially in STEM disciplines. Works authored in PreTeXt can be converted to HTML, PDF, braille, and many other formats. This book was written in PreTeXt!

PreTeXt is particularly well-suited for the creation of interactive and accessible Open Educational Resources in mathematics and computer science. Works authored in PreTeXt can be deployed to [Runestone Academy](#)², allowing students to log into their textbook and persist progress on exercises and activities.

The [PROSE Consortium](#)³ forms the broader ecosystem serving open-source STEM open educational resources and offers regular drop-in meetings for community members to learn more about its products, which also include [WebWork](#)⁴ and [Doenet](#)⁵.

[Getting Started with PreTeXt](#)⁶ is a tutorial that uses GitHub Codespaces to get authors up and writing quickly, and helps them share their works on GitHub and GitHub Pages.

6.2 π -Base Community Database of Topological Counterexamples

To paraphrase Mary Ellen Rudin, “topology is a dense forest of counterexamples, and a usable map of the forest is a fine thing.”. The π -Base¹ aims to serve as this atlas, turning the classic text [Counterexamples in Topology](#)² into an open, living, and interactive database supported by a community of researchers, instructors, and students.

¹pretextbook.org/

²runestone.academy

³prose.runestone.academy

⁴openwebwork.org/

⁵www.doenet.org/

⁶stevenclontz.github.io/pretext-getting-started-2023

¹topology.pi-base.org/

²zbmath.org/0211.54401

To learn how to contribute to the π -Base, a [tutorial](#)³ is available to walk through the process of editing its data and previewing it in your own Codespace.

6.3 Lean Theorem Prover

The [Lean theorem prover](#)¹ is an interactive proof assistant that allows mathematicians to formally verify their proofs by computer.

The textbook [Mathematics in Lean](#)² provides an excellent introduction to authoring in Lean, with GitHub Codespaces support.

(For a more casual experience outside GitHub, the [Lean game server](#)³ has fun tutorials for both Peano axioms and naive set theory.)

6.4 code4math

In December 2023, the American Institute of Mathematics hosted a workshop of researchers to explore the future of sociotechnical infrastructure for mathematics. One outcome of this work was the establishment of <https://code4math.org>, a community for mathematicians engaged in this work.

³github.com/pi-base/data/wiki/Tutorial

¹leanprover-community.github.io/

²github.com/leanprover-community/mathematics_in_lean?tab=readme-ov-file#to-use-this-repository-with-github-codespaces

³adam.math.hhu.de/

Appendix A

Additional Reading

- [Version Control with Git](#)¹
- [Programming with Python](#)²

¹swcarpentry.github.io/git-novice/

²swcarpentry.github.io/python-novice-inflammation/

Appendix B

Additional Topics

B.1 GitHub Desktop

TODO

B.2 VS Code Application

TODO

B.3 Using the Terminal

TODO

Appendix C

Definitions and Notes Quick Ref

Section 2.1 Making an Account

[Note 2.1.1](#)

[Note 2.1.2](#)

Section 2.4 Using GitHub.dev

[Note 2.4.1](#)

[Note 2.4.2](#)

Section 3.1 Creating a Simple Webpage

[Note 3.1.1](#)

[Note 3.1.3](#)

[Note 3.1.4](#)

Section 3.2 Using a Template

[Definition 3.2.1](#)

[Note 3.2.3](#)

Section 4.1 Codespaces

[Definition 4.1.1](#)

[Note 4.1.2](#)

[Note 4.1.3](#)

[Note 4.1.4](#)

Section 4.2 Python and Jupyter Notebooks

[Note 4.2.1](#)

Section 4.4 Managing Your Codespaces

[Note 4.4.1](#)

(Continued on next page)

Section 5.2 Collaborators and Pull Requests

[Definition 5.2.1](#)

[Note 5.2.3](#)

[Definition 5.2.4](#)

[Note 5.2.5](#)

[Note 5.2.6](#)

[Note 5.2.7](#)

[Note 5.2.8](#)

[Note 5.2.9](#)

Section 5.3 Forks

[Definition 5.3.1](#)

[Note 5.3.2](#)

Colophon

This book was authored in PreTeXt.