What is version control?

Source control, also known as version control or revision control, is a system or process used in software development to manage changes to the source code of a project. Source control helps developers collaborate on code, track changes over time, and maintain a history of code revisions. It provides a structured way to manage codebase, coordinate the work of multiple developers, and safeguard against errors or data loss.

The Collaborative Novel Writing Analogy

Imagine you and a group of friends decide to write a novel together. Each of you will contribute different chapters to create the final masterpiece. However, there's a challenge – you need a way to manage everyone's contributions, keep track of changes, and ensure that the novel maintains a coherent storyline.

This is where source control comes into play, represented by the role of a "Story Control Manager."

1. The Original Manuscript:
    - The story begins with a blank manuscript representing the initial state of your novel.
2. Forking the Manuscript:
    - Each friend creates their own copy of the manuscript. These copies are like branches in source control. Each friend's copy is their "fork."
3. Writing Chapters:
    - Everyone starts writing their chapters independently in their respective copies. These are like code changes in a software project.
4. Merging Chapters:

- When a friend finishes their chapter, they submit it to the Story Control Manager.
- The Manager reviews the chapter, makes sure it fits the story, and merges it into the master manuscript.

5. Conflicts:
   - Sometimes, two friends write chapters that conflict with each other (e.g., contradictory plotlines). This is like a merge conflict in source control.
   - The Manager resolves these conflicts by deciding which version of the chapter to include or finding a compromise.
6. Rollback:
   - If a friend submits a chapter that doesn't work well, the Manager can roll back to a previous version of the manuscript (similar to reverting changes in source control).
7. Version History:
   - The Story Control Manager keeps a log of who contributed which chapters, when they did it, and the changes they made. This is like the version history in source control.
8. Collaboration and Feedback:
   - Throughout the process, friends can review each other's chapters, leave comments, and suggest improvements – just like code reviews in software development.
9. The Final Novel:
   - As the story progresses, the manuscript gradually transforms into a complete novel, and the end result is a well-crafted, collaborative masterpiece.
10. Ongoing Editing:

- Even after the novel is published, you can continue to make revisions or create sequels by extending the same source control principles.

In this analogy, source control is the Story Control Manager who oversees the collaborative novel-writing process, manages changes, resolves conflicts, and maintains a clear history of the story's evolution. Just as source control helps software developers work

together on code, the Story Control Manager ensures that your group of friends can write a cohesive and successful novel together.

Key Aspects of source control

Key aspects of source control include:

Version Tracking: Source control systems keep track of changes made to the codebase. This includes who made the change, when it was made, and what exactly was changed. This history is invaluable for understanding the evolution of the code and for troubleshooting issues.

Collaboration: Developers can work on the same codebase concurrently without interfering with each other's changes. Source control systems facilitate merging different branches of code and resolving conflicts when they occur.

Backup and Recovery: Source control acts as a backup system for code. If something goes wrong, such as code corruption or accidental deletion, previous versions can be restored from the source control history.

Branching and Forking: Developers can create branches or forks of the codebase to work on new features or experiment without affecting the main codebase. These branches can be merged back into the main codebase when ready.

Code Review: Source control systems often integrate with code review tools, making it easier for developers to review each other's code and provide feedback.

Documentation: Along with code changes, developers can add comments and documentation to explain why specific changes were made, making it easier for others to understand the code.

Common source control systems include Git, Subversion (SVN), Mercurial, and Perforce. Git, in particular, has gained widespread popularity due to its distributed nature and efficiency in handling branching and merging.

Source control is considered a fundamental best practice in software development, enabling teams to work more efficiently, reduce errors, and maintain a clean and organized codebase.

What is github?

GitHub is a web-based platform and service that provides a range of tools and features for version control, collaboration, and software development. It is widely used by developers and teams for hosting, sharing, and collaborating on code repositories. Here are some key aspects of GitHub:

Version Control: GitHub is built around the Git version control system, which allows developers to track changes in their codebase over time. Git enables users to create branches, make changes, and merge those changes back into the main codebase. GitHub provides a user-friendly interface for managing Git repositories.

Code Hosting: Developers can host their Git repositories on GitHub for free. This makes it easy to share code with others, whether they are collaborators, open-source contributors, or the broader community. GitHub provides a secure and reliable platform for storing and accessing code.

Collaboration: GitHub offers powerful collaboration features, including pull requests, code reviews, and issue tracking. Teams can work together seamlessly, review each other's code, and discuss changes within the platform. This facilitates effective teamwork and code quality control.

Public and Private Repositories: GitHub allows developers to create public repositories that are open for anyone to view and contribute to. It also offers private repositories for projects that need to remain confidential or limit access to specific collaborators.

Community and Social Interaction: GitHub is a hub for the global developer community. Developers can follow projects, star repositories they find interesting, and discover open-source software. It encourages social coding and knowledge sharing.

Continuous Integration and Continuous Deployment (CI/CD): GitHub integrates with various CI/CD tools and services, making it easier to automate the build, test, and deployment processes for software projects. This helps ensure code quality and streamlines the development workflow.

GitHub Actions: GitHub Actions is a built-in CI/CD and automation service that allows developers to create custom workflows for their repositories. It can automate tasks such as testing, building, and deploying applications.

Packages and Container Registry: GitHub provides features for hosting and managing software packages, Docker containers, and other artifacts, making it a comprehensive platform for managing dependencies and artifacts.

Documentation and Wikis: Developers can create documentation for their projects using GitHub's built-in wiki system or by integrating with tools like GitHub Pages. This helps maintain project documentation and user guides.

Security and Code Scanning: GitHub offers security features to scan code for vulnerabilities and provides insights into code

quality. It helps developers identify and fix security issues early in the development process.

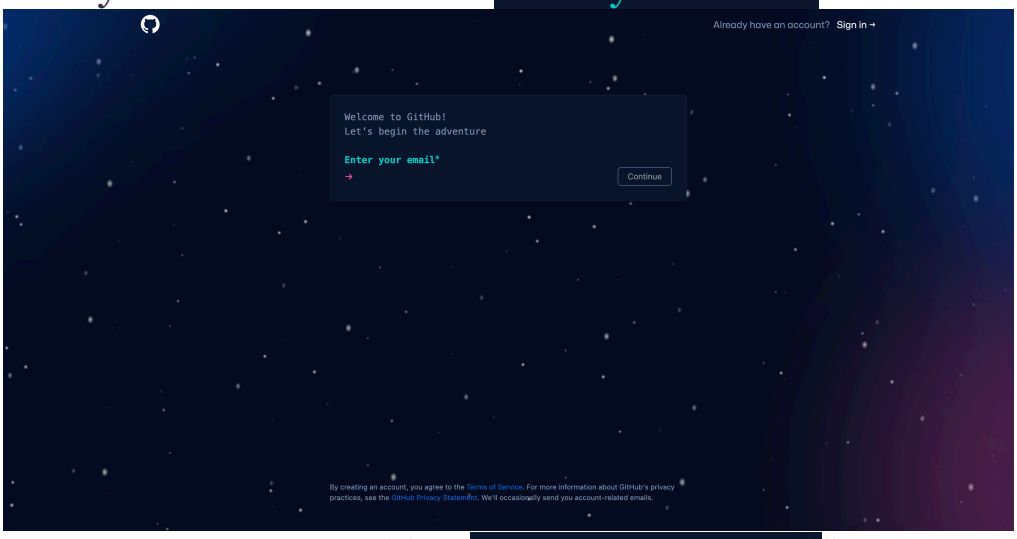
GitHub is an essential tool for individual developers, open-source projects, and organizations of all sizes. It promotes collaboration, code quality, and efficient software development workflows. Many companies and communities use GitHub as a central hub for their software projects and code sharing.

Make a repository on GitHub

1. Create a github Account

- Open a new browser tab
- Navigate to https://github.com/
- Create an account

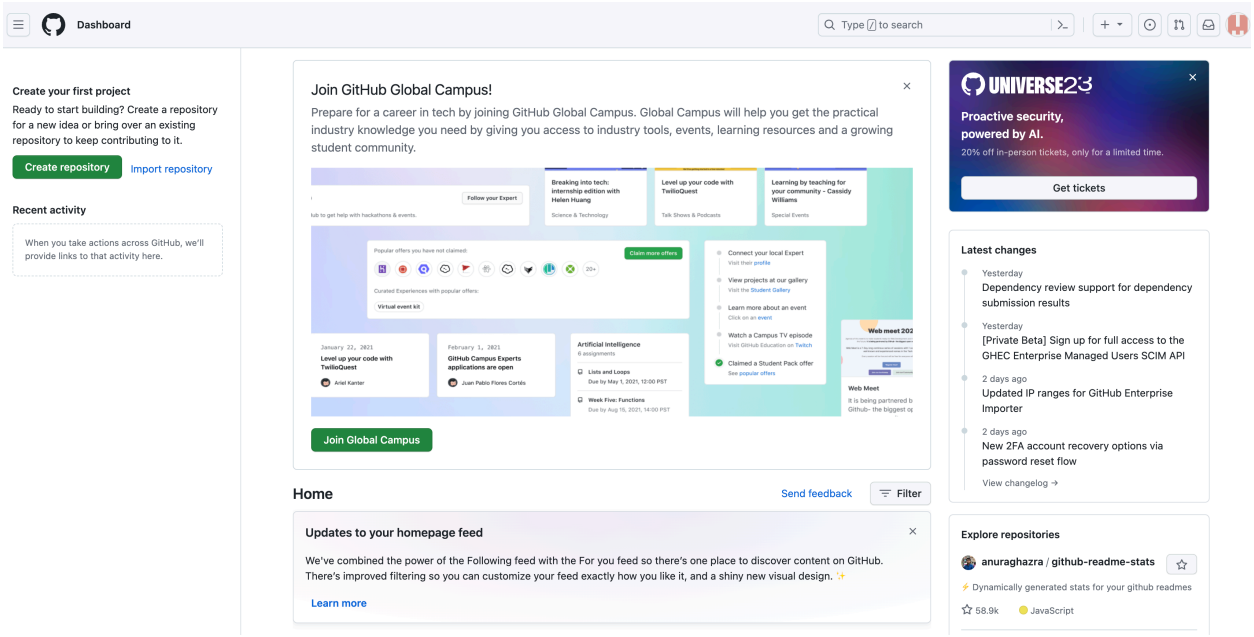
click on ‘Create an Account’ and follow the below steps.
Give your emailID in the “Enter your email*”



Enter your password in “Create a password*”
Enter a username in “Enter a username*”
Provide your choice of “Would you like to receive product updates and announcements via email?”

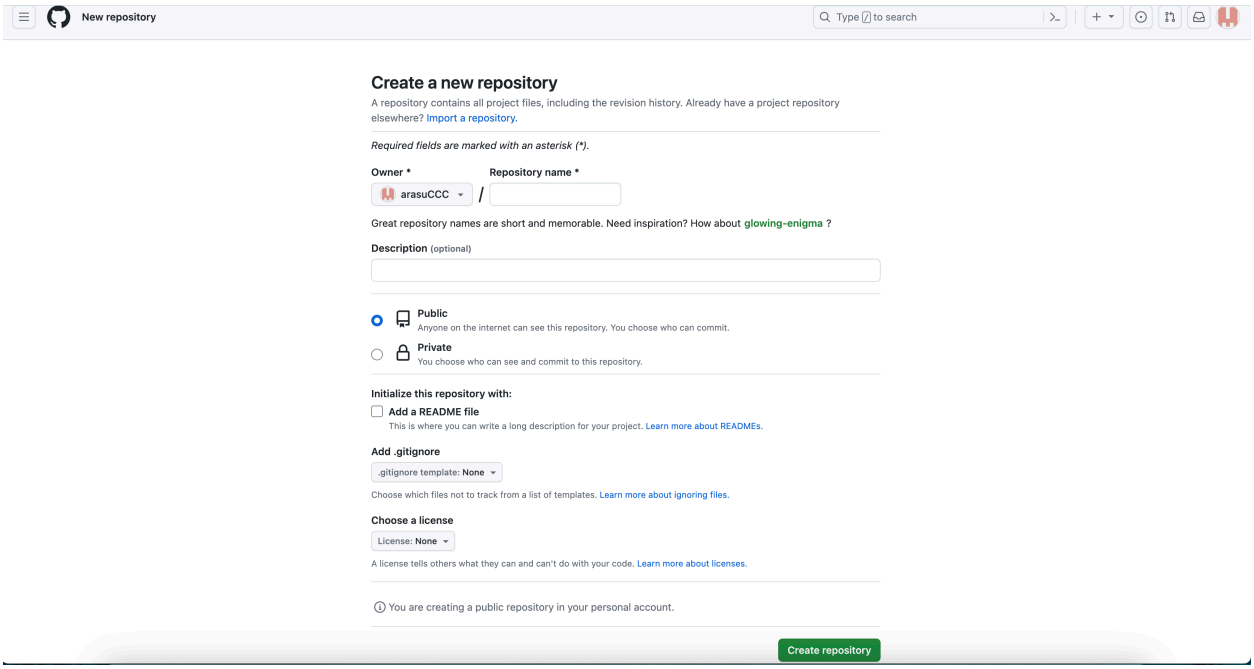
Verify your account through a puzzle and click Continue

Provide the personalization information and click Continue. You will get the below screen



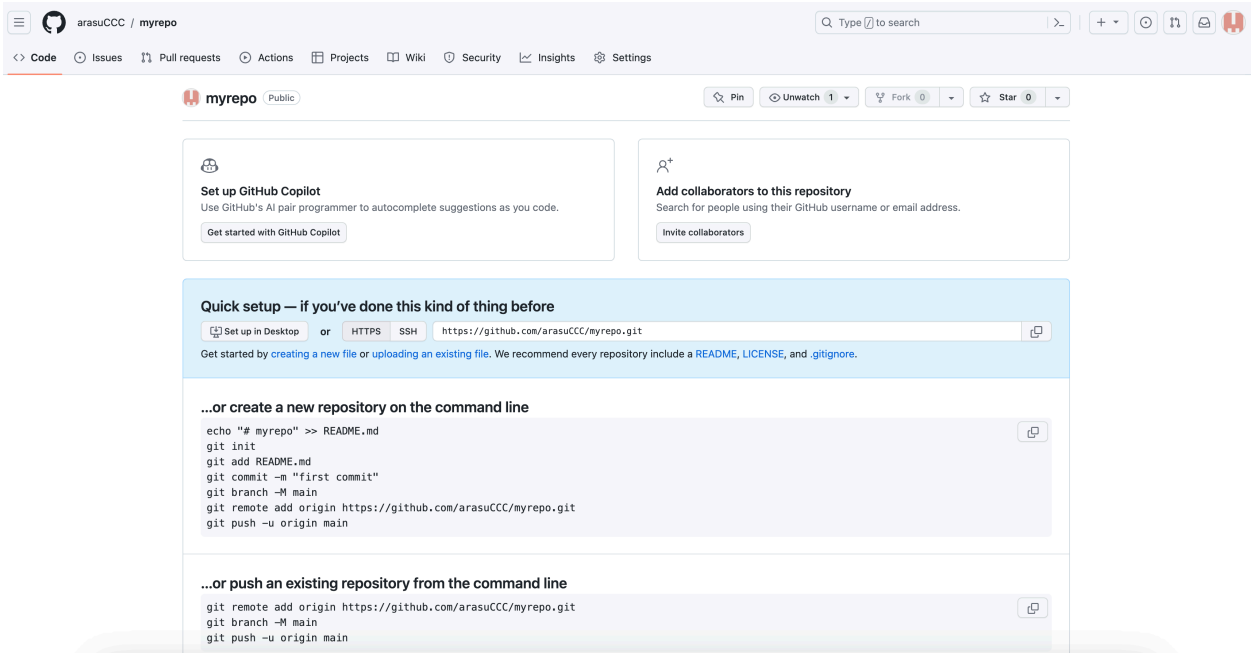
2. Create a GitHub repository.

Click the “Create repository” button, Give your repository a name and a description in the below screen. You can leave the “public” and “private” option as “public”



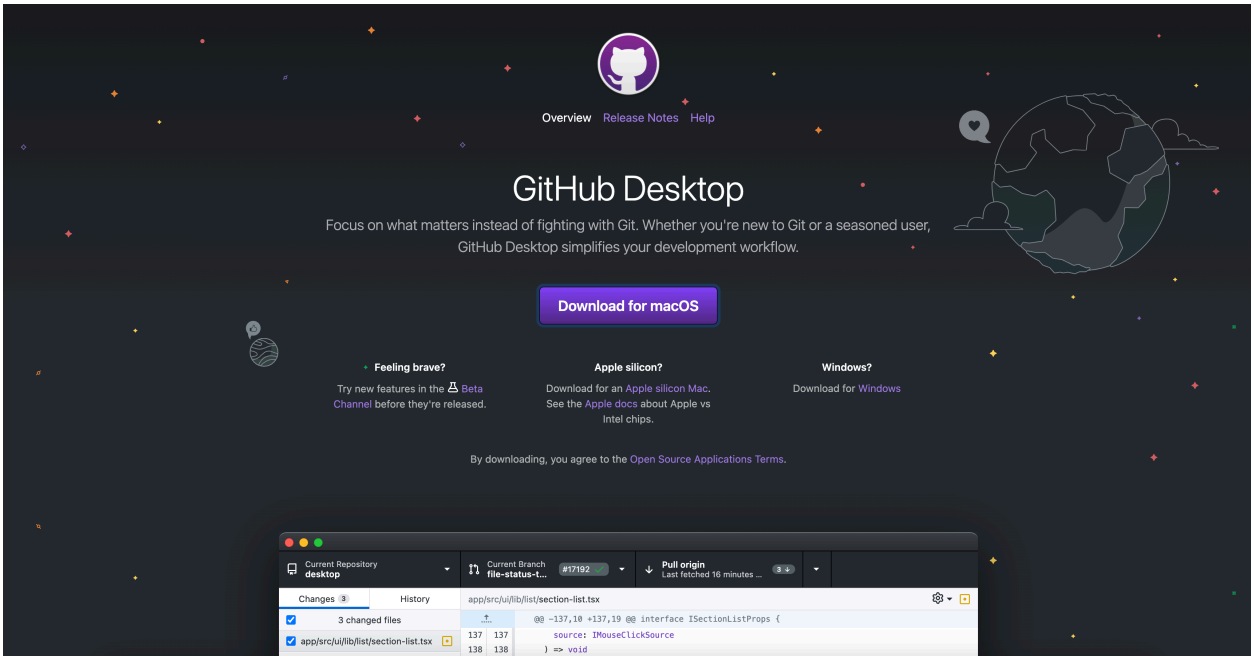
Check the box at Initialize this repository with a README (the readme is a simple text file in which you can put information about the current project)

Once you click “Create Repository” button, you will find the below page.



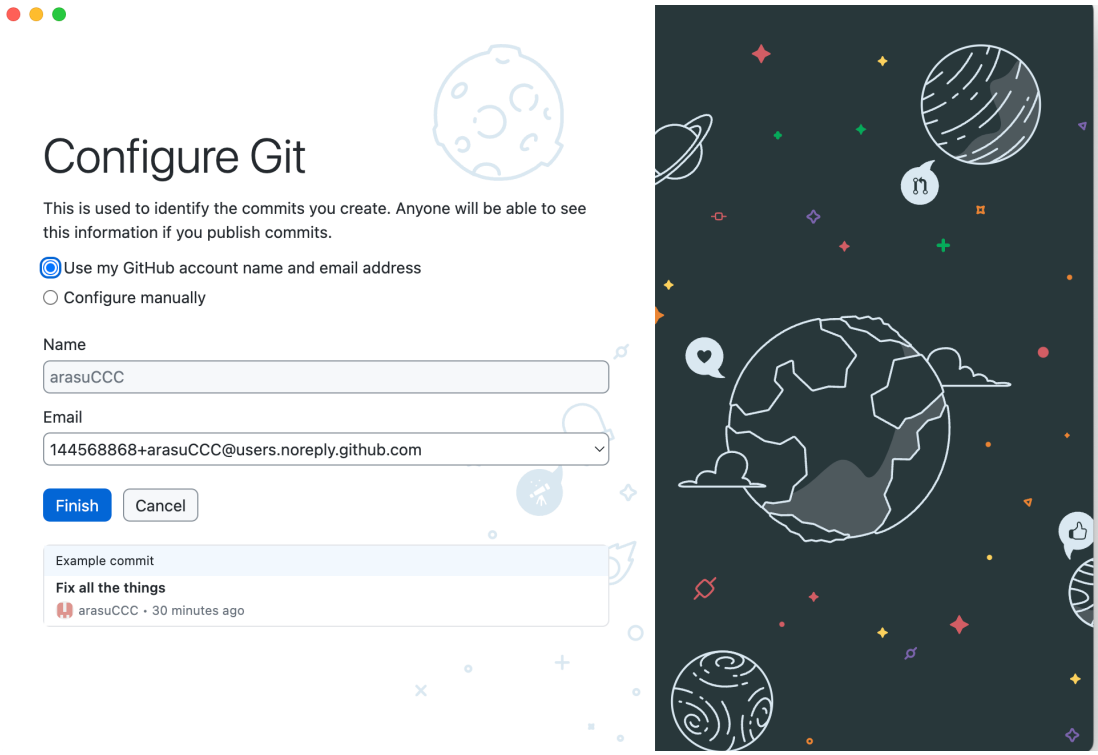
3. Clone the repository to your local machine.

If you would like to work on your code and maintain it in the github, you need to get this repository onto your pc. This is done by cloning it. The easiest way to get this done is through GitHub Desktop. You can go to https://desktop.github.com/ and download the GitHub Desktop.

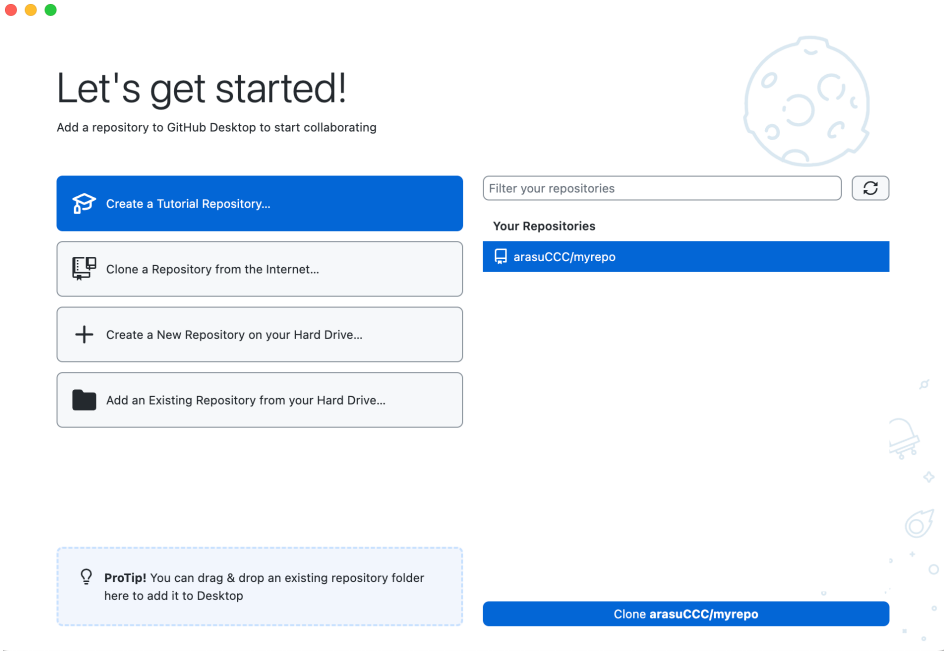


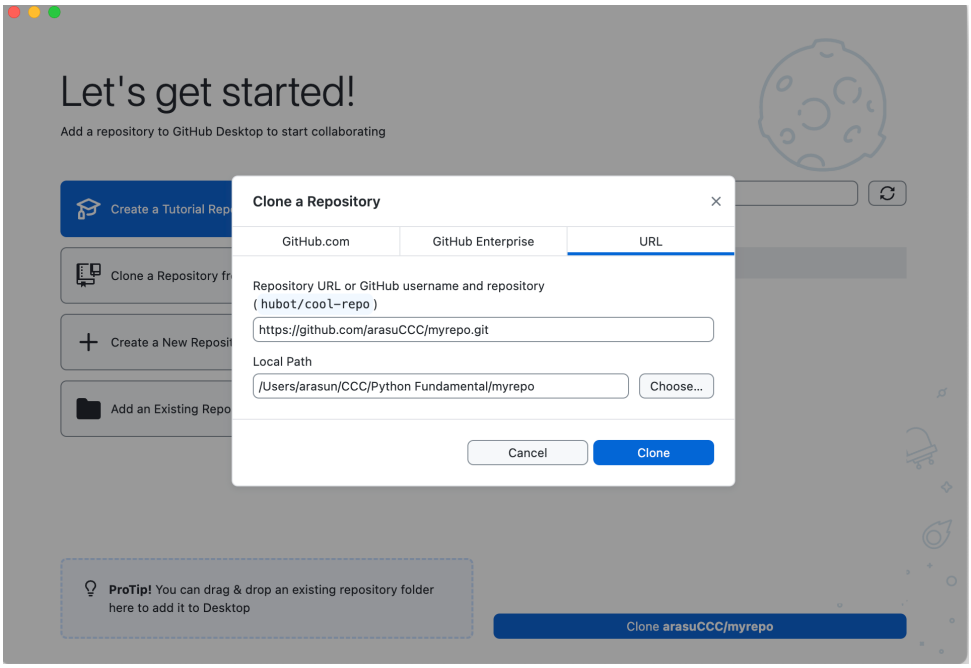
For the mac, you will find a GitHubDesktop-x64.zip in the download folder and extract the zip file in your folder.

Open the GitHub Desktop and provide the login information in the desktop application



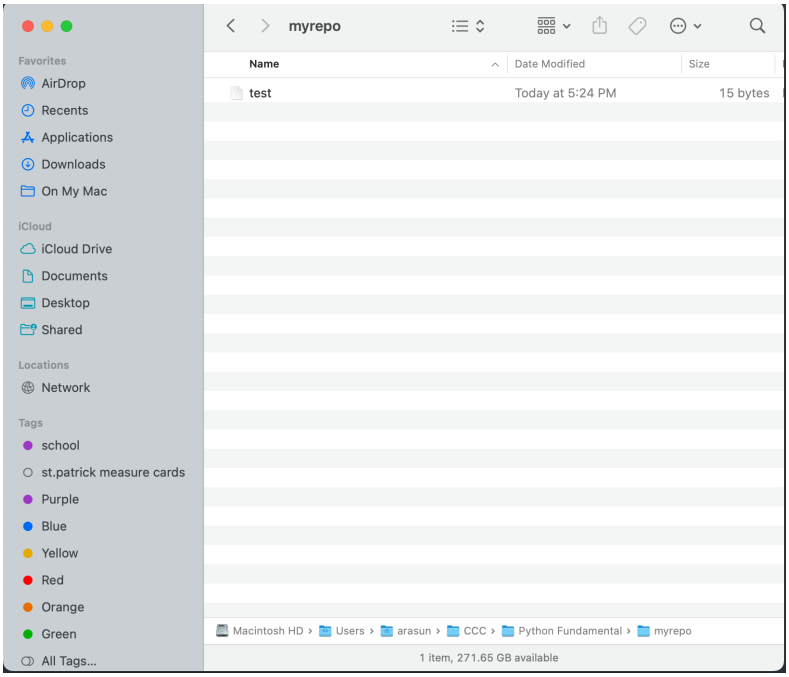
Click the repository and clone it locally, provide the local folder and click “clone” button. You can find the repository in the local folder.





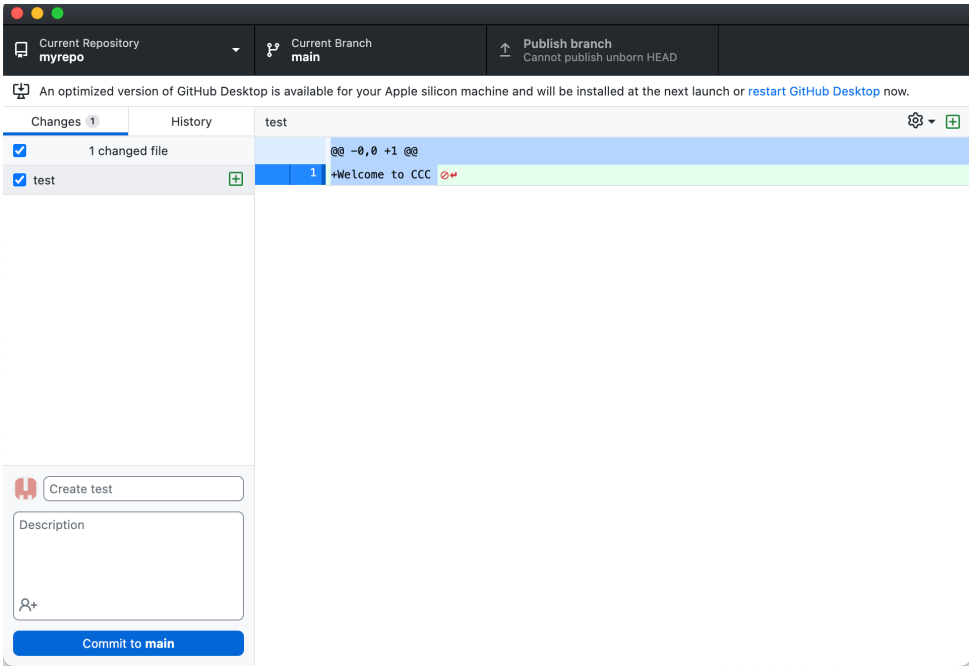
3. Add a code file to your repository

For this step, ignore GitHub. Just browse on your pc to the location where you have cloned the repository and create a test file:



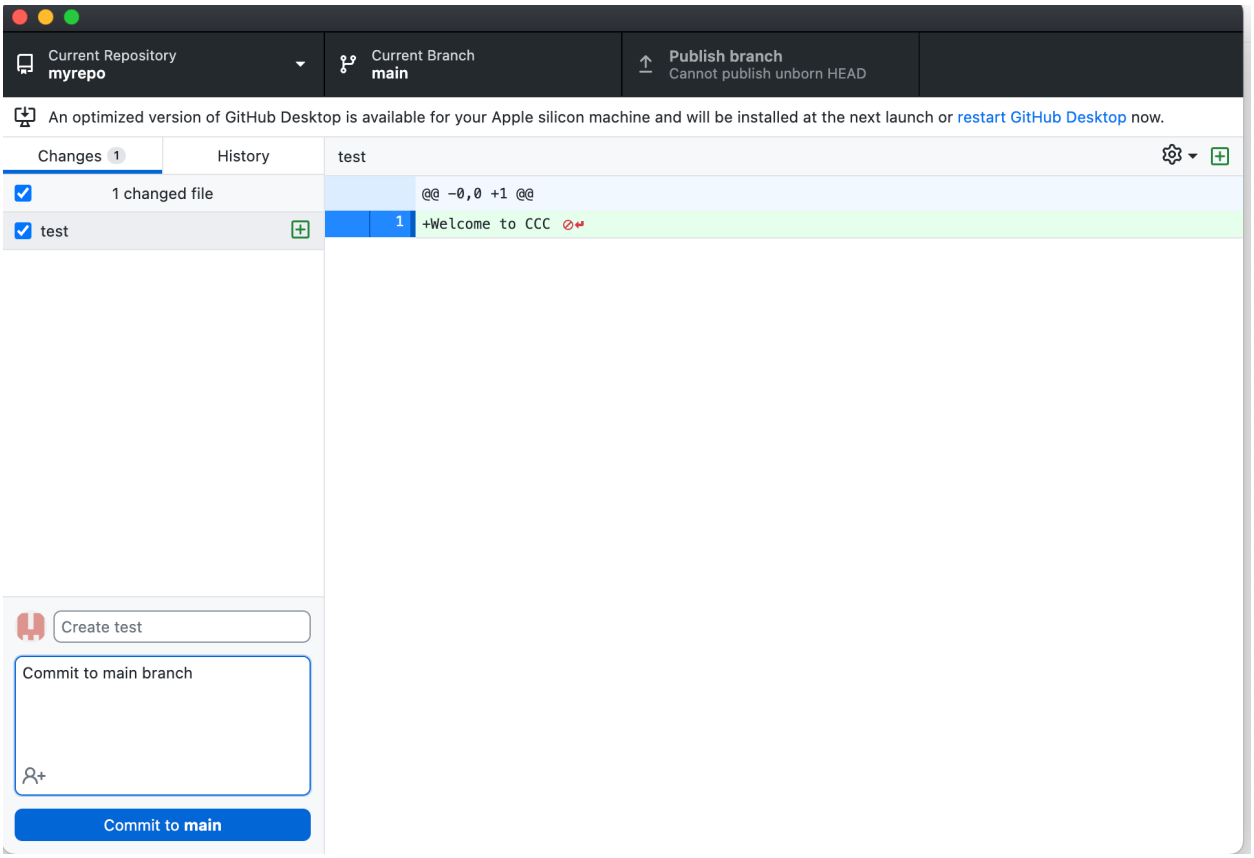
4. Add a code file to your repository.

Our repository has now changed, and we should update the new version to GitHub. Go back to GitHub Desktop and see that it has already noticed that you changed something:



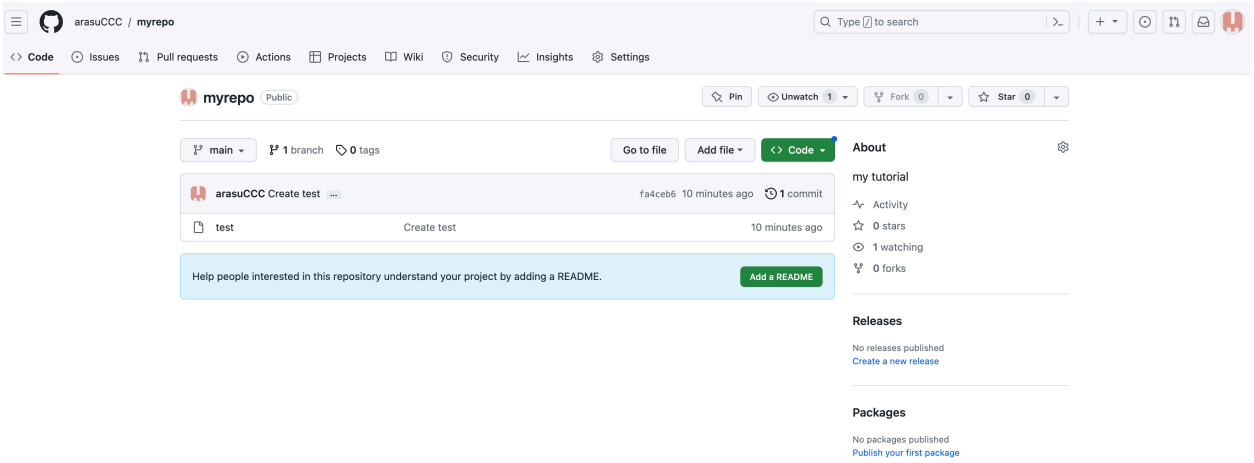
Two steps have to be undertaken to upload your code:

First, type a small comment like ‘Create test’ and click on “Commit to main branch”. Second, click on ‘Public branch’:



5. Verify that the code has changed on GitHub

You are all done with your first Github repository! Hurray! To verify that it really worked, you can now see your change on GitHub. Just go to the GitHub page of your repository and you'll see that the code has changed!



Git Setup for Mac and Windows

Next, we will set up Git on your personal computer. Follow the instructions for your operating system.

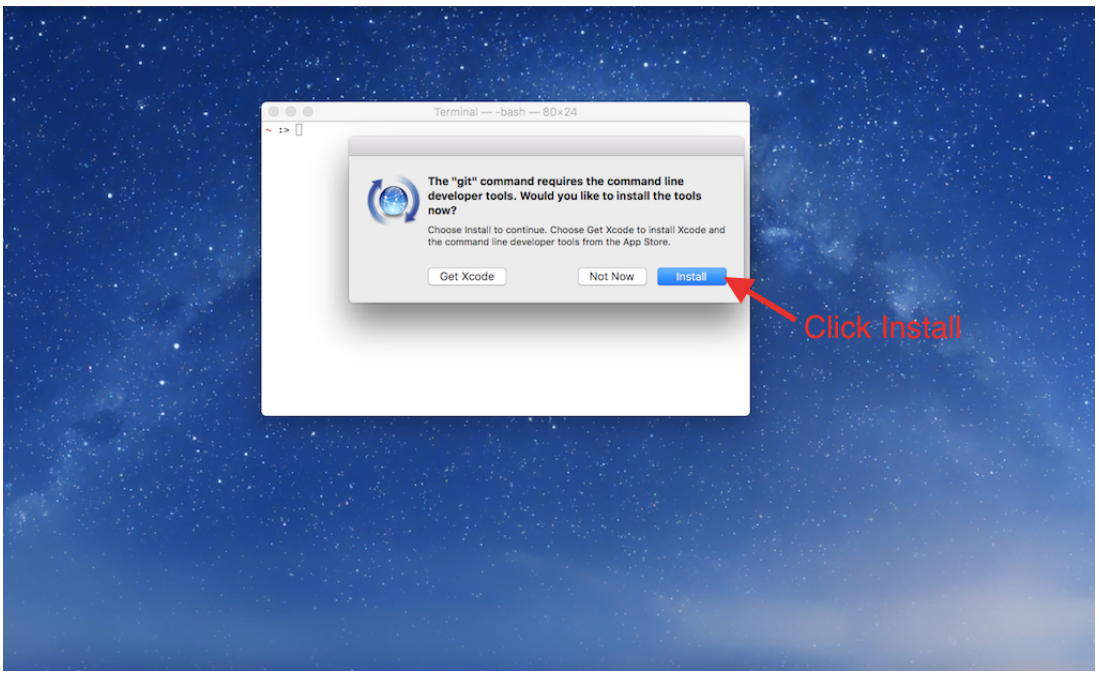
Mac users:

1. Launch the Terminal application.

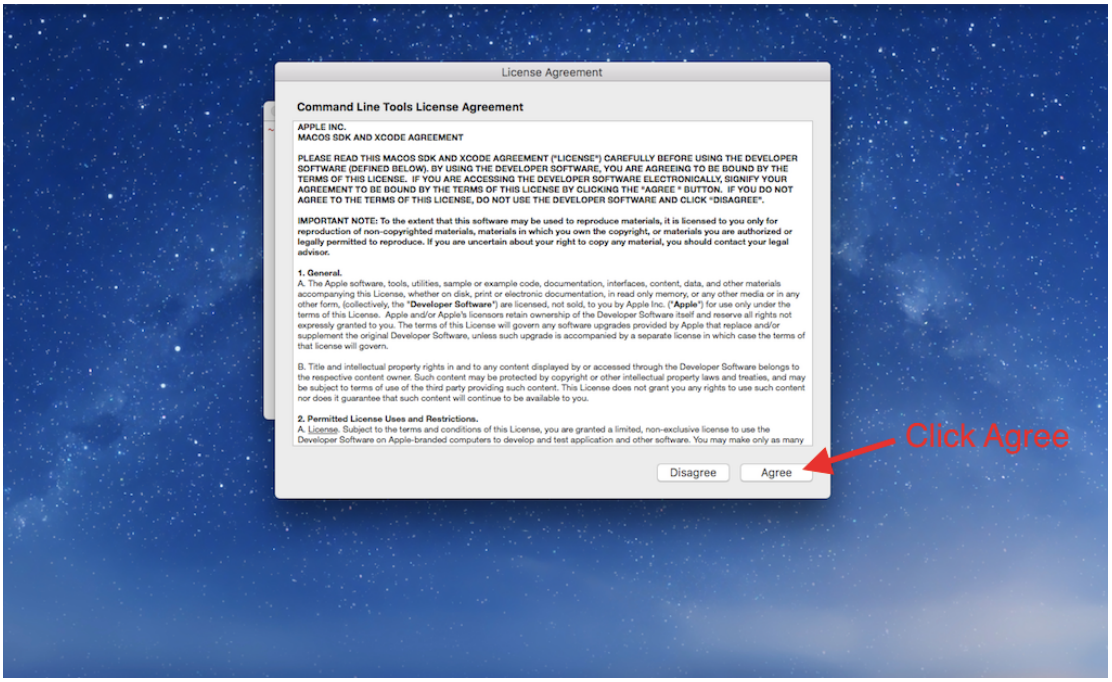
You can find it in /Applications/Utilities/. You can also use the Spotlight search tool (the little magnifying glass in the top right of your screen) to search for Terminal. Once Spotlight locates it, click on the result that says Terminal.

2. When Terminal opens, type in git and press enter.

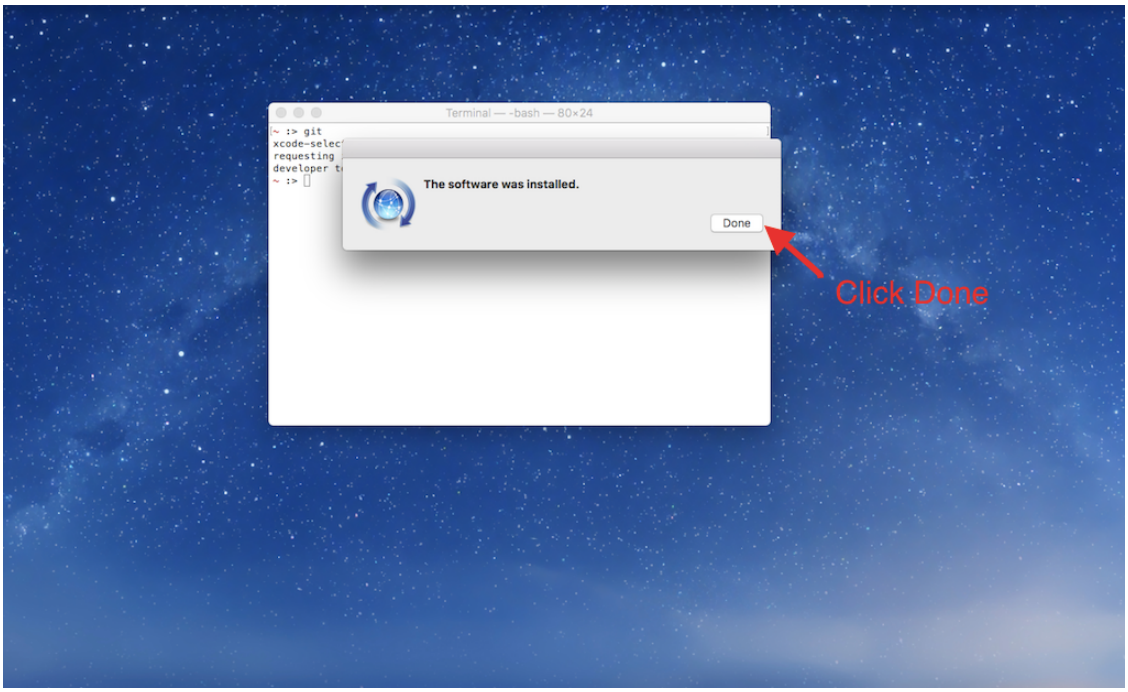
3. If you don't already have Git installed, a dialog will appear saying that "The 'git' command requires the command line developer tools. Would you like to install the tools now?" Click "Install".



Then click “Agree to the Terms of Service” when requested.



4. When the download finishes, the installer will go away on its own signifying that Git is now installed! Click “Done” to finish the installation process.



5. Navigate to GitHub's articles on setting up your Git username and email and follow the instructions for each using Terminal.

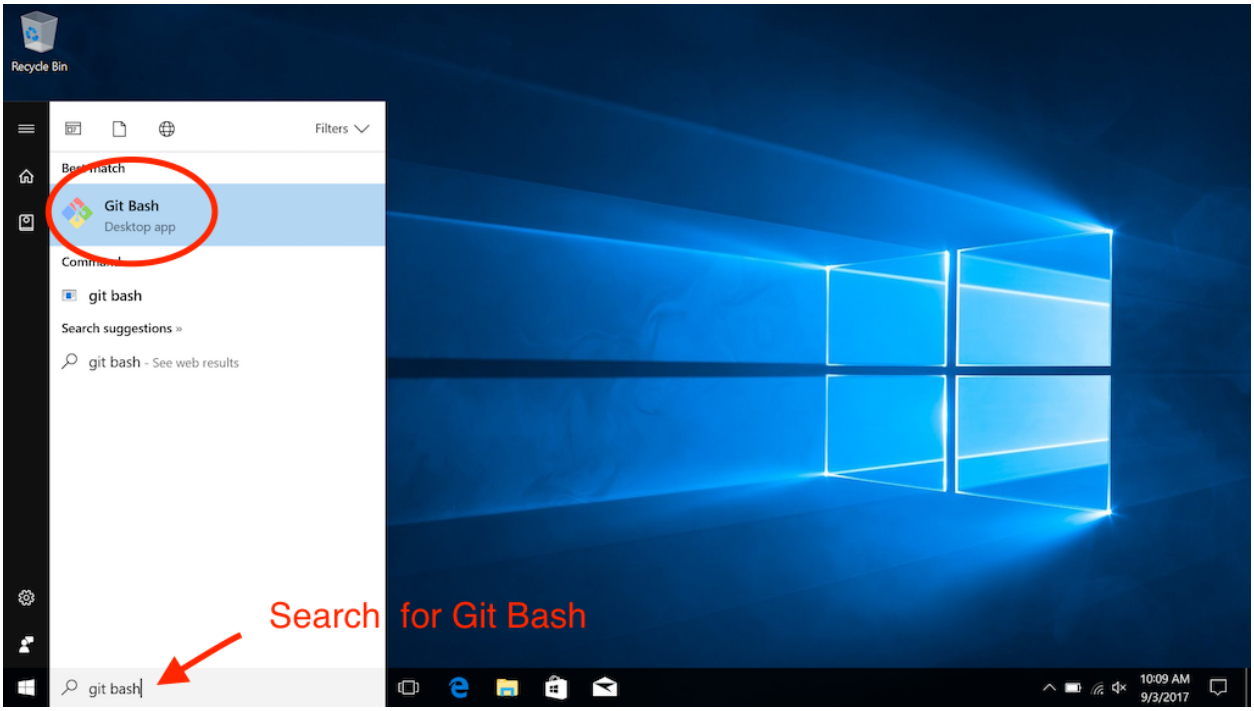
6. GitHub offers two authentication options, HTTPS and SSH, to keep your work secure. This is a security measure which prevents anyone who isn't authorized from making changes to your GitHub repository. In this article, we will use HTTPS. Navigate to GitHub's article on caching your password and follow the instructions to configure your computer to be able to use HTTPS.

Now skip down to the "Try it Out!" section below.

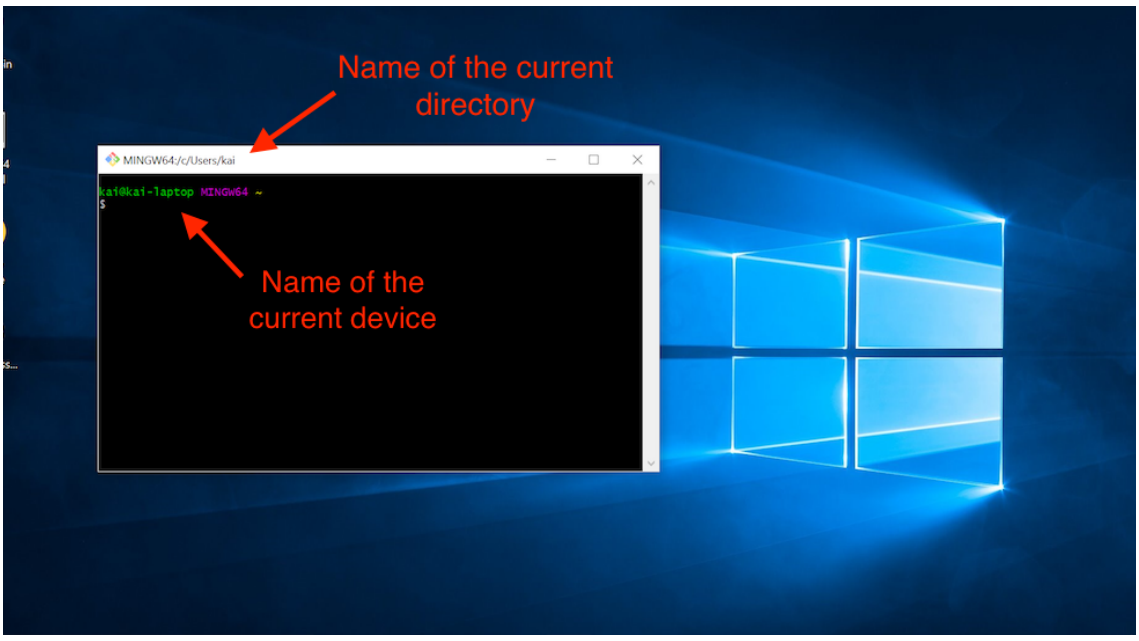
Windows users:

This portion of the guide assumes you have already installed a program called Git Bash which allows us access to Git on Windows. If you have not installed Git Bash, please refer to the previous tutorial on Command Line Interface (CLI) Setup and follow the instructions for installing Git Bash on Windows. Once you complete that you can continue with this guide.

1. Open the Start menu and search for the app, git bash. You should see ‘Git Bash Desktop app’ appear. Press Enter or click on the Git Bash icon to open the app.

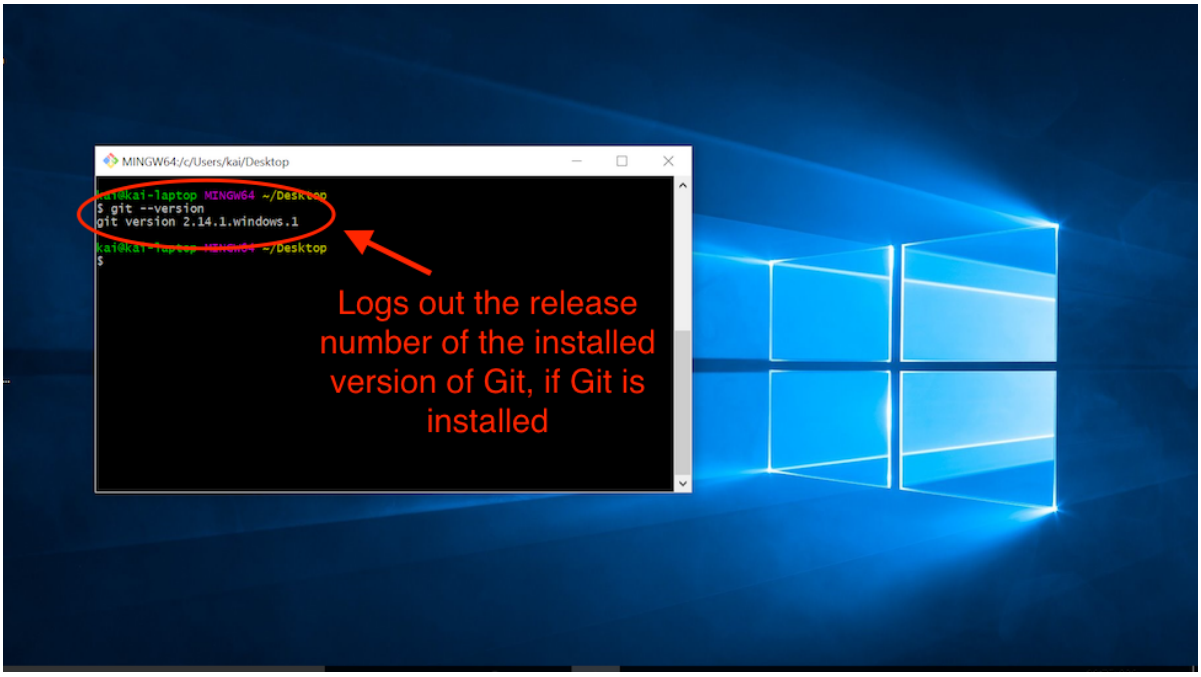


A new window will open that looks like this:



This window is our CLI, where we will use our Git commands.

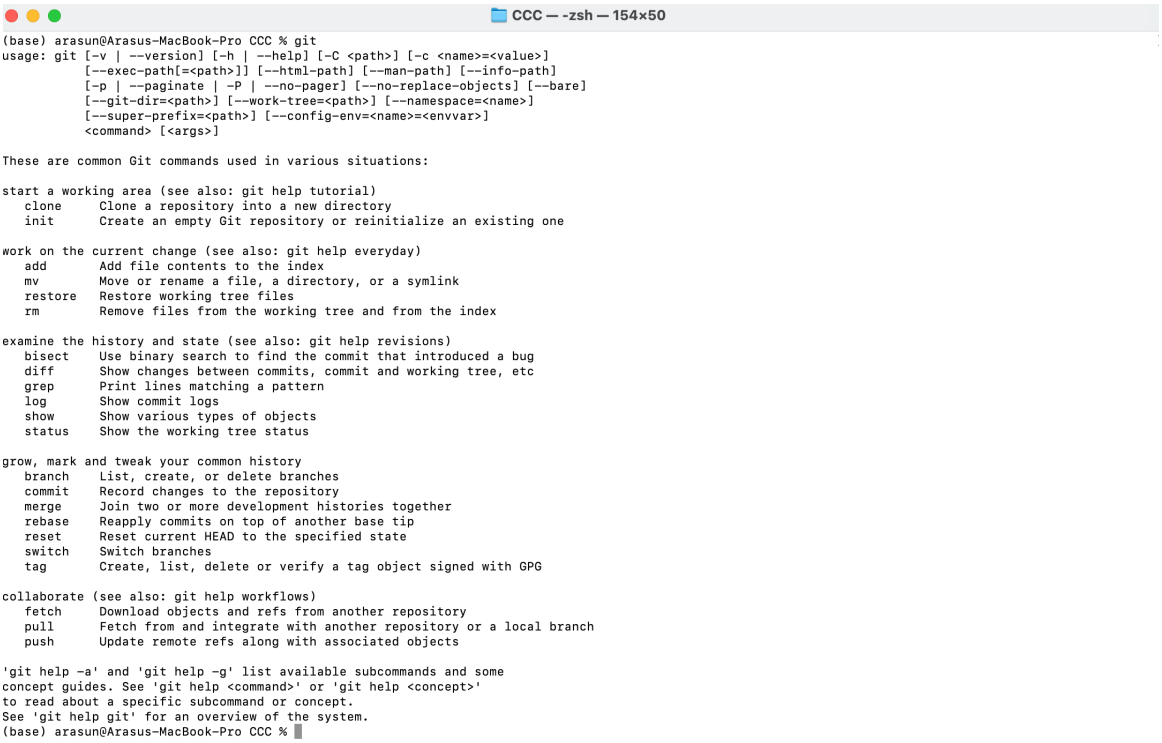
2. If you want to make sure that Git is installed, run git --version in the CLI. You should see a response that gives you the version of Git installed. It will look like this:



Git can now be used in the Git Bash app!

3. Navigate to GitHub's articles on setting up your Git username and email and follow the instructions for each using Git Bash.
4. GitHub offers two authentication options, HTTPS and SSH, to keep your work secure. This is a security measure which prevents anyone who isn't authorized from making changes to your GitHub repository. In this article, we will use HTTPS. Navigate to GitHub's article on caching your password and follow the instructions to configure your computer to be able to use HTTPS.

Here is the response for “git” command, it will show all the options on the git command.



Here are some basic GitHub commands with examples that you can use in your terminal:

1. Clone a Repository

   To clone (download) a repository from GitHub to your local machine:

   git clone <repository_url>

   Example:

   git clone https://github.com/arasuCCC/myrepo.git

2. Check a Repository Status

To check the status of your local repository and view changes:

git status

Example:

```
(base) arasun@Arasus-MacBook-Pro myrepo % git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(base) arasun@Arasus-MacBook-Pro myrepo %
```

3. Adding changes to staging area

To stage changes for commit:

git add <file_or_directory>

Example:

```
(base) arasun@Arasus-MacBook-Pro myrepo % ls
test    test1
(base) arasun@Arasus-MacBook-Pro myrepo % git add test1
(base) arasun@Arasus-MacBook-Pro myrepo %
```

4. Committing changes

To commit staged changes with a message:

git commit -m “your commit message”

Example

```
(base) arasun@Arasus-MacBook-Pro myrepo % ls
test    test1
(base) arasun@Arasus-MacBook-Pro myrepo % git commit -m "added the new file"
[main c9e07d1] added the new file
 1 file changed, 1 insertion(+)
 create mode 100644 test1
(base) arasun@Arasus-MacBook-Pro myrepo %
```

5. Pushing changes to GitHub

To push your local commits to the remote GitHub repository:

```
git push
```

6. Pushing changes from GitHub

To pull changes from the remote GitHub repository to your local repository:

```
git pull
```

7. Creating a new branch

To create a new branch

```
git branch <branch_name>
```

Example

```
git branch myrepo2
```

8. Switching to a branch

To switch existing branch

```
git checkout <branch_name>
```

Example

```
git checkout myrepo2
```

How to fix : Authentication Failed from the Command Line

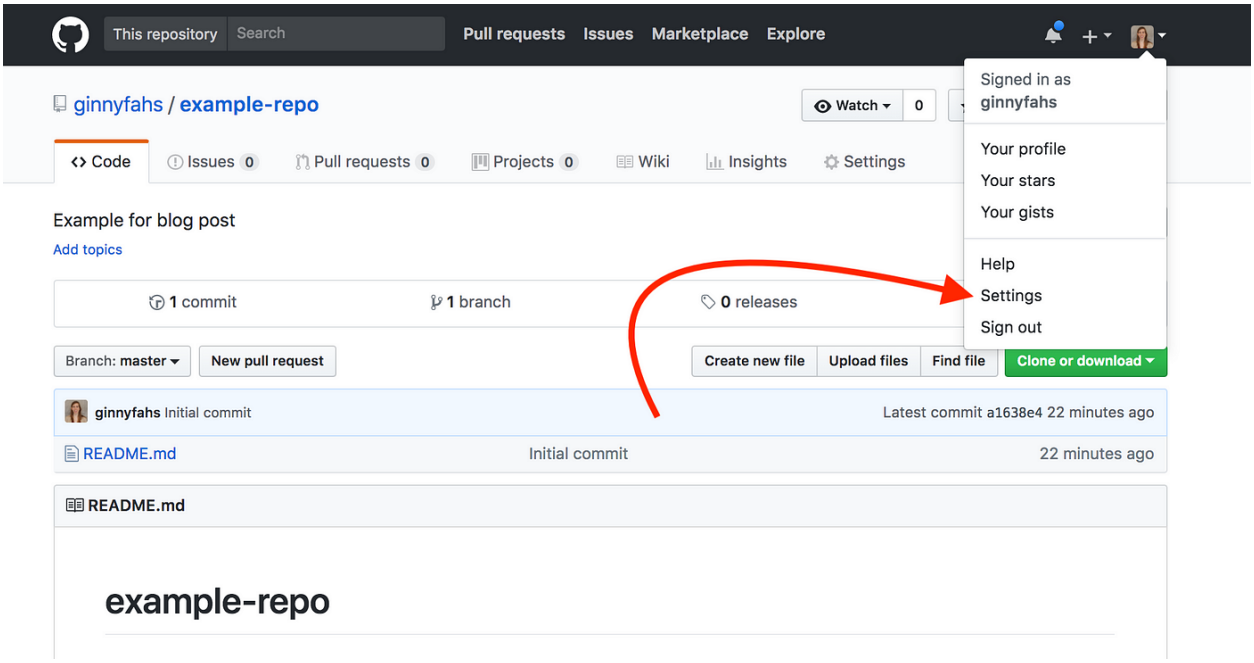
After setting up 2 Factor Authentication on GitHub, if you couldn't push your remote repositories from the command line anymore follow the below instructions.

~ :> git push origin my-branch
Username for 'https://github.com': myusername
Password for 'https://myusername@github.com': mypassword
remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/my-repository'

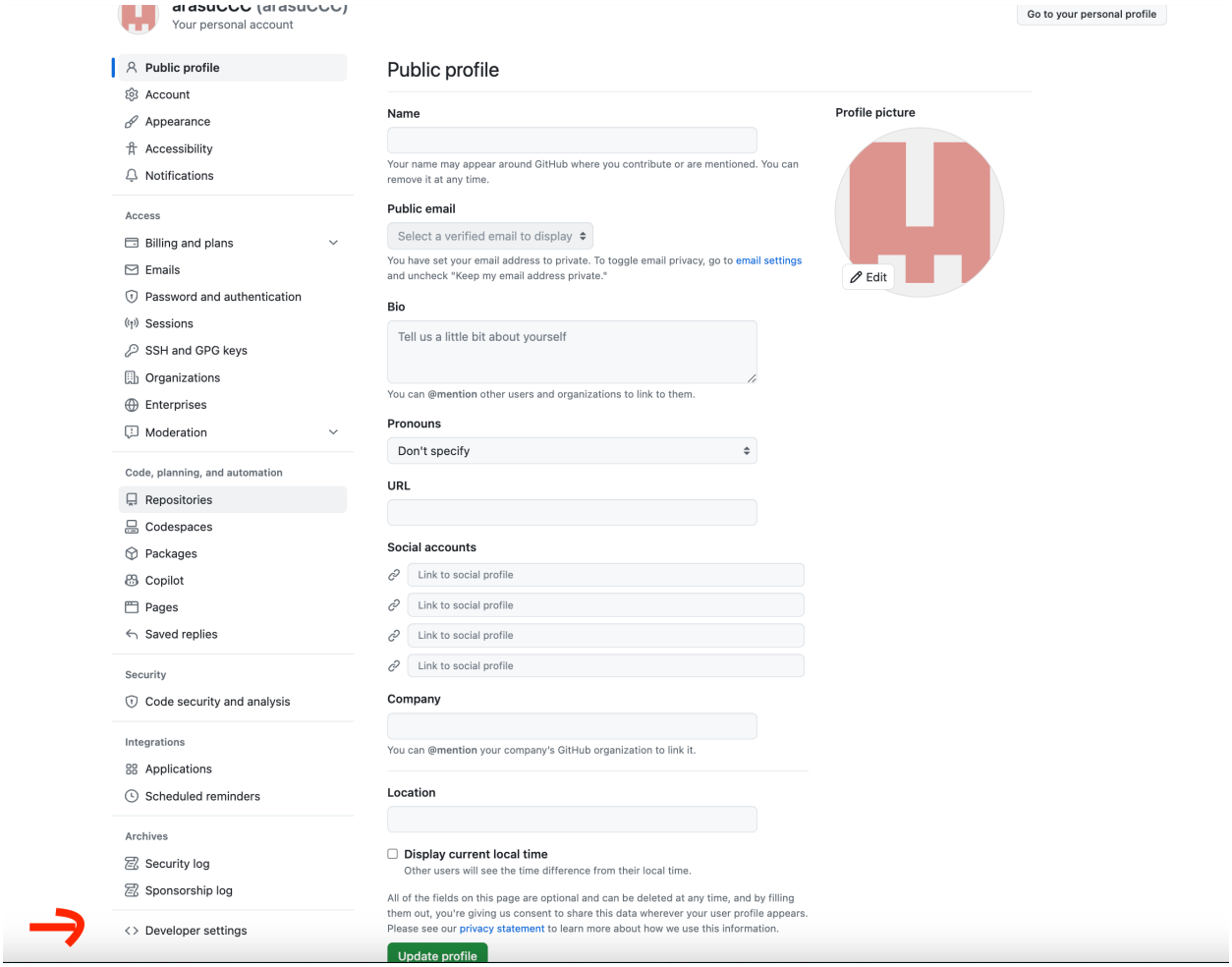
How to Authenticate on GitHub with 2FA

Command line authentication requires a personal access token.

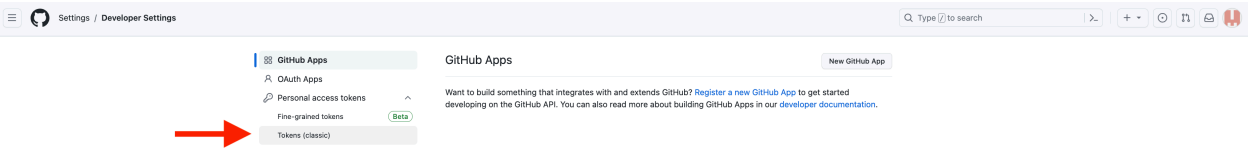
Go to Settings:



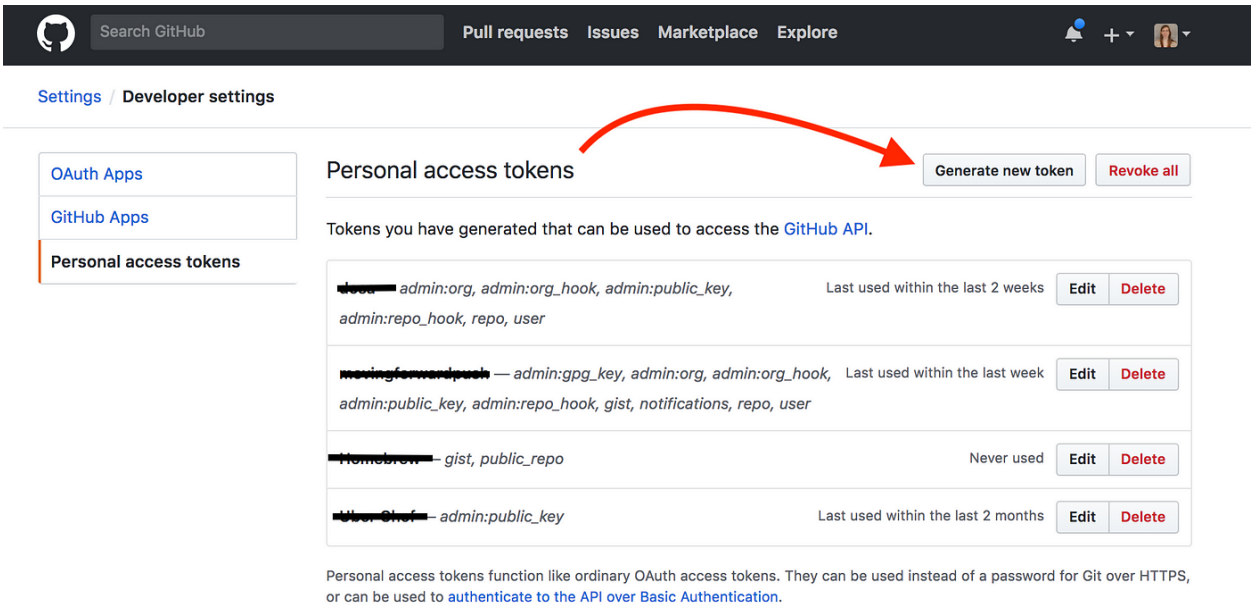
Then Developer settings:



Then Personal access tokens >> Tokens (classic):



Generate a new Personal Access Token (also classic). Make sure you copy the Personal Access Token as soon as it gets generated — you won't be able to see it again!



Paste the Personal Access Token into the “Password” field when you authenticate via the command line.

```
~ :> git push origin my-branch


Username for 'https://github.com': myusername

Password for 'https://myusername@github.com':  <-- Here :)

remote: Invalid username or password.

fatal: Authentication failed for 'https://github.com/my-repository'
```