Git Made Simple

A Comprehensive Guide for Beginners

Version: 1.0

Prepared by:
Kausshik Manojkumar

Date:
October 1, 2023

Table of Contents

0. Audience and Introduction

Audience

This user manual is tailored for individuals who are embarking on their journey into the world of version control with Git and GitHub. It's especially beneficial for beginners with little to no experience in handling version control systems. The guide assumes basic familiarity with terminal or command prompt operations, but even those new to these environments will find the step-by-step instructions accessible and easy to follow. The content is crafted to be clear and concise, making it a suitable reference for individuals from diverse backgrounds.

Introduction

Welcome to the Git and GitHub User Manual, your comprehensive guide to navigating the essential operations in Git, a powerful version control system used by developers worldwide. This manual guides you through the initial setup, demystifying the jargon and elucidating the processes, ensuring a smooth and confident start to your version control journey.

In this manual, you'll learn to:

- Set up Git and GitHub, ensuring your system is ready for action.
- Create and clone a repository, allowing you to work on projects from your local machine.
- Make changes and commit them, ensuring your project stays up-to-date.
- Push changes to a remote repository, facilitating collaboration and backup.

Embark on this learning expedition and emerge with a robust understanding and hands-on experience of Git's fundamental operations, ready to contribute effectively to the global developer community. Let’s dive in and start mastering Git and GitHub together!

1. Introduction

In the digital realm, managing various versions of your projects is a task that is as essential as it is tedious. It's easy to find yourself drowning in different versions of your project, unsure of what changes were made and when. Git emerges as a beacon of hope in such scenarios, ensuring you can seamlessly manage and track various versions of your projects.

1.1. Why Git?

Consider this: You're working on a document named "ProjectVersion1". After a review, you make some changes and rename it to "ProjectVersion2". You share it with a colleague, and they suggest some alterations, leading you to create "ProjectVersion3". Amidst all these changes, you remember a point you made in the very first version that should be included. But alas, navigating back to that original content can be a hassle if you didn't keep each version.

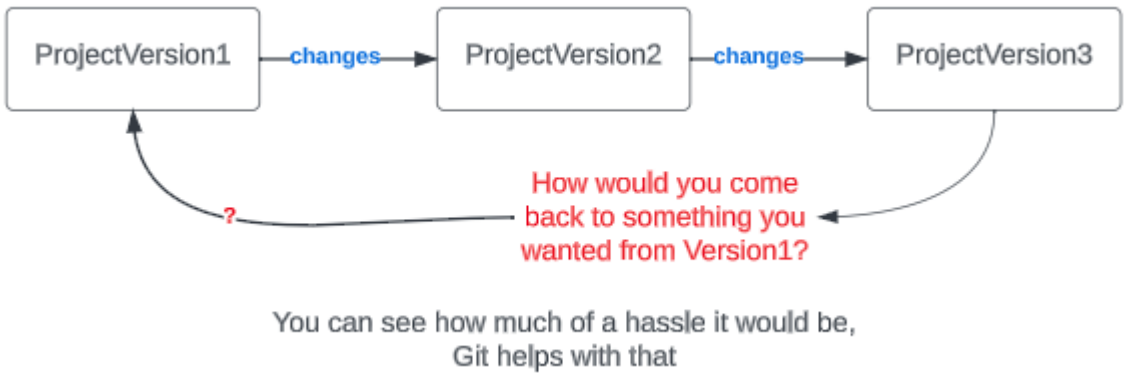

Fig. 1: A graphical representation of what when you do not use Git

This is a common challenge, and Git is the solution. It keeps a record of the evolution of your files, allowing you to revisit older versions, compare amendments, and, if necessary, revert to a previous version. Though Git is not fully compatible with binary files (like those of Word documents), it is impeccable for managing plain text files used for scripting, programming, and typesetting systems like LaTeX.

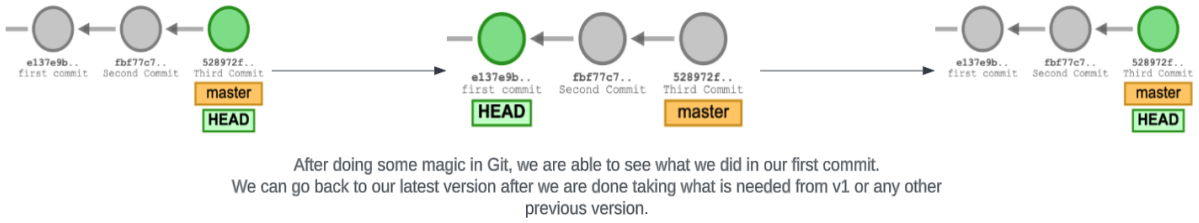

Fig. 2: A graphical representation of what happens in the background when you use Git on a file

Time to introduce our first few technical terms in a non-technical manner:

1. **COMMIT:** Think of a commit as a snapshot or a save point in a video game. It's a record of the changes you made to your files at a particular point in time. You can always go back to this snapshot if you need to.
2. **HEAD:** Think of HEAD as a bookmark in your project that tells you what part you're currently looking at. It keeps track of where you are.

1.2. Git vs. GitHub: Understanding the Difference

Git and GitHub, while related, serve different functions. Git is software that runs on your local computer, tracking changes in your files. It meticulously manages and stores different versions of your project without requiring GitHub.

On the other hand, GitHub is an online platform. It hosts your project repositories online for backup and collaborative purposes, synchronizing with the local repositories managed by Git. You can share your GitHub repositories with others and also access repositories shared by others.

git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the **Git** repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

Fig. 3: Graphical Table explaining the difference between Git and GitHub

1.3. Benefits of Using Git

- Version Control: Git allows you to maintain various versions of your project, giving you the freedom to revisit and revert to previous versions anytime.
- Collaboration: GitHub facilitates collaboration by hosting your project online, where other contributors can access and make changes.
- Backup: GitHub acts as an online backup for your project, ensuring you don't lose your work.
- Open Source: Many GitHub projects are open-source, allowing you to explore and learn from other developers' work.

In the next section, we'll get started by installing Git on your computer. It's the first step to making your work easier and more organized. Let's dive in!

2. Installing Git

Before we dive into the world of Git, we first need to ensure it's installed on your computer. Don't worry, the process is straightforward and quick.

2.1. Installation Steps for Various Operating Systems

FOR WINDOWS:

Go to the Website:

1. Visit the following website: gitforwindows.org. (this site will provide you with the latest version of Git for Windows.)
2. Click on the "Download" button to download the installer.

Install the Downloaded File:

3. Locate the downloaded file on your computer.
4. Double-click on the file to run the installer.
5. Follow the prompts, and accept the default settings unless you have a specific preference for something else.

Verify the Installation:

6. To make sure that Git has been installed correctly, open a new Command Prompt window.
7. Type the following command and press "ENTER"

```
git --version
```

8. If Git is installed correctly, you'll see a message with the version number of Git. If not, you may need to revisit the installation steps.

FOR MACOS (should be installed by default; start at Step 3):

Using HomeBrew:

1. If you have Homebrew installed, just type the following command in your terminal:

```
brew install git
```

2. If not, you can install Homebrew by following the instructions here: https://brew.sh/

Verify the Installation:

3. To make sure that Git has been installed correctly, open a new Terminal window.
4. Type the following command and press “ENTER”

```
git --version
```

5. The version number of Git should appear. If it does not, you may want to retry the process.

FOR LINUX USERS (should be installed by default; start at Step 3):

Use your distribution's package manager:

1. For Debian-based distributions like Ubuntu, type the following command:

```
sudo apt-get install git
```

2. For Red Hat-based distributions like Fedora, type:

```
sudo dnf install git
```

Verify the Installation:

3. To make sure that Git has been installed correctly, open a new Terminal window.
4. Type the following command and press “ENTER”

```
git --version
```

5. The version number of Git should appear. If it does not, you may want to retry the process.

2.2. Troubleshooting Git Installation

Encountering issues while installing Git? Don’t worry. Here’s a list of common problems and their solutions to help you troubleshoot Git installation.

1. Permission Denied

   Problem: You’re trying to install Git, but you don’t have the necessary administrative privileges.

   Solution: Run the installation command with ‘sudo’ to grant administrative permissions.

```
sudo apt-get install git
```

2. Command Not Found

   Problem: The terminal can't find the package manager (like apt or brew) you're using to install Git.

   Solution: Ensure the package manager is installed. For macOS users, you can install Homebrew using the following command:

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   ```

   And then use it to install Git:

   ```
   brew install git
   ```

3. Unable to Locate Package

   Problem: The package manager can't find the Git package.

   Solution: Update your package manager's list of available packages. For example, on Ubuntu, run:

   ```
   sudo apt-get update
   ```

   And then try to install Git again.

4. Installation Fails

   Problem: After installation, the system can't verify the Git installation.

   Solution: Ensure that Git is installed properly by running the following command:

   ```
   git --version
   ```

   This will return the installed Git version if the installation was successful.

NOTE:
Troubleshooting can feel frustrating, but patience is key. Review the error messages carefully and try the suggested solutions. If you still can't resolve the issue, seek help from online forums or the Git community. Your issue is likely a common one that other users have encountered and resolved before.

3. Introduction to Setting Up GitHub and SSH Keys

Before diving into the world of Git, it's crucial to set up your GitHub account and configure SSH keys. This setup is a one-time process that will save you from the hassle of entering your username and password each time you interact with GitHub. SSH keys provide a secure connection between your computer and GitHub, ensuring your code and data's security and integrity while working on projects. By following the upcoming steps, you'll establish a secure and efficient bridge between your local machine and your GitHub account, allowing you to focus more on coding and less on access management.

In this section, you will learn to:

1. Create a GitHub Account: A personal space for hosting and managing your projects and code.
2. Generate SSH Keys: Secure keys for identification and secure access to your GitHub account.
3. Add SSH Keys to GitHub: Link your computer to GitHub for seamless and secure operations.
4. Configure Git: Ensure your local Git is ready and personalized for use with GitHub.

Now, let's begin this simple and essential setup to ensure a smooth and secure coding journey with Git and GitHub.

3.1. Getting Started with GitHub

1. Visit GitHub:
   Launch your web browser and go to www.github.com.
2. Sign Up:
   Visit GitHub.com and register. Use a genuine email address, as it will be used to identify your contributions.

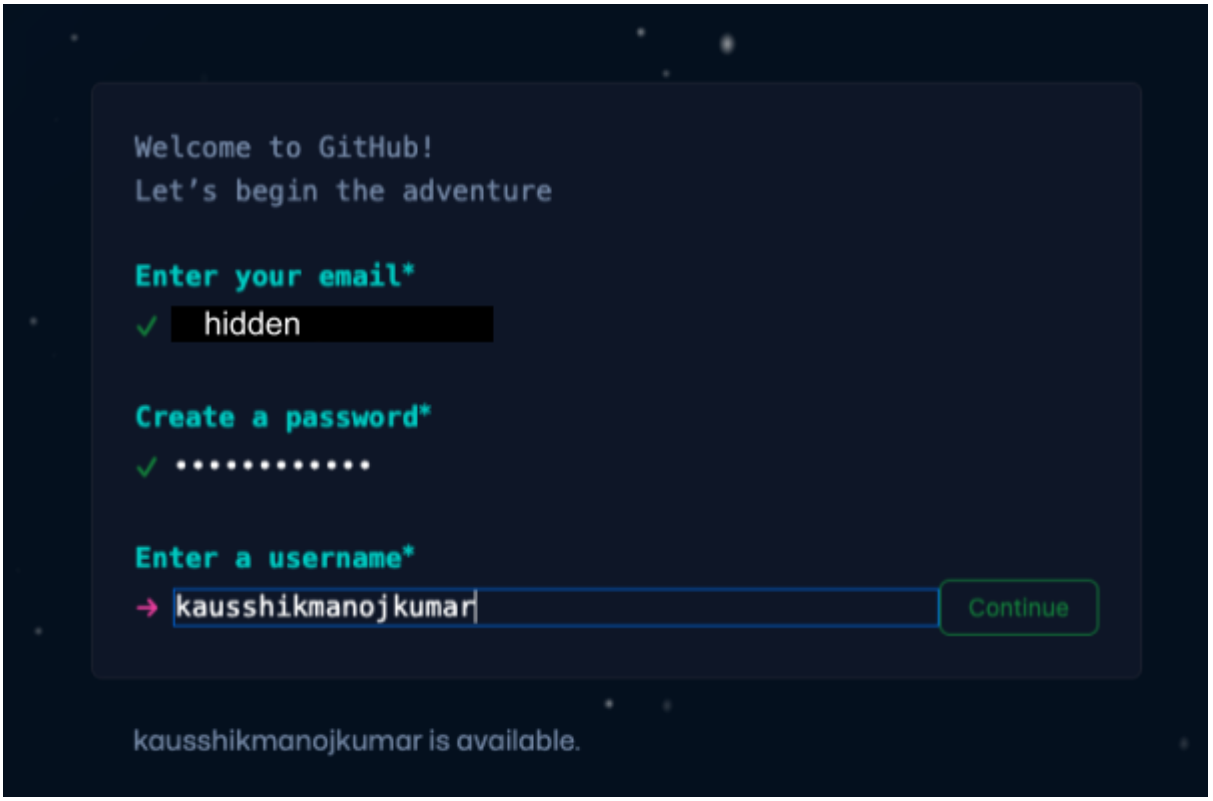

Fig. 4: GitHub Sign Up Page

3. GitHub Dashboard:

   Upon Successful registration, you should be able to see the following things on your dashboard

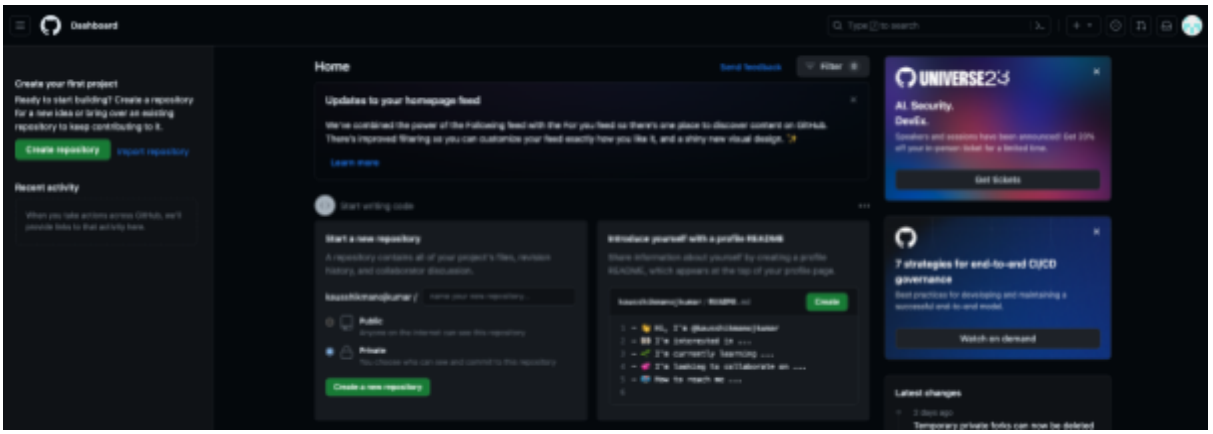

Fig. 5: GitHub Dashboard after signing up

3.2. Setting Up Git for GitHub

1. For smooth functioning, Git requires your identification to connect a local Git user to GitHub, enabling clear visibility of your contributions to team projects.

   Execute the following commands in your terminal. Ensure to substitute with your own details within the quotes:

```
git config --global user.name "Your Name"
git config --global user.email "yourname@example.com"
```

2. GitHub now uses “main” as the default branch for new repositories, a shift from the previous “master.” Adjust the default branch using this command:

```
git config --global init.defaultBranch main
```

What is a branch?

BRANCH: it is essentially a unique set of code changes (or just changes in general) with a unique name.

3. In addition, it’s recommended to set your default branch behavior to merging with this command (this is an advanced command, and we do not need to know what this means for our understanding of Git):

```
git config --global pull.rebase false
```

4. Verify your setup by matching the output of these commands with your entered name and email:

```
git config --get user.name
git config --get user.email
```

Sample Output:

```
(base) Kausshiks-MacBook-Pro:~ kausshik$ git config --get user.name
KAUSSHIK
(base) Kausshiks-MacBook-Pro:~ kausshik$ git config --get user.email
kausshikmanojkumar@gmail.com
```

5. **ATTENTION** macOS Users: Prevent Git from tracking .DS_Store files (metadata files created by Finder) using these commands:

```
echo .DS_Store >> ~/.gitignore_global
git config --global core.excludesfile ~/.gitignore_global
```

3.3. Setting up SSH Keys

An SSH key acts like a lengthy password, identifying your machine securely. GitHub employs SSH keys for password-free repository uploads. An SSH Key is used to prove your identity, so you don’t have to enter your username and password to the Git server every time you upload code to the servers. Using SSH keys is not required to use Git, but they make long-term use simpler.

Creating an SSH Key:

1. Check for existing SSH Keys by running the following command:

```
ls ~/.ssh/id_ed25519.pub
```

2. If the terminal shows “No such file or directory,” create a new SSH key with:

```
ssh-keygen -t ed25519 -C "your@email.com"
```

Associating the SSH Key with GitHub:

The next step would be to inform GitHub about your SSH key for passwordless code pushes.

1. Log into GitHub, click your profile picture, and choose Settings.
2. Click SSH and GPG keys, and then New SSH Key.
3. Name your key descriptively.
4. To copy your public SSH key, use the cat command on the terminal:

```
cat ~/.ssh/id_ed25519.pub
```

5. Copy the displayed key (beginning with ssh-ed25519) and paste it into GitHub’s key field. Set the key type as Authentication Key and click Add SSH Key. Congratulations on successfully adding your SSH key!

4. Working Toward Your First Commit

In this section, you will learn how to create your first Git repository, clone it to your local desktop, and make your first commit. Here is a step-by-step guide, along with explanations of Git terms, that will be helpful for beginners.

4.1. Creating A Repository on GitHub

First, let us begin by defining what a repository is.

REPOSITORY (Repo): A repository is like a folder for your project that contains all of your project's files and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private.

Now, let's create one on our new GitHub Account:

1. Navigate to GitHub (www.github.com).
2. Log in to your account.
3. Click the '+' sign in the upper right corner and select 'New repository'.

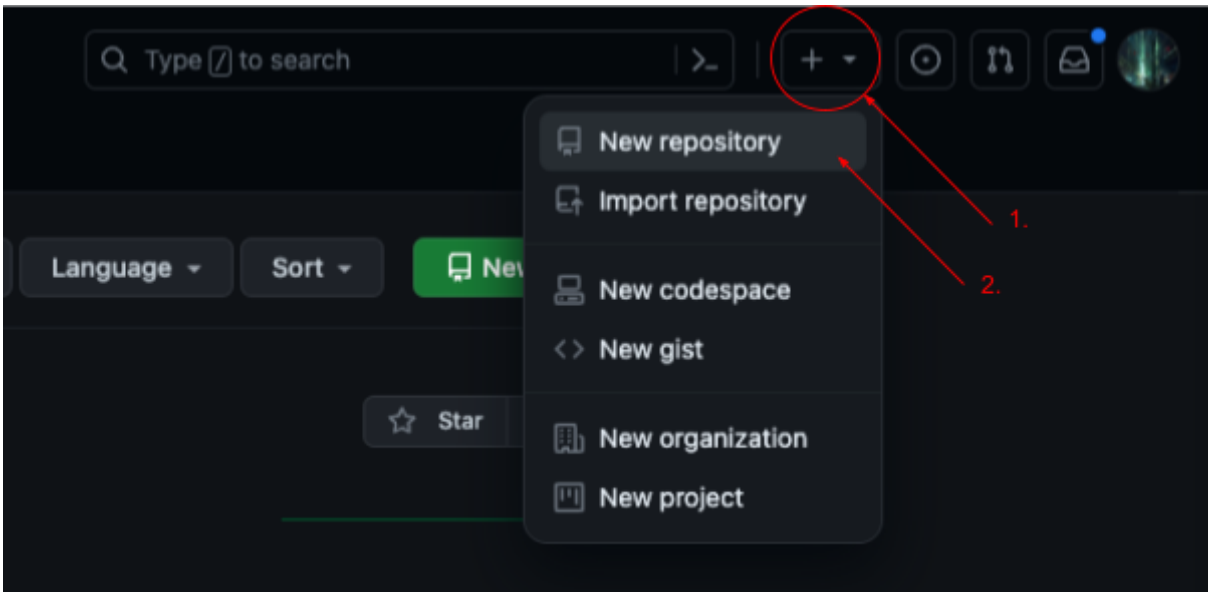

Fig. 6: Create New Repository Button on the dashboard

4. Name your repository and provide a brief description.
5. You can also add a README file. This file is mainly used for when you want to provide the reader with any information about the repository that they may find helpful.
6. Choose to make the repository public (visible to everyone) or private (visible only to you and the collaborators you add).

7. Click 'Create repository'.

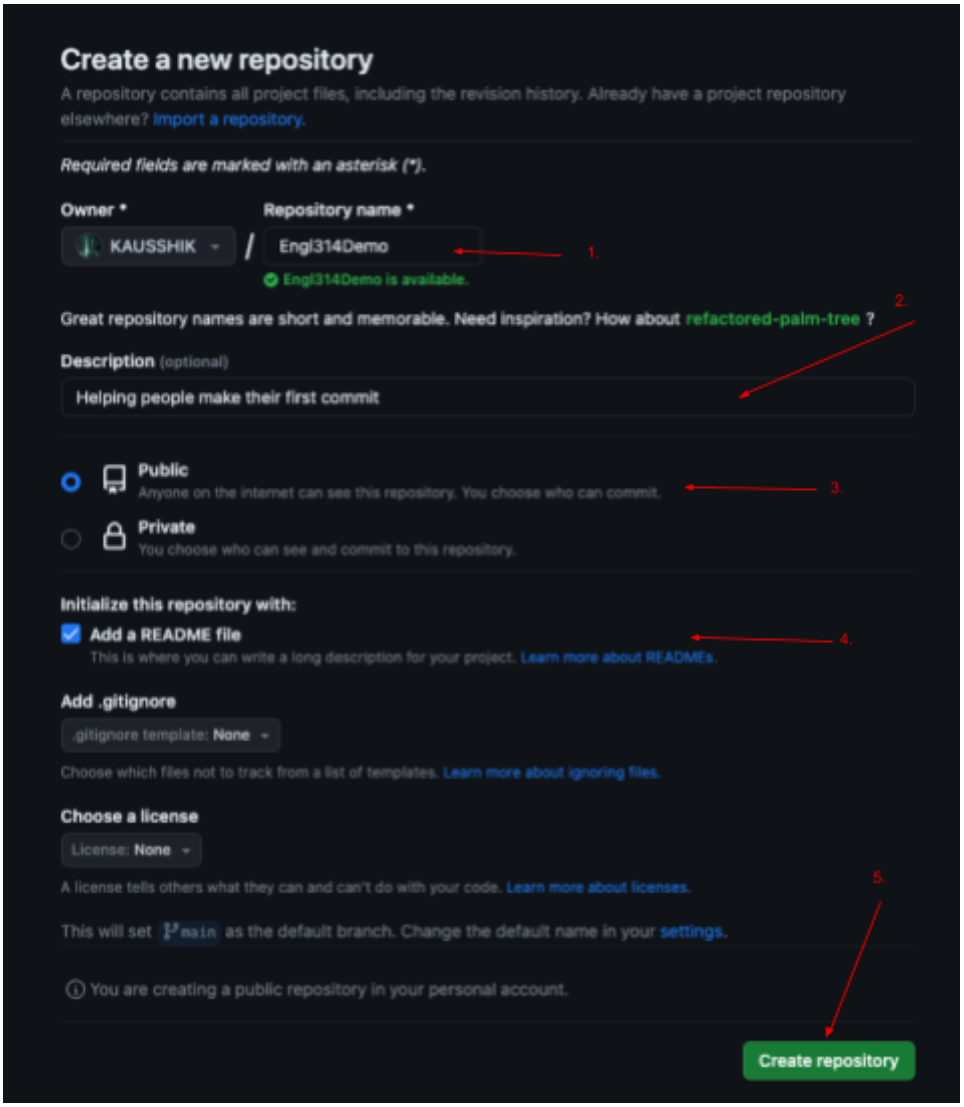

Fig. 7: Page to Create a New Repository

8. Hooray, you made your first repository. This is how it may look:

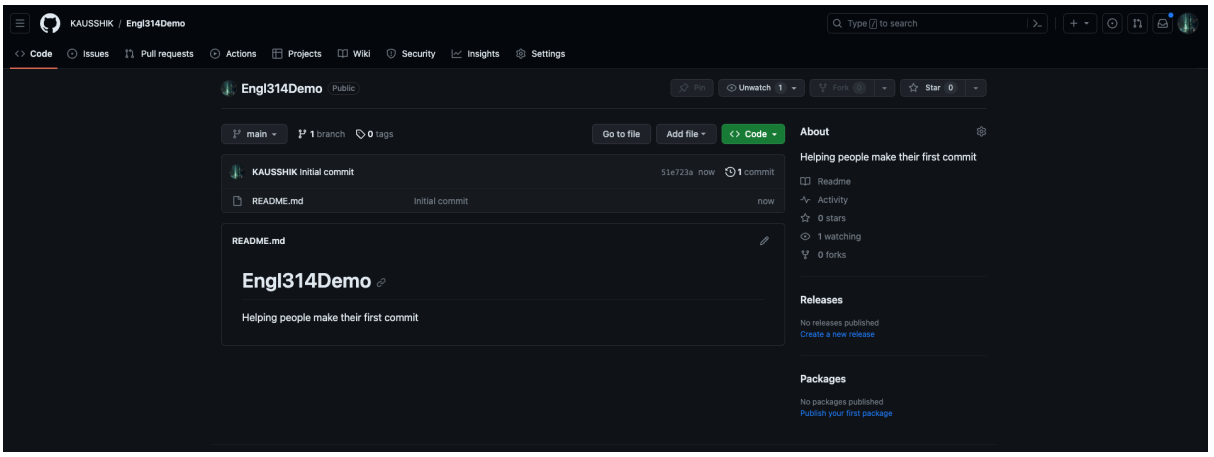

Fig. 8: Image of the new Repository created

4.2. Cloning the Repository to Your Local Machine

Cloning a repository copies it from the platform (like GitHub) to your local machine, allowing you to work on the project from your own computer. Think of a repository as a folder on the web; when cloned to your local machine, a new folder appears, housing the contents of the repository. Crucially, this local folder retains a link with the original online repository, enabling seamless synchronization of changes between your local copy and the GitHub repository.

In the steps below, I will be working with a repo called “Engl314Demo” on my MacOS laptop. Keep in mind that while this is a specific example, you can follow the same steps for any repository you wish to clone and work with on your local machine.

1. On the main page of your new GitHub repository, click the 'Code' button.
2. Copy the URL shown.

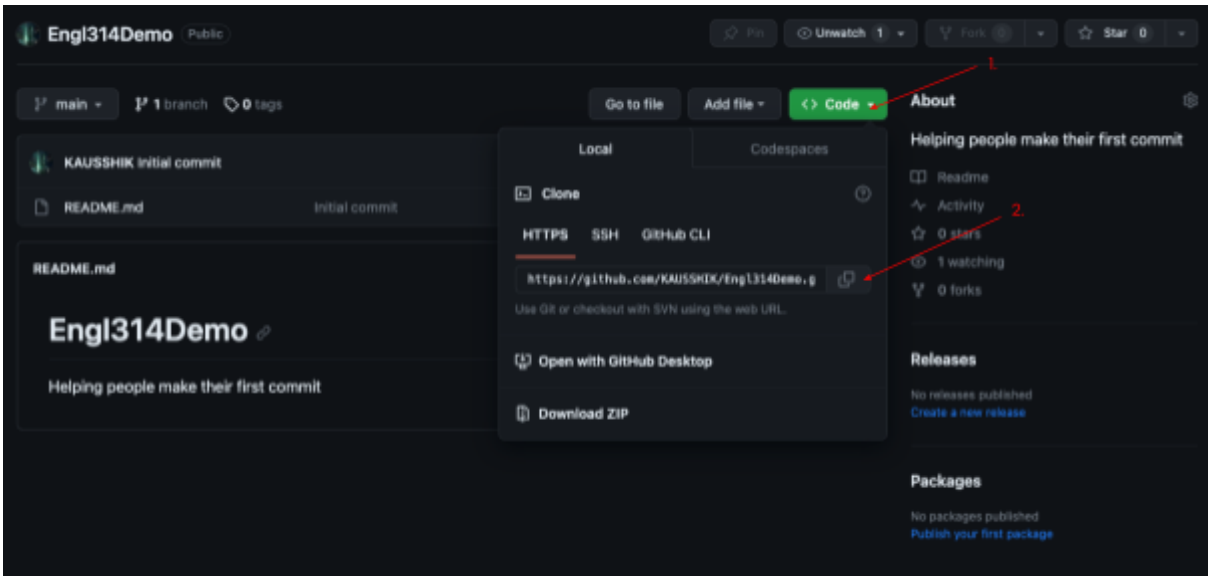

Fig. 9: Instruction to copy the link of the repository we need to clone

3. Open your computer's terminal or command prompt.
4. Navigate to the location where you want to store the cloned repository using the cd command
   For Example:
   I want to store mine inside a directory (folder) called “Sandbox.”

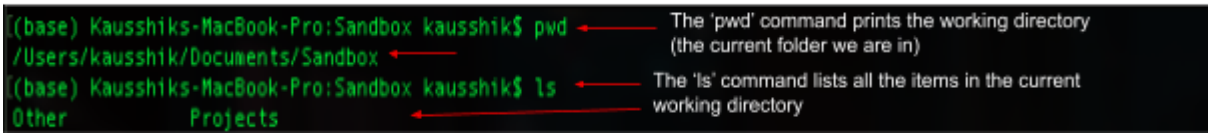

Fig. 10: Terminal Commands to navigate to the right directory

Notice how there are folders called Other and Projects inside the Sandbox folder. We will see how that changes after we clone the repository.

5. Now that you are in the right directory, you can run the 'git clone' command as follows:

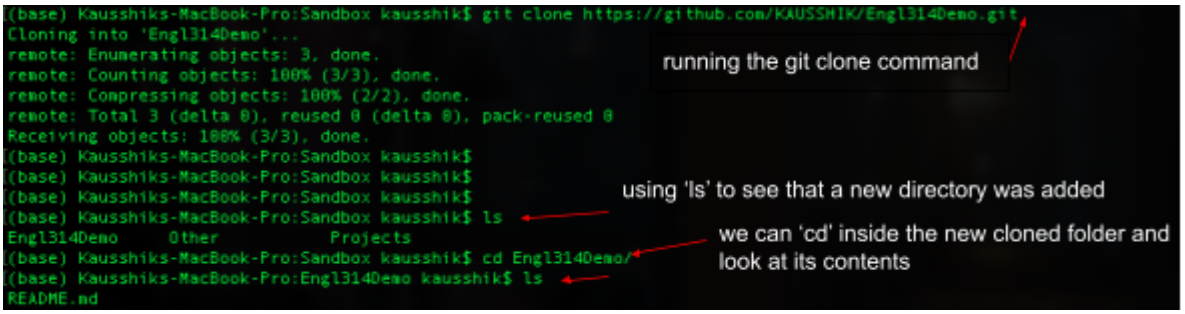

Fig. 11: Terminal Commands execute git clone

6. We have now successfully cloned the folder, and you can see that the Sandbox folder now has the cloned repository "Engl314Demo," with a README file inside it, which we created upon initializing it in GitHub.

   WAIT! You just encountered the first git command. It is best to note it down for future reference:

GIT CLONE
Command: git clone <repository URL>
Definition: This command is used to create a copy of a repository on your local machine from a remote server (such as GitHub).

4.3. Making Changes and Your First Commit

After you have cloned your repository to your local machine, you may want to make changes to the files and save (or "commit") these changes. In this section, we'll walk through the process of making a commit.

A commit in Git is like a snapshot of your project at a specific point in time. It contains all the changes to your files and directories up to that point.

Every commit has a unique ID that allows you to keep track of the changes and who made them.
Before committing, let's make a change. For this example, create a directory (folder) within the new repo named “Folder1”. Then navigate into that folder using the cd command and generate a new text file titled “hello.txt”.

Making a Change:

1. Make a new Folder

   Make sure you are inside your cloned repo folder, then create a new folder, “Folder1” using:

```
mkdir Folder1
```

```
(base) Kausshiks-MacBook-Pro:Sandbox kausshik$ pwd
/Users/kausshik/Documents/Sandbox
(base) Kausshiks-MacBook-Pro:Sandbox kausshik$ cd Engl314Demo/
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ pwd
/Users/kausshik/Documents/Sandbox/Engl314Demo
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ mkdir Folder1
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ ls
Folder1         README.md
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$
```

Fig. 12: Terminal Command to make a New Folder

2. Navigate into the New Folder

   Make sure to cd into the new Folder1 you created, using:

```
cd Folder1
```

```
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ cd Folder1/
(base) Kausshiks-MacBook-Pro:Folder1 kausshik$ pwd
/Users/kausshik/Documents/Sandbox/Engl314Demo/Folder1
(base) Kausshiks-MacBook-Pro:Folder1 kausshik$
```

Fig. 13: Terminal Command to navigate to the New Folder

3. Create a New Text File

   Create a new text file, or any other type of file, inside Folder1 using:

   For MacOS/Linux:

```
touch hello.txt
```

   For Windows:

```
type nul > hello.txt
```

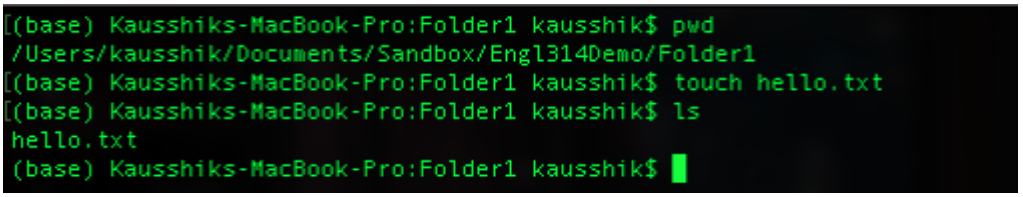

Fig. 14: Terminal Command to create a new Text File

Now, we have added a new folder and a new file. These changes, although empty, are still changes. So we can go ahead and commit them.

Making a Commit:

1. Check Git Status

   To see the changes you have made (if any) try running

   ```
   git status
   ```

   in the Terminal and look at its output

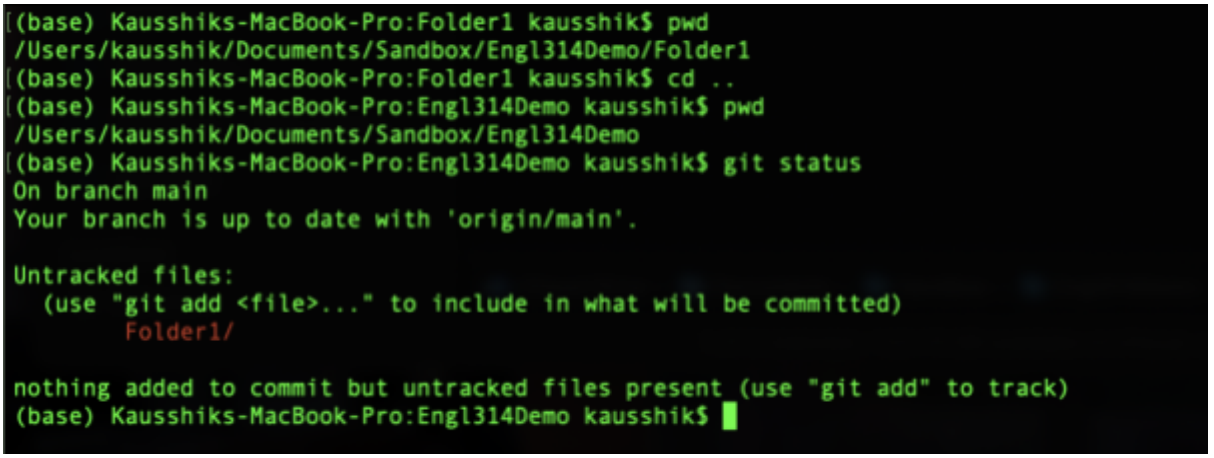

Fig. 15: Terminal Command to execute git status

This shows us that there are some untracked files; what is an untracked file, you may ask.

An untracked file in Git is a file that exists in your directory but has not yet been added to the version control system. These are files that Git is not monitoring for changes, and they are not included in commits unless explicitly added.

WAIT! You just encountered another git command. It is best to note it down for future reference:

GIT STATUS
Command: git status
Definition: This command is used to display the changes made in your repository, listing modified, added, or removed files.

2. Add Files to the Staging Area

   Before making a commit, you need to add the files to the staging area. The staging area is a place where Git tracks the changes you want to commit to the version control history. Here, you decide what changes you want to commit by adding files to or removing files from the staging area.

   To add untracked files or changes to the staging area, use the git add command followed by a period "." . For example:

   ```
   git add .
   ```

   This command tells Git to track and add all the untracked and modified files to the staging area at once and include it in the next commit.

   WAIT! You just encountered another git command. It is best to note it down for future reference:

   GIT ADD
   Command: git add .
   Definition: This command is used to add the changes to the staging area, preparing them for the next commit.

3. Verify Changes are Staged:

   After adding the files to the staging area, it's a good practice to verify the changes before making a commit. Run:

   ```
   git status
   ```

   Now, the files will appear under "Changes to be committed," indicating they are staged and ready for commit.

```
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ git add .
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Folder1/hello.txt

(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$
```

Fig. 16: Terminal Command to execute git add . and check status

4. Make the Commit:

   Now that your changes are staged, you can commit them to your repository with the git commit command. This command requires a message that describes the changes you are committing. Use the "-m" option to add a commit message:

```
git commit -m "Add hello.txt file"
```

   This command commits the staged changes to the repository with a descriptive message, providing a clear history of your project and the changes you have made.

```
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$ git commit -m "Add hello.txt file"
[main a1685b5] Add hello.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Folder1/hello.txt
(base) Kausshiks-MacBook-Pro:Engl314Demo kausshik$
```

   WAIT! You just encountered another git command. It is best to note it down for future reference:

GIT COMMIT
Command: git commit -m "Your commit message"
Definition: This command is used to save your changes to the local repository. The `-m` flag allows you to add a message that describes the commit. Writing clear, comprehensive commit messages is essential for maintaining a readable history of your project.
Example: After adding your files to the staging area with `git add .`, use `git commit -m "Initial commit"` to save the changes with a message describing the commit as the initial commit.

You can see your commit in GitHub. It may look similar to this:

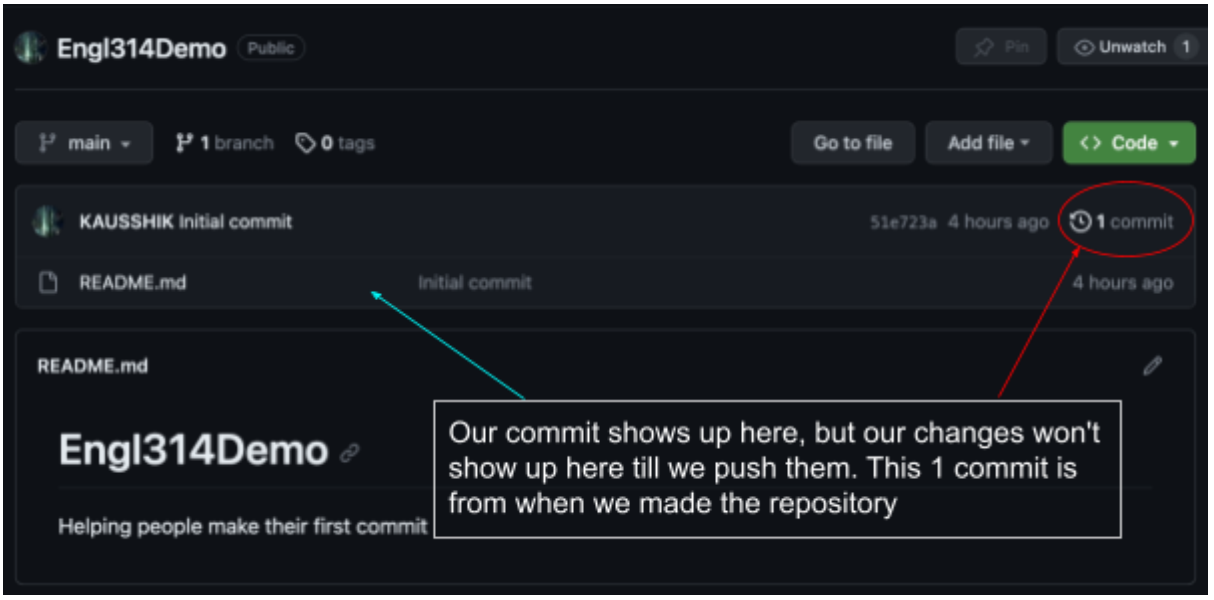

Fig. 17: Checking GitHub after a Commit

5. Git Push and Beyond

At this stage, you've successfully made changes and committed them. Now, let's push these changes to the remote repository and delve a bit deeper into the next steps, including making additional changes, commits, and pushes.

Pushing, in Git terms, refers to the process of uploading your local commits to the remote repository or server (in our case: GitHub).

5.1. Pushing Changes to Remote Repository

1. Making your First Push:

   Let us push these files to the remote repository (GitHub) and look at the output. Run the following command to do so:

   git push

   Here is what we see in the terminal:

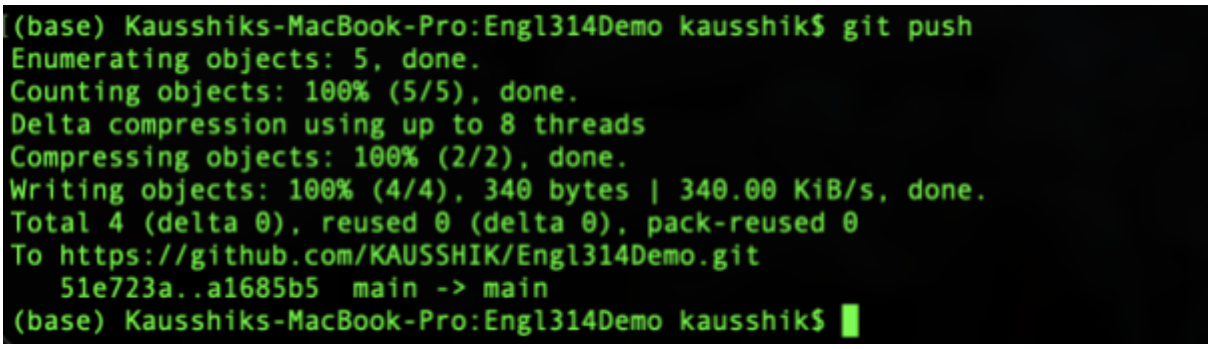

Fig. 18: Terminal Commands to git push

   Now your code has successfully been pushed from your end (your local machine), it will now be reflected on GitHub.

   WAIT! You just encountered another git command. It is best to note it down for future reference:

GIT PUSH
Command: git push
Definition: The git push command is used to upload local repository content to a remote repository.

NOTE:
If you are pushing for the first time, you might be prompted for your GitHub credentials.

Here is what we see on GitHub:

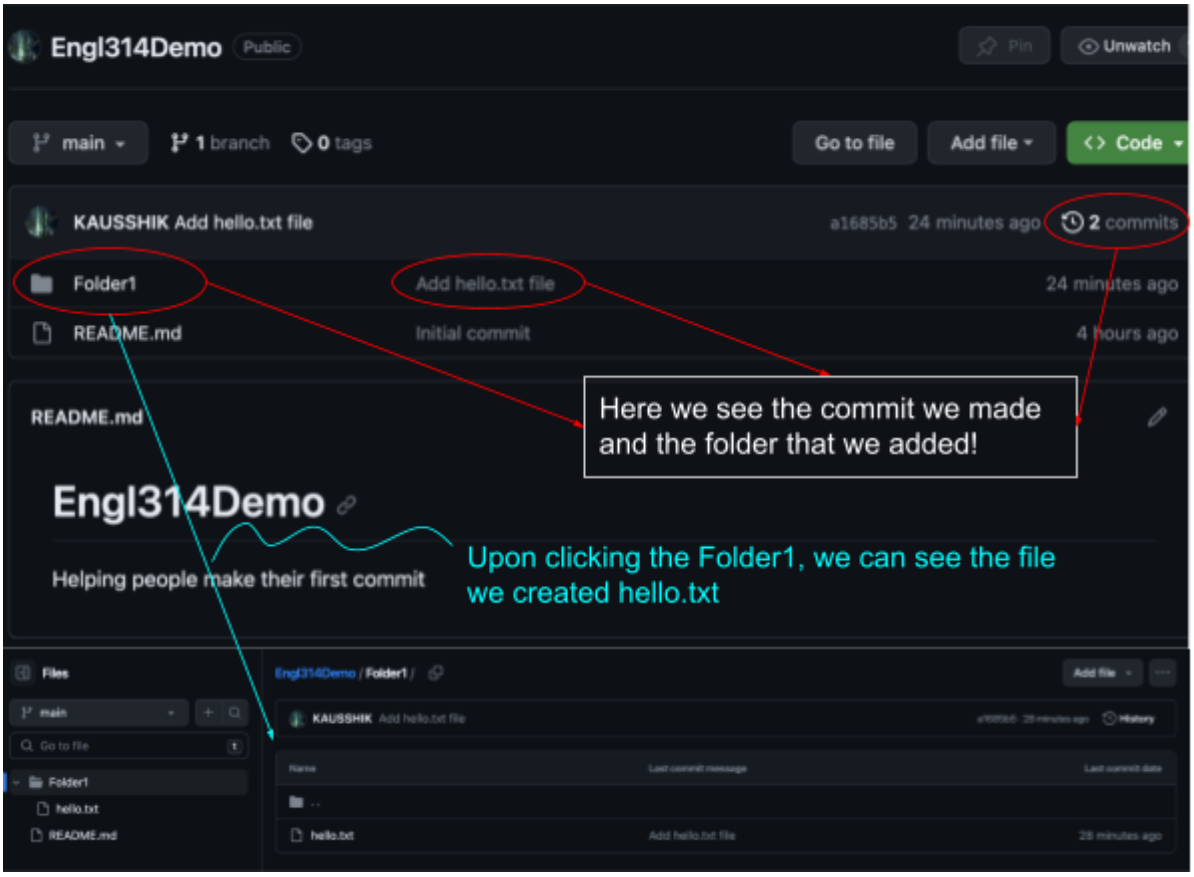

Fig. 19: GitHub after a Push is executed

5.2. Making More Changes and Commits

You now know the basics, from making a repository to committing and pushing new changes. You can practice more by repeating the same steps:

1. Make Additional Changes:
   Modify files or create new ones in your local repository as needed.
2. Check the Status:
   Command:
   ```
   git status
   ```
   Description: This will show the modified and untracked files.

3. Add Changes to Staging Area:
   Command:
   ```
   git add .
   ```
   Description: This adds all changes to the staging area, preparing them for a commit.
4. Commit the Changes:
   Command:
   ```
   git commit -m "Describe the changes made"
   ```
   Description: This will commit the changes with a message describing what was done.
5. Push the Changes:
   Command:
   ```
   git push
   ```
   Description: This pushes the new changes to the remote repository on GitHub.

Repeat these steps as needed whenever you make changes to your project. With each cycle, your changes will be tracked, committed, and stored both locally and remotely, ensuring a safe and collaborative environment for your projects.

6. Conclusion

Congratulations on mastering the fundamentals of Git! You've journeyed from setting up Git and GitHub, creating and cloning repositories, to making, committing, and pushing changes. These steps form the backbone of effective version control systems, which are instrumental in facilitating seamless teamwork and robust project management, while ensuring the enduring security and integrity of your projects.

Your initiation into the world of Git has equipped you with a solid foundation. The pathway to advanced proficiency is laid with the bricks of continuous learning and unhesitant experimentation. Do not fear mistakes; consider them stepping stones towards mastering Git. Beyond these essentials, delve into the realms of branching, merging, and other advanced Git features to augment your version control skills further.

In your ongoing Git adventure, keep this manual as a handy companion. Refer back to it whenever you find yourself in the mazes of commands and operations. Your next destination awaits – continue exploring and utilizing the expansive tools and possibilities within Git to bolster your coding journey.

Happy coding!

7. Examples of Some GitHub/GitLab Repositories:

Below is an example of how I used Git through GitLab (a GitHub equivalent) for COM S 309, a group project.

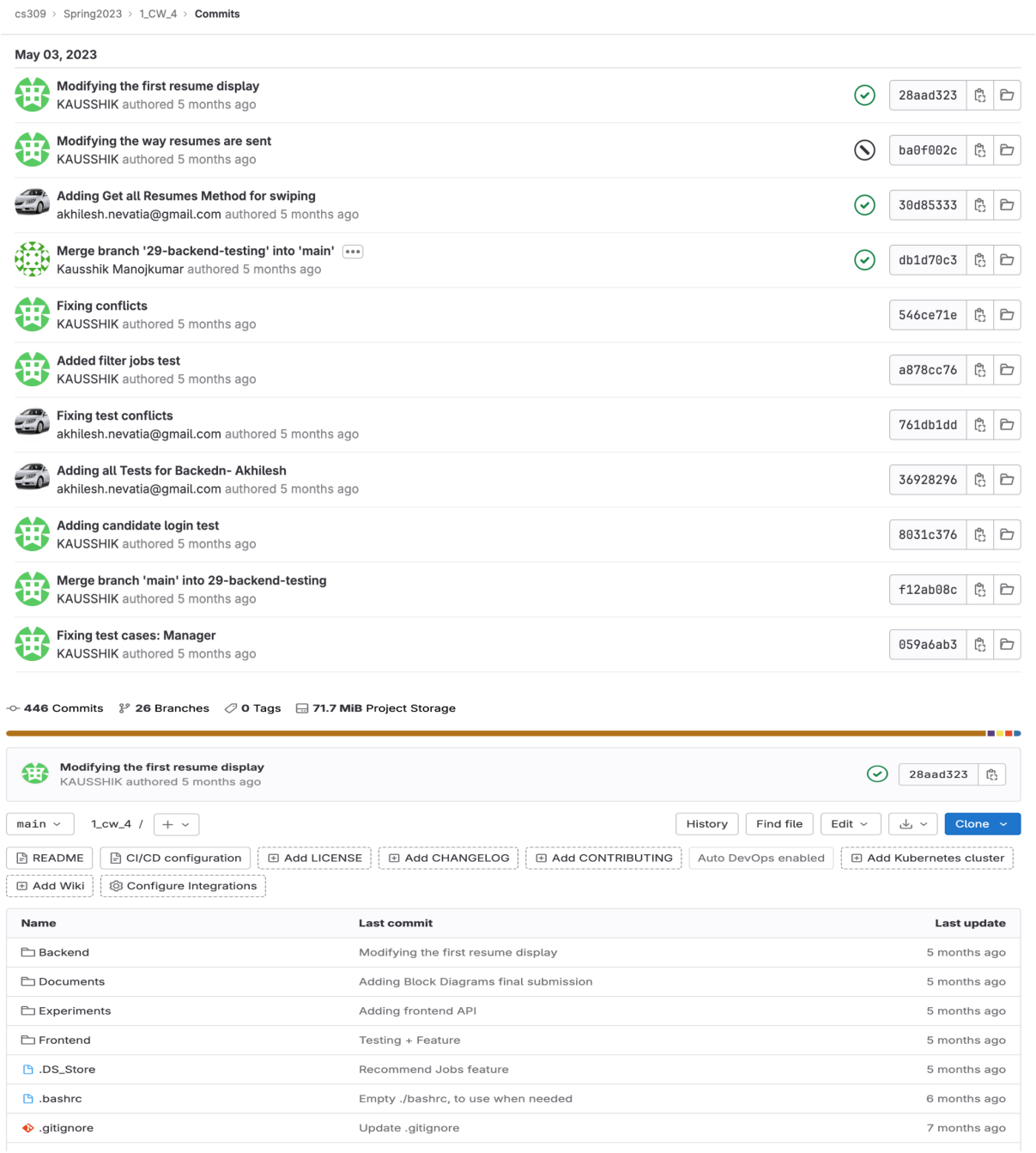

Fig. 20: GitLab Commit History from author's COMS309 Class

8. Works Citation and References:

1. All images are personal screenshots from my work, code, and GitHub.
2. Fig. 3:
   Graphical Table explaining the difference between Git and GitHub. *Https://Andersenlab.Org/Dry-Guide/2022-03-09/Github/*.
3. Fig. 2:
   Visualizing Git, git-school.github.io/visualizing-git/. Accessed 2 Oct. 2023.

4. Martin, Andrew C.R. C.R. "An Introduction to Git and GitHub." (http://www.bioinf.org.uk/teaching/splats/git.pdf)