

# How to get started in Git? For Technical Writers using Windows

## Introduction

GitHub and Git are version control systems used primarily by developers to track changes made in collaborative code. The keyword in this statement is “developers” - Git and GitHub are developer tools, and this quality is evident in the usage of terminal commands and code repositories. Thus, using Git for technical writing may seem daunting and even rather unintuitive. However, since technical writing is often so closely intertwined with development teams, many companies and open-source projects now expect that technical writers also utilize Git version control for documentation. Since it is such a developer-oriented space, seeing the interface for the first time may make you want to throw your hands up and take up copywriting instead. However, this guide is designed to simplify the learning process and make it as quick and painless as possible. The universe only knows how much time you’ve wasted trying to understand every piece of tech thrown at you thus far- this guide compresses only what you need to know to get into the swing of things. In this guide, I will cover how to set up Git on your computer, how to clone a repo from GitHub onto your local machine, how to commit and add changes, and how to push changes made on a local machine back to a remote repo, all in a timely manner.

Before you begin this tutorial, here is a list of prerequisites that you should have downloaded/completed.

**[Microsoft VS Code](#):** Standard text editor. This is where you will be using the terminal from. You can also use the terminal directly through Git Bash, but for the sake of convenience, you’ll only be using VS Code.

**[Git and Git Bash](#):** Git is the version control system used on your local machine to track changes on files. Git Bash is an application made for Microsoft Windows environments that allows users to interact with Git through the terminal.

**A [GitHub Account](#):** This is where your remote repository will be. From here, you will create a repo, clone it to your local machine, and then have it pushed from your local machine back to the remote repo along with any changes made.

**Proficiency in a markup language such as [Markdown](#) or [AsciiDoc](#):** Since Git and GitHub are developer environments, you will primarily be working with markup language files. Markup languages are text-encoding systems that structure and format text through the use of certain symbols. These languages automatically format the documentation that will then be added to repositories.

The installations/requirements for these are relatively straight forward, so I won’t go into much detail on them. Once you have all three set up, it is time to start the guide.

Now, a brief crash course into what each term means.

### What are Git and GitHub used for? What is the difference between them?

Git is a version control system used on your computer. When you are making changes to files on your computer, these changes will be tracked by Git. This is useful in developer environments when code

must be branched off and edited by individual members. GitHub is a cloud hub for repos. You interact between these two, cloning and pushing repos back and forth, using Git commands typed in the terminal.

### What is a terminal?

A terminal, also known as a CLI (Command Line Interface), is an interface in which you can type in and execute text-based commands. This will be your primary environment for using Git.

### What is a repository? A directory?

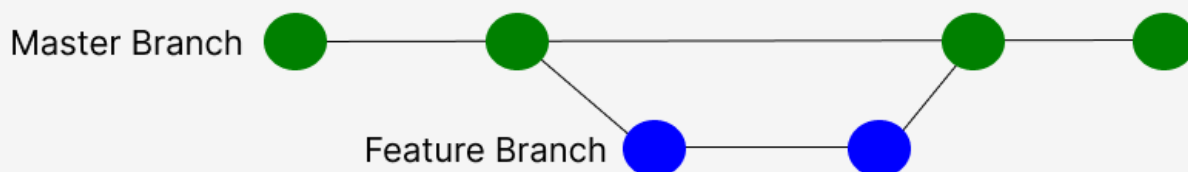
A repository, more commonly referenced as a repo, can be imagined as a place that stores your code. A directory is essentially a file folder on your hard disc for your code. Directories work hierarchically in Git, meaning that you must switch between directories in order to track changes for certain files. This will be important later when committing changes. Directories will be displayed and controlled via the terminal.

### What is an SSH key?

SSH keys are a Unix security feature that allow computers to communicate remotely over insecure networks. SSH keys encrypt your data and facilitate data transfer and network communication. This guide will also cover how to generate your SSH key via the terminal, as it required for file transfers between your local machine and GitHub.

### Commits? Branches?

Committing and branching are two fundamental functions in the Git and GitHub version control environment. Branching allows for simultaneous collaboration on projects. Each project starts out with a master/main branch. The main branch comprises the original source code, and the action of branching makes a copy of the master branch while preserving the original code in the master branch. The separate branch is then worked on and later merged back into the master branch. This differs from cloning/copying repositories in GitHub, in which the entire repository is copied and then worked on in your local machine, with changes being pushed back up to the remote repository. The graphic below visualizes the workflow of a Git repository with a feature branch. As you can see, the branch retains the source code but splits off from the master branch and is then merged back in as an update or patch.



Commits simply refer to changes tracked and saved within Git. For example, if you were to change a line of text, you would track the change and commit it using Git commands to ensure that the change is logged on the file. The changes are then visible in new versions of the file.

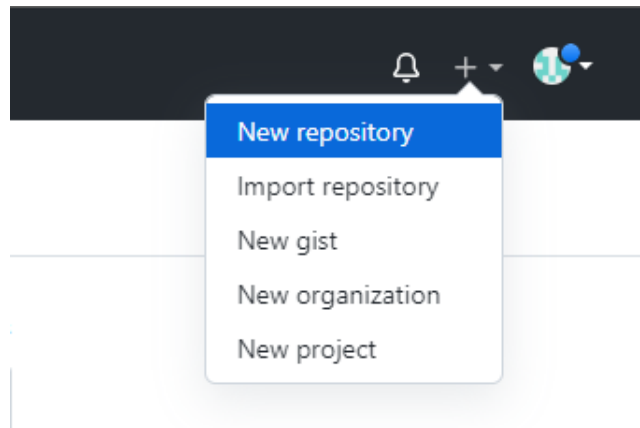
## Main Guide

For the purposes of this tutorial, you will be creating the initial repo on GitHub and then cloning it onto your local machine, where you can make changes and track them with Git. You will then push the

changes made on your local machine back to GitHub. Basically, remote repo → local machine → remote repo.

## Creating a repo



You'll start off on GitHub. On your GitHub Account, in your profile section, select create new repo from the top right menu. You can name this repo whatever you like.



### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner \*

 / Test3 


Great repository names are short and memorable. Need inspiration? How about [automatic-journey?](#)

Description (optional)

Test 3 For Git Demo

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

**Add .gitignore**


Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None** ▼

**Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

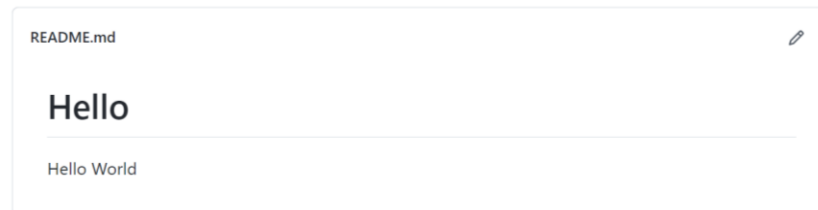
License: **None** ▼

 You are creating a public repository in your personal account.

**Create repository**

Here, you can see in the example that the Repository name is **Test3**.

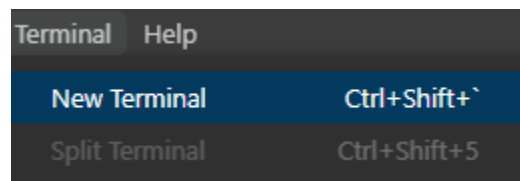
After creating the repo, you'll add a markdown file named **README.md** that you will later edit on our local machine. Select "**creating a new file**," name the file **README.md**, and add some text with a header in a markup language. Once you are finished, select **commit changes** at the bottom. This will commit the changes to GitHub.



The README.md file as it appears on GitHub, formatted with Markdown. This will be the file you edit on your local machine.

### Generating an SSH key

Before you can clone this repo onto our local machine, you must verify to GitHub that you do in fact own your local machine. This may sound rather unintuitive, but this is where the SSH key security protocol comes into play. Open VS Code, and from the top ribbon, select **Terminal**, and then **New Terminal**.



- 1.) The terminal will display in the bottom portion of the VS Code Interface. Now, in the terminal, you are going to type in the command below to generate your SSH Key. At the end of this key, it is critical that you type in the same email that you used for your GitHub account.

**ssh-keygen -t rsa -b 4096 -C "email@example.com"**

- 2.) The next line will prompt you to enter a file in which to save the generated key. It is suggested to find the directory referenced in this line beforehand and enter a filename that will be easy to access. For the sake of this example, I will type in "Test 3" as the file to save the key.
- 3.) Afterwards, it will ask you to enter a passphrase. This is optional, and you can simply hit enter to skip this if you prefer. Once you enter/skip the passphrase, your SSH key will have been generated in the file location you added.

### Steps 1-3:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ssh-keygen + v [ ] [X] ^ X
```

Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/powershell>

```
PS C:\Code Repository\Code Bin\reptest\Test 3> ssh-keygen -t rsa -b 4096 -C "email@example.com"
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\user\Documents\.ssh\id_rsa): Test3
Enter passphrase (empty for no passphrase):
```

- 4.) If you have entered a custom file name, type in the following to switch to the directory in which the file has been stored (*YourUser* stands for whatever the username is on your computer). Otherwise, your file will be stored as a .pub file in the .ssh folder in your “Users” directory.

```
cd C:\Users\YourUser/
```

- 5.) Type in the following to list all of your generated keygens in this location. The .pub file represents your SSH public key that you will paste to GitHub in order to link your local machine with GitHub.

Is **C:\Users\YourUser/\*.pub**

```
PS C:\Code Repository\Code Bin\Portfolio> ls C:\Users\Bart\*/*.pub

Directory: C:\Users\Bart\

Mode                LastWriteTime         Length Name
----                -
-a----           10/31/2022  12:21 AM             576 Test3.pub
```

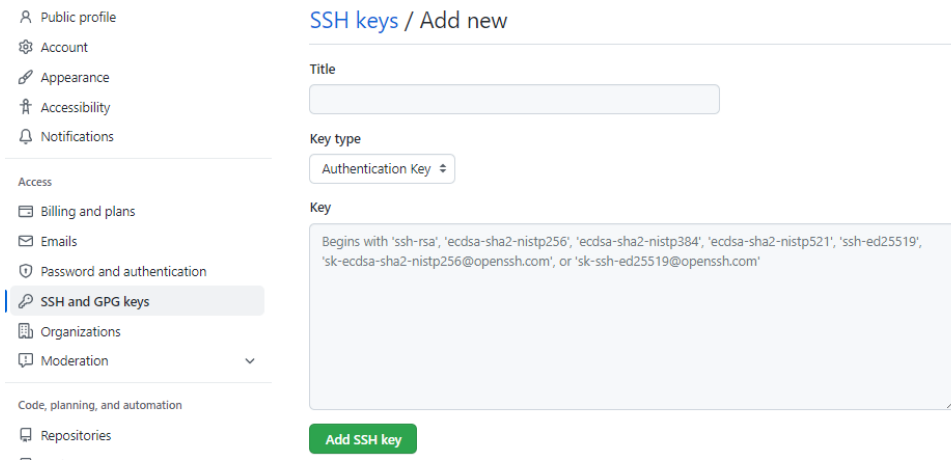
- 6.) Finally, type in the following to view your generated key (*YourSSHKey* is a stand-in for whatever you named your file when you generated the key in Step 2. It should be listed when typing in the previous command).

```
cat C:\Users\YourUser\YourSSHKey.pub
```

```
PS C:\Code Repository\Code Bin\Portfolio> cat C:\Users\Himanshu\I\Test3.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDAK...
PS C:\Code Repository\Code Bin\Portfolio>
```

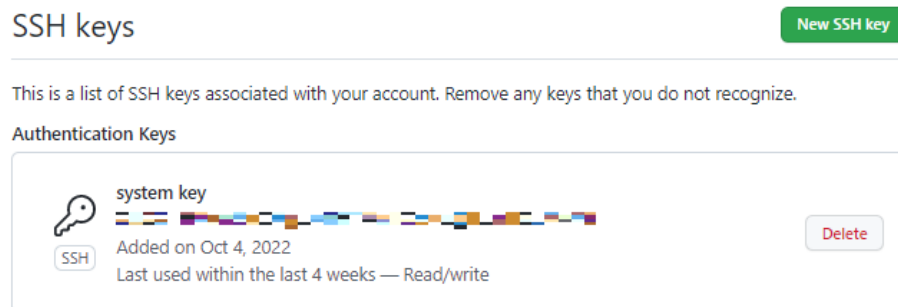
The blurred-out text in the above screenshot represents the generated key. Do not share this key with anyone.

- 7.) Copy the key that is listed in the terminal. Now, back to GitHub. Go to settings and hit the **SSH and GPG keys** tab. Paste your key into the space, and you're connected!



The screenshot shows the GitHub 'SSH keys / Add new' page. On the left is a sidebar with navigation links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, SSH and GPG keys (highlighted), Organizations, Moderation, Code, planning, and automation, and Repositories. The main form has a 'Title' input field, a 'Key type' dropdown set to 'Authentication Key', and a 'Key' text area containing a list of valid key formats: 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', and 'sk-ssh-ed25519@openssh.com'. A green 'Add SSH key' button is at the bottom.

Once you've added your SSH to your GitHub Account, it will appear on your profile like this:



The screenshot shows the 'SSH keys' page on a GitHub profile. It features a 'New SSH key' button in the top right. Below the header, a message states: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' Under the 'Authentication Keys' section, a single key is listed: 'system key'. It includes a key icon, a visual representation of the key, the text 'Added on Oct 4, 2022', 'Last used within the last 4 weeks — Read/write', and a 'Delete' button.

## Using Git

Before you dive into the fun of Git commands, I'll introduce a set of Git commands to be typed into the terminal. This list is by no means comprehensive but is enough to get you started for the purposes of this guide. Come back to this list often or have it written down as you continue the tutorial. This is one of those things that only practice improves. Git stops being useful and fun when you use commands in the wrong context and errors start flying at you from the terminal. It is important to make sure you use each command in the right context and to be proactive when you encounter issues.

**git status:** lists all the files that have been altered but have not yet been saved in a commit.

**git commit:** A command that consolidates changes into repo you are working on. Must be followed by a message.

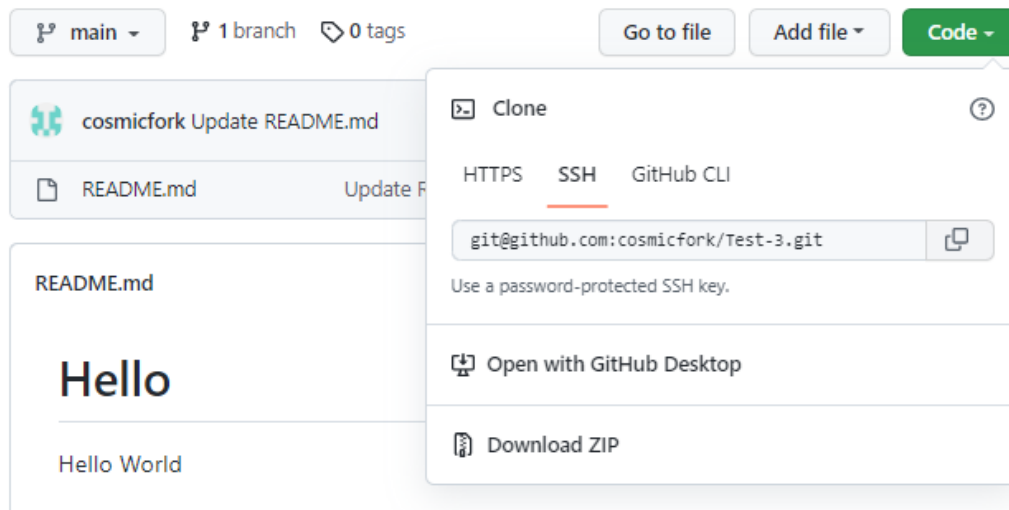
**git clone:** allows for remote repos to be added to local machines.

**git push:** pushes the repo (along with all local commits) back to the remote repo.

**git add (.):** Tracks selected files that are to be staged for a commit. If a "." is added after, all untracked files are tracked and staged for commit.

**cd:** changes the file directory in the terminal.

Next, go back to the repo you made, and on the green code button, copy the SSH link presented. Then, return to the terminal and type in the command "git clone" along with the link you've copied.



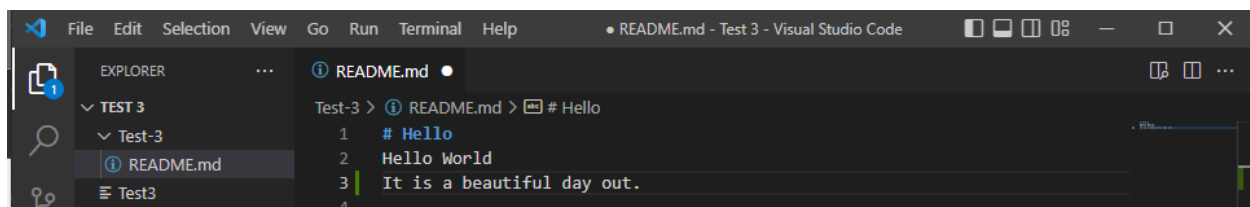
```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Code Repository\Code Bin\repotest\Test 3> git clone git@github.com:cosmicfork/Test-3.git
Cloning into 'Test-3'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (3/3), done.
Receiving objects: 100% (8/8), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
PS C:\Code Repository\Code Bin\repotest\Test 3> 
```

The git clone command

The README.md file that you created earlier in GitHub should now be visible in the file explorer on the left. Go ahead and click on the file to make some edits. Each addition/deletion will show a green or red line on the side, respectively. These represent changes made to the cloned repo.



Before you can track the changes, you must change into the directory that has been cloned. (*nameofyourrepo* is a stand in for whatever your repo file is called).

**cd nameofyourrepo/**

Type in **git status** in the terminal to see what changes have been made but not yet saved. If no changes have been made, you will receive the output *“Your branch is up to date with ‘origin/main’”*. Once you have made changes, you will receive an output that lists your untracked changes in red font.

```
PS C:\Code Repository\Code Bin\repotest\TestRepoTest> cd Test-3
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3> git status
On branch main
Your branch is up to date with 'origin/main'.
```

```
PS C:\Code Repository\Code Bin\repotest\Test 3> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  Test-3/

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Code Repository\Code Bin\repotest\Test 3>
```

In order to track these changes, type in the command **git add .** to track all the changes. The **“.”** in the command signifies that all changes will be tracked; if you want to track specific files, use the **git add** command along with the file name. Now, when you use the **git status** command, it will display the tracked changes that are not yet staged for the commit.

```
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3> git add .
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3>
```

To commit the changes, you must type in **git commit -m “message”** (*message* is a stand-in for whatever title you may want to add). You can add another **-m** after the first to add a description. The purpose of these is to give descriptions to commits for other users to see, so both messages will display on GitHub after you push the document back to the remote repository. After committing the change, you should see the output *“1 file changed, 1 insertion(+)”*. This refers to the changes tracked on your local machine, where you added some text to the document. This output will vary depending on how many insertions or deletions you made.

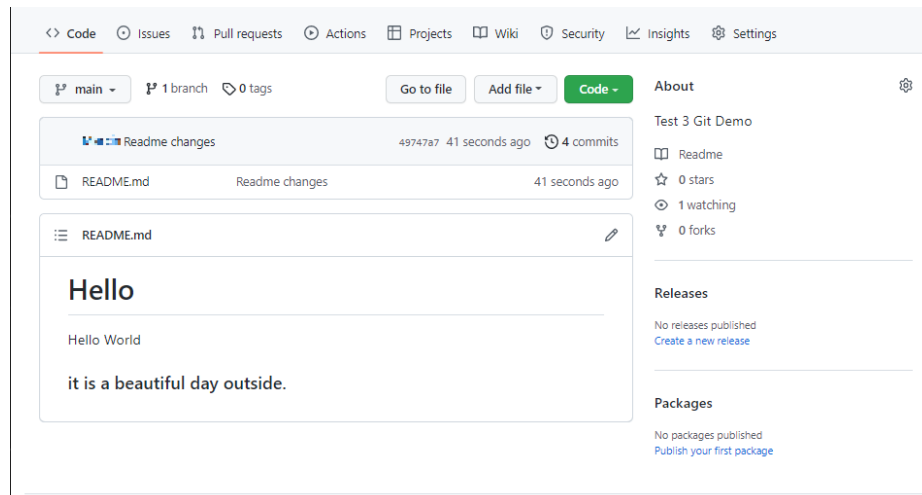
```
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3> git commit -m "Readme changes"
[main 49747a7] Readme changes
1 file changed, 1 insertion(+)
```

Finally, to push this file back to GitHub, type in **git push origin main**. Origin represents the location of our Git repository, and main is the branch that you are merging to in GitHub.

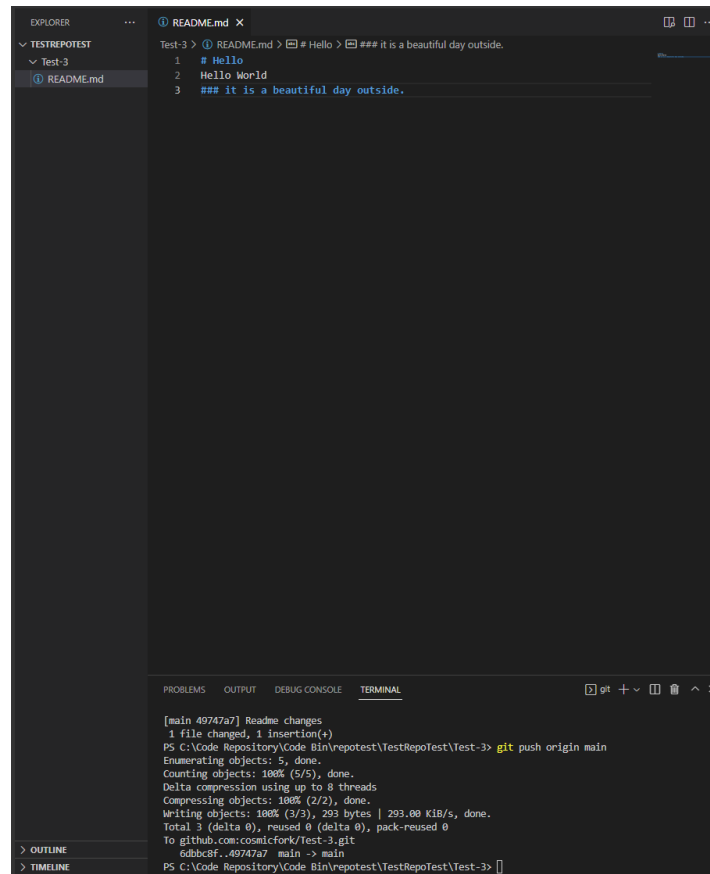


```
PS C:\Code Repository\Code Bin\reptest\TestRepoTest\Test-3> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:cosmicfork/Test-3.git
6dbbc8f..49747a7  main -> main
```

Once you go back to GitHub, you should see all of the changes you've committed on Git visible on GitHub.



The changes that were made on Git are now visible on your remote repo on GitHub.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project named 'TESTREPOTEST' with a subdirectory 'Test-3' containing a file 'README.md'. The main editor area displays the content of 'README.md', which contains three lines of text: '# Hello', 'Hello World', and '### it is a beautiful day outside.'. At the bottom, the TERMINAL panel is active, showing the output of a 'git push' command. The output indicates that one file was changed and one insertion was made, and the push was successful, creating a new commit on the 'main' branch of the remote repository 'github.com:cosmicfork/Test-3.git'.

```
Test-3 > README.md > # Hello > ## it is a beautiful day outside.
1 # Hello
2 Hello World
3 ### it is a beautiful day outside.

[main 49747a7] Readme changes
1 file changed, 1 insertion(+)
PS C:\Code Repository\Code Bin\reptest\TestRepoTest\Test-3> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:cosmicfork/Test-3.git
6dbb8f1..49747a7 main -> main
PS C:\Code Repository\Code Bin\reptest\TestRepoTest\Test-3>
```

How the changes made on the local machine appear on VSCode.

That's all there is to it. Well, not really; you haven't yet gotten to branching or initializing repos on your local machine, and Git gets a lot more complicated than this. Hopefully, however, this brief, simplified guide was able to illustrate how to start out with Git and how to simply edit files before pushing them back on to remote repos. This foundation is necessary for the more complicated parts later. It is suggested to practice often with these commands and to become comfortable with the terminal before merging/pulling branches.

## Troubleshooting

If your files are visible on GitHub but the changes you made are not, try saving on VS Code before adding and committing changes. This should allow all changes to be visible on GitHub after a push.

If your changes are not saving after a commit, make sure you are in the right directory. You can switch directories with the **cd** command.

Make sure you are not confusing the commands **git add** with **git commit** or forgetting to use either in the correct circumstances. Remember, **git add** tracks which files have been changed and are staged for the commit; **git commit**, done after **git add**, saves the changes to the branch.

```
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3> git add .
PS C:\Code Repository\Code Bin\repotest\TestRepoTest\Test-3> git commit -m "Readme changes"
[main 49747a7] Readme changes
1 file changed, 1 insertion(+)
```

If you are receiving a **fatal: not a git repository (or any of the parent directories): .git** message when typing in the **git status** command, type in **git init** at the root directory to initialize a repository. This should solve the issue and allow for you to add and track changes made in your repo.

When typing in commands, make sure that you are in the correct directory in the terminal. The directory below is listed as `C:\Code Repository\Code Bin\repotest\Test 3>`. The directory displayed will be the directory in which commands will be applied. Use the **cd** command to change directories. For example, to switch to the `repotest` directory, you would type in `cd C:\Code Repository\Code Bin\repotest` (Note that file names will differ for every computer; your computer's files names will be different from the example).

```
PS C:\Code Repository\Code Bin\repotest\Test 3> ssh-keygen -t rsa -b 4096 -C "email@example.com"
```