**Table of Contents**

* [**What is a Version Control System**](https://www.softwaretestingmaterial.com/git-tutorial/#h-what-is-a-version-control-system)
* [**Types of Version Control Systems**](https://www.softwaretestingmaterial.com/git-tutorial/#h-types-of-version-control-systems)
* [**Local version control systems:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-local-version-control-systems)
* [**Centralized version control systems:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-centralized-version-control-systems)
* [**Distributed version control systems:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-distributed-version-control-systems)
* [**What is Git?**](https://www.softwaretestingmaterial.com/git-tutorial/#h-what-is-git)
* [**Git Installation**](https://www.softwaretestingmaterial.com/git-tutorial/#h-git-installation)
* [**What is GitHub and why use GitHub?**](https://www.softwaretestingmaterial.com/git-tutorial/#h-what-is-github-and-why-use-github)
* [**Git vs GitHub:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-git-vs-github)
* [**Git Lifecycle**](https://www.softwaretestingmaterial.com/git-tutorial/#h-git-lifecycle)
* [**What is a Repository in GitHub?**](https://www.softwaretestingmaterial.com/git-tutorial/#h-what-is-a-repository-in-github)
* [**How to setup GitHub account:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-how-to-setup-github-account)
* [**How to create a GitHub repository**](https://www.softwaretestingmaterial.com/git-tutorial/#h-how-to-create-a-github-repository)
* [**Fundamental Git commands:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-fundamental-git-commands)
* [**Initializing a Git Repository**](https://www.softwaretestingmaterial.com/git-tutorial/#h-initializing-a-git-repository)
* [**Adding & Removing files:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-adding-removing-files)
* [**Git Commit :**](https://www.softwaretestingmaterial.com/git-tutorial/#h-git-commit)
* [**Cloning and Pushing :**](https://www.softwaretestingmaterial.com/git-tutorial/#h-cloning-and-pushing)
* [**Pushing Remotely**](https://www.softwaretestingmaterial.com/git-tutorial/#h-pushing-remotely)
* [**Pulling**](https://www.softwaretestingmaterial.com/git-tutorial/#h-pulling)
* [**Reset & Checkout:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-reset-checkout)
* [**Branches:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-branches)
* [**Stash:**](https://www.softwaretestingmaterial.com/git-tutorial/#h-stash)
* [**Fetching (git fetch):**](https://www.softwaretestingmaterial.com/git-tutorial/#h-fetching-git-fetch)
* [**Merging (git merge) :**](https://www.softwaretestingmaterial.com/git-tutorial/#h-merging-git-merge)

**Version Control**

In software engineering, version control systems are a category of software tools that help a software team (multiple developers, designers, and team members) to work together on the same project and allow them to manage changes to computer programs, documents, websites, and other information over time.

Version Control Systems maintains all the edits and historic versions of the project. So that teams can recall specific versions later.

**Types of Version Control Systems**

**Local Version Control Systems:**

These store all changes to files in a local database. Examples include RCS (Revision Control System).

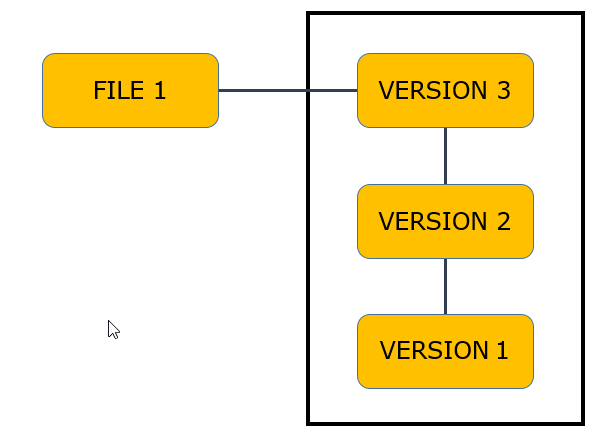
**Centralized Version Control Systems (CVCS):**

These use a single server to store all files and allow multiple clients to check out files. Examples include CVS (Concurrent Versions System) and Subversion (SVN).

**Distributed Version Control Systems (DVCS):**

These clients fully mirror the repository, including its full history. Examples include Git and Mercurial. DVCSs provide better redundancy and collaboration capabilities compared to CVCS.

**Local version control systems:**



### ****Centralized version control systems:****

### 

### ****Distributed version control systems:****

### 

## ****What is Git?****

Git is one of the most popular version control systems. It is one of the distributed version control systems. It is an open-source project which is compatible with many operating systems and IDEs.  
It allows us to track changes in an application, or in a folder, or in a single file over time across different users, and different computers. Git takes a snapshot of all files whenever we create a new commit. Every committer on a project always has a copy of the whole repository on their machine. So, we can commit to the Git file system also in offline mode.

### ****Why is GIT important?****

GIT is important because it allows developers to track changes to code over time, collaborate with others on the same project, and manage multiple project versions. It also provides a robust set of features for branching and merging, which can help teams work more efficiently and avoid conflicts.

### ****What are some common GIT commands?****

Some common Git commands include:  
1. git init: Creates a new Git repository  
2. git add: Adds changes to the staging area in preparation for the next commit.  
3. git commit: Records changes to the repository  
4. git push: Upload changes to a remote repository  
5. git pull: Downloads changes from a remote repository and merge them into the local branch  
6. git branch: Lists or creates branches  
7. git merge: Combine the modifications made in one branch with another branch.  
8. git status: Displays the current state of the local repository.

### Is GIT Centralized or Decentralized?

Git is decentralized. It has a distributed architecture where each user has their own complete copy of the repository. This allows users to work independently, offline if needed, and have access to the full history and files.

Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**.

## How to Use GIT?

GIT can be used through a command line interface (CLI) or a graphical user interface (GUI). Here are the basic steps to using GIT:

* Install Git on your computer.
* Create a new repository.
* Download a copy of the repository to your local computer using “git clone”.
* Edit the files in the repository to make changes.
* Stage the changes for the next commit using “git add”.
* Commit the changes to the repository with a descriptive message using “git commit”.
* Upload the changes to the main repository with “git push”.



## ****What is GitHub and why use GitHub?****

GitHub is a cloud-based Git repository hosting service. It lets individuals and teams work together on projects.

Git is a command-line tool whereas GitHub comes with a web-based graphical interface.

GitHub is an application allowing you to store remote repositories on their servers. It also provides a user-friendly platform to interact with and manage your repositories. It is a public platform that allows millions of users to share their projects with the world.

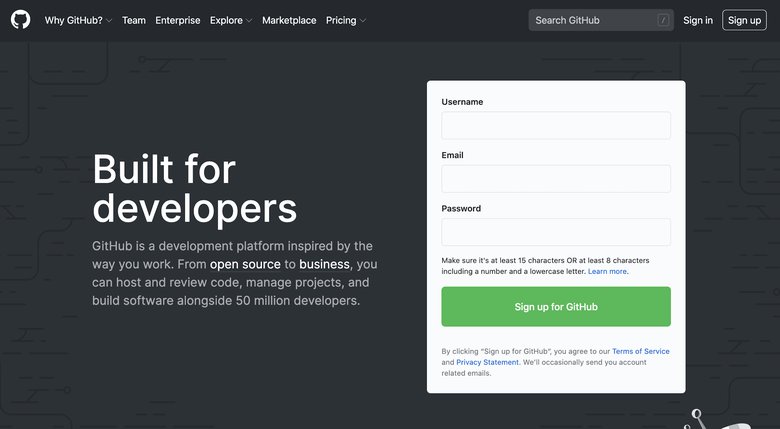
The main purpose is to allow people to collaborate together to build projects. But it’s not just limited to collaboration. On top of that, it can be used as a portfolio for your best work. One of the most useful features of GitHub is being able to access your repository from any location. Also, it’s an industry standard for hosting Git repositories.

**GitHub**

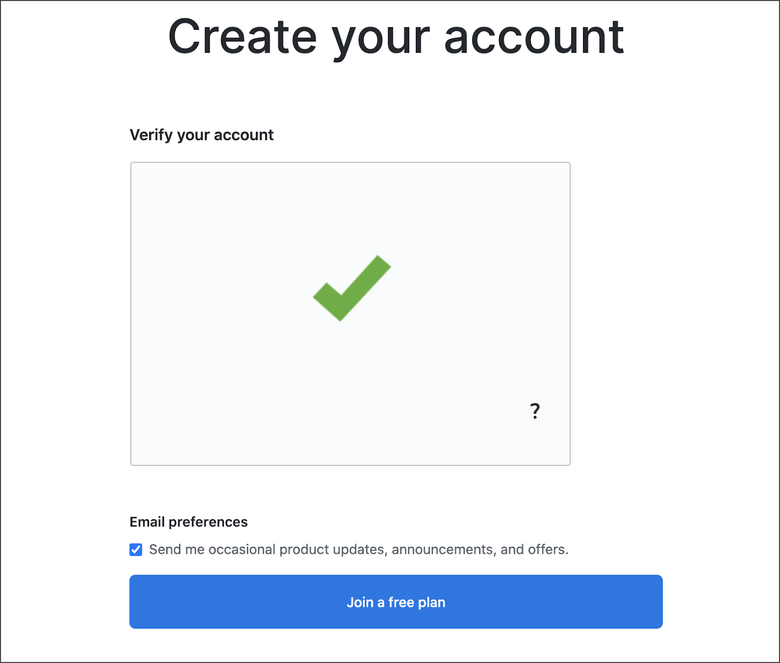
GitHub is a web-based platform that uses Git for version control. It is particularly valuable for software testers for several reasons:

## ****How to setup GitHub account:****

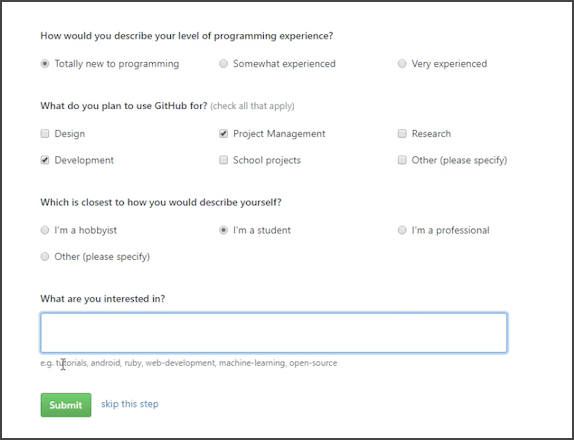
1. Go to [https://github.com](https://github.com/) and **signup** for a GitHub account using username, email, and password.



2. Once you fill out your information, you will see the verify account screen. Verify the captcha and click the Join a free plan button.



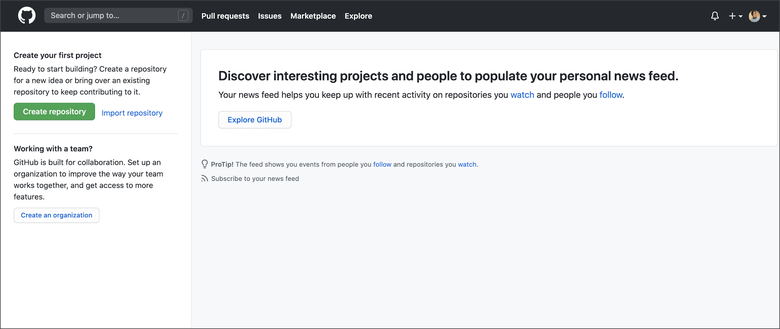
3. Now you need to fill out some other information related to programming experience and kind of work etc.



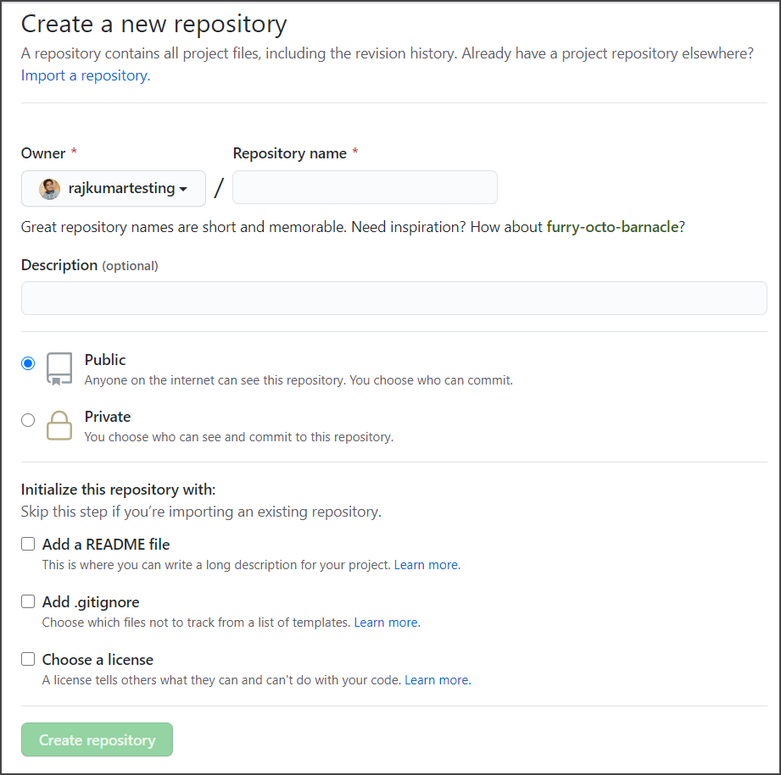
4.Now your GitHub account is created.

## ****How to create a GitHub repository****

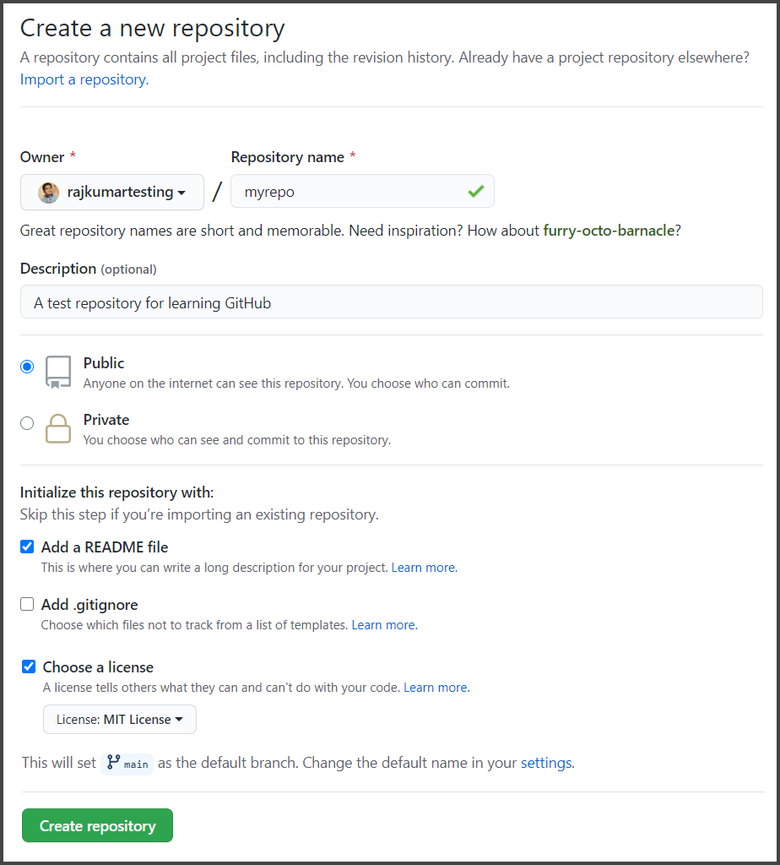
1. Go to the GitHub home page.



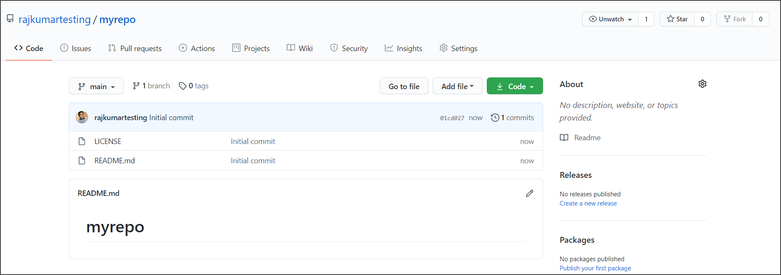
2. Click on Create repository. Add repo name, description and you can choose Private or Public. You can’t choose Private unless you have a paid plan.



3. For now, let’s initialize this repository “**myrepo**” with a README file. You can add a license too. There are many different licenses for different things. For now, let’s choose an MIT license because this is the license that allows people to freely distribute it and copy it however they want.



4. This is what a repository looks like. There are a bunch of things we can see. There are a number of total commits, then we have the number of branches, number of releases, contributors, and license.



5. There are a bunch of other tabs at the top of the page.

**Issues:** Issues can be labeled by contributors. Anyone who is working on your project can actually find an issue and list it. What you and other contributors can do is go and look through and find these issues and try and fix them. So it’s a really useful feature on Github.

**Pull Request:** To request access to pull your project onto their local machine.

**Projects:** This is just a bunch of workflow management. Projects can be managed in the same place you keep your code.

**Wiki:** It is actually a bigger place to layout something like a summary. It’s just a kind of quick and concise place to list your project. You can list documentation of some of the application libraries or other things.

**Insights:** Insights allow you to see a lot of different features of your repository. It is just like Google Analytics or any other analytics program. It includes merged pull requests, open pull requests, and a bunch of different metrics.

**Settings:** In settings, we can change some of the things about our repository. You can change the name of your repo, delete your repository. You can invite collaborators from here. There is an option to switch between the default branches and other branches.

## ****Fundamental Git commands:****

**1. git –help:** When this command is hit in our terminal, Git will give a list of its most common commands. If it is followed by a specific command then it will give you a full detailed description of that particular command.

## ****Initializing a Git Repository****

To start using Git, the first step is to initialize a repository, a location for files, and their revision history.

Let’s create a new repository now. Create a folder or a directory “**MySoftware**”. Then switch to that directory. Once inside the folder, use the git init command to convert the directory into an empty Git repository.

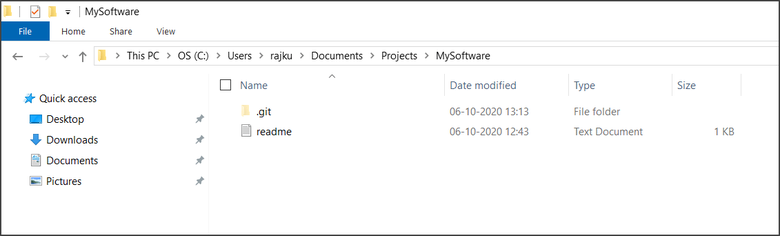
**git init**

## ****Adding & Removing files:****

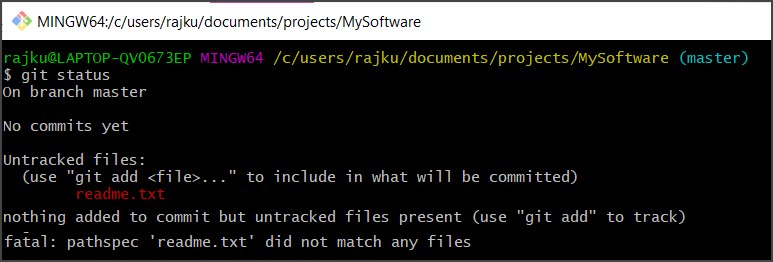
Now we are going to be working with the staging area. In general, files in a repository can have the following statuses:

1. Not tracked  
2. Staged  
3. Committed

To find out the actual status of files in a repository, the **git status** command is used. Let’s say we have created a file ‘readme.txt’ in our repository.

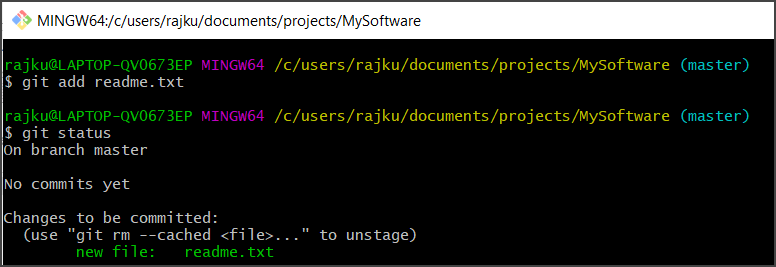


– Running **git status** will have the following output:



– As you can see the file **readme.txt** is untracked. Untracked means that the file is not being tracked by Git for changes. We should explicitly say which files it should follow/track.

– The **git add ‘file name’** command tells Git to track the file. This step is called **staging**. Let’s add our readme.txt file:



– Now, our file is in the staging area. Let’s say we have more untracked files. If now we will do ***git status***, we can see there are untracked files. To add all files in the staging area together, we use “***git add***.” command

## ****Git Commit :****

The **git commit** command saves the state of your project by adding snapshots of staged files to the repository. This command includes the **-m** flag with a message describing what we have changed.

The command is: ***git commit -m “message”***

Executing **git status** after a git commit verifies that tracked files are up to date. Only changes to staged files(added using the **git add** command) will be added to the repository with the **git commit** command.

### ****Pushing Remotely****

After making our local changes and commits, it’s time to push the changes to the remote repository. The push command lets you transfer commits from local to the remote repository.

***git push -u origin master***

### ****Pulling****

**git pull** command simply downloads the latest content from the remote repository and copies the latest changes to the local repository.

The command to pull new changes on our local repository is:

***git pull origin master***

