GITHUB

Version Control System (VCS):

In Software Engineering, **Version Control System (also known as Revision Control, Source Control or Source Code Management)** is a class of systems responsible for managing changes to computer programs, documents, large web sites, or other collections of information.

What is a "Version Control System"?

Version Control Systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

Why Version Control System is so Important?

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So in order to contribute to the product, they made modifications to the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

Basically Version Control System keeps track on changes made on a particular software and take a snapshot of every modification. Let's suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as the version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version.

Benefits of the version control system:

- Enhances the project development speed by providing efficient collaboration,
- Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
- Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
- Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this **VCS,**
- For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is **Git, Helix core, Microsoft TFS,**
- Helps in recovery in case of any disaster or contingent situation,
- Informs us about Who, What, When, Why changes have been made.

Use of Version Control System:

- **A repository:** It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
- **Copy of Work (sometimes called as checkout):** It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.
- **Working in a group:** Consider yourself working in a company where you are asked to work on some live project. You can’t change the main code as it is in production, and any change may cause inconvenience to the user, also you are working in a team so you need to collaborate with your team to and adapt their changes. Version control helps you with the, merging different requests to main repository without making any undesirable changes. You may test the functionalities without putting it live, and you don’t need to download and set up each time, just pull the changes and do the changes, test it and merge it back. It may be visualized as.

Types of Version Control Systems:

1. *Local Version Control Systems (LVCS)*
2. *Centralized Version Control Systems (CVCS)*
3. *Distributed Version Control Systems (DVCS)*

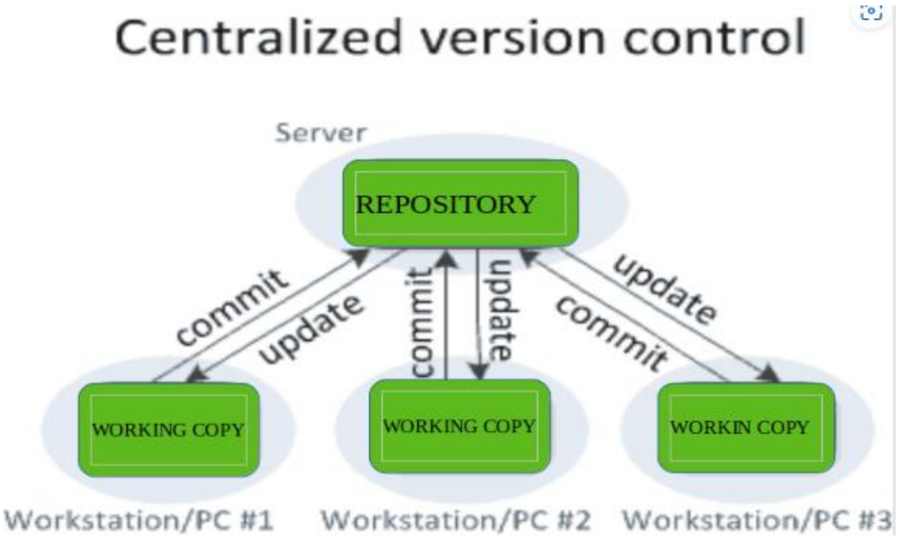
1. **Local Version Control Systems (LVCS):** It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special

format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

2. **Centralized Version Control Systems (CVCS):** Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

   Two things are required to make your changes visible to others which are:

   - ***You commit***
   - ***They update***



The **benefit** of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

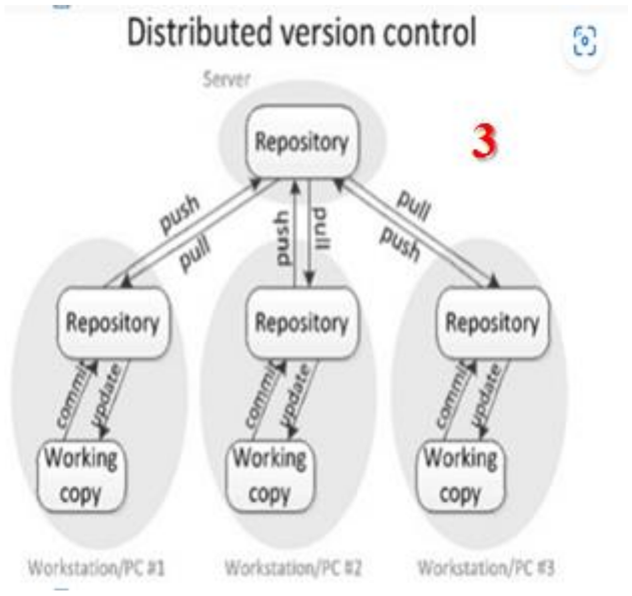
It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.

3. **Distributed Version Control Systems (DVCS):** Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your

changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

- ***You commit***
- ***You push***
- ***They pull***
- ***They update***



The most popular distributed version control systems are **Git, and Mercurial**. They help us overcome the problem of single point of failure.

Purpose of Version Control:

- ✓ Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.
- ✓ It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- ✓ It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line

of a file, then human assistance is requested by the version control system in deciding what should be done.

- ✓ Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

What is Git?

- ✓ Git is a **free and open-source distributed version control system** designed to handle everything from small to very large projects with speed and efficiency.
- ✓ Git relies on the basis of distributed development of software where more than one developer may have access to the source code of a specific application and can modify changes to it that may be seen by other developers.
- ✓ Initially designed and developed by **Linus Torvalds** for Linux kernel development in **2005.**
- ✓ Every git working directory is a full-fledged repository with complete history and full version tracking capabilities, independent of network access or a central server.
- ✓ Git allows a team of people to work together, all using the same files. And it helps the team cope with the confusion that tends to happen when multiple people are editing the same files.

Characteristics of Git:

A. Strong support for non-linear development

- ✓ Git supports rapid branching and merging and includes specific tools for visualizing and navigating a non-linear development history. A major assumption in Git is that a change will be merged more often than it is written.
- ✓ **Branches** in Git **are very lightweight**.

B. Distributed development

- ✓ Git **provides** each developer **a local copy** of the entire development history, and changes are copied from one such repository to another.
- ✓ The changes can be merged in the same way as a locally developed branch very efficiently and effectively.

C. Compatibility with existing systems/protocol

- ✓ Git has a CVS server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories.

D. Efficient handling of large projects

- ✓ Git is very fast and scalable compared to other version control systems.
- ✓ The fetching power from a local repository is much faster than is possible with a remote server.

E. Data Assurance

- ✓ The Git history is stored in such a way that the ID of a particular version depends upon the complete development history leading up to that commit.
- ✓ Once published, it is not possible to change the old versions without them being noticed.

F. Automatic Garbage Collection

- ✓ Git automatically performs garbage collection when enough loose objects have been created in the repository.
- ✓ Garbage collection can be called explicitly using git gc –prune.

G. Periodic explicit object packing

- ✓ Git stores each newly created object as a separate file. It uses packs that store a large number of objects in a single file (or network byte stream) called packfile, delta-compressed among themselves.
- ✓ A corresponding index file is created for each pack file, specifying the offset of each object in the packfile.
- ✓ The process of packing can be very expensive computationally.
- ✓ Git allows the expensive pack operation to be deferred until later when time does not matter.
- ✓ Git does periodic repacking automatically but manual repacking can be done with the git gc command.

What is GitHub?

GitHub is an immense platform for code hosting. It supports version controlling and collaboration and allows developers to work together on projects. It offers both distributed version control and source code management (SCM) functionality of Git. It also facilitates collaboration features such as bug tracking, feature requests, task management for every project.

- Development of the **GitHub.com** platform began on **October 19, 2007**.
- The site was launched in **April 2008 by Tom Preston-Werner, Chris Wanstrath, P.J Hyett and Scott Chacon** after it had been made available for a few months prior as a beta release.
- GitHub has an annual keynote called GitHub Universe
- **Microsoft acquired GitHub in June 2018**

Essential components of the GitHub are:

- ✓ *Repositories*
- ✓ *Branches*
- ✓ *Commits*
- ✓ *Pull Requests*
- ✓ *Git (the version control tool GitHub is built on)*

Repository:

A GitHub repository can be used to store a development project. It can contain folders and any type of **files** (HTML, CSS, JavaScript, Documents, Data, Images). A GitHub repository should also include a **licence** file and a **README** file about the project. A GitHub repository can also be used to store ideas, or any resources that you want to share.

Branch:

A GitHub branch is used to work with different **versions** of a repository at the same time. By default a repository has a **master** branch (a production branch). Any other branch is a **copy** of the master branch (as it was at a point in time). New Branches are for bug fixes and feature work separate from the master branch. When changes are ready, they can be merged into the master branch. If you make changes to the master branch while working on a new branch, these updates can be pulled in.

Commits:

At GitHub, changes are called commits. Each commit (change) has a description explaining why a change was made.

Pull Requests:

Pull Requests are the heart of GitHub **collaboration**. With a pull request you are **proposing** that your changes should be **merged** (pulled in) with the master. Pull requests show content **differences**, changes, additions, and subtractions in **colors** (green and red).

Note: As soon as you have a commit, you can open a pull request and start a discussion, even before the code is finished.

Advantages of GitHub

GitHub can be separated as the **Git and the Hub**. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows:

- ✓ It is easy to contribute to open source projects via GitHub.
- ✓ It helps to create an excellent document.
- ✓ You can attract the recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
- ✓ It allows your work to get out there in front of the public.
- ✓ You can track changes in your code across versions.

Features of GitHub

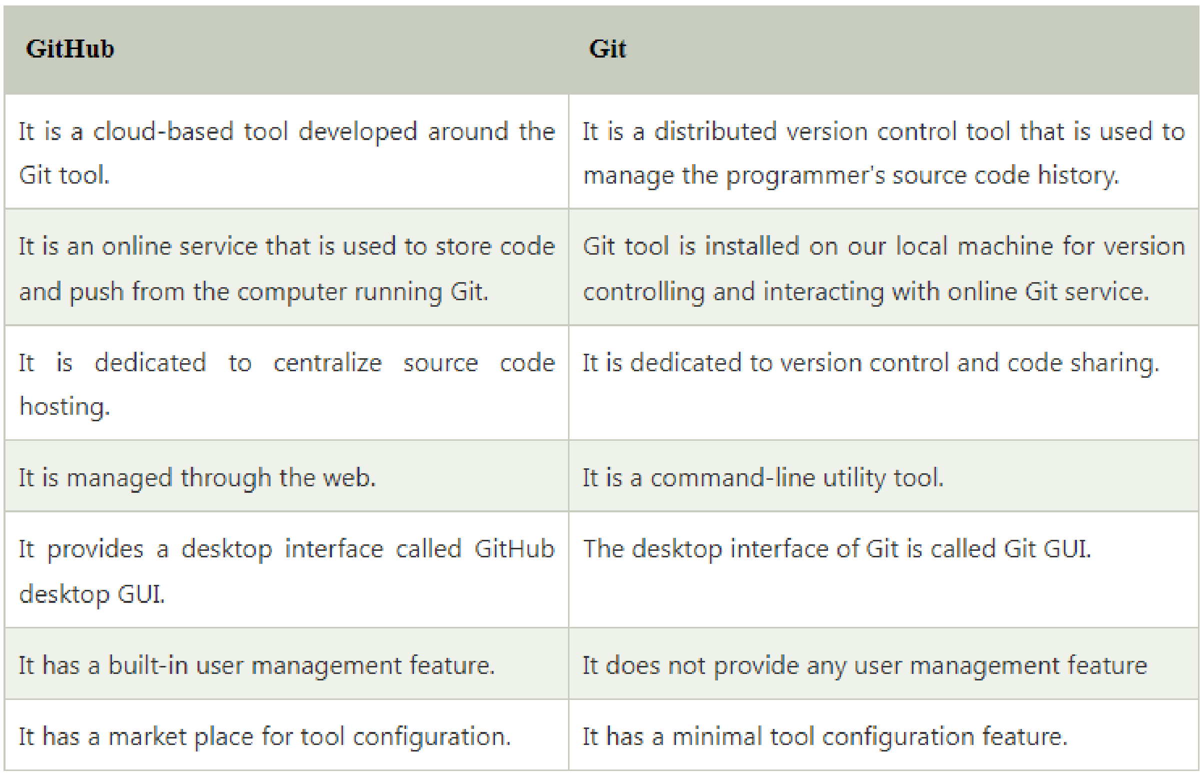
GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open source projects and codes of various programming languages.

Some of its significant features are as follows:

- ✓ *Collaboration*
- ✓ *Integrated issue and bug tracking*
- ✓ *Graphical representation of branches*
- ✓ *Git repositories hosting*
- ✓ *Project management*
- ✓ *Team management*
- ✓ *Code hosting*
- ✓ *Track and assign tasks*
- ✓ *Conversations*

GitHub vs. Git

Git is an open-source distributed version control system that is available for everyone at zero cost. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work among programmers. The version control allows you to track and work together with your team members at the same workspace.



GitHub	Git
It is a cloud-based tool developed around the Git tool.	It is a distributed version control tool that is used to manage the programmer's source code history.
It is an online service that is used to store code and push from the computer running Git.	Git tool is installed on our local machine for version controlling and interacting with online Git service.
It is dedicated to centralize source code hosting.	It is dedicated to version control and code sharing.
It is managed through the web.	It is a command-line utility tool.
It provides a desktop interface called GitHub desktop GUI.	The desktop interface of Git is called Git GUI.
It has a built-in user management feature.	It does not provide any user management feature
It has a market place for tool configuration.	It has a minimal tool configuration feature.

How to Use GitHub?

This question is prevalent for the developers who have never used GitHub. There is nothing to worry about, the necessary steps for the using GitHub are as follows:

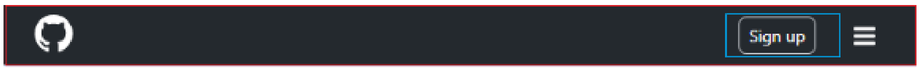
- ✓ ***Create a GitHub account***
- ✓ ***GitHub login***
- ✓ ***GitHub Repository***
- ✓ ***Create a repository***
- ✓ ***Create a file***
- ✓ ***Create Branches***

1. Create a GitHub Account

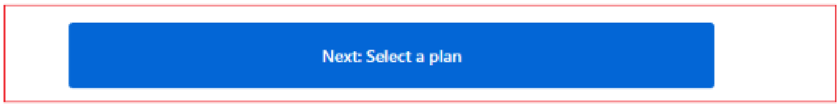
The first step to explore the benefits of GitHub is to create a GitHub account. GitHub provides both the free and pro membership to its user. We can explore many exciting and useful things in its pro account. We can explore unlimited private repository and can control the user access.

To create a GitHub account, visit **GitHub**.

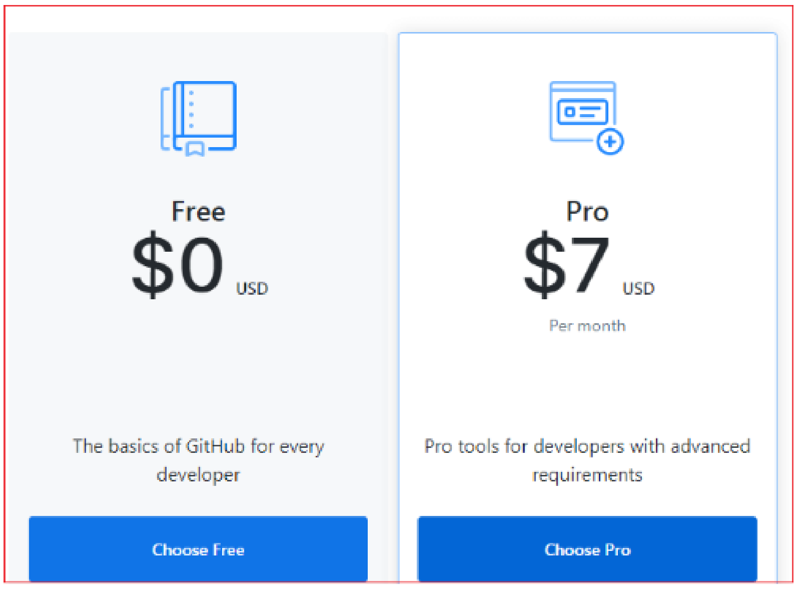
Click on the **Signup** option at the upper right corner.



Fill the necessary details under sign up like your name, email address, and password. Then click on the **Next: Select a plan** option.



Under the above option, you will see the plan. Select your plan, whether you want to be a pro member, or would like to continue with a free account.


Free
$0 USD
The basics of GitHub for every developer
Choose Free
Pro
$7 USD
Per month
Pro tools for developers with advanced requirements
Choose Pro

After selecting a plan, a confirmation link will send to your email address. Activate your account by clicking on the received link, and you are ready to go with GitHub.

2. GitHub Login

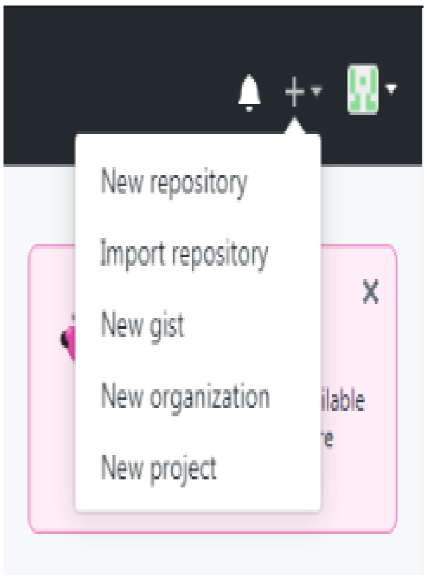
Log in to your GitHub account to use the GitHub service. To login to your account, click on the **Sign-in** option on the upper right corner. It will ask you for your email id and password. You can log in by entering your credentials. At your first login, the homepage will ask you to create your first repository and some other options like exploring the repository.

3. GitHub Repository

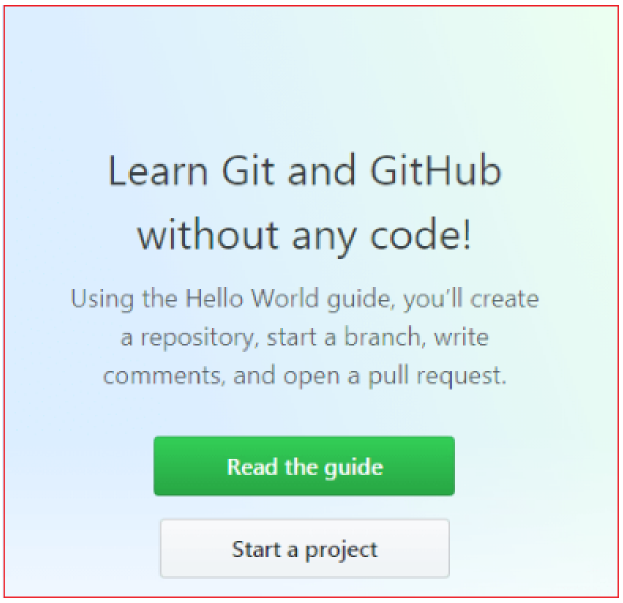
The repositories are the data structures used by GitHub to store metadata for files and directories. It encloses the collection of the files as well as the history of changes made to those files. Generally, the repository is considered a project folder. A single project can have more than one repository.

4. Create a Repository

We can create an unlimited public repository and unlimited private repository (For the pro user) on GitHub. To create a repository on GitHub, click on the '+' symbol on the upper right corner on the login screen.



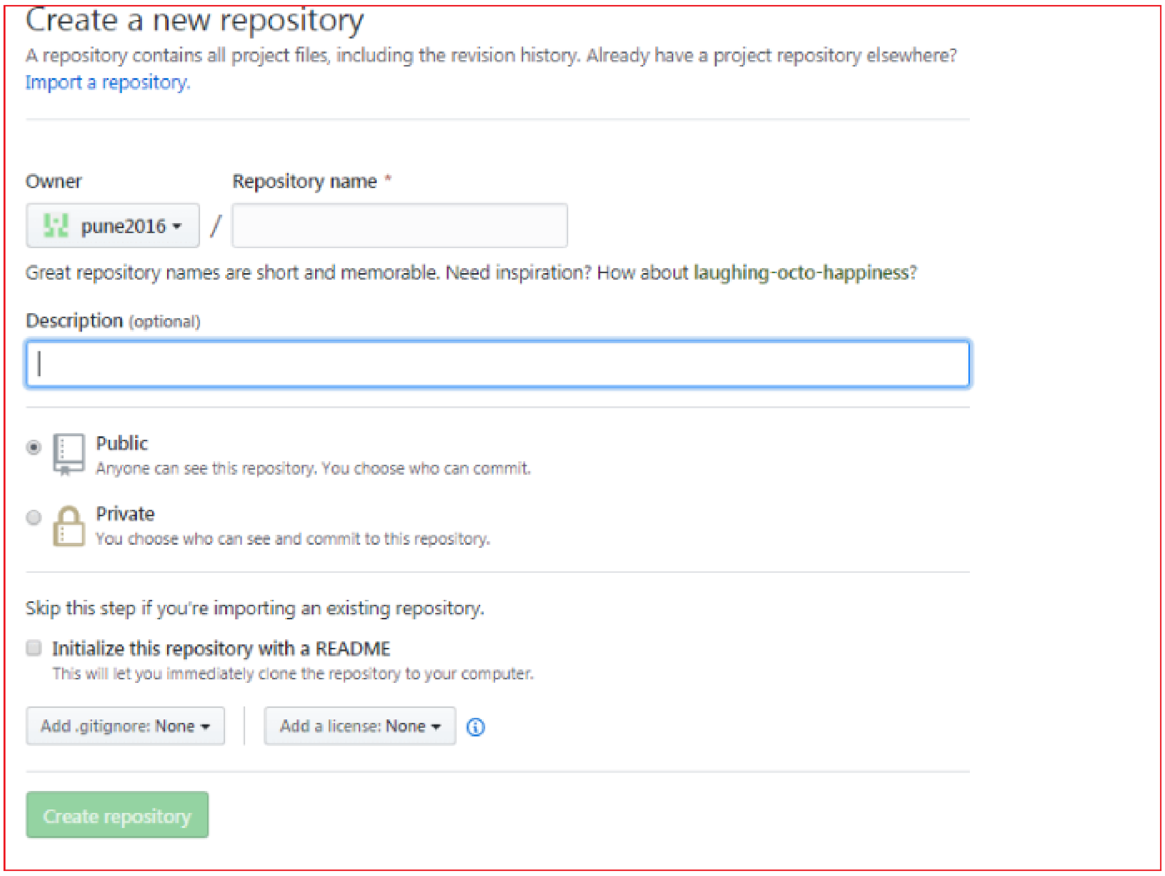
There are some other options available like import repository, gist, organization, and new project. To create a repository, choose **New repository** option from the given list. When you first log in to your account, you will see the UI as follows:


Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

GitHub asks you to learn Git and GitHub without any code. It will ask you to read the hello world guide for the first uses. Also, you can create a repository (Project) from here.

Click on the new repository option and then fill the required details like repository name, description, and select the access of this repository. You can also initialize the repository with a README file. After filling all the details, click on the **Create Repository** option. It will create a repository for you. Consider the below image:


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner: pune2016 / Repository name *

Great repository names are short and memorable. Need inspiration? How about laughing-octo-happiness?

Description (optional)

[✓ Public] Anyone can see this repository. You choose who can commit.

Private: You choose who can see and commit to this repository.

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

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None

Create repository

Hence, we have created a public repository.

How to create a private repository?

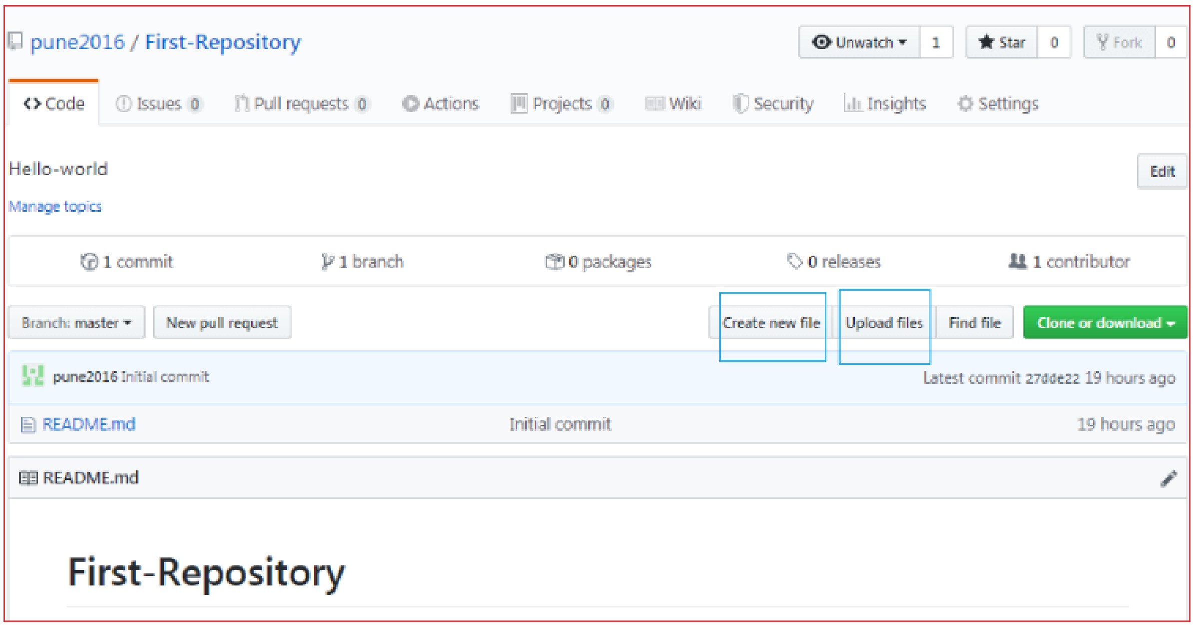
We can set up a repository as private during initialization and make and manage the accessibility of the repository. The private repository feature is only allowed to pro members. Pro members can create unlimited repositories and set access for them.

If we are on GitHub free account and want to change a repository's visibility from public to private, we will lose access to features like protected branches and GitHub Pages. The GitHub pages site automatically removed from our account.

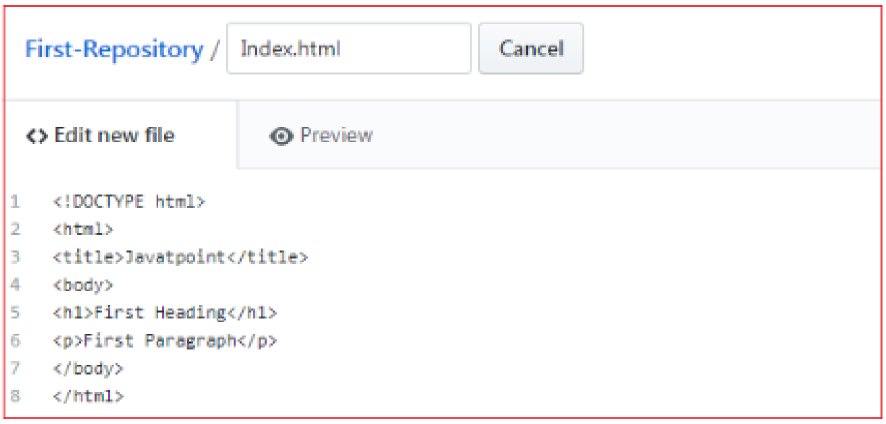
Hence we have created a repository and set its access. Now, we are all set to create our first file. Let's create a file:

5.Create a file

In GitHub, creating a file is a straight forward process. Let's create a file in our newly created repository. Consider the below snap of our repository:



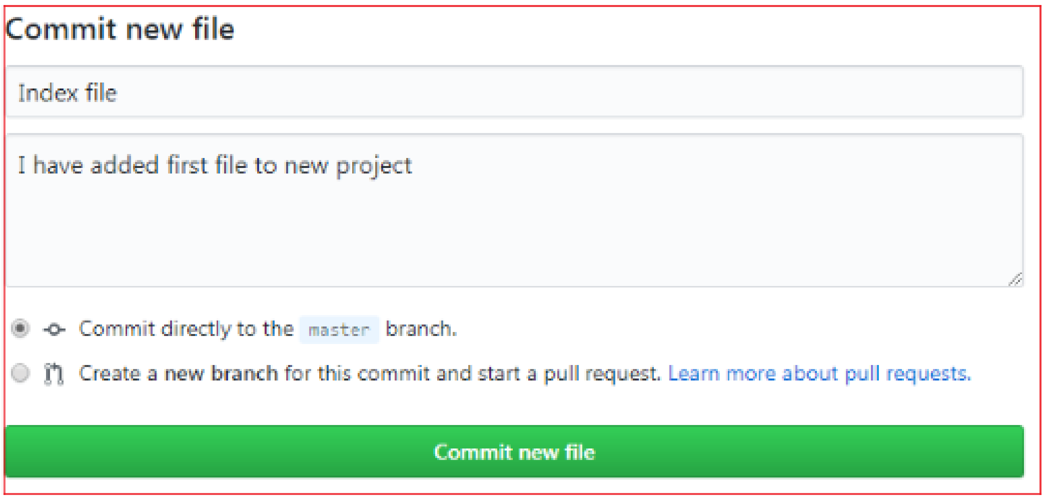
There are distinct options available to add files to the repository. GitHub allows us to design and upload files. To create a file, click on the '**Create new file**' option. It will open a file structure, and it will look like as follows:



```
<!DOCTYPE html>
<html>
<title>Javatpoint</title>
<body>
<h1>First Heading</h1>
<p>First Paragraph</p>
</body>
</html>
```

Enter the file name on the box and type the code on the editor area.

At the bottom of the page, the commit options are available. Consider the below snap:



Index file

I have added first file to new project

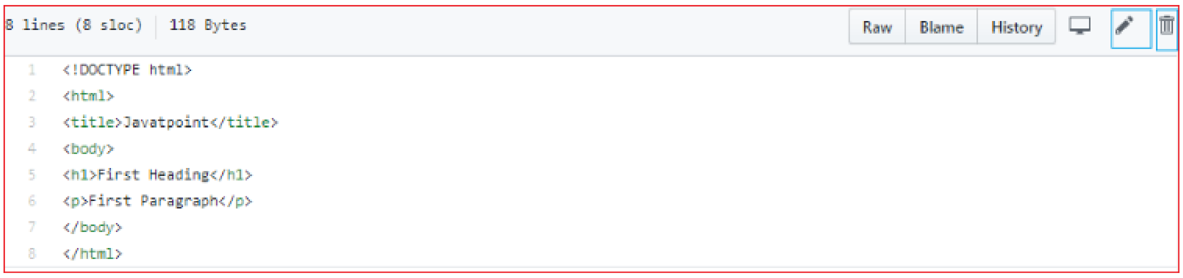
[✓ Commit directly to the master branch.]

Create a new branch for this commit and start a pull request.

In the above image, we can give the commit message in the first text area and the description in the second text area. Also, we can specify whether we want to commit it to the master branch or want to create a new branch.

Click on the '**commit new file**' option. We have successfully added and committed a new file to our repository.

We can edit and delete this file from our project. There are many options available, like edit, delete, Raw, Blame, and history. Consider the below snap of the file.


8 lines (8 sloc) | 118 Bytes

```
<!DOCTYPE html>
<html>
<title>Javatpoint</title>
<body>
<h1>First Heading</h1>
<p>First Paragraph</p>
</body>
</html>
```

Hence we have learned how to create a file and commit changes. Now we will see how to create a new branch.

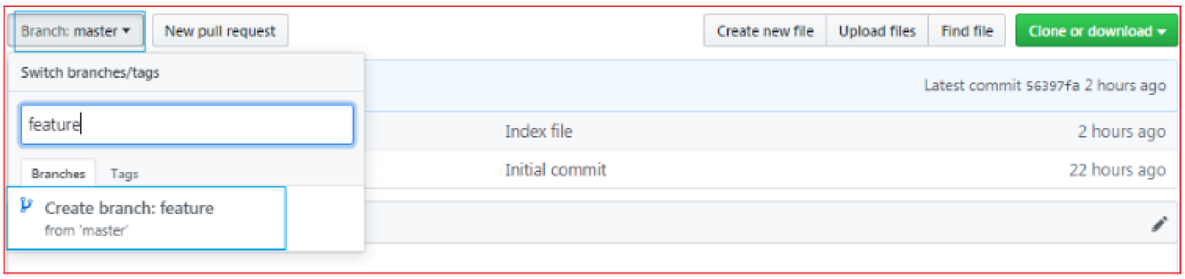
6. How to create a new branch?

Branches are the pointer to snapshots of changes. Branches are created for a particular purpose like fixing a bug, testing, release, and more. To understand the types of branches, visit Git Flow.

It is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.

The **master branch** is the **default** branch of the repository.

Let's understand how to create a branch in GitHub. To create a new '**feature**' branch, drag the branch option under the repository. This option will list the available branches. A search option is available under the branch. It will search for the requested branch if it is not in the repository, then it will create a new branch by the given name. Consider the below image:



Also, we can create the branch when we add a file or make some commit. It asks to commit the changes in the existing branch or create a new one.

Connect GitHub with your computer

GitHub Download

We can connect the GitHub with our computer. GitHub allows downloading its desktop application. Also, we can connect the GitHub repository with our computer by Git.

There are different kinds of audiences, some people love Git commands, and some love the attractive user interface for their work. The people who love the user interface, the GitHub desktop application is one of the best Git clients for them.

Features of GitHub Desktop

The desktop application of GitHub has incredible features that make collaboration easy for the contributor. Some of its attractive features are as follows:

- ✓ Attributing commits with collaborators easily.
- ✓ Checkout branches and create a PR(pull request)
- ✓ Broad editor and shell integration
- ✓ Open-source

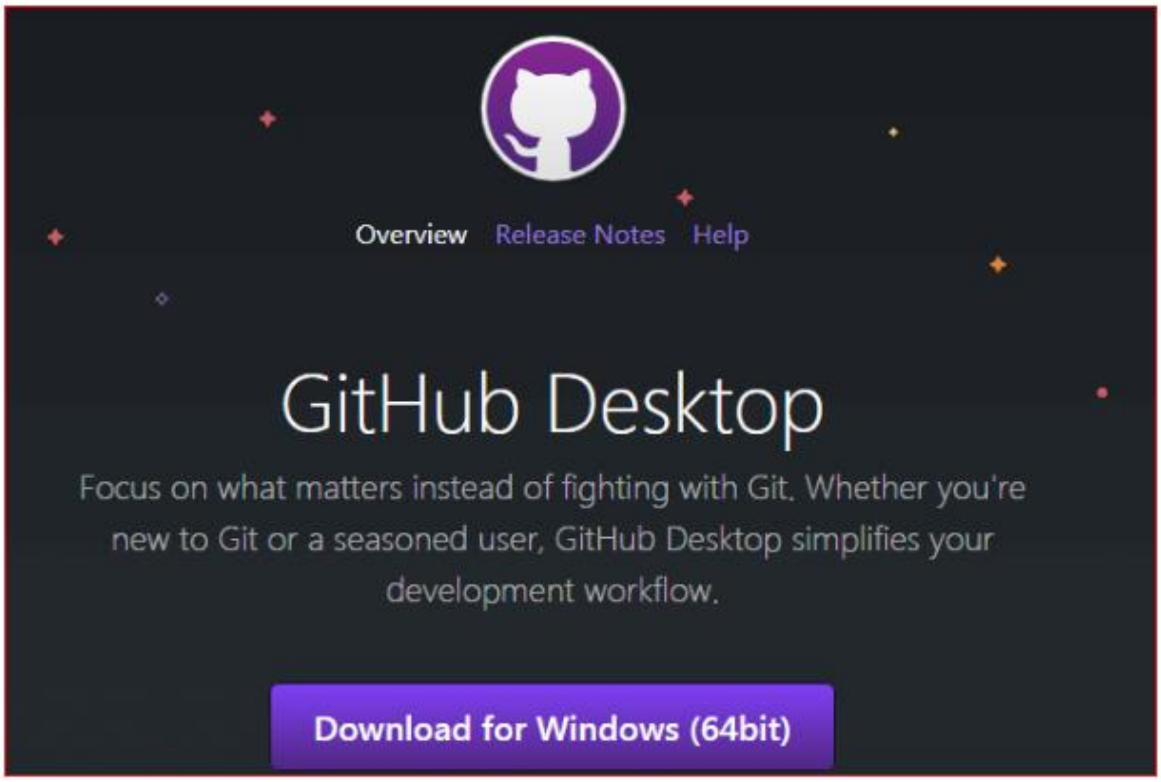
GitHub Desktop for Windows

To setup GitHub Desktop, we must already have a GitHub account. It is a fast and straight forward way to contribute to GitHub. It is developed to make straight-forward all the processes of GitHub.

GitHub Desktop is an open-source that can be downloaded. If we talk about its technical specification, it is written in Typescript and uses react. It is available for Microsoft Windows or macOS operating systems.

Below are the steps to install the GitHub desktop:

Step1: Visit https://desktop.github.com for Github desktop. To download the setup, click on the '**Download for Windows (64bit)**' option. Consider the below image:

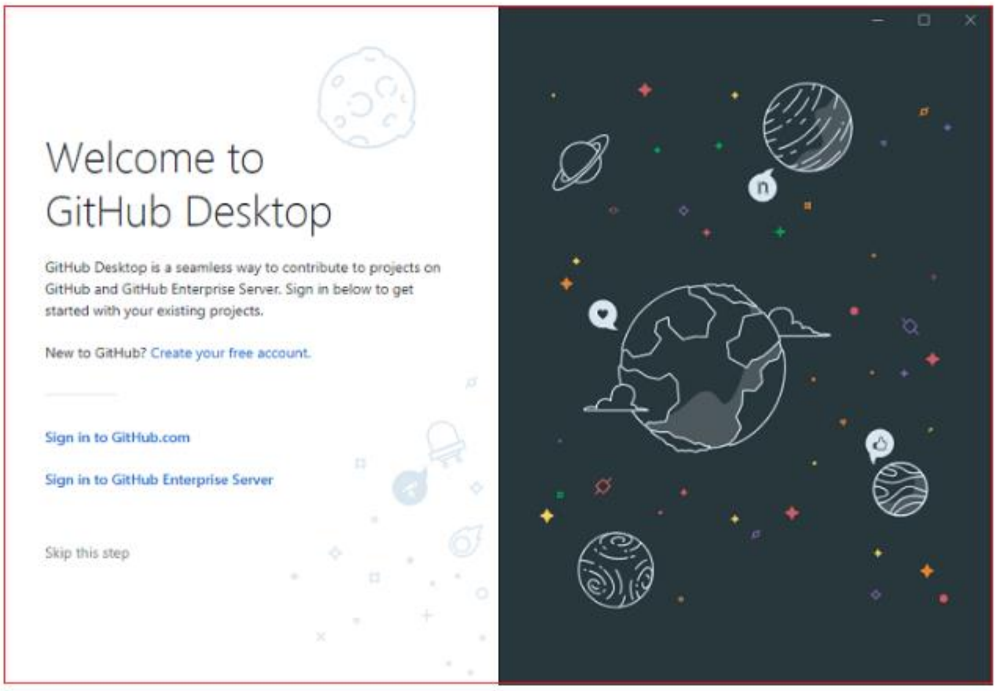


GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Step2: Installation:

Install the GitHub desktop application by running the installer file. There we can see options like create an account, sign in. However, we can skip this step. It looks like as follows:



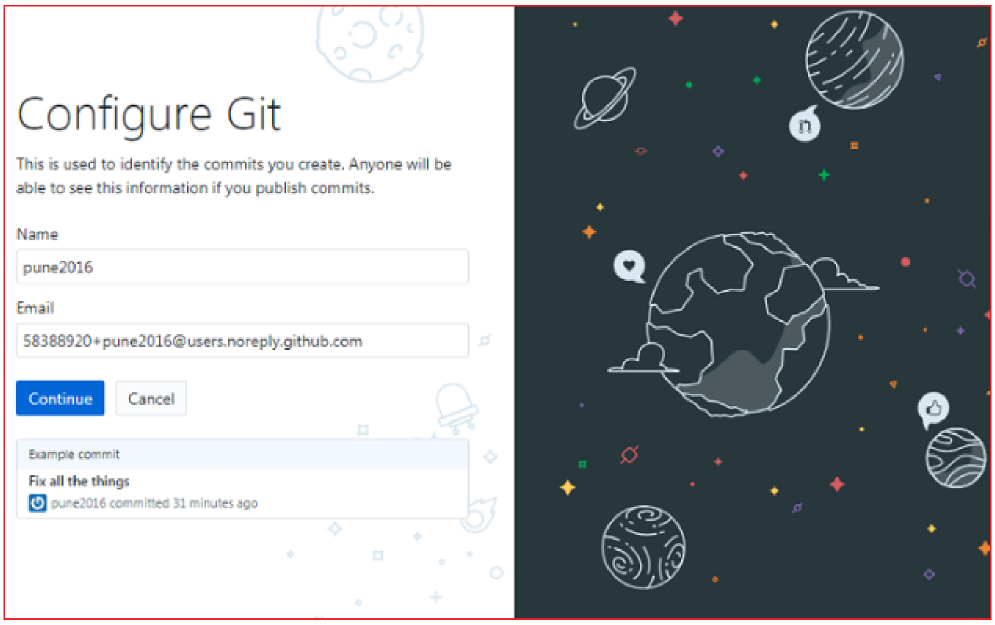
Welcome to GitHub Desktop

GitHub Desktop is a seamless way to contribute to projects on GitHub and GitHub Enterprise Server. Sign in below to get started with your existing projects.

New to GitHub? Create your free account.

Step3: Setting up the desktop application:

The next step after installing the GitHub is to customize it. However, we can skip this step. Sign in to your GitHub account. The configuration will look like as follows:



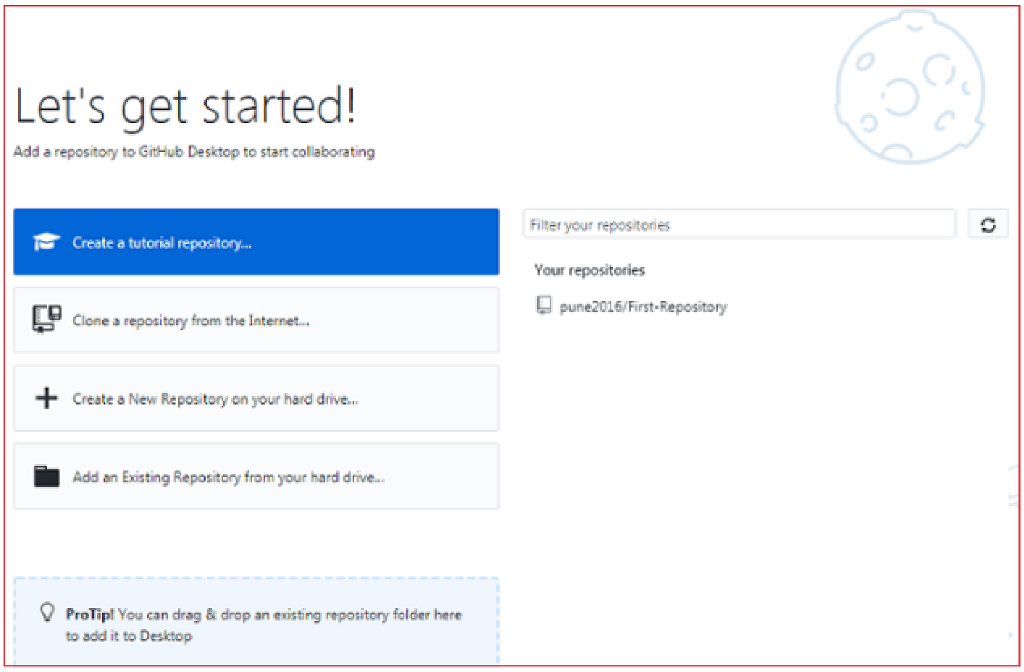
Configure Git

This is used to identify the commits you create. Anyone will be able to see this information if you publish commits.

Name: pune2016

Email: 58388920+pune2016@users.noreply.github.com

Now we are all set to go with Desktop application of GitHub. The UI of the GitHub desktop will look like as follows:



Let's get started!

Add a repository to GitHub Desktop to start collaborating

Create a tutorial repository...
Clone a repository from the Internet...
Create a New Repository on your hard drive...
Add an Existing Repository from your hard drive...
Filter your repositories
Your repositories
pune2016/First-Repository

ProTip! You can drag & drop an existing repository folder here to add it to Desktop

We can create, clone, or upload a repository to our project with GitHub desktop. It also lists your existing GitHub repositories.

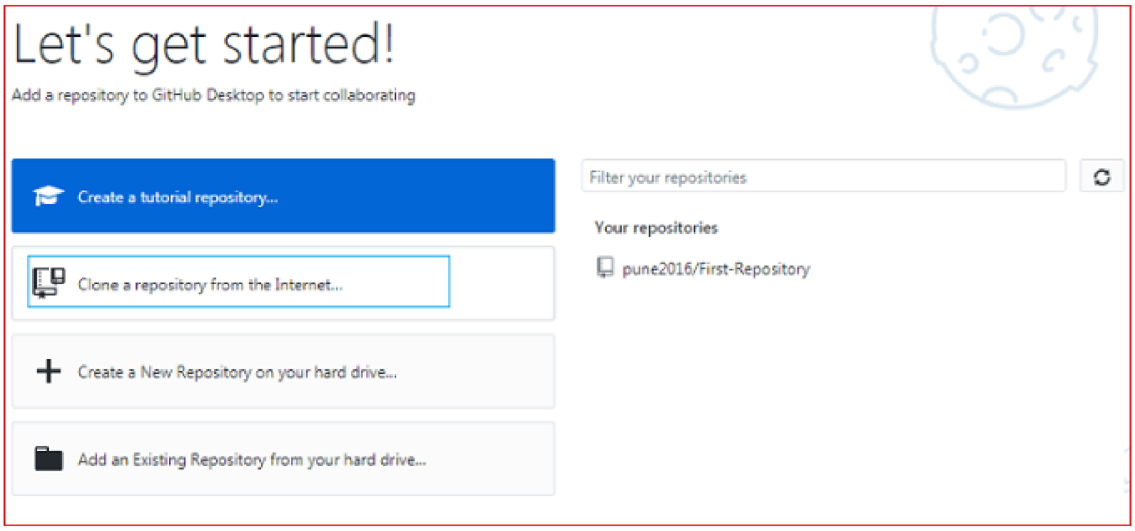
How to clone (copy) GitHub repository to our PC

There are many ways to copy a GitHub project. We can make a copy of a GitHub project on our local machine. To do so either, we can use the GitHub desktop application or Git Bash. Since here we are talking about GitHub so let's see how to copy by GitHub desktop application.

To copy from GitHub desktop application, follow the below steps:

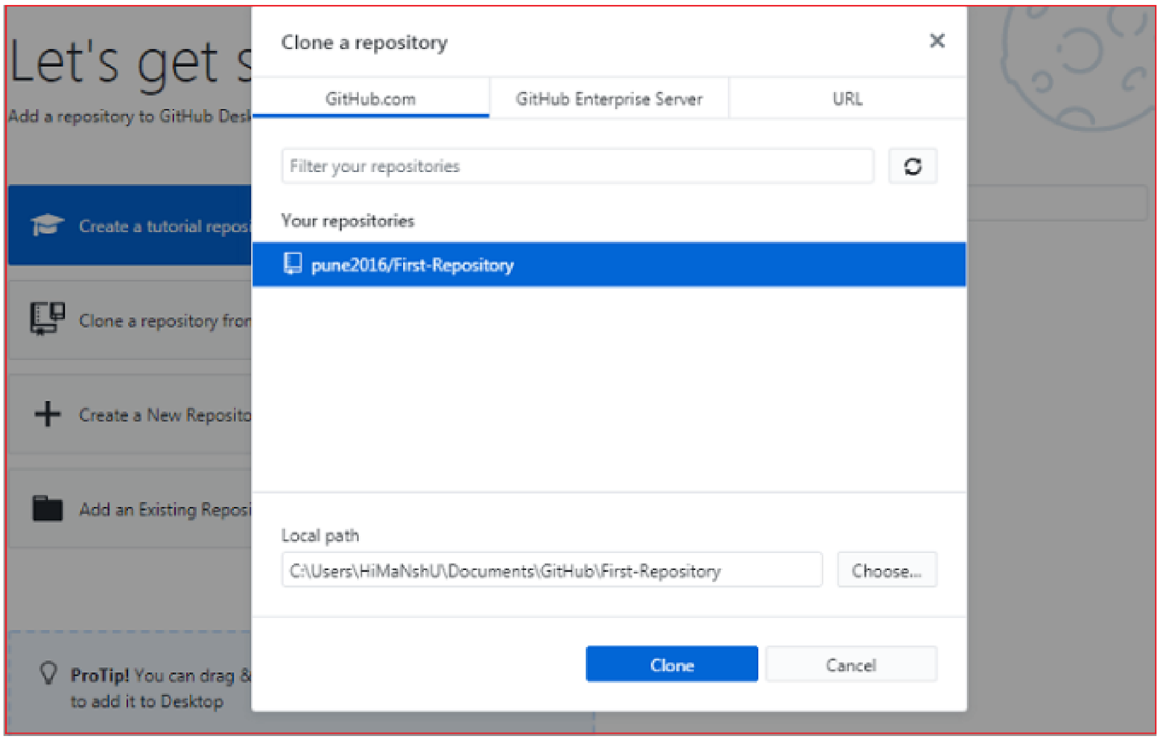
Step1: Open the GitHub desktop

Open the application, if you have not logged in yet, log in to the application by using your account credential. Select the option '**clone a repository from the internet**.' Consider the below image:


Let's get started!
Add a repository to GitHub Desktop to start collaborating
Create a tutorial repository...
Clone a repository from the Internet...
Create a New Repository on your hard drive...
Add an Existing Repository from your hard drive...
Filter your repositories
Your repositories
pune2016/First-Repository

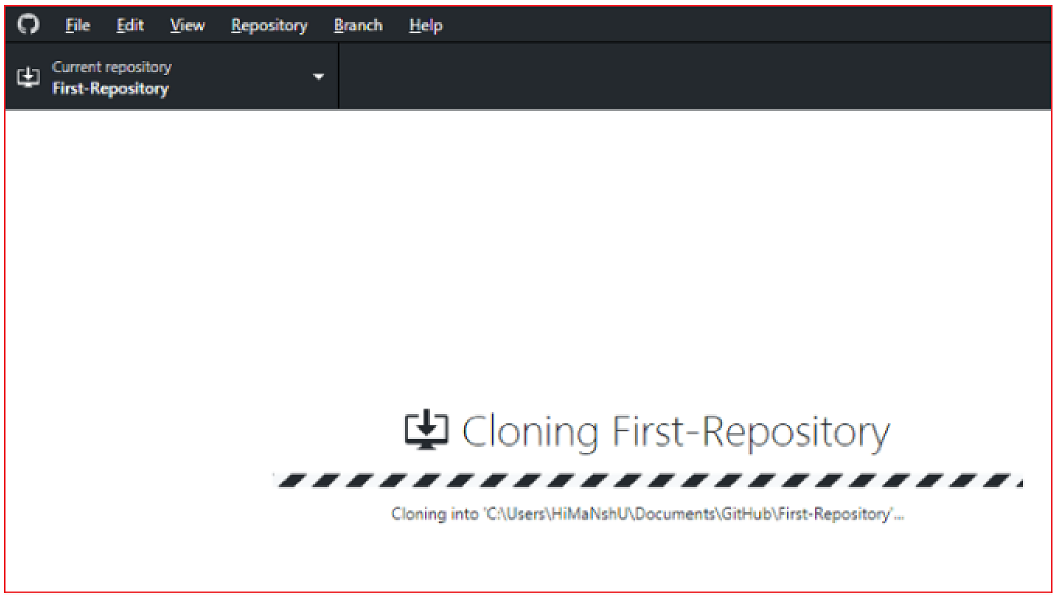
Step2: Select the repository

After selecting the clone option, it will list all the available repository on your GitHub account. Consider the below output:



Step3: Clone the Repository

Select your desired directory which you want to clone and click on the **clone** option. It will start copying the project. Consider the below image:



Cloning First-Repository

Cloning into 'C:\Users\HiMaNshU\Documents\GitHub\First-Repository'...

It will take a while to copy the project.

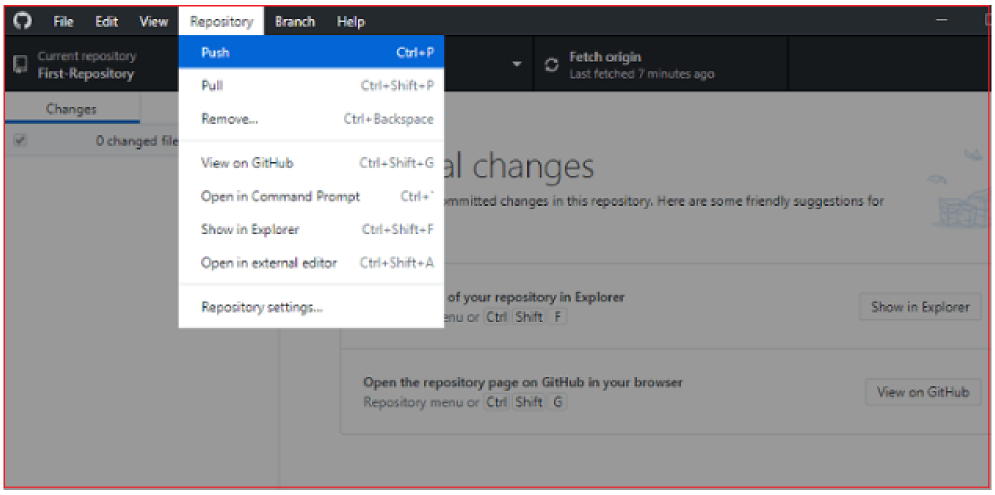
To clone a repository by Git Bash, run the clone command as follows:

```
$ git clone <repository URL>
```

How to pull from GitHub

We need to pull the data from GitHub to keep the local repository updated with the GitHub repository. Suppose any team member made or propose changes for our project. If you want to merge with your local directory, make a pull.

We can pull the updates from the GitHub project by GitHub desktop and Git Bash. To pull the changes by GitHub desktop, navigate to the **repository menu**, and select the pull option.



It will pull the GitHub repository.

To pull the repository by the Git Bash, run the below command:

```
$ git pull origin master
```

What is GitHub pull request?

Pull request is a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.

How to commit and push to GitHub

Pushing is the act of transferring the local changes to GitHub. Suppose we made some changes to your local repository and share it on GitHub. To do so, we can push the changes.

We can commit changes from our GitHub desktop application as well as Git Bash. To commit the changes from the GitHub desktop application, follow the below steps:

Step1: Open the file explorer

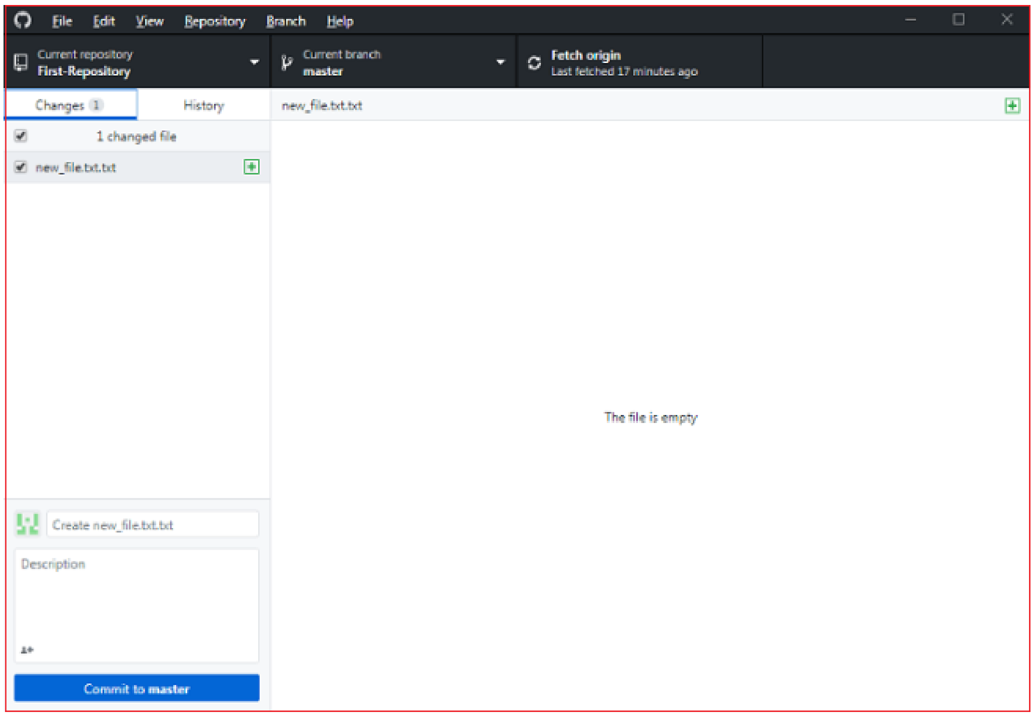
Open the file explorer from GitHub desktop. To open the file explorer, either press the '**Ctrl +Shift +F'** keys together, or we can select it from the menu.

Step2: Made the changes

Now, you are in the file explorer made the desired changes. In our case, we have created a file **new_file.txt.**

Step3: Commit the changes

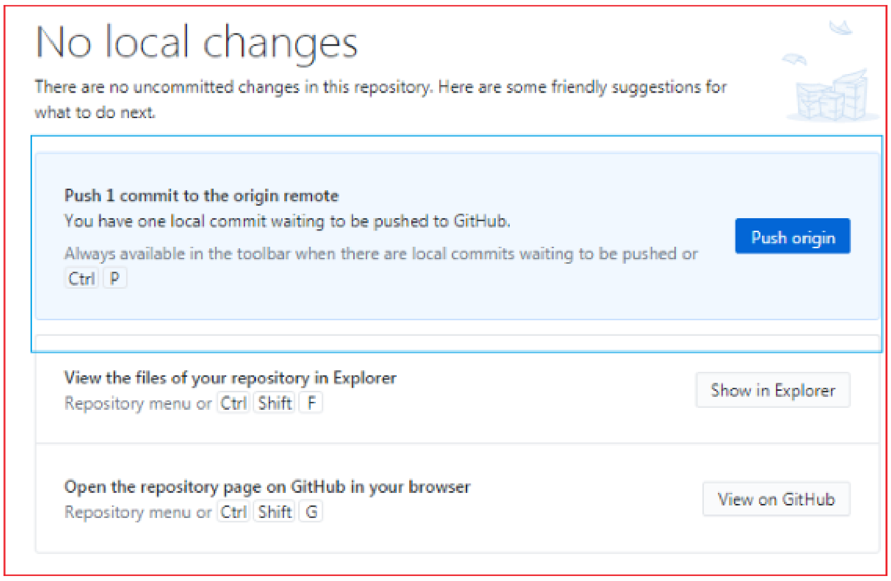
It is necessary to commit the changes to share it on GitHub. To commit the changes, open GitHub desktop; here, we can see the changes that we made. Consider the below image:



To commit the changes, type commit message, and description. After that, click on the commit option as displayed on the above image. Now, you have successfully made a commit. The next step is to push it to GitHub account.

Step4: Push the changes to GitHub account

The change that we have made is now ready to be pushed on GitHub account. The GitHub desktop application starts displaying a notification like a commit is ready to be pushed. Consider the below image:



No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Push 1 commit to the origin remote
You have one local commit waiting to be pushed to GitHub.
Always available in the toolbar when there are local commits waiting to be pushed or Ctrl P

View the files of your repository in Explorer
Repository menu or Ctrl Shift F

Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G

We can use the '**Ctrl + P**' keys or '**push origin**' option to push the changes to the GitHub repository.

Now, we have these changes in our GitHub repository.

To push the changes by Git Bash, run the below command:

```
$ git push origin master
```

------------------------**************************--