 A logo of a company

Description automatically generatedA red circle with white text and a letter on it

Description automatically generated



Lab Manual

**Introduction To Information And Communication Technology**

**Semester : Fall 2023**

**Program : BS**

**Course Title and Name : CSC 111**

**Credits : 0+1**

**Faculty : Mehar Khan Niazi**

**Student Name :**

**Student ID :**

**Total Marks : 100**

**Obtained Marks :**

**Submitted Date :**

**LAB 13**

**Objective:**

The objective of this lab is to introduce students to Git and GitHub – Version Control.

**Description:**

**How to Use Git and GitHub – Version Control?**



A Version Control System is a tool you use to track, make, and manage changes to your software code. It's also called source control.

A version control system helps developers store every change they make to a file at different stages so they and their teammates can retrieve those changes at a later time.

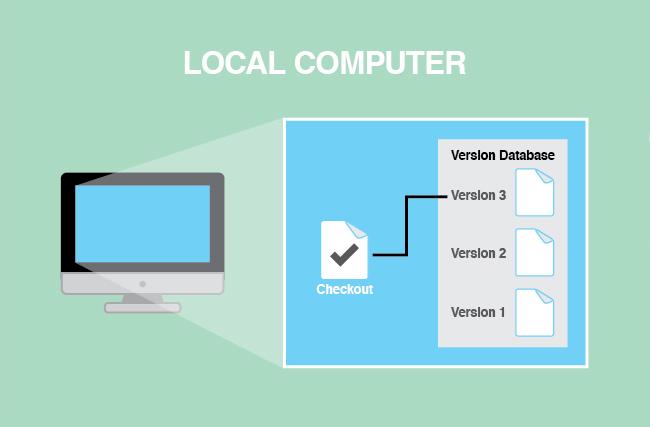
There are three types of version control systems, which are:

* Local Version Control Systems
* Centralized Version Control Systems
* Distributed Version Control Systems.

**What is a Local Version Control System (LVCS)?**

This is a type of version control system that is very common and simple to use. But this method is quite prone to errors and attacks because the files are being stored in your local system.

This means that you might lose the system file or accidentally forget the directory/folder of the file you are working (and then write in another directory).

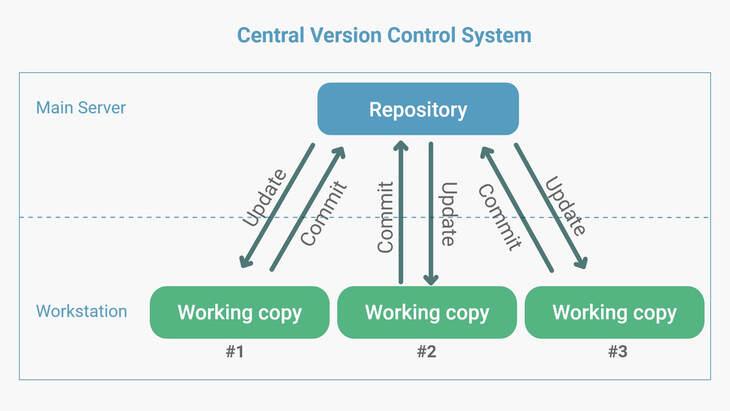


**What is a Centralized Version Control System (CVCS)?**

In this type of version control, a server act as a repository that stores each version of the code. The CVCS helps different developers collaborate together.

Despite the helpful collaboration and communication between developers, if a server goes down for few seconds or gets corrupted, there's the chance you'll lose your work. Unfortunately, this is a very big problem with the CVCS.

In CVCS, only a few developers can work together on a project.

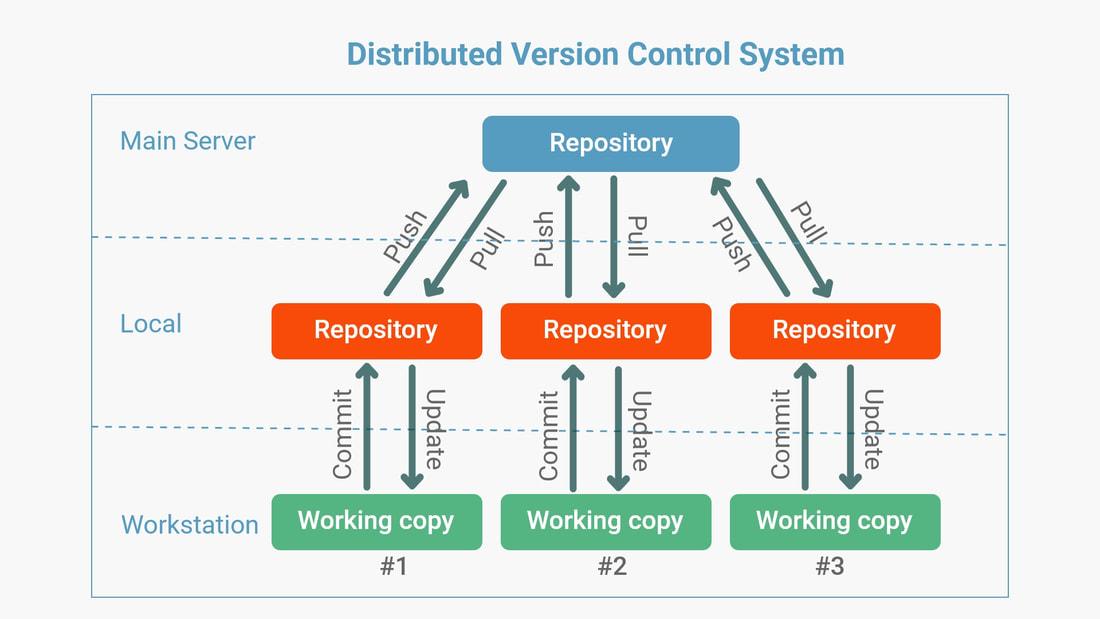


**What is a Distributed Version Control System (DVCS)?**

This is the latest and most commonly used type of version control system these days.

In a DVCS all developers have a full back up (clone) of all the data in the server. This means that whenever the server is down or faulty, you can still work on your project and you can copy or back up your repositories to the server to restore them.

When you're using a DVCS, many developers can work together on a project. One popular DVCS is Git, which we'll talk about more now.



**What is Git?**

Git is a free open source distributed version control system you can use to track changes in your files. You can work on all types of projects in Git, from small to large.

With Git, you can add changes to your code and then commit them (or save them) when you're ready. This means you can also go back to changes you made before.

Git works hand in hand with GitHub – so what is GitHub?

**What is GitHub?**

GitHub is a web interface where you store your Git repositories and track and manage your changes effectively. It gives access to the code to various developers working on the same project. You can make your own changes to a project at the same time as other developers are making theirs.

If you accidentally mess up some code in your project while making changes, you can easily go back to the previous stage where the mess has not occurred yet.

**Why use GitHub**

There are so many reasons you should learn and use GitHub. Let's look at a few of them now.

* **Effective Project Management**

GitHub is a place where your Git repositories are stored. GitHub makes it easy for developers working on the same project but in different locations to be on the same page.

With GitHub, you can easily track and manage the changes you have made and check on the progress you've made in your project.

* **Easy Collaboration and Cooperation**

With GitHub, developers from all over the world can work together on a project without having any problems.

Teams are able to stay on the same page while working on a project together and can easily organize and manage the project effectively.

* **Open Source**

GitHub is a free and open source system. This means that developers can easily access different types of code/projects which they can use in learning and developing their skills.

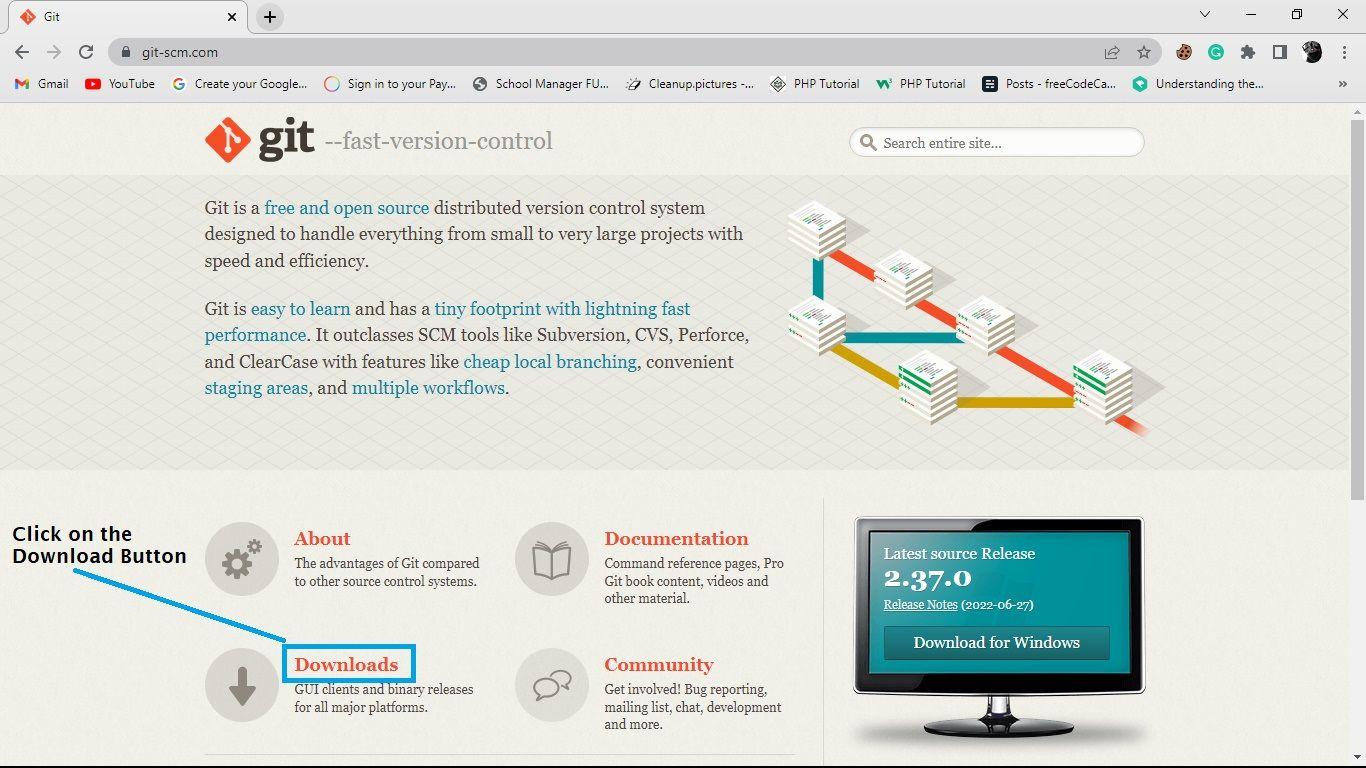
* **Versatility**

This attribute of GitHub is very important. GitHub is not a web interface for only developers. It can be used by designers, writers, and anyone who wants to keep track of the history of their projects.

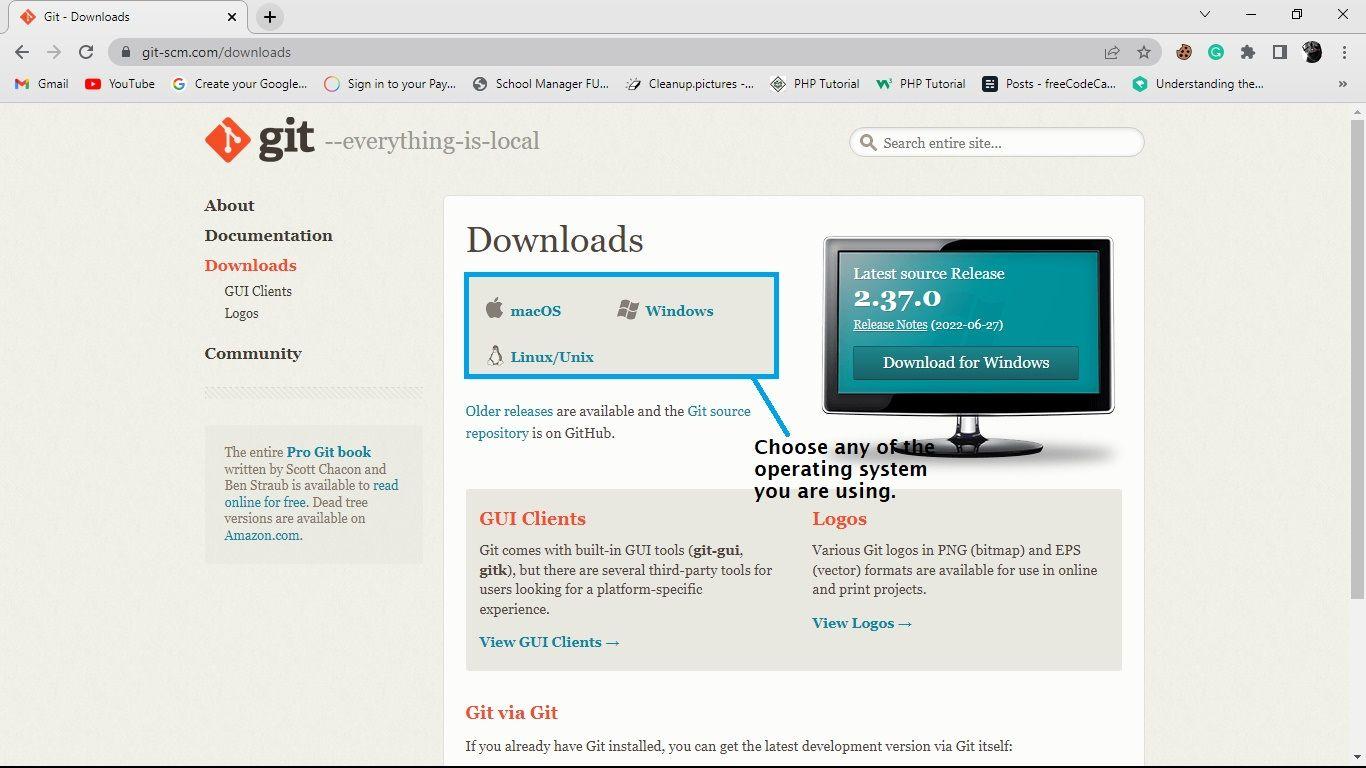
**How to Setup Git**

To start using Git, you'll need to download it to your computer if you haven't already. You can do this by going to their official website (<https://git-scm.com/>).

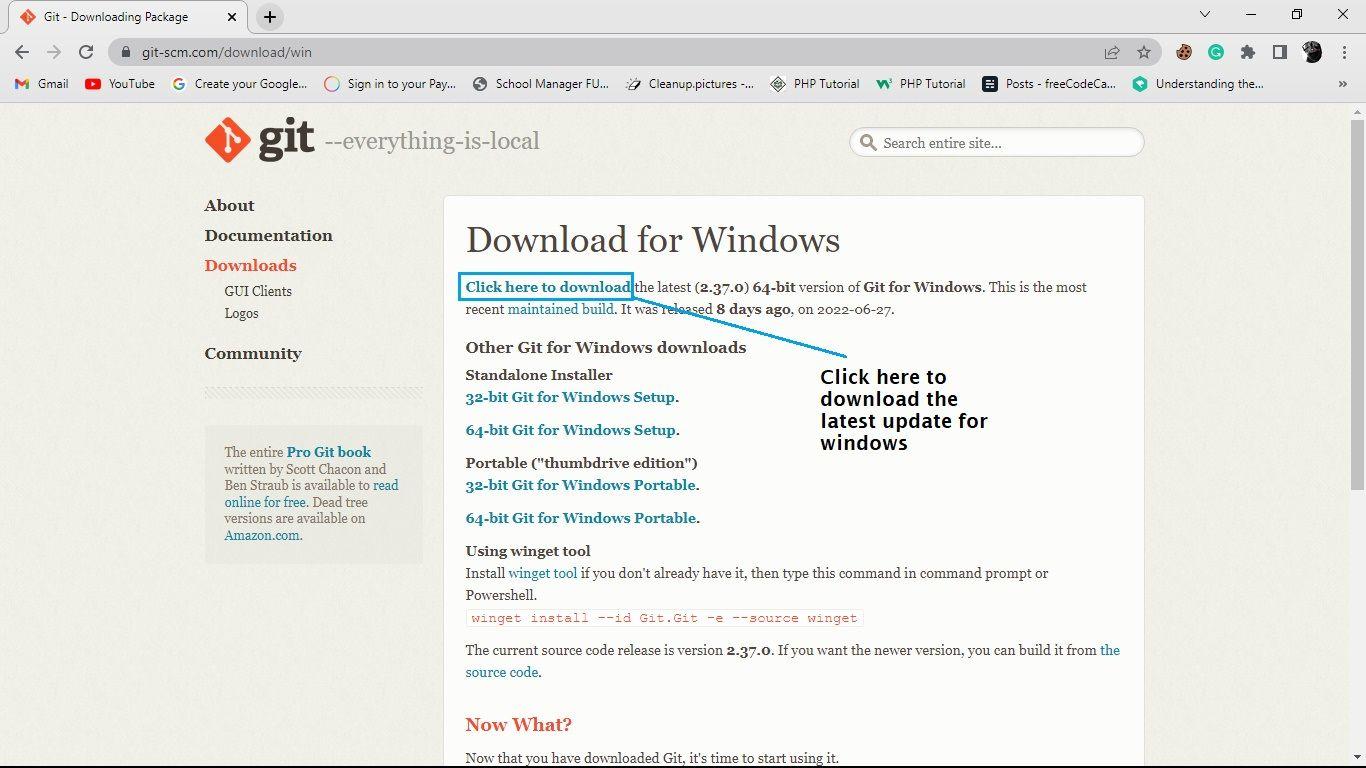
When Git opens, scroll down a bit and you should see a download button. Go ahead and click on it.



Choose your operating system whether it's Windows, MacOS, Linux/Unix. In my case, I will be choosing the Windows option because I am using a Windows computer:



Click on the first link at the very top of the page to download the latest version of Git.



When the download is complete, then go ahead and install Git to your computer. You'll need to go to the location where the file has been downloaded and install it.

After the installation, you'll want to make sure that Git is successfully installed on your system. Open your command prompt or Git bash (whichever one you choose to use) and run the command:

git –version

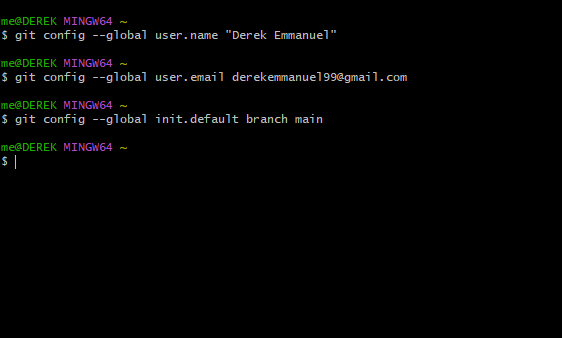
bandicam-2022-07-05-20-48-13-907

If Git was successfully installed on your computer, it should display the current version of Git below the command you just ran. If the current version is being displayed, congratulations!

**How to Configure Git**

Now that we have installed Git on our computer, we have to configure it. We do this so that any time we are working in a team on a project, we can easily identify the commits we have made in the repository.

To configure Git, we need to specify the name, email address, and branch by using the **git config --global** command. For example:



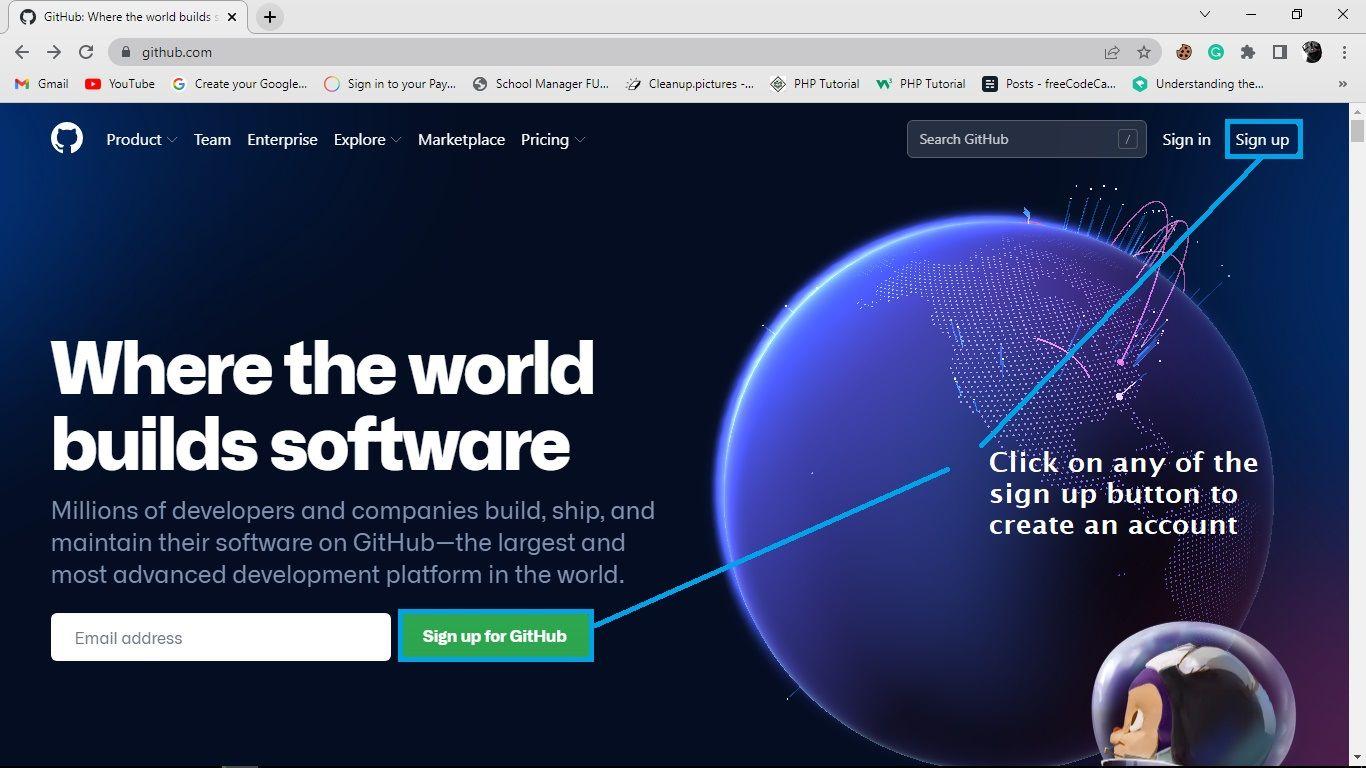
From the image above, we used **git config --global user.name** to configure the username. In my case I used name “Derek Emmanuel”. The same applies for the **git config --global user.email**.

Git comes with a default branch of master, so I changed it to be called the main branch by using the **git config --global init.default branch main** command.

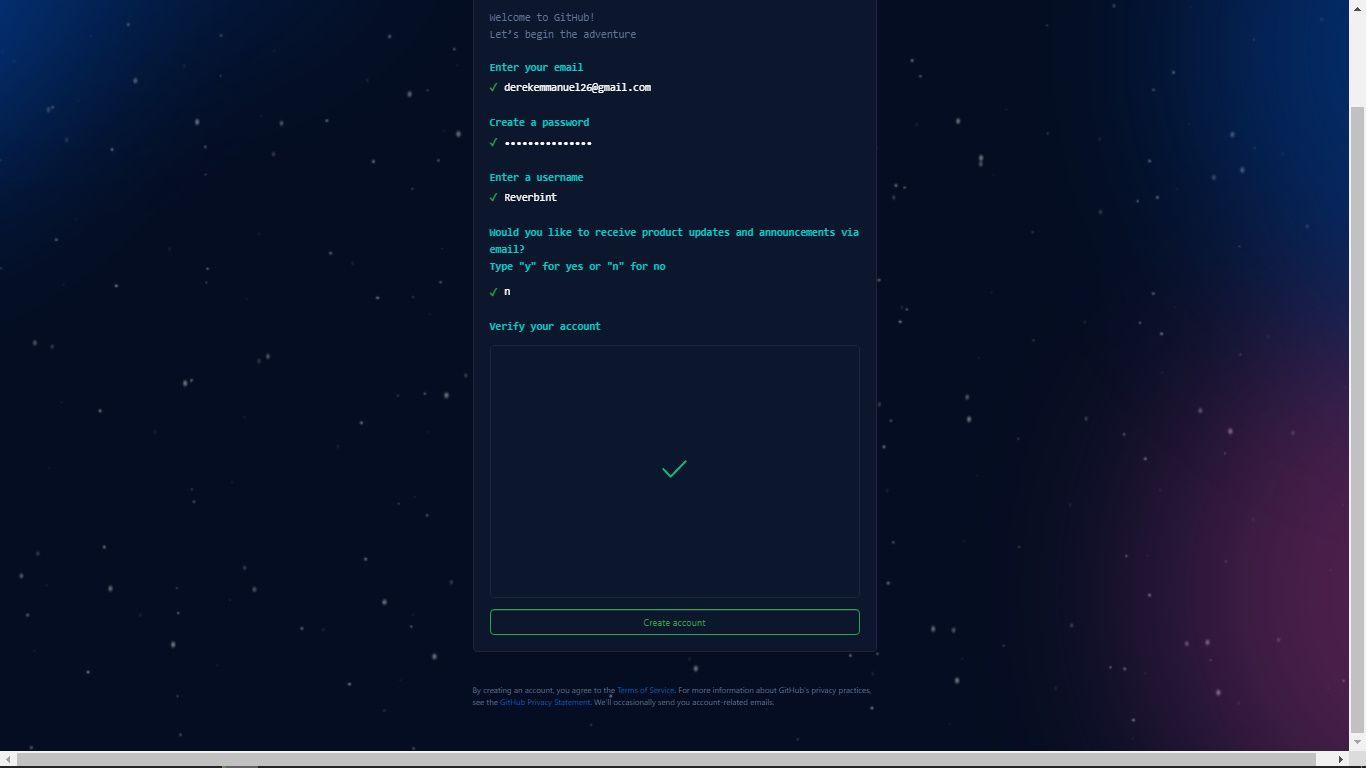
Now you're ready to start using Git.

**How to Setup a GitHub Account**

To set up a GitHub account, visit their official website(<https://github.com/>). Click on the sign up button in the upper right corner:



When the sign up form opens up, enter your email, create a password, enter your username, and then verify your account before clicking on the create account button.



**Files states in Git**

In Git, there are three primary states (conditions) in which a file can be: modified state, staged state, or committed state.

1. **Modified state**

A file in the modified state is a revised — but uncommitted (unrecorded) — file.

In other words, files in the modified state are files you have modified but have not explicitly instructed Git to monitor.

1. **Staged state**

Files in the staged state are modified files that have been selected — in their current state (version) — and are being prepared to be saved (committed) into the .git repository during the next commit snapshot.

Once a file gets staged, it implies that you have explicitly authorized Git to monitor that file’s version.

1. **Committed state**

Files in the committed state are files successfully stored into the .git repository.

Thus, a committed file is a file in which you have recorded its staged version into the Git directory (folder).

Note: The state of a file determines the location where Git will place it.

**File locations**

There are three key places versions of a file may reside while version controlling with Git: the working directory, the staging area, or the Git directory.

1. **Working directory**

The working directory is a local folder for a project's files. This means that any folder created anywhere on a system is a working directory.

**Note:**

* Files in the modified state reside in the working directory.
* The working directory is different from the .git directory. That is, you create a working directory while Git creates a .git directory.

1. **Staging area**

The staging area — technically called “index” in Git parlance — is a file, usually located in the .git directory, that stores information about files next-in-line to be committed into the .git directory.

**Note:**

* Files in the staged state reside in the staging area.

**Git directory**

The .git directory is the folder (also called “repository”) that Git creates inside the working directory you have instructed it to track.

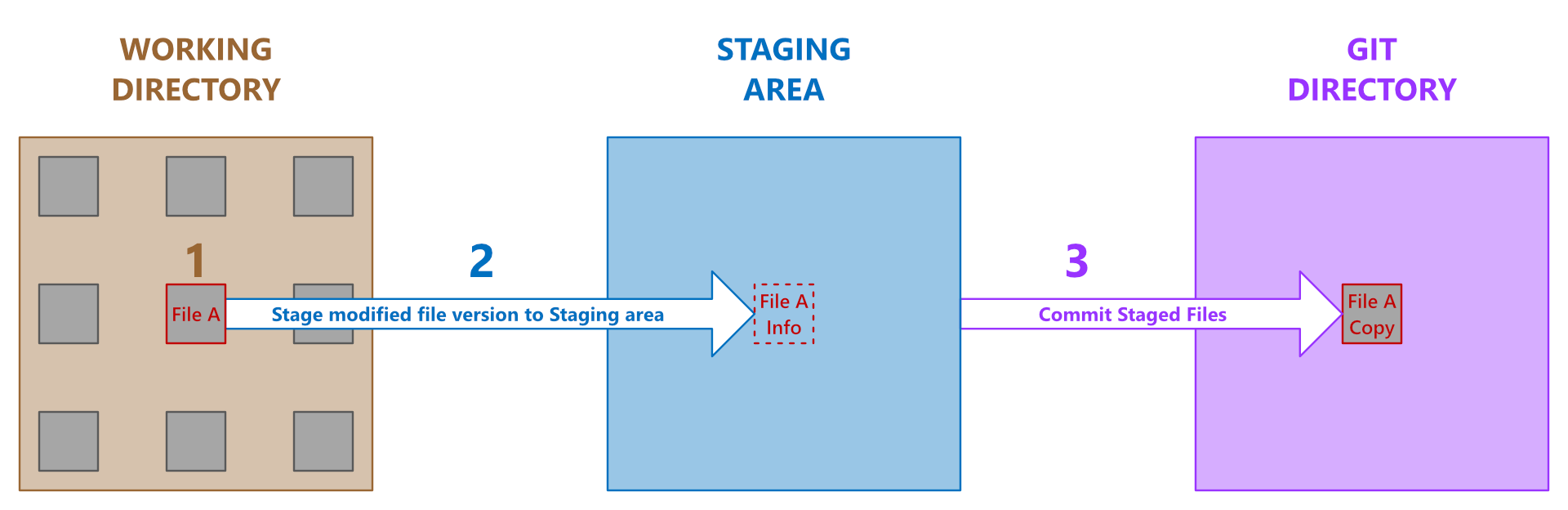
Also, the .git folder is where Git stores the object databases and metadata of the file(s) you have instructed it to monitor.

**Note:**

* The .git directory is the life of Git — it is the item copied when you clone a repository from another computer (or from an online platform like GitHub).
* Files in the committed state reside in the Git directory.

**The basic Git workflow**

Working with the Git Version Control System looks something like this:



1. Modify files in the working directory. Note that any file you alter becomes a file in the modified state.
2. Selectively stage the files you want to commit to the .git directory. Note that any file you stage (add) into the staging area becomes a file in the staged state. Also, be aware that staged files are not yet in the .git database. Staging means information about the staged file gets included in a file (called "index") in the .git repository.
3. Commit the file(s) you have staged into the .git directory. That is, permanently store a snapshot of the staged file(s) into the .git database. Note that any file version you commit to the .git directory becomes a file in the committed state.

**GitHub Demystified**

GitHub is a web-based platform where users can host Git repositories. It helps you facilitate easy sharing and collaboration on projects with anyone at any time.

GitHub also encourages broader participation in open-source projects by providing a secure way to edit files in another user's repository.

To host (or share) a Git repository on GitHub, follow the steps below:

**Step 1: Signup for a GitHub account**

The first step to begin hosting on GitHub is to create a personal account. Visit the official registration page to sign up.

**Step 2: Create a remote repository in GitHub**

After signing up for an account, create a home (a repository) in GitHub for the Git repository you want to share.

**Step 3: Connect the project’s Git directory to the remote repository**

Once you've created a remote repository for your project, link the project’s .git directory — located locally on your system — with the remote repository on GitHub.

To connect to the remote repository, go inside the root directory of the project you want to share via your local terminal, and run:

git remote add origin https://github.com/yourusername/yourreponame.git

**Note:**

* Replace yourusername in the code above with your GitHub username.
* Likewise, replace yourreponame with the name of the remote repository you want to connect to.
* The command above implies that git should add the specified URL to the local project as a remote reference with which the local .git directory can interact.
* The origin option in the command above is the default name (a short name) Git gives to the server hosting your remote repository.
* That is, instead of the server's URL, Git uses the short name origin.
* It is not compulsory to stick with the server’s default name. If you prefer another name rather than origin, simply substitute the origin name in the git remote add command above with any name you prefer.
* Always remember that a server’s short name (for example, origin) is nothing special! It only exists — locally — to help you easily reference the server’s URL. So feel to change it to a short name you can easily reference.
* To rename any existing remote URL, use the git remote rename command like so:
* git remote rename theCurrentURLName yourNewURLName

Whenever you clone (download) any remote repo, Git automatically names that repo’s URL origin. However, you can specify a different name with the git clone -o yourPreferredName command.

To see the exact URL stored for nicknames like origin, run git remote -v command.

**Commonly Used Git Commands**

There are some basic Git commands that every developer should know how to use:

* git config
* git init
* git add
* git commit
* git clone
* git push
* git rm
* git branch

Let's go through each of these briefly so you know how to use them.

**How to Use the git config Command**

You use this command to set the username, email, and branch of a user so as to identify who made a commit when working on a project. This command is used when you have downloaded git into your computer and you want to customize it for your use.

For example:

git config --global user.name “ [username]”

git config --global user.email [email address]

**How to Use the git init Command**

You use the **git init** command to start Git in your project. This git command is used when you are working on a project and would like to initialize git to the project in order to keep track of the changes made in the project.

**For example:**

git init

When you run this command, you should see a folder named **.git** being created automatically in the current folder you are working on.

**How to Use the git add Command**

This command adds your file to the staging area. The staging area is the area where files we make changes to are added and where they wait for the next commit.

To add a file to the staging area, you use the **git add** command. It adds all the files in the folder to the staging area.

**git add (file name)** adds the name of the particular file you want to commit in the staging area.

Use this command when you have made changes to a file and want to commit them to your project.

**How to Use the git commit Command**

This commits any file you added with the git add command as well as every file in the staging area.

**For example:**

**git commit –m “first commit”**

This command saves a file permanently to the Git repository. You use it whenever a file has been added to the staging area using the **git add** command.

**How to Use the git clone Command**

You use the **git clone** command to copy an existing repository in another location to the current location where you want it to be.

For example:

**git clone (repository name)**

You use this command when you want to duplicate a Git repository from GitHub into your local storage.

**How to Use the git push Command**

You use this command to upload/push files from the local repository/storage to another storage, like a remote storage such as GitHub.

**For example:**

**git push (remote storage name)**

You only use this command when you're satisfied with the changes and commits you've made on a project and finally want to upload/push it to the Git repository in GitHub.

**How to Use the git rm Command**

You use this Git command to remove a file from a working repository. For example:

**git rm (filename)**

You use this command only when you wish to get rid of an unwanted changes/files from the Git repository.

**How to Use the git branch Command**

You use this command to check the current branch you are working on, either main or master.

**For example:**

**git branch**

This command helps you know the current branch you are working on.