

HANDBOOK

# Version Controlling & Hosting



# Table of Contents

## Table of Contents

.....	
.....	<b>0</b>
<b>Table of Contents .....</b>	<b>1</b>
<b>Course Objectives .....</b>	<b>2</b>
<b>Introduction to GitHub .....</b>	<b>3</b>
Definition of Git and Github .....	3
Account Creation .....	6
Creating Repositories .....	8
Commit the change .....	9
Commands .....	12
<b>Streamlit Community Cloud .....</b>	<b>14</b>
Creating Account on Streamlit Community Cloud .....	14
File Organization for Community Cloud App .....	16
Connect GitHub Account and Deploying App .....	18
<b>References .....</b>	<b>20</b>

## Course Objectives

After completing this handbook, learner will be able to

- Apply modern web development skills using AI-assisted tools like GitHub Copilot and Cursor AI.
- Enable rapid prototyping and data visualization through Streamlit for building interactive web applications.
- Introduce AI integration techniques using tools like Gemini API for smart feature development (e.g., chatbots, content generation).
- Develop proficiency in version control and cloud deployment using Git, GitHub, and Streamlit Community Cloud.
- Empower faculty to transfer industry-relevant skills to students, enhancing employability and innovation in academic projects.

# Chapter 4: Version Controlling & Hosting

## Learning Outcomes:

- Understand what Git and Github is?
- Explain the key features and differences between Git and GitHub.
- Create a GitHub account and navigate its features such as repositories, dashboards, and profile settings.
- Organize and structure files appropriately for Streamlit Community Cloud applications.
- Sign up for and configure Streamlit Community Cloud using GitHub or other authentication methods.
- Connect a GitHub repository to Streamlit Community Cloud and deploy a Streamlit application.
- Demonstrate the complete workflow of building, versioning, hosting, and deploying simple web applications using Git, GitHub, and Streamlit Community Cloud.

## Introduction to GitHub

In modern software development, collaboration, version management, and efficient tracking of changes are essential. Tools that support these activities enable teams to work on large and complex projects without losing track of contributions and modifications. Git and GitHub are two closely related technologies that serve as fundamental components of this workflow. Although often mentioned together, Git and GitHub are different in nature and functionality. Understanding their roles is crucial for any developer or team working on projects that involve coding or document versioning.

## Definition of Git and Github

**Git:** Git is a distributed version control system (VCS) primarily used to track changes in source code during software development. The term ***distributed*** means that every developer working on a project has a complete copy of the repository, including its full history of changes, stored locally on their own machine. This decentralization ensures that work can continue even when there is no internet connection.

The main objectives of Git are:

1. **Tracking Changes:** Git records every change made to files. Developers can review, revert, or merge changes at any time.

2. **Collaboration:** Multiple developers can work on the same project simultaneously. Git ensures that their changes are integrated properly without overwriting each other's work.
3. **Speed and Efficiency:** Git is designed to perform tasks like commits, merges, and branching very quickly, even for large projects.
4. **Data Integrity:** Every version of a file and its history is stored in a way that ensures data cannot be corrupted without detection.
5. **Non-linear Development:** Git supports creating and managing multiple branches, enabling experimentation with new features without affecting the main version of the project.

Although Git was originally developed for software development, it can also be used for any work that requires version control, such as managing documentation, presentations, or datasets.

**GitHub:** This is a cloud-based platform that hosts Git repositories and enhances them with additional tools for collaboration, project management, and security. It is not a version control system by itself; rather, it is a service built on top of Git that provides an accessible interface for storing, sharing, and managing repositories online.

**Key features of GitHub include:**

1. **Remote Hosting of Git Repositories:** GitHub allows developers to store their repositories on a central server. This enables sharing and collaboration from anywhere.
2. **Collaboration Tools:** It provides features like pull requests, code reviews, and discussions, which make teamwork more organized and transparent.
3. **Project Management:** GitHub offers boards, issues, and milestones for tracking progress and managing tasks.
4. **Community and Networking:** Developers can contribute to open-source projects, follow other developers, and showcase their work.
5. **Integration with CI/CD Tools:** GitHub integrates with tools for continuous integration and deployment, improving automation in the development process.
6. **Security and Backup:** Hosting code on GitHub ensures that there is an additional backup of the repository in the cloud, along with security features like vulnerability scanning.

While Git can be used entirely offline, GitHub acts as a central hub where team members synchronize their local changes and collaborate effectively.

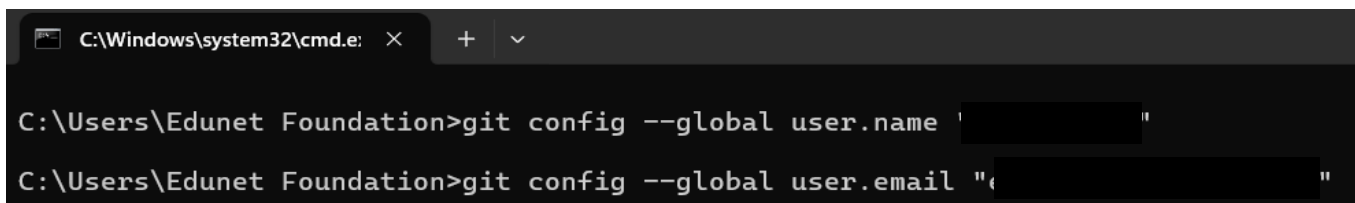
Below is a table of differences between Git and GitHub:

Aspect	Git	GitHub
Type	Distributed version control system (software/tool)	Web-based hosting platform built on top of Git
Functionality	Tracks and manages changes to files locally and in distributed setups	Provides cloud storage for repositories with additional collaboration features
Connectivity	Can work completely offline	Requires internet access for hosting, syncing, and collaboration
Scope	Focuses on version control and branching	Adds features like pull requests, code reviews, issue tracking, and project boards
Usage	Used from the command line or GUI clients	Accessed through a web interface and integrations
Ownership	Free and open-source software maintained by the community	Commercial platform (offers free and paid plans)
Primary Role	Version control and code history management	Centralized repository hosting and collaboration environment

### Install Git on Windows: Git for Windows stand-alone installer

- Download the latest Git for Windows installer - <https://git-scm.com/downloads/win>
- After the download is completed, start the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.
- Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).
- Run the following commands to configure your Git username and email using the following commands, replacing Emma's name with your own.
- These details will be associated with any commits that you create:

```
git config --global user.name "Emma Paris"  
git config --global user.email eparis@atlassian.com
```



```
C:\Windows\system32\cmd.exe  X  +  v  
  
C:\Users\Edunet Foundation>git config --global user.name "[REDACTED]"  
C:\Users\Edunet Foundation>git config --global user.email "[REDACTED]"
```

## **Account Creation**

### **1. Open the GitHub Website**

- Open any web browser (such as Google Chrome, Edge, Firefox).
- Type <https://github.com> in the address bar and press Enter.
- This will take you to the GitHub home page.

### **2. Start the Sign-Up Process**

- On the home page, locate and click the “Sign up” button, usually found in the top-right corner.

### **3. Enter Your Email Address**

- You will be asked to enter your valid email address.
- Make sure to use an email that you have access to, as you will need it for verification.
- Click on “Continue”.

### **4. Create a Password**

- Enter a strong password that you can remember.
- Use a combination of uppercase letters, lowercase letters, numbers, and special symbols.
- Click “Continue”.

### **5. Choose a Username**

- Provide a unique username. This will be visible to others on GitHub.
- If your chosen username is already taken, GitHub will suggest alternatives or you can try a different one.
- Click “Continue”.

### **6. Verify You Are Not a Robot**

- Complete the captcha verification.

### **7. Verify Your Email Address**

- After completing the previous steps, GitHub will send a verification code to the email address you provided.
- Open your email inbox, find the mail from GitHub, copy the code, and paste it into the verification box on the GitHub page.

## 8. Choose Your Plan

- Once your email is verified, you will be asked to choose a plan:
  - Free Plan: Suitable for beginners, allows unlimited public and private repositories.
  - Paid Plans: Provide additional features for advanced use.
- Select the Free plan if you are just starting and click “Continue”.

## 9. Access Your GitHub Dashboard

- After completing these steps, your GitHub account will be created.
- You will be directed to your GitHub dashboard (home page), where you can:
  - Create a new repository
  - Explore other projects
  - Edit your profile details

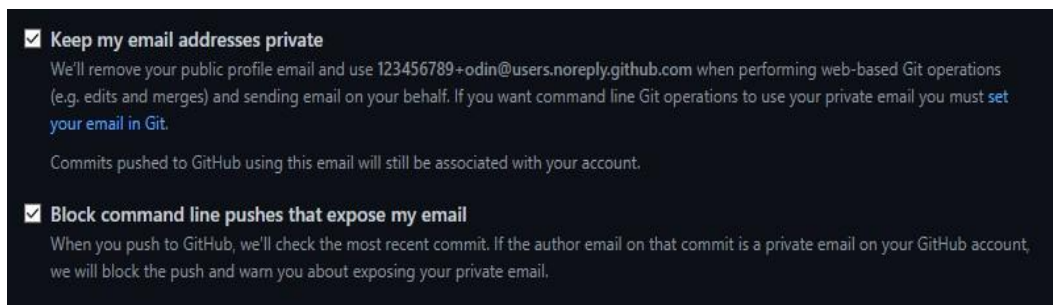
### Tips After Creating Your Account (Optional)

1. **Add a Profile Picture:** Go to your profile settings and upload a profile picture.
2. **Enable Two-Factor Authentication:** This adds an extra layer of security to your account.
3. **Install Git:** If you plan to work locally with repositories, install Git on your computer.
4. **Learn Basic GitHub Navigation:** Explore tabs like **Repositories**, **Pull Requests**, and **Issues**.



## Creating Repositories

1. Navigate to to <https://github.com/> and create an account, if you **DO NOT HAVE AN ACCOUNT!**
2. During the account setup, it will ask you for an email address. Enter your email and other required details to complete the account creation process.
3. If you are privacy conscious, or just don't want your email address to be publicly available, make sure you tick the following two boxes on the Email Settings page after you have signed in:

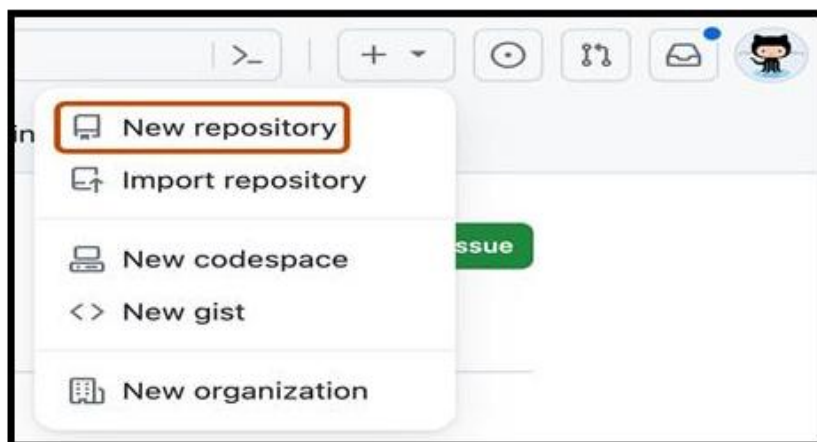


Source: Screenshot

Having these two options enabled will prevent accidentally exposing your personal email address when working with Git and GitHub.

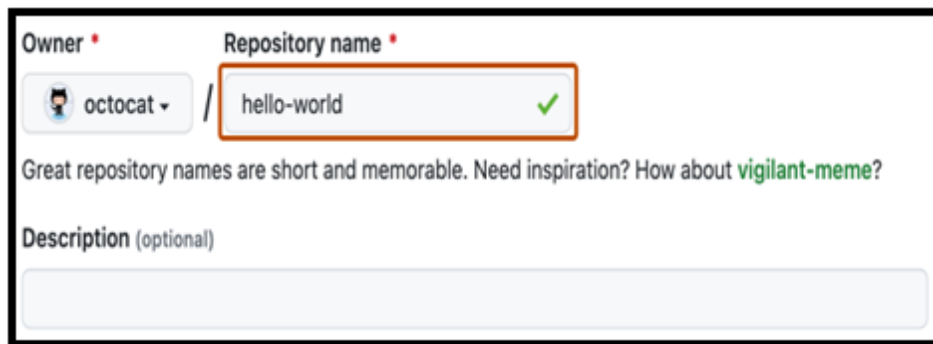
GitHub repositories store a variety of projects. In this guide, you'll create a repository and commit your first change.

1. In the upper-right corner of any page, select, then click New repository.



Source: Screenshot

2. Type a short, memorable name for your repository. For example, "hello-world".



Source: Screenshot

3. Optionally, add a description of your repository. For example, "My first repository on GitHub."
4. Choose a repository visibility.
5. Select Initialize this repository with a README.
6. Click Create repository.

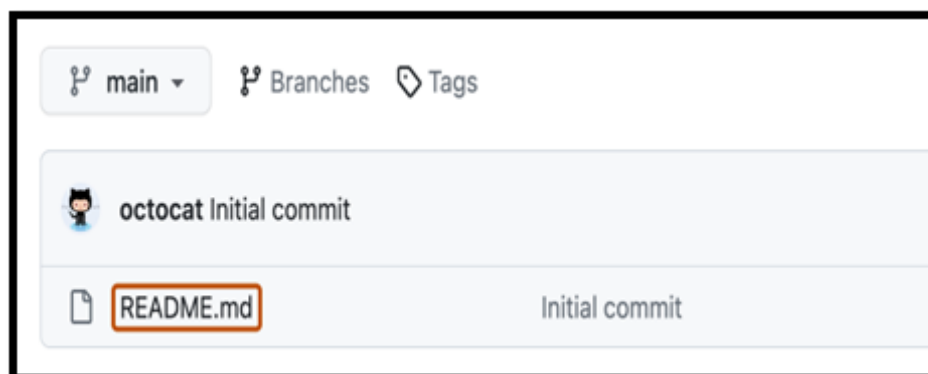
### Commit the change

A commit is like a snapshot of all the files in your project at a particular point in time.

When you created your new repository, you initialized it with a README file. README files are a great place to describe your project in more detail, or add some documentation such as how to install or use your project. The contents of your README file are automatically shown on the front page of your repository.

Let's commit a change to the README file.

1. In your repository's list of files, select README.md.



Source: Screenshot

2. In the upper right corner of the file view, click to open the file editor.



Source: Screenshot

3. In the text box, type some information about yourself.
4. Above the new content, click Preview.



Source: Screenshot

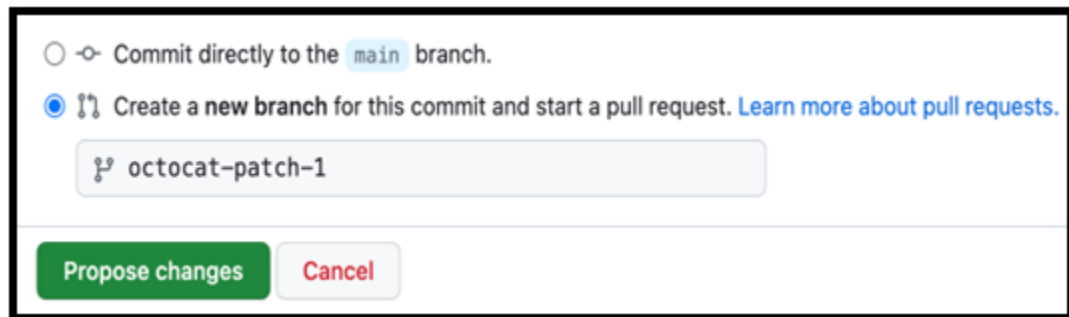
5. Review the changes you made to the file. If you select Show diff, you will see the new content in green.



Source: Screenshot

6. Click Commit changes...
7. In the "Commit message" field, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "Creating a commit with multiple authors."

8. Below the commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit and then create a pull request. For more information, see "Creating a pull request."
9. Click Commit changes or Propose changes.



Source: Screenshot

### Note:

Before starting the execution of commands, you need to make sure you have the PAT to access GitHub through cmd.

### Step 1: Generate a Personal Access Token

- Go to GitHub → Settings (top-right profile menu)
- Go to Developer settings → Personal access tokens → Tokens (classic)
- Click Generate new token → Generate new token (classic)

### Select scopes:

- Check repo (for full control of private/public repositories)
- Click Generate token
- Copy the token (you will not see it again)

### Step 2: Use Username and Tokens

Enter the username and password:

- Username: Your GitHub **username**
- Password: Paste the **Personal Access Token**

### Check the correct remote URL from GitHub

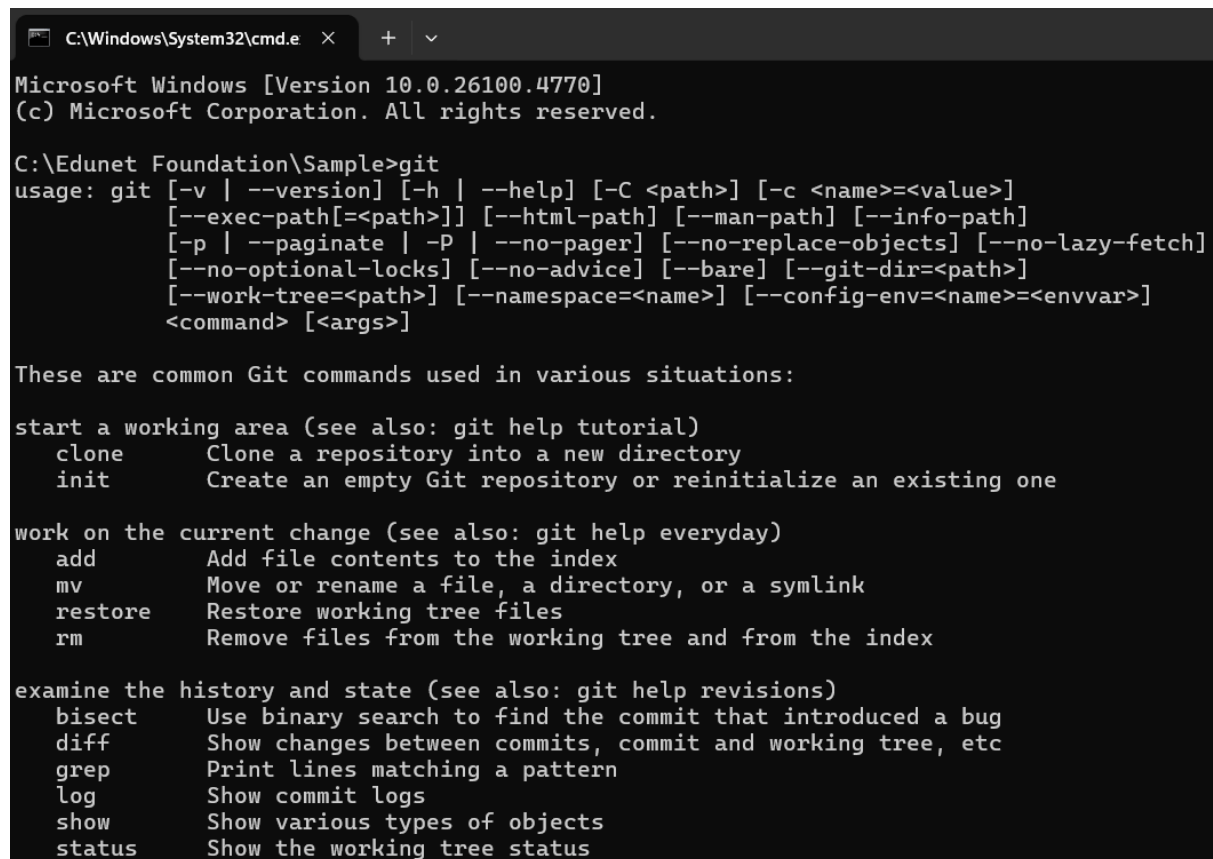
- Go to your repository on GitHub and create a new repository
- Click the green Code button
- Copy the HTTPS URL, which will look like:  
[https://github.com/<username>/<Repository\\_name>](https://github.com/<username>/<Repository_name>)

## Commands

Before we initiate the commands create a new folder. Here we have created the folder **Sample** within which we will execute the commands.

### 1. git

git is the main command-line tool for interacting with Git. It is used with subcommands (like init, add, commit, push) to manage your repository. Just typing git shows you a list of common Git commands.



```
C:\Windows\System32\cmd.e  X  +  v

Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Edunet Foundation\Sample>git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
          [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]
          [--work-tree=<path>] [--namespace=<name>] [--config-env=<name>=<envvar>]
          <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

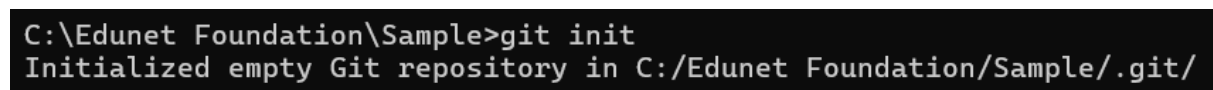

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index


examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status
```

Source: Screenshot

### 2. git init

git init creates a new empty Git repository in the current folder. After this, Git will start tracking changes in that folder.



```
C:\Edunet Foundation\Sample>git init
Initialized empty Git repository in C:/Edunet Foundation/Sample/.git/
```

Source: Screenshot

### 3. git add

git add tells Git to stage files (mark them for the next commit). You can add a single file, multiple files, or all files in the folder. Make sure to have the file with the same name present within the folder.

**Commands:** To add a specific file:

```
C:\Edunet Foundation\Sample>git add trial.txt
```

Source: Screenshot

To add all the files present in the folder

```
C:\Edunet Foundation\Sample>git add .
```

Source: Screenshot

### 4. git commit

git commit saves your staged changes into the repository with a message describing the changes. This creates a version (snapshot) of your code.

```
C:\Edunet Foundation\Sample>git commit -m "Hello World!"
[master (root-commit) d274ca0] Hello World!
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 trial.txt
```

Source: Screenshot

## 5. git push

git push uploads your commits to a remote repository (like GitHub). Before you can push, you need to have:

A remote repository (for example, on GitHub)  
Added the remote using git remote add Sample <URL>  
Pulled/pushed once to set the upstream branch (if first time)

Commands:

First time (if remote is not yet added):

```
C:\Edunet Foundation\Sample>git remote add Sample https://github.com/S-Edunet/Trial.git
C:\Edunet Foundation\Sample>git branch -M main
C:\Edunet Foundation\Sample>git push -u origin main
Username for 'https://github.com': S-Edunet
Password for 'https://S-Edunet@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 310 bytes | 77.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/S-Edunet/Trial.git
   ec5cfb4..4ff56d4  main -> main
branch 'main' set up to track 'origin/main'.
```

## Streamlit Community Cloud

### Creating Account on Streamlit Community Cloud

#### 1. Open the Streamlit Community Cloud Website

- Open a web browser (Google Chrome, Edge, or Firefox).
- Go to <https://streamlit.io/cloud> or <https://share.streamlit.io/>.
- This will take you to the Streamlit Community Cloud sign-up/sign-in page.

#### 2. Click on “Sign up”

- Look for the “Sign in” or “Sign up” option.
- If you are visiting for the first time, click “Sign up”.

#### 3. Choose a Sign-Up Method

Streamlit Community Cloud allows sign-up using:

- GitHub account (most commonly used)

- Google account
- Email/Password (if shown)

**Recommendation:** Use your **GitHub account** because Streamlit deploys apps directly from GitHub repositories.

#### 4. Authorize Streamlit (if using GitHub)

- If you choose GitHub:
  - You will be redirected to GitHub for authorization.
  - Sign in with your GitHub credentials if prompted.
  - Click “Authorize Streamlit” to allow Streamlit to access your repositories (so that you can deploy apps).
- If you choose Google, you will be redirected to a Google login page and asked for permissions.

#### 5. Set Up Your Streamlit Profile

- After authorization, you may be asked to:
  - Confirm your **email address**.
  - Choose your **display name** or organization (optional).
  - Accept the **terms and conditions**.

#### 6. Verify Your Email

- If an email verification step appears:
  - Open your email inbox.
  - Find the verification mail from Streamlit.
  - Click on the verification link.

#### 7. Access the Streamlit Dashboard

- Once sign-up is complete, you will be redirected to your Streamlit Community Cloud dashboard.
- From here, you can:
  - Deploy a new app (by connecting a GitHub repository containing your Streamlit project).
  - View and manage existing apps.

#### 8. Optional Post-Signup Actions

1. **Connect GitHub Repositories:** Ensure your app code is in a public GitHub repository.



2. **Check Resource Limits:** Free tier allows a limited number of apps and compute resources.
3. **Explore Documentation:** Learn how to structure your Streamlit project for deployment.

## File Organization for Community Cloud App

### 1. Create a Project Folder on Your Computer

- Make a new folder for your Streamlit project.
- Give it a short and meaningful name (e.g., my\_streamlit\_app).
- This folder will contain all the files required for your app.

### 2. Create Your Main Python File

- Inside this folder, create your main Python file.
- Name it clearly, for example:
  - streamlit\_app.py (recommended) or app.py
- Streamlit Community Cloud will look for this file during deployment.
- This file must contain your Streamlit code (starting with import streamlit as st).

### 3. Prepare a requirements.txt File

- This file tells Streamlit Community Cloud which Python libraries your app depends on.
- Create a text file named requirements.txt (all lowercase, no extra characters).
- Inside, list each required library on a separate line. Example:  

```
streamlit  
  
pandas  
  
matplotlib
```
- Save it in the same folder as your app file (root of the project).

### 4. Add Other Necessary Files (if needed)

- If your app uses datasets, images, or configuration files:
  - Create subfolders like data, images, etc.
  - Keep the structure clean and simple.

## 5. Initialize a Git Repository Locally (Optional, but Best Practice)

- Open a terminal/command prompt in the project folder.

- Run:

```
git init
```

```
git add .
```

```
git commit -m "Initial commit for Streamlit app"
```

## 6. Push the Folder to GitHub

- Create a new repository on **GitHub** (preferably public).

- Connect your local folder to the repository:

```
git remote add origin https://github.com/your-username/repository-name.git
```

```
git branch -M main
```

```
git push -u origin main
```

- Ensure that your main .py file and requirements.txt file are visible in the root of the repository on GitHub.

## 7. Verify File Structure on GitHub

Your repository structure on GitHub should look like this:

- repository-name/
- streamlit\_app.py
- requirements.txt
- data/
  - dataset.csv

## 8. Deploy on Streamlit Community Cloud

- Go to <https://streamlit.io/cloud>.
- Click "New app".
- Select the GitHub repository where you uploaded your files.
- Choose the main branch and main Python file (streamlit\_app.py).
- Click Deploy.

## 9. Best Practices for File Organization

- **Keep the app simple:** Avoid unnecessary files in the root folder.
- **Correct file names:** Use lowercase letters, underscores, and avoid spaces.
- **Update requirements.txt:** Each time you add a new Python library, update this file and push changes to GitHub.

## Key Notes

- Without requirements.txt, Streamlit may fail to deploy your app.
- Always make sure your main Python file is at the top level of your repository.

## Connect GitHub Account and Deploying App

### 1. Sign in to Streamlit Community Cloud

- Open a web browser and go to <https://streamlit.io/cloud>.
- Click Sign in.
- Use the same method you used while creating your Streamlit account (GitHub or Google).  
*If you originally signed up using Google, you will still be able to connect GitHub separately during this process.*

### 2. Go to the App Deployment Page

- Once logged in, click on “New app” or “Deploy an app”.
- If this is your first deployment, Streamlit will ask you to connect a GitHub account.

### 3. Authorize Streamlit to Access GitHub

- Click on Connect to GitHub.
- You will be redirected to the GitHub login page (if you are not already signed in).
- Enter your GitHub username and password.
- GitHub will show a permissions screen asking if you want to allow Streamlit to access your repositories.
- Click Authorize Streamlit.

### 4. Select Repositories for Streamlit

- GitHub may ask if you want Streamlit to:
  - Access all repositories, or
  - Access only selected repositories.
- Choose only selected repositories if you prefer privacy.  
Then, specifically select the repository that contains your Streamlit app.

### 5. Return to Streamlit Community Cloud

- After authorization, you will be redirected back to the Streamlit interface.
- Now, you will be able to see a dropdown list of your GitHub repositories.

## 6. Deploy Your First App

- Select:
  - The repository containing your Streamlit app,
  - The branch (usually main),
  - And the main Python file (streamlit\_app.py).
- Click Deploy.

## 7. Verify Connection

- Once connected, Streamlit will remember your GitHub account.
- For future deployments, you will not need to re-authorize unless permissions change.

## 8. Best Practices After Connecting GitHub

1. Keep your repository updated: Push changes from your local machine to GitHub whenever you update your app.
2. Public vs. Private Repositories:
  - Public repositories are simpler to deploy.
  - For private repositories, make sure Streamlit has permission to access them.
3. Branch Management: Deploy from a stable branch (commonly main).

## Key Notes

- Connecting GitHub is mandatory for deploying apps because Streamlit Community Cloud directly fetches code from a GitHub repository.
- If you do not see your repository after connecting, check:
  1. If it is private and not selected during authorization.
  2. If the main file and requirements.txt exist in the root of the repository.

## References

- Atlassian. (n.d.). What is Git | Atlassian Git Tutorial.  
<https://www.atlassian.com/git/tutorials/what-is-git>
- GeeksforGeeks. (2025, July 16). A Beginners guide to Streamlit.  
GeeksforGeeks. <https://www.geeksforgeeks.org/python/a-beginners-guide-to-streamlit/>
- Git. (n.d.). <https://git-scm.com/>
- Streamlit • A faster way to build and share data apps. (n.d.).  
<https://streamlit.io/>
- Wikipedia contributors. (2025, July 24). GitHub. Wikipedia.  
<https://en.wikipedia.org/wiki/GitHub>
- Working with GitHub in VS Code. (2021, November 3).  
<https://code.visualstudio.com/docs/sourcecontrol/github>