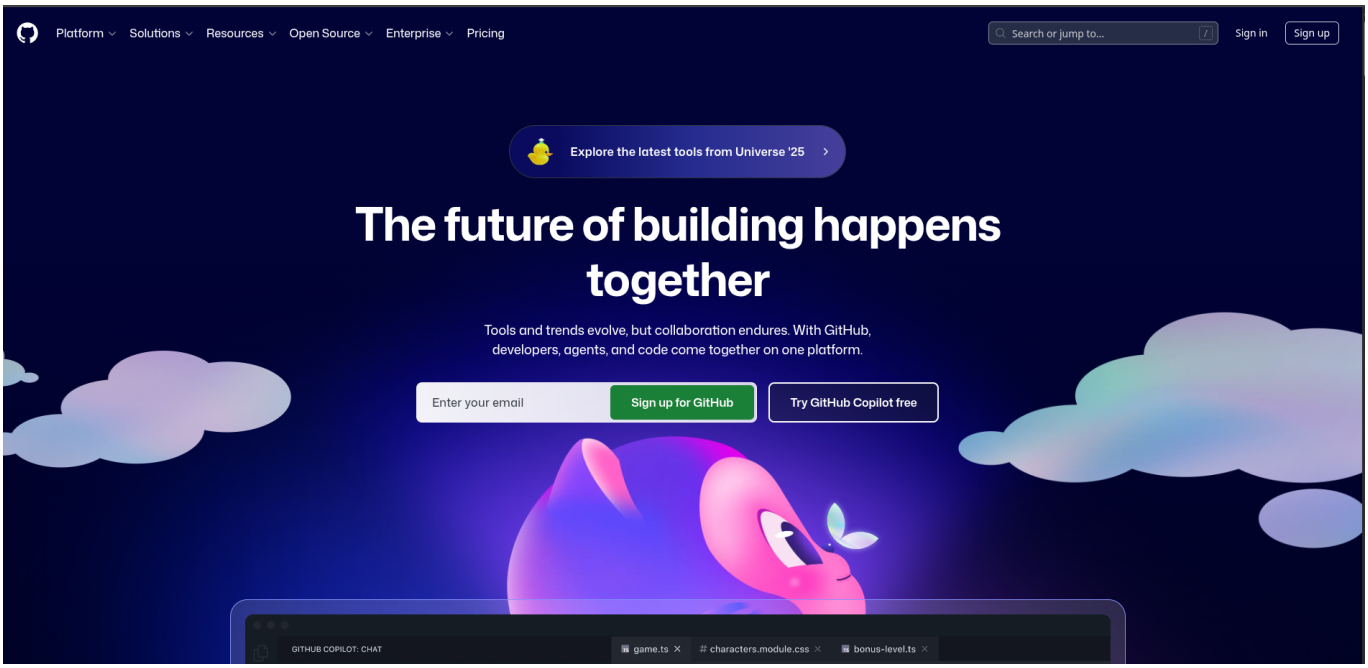
Introduction to Version Control with Git and GitHub

More information can be found on the official GitHub documentation.



GitHub Landing Page

What we will cover:

- Fundamentals of version control
    - Principles of version control systems
    - Differences between centralized and distributed version control
- Introduction to Git and GitHub
    - Git introduction
        - What is Git?
        - Prerequisites and installation of Git on different operating systems
        - Basic configuration of Git (user.name, user.email)
    - GitHub introduction
        - What is GitHub and why use it?
        - Creating a GitHub account
    - Connecting Git with GitHub
        - Setting up SSH keys for secure communication with GitHub
            - Generating SSH keys

        - Adding SSH keys to your GitHub account
    - Working with Personal Repositories
        - Initialization of a new Git repository (git init)
        - Verifying the status of your repository (git status)
        - Adding changes to the staging area (git add)
        - Committing changes to the repository (git commit)
        - Saving changes to a remote repository on GitHub (git push)
        - Calling up Git help and self-help options

Fundamentals of Version Control

- Imagine you are working on a document and would like to:
    - Keep track of changes you make
    - Revert to previous versions if needed
    - Invite others to work with you
    - Accept changes from others without overwriting your work
- Version control systems (VCS) help you achieve these and more.

What is Version Control?

- A system that records changes to a file or set of files over time.
- Think of it as an intelligent notebook that keeps track of every change you make.
- Allows you to revert to previous versions, compare changes, and collaborate with others.
- A version control system tracks four main things:
    - **WHAT was changed** (the content): *line 5 in document X was deleted, one paragraph in file Y was modified, image Z was added*
    - **WHO made the change** (the author):*the username of the person who made the change*
    - **WHEN the change was made** (the timestamp): *date and time of the change*
    - **WHY the change was made** (the commit message): *a brief description of the reason for the change written by the author*

Differences Between Centralized and Distributed Version Control

- There are two basic architectures that version control systems can have.
    - Centralized Version Control System (CVCS)
    - Distributed Version Control System (DVCS)
- Both follow the above principles, but in different ways.

Centralized Version Control System (CVCS)

- Imagine a library with a single, large bookshelf in the middle of the room
- that's the **central server**.
- On this shelf is the **ONLY** original copy of the entire project history.

- When you want to work on something,
    - you go to this shelf,
    - borrow a copy of the latest version of a file (a "checkout").
    - and work on your copy at your desk.
- When you're done,
    - you return the file
    - submit your changes ("commit").
    - Your changes are saved directly on the central server.
- **Advantages:**
    - Very easy to understand and manage.
    - There's always a clear understanding of what the "official" version is.
- **Disadvantages:**
    - **Single Point of Failure:**
        - If the central server fails (e.g., power outage, hard drive crash), no one can work.
        - No one can commit, compare versions, or retrieve changes.
        - the entire project history is lost.
    - **Online Requirement:**
        - For almost all actions (committing, viewing history, creating branches), you need a connection to the central server.
        - Offline work (e.g., on a train) is very limited.
- **Famous examples:** Subversion (SVN), CVS, Perforce.

Distributed Version Control System (DVCS)

- Now imagine another system.
- Instead of a central library, there is a main server, but every developer receives not just the latest version when checking out, but a **complete 1:1 copy of the entire library** for their own desk.
- Every developer thus has the complete project history locally on their own computer.
- You work on your project and make commits.
    - These commits are stored in your *local* copy of the repository.
    - You can create branches, view history, compare versions – all lightning fast and without an internet connection.
- Only when you want to share your work with others, can you
    - connect to the main server and
    - push your changes there ("push").
    - To receive others' changes, you fetch them from the server ("pull").
- **Advantages:**

    - **No Single Point of Failure:**
        - If the main server fails, it's annoying but not tragic.
        - Every developer has a complete copy of the project history on their computer.
        - The project can simply be restored from one of these copies.
    - **Excellent Offline Capability:**
        - You can sit on a plane and make dozens of commits, create new branches, and organize all your work.
        - Only when you have a connection again do you share your work.
    - **Speed:**
        - Since most operations (commit, view history) happen locally, they're extremely fast.
        - There's no network delay.
    - **Flexible Branching & Merging:**
        - Creating branches in a DVCS is extremely easy and fast, greatly encouraging experimental and parallel development workflows.

- **Disadvantages:**

    - The learning curve is initially a bit steeper because you have to understand the distinction between the local and remote repository (e.g., concepts like push & pull).

- **Famous examples: Git** (by far the most popular), Mercurial.

Introduction to Git and GitHub

Git Introduction

- **What is Git?**
    - Git is a distributed version control system (DVCS) created by Linus Torvalds in 2005.
    - It allows multiple developers to work on the same project simultaneously without overwriting each other's changes.
    - Git tracks changes to files, enabling you to revert to previous versions, compare changes, and collaborate effectively.

Prerequisites and Installation of Git

- Git is free and open-source software.
- It can be installed on various operating systems, including Windows, macOS, and Linux.
- Here is the official Git website for download and installation instructions: https://git-scm.com/install/
- **Installing Git on Linux:(Link to the official Git installation page)**
    - Open your terminal(ctrl + alt + t)
    - Check if you have Git installed with:

```
git --version
```

- If Git is not installed, you can install it using your package manager.
- Now update and upgrade your package lists:

```
sudo apt update
sudo apt upgrade
```

- Then install Git with:

```
sudo apt-get install git
```

- Verify the installation with:

```
git --version
```

- **Installing Git on Windows:**
  - Download the Git(and Git Bash) installer from https://gitforwindows.org/.
  - Run the installer and follow the setup instructions.

Basic Configuration of Git

- After installing Git, it's essential to configure your user information (The WHO that the version control needs).
- This information will be associated with your commits (the WHY part).
- This is a **ONE-TIME setup** when you first install Git on a new machine.
- You will need to provide **THE SAME EMAIL ADDRESS**that you use for your GitHub account.
- Open your terminal (or Git Bash on Windows) and run the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

- You can verify your configuration with:

```
git config --list
```

- If neccessary, set your defaut branch name to "main":

```
git config --global init.defaultBranch main
```

    - This ensures that new repositories you create will use "main" as the default branch name instead of "master".
    - This change came about to promote inclusive language in software development.
    - Many platforms, including GitHub, have adopted "main" as the default branch name for new repositories.
- Verifying your configuration again with:

```
git config --list
```

- You should see your user name and email listed among the configuration settings. For example mine looks like this:

```
user.email=myprivate@gmail.com
user.name=Ndimofor
init.defaultbranch=main
```

GitHub Introduction

- **What is GitHub and Why Use It?**
    - GitHub is a cloud-based platform where you can
        - store,
        - share,
        - and work together with others to write code or other text-based files using Git.
    - It provides a web interface for Git repositories, making it easier to manage and collaborate on projects.
    - GitHub offers additional features such as issue tracking, project management tools, and integrations with other services.
    - Think of it as a storage location for your code, markdown files, and other project-related documents that you can access from anywhere and share with others.

Creating a GitHub Account

- To use GitHub, you need to create an account.
- PLEASE USER YOU PERSONAL EMAIL ADDRESS FOR THIS!
- Open your personal Email inbox in a new tab to verify your account later.
- Make sure to remember your username and password.
- Follow this link to sign up

Connecting Git with GitHub

- **How Git and GitHub Work Together**

- Git tracks changes to your files locally on your computer (your *local* repository).
- GitHub is the online platform where you can store this Git repository (your *remote* repository).
- When you upload files to GitHub, you'll store them in a "Git repository."
- When you make changes to your files locally, you can use Git to track those changes.
- To update your remote repository on GitHub with your local changes, you "push" those changes to GitHub.
- For all of this to work smoothly, you need to set up a secure connection between your local Git installation and your GitHub account.
- We will do this using SSH keys.

Setting Up SSH Keys for Secure Communication with GitHub

- **What are SSH Keys?**
  - SSH (Secure Shell) keys are a pair of cryptographic keys used to authenticate your computer with GitHub securely.
  - They consist of a public key (which you share with GitHub) and a private key (which you keep secret on your computer).
  - Using SSH keys allows you to connect to GitHub without needing to enter your username and password every time you push or pull changes.
- **Generating SSH Keys (Linux Ubuntu)**
  - Open your terminal (or Git Bash on Windows).
  - Check if you already have SSH keys by running:

```
ls -al ~/.ssh
```

  - If you see files named `id_rsa` and `id_rsa.pub`, you already have SSH keys.
  - You can skip to adding the key to GitHub.
  - If not, generate a new SSH key pair with the following command (replace "your_email@example.com" with your actual email address):

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

    - When prompted to "Enter a file in which to save the key," press Enter to accept the default location.
    - When prompted for a passphrase, press Enter
    - When prompted to confirm the passphrase, press Enter again.
  - This will create a new SSH key pair in the `~/.ssh` directory.
- **Adding SSH Keys to Your GitHub Account**
  - Start the SSH agent in the background by running:

```
eval "$(ssh-agent -s)"
```

    - you should see a message like `Agent pid 59566`.

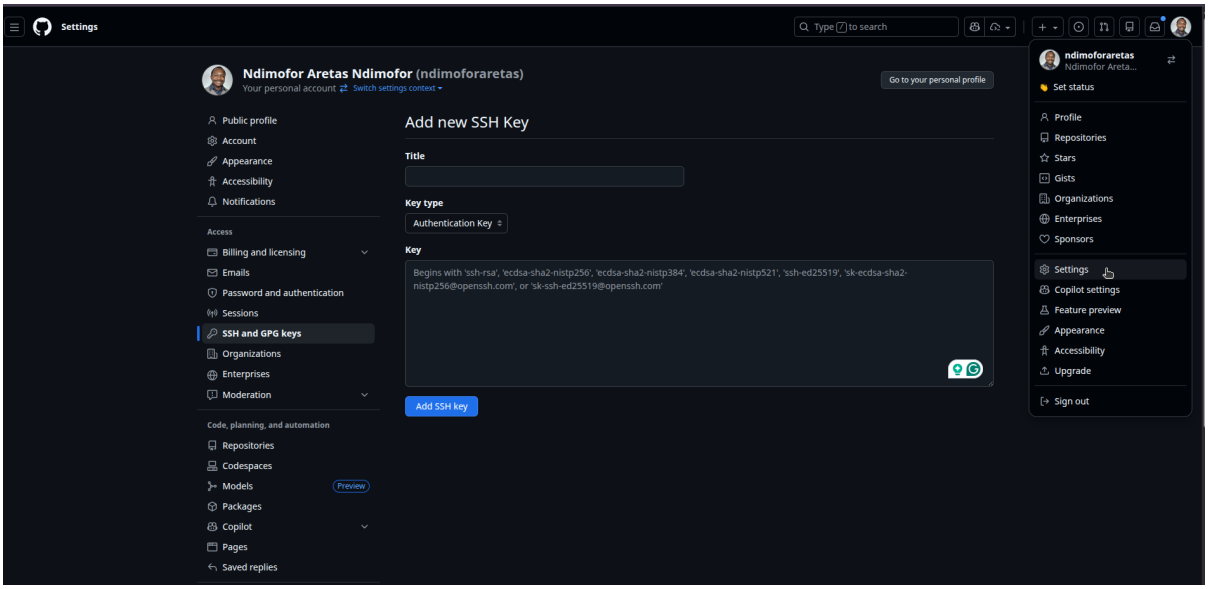
- Add your SSH private key to the SSH agent with:

```
ssh-add ~/.ssh/id_ed25519
```

- **Adding the SSH Key to Your GitHub Account**
  - Copy the contents of your public SSH key to your clipboard by running:

```
cat ~/.ssh/id_ed25519.pub
```

    - Select and copy the entire output (it starts with `ssh-ed25519` and ends with your email address).
  - Log in to your GitHub account.



SSH Key Addition Steps

  - Click on your profile picture in the top-right corner.
  - Go to **Settings** > **SSH and GPG keys** > **New SSH key**.
  - Give it a descriptive title, like "DCI Linux Laptop SSH Key".
  - Paste the copied key into the "Key" field.
  - Click **Add SSH key** to save it.
- ****Testing the SSH Connection**
  - To test if everything is set up correctly, run:

```
ssh -T git@github.com
```

  - You should see a message like:

```
Hi username! You've successfully authenticated, but GitHub does
not provide shell access.
```

    - This means your SSH key is correctly set up, and you can now securely connect to GitHub using Git.

Working with Personal Repositories

- There are two types of repositories you can work with on GitHub:
    - **Local Repository:**
        - This is the repository you create on your own computer.
        - You then have to push this local repository to GitHub to create a remote repository.
    - **Remote Repository:**
        - This is the repository you create directly on GitHub.
        - You then have to clone this remote repository to your computer to create a local repository.
- We will focus on creating a local repository first and then pushing it to GitHub to create a remote repository.
- For today we will do this using the VS Code Version Control Interface.