# Git and GitHub Essentials: A Crash Course for Version Control and Collaboration

## Introduction

**What is version control?**

A system that manages changes made to documents, code, or any digital content over time

**Why version control?**

Helps track, organize, and document modifications to files, enabling collaboration, history tracking, and error recovery

**Key features of version control:**

- **Recording changes:** Stores a historical record of every change made to a file.
- **Multiple users:** Allows multiple people to work on the same project together.
- **Branching:** Allows you to make different copies of your project to try out new ideas without messing up the main project.
- **Merge and Conflict Resolution:** Helps combine changes made by different people without causing conflicts. It's like ensuring that when two people edit the same document, their changes can peacefully coexist.
- **Rollback:** Allows you to easily go back to a previous version of your project, like using a "time machine" for your work. This is handy when you make a mistake or need to revisit an earlier state.
- **Access Control:** Lets you control who can view or modify your project. It's similar to deciding who can enter a club or a building, ensuring only the right people have access.
- **Large File Handling:** Manages large files efficiently, preventing them from slowing down your project. It's like a fast and organized filing cabinet for your digital documents.

**Benefits of version control:**

- **Collaboration:** Facilitates teamwork by merging changes from different contributors.
- **History:** Provides a complete history of changes, useful for troubleshooting and auditing.
- **Reversion (Going back version(s)):** Allows you to return to a previous version(s) of your work if needed.

**Why Git and GitHub?**

- **Git: Version Control**
    - **Easy tracking of changes:** Git makes it simple to see what you've changed in your project (e.g., editing a document).
    - **Efficient Collaboration:** Git allows you and your friends can work together on a project without messing up each other's work (e.g., you and your friend working on the same art project without messing up each other's work).
    - **Branching and Experimentation:** You can use Git to create branches (special spaces) to try new things in your project (e.g., having a sandbox to play around with new ideas safely).
    - **Reversion:** Git is like a "time machine" for your project, so if you make a mistake, you can easily go back to how things were before, just like hitting the "undo" button.
- **GitHub: Collaboration**
    - **Teamwork:** GitHub allows you to edit, add, and work on a project together, making sure you don't accidentally interfere with each other's work.
    - **Easy Access:** Your project is accessible from anywhere with an internet connection. It is like having a library where you can access your work from different computers.
    - **Backup and Security:** GitHub acts as a safe vault for your work, making sure it's secure and backed up.
    - **Community and Teamwork:** GitHub is a platform where a community of developers hosts, collaborates on, and shares their projects. You can learn from others, showcase your work, and contribute to their projects too.

## Git Basics

### Installation and Setup

1. Go to https://git-scm.com/downloads and download for your operating system
2. Locate the download in your "Downloads" folder, follow installation instructions (use default settings if you are unsure)
3. Once Git is installed, open your Terminal or Command Prompt
4. Once your Terminal or Command Prompt is open, configure your username and password

```
git config --global user.name [insert your name here]
```

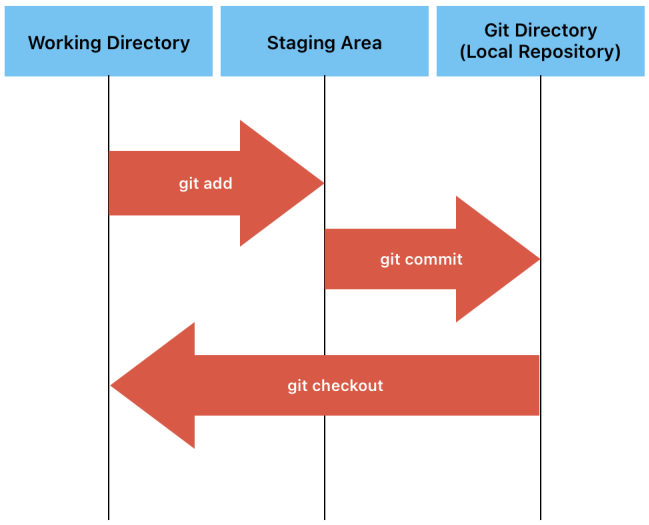
```
git config --global user.email [insert your email address here]
```

5. Verify your installation to make sure Git is installed correctly
   `git --version`

### Creating a Repository

1. Open the Terminal or Command Prompt
2. Navigate to Your Project's Folder (e.g., `cd /Desktop/my_project_folder`)
3. Initialize a Git Repository typing `git init` in the Command Prompt or Terminal

### The Three Stages in Git




**Working Directory:**

- The working directory is a spot where you can create, edit, and organize your files (e.g., working on your code in Visual Studio, Eclipse, IntelliJ, or any remote IDE).
- When you make changes to your project, those changes are initially in the working directory.
- Git constantly watches the working directory for any modifications.

**Staging Area:**

- Think of the staging area as a prep table in your workshop. It would be where you select and prepare the changes you want to save.
- When you are satisfied with certain changes in your working directory, you add them to the staging area. This is where you can decide what changes you want to keep.

**Repository:**

- The repository is like a history book of your project. It snapshots your project at different points in time.
- The repository retains a record of all your commits, making it possible to review and revert to earlier versions of your project.

**Basic Git Workflow**

- **Modify:** Make changes to your project in the working directory.
- **Stage:** Select the changes you want to save by adding them to the staging area.
- **Commit:** Record the staged changes in the repository, creating a new snapshot of your project.

**Checking the status**

To check the status, type in `git status` in the Terminal or Command Prompt.

- **Changes to be committed:** files that are in the staging area, ready to be committed
- **Changes not staged for commit:** modified files that need to be staged
- **Untracked files:** files Git is not currently tracking
- **Branch information:** current branch, and whether it is up to date

**Making Commits**

Making commits in Git means saving the changes you've staged in your project.

```
git commit -m "Description of the commit"
```

**Viewing Commit History**

To view the commit history of your Git repository, type in the `git log` command.

Each commit will show:

- A unique commit hash (SHA).
- The author's name and email.
- The date and time of the commit.
- The commit message, describing what was done in that commit.

**Working with Branches**

**Branch:** A separate playground where to work on a specific part of your project.

- Branching is essential because it enables you to experiment and develop new features without affecting the main project.
- Branches can also be used for collaboration, because one person can work on the same code or document without hurting the other person’s progress by working on different branches
- To create a new branch, type in the `git branch` command
    - An example would be if you were to create a branch named “backup,” you would create the branch from the command `git branch backup`.

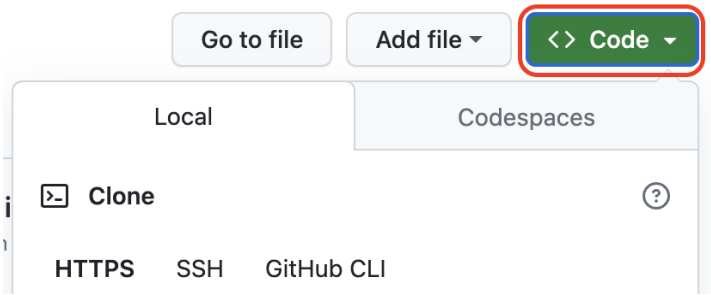
## Collaborating with GitHub

### Creating a GitHub Account

1. Visit https://github.com/ on your web browser.
2. Click the button that says “Sign Up” or “Sign Up for GitHub” button.
3. Provide Details (email address, username, strong password)
4. Complete the CAPTCHA to verify that you are not a robot
5. Choose a Plan (there is a free plan)
6. Verify your email
7. Congratulations! You are registered

### Pushing to GitHub

1. Ensure that you have created a repository on GitHub
2. Click on “<> Code” button, and find a link that begins with “https://github.com/” and ends with “.git”




3. Connect your local repository with this command below (Replace "your-username" with your GitHub username and "your-repo" with your repository name)

```
git remote add origin https://github.com/[your-username]/[your-repo].git
```

4. Push your local changes using this command below (Replace “main” with the name of your branch if it's a different name than “main”)

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

5. Authentication may be required. This means, you may be required to enter your GitHub username and password or use your SSH key for authentication.
6. After you push, check your changes to make sure they are saved into your repository.

### Pulling from GitHub

1. Go to your Command Prompt, Terminal, or Terminal in your IDE (VSCode, Eclipse, …).
2. Go into your local repository folder, which would be the folder you pushed to your remote repository (e.g., GitHub) by typing in `cd /[path/to/your/local/repository]`. Replace [path/to/your/local/repository] to the path to your local repository. This ensures your local copy is up to date with the remote repository (e.g., GitHub).

**Cloning a Repository**

- Cloning from a repository means creating a local copy of your remote repository.
- It downloads all the project files and version history, allowing you to work on the project locally.
- You need to specify the URL you want to clone from
- Here are the steps:
    1. **Get the Repository URL:** Click the "Code" button on the repository's page. Copy the URL provided (you can choose either HTTPS or SSH, depending on your preference).
    2. **Open Your Terminal or Command Prompt:** Open the terminal or command prompt on your computer.
    3. **Navigate to Your Desired Directory:** Use the cd command to navigate to the directory where you want to clone your forked repository. For example, `cd /[path/to/your/desired/directory]`, where you can replace "[path/to/your/desired/directory]" with the name of the directory you would like to clone to.
    4. **Clone the Repository:** Type in git clone `https://github.com/[your-username]/[your-repo].git` and you will see a downloaded folder of the files and directories you cloned. Replace "your-username" with your GitHub username and "your-repo" with the repository name you would like to clone.
    5. **Press Enter:** Hit Enter, and Git will clone the repository to your local computer.

**Forking a Repository**

- Forking on GitHub is like making your own copy of someone's project, which you can edit without changing the original.
- To fork a repository on GitHub, follow these steps:
    1. **Go to the Repository:** Visit the GitHub repository you want to fork.
    2. **Click "Fork":** In the top right corner of the repository's page, click the "Fork" button. This action creates a copy of the repository under your GitHub account.
    3. **Select the Account:** If you belong to multiple organizations or have multiple accounts, select the account where you want to fork the repository.
    4. **Wait for Fork:** GitHub will duplicate the repository, and you will be redirected to your own forked copy.

**Cloning from a Forked Repository**

1. **Find Your Forked Repository:** Go to your forked repository on GitHub.
2. **Get the Repository URL:** Click the "Code" button on the repository's page. Copy the URL provided (you can choose either HTTPS or SSH, depending on your preference).
3. **Open Your Terminal or Command Prompt:** Open the terminal or command prompt on your computer.
4. **Navigate to Your Desired Directory:** Use the cd command to navigate to the directory where you want to clone your forked repository. For example, `cd /[path/to/your/desired/directory]`, where you can replace "[path/to/your/desired/directory]" with the name of the directory you would like to clone to
5. **Clone the Repository:** Type in git clone `https://github.com/[your-username]/[your-forked-repo].git` and you will see a downloaded folder of the files and directories you cloned. Replace "your-username" with your GitHub username and "your-forked-repo" with the name of your forked repository you would like to clone.
6. **Press Enter:** Hit Enter, and Git will clone the repository to your local computer.

## Advanced Tips

**Merging Branches**

- Merging is how you blend changes from one branch into another. You can choose which parts or all of the code from one branch to keep, or you can use all of the code from the other branch.
- Here is a sample Command Line or Terminal prompt for merging the "demo" branch into the "main" branch:

```
git checkout main  # Switch to the main branch
git merge demo  # Merge demo into main
```

**Pull Requests**

- A Pull Request on GitHub is like raising your hand to propose code changes. It's a way to present your work, gather input from others, and, when approved, merge those changes into the main code.
- Here are steps for a successful pull request:
    1. **Create a Pull Request:** You propose changes.
    2. **Review and Feedback:** Others review and comment.
    3. **Address Comments:** Make necessary changes.
    4. **Approval:** Get approval from a reviewer.
    5. **Pass Tests:** Ensure automated tests pass.
    6. **Merge:** A collaborator merges the changes if everything is good.