Beginner's Guide to Git and GitHub

Written by Berkeley Conkling (With the help of ChatGPT)

Note to readers:
Anytime Git bash is being used to run commands, a linux terminal can also run all of these commands. If you are on linux, this guide should still apply to you.
Also, this guide follows how to use git bash. If you are using a visual tool like the github app, sourcetree, or some other app, the concepts are the same (pulling, branching, pushing etc) you just need to determine what the equivalent of the shown command is in your app. With this said, I would recommend trying to learn Git Bash since it helps you understand how git works at a deeper level than a GUI git app.

Getting Started with GitHub and Git Bash

As collaborative platforms for version control, Git and GitHub have become essential in modern software development, allowing multiple individuals to work on projects simultaneously without conflict. Understanding how to utilize these tools is crucial for anyone looking to engage in both personal and professional coding projects.

One of the foundational concepts in Git is the workflow. A well-defined Git workflow helps developers manage changes systematically, ensuring that the project remains organized and that all contributions are tracked. Git workflows can vary based on the needs of the project, but they generally involve creating branches, making changes, and submitting those changes for review.

Creating branches is a core aspect of using Git effectively. A branch allows developers to work on features or fixes independently from the main codebase (often referred to as the "main" or "master" branch). This separation not only keeps the main branch stable but also provides a safe space to experiment with new ideas. Once changes are made and tested on a branch, they can be merged back into the main branch.

After making changes in your branch, the next step is to submit a pull request. A pull request is a request to merge your changes into the main project. This process encourages code review and discussion among team members, ultimately leading to better quality code. It allows for feedback and collaboration, which are vital components of successful software development.

By familiarizing yourself with these concepts and tools, you will be equipped to manage your projects more effectively and contribute to collaborative efforts with confidence.

What is Git and GitHub?

Git is a distributed version control system that enables developers to track changes in their code over time. Created by Linus Torvalds in 2005, Git allows multiple developers to work on the same project concurrently without overwriting each other's work. This system is essential for managing code, as it records every modification made to a project, including who made the change and when it happened. With Git, developers can easily revert to previous versions of their code, compare different revisions, and maintain a complete history of their project, providing a clear audit trail.

Installing Git and Git Bash

Installing Git and Git Bash is a straightforward process that enables you to manage your code repositories effectively. Follow these step-by-step instructions to get started.

Step 1: Visit the Official Git Website

Begin by navigating to the official Git website at git-scm.com. This site contains the latest versions of Git for various operating systems and provides helpful resources.

Step 2: Download Git for Your Operating System

Once on the website, you will see a prominent download button that detects your operating system automatically. Click the button to download the appropriate installer for Windows, macOS, or Linux. If you are using Linux, you may prefer to use your package manager for installation (e.g., `apt` for Ubuntu).

Step 3: Run the Installer

After the download completes, locate the installer file and double-click to run it. Follow the on-screen prompts to proceed with the installation. It is generally recommended to keep the default settings, which include options for using Git from the command line and enabling Git Bash.

Step 4: Configure the Installation Options

During installation, you'll be prompted to select options for adjusting your PATH environment. If you're unsure, choose the option that allows you to run Git from the Windows Command Prompt. You may also want to enable the option to use the OpenSSH authentication.

Step 5: Complete the Installation

Continue through the installation process by clicking "Next" until the installation is complete. Once finished, you can choose to launch Git Bash immediately.

Step 6: Open Git Bash

To access Git Bash, you can either launch it directly from the installer or find it in your Start Menu (Windows) or Applications folder (macOS). Git Bash provides a command-line interface where you can use Git commands to manage your projects efficiently.

With Git and Git Bash installed, you are now ready to start using version control to manage your code repositories and collaborate effectively with others.

Basic Bash/Linux Commands

Bash is a command-line interface that allows users to interact with the operating system through text-based commands. In Git Bash, a popular terminal for Windows users, several basic commands are essential for navigating directories and managing files. Here, we will explore some fundamental Bash commands: `pwd`, `ls`, `cd`, `mkdir`, `touch`, `rm`, and `clear`.

`pwd`

The command `pwd` stands for "print working directory." When executed, it displays the current directory you are in within the file system. This command is helpful for keeping track of your location while navigating through directories.

`ls`

The `ls` command lists the files and directories in your current location. By default, it shows only the names of files and folders. You can enhance its functionality with options, such as `ls -l` for detailed information or `ls -a` to include hidden files in the output.

`cd`

The `cd` command, short for "change directory," allows you to navigate between directories. For example, typing `cd Documents` will move you into the Documents directory. To go back to the previous directory, you can use `cd ..`. To return to your home directory, simply type `cd` without any arguments.

`mkdir`

The `mkdir` command is used to create a new directory. For instance, if you want to create a folder named "Projects," you would type `mkdir Projects`. This command helps organize files by allowing users to create structured directories.

`touch`

The `touch` command is employed to create a new, empty file or update the timestamp of an existing file. For example, `touch notes.txt` creates a new text file named "notes.txt" in the current directory, ready for you to edit.

`rm`

The `rm` command removes files or directories. For instance, `rm file.txt` deletes "file.txt." Be cautious when using this command, as deleted files typically cannot be recovered. To remove a directory, you need to use `rm -r directory_name`, which recursively deletes the directory and its contents.

`clear`

Finally, the `clear` command is a simple yet effective way to refresh the terminal display, removing all previous commands and output from view. This can help maintain a tidy workspace, especially when working with long command histories.

By mastering these basic Bash commands, users can efficiently navigate their file systems and manage files in Git Bash, forming a solid foundation for more advanced operations.

Setting Up Git

Configuring your Git identity is an essential step after installing Git, as it allows you to associate your commits with your name and email. This information is important for collaboration and tracking contributions made to a project. Below are the step-by-step commands to set up your Git user information.

Step 1: Open Git Bash

Begin by launching Git Bash on your computer. You can find it in your Start Menu (Windows) or Applications folder (macOS). Once opened, you will see a command-line interface ready for input.

Step 2: Set Your User Name

To set your user name, enter the following command:

```
git config --global user.name "Your Name"
```

Replace `"Your Name"` with your actual name. This command tells Git to record this name as the author of your commits.

Step 3: Set Your Email Address

Next, you need to configure your email address. Input the following command:

```
git config --global user.email "you@example.com"
```

Make sure to replace `"you@example.com"` with your actual email address. Similar to your name, this email will be associated with your commits.

Step 4: Verify Your Settings

To ensure that your settings have been correctly applied, you can verify your configuration by running:

```
git config --global --list
```

This command will display a list of all global configuration settings, including your user name and email. You should see output similar to the following:

```
user.name=Your Name

user.email=you@example.com
```

Conclusion

By completing these steps, you have successfully set up your Git identity. This configuration is crucial for maintaining accurate records of contributions and fostering collaboration in your development work.

Working with a GitHub Repository

Working with a GitHub repository involves several key activities that allow you to collaborate effectively and manage your code changes. Below are detailed instructions for each core activity: cloning a repository, creating a new branch, making changes, committing changes, pushing changes, and submitting a pull request.

Cloning a Repository

To begin working with a GitHub repository, you first need to clone it to your local machine. To do this, follow these steps:

1. **Navigate to the repository on GitHub:** Open your web browser and go to the GitHub repository you want to clone.
2. **Copy the repository URL:** Click on the green "Code" button and copy the HTTPS URL or SSH URL provided.
3. **Open Git Bash:** Launch Git Bash on your computer.

4. **Run the clone command:** In Git Bash, type the following command, replacing `<URL>` with the copied repository URL: “git clone <URL>”

This command creates a local copy of the repository on your machine.

Creating a New Branch

Once you have cloned the repository, it’s best practice to create a new branch for your changes. Here’s how:

1. **Navigate to the repository directory:** Use the `cd` command to change into the cloned repository directory:
2. **Create a new branch:** Use the following command to create a new branch: “git switch -c <name>”

This command creates a new branch and switches you to the new branch, allowing you to work independently of the main branch. If you want to switch to a branch without creating a new branch, the command is the same except without the “-c”

Making Changes

With your new branch created, you can make the necessary changes to the code. Use your preferred text editor or IDE to edit files as needed.

Committing Changes

After making changes, you need to commit them to your branch:

1. **Stage your changes:** Use the following command to stage all modified files: “git add .” (The period is part of the command) or if you want to only stage a certain file “git add <filename>”
2. **Commit your changes:** Next, commit your changes with a descriptive message: “git commit -m <message>” note that the message should be in quotation marks.

Pushing Changes

To share your changes with the remote repository, push your branch:

1. **Push to the remote repository:** Run the following command: “git push origin <branch name>”

This command uploads your branch and its commits to GitHub.

Submitting a Pull Request

Finally, to propose your changes to the main codebase, submit a pull request:

1. **Go to the repository on GitHub:** Navigate back to the GitHub repository in your web browser.
2. **Open the Pull Requests tab:** Click on the "Pull Requests" tab.
3. **Create a new pull request:** Click the "New Pull Request" button and select your branch from the dropdown menu.
4. **Review and submit:** Add a title and description, then click "Create Pull Request."

Sometimes there will be collisions, where your version of the code is not compatible with the remote version you are trying to merge into. These collisions will be highlighted in the file, and must be resolved before a pull request can go through.

Next Steps:

Someone with review privileges (For example a Project Manager) will review the pull request. If the pull request is good to merge, the reviewer will sign off on it and merge it. If there are parts that still need work, it will be rejected with a descriptive message on what to improve to get approved and merged.

Final Note:

Git can be very confusing at first. Some helpful resources are listed below:

- Learn git branching visually: https://learngitbranching.js.org/?locale=en_US
- Code academy Git and GitHub course: https://www.codecademy.com/learn/learn-git
- ChatGPT: Extremely helpful if you can't remember a command you want to use, want to learn new commands, or have a collision or other issue you need help resolving. **Obviously the club does not condone cheating in any form, only use chatGPT if you are allowed to for school purposes.** For the purpose of learning git in this club, chatGPT is completely ok. In fact it would be recommended.
- Asking in Discord!: Always feel free to reach out in discord to a PM or in your teams respective channels to ask questions to your peers.