A group of people looking at a rocket launch

Description automatically generatedA yellow line drawing of a hand holding a megaphone

Description automatically generatedA yellow line drawing of a hand holding a megaphone

Description automatically generatedA yellow line drawing of a hand holding a megaphone

Description automatically generatedA yellow line drawing of a hand holding a megaphone

Description automatically generated

**Sit With Git By John Smith Smith**

A yellow line drawing of a hand holding a megaphone

Description automatically generated

Contents

[Introduction 2](#_Toc187271169)

[Basic Git Commands 2](#_Toc187271170)

[Initialize Git Repository (git init) 2](#_Toc187271171)

[What is the Staging Area in Git (git add) 3](#_Toc187271172)

[How to Rename and Remove file (git mv and git commit) 3](#_Toc187271173)

[Shows you a list of files in your repository (git ls-files) 4](#_Toc187271174)

[Configuration Variables (git config) 4](#_Toc187271175)

[Review the files to be committed and which are not (git status) 5](#_Toc187271176)

[Sourcetree 8](#_Toc187271177)

[Add or Clone Repository 8](#_Toc187271178)

[Overview of File Status, History, and Search 8](#_Toc187271179)

[References 10](#_Toc187271180)

# Introduction

Git is a distributed version control system that allows multiple people to work on a project simultaneously without overwriting each other's changes. It was created by Linus Torvalds in 2005 and has since become an essential tool for developers.

Every developer has a complete copy of the repository, including its history, on their local machine. This ensures that the project is not dependent on a central server and reduces the risk of data loss.

Git supports powerful branching and merging capabilities, allowing developers to work on different features or fixes simultaneously. Branches are lightweight and easy to create, which encourages experimentation and parallel development.

Before committing changes to the repository, developers can add files to a staging area. This allows them to review and selectively commit changes, providing greater control over the project's history.

Git maintains a history of all changes made to the repository. Each commit is identified by a unique hash, and developers can view the history to track changes, revert to previous states, and understand the evolution of the project.

Git enables collaboration through remote repositories hosted on platforms like GitHub, GitLab, and Bitbucket. Developers can clone repositories, push and pull changes, and collaborate on code through pull requests and code reviews.

# Basic Git Commands

## Initialize Git Repository (git init)

The **git init** command creates a hidden directory named **period git** inside your current directory. This directory contains all the necessary files and subdirectories that Git uses to manage and track changes in your project. The **period git** directory includes subdirectories for objects, refs, and template files. It also creates essential files like HEAD, config, and description.

Simply type the following command to initialize Git repository and press Enter:

**git init**

## What is the Staging Area in Git (git add)

The **staging area is a file**, generally contained in your Git directory, that stores information about what will go into your next commit. It allows you to review and control changes before they are committed.

**Stage a File**: To stage a specific file, run:

**git add file\_name**

**Stage All Changes:** To stage all changes at once, you can use**:**

**git add \***

## How to Rename and Remove file (git mv and git commit)

First, you need to initialize and stage the file in the Git repository. This command essentially stages the changes required to rename or move the file, and then you can commit those changes to the repository. **For renaming, it's a simple two-step process**:

1. Rename or move a file or directory within a Git repository**.**

 **git mv**: This is the command to rename or move a file in the Git repository.

 **old\_file\_name**: This is the current name or path of the file or directory you want to rename or move.

 **new\_file\_name**: This is the new name or path for the file or directory.

**git mv old\_file\_name new\_file\_name**

1. **Commit the changes**.

 **git commit**: This command is used to commit your staged changes to the repository. A commit captures a snapshot of the project's currently staged changes.

 **-m**: This flag specifies that the commit message will be provided directly in the command line, rather than opening an editor.

 **“Renamed file":** This is the commit message. It describes the changes that are being committed. In this case, it indicates that a file was renamed.

**git commit -m "Renamed file"**

**For removing a file**:

1. Use the following commandto delete the file.

**git rm file\_name**

1. Commit the removal.

**git commit dash m "Removed file"**

## Shows you a list of files in your repository (git ls-files)

Git ls-files is like **Git's inventory checker**. It shows you a list of files in your repository that Git knows about. This includes tracked files, which means files that are part of your Git history, and optionally, files that would be excluded by .gitignore. It's super handy to get a quick overview of what's in your repo.

**git ls-files**

A screenshot of a computer

Description automatically generated

## Configuration Variables (git config)

* To **check the variables configured** in Git, you can use the following command:

**git config --list**

* To set the **username in the configuration file** for commits, you can use the following command.

**git config user.name "Your Name"**

* To set the **email** **in the configuration file** for commits, you can use the following command.

**git config user.email "your.email@example.com"**

## Review the files to be committed and which are not (git status)

The **git status** command is a fundamental tool for understanding the current state of your Git repository. When you run **git status** in your terminal, it provides a detailed report on the following:

* **Changes to be committed:** These are modifications that have been staged using git add and are ready to be committed to the repository.
* **Changes not staged for commit:** These are modifications that have been made to tracked files but have not yet been staged.
* **Untracked files:** These are files that are not currently being tracked by Git and haven't been added to the repository.

In summary, git status helps you keep track of your progress and understand what changes have been made, which ones are ready to be committed, and which ones still need to be staged or committed.

We begin with **nothing being staged**, meaning no files have been put into the staging environment. Staging is where you place code that is waiting to be pushed to a remote server like GitHub. You'll see that your main branch is up to date with no pending commits.

A screenshot of a computer

Description automatically generated

Next, I updated the Navigation Bar on the Index Page by **adding a link to the new About Page** . Then I copied the About files into JetBrains WebStorm.

A screen shot of a computer

Description automatically generated

After saving all changes, I check the status again. This time, the status indicates that the Index file has been modified, but **no changes have been staged** for commitment. This means I haven't yet placed the files into the staging area to prepare them for commitment to my remote repository.

In Git, an untracked file is a file that exists in your working directory but hasn’t yet been added to the repository’s tracking system. Essentially, Git is unaware of this file. Untracked files are typically new files that you've created but haven’t yet added to version control. The three About Files are untracked.

A screen shot of a computer

Description automatically generated

I **staged the index file** but not the about files.

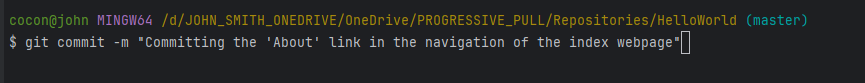


Observe that the **index file is highlighted in green**. This indicates that the file has been moved into the staging environment, but it has not yet been committed.

A screen shot of a computer

Description automatically generated

Finally, I commit whatever has been staged, which in this **case is the Index file**. The About files remain untracked.

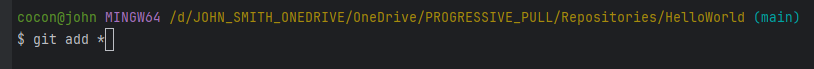


Checking the status once more, it confirms that changes to the **About files have still been untracked**.

A screen shot of a computer

Description automatically generated

I stage all the About files command, where the asterisk () means to commit all the files.

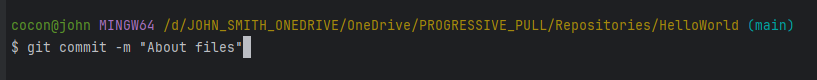


Now, I check the status again to **see all the files are staged**.

A computer screen with a red arrow pointing to the screen

Description automatically generated

After staging the files, **I commit them**.



You'll **notice that I am two commits behind** the main branch on the GitHub remote. This is because I made two separate commits: one for the Index file and another for the About files. That’s why I'm behind two commits.

A computer screen with a red arrow pointing to a black background

Description automatically generated

# Sourcetree

## Add or Clone Repository

Let's start with Sourcetree. Open Sourcetree, where **you have two options**: add a local repository or clone one from a cloud-based server like GitHub.

A screenshot of a computer

Description automatically generated

Once you clone the repository using a tool like GitHub Desktop, you can open the **codebase with the Add feature**.

A screenshot of a computer

Description automatically generated

**We'll begin with cloning one**. Copy the URL into Sourcetree and give it a name. Create a folder to save the code you download from the Git repository.

A screenshot of a computer

Description automatically generated

## Overview of File Status, History, and Search

In the **Workspace Menu** you can check the status, view the history, and search for commit messages.

A screenshot of a computer

Description automatically generated

By selecting the search area, you can **type the name of the author** in the search bar and then you can pull up all the commit done by the author.

A screenshot of a computer

Description automatically generated

By choosing 'History' in the Workspace, you can view all changes along with their **corresponding commit messages**.

A screenshot of a computer

Description automatically generated

**To commit in Sourcetree**, stage your file changes first, then add a commit message and make the commit. New changes will be highlighted in green, and modified parts in pink, showing what's been added or removed.

A screenshot of a computer

Description automatically generated

# References

* [Get started with Sourcetree - Sourcetree Support](https://confluence.atlassian.com/get-started-with-sourcetree/get-started-with-sourcetree-847359026.html)
* [Git Official Reference Documentation](https://git-scm.com/docs/git)
* [GitHub tutorial - Cameron McKenzie - YouTube](https://www.youtube.com/playlist?list=PL_RrEj88onS8-8OfcYMuLXnTfnSIGf37k)