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 3](#_Toc188292543)

[Basic Git Commands 3](#_Toc188292544)

[Initialize Git Repository (git init) 3](#_Toc188292545)

[What is the Staging Area in Git (git add) 4](#_Toc188292546)

[How to Rename and Remove file (git mv and git commit) 4](#_Toc188292547)

[Shows you a list of files in your repository (git ls-files) 5](#_Toc188292548)

[Introduction of Configuration Variables 6](#_Toc188292549)

[System-level Configuration (sudo git config --system <variable> <value>) 6](#_Toc188292550)

[Global-level Configuration (git config --global <variable> <value>) 6](#_Toc188292551)

[Local-level Configuration (git config <variable> <value>) 6](#_Toc188292552)

[Configuration Variables (git config) 7](#_Toc188292553)

[Review the files to be committed and which are not (git status) 7](#_Toc188292554)

[Cloning HelloWorld from GitHub (git clone <url>) 11](#_Toc188292555)

[Introduction of Branching and Merging 11](#_Toc188292556)

[Lists all branches (git branch) 12](#_Toc188292557)

[Creates a new branch (git branch <branch-name>) 12](#_Toc188292558)

[Switches to the specified branch (git checkout <branch-name>) 12](#_Toc188292559)

[Delete Branch (git branch <branch-name> -D) 13](#_Toc188292560)

[Merges the specified branch into the current branch (git merge <branch-name>) 13](#_Toc188292561)

[Example code for the Merge operation 13](#_Toc188292562)

[GitHub Branches 14](#_Toc188292563)

[Git Merge Example 16](#_Toc188292564)

[Git Commit Example 17](#_Toc188292565)

[Delete Branch Example 17](#_Toc188292566)

[Sourcetree 18](#_Toc188292567)

[Add or Clone Repository 18](#_Toc188292568)

[Add Repository 18](#_Toc188292569)

[Clone Repository 19](#_Toc188292570)

[Overview of File Status, History, and Search 19](#_Toc188292571)

[Search 19](#_Toc188292572)

[Branches 19](#_Toc188292573)

[History 20](#_Toc188292574)

[File Status 21](#_Toc188292575)

[References 21](#_Toc188292576)

# 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 **.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 **.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:

A screen shot of a computer

Description automatically generated

## 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:



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

## 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.

A black background with yellow and white text

Description automatically generated

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.

A computer screen shot of a computer code

Description automatically generated

**For removing a file**:

1. Use the following commandto delete the file.

A black screen with yellow text

Description automatically generated

1. Commit the removal.

A screen shot of a computer

Description automatically generated

## 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.

A screen shot of a computer

Description automatically generated

## Introduction of Configuration Variables

Configuration variables in Git are settings that control various aspects of how Git operates. These variables can be set at different levels: **system**, **global**, and **local**.

The **System level** applies configurations to all users on the system, ensuring consistency across all repositories and users on a machine.

The **Global level** applies configurations to a specific user across all their repositories. It's ideal for personal preferences like username, email, or default editor.

The **Local level** applies configurations to a specific repository. Use git config --local (or simply git config) for settings like hooks or branch settings that are specific to a repository.

To set configuration variables in Git for system, global, or local levels, you can use the git config command with different flags. Here's how you can do it:

### System-level Configuration (sudo git config --system <variable> <value>)

To set a system-level configuration variable, you need to have administrative privileges. Use the following command:

**sudo git config --system <variable> <value>**

For example:

**sudo git config --system user.name "Your Name"**

### Global-level Configuration (git config --global <variable> <value>)

To set a global configuration variable for the current user, use:

**git config --global <variable> <value>**

For example:

**git config --global user.email** [**your.email@example.com**](mailto:your.email@example.com)

### Local-level Configuration (git config <variable> <value>)

To set a configuration variable for a specific repository, navigate to the repository's directory and use:

**git config <variable> <value>**

For example:

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

### Configuration Variables (git config)

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

A screenshot of a computer program

Description automatically generated

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

A black background with yellow and white text

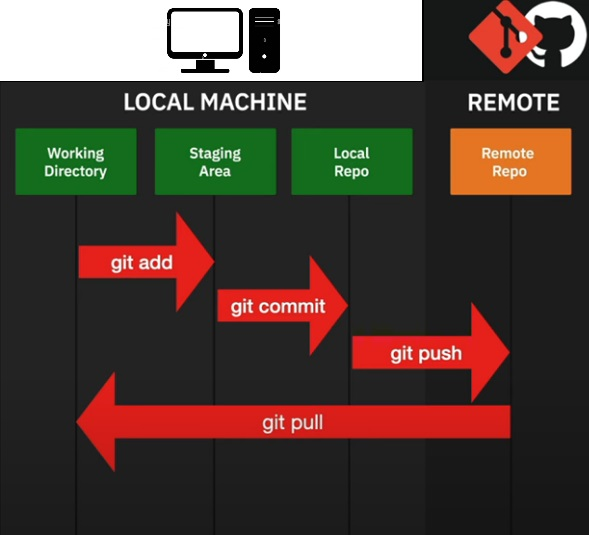
Description automatically generated

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

A black background with yellow and white text

Description automatically generated

## Git Workflow



## 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 black screen with white text

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 screenshot of a computer program

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 program

Description automatically generated

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

A black screen with yellow text

Description automatically generated

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 computer screen shot of a program

Description automatically generated

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

A screen shot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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

A computer screen with text and a red arrow pointing to the screen

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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 screen shot of a computer

Description automatically generated

## Cloning HelloWorld from GitHub (git clone <url>)

Now, let's clone this repository. The first step is to run git clone and get the clone URL from GitHub. Once we have the clone URL, we'll paste it into our terminal and clone the repository.

A screenshot of a computer

Description automatically generated

<https://github.com/progressivepull/HelloWorld.git>

After cloning, if you run git branch, you might notice there's no repository here because we need to change directories into the "HelloWorld" folder. That's a common mistake.

A screenshot of a computer screen

Description automatically generated

## Introduction of Branching and Merging

A **git branch** is a key feature in Git, a version control system used by developers to manage and track changes in their code. Think of a branch as a separate line of development that diverges from the main project. This allows you to work on new features, bug fixes, or experiments without affecting the main codebase.

### Lists all branches (git branch)

By typing git branch without any arguments, you can see a list of all the branches in your repository. The current branch will be highlighted with an asterisk (\*) next to its name.

A black screen with yellow text

Description automatically generated

### Creates a new branch (git branch <branch-name>)

You can create a new branch by specifying the branch name after the git branch command. The command git branch temp-animals will create a new branch in your Git repository called temp-animals.



The current branch is the main, not temp-animals.

A black screen with yellow text

Description automatically generated

### Switches to the specified branch (git checkout <branch-name>)

In Git, the checkout command is used to switch between branches or restore working tree files. When you use git checkout temp-animals, you are switching your working directory to the branch named temp-animals.

A screen shot of a computer

Description automatically generated

### Delete Branch (git branch <branch-name> -D)

The command git branch <branch-name> -D is used to delete a branch in Git forcefully. Here’s a breakdown of the command:

* git branch: This is the Git command used to manage branches.
* <branch-name>: This is a placeholder for the name of the branch you want to delete.
* -D: This flag is used to forcefully delete the branch, even if it has unmerged changes.

For example, if you have a branch named temp-colors that you want to delete, you will use the command **git branch temp-colors -D**. This will remove the feature-x branch from your repository, even if it has changes that haven't been merged into any other branch.

Keep in mind that using the -D flag can be risky, as it will permanently delete the branch and its changes, so be sure you no longer need it before running this command.

A black background with yellow and white text

Description automatically generated

### Merges the specified branch into the current branch (git merge <branch-name>)

I've created a playground on GitHub that you can freely clone. Avoid modifying the main, note-animals, or note-colors branches. Instead, create temporary branches in your local git playground to practice git merging.

### Example code for the Merge operation

I start out with this example by adding a new folder called "merge." In the "merge" folder, I have two folders. One folder is called animals, and the other is called colors. Both folders contain a file named note. You can use the Note file to practice merging.

The Animals folder contains a note file, and at the bottom, there is a list of animals.

A screenshot of a computer

Description automatically generated

The Colors folder contains a note file, and at the bottom, there is a list of colors.

A screenshot of a computer

Description automatically generated

We're going to create two branches and then merge them to demonstrate how to deal with conflicts.

### GitHub Branches

Go into my GitHub site and access the HelloWorld repository. You'll notice I have three different branches here. So, to summarize:

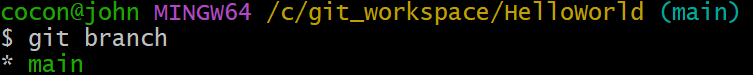
* The **main branch** does not have the notes file.
* The **note-colors branch** has the notes file with colors.
* The **note-animals branch** has the notes file with animal information at the bottom.

A screenshot of a computer

Description automatically generated

These are the three branches I have set up.

Once inside the "HelloWorld" directory, the running **git branch** will show that there's only one local branch.



To view remote branches in GitHub, use **git branch -r**.

A black screen with yellow text

Description automatically generated

To check out a specific branch, such as the note-animals branch, use **git checkout note-animals**. Repeat this process for other branches like "notes-colors" by using **git checkout notes-color**.

A screen shot of a computer

Description automatically generated

The git branch will show all branches on your local machine after checking out all branches.

A black screen with yellow text

Description automatically generated

To avoid modifying any of the three branches directly, I'll create a temporary branch for practice. In the terminal, switch to git bash (the preferred environment). Assuming you're on the "notes-color" branch, create a temporary branch with **git branch temp-colors**.

A computer screen with yellow and white text

Description automatically generated

Switch to the new branch using **git checkout temp-colors**. Now, to practice merging, try merging the "notes-animal" branch into "temp-colors" with git merge notes-animal.

A black screen with yellow text

Description automatically generated

Let's open JetBrains WebStorm. Different editors handle Git merge operations uniquely, which is why I'll be using WebStorm for this demonstration.

A screenshot of a computer

Description automatically generated

### Git Merge Example

If you examine it, you'll see that you're currently on the temp-colors branch. This is your active branch, and you want to merge it into their branch, which is the notes-animals branch.

A screen shot of a computer screen

Description automatically generated

You can review the merge conflict by opening the note.md file.

A screenshot of a computer

Description automatically generated

If there's a conflict during the merge, Git will prompt you to resolve it by accepting either "Accept Yours" or "Accept Theirs". You can abort the merge operation.

A screenshot of a computer

Description automatically generated

### Git Commit Example

Accept the appropriate changes staging with **git add \*** and then commit them with **git commit -m "Committing changes after the merge".**

A screen shot of a computer screen

Description automatically generated

### Delete Branch Example

Once you’re done practicing, you can delete the temporary branch with **git branch temp-colors -D**. Before deleting temp-colors must change to another branch. This allows you to continue practicing in a safe environment without affecting the original branches.

A screen shot of a computer

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

### Add Repository

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

### Clone Repository

**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

### Search

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

### Branches

SourceTree is a graphical user interface (GUI) for managing Git repositories. One of its key features is the ability to handle branches. Branches in SourceTree (and Git in general) are a way to diverge from the main line of development and continue to work on changes independently.

Here's a breakdown of what branches are and how you can use them in SourceTree:

**What Are Branches?**

* **Branches** are essentially pointers to specific commits in a repository. They allow you to create multiple lines of development within the same project.
* The **default branch** when you create a repository is called **main or master**.
* **Feature branches** can be created for working on specific features or tasks without affecting the main branch.

A screenshot of a computer

Description automatically generated

### History

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

### File Status

**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)