GIT(Global Information Tracker)

Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel.

Git is a version control system.

Git helps you keep track of code changes.

Git is used to collaborate on code.

**What is Git?**

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

**It is used for:**

* Tracking code changes
* Tracking who made changes
* Coding collaboration

**What does Git do?**

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

**Working with Git**

* Initialize Git on a folder, making it a **Repository**
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added or deleted, it is considered **modified**
* You select the modified files you want to **Stage**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

Why Git?

* Over 70% of developers use Git!
* Developers can work together from anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to earlier versions of a project.

What is GitHub?

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
* In this tutorial, we will focus on using Git with GitHub.

Why is Git so popular?

One of the biggest advantages of Git is its **branching capabilities**. Unlike centralized version control systems, Git branches are cheap and easy to merge. This facilitates the feature branch workflow popular with many Git users. Feature branches provide an isolated environment for every change to your codebase.

What is difference Git and GitHub?

While Git is a tool that's used to manage multiple versions of source code edits that are then transferred to files in a Git repository, GitHub serves as a location for uploading copies of a Git repository. I

A remote repository in Git, also called a remote, is **a Git repository that's hosted on the Internet or another network**.

What are the types of repository?

There are exactly two types of repositories: **local and remote**:

What is local and remote repository?

Local repositories reside on the computers of team members. In contrast, remote repositories are hosted on a server that is accessible for all team members - most likely on the internet or on a local network.

Is git a remote repository platform?

Distributed Version Control System: **Git has a remote repository** which is stored in a server and a local repository which is stored in the computer of each developer.

## Configure Git

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

**Note:** Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

**Creating Git Folder**

Now, let's create a new folder for our project:

Example

mkdir myproject

cd myproject

mkdir **make**s a **new directory**.

cd **changes** the **current working directory**.

Now that we are in the correct directory. We can start by initializing Git!

**Note:** If you already have a folder/directory you would like to use for Git:

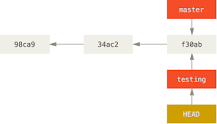
Navigate to it in command line, or open it in your file explorer, right-click and select "Git Bash here"

## Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

git init - Initialized empty Git repository in /Users/user/created folder/.git/

What is the default main branch of git?



**master**

The default branch name in Git is **master** . As you start making commits, you're given a master branch that points to the last commit you made.

## Git Adding New Files/Tracking files/add to staging

You just created your first local Git repo. But it is empty.

So let's add some files, or create a new file using your favourite text editor. Then save or move it to the folder you just created. And save it to our new folder

Let's go back to the terminal and list the files in our current working directory:

ls

then we check the Git status and see if it is a part of our repo:

git status

Now Git is **aware** of the file, but has not **added** it to our repository!

Files in your Git repository folder can be in one of 2 states:

* Tracked - files that Git knows about and are added to the repository
* Untracked - files that are in your working directory, but not added to the repository

 When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

## Git Staging Environment

## Staging is the area where your files sits until we commit them

One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

**Staged** files are files that are ready to be **committed** to the repository you are working on. You will learn more about commit shortly.

For now, we are done working with file. So we can add it to the Staging Environment:

### Example

git add FileName //Particular file

git add . // All files

git add -A // All files

**Note:** The shorthand command for git add --all is git add -A

## Git Commit

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when

### Example

git commit -m "First release of Hello World!

The commit command performs a commit, and the -m "*message*" adds a message.

The Staging Environment has been committed to our repo, with the message:  
"First release of Hello World!"

**GIT Restore:** restore the committed file to staging area

9o

**Git Commit without Stage**

Sometimes, when you make ismall changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file.

And check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way:

### Example

git status --short

**Note:** Short status flags are:

* ?? - Untracked files
* A - Files added to stage
* M - Modified files
* D - Deleted files
* We see the file we expected is modified. So let's commit it directly:

### Example

* git commit -a -m "Updated index.html with a

**Warning:** Skipping the Staging Environment is not generally recommended.

Skipping the stage step can sometimes make you include unwanted changes.

## Git Commit Log

To view the history of commits for a repository, you can use the log command:

### Example

git log

**Git Help**

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line:

* git *command* -help -  See all the available options for the specific command
* git help --all -  See all possible commands

**Note:** If you find yourself stuck in the list view, SHIFT + G to jump the end of the list, then q to exit the view.

**MY Notes:**

Git has three Environment

Working Files – Files you will be working on

Staging Area – An area where your works sits , tracking files but commit to track the history or restore

Commit – adding to the history for records

COMMANDS

mkdir  - **make**s a **new directory**.

Cd -  **changes** the **current working directory**.

Git --version - Gives the version of the software

git init - Initialized empty Git repository in /Users/user/created folder/.git/

git config --global user.name "Name"

git config --global user.email "Email"

mkdir DirectoryNAme – create a directory or folder

ls – list all the files in the directory

git status – provides the status of the git

git add FileName //Particular file

git add . // All files

-A, --all - add changes from all tracked and untracked files

**Note:** The shorthand command for git add --all is git add -A

git commit -m "First release of Hello World! - The commit command performs a commit, and the -m "message" adds a message.

git status –short - when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file.

Git rm -r chached – deletes files from your local git repo

Does git rm -- cached delete the file?

**--cached**. **Removes the file only from the Git repository, but not from the filesystem**. By default, the git rm command deletes files both from the Git repository as well as the filesystem. Using the --cached flag, the actual file on disk will not be deleted.

**Cat** - **print the content of a file onto the standard output stream**

Git restore --staged filename – restore the committed file

**Echo filename –** creates and inputs the text

**Note:** Short status flags are:

* ?? - Untracked files
* A - Files added to stage
* M - Modified files
* D - Deleted files

git log - To view the history of commits for a repository

**git commit -a -m** “By pass staging and commit” –

The -a Option

The -a stands for all. **This option automatically stages all modified files to be committed**. If new files are added the -a option will not stage those new files. Only files that the Git repository is aware of will be committed.

**Git rm filename.extension** – removes individual file or a collection of files. The primary function of Git remove is to remove the tracked files from the Git Index, Additionally, git rm can be used to files from both the staging index and working Directory

**Git restore -**

**Git reset HEAD~1** – to restore the staged file(only if the file was remove on the staged area)