WHAT IS GIT?

Git is a widely used version control system designed to track changes made to files over time, enabling users to manage and maintain histories of their projects efficiently. Instead of saving multiple copies of a file—as you might do with versions like “resume v1,” “resume v2,” and so on—Git allows you to maintain just one file while keeping a detailed record of all changes made to it. This system supports collaborative work by providing mechanisms to track, review, and merge changes from multiple contributors seamlessly.

At its core, Git organizes your workflow through several key concepts: **the working directory (where you modify files)**, **the staging area (a preparatory space to review changes before saving)**, **and the local repository (a complete history of your project stored on your machine).** Additionally, **Git facilitates collaboration via remote repositories shared over networks or the internet, allowing teams to push and pull changes as needed. Features like branches enable developers to work on separate features or fixes independently without disrupting the main project, while pull requests and merging provide structured ways to review and integrate changes.**

Git is not the same as GitHub, which is a cloud-based platform built on Git technology that offers additional collaboration tools and code hosting services. To use Git, you must install it on your computer, configure your identity with commands specifying your username and email, and then initialize repositories for your projects. Common commands include git status to check changes, git add to stage modifications, and git commit to save those changes with descriptive messages. Overall, Git empowers developers to efficiently manage code versions, collaborate with others, and maintain a clean, organized project history.

WHAT IS GITHUB?

GitHub is a web-based platform that acts as a remote repository hosting service, allowing developers to store, manage, and collaborate on their code projects in the

cloud. While Git itself is a version control system that tracks changes to files locally on your computer, GitHub extends this functionality by providing a centralized online space where developers can push their local repositories, share code, and work together with others. It facilitates collaboration by enabling multiple contributors to pull, push, and merge changes, making teamwork more efficient.

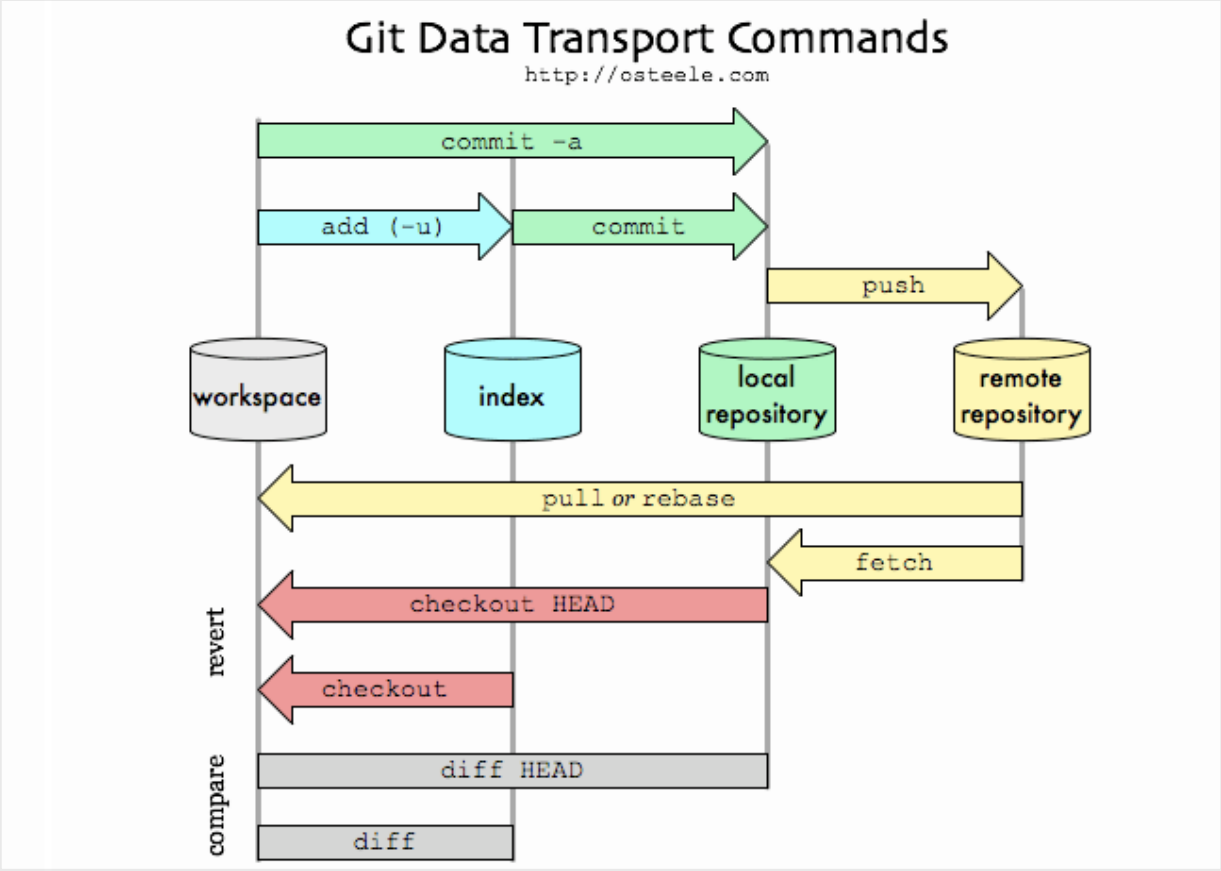
GitHub also supports essential collaborative features such as pull requests, which let developers propose changes from one branch to another, enabling code review and discussion before merging those changes into the main project. Additionally, GitHub offers tools for issue tracking, project management, and continuous integration, helping teams build, scale, secure, and maintain their software projects more effectively. Importantly, GitHub is not the same as Git—the former is a platform built on top of the latter—working together to streamline the software development process by combining version control with cloud-based collaboration and hosting.

LOCAL VS REMOTE REPOSITORY?

A **local repository** in Git is the complete history of your project stored on your own computer. It includes all the commits and branches you create as you work, acting as your personal record of changes. Your local repository resides on your machine and allows you to work independently, making changes, staging them, and committing to build a detailed timeline of your project's evolution without needing an internet connection. It reflects the state of your project's files in your

working directory and staging area.

On the other hand, a **remote repository** is a version of your project that is hosted on the internet or a network, such as platforms like GitHub or GitLab. It serves as a centralized shared space that multiple collaborators can access to push their local changes and pull updates made by others. This repository enables teamwork by synchronizing code changes between various contributors, facilitating collaboration, code review, and integration of different branches. While your local repository is private to your machine, the remote repository is public or shared depending on project settings, allowing distributed development and backup of your work.



HOW DO WE GET DATA FROM OUR LOCAL COMPUTER TO GITHUB?

1. **Initializing Git (git init)**
   This command sets up a new Git repository in your current project folder. It creates a hidden .git directory where Git stores all metadata and history for version control. This is the *first step* you perform in a new project to enable Git tracking. Without initializing, Git commands like git status will return errors because Git isn't managing the folder yet.
   - **Purpose:** Start version control tracking on a folder.
   - **Effect:** Creates a local repository on your machine.

- **Example:**git init
  - **Output:**
    - Initialized empty Git repository in /path/to/project/.git/

2. **Staging Changes (git add)**
   Staging is the process of preparing your file changes before committing them. When you edit or create files, they exist in the *working directory* (your current workspace). Running git add moves these changes to the *staging area* (or index), a draft zone where you finalize what will be included in the next commit. This allows selective commits and review before saving history.
   - **Purpose:** Add changes from working directory to staging area.
   - **Effect:** Prepares files to be committed; they won't be saved until committed.
   - **Example:**
     - git add .
   - **Note:** After git add, git status shows staged files ready to commit.
3. **Checking Git Remote (git remote)**
   Git remotes are references to versions of your repository hosted elsewhere, usually on the internet or a network (like GitHub). Checking remotes lets you see which remote repositories your local repo is connected to. This is crucial for collaboration, enabling you to push your commits to shared codebases or pull updates from others.
   - **Purpose:** View or manage remote repositories linked to your local repo.
   - **Effect:** Shows URLs and names of remote repositories (e.g., origin).
   - **Example:**git remote -v
     - **Output:**
       - origin https://github.com/username/repo.git (fetch)
       - origin https://github.com/username/repo.git (push)
   - This tell us the URL where the code(data) is going to be pushed
4. **Committing Changes (git commit)**
   Committing saves the staged changes into your local repository's history with a descriptive message. It creates a snapshot of your project at that point, allowing you to track progress and revert if needed. The commit includes author information configured earlier (user.name and user.email).
   - **Purpose:** Save staged changes permanently in local Git history.
   - **Effect:** Creates a new commit object with changes and message.
   - **Example:**
     - git commit -m "initial commit"
   - **Output:**
     - [main (root-commit) abc1234] initial commit

        - 1 file changed, 1 insertion(+)
        - create mode 100644 hello.md
5. **Pushing Changes (git push)**
   Pushing uploads your committed changes from your local repository to a remote repository, making your updates available to collaborators or as a backup in the cloud (e.g., GitHub). Without pushing, commits exist only on your local machine.
    - **Purpose:** Transfer local commits to a remote repository.
    - **Effect:** Updates remote branch with your local commits.
    - **Example:**
        - git push origin main
    - **Note:** Requires a remote repository to be configured (git remote add origin).

HOW DO WE GET DATA FROM OUR GITHUB TO OUR LOCAL COMPUTER?

- **What are Git Branches?**
    - Branches are parallel versions of your project within a Git repository.
    - They allow you to work on different features, fixes, or experiments independently without affecting the main project (usually the “main” or “master” branch).
    - Branches help maintain a clean and organized workflow by isolating changes until they are ready to be merged back.
    - Example: You might have a branch called feature-login to develop a login feature, while the main branch remains stable. Once the feature is complete, you merge feature-login back into the main branch.
    - This ensures that ongoing development does not disrupt the stable codebase.
- **How Branches Work in Practice:**
    - You can create a new branch with git branch branch-name.
    - Switch to a branch using git checkout branch-name.
    - Work on your changes and commit them without affecting other branches.
    - Once ready, merge the branch back into the main branch with a pull request or a merge command.
- **How to Git Pull on the Master Branch:**
    - The command git pull is used to fetch and integrate changes from a remote repository into your local branch.
    - To pull updates specifically on the master branch, first ensure you are on the master branch using:git checkout master
    - Then, run the pull command:

        - git pull origin master
    - This fetches the latest commits from the remote repository's master branch and merges them into your local master branch.
    - It keeps your local repository up-to-date with the remote changes, which is essential when collaborating with others.
- **Summary:**
    - **Branches** allow isolated development streams in Git, enabling safer and more organized coding workflows.
    - **Pulling on master branch** ensures your local main codebase reflects the latest remote changes by switching to master and running git pull origin master

GIT COMMANDS FOR COMMAND LINE(TERMINAL)

Topic	Description	Example Command(s)	Output / Notes
Version Control Concept	Tracks file changes, enabling you to view history and revert to previous versions without duplicating files.	N/A	Example: Instead of "resume v1", "resume v2", etc., use Git to track changes in a single file.
Working Directory	Your current workspace where you modify files; changes here are not yet tracked by Git.	N/A	Files start here before being staged or committed.
Staging Area (Index)	A preparation zone where changes are reviewed and staged before committing to the repository.	git add or git add .	Staged files are ready for commit, allowing selective commits.

Local Repository	The Git repository on your machine storing the full project history, including commits and branches.	N/A	Acts as your personal record of project changes.
Remote Repository	A version of your project hosted online (e.g., GitHub) to enable collaboration by pushing and pulling changes.	N/A	Enables team collaboration and backup of code.
Branches	Independent lines of development for new features or fixes, isolated from the main branch until merged.	git branch	Allows parallel development without affecting the main project.
Pull Request	A request to merge changes from one branch to another, often used for code review and collaboration.	N/A	Facilitates team discussions and quality control before merging.
Merging	The process of integrating changes from one branch into another, combining histories into one unified timeline.	git merge	Combines different branches, resolving conflicts if any.

Installing Git (macOS)	Use Homebrew to install the latest Git version instead of relying on pre-installed versions.	brew install git	After installation, running git shows available Git commands.
Installing Git (Windows)	Download installer from Git website, follow prompts, set default branch to “main”, and finish installation.	N/A	After installation, open terminal and run git to confirm commands are available.
Configuring Git	Set your identity to attribute commits properly with username and email.	git config --global user.name "Name" git config --global user.email "email@example.com"	Check config with git config --list; used to credit work done.
Create Project Folder	Make a new directory and navigate into it as your project workspace.	mkdir git-practice cd git-practice	Prepares folder for Git repository initialization.
Create Files	Create new files to track in Git.	touch hello.md touch newfile.js	Files appear in working directory, ready to be staged.
Initialize Git Repo	Start tracking files by initializing a Git repository.	git init	Output: Initialized empty Git repository in ...
Check Status	View changes and file states (untracked, staged, modified).	git status	Shows which files are untracked, staged, or modified.

Add Changes to Staging	Move changes from working directory to staging area so they can be committed.	git add .	Stages all new and modified files.
Commit Changes	Save staged changes to the repository with a descriptive message.	git commit -m "initial commit"	Output includes commit hash and summary of changes.

GENERIC COMMANDS FOR COMMAND LINE(TERMINAL)

Command	Description
mkdir	Creates a new directory named ``. This is used to organize your project files into folders.
cd	Changes the current directory to the specified ``. This lets you navigate **into** a child directory.
cd ..	Moves **up one level** to the parent directory of your current folder.
cd .	Refers to the **current directory**; running cd . keeps you in the same place, useful in scripts or chaining commands.
pwd	Prints the full path of your **current working directory**, helping you confirm where you are in the filesystem.
ls	Lists all files and directories in the current directory, showing your project contents.

touch	Creates a **new empty file** named `` in the current directory. By specifying the filename after touch, you define exactly what the new file will be called. For example, touch hello.md creates a blank file named hello.md. This is often the first step before editing or adding content to a new file.
rm	Deletes the specified file `` from your directory.
rm -r	Recursively deletes a directory and all its contents, used for removing folders and their files.
clear	Clears the terminal screen output for improved readability.