What Is GIT | Mechanism, Lifecycle, Architecture, Workflow, & More

Git is an open-source, distributed version control tool that helps track changes made in computer files/ codes. It facilitates collaboration and coordination amongst teams of developers while maintaining data integrity.

Shivani Goyal

Listen

Git is a distributed version control software/ system that allows developers to collaborate on software projects with ease. It was developed by Linus Torvalds in 2005 for the development of the Linux kernel and has since become the most widely used version control system in the world. In this article, we will explore what is Git in detail, how it works, the life cycle of git, and why it is such an essential tool for developers.

What Is Git? Basics

worrying about interfering with the work of others.

Git is designed to be fast and efficient, even with very large codebases. It uses a content-addressable file system to store all versions of files in the repository, which allows for quick access and easy retrieval of past/ earlier versions of code.

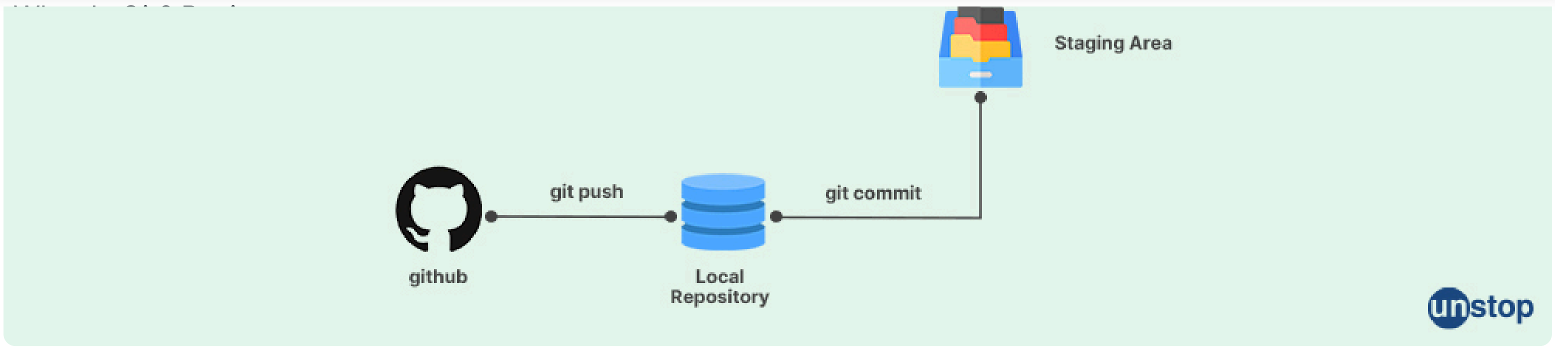
How Git Works?

The Git working mechanism is dependent on a few key components/ features. They are-

- **Commits**: Git works by maintaining a history of changes to a codebase in the form of commits. A commit is a snapshot of the entire codebase at a particular point in time. When a developer makes changes to the code, they create a new commit, which contains a record of all the changes they made. Each commit is identified by a unique SHA-1 hash, which allows Git to track changes and ensure the integrity of the commit and Git repository.
- **Git Branches**: Git uses branches to allow multiple developers to work on different parts of the codebase simultaneously. A branch is essentially a separate version of the codebase that can be worked on independently. When a developer creates a new branch, they are creating a copy of the codebase at the current point in time. They can then make changes to their branch without affecting the main branch, known as the "master" branch.
- **Merge**: Once a developer has made changes to their branch, they can merge their changes back into the master branch. Git uses a merge algorithm to determine how to combine changes from different branches, and it will attempt to automatically resolve any conflicts that arise.

Git also has a powerful set of tools for managing and reviewing changes to a codebase. It gives developers some additional benefits to compare different versions of code, track changes made by individual developers, and easily revert changes if necessary.

The Git Lifecycle



The lifecycle of Git consists of four main stages:

1. Create: The first stage in the lifecycle of Git is to create a new repository. You can think of it just like a project folder. This can be done either locally on your computer or on a remote server. To create a new repository, you can use the command "git init" followed by the name of the repository.

2. Modify: The next stage in the lifecycle of Git is to modify the files in the repository. This can be done using any text editor or integrated development environment (IDE) of your choice. When you make changes to a file, Git automatically detects those changes and marks the files as "modified."

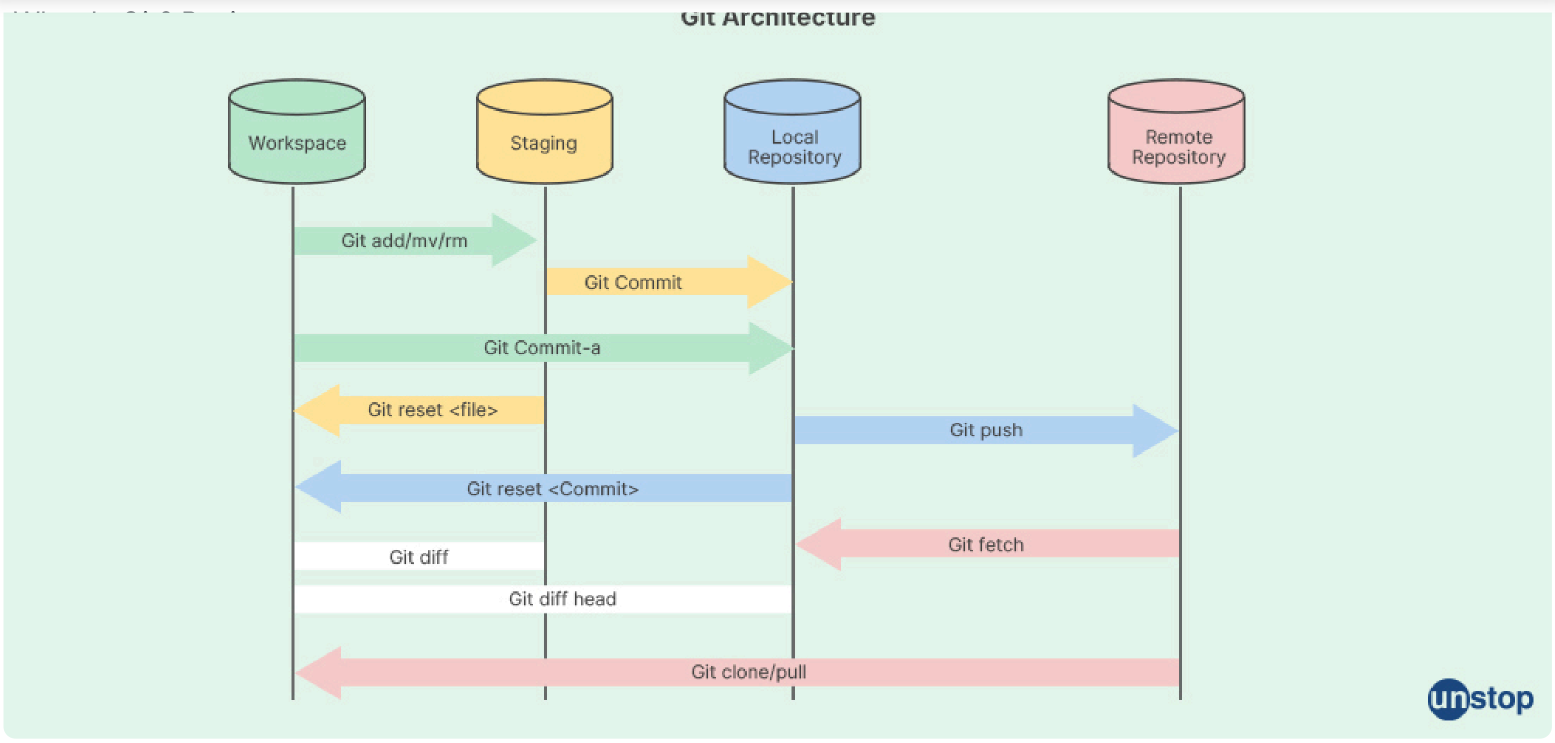
3. Stage: The third stage in the lifecycle of Git is to stage the changes you have made. Staging is the process of preparing your changes to be committed to the repository. You can stage your changes using the "git add" command followed by the name of the file you want to stage. You can also use the "git add" command to stage all changes in the repository.

4. Commit: The final stage in the lifecycle of Git is to commit your changes to the repository. A commit is a permanent snapshot of the changes you have made to the repository, and by default, it goes to the master branch. When you commit your changes, Git creates a new commit object that contains the changes you have made, along with a message describing what changes were made. To commit your changes, you can use the "git commit" command followed by a message describing the changes you have made.

Once you have committed your changes, you can push those changes to a remote repository using the "git push" command. This will upload your changes to the remote repository, making them available for other developers to pull down and work with.

About The Git Architecture

The architecture of Git is designed to provide a flexible, scalable, and efficient way to manage source code changes. Git has a client-server architecture that allows multiple developers to work on the same codebase at the same time without the need for a centralized server. Each developer has a



Here are the key components of the Git architecture:

1. Git Repository: For all those who don't know what is a Git repository, well, it is the central data store for all of the files, directories, and project history in a project folder. In other words, this is the place where all of the changes that have been made to the codebase are stored. The repository can be located on a local machine, on a network server, or on a cloud-based platform.

2. Commit: A commit is a snapshot of the codebase at a particular point in time. It represents a complete set of changes made to the codebase, including new files, modifications to existing configuration files, edit files, and deleted file contents. Each commit has a unique identifier that can be used to reference it at any point in the future.

3. Branch: A branch is a separate line of development that is created from a specific commit in the codebase. Each Git branch can have its own set of changes and commits, which can be merged back into the main branch of the codebase when they are ready.

4. Merge: A merge is a process of combining changes from one branch into another branch. When two branches have diverged, a merge can be used to combine the changes from both branches into a single codebase.

5. Remote: A remote is a copy of the repository that is stored on a different machine or server. Remotes can be used to collaborate with other developers and to synchronize changes between different copies of the repository.

7. Pull: A pull is the process of downloading changes from a remote repository and merging them into the local copy of the repository.

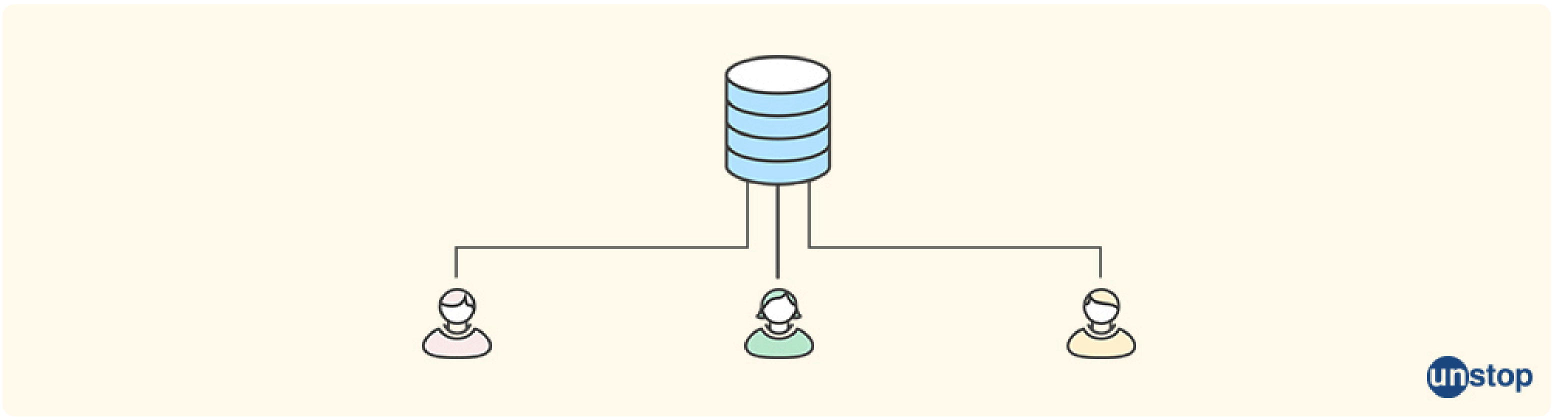
8. Push: A push is a process of uploading changes from a local copy of the repository to a remote repository.

The Different Types Of Git Workflows

We already know that Git allows multiple developers to work collaboratively on any project. Now with multiple developers working on different sections of a code without affecting the master code, there has to be a direction in which the work flows, to avoid confusion as to who is changing or merging what and so on. Well, this is where Got workflows come into play.

Git workflows help teams establish a structured approach to managing code changes, allowing them to work together more effectively and avoid common pitfalls such as merge conflicts and code duplication. Some of the most common Git workflows used by development teams are as follows:

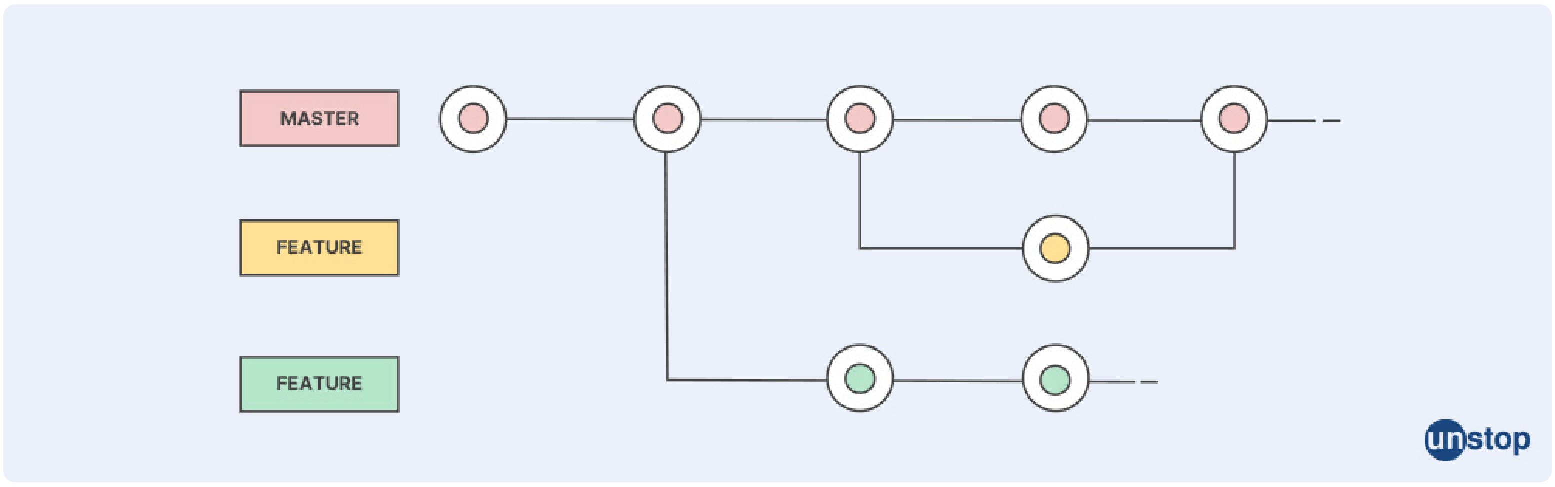
Centralized Workflow



The centralized workflow is a simple and straightforward workflow that is ideal for small teams working on simple projects. In this workflow, there is a single central repository that serves as the definitive source of truth for the codebase. Each developer clones the repository to their local machine and works on their own copy of the code. When they are ready to share their changes, they push them to the central repository.

The downside of this workflow is that it can lead to merge conflicts when multiple developers are working on the same code at the same time. To mitigate this risk, teams using this workflow often use a strict code review process to ensure that changes are properly reviewed before they are merged into the central repository.

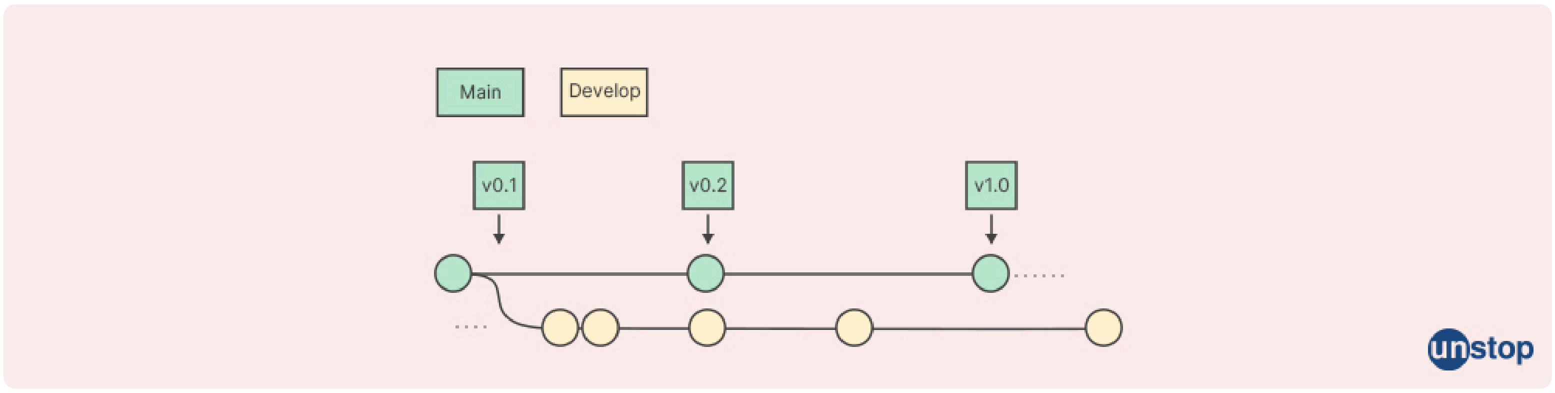
codebase that is dedicated to a specific feature or bug fix. Once the developer has completed their work, they submit a pull request to the main branch of the repository.



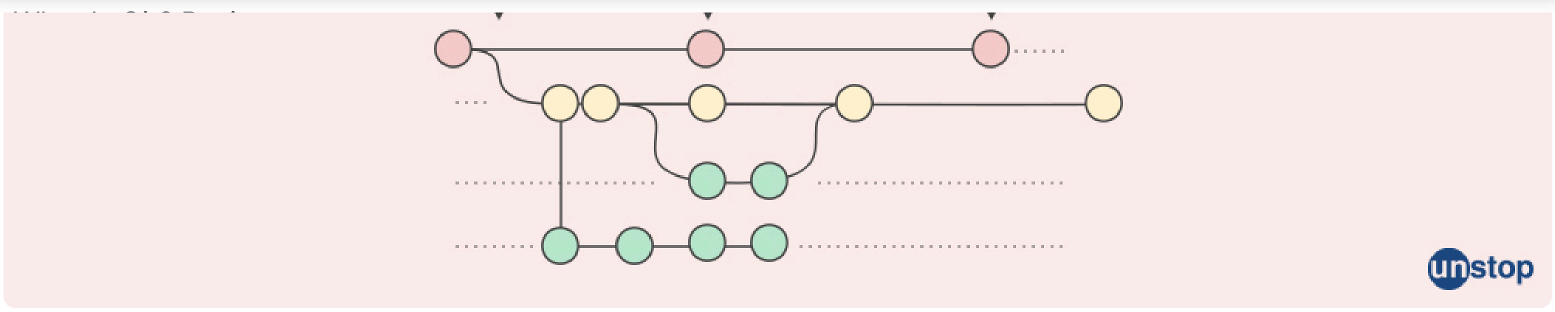
This workflow allows teams to work on multiple features or bug fixes simultaneously without interfering with each other's work. It also allows for better code review, as each pull request can be reviewed individually before it is merged into the main branch.

The Gitflow Workflow

The Gitflow workflow is a more complex workflow that is ideal for large teams working on long-lived projects. It is based on the idea of having separate branches for different stages of development, such as feature development, bug fixes, and release preparation.



In this workflow, the main branch of the repository is always stable, and new features and bug fixes are developed on separate branches. Once a feature or bug fix is complete, it is merged into a development branch, where it is tested and reviewed. Once the development branch is stable, it is merged into a release branch where final testing and bug fixing takes place. Once the release branch is stable, it is merged into the main branch and released to users.



This workflow is more complex than the other workflows, but it can be highly effective for large and complex projects where multiple teams are working on different parts of the codebase.

In conclusion, Git workflows are an essential version control tool for development teams working on collaborative software projects. By establishing a clear and structured approach to managing code changes, teams can work together more effectively and avoid common pitfalls such as merge conflicts, file corruption, and code duplication. Whether your team is working on a small or large project, there is a Git workflow that is right for you.

Common Functions Of Git

Git provides a range of functions that make it an essential and powerful tool for any software development team. Here are some of the key functions of Git:

1. Version control: One of the primary functions of Git is version control. It allows developers to track changes to their codebase/ code over time and to maintain a complete version history of all changes in the current version. This function enables developers to roll back changes, review earlier code versions, and collaborate more effectively.

2. Branching and merging: Git allows developers to create branches, which are separate lines of development that can be used to work on different features or fixes. Branches can be merged back into the main codebase when the work is complete. This function allows teams to work in parallel on different features without interfering with each other's work.

3. Collaboration: Git makes it easy for developers to collaborate on a codebase. Multiple developers can work on the same repository, and changes can be tracked, reviewed, and merged back into the main codebase. This function allows teams to work together efficiently and avoid conflicts and errors and check for non-linear development in the code base.

4. Code reviews: Git provides tools for code reviews, which enable developers to review and comment on each other's code changes. This function helps to improve the quality of code and ensures that changes are consistent with the overall architecture and coding standards.

6. Tagging: Git allows developers to tag specific versions of their codebase and project files. This function is useful for marking significant milestones or releases and for keeping track of different versions of a project.

7. Stashing: Git provides a stash function, which allows developers to save changes to their codebase without committing them. This function is useful for saving work in progress or for temporarily setting aside changes while working on something else.

8. Integration with other tools: Git integrates with a range of other tools, including continuous integration and deployment tools, issue tracking systems, and code review tools. This function allows developers to streamline their workflow and to work more efficiently.

9. Speed: Git provides a very fast and huge speed of connection to complete all the tasks efficiently, along with being in a connection to a centralized version control system.

We are sure that by now, you must know all you need about what Git is and how you, too, can benefit from using it. Here are some more articles that you might be interested in reading, do check them out:

1. What Is GitHub? An Introduction, How-To Use It, Components & More!
2. Linux Kernel In A Nutshell To Help You Prepare For Linux Interview Questions
3. Git Submodule: Add, Remove, Pull Changes & More (With Examples)
4. What Is Bash? Features, Major Concepts, Commands, & More!

Shivani Goyal
Manager, Content

I am an economics graduate using my qualifications and life skills to observe & absorb what life has to offer. A strong believer in 'Don't die before you are dead' philosophy, at Unstop I am producing content that resonates and enables you to be #Unstoppable. When I don't have to be presentable for the job, I'd be elbow deep in paint/ pencil residue, immersed in a good read or socializing in the flesh.

Comments

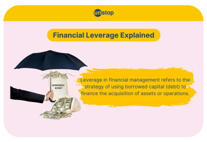
Blogs you need to hog!

Convert Word Doc To PDF & Sign Them
(Easy Step-wise Explanation)
DOC

How To Convert A Word Document To PDF And Sign It Online With Ease

What Is Dividend In Finance?
Dividend is a portion of the company's earnings returned to investors as a reward for their investment in the business or rewards to shareholders from a company's net profit.
DIVIDENDS

Dividend: Definition, Types, Examples And Impact On Investments



Financial Leverage: Definition, Types, Formula, Risks And Benefits

Financial Management Explained
Financial management is the strategic planning, organizing, directing, and controlling of an organization's financial activities.

Financial Management: Definition, Types, Importance And Functions

Share