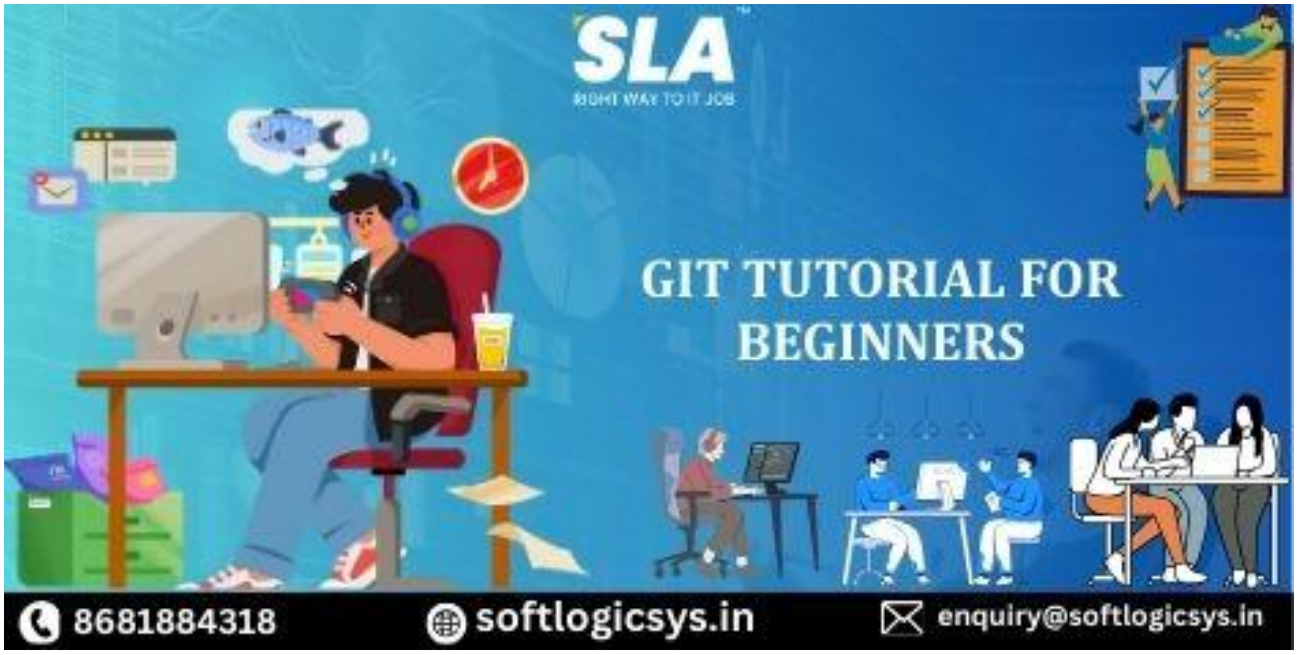

Git Tutorial for beginners

Introduction

Job hunting frequently requires mastery of Git, the fundamental version control tool for collaborative coding. Don't let maze-like workflows or configuration roadblocks hold you back! This Git tutorial for beginners breaks the technical jargon barrier, addressing typical starter headaches so you can confidently demonstrate your skills. Ready to become a master version controller? View our full **Git course syllabus** and begin today!

Why Students or Freshers Learn Git?

Freshers and students must learn Git since it is a required industry skill for practically every job in technology.

- **Version Control:** With Git, you can track all changes in your code. This implies that you can revert errors with ease, safely experiment, and view the history of a project.

- **Team Collaboration:** It is the de facto tool for collaboration. You'll get to merge contributions, fix conflicts, and work effectively on collaborative projects with remote colleagues from across the globe.
- **Portfolio Showcase:** HRs want to see your projects on GitHub (a Git-built platform). It demonstrates you can code, work with others, and oversee a project life cycle.
- **Career Readiness:** Git mastery is often a minimum expected standard for junior developer and DevOps roles, increasing your job prospects.

Prepared for your tech interview? Download our free list of most important **Git interview questions and answers** today!

Step-by-Step Git Tutorial for Beginners

Git is the version control standard used by the industry, enabling developers to track history, collaborate effortlessly, and control project history. This in-depth Git tutorial for beginners will take you through installing Git, learning the basic commands, and grasping the essentials.

Step 1. Installation and First-Time Setup

Git is a command-line application. You must install it on your machine and configure your identity next.1

1.1 Install Git

Operating System	Installation Method

Windows	Download the installer from the official Git website and follow the prompts. The installer includes Git Bash, a Linux-like terminal that works great with Git.
macOS	The easiest way is to open your Terminal and type git. If it's not installed, macOS will prompt you to install it via the Command Line Developer Tools. Alternatively, use a package manager like Homebrew: brew install git.
Linux (Debian/Ubuntu)	Use your distribution's package manager: sudo apt update and then sudo apt install git.

1.2 Set Up User Identity

Once installed, you need to inform Git about yourself. This is added to each change (commit) that you make.

Open your terminal (Git Bash on Windows) and execute the following commands, filling in the place holders with your own name and email: (BASH)

git config –global user.name "Your Name"

git config –global user.email "your.email@example.com"

The –global flag makes this setting apply to all your Git projects.

1.3 Test Installation

Check that Git was installed properly by requesting its version: Bash

git –version

Step 2. Core Concepts: The Git Workflow

Before we get into commands, learn about the three primary states or spaces in a Git project.

- **Working Directory (or Working Tree):** The files that exist and you actually see and edit on your computer. Here is where you make modifications.
- **Staging Area (or Index):** A staging area where you choose which particular changes (files or file portions) you would like to have included in your next commit. It is like getting your photo album ready before you take the photograph.
- **Git Repository (or .git folder):** Where Git keeps the metadata and object database for your project permanently, all history and commits.

The basic Git workflow involves: Modifying files in the Working Directory, Staging the intended changes, and Committing the staged changes to the Repository.

Step 3. Starting a New Project

There are two ways to start using Git for a project: creating a new repository or cloning an existing one.

3.1 Creating a New Repository

Go to your project folder using the cd (change directory) command and initialize Git inside it.

 # 1. Navigate to your project folder (e.g., a “website” folder)

cd ~/Documents/website

 # 2. Initialize a new Git repository

git init

This command initializes a hidden .git directory within your project directory, transforming it into a proper local Git repository.

3.2 Existing Repository Cloning

When you prefer to begin working on an already existing project hosted on a service such as GitHub, you employ the ***clone*** command.

 # This downloads the entire project history from the remote server

git clone https://docs.github.com/en/get-started/git-basics/managing-remote-repositories

Example:

git clone https://github.com/octocat/Spoon-Knife.git

This does the initialization of the local repository automatically and hooks it up to the source remote one.

Step 4. Committing and Tracking Changes

Committing is taking the action of permanently storing a snapshot of your changes into the Git history.

4.1 Status Check

The status command is your friend. It displays to you the status of your Working Directory and Staging Area in relation to the most recent commit.

git status

Output Interpretation:

- “Untracked files”: New files that Git is not yet tracking.

- “Changes not staged for commit”: Files you’ve changed but have not added to the Staging Area.
- “Changes to be committed”: Files in the Staging Area, ready for the next commit.

4.2 Staging Changes

To move changes from the Working Directory to the Staging Area, use git add.

a) Stage a specific file:

git add index.html

b) Stage all changes in the current directory: (Use with caution!)

git add .

4.3 Viewing Staged Changes (Diff)

Before committing, it’s good practice to review exactly what you’ve staged.

 # Shows the differences between the Staging Area and the last commit

git diff –staged

4.4 Committing Changes

After your changes are in the Staging Area, you can commit them officially with git commit. A commit needs to contain a thoughtful comment telling what changed and why.

 # The -m flag lets you provide a message directly

git commit -m “Added navigation bar and updated page title”

A commit makes a permanent snapshot with an ID (a SHA-1 hash), your name and email, and the commit message.

4.5 Looking at Commit History

To have a list of all the commits in the order they were made and in the branch you're currently in, use git log.

git log

Tip: For a tidier, single-line history, use: git log –oneline –graph

Step 5. Branching

Branching is the most powerful aspect of Git. It enables you to work on new features, squash bugs, or try out something without messing with the production version of your code. A branch is like an independent line of development.

5.1 Getting to Know the Main Branch

By default, when you create a repository, Git initializes a branch called master or main. This is the branch where traditionally the stable, production-ready code resides.

5.2 Creating a New Branch

Create a new branch for a feature (e.g., a "contact-form" feature).

git branch contact-form

5.3 Switching Branches

Before you can begin working on the new branch, you must switch (checkout) to it.

git checkout contact-form

Your Working Directory now represents the contact-form branch's code state.

5.4 One-Step Creating and Switching

The shortcut most often used is combining both of them:

git checkout -b new-feature-name

5.5 Showing Branches

Look at all of the local branches and which one you're on (the one with an asterisk *).

git branch

5.6 Merging Branches

After your feature is developed and tested in the contact-form branch, you must introduce those changes into the main codebase.

Switch back to the branch that you want to merge into (most likely main).

git checkout main

Execute the merge command:

git merge contact-form

If Git is able to merge the histories automatically, it does a "Fast-Forward" merge. If both branches changed the same lines (Conflict changes), Git will interrupt the merge and prompt you to fix the Merge Conflict.

5.7 Deleting a Branch

You can remove the feature branch safely once a merge has succeeded.

git branch -d contact-form

(Use -D instead of -d to force-remove an unmerged branch.)

Step 6. Remote Repositories (GitHub/GitLab)

The Git process normally consists of a remote repository (such as GitHub, GitLab, or Bitbucket) that is the team's focal point and your project's backup.

6.1 Remote to Local

You must first initialize an empty repository on your preferred platform (such as GitHub). The platform will offer you the remote URL. You then connect your local repository to it.

 # Sets the remote repository URL and names it 'origin'

git remote add origin [Your Remote Repository URL]

 # Verify the connection (shows the fetch and push URLs)

git remote -v

6.2 Pushing Changes

Once you've made changes locally, you must push them from your local repository to the remote server so others (or the remote backup) can view them.

 # Push the commits from your local 'main' branch to the 'origin' remote

git push -u origin main

The -u flag (for "set upstream") is only used once to associate your local branch with the remote branch; later pushes can simply use git push.

6.3 Pulling Changes

To download and merge changes others (or you made on a different computer) did into your local branch of work, you employ git pull.

git pull origin main

This command is actually two operations combined: git fetch (get the data) and git merge (merge the data).

6.4 Fetching Changes

If you would like to preview what changes are on the remote but not actually commit them to your local branch, use git fetch.

git fetch origin

This refreshes your remote-tracking branches (e.g., origin/main) so that you can review the changes before pulling.

Step 7. Undoing Things (A Safety Net)

Git is engineered so that it is very hard to lose data forever. Here are typical ways to undo or alter history.

7.1 Amending the Last Commit

If you had forgotten to stage a file or need to fix the commit message of the last commit, use git commit –amend.

 # Stage the forgotten file(s)

git add forgotten-file.js

 # Amend the last commit (this opens an editor to change the message)

git commit –amend

Note: Never amend a commit that is already on a shared remote repository.

7.2 Discarding a Staged File

If you have staged a file by mistake, which you did not intend to commit, use git reset.

 # Moves the file back from Staging Area to Working Directory

git reset HEAD path/to/file.js

The changes are still in your file; they're just no longer staged.

7.3 Discarding Local Changes

If you wish to discard permanently all changes that haven't been saved in a file and bring it back to the last commit state:

 # WARNING: This destroys local, unsaved changes!

git checkout — path/to/file.js

7.4 Resetting the History

A more aggressive action to reset the branch pointer to an earlier commit.

 # Reset to a specific commit (e.g., using the first 7 characters of its ID)

git reset –soft [commit-hash]

 # The changes are preserved in the Staging Area

git reset –mixed [commit-hash] # Default

 # The changes are preserved in the Working Directory (unstaged)

git reset –hard [commit-hash]

 # WARNING: Discards all local changes and commits since that point!

Git has a steep initial learning curve, but constant practice is the only way to master it. Start using Git for every personal project, even simple ones.

Focus on the core workflow: git status → git add → git commit → git push.

Ready to test your knowledge with real-world scenarios? Master common branching mistakes, merge conflict resolution, and history manipulation with our comprehensive **Git challenges and solutions** now!

Real Time Examples for Git Tutorial for Learners

Git is easy to understand when you look at it in practice. Below are real time examples of the fundamental Git workflow:

Saving Work (Commit):

Let's say you've completed the styling of your website's header (style.css). Rather than merely saving the file, you commit with the description, *"Feature: Completed responsive header styling."* This leaves a clear historical snapshot.

If later design modifications cause the layout to be broken, you can immediately switch back to this ideal header state.

Repairing a Bug (Branching):

You have your core application code stable, but a customer reports a serious bug (e.g., the login button is not working). You create and branch to an immediately new isolated branch named fix/login-bug.

You repair the bug on this isolated branch without altering the stable main code so that the rest of the application stays unaltered and deployable.

Team Collaboration (Pull & Push):

Your friend, Sarah, created a new footer.js file and pushed it to GitHub. When you want to begin working on your side first, you employ git pull to fetch Sarah's updates and

merge them into your local repository, ensuring that you work with the latest version of code. After completing work, you git push your updates again up for Sarah to notice.

Undoing an Error (Checkout):

You've been playing around with a file (config.yaml) and had the realization that you've created a dreadful mess that it's easier to throw away than to correct.

You use git checkout — config.yaml to restore the file to its previous committed state, erasing all the experimental modifications in a snap.

Eager to create your developer portfolio? Check out our collection of easy-to-use **Git project ideas** and start programming today!

FAQs About Git Tutorial for Beginners

1. What is Git used for?

Git is employed for version control so that developers can monitor all changes in source code, work on projects without conflict, and go back to previous working versions.

2. What does Git mean?

The author, Linus Torvalds, initially defined Git as a "stupid content tracker." It is not an acronym but rather a British slang word that refers to an obnoxious individual, showing Torvalds' humility.

3. What is the difference between Git and Github?

Git is the local computer program/installation used on your machine for version control. GitHub is an online platform (a business) that stores Git repositories online for collaboration and offsite backup.

4. Is Git like Python?

No. Python is a high-level programming language used to develop applications. Git is a version control system (a tool) to maintain the code in languages such as Python.

5. Is Git difficult to learn?

The fundamental commands (add, commit, push, pull) are easy to pick up. Advanced topics such as intricate merging, rebasing, and fixing deep conflicts require more practice but are a must to be proficient.

6. Which language is Git?

Git is written mainly in C. This was selected for performance. The commands you run (such as git add or git commit) are run through your computer's terminal.

7. What is Git example?

A good example is taking a website to develop using Git. You make a branch for a feature, commit changes in your local Git repository, and then push them onto GitHub for your team to see.

8. Is coding 2 hours a day enough?

Yes, persistent, intense coding for 2 hours a day is a great and feasible method to develop skills, finish projects, and create a solid learning habit.

9. Is Git a DevOps skill?

Yes, Git is an essential DevOps skill. It is important for source code management, continuous integration (CI), and pipeline automation processes in a DevOps setting.

10. What are the two basic jobs of Git?

The two fundamental tasks of Git are tracking and recording changes (version control) and enabling collaboration among multiple developers over one codebase.

11. Is Git a CI/CD tool?

No, Git is not a CI/CD tool. Git is the source code manager. CI/CD tools (such as Jenkins or GitHub Actions) build on top of Git and automatically build and deploy code when changes are pushed.

12. Are Jira and Git the same?

No. Git is a system of version control for source code. Jira is a project management application utilized for following tasks, bugs, and project workflow. Both are frequently used in conjunction.

Conclusion

You have now mastered the essential Git workflow: initializing repositories, staging changes, committing milestones, and collaborating via branching and remote pushing through this Git Tutorial for beginners. This foundation is crucial for any modern technical role. While Git has advanced features, consistently practicing add, commit, and push/pull will build your confidence quickly.

Ready to dig deeper into merge conflicts, rebasing, and pro Git tactics? Sign up for our complete, full **Git course in Chennai** today and make a basic skill your greatest strength!