

What is Version Control?

Version control helps developers keep track of code changes, work together smoothly, and fix problems easily.

Version control is a tool that helps you save your code step by step.

Why Do Developers Use It?

1. Work Together – Many people can work on the same project without confusion.
2. Undo Mistakes – You can go back if something goes wrong.
3. See Changes – You can see what was changed and who changed it.
4. Try Ideas – You can test new code without changing the main project.
5. Stay Organized – It helps keep everything neat and safe.

Popular Tool

- **Git** – The most common version control tool
- Used with websites like GitHub to share and manage code online
- You use Git to manage your code, then use **GitHub** to upload it, share it, or work with others.

Git

Git is a tool you install on your computer.

It helps you track changes to your code.

Works offline (on your local machine).

Used to create commits, branches, and versions.

Created by developers for tracking code.

GitHub

GitHub is a website where you can store and share your Git projects online.

It helps you store code online, collaborate with others, and show your projects.

Works online, like Google Drive for code.

Used to host repositories, review code, and manage projects with teams.

Owned by Microsoft and used by millions to share code

git --version command: -

is used to check if Git is installed on your computer and to see which version you have.

git config: -

The git config command is used to set up and manage Git settings.

You use it mainly to tell Git who you are (your name and email) and to configure other preferences like your default editor or how Git behaves.

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

This is important because Git uses your name and email in every commit.

git config --list : - show all settings

What does --global mean?

--global: Applies the setting to all your Git projects.

Without --global: Applies the setting to only the current project.

The pwd command stands for:

Print Working Directory

git status — What It Does?

The git status command shows you the current state of your Git project.

It tells you what's going on in your folder, like:

- Which files are changed
- Which files are staged (ready to commit)
- Which files are not tracked by Git
- Whether you're on the main branch, or a different one

.git — What Is It?

The .git folder is a hidden directory that Git creates when you run:

'git init'

Git creates a **hidden** folder called .git in your project

What Does It Do?

The .git folder stores all the information about your Git project, including:

- The full history of your changes
- All your commits
- Your branches
- Configuration files
- And more

It's basically the brain of your Git project.

git add :-

git add is a Git command used to tell Git **which files you want to include** in your next commit.

When you change a file, Git doesn't save it automatically.

You must add it first using:

'git add filename' Then you can commit it using git commit.

Or, to add all files (new + changed):

'git add .'

git commit: -

git commit -m "Your message" means:

Save your changes with a short message that describes what you did.

git clone

git clone copies a remote Git project (repository) to your computer so you can work on it.

git clone <repository> foldername/ - where you clone

What does git diff do?

- It shows you what changes you made in your files that are not saved (committed) yet.
- You change a file.
- Run git diff to see what you changed.

Life Cycle of a Change in Git

1. Modify Files

You change your files on your computer. These changes are only on your disk — Git doesn't track them yet.

2. Check Changes

Run:

```
git diff
```

to see what changes you made (not saved to Git yet).

3. Stage Changes

```
git add filename
```

or

```
git add .
```

to tell Git: "I want to include these changes in the next save."

4. Check Staged Changes

```
git diff --staged
```

to see what changes are ready to be saved (committed).

5.Commit Changes

`git commit -m "Your message here"`

This saves your changes in Git history with a message describing what you did

6.Push Changes (optional)

If you use a remote repository (like GitHub), run:

`git push`

to send your saved changes to the remote server.

What is a Commit ID?

Every time you save your changes with `git commit`, Git creates a unique ID for that commit.

This unique ID is called the commit ID or commit hash.

It looks like a long string of letters and numbers.

This ID helps Git and you to identify exactly which commit you are talking about.

Why is Commit ID useful?

- To see details of a specific commit.
- To go back to an old commit (undo changes).
- To share or refer to a specific commit with others.

How to see commit IDs?

Run this command: **`git log`**

You will see a list of commits with:

- Commit ID (long hash)
- Author
- Date
- Commit message

Example output:

Commit: e4f3a1b7c9d2e8f0a1234567890abcdef12345678

Author: Akash

Date: Tue May 28 10:00:00 2025 +0530

Fixed bug in login feature

Short commit ID

You usually only need the first 7-8 characters to refer to a commit, for example:

e4f3a1b

: q!

If you are inside the Git commit editor (usually Vim) and want to exit without saving anything

: q! means "Quit and ignore any changes".

1. Press the **Esc key** on your keyboard (to make sure you are not typing).
2. **Type: q!** (colon, then q, then exclamation mark).
3. Press **Enter**.

If you want to save your commit message and exit:

Press the Esc key on your keyboard (just to be safe).

Type: wq

(That means colon , then w, then q)

Press **Enter**.

This will save your message and exit the editor.

What does git log -3 do?

- It shows you the **last 3 saved changes** (commits) in your project.
- Run command: **git log -3**

git log -p

shows commit **history** plus the **exact code**/text changes (diff) made in each commit.

Commit info (ID, author, date, message)

Then, the diff showing:

- **Lines added** (with +)
- **Lines removed** (with -)

This helps you see what changed inside files for every commit.

What does git log --stat do?

- It shows the commits with summary of changes:
 - o Which files changed
 - o How many lines added or removed in each file

Command: -

git log --stat

Example output:

commit a1b2c3d4

Author: Akash

Date: Thu May 29 2025

Fixed bug in login

src/login.js | 10 +++++-----

src/utils.js | 4 ++--

2 files changed, 7 insertions(+), 7 deletions(-)

Quickly see what files changed in each commit and how big the changes were without seeing full code diff.

git log --oneline

- Shows a short and simple list of your commits.
- Each commit is shown in one line with:
- A short commit ID (7 characters)
- The commit message (short description)

Run this command inside your Git project folder:

git log -oneline

Example output:

a1b2c3d Fixed login bug

f4e5d6a Added new feature

123abcd Initial commit

This is useful when you want a quick overview of all commits without too much detail.

What does git show do?

git show shows you details of the **most recent commit** (or **any specific commit** you ask for).

It includes:

- Commit ID
- Author name
- Date
- Commit message
- What changed (like a diff)

git show

This shows the latest commit with all the changes

git show <commit-id>

If you want to see a specific commit, use its commit

What is HEAD in Git?

- **HEAD means:**

👉 “Where you are right now” in your Git project history.

It points to the latest commit on the current branch.

HEAD = your current position in the Git history.

Show current commit using HEAD

git show HEAD

Go back to previous commit temporarily

git checkout HEAD~1

You move one step behind current commit

Meaning of HEAD~1:

- Refers to 1 commit before the current one
- You can do HEAD~2 for 2 commits back, and so on

What does git restore do?

- git restore is used to undo changes in your files.
- It helps you go back to the last saved version (from your last commit or stage).

When do you use it?

- Imagine you changed a file, but now you don't want that change — you want to bring it back to how it was before.
- **git restore --source id <filename>**

This will remove all the changes in that file and bring it back to the last commit.

git commit -am "your message"

This command:

1. Adds all modified (already tracked) files to staging, and
2. Commits them with a message.

Add + commit modified files (only tracked)

What is .gitignore?

- It's a special file in your Git project.
- It tells Git:

👉 "Don't track these files or folders."

Why use .gitignore?

Some files should not go into Git, like:

- Passwords or secrets
- Temporary files
- node_modules/, dist/, *.log files, etc.

These files are usually for your computer only — not for sharing.

Git will only **ignore untracked** files.

If a file is already being tracked, .gitignore won't stop it.

Use this to **untrack it**:

git rm --cached filename

git restore .gitignore

This command restores the .gitignore file to its last committed version.

So if you changed .gitignore but didn't commit yet, this will undo your changes

What is a branch in Git?

- A branch is like a separate workspace where you can try changes without affecting the main code.
- The default branch is usually called main or master.

Why use branches?

- To work on new features, fix bugs, or experiment.
- So you don't break the main project while trying something new.

See all branches: **git branch**

Shows the list of branches.

The branch with * is your current branch.

Create a New Branch: - **git branch new-branch-name**


Creates a new branch but doesn't switch to it.

git checkout branch-name or git switch branch-name: -

Changes your working branch.

Create & switch in one step:-

git checkout -b new-branch or git switch -c new-branch

 Delete a branch: -

git branch -d branch-name

What is git merge?

- git merge is a Git command used to combine the changes from one branch into another

Basic Syntax: **-git merge <branch-name>**

Example Situation:

1. You're working on the main branch
2. You create a new branch called feature1 to work on a new feature.
3. After finishing your changes in feature1, you want to bring those changes into main.

Commands:

`git checkout main` # Step 1: Switch to main branch

`git merge feature1` # Step 2: Merge changes from feature1 into main

Now, the main branch will include all the commits from feature1.

What if there's a conflict?

Git will pause and tell you:

CONFLICT (content): Merge conflict in file.txt

You need to open the file, fix it, then run:

`git add file.txt`

`git commit`

What is a tag in Git?

- A tag in Git is like a label or bookmark that points to a specific commit in your project's history.

Why Use Tags?

To mark important points, like:

v1.0

v2.0-beta

release-2025

Commonly used for:

Releases

Deployments

Milestones

Types of Tags

1. Lightweight Tag: - `git tag v1.0`

A simple pointer to a commit (like a branch but doesn't move).

2. Annotated Tag (recommended):- `git tag -a v1.0 -m "Version 1.0 release"`

Includes metadata: tag message, author, date, and can be signed.

Working with Tags

List all tags: - `git tag`

Show details of a specific tag: - `git show v1.0`

Tag an old commit: - `git tag -a v1.0 <commit-id> -m "Tagging old version"`

Delete a tag: - `git tag -d v1.0`

Sharing Tags with Remote (like GitHub)

Push a single tag: - `git push origin v1.0`

Push all tags: - `git push origin --tags`

What is git stash?

git stash lets you temporarily save your uncommitted changes, so you can switch branches or pull updates without losing your work.

It's like putting your work in a safe place for now and restoring it later.

When to use it?

Imagine this:

- You're working on something.
- You haven't committed yet.
- You suddenly need to switch to another branch.

Instead of committing incomplete work, you use: - **git stash**

Now you're working directory is clean, and you can switch branches.

See your stash list: - **git stash list**

Apply your stashed work back: `git stash apply`

(Brings back the latest stashed changes)

Remove the stash after applying `git stash drop` or `git stash pop`

What is "upstream" in Git?

Upstream means the remote branch your local branch is tracking (usually from GitHub or another remote server).

It helps Git know where to pull changes from and where to push changes to.

Example Situation:

You clone a project from GitHub:

git clone https://github.com/user/project.git

Now your local main branch automatically tracks the remote origin/main.

Here, origin/main is your upstream branch.

Why is it useful?

- When you type: - **git push**

Git knows which remote branch to push to (the upstream).

- When you type: - **git pull**

Git pulls from the upstream branch.

How to set upstream manually?

git push --set-upstream origin your-branch-name

Now your branch will track origin/your-branch-name.

Check current upstream: -

git branch -vv

It will show:

- main abc1234 [origin/main] Your latest commit message

git push -u origin master

This command pushes your local master branch to the remote repository (called origin), and:

Sets the upstream for master

So, in the future, you can just run:

git push

You created a new local branch:

git checkout -b feature-login

Now push it and set upstream:

git push -u origin feature-login

From now on, just run git push or git pull from this branch — Git knows where.

Undo commits: -

To undo commits in Git, there are different ways depending on what you want to do

Undo the last commit, but keep changes in your files (soft reset):

git reset --soft HEAD~1

Removes the last commit

Keeps your changes staged (ready to recommit)

Undo the last commit, and unstage the files (mixed reset — default):

git reset HEAD~1

Removes the last commit

Keeps your changes, but unstaged

Undo the last commit and remove changes completely (hard reset):

git reset --hard HEAD~1

Danger! This deletes the commit and your changes.

Undo a pushed commit (safe way) using revert:

git revert <commit-id>

Adds a new commit that reverses changes

Safer for shared repos (GitHub, team work)

What git commit --amend

It modifies your last commit. You can:

1. Change the commit message
2. Add files you forgot to include

Git GUIs (Graphical User Interfaces)

If you prefer using buttons and visuals instead of terminal commands, Git GUIs are perfect for you!

Here are some popular ones with simple descriptions:

GUI Tool Best for Platform

GitHub Desktop GitHub users (easy UI) Windows, macOS

SourceTree Visual Git control Windows, macOS

GitKraken Pro Git workflow Windows, macOS, Linux

TortoiseGit File Explorer users Windows

VS Code Git GUI Code + Git in one app All platforms