* To know the version: $ git --version
* Git is a popular version control system. 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

Key Git Concepts

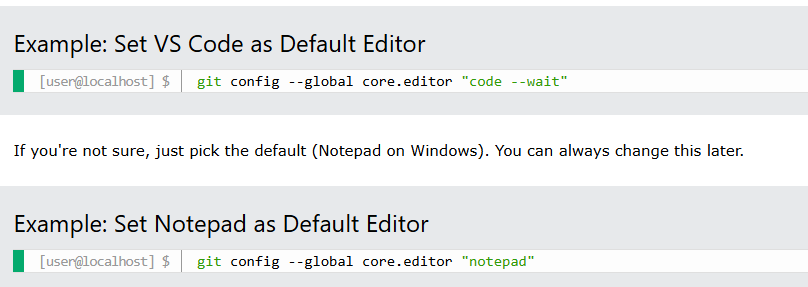
* **Repository:** A folder where Git tracks your project and its history.
* **Clone:** Make a copy of a remote repository on your computer.
* **Stage:** Tell Git which changes you want to save next.
* **Commit:** Save a snapshot of your staged changes.
* **Branch:** Work on different versions or features at the same time.
* **Merge:** Combine changes from different branches.
* **Pull:** Get the latest changes from a remote repository.
* **Push:** Send your changes to a remote repository.

**Note:** Most Git actions (like staging, committing, and viewing history) happen on your own computer. Only **Push** and **Pull** interact with remote servers like GitHub, GitLab, or Bitbucket to upload or download changes.

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.

Default Editor:



**How to Add Git to PATH after Installation**

**Windows:**

* + 1. If you missed the option during installation, search for "Environment Variables" in the Start menu and open it.
    2. Click "Environment Variables..." and find the "Path" variable under "System variables".
    3. Click "Edit", then "New", and add the path to your Git bin and cmd folders  
       (e.g., C:\Program Files\Git\bin and C:\Program Files\Git\cmd).
    4. Click OK to save. Restart your terminal.

**Linux:**

* + Most package managers add Git to PATH automatically.
  + If not, add this line to your ~/.bashrc or ~/.profile:
  + export PATH="/usr/bin:$PATH"
  + Save the file and run source ~/.bashrc or source ~/.profile.

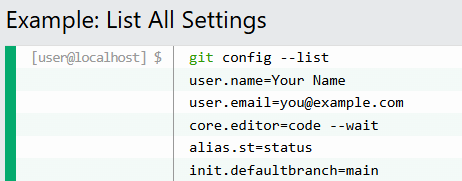
Update git: **brew upgrade git or sudo apt-get upgrade git**

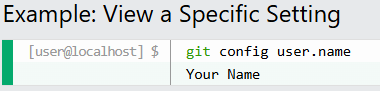
**Configure Git:**

User name: $ git config --global user.name “Your Name”  
Emai: $ git config –global user.email “Your email

Use --global to set the value for **every repository** on your computer.

Use --local (the default) to set it only for the current repository.



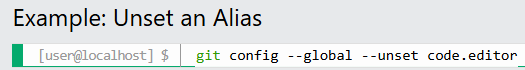


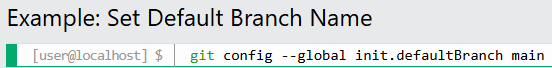
**Changing or Unsetting Config Values:**

To change a value, just run the git config command again with the new value.

To remove a setting, use --unset:

**Now, create one repo in github as well, so that you can copy or store all your files into github.**

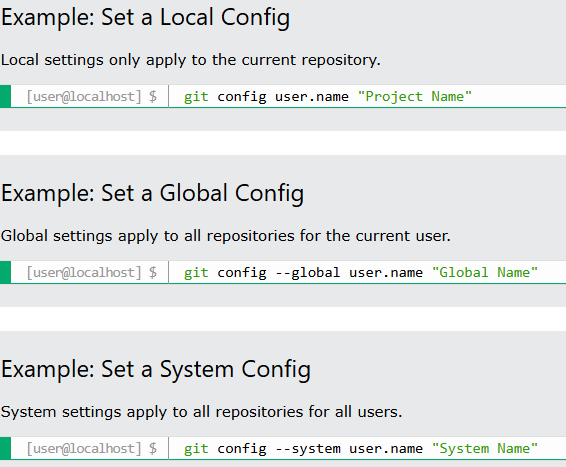


Default Branch Name: Set the default branch name for new repositories (for example, main instead of master): 

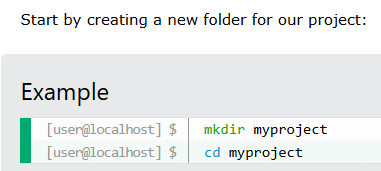
Configuration Levels:

There are three levels of configuration:

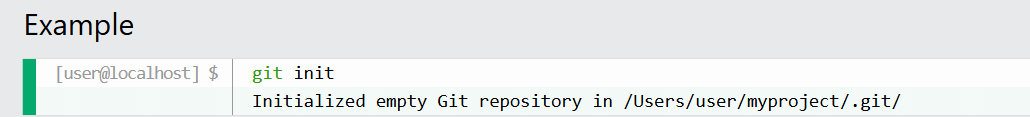
* **System** (all users): git config --system
* **Global** (current user): git config --global
* **Local** (current repo): git config –local



**Get Started with Git**



Initialize Git:



What Happens When You Run git init?

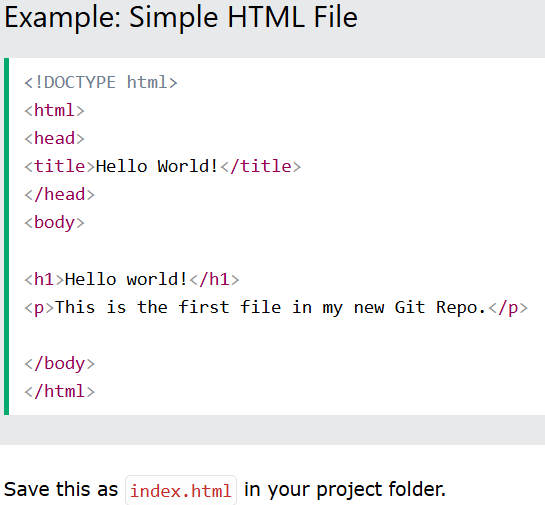
Git creates a hidden folder called .git inside your project. This is where Git stores all the information it needs to track your files and history.

**Git new files:** A new file is a file that you have created or copied into your project folder, but haven't told Git to watch.

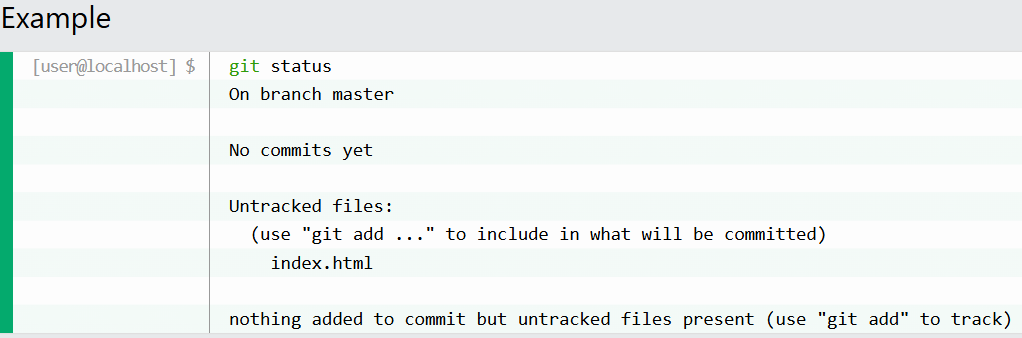
Here are the key things to know:

* Create a new file (with a text editor)
* ls - List files in the folder
* git status - Check which files are tracked
* Understand **untracked** and **tracked** files

Create a new file:



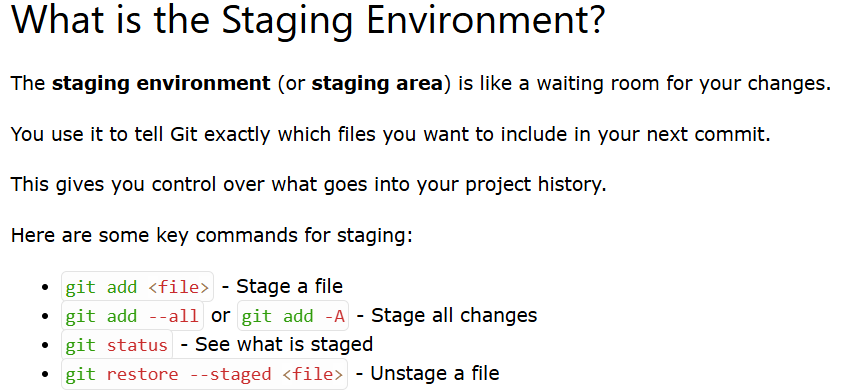
To see which files are in your project folder, use the ls command & Check File Status with git status.



**What is an Untracked File?**

An **untracked file** is any file in your project folder that Git is not yet tracking.

Tracked File: A **tracked file** is a file that Git is watching for changes.



Check Staged Files with git status

* **Staged the wrong file?** Use git restore --staged <file> to unstage it.
* **Forgot to stage a file?** Just run git add <file> again before you commit.
* **Not sure what's staged?** Run git status to see what will be committed.



Commit All Changes Without Staging (-a): You can skip the staging step for **already tracked files** with git commit -a -m "message".

What is Version Control System (VCS)?

- Helps track and manage code changes.

- Two main problems solved:

1. Code Sharing: Combining work from multiple developers.

2. Versioning: Moving between historical versions of code.

Why Git Became Popular

- Older VCS: CVS, SVN (centralized).

- Git: Distributed VCS allowing developers to work offline and push changes.

- Solves single point of failure issue.

Centralized vs Distributed VCS

Centralized:

- Requires central server (e.g., SVN).

- Server downtime breaks collaboration.

Distributed:

- Each developer has full repo copy.

- Changes can be made locally and pushed later.

Analogy: Centralized = Shared Google Doc, Distributed = Personal copies synced via GitHub.

What is a Fork?

- A complete personal copy of a repository.

- Used for experimentation or proposing changes to original project.

- Common in open-source contributions.

Git vs GitHub

Git:

- CLI tool for version control.

- Installed locally.

GitHub:

- Web-based hosting service for Git repos.

- Adds collaboration tools (issues, pull requests, CI/CD).

Analogy: Git = Engine, GitHub = Car UI + Dashboard.

Git & GitHub - DevOps Zero to Hero (Day 9) Revision Sheet

Common Git Commands

git init - Initialize repo

git add - Stage files

git commit - Save version

git status - View repo status

git diff - See file changes

git log - View commit history

git reset - Revert to earlier commit

Using GitHub

- Create account at github.com

- Create a new repository (public/private)

- Push local repo using remote URL

- Use README.md for project details

- Collaborate via pull requests, forks, and issues

Interview Questions

1. Difference: Git vs GitHub

2. What is VCS?

3. Centralized vs Distributed VCS?

4. What is a fork?

5. How to roll back changes in Git?

6. Basic Git commands

**Git & GitHub Flashcards (Q&A Format)**

**🔹 Basics**

**Q:** What is Git?  
**A:** A distributed version control system used to track changes in source code.

**Q:** What is GitHub?  
**A:** A cloud-based platform that hosts Git repositories and adds collaboration features like pull requests, issues, and CI/CD tools.

**🔹 Version Control**

**Q:** What is a Version Control System (VCS)?  
**A:** A system that tracks changes to files over time and allows multiple people to collaborate on code.

**Q:** What problems does VCS solve?  
**A:** 1. Code sharing between developers. 2. Managing versions and rollback of code.

**🔹 Centralized vs Distributed**

**Q:** What is a centralized version control system?  
**A:** A VCS with a single central server (e.g., SVN, CVS). All changes flow through it.

**Q:** What is a distributed version control system?  
**A:** Each user has a complete local copy of the repository (e.g., Git), enabling offline commits and fault tolerance.

**🔹 Git Commands**

**Q:** What does git init do?  
**A:** Initializes a new Git repository.

**Q:** What does git add do?  
**A:** Stages changes to be committed.

**Q:** What does git commit -m "message" do?  
**A:** Saves staged changes with a commit message.

**Q:** What does git status do?  
**A:** Shows the current state of the working directory and staging area.

**Q:** What does git diff show?  
**A:** Line-by-line changes between the working directory and the last commit.

**Q:** What does git log show?  
**A:** Commit history.

**Q:** How to revert to a previous version in Git?  
**A:** Use git reset --hard <commit-id>.

**🔹 GitHub Specific**

**Q:** What is a fork in GitHub?  
**A:** A full copy of another user's repository under your account.

**Q:** What is the purpose of a README file?  
**A:** To describe the project, usage, and any important notes for users.

**Q:** How do you push code to GitHub?  
**A:** Add remote: git remote add origin <url>  
Push code: git push -u origin main

**Q:** Public vs Private repository – difference?  
**A:** Public repos are visible to everyone. Private repos are restricted to invited users.

**🔹 Interview Tips**

**Q:** Git vs GitHub – Key Difference?  
**A:** Git is the tool. GitHub is the platform built on top of Git for hosting and collaboration.

**Q:** What are common Git interview questions?  
**A:** Fork vs clone, Git vs GitHub, commands like add, commit, push, reset, and how to resolve merge conflicts.

**What is Git Branching?**

**🔸 Definition:**

A **branch** in Git is a parallel version of a repository. It allows you to develop features, fix bugs, or test ideas **independently** of the main production code.

**🔸 Analogy:**

Think of a branch as a **sandbox** where you can make changes without affecting the main product. Once you're confident, you **merge** the changes back.

**🔸 Real-World Example:**

Imagine Uber adds a new **"bike"** service:

* Uber creates a feature-bike branch.
* Devs work independently on this branch.
* Once tested and verified, the branch is merged into the main code (e.g., main or master).

**🧱 Why Use a Branching Strategy?**

A branching strategy defines how your team uses branches to develop and deliver software efficiently. It:

* Encourages **team collaboration**.
* Prevents **production issues** during active development.
* Helps with **code isolation**, **versioning**, and **release planning**.
* Ensures **continuous delivery** to customers (e.g., every 15 days, 1 month, etc.)

**📂 Types of Git Branches**

Abhishek introduces 4 key branch types:

**1. 🧵 Main/Master (aka Trunk)**

* **Default** primary branch.
* Used for **active development**.
* Must always reflect the latest **stable state**.
* All feature, hotfix, and release branches are eventually merged into this.

**2. 🌿 Feature Branch**

* Created to build a **new functionality** or module.
* Example: feature/percentage, feature/bikes
* Developers collaborate here until the feature is stable.
* Merged back into main once complete and tested.

**🔄 Life Cycle:**

* Branch → Develop → Test → Merge → Delete

**3. 🚚 Release Branch**

* Created from main when you're ready to prepare a **new version for deployment**.
* Used for **final testing**, bug fixes, and staging before customer delivery.
* Keeps development on main isolated from the version being released.

**🔄 Example:**

text

CopyEdit

main → release/v2.0 → QA Testing → Production Deployment

**4. 🔥 Hotfix Branch**

* Used when a **critical production bug** is reported.
* Created immediately to fix and deliver fast.
* Changes are merged back into both release and main.

**🔄 Rule:**

Changes in a hotfix must be merged to:

* The **active release branch** (to patch the bug for customers).
* The **main branch** (to keep codebase current).

**📘 Example: Calculator Application**

**Initial Setup:**

* main has basic calculator: add, subtract, multiply, divide.

**Feature Development:**

* Product owner wants **percentage** feature → Create feature/percentage
* Later, wants **exponential** → Create feature/exponent

**Release Planning:**

* Once testing is done, features are merged to main.
* From main, create a release/v2.0 branch to ship this version.

**Post-Release Hotfix:**

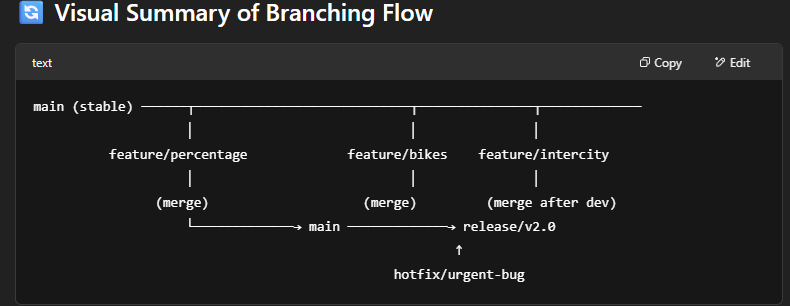
* After v2.0 is released, a production bug is reported.
* Create hotfix/rounding-bug, fix, test, and merge to main + release/v2.0.

**📊 Real Project Example: Kubernetes**

Kubernetes uses the **same strategy**:

* **Main branch** → continuous development
* **Feature branches** → feature/workload-GA, feature/server-set
* **Release branches** → release-1.26, release-1.27
* Many contributors (>3300) work concurrently using branches.

📍 Tip: Check [Kubernetes GitHub repo](https://github.com/kubernetes/kubernetes) to observe live branching in action.



**Branching & Merging**

bash

CopyEdit

git branch feature-x # Create new branch

git checkout feature-x # Switch to it

git checkout -b feature-x # Create & switch

# After changes

git add . && git commit -m "feature done"

git checkout main # Switch to main

git merge feature-x # Merge feature into main

**✅ 9. Pull & Push**

bash

CopyEdit

git push origin main # Push local to GitHub

git pull origin main # Pull updates from GitHub

**✅ 10. Cloning a Repo**

bash

CopyEdit

git clone https://github.com/username/repo.git

**✅ 11. .gitignore File**

* Used to **ignore files** from being tracked by Git.

plaintext

CopyEdit

node\_modules/

.env

\*.log

**🎯 Final Simple Project — *To-Do App with Git & GitHub***

**📝 Step-by-Step**

1. **Create folder**: mkdir todo-app && cd todo-app
2. **Create file**: touch index.html
3. Add basic HTML:

html

CopyEdit

<!-- index.html -->

<!DOCTYPE html>

<html>

<body>

<h1>My To-Do List</h1>

<ul>

<li>Learn Git ✅</li>

<li>Push to GitHub ✅</li>

</ul>

</body>

</html>

1. **Initialize Git**:

bash

CopyEdit

git init

git add .

git commit -m "Initial commit"

1. **Create GitHub Repo** (on GitHub) and **connect**:

bash

CopyEdit

git remote add origin https://github.com/yourusername/todo-app.git

git branch -M main

git push -u origin main

🎉 Your first GitHub project is live!

**🧪 Interview Questions (Basic)**

1. What is Git and how is it different from GitHub?
2. What does git add do?
3. Explain the purpose of a commit.
4. What is a branch and why is it useful?
5. How do you resolve a merge conflict?
6. Difference between git pull and git fetch?
7. What’s the use of .gitignore?