# Git (Global Information Tracker)

Git is a distributed version control system.

# What is Version Control?

When working on a project, we often make multiple updates and save these updates in different folders to access previous versions when needed. Version control simplifies this process by tracking changes in a structured way.

# Why Use Git?

***Collaboration***

• Git allows multiple developers to work on the same project simultaneously without overwriting each other’s changes.  
• Using branches, team members can work on different features or bug fixes in isolation and then merge their work into the main project when ready.  
• It provides tools to resolve conflicts if two people modify the same part of the code.

***Example:***  
Developer A works on a new feature, and Developer B fixes a bug. Both can work independently, and Git will help integrate their changes.

***Versioning***

Git is a Version Control System (VCS) that tracks changes to your codebase over time.  
Each time you make a change and commit it, Git creates a snapshot of your project, allowing you to:  
• Revert to a previous state if something breaks.  
• Compare different versions of the code.  
• Identify when and by whom a change was made.

***Benefits of Versioning:***

• Traceability: You can see the entire history of changes and who made them.  
• Recovery: If you introduce a bug, you can roll back to a previous working version.  
• Experimentation: You can try new ideas on a separate branch without affecting the main codebase.

***Example:***  
Version 1.0 of your project is stable, and you want to add a new feature. Using Git, you can branch out, develop the feature, and return to Version 1.0 anytime if needed.

# Types of Version Control Systems

# 1. Local Version Control

In this system, developers save projects on their local machines.

Disadvantage: If the machine crashes, all files are lost.

# 2. Centralized Version Control System (CVCS)

A central repository is used for collaboration. Developers copy the repository to their local machines, make changes, and commit these changes back to the central server.

Advantages:  
• Everyone has access to the central repository.  
• Changes are visible to all collaborators after each commit.

Disadvantages:  
• If the central server goes down, all progress and access to the repository are lost.

# 3. Distributed Version Control System (DVCS)

Each user has a full copy of the repository, including the complete version history.

Advantages:  
• Local backups of the repository exist for every user.  
• Collaboration is seamless, even if the central server goes down.

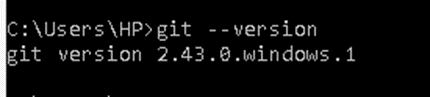
# Downloading Git

Git download link:

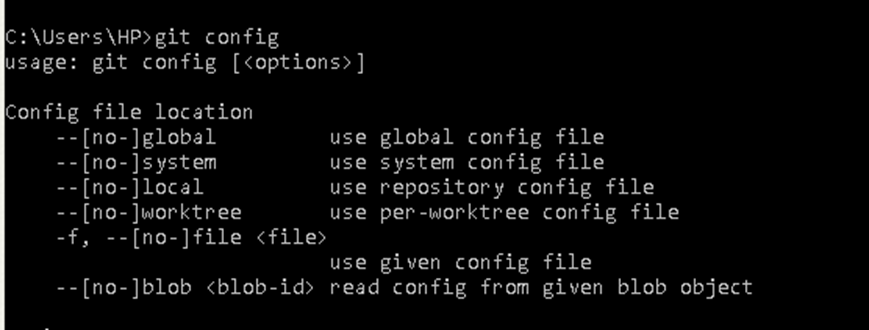
https://git-scm.com/downloads

After installation, verify Git by running

git --version

[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/0.png)

# Configuring Git

Git needs to be configured so we know who is making changes.  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/1.png)

# Check Existing Configuration

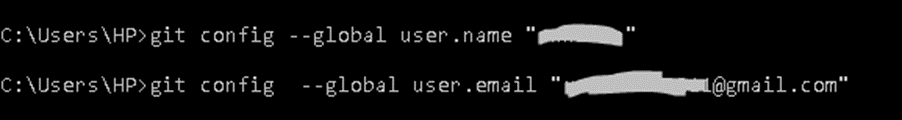
git config --global –list

[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/2.png)

Set User Name and Email

git config --global user.name "your-name"

git config --global user.email "your-email"

[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/3.png)

We configure Git to provide information about who is making changes to the repository.  
Configuration ensures that all changes, commits, and contributions are properly attributed to the correct user, making it easier to track and manage work in collaborative environments.

***Identify Contributors***  
Each commit is tagged with the user.name and user.email of the person who made the change, helping teams track contributions.

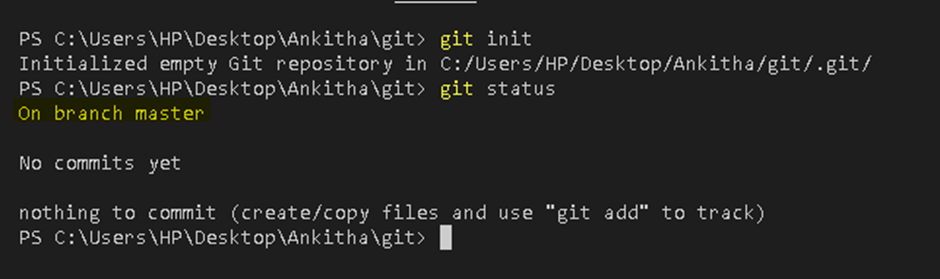
***Improve Collaboration***  
In a team setting, knowing who made specific changes helps in communication and accountability.

***Enable Git Functionality***  
Git requires a user identity (user.name and user.email) to associate commits with a specific person. Without this configuration, Git may display warnings or fail to commit changes.

# Creating a Local Repository

# Initialize a Repository:

git init

By default, Git creates a branch named master.  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/4.png)

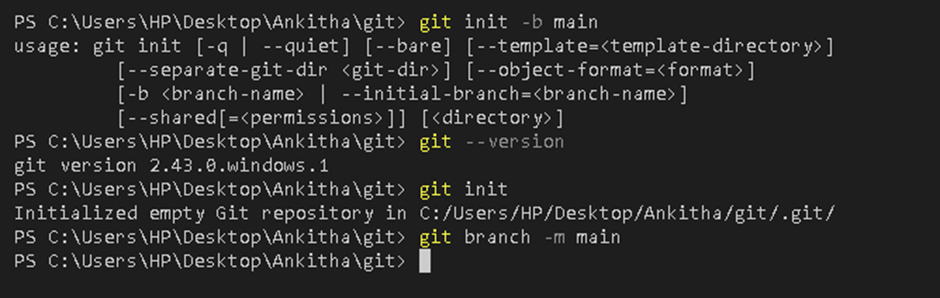
# Initialize with a Custom Branch Name:

git init -b main

• If your Git version is older than 2.28.0, the -b option won't work.  
• To update Git, download the latest version from the Git website.  
• Alternatively, use the following commands to rename the default branch:

git init

git branch -m main or git branch -M main

[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/5.png)

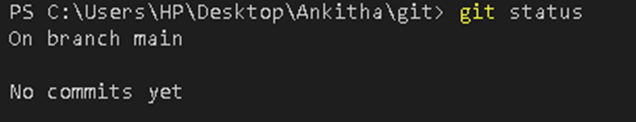
# Understanding the .git Folder:

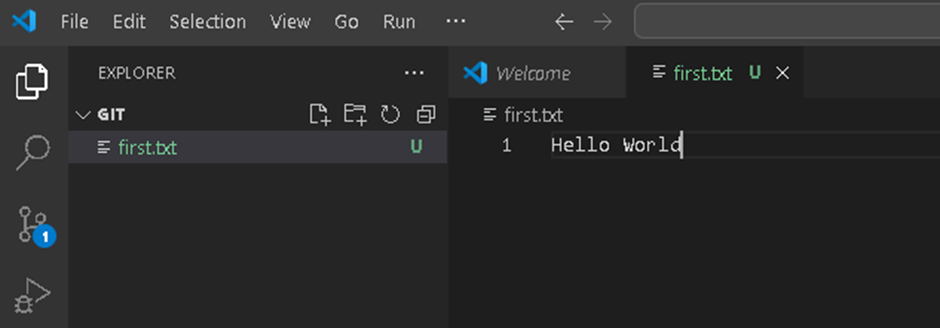
The .git folder contains all the metadata for the repository, including:  
• The staging area  
• Commit history  
• Other tracking information

# Basic Git Commands

1. Check the Project Status

git status

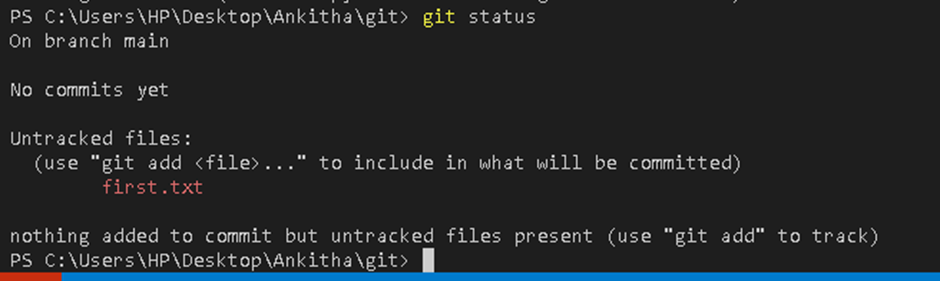
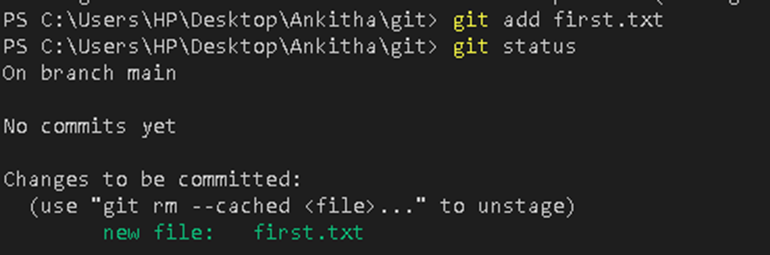
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/6.png)

1. Create a File :  
   Create a new file in the working directory.  
   [](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/7.png)
2. Track Changes  
   Add the file to the staging area:

git add filename

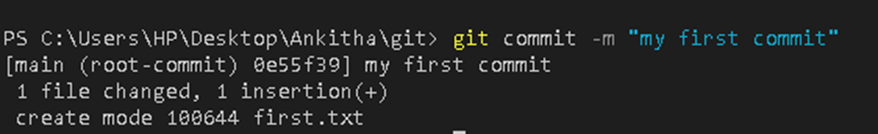
if you want to add multiple files

git add .

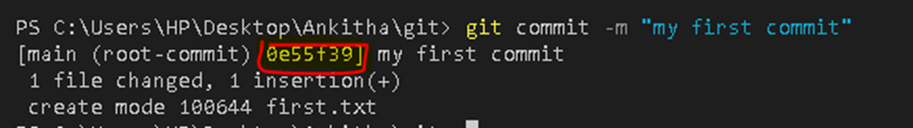
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/8.png) [](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/9.png)

1. Save or Commit Changes  
   Commit the changes with a message:

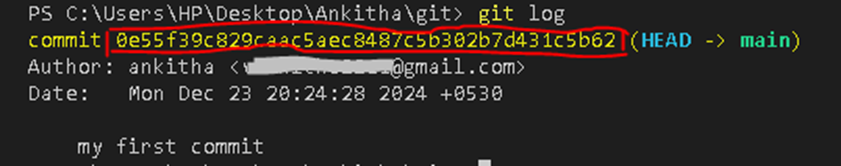
git commit -m "your-message"

After committing, Git starts tracking the file.  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/10.png)

# Git and Checksum

• Git creates a checksum for every commit to uniquely identify it.  
• The checksum is a 40-character hexadecimal string.  
• In some cases, Git displays only the first 7 characters of the checksum for brevity  
Example of a checksum:  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/11.png)

# Viewing Commit History

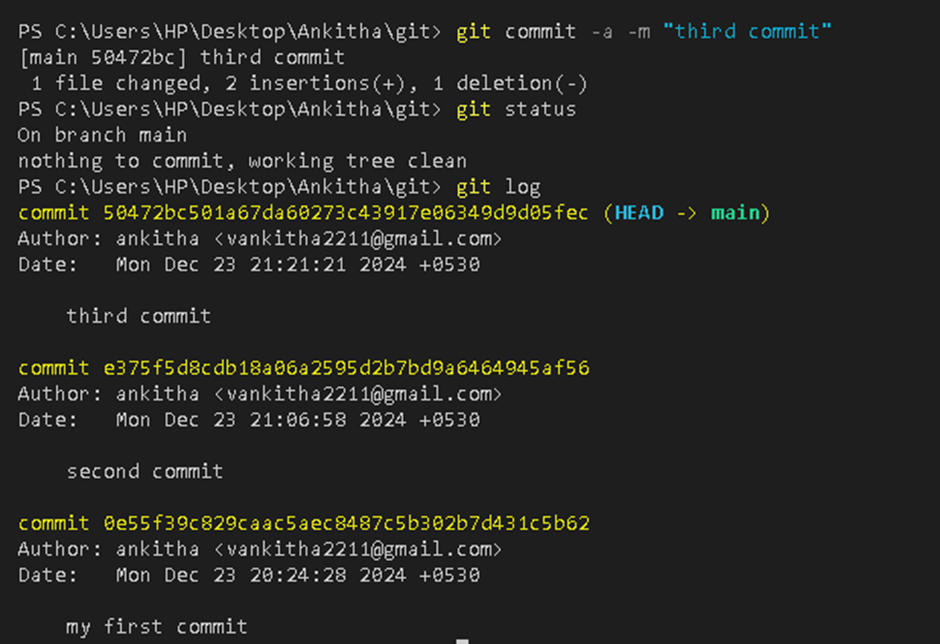
• Use git log to see the log of all commits.  
• The log includes details such as:  
• Full 40-character checksum  
• Commit author  
• Date of the commit  
• Commit message  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/12.png)

# Skipping the Staging Step

If you want to skip adding changes to the staging area before committing, use the -a flag.

Command:

git commit -a -m "message"

This automatically stages all modified files (but not new, untracked files) and commits them in one step.  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/13.png)

# Git Commands for Managing Changes and Files

1. Git diff Command  
   The git diff command is used to find differences between changes you have made and the previous state of the repository. It helps in identifying what has changed in the files.  
   Purpose  
   • To see the changes made in your working directory compared to the last commit.  
   • To review modifications before staging or committing them.  
   Common Scenarios
2. View Changes in the Working Directory  
   Show differences between files in the working directory and the latest commit:

git diff

1. View Staged Changes  
   Show differences between staged changes (added with git add) and the latest commit:

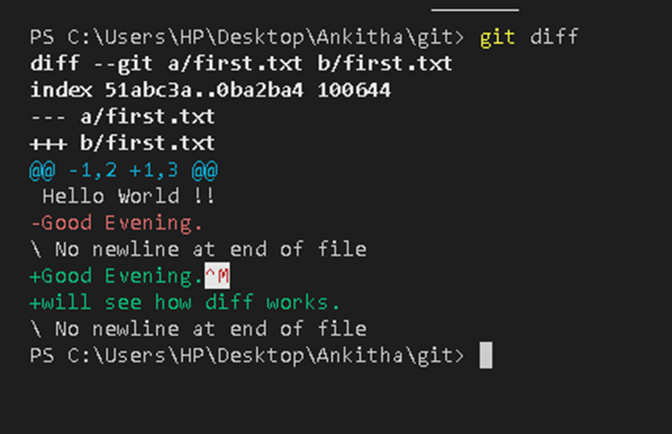
git diff --cached

1. Compare Two Commits  
   Show differences between two specific commits:

git diff commit1 commit2

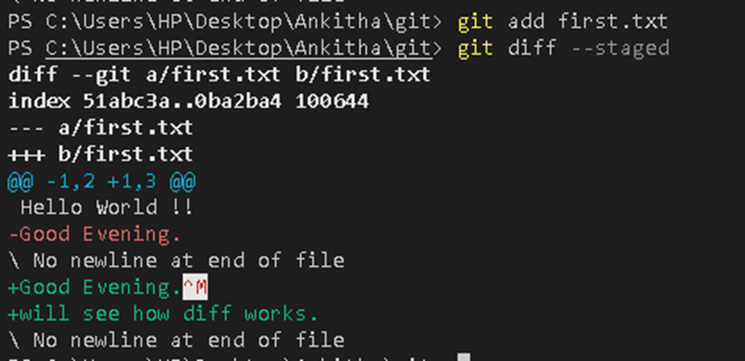
1. Compare a Branch with the Current Branch  
   Show differences between a branch and your current branch:

git diff branch\_name

Output  
• The output shows line-by-line differences:  
• Lines added are prefixed with +.  
• Lines removed are prefixed with -.  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/14.png)

# To see differences between the staging area and the last commit:

git diff --staged

[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/15.png)

***1. Remove a File from Git Repository***

To remove a file from the Git repository while keeping it in the local file system, use:

git rm --cached filename

***2. Clone a Repository***

If you have a repository and want to copy it to your local machine, use the git clone command:

git clone repository-url

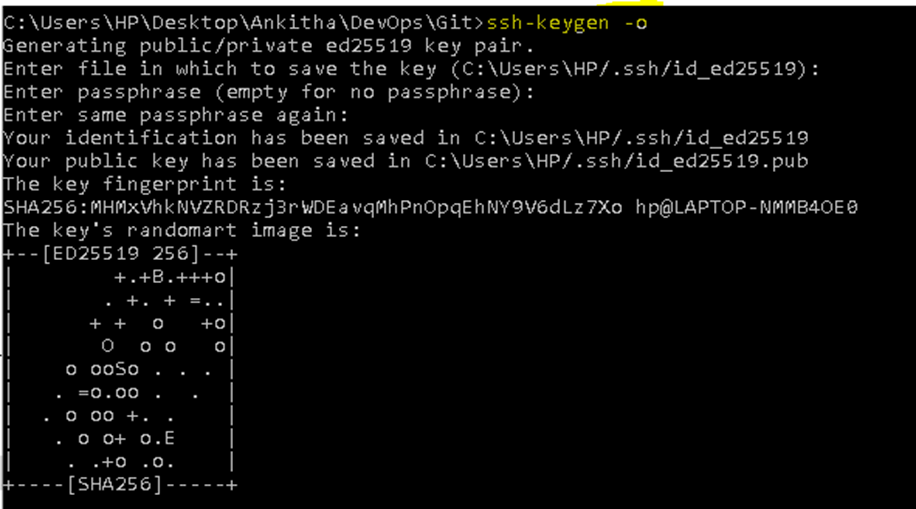
• The repository-url can be an HTTPS URL, SSH URL, or a local file path.  
• After cloning, you will have a complete copy of the repository, including all files and version history.

# How to Connect a Local Repository to a Remote Repository

1. Generate an SSH Key (One-Time Setup)  
   To enable secure communication between your local machine and a remote repository (e.g., GitHub):
2. Run the following command to generate an SSH key:

ssh-keygen -o

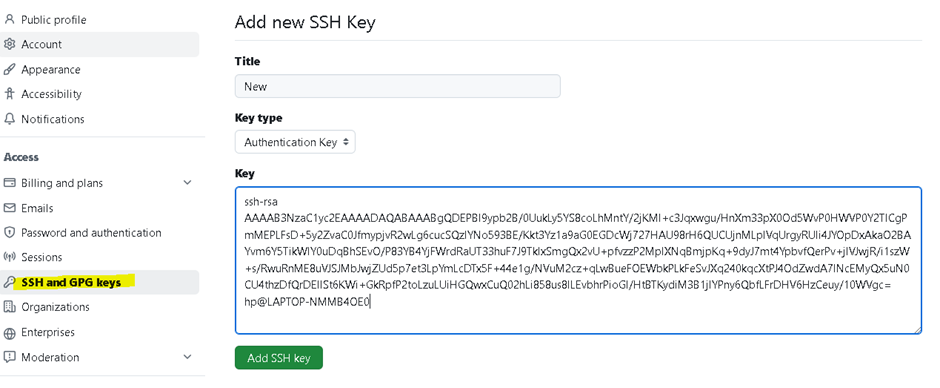
The key will be saved in the .ssh directory on your machine.

1. Locate the generated key in the .ssh folder (typically ~/.ssh/id\_rsa.pub).  
   [](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/16.png) [](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/17.png)
2. Add the SSH Key to GitHub
3. Copy the contents of the public key file (id\_rsa.pub). You can use the following command:

cat ~/.ssh/id\_rsa.pub

1. Log in to your GitHub account.
2. Navigate to Settings → SSH and GPG keys.
3. Click on New SSH key.
4. Provide a title for the key (e.g., "My Laptop SSH Key").
5. Paste the copied key into the Key field.
6. Click Add SSH key.

Now, your local machine (client) and GitHub (server) are connected. This allows GitHub to verify your identity when you push changes.

[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/18.png)

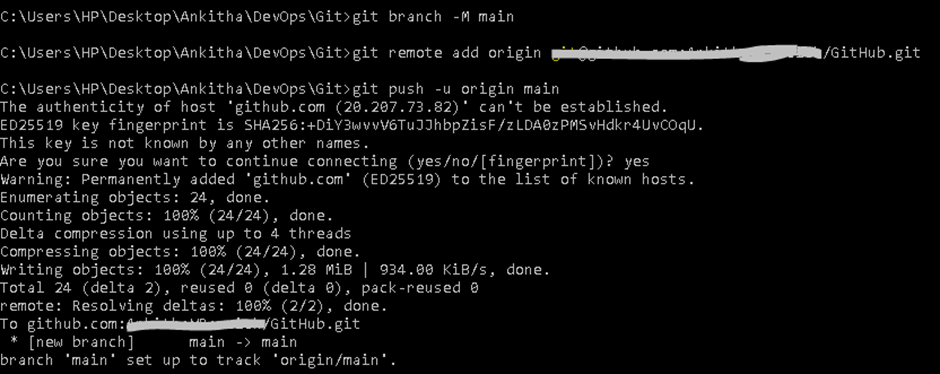
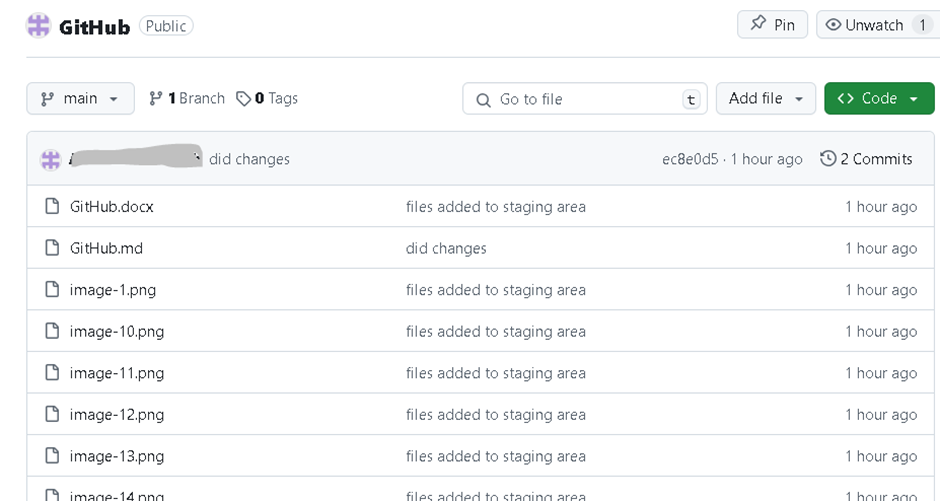
1. Connect the Local Repository to a Remote Repository  
   Run the following command to link your local repository to the remote repository:

git remote add origin <repository-url>

• Replace with the SSH URL of your GitHub repository.  
• The origin keyword is an alias for the remote repository.

1. Push Code from Local to Remote  
   Use the following command to push your code to the remote repository for the first time:

git push -u origin main

• -u sets the upstream branch so that future pushes can be done using git push.  
• main is the branch name. Replace it with the appropriate branch name if different.  
[](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/19.png) [](https://github.com/AnkithaVRamaiah/GitHub/blob/main/image/20.png)

# Tagging

1. View Tags: Displays all tags associated with the project.

git tag

1. Create a Tag:  
   • Lightweight Tagging: A simple tag pointing to a commit without additional metadata. git tag tagname  
   • Annotated Tagging: Includes additional metadata such as the author's name, date, and a message. This is typically used for releases.

git tag -a tagname -m "annotation message"

1. View Tag Details: Displays detailed information about the specified tag.

git show tagname

1. Push a Tag to Remote:

git push origin tagname

1. View Commit Logs with Tags:Shows commit checksums along with their messages.

git log --pretty=oneline

# Branching

Branches allow you to work on a specific feature or fix bugs without affecting the main branch.

1. Why Use Branching?  
   • Prevent errors in the main branch during development.  
   • Facilitate isolated work on new features.
2. Create a Branch:  
   • Using checkout:

git checkout -b branchname

• Using switch (recommended for newer Git versions):

git switch -c branchname

1. View Branches: • Show local branches:

git branch

• Show all branches (local and remote):

git branch --all

• The \* symbol indicates the branch you are currently on.

1. Switch Between Branches: • Switch to a specific branch:

git switch branchname

• Switch to the previous branch:

git switch -

• Using checkout (older syntax):

git checkout branchname

1. Delete a Branch:

git branch -d branchname

1. Push a Branch to Remote:

git push origin branchname

# Commits and Snapshots

1. How Git Tracks Changes:  
   • Each commit creates a snapshot of your files and assigns a unique identifier called a checksum.
2. Pointers:  
   • Git uses pointers to track the latest commit in a branch.  
   • When creating a new branch, the pointer initially points to the same commit as the main branch.

# Merging Branches

1. Why Merge?  
   • Combine changes from one branch into another, typically merging a feature branch into the main branch.
2. Steps to Merge:  
   • Ensure you are on the target branch (e.g., main):

git switch main

• Pull the latest changes to ensure the branch is up-to-date:

git pull origin main

• Merge the desired branch into the main branch:

git merge branchname

# Stashing

Temporarily save changes without committing them:

git stash

Apply stashed changes:

git stash apply

# Rebasing

Move or combine commits from one branch onto another:

git rebase branchname

# Resolving Merge Conflicts

When merging, conflicts may arise if changes overlap. Git will indicate the conflicting files. Resolve conflicts manually in your code editor, then add and commit the resolved files:

git add conflicted\_file

git commit -m "Resolved merge conflict"

# Fetching and Pulling

Fetch: Retrieve changes from the remote without applying them to your local branch:

git fetch

Pull: Fetch and apply changes from the remote to your local branch:

git pull origin branchname

### Git and GitHub Detailed Documentation with Questions and Answers

### 1. **What is Git?**

**Answer:**  
Git is a distributed version control system that allows multiple developers to track changes in files, collaborate on code, and manage different versions of a project efficiently. It works by storing snapshots of files and their history, allowing for easy rollback, collaboration, and version management.

### 2. **What are the main features of Git?**

**Answer:**

* **Distributed Architecture**: Each user has a full copy of the repository, making it resilient and fast.
* **Branching and Merging**: Enables parallel development through branching and merging.
* **Lightweight Tags**: Used to mark specific points in history (usually for releases).
* **Staging Area**: Allows users to prepare changes before committing them.
* **High Performance**: Optimized for handling large projects efficiently.

### 3. **What is the difference between Git and GitHub?**

**Answer:**

* **Git**: A version control system used to track changes locally on a machine.
* **GitHub**: A cloud-based platform for hosting Git repositories that adds collaboration features such as pull requests, issue tracking, and CI/CD integrations.

### 4. **How do you check the version of Git installed?**

**Answer:**  
Run the following command in your terminal:

git --version

### 5. **What is a repository in Git?**

**Answer:**  
A Git repository is a directory where Git tracks and stores all versions of your project's files, along with its history and configuration.

### Initial Setup

### 6. **How do you initialize a Git repository?**

**Answer:**  
Use the command:

git init

This command initializes an empty Git repository by creating a .git folder in your project directory.

### 7. **How do you configure a Git username and email?**

**Answer:**  
Use the following commands:

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

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

### 8. **How do you clone a repository?**

**Answer:**  
Use the following command to clone a repository from GitHub or any remote server:

git clone <repository-url>

### 9. **What is the .gitignore file?**

**Answer:**  
The .gitignore file tells Git which files or directories to ignore when committing. It helps in excluding temporary files, build outputs, or sensitive data.

### Staging and Committing

### 10. **What is the staging area in Git?**

**Answer:**  
The staging area is a place where files are stored before they are committed to the Git repository. You can add files to the staging area using git add.

### 11. **How do you commit changes in Git?**

**Answer:**  
Use the following command to commit changes:

git commit -m "Your commit message"

### 12. **How do you check the status of a repository?**

**Answer:**  
Use the command:

git status

This shows which files are staged, modified, or untracked.

### 13. **What is git diff used for?**

**Answer:**  
git diff shows the differences between the working directory and the staging area, or between commits.

### 14. **How do you view commit history?**

**Answer:**  
Use the command:

git log

This shows the commit history in the repository.

### Branching and Merging

### 15. **What is a branch in Git?**

**Answer:**  
A branch is a pointer to one of the commits in the repository. It allows developers to work on different features or fixes simultaneously without affecting the main project.

### 16. **How do you create a new branch?**

**Answer:**  
Use the following command:

git branch <branch-name>

### 17. **How do you switch branches?**

**Answer:**  
Use the command:

git checkout <branch-name>

Alternatively, you can use:

git switch <branch-name>

### 18. **What is the difference between git merge and git rebase?**

**Answer:**

* **git merge**: Combines the histories of two branches, preserving the commit history.
* **git rebase**: Re-applies commits on top of another branch, effectively rewriting history to make the history linear.

### 19. **What is a merge conflict?**

**Answer:**  
A merge conflict occurs when changes made in two different branches overlap. Git cannot automatically merge them and requires the user to manually resolve the conflict before committing.

### 20. **How do you delete a branch?**

**Answer:**

* **Local**: git branch -d <branch-name>
* **Remote**: git push origin --delete <branch-name>

### Tags and References

### 21. **What is a tag in Git?**

**Answer:**  
A tag is a reference to a specific commit in the repository, often used to mark a release point (e.g., v1.0).

### 22. **How do you create a lightweight tag?**

**Answer:**  
Use the command:

git tag <tag-name>

### 23. **What is HEAD in Git?**

**Answer:**  
HEAD points to the current branch reference in the repository, usually the most recent commit.

### 24. **What is a detached HEAD?**

**Answer:**  
A detached HEAD occurs when Git points to a specific commit rather than a branch. This happens when checking out a commit directly instead of a branch.

### 25. **What is git cherry-pick?**

**Answer:**  
git cherry-pick applies a specific commit from one branch to another branch, without merging the entire branch.

### Undoing Changes

### 26. **How do you undo the last commit?**

**Answer:**

* **Soft reset** (keeps changes staged):

git reset --soft HEAD~1

* **Hard reset** (discards changes):

git reset --hard HEAD~1

### 27. **What is git stash?**

**Answer:**  
git stash temporarily saves changes in the working directory that are not ready for commit. You can retrieve them later using git stash pop.

### 28. **How do you revert a commit?**

**Answer:**  
Use the following command to create a new commit that undoes the changes:

git revert <commit-hash>

### 29. **What is git clean?**

**Answer:**  
git clean removes untracked files from the working directory. Use cautiously as it deletes files not tracked by Git:

git clean -f

### GitHub Basics

### 30. **What is GitHub?**

**Answer:**  
GitHub is a platform for hosting Git repositories with additional features like collaboration tools, pull requests, issue tracking, and CI/CD pipelines.

### 31. **How do you fork a repository on GitHub?**

**Answer:**  
Go to the repository's page on GitHub and click on the "Fork" button. This creates a personal copy of the repository under your GitHub account.

### 32. **How do you create a pull request on GitHub?**

**Answer:**

1. Fork the repository and make changes.
2. Go to the "Pull Requests" tab and click "New Pull Request."
3. Select the base repository and branch, then click "Create Pull Request."

### 33. **What is GitHub Actions?**

**Answer:**  
GitHub Actions is a CI/CD platform that automates workflows like build, test, and deployment directly within GitHub repositories.

### 34. **How do you secure your GitHub repository?**

**Answer:**

* Enable branch protection rules.
* Use Two-Factor Authentication (2FA).
* Limit and review access permissions for collaborators.

### Advanced Git

### 35. **What is git reflog?**

**Answer:**  
git reflog keeps track of all changes to the HEAD reference, including changes that are not part of the commit history.

### 36. **How do you compare two commits?**

**Answer:**  
Use the command:

git diff <commit1> <commit2>

### 37. **What is git blame?**

**Answer:**  
git blame shows which author made changes to each line in a file and when the change was made.

### Submodules and Hooks

### 38. **What is a submodule in Git?**

**Answer:**  
A submodule is a repository embedded inside another repository. It is often used to manage dependencies or separate projects.

### 39. **What are Git hooks?**

**Answer:**  
Git hooks are scripts that automatically run at certain points during Git's lifecycle (e.g., pre-commit, post-merge) to enforce rules or trigger actions like testing.

### 40. **How do you set up a global .gitignore file?**

**Answer:**  
Create the file and configure it globally with:

touch ~/.gitignore\_global

git config --global core.excludesfile ~/.gitignore\_global

### Collaboration and Workflow

### 41. **What is the purpose of git fetch?**

**Answer:**  
git fetch downloads changes from the remote repository without merging them into your local branch. It allows you to review changes before applying them.

### 42. **How do you resolve a merge conflict?**

**Answer:**

* Manually edit conflicting files.
* Stage the resolved files using git add.
* Commit the resolution with git commit.

### 43. **What is the difference between SSH and HTTPS cloning in Git?**

**Answer:**

* **SSH**: Secure and requires an SSH key for authentication.
* **HTTPS**: Requires a username and password for each interaction.

### 44. **What is git bisect?**

**Answer:**  
git bisect helps identify which commit introduced a bug by performing a binary search through the commit history.

### 45. **How do you squash commits?**

**Answer:**  
Use the following interactive rebase command:

git rebase -i <base-branch>

Choose "squash" to merge multiple commits into one.

### Advanced Branching Strategies

* **Git Flow**: A branching model with distinct branches for features, releases, and hotfixes.
* **GitHub Flow**: A simpler model with just main and feature branches, ideal for CI/CD.
* **Trunk-based Development**: Small, frequent merges directly into the main branch.

### Setting Up CI/CD Pipelines with GitHub Actions

GitHub Actions automates tasks like build, test, and deployment directly in GitHub. Features include caching, secrets management, and matrix builds.

### Handling Large Repositories with Git LFS

**Git LFS** is used to store large binary files like images and videos outside of the repository, tracking them with pointers.

git lfs install

git lfs track "\*.psd"

git add .gitattributes

### Automating Workflows Using Git Hooks

Git hooks trigger actions during Git operations. Examples include pre-commit checks or post-push notifications.

Example pre-commit hook:

#!/bin/sh

npm test

### Migrating Repositories to GitHub

1. Create a new repository on GitHub.
2. Add GitHub as a remote to your local repository:

git remote add origin https://github.com/username/repo.git

1. Push the repository:

git push -u origin main

### Practice Questions

1. **What is the difference between git fetch and git pull?** **Answer**: Fetch downloads updates without merging, while pull downloads and merges updates.
2. **How do you view a specific commit?** **Answer**: Use git show <commit-hash>.
3. **What is the purpose of .gitignore?** **Answer**: To specify files and directories that Git should ignore.
4. **How do you revert a commit?** **Answer**: Use git revert <commit-hash> to undo the changes.
5. **How do you create a lightweight tag in Git?** **Answer**: Use `git tag <tag-name