**Basic Git Questions**

**1. What is Git and why is it used?**

Git is a distributed version control system that helps developers track changes in files and coordinate work among multiple people. It is widely used in software development to manage source code efficiently. Git allows developers to work independently on different features, commit their changes, and merge them with the main project without overwriting each other's work.

**The key advantages of Git include**:

* **Version Control**: Every change made to the project is recorded, allowing users to revert to previous versions if necessary.
* **Branching and Merging**: Developers can create branches to work on new features without affecting the main project and later merge them.
* **Collaboration:** Multiple developers can work on a project simultaneously.
* **Distributed System:** Every developer has a full copy of the repository, ensuring no data loss even if the central server goes down.

**2.Explain the difference between Git and GitHub.**

Git and GitHub are related but serve different purposes:

* Git is a command-line tool that provides version control functionalities. It runs locally on a developer’s machine and allows tracking changes, creating branches, merging code, and managing repositories.
* GitHub is a cloud-based platform that hosts Git repositories, enabling collaboration between developers. GitHub provides additional features such as:
* Remote Repository Hosting: Developers can store their Git repositories on GitHub and access them from anywhere.
* Pull Requests and Code Review: Developers can request feedback on code before merging it into the main branch.
* Issue Tracking and Project Management: GitHub offers tools to track bugs, manage tasks, and streamline development workflows.
* Alternative platforms to GitHub include GitLab, Bitbucket, and Azure DevOps, which provide similar hosting services for Git repositories.

**3.How do you install Git on your machine?**

Installing Git depends on the operating system:

**Windows:**

Download Git from [git-scm.com](https://git-scm.com/).

Run the installer and follow the setup instructions.

After installation, open Command Prompt and verify by running:

git --version

**macOS:**

Open Terminal and check if Git is installed using git --version.

If not installed, use Homebrew:

brew install git

**Linux:**

Use the package manager to install Git:

sudo apt update && sudo apt install git

sudo yum install git

sudo dnf install git

**4.How do you configure your username and email in Git?**

After installing Git, it’s important to set up your username and email, as these details are associated with every commit you make. Run the following commands:

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

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

The --global flag ensures these settings apply to all Git repositories on your machine. If you want to configure them for a specific project only, remove the --global flag and run the commands inside the project directory.

To verify your settings, run:

git config --global --list

This command will display the currently configured username and email.

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

A repository (repo) in Git is a storage location that contains all files, folders, and the complete history of changes for a project. It tracks every modification made to the code and allows developers to collaborate efficiently.

**There are two types of repositories in Git:**

**Local Repository:** Stored on a developer’s computer and used for making changes before pushing them to a remote repository.

**Remote Repository:** Hosted on platforms like GitHub, GitLab, or Bitbucket, allowing multiple developers to contribute.

To initialize a repository, use:

git init

This creates a .git directory that stores the project history.

**6.How do you create a new Git repository?**

To create a new Git repository, follow these steps:

Step1:Navigate to your project folder in the terminal.

Step2:Run the following command:

git init

step3:This initializes an empty Git repository in the project directory, allowing you to start tracking changes.

Step4:To add files, use:

git add .

step5:To make an initial commit, use:

git commit -m "Initial commit"

**7.How do you clone a repository from GitHub?**

Cloning a repository creates a local copy of an existing remote repository.

To clone a repository, use:

git clone <repository\_url>

Example:

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

This will download all files, branches, and commit history to your local machine.

**8.What is the purpose of the .gitignore file?**

The .gitignore file tells Git which files and directories to ignore, preventing them from being tracked in the repository.

Common examples include:

Log files

Compiled binaries

System-generated files like .DS\_Store (macOS) or Thumbs.db (Windows)

Environment files (.env)

Example .gitignore content:

node\_modules/

\*.log

.env

dist/

To apply the .gitignore file, add and commit it:

git add .gitignore

git commit -m "Added .gitignore file"

**9.How do you check the status of your working directory in Git?**

Use:

git status

This command shows:

Which files have been modified

Files staged for commit

Untracked files

It helps developers understand the current state of their repository.

**10.How do you add files to the staging area in Git?**

The staging area is where changes are prepared before committing.

To add a single file:

git add filename.txt

To add all changes:

git add .

To commit the staged changes:

git commit -m "Added new feature"

This ensures that only selected changes are committed.

**Intermediate Git Questions**

**11.Explain the concept of commits in Git.**

A commit in Git is a snapshot of the changes made to a repository at a specific point in time. Each commit records a set of changes along with metadata such as the author's name, email, date, and a commit message. Commits allow developers to track modifications, revert to previous versions, and collaborate efficiently.

**Key features of commits:**

* Atomicity: Each commit is an independent unit of change.
* Unique Hash: Each commit has a unique SHA-1 hash that identifies it.
* Commit Message: A description that explains the purpose of the changes.
* To create a commit:
* git add <filename> # Stage changes
* git commit -m "Descriptive commit message" # Commit changes
* To view the commit history:
* git log

**12.How do you create a new commit in Git?**

* Creating a commit follows these steps:
* Stage changes: Add modified or new files to the staging area using:

git add <filename>

* or to stage all changes:

git add .

* Commit changes: Create a commit with a meaningful message:

git commit -m "Implemented feature X"

* Push changes (if working with remote repositories):

git push origin <branch\_name>

**13.What is the purpose of the git log command?**

The git log command displays the commit history of a repository, helping developers track past changes.

Common usages:

git log # Shows the full commit history

git log --oneline # Displays one commit per line

git log --graph # Shows a graphical representation of branches

git log --author="Your Name" # Filters commits by author

**14.How do you view the history of commits in a repository?**

The git log command provides detailed commit history:

git log

To view commits in a concise format:

git log --oneline --graph --decorate

To view changes in a specific commit:

git show <commit\_hash>

**15.How do you view the changes made in a commit?**

To see the changes introduced in a specific commit:

git show <commit\_hash>

To compare changes between two commits:

git diff <commit\_hash1> <commit\_hash2>

**16.What is branching in Git and why is it useful?**

Branching allows developers to work on new features or fixes independently without affecting the main codebase.

Benefits of branching:

Isolation: Separate features from production code.

Parallel Development: Multiple developers can work simultaneously.

Safe Experimentation: Try new ideas without affecting the stable version.

**Common branching commands:**

git branch <new\_branch> # Create a new branch

git switch <branch\_name> # Switch to a branch

git merge <branch\_name> # Merge a branch into the current branch

**17.How do you create a new branch in Git?**

To create a new branch:

git branch feature-branch

To create and switch to a branch in one step:

git switch -c feature-branch

**18.How do you switch between branches in Git?**

To switch to an existing branch:

git switch <branch\_name>

Alternative (older) command:

git checkout <branch\_name>

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

Merge: Combines changes from one branch into another, keeping the history intact.

git merge <branch\_name>

Rebase: Reapplies commits from one branch onto another, creating a linear history.

git rebase <branch\_name>

Merging is safer, while rebasing creates a cleaner commit history.

**20.How do you resolve merge conflicts in Git?**

When merging branches, conflicts may arise if changes overlap. Steps to resolve:

* Identify conflicts using:

git status

* Manually edit conflicted files.

Stage the resolved files:

git add <filename>

* Complete the merge with:

git commit -m "Resolved merge conflicts"

* If necessary, abort the merge:

git merge --abort

**Git Exercises**

**21.Create a new Git repository and configure your username and email.**

git init

git config --global user.name "Leelavathi chokkapu"

git config --global user.email "leelavathichokkapu@puropalecreations.com"

**22.Create a file, add some content to it, and commit the changes.**

echo "Hello, Git!" > file.txt

git add file.txt

git commit -m "Added file.txt"

**23.Create a .gitignore file and add rules to ignore specific files and directories.**

echo "node\_modules/" > .gitignore

git add .gitignore

git commit -m "Added .gitignore"

**24.Clone an existing repository from GitHub and make some changes.**

git clone <repository\_url>

cd <repository\_name>

echo "New changes" >> file.txt

git add file.txt

git commit -m "Updated file.txt"

**25.Create a new branch, make some changes, and switch back to the main branch.**

git branch new-feature

git switch new-feature

echo "Feature changes" > feature.txt

git add feature.txt

git commit -m "Added feature.txt"

git switch main