**Git Task**

**1.Explain the workflow of git with neat diagram?**

**Ans:**

Git is a distributed version control system that allows multiple developers to work on a project simultaneously without interfering with each other's work.

**Git workflow:**

**1.Working directory:**

* This is your local workspace where you make changes to files.
* Files can be in one of three states: modified, staged, or committed.

**git status # Check changes**

**git diff # View changes in modified files**

**2.Staging Area:**

* After modifying files, you selectively add them to the staging area using git add.
* The staging area is a snapshot of the changes you want to include in your next commit.

**git add file.txt # Add a specific file**

**git add . # Add all changes**

**git reset file.txt # Unstage a file**

**3.Local Repository:**

* Once changes are staged, you commit them to the local repository using git commit.
* Each commit creates a unique snapshot of your project, along with a commit message describing the changes.

**git commit -m "Added new feature" # Commit changes**

**git log # View commit history**

**git reset --soft HEAD~1 # Undo last commit but keep changes staged**

**4.Remote Repository:**

* After committing changes locally, you can push them to a remote repository (e.g., GitHub, GitLab) using git push.
* This allows other developers to access your changes.

**git remote add origin <repo\_url> # Link to remote repository**

**git push origin main # Push changes to the remote repository**

**git pull origin main # Pull latest changes from the remote repository**

**5.Fetching and Pulling:**

* To get updates from the remote repository, you use git fetch to download changes or git pull to fetch and merge them into your local branch.

**git fetch origin # Fetch all changes from the remote repository**

**git fetch origin main # Fetch changes only for the main branch**

**git log origin/main # View remote commits without merging**

**git pull origin main # Fetch and merge changes from remote main branch**

**6.Branching and Merging:**

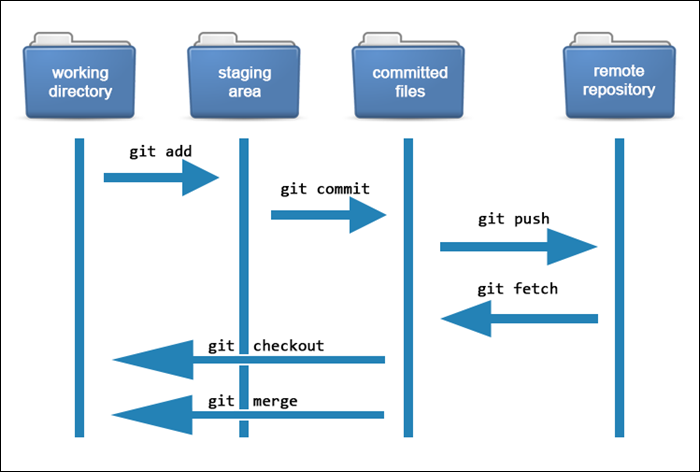
* Git allows you to create branches to work on new features or fixes without affecting the main codebase.
* Once work is complete, you can merge branches back into the main branch using git merge.

**git branch feature-branch # Create a new branch**

**git checkout feature-branch # Switch to the branch**

**git merge feature-branch # Merge into main branch**

**git branch -d feature-branch # Delete a branch**



The Git workflow revolves around the Working Directory, Staging Area, Local Repository, and Remote Repository. Developers make changes locally, stage them, commit them, and then push them to a shared remote repository. Collaboration is facilitated through branching, merging, and syncing with the remote repository. This workflow ensures version control and seamless collaboration among team members.

**2.** **What is git, how to stage a file with git add?**

Git is a **distributed version control system (DVCS)** designed to track changes in source code during software development. It allows multiple developers to collaborate on a project, maintain a history of changes, and manage different versions of the codebase efficiently. Git is widely used in modern software development due to its flexibility, speed, and robust branching and merging capabilities.

Key features of Git:

* **Distributed:** Every developer has a full copy of the repository, including its history.
* **Branching and merging:** Easily create branches for new features or bug fixes and merge them back into the code mainbase.
* **Version Control:** Track changes to files over time, allowing you to revert to previous versions if needed.
* **Collaboration:** Enables teams to work on the same project without overwriting each other's work.

**How to Stage a File with git add**

Staging a file in Git means preparing it to be included in the next commit. When you stage a file, you are adding it to the **staging area** (also called the **index**), which is a snapshot of the changes you want to save in your next commit.

**Steps to Stage a File:**

**1.Check the status of Your Working Directory:**

Before staging files, you can check the status of your working directory to see which files have been modified, added, or deleted.

**git status**

**This will show you:**

* **Untracked files: Files that are not yet being tracked by Git.**
* **Modified files: Files that have been changed since the last commit.**
* **Staged files: Files that are ready to be committed.**

**2.Stage a specified file:**

To stage a specific file, use the git add command followed by the file name:

**git add <file\_name>**

**3.Stage All Changes:**

To stage all changes in the working directory (including new, modified, and deleted files), use:

**git add . or git add –all**

**4.Stage Specific Types of Changes:**

Stage only modified and deleted files (ignore new untracked files):

**git add -u**

Stage only deleted files:

**git add -A**

**5.Verify Staged Files:**

After staging, you can use git status again to confirm that the files have been added to the staging area:

**git status**

Staged files will appear under the section **"Changes to be committed"**.

**6.Commit the Staged Files:**

Once files are staged, you can commit them to the local repository:

**git commit -m "Your commit message"**

🡪Git is a powerful version control system for tracking changes and collaborating on projects.

🡪To stage a file, use the git add command followed by the file name or use git add . to stage all changes.

🡪Staging prepares files for the next commit, which is then saved in the repository with git commit.

**3.Differences between Git, GitHub, GitLab?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Git** | **GitHub** | **GitLab** |
| **Type** | Version Control System (VCS) | Cloud-based Git repository hosting service. | Cloud-based Git repository hosting service. |
| **Purpose** | Tracks code changes locally. | Hosts git repositories for collaboration. | Hosts git repositories with DevOps tools. |
| **Installation** | Installed on local machine | Web-based requires account | Web-based (self-hosted option available) |
| **Main Use** | Local version control | Collaboration and code sharing | Collaboration, CI/CD, DevOps. |
| **Access** | Command-line tool | Web UI + CLI | Web UI + CLI |
| **Repository Hosting** | No hosting (Local only) | Hosts repositories remotely | Hosts repositories remotely |
| **CI/CD(Continuous Integration/Deployment)** | Not included | Requires external CI tools (GitHub Actions) | Built-in CI/CD (Gitlab CI/CD) |
| **Best For** | Local Development | Open-source collaboration, team projects. | Enterprise projects, DevOps integration. |
| **Pricing** | Free and open-source | Free for public repos; paid for private repos. | Free for basic features; paid for advanced. |

**4.How to create a tag and how can we send it?**

Tags in Git are used to mark specific points in a project’s history. They are like Bookmarks for important commits. It is often used to make release points.

**Why Use Tags?**

* Mark **release versions** (e.g., v1.0, v2.0).
* Identify **stable points** in the project.
* Easy to **switch** between versions.

**1.Create a Tag:**

There are 2 types of tags in Git:

* **Lightweight tags:** Just a name for a specific commit.
* **Annotated tags:** Include additional metadata like the tagger’s name , email, date, and a massage.

**Create a Lightweight tag:**

**git tag <tag\_name>**

**Create an Annotated tag:**

**git tag -a <tag\_name> -m "Tag message"**

**Tag a Specific Commit**

**git tag -a <tag\_name> <commit\_hash> -m "Tag message"**

**2.View Tags:**

🡪To list all the tags in your repository:

**git tag**

🡪 To view details about a specific tag:

**git show <tag\_name>**

**3.Push tags to a Remote Repository**

By default, tags are not pushed to the remote repository when you run git push. You need to explicitly push tags.

**Push a Specific Tag**

**git push origin <tag\_name>**

**Push All Tags**

**git push origin –tags**

**4.Delete Tags**

**Delete a local tag:**

**git tag -d <tag\_name>**

**Delete a remote tag:**

**git push origin --delete <tag\_name>**

**5.Explain the steps which are necessary to send a project from local repo to remote repo?**

🡪Follow these steps to push your project from a local repository to the remote repository.

**1.Initalize a git repository:**

**git init**

**2. Add Files to the Staging Area**

🡪Add the files you want to track to the staging area:

**git add .**

🡪The adds all files in the current directory. You can also specify individual files.

**git add <file\_name>**

**3. Commit the Changes**

🡪Commit the staged files to the local repository with a meaningful commit message**:**

**git commit -m "Initial commit"**

**4. Create a Remote Repository**

* Go to your remote hosting service (e.g., GitHub, GitLab, Bitbucket).
* Create a new repository (usually by clicking a "New Repository" button).
* Do not initialize the repository with a README, .gitignore, or license if your local repository already has these files.

**5. Link the Local Repository to the Remote Repository**

🡪Add the remote repository URL to your local repository.

**git remote add origin <remote\_repository\_URL>**

**Verify the remote URL:**

**git remote -v**

**6. Push the Local Repository to the Remote Repository**

🡪Push your local commits to the remote repository:

**git push -u origin main**

* -u sets the upstream branch, so future pushes can be done with just git push.
* Replace main with the name of your default branch if it's different (e.g., master).

**7. Verify the Push**

* Go to your remote repository on the hosting service (e.g., GitHub).
* Check if all the files and commits have been successfully pushed.

**6.What is the pull request, Explain it’s procedure how to raise , when to accept?**

🡪A pull request (PR) is a feature of Git-based platforms like GitHub, GitLab, and Bitbucket that allows developers to propose changes to a codebase.

🡪It is a way to request that the changes in your branch be merged into another branch (usually the main branch). Pull requests facilitate code review, collaboration, and discussion before changes are integrated into the main codebase.

**What is a Pull Request?**

* A pull request is a request to merge changes from one branch into another.
* It provides a platform for code review, where team members can comment on, approve, or request changes to the proposed code.
* It is commonly used in collaborative workflows, especially in open-source projects and team-based development.

**When to Raise a Pull Request?**

* When you have completed a feature, bug fix, or any changes in a branch and want to merge it into the main branch.
* When you want to get feedback from your team or maintainers before merging your changes.
* When you want to ensure that your changes are reviewed and tested before being integrated into the main codebase.

**Procedure to Raise a Pull Request**

**1. Create a Feature/Bugfix Branch**

* Before making changes, create a new branch from the main branch:

**git checkout -b feature-branch**

**2. Make Changes and Commit**

* Make your changes in the new branch.
* Stage and commit the changes:

**git add .**

**git commit -m "Add new feature"**

**3. Push the Branch to the Remote Repository**

* Push the branch to the remote repository:

**git push origin feature-branch**

**4. Open a Pull Request**

* Go to the repository on the hosting platform (e.g., GitHub, GitLab).
* You will see a prompt to create a pull request for the recently pushed branch.
* Click "Compare & pull request" or navigate to the Pull Requests tab and click New Pull Request.

**5. Fill in the Pull Request Details**

* Base branch: The branch you want to merge into (e.g., main).
* Compare branch: The branch containing your changes (e.g., feature-branch).
* Add a title and description explaining the purpose of the pull request.
* Optionally, assign reviewers, add labels, or link issues.

**6. Submit the Pull Request**

* Click Create Pull Request to submit it for review.

**What Happens After Raising a Pull Request?**

1. **Code Review:**
   * Team members or maintainers review the changes.
   * They can leave comments, suggest improvements, or approve the PR.
2. **Continuous Integration (CI) Checks:**
   * If the repository has CI/CD pipelines, automated tests will run to ensure the changes don’t break the codebase.
3. **Address Feedback:**
   * If reviewers request changes, make the necessary updates in the same branch.
   * Push the changes to the remote branch. The PR will automatically update.
4. **Approve the Pull Request:**
   * Once the changes are approved and all checks pass, a maintainer or reviewer can merge the PR.

**When to Accept a Pull Request?**

A pull request should be accepted (merged) when:

1. **Code Review is Complete:**
   * All reviewers have approved the changes.
   * Any feedback has been addressed.
2. **Automated Checks Pass:**
   * CI/CD pipelines (e.g., tests, linting, builds) have successfully completed.
3. **Changes Align with Project Goals:**
   * The changes meet the requirements and align with the project’s coding standards.
4. **No Conflicts:**
   * The branch has no merge conflicts with the base branch. If conflicts exist, they must be resolved before merging.

**How to Merge a Pull Request**

1. Go to the pull request on the hosting platform.
2. Click the Merge button.
3. Choose a merge strategy:
   * **Merge commit**: Creates a new commit for the merge.
   * **Squash and merge**: Combines all commits into a single commit.
   * **Rebase and merge**: Replays the commits on top of the base branch.
4. Confirm the merge.

**Benefits of a Pull Request**

✔ Ensures code quality through **reviews**.  
✔ Enables **team collaboration** and discussions.  
✔ Prevents direct changes to the **main branch**.

**7.What is fork, why it is necessary?**

**What is Fork in Git?**

🡪A Fork is a process of creating a copy of an existing remote repository into your own GitHub, GitLab, or Bitbucket account.

🡪It allows developers to independently experiment, modify, or contribute to a project without affecting the original repository.

**Why is Fork Necessary?**

1. **Open Source Contribution:**  
   It allows external developers to propose changes without direct access to the original project.
2. **Independent Development:**  
   Developers can customize or improve the project without disturbing the main repository.
3. **Code Collaboration:**  
   Forking enables users to collaborate on a project by submitting **pull requests** after making changes.
4. **Backup and Experimentation:**  
   Forking creates a personal copy where developers can experiment without breaking the original project.

**How to Fork a Repository?**

1. Go to the GitHub/GitLab repository page you want to fork.
2. Click the **Fork** button (usually in the top-right corner).
3. The forked repository will be created under your account.

**Working with a fork:**

**1.Clone your fork:**

🡪Clone the forked repository to your local machine:

**git clone https://github.com/your-username/repository-name.git**

**2.Add the original repository a s a remote:**

🡪To sync your fork with the original repository, add the original repository as a remote:

**git remote add upstream https://github.com/original-owner/repository-name.git**

**3. Sync Your Fork with the Original Repository**

🡪Fetch changes from the original repository:

**git fetch upstream**

🡪Merge changes into your local branch:

**git checkout main**

**git merge upstream/main**

**🡪**Push the updated branch to your fork:

**git push origin main**

**4. Make Changes and Push to Your Fork**

🡪Create a new branch for your changes:

**git checkout -b feature-branch**

🡪Make changes, commit them, and push to your fork:

**git add .**

**git commit -m "Add new feature"**

**git push origin feature-branch**

**5. Create a Pull Request**

🡪Go to your fork on GitHub/GitLab.

🡪Click **Compare & pull request** to propose your changes to the original repository.

**When to Use Fork?**

🡪Contributing to open-source projects.

🡪Making personal customizations to a public project.

🡪Testing new features without affecting the main codebase.