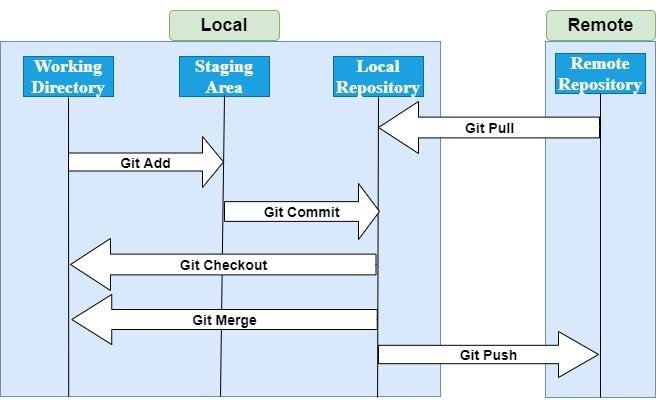
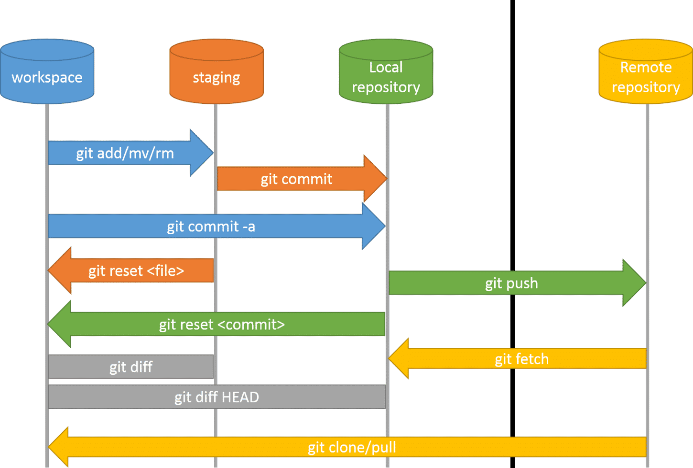
**TASKS ON GIT**

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

ANS: Git Workflow:

Git is a distributed version control system designed to manage and track changes in source code during software development. It enables collaboration among multiple developers by maintaining a history of changes, facilitating branching and merging, and ensuring consistency across local and remote repositories. The Git workflow can be broken down into four key stages: Working Directory, Staging Area (Index), Local Repository, and Remote Repository.



**Git Workflow Stages**

1. **Working Directory**
2. **Staging Area**
3. **Local Repository**
4. **Remote Repository**
5. **Synchronization (Pulling & Fetching Changes)**

**1.Working Directory (Untracked & Modified Files)**

* The working directory contains all project files.
* Changes made in this directory are **not tracked** until they are added to the staging area.
* Any modifications to files are stored temporarily until they are explicitly added for tracking.

Commands used:

git status - Check the status of modified and untracked files

git diff - View differences between working directory and last commit

git checkout -- <filename> - Discard changes in a file.

**2. Staging Area (Indexed Changes)**

* Files added to the **staging area** are prepared for commit.
* The staging area allows developers to selectively commit changes.
* The staging area acts as an intermediate storage before permanently saving changes in the repository.

Commands used:

git add <filename> -Add a specific file to the staging area

git add . -Add all modified files

git reset <filename> - Remove a file from staging

git status -View staged files

git diff --staged - View changes in the staging area

**3. Local Repository (Committed Changes)**

* Committing saves a snapshot of the project.
* Each commit has a unique **hash ID** for tracking.
* The local repository acts as a historical record of all changes, which can be referenced and reverted if needed.

Commands used:

git commit -m "Commit message" -Commit staged changes

git commit -a -m "Commit all tracked files" -Commit all modified files

git log -View commit history

git show <commit\_hash> - View a specific commit

git revert <commit\_hash> - Revert to a previous commit

git reset --hard <commit\_hash> - Reset the repository to a previous state

**4. Remote Repository (GitHub/GitLab/Bitbucket)**

* The remote repository is a **centralized storage** for collaboration.
* Changes are pushed from the local repository to a remote repository.
* Developers can clone, pull, and merge changes from a remote repository.

Commands used:

git remote add origin <repo\_url> - Link local repository to remote

git remote -v -View linked remote repositories

git push origin <branch> - Push commits to remote repository

git push -u origin main -Push for the first time

**5. Synchronization (Pulling & Fetching Changes)**

* Developers **fetch and merge** updates from the remote repository.
* Pulling ensures the local repository remains up-to-date.
* git pull combines git fetch and git merge.

Commands used:

git pull origin <branch> - Fetch and merge remote changes

git fetch -Retrieve changes without merging

git merge <branch> - Merge fetched changes

git rebase <branch> - Apply changes from another branch while preserving history

**Advantages of Git Workflow**

**Pros**

* **Efficient Collaboration:** Allows multiple developers to work on the same project simultaneously.
* **Version Control:** Maintains a complete history of changes.
* **Branching and Merging:** Enables seamless feature development and bug fixes.
* **Distributed System:** Every developer has a full copy of the repository.
* **Backup and Recovery:** Data is safely stored in remote repositories.
* **Integration with CI/CD:** Automates build and deployment processes.

**Cons**

* **Complexity:** Can be challenging for beginners to understand.
* **Merge Conflicts:** Can occur when multiple developers modify the same file.
* **Large Repositories:** Can become slow with large files if not managed properly.
* **Requires Internet for Collaboration:** Remote operations need network access.
* **No Built-in Access Control:** Permissions need to be managed separately.

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

ANS: **Introduction**

Git is a distributed version control system (DVCS) that enables developers to efficiently track changes, collaborate on projects, and manage different versions of code. It plays a crucial role in modern software development, allowing teams to work on the same codebase while ensuring version integrity and history tracking. This document provides an in-depth explanation of the Git workflow, best practices, and essential commands for effective project management.

**What is Git?**

Git is a **distributed version control system (DVCS)** that helps developers efficiently manage and track changes in source code. It enables multiple developers to work on a project simultaneously while maintaining a complete history of all modifications.

**Key Features of Git:**

**Version Control:** Keeps track of all changes made to files.  
**Branching & Merging:** Allows developers to work on features independently.  
**Distributed System:** Every developer has a complete copy of the repository.  
**Collaboration:** Enables multiple contributors to work on the same project. **Backup & Recovery:** Provides a complete history, allowing rollback to previous versions.

**How to Stage a File with git add?**

In Git, staging a file means preparing it for a commit. The **staging area (index)** acts as a temporary storage where changes are added before being committed to the local repository.

**Why is Staging Important?**

* Allows selective commits (you can choose which changes to include).
* Helps organize changes before committing.
* Prevents accidental commits of incomplete code.

**Steps to Stage a File in Git**

**Step 1: Check the status of your working directory**

Before staging, check which files have been modified, added, or deleted.

Syntax: git status

**Red-coloured files** → Modified but not staged.

**Green-coloured files** → Staged and ready for commit.

**Step 2: Add files to the staging area**

1️. **Stage a specific file:**

git add <filename>: syntax

🡪 This adds only the specified file to the staging area.

**2.Stage multiple files:**

git add file1.txt file2.txt: syntax

🡪 This adds only the specified file to the staging area.

**3.Stage all modified files:**

git add . : syntax

🡪 This command stages all changes in the current directory.

**4.Stage files within a specific folder:**

git add <folder\_name> : syntax

🡪 This stages all files within a particular directory.

**Step 3: Verify Staged Changes**

To confirm which files have been staged, run:

Syntax: git status

🡪 This will display files that are ready for commit.

**Step 4: View Changes Before Committing**

To see what changes have been staged:

Syntax : git diff –staged

🡪This will list all modifications that will be included in the next commit.

**Step 5: Remove a File from the Staging Area (Before Committing)**

If you accidentally staged a file and want to remove it from the staging area:

git reset <filename> : syntax

🡪 This un stages the file but keeps the changes in the working directory.

To un stage all files: git reset

**3. what is git, git hub and git lab and differences between them.**

ANS: **what is Git?**

**What is Git?**

Git is a **distributed version control system (DVCS)** that helps developers efficiently manage and track changes in source code. It enables multiple developers to work on a project simultaneously while maintaining a complete history of all modifications.

**Key Features of Git:**

**Version Control:** Keeps track of all changes made to files.  
 **Branching & Merging:** Allows developers to work on features independently.  
 **Distributed System:** Every developer has a complete copy of the repository.  
 **Collaboration:** Enables multiple contributors to work on the same project.  
 **Backup & Recovery:** Provides a complete history, allowing rollback to previous versions.

**What is GitHub?**

GitHub is a **cloud-based hosting service** for Git repositories. It provides tools for version control, collaboration, and project management. GitHub is widely used for **open-source and private repositories**, offering features such as pull requests, issue tracking, and CI/CD integration.

**Key Features of GitHub:**

**Remote Repository Hosting** – Store and manage Git repositories online.  
 **Collaboration Tools** – Supports pull requests, code reviews, and discussions.  
 **Issue Tracking & Project Management** – Includes built-in tools for managing tasks and tracking bugs.  
 **Continuous Integration (CI/CD)** – Integrates with automation tools for testing and deployment.  
 **Security & Access Control** – Allows managing user roles and repository visibility.

**What is GitLab?**

GitLab is an **open-source DevOps platform** that provides **Git repository management, CI/CD pipelines, and security tools** all in one system. Unlike GitHub, which focuses mainly on hosting repositories, GitLab offers a **complete DevOps lifecycle solution**.

**Key Features of GitLab:**

**Integrated CI/CD Pipelines** – Automates testing and deployment workflows.  
 **Issue Tracking & Agile Planning** – Supports Kanban boards and sprint planning.  
 **Self-Hosted Option** – Can be deployed on private servers for more control.  
 **Advanced Security Features** – Includes vulnerability scanning and compliance management.  
 **Built-in Container Registry** – Stores Docker images for easy deployment.

**Differences between the git, git hub and git lab.**

|  |  |  |
| --- | --- | --- |
| Git | Git Hub | Git Lab |
| 1.version control system | 1. cloud based git repository hosting | 1. DevOps platform with git hosting |
| 2.track code changes locally and remotely | 2. provides cloud storage and collaboration features for git | 2. offers full devops lifecycle management |
| 3. not a hosting service | 3.hosted on git hub servers | 3. can be self hosted or cloud based |
| 4. local and remote repositories | 4. supports pull requests, code, reviews and codes | 4.provides advanced issue tracking and planning tools |
| 5.used for version control in local and remote repositories | 5. used for open-source and enterprise projects | 5. used for devops and ci/cd integration |

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

ANS: Git tags are used to mark specific points in the commit history, usually for **versioning software releases**. Tags help in identifying important commits, such as **release versions** (e.g., v1.0.0). They are useful for referring to specific commits, making it easier to navigate project history.

**Types of Git Tags**

Git provides two types of tags:

1. **Lightweight Tags** – A simple reference to a specific commit without extra metadata.
2. **Annotated Tags** – A full object in the Git database, including metadata like the author's name, email, date, and an optional message.

**Creating a Git Tag**

Syntax: git tag <tag name>

Example: git tag v1.0.0

**Create an annotated tag** (recommended for releases since it includes metadata):

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

Example: git tag -a v1.0.0 -m "Version 1.0.0 release"

**Create a tag for a specific commit**

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

example: git tag -a v1.0.0 1a2b3c4d -m "Tagging commit 1a2b3c4d as v1.0.0"

**Tagging with signed verification (GPG-signed tag)**:

Syntax: git tag -s <tag\_name> -m "Tag message"

🡪 This adds cryptographic verification to the tag using GPG keys.

**Pushing Tags to a Remote Repository**

Once a tag is created locally, it must be pushed to a remote repository for sharing.

1️.**Push a single tag:**

Syntax: git push origin <tag\_name>

Example: git push origin v1.0.0

**2.Push all local tags to the remote repository:**

**Syntax:** git push origin –tags

🡪 This pushes all existing tags to the remote repository at once.

**Listing and Deleting Tags**

**List all tags:**

git tag :syntax

🡪 This displays all tags in the repository.

**View details of a specific tag:**

git show <tag\_name> :syntax

example: git show v1.0.0

**Delete a local tag:**

git tag -d <tag\_name> :syntax

example: git tag -d v1.0.0

**Delete a tag from the remote repository**

git push --delete origin <tag\_name> :syntax

example: git push --delete origin v1.0.0

**Delete all local tags at once:**

git tag -d $(git tag) :syntax

🡪 This removes all tags from the local repository.

**Delete all remote tags:**

for tag in $(git tag); do git push --delete origin $tag; done : syntax

🡪This removes all tags from the remote repository.

**Checking Out a Tag**

To check out a specific tag (switch to a tagged version without creating a new branch):

git checkout <tag\_name>: syntax

example : git checkout v1.0.0

However, this puts the repository in a **detached HEAD** state, meaning no new commits can be made directly. To work on a tagged version, create a new branch from the tag:

git checkout -b <new\_branch\_name> <tag\_name> :syntax

example:

git checkout -b hotfix-branch v1.0.0

**Best Practices for Tagging**

✔ Use **annotated tags** for releases to include metadata.  
✔ Follow **semantic versioning** (vMAJOR.MINOR.PATCH, e.g., v2.1.0).  
✔ Always push tags to remote repositories for consistency.  
✔ Delete obsolete tags to maintain a clean repository.  
✔ Use **signed tags** for security when releasing stable versions.

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

ANS: **Steps to Send a Project from Local Repository to Remote Repository**

When working with Git, you often need to push a local project to a remote repository like GitHub, GitLab, or Bitbucket. Follow these steps to successfully send your project from your local repository to a remote repository.

**Step 1: Initialize a Git Repository (If Not Already Initialized)**

If your project is not already initialized as a Git repository, run:

git init : syntax

🡪 This creates a new Git repository in your project folder

**Step 2: Add a Remote Repository**

To link your local repository to a remote repository, run:

Syntax: git remote add origin <remote\_repository\_URL>

Example: git remote add origin <https://github.com/username/repository.git>

🡪 Use git remote -v to verify the remote repository.

**Step 3: Add Files to Staging Area**

Before pushing, ensure all required files are added to Git’s tracking system:

git add . :syntax

example: This stages all modified and new files.

**Step 4: Commit Changes**

Once files are staged, commit them with a meaningful message:

git commit -m "Initial commit":syntax

**Step 5: Push Code to Remote Repository**

Push the local commits to the remote repository:

git push -u origin main:syntax

**Step 6: Verify the Remote Repository**

After pushing, visit your remote repository (e.g., GitHub) to confirm that your files are uploaded.

6.**What is the pull request , explain it procedure.**

ANS: **What is a Pull Request?**

A **Pull Request (PR)** is a feature in Git hosting platforms like **GitHub, GitLab, and Bitbucket** that allows developers to propose changes to a repository. It facilitates **code review, discussion, and collaboration** before merging changes into the main branch.

**When to Use a Pull Request?**

* When contributing to an open-source project.
* When working in a **team** and requiring a review before merging code.
* When integrating **feature branches** into the main branch.
* When submitting **bug fixes or enhancements** for approval.

**Steps to Raise a Pull Request**

**Step 1: Fork the Repository (If Contributing to an External Project)**

If you're contributing to someone else’s project, fork it first:

1. Go to the GitHub repository.
2. Click on Fork (top right corner).
3. This creates a copy in your GitHub account**.**

**Step 2: Clone the Repository Locally**

git clone <repository\_URL> :syntax

example: git clone <https://github.com/your-username/repository.git>

🡪 This downloads the project to your local machine.

**Step 3: Create a New Branch**

Before making changes, create a new branch:

git checkout -b feature-branch: syntax

Example:

git checkout -b add-new-feature

🡪This isolates your work from the main branch.

**Step 4: Make and Commit Changes**

Edit files as needed, then add and commit them:

Syntax:

git add .

git commit -m "Added a new feature"

**Step 5: Push Changes to Your GitHub Repository**

Once committed, push your branch to your remote repository:

Syntax:

git push origin feature-branch

**Step 6: Create a Pull Request on GitHub**

1. Go to your repository on GitHub.
2. Click on Pull Requests → New Pull Request.
3. Select the base branch (e.g., main) and the compare branch (your feature branch).
4. Add a title and a description explaining your changes.
5. Click Create Pull Request.

**Step 7: Review and Merge the Pull Request**

* Other developers review the code and may suggest changes.
* If changes are requested, modify your code and push again:

Syntax:

git add .

git commit -m "Fixed review comments"

git push origin feature-branch

🡪 Once approved, click Merge Pull Request.

**Who Can Access a Pull Request?**

* **Repository Owners & Maintainers:** Can review, approve, and merge PRs.
* **Contributors:** Can create PRs but may require approval to merge.
* **Reviewers:** Assigned team members who check the code before merging
* **What is Forking and Why is it Necessary?**
* A **fork** is a personal copy of another user’s repository that exists in your GitHub, GitLab, or Bitbucket account. Forking allows developers to make changes to a project without affecting the original repository.
* **Why is Forking Important?**
* **Contributing to Open Source:** Developers can contribute to public repositories without directly modifying them.  
   **Isolating Changes:** Forking allows you to experiment with changes before submitting them via a pull request.  
   **Backup and Experimentation:** Developers can use forks to store modified versions of projects for testing purposes.  
   **Collaboration:** Team members can work on different features without interfering with the main repository.

**Steps to Fork a Repository**

1️ **Navigate to the Repository:**

* Go to the GitHub repository you want to fork.

2️ **Click on the Fork Button:**

* Located at the top right corner of the repository page.

3️ **Clone the Forked Repository Locally:**

**Syntax:**

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

**4.Add the Original Repository as an Upstream Remote:**

Syntax:

git remote add upstream <https://github.com/original-owner/repository.git>

🡪 This allows you to sync changes from the original repository.

**5.Fetch Updates from the Original Repository:**

**Syntax:**

git fetch upstream

🡪Keeps your fork updated with the latest changes.

**6.Create a New Branch for Your Changes:**

Syntax:

git checkout -b feature-branch

**7.Make Changes and Push to Your Forked Repository:**

Syntax:

git add .

git commit -m "Added new feature"

git push origin feature-branch

**8.Create a Pull Request:**

* Open a pull request to merge your changes into the original repository.

**Difference Between Linux and Windows**

|  |  |  |
| --- | --- | --- |
| Feature | Linux | Windows |
| Type | Open-source operating system | Proprietary operating system |
| Cost | Free (various distributions) | Paid (license required) |
| Customization | Highly customizable | Limited customization options |
| Security | More secure, fewer viruses | More vulnerable to malware |
| Performance | Generally faster and lightweight | Can be resource-heavy |
| User Interface | Command-line and GUI-based | Mostly GUI-based |
| Software Support | Supports open-source apps | Supports commercial software |
| File System | Ext4, XFS, Btrfs, etc. | NTFS, FAT32, exFAT |
| Usage | Preferred by developers, servers | Common for personal & business use |
| Updates | Frequent updates, user-controlled | Automatic updates from Microsoft |
| Driver Support | Limited official driver support | Wide range of driver compatibility |