Name: Hitain Shroff
Roll No: 2201102
Div: T22

SEPM Experiment 3

Aim: To Perform various GIT operations on local and Remote repositories using GIT Cheat-Sheet.

Theory:

Git Operations on Local and Remote Repositories: A Git Cheat-Sheet

Git is a powerful tool for managing version control, and it offers many commands to perform operations both on local and remote repositories. Here's a cheat-sheet to guide you through common Git operations.

Basic Git Commands for Local Repository

1. Initializing a Repository

- **Command**:

bash

git init

- **Purpose**: Initializes a new Git repository in the current directory. This creates a .git directory, which contains all the version control information.

2. Cloning a Repository

- **Command**:

bash

git clone <repository_url>

- **Purpose**: Creates a local copy of a remote repository. It's typically used to copy a project from GitHub or another remote server to your local machine.

3. Checking the Status of the Repository

- **Command**:

bash

git status

- **Purpose**: Displays the current status of the working directory and staging area. It tells you which files are modified, staged for commit, or untracked.

4. Adding Files to Staging Area

- **Command**:

bash

git add <file_name>

- **Purpose**: Adds a file to the staging area, preparing it for commit. You can also add all files by using git add . or git add -A.

5. Committing Changes

- **Command**:

bash

git commit -m "Your commit message"

- **Purpose**: Commits the staged changes to the local repository with a message explaining what was changed.

6. Viewing Commit History

- **Command**:

bash

git log

- **Purpose**: Shows the commit history in reverse chronological order. You can use git log --oneline to see a simplified version with commit hashes.

7. Checking the Differences Between Changes

- **Command**:

bash

git diff

- **Purpose**: Shows the differences between the working directory and the staging area. You can use git diff <file_name> to compare specific files.

8. Creating a New Branch

- **Command**:

bash

git branch <branch_name>

- **Purpose**: Creates a new branch. This branch is a separate line of development from the main branch.

9. Switching Between Branches

- **Command**:

bash

git checkout <branch_name>

- **Purpose**: Switches to the specified branch. You can also use git switch <branch_name> in newer Git versions.

10. Merging Branches

- **Command**:

bash

git merge <branch_name>

- **Purpose**: Merges the specified branch into the current branch.

11. Deleting a Branch

- **Command**:

bash

git branch -d <branch_name>

- **Purpose**: Deletes the specified branch locally. If the branch has unmerged changes, use git branch -D <branch_name> to force delete it.

Basic Git Commands for Remote Repository

1. Adding a Remote Repository

- **Command**:

bash

git remote add origin <remote_repository_URL>

- **Purpose**: Adds a remote repository to your local project, allowing you to push and pull changes from/to it.

2. Viewing Remote Repositories

- **Command**:

bash

git remote -v

- **Purpose**: Lists the remote repositories associated with your local repository, showing the URLs for fetch and push operations.

3. Fetching Changes from Remote

- **Command**:

bash

git fetch origin

- **Purpose**: Fetches changes from the remote repository but doesn't automatically merge them into your local repository. It updates your remote-tracking branches.

4. Pulling Changes from Remote

- **Command**:

bash

git pull origin <branch_name>

- **Purpose**: Fetches changes from the remote repository and merges them into your current local branch. This is often used to keep your local branch up to date with the remote branch.

5. Pushing Changes to Remote

- **Command**:

bash

git push origin <branch_name>

- **Purpose**: Pushes your local commits to the remote repository on the specified branch.

6. Creating a Remote Branch

- **Command**:

bash

git push origin <branch_name>

- **Purpose**: Creates a new branch on the remote repository and pushes the local branch to it.

7. Deleting a Remote Branch

- **Command**:

bash

git push origin --delete <branch_name>

- **Purpose**: Deletes the specified branch from the remote repository.

8. Renaming a Remote Branch

- **Command**:

Bash

git push origin :<old_branch_name>

git push origin <new_branch_name>

- **Purpose**: Deletes the old branch from the remote and pushes the newly renamed branch.

Advanced Git Operations

1. Rebasing (Rewriting Commit History)

- **Command**:

bash

git rebase <branch_name>

- **Purpose**: Re-applies commits from the current branch onto another branch, creating a cleaner project history.

2. Stashing Changes

- **Command**:

bash

git stash

- **Purpose**: Temporarily saves uncommitted changes to a "stash" so you can switch branches without committing them. You can apply the stashed changes later using git stash apply.

3. Resetting Changes

- **Command**:

bash

git reset <file_name>

- **Purpose**: Unstages a file from the staging area, while keeping the changes in the working directory. You can also use git reset --hard to reset the working directory to the last commit.

4. Tagging a Commit

- **Command**:

bash

git tag <tag_name>

- **Purpose**: Marks a specific commit with a tag, typically used for marking release points like v1.0.0.

5. Cherry-Picking Commits

- **Command**:

bash

git cherry-pick <commit_hash>

- **Purpose**: Applies the changes from a specific commit (identified by its hash) to your current branch.

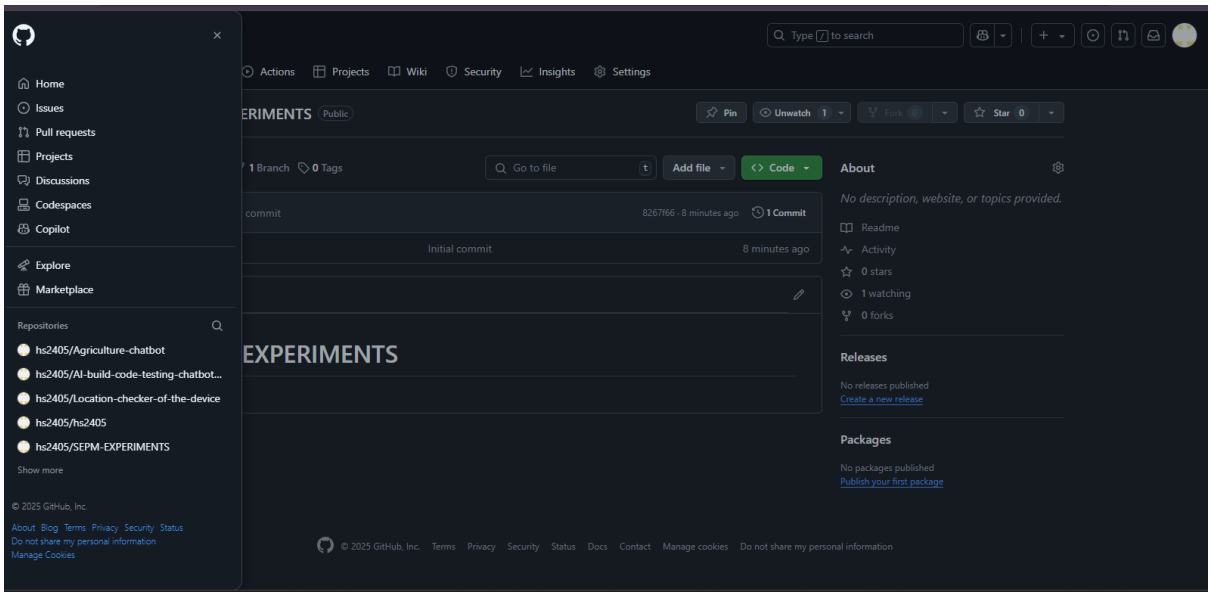
Git Workflow Summary

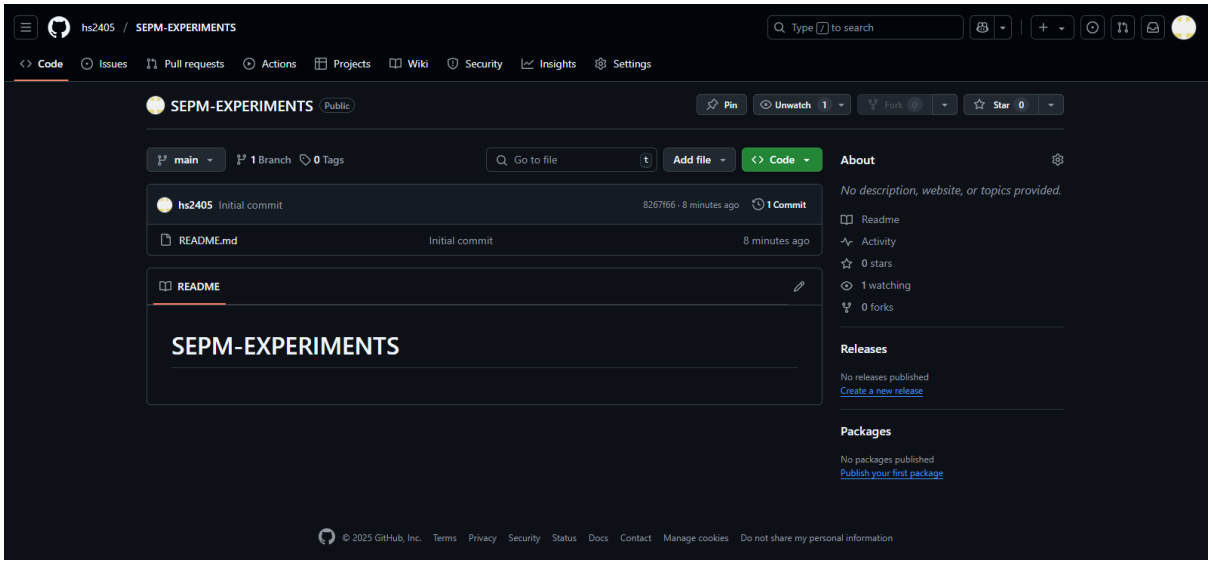
- **Starting a Project**: git init
- **Cloning an Existing Project**: git clone <repo_url>
- **Tracking Changes**: git status, git add ., git commit -m "message"
- **Branching**: git branch, git checkout <branch_name>, git merge <branch_name>
- **Collaborating**: git push, git pull, git fetch

Name: Hitain Shroff
Roll No: 2201102
Div: T22

- **Working Remotely**: git remote add origin <url>, git push origin <branch>

Output screenshots





Name: Hitain Shroff
Roll No: 2201102
Div: T22

Conclusion:

This Git cheat-sheet outlines key operations and workflows for managing code with Git, both locally and remotely. Mastering these commands will allow you to work effectively in team-based and individual development environments, whether you're managing a personal project or collaborating on a team. By using Git, you can track changes, collaborate seamlessly with others, and maintain a history of your project's evolution.