EX. NO: 6	IMPLEMENTATION OF CORE REVIEW IN GIT

Introduction

Git is a powerful tool for version control, but it’s also a crucial tool for collaborating with remote teams. When working remotely, effective communication and collaboration are key to ensure that your team is working together towards a common goal.

Use a shared repository

Using a shared repository is the foundation of effective collaboration with Git. By hosting your repository on a remote server, you can give your team members access to the latest version of your code, regardless of their location.

Create a new repository: **git init**

Clone a repository: **git clone [respository URL]**

Add a remote repository: **git remote add [name] [repository URL]**

Fetch changes from a remote repository: **git fetch**

Pull changes from a remote repository**: git pull**

Push changes to a remote repository: **git push**

Use branching and merging

Branching and merging are powerful features of Git that allow you to work on multiple versions of your code simultaneously, without interfering with each other’s work. By using branching and merging effectively, your team can work on different features or bug fixes in parallel, without causing conflicts.

Create a new branch: **git branch [name]**

Switch to a branch: **git checkout [name]**

Merge a branch: **git merge [name]**

Resolve merge conflicts: **git mergetool**

Delete a branch: **git branch -d [name]**

Use Pull requests

Pull requests are a great way to review and merge changes from different team members before they are merged into the main codebase. By using pull requests, you can ensure that your code is reviewed and tested before it is merged into the main branch, which can help prevent bugs and other issues.

Create a new pull request: **git request-pull [branch] [repository URL]**

Review and merge a pull request**: git pull-request**

Close a pull request: **git request-pull -C [branch] [repository URL]**

Use Issue Tracking

Issue tracking is a great way to keep track of bugs, feature requests, and other issues that need to be addressed in your code. By using an issue tracking system like GitHub Issues, you can assign tasks to team members, track progress, and ensure that all issues are addressed in a timely manner.

Create a new issue: **git commit -m "[issue number] [commit message]"**

Assign an issue to a team member: **git assign [username]**

Clone an issue: **git clone [issue number]**

Reopen an issue: **git reopen [issue number]**

Communicate Effectively

Effective communication is key to collaboration, especially in a remote team setting. Use tools like Slack or Microsoft Teams to stay in touch with your team members, and use video calls or screen sharing to discuss code changes or work on problems together.

Start a video call: **git video-call [username]**

Share your screen**: git screen-share**

Send a message: **git message [username] [message]**

Use Git hooks

Git hooks are scripts that Git runs automatically at certain points in the Git workflow. You can use Git hooks to automate repetitive tasks, enforce coding standards, or perform other tasks that are important to your team's workflow.

Install a git hook: **git init [hook name]**

Write a git hook script: **nano.git/hooks/[hook name]**

Make the Git hook script executable: **chmod +x .git/hooks/[hook name]**

Use Git submodules

Git submodules are repositories that are embedded inside other repositories. You can use Git submodules to manage dependencies, or to include shared code in multiple projects.

Add a Git submodule: **git submodule add [repository URL]**

Update a Git submodule: **git submodule update**

Remove a Git submodule: **git submodule deinit [submodule path]**

Use Git aliases

Git aliases are shortcuts for commonly used Git commands. You can use Git aliases to save time and improve your productivity when working with Git.

Set up a Git alias: **git config –global alias.[alias name] '[Git command]'**

Use a Git alias: **git [alias name]**