NAME: VOMMIDAPU CHINNI **BATCH NO:** 120

MAIL.ID: vommidapuchinni@gmail.com **DATE:** 11/04/24

TRAINER: Mr. Madhukar **MODULE:** GIT

ASSIGNMENT – 6

1. **Explain the differences between git reset, git rebase and git revert?**

git reset: "git reset" is used to reset the current HEAD to a specified state. It is often used to undo changes in the working directory or staging area. This will discard commits. Changes which commit a branch HEAD is currently pointing at. It alter the existing commist. Can be used to unstage a file. In it we have three types:

1. Soft
2. Mixed
3. Hard

1. git reset --soft: Reset data from local repository.
   Syntax: git reset --soft <commit_id>
2. git reset --mixed: Reset data from local repository and staging area. The mixed option is default if we don't provide git reset with an option it will take mixed option.
   Syntax: git reset --mixed<commit_id>
3. git reset --hard: Reset data from local repository, staging area and working directory also.
   Syntax: git reset --hard<commit_id>

git rebase: git rebase is used to integrate changes from one branch into another by reapplying commits on top of another base commit.

It helps to maintain a linear project history.

Workflow:

1. Choose a base commit.
2. Select a branch to rebase onto the base commit.
3. Git will rewind the branch to the common ancestor commit of the branch and the base commit.
4. Git then applies each commit from the branch one by one on top of the base commit.

It is used when It's often used to keep feature branches up-to-date with the main branch or to rewrite commit history before merging a feature branch.

Syntax: git rebase or git rebase <branchname>

git revert: It is used when we want to se the changes in the remote repository. It is used to create a new commit that undoes the changes made by a specific commit or a range of commits. It's a safe way to undo changes as it doesn't alter the commit history.

Workflow:

1. Specify the commit to be reverted.
2. Git creates a new commit that applies the inverse of the changes introduced by the specified commit.

It's useful when you want to undo changes introduced by a commit without altering the project's commit history.

Syntax: git revert HEAD <commit_id>

2. Explain the Branching Strategy?

Branching strategy refers to a set of rules and practices that dictate how branches are created, managed, and merged within a version control system like Git. A well-defined branching strategy helps streamline development workflows, facilitates collaboration among team members, and ensures a stable and organized codebase**.**

When master branch is from error free, then you want to add some more code to that file in this case we are using branches. When a new branch is created and add code to it when that branch is form error free. Then merge with default master branch.

Properties of Branches:

- Each task has one separate branch.
- After done with coding, merge other branches with master.
- This concept is useful for parallel development.
- You can create any number of branches.
- Changes are personal to that particular branch.
- Default branch is ‘master’.
- File created in workspace will be visible in any of the branch workspace until you commit once you commit, then that file belongs to that particular branch.

Syntaxs:

To create a branch: git branch <branch name>

To create a branch and switch to that branch: git checkout -b <branch name>

To switch a branch: git checkout <branch name>

To delete a branch: git branch -d <branch name>

git branch -D <branch name> (to delete forcefully)

To list branches: git branch

To merge branches: git merge <branch name>