Name : K.Taraka Lakshmi Prasanna

Email : 22bq1a4268@vvit.net

Superset ID : 6314755

Week : 8

Topic : Git

Git Basics

Git is a **distributed version control system** used to track changes in code, collaborate with teams, and manage project history.
Below are some essential Git commands and their purposes:

Command	**Description**	**Example**
git init	Initializes a new local Git repository in the current directory.	git init
git status	Shows the current state of the working directory and staging area.	git status
git add	Adds files to the staging area for the next commit.	git add file.txt or git add .
git commit	Saves changes from the staging area to the local repository with a message.	git commit -m "Added new feature"
git push	Uploads commits from the local repository to a remote repository.	git push origin main
git pull	Fetches and merges changes from the remote repository into the current branch.	git pull origin main

What is .gitignore?

.gitignore is a special text file in a Git repository that tells Git **which files or folders to ignore**.
Files listed in .gitignore will:

- Not be tracked by Git.
- Not appear as *untracked* in git status.
- Not be pushed to the remote repository.

This is useful for **temporary, sensitive, or system-generated files** like:

- Log files (.log)
- Temporary build files (/build/)

- IDE settings (.vscode/, .idea/)
- Secrets (config.env)

```
touch .gitignore
# Ignore all .log files
*.log
# Ignore a specific file
secret.txt
# Ignore a folder
node_modules/
# Ignore all .tmp files in any folder
*.tmp
# Ignore all files in "log" folder
log/
git add .gitignore
git commit -m "Add .gitignore to ignore unwanted files"
```

Branching

- **What it is:**
  A branch in Git is a separate line of development. It allows you to work on new features or bug fixes **without affecting the main codebase**.
  The default branch is usually main or master.
- **Why it's useful:**
  - Isolates changes until they're ready.
  - Allows multiple developers to work in parallel.
  - Keeps the main branch stable.
- **Creating and Switching Branches:**

```
# Create a new branch
git branch feature-branch
# Switch to it
git checkout feature-branch
# Or create and switch in one step
git checkout -b feature-branch
```

Merging

- **What it is:**
  Merging is the process of combining changes from one branch into another (usually from a feature branch into main).
- **How to Merge Locally:**

```
# Switch to the branch you want to merge into (main)

git checkout main

# Merge the feature branch into main

git merge feature-branch
```

Creating a Branch in GitLab

1. Go to your GitLab project.
2. Navigate to **Repository → Branches**.
3. Click **New branch**.
4. Enter the branch name (e.g., feature-login) and select the source branch (e.g., main).
5. Click **Create branch**.
6. You can now clone or checkout this branch locally to start making changes.

Creating a Merge Request in GitLab

A **Merge Request (MR)** in GitLab is a request to merge changes from one branch into another, typically after code review.

Steps:

1. Push your branch with changes to GitLab:

bash

CopyEdit

```
git push origin feature-login
```

2. In GitLab, go to your project.
3. GitLab usually shows a banner suggesting to create a merge request. Click it.
   Or navigate to **Merge Requests → New Merge Request**.
4. Select:
   - **Source branch**: Your feature branch (feature-login)
   - **Target branch**: The branch you want to merge into (main or master)
5. Add:
   - Title (short description of the changes)
   - Description (detailed info, related issues, testing steps)
   - Reviewers (optional, for code review)

6. Click **Create Merge Request**.
7. Once approved and CI/CD checks pass, click **Merge**.

What is a Merge Conflict?

A **merge conflict** occurs when Git cannot automatically combine changes from two branches because both modified the same part of a file, or one branch deleted a file the other modified.
This often happens when **multiple users update the same code** in the trunk (main/master) and in their feature branch.

How to Resolve a Merge Conflict

1. Identify the Conflict

When you try to merge:

```
git checkout master

git merge feature-branch
```

```
Auto-merging file.txt

CONFLICT (content): Merge conflict in file.txt

Automatic merge failed; fix conflicts and then commit the result.
```

2. Open the Conflicted File

```
<<<<<<< HEAD

Content from current branch (master)

=======

Content from feature branch

>>>>>>> feature-branch
```

3. Decide How to Merge

Options:

- Keep **only your changes**.
- Keep **only the incoming changes**.
- Combine both changes logically.

Edit the file to remove the conflict markers (<<<<<<<, =======, >>>>>>>) and keep the correct code.

4. Mark the Conflict as Resolved

After editing:

```
git add file.txt
```

5. Commit the Merge

```
git commit -m "Resolve merge conflict between master and feature-branch"
```

6. Push the Resolved Changes

```
git push origin master
```

Conflict Resolution with a Merge Tool

You can use visual tools like **P4Merge**, **Meld**, or VS Code's merge editor:

```
git mergetool
```

The tool will show **side-by-side differences** and allow you to choose or combine changes interactively.

Explain how to clean up and push back to remote Git.

In Git, "clean up" can refer to:

- **Removing untracked files** (files not in Git yet)
- **Discarding local changes** (resetting to last commit)
- **Cleaning old/merged branches**
- **Tidying commit history** before pushing

```
git checkout main

git status

git clean -fd

git reset --hard HEAD

git pull origin main

git push origin main
```