1.1 Why Git?

What is Version Control?

- A system that tracks changes to files over time, enabling:
  - **Collaboration**: Multiple people can work on the same files without overwriting each other's changes.
  - **History Tracking**: Allows you to revert to previous versions of a file.

Why Git is Better Than Other VCS?

- **Distributed System**: Every developer has a full copy of the repository, enabling offline work.
- **Speed**: Git operations like commits, branching, and merging are very fast due to local operations.
- **Open Source and Widely Supported**: Works with many platforms, tools, and hosting services (GitHub, GitLab, Bitbucket).

Configuration:-

Run these commands after installation:

Set your name:
git config --global user.name "Vikash Kumar"

Set your email:

Git config --global user.email "vikashkr@gmail.com"

View Configuration:

git config --list

1.3.1 Initializing a Repository

git init

1.3.2 Cloning a Repository

git clone <repository-url>

1.3.3 Checking Repository Status

git status

1.3.4 Adding Files

git add <file name> or [git add .] → It will add all files

1.3.5 Committing Changes

git commit -m "Your commit message"

1.3.6 Viewing Commit History

git log or [git log --oneline]

**********Chapter 2**********

2.1 What Are Branches in Git?

- A **branch** in Git is like a parallel line of development.
- It allows you to work on a feature, bug fix, or experiment without affecting the main codebase (usually called `main` or `master`).

Why Use Branches?

- **Isolation**: Keep work on features/bugs separate from the main branch.
- **Collaboration**: Multiple developers can work on different branches simultaneously.
- **Version Control**: Helps track and manage different versions of the codebase.

Structure of Branches

- **HEAD**: A pointer to the current branch you're working on.
- **Default Branch**: Typically `main` or `master`

2.2 Basic Branching Commands

2.2.1 Creating a Branch

- git branch <branch name>

2.2.2 Viewing Branches

git branch

To view remote branches:- [git branch –r]

To view both local and remote branches:- [git branch -a]

2.2.3 Switching Branches

git switch <branch_name> or git checkout <branch_name>

2.3.1 Merging a Branch

First, switch to the branch you want to merge into (e.g., main):→ git switch main

Merge the feature branch:→ git merge feature/login

2.3.2 Fast-Forward Merge

- If the main branch hasn’t changed since the feature branch was created, Git simply moves the main branch pointer forward.

2.3.3 Three-Way Merge

- If both branches have changes, Git creates a new commit that combines the histories of the branches.

2.3.4 Resolving Merge Conflicts

- ## What is a Conflict?

  Happens when changes from two branches conflict.

Steps to Resolve:

1. Git will show conflict markers in the file:

   <<<<<<< HEAD

   code from main branch

   =======

   code from feature/ branch name

   >>>>>>> feature/branch

2. Manually edit the file to resolve the conflict.
3. Add the resolved file to the staging area:
   git add <file>
4. Complete the merge:
   git commit

Chapter 2: Working with Branches

2.1 What Are Branches in Git?

- A **branch** in Git is like a parallel line of development.
- It allows you to work on a feature, bug fix, or experiment without affecting the main codebase (usually called main or master).

Why Use Branches?

- **Isolation**: Keep work on features/bugs separate from the main branch.
- **Collaboration**: Multiple developers can work on different branches simultaneously.
- **Version Control**: Helps track and manage different versions of the codebase.

Structure of Branches

- **HEAD**: A pointer to the current branch you're working on.
- **Default Branch**: Typically main or master.

2.2 Basic Branching Commands

2.2.1 Creating a Branch

```
git branch branch_name
```

- Example:→ git branch feature/login
- This creates a new branch feature/login.

2.2.2 Viewing Branches

```
git branch
```

  - Shows all local branches. The current branch is marked with *.
- To view remote branches:

```
git branch -r
```

- To view both local and remote branches:

```
git branch -a
```

2.2.3 Switching Branches

```
git switch branch_name
      Or
git checkout branch_name
Ex:- git switch feature/login
```

- Git moves HEAD to the new branch, allowing you to work on it.

2.3 Merging Branches

What is Merging?

- Merging integrates changes from one branch into another.
- Usually, changes from a feature branch are merged into the main branch.

2.3.1 Merging a Branch

- First, switch to the branch you want to merge into (e.g., main):
  git switch main
- Merge the feature branch:
  git merge feature/login

2.3.2 Fast-Forward Merge

- If the main branch hasn't changed since the feature branch was created, Git simply moves the main branch pointer forward.

2.3.3 Three-Way Merge

- If both branches have changes, Git creates a new commit that combines the histories of the branches.

2.3.4 Resolving Merge Conflicts

- **What is a Conflict?**
  - Happens when changes from two branches conflict.
- **Steps to Resolve**:
  1. Git will show conflict markers in the file:

     <<<<<<< HEAD
     code from main branch
     =======
     code from feature/login branch
     >>>>>>> feature/login

  2. Manually edit the file to resolve the conflict.
  3. Add the resolved file to the staging area:
     git add <file>
  4. Complete the merge:
     git commit

2.4 Deleting Branches

Why Delete a Branch?

- Once a feature is merged, the branch is no longer needed and can be deleted to keep the repository clean.

Delete a Local Branch

git branch –d <branch_name>

- Force delete (if the branch is not fully merged):

  git branch -D <branch_name>

Delete a Remote Branch

git push origin --delete <branch_name>