Git Fundamentals

Outlines

- Introduction to Git
- Core Git Concepts
- Basic Git Workflow
- Working with Remote Repositories
- Branching and Merging
- Undoing Changes
- Pull Requests (PRs) and Code Reviews
- Q&A and Hands-on Practice

Introduction to Git

What is Git? Why use it?

- Git is a **Version Control System (VCS)** that helps track changes in files over time.
- Allows multiple people to collaborate on the same project.
- Prevents losing previous versions of work.
- Helps manage code history and roll back changes if needed.

Introduction to Git

Git vs. GitHub/GitLab/Bitbucket

- **Git** = a **tool** for managing code versioning on your **local** machine.
- **GitHub, GitLab, Bitbucket** = hosting services for Git repositories to collaborate online.

Introduction to Git

Installing and Configuring Git

- Install Git: https://git-scm.com/downloads.
- Check if Git is installed:
  - **git –version**
- Configure user information (one-time setup):
  - **git config --global user.name "Your Name"**
  - **git config --global user.email "your.email@example.com"**

Core Git Concepts

Repositories (Repos)

- A repository is like a **folder** where Git tracks changes.
- There are two types:
  - **Local repository** (on your computer).
  - **Remote repository** (hosted on GitHub/GitLab/etc.)

Core Git Concepts

Working Directory, Staging Area, and Commit History

- **Working Directory** → Where you edit files.
- **Staging Area (Index)** → Where you prepare files before committing.
- **Commits (History)** → A record of changes stored in Git.
- Visual representation:
  - Working Directory → Staging Area → Repository

Basic Git Workflow

Creating a Repository

- Initialize a new Git repository:
  - **git init**

Cloning an Existing Repository

- Download a copy of an existing project:
  - **git clone <repo-url>**

Basic Git Workflow

<u>Checking Repository Status</u>

- See which files are changed, staged, or untracked:
  - **git status**

<u>Adding and Committing Changes</u>

- Add a file to the staging area:
  - **git add filename.txt**
- Add all files:
  - **git add** .
- Commit changes:
  - **git commit -m "Meaningful commit message"**

Basic Git Workflow

Viewing Commit History

- Check past commits
  - **git log**
- Simplified one-line history:
  - **git log –oneline**

Working with Remote Repositories

<u>Connecting to a Remote Repository</u>

- Add a remote repository (e.g., GitHub):
  - **git remote add origin <repo-url>**

<u>Pushing Changes to Remote:</u>

- Send local commits to the remote repo:
  - **git push origin main**

Working with Remote Repositories

Pulling Updates from Remote

- Get the latest changes from the remote:
  - **git pull origin main**

Fetching Without Merging

- Fetch updates without applying them:
  - **git fetch**

Branching and Merging

What is a Branch?

- A branch is like a **copy of your code** where you can make changes without affecting the main project.
- The default branch is usually main or master:

Creating and Switching Branches

- Create a new branch:
  - **git branch feature-branch**
- Switch to the new branch:
  - **git checkout feature-branch** or **git switch feature-branch**.

Branching and Merging

Merging Branches

- Merge changes from a branch into main:
  - **git checkout main**
  - **git merge feature-branch**

Handling Merge Conflicts

- When Git cannot automatically merge, it marks conflicts in files.
- Open the file, resolve conflicts, then commit:
  - **git add** .
  - **git commit -m "Resolved merge conflict"**

Undoing Changes

Unstaging a File

- If you added a file by mistake:
  - **git reset HEAD filename.txt**

Undoing the Last Commit

- Keep changes but remove the commit:
  - **git reset --soft HEAD~1**
- Delete last commit permanently:
  - **git reset --hard HEAD~1**

Undoing Changes

Reverting a Commit (Safer than Reset)

- Undo a commit while keeping history:
  - **git revert <commit-hash>**

Pull Requests (PRs) and Code Reviews

- PRs are used when working with remote repositories (e.g., GitHub).
- Developers review and approve before merging changes.

Ignoring files with .gitignore

- Create a **.gitignore** file to avoid tracking unnecessary files:
- **node_modules/**
- **.env**
- ***.log**

Pull Requests (PRs) and Code Reviews

How Does a Pull Request Work?

1. Developer Creates a Feature Branch

- Before making changes, the developer creates a new branch (e.g., feature-login), **Example**:
- **git checkout -b feature-login**

2. Developer Makes Changes and Pushes to Remote Repository

- After editing files, they add and commit the changes:
    - **git add .**
    - **git commit -m "Added login functionality"**
    - **git push origin feature-login**

Pull Requests (PRs) and Code Reviews

How Does a Pull Request Work?

3. Create a Pull Request (PR)

- On GitHub (or GitLab, Bitbucket), the developer **opens a PR** from the feature-login branch into **main.**
- They provide a **description of the changes** and why they are necessary.

4. Code Review Process

- Other developers **review the code**, leave comments, and suggest improvements.
- The team can **approve or request changes.**

Pull Requests (PRs) and Code Reviews

How Does a Pull Request Work?

5. Merging the PR

- Once approved, the PR can be **merged** into the **main** branch.
- Example command (if merging locally):
  - **git checkout main**
  - **git merge feature-login**

Pull Requests (PRs) and Code Reviews

Step-by-Step Guide to Creating a Pull Request

1. Fork the Repository (if necessary):

- If you don't have write access to the repository, you'll need to fork it first. Go to the repository page on GitHub and click the "Fork" button in the top-right corner.

2.Clone the repository:

- Clone the repository to your local machine. If you forked the repository, clone your fork.
  - **git clone https://github.com/your-username/repository-name.git**

Pull Requests (PRs) and Code Reviews

Step-by-Step Guide to Creating a Pull Request

3. Create a New Branch:

- Navigate to the repository directory and create a new branch for your changes.
  - **cd repository-name**
  - **git checkout -b feature-branch**

4. Make Your Changes:

- Make the necessary changes to the files in your local repository.

Pull Requests (PRs) and Code Reviews

Step-by-Step Guide to Creating a Pull Request

5. Commit Your Changes:
   - Stage and commit your changes.
     - **git add** .
     - **git commit -m "Description of the changes made"**

6. Push Your Branch to GitHub:
   - Push your branch to your GitHub repository.
     - **git push origin feature-branch**

Pull Requests (PRs) and Code Reviews

Step-by-Step Guide to Creating a Pull Request

7. Create the Pull Request:

- Go to the original repository on GitHub.
- You should see a prompt to create a pull request for your recently pushed branch.
- Click on "Compare & pull request."
- Fill in the details for your pull request, including a title and description.
- Click "Create pull request."

Q&A and Hands-on Practice

Exercise: Create a repository, add a file, make changes, commit, and push it.

Thank you

Shurok KHOZAM

shurok.khozam@telecom-sudparis.eu