The Git Course - Web Development Training

The Utility of Git

1. **Collaboration**: Git enhances teamwork by enabling several individuals to work simultaneously on the same project. Each developer can create their own branch and operate independently.
2. **Version Control**: Git records a history of all modifications made to a project. This feature allows you to monitor the changes made, the individuals behind them, and the reasons for these changes.
3. **Backup and Restore**: As all previous file versions are preserved, it becomes straightforward to restore earlier versions if a problem arises.
4. **Branching and Merging**: The capacity to create branches and merge them makes it convenient to experiment with new ideas and incorporate them into the main project if they prove beneficial.

Detailed Explanation of the Git Workflow

- In Git, your project's workflow revolves around three main areas: The working directory, the staging area, and the Git directory (repository).
    1. **Working Directory**: This is where you'll be making changes to your files. At this stage, the files can either be untracked (new files) or tracked (files that have been previously staged or committed).
    2. **Staging Area**: After making changes, you can add these changes to the staging area using `git add <filename>` or `git add .` for all changes. The staging area is a snapshot of your working directory, representing the changes you wish to commit to your project history.
    3. **Git Directory (Repository)**: This is where Git stores the metadata and object database for your project. It's the most integral part of Git and is what is copied when you clone a repository from another computer.

Setup Git (Installation)

- Ensure Git is installed on your computer. If not, download and install it from **here**.
- Open your terminal and type the following command to verify the installation: `git --version`.

Configure Your Name & Email

- Set your name and email in git configuration by running the following commands in your terminal:

```
bashCopy code

git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

- Replace `"Your Name"` and `"your@email.com"` with your actual name and email.

Initialize a Repository

- Navigate to your project folder using the terminal.
- Run the command `git init` to create a new repository.

Cloning an Existing Repository

- In Git, you can also work with an already existing repository by cloning it. This process copies all the data from the source repository, allowing you to work with it locally.
- To clone a repository, use the following command: `git clone <repo-url>`. Replace `<repo-url>` with the URL of the repository you wish to clone.
- After running the command, a new directory will be created with the repository's contents.

Set up GitLab

- Sign up/in to GitLab and create a blank project.

Throughout the course, we will be using GitLab as a remotely hosted platform to push our work and collaborate using Git.

- If you do not already have one, start by creating a GitLab account: **https://gitlab.com/users/sign_in**
- GitLab might guide you through the creation of your very first project. Go to your **dashboard** by clicking on the top left GitLab logo:
- Then click on “New Project”:
- Select the first option “Create blank project”
- Type “Git Course” as a project name and select “Public” as a visibility level. This way, other people will be able to clone and download the repository but won’t be able to push on it:
- Click on “Create project”

Working with Remote Repositories

- In addition to local repositories, Git allows you to work with remote repositories. These are versions of your project that are hosted on the internet or network somewhere.
- You can connect your local repository to a remote server by adding a "remote". Use this command to add a remote: `git remote add origin <repo-url>`. Replace `<repo-url>` with the URL of your remote repository.
- After adding a remote, you can push your changes to it using `git push -u origin master`. This command pushes changes from your local "master" branch to the "master" branch on the remote repository.

Staging and Committing Code

- Changes in files are first added to the staging area using the `git add <filename>` command. You can add all changes using `git add .`.
- After staging the changes, you can commit them to the history of your project using the `git commit -m "Commit message"` command. The commit message should be a brief

description of the changes made.

Branches

- In Git, branches are effectively pointers to a specific commit. They are useful for developing new features or testing out ideas, without affecting the main project (usually the "master" branch).
- You can create a new branch using the `git branch <branch-name>` command, and switch to it using `git checkout <branch-name>`.
- You can do both at once using the `git checkout -b <branch-name>` command.

Resolving Merge Conflicts

- Sometimes, when you try to merge branches, Git may not be able to resolve differences between the branches. This results in a conflict.
- You will have to manually resolve these conflicts. Open the file with conflicts, and you will see the differences demarcated clearly. Edit the file until it is as desired, then stage and commit it.

▼ For further learning, refer to:

- Git official documentation: **https://git-scm.com/doc**
- About GitHub: **https://guides.github.com/**

More?

Index, cheatsheet, best practices, fundamental (1)

Exercices

Exercise 1: Building a Personal Website with Git and GitLab (1)

Exercise 2: Creating a Branch and Changing Styles (1)

Peer-Exercice : Exercise 3: Collaborative Website Development with Git and GitLab (1)

Peer-Exercice : Exercise 4: Collaborative Styling Update with Branches (1)

BONUS EXERCICES

BONUS EXERCICES (1)