GIT PRACTICE LAB JOURNEY

LAB OVERVIEW

Welcome students to the Git Practice Lab! In this engaging lab, you'll learn how to use Git, a powerful tool for tracking changes in your code. Git is essential for developers, as it allows you to manage your project's history efficiently, collaborate with others, and revert to previous versions of your work if necessary.

We'll start by setting up Git on your computer. This includes downloading and installing the required software, configuring your Git environment, and verifying that everything is working correctly.

Once Git is up and running, you'll create your first repository. A repository, often referred to as a "repo," is where all your project files and their revision history are stored. You'll learn how to initialize a new repository and understand the structure of files within it.

After setting up your repository, the next step will be making your first commit. A commit in Git is like a snapshot of your project at a specific point in time. You will learn how to stage changes, commit them with meaningful messages, and understand the importance of each commit in maintaining a clear project history.

As we go along, remember that I'm here to help. Don't hesitate to ask questions whenever you're unsure or when you encounter challenges. Git can seem overwhelming at first, but with practice and guidance, you'll become proficient in no time. Let's dive into version control with Git and unlock the full potential of your coding projects!

SECTION 1: SETTING UP GIT AND CREATING A REPOSITORY

In this section, we'll get Git up and running on your machine. First, let's check if Git is already installed. Open your terminal and type the command `git --version`. If you see a version number displayed, congratulations! Git is installed and ready for use. If you encounter any issues or the command

returns an error, you'll need to follow the installation steps relevant to your operating system.

INSTALLING GIT

For Windows users, you can download the Git installer from the official Git website. Make sure to follow the prompts during installation, setting the options that best suit your development needs. Mac users can install Git via Homebrew by running `brew install git` in the terminal. Linux users can use their package manager (e.g., `sudo apt-get install git` for Ubuntu) to get Git up and running.

CONFIGURING GIT

Once Git is installed, it's important to configure it with your personal information. This step ensures that your commits are tagged with your name and email address. Use the following commands in your terminal to set up your configuration:

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

CREATING A NEW PROJECT FOLDER

Now that Git is installed and configured, let's create your first project folder. Choose a location on your computer where you'd like to create this folder, and use the terminal to navigate there. Create a new directory with the command:

```
mkdir my_first_repo
cd my_first_repo
```

INITIALIZING A GIT REPOSITORY

Inside your new project folder, initialize a Git repository by executing:

```
git init
```

This command sets up a new Git repository, making it ready to track your files. To verify that everything is set up correctly, you can check the status of your repository with:

```
git status
```

Make sure to take a screenshot of this output; it will serve as proof of your successful setup! 📸

SECTION 2: STAGING AND COMMITTING CHANGES

Now that your Git repository is initialized, it's time to stage files and make your first commit. Start by creating a `README.md` file, which is a fundamental component of any Git repository. This file typically contains information about your project, including its purpose, features, and how to use it. You can create the file using the command:

```
echo "# My First Repository" > README.md
```

Once the `README.md` file is created, you'll want to stage it for commit. Staging is the process of preparing your changes to be committed. To stage your file, run:

```
git add README.md
```

After staging the file, you can confirm its status by running:

```
git status
```

This command will show you which files are staged for commit, ensuring that your changes are ready to go.

Next, it's time to commit your staged changes. A commit in Git is like taking a snapshot of your project at a specific point in time, so it's essential to include a clear and descriptive message. You can create your first commit with the following command:

```
git commit -m "Add README.md file"
```

This command records the staged changes and associates them with your message.

To further enhance your project, you should create a `.gitignore` file. This file specifies intentionally untracked files that Git should ignore, preventing them from being included in commits. To create a `.gitignore` file, you can use:

```
touch .gitignore
```

Open the file in a text editor and list any file types or directories you want Git to exclude, such as:

```
*.pyc
__pycache__/
```

Finally, enhance your project by adding a Python script. Create a simple script, stage it, and commit all new changes just like you did with the `README.md`. For example, if you create a file named `script.py`, stage it using `git add script.py`, and commit it with a message like `Add initial Python script`.

As a part of your requirements, remember to capture a screenshot of your commit history using the command:

```
git log --oneline
```

This will help you document your progress in the lab!

SECTION 3: INSPECTING HISTORY AND MANAGING CHANGES

Now that you have made your initial commits, it's time to inspect the history of your work and learn how to manage changes effectively. Understanding

your commit history is crucial for navigating your project and ensuring that you can track the evolution of your code.

To begin, utilize the `git log` command in your terminal. This command will display a detailed list of all the commits made in your repository, including the commit hash, author, date, and the commit message. For a more concise view, you can use `git log --oneline`, which summarizes each commit in a single line, making it easier to read and analyze.

As you inspect your commit history, you may want to make changes to your project. Open your `script.py` file and make some edits. After making these changes, you can check the differences between your last commit and the current state of your working directory by using the command:

```
git diff
```

This command will show you the lines that have been added or removed, allowing you to review your modifications before proceeding.

If you realize you need to amend something from your last commit, Git provides a powerful feature to do so. After staging any new changes with `git add`, you can use the command:

```
git commit --amend
```

This command allows you to modify the previous commit, either by changing the commit message or adding new changes. It’s a handy way to keep your commit history clean and relevant.

Finally, as a part of your lab experience, don’t forget to capture a screenshot of your amended commit history using `git log --oneline`. This will serve as a visual representation of the changes you've made and the evolution of your project.

SECTION 4: BRANCHING AND MERGING

In this section, we'll explore the fundamental concept of branching in Git, which allows you to work on different features or tasks simultaneously without affecting the main codebase. Branching is a powerful tool that enables developers to experiment and develop new ideas in isolation.

To create a new branch, you can use the command:

```
git branch feature-branch
```

This command creates a new branch named `feature-branch`. However, simply creating a branch is not enough; you need to switch to it to start working on your changes. You can do this with the command:

```
git checkout feature-branch
```

Now that you are on the `feature-branch`, go ahead and make some changes to your project files. For example, you might modify `script.py` to add new functionality or update the `README.md` to reflect recent changes. Once you've made your desired changes, stage and commit them as you have learned in previous sections:

```
git add .
git commit -m "Implement new feature in feature-branch"
```

After committing your changes, it's time to return to the main branch. You can switch back to the main branch using the command:

```
git checkout main
```

Now, you can merge the changes from your `feature-branch` into the main branch. The command for merging is:

```
git merge feature-branch
```

This command incorporates the changes from `feature-branch` into the main branch, allowing you to integrate the new feature into your primary codebase.

Once you're satisfied with the merge, it's good practice to clean up by deleting the feature branch. You can do this using:

```
git branch -d feature-branch
```

This command deletes the `feature-branch`, keeping your branch list organized and manageable. Through this hands-on experience, you will solidify your understanding of managing features and tasks in Git, making your workflow more efficient and effective.

FINAL SUBMISSION INSTRUCTIONS

To wrap up the lab, you must submit your work professionally. Begin by creating a folder named `Git-LabSubmission`. This folder will serve as a container for all the necessary files that demonstrate your progress and accomplishments throughout the lab. Within this folder, include the following required screenshots: `git-status.png` (captured from Section 1), `git-log.png` (from Section 2), and `git-log-amended.png` (from Section 3).

Ensure that each screenshot is clear and properly labeled, as they will provide visual evidence of your understanding and execution of Git commands. Once you have collected all required files, the next step is to compress the folder into a .zip file. Name this file `Git-Lab-Submission.zip`.

To create the .zip file, you can right-click on the `Git-LabSubmission` folder and select the option to compress or zip the folder, depending on your operating system. This will create a single, manageable file that contains all your work.

Finally, follow the submission instructions provided in your course materials to submit this zip file. Be sure to double-check that all required files are included before submitting. Congratulations on completing your Git practice lab! 🎉 You've worked hard to learn the essentials of version control, and this final submission is a testament to your dedication and progress.