Git Basics Lab Exercise

Introduction and Objectives

This lab exercise is designed to help you understand and practice the fundamental Git commands. The goal is to provide a hands-on experience with Git by working through real-world scenarios. By the end of this lab, you will be able to create a Git repository, track changes, and understand the workflow of committing changes in Git.

Prerequisites

1. Git installed on your system (Windows, macOS, or Linux).
2. Basic knowledge of the command line.

Real-World Use Case Scenario

Imagine you're a software developer working on a project called **'AwesomeApp**.' You'll use Git to manage the project's source code, track changes, and collaborate with your team.

Exercise 1: Creating a Repository

Objective:

Create a new local Git repository for your project.

Instructions:

1. Create a Project Directory:

   a. Open your terminal or command prompt.

   b. Navigate to your workspace or preferred directory.

   c. Create a new directory for your project:

```
mkdir AwesomeApp

cd AwesomeApp
```

2. Initialize a Git Repository:

```
git init
```

3. Check the Repository Status:

```
git status
```

Explanation:

The 'git init' command creates a new Git repository. It adds a .git directory in your project, where all the version control information is stored.

The 'git status' command shows the current status of the working directory and staging area. It indicates which files are untracked, modified, or staged for the next commit.

Exercise 2: Adding Files to the Repository

Objective:

Add files to the Git repository and track them.

Instructions:

1. Create a New File:

```
echo "<!DOCTYPE
html><html><head><title>AwesomeApp</title></head><body><h1>Welcome
to AwesomeApp</h1></body></html>" > index.html
```

2. Check the Status:

```
git status
```

3. Add the File to the Staging Area:

```
git add index.html
```

4. Verify the Staging Status:

```
git status
```

Explanation:

This command creates an 'index.html' file with basic HTML content.

Git will show that the 'index.html' file is untracked.

The 'git add' command adds changes in the working directory to the staging area. This means that Git is now tracking the file.

The file 'index.html' will now appear in the 'Changes to be committed' section.

Exercise 3: Committing Changes

Objective:

Commit your staged changes to the repository.

Instructions:

1. Commit the Changes:

```
git commit -m "Add initial index.html with welcome message"
```

2. Check the Commit Log:

```
git log
```

Explanation:

The 'git commit' command saves your changes to the local repository. The '-m' flag allows you to add a commit message, which describes the changes made.

The 'git log' command shows the history of commits in the repository. Each commit has a unique identifier (hash), the author, date, and commit message.

Exercise 4: Modifying Files and Viewing Changes

Objective:

Modify an existing file and view the changes.

Instructions:

1. Modify the 'index.html' File:

```
echo "<p>This is a new paragraph.</p>" >> index.html
```

2. Check the Status:

```
git status
```

3. View the Differences:

```
git diff index.html
```

Explanation:

Git will show that 'index.html' has been modified.

The 'git diff' command shows the changes between the working directory and the staging area. It highlights the differences line by line.

Exercise 5: Staging and Committing Changes

Objective:

Stage and commit the modifications.

Instructions:

1. Stage the Changes:

```
git add index.html
```

2. Commit the Changes:

```
git commit -m "Update index.html with a new paragraph"
```

3. Verify the Commit:

```
git log
```

Exercise 6: Viewing the Commit History

Objective:

Explore the commit history to understand how your project evolved.

Instructions:

1. View the Commit Log:

```
git log
```

Explanation:

The 'git log' command provides detailed information about each commit, including the commit hash, author, date, and commit message. This is useful for tracking the progress and understanding the changes made over time.

Exercise 7: Creating a .gitignore File

Objective:

Ignore specific files or directories from being tracked by Git.

Instructions:

1. Create a '.gitignore' File:

```
echo "node_modules/" > .gitignore

echo "*.log" >> .gitignore
```

2. Add and Commit the '.gitignore' File:

```
git add .gitignore
```

git commit -m "Add .gitignore to ignore node_modules and log files"

3. Verify the '.gitignore':

```
echo "This is a log file." > error.log

git status
```

Explanation:

The '.gitignore' file specifies files or directories that Git should ignore. In this example, we're ignoring the 'node_modules' directory (commonly used in JavaScript projects) and any '.log' files.

The 'error.log' file should not appear in the 'git status' output because it's ignored by '.gitignore'.

Exercise 8: Conclusion and Cleanup

Objective:

Clean up the workspace and review what you've learned.

Instructions:

1. Review the Files:

```
ls -a
```

2. Cleanup (Optional):

If you want to start fresh, you can delete the '.git' directory to remove version control:

```
rm -rf .git
```

3. Review What You've Learned:

You've now created a Git repository, tracked changes, committed those changes, modified files, and used '.gitignore'. These are the basic building blocks of working with Git in any real-world project.