Lab Experiment - VS Code and Version Control

This lab manual will guide you through the essential tools for Linux development: VS Code Editor and git.

VS Code

VS Code is a popular, versatile code editor with excellent support for various programming languages and tools.

Learning Resources:

* VS Code Introduction Videos: [[click here](https://code.visualstudio.com/docs/introvideos/basics" \t "_blank)](https://code.visualstudio.com/docs/introvideos/basics)

Exercises:

* **Setting Up VS Code:**
  + Install VS Code for your chosen platform (Linux/WSL) and explore the interface.
* **Shell Integration:**
  + Learn to run shell commands directly from the VS Code terminal.
* **Extensions:**
  + Install extensions for specific languages like Python or Git to enhance functionality.

**Tip:** During the Git section, you'll learn how to integrate Git with VS Code for a seamless workflow.

# Git

Git is a version control system that allows you to track changes in your code, collaborate with others, and revert to previous versions if needed.

## Learning Resources:

* Microsoft Learning Path: [click here](https://learn.microsoft.com/en-us/training/modules/intro-to-git/)
* MIT OpenCourseware: [click here](https://missing.csail.mit.edu/2020/version-control/)
* Setting Up SSH Keys for Secure Communication: [click here](https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account)
* Using Git with VS Code: [click here](https://code.visualstudio.com/docs/sourcecontrol/intro-to-git)

## Exercises:

* **Git Basics:**
* Learn core Git commands like init, add, commit, push
* **Branching and Merging:**

1. **Create a Feature Branch:**
   * Create a new branch for a specific feature, make modifications, commit those changes, and then merge the branch back into the main branch.
2. **Resolving Merge Conflicts:**
   * Simulate a merge conflict scenario where changes were made to the same line of code in different branches. Guide them through manually resolving the conflict and committing a successful merge.
3. **Remote Repositories:**
   * **Clone a Public Repository:**
     + Have trainees clone an existing public repository from a platform like GitHub. This helps them understand how to work with remote codebases.
   * **Contributing to a Public Repository:**
     + Fork an existing public repository, make changes locally, push those changes to their forked repository, and then create a pull request for the original maintainers to review. This teaches them the contribution workflow.
4. **Understanding Git Stash:**
   * Introduce the git stash command that allows temporarily saving uncommitted changes. Practice using stash to keep your working directory clean.
5. **Ignoring Files/Directories:**
   * Create a .gitignore file to specify files or directories that should be excluded from version control (e.g., compiled files, configuration files).
6. **Git Tags:**
   * Demonstrate using Git tags to mark specific points in the development history. Trainees can create tags to signify project milestones or releases.

**Additional Tips:**

* **Visualizing Git Workflow:** Create diagrams or animations to illustrate Git concepts like branching and merging. It will help you visualizing the essence of version control.
* **Real-world Scenarios:** Think about practical use cases for Git, like collaborating on a group project or managing different versions of your own code.
* **Interactive Learning:** Utilize online Git tutorials with interactive exercises to solidify their understanding.