Git and GitHub Basics

1. **Introduction to Version Control**

* **Session 1:**
  + What is Version Control?
  + Benefits of Version Control in Software Development
  + Overview of Git and GitHub
* **Session 2:**
  + Setting up Git: Installation and Configuration
  + Basic Git Terminology (Repository, Commit, Branch, etc.)
  + Hands-on: Creating your first Git repository
* [**Exercises**](#WS01_Introduction)

1. **Core Git Concepts and Commands**

* **Session 3:**
  + Understanding Git Workflow (Working Directory, Staging Area, Repository)
  + Adding Changes to the Staging Area (git add)
  + Saving Changes with Commits (git commit)
* **Session 4:**
  + Viewing Commit History (git log)
  + Reverting Changes (git checkout, git revert)
  + Hands-on Exercises
* [**Exercises**](#WS02_Git_Workflow)

1. **Branching and Merging**

* **Session 5:**
  + Introduction to Branching: Why It is Important
  + Creating and Switching Branches (git branch, git checkout)
  + Best Practices for Branching
* **Session 6:**
  + Merging Branches (git merge)
  + Resolving Merge Conflicts
  + Hands-on: Collaborative Branching and Merging
* [**Excercises**](#WS03_Branching_Merging)

**Week 4: Collaborating with GitHub**

* **Session 7:**
  + Setting up a GitHub Account
  + Creating Repositories on GitHub
  + Pushing Code to GitHub (git push)
* **Session 8:**
  + Cloning Repositories (git clone)
  + Pulling Changes (git pull)
  + Managing Collaborators and Permissions on GitHub

**Week 5: Advanced Git Techniques**

* **Session 9:**
  + Stashing Changes (git stash)
  + Using Tags (git tag) for Versioning
  + Introduction to Git Rebase
* **Session 10:**
  + Git Aliases for Productivity
  + Undoing Mistakes (git reset, git reflog)
  + Hands-on: Complex Scenarios

**Week 6: GitHub Workflow and Automation**

* **Session 11:**
  + Working with Issues and Pull Requests
  + Reviewing and Merging Pull Requests
  + Hands-on: Open Source Contribution Workflow
* **Session 12:**
  + Using GitHub Actions for Automation
  + Hands-on: Setting up a CI/CD Pipeline with GitHub Actions

**Week 7: Best Practices and Troubleshooting**

* **Session 13:**
  + Git Workflow Best Practices (e.g., Feature Branching, Commit Message Guidelines)
  + Troubleshooting Common Git Issues
* **Session 14:**
  + Managing Large Repositories (Git LFS)
  + Course Wrap-Up and Q&A

**Assessment and Project Work**

* **Weekly Assignments:** Exercises on the topics covered.
* **Final Project:** A collaborative GitHub repository with branches, issues, pull requests, and CI/CD setup.

**Learning Outcomes**

By the end of this course, students will be able to:

1. Understand and apply Git concepts in software projects.
2. Use GitHub for collaboration and project management.
3. Manage version control in solo and team environments.
4. Automate workflows using GitHub Actions.

**Workshop: Introduction to Version Control**

**Hands-On Exercise 1: Understanding Version Control Through a Real-World Scenario**

**Objective:** Illustrate version control concepts in a relatable way.

1. **Scenario:** Use a document or code file saved locally to simulate the challenges of versioning without Git.
   * Edit the document multiple times and save each version with a different name (e.g., file\_v1, file\_v2, etc.).
   * Discuss the confusion this approach might create in large projects.
2. **Activity:**
   * Create a table comparing manual versioning to Git versioning.
   * Reflect on how Git solves these problems, such as tracking history, reverting changes, and collaboration.

**Session 2: Setting Up Git and Understanding Basics**

**Hands-On Exercise 2: Setting Up Git Locally**

**Objective:** Install and configure Git for personal use.

1. **Steps:**
   * Download and install Git on your local machine.
   * Configure Git with your username and email using the commands:

bash

Copy code

git config --global user.name "Your Name"

git config --global user.email "youremail@example.com"

* + Verify the configuration with:

bash

Copy code

git config --list

1. **Outcome:** Students should have Git installed and ready for use.

**Hands-On Exercise 3: Create Your First Git Repository**

**Objective:** Set up and initialize a Git repository to understand Git basics.

1. **Steps:**
   * Create a new folder on your local machine, e.g., MyFirstRepo.
   * Navigate to the folder using the terminal and initialize it as a Git repository:

bash

Copy code

git init

* + Create a simple text file, hello.txt, and add some content to it.
  + Stage the file using:

bash

Copy code

git add hello.txt

* + Commit the file to the repository with a meaningful message:

bash

Copy code

git commit -m "Initial commit: Added hello.txt"

1. **Outcome:** Students will understand how to set up a Git repository and make their first commit.

**Hands-On Exercise 4: Exploring Basic Git Commands**

**Objective:** Learn the essential Git commands through practice.

1. **Steps:**
   * Check the current repository status:

bash

Copy code

git status

* + Make changes to hello.txt, such as adding a new line.
  + View the changes:

bash

Copy code

git diff

* + Stage and commit the changes:

bash

Copy code

git add hello.txt

git commit -m "Updated hello.txt with new content"

* + View the commit history:

bash

Copy code

git log

1. **Outcome:** Students will gain confidence in working with Git repositories and tracking changes.

**Bonus Exercise: Fun Git Exploration**

**Objective:** Reinforce the concept of Git tracking by modifying files.

1. **Activity:**
   * Create another file, funfact.txt, and add some fun facts about Git.
   * Repeat the steps of staging, committing, and logging history for this new file.
   * Encourage students to explore other commands like git show to view commit details.

These exercises provide a mix of conceptual clarity and practical experience, ensuring participants can follow along and apply their learning immediately.Top of Form

**Understanding Git Workflow and Saving Changes**

**Exercise 1: Exploring the Git Workflow**

**Objective:** Understand the roles of the Working Directory, Staging Area, and Repository.

1. **Steps:**
   * Create a folder GitWorkflowDemo and initialize it as a Git repository:

bash

Copy code

git init

* + Create a new file, notes.txt, and add some text (e.g., "Learning Git Workflow").
  + Run git status to see the file listed as untracked.
  + Stage the file using:

bash

Copy code

git add notes.txt

* + Run git status again and discuss the transition to the Staging Area.
  + Commit the file to the repository:

bash

Copy code

git commit -m "Added notes.txt with initial content"

* + Show how the file has moved to the Repository by running:

bash

Copy code

git log

1. **Outcome:** Students will understand how files transition through the Git workflow.

**Exercise 2: Adding Changes to the Staging Area**

**Objective:** Learn how to add and stage changes using git add.

1. **Steps:**
   * Make edits to notes.txt (e.g., add another line: "Understanding git add").
   * Stage the changes:

bash

Copy code

git add notes.txt

* + Use git status to confirm the changes are staged.

**Exercise 3: Saving Changes with Commits**

**Objective:** Practice saving changes to the repository using git commit.

1. **Steps:**
   * Commit the staged changes with a descriptive message:

bash

Copy code

git commit -m "Updated notes.txt with an explanation of git add"

* + Check the commit history with:

bash

Copy code

git log

* + Add multiple files (create tips.txt and examples.txt) and commit them together.

**Session 4: Viewing Commit History and Reverting Changes**

**Exercise 4: Viewing Commit History**

**Objective:** Explore git log and understand commit history.

1. **Steps:**
   * Use git log to display commit history.
   * Use git log --oneline to view a condensed history.
   * Use git log -p to display the changes introduced in each commit.

**Exercise 5: Reverting Changes with Git**

**Objective:** Undo changes using git checkout and git revert.

1. **Steps for git checkout:**
   * Make an edit to notes.txt but don’t stage the changes.
   * Revert the working directory to the last committed state:

bash

Copy code

git checkout -- notes.txt

* + Confirm that the changes are undone.

1. **Steps for git revert:**
   * Revert a specific commit from history:

bash

Copy code

git revert <commit-hash>

* + Verify that a new commit was created to undo the changes.

**Exercise 6: Practice with Mistakes**

**Objective:** Simulate common mistakes and their fixes.

1. **Scenario:** You accidentally staged a wrong file (wrongfile.txt).
   * Unstage it using:

bash

Copy code

git reset wrongfile.txt

* + Confirm it’s no longer staged by running git status.

**Bonus Exercises: Git Workflow Challenge**

1. Create three files (file1.txt, file2.txt, file3.txt) and perform the following:
   * Add changes to the staging area.
   * Commit the changes.
   * Make further edits and view the difference with git diff.
   * Revert one of the commits and observe the changes.
2. **Debugging Practice:**
   * Simulate a merge conflict by editing the same lines in two branches.
   * Practice resolving the conflict manually.

These exercises focus on both foundational commands and practical problem-solving to ensure a solid grasp of core Git concepts.

**Introduction to Branching and Merging**

**Exercise 1: Creating and Switching Branches**

1. **Objective**: Learn how to create and navigate between branches.
2. **Steps**:
   * Initialize a Git repository:

bash

Copy code

git init branching-workshop

cd branching-workshop

echo "Welcome to Branching Workshop" > readme.md

git add readme.md

git commit -m "Initial commit"

* + Create a new branch named feature-1:

bash

Copy code

git branch feature-1

* + Switch to feature-1 branch:

bash

Copy code

git checkout feature-1

* + Modify readme.md (e.g., add a line like Learning branching), then commit:

bash

Copy code

echo "Learning branching" >> readme.md

git add readme.md

git commit -m "Add a line to readme.md in feature-1 branch"

* + Switch back to the main branch:

bash

Copy code

git checkout main

* + Check the differences between the two branches:

bash

Copy code

git log --oneline --graph --all

**Exercise 2: Best Practices for Branching**

1. **Objective**: Understand naming conventions and purpose-driven branches.
2. **Scenario**: Assume you’re working on a project with the following tasks:
   * Create a branch for a bug fix: bugfix/user-login-error.
   * Create a branch for a feature addition: feature/add-user-profile.
3. **Task**:
   * Create and switch to these branches using the following commands:

bash

Copy code

git branch bugfix/user-login-error

git branch feature/add-user-profile

* + List all branches:

bash

Copy code

git branch

**Session 6: Merging and Resolving Conflicts**

**Exercise 3: Merging Branches**

1. **Objective**: Practice merging changes from a feature branch into the main branch.
2. **Steps**:
   * Merge feature-1 into main:

bash

Copy code

git checkout main

git merge feature-1

* + Check the merge results:

bash

Copy code

cat readme.md

**Exercise 4: Resolving Merge Conflicts**

1. **Objective**: Understand and resolve merge conflicts.
2. **Steps**:
   * Create a conflict:

bash

Copy code

git checkout main

echo "Main branch content" >> readme.md

git add readme.md

git commit -m "Update readme in main branch"

git checkout feature-1

echo "Feature branch content" >> readme.md

git add readme.md

git commit -m "Update readme in feature-1 branch"

git checkout main

git merge feature-1

* + Git will notify you of a merge conflict. Open readme.md in a text editor to resolve it.
  + Edit the file to combine changes:

text

Copy code

Main branch content

Feature branch content

* + Mark the conflict as resolved:

bash

Copy code

git add readme.md

git commit -m "Resolve merge conflict"

**Exercise 5: Collaborative Branching and Merging**

1. **Objective**: Simulate a real-world collaborative environment.
2. **Steps**:
   * Divide participants into pairs or small groups.
   * One participant initializes a repository and pushes it to GitHub:

bash

Copy code

git remote add origin <repository\_url>

git push -u origin main

* + Each team member:
    - Clones the repository.
    - Creates their branch (feature/team-member-name).
    - Makes changes and commits.
    - Pushes their branch to the remote repository:

bash

Copy code

git push origin feature/team-member-name

* + One participant merges all branches into the main branch:
    - Pull changes from the remote repository.
    - Merge branches into main locally.
    - Push the updated main branch to the remote repository.

**Additional Tips**

* Use visual tools like git log --oneline --graph --all to help students visualize branching and merging.
* Encourage the use of GUI tools like GitHub Desktop, GitKraken, or VSCode Git extensions for an alternative perspective.
* Provide pre-configured merge conflict scenarios for additional practice if time permits.

**Expected Outcomes**

* Students will understand the importance of branching.
* They will be able to create, switch, and merge branches.
* They will gain hands-on experience in resolving merge conflicts, an essential skill in collaborative projects.