**Linux Operating System and Source Code Management with Git-2.0**

**Assignment 1:** Initialize a new Git repository in a directory of your choice. Add a simple text file to the repository and make the first commit.

here's a step-by-step guide to initialize a new Git repository, add a simple text file, and make the first commit:

1. **Create a new directory (optional) and navigate to it**:

sh

Copy code

mkdir my\_new\_repo cd my\_new\_repo

1. **Initialize a new Git repository**:

sh

git init

This command initializes an empty Git repository in the current directory.

1. **Create a simple text file**:

sh

echo "This is a simple text file." > example.txt

This command creates a file named **example.txt** with the content "This is a simple text file."

1. **Add the text file to the staging area**:

sh

git add example.txt

This command stages the **example.txt** file, preparing it to be committed.

1. **Make the first commit**:

sh

git commit -m "Add initial text file"

This command commits the staged file to the repository with the commit message "Add initial text file."

**Summary of Commands**

Here are all the commands together for easy reference:

sh

mkdir my\_new\_repo

cd my\_new\_repo

git init echo "This is a simple text file." > example.txt

git add example.txt

git commit -m "Add initial text file"

**Explanation**

1. **mkdir my\_new\_repo**: Creates a new directory named **my\_new\_repo**.
2. **cd my\_new\_repo**: Navigates into the newly created directory.
3. **git init**: Initializes a new Git repository in the current directory.
4. **echo "This is a simple text file." > example.txt**: Creates a new text file named **example.txt** with the specified content.
5. **git add example.txt**: Stages the **example.txt** file, adding it to the staging area.
6. **git commit -m "Add initial text file"**: Commits the staged file to the repository with a descriptive commit message.

**Assignment 2:** Branch Creation and Switching

Create a new branch named 'feature' and switch to it. Make changes in the 'feature' branch and commit them.

the steps to create a new branch named **feature**, switch to it, make changes, and commit those changes.

1. **Ensure you are in your Git repository directory**:

sh

cd my\_new\_repo

1. **Create a new branch named feature**:

sh

git branch feature

This command creates a new branch named **feature**.

1. **Switch to the feature branch**:

sh

git checkout feature

Alternatively, you can create and switch to the new branch in one step using:

sh

git checkout -b feature

1. **Make changes in the feature branch**: For example, let's modify the **example.txt** file:

sh

echo "Adding some new content in the feature branch." >> example.txt

This command appends the line "Adding some new content in the feature branch." to the **example.txt** file.

1. **Stage the changes**:

sh

git add example.txt

This command stages the modified **example.txt** file.

1. **Commit the changes**:

sh

git commit -m "Add new content in the feature branch"

This command commits the staged changes with the message "Add new content in the feature branch."

**Summary of Commands**

Here are all the commands together for easy reference:

sh

cd my\_new\_repo

git branch feature

git checkout feature

# or combine the above two steps with: git checkout -b feature

echo "Adding some new content in the feature branch." >> example.txt

git add example.txt

git commit -m "Add new content in the feature branch"

**Explanation**

1. **cd my\_new\_repo**: Navigate to your Git repository directory.
2. **git branch feature**: Create a new branch named **feature**.
3. **git checkout feature**: Switch to the **feature** branch.
   * Alternatively, **git checkout -b feature** creates and switches to the **feature** branch in one step.
4. **echo "Adding some new content in the feature branch." >> example.txt**: Append new content to the **example.txt** file.
5. **git add example.txt**: Stage the changes to the **example.txt** file.
6. **git commit -m "Add new content in the feature branch"**: Commit the staged changes with a descriptive commit message.

**Assignment 3:** Feature Branches and Hotfixes

Create a 'hotfix' branch to fix an issue in the main code. Merge the 'hotfix' branch into 'main'

To create a 'hotfix' branch to fix an issue in the main code and merge it back into 'main', you can follow these steps in Git. Here is a step-by-step guide:

1. **Checkout to the main branch**: Ensure you are on the 'main' branch and that it's up to date.

bash

git checkout main git pull origin main

1. **Create a new 'hotfix' branch**: Create a new branch from 'main' to work on the hotfix.

bash

git checkout -b hotfix-branch

1. **Make the necessary changes**: Edit the files to fix the issue. Use your preferred text editor or IDE to make the changes.
2. **Stage the changes**: Add the modified files to the staging area.

bash

git add .

1. **Commit the changes**: Commit the changes with an appropriate commit message.

bash

git commit -m "Fix: [Brief description of the issue fixed]"

1. **Push the 'hotfix' branch to the remote repository**: Push the new branch to the remote repository.

bash

git push origin hotfix-branch

1. **Create a pull request (optional but recommended)**: On your Git hosting service (like GitHub, GitLab, Bitbucket), create a pull request to merge the 'hotfix' branch into 'main'. This allows for code review and automated testing before merging.
2. **Merge the 'hotfix' branch into 'main'**: After the pull request is reviewed and approved, merge it into the 'main' branch. If you are not using pull requests, you can merge directly using Git:

bash

git checkout main git pull origin main # Ensure your local 'main' is up-to-date git merge hotfix-branch

1. **Resolve any merge conflicts (if any)**: If there are any merge conflicts, resolve them manually, then add and commit the resolved files.

bash

git add . git commit -m "Resolve merge conflicts"

1. **Push the updated 'main' branch to the remote repository**: Push the merged changes to the remote repository.

bash

git push origin main

1. **Delete the 'hotfix' branch (optional)**: If the hotfix branch is no longer needed, you can delete it locally and remotely.

bash

git branch -d hotfix-branch # delete locally git push origin --delete hotfix-branch # delete remotely

**Example Commands**

bash

# Step 1: Checkout to main git checkout main git pull origin main

# Step 2: Create a hotfix branch git checkout -b hotfix-branch

# Step 3: Make changes (use your editor)

# Step 4: Stage changes git add.

# Step 5: Commit changes git commit -m "Fix: [Brief description of the issue fixed]"

# Step 6: Push the hotfix branch to remote git push origin hotfix-branch

# Step 8: Merge the hotfix branch into main git checkout main git pull origin main git merge hotfix-branch

# Step 10: Push the main branch to remote git push origin main

# Step 11: Delete the hotfix branch git branch -d hotfix-branch git push origin --delete hotfix-branch

Following these steps ensures that you create a dedicated branch to address the hotfix, review the changes before merging, and maintain a clean and up-to-date main branch.