**Placement Empowerment Program**

***Cloud Computing and DevOps Centre***

Set Up Git Branching: Create a new branch in your Git repository for testing. Add a new feature and merge it

Name: Kirubalini S I Department : IT



**Introduction**

Git branching allows developers to work on isolated features or fixes without disrupting the main project. It creates separate development lines called branches, diverging from the main branch (often "main" or "master"). Changes within a branch remain isolated until a deliberate merge. Merging integrates the branch's changes back into the main codebase. This enables parallel development by multiple developers and helps manage complex projects. Branches are easily created, switched between, and deleted, offering workflow flexibility. Git branching is fundamental for effective version control, promoting organized and collaborative development.

**Overview**

Setting up Git branching involves these key steps: First, you create a new branch to isolate your work. This branch acts like a separate copy of your project where you can experiment. Next, you implement your new feature within this branch, committing your changes as you go. Finally, once you're satisfied, you merge the branch back into the main branch, integrating your new feature into the main project. This process keeps your main codebase stable while allowing for safe experimentation and parallel development.

**Objective**

The objectives of setting up Git branching, creating a new branch, adding a feature, and merging it are:

**Isolation:** To isolate new feature development or bug fixes from the main codebase, preventing instability or conflicts.

**Parallel Development:** To enable multiple developers to work on different features simultaneously without interfering with each other's work.

**Experimentation:** To provide a safe space to experiment with new ideas or code changes without risking the stability of the main project.

**Organized Development:** To manage complex projects with multiple ongoing features in a structured and controlled manner.

**Code Stability:** To ensure the main codebase remains stable and functional while new features are being developed and tested.

**Feature Integration:** To integrate completed and tested features back into the main project smoothly and efficiently.

**Importance**

Setting up Git branching, creating a new branch, adding a feature, and merging it are crucial for modern software development due to several key benefits:

**Isolation and Stability:** Branching isolates new work, preventing unstable or incomplete features from affecting the main codebase. This ensures the main branch remains stable and deployable.

**Parallel Development:** Multiple developers can work on different features simultaneously without stepping on each other's toes. This significantly speeds up development cycles.

**Safe Experimentation:** Branches provide a sandbox for experimenting with new ideas. If a change doesn't work out, the branch can simply be abandoned without impacting the main project.

**Version Control and History:** Branching maintains a clear history of all changes, making it easy to track progress, revert to previous versions, and understand how features were developed.

**Organized Workflow:** Branching promotes a structured and organized development workflow, especially for complex projects with multiple features in progress.

**Efficient Collaboration:** Branching facilitates collaboration by allowing developers to work on separate parts of a project and then easily integrate their changes.

**Simplified Testing:** Features can be thoroughly tested within their branch before being merged into the main branch, improving code quality.

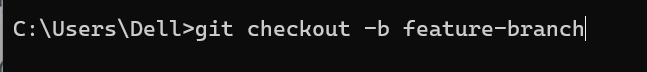
**Controlled Releases:** Branching allows for controlled releases by enabling developers to prepare and test features in a separate branch before deploying them to production.

**Step-by-Step Overview**

Step1:

Create a New Branch:

First, navigate to your Git repository in the terminal and check out a new branch for your feature.



This creates and switches you to a new branch called feature branch.

Step 2:

Add a New Feature:

Make changes to your code to add the new feature. Once you’ve made the necessary changes, save the files.

Step 3:

Stage the Changes:

After editing your files, stage the changes for commit.



Step 4:

Commit the Changes:

Next, commit the changes with a relevant message.



Step 5:

Push the New Branch:

If you’re working with a remote repository (like GitHub or GitLab), push the branch to the remote.



Step 6:

Switch Back to the Main Branch:

Once you’re done with the feature, switch back to the main branch (or whatever your main branch is called, e.g., main or master)



Step 7:

Merge the Feature Branch:

Now, merge the new feature branch into the main branch.



Step 8:

Push the Changes to the Remote:

If everything looks good, push the updated main branch to the remote repository.



Step 9:

Clean Up:

Once the feature is merged, you can delete the feature branch both locally and remotely if no longer needed.





**Expected Outcome**

The outcome of creating a simple backup script for your Git repository is a reliable, automated system that safeguards your project data. Specifically, you'll achieve the following:

* **Automated Daily Backups:** The script will run automatically (e.g., via cron) each day, creating a fresh backup of your Git repository without any manual intervention.
* **Local Copy of Your Repository:** A complete copy of your Git repository, including all files, directories, and Git history, will be stored in a designated local folder.
* **Protection Against Data Loss:** This local backup acts as a safeguard against accidental deletions, hard drive failures, remote repository issues (e.g., server outages, account problems), or other unforeseen events that could lead to data loss.
* **Improved Recovery Time:** In the event of data loss, restoring your repository from a local backup will be significantly faster and easier than cloning from a remote server, saving you valuable time and effort.
* **Increased Peace of Mind:** Knowing that your project is backed up daily will give you peace of mind and allow you to focus on your work without worrying about data loss.
* **Enhanced Data Security:** Having a local backup adds an extra layer of security, protecting your project from potential vulnerabilities or issues with your remote Git hosting provider.
* **Practical Scripting Skills:** You'll gain valuable experience in shell scripting, automation, and working with Git commands, enhancing your technical skills.