



**Placement Empowerment Program**

***Cloud Computing and DevOps Centre***

Create a new branch in your git repository for testing and add a new feature and merge it

Name:M Anumitha Department:AML



**INTRODUCTION:**

To create a proof of concept (POC) for a new feature in your Git repository, start by creating a new branch. This ensures that your POC work won’t interfere with the main branch. First, make sure you're on the main branch and pull the latest updates. Then, create a new branch using git checkout -b feature-poc and switch to it. Once on the new branch, proceed with developing the feature for your POC. After implementing the necessary changes, stage them with git add ., commit the changes with a message like git commit -m "Add POC for new feature", and push the branch to the remote repository using git push origin feature-poc. Once you’ve tested and are satisfied with the POC, switch back to the main branch using git checkout main, merge the feature branch with git merge feature-poc, and resolve any conflicts if necessary. Finally, push the merged changes to the remote repository using git push origin main. If the POC is complete and you no longer need the feature branch, you can clean up by deleting the branch locally with git branch -d feature-poc and remotely with git push origin --delete feature-poc. This process allows you to experiment and test new ideas without disrupting the main codebase.

**OVERVIEW:**

Creating a new branch in your Git repository for a proof of concept (POC) is an effective way to experiment with new features or ideas without disrupting the main codebase. This method ensures that your POC work remains isolated, allowing you to freely test and develop without affecting the stability of the main branch. The process begins by ensuring you're on the latest version of the main branch and then creating a dedicated branch specifically for the POC. Once the new branch is created, you can implement the feature you're testing, whether it's a new functionality, design, or concept.

After completing your changes, you can stage, commit, and push your changes to the remote repository for collaboration or backup. Once you’re confident in your POC and have tested it thoroughly, the next step is to merge the feature branch back into the main branch, ensuring your new work integrates smoothly with the rest of the project. If there are any merge conflicts, they can be resolved manually. After merging, you push the updated main branch to the remote repository.

Once the POC is fully integrated, you may want to delete the feature branch to keep your repository clean and organized. This is especially helpful for temporary branches that are no longer needed after the feature has been validated. Overall, using a separate branch for POC work allows you to safely develop and test new ideas while maintaining a stable and well-structured codebase.

**Top of Form**

**Bottom of Form**

**OBJECTIVE:**

1.Isolate POC Work: Keep experimental features or new ideas separate from the main codebase to avoid disrupting the stable version of the project.

2. Safe Testing and Development: Provide a safe environment to test and develop new features without risk to the main branch, ensuring that changes can be evaluated and discarded if needed.

3. Clear Version Control: Track changes specific to the POC, making it easier to understand what was tested, when, and how, without cluttering the main branch's commit history.

4. Facilitate Collaboration: Allow team members to review, comment, or contribute to the POC without affecting other parts of the project.

5. Easy Merging and Integration: Once the POC is complete and validated, easily merge the feature branch back into the main branch to integrate the new functionality or improvements.

6. Conflict Resolution: Handle potential merge conflicts during integration, ensuring that the final code is stable and free of issues.

7. Maintain Repository Cleanliness: After the POC is merged or discarded, delete the feature branch to maintain an organized and manageable repository.

8. Experimentation without Risk: Enable experimentation with minimal risk, knowing that any issues can be isolated to the feature branch and resolved without impacting the main codebase.

**Importance of Setting Up a Local Repository**

1. **Preserves Code Stability**: By working on a POC in a separate branch, you ensure that the main branch remains stable. This prevents unfinished or experimental features from accidentally affecting the main project, which could lead to bugs, errors, or broken functionality.
2. **Isolation of Work**: POC development often involves trying out new ideas, testing different approaches, or exploring untested features. Having a dedicated branch ensures that these changes remain isolated from the rest of the codebase, reducing the risk of introducing unwanted changes or inconsistencies.
3. **Safe Experimentation**: A separate branch allows for free experimentation without worrying about breaking anything. If the POC turns out to be unsuccessful or unnecessary, you can simply discard the branch without impacting the main project. This encourages innovation and testing new ideas without fear of consequences.
4. **Simplified Collaboration**: If you're working in a team, setting up a POC branch makes it easier for others to review, collaborate, or provide feedback without interrupting the work being done on the main branch. It ensures a smoother workflow, particularly in teams where multiple features might be under development simultaneously.
5. **Better Version Control**: Using a separate branch helps keep the commit history of the main branch clean and organized. The commits related to the POC are contained within the feature branch, making it easier to track the development of that specific feature or idea, and reducing clutter in the main branch’s history.
6. **Easier Rollback or Rework**: If the POC needs to be rolled back or significantly reworked, it's much easier to do so on a dedicated branch. You can either delete the branch entirely or make changes without affecting other ongoing development work.
7. **Streamlined Merging**: Once the POC is tested and you’re ready to integrate it into the main branch, having it on a separate branch makes the merging process more controlled and less risky. You can merge the feature when it's fully validated, ensuring it integrates smoothly with the existing code.
8. **Keeps the Repository Clean**: By deleting the POC branch after it's no longer needed, you maintain a tidy, well-structured repository. This is particularly important for long-term project maintenance and scalability, as it avoids unnecessary clutter and makes it easier for new developers to navigate the codebase.

I

Top of Form

Bottom of Form

**STEP BY STEP OVERVIEW:**

**1.Initialize the git repository:**

****initialize a git repository if it doesn’t exist.

**2.give a commit message that defines that it is a initial commit.**

****

**3.**

**In Git, the purpose of a file like feature.py (or any file) is usually to track changes, manage versions, and collaborate with others on your project. When you make modifications to files like feature.py, you can use Git to:**

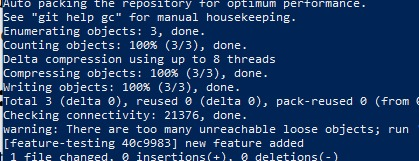
1. **Track Changes: Git tracks changes to your file, including additions, deletions, and modifications, over time. This helps you keep a history of all the changes made to the project.**
2. **Version Control: Each change you commit creates a new version of the file, making it easy to roll back to previous versions if something goes wrong.**
3. **Collaboration: Git enables multiple people to work on the same project at the same time, allowing you to merge changes made by different team members.**
4. **Branching: You can use branches to work on new features (e.g., adding a new feature in feature.py) without affecting the main codebase. Once your feature is complete, you can merge it back into the main branch.**

****

**4.again give a commit message after the new feature file added to the new branch for tracking changes**

****

**5.1 file changed with zero modifications**

****

**6.create a new branch named feature -testing using this command:**

****

**7.To view the status of main branch add a command that includes:**

****

**8.**

****

**9.Merge the feature-testing branch into the main branch:**

****

**Expected outcome:**

In Git, the purpose of a file like feature.py (or any file) is usually to track changes, manage versions, and collaborate with others on your project. When you make modifications to files like feature.py,

1. Track Changes: Git tracks changes to your file, including additions, deletions, and modifications, over time. This helps you keep a history of all the changes made to the project.
2. Version Control: Each change you commit creates a new version of the file, making it easy to roll back to previous versions if something goes wrong.
3. Collaboration: Git enables multiple people to work on the same project at the same time, allowing you to merge changes made by different team members.
4. Branching: You can use branches to work on new features (e.g., adding a new feature in feature.py) without affecting the main codebase. Once your feature is complete, you can merge it back into the main branch.

The expected outcome of the command you provided (echo "feature added" >> feature.py) is that the text "feature added" will be appended to the feature.py file.

1. Appending to the File: The echo command writes the string "feature added" to the file feature.py. The >> operator ensures that the content is appended to the file rather than overwriting it.
   * If feature.py already exists, the text will be added at the end of the file.
   * If feature.py does not exist, the file will be created, and the text will be written to it.
2. Git Stage: If you're planning to use Git to track the changes, the file will have the new text, and you'll need to stage and commit this change. The typical workflow would be:
   * Staging: git add feature.py (this stages the file for commit).
   * Committing: git commit -m "Add feature to feature.py" (this commits the change with a descriptive message).

After this, you'll have a versioned history of the changes, and you can push it to a remote repository (e.g., GitHub or GitLab).

The expected outcome of merging a feature branch into the main branch in Git is that any changes made in the feature branch (such as modifications to feature.py) will be integrated into the main branch.

Steps to Merge the feature Branch into main:

1. Ensure You’re on the main Branch: Before merging, you need to switch to the main branch:
2. Update the main Branch: It's good practice to ensure that your main branch is up to date, especially if others may have made changes. Run:
3. Merge the feature Branch: Now that you're on the main branch, you can merge the feature branch into it: