😊 Git Branching Model

The **Git branching model** refers to the strategy or methodology used to manage and organize branches in a Git repository. A branch is an independent line of development, and adopting a consistent branching model helps streamline collaboration, development, and deployment workflows. Below are some common branching models and concepts:

Common Git Branching Models

1. Centralized Workflow

- A simple model where there is a single main branch (e.g., main or master).
- Developers pull from and push changes directly to this branch.
- Useful for small teams or solo projects but lacks isolation for features or bug fixes.

2. Feature Branch Workflow

- Each new feature or task is developed in a dedicated branch.
- Branches are usually named descriptively, e.g., feature/login-page.
- After development, the branch is merged into the main branch (e.g., main) via pull requests (PRs).
- Helps isolate changes and makes code reviews easier.

3. Git Flow

- A robust branching model suitable for large projects.
- Main branches:
    - **main**: Holds production-ready code.
    - **develop**: Used for ongoing development.
- Supporting branches:
    - **Feature branches** (feature/*): For new features.
    - **Release branches** (release/*): For preparing a release.
    - **Hotfix branches** (hotfix/*): For urgent fixes in production.
- Provides clear workflows for feature development, releases, and bug fixes but can be complex for small teams.

4. GitHub Flow

- A lightweight branching model often used in CI/CD environments.
- Main principles:
    - Always deploy from the main branch.

    - Create branches for new features (e.g., feature/add-user-profile).
    - Open a pull request for code review and discussion.
    - Merge the branch into main after testing and approval.
    - Deploy immediately after merging.

5. GitLab Flow

- A flexible branching model combining GitHub Flow with support for environments like staging and production.
- Involves creating environment-specific branches (e.g., staging, production).
- Supports issue tracking, CI/CD, and merge requests.

Key Branching Concepts

1. **Main Branches**
    - main or master: The default branch where the stable or production-ready code resides.
    - develop: A secondary branch for integrating features before releasing.
2. **Feature Branches**
    - Used for implementing individual features or tasks.
    - Typically branched off develop or main.
3. **Release Branches**
    - Used to prepare code for release.
    - Allows final adjustments like documentation updates or version bumps.
4. **Hotfix Branches**
    - Used to quickly address critical issues in the main branch.
5. **Branch Naming Conventions**
    - Use descriptive and consistent names, e.g., feature/, bugfix/, hotfix/, or release/.

Choosing the Right Branching Model

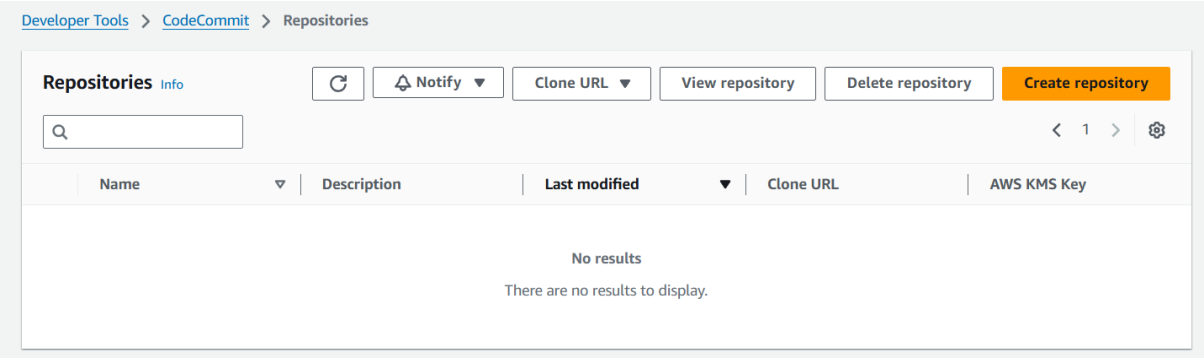
The choice of branching model depends on:

- **Team size**: Smaller teams may prefer simpler workflows like GitHub Flow.
- **Project complexity**: Large projects with multiple releases may benefit from Git Flow.
- **Deployment frequency**: Rapid deployment cycles align well with GitHub Flow or GitLab Flow.

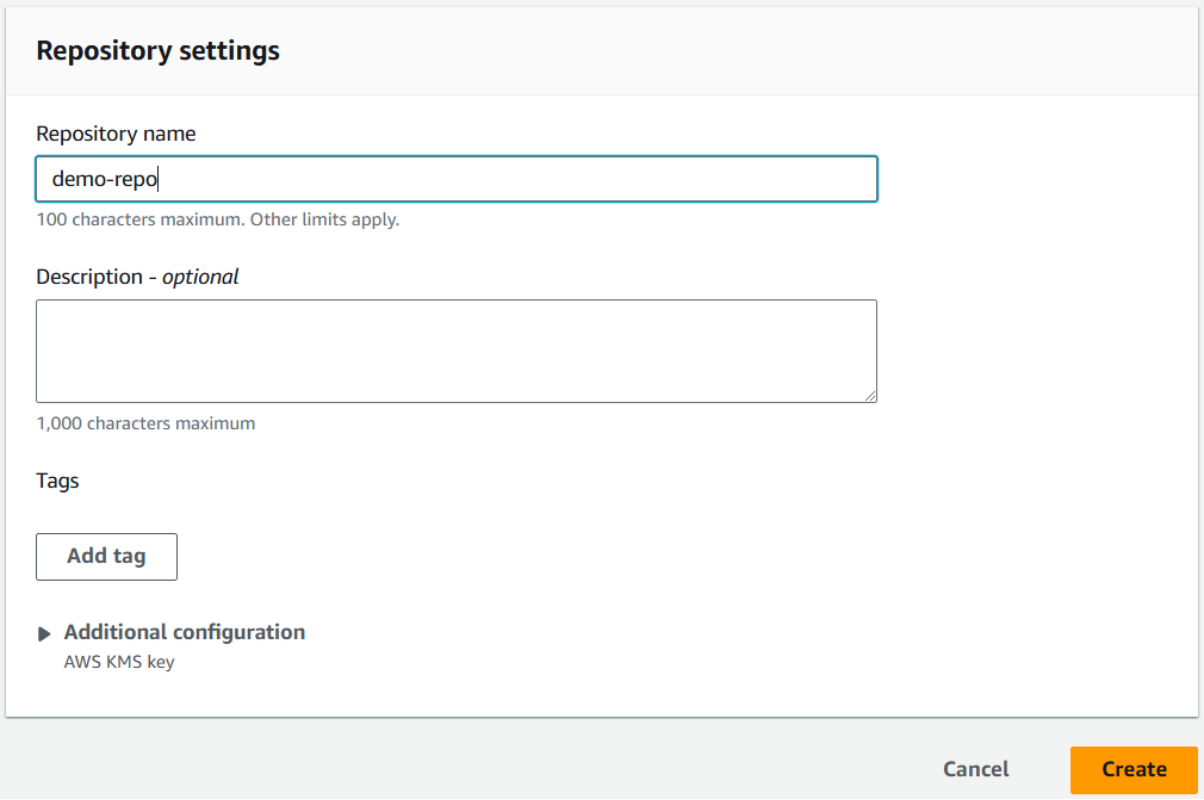
Adopting a clear branching model can enhance collaboration, reduce conflicts, and improve code quality.

😄 To begin with the Lab:

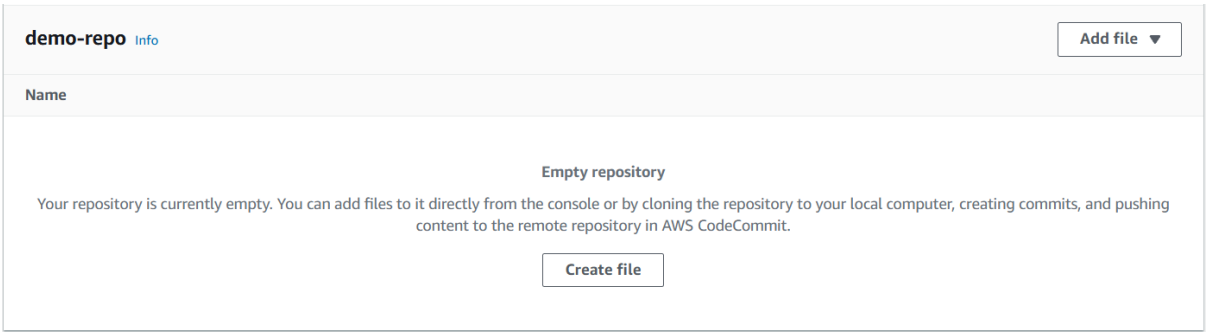
1. In this lab, we will understand the concept of Git branches through a small example: We will create a repository, and from the main branch, we will create a feature branch to showcase a better workflow environment.
2. First, in your AWS console open Code Commit and create a new repository.



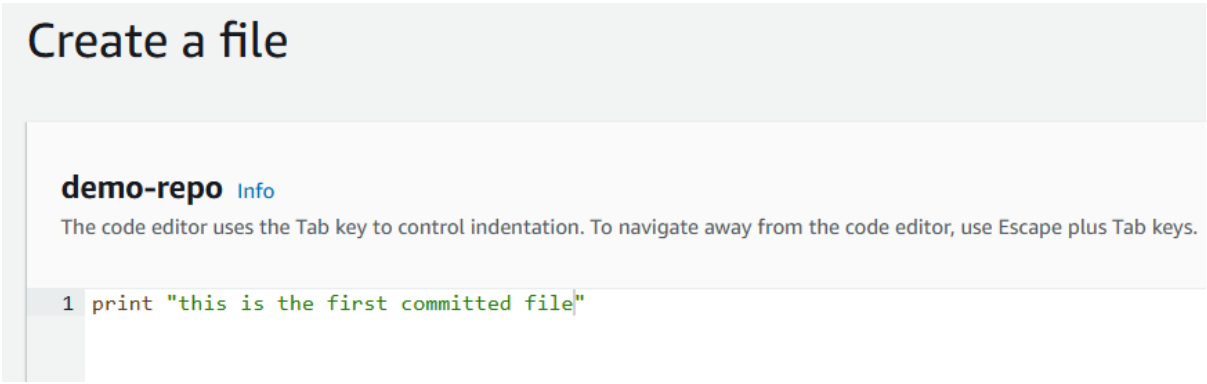
3. Here you need to give your repository a name and click on create.

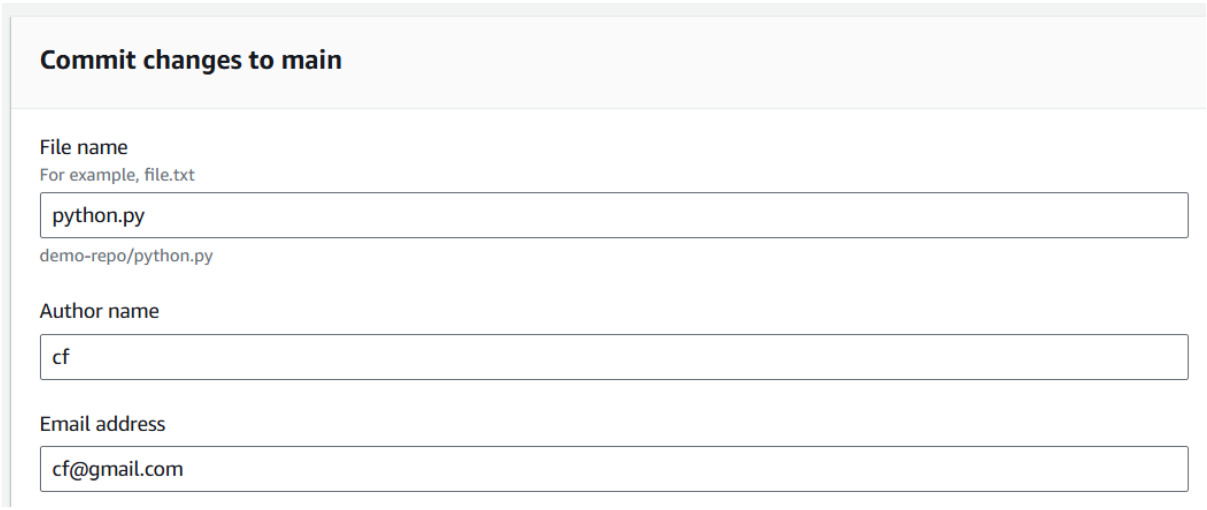


4. Then you need to create a file in your repository.

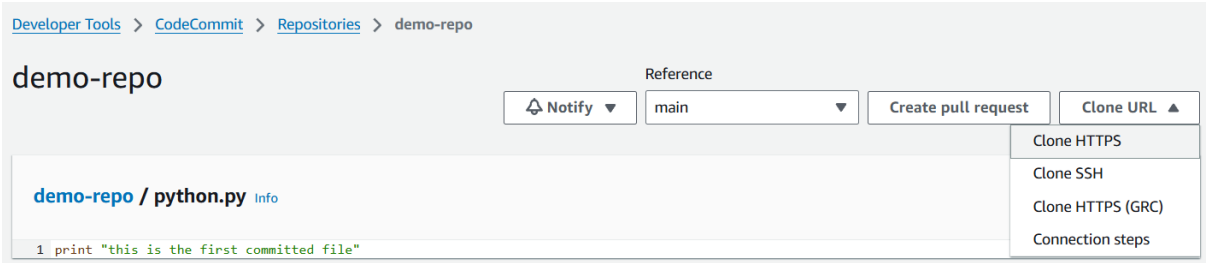


5. We will create a Python file and add a print statement here. Then give a name to your file, fill in the details, and click on commit changes.

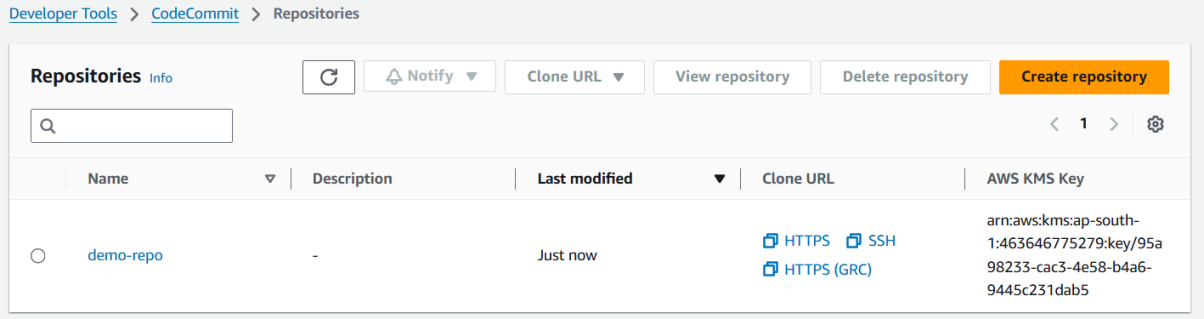




6. Now you can see that the file has been created.



7. Then from your repository you need to copy the HTTPS clone URL to clone your repository locally on your laptop.



8. So, use the git clone command to clone your repository in any folder on your laptop.

```
      @LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat
$ git clone https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/demo-repo
Cloning into 'demo-repo'...
remote: Counting objects: 3, done.
Unpacking objects: 100% (3/3), 223 bytes | 24.00 KiB/s, done.

      @LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat
$ ls
demo-repo/
```

9. After that go inside the demo-repo and look at which branch you are currently in. as you can see, we are in the main branch.

```
      @LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat
$ cd demo-repo/

      @LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (main)
$ git branch
* main
```

10. Now we created a new branch with the name little-feature and now we can see that we are in the little-feature branch.

    git checkout -b little-feature
    git branch

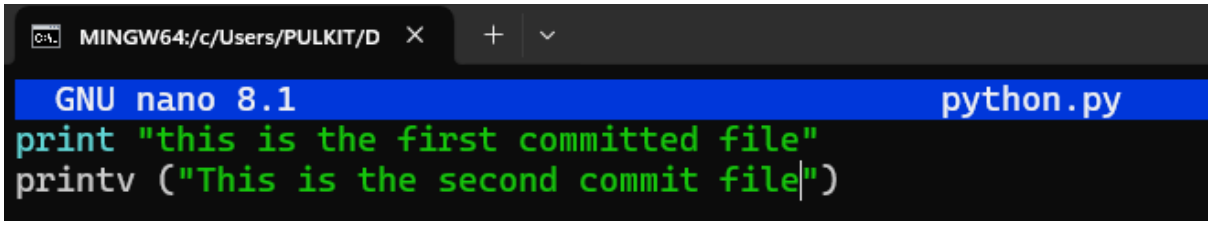
```
      @LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (main)
$ git checkout -b little-feature
Switched to a new branch 'little-feature'

      @LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (little-feature)
$ git branch
* little-feature
  main
```

11. So, what we have done is from our main or master branch we have forked a new branch whose name is little-feature. Also, this little feature branch will have all the files that were present in the main or master branch.
12. Let's open the python.py file in our little-feature branch to add some changes for that run the command given below.

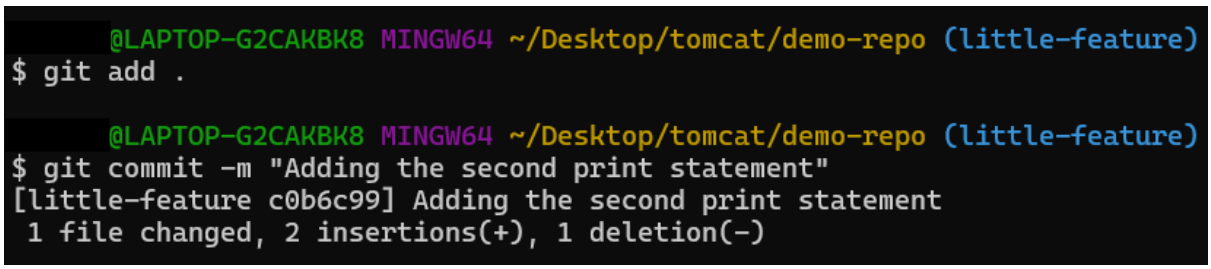
nano python.py

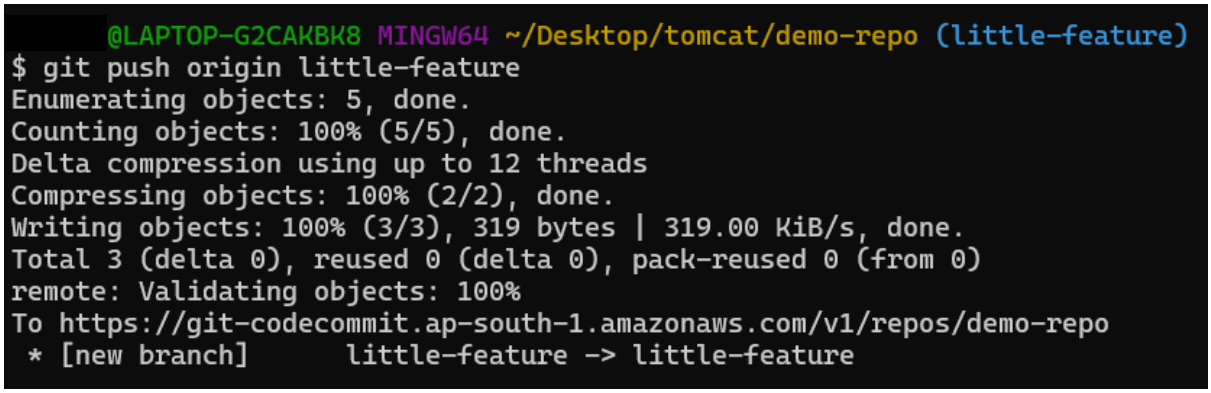
13. As you can see below, we have added a new print statement and saved it in the python.py file through the little-feature branch.



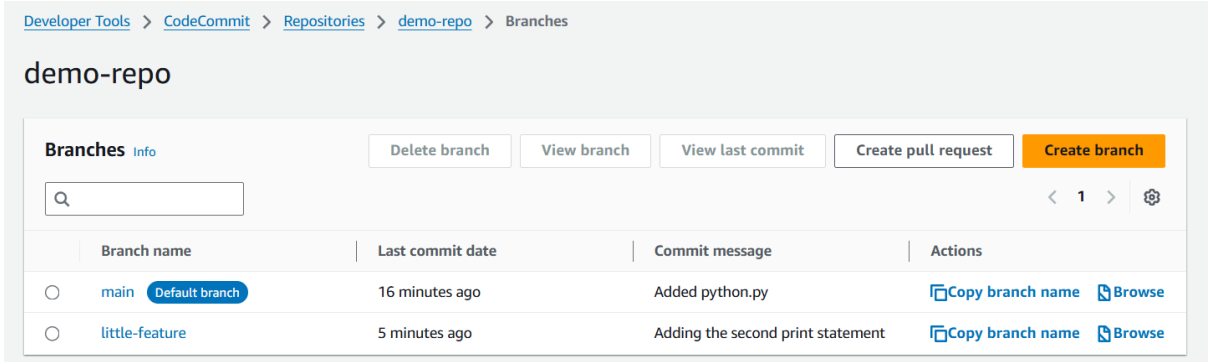
14. After that we added the new changes, committed them, and pushed the new changes to the repository.

git add .
git commit -m "--"
git push origin little-feature

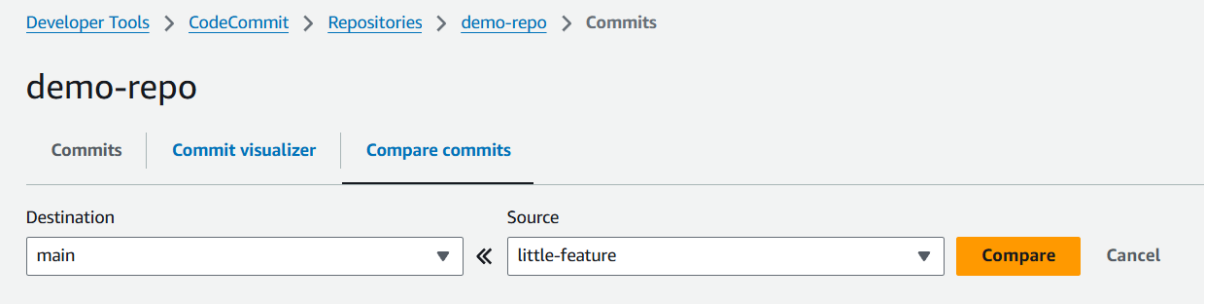




15. Once the changes have been pushed now you need to open the console go to code commit, open your repository, and go to branches. Here you will see 2 branches.



16. Now if you want to see what are changes between your main and little-feature branches then you can go to commits from the left pane and compare the commits. Choose your main branch and little-feature branch, click on compare.



17. Here it will show you the comparison. On the left side, we have the main branch and on the right side we have the little-feature branch in the little-feature branch, we can see that the green line has been added. Also, there is no change on the first line.