



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

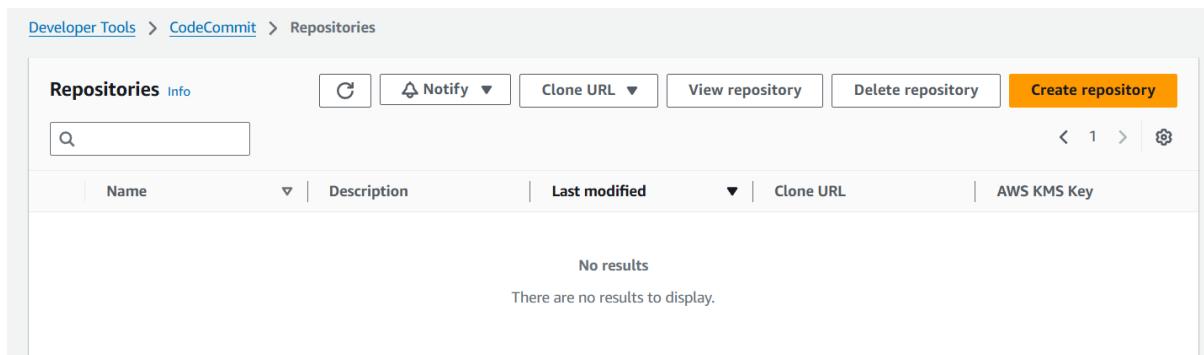
The choice of branching model depends on:

- **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.

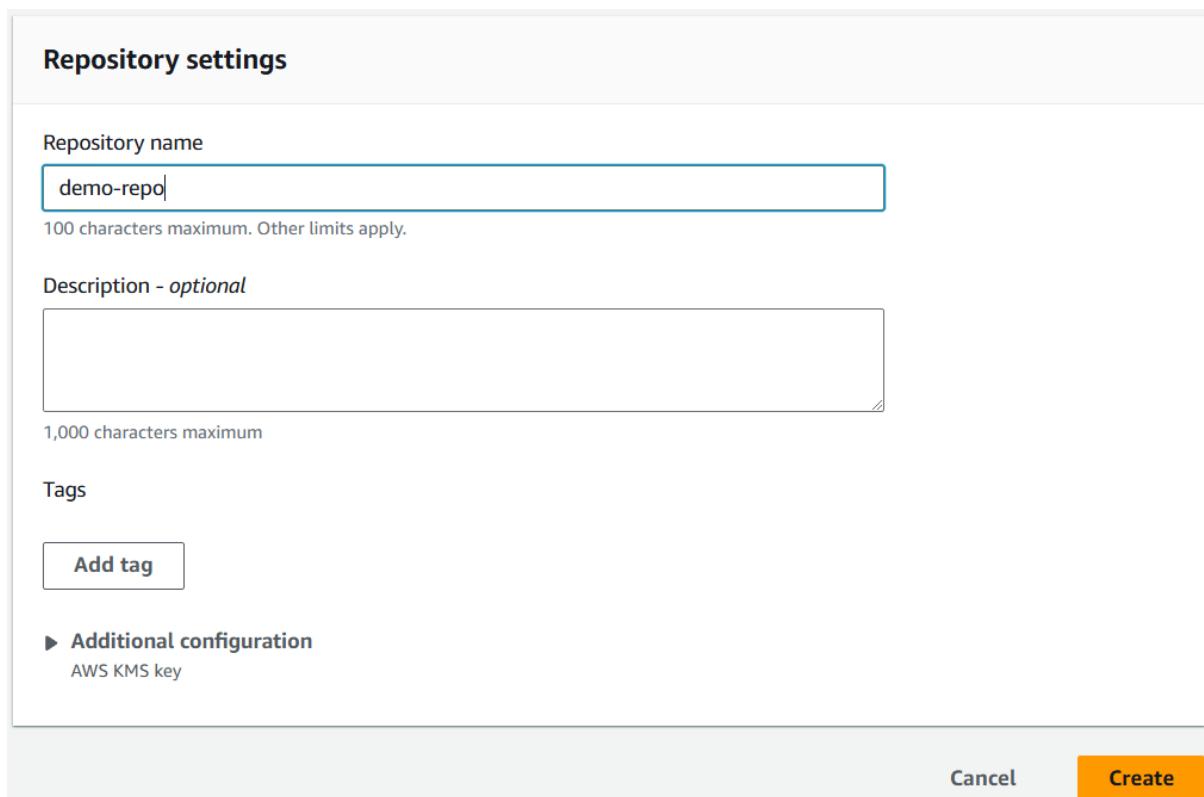
To begin with the Lab:

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.



The screenshot shows the AWS CodeCommit interface. At the top, there's a breadcrumb navigation: Developer Tools > CodeCommit > Repositories. Below that is a search bar and a toolbar with buttons for Create, Notify, Clone URL, View repository, Delete repository, and Create repository (which is highlighted in orange). A search input field is empty. The main area has a header 'Repositories' with a 'Info' tab selected. It includes columns for Name, Description, Last modified, Clone URL, and AWS KMS Key. A message 'No results' indicates there are no repositories to display.

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



The screenshot shows the 'Repository settings' dialog. It starts with a 'Repository name' field containing 'demo-repo'. A note below says '100 characters maximum. Other limits apply.' Below that is a 'Description - optional' text area which is currently empty. A note below it says '1,000 characters maximum'. Under 'Tags', there's an 'Add tag' button. At the bottom, there's a section titled 'Additional configuration' with a 'AWS KMS key' option. At the very bottom right are 'Cancel' and 'Create' buttons, with 'Create' being highlighted in orange.

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

The screenshot shows the AWS CodeCommit interface for a repository named 'demo-repo'. At the top right is a 'Add file ▾' button. Below it is a 'Name' input field. In the center, the text 'Empty repository' is displayed with the message: 'Your repository is currently empty. You can add files to it directly from the console or by cloning the repository to your local computer, creating commits, and pushing content to the remote repository in AWS CodeCommit.' A 'Create file' button is located at the bottom right of this section.

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.

The screenshot shows the 'Create a file' interface for the 'demo-repo'. The title bar says 'Create a file'. Below it is a code editor window with the following content:

```
1 print "this is the first committed file"
```

The code editor has a note: 'The code editor uses the Tab key to control indentation. To navigate away from the code editor, use Escape plus Tab keys.'

The screenshot shows the 'Commit changes to main' interface. It includes fields for 'File name' (set to 'python.py'), 'Author name' (set to 'cf'), and 'Email address' (set to 'cf@gmail.com').

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

The screenshot shows the AWS CodeCommit interface for the 'demo-repo'. The navigation bar at the top shows 'Developer Tools > CodeCommit > Repositories > demo-repo'. The repository name 'demo-repo' is at the top left. A 'Reference' dropdown is set to 'main'. To its right are 'Notify' (dropdown), 'Create pull request', and 'Clone URL' (dropdown). The 'Clone URL' dropdown menu is open, showing options: 'Clone HTTPS', 'Clone SSH', 'Clone HTTPS (GRC)', and 'Connection steps'. Below the reference dropdown, the file 'python.py' is listed with the content: '1 print "this is the first committed file"'.

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

Repositories					
Info		Notify	Clone URL	View repository	Delete repository
<input type="text"/> 1					
Name	Description	Last modified	Clone URL	AWS KMS Key	
demo-repo	-	Just now	HTTPS SSH HTTPS (GRC)	arn:aws:kms:ap-south-1:463646775279:key/95a98233-cac3-4e58-b4a6-9445c231dab5	

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'

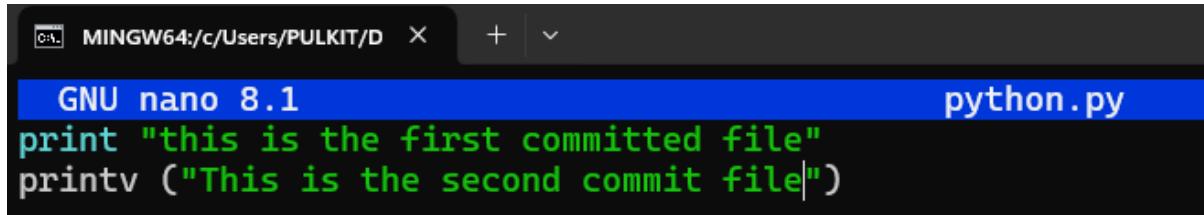
@LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (little-feature)
$ git branch
* little-feature
  main
```

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

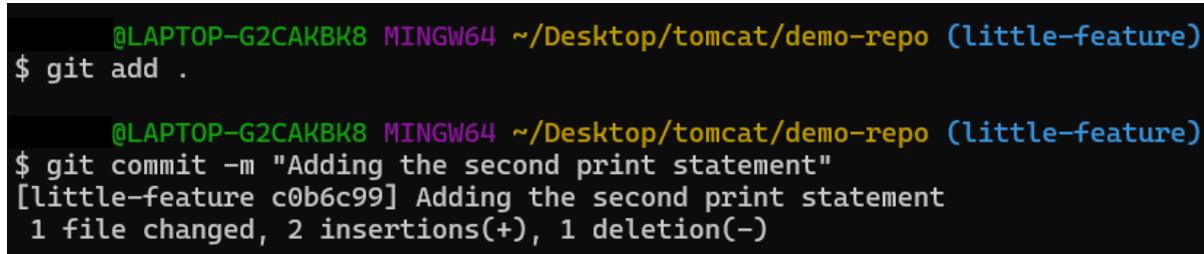
13. As you can see below, we have added a new print statement and saved it in the python.py file through the little-feature branch.



```
MINGW64:/c/Users/PULKIT/D X + v
GNU nano 8.1 python.py
print "this is the first committed file"
printv ("This is the second commit file")
```

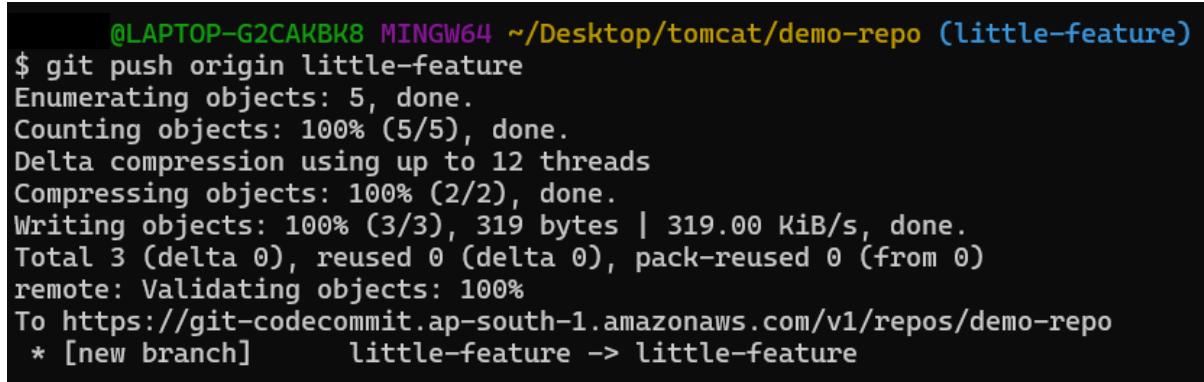
14. After that we added the new changes, committed them, and pushed the new changes to the repository.

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



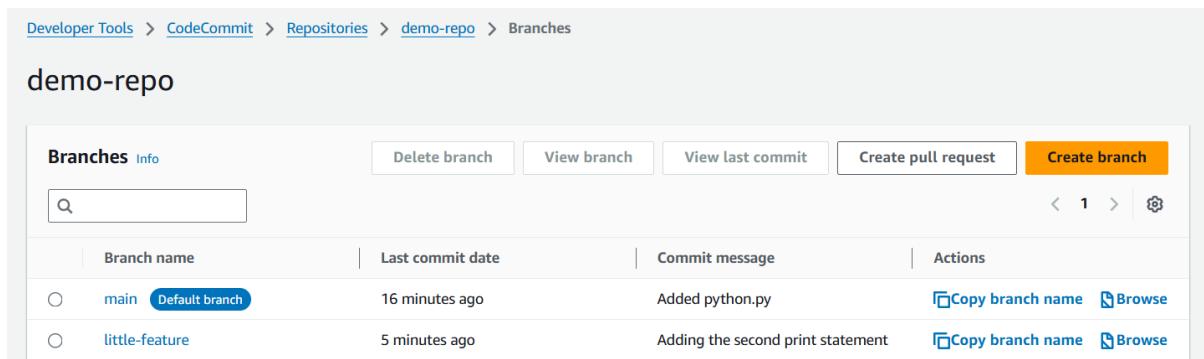
```
@LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (little-feature)
$ git add .

@LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (little-feature)
$ git commit -m "Adding the second print statement"
[little-feature c0b6c99] Adding the second print statement
 1 file changed, 2 insertions(+), 1 deletion(-)
```



```
@LAPTOP-G2CAKBK8 MINGW64 ~/Desktop/tomcat/demo-repo (little-feature)
$ git push origin little-feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 319 bytes | 319.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Validating objects: 100%
To https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/demo-repo
 * [new branch]      little-feature -> 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.



Developer Tools > CodeCommit > Repositories > demo-repo > Branches

demo-repo

Branches				Info	Delete branch	View branch	View last commit	Create pull request	Create branch
				<input type="text"/>	<	1	>		
	Branch name	Last commit date	Commit message	Actions					
<input checked="" type="radio"/>	main <small>Default branch</small>	16 minutes ago	Added python.py	Copy branch name Browse					
<input type="radio"/>	little-feature	5 minutes ago	Adding the second print statement	Copy branch name Browse					

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.

The screenshot shows the AWS CodeCommit interface for comparing commits. At the top, there's a navigation bar: Developer Tools > CodeCommit > Repositories > demo-repo > Commits. Below that, the repository name "demo-repo" is displayed. A horizontal menu bar has three items: Commits, Commit visualizer, and Compare commits, with "Compare commits" being the active tab. Underneath, there are two dropdown menus: "Destination" set to "main" and "Source" set to "little-feature". To the right of these dropdowns are two buttons: "Compare" (highlighted in orange) and "Cancel".

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.

The screenshot shows the AWS CodeCommit interface for comparing commits, specifically for the file "python.py". The "Compare" button has been clicked, and the interface now displays a diff view. At the top, there are buttons for "Hide comments", "Hide whitespace changes", and "Unified" (which is selected, indicated by a checked radio button). To the right of these is a "Split" button, which is also checked and highlighted with a red box. Below this, the file content is shown in a split pane. The left pane shows line 1: `print "this is the first committed file"`. The right pane shows line 1: `print "this is the first committed file"` and line 2: `+ printv ("This is the second commit file")`. There is a green highlight under the entire line 2 in the right pane, indicating it is a new addition. At the bottom of the diff view, there are buttons for "Browse file contents" and "Comment on file".