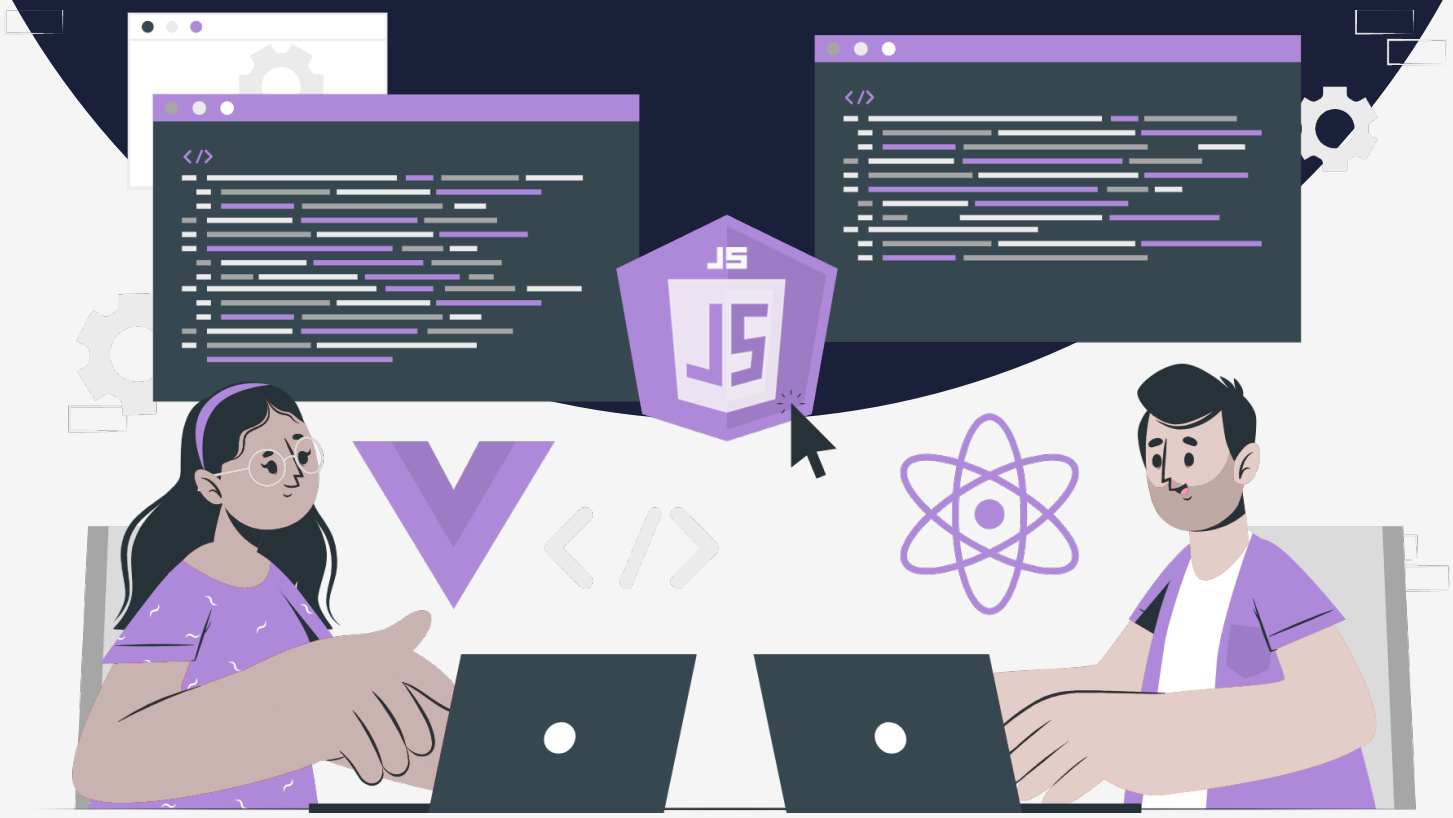
Lesson:

Creating Branch, Merging Branch, & Checkout to main branch

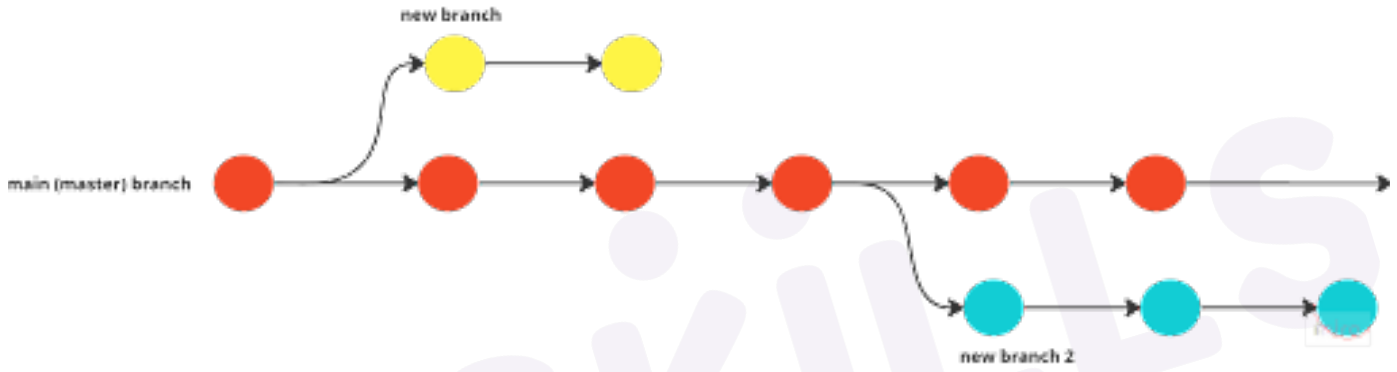


Topics Covered

- What is Branch in Git
- How to create a new Branch and Checkout to it
- How to Merge Branch

What is Branch in Git

A branch in Git is like a separate path or timeline of changes for your project. When you create a branch, you're essentially creating a copy of your project at a specific point in time, usually the latest commit on the main branch (which is often called the "master" branch). This copy allows you to make changes to your project without affecting the original main (master) branch.



Each branch has its own unique name and points to a specific commit in the project's history. Think of a commit as a snapshot of the project at a specific point in time. When you create a new branch, it points to the same commit as the branch you created it from. But as you make changes to the new branch, it will start to diverge from the original branch.

This means that you can work on different features or fixes in parallel, without affecting each other's work. For example, you can create a new branch to add a new feature to your project, while another team member creates a separate branch to fix a bug. When you're ready to merge your changes back into the main branch, Git will automatically combine the changes in a way that makes sense.

Overall, branches in Git provide a flexible and powerful way to work on your project without interfering with each other's work, and they allow you to experiment with new ideas and features without worrying about breaking your main codebase.

When you create a branch in Git, you can make changes to the codebase and commit those changes to the branch, without affecting the **master** branch or any other branches that may exist. This allows you to experiment with new features or ideas without risking the stability of the main codebase.

Once you're happy with the changes you've made on a branch, you can merge those changes back into the **master** branch (or another branch), which incorporates the changes from the branch into the main codebase.

Here to complete the merge, the pull request concept comes into the picture. The purpose of a pull request is to propose changes to a codebase managed by Git. Pull requests allow developers to share their changes with others and get feedback before merging their changes into the main codebase. Pull requests facilitate collaboration, improve code quality, and provide transparency in the codebase.

When a developer creates a pull request, it creates a discussion thread where other developers can review the changes, ask questions, and suggest improvements. This helps catch bugs and improve the quality of the code. Pull requests also provide a way to track changes to the codebase over time, making it easier to understand the evolution of the codebase and revert changes if necessary.

Git branches are lightweight and easy to create, which makes them an essential part of Git workflows. They allow developers to work on multiple features or bug fixes simultaneously, without worrying about conflicts or impacting the main codebase. Git branches also enable collaboration by allowing multiple developers to work on the same codebase simultaneously, each on their own branch.

How to create a new Branch and Checkout to it

To create a new branch in Git, you can use the git branch command followed by the name of the new branch you want to create. Here are the steps to create a new branch:

1. Open your terminal or command prompt and navigate to the directory of the Git repository where you want to create the new branch.
2. Use the **git branch** command followed by the name of the new branch you want to create. For example, to create a new branch named **new-feature**, you would use the following command:

```
JavaScript
git branch new-feature
```

3. Once the new branch is created, you can switch to it using the **git checkout command**. For example, to switch to the **new-feature** branch, you would use the following command:

```
JavaScript
git checkout new-feature
```

Alternatively, you can create a new branch and switch to it in one command using the **-b** option with the **git checkout command**. For example, to create a new branch named **new-feature** and switch to it, you would use the following command:

```
JavaScript
git checkout -b new-feature
```

That's it! You've now created a new branch in Git and switched to it. You can make changes to the codebase on this branch and commit those changes to the branch without affecting the main branch (usually called **master** branch). Once you're done with the changes, you can merge them back into the main branch or any other branch as needed.

How to Merge Branch

To merge a branch in Git, you need to follow these basic steps:

1. Switch to the branch that you want to merge into. For example, if you want to merge a branch named **new-feature** into the master branch, you would switch to the **master** branch:

```
JavaScript
git checkout master
```

2. Use the **git merge** command followed by the name of the branch that you want to merge into the current branch. For example, to merge the **new-feature** into the **master** branch, you would use the following command:

```
JavaScript
git merge new-feature
```

3. Git will attempt to merge the changes from the new-feature into the master branch. If there are any conflicts between the changes made on both branches, Git will prompt you to resolve the conflicts manually. You can use a text editor or a Git merge tool to resolve the conflicts.

4. Once the conflicts are resolved, save the changes and commit them to complete the merge:

Note that when you merge a branch, the changes made on that branch are applied to the current branch. This means that any changes made on the merged branch will be included in the current branch, and the merged branch will no longer be needed.

Merge Conflict

A merge conflict is a situation that occurs when two or more branches of code have conflicting changes that Git cannot automatically resolve when merging them together. It typically happens when two or more developers make changes to the same file or code block, and those changes conflict with each other.

When a merge conflict occurs, Git will alert the developer that there is a problem and cannot automatically merge the changes. The developer must then resolve the conflict by manually editing the conflicting code and choosing which changes to keep or discard. Once the conflict is resolved, the developer must commit the changes and complete the merge.

Resolving merge conflicts can be time-consuming and require careful attention to detail to ensure that the codebase remains stable and functional. Effective communication and collaboration between developers can help avoid merge conflicts and minimize their impact when they occur.

Example of Merge Conflict and it's Solution

Suppose two developers, Dev-one and Bob, are working on the same project and both make changes to the same file at the same time. Dev-one changes the following lines in a file called example.txt:

```
JavaScript
This is some sample text.
Dev-one  added this line.
```

Meanwhile, Dev-two changes the same lines to

```javascript
This is some sample text.
Dev-two added this line.
```

2. Use the **git merge** command followed by the name of the branch that you want to merge into the current branch. For example, to merge the **new-feature** into the **master** branch, you would use the following command:

```javascript
This is some sample text.
Dev-two added this line.
```

When Dev-one tries to merge her changes into the main branch, Git detects a conflict because Dev-two has made conflicting changes to the same lines of code. Here's how Dev-one can use a text editor or a Git merge tool to resolve the conflict:

Text Editor Method:

1. Dev-one runs git merge main to attempt to merge her changes into the main branch.
2. Git detects a conflict and displays a message indicating which file has a conflict.
3. Dev-one opens example.txt in a text editor and finds the section of code that has a conflict, which now looks like this:

```javascript
This is some sample text.
<<<<<<< HEAD
Dev-one  added this line.
=======
Dev-two added this line.
>>>>>>> main
```

4. Dev-one manually edits the code to remove the conflict and ensure that both her changes and Dev-two's changes are included, for example,

```javascript
This is some sample text.
Dev-one  added this line.
Dev-two added this line.
```

5. Dev-one saves the file and stages the changes with git add example.txt.

6. Dev-one completes the merge with the git commit.

Git Merge Tool Method

1. Dev-one runs git merge main to attempt to merge her changes into the main branch.
2. Git detects a conflict and prompts Dev-one to use a merge tool.
3. Dev-one runs git mergetool to open the merge tool.
4. The merge tool displays the conflicting sections of code side by side, with options to select which changes to keep or discard. In this case, the tool might display something like this:

```
JavaScript
This is some sample text.
Dev-one  added this line.     |       Dev-two added this line.
                           >>
```

5. Dev-one selects the changes to keep and saves the merge result, for example,

```
JavaScript
This is some sample text.
Dev-one  added this line.
Dev-two added this line.
```

6. Dev-one stages the changes with git add example.txt.
7. Dev-one completes the merge with the git commit.

By using a text editor or a Git merge tool to resolve conflicts, developers can ensure that the codebase remains stable and functional even when multiple people are making changes at the same time.