Git Branch

Every developer has done it—you make a mistake but fail to notice it until you've already written a lot of great code in the meantime. If you want to revert your changes to fix the error, you risk losing all that work!

Luckily, a powerful tool called **Git branching** helps prevent scenarios like this and makes it easier to fix problems when they do arise. Branching allows team members to work on separate features at the same time while minimizing conflicts. For instance, you can easily switch to a colleague's branch to help them with their code or ask colleagues to review your own code before merging it with the main codebase.

You'll likely use branching every day in your development career. Whenever you need to fix a bug or build a new feature, you'll create a new branch—an isolated environment where you can write and test code without messing up the main codebase. This way, you always have the option to return to an earlier version without losing everything. Once the work is complete, you can merge the branch with the main codebase and move on!

In this activity, we'll learn the concept of branching by using Git commands to create a new branch, complete a feature in the new branch, merge the branch with `main`, and close the branch when finished.

Instructions

- Navigate to a directory where you usually store your coding files using your command line.

- Create a new directory called `git_branch_demo`. We could create a new repo in Github as well, but because we're working locally for this activity, we can skip that step and instead issue the following command:

  ```
  mkdir git_branch_demo
  ```

- Now use the `cd` command to navigate into the newly created directory. Initialize an empty Git repo in the directory with the following command:

  ```
  cd git_branch_demo
  git init .
  ```

- Open the `git_branch_demo` directory in VS Code.

- Now that we're ready to create the first feature, we need to create a new branch. Remember, the goal is to avoid working in the `main` branch, so that we can make mistakes on the new feature without damaging code that already works.

- Now before we can create a new branch, we first need to actually commit something to our newly initiated `main` branch. Let's go ahead and create a top-level README. Top-level here means its at the root of the directory.

```
touch README.md
git add .
git commit -m "creating a top level readme"
```

- The Git command for creating a new named branch in your repo is `git checkout -b feature/<feature-name>`. Pick a name that is associated with the feature that you will be working on. `feature/` reminds us that each branch is dedicated to a specific feature, while `<feature-name>` is the name of the feature. In this case, we'll be creating a JS file, so let's call it `create-js-file`, as follows:

  ```
  git checkout -b feature/create-js-file
  ```

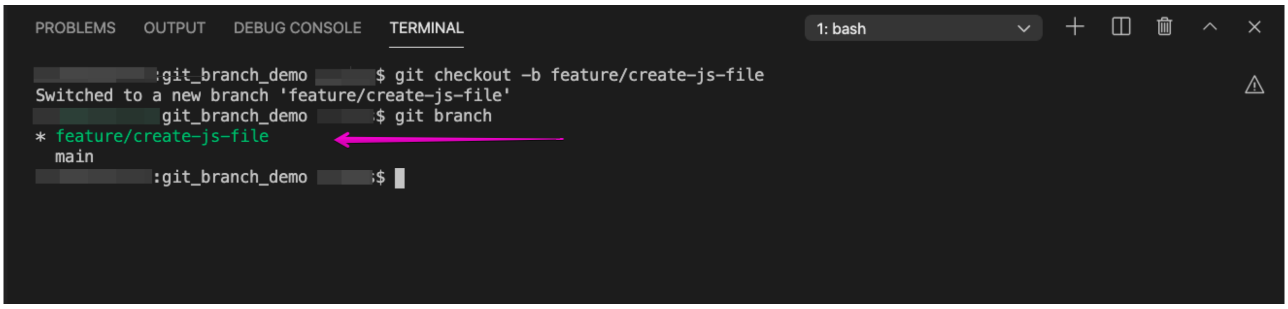
- This command will create the new branch, and then `checkout` to it. So you should now be on the `feature/create-js-file` branch.

- The Git command `git branch` allows us to see a list of existing branches. Run the following command so that we can confirm that the `feature/create-js-file` branch was created:

  ```
  git branch
  ```

- You should see the `feature/create-js-file` branch and a `*` next to `main`, as shown in the following image:

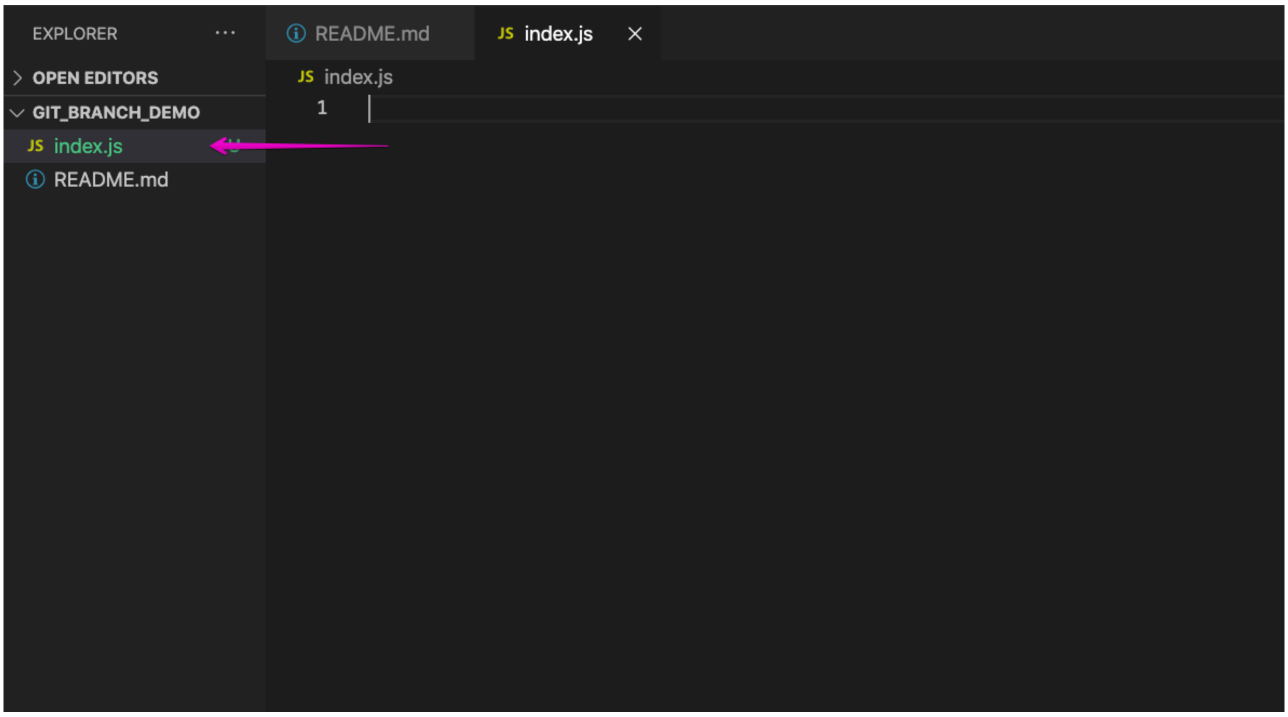
- We should now see a `*` by `feature/create-js-file`, letting us know this is the branch we are on, as shown in the following image:



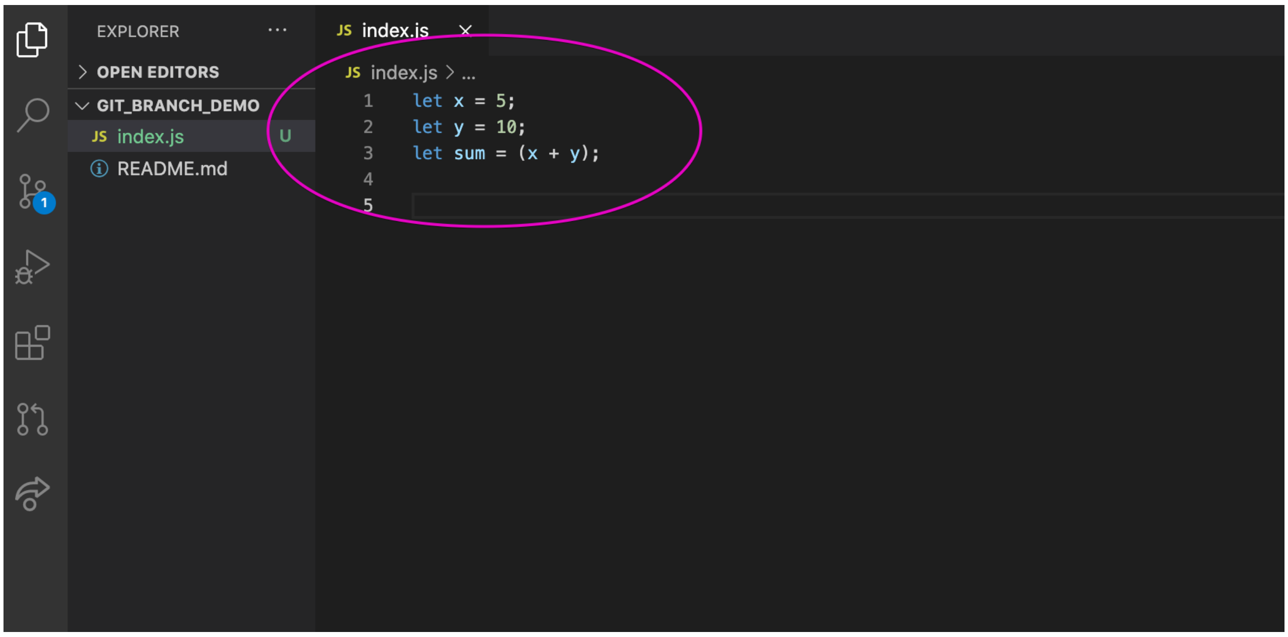
```
[redacted]:git_branch_demo [redacted]$ git checkout -b feature/create-js-file
Switched to a new branch 'feature/create-js-file'
[redacted]git_branch_demo [redacted]$ git branch
* feature/create-js-file
  main
[redacted]:git_branch_demo [redacted]$
```

> **Important:** We also have the option to create a branch and switch over to it at the same time by entering `git checkout -b <branch-name>`.

- Now that we're on the correct feature branch, we need to work on the feature. Create an `index.js` file in the `git_branch_demo` directory, as shown in the following image:



- Add some a few random variables to the `index.js` file, so that it looks something like the following image:



```
let x = 5;
let y = 10;
let sum = (x + y);
```

- Finally, add and commit the changes that you made, as follows:

```
git add .
git commit -m "Created index.js and added text to the file"
```

- Now that the feature is complete, you can merge the feature branch with `main`. First you need to switch back to `main` from `feature/create-js-file`. Remember, it's always a good idea to confirm that you're on the correct branch using the `git branch` command. See the following example:

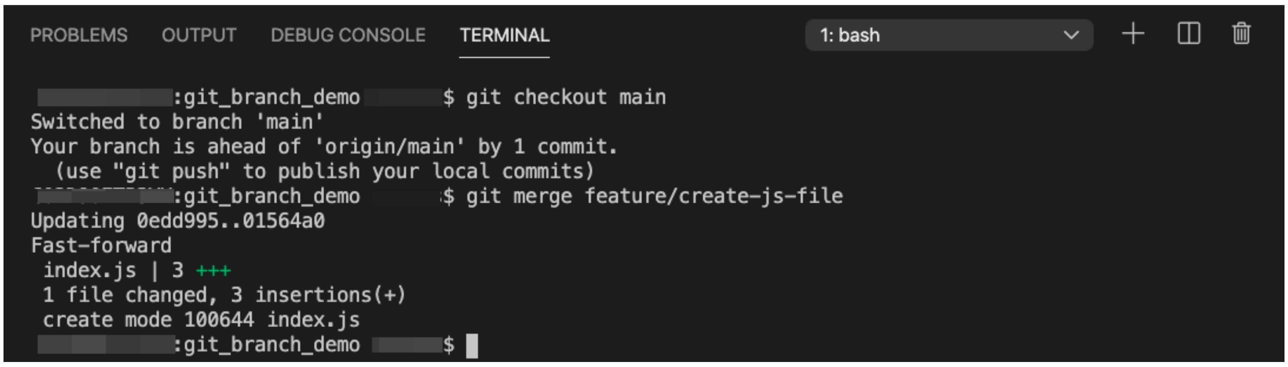
```
git checkout main
git branch
```

> **Important:** Git won't let you switch to a different branch until you have added, committed, and pushed any changes that you made to your feature branch. If you try to switch without pushing your code, Git will send you a reminder to push the changes that you made before switching branches.

- Once we're in `main`, notice that we no longer have a index.js file in the directory. What happened? `main` is currently behind `feature/create-js-file` and we still need to merge the feature branch with the main codebase. To merge, add the following code to the command line:

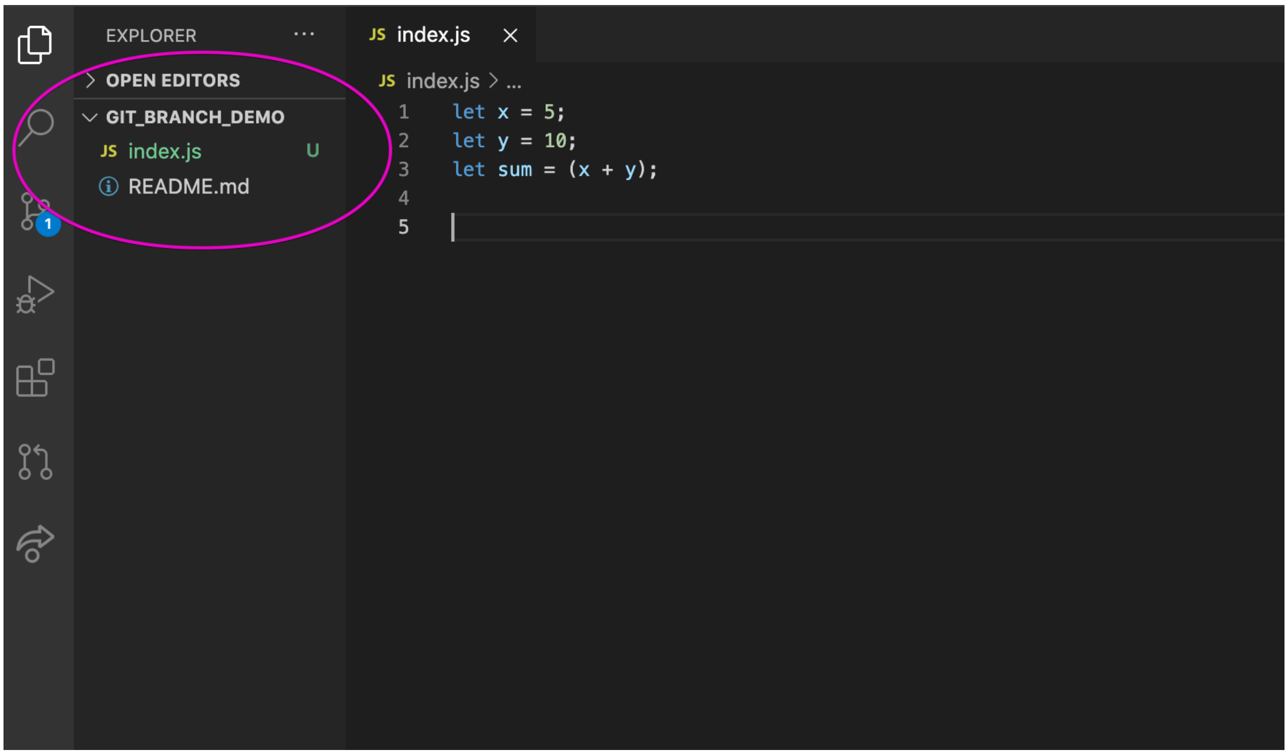
```
git merge feature/create-js-file
```

- If the merge is successful, we should see something similar to the following image:



```
[redacted]:git_branch_demo [redacted]$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
[redacted]:git_branch_demo [redacted]$ git merge feature/create-js-file
Updating 0edd995..01564a0
Fast-forward
 index.js | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 index.js
[redacted]:git_branch_demo [redacted]$
```

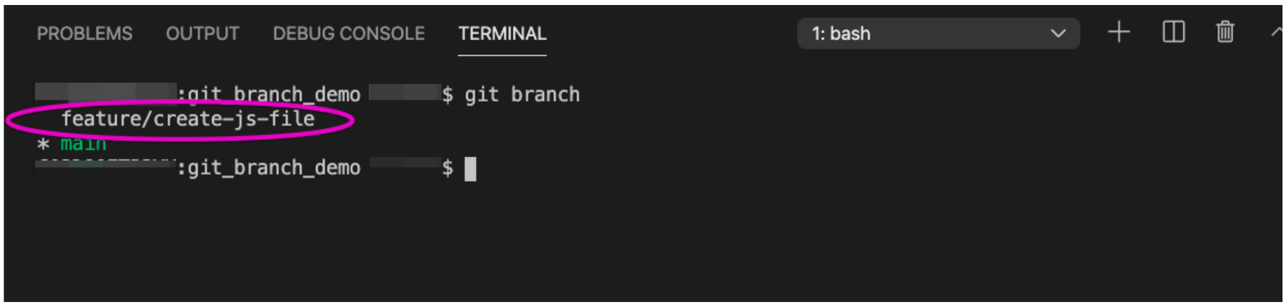
- Now the directory should include the `index.js` file that we created, along with whatever text we added to the file. We asked Git to merge the code that we wrote in the feature branch with the code that already existed in `main`. At this point, the new feature is a part of the main codebase, as you can see in the following image:



```
let x = 5;
let y = 10;
let sum = (x + y);

```

- We're all done with the feature, and the code that we wrote safely exists in `main`. But if you enter the `git branch` command, you'll see that `feature/create-js-file` still exists, as shown in the following image:

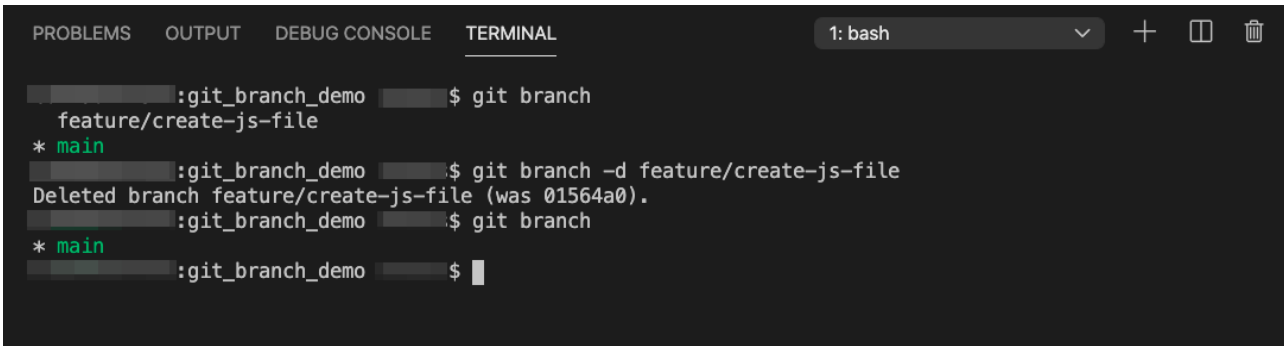


```
[redacted]:git_branch_demo [redacted]$ git branch
  feature/create-js-file
* main
[redacted]:git_branch_demo [redacted]$
```

- We could potentially generate a huge list of feature branches while working on a large project. So to avoid confusion and stay organized, it is good practice to close a branch once a feature is completed and merged. Because we're finished with this feature and the code is now included in `main`, we no longer need the isolated environment of that branch. We can always open another branch to fix future problems. But for now, we can safely close the feature branch by issuing the following command:

```
git branch -d feature/create-js-file
```

- When we run `git branch`, we should see something similar to the following image:



```
[redacted]:git_branch_demo [redacted]$ git branch
  feature/create-js-file
* main
[redacted]:git_branch_demo [redacted]$ git branch -d feature/create-js-file
Deleted branch feature/create-js-file (was 01564a0).
[redacted]:git_branch_demo [redacted]$ git branch
* main
[redacted]:git_branch_demo [redacted]$ 
```

- Congratulations, you've now completed your first branch lifecycle! You created an isolated environment on a new branch so that you could write and test code for a new feature, the `index.js` file. Once you finished adding text to `index.js`, you merged the feature branch with the main codebase on `main`. You then closed the feature branch, because you no longer needed to work on the `index.js` file.

Hints

- You'll come up with your own naming conventions for branches when you're working on your own project. Try to be descriptive but concise to help other developers (or your future self) understand what is happening in each branch.

- Popular naming conventions in the field include `feature/<feature-name>`, `issue/<issue-reference>`, etc. So it is a good idea to practice these conventions while you're learning.

- Avoid including your own name in the branch name, because multiple developers might end up working in the same branch. So, for example, instead of `git branch rachels-feature`, try using `git branch user-login-page`.

- More information can be found in these Atlassian tutorials on branching.
