

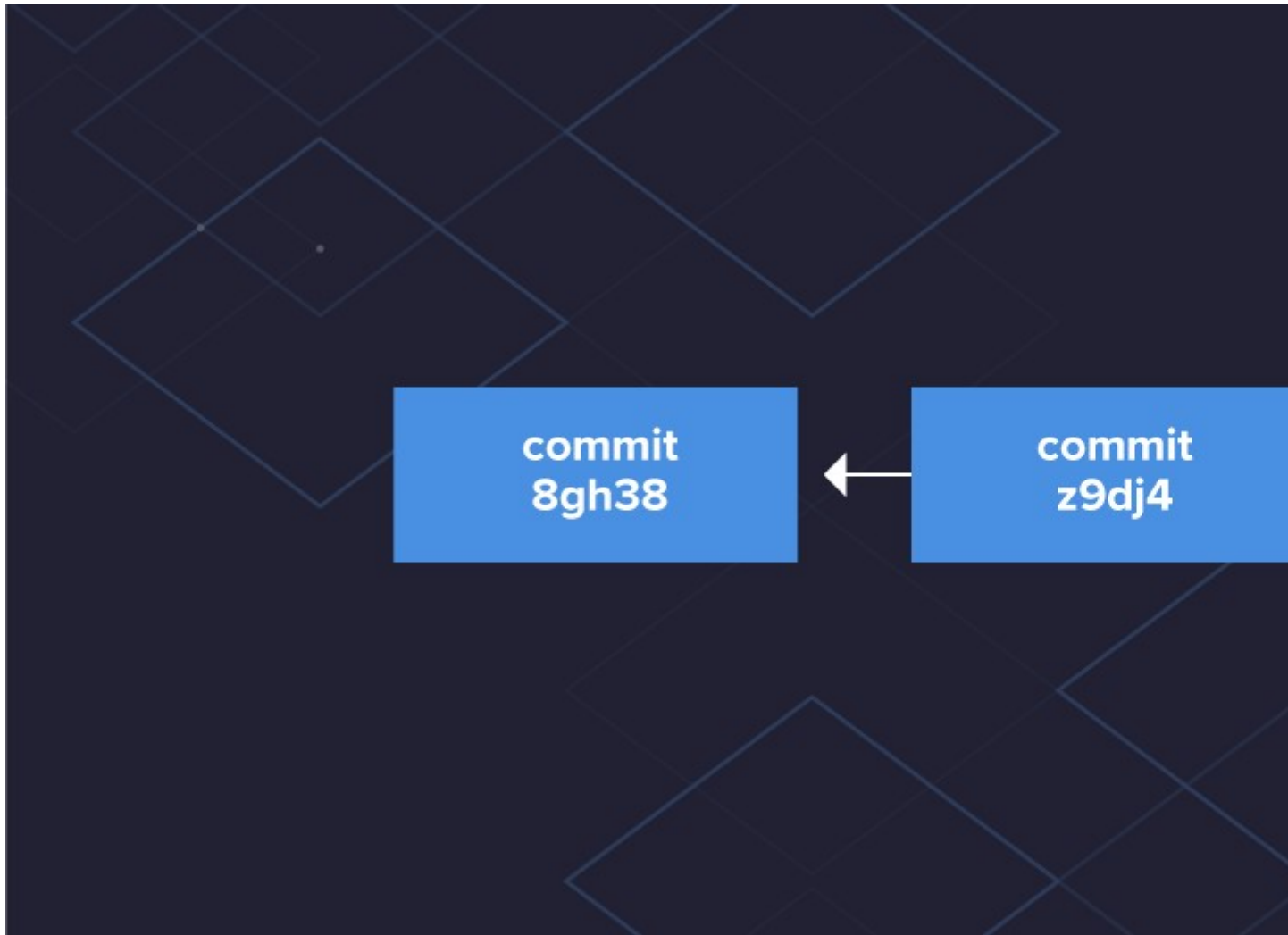
What is Git Branching?



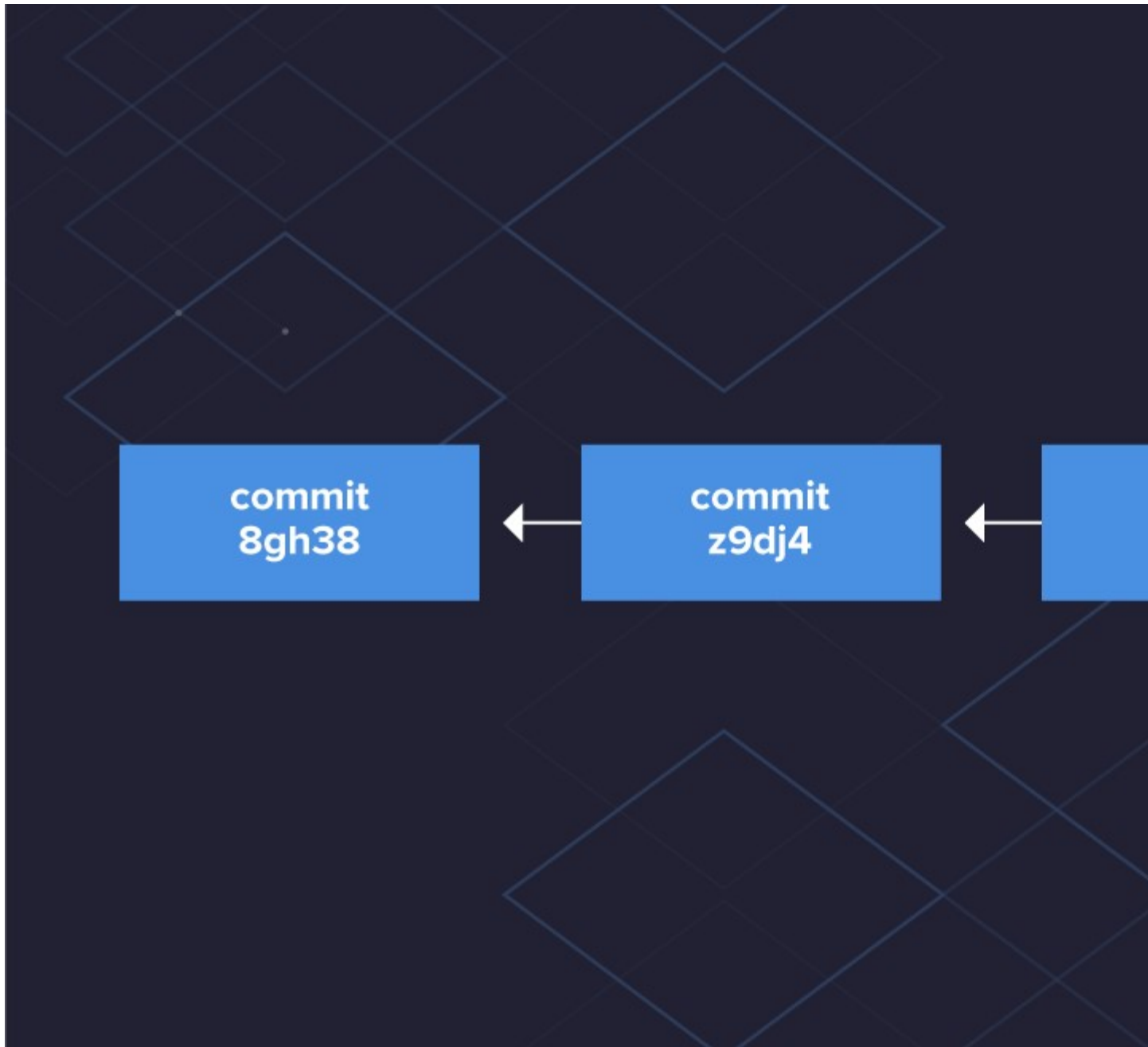
Git branching allows developers to diverge from the production version of code to fix a bug or add a feature. Developers create branches to work with a copy of the code without modifying the existing version. You create branches to isolate your code changes, which you test before merging to the main branch (more on this later).

There is nothing special about the main branch. It is the first branch made when you initialize a Git repository using the `git init` command.

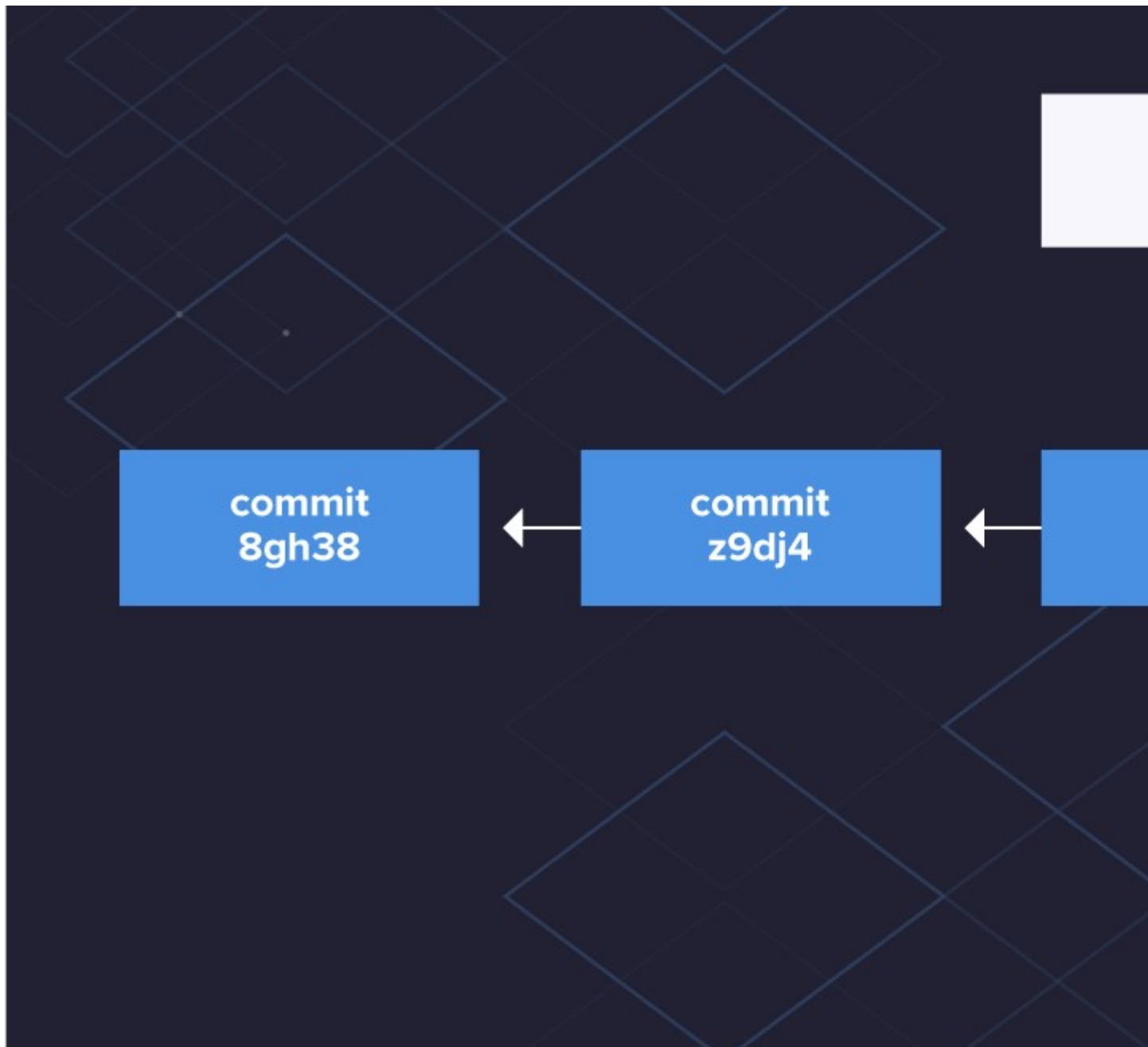
When you create a commit, Git identifies that snapshot of files with a unique SHA-1 hash. When you initially create a branch, Git creates a new pointer to the same commit the main branch is currently on. The diagram below shows both branches have the same snapshot of code at this point.



As you create commits in the new branch, Git creates new pointers to track the changes. The latest commits are now ahead of the main branch commits. As you continue to make commits, each branch keeps track of its version of files.



Git knows which branch you have checked out by using a special pointer called HEAD. When you create a new branch, Git doesn't immediately change the HEAD pointer to the new branch. You'll see HEAD in the tutorial when you create branches and view the commit log.



This branching function is what makes Git really powerful. Multiple people create separate branches to work on their code and merge their changes into the main branch. Branches are meant to be temporary and should be deleted when work is completed.

Branch Naming Strategies

Branch names can be anything you'd like. However, your organization or project may have standards outlined for branch naming. For example, naming the branch based on the person responsible for working on the branch and a description or work item:

- username/description
- username/workitem

You can name a branch to indicate the branch's function, like a feature, bug fix, or hotfix:

- bugfix/description
- feature/feature-name
- hotfix/description

Another branching strategy is having branches dedicated to the different development cycles, like feature or hotfix. As work items come up, you create a branch for that item from its respective branch. Yes, you can create branches from branches! Check out Option 4 below for an example.

How to Create a Branch in Git

Enough theory, let's create some branches! These examples will be using PowerShell 7 on a Windows 10 system; however, you can use any terminal that supports Git commands.

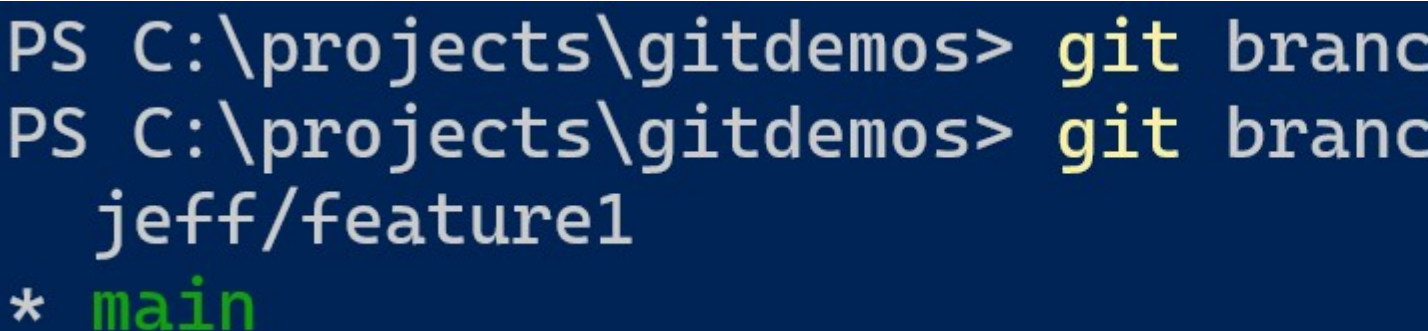
Option 1: Creating a Branch

To create a branch, use the `git branch` command followed by the name of the branch. After making the branch, use `git branch` again to view available branches.

Notice that creating a branch this way does not automatically switch to the new branch. Git uses an asterisk and a different colored font to identify which branch is active. This designation represents the HEAD pointer showing which branch is active.

```
git branch <branch name>
```

```
git branch
```



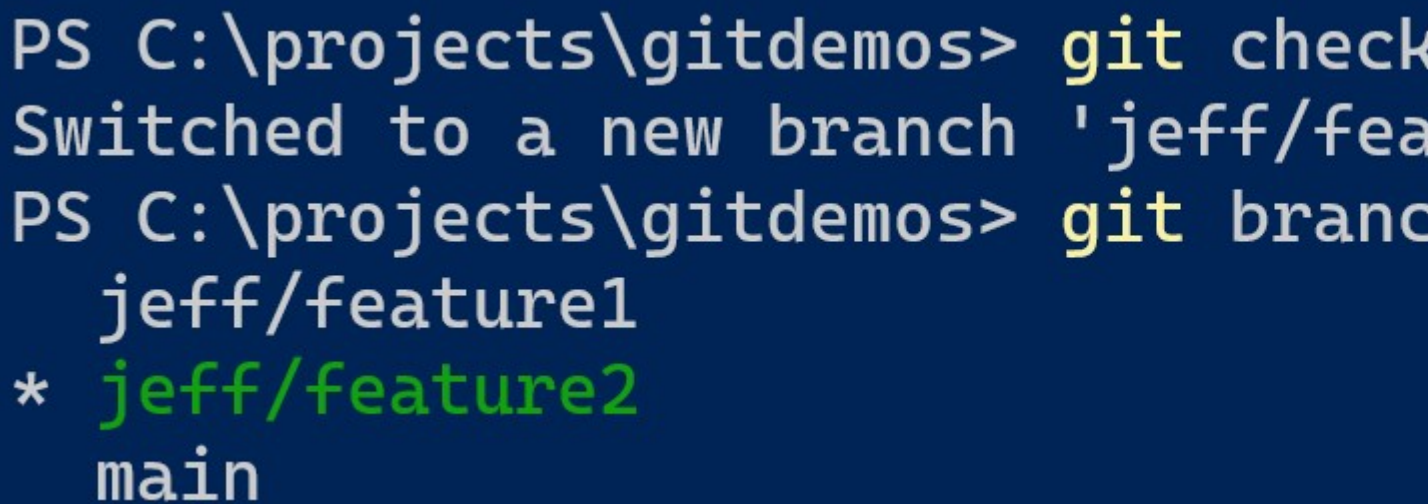
```
PS C:\projects\gitdemos> git branch
PS C:\projects\gitdemos> git branch
    jeff/feature1
* main
```

Option 2: Creating a Branch using Checkout

If you want to create a branch and checkout the branch simultaneously, use the `git checkout` command. The switch `-b` specifies the name of the branch. Note that after command completion, Git has moved HEAD to the new branch.

```
git checkout -b <branch name>
```

```
git branch
```

A terminal window with a dark blue background and white text. The first command is 'git checkout -b jeff/feature1', which is partially visible as 'git check' and 'Switched to a new branch \'jeff/fea'. The second command is 'git branch', which is partially visible as 'git bran'. The output shows three branches: 'jeff/feature1', '* jeff/feature2' (with an asterisk indicating it's the current branch), and 'main'.

```
PS C:\projects\gitdemos> git check  
Switched to a new branch 'jeff/fea  
PS C:\projects\gitdemos> git bran  
jeff/feature1  
* jeff/feature2  
main
```

Option 3: Creating a Branch from a Commit

You can create a branch from a previous commit on an existing branch. Remember, a commit is just a snapshot in time of the files in a repository. You create a branch from a commit if you want to work on a specific snapshot of the files.

Before creating the branch, you need the SHA-1 identifier of the commit. To find the identifier, use the `git log` command to view previous commits. Each commit will have a complete SHA-1 hash as the identifier. However, you only need the first few characters to identify the commit.

Next, use the same `git branch` command from Option 1 but append the commit identifier at the end. This example is using **40b4d7** from the second commit as the identifier.

Note the HEAD designator is on the main branch, which is the active branch. The other branches *jeff/feature1* and *jeff/feature2* point to the same commit when you created them earlier. Both point to the same snapshot as each branch has not had additional commits made to each one since creation.

```
git log  
git branch <branch name> <identifier>
```



```
PS C:\projects\gitdemos> git log
commit 3d7f42658e1409e9916f32baca
Author: Jeff Brown Tech <jeff@jeff
Date: Thu Mar 25 14:44:53 2021 -
```

modified code

```
commit 40d4b7ec951dac94cf00b7aece6
Author: Jeff Brown Tech <jeff@jeff
Date: Thu Mar 25 14:44:14 2021 -
```

add info about repo purpose

```
commit 123847d0712a7179a9f22c3f961
Author: Jeff Brown Tech <jeff@jeff
Date: Thu Mar 25 14:35:11 2021 -
```

init commit

```
PS C:\projects\gitdemos> git branch
```

```
PS C:\projects\gitdemos> git branch
```

jeff/feature1

jeff/feature2

jeff/main

Option 4: Creating a Branch from Another Branch

If you use branches dedicated to hotfixes or features, you create branches from these other branches to work on the item. Creating a branch from another branch is no different from creating from the main branch. You just need to specify the name of the other branch as the starting point. This example shows creating the *feature4* branch from the develop branch.

```
git checkout -b feature4 develop
```

Option 5: Download Branch from Remote Repository

While you have a local copy of a repository to work with, so do other developers. These developers will have branches they are working on, and they can push their branches to a remote repository.

Along the way, you may need to work on another branch that isn't local on your system. You can pull or download specific branches from a remote repository to use on your system.

In a central repository hosted in GitHub, notice the branches available are the same ones on the local system (main, feature1, feature2, and hotfix1). However, another developer named Maggie has a branch for hotfix2 that is not on the local system. Maggie requests your assistance working on a hotfix, so you need to download this branch to your system.

JeffBrownTech / gitdemos

<> Code

! Issues

🔗 Pull requests

▶ Actions

Overview

Active

Stale

Default branch

`main` Updated 6 hours ago by JeffBrownTech

Active branches

`maggie/hotfix2` Updated 6 hours ago by JeffBrownTech

`jeff/hotfix1` Updated 6 hours ago by JeffBrownTech

`jeff/feature2` Updated 6 hours ago by JeffBrownTech

`jeff/feature1` Updated 6 hours ago by JeffBrownTech

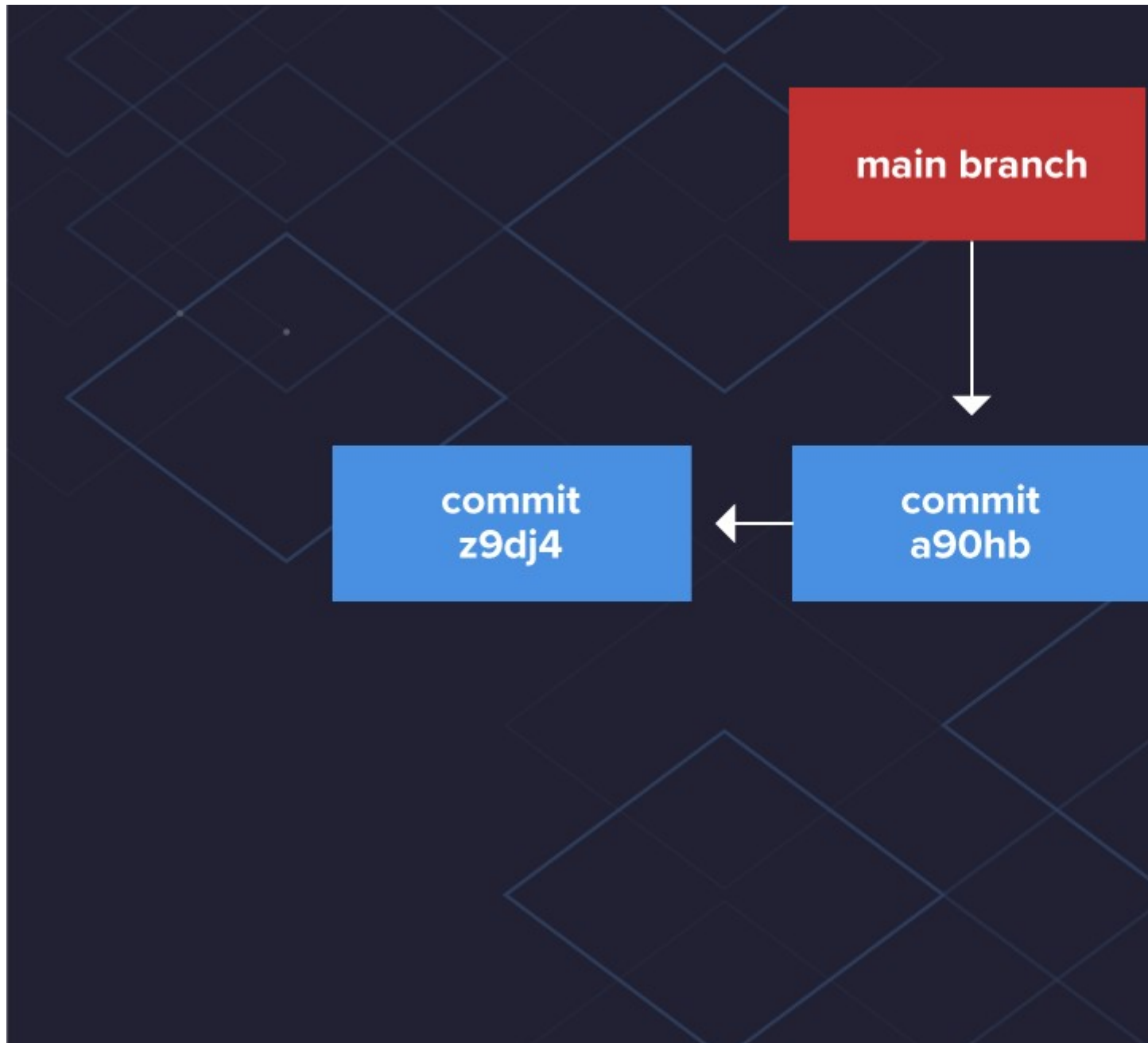
To retrieve the branch from the remote repository, use `git pull` against the origin and specify the name of the branch. If you check available local branches, the new branch doesn't appear automatically. However, you can check out the branch and begin working on this new branch.

```
git pull origin <branch name>
git branch
git checkout <branch name>
git branch
```

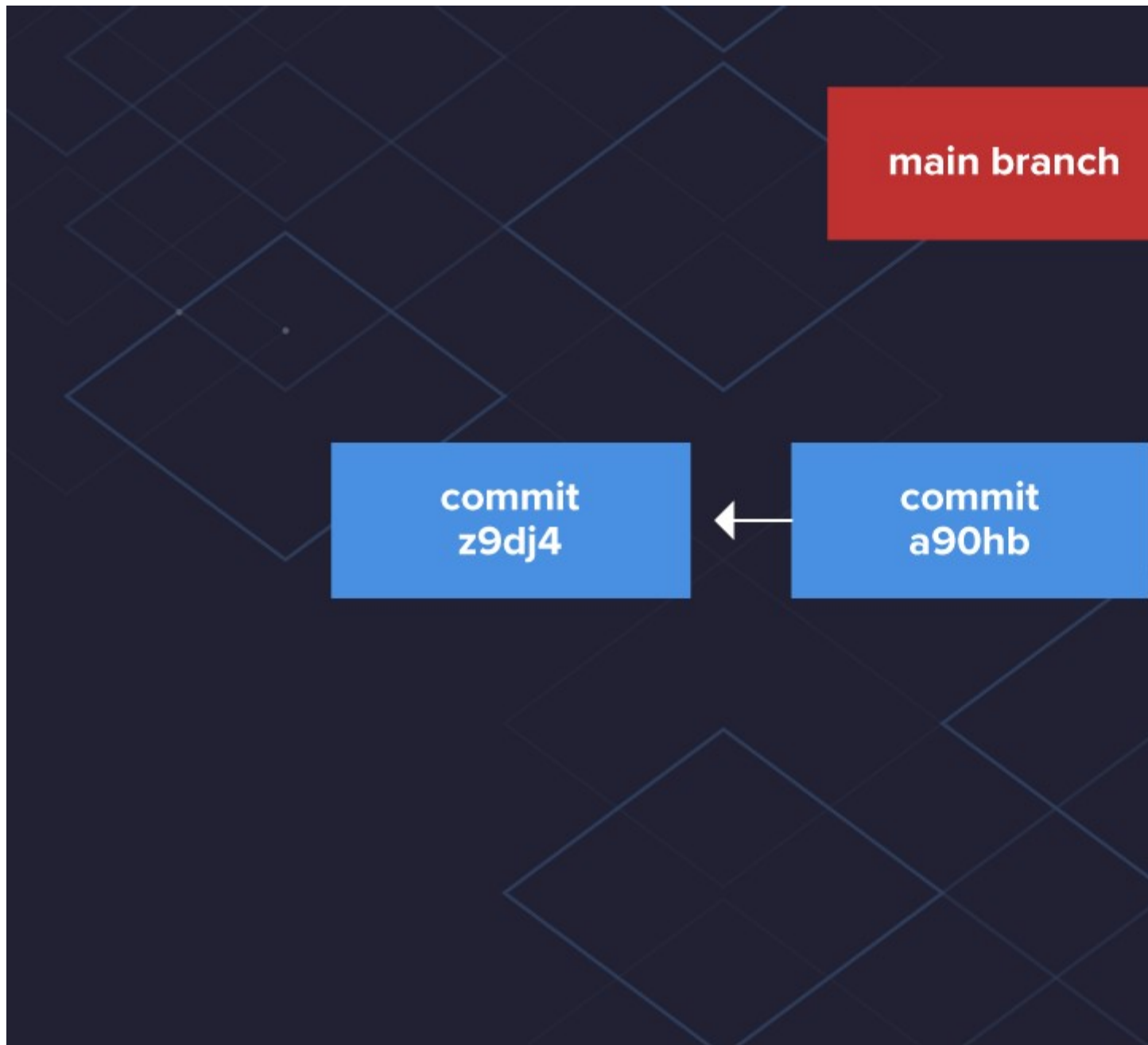
```
PS C:\projects\gitdemos> git pull
From github.com:JeffBrownTech/gitdemos
* branch                maggie/hotfix2
Already up to date.
PS C:\projects\gitdemos> git branch
  jeff/feature1
  jeff/feature2
  jeff/hotfix1
* main
PS C:\projects\gitdemos> git checkout -b maggie/hotfix2
Switched to a new branch 'maggie/hotfix2'
Branch 'maggie/hotfix2' set up to track the new branch 'maggie/hotfix2' from 'origin'.
PS C:\projects\gitdemos> git branch
  jeff/feature1
  jeff/feature2
  jeff/hotfix1
* maggie/hotfix2
  main
```

Merging Branches

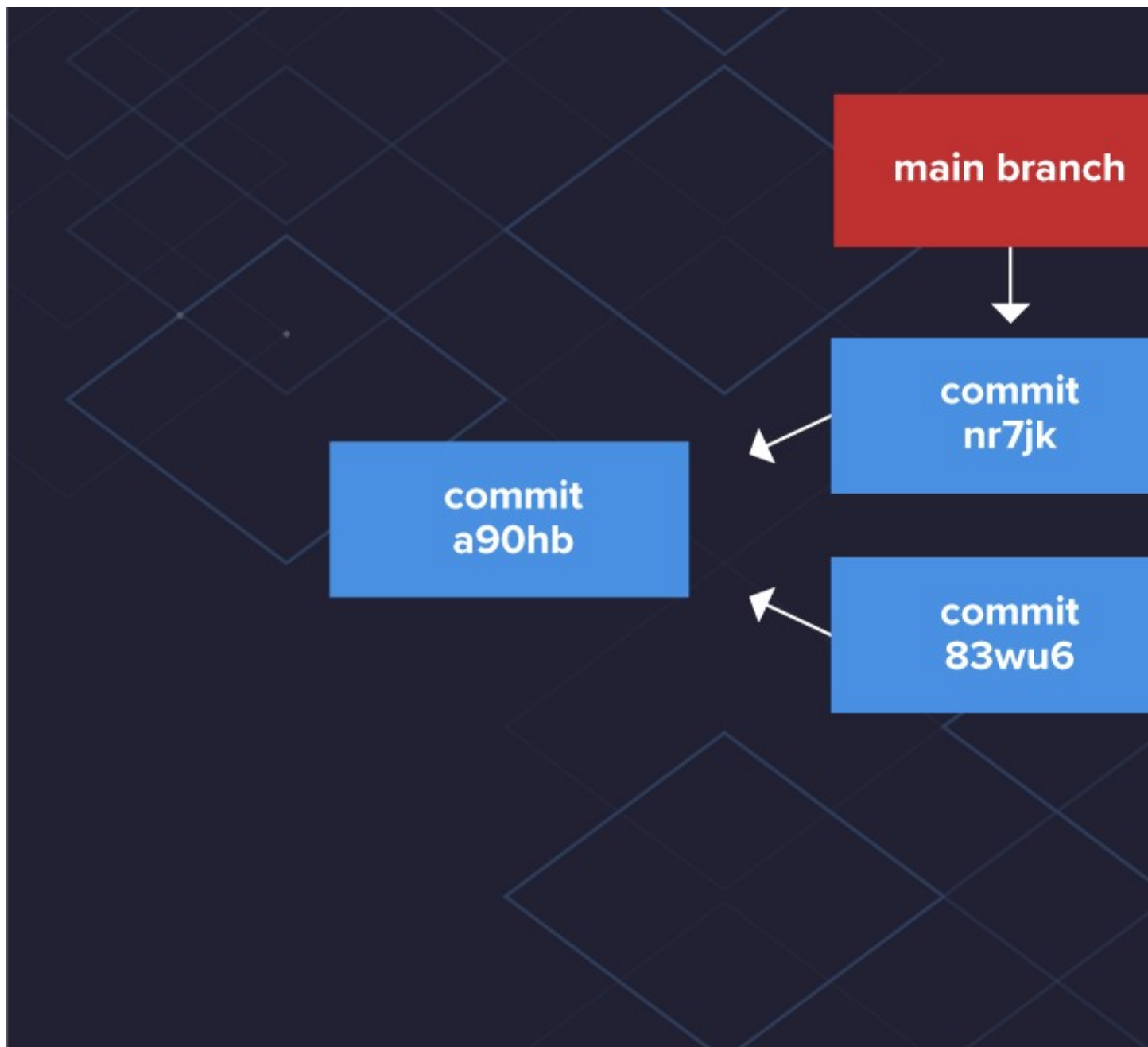
Once you've completed work on your branch, it is time to merge it into the main branch. Merging takes your branch changes and implements them into the main branch. Depending on the commit history, Git performs merges two ways: fast-forward and three-way merge. Let's examine both of these based on the branches and commit history in the following diagram.



When you merge the *hotfix* branch into the *main* branch, Git will move the main branch pointer forward to *commit nr7jk*. Git does this because the *hotfix branch* shares a direct ancestor commit with the *main* branch and is directly ahead of its commit. This commit is a **fast-forward** merge.

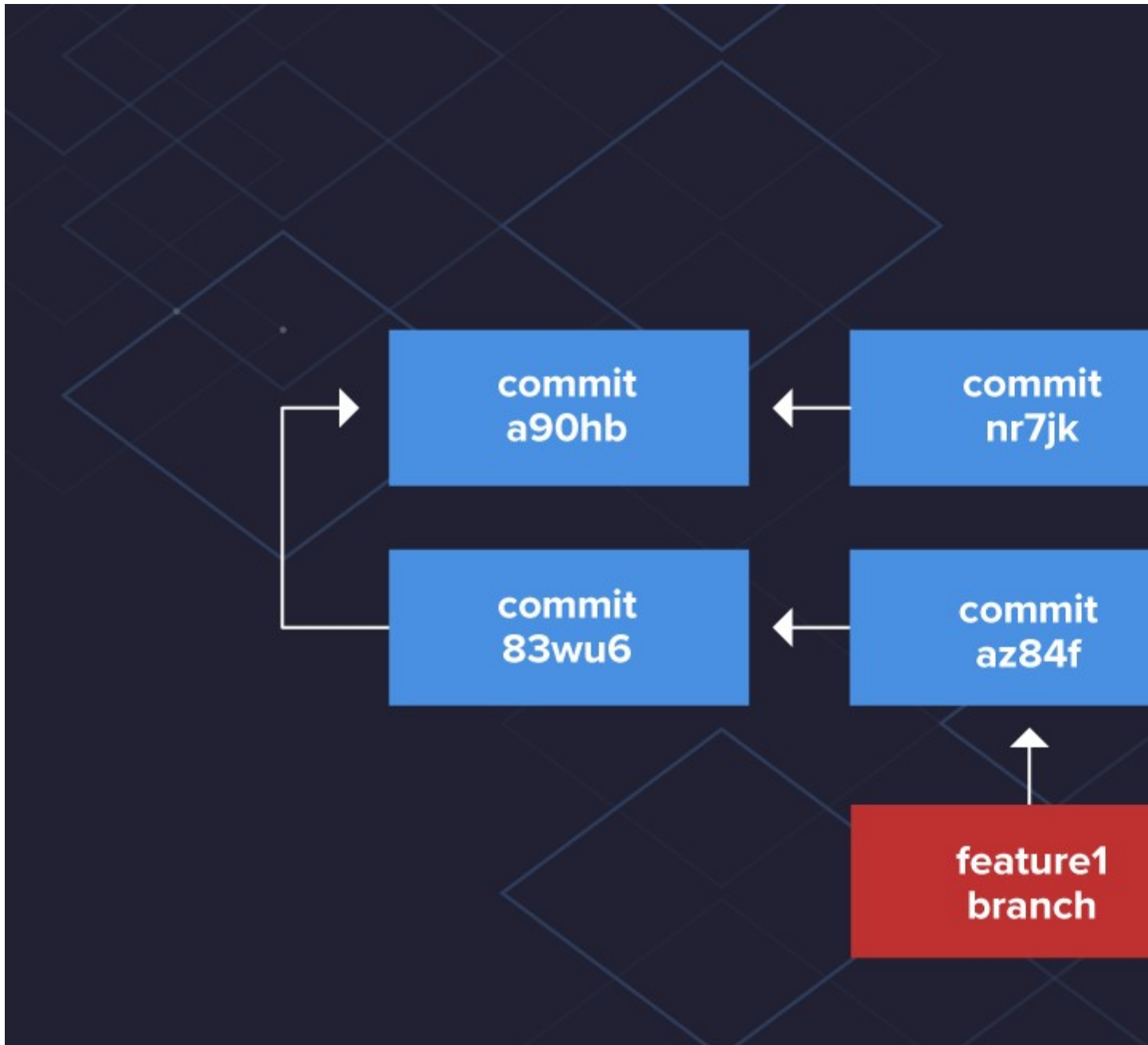


Once you merge the *hotfix branch*, continue working on the *feature1* branch. As you continue making commits on the *feature1* branch, the commit history diverges.



Git is unable to move the pointer to the latest commit like in a fast-forward commit. To bring the *feature1* branch into the *main* branch, Git performs a **three-way merge**. Git takes a snapshot of three different commits to create a new one:

- The common commit both branches share (a90hb)
- The latest commit of the branch (az84f)
- The commit of the branch to merge into (nr7jk)



Merging Branches in a Local Repository

To merge branches locally, use `git checkout` to switch to the branch you want to merge into. This branch is typically the *main* branch. Next, use `git merge` and specify the name of the other branch to bring into this branch. This example merges the *jeff/feature1* branch into the *main* branch. Note that this is a **fast-forward** merge.

```
git checkout main
git merge jeff/feature1
```

```
PS C:\projects\gitdemos> git check
Already on 'main'
Your branch is up to date with 'or
PS C:\projects\gitdemos> git merge
Updating 3d7f426..4c0ec58
Fast-forward
 README.md | 2 ++
1 file changed, 2 insertions(+)
```

Work continues on the main and other branches, so they no longer share a common commit history. Now a developer wants to merge the *jeff/feature2* branch into the main branch. Instead, Git performs a three-way (or recursive) merge commit.

```
git checkout main
git merge jeff/feature2
```

```
PS C:\projects\gitdemos> git check
Switched to branch 'main'
PS C:\projects\gitdemos> git merge
Merge made by the 'recursive' stra
README.md          | 2 ++
projectfile.txt    | 1 +
2 files changed, 3 insertions(+)
create mode 100644 projectfile.tx
```

Merging Branches to Remote Repository

If you create a branch in your local repository, the remote repository is not aware of the branch's existence. Before you can push the branch code in the remote repository, you set the remote repository as the upstream branch using the `git push` command. This command simultaneously sets the upstream branch and pushes the branch contents to the remote repository.

```
git push --set-upstream origin <branch name>
```



```
PS C:\projects\gitdemos> git push
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 8 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (18/18), 1.5 MiB | 1.5 MiB/s, done.
Total 18 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote:
remote: Create a pull request for 'jeff/bugfix1' on GitHub by visiting
remote:      https://github.com/JeffBrownTech/gitdemos/pull/new/jeff/bugfix1
remote:
To github.com:JeffBrownTech/gitdemos
 * [new branch]      jeff/bugfix1 -> jeff/bugfix1
Branch 'jeff/bugfix1' set up to track remote branch 'jeff/bugfix1' from 'origin'
```

Merging Main into a Branch

While you are working on your branch, other developers may update the *main* branch with their branch. This action means your branch is now out of date of the *main* branch and missing content. You can merge the *main* branch into your branch by checking out your branch and using the same `git merge` command.

```
git checkout <branch name>
git merge main
```



```
PS C:\projects\gitdemos> git check
Switched to branch 'jeff/feature2'
PS C:\projects\gitdemos> git merge
Updating 5020537..638b7ae
Fast-forward
 projectfile1.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 projectfile1.t
```