



Week 3 : SOFTWARE DEVELOPMENT TOOLS AND ENVIRONMENTS

# **Week 3**

## **Branches and Working with Others**

## Week 3 Branches and Working with Others

- **Review:**

- So far we've learned how to create repositories, add changes to the stage, and commit them to the repository.
- We've also learned how to push and pull code back and forth from local machines to remote branches on GitHub.

# Week 3 Branches and Working with Others

- **Week 3**

- Today it's time to learn about a critical concept in Git: **branches**.
- Branches allow us to organize a repository and split it apart so multiple people can work on it or so a solo developer can work on different aspects of a project on a separated work.

# Week 3 Branches and Working with Others

- **Week 3 Topics:**

- Master/Main Branch and Branches
- Understanding HEAD
- Git Branch Commands:
  - **git branch, git switch, git checkout**
- Delete or Rename Branch
- Merging Branches and Conflicts
- Using **git diff**
- Exercise and Solution

**Let's get started!**

# **Week 3**

## **Branches**

## Week 3 Branches and Working with Others

- Let's review what our current commit process looks like...



# Week 3 Branches and Working with Others

- **Commit Process**

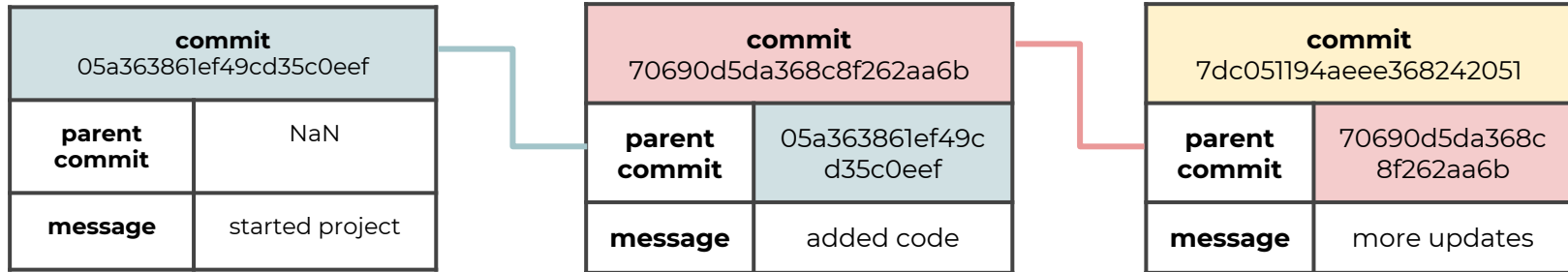
- As we create commits, we are linking to a parent commit, showing the log of the commit history.

commit 05a363861ef49cd35c0eef	
parent commit	NaN
message	started project

# Week 3 Branches and Working with Others

- **Commit Process**

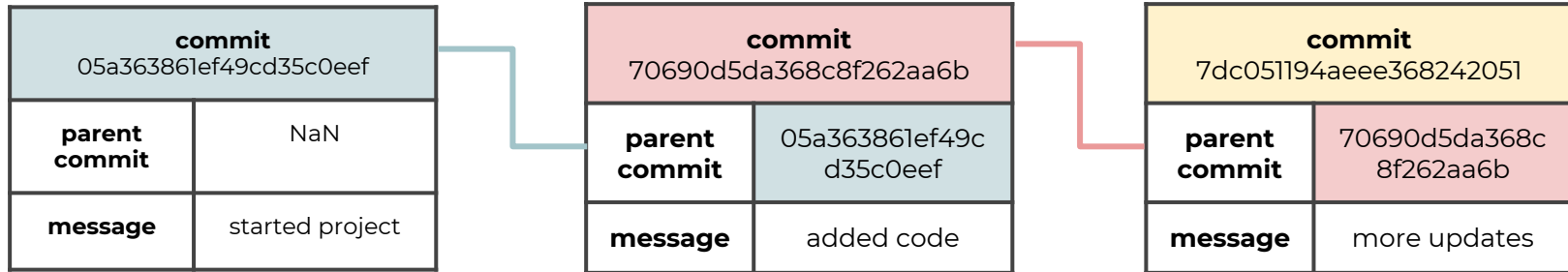
- As we create commits, we are linking to a parent commit, showing the log of the commit history.



# Week 3 Branches and Working with Others

- **Commit Process**

- As we need incorporate the workflows of others or be able to focus on new updates without breaking old code, we need **branches**.



# Week 3 Branches and Working with Others

- **Branches**

- A branch represents an independent line of development.
- Branches serve as an abstraction for the edit/stage/commit process.
- They are a way to request a brand new working directory, staging area, and project history.

## Week 3 Branches and Working with Others

- **Branches**

- Branches are just pointers to commits.
- When you create a branch, all Git needs to do is create a new pointer, it doesn't change the repository in any other way.
- Let's explore why branches are useful for workflows...

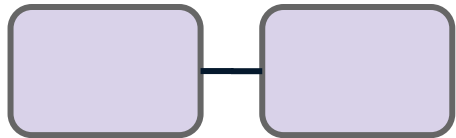
# Week 3 Branches and Working with Others

- **Branches**



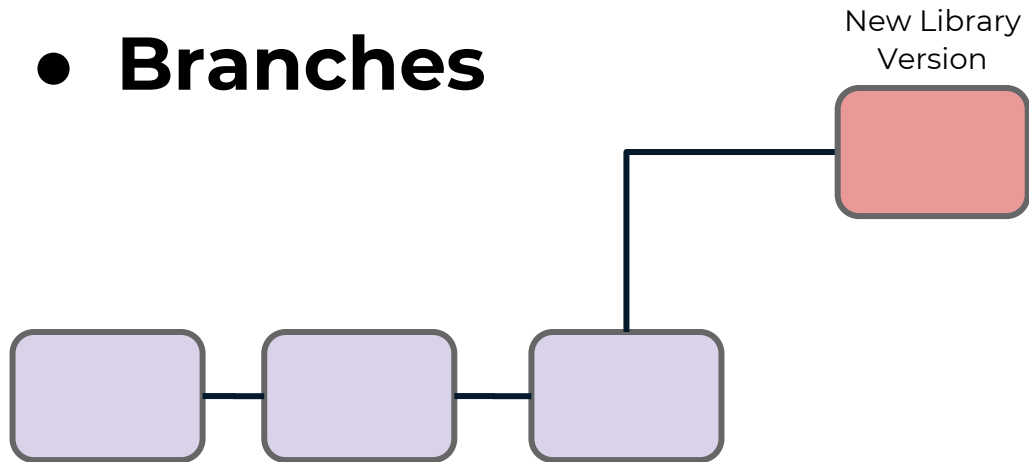
# Week 3 Branches and Working with Others

- **Branches**



# Week 3 Branches and Working with Others

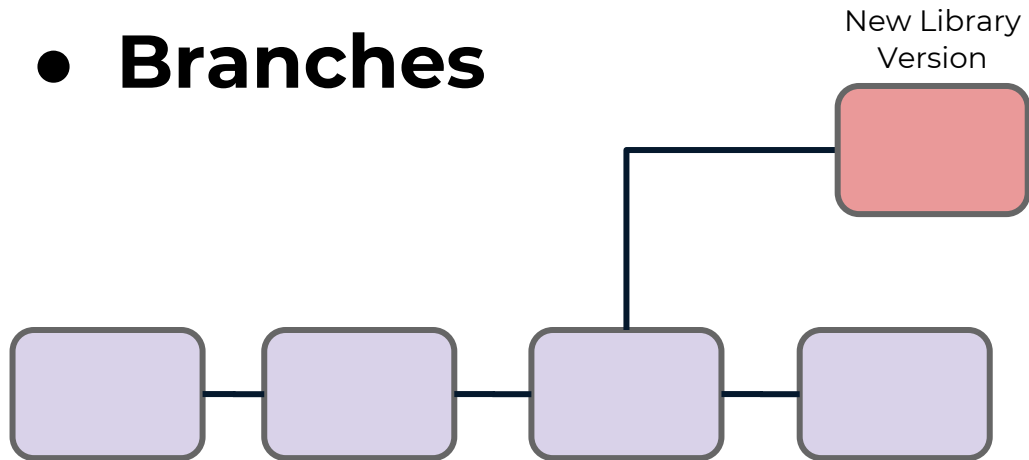
- **Branches**





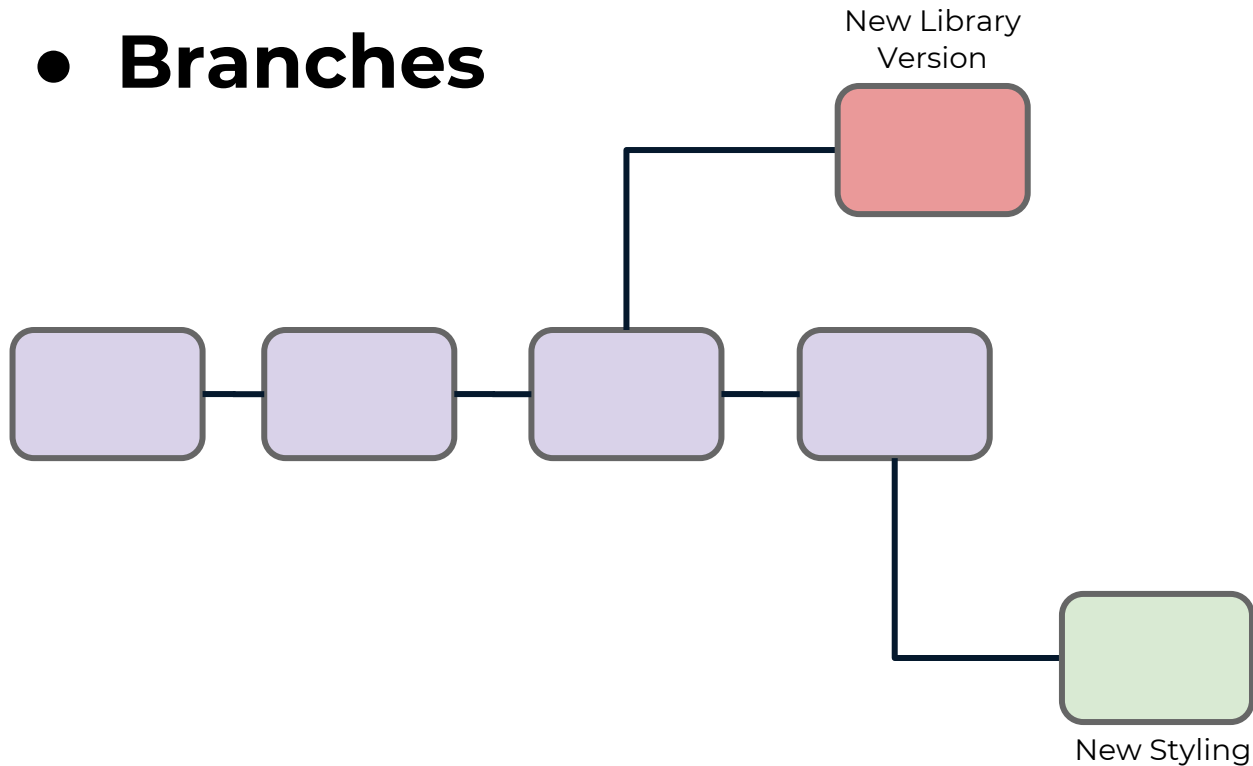
# Week 3 Branches and Working with Others

- **Branches**



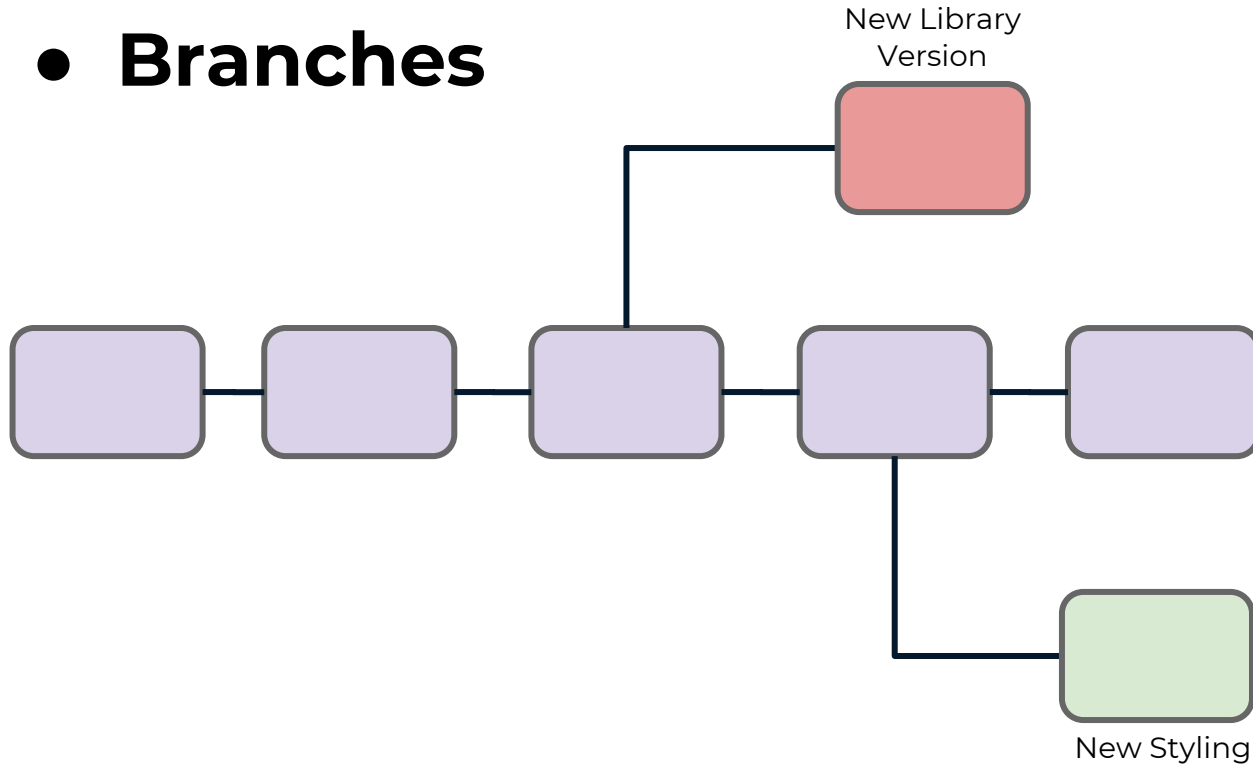
# Week 3 Branches and Working with Others

- **Branches**



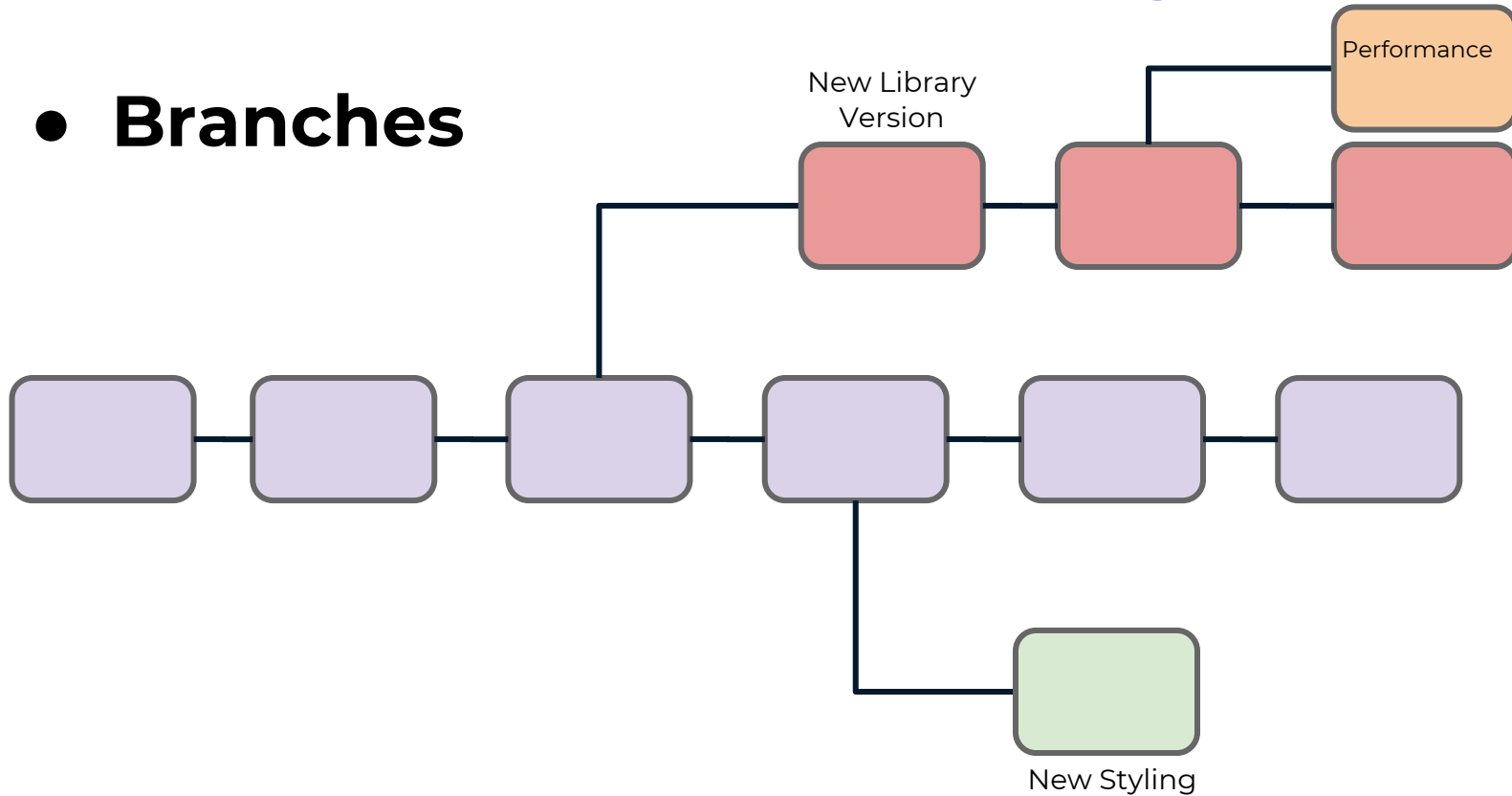
# Week 3 Branches and Working with Others

- **Branches**



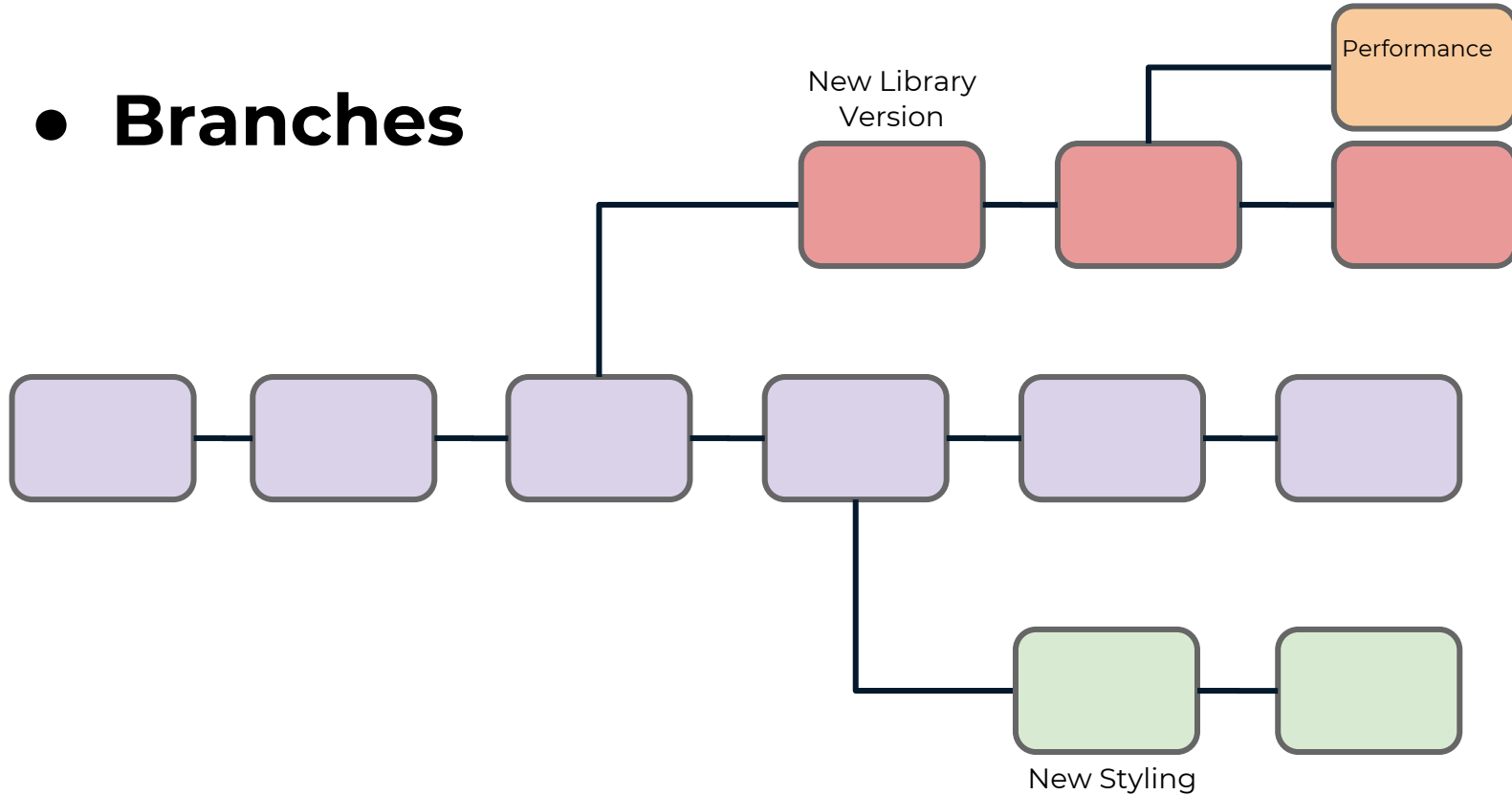
# Week 3 Branches and Working with Others

- **Branches**



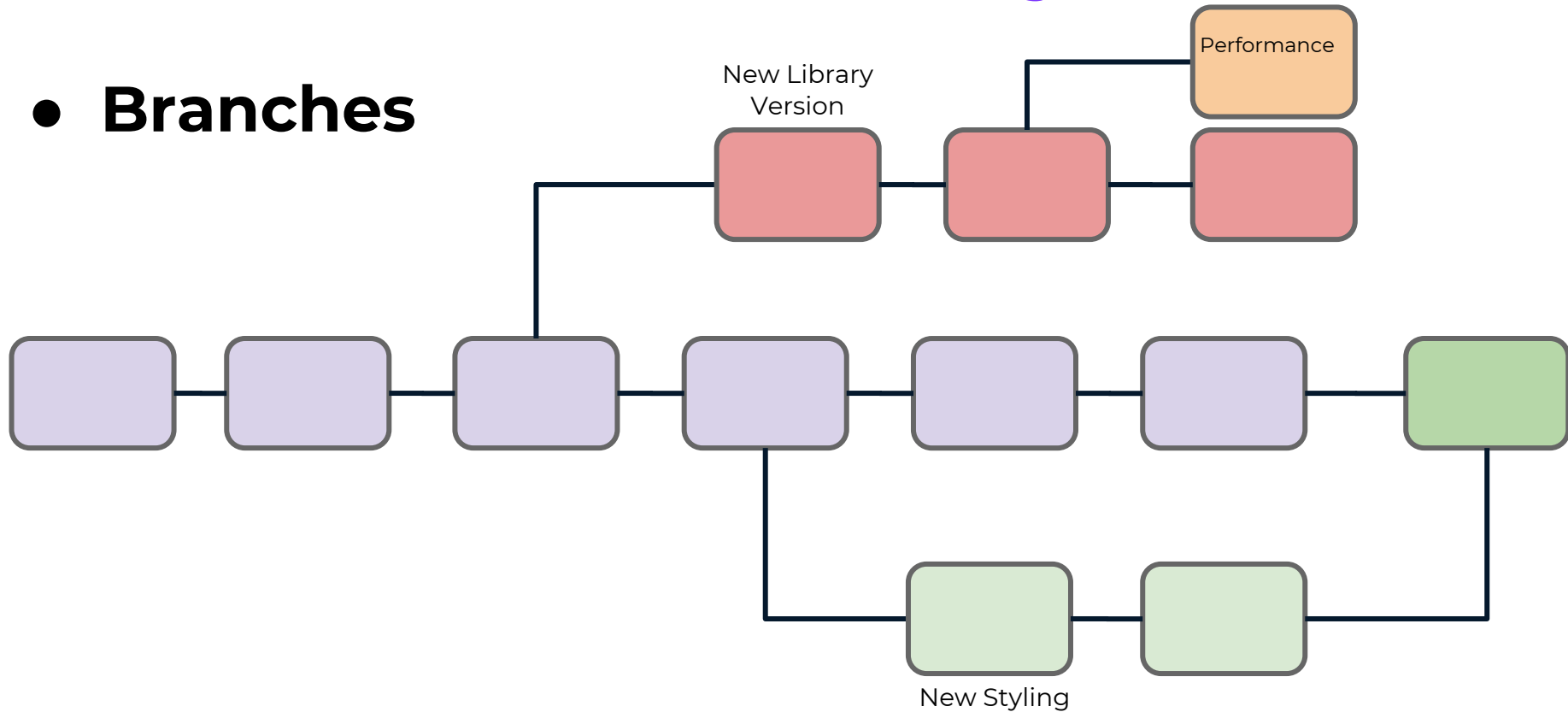
# Week 3 Branches and Working with Others

- **Branches**



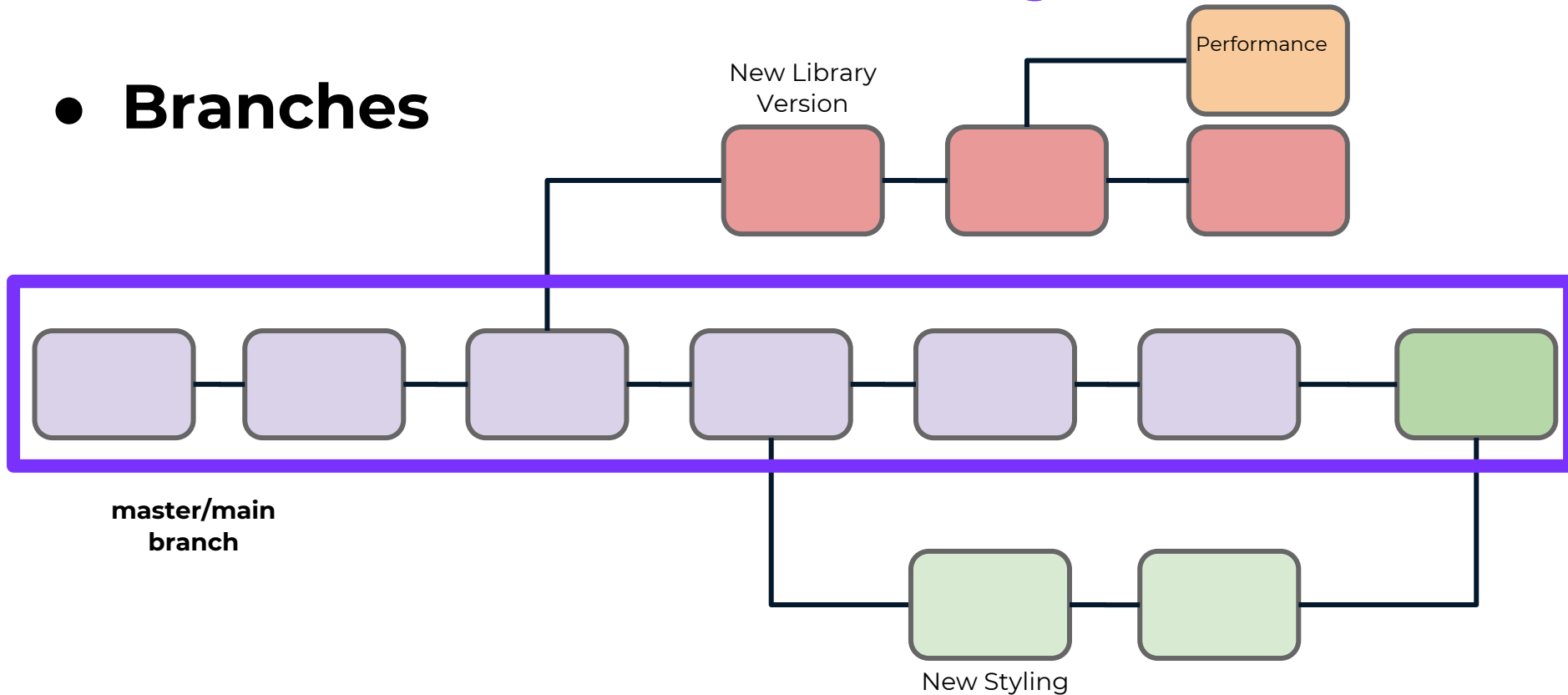
# Week 3 Branches and Working with Others

- **Branches**



# Week 3 Branches and Working with Others

- **Branches**



# Week 3 Branches and Working with Others

- **Branches**

- Upon creating a new repo with **git init** you create a new branch called the **master branch** (or **main branch**).
- This is a branch just like any other, but it's simply the first one created.
  - *Should code pushed to master branch always be in working condition?*



## Week 3 Branches and Working with Others

- **Branches**

- While organizations and developers often treat this master branch as the official branch for things like deployment, this is not a requirement.
- You can use any branch for code deployment or code that's actually "in-use".

# Week 3 Branches and Working with Others

- **Branches**

- Master vs. Main

- As we've discussed previously, GitHub has changed the nomenclature for this initial branch to be **main branch** while Git is still using **master branch** (but this may change in the future).
    - You can also rename any branch (**trunk branch**).

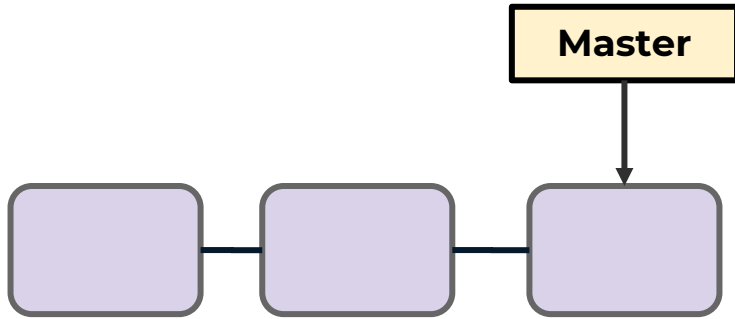
## Week 3 Branches and Working with Others

- **Creating a New Branch**

- Before we conclude, let's quickly go into more detail about what happens when first create a new branch.
- Branches are just pointers to commits.
- When you create a branch, all Git needs to do is create a new pointer, it doesn't change the repository in any other way.

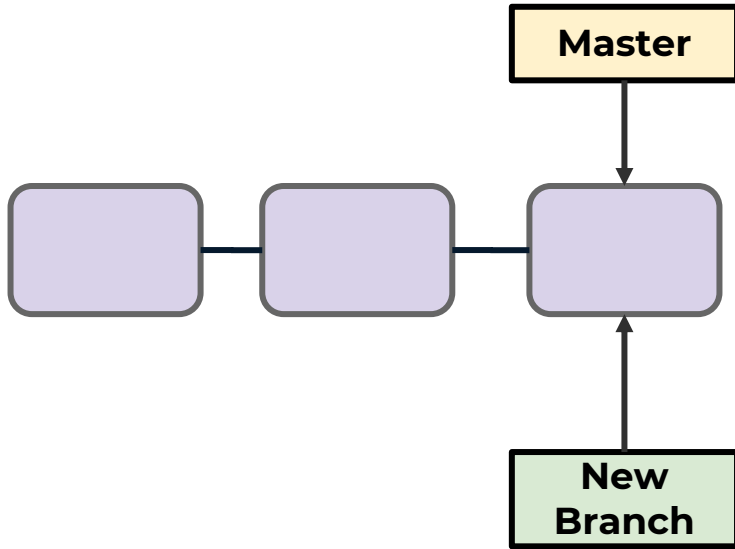
# Week 3 Branches and Working with Others

- **Creating a New Branch**



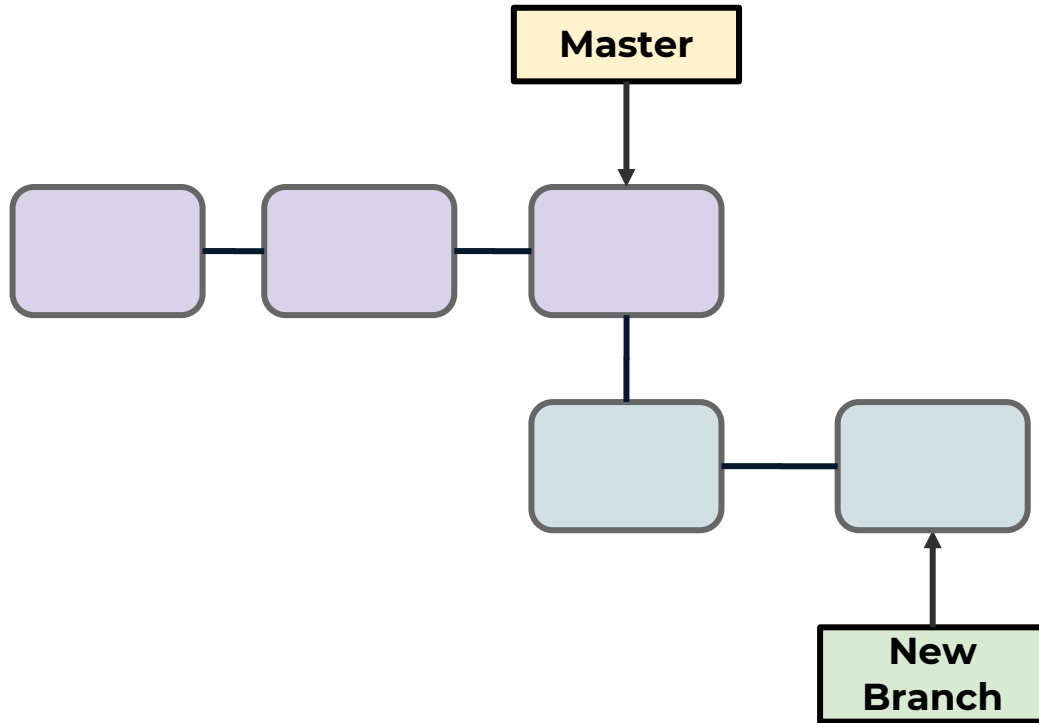
# Week 3 Branches and Working with Others

- **Creating a New Branch**



# Week 3 Branches and Working with Others

- **Creating a New Branch**



## Week 3 Branches and Working with Others

- Now that we've seen how branches point to commits, we need to learn about HEAD.
- HEAD will help us understand what we are currently “viewing” or where we are “located” in regards to branches and commits.

## Week 3 Branches and Working with Others

- **Up Next:**

- We'll explore and visualize specific actions and commands related to branches, including **HEAD**, **git checkout**, **git branch**, **git switch**, and more.



# **Week 3**

## **Understanding HEAD**

## Week 3 Branches and Working with Others

- **Branches**

- As we work more with branches, you will probably notice a term show up during your commits: **HEAD**.
- When viewing the most recent commit using **git log** you may see:
  - **commit 05as..3e2 (HEAD -> master)**

## Week 3 Branches and Working with Others

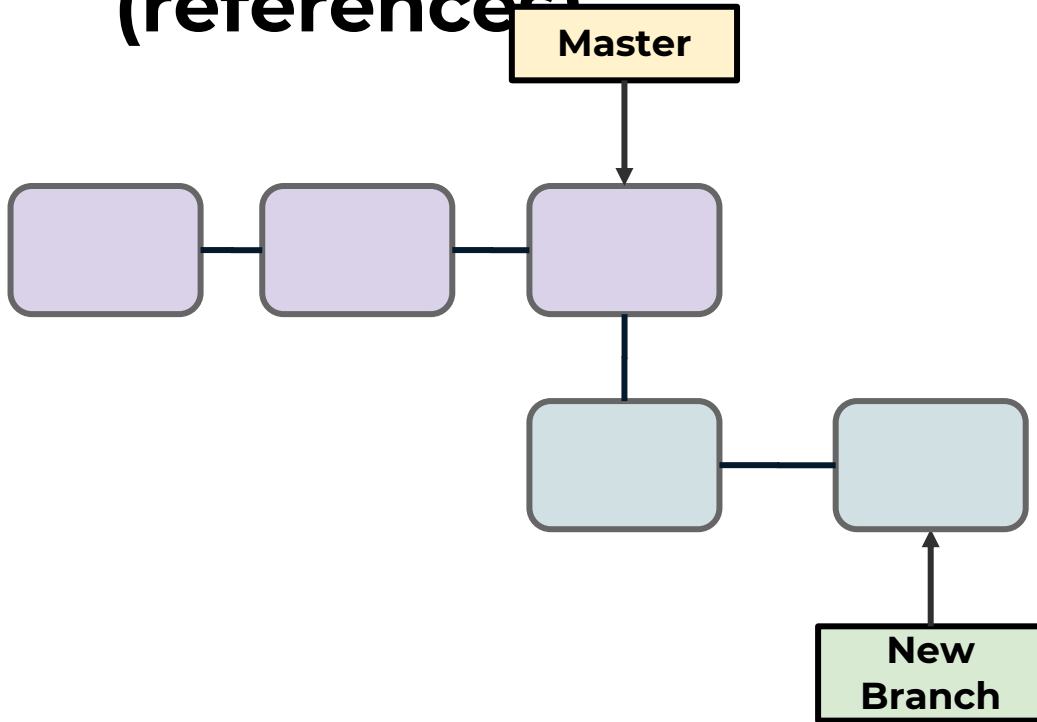
- **HEAD**

- In all of our examples so far, HEAD has always been pointing to the most recent commit in the master branch.

- **HEAD -> master**

# Week 3 Branches and Working with Others

- Recall we have branch points (references)



## Week 3 Branches and Working with Others

- **Branches and Commits**

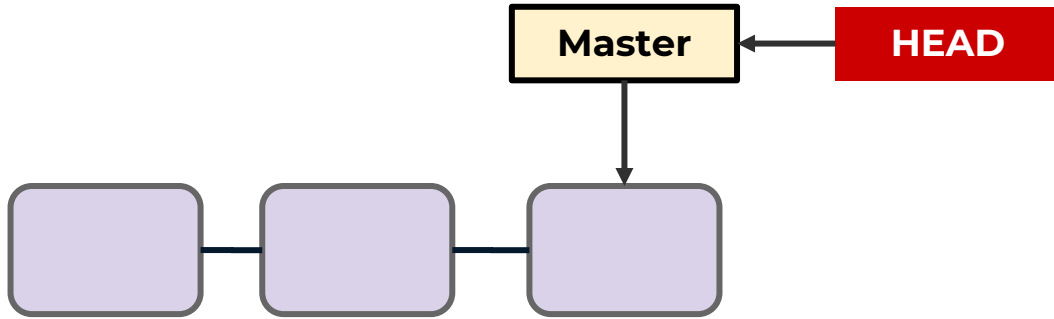
- Git stores a branch as a reference to a commit.
- In this sense, a branch represents the tip of a series of commits—it's not a container for commits.
- The history for a branch is extrapolated through the commit relationships.

## Week 3 Branches and Working with Others

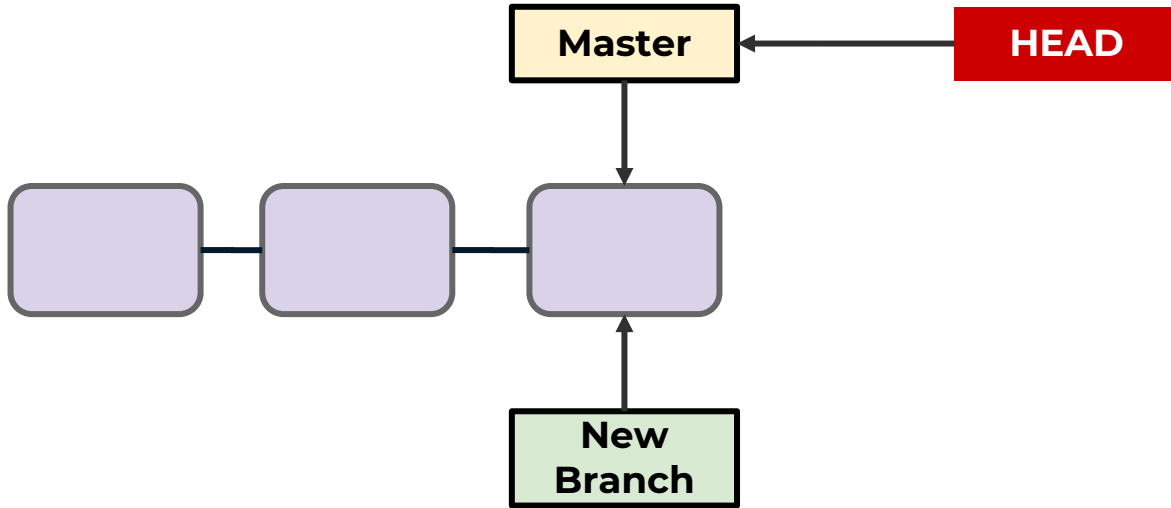
- **HEAD**

- A HEAD is simply a reference to a commit object.
- We can think of HEAD as pointing to a specific commit in a branch that we are currently viewing.

# Week 3 Branches and Working with Others

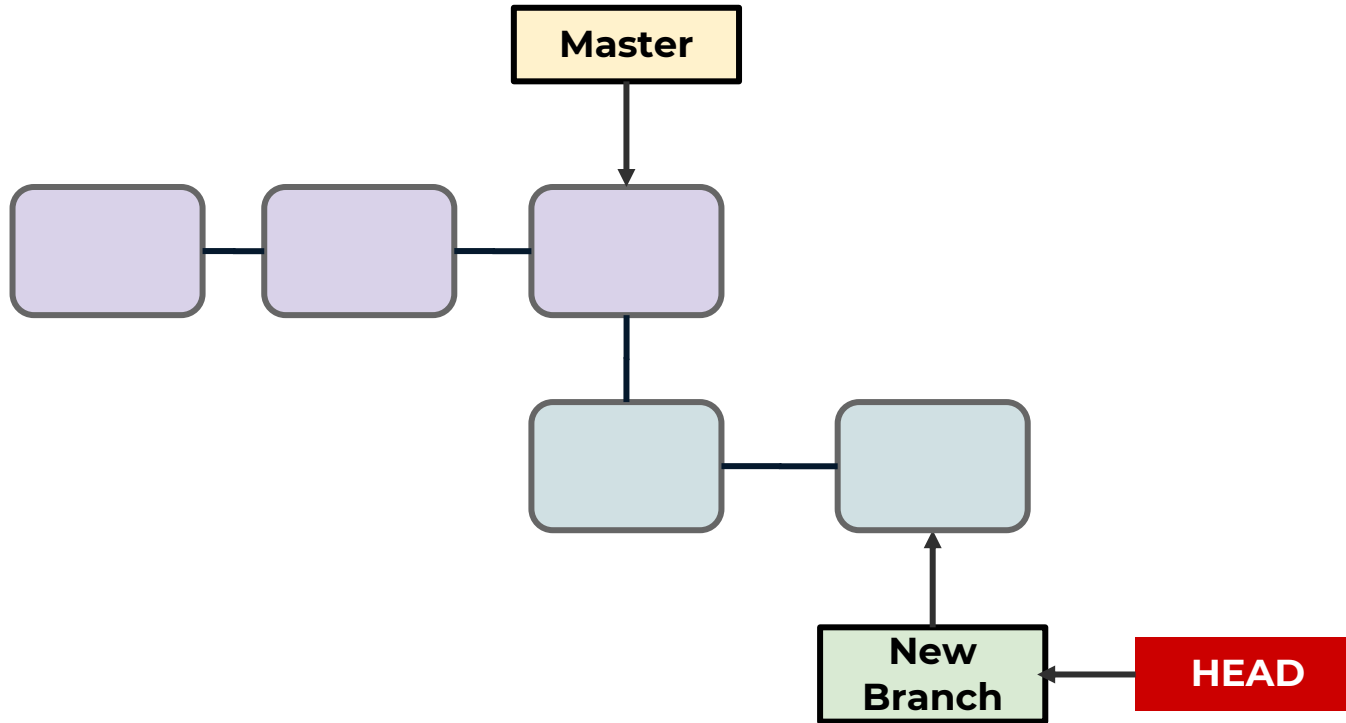


# Week 3 Branches and Working with Others





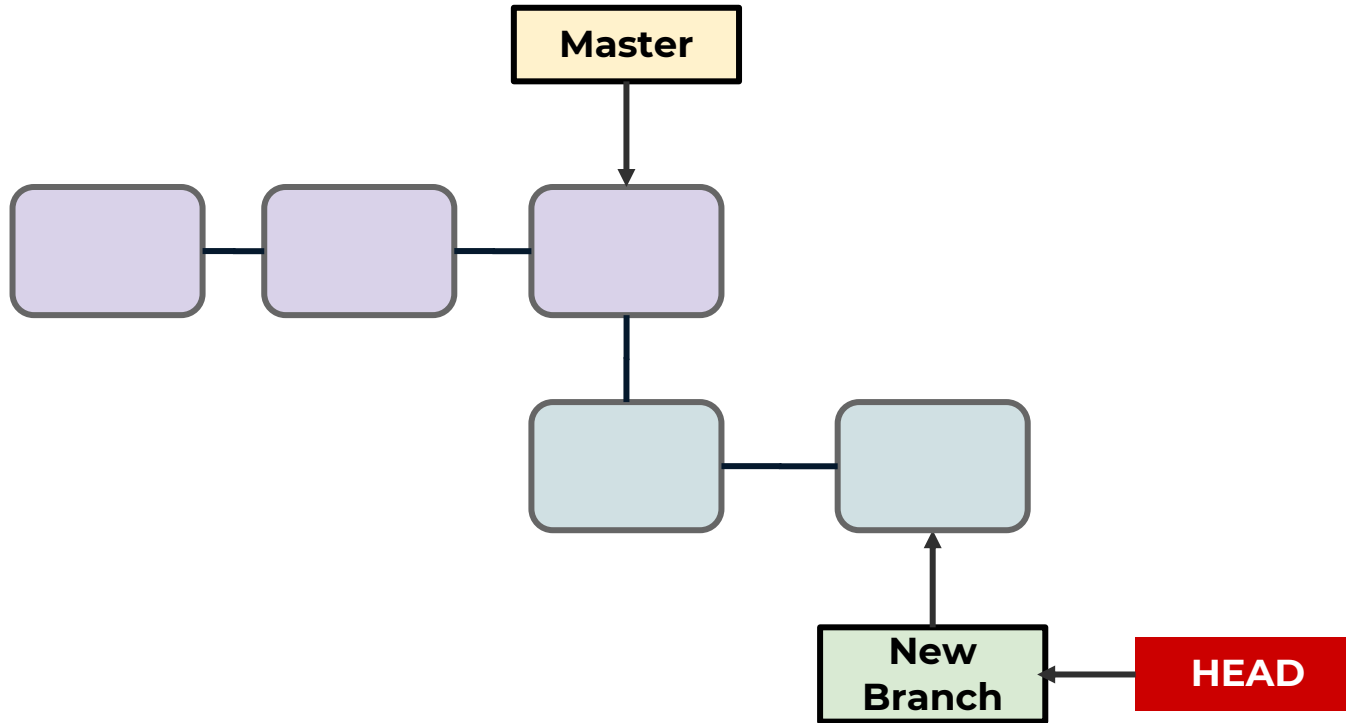
# Week 3 Branches and Working with Others



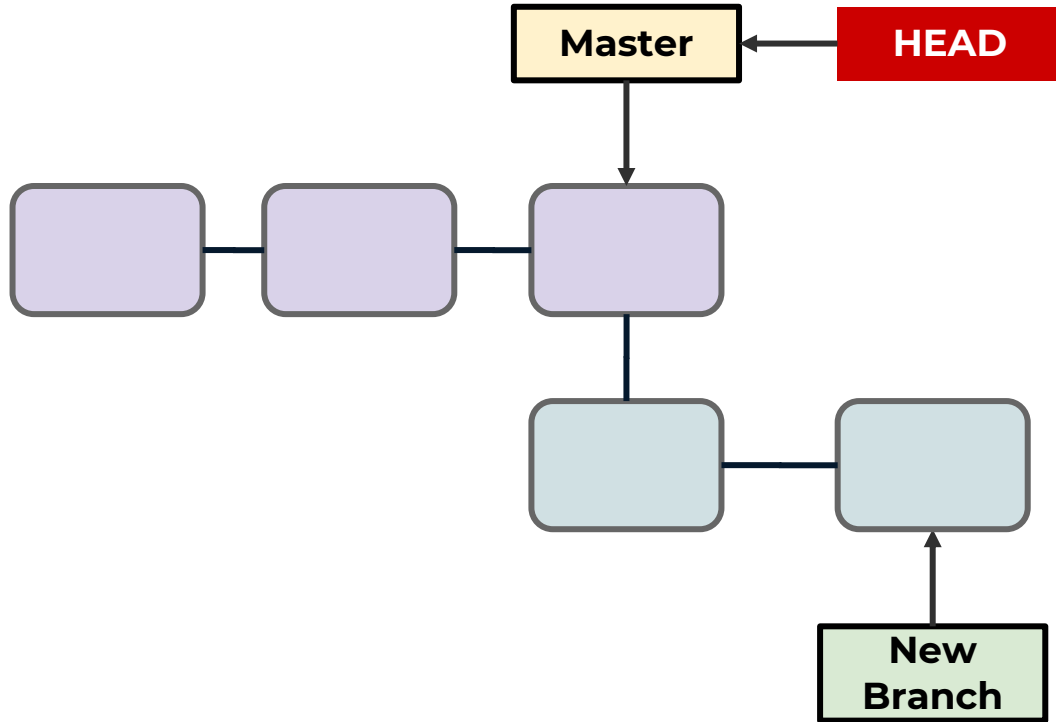
## Week 3 Branches and Working with Others

- We can think of these branches as just references to a commit.
- Using HEAD tells us which branch reference we are currently “checking out”.
- We can always switch back out HEAD to some other branch (which is a pointer to a commit reference).

# Week 3 Branches and Working with Others



# Week 3 Branches and Working with Others



## Week 3 Branches and Working with Others

- **Up Next:**

- Now that we understand the theory behind branches and HEAD, let's begin to explore the actual commands that let us create branches and navigate between them.

# **Week 3**

## **Git Branch Commands**

# Week 3 Branches and Working with Others

- **Git Branch Commands**

- Create a New Repo
- Add File
- Create a New Branch
  - **git branch <branch\_name>**
- Report Branches
  - **git branch**
- Switch Branches
  - **git switch**

## Week 3 Branches and Working with Others

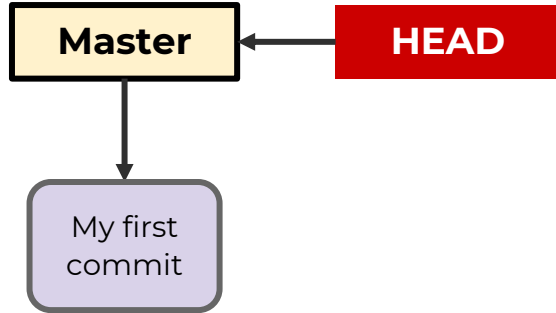
- **Git Branch Commands**

- Add and Commit Changes on New Branch
- Use **git log** and **git switch** to explore differences between branches.



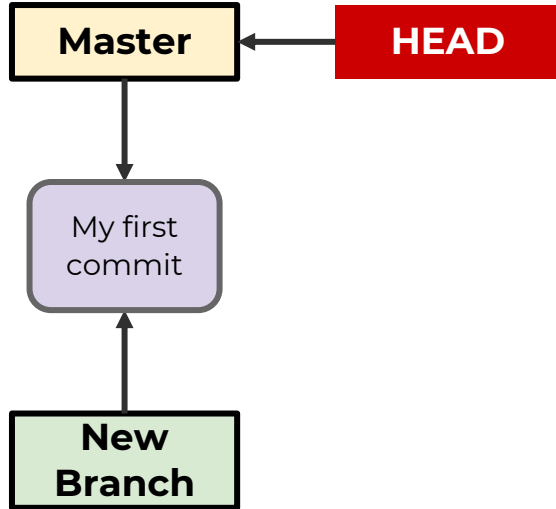
# Week 3 Branches and Working with Others

- **git init, git add, git commit**



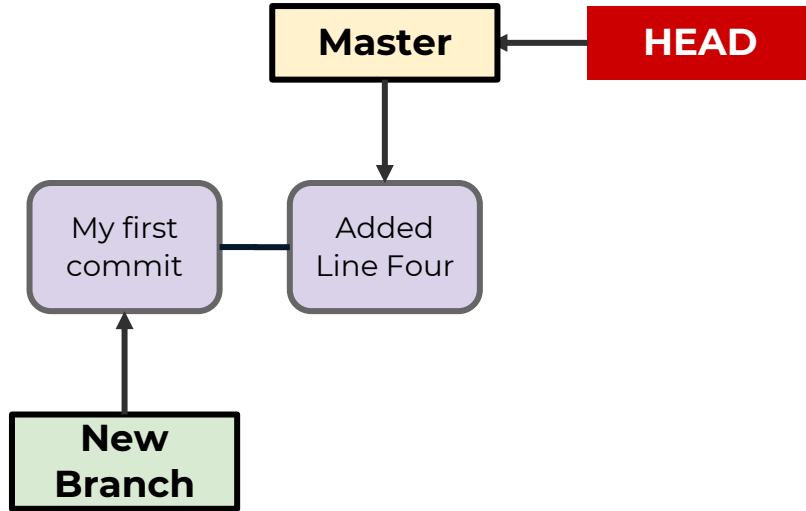
# Week 3 Branches and Working with Others

- **git branch new\_branch**



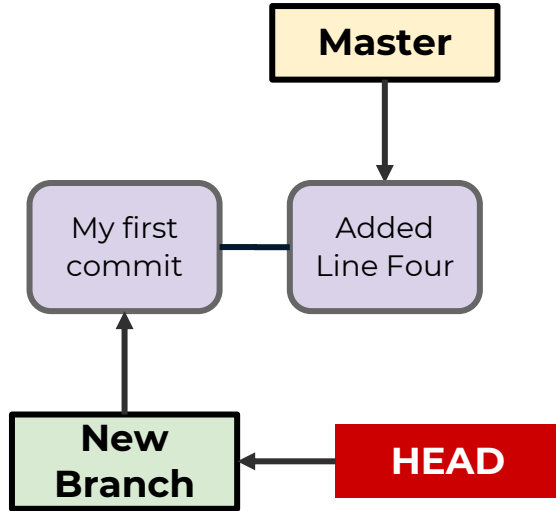
# Week 3 Branches and Working with Others

- **git add, git commit, git log**



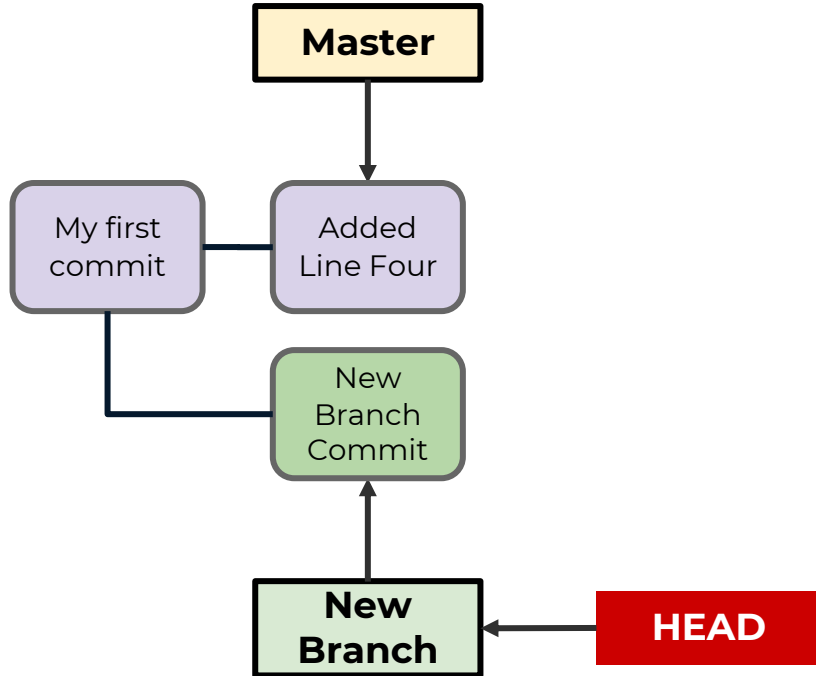
# Week 3 Branches and Working with Others

- **git switch new\_branch**



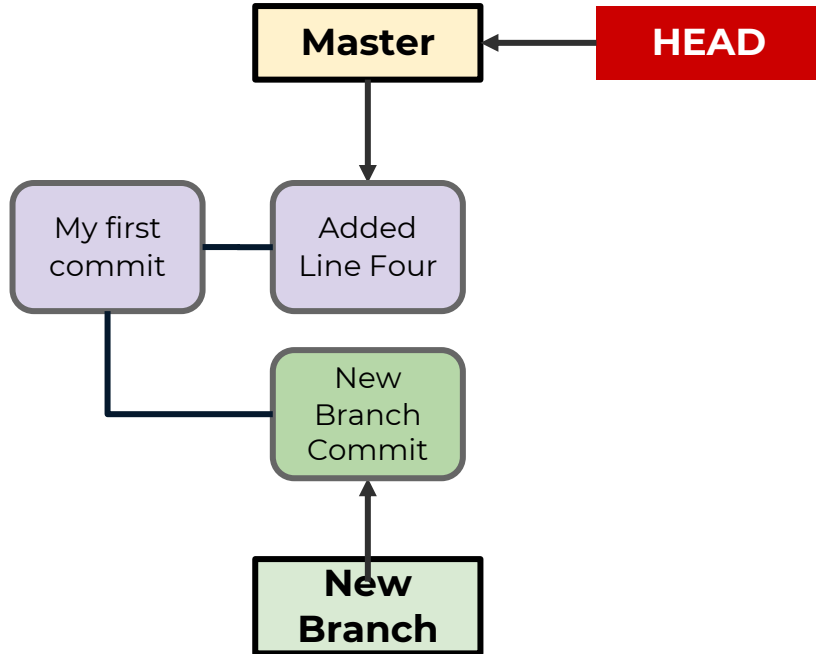
# Week 3 Branches and Working with Others

- **git add , git commit, git log**



# Week 3 Branches and Working with Others

- **git switch master**



# **Week 3**

## **Delete and Rename Branches**

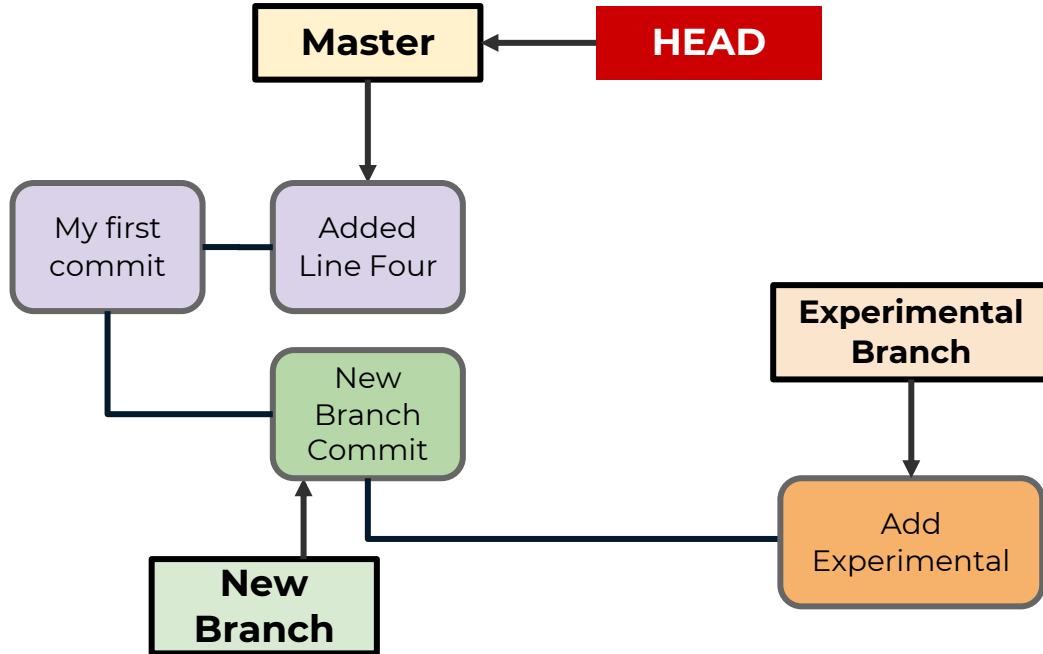
## Week 3 Branches and Working with Others

- Let's quickly explore how to rename and delete branches.
- Keep in mind that we still need to learn how to merge branches together.

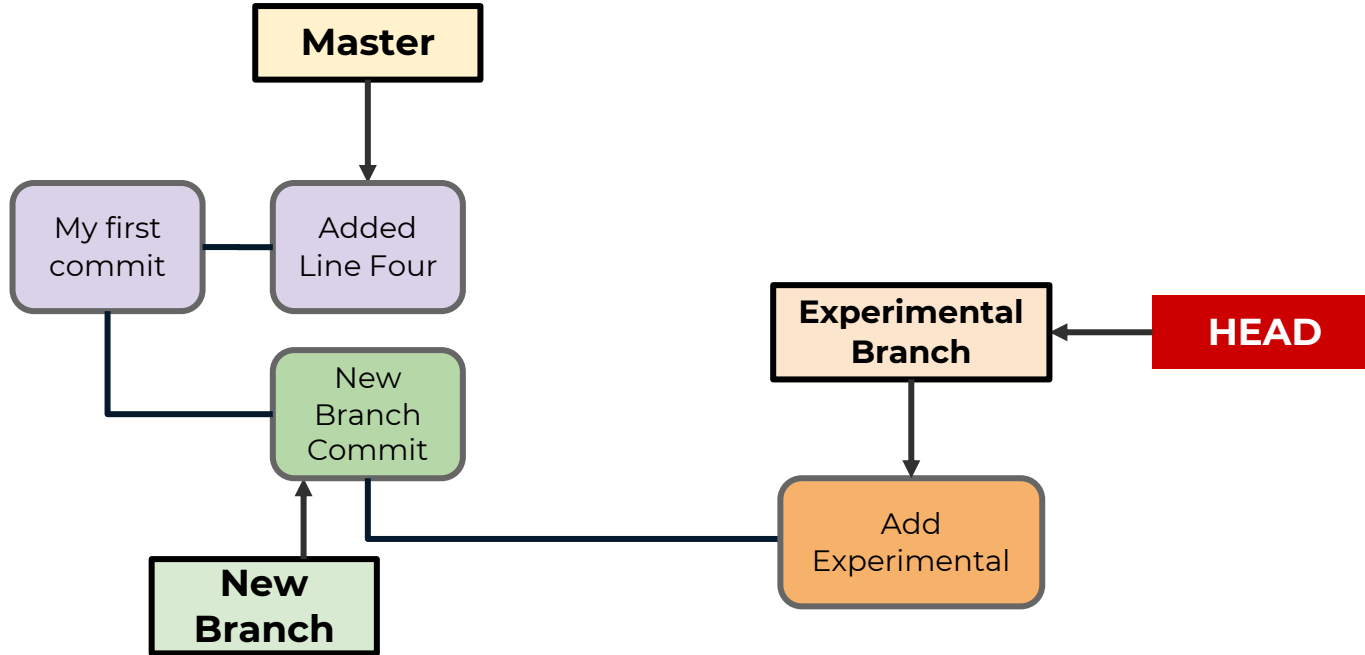


# Week 3 Branches and Working with Others

- **Previously:**



# Week 3 Branches and Working with Others

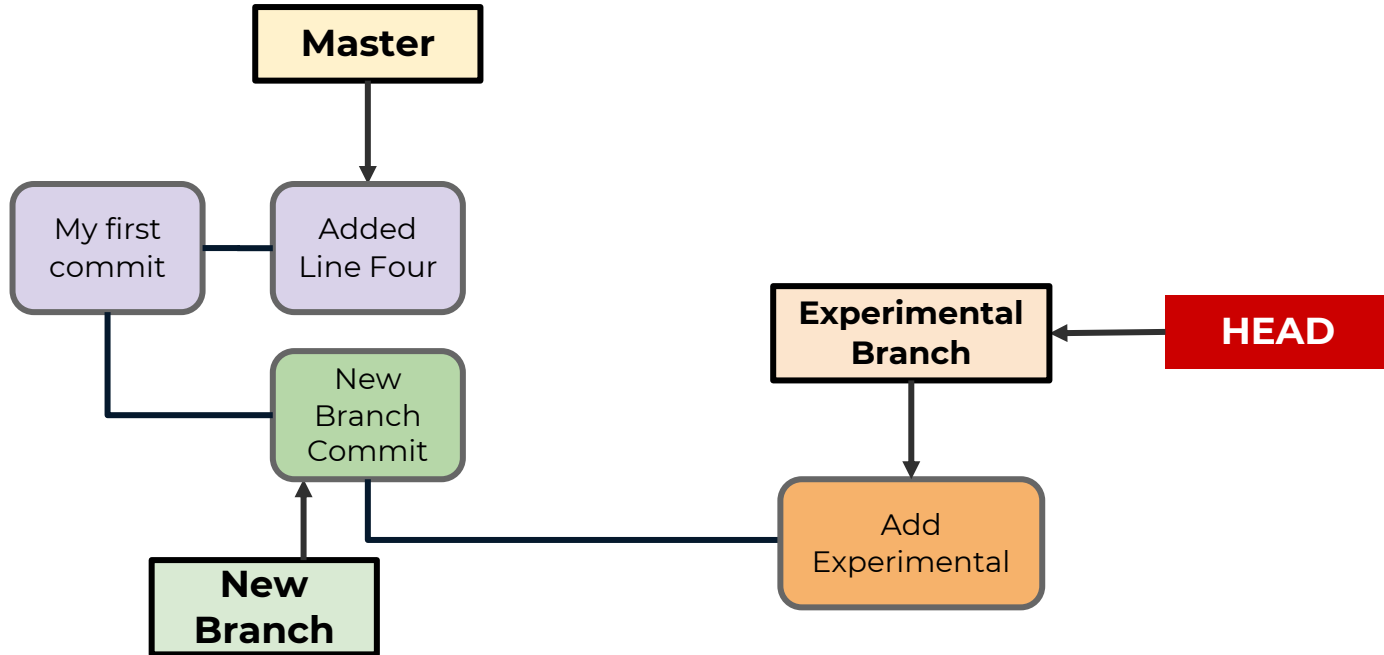


## Week 3 Branches and Working with Others

- **Renaming a Branch**
  - **git switch branch\_to\_rename**
  - **git branch -m new\_name**
    - You must be checked out on the branch you will rename.

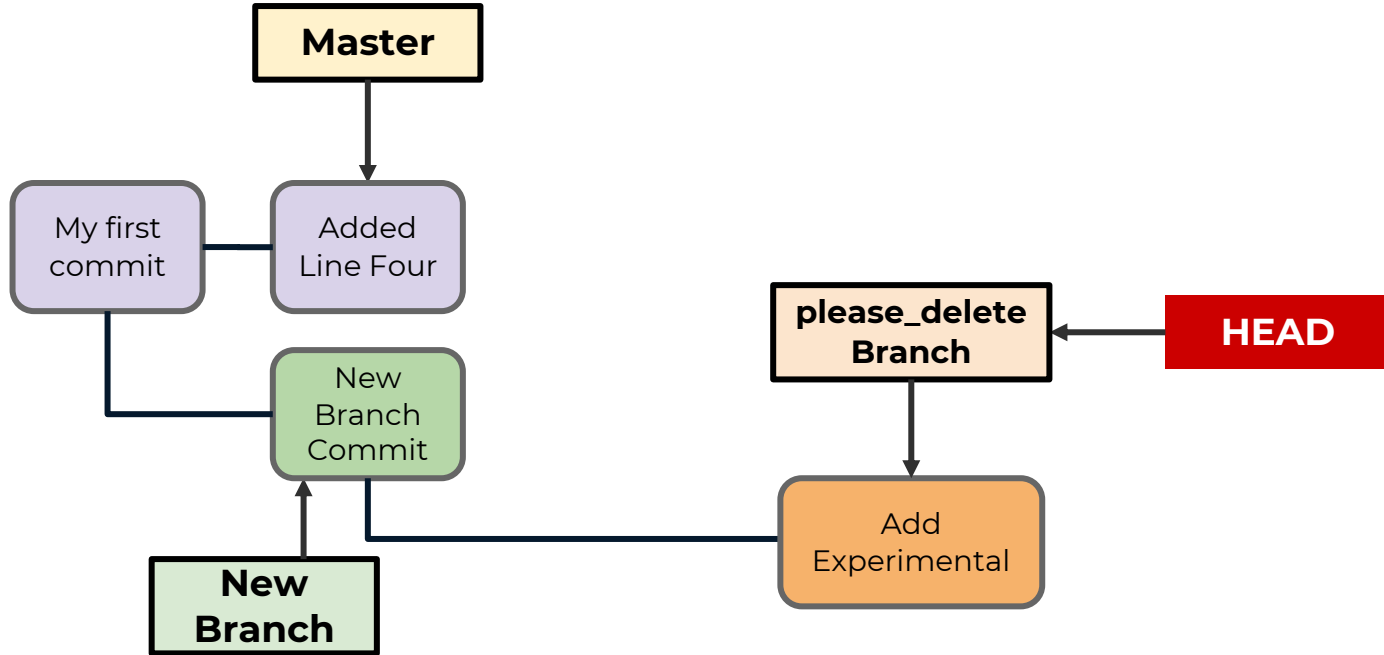
# Week 3 Branches and Working with Others

- **git switch experimental**



# Week 3 Branches and Working with Others

- **git branch -m please\_delete**



## Week 3 Branches and Working with Others

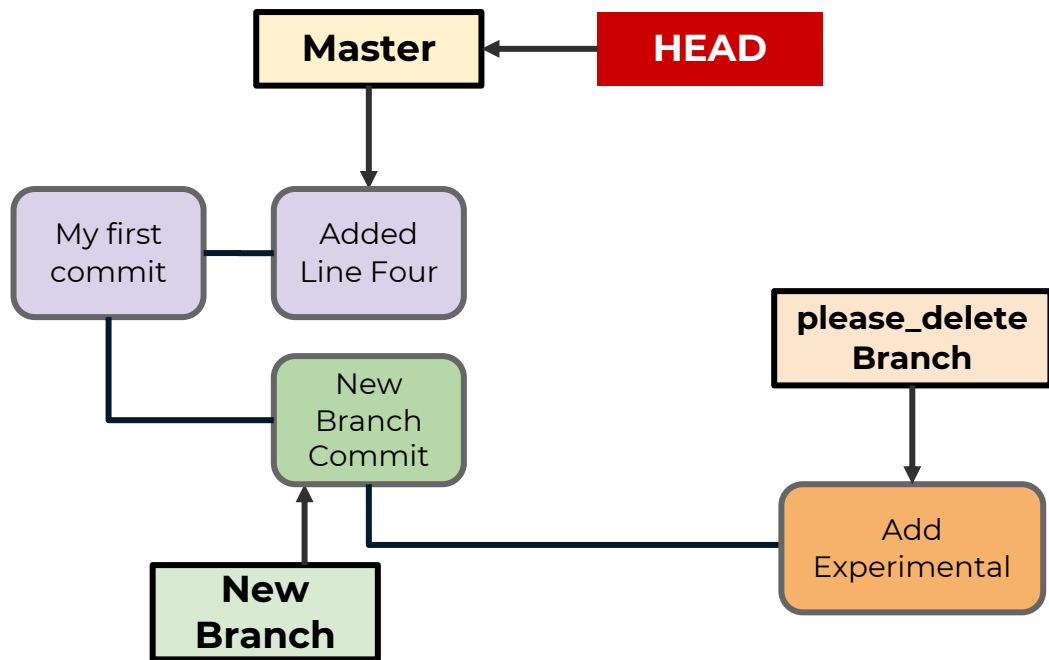
- **Deleting a Branch**

- **git branch -d branch\_to\_delete\_name**
  - You can not delete a branch you are checked out at.
  - You also will get a warning if the branch is not merged.
    - You can confirm you want to do this anyways with **-D**



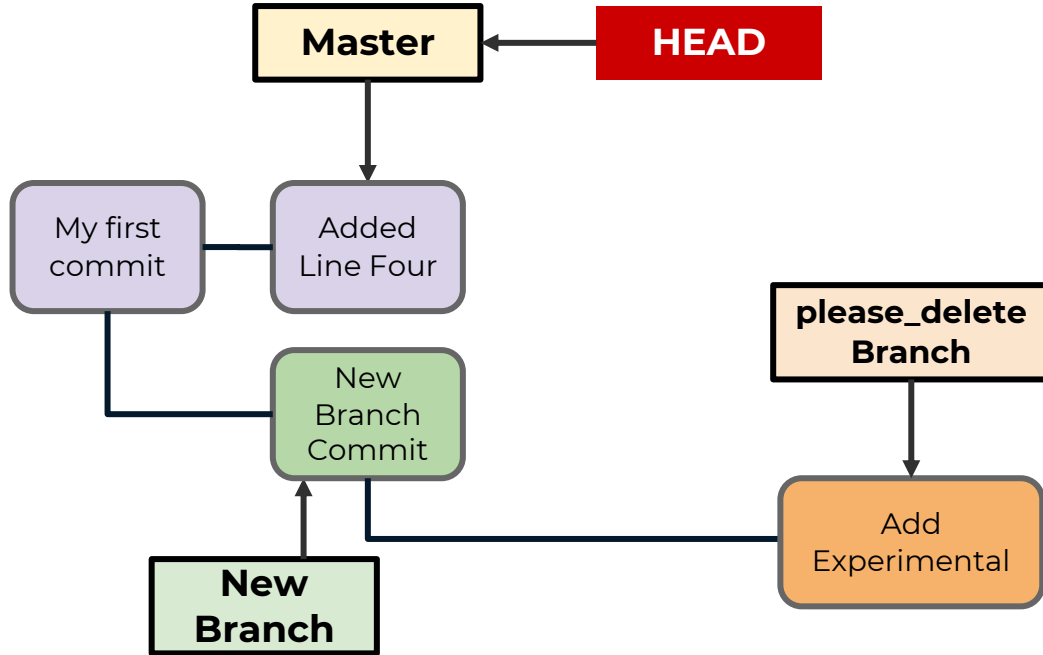
# Week 3 Branches and Working with Others

- **git switch master**



# Week 3 Branches and Working with Others

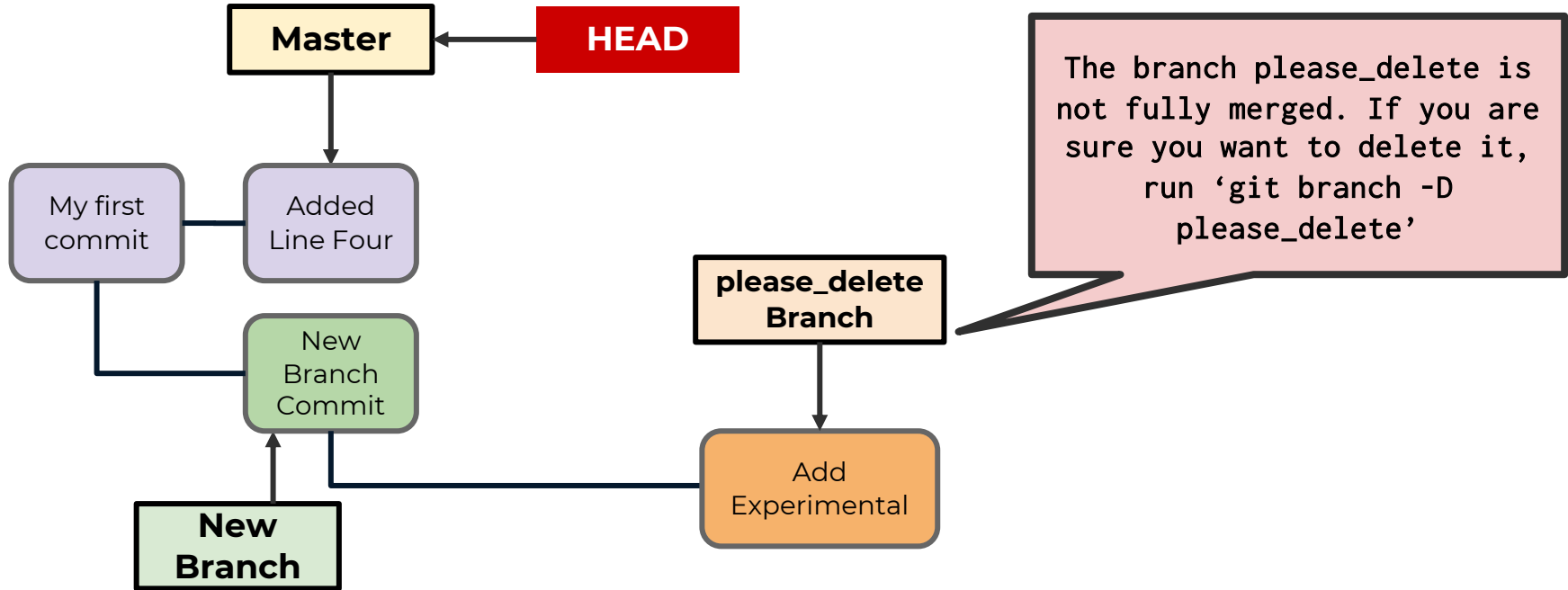
- **git branch -d please\_delete**





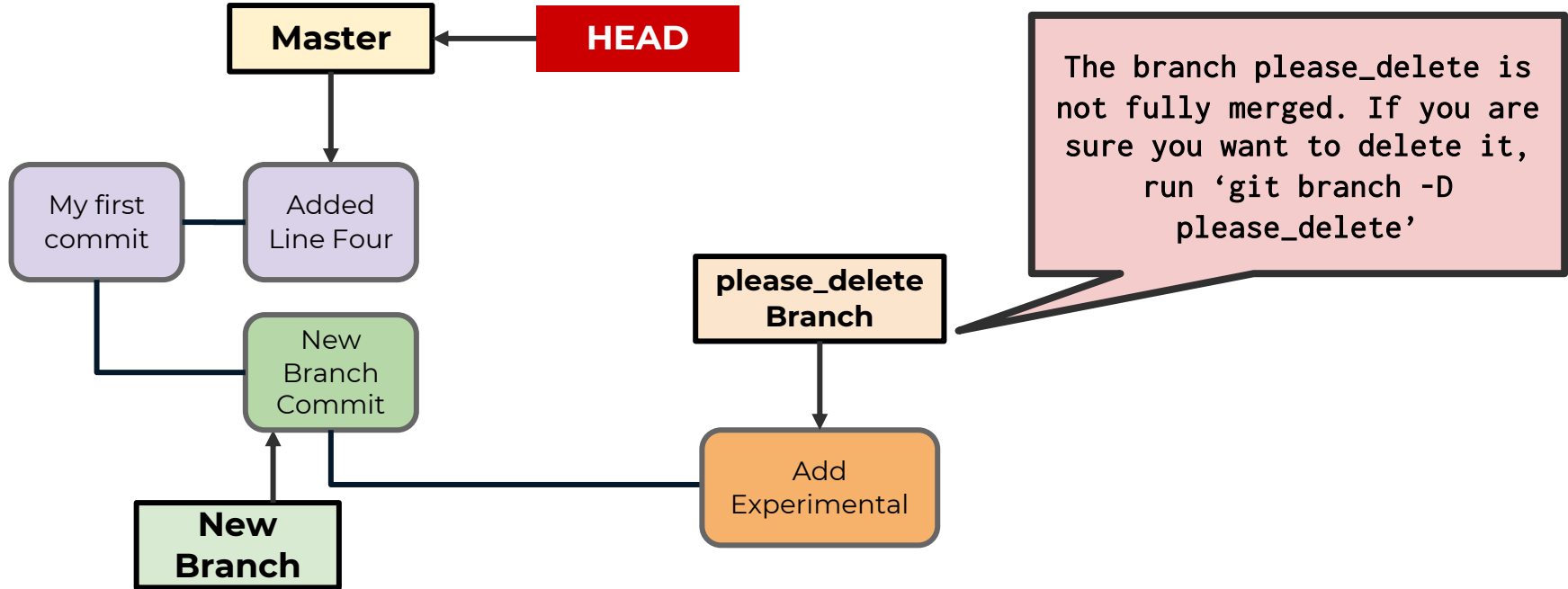
# Week 3 Branches and Working with Others

- **git branch -d please\_delete**



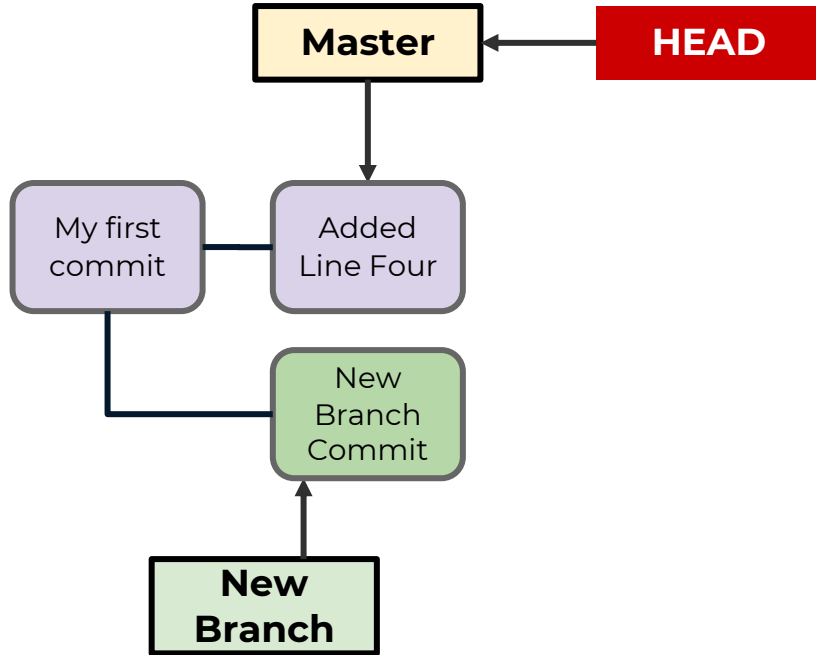
# Week 3 Branches and Working with Others

- **git branch -D please\_delete**



# Week 3 Branches and Working with Others

- **git branch -D please\_delete**



# **Week 3**

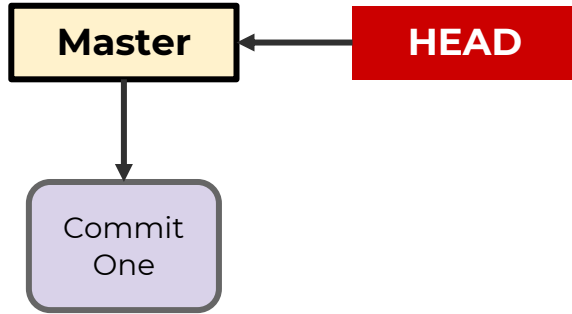
## **Merging Branches and Conflicts**

## Week 3 Branches and Working with Others

- Now that we understand creating new branches, let's shift focus to merging branches back together.
- Let's explore a simple type of merge, where a new branch is created, but the original branch it stemmed from has no additional commits.
  - This is known as a “fast-forward” merge

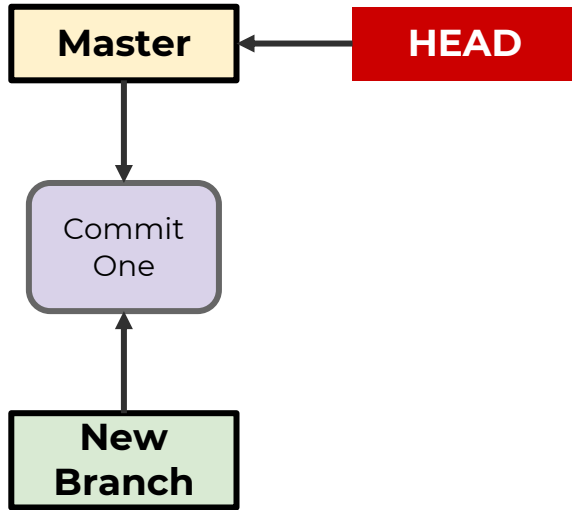
# Week 3 Branches and Working with Others

- **“Fast Forward” Merge**



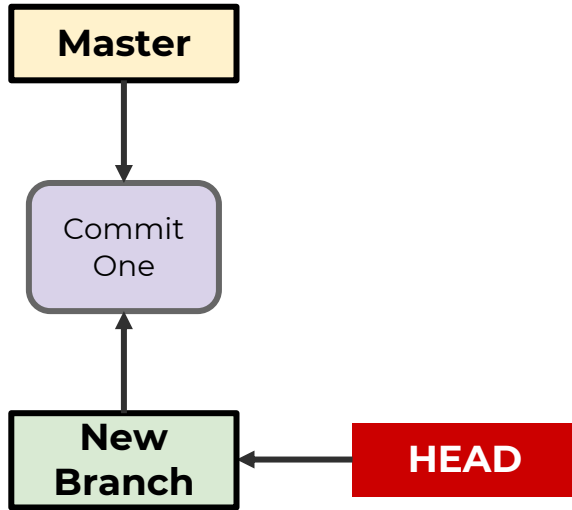
# Week 3 Branches and Working with Others

- **“Fast Forward” Merge**



# Week 3 Branches and Working with Others

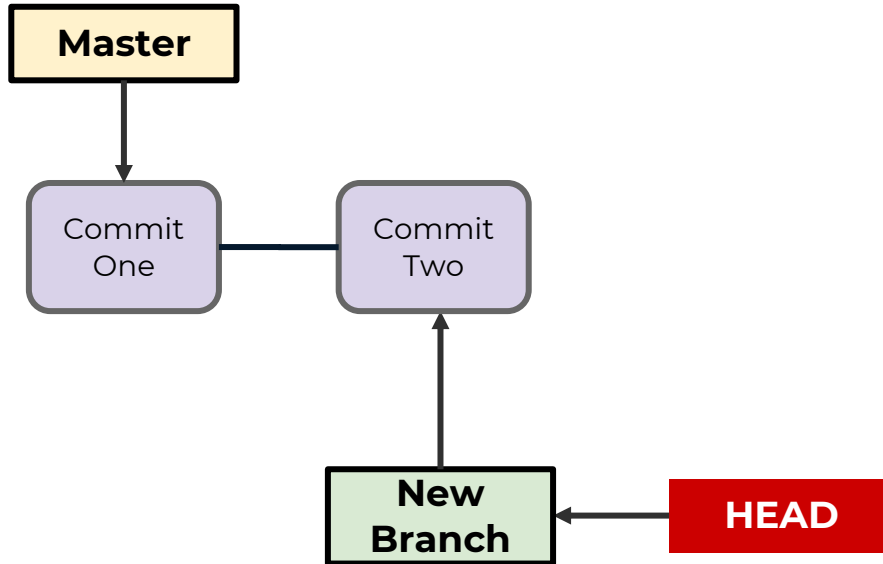
- **“Fast Forward” Merge**





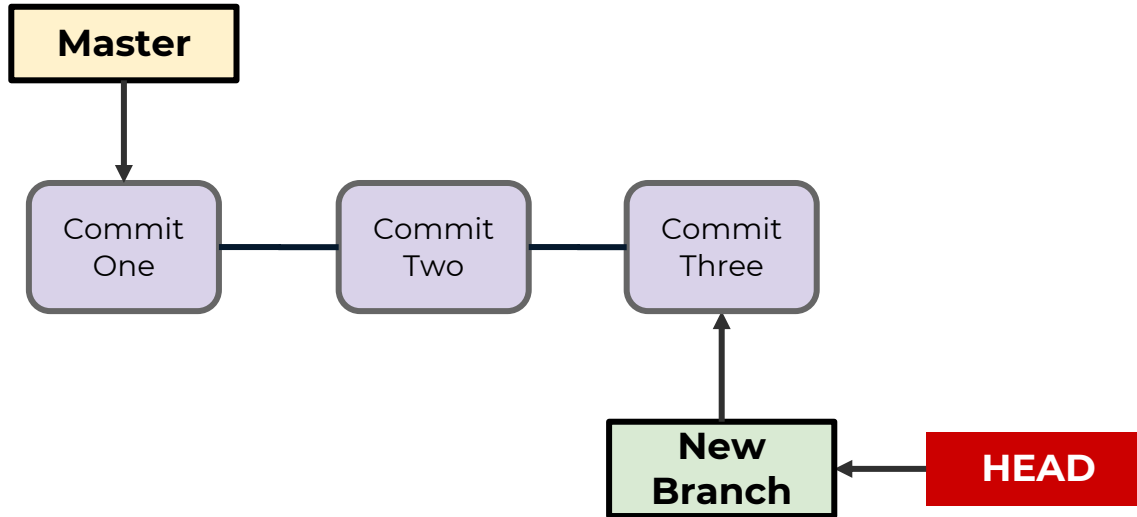
# Week 3 Branches and Working with Others

- **“Fast Forward” Merge**



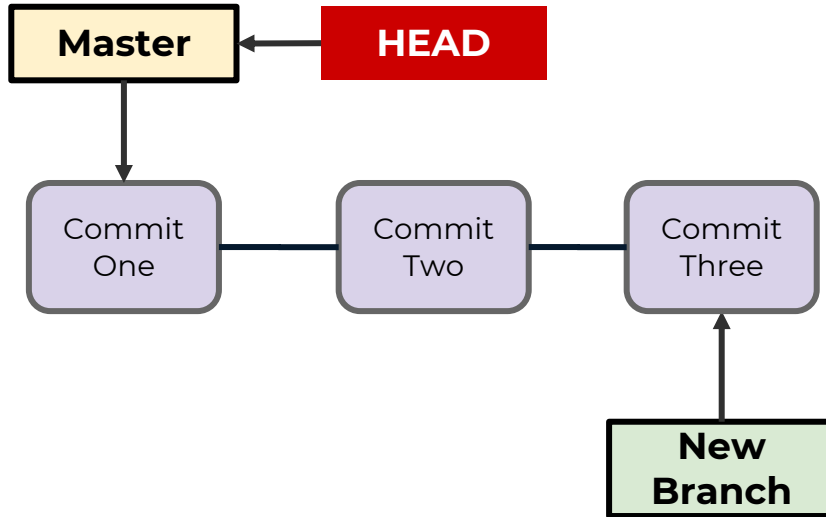
# Week 3 Branches and Working with Others

- **“Fast Forward” Merge**



# Week 3 Branches and Working with Others

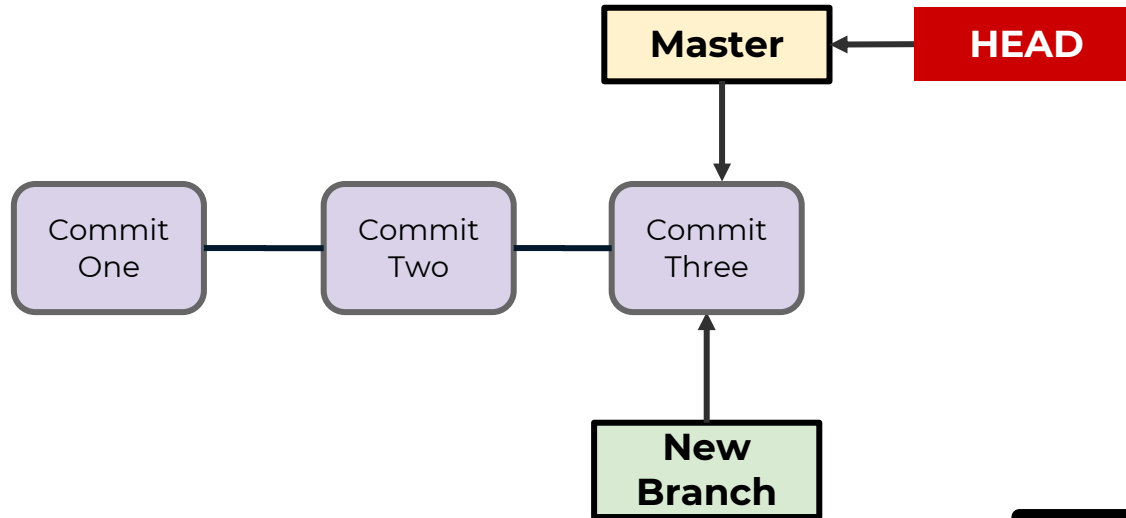
- “Fast Forward” Merge



```
>> git switch master
```

# Week 3 Branches and Working with Others

- **“Fast Forward” Merge**



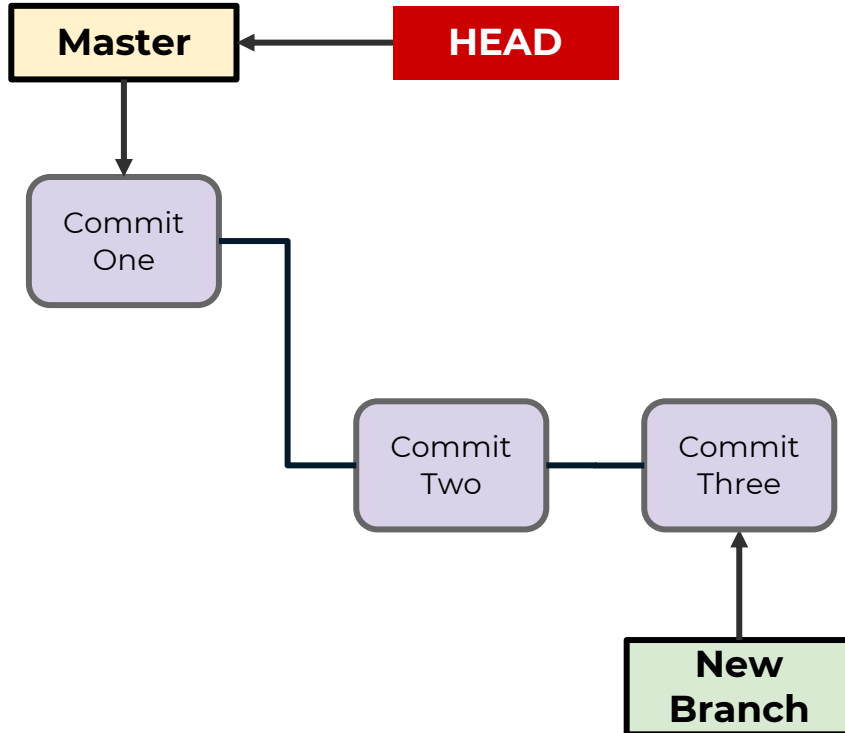
```
>> git merge  
new_branch
```

## Week 3 Branches and Working with Others

- Now let's explore what happens for a merge where we have different commits in the branches.

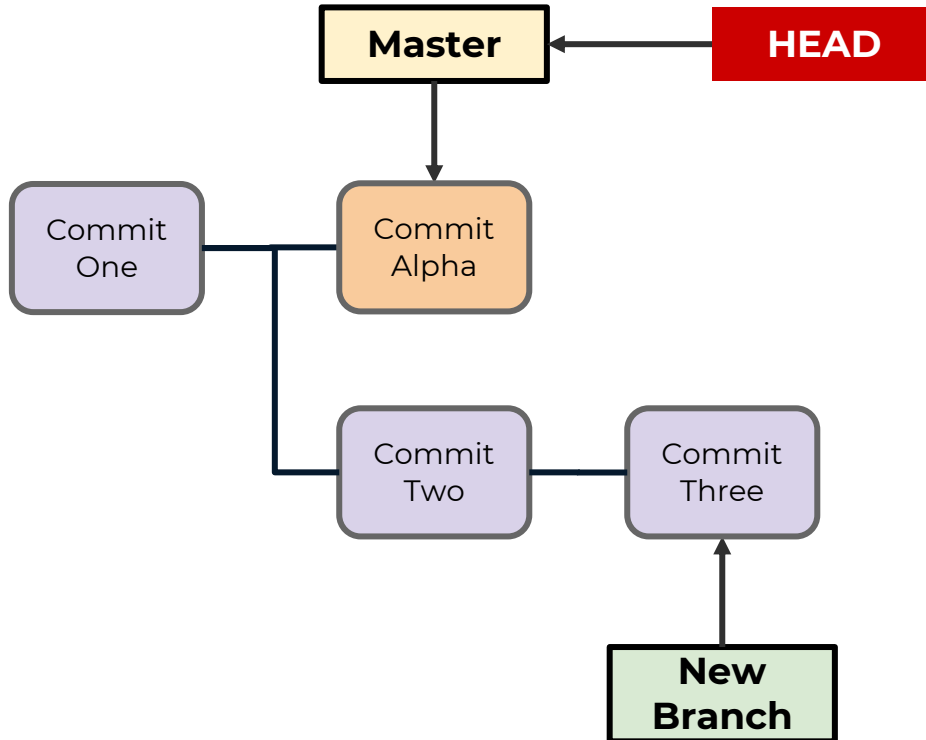
# Week 3 Branches and Working with Others

- **Git Merge**



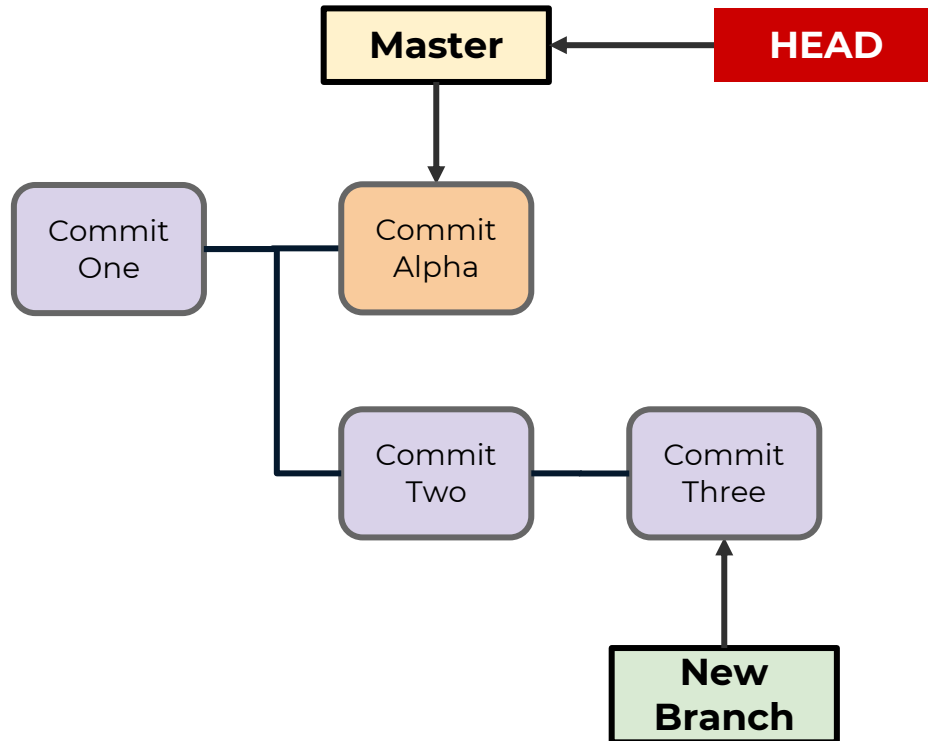
# Week 3 Branches and Working with Others

- **Git Merge**



# Week 3 Branches and Working with Others

- **Git Merge**

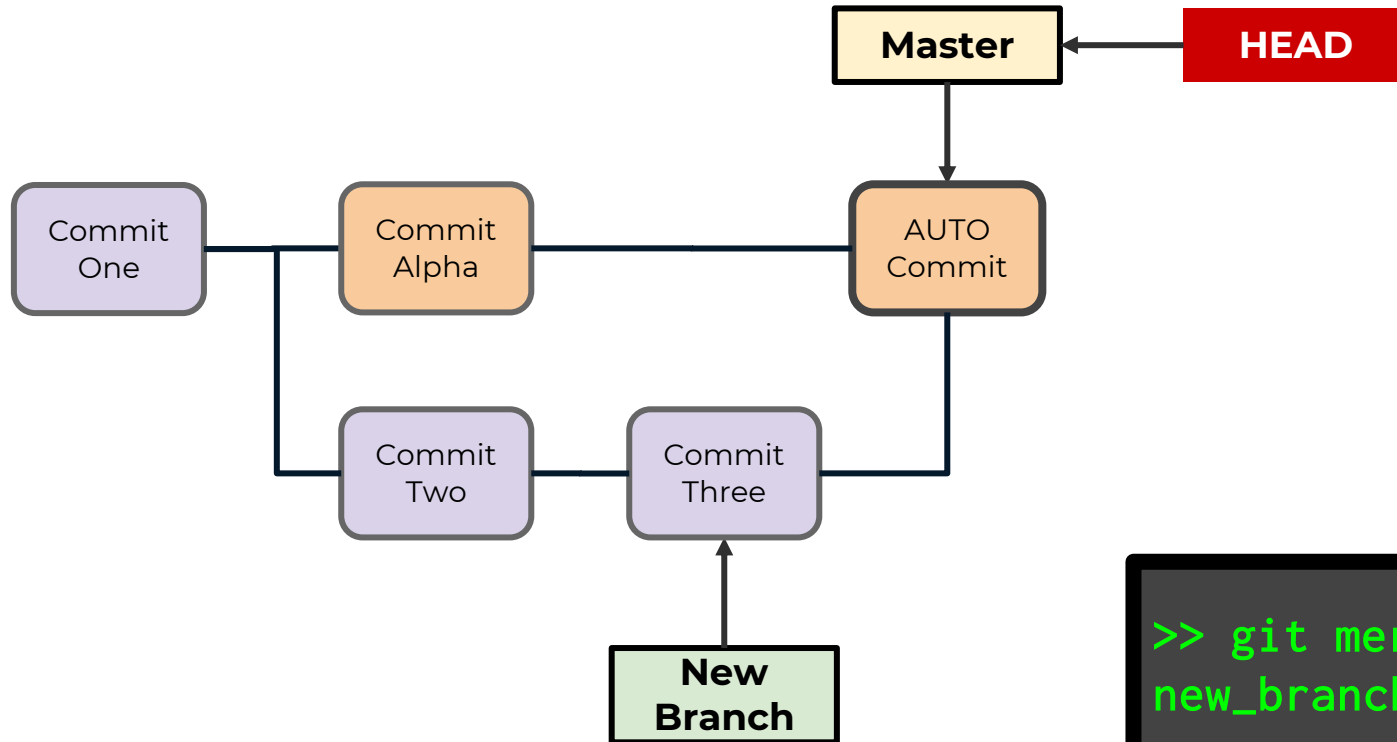


```
>> git merge  
new_branch
```



# Week 3 Branches and Working with Others

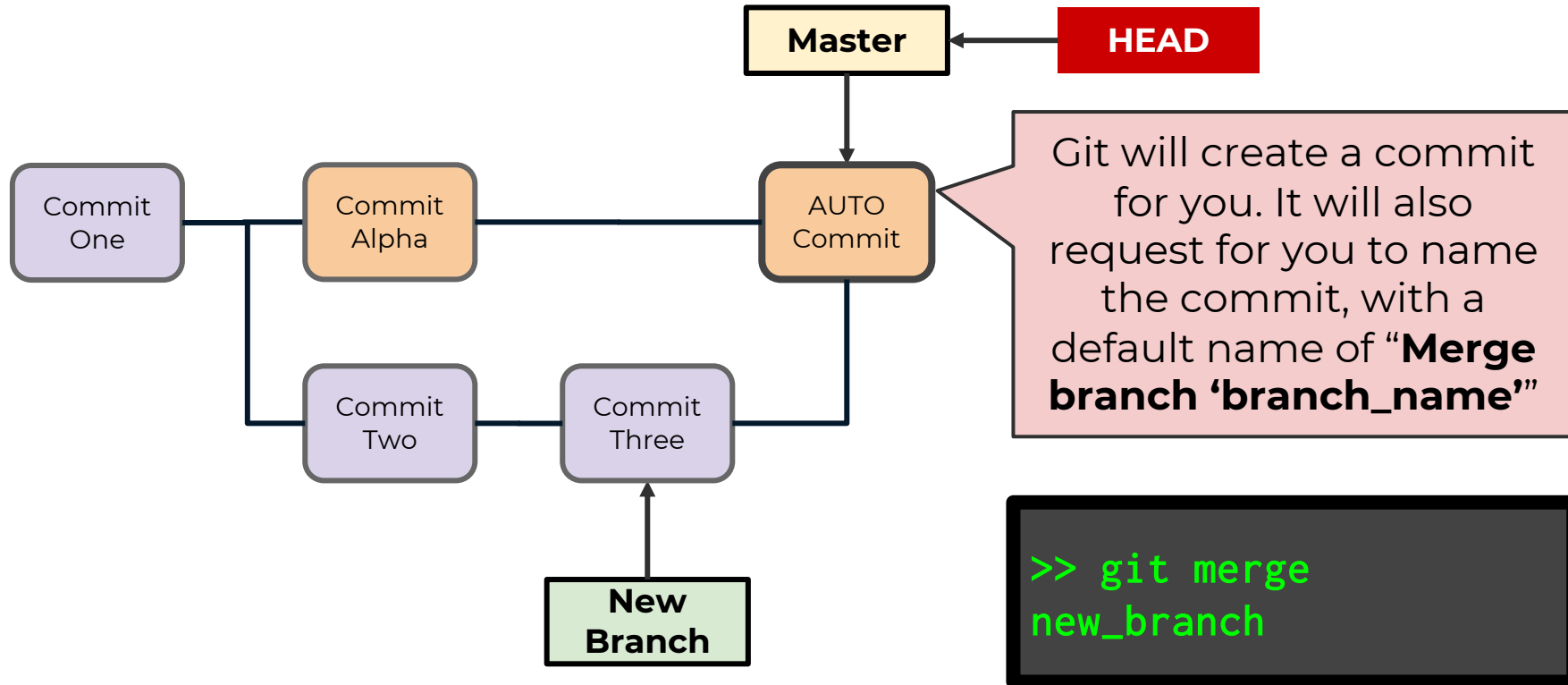
- **Git Merge**



```
>> git merge  
new_branch
```

# Week 3 Branches and Working with Others

## ● Git Merge



## Week 3 Branches and Working with Others

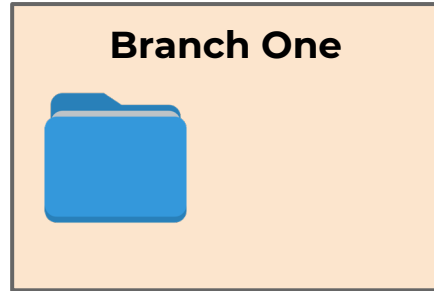
- Git creates the new commit for us, and will attempt the merge.
- Sometimes there are no conflicts, for example:
  - The branch only focused on files not in the receiving branch, thus the merge simply adds the new files to the receiving branch.

## Week 3 Branches and Working with Others

- Git will try to automatically create the merge, and can do so in cases where no information from a specific branch would be lost, for example:

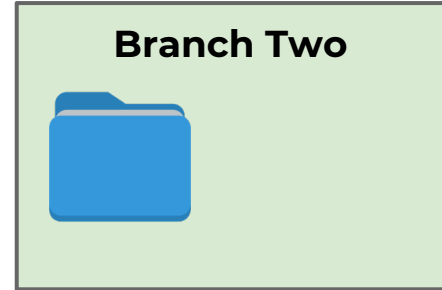
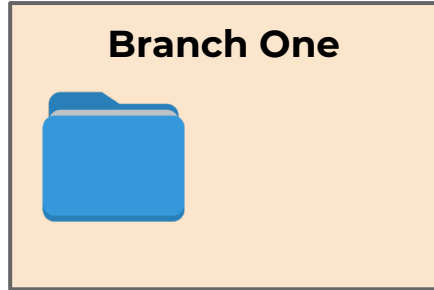
## Week 3 Branches and Working with Others

- Git will try to automatically create the merge, and can do so in cases where no information from a specific branch would be lost, for example:



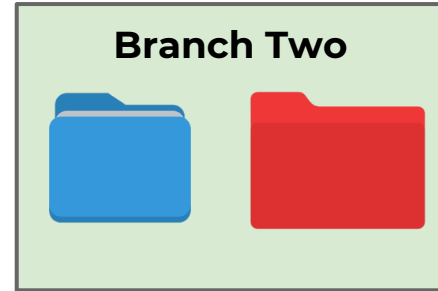
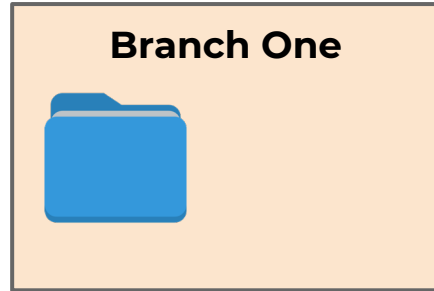
## Week 3 Branches and Working with Others

- Git will try to automatically create the merge, and can do so in cases where no information from a specific branch would be lost, for example:



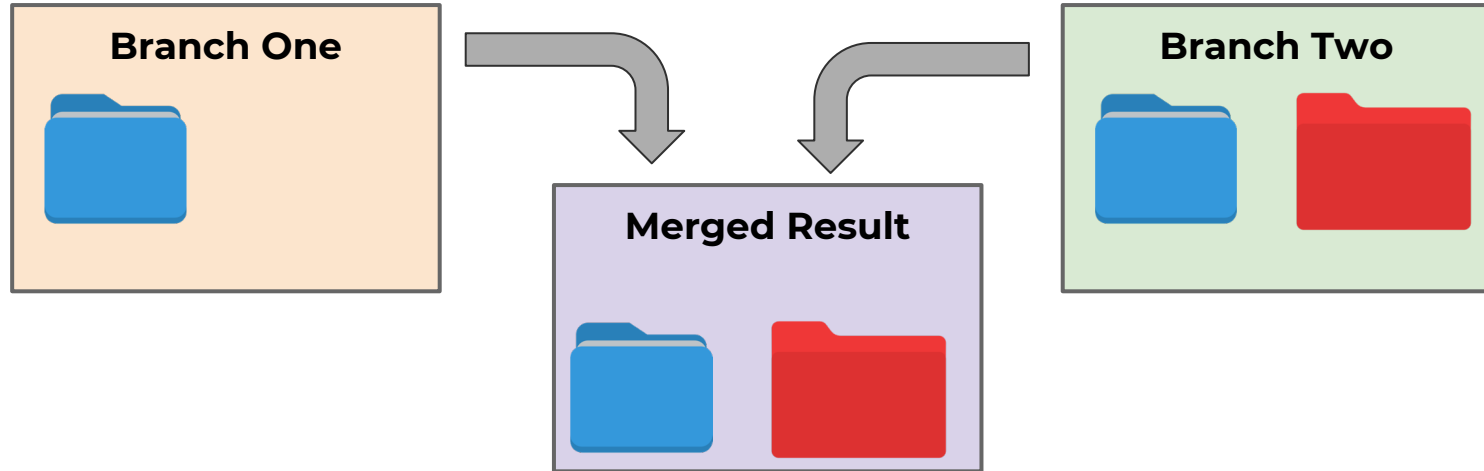
## Week 3 Branches and Working with Others

- Git will try to automatically create the merge, and can do so in cases where no information from a specific branch would be lost, for example:



## Week 3 Branches and Working with Others

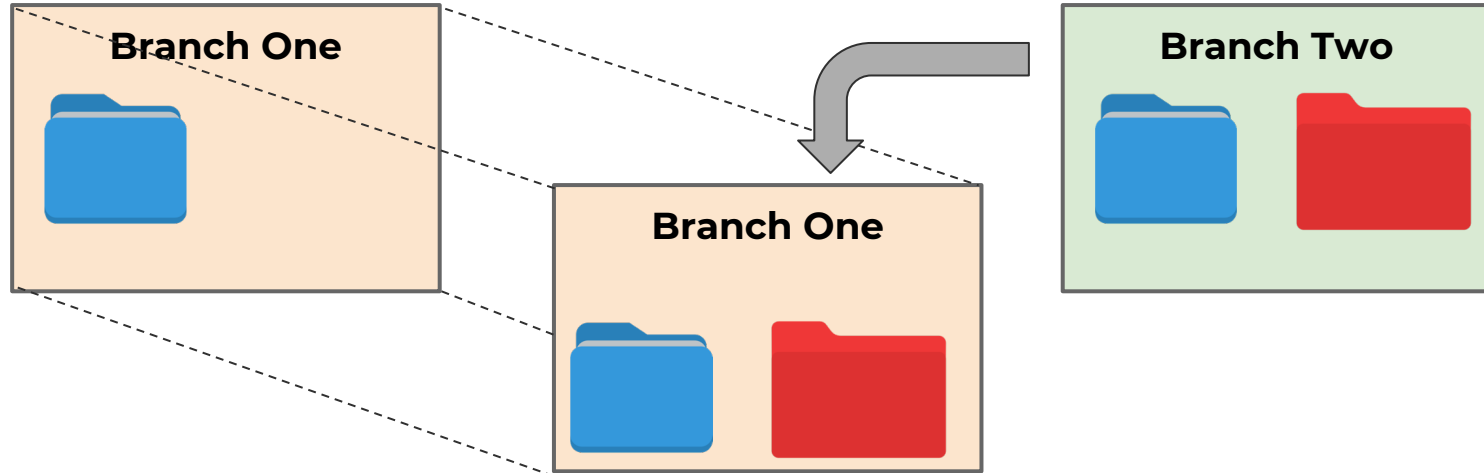
- Git will try to automatically create the merge, and can do so in cases where no information from a specific branch would be lost, for example:





## Week 3 Branches and Working with Others

- Git will try to automatically create the merge, and can do so in cases where no information from a specific branch would be lost, for example:



## Week 3 Branches and Working with Others

- However, there will be many instances where there are conflicts, for example changes in the file on lines that are different between the branches.
- These are known as **merge conflicts**, and we need to resolve (fix) the conflicts between the branches in order to merge them.

## Week 3 Branches and Working with Others

- Git will warn you about files in conflict.
- Then you must edit the files in order to remove the conflicts.
  - Fortunately, Git also provides specialized markdown to indicate the differences between the files and what differences come from which branch.
  - Modern editors (e.g. VS Code) have syntax highlighting to reflect this.

# Week 3 Branches and Working with Others

- Merge Conflict Example

```
$ cat merge.txt
<<<<<< HEAD
Some content from the text file
=====
Different content from the other branch
>>>>>> new_branch
```

# Week 3 Branches and Working with Others

- Merge Conflict Example

```
$ cat merge.txt
```

```
<<<<<< HEAD
```

```
Some content from the text file
```

```
=====
```

```
Different content from the other branch
```

```
>>>>>> new_branch
```

Content below this and above the ===== means that the content already exists in the current HEAD branch.

# Week 3 Branches and Working with Others

- Merge Conflict Example

```
$ cat merge.txt
```

```
<<<<<< HEAD
```

```
Some content from the text file
```

```
=====
```

```
Different content from the other branch
```

```
>>>>>> new_branch
```

Division line between the conflicting content between the branches.

# Week 3 Branches and Working with Others

- Merge Conflict Example

```
$ cat merge.txt
```

```
<<<<<< HEAD
```

```
Some content from the text file
```

```
=====
```

```
Different content from the other branch
```

```
>>>>>> new_branch
```

Content between ===== and >>>>branch is the content from the branch you are trying to merge from.