DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Branching, Merging, and Rebasing in Git

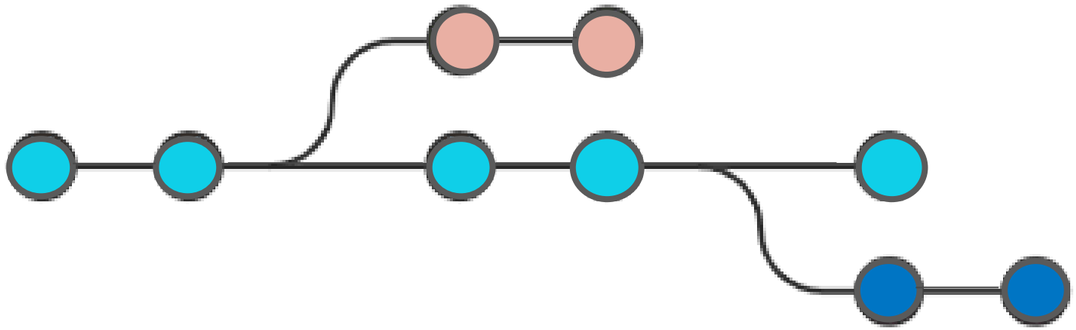
Learning Objectives

By the end of this lesson, you will be able to:

- Create, merge, and resolve conflicts in Git branches
- Explain fast forward and recursive merge in Git
- Demonstrate stashing and rebasing in Git

Branching in Git

Introduction to Branches



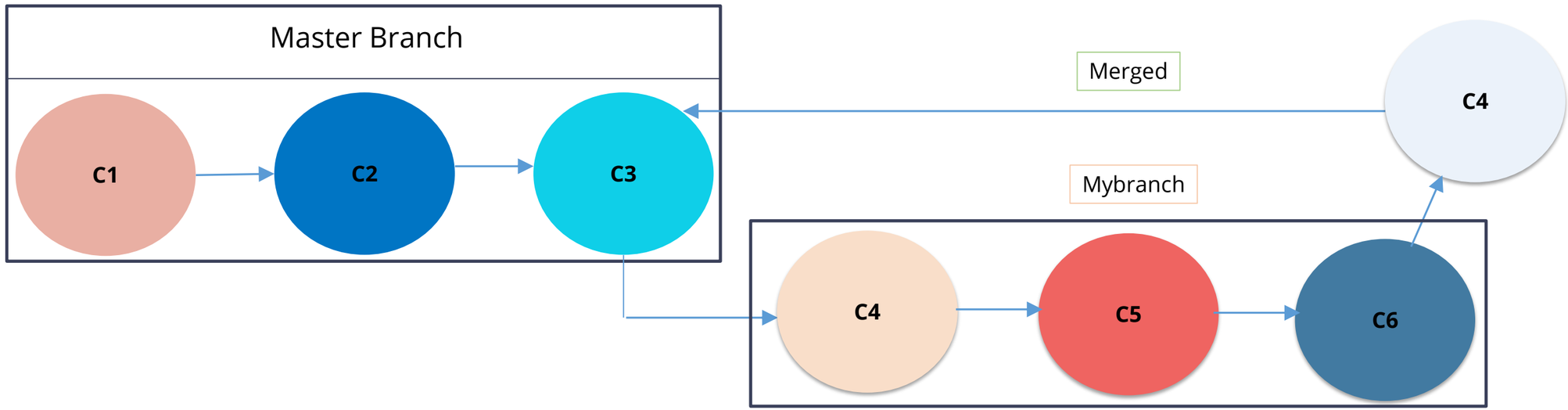
Git branches contain different lines of codes.
It is a separate workspace, usually created when you need to experiment or test something, without impacting the main code base.

Git branch is used to:

- Manage releases
- Create hotfixes to the production branches

Branch: Example

A branch is a method of requesting a new working directory and staging area.



NOTE

A Git branch is just a pointer to the commits; it does not change the contents of the repository.

Branch Commands

$ git branch <branch name>	Creates a new branch
$ git branch	Lists all branches in the current repository
$ git checkout <branch name>	Switches to branches
$ git merge <branch name>	Merges branches

NOTE

To create a new branch and switch to it, execute: $ git checkout -b<branch-name>

Assisted Practice

Branching in Git

Problem Statement: You are working on your project and have a couple of commits already on the master branch. You've decided that you're going to work on a new issue in a new branch. Create a new branch and switch to it at the same time.

Steps to Perform:

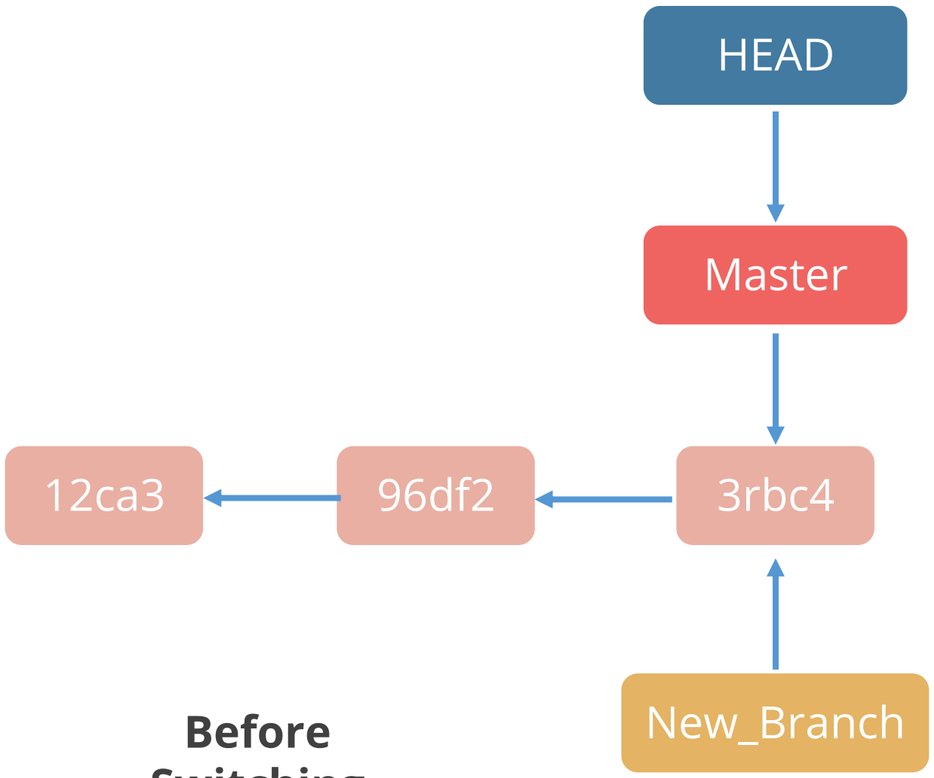
1. Listing all the branches in your repository
2. Creating a new branch
3. Verifying the creation of the new branch
4. Switching to different branches
5. Renaming an existing branch
6. Deleting the branch
7. Verifying the deletion of the branch

Switch between the Branches

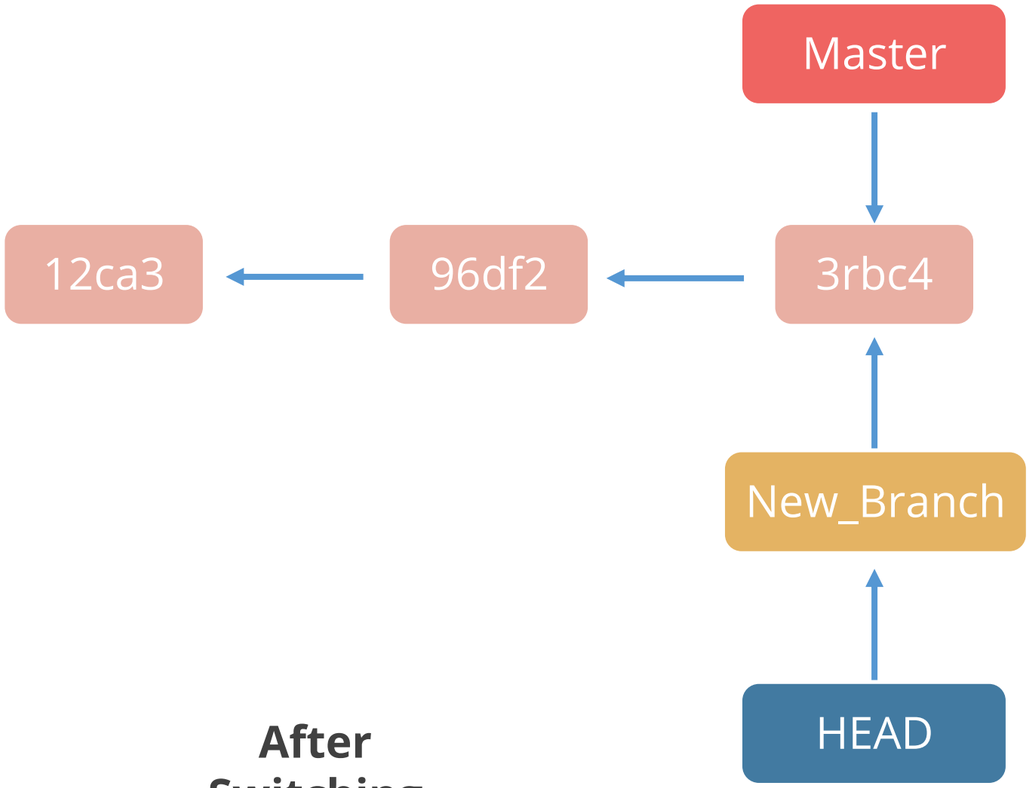
Switching Branches

"**Git checkout <existing branch>**" can be used to switch to an existing branch.

Git checkout New_Branch



Before Switching



After Switching

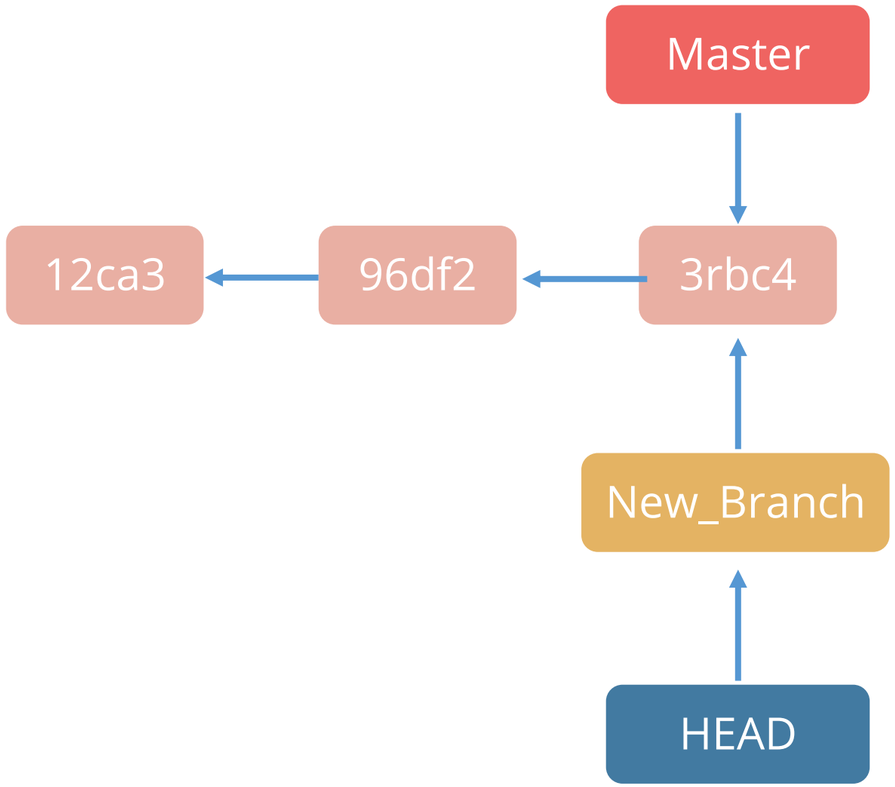
Caltech | Center for Technology & Management Education

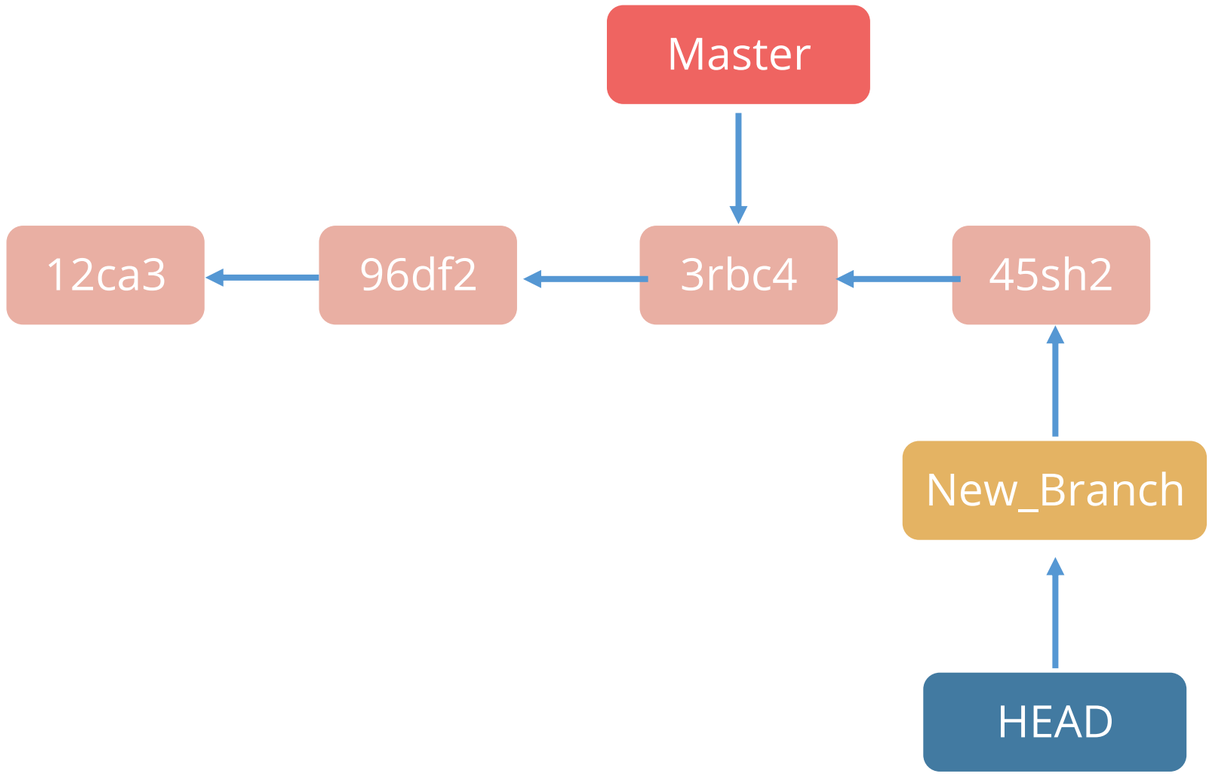
simplilearn

Switching Branches

Execute **vi <file_name>** followed by **git commit –a –m “message.“**

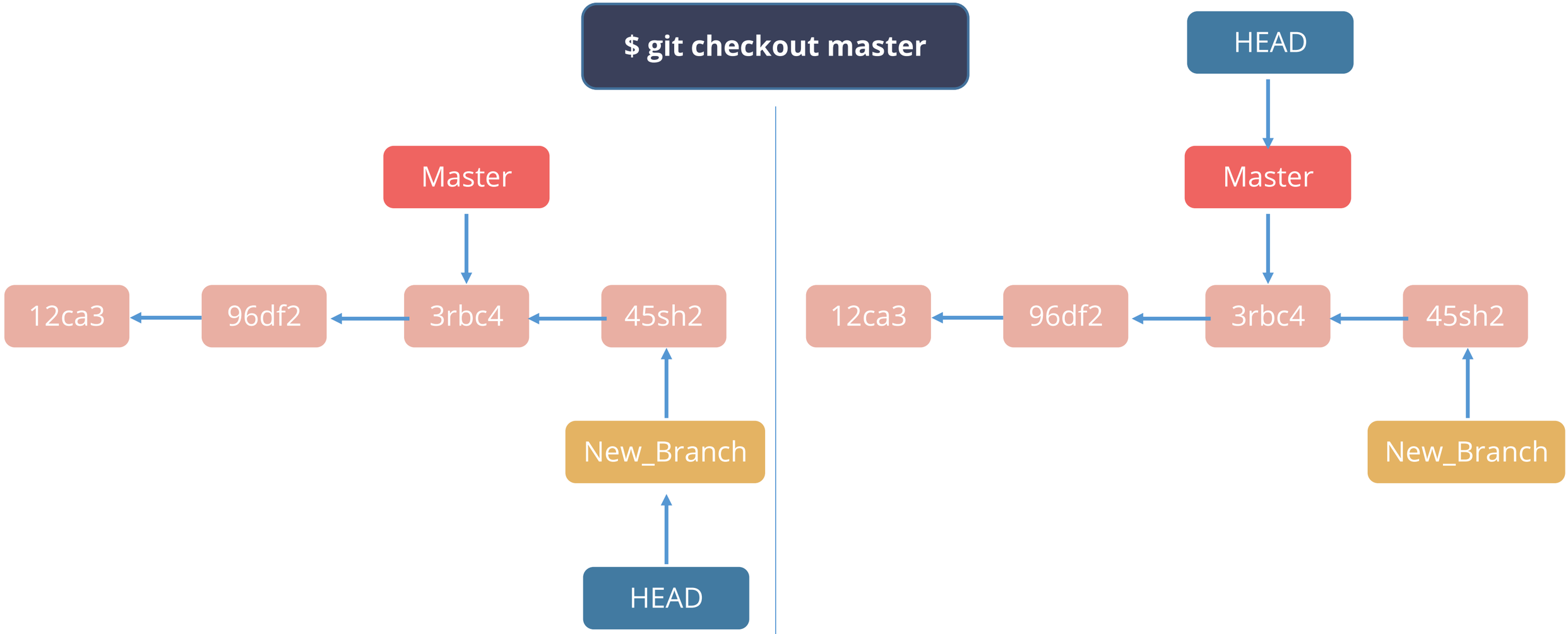
```
$ vi index.html
$ git commit –a –m “file edited”
```





Switching Branches

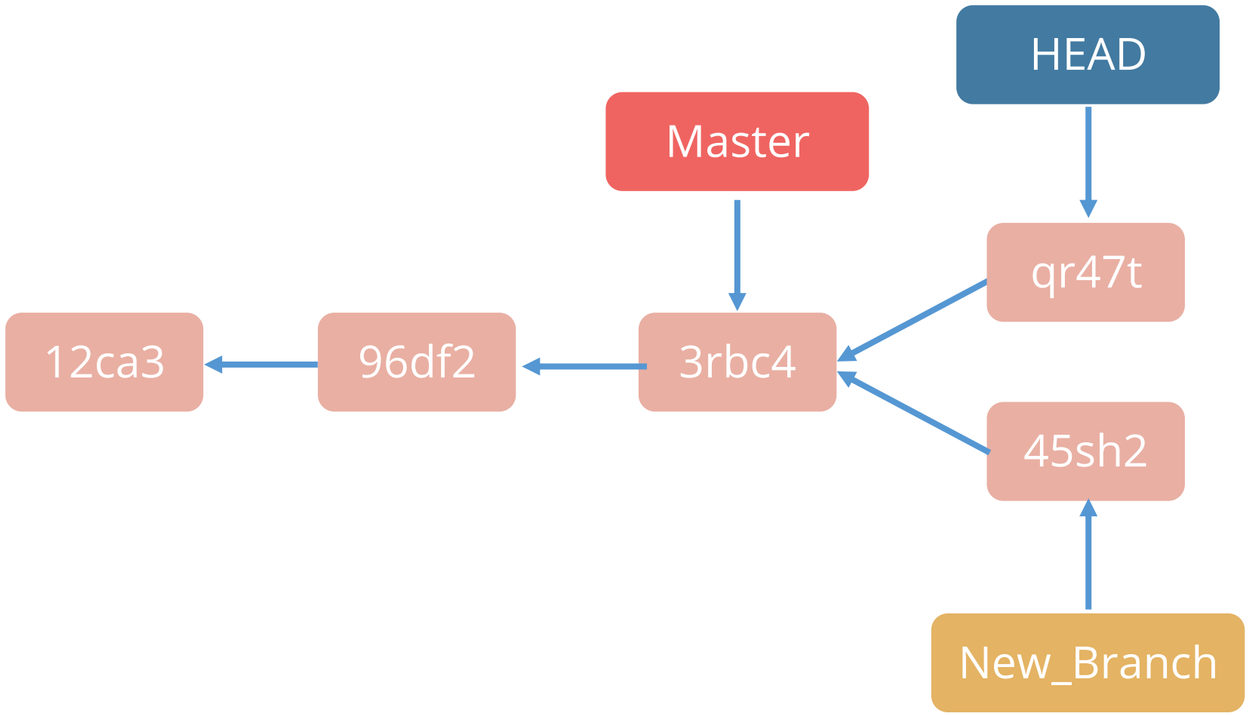
Execute **git checkout master** to move the HEAD pointer back to the master branch.



Switching Branches

Execute **vi <file_name>** followed by **git commit –a –m "message"** to make few more changes and commit again.

```
$ vi index.html
$ git commit –a –m "some more changes"
```



The branch histories have diverged.

Assisted Practice

Switching between Branches

Problem Statement: While working in your local repository, you wish to checkout and work on branch code rather than the main code line. Switch over to the new branch and add commits to it.

Steps to Perform:

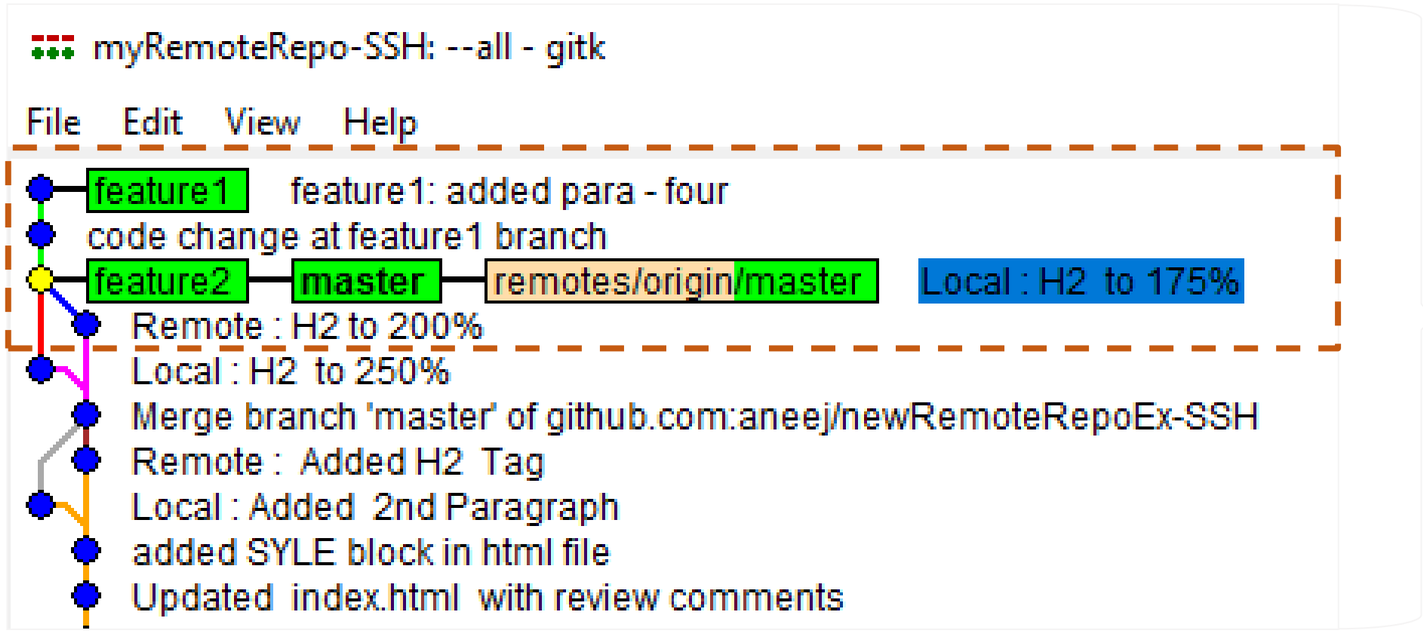
1. Creating a new branch
2. Switching to the new branch
3. Creating a file and committing the changes
4. Checking the git status
5. Reviewing the git logs

Merging Branches in Git

Merging Branches: Prerequisites

Analyze the difference → Prevent conflicts

$ git diff



This comes in handy when there are multiple users working across multiple branches.

Steps to Merge Branches

Switch to the branch you want to merge

Execute: **$ git merge <branch name>**

Assisted Practice

Merging Branches in Git

Problem Statement: Suppose your work on a new merge is complete and ready to be merged into your master branch. In order to do that, merge your issue into the master branch.

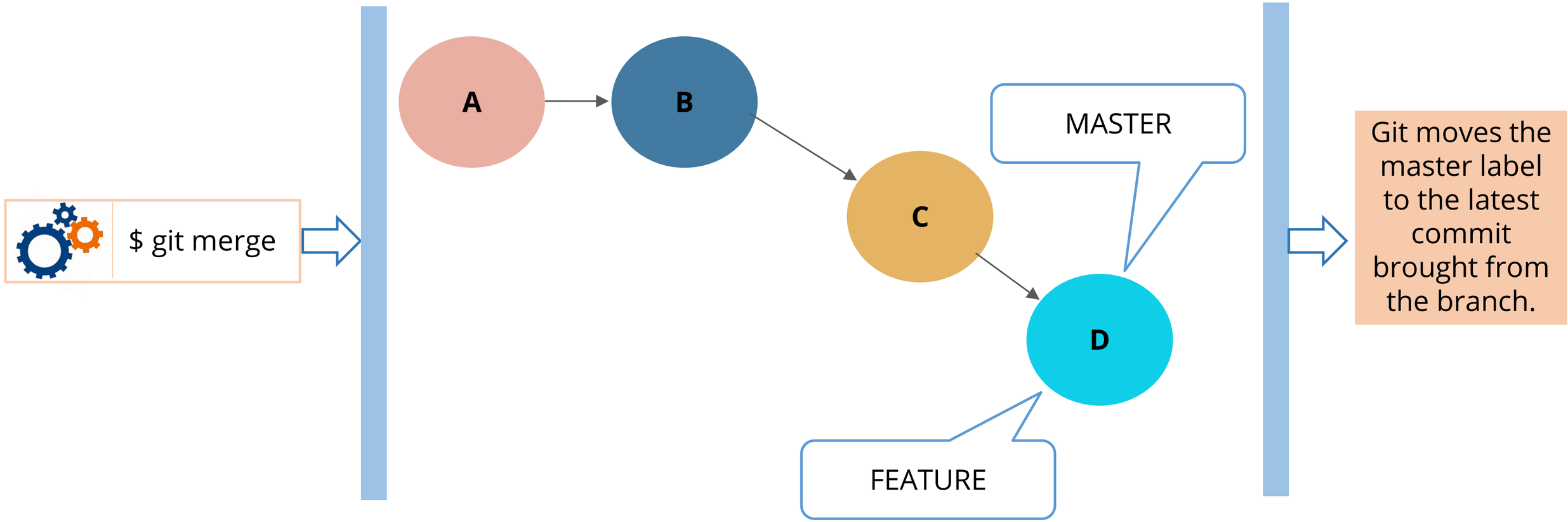
Steps to Perform:

1. Creating a new branch
2. Creating a new file in the new branch
3. Switching to the master branch
4. Merging the branches

Fast Forward and Recursive Change

Fast Forward Merge

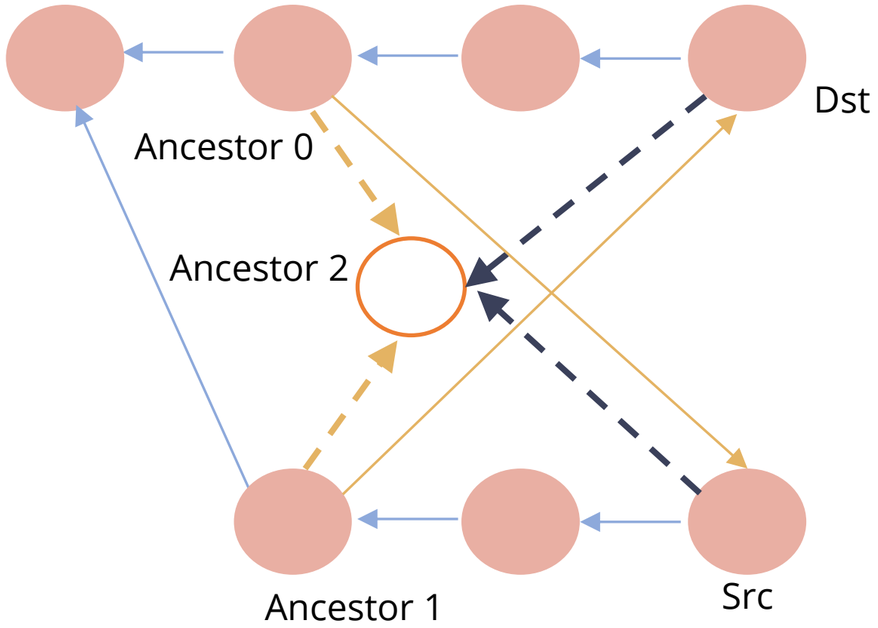
Fast forward merge is employed when you merge into a branch, the latest commit of which is your parent.



Hotfix is needed over the production code base → Used to fix bugs

Recursive Merge

This is the default merge strategy when pulling or merging one branch. When more than one valid ancestor is found, the recursive-merge strategy will create a new unique "virtual ancestor" merging the ones initially found.



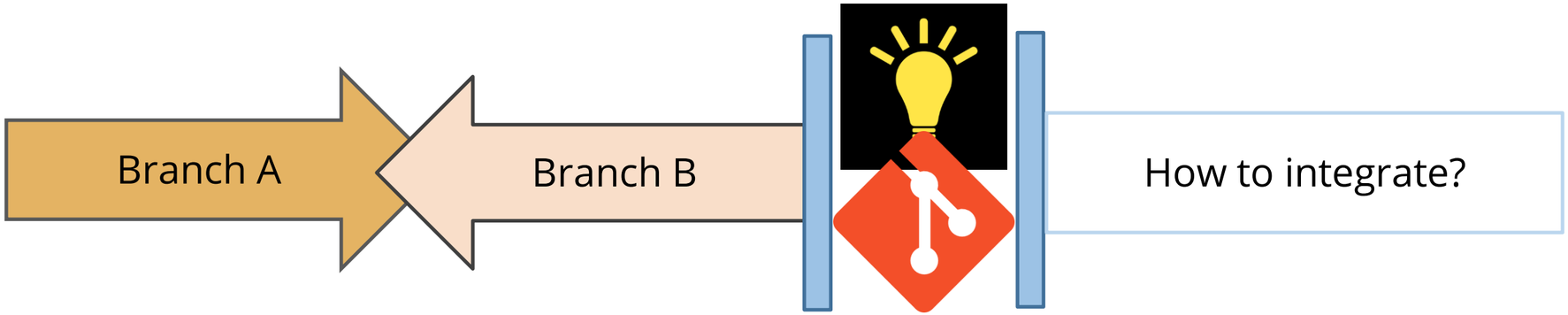
Merge ancestors to create a new virtual tree which will be the ancestor of the next image

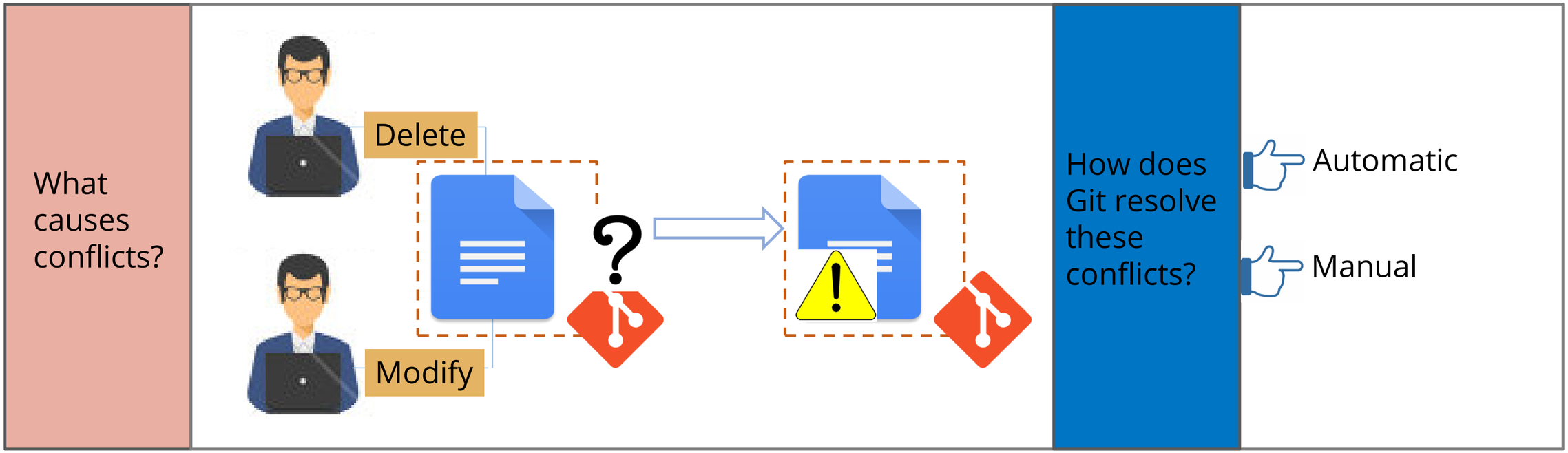
Fast Forward Merge Vs. Recursive Merge

Fast Forward Merge	Recursive Merge
No new commits on the master	New commits on the master
Linear History	Commit two parents
No merge commits	Merge commit is created
git rebase	git merge–no-ff

Resolving Merge Conflicts in Git

Merging Conflicts Across Branches





Assisted Practice

Resolving Merge Conflicts on Delete

Problem Statement: You edited a file in one branch and your co-worker removed the same file in another branch in the same repository. You'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

Steps to Perform:

1. Navigating to the local Git repository that has the merge conflict
2. Generating a list of the files affected by the merge conflict
3. Adding the removed file back to the repository
4. Committing the changes with a comment

Assisted Practice

Resolving Merge Conflicts on Modifications

Problem Statement: A merge conflict has occurred by competing line changes, such as when you make different changes to the same line of the same file on different branches in your Git repository. Resolve the conflicts before the final merge.

Steps to Perform:

1. Creating a file with some content in it
2. Adding the file to the repository and committing it
3. Creating a new branch
4. Overwriting the content in the previously created file
5. Committing the new content
6. Checking the Git status to identify the merge conflicts
7. Using git add to stage the newly merged content

Stashing in Git

Stashing Across Branches

Revert to the last commit, without interrupting the current work.

$ git stash

Records

Reverts

$ git stash list

Snapshots

$ git stash apply

$ git stash pop command

The “pop” command removes the stash snapshot from the stash list.

Assisted Practice

Stashing in Git

Problem Statement: When you were working on a part of the project, things were in a complex state, and you wanted to switch branches for a while to work on something else. Use stashing to save your modified tracked files and staged changes on a stack of unfinished changes that you can reapply any time.

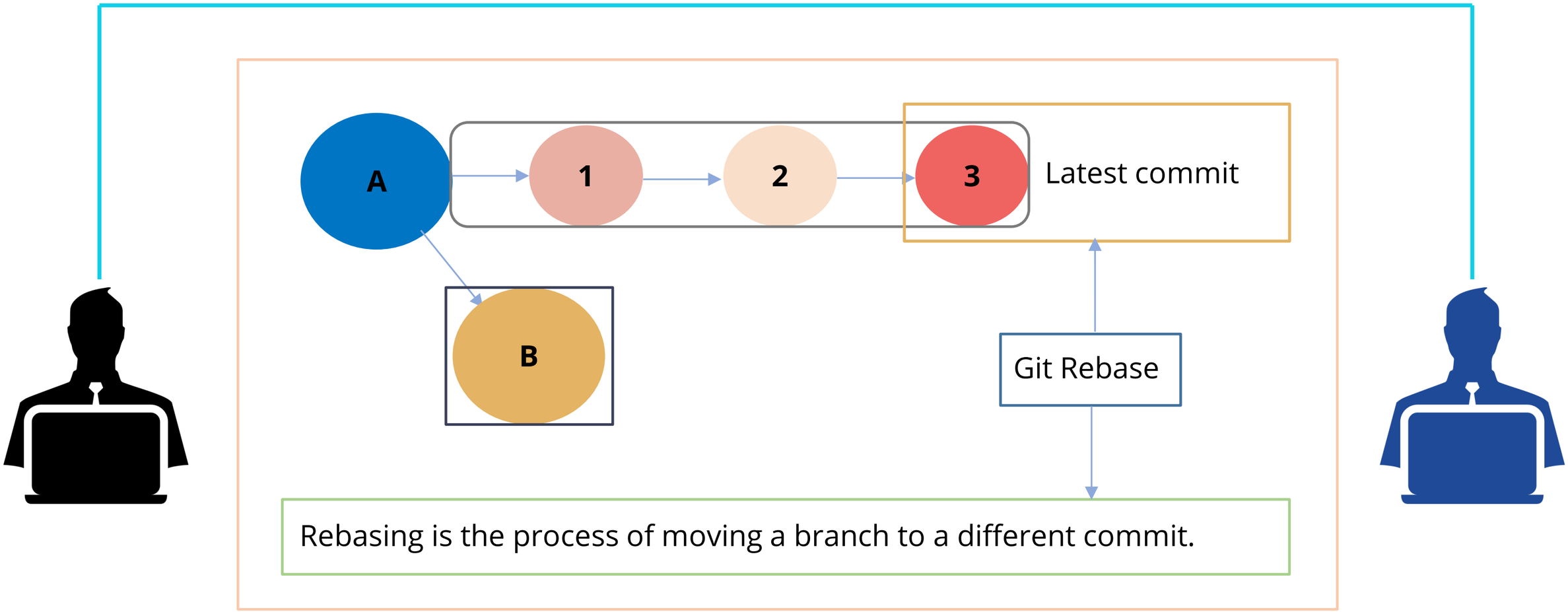
Steps to Perform:

1. Working on a file and stage one of the changes
2. Checking the git status
3. Using git stash to stash the changes
4. Using git stash list to see which stashes you've stored

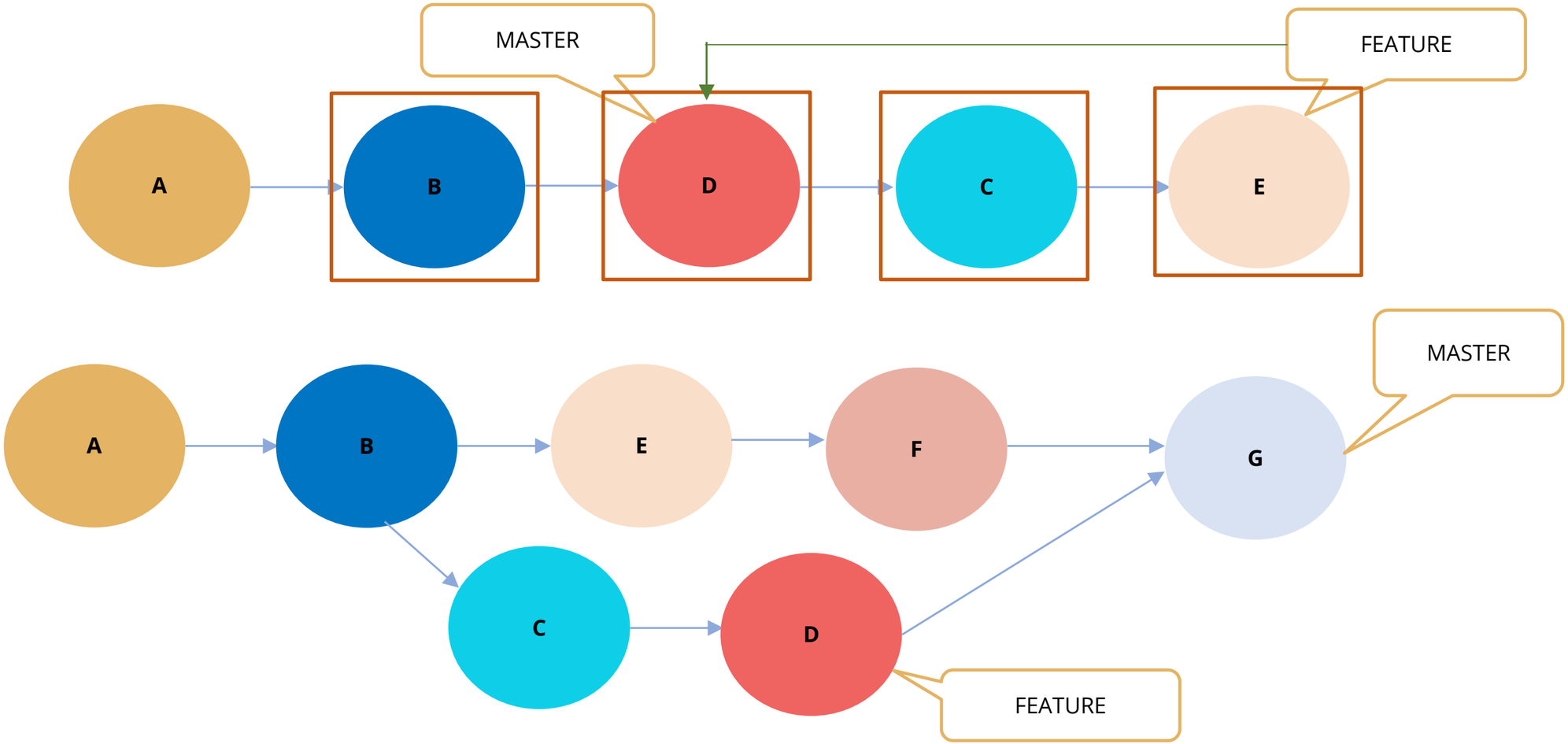
Rebasing in Git

Why Rebase?

Collaboration is essential in a development environment.



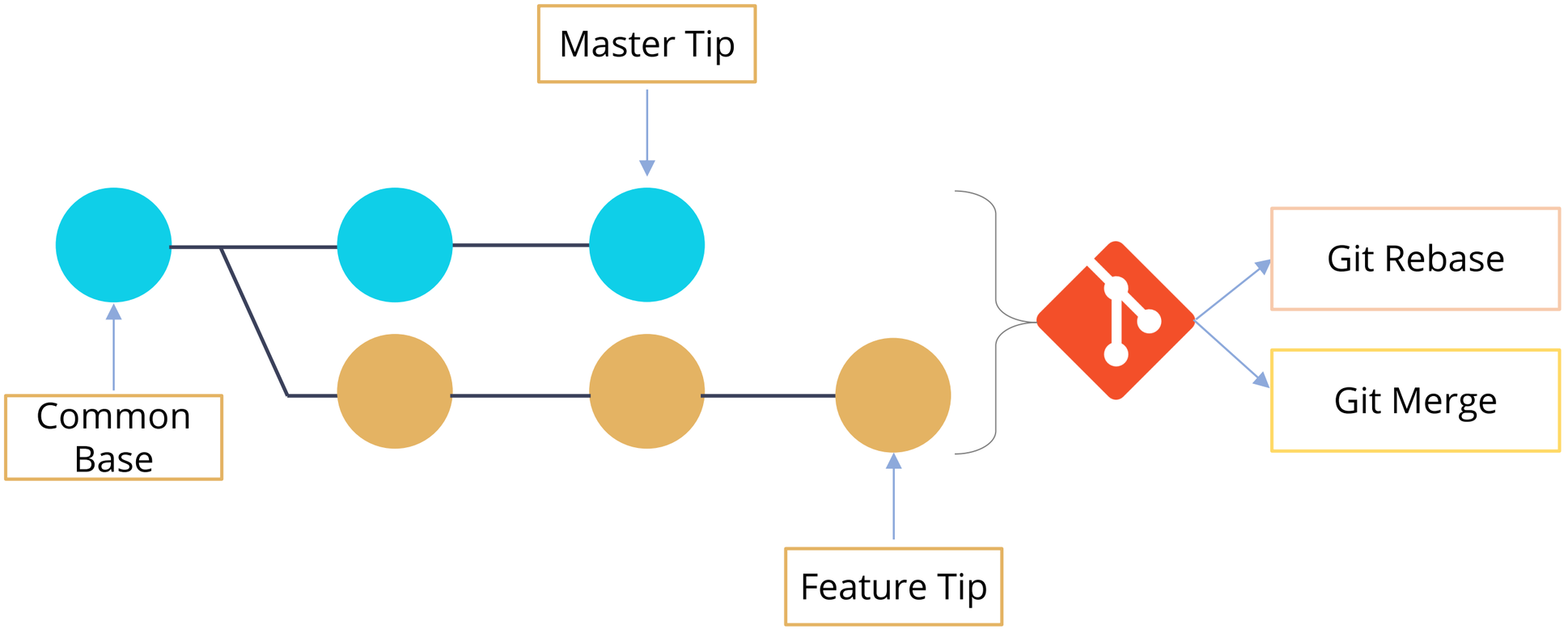
Rebase: Example



$ git rebase master → Rebase the current branch to the latest commit in master

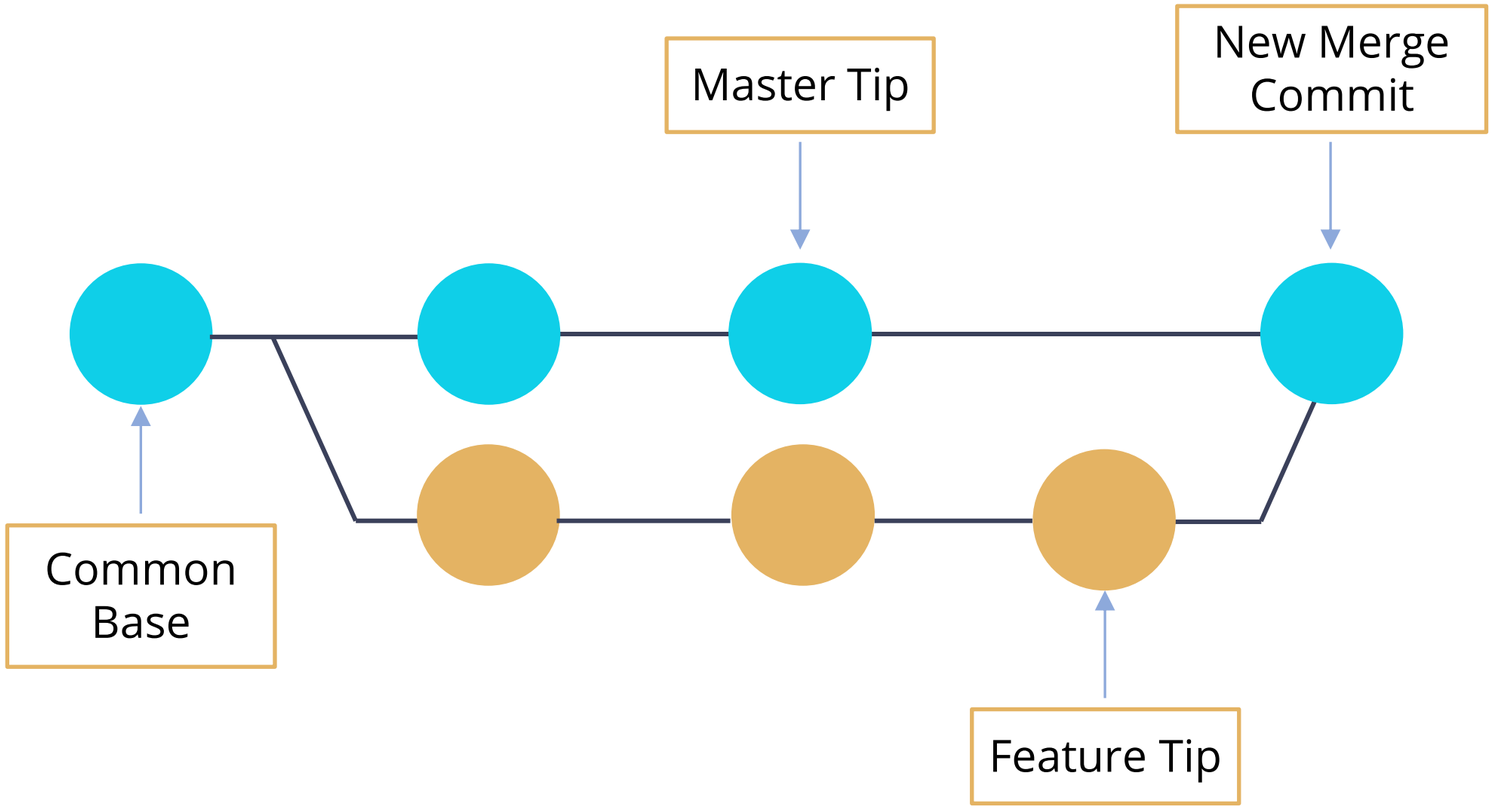
Rebase vs. Merge

Git Rebase | Git Merge → Integrate changes across branches



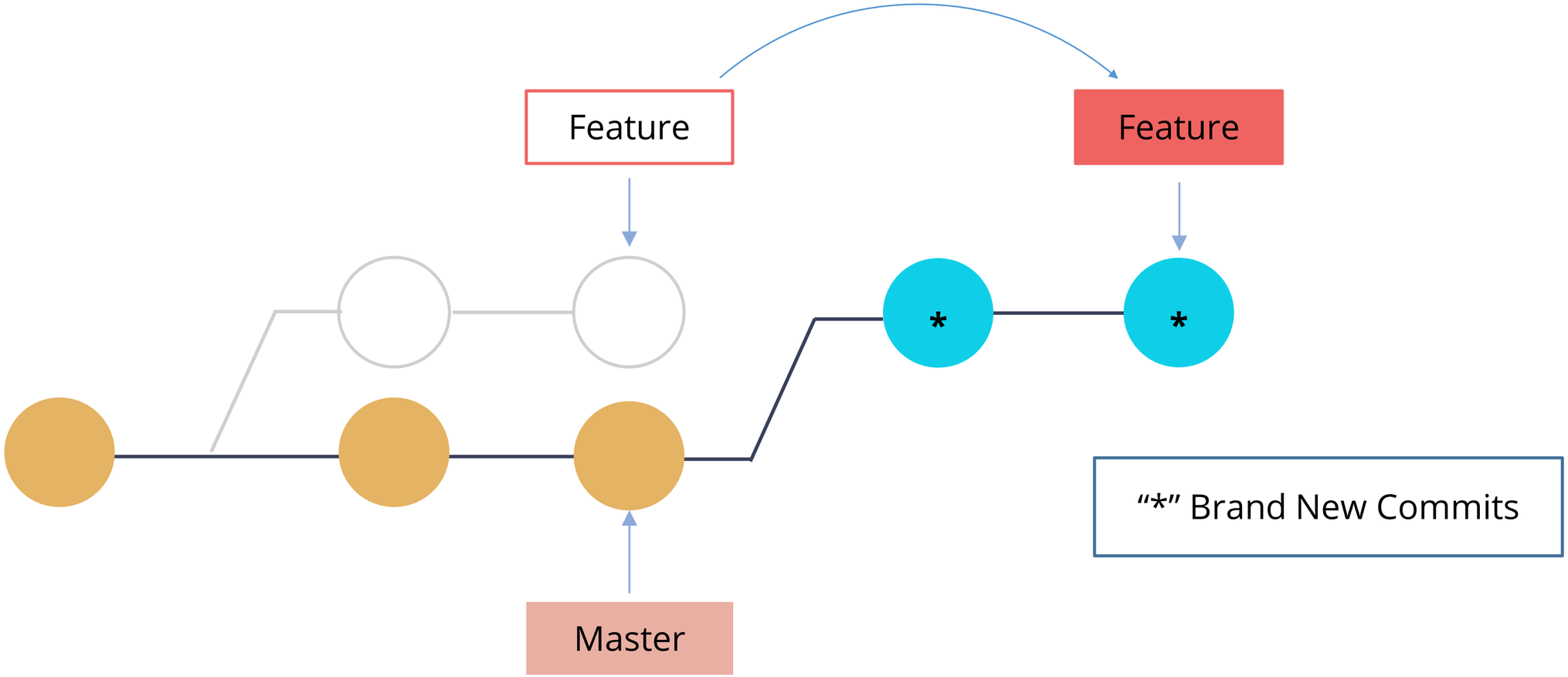
Rebase vs. Merge

Merge: It takes the contents of the feature branch and integrates it with the master branch.



Rebase vs. Merge

Rebase: It moves or combines a sequence of commits to a new base commit.



NOTE

Never use a "git rebase" on public branches.

Assisted Practice

Rebasing in Git

Problem Statement: The master branch has progressed since you started working on a feature branch. You want to get the latest updates to the master branch in your feature branch, but you want to keep your branch's history clean. Use rebasing to maintain a linear project history and the get the latest updates.

Steps to Perform:

1. Creating a feature branch
2. Editing files in the master branch
3. Executing git rebase

Key Takeaways

- Fast forward merge is employed when you merge into a branch for which the latest commit is the parent branch.
- A Git branch is just a pointer to the commits; it does not change the contents of the repository.
- Rebasing is the process of moving a branch to a different commit.

Lesson-End Project

Fix Merge Conflicts

Problem statement: You have created a repository with an attached remote repository. Another developer has worked on the same file and this has caused conflicts between the two changes. Resolve the conflicts before merging the changes.

Following events have occurred:

1. Developer A and Developer B fetch the remote repository for the latest commits
2. Developer A commits changes to a file in their local repository and push the changes to the remote repository
3. Developer B commits changes to the same file in their local repository and try to push the changes to the remote repository

In this scenario, Git is unable to fast-forward the changes from Developer B, because Developer A has made several changes. This will create a merge conflict due to different versions of the same file in local and remote repository.