Git and GitHub Training

Branching, Merging, and Rebasing in Git

simplilearn

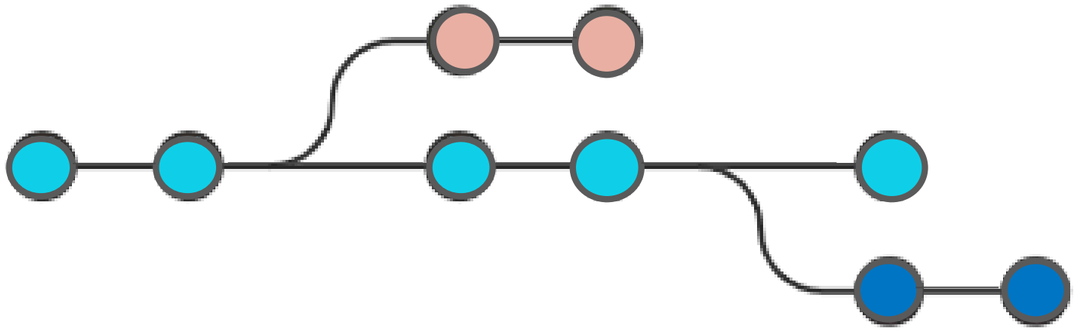
Learning Objectives

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



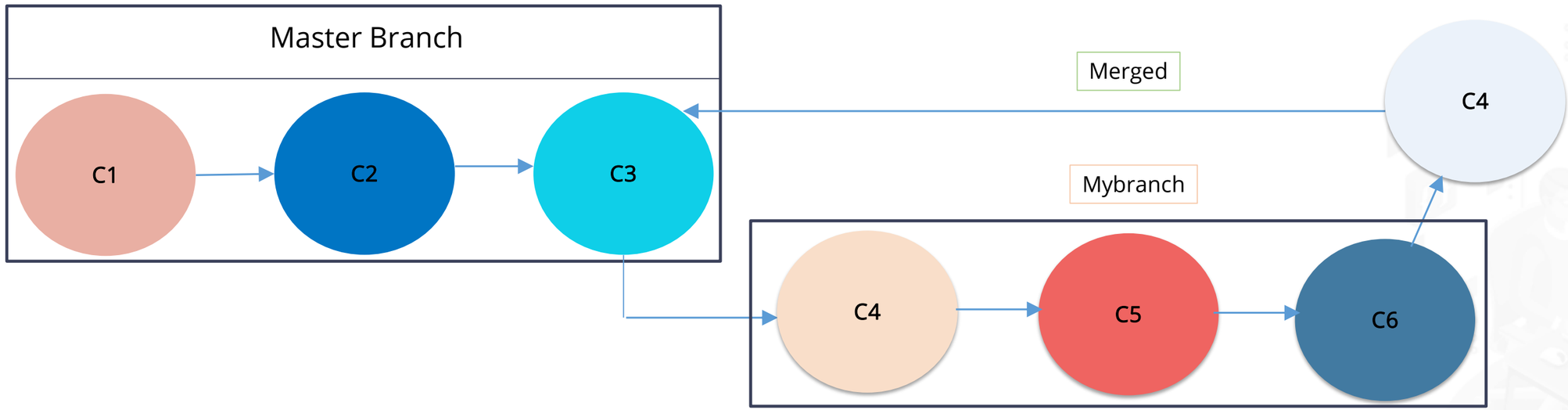
Git branches contain different lines of codes.
It is a separate workspace, usually created when you need to experiment or test something, without impacting the main code base.

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>

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:

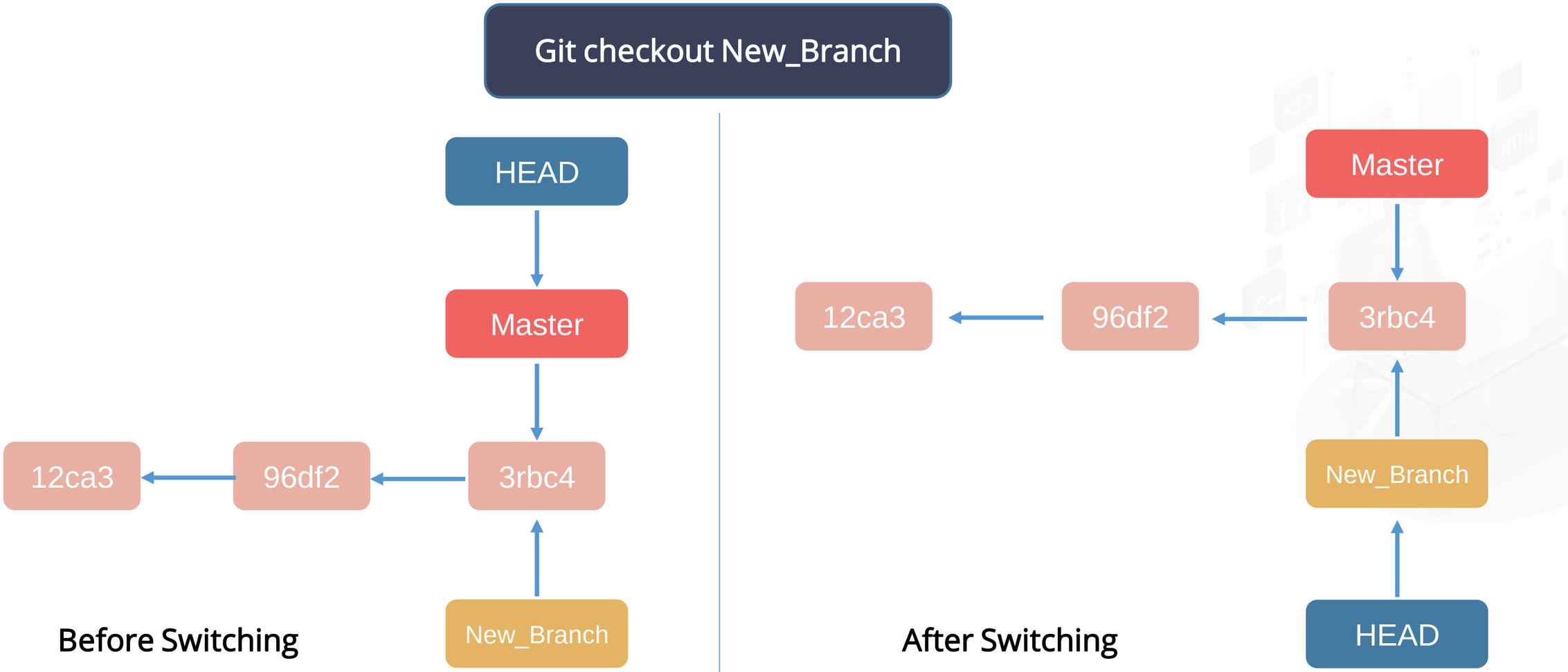
- Create a new branch
- Perform some operations on the master branch
- Switch to the new branch
- Rename the new branch
- Delete the new branch

ASSISTED PRACTICE

Switch between the Branches

Switching Branches

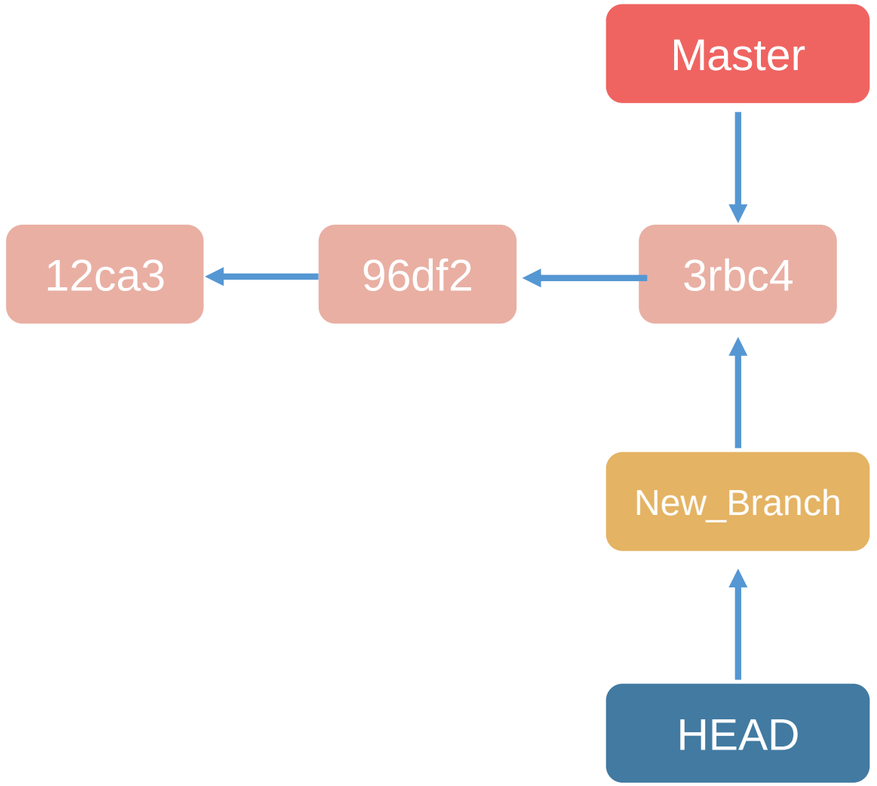
"**Git checkout <existing branch>**" can be used to switch to an existing branch.

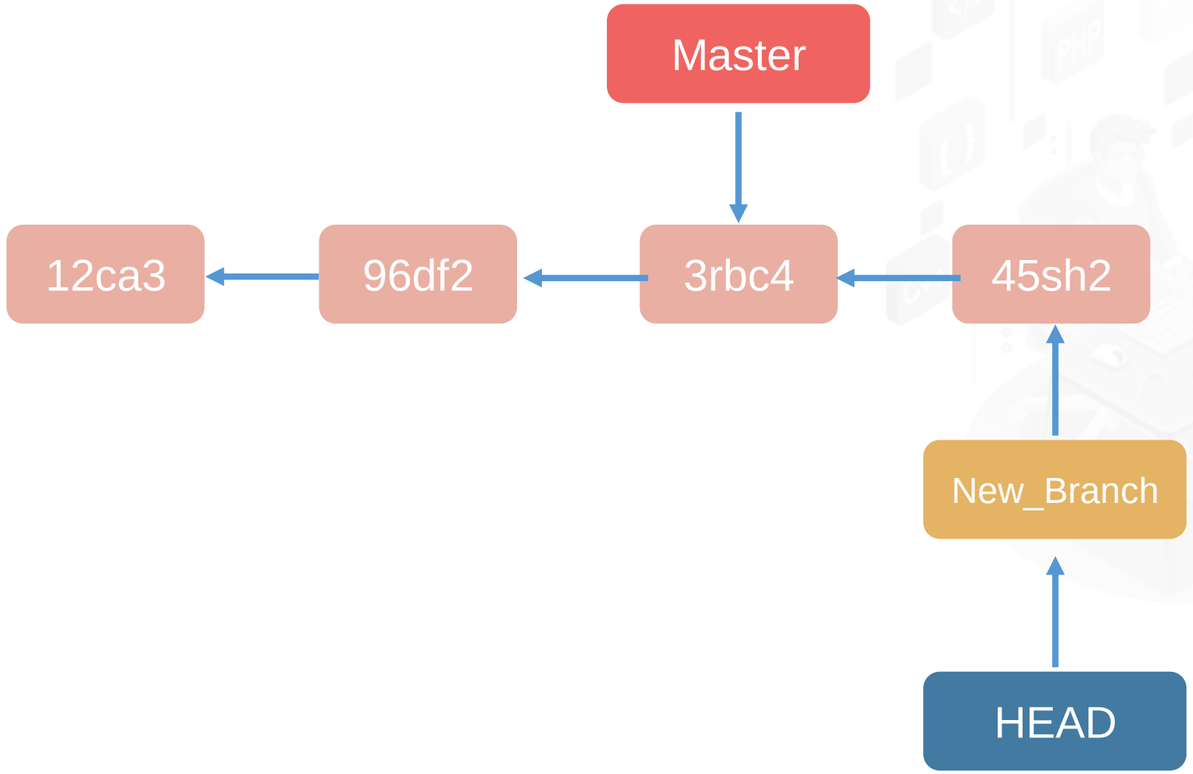


Switching Branches

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

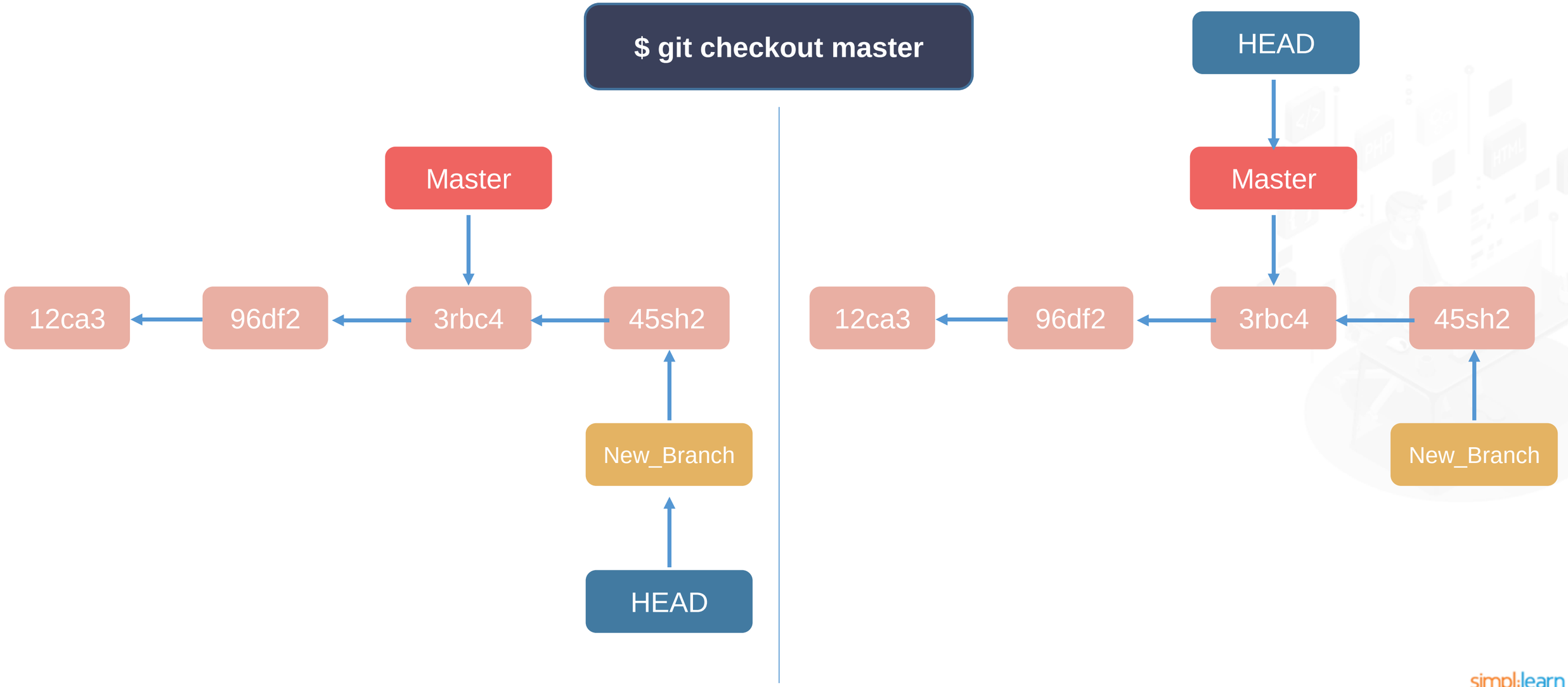
```
$ vi index.html
$ git commit –a –m “file edited”
```





Switching Branches

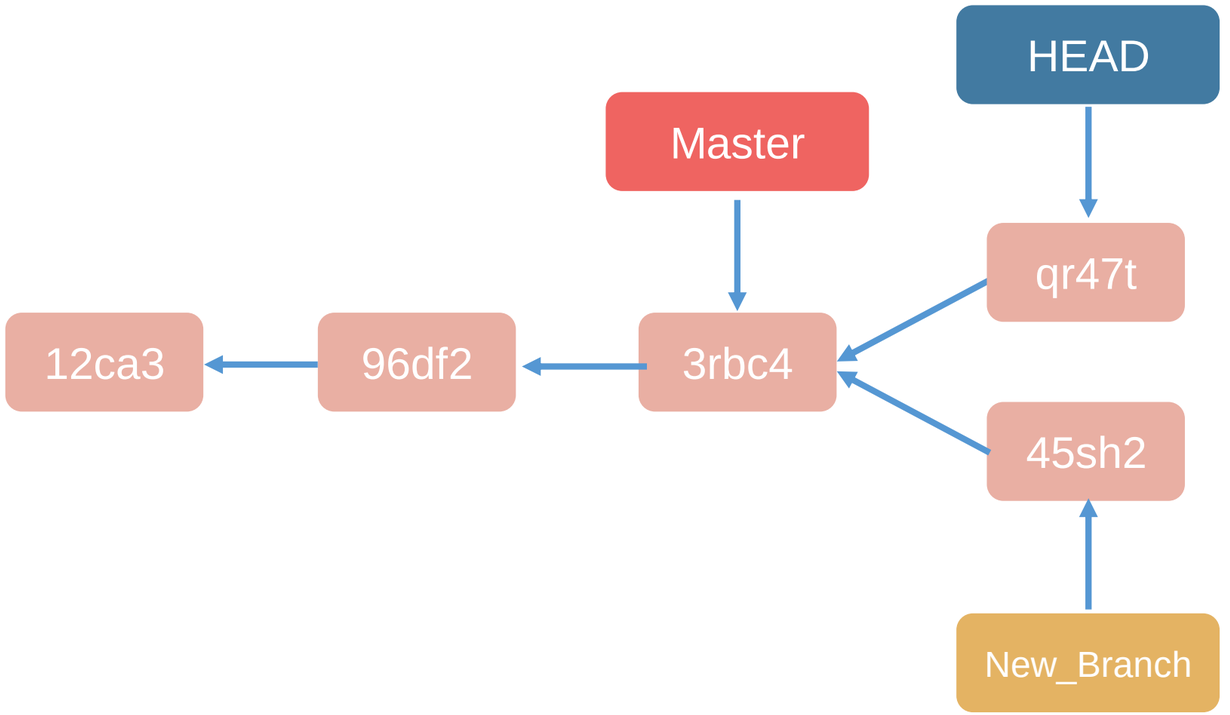
Execute **git checkout master** to move the HEAD pointer back to the master branch.



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.

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:

- Create a new branch
- Switch to the new branch
- Perform operations on the new branch
- Switch back to master branch

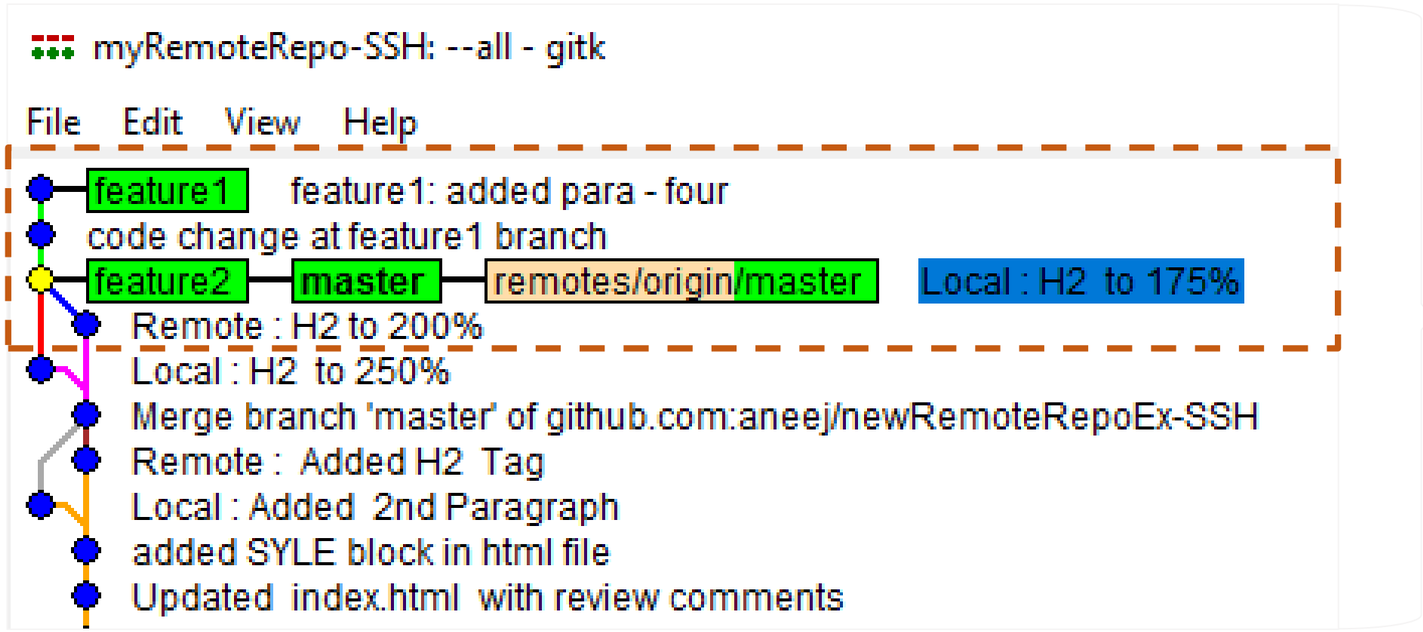
ASSISTED PRACTICE

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>**

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:

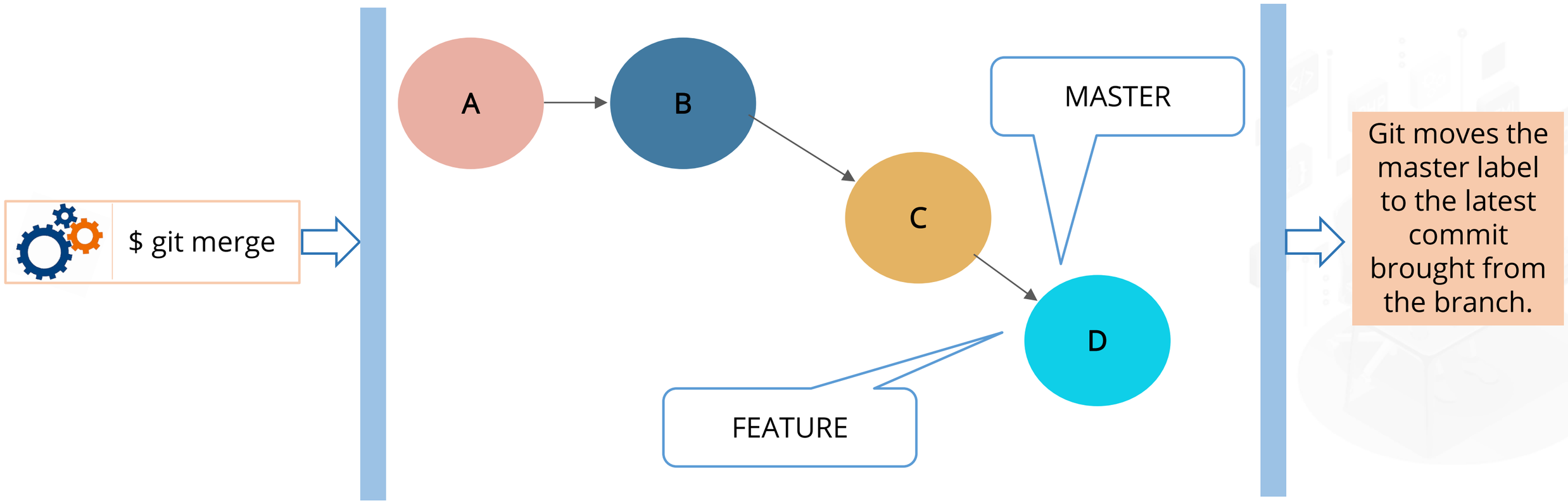
- Create a new branch
- Switch to the branch
- Do some work on the branch
- Commit the changes
- Check out the branch you wish to merge into and then run the **git merge** command

ASSISTED PRACTICE

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

Fast Forward Merge in Git

Problem Statement: Demonstrate fast-forward merge to integrate the histories of commit by moving the current branch tip to the target branch tip.

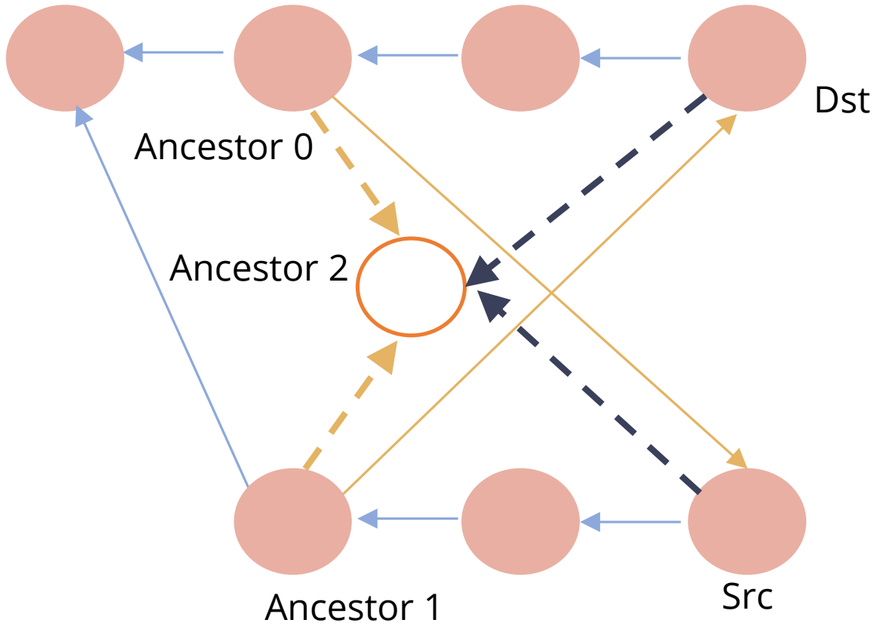
Steps to Perform:

- Start a new feature
- Edit some files
- Merge in the new feature branch
- Delete the feature branch

ASSISTED PRACTICE

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

Recursive Merge in Git

Problem Statement: Suppose you have found two common ancestors. Use recursive-merge strategy to create a new virtual ancestor, merging the ones initially found.

Note: Ancestor is the changeset (or commit) which is the nearest parent of the source and the destination.

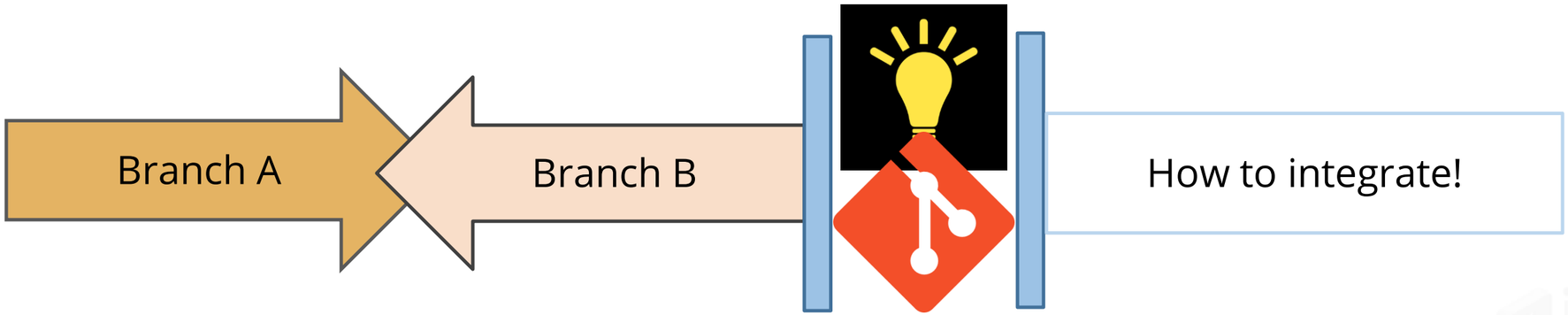
Steps to Perform:

- Create a new branch and perform multiple commits on a file (Example: Index.html)
- Switch to the master branch
- Modify the file and commit the changes
- Execute **git merge <branch name>** command to merge the feature branch into the master
- Execute **git log** which shows the new commit

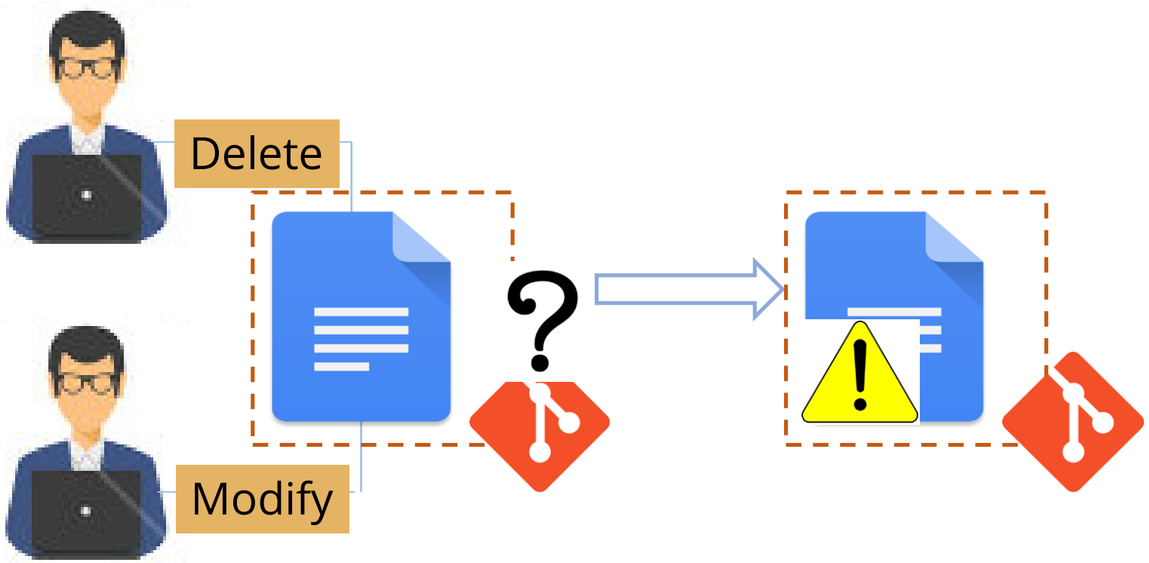
ASSISTED PRACTICE

Resolving Merge Conflicts in Git

Merging Conflicts Across Branches



What causes conflicts?



How does Git resolve these conflicts?

- Automatic
- Manual

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:

- Navigate to the local Git repository that has the merge conflict
- Generate a list of the files affected by the merge conflict
- Open your favorite text editor and navigate to the file that has merge conflicts
- Execute **git add <file name>** to add the removed file back to your repository
- Commit your 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:

- Create a file with some content in it
- Add the file to the repository and commit it
- Create and checkout a new branch
- Overwrite the content in the previously created file
- Commit the new content
- Check the git status to identify the merge conflicts
- Use **git add <file name>** to stage the new merged content
- Create a new commit to finalize the merge

ASSISTED PRACTICE

Stashing in Git

Stashing Across Branches

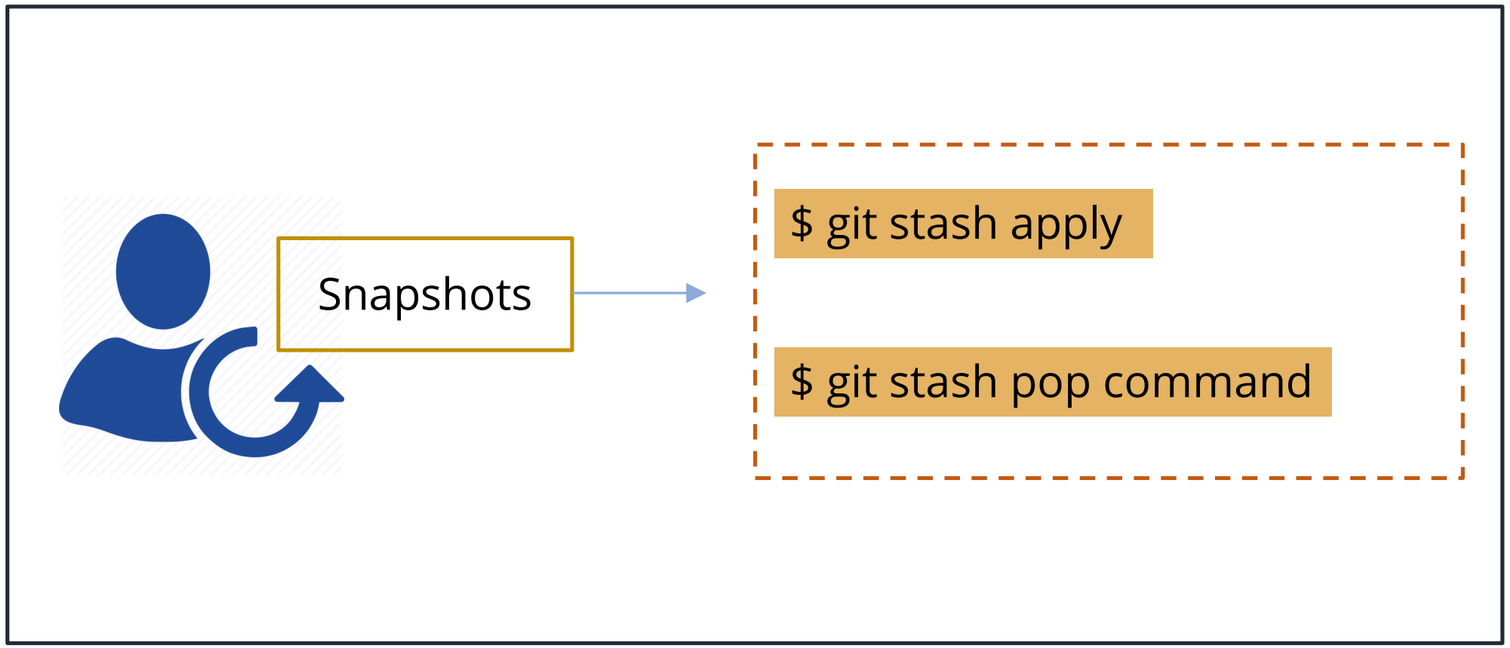
Revert to the last commit, without interrupting the current work.

$ git stash

Records

Reverts

$ git stash list



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

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:

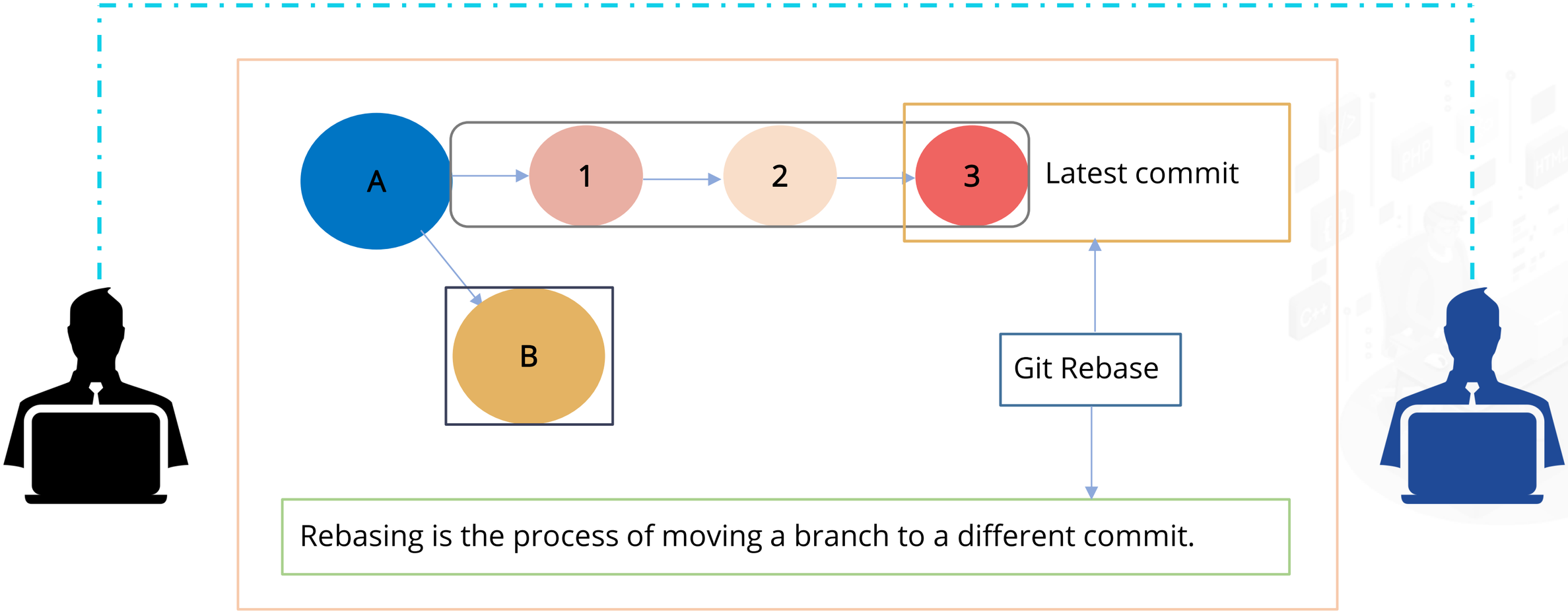
- Work on a 2-3 files and stage one of the changes.
- Check the git status
- Use **git stash** to stash the changes
- Use git stash list to see which stashes you've stored

ASSISTED PRACTICE

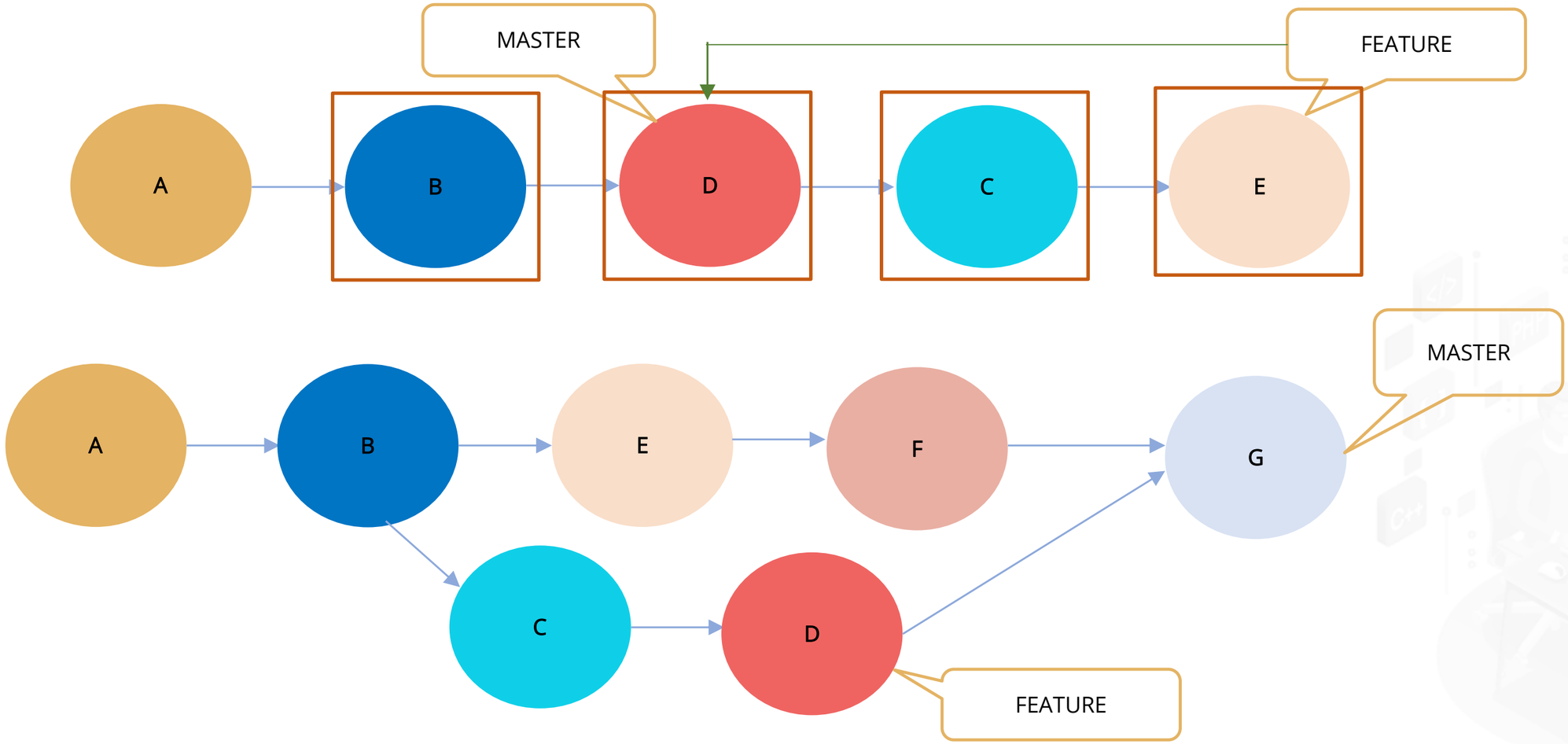
Rebasing in Git

Why Rebase?

Collaboration is essential in a development environment.



Rebase: Example

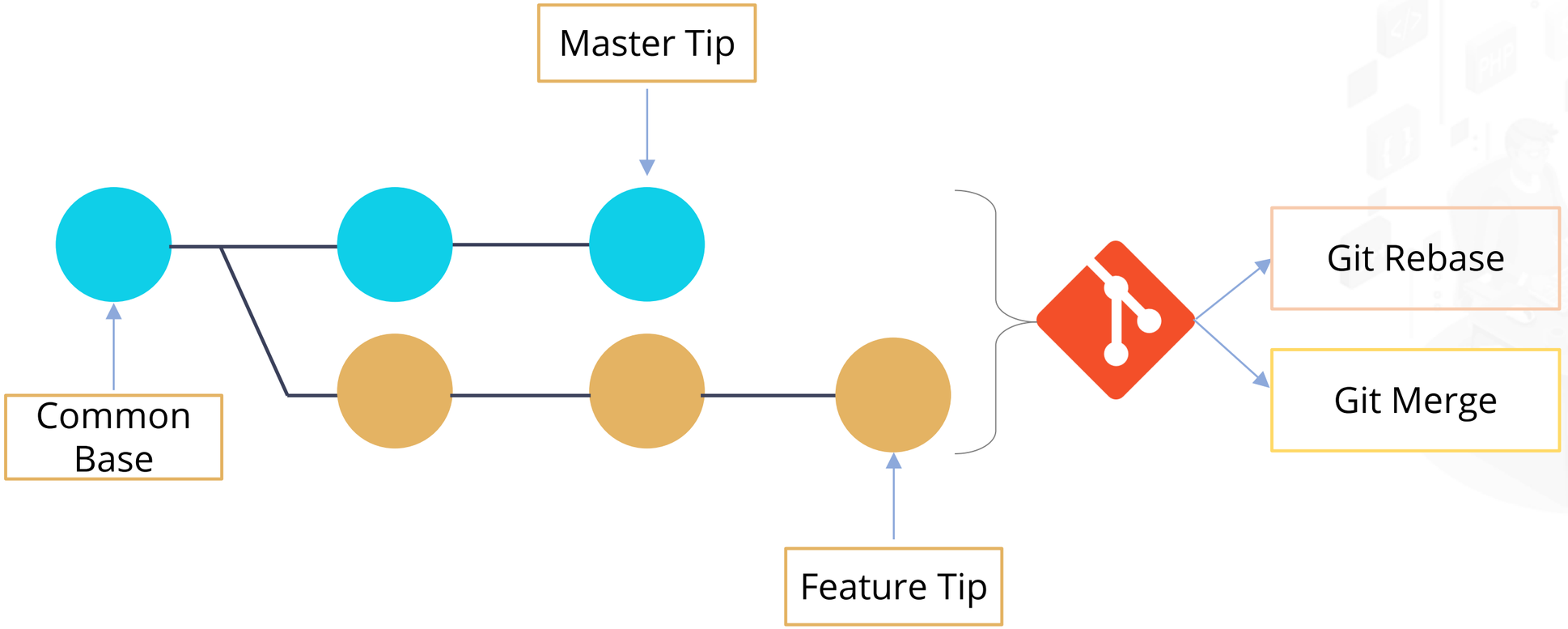


$ git rebase master

Rebase the current branch to the latest commit in master

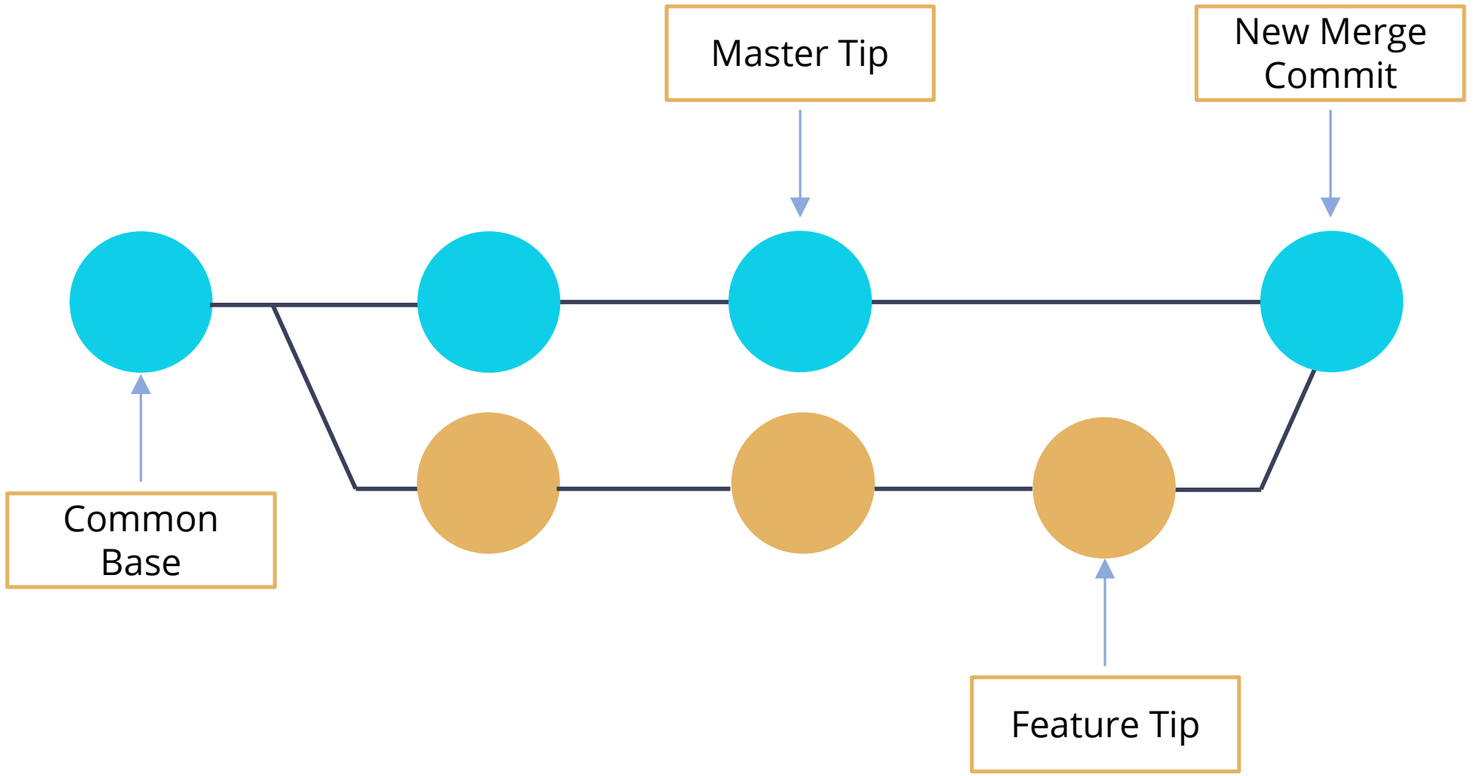
Rebase vs. Merge

Git Rebase | Git Merge → Integrate changes across branches



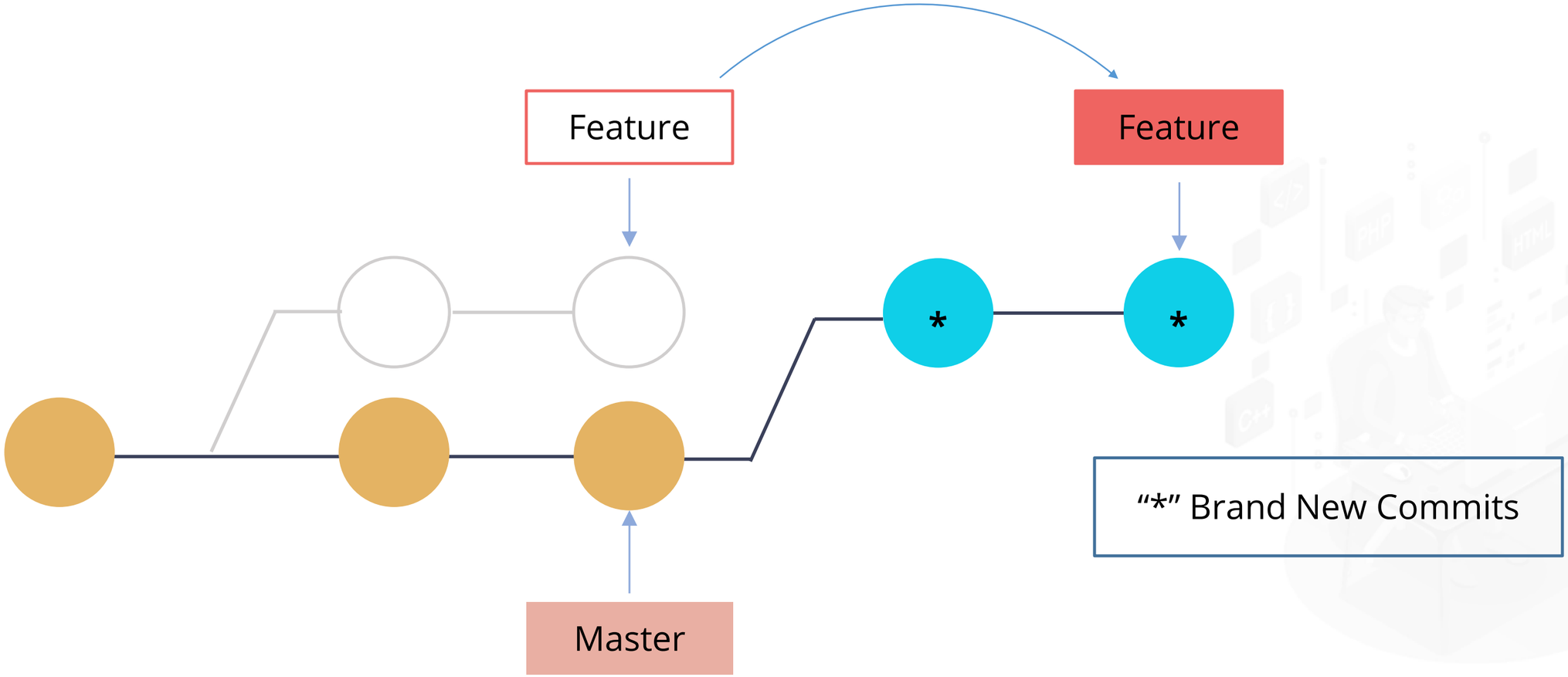
Rebase vs. Merge

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.

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:

- Create a feature branch
- Edit files
- Execute git rebase <base>

ASSISTED PRACTICE

Key Takeaways

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

Knowledge Check

Knowledge Check 1	Which of the following is the correct Git command to switch to a newly created branch?

a. Command: git checkout –b <branch-name>

b. Command: git checkout <branch-name>

c. Command: git branch <branch-name>

d. Command: git branch –b <branch-name>

Knowledge Check 1	Which of the following is the correct Git command to switch to a newly created branch?

a. Command: git checkout –b <branch-name>

b. Command: git checkout <branch-name>

c. Command: git branch <branch-name>

d. Command: git branch –b <branch-name>

The correct answer is **a**

git checkout –b <branch-name> is used to switch to a newly created branch.

Knowledge Check

2

When would Git employ the “fast-forward” merge algorithm?

a. At all times

b. Only when you merge master commits to branches

c. Only when the branch has branched off the latest commit you want to merge to

d. Only when the branch is branched off from an older commit on the master branch

Knowledge Check

2

When would Git employ the “fast-forward” merge algorithm?

a. At all times

b. Only when you merge master commits to branches

c. Only when the branch has branched off the latest commit you want to merge to

d. Only when the branch is branched off from an older commit on the master branch

The correct answer is **c**

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

Knowledge Check

3

How can you view all the branches that exist in the repository that you cloned?

a. You can only view the master branch of the cloned repository

b. You can only view the active branch of the cloned repository

c. You can view them using the command “git branch –a”

d. You can view them using the command “git branch –show”

Knowledge Check 3	How can you view all the branches that exist in the repository that you cloned?

a. You can only view the master branch of the cloned repository

b. You can only view the active branch of the cloned repository

c. You can view them using the command “git branch –a”

d. You can view them using the command “git branch –show”

The correct answer is **c**

The “git branch –a” command will show all the branches.

Knowledge Check

4

How different is rebase from the merge functionality?

a. In merge, all commits from the point of divergence are brought into a new commit, whereas in rebase the branch commits are moved sequentially after the latest master commit.

b. Merge and rebase are the same.

c. Merge and rebase are the same except that for rebase to work, the latest commit in the master branch has to be the parent of the feature branch.

d. In merge, all commits from the point of divergence are merged into the latest master commit, whereas in rebase the branch commits are moved sequentially after the latest master commit.

Knowledge Check

4

How different is rebase from the merge functionality?

a. In merge, all commits from the point of divergence are brought into a new commit, whereas in rebase the branch commits are moved sequentially after the latest master commit.

b. Merge and rebase are the same.

c. Merge and rebase are the same except that for rebase to work, the latest commit in the master branch must be the parent of the feature branch.

d. In merge, all commits from the point of divergence are merged into the latest master commit, whereas in rebase the branch commits are moved sequentially after the latest master commit.

The correct answer is **a**

In merge, all commits from the divergent point are brought into a new commit, whereas in rebase the branch commits are moved sequentially after the latest master commit.

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:

- In your environment the changes from a remote repository has been fetched
- Developer A pulls the latest changes from Developer B.
- Developer B commits changes to their local repository.
- Developer A commits non-conflicting changes to their local repository.
- Developer A pulls the latest changes from Developer B.

In this scenario, Git is unable to fast-forward the changes from Developer B, because Developer A has made several changes.

LESSON-END PROJECT