Code.Hub

The first Hub for Developers

Ztoupis Konstantinos

Version Control Systems - Git

Code.Learn Program:
React

How developers work?

- Work in a team, probably on particular components?
- Integrate your code together.
- Make copies of your files in case something you lose them. (Not really?)

So what's version control?

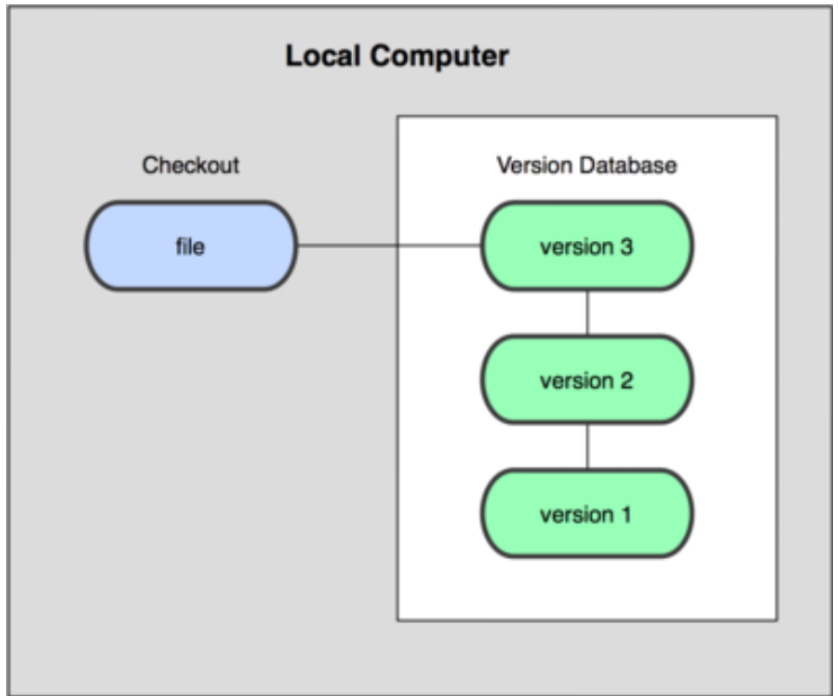
- Version control is the management of changes to documents, primarily computer programs.
- Also known as revision control or source control.
- Examples: git, mercurial, subversion

Why version control?

- Makes working in a team easy!
  1. Code without interference.
  2. Go back to a previous version
  3. Integrate code of multiple developer's easily.
  4. Know who did what, when.

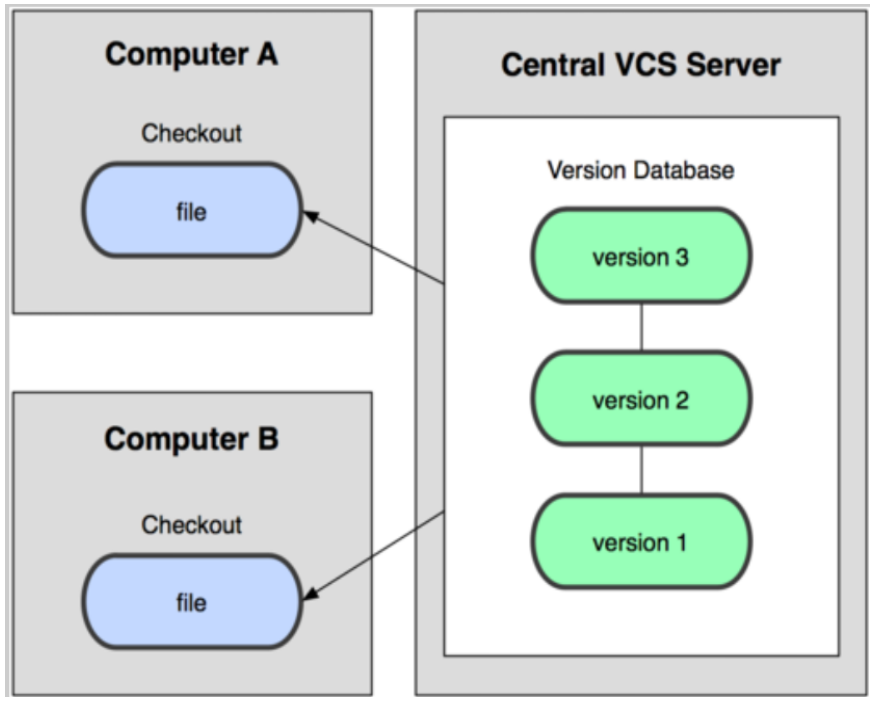
- Keep your code secure.

Version Control Systems (VCS)



- Very simple
- No cooperation between devs
- Not online

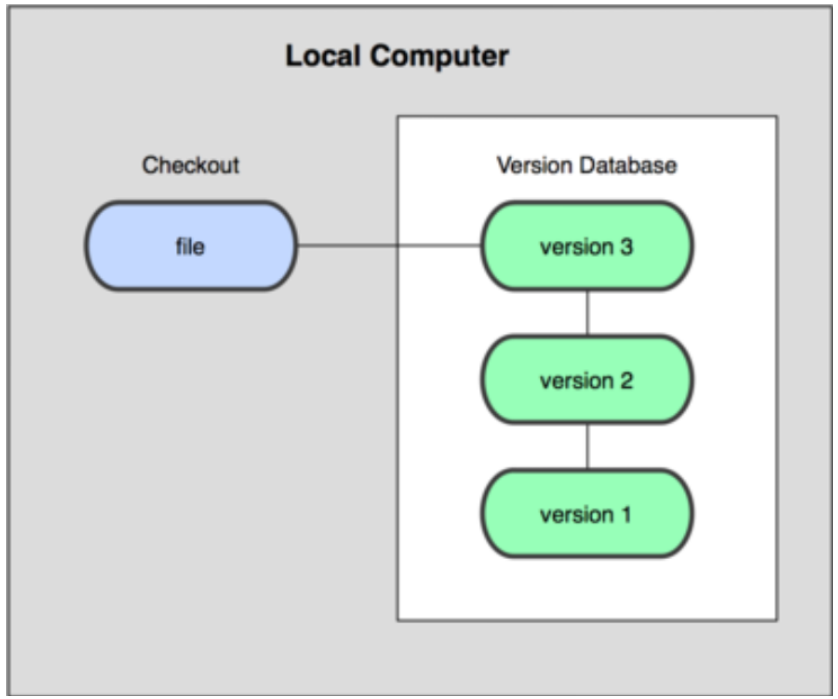
Version Control Systems (VCS)



- Ability to work online
- Source code in one place
- Difficult to cooperate
- No offline working
- Prone to errors on merging
- Downtime on server is a huge problem

Code.Hub

Version Control Systems (VCS)



- Clients fully mirror the Repository
- If Server or Clients die, source code still exists on other Clients / Servers
- Much faster than older VCS
- Ability to work offline
- Way better when cooperating

Code.Hub

Git

- an amazing revision control system for any type of document-based projects
- trackes changes in source code during software development
- is designed for coordinating work among programmers
- its goals include speed, data integrity, and support for distributed, non-linear workflows
- is free and open-source software

Code.Hub

Why Git and not other VCS?

- Git's the most popular version control system in the industry, by far.
- A proper and detailed understanding of Git will allow you to make a transition to any other distributed VCS easily.
- Most popular VCS are similar to Git

How GIT works

- Git thinks of data like snapshots of a file system
- Each commit is a snapshot of the current state of the repository and has a unique name (SHA-1)
- Every commit represents a set of changes
- Most of the work (commands) is done locally
- Git has integrity (no change is made in a git folder that git doesn't know about it, except if we explicitly tell it to ignore it)

Installing Git

- Things you'll need:
  1. You need Git installed on your system, and you can access it in a UNIX Terminal, either the Terminal on the Mac or Git Bash on Windows.
  2. Download Git from the following link: https://git-scm.com/downloads

Version Control Terminology

- Version Control System (VCS) or (SCM)
- Repository
- Commit
- SHA
- Working Directory
- Checkout
- Staging Area/Index
- Branch

Version Control Terminology

- Version Control System :
  A VCS allows you to: revert files back to a previous state, revert the entire project back to a previous state, review changes made over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

- Repository:
  A directory that contains your project work which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer

Version Control Terminology

- Commit
  Git thinks of its data like a set of snapshots of a mini file system.
  Think of it as a save point during a video game.

- SHA
  A SHA is basically an ID number for each commit.
  Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6

Version Control Terminology

- Working Directory
  The files that you see in your computer's file system. When you open your project files up on a code editor, you're working with files in the Working Directory.

- Checkout
  When content in the repository has been copied to the Working Directory. It is possible to checkout many things from a repository; a file, a commit, a branch, etc.

Version Control Terminology

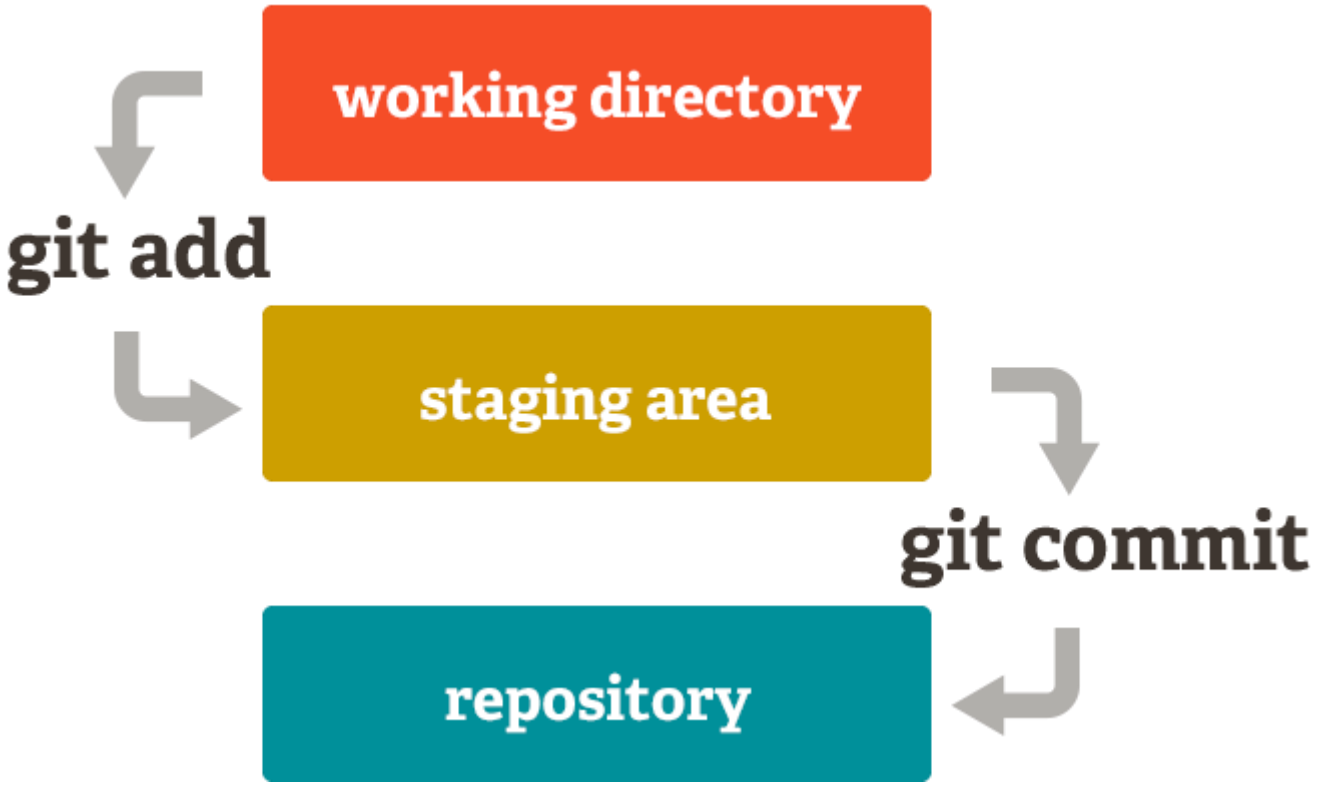
- Staging Area
  You can think of the staging area as a prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repository.

- Branch
  A branch is when a new line of development is created that diverges from the main line of development. This alternative line of development can continue without altering the main line.

Basic Git Commands

- git init
- git status
- git add <filename> / git add .
- git commit / git commit -m “commit message”
- git log -oneline / git log —stat
- git clone
- Other commands like git show, git ignore, git diff etc.

Basic Git model locally



Basic Git Commands

- git init – Initialize a Git repository/working directory
- git status – Status of your working directory
- git add <filename> or git add . (for all files in your working directory)
- git commit – Stash changes in your working directory
- git log –oneline – View your commit history
- git clone – Create an identical copy

Code.Hub

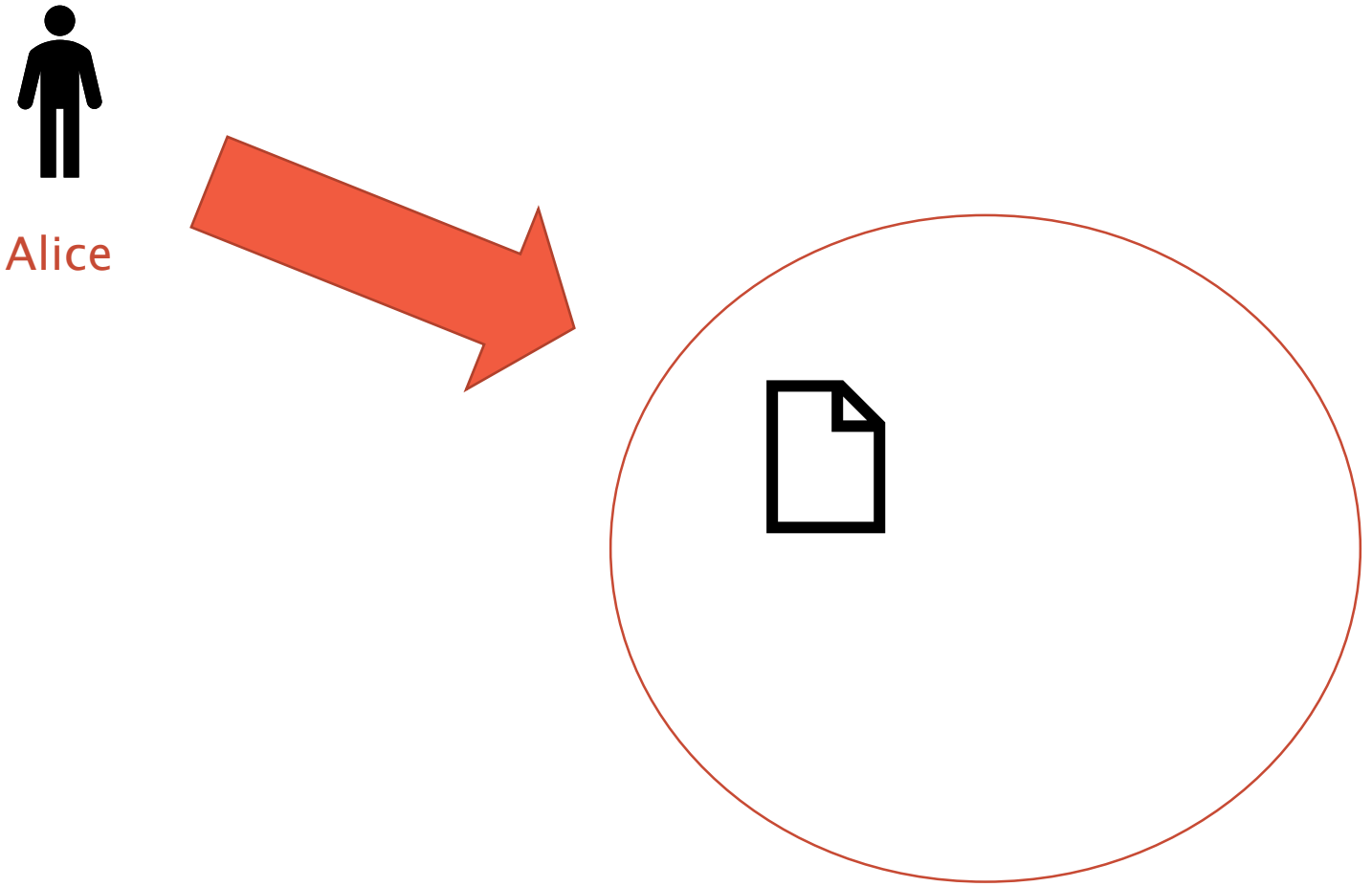
Working with a remote repository

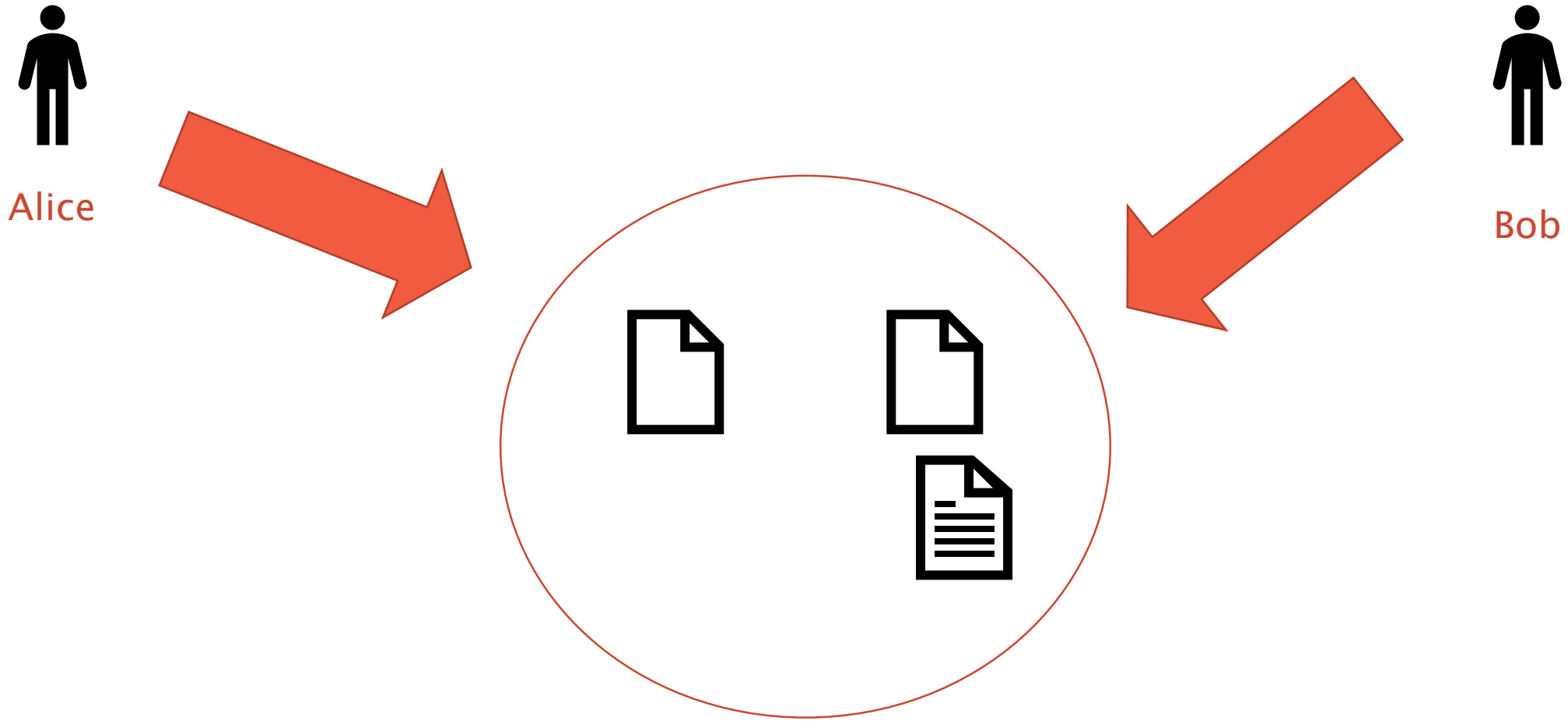
- Remote?
  - It's the place where your code is stored.
  - By default, remote name is origin and default branch is master.

- Certain things that come to play, namely collaboration.
  - How are we going to handle that with Git.

So here comes, push, pull, branching, merging, forking.

Code.Hub

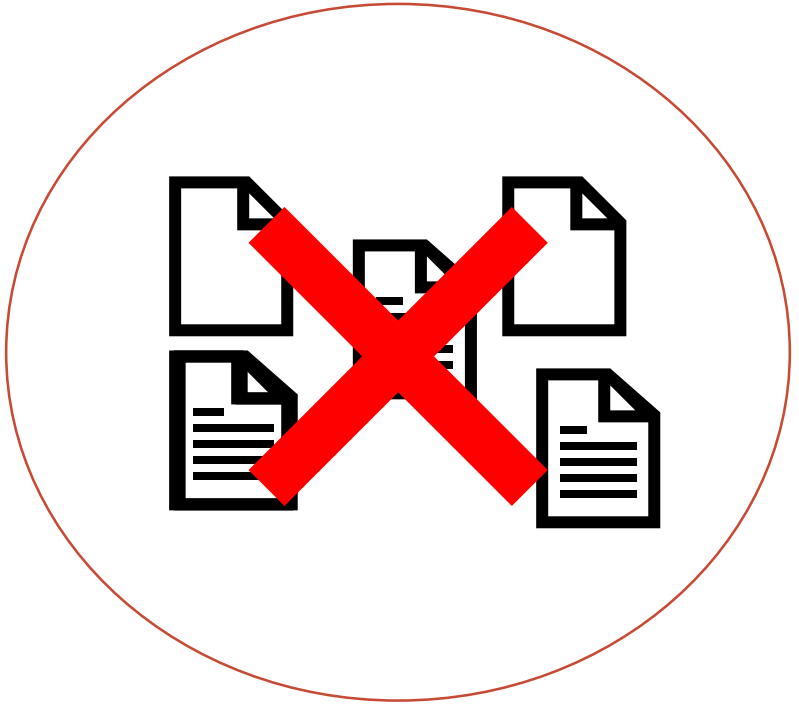

Alice


Alice
Bob


Alice
Joe
Bob

Alice

Joe



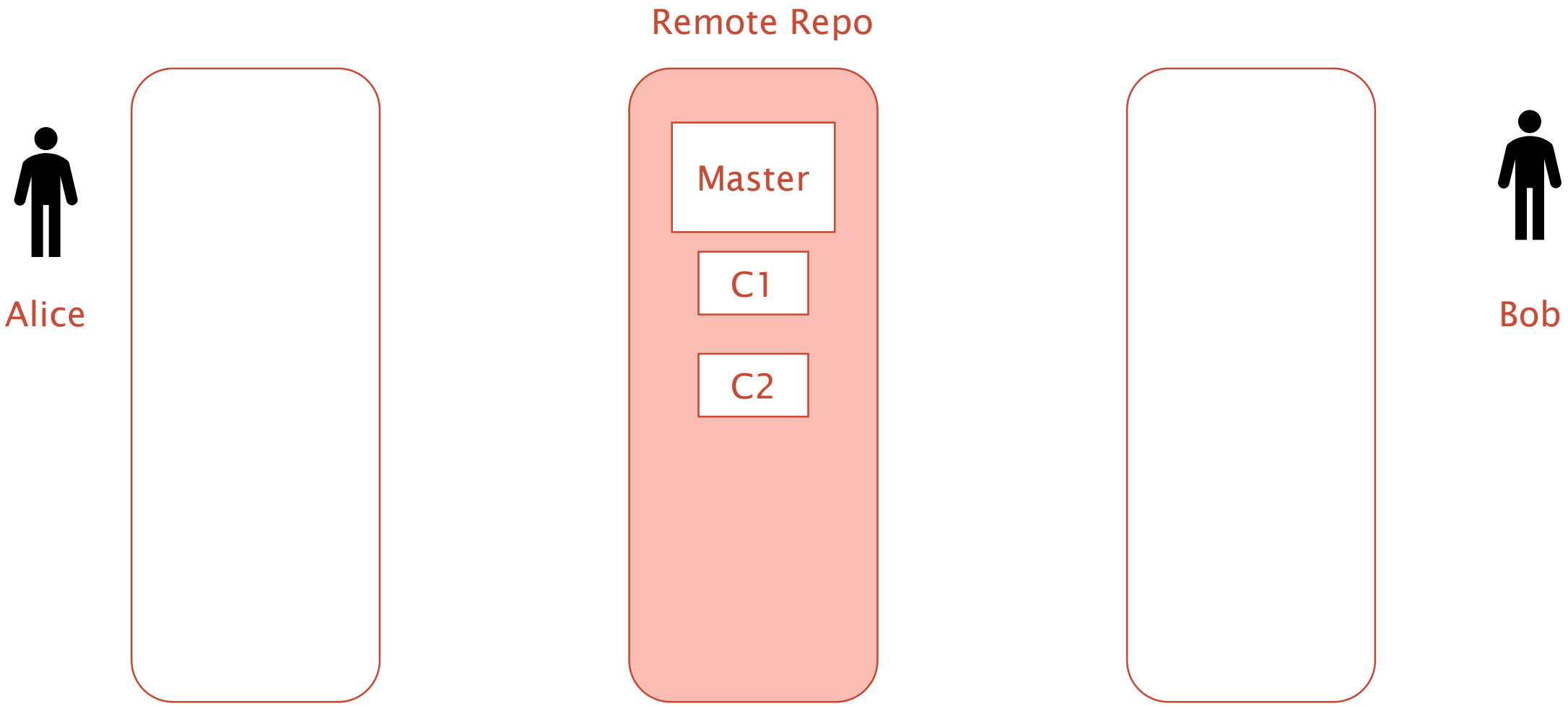
Who replaced the files? When ?

Bob

More Git commands

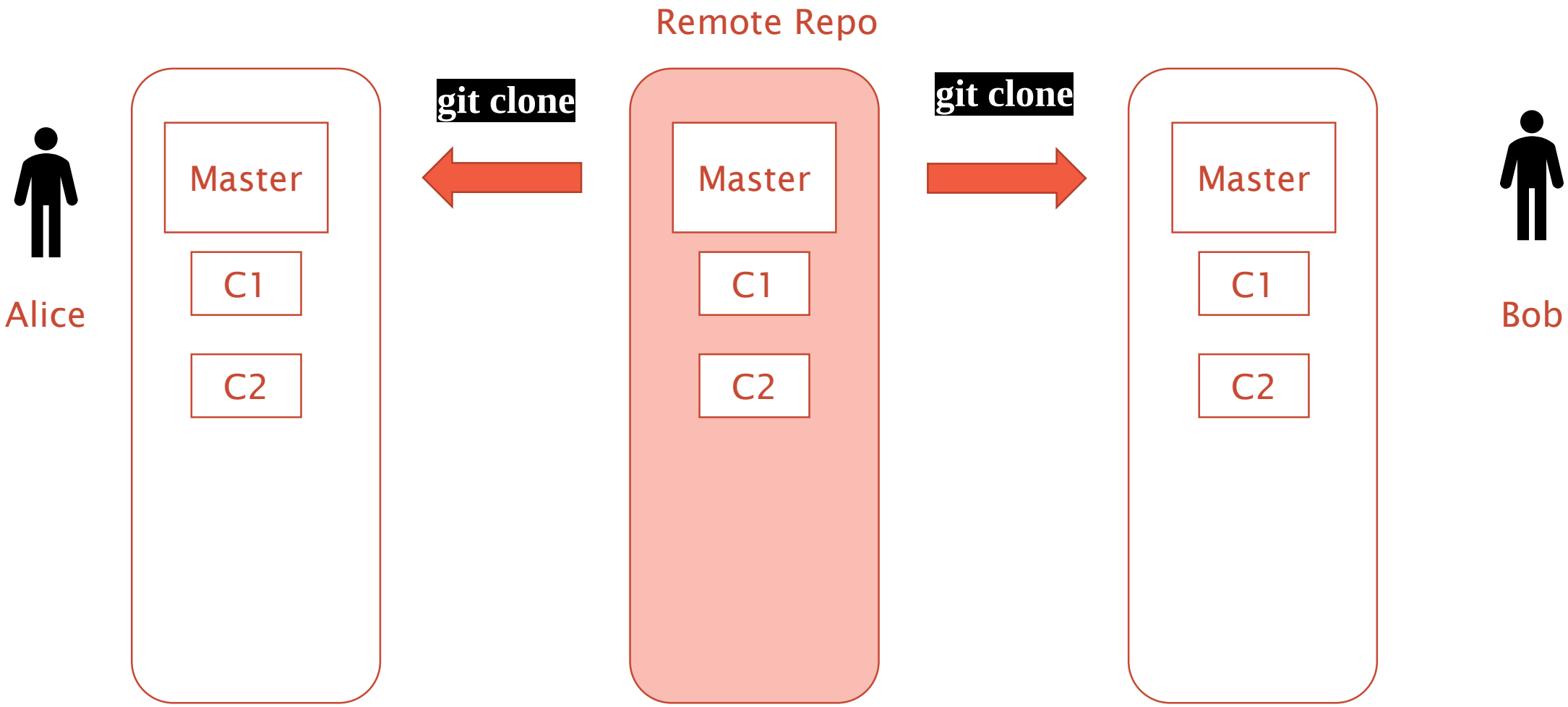
- git push – push your changes into the remote repository
- git pull – pull your latest changes from the remote repository
- git branch – view branches in your repository
- git branch <branchname> - create a branch
- git checkout <branchname> - move to that branch
- git merge <branchname> - merge into that branch
- git revert <commit sha>

Collaborate

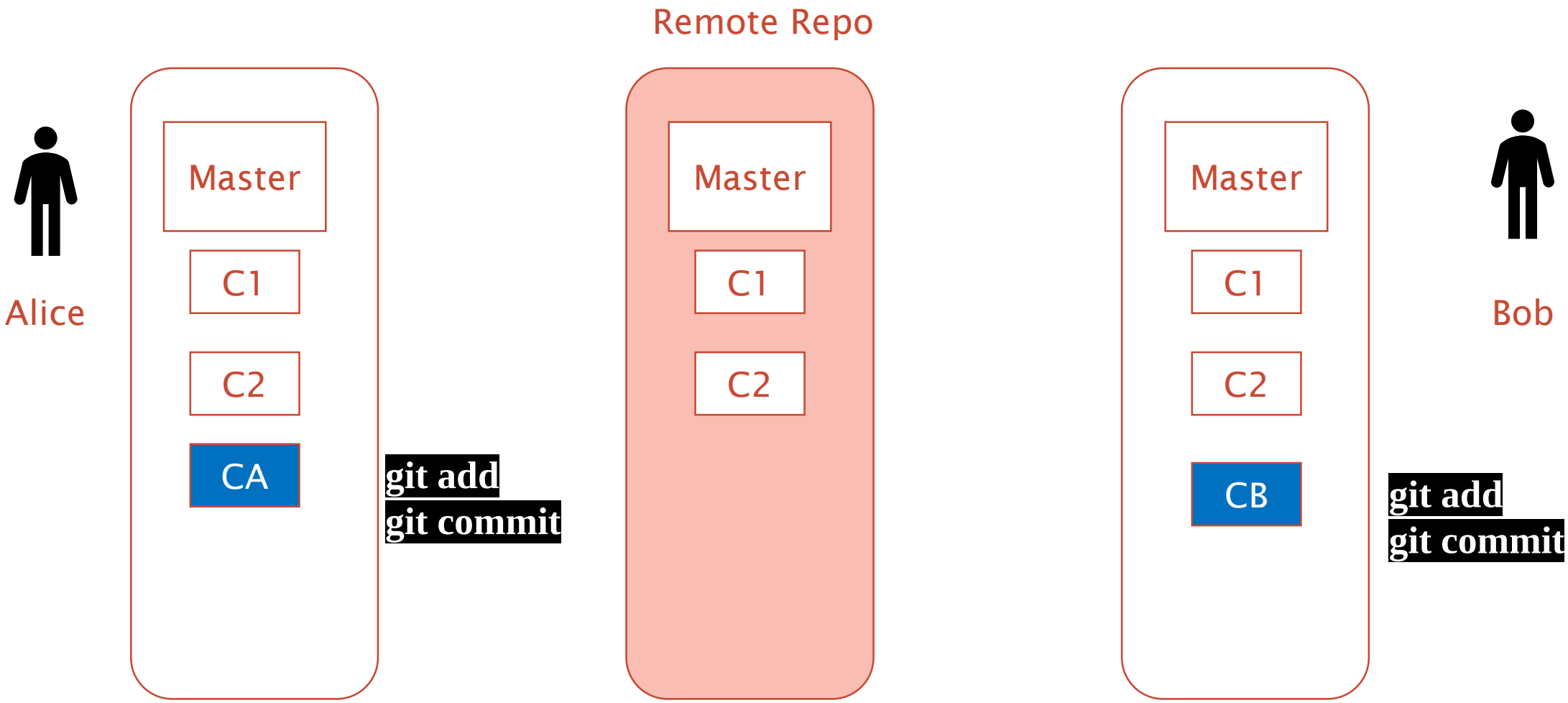


Code.Hub

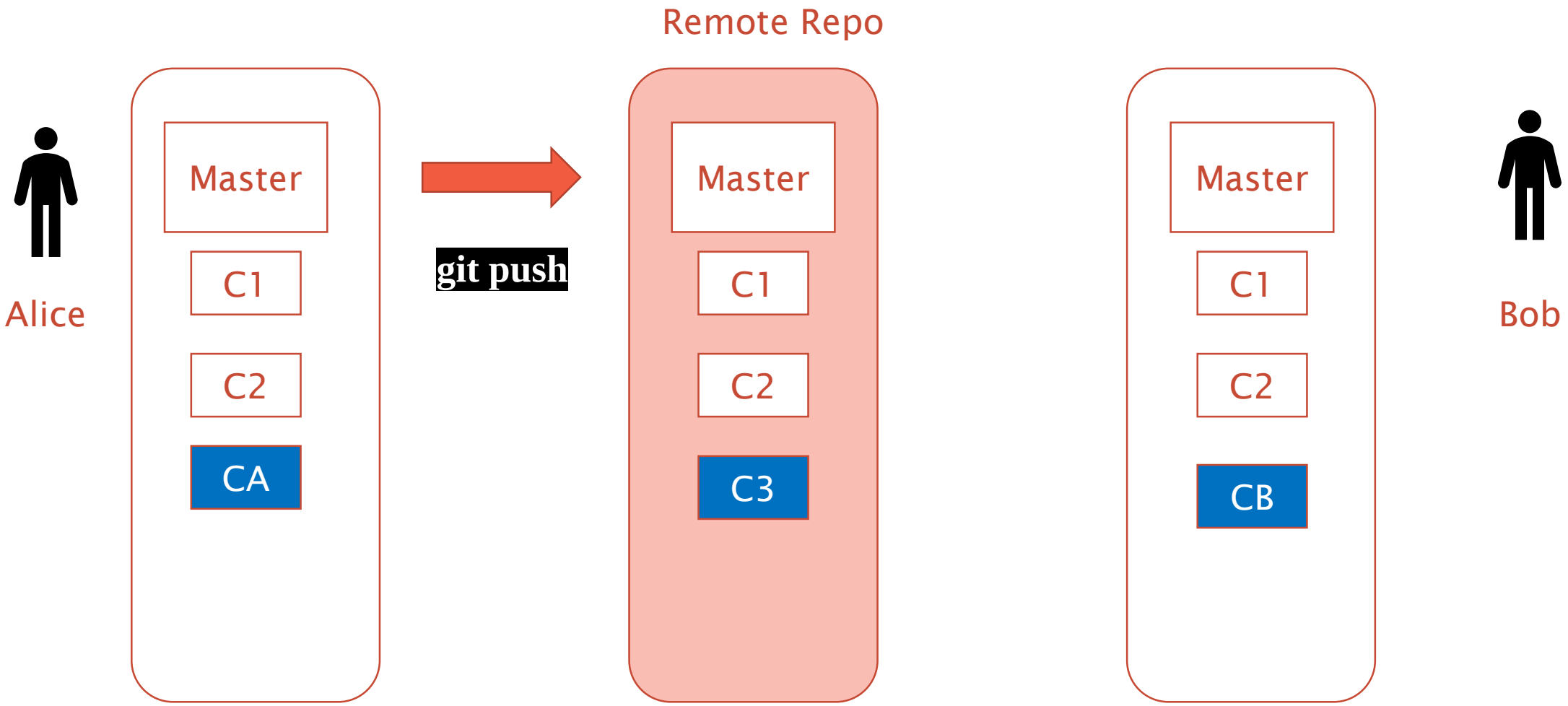
Collaborate



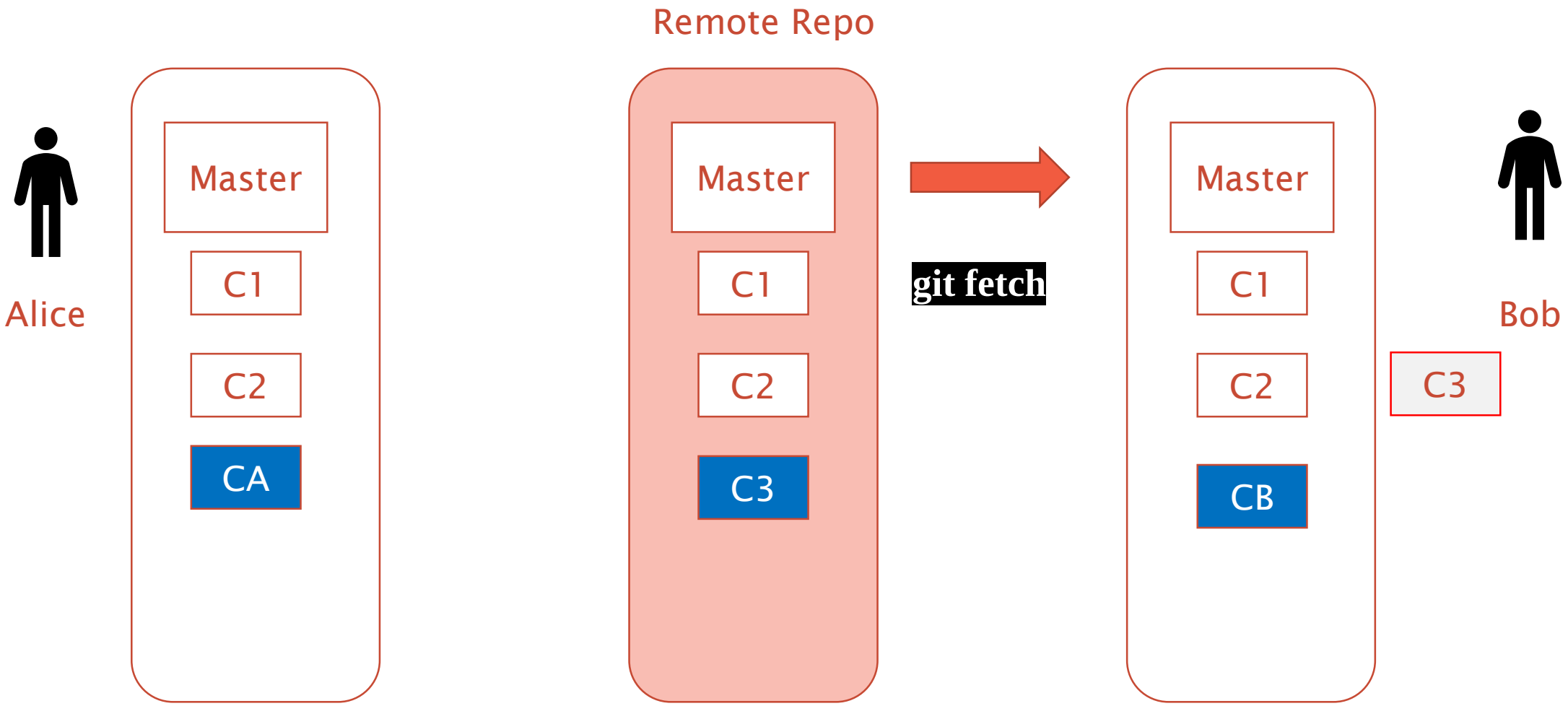
Collaborate



Collaborate

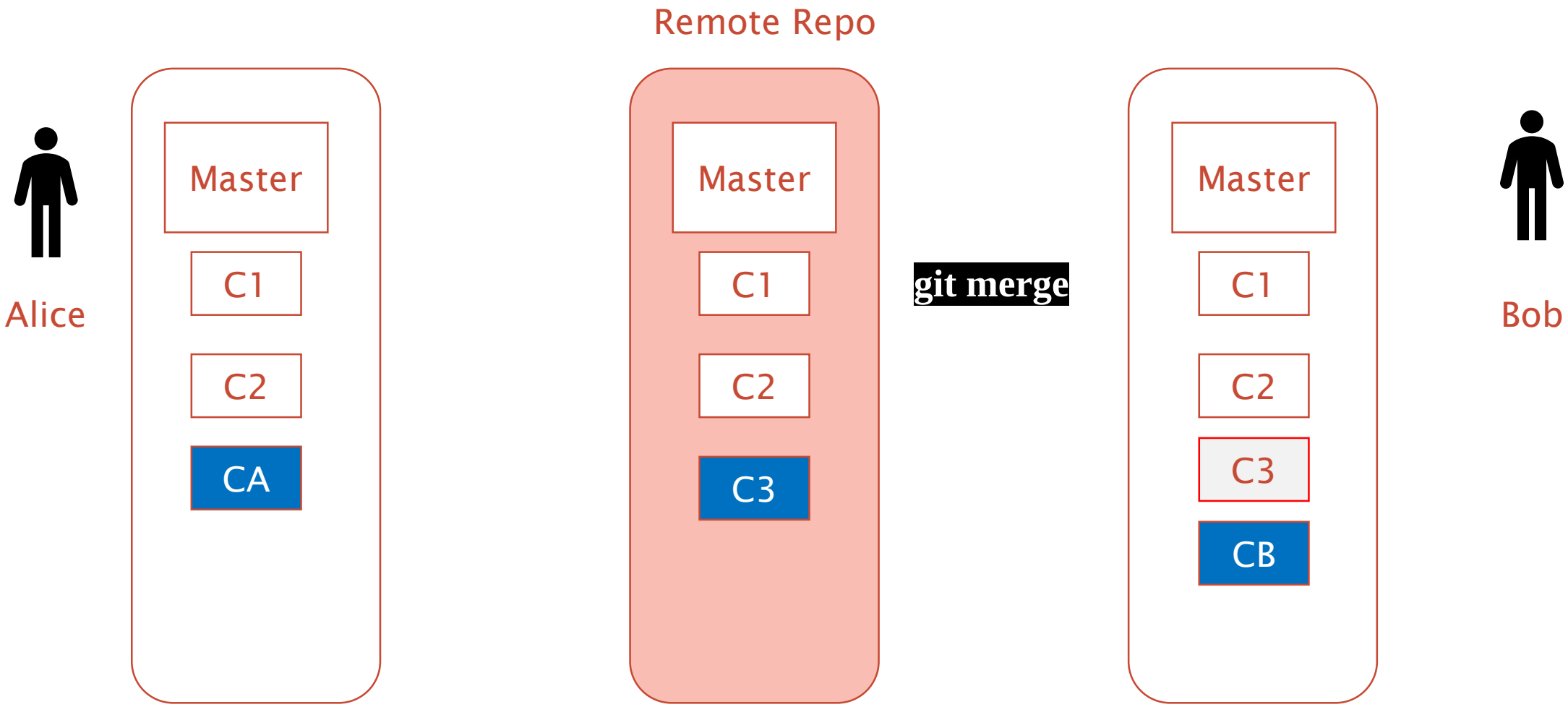


Collaborate

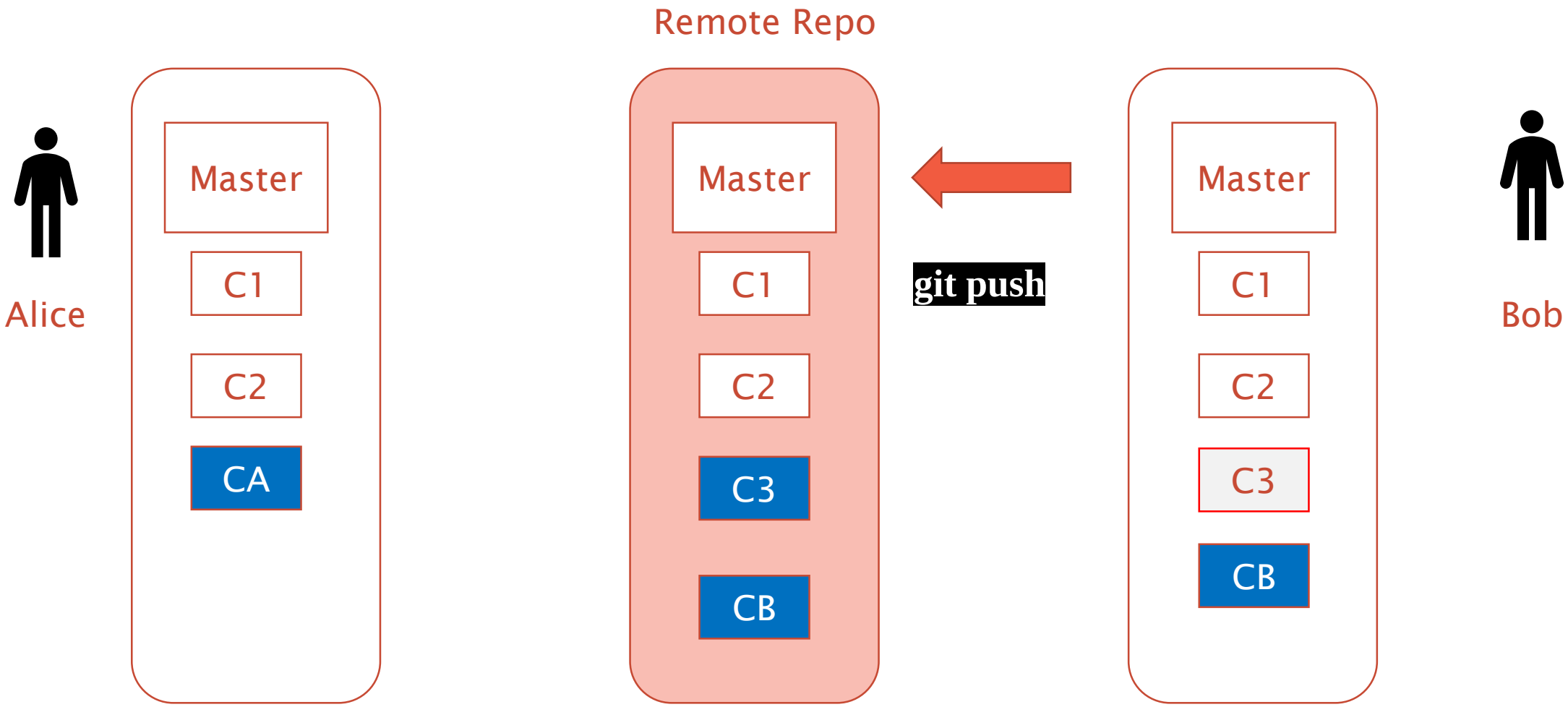


Code.Hub

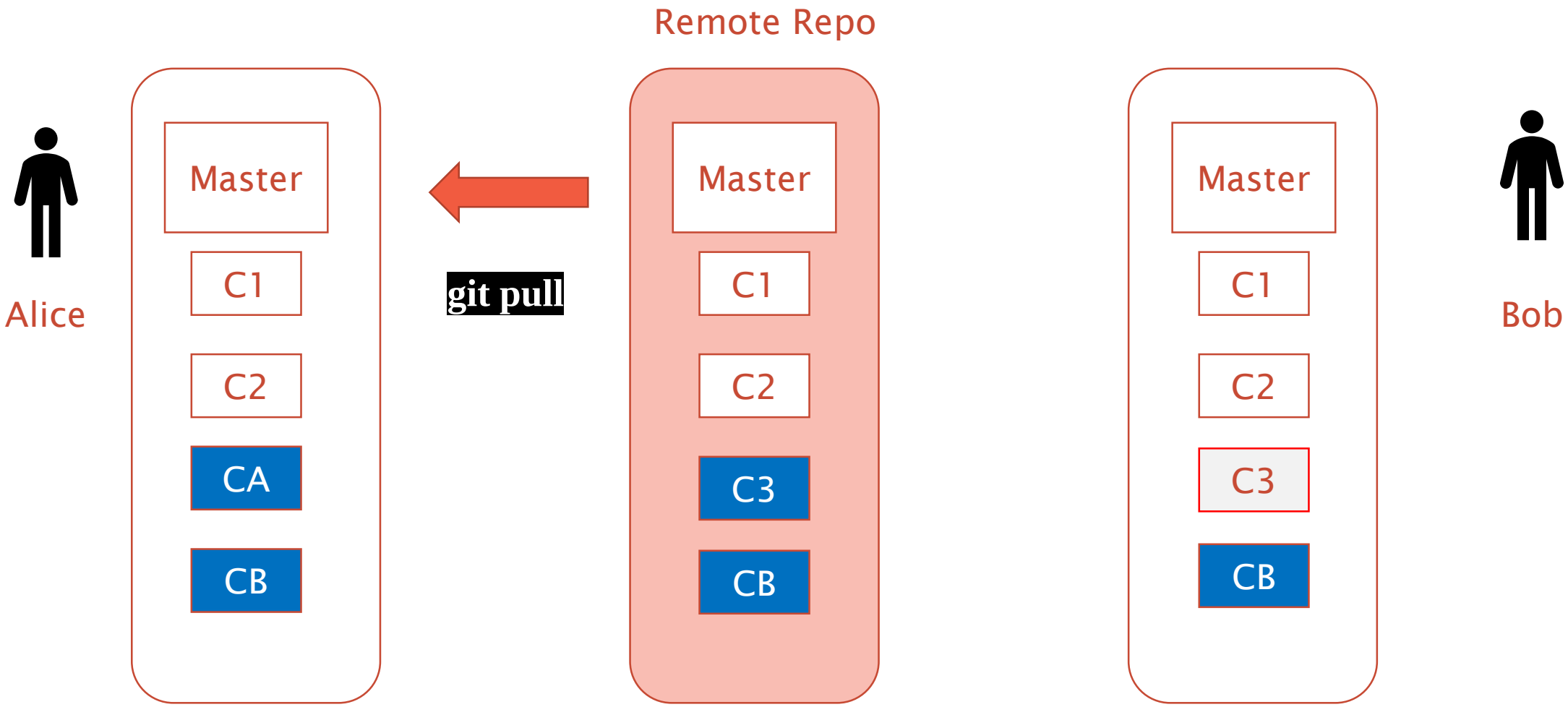
Collaborate



Collaborate



Collaborate



Dealing with Merge Conflicts

- Two typical cases of merge conflicts
  - Normal merge where you're a collaborator
  - Pull request (handled by the repository owner)

Merge conflicts

- The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

branch 1's version

branch 2's version

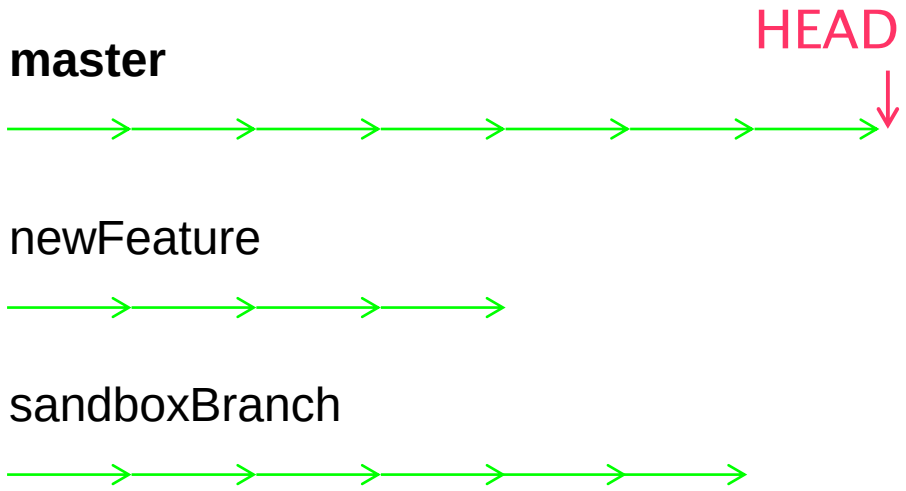
- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

Branches in Git

- All this time we were working on our main branch. (master)
- We can actually have many branches and each one can hold different features and changes.

Branches in Git

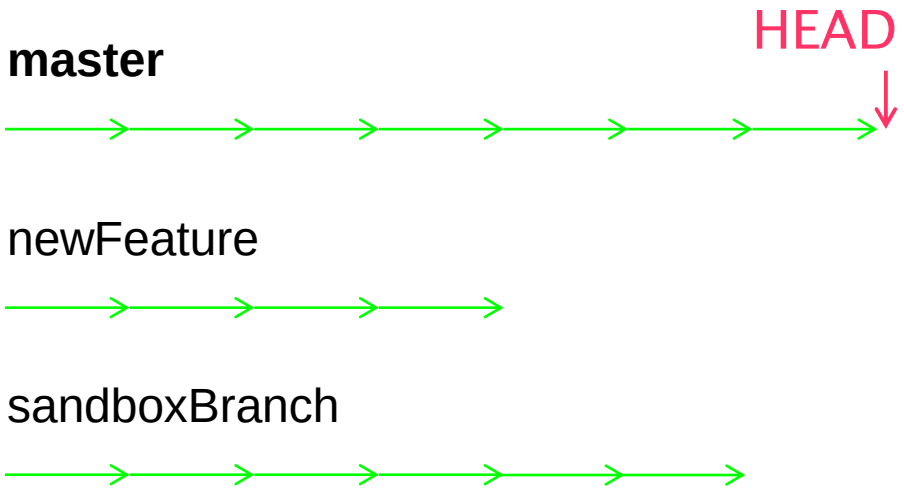
- **git branch** command will show all local branches



Branches in Git

- To move between branches just use the **git checkout <branch_name>** command

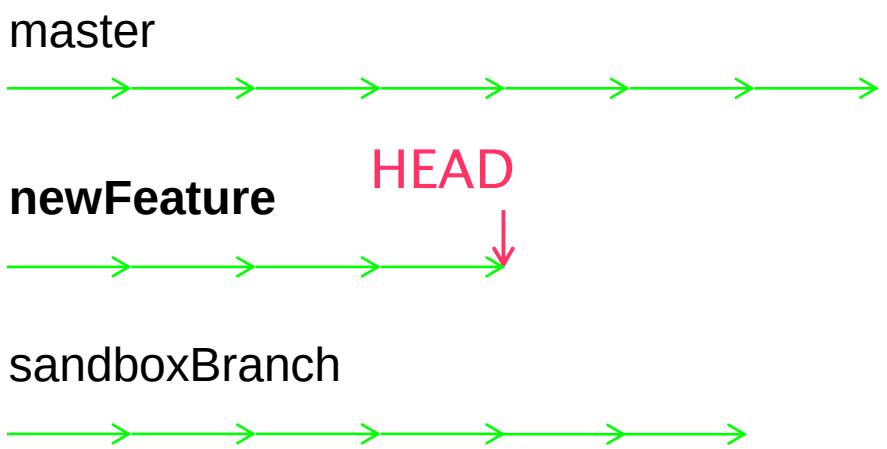
- So: **git checkout newFeature**



Code.Hub

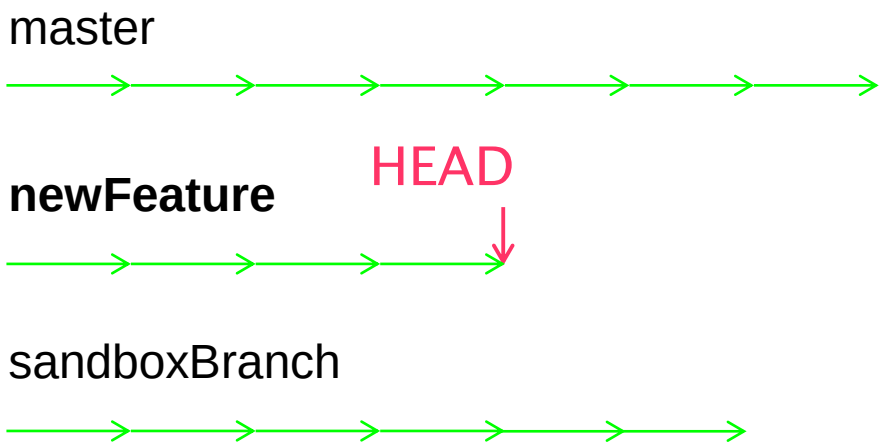
Branches in Git

- To move between branches just use the **git checkout <branch_name>** command
- So: **git checkout newFeature**



Branches in Git

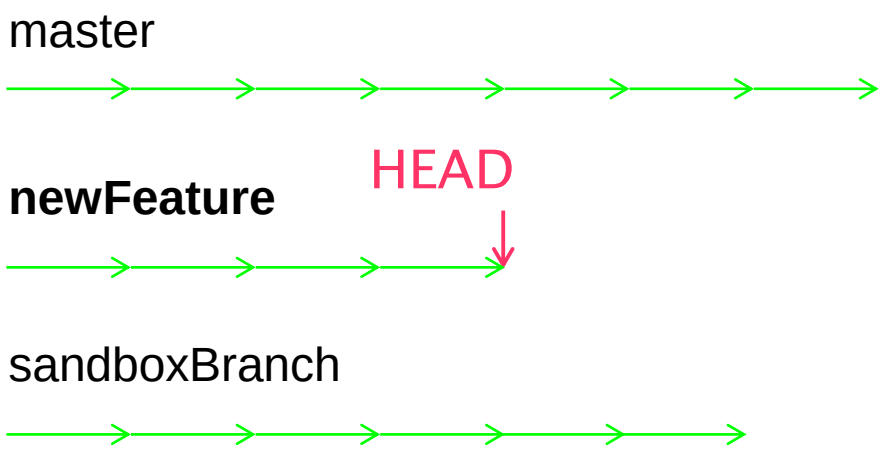
- Think of HEAD as the recording module of a tape recorder. It is the edge of the branch. It is where the new commits you will create will be added.
- So, when checking out a different branch HEAD actually points to a different commit.



Code.Hub

Branches in Git

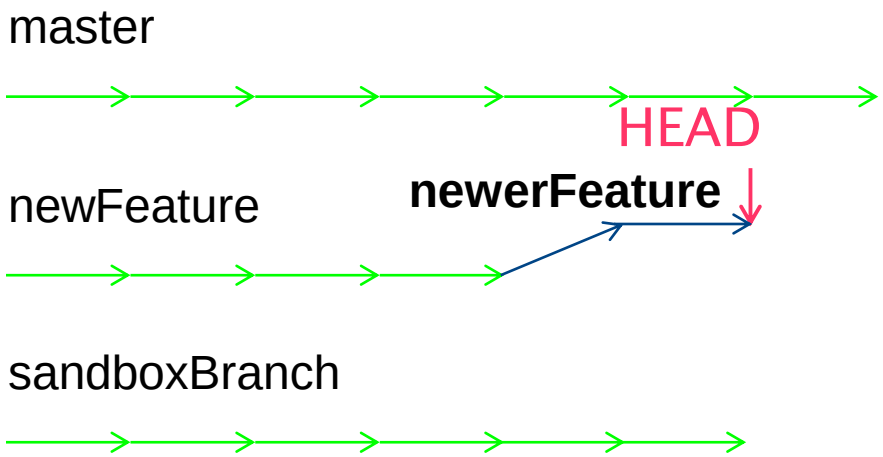
- But if we want to create a new one we use:
  git checkout -b “newerFeature”



Branches in Git

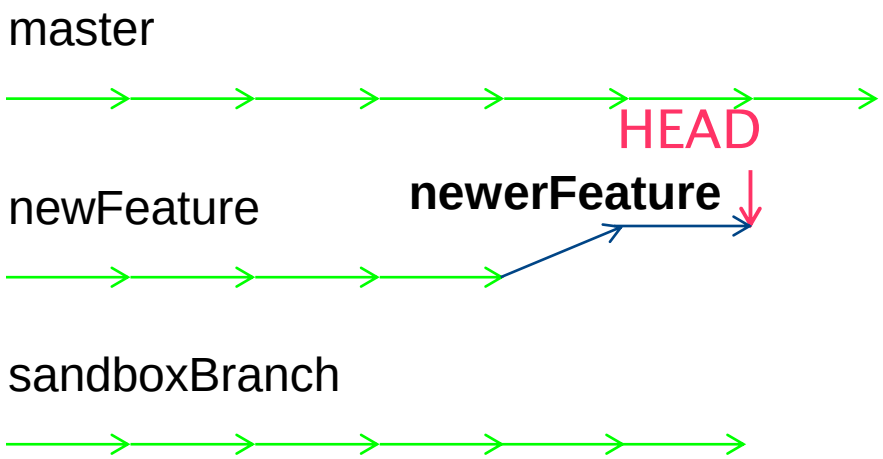
- The commit history of our new branch called “newerFeature” will be the same as newFeature since we created the new branch while HEAD was pointing that branch.

- And we add 2 new commits to that branch.



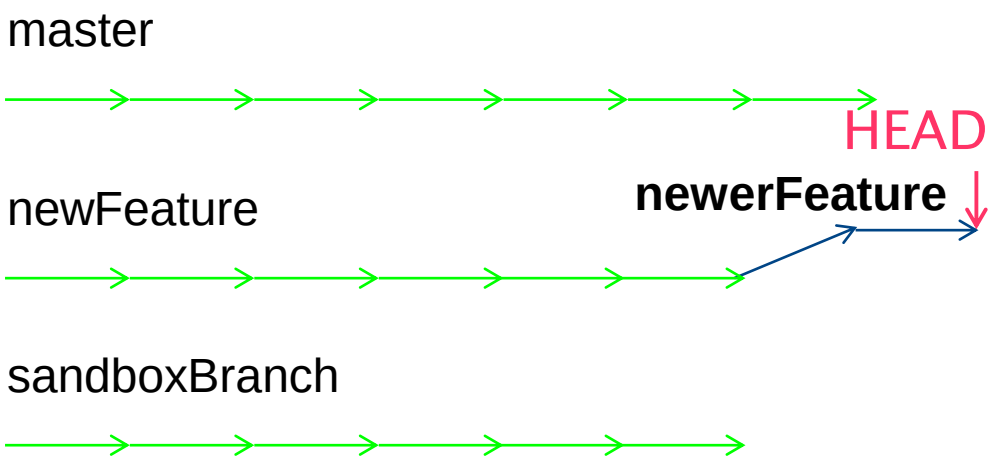
Branches in Git

- Let's suppose that newFeature branch has 2 new commits
- How do I get my newerFeature branch updated?



Branches in Git

- **git rebase newFeature**
- "Rebasing will actually fetch all new commits and place them right behind our commits on the current branch"

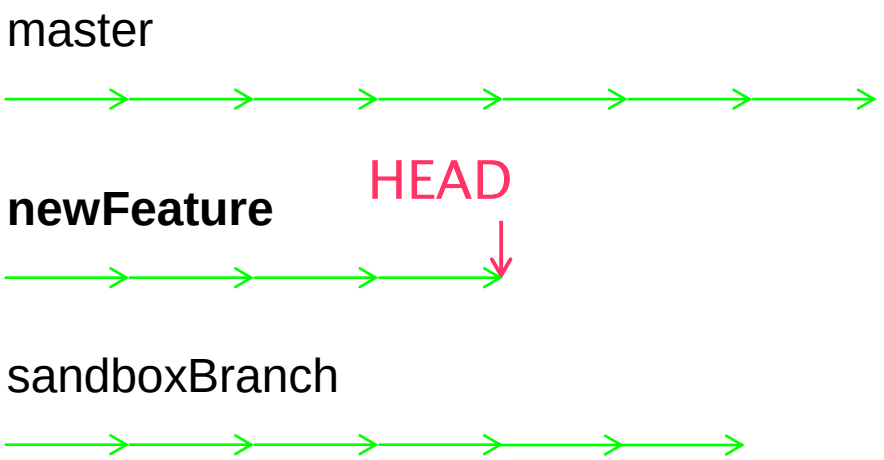


Code.Hub

Branches in Git

- How do I push all my commits/branches in server?

- There is a command just for that and it is:
- **git push** (for pushing only commits)
  OR
- **git push origin newFeature -u** (for pushing branches)



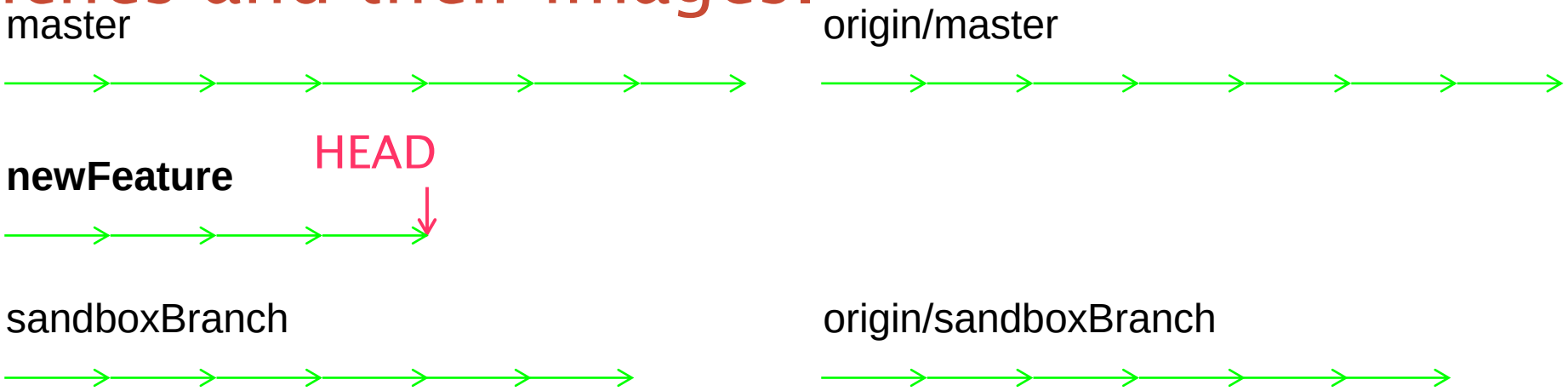
Code.Hub

Branches in Git

- And what is **Origin**?
- Think of Origin as a **local image of the Repository** that exists somewhere in a git repository host (like Github, Bitbucket etc.)

Branches in Git

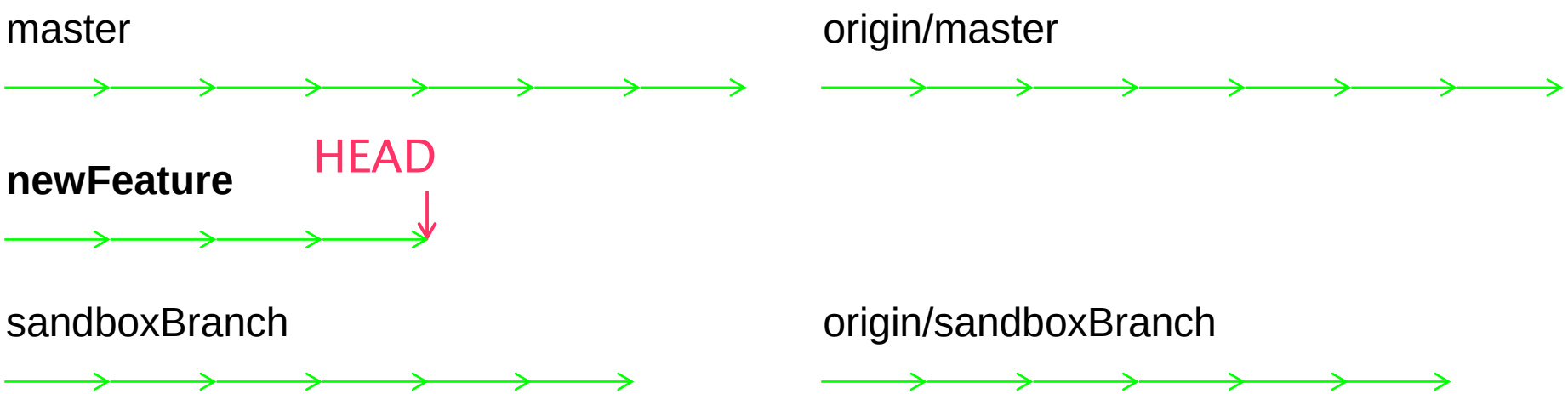
- So besides your local braches you actually posses an image of all branches your coworkers have pushed in Origin.
- But let's focus on our own for the time being. Our own branches and their images.



Code.Hub

Branches in Git

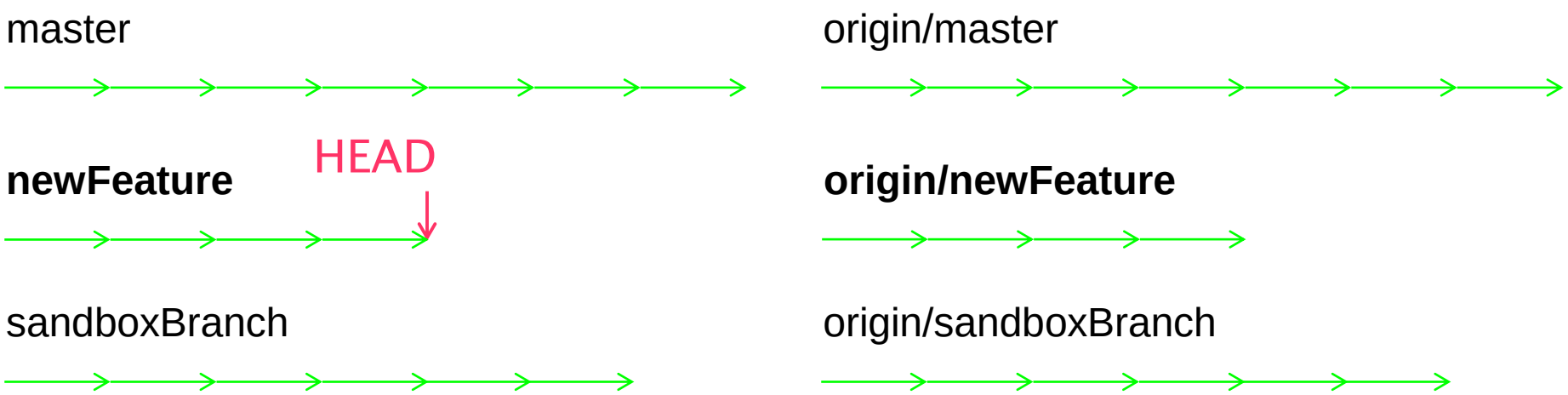
- Let's push the newFeature branch in Origin.
- **git push origin newFeature -u**



Code.Hub

Branches in Git

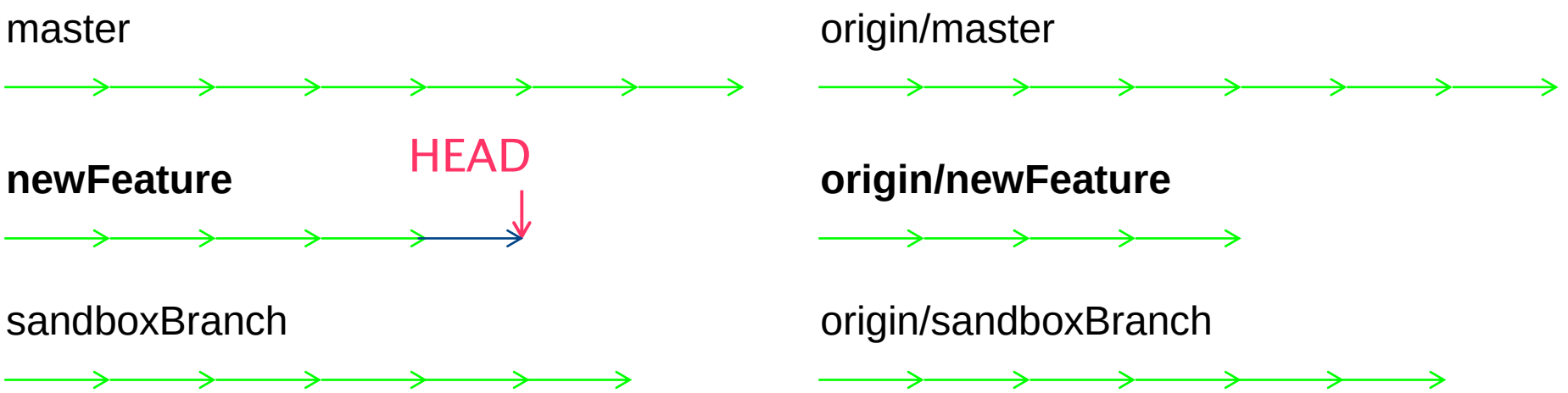
- Let's push the newFeature branch in Origin.
- **git push origin newFeature -u**



Code.Hub

Branches in Git

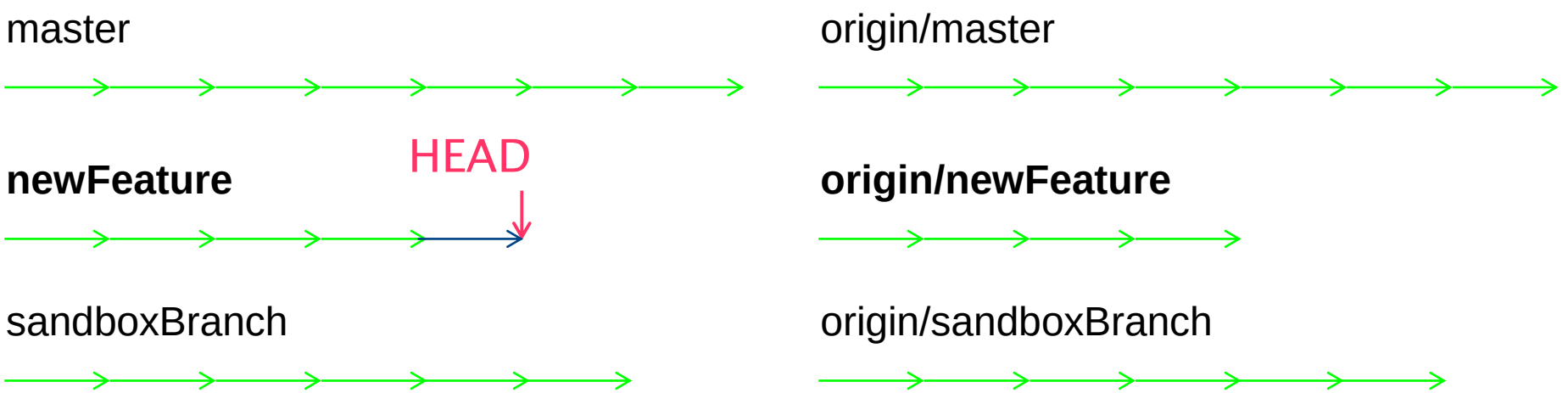
- Now we create a new commit in our newFeature branch
- origin/newFeature still hasn't got the new commit



Code.Hub

Branches in Git

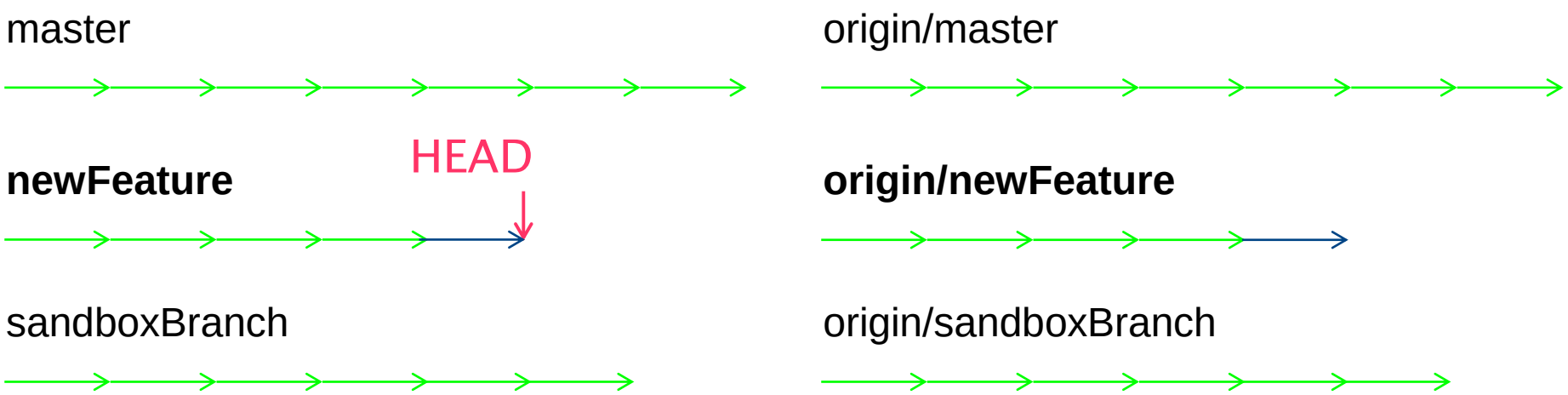
- We hit **git push**



Code.Hub

Branches in Git

- Now origin/newFeature is updated along with the server.



Branches in Git

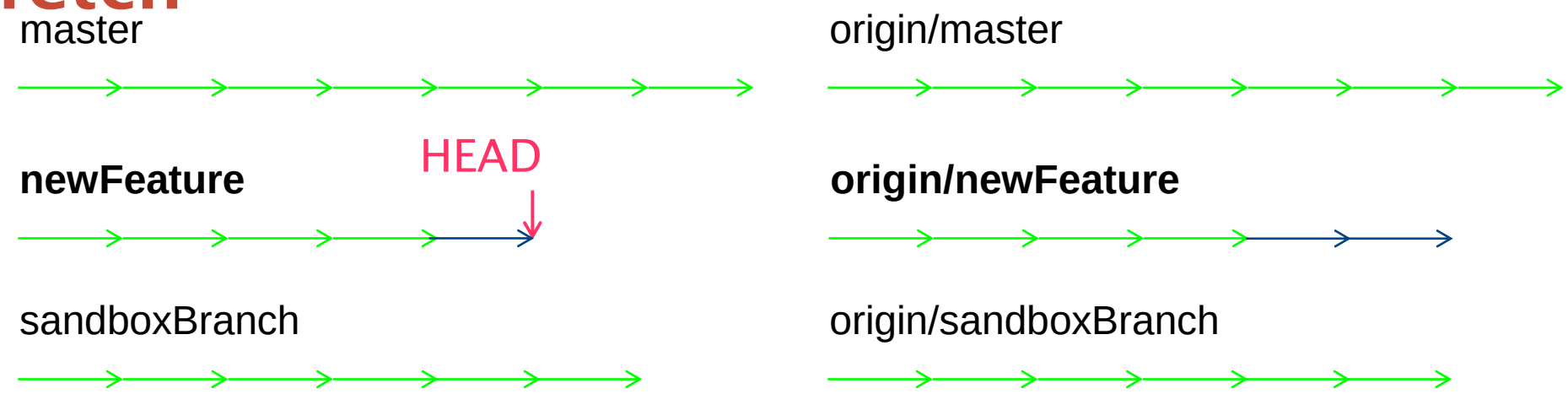
- Finally, let's say that our co worker want's to join the fun and get our branch to work on his own and commit his/her changes.

- The only commands that he/she would have to use are:
- **git fetch** (updates origin with changes that happened to branches)
- **git checkout newFeature** (go to that branch – create it if you go for the first time)

Branches in Git

- So our co-worker checkout our branch and committed 1 new commit and pushed it.

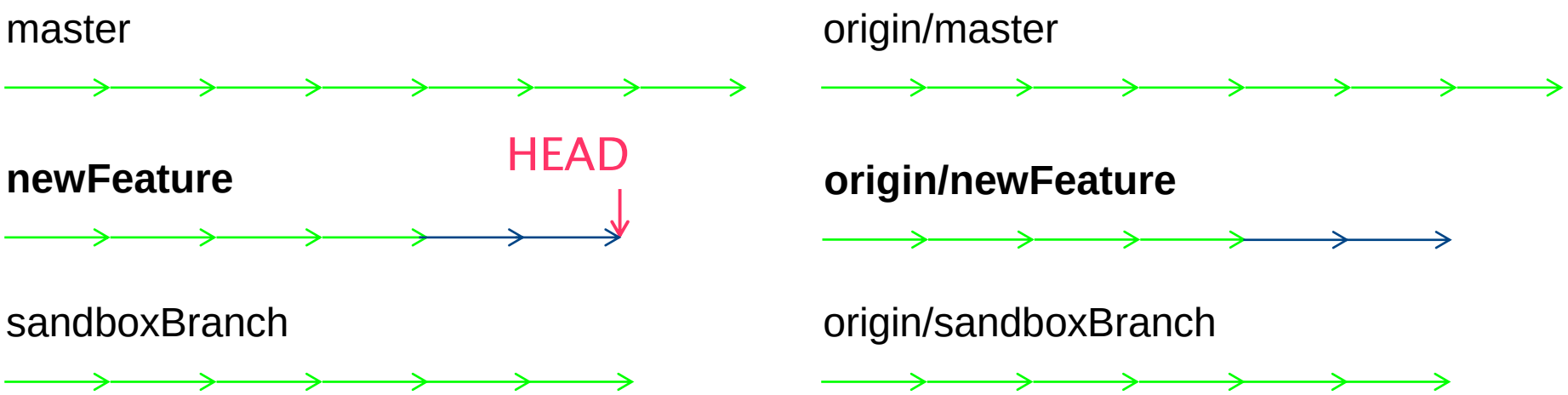
- Now we have to fetch and rebase our branch with the new changes.
- **git fetch**



Code.Hub

Branches in Git

- So our co-worker checkout our branch and committed 1 new commit and pushed it.
- Now we have to fetch and rebase our branch with the new changes.
- **git fetch**
- **git rebase origin/newFeature** OR **git pull -- rebase**



Code.Hub

Branches in Git

- And If I want to know all the branches that exist out there in our application just use the same command as you use to see local branches but with -r param

- **git branch -r**

Git Commands

- git branch
- git branch -r
- git checkout -b <new_branch>
- git checkout <different_branch>
- git fetch
- git rebase <branch_name>
- git pull --rebase
- git push
- git push origin <branch_name> -u

Code.Hub

Github

- a web-based version of git
- provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project
- is a subsidiary of Microsoft, which acquired the company in 2018 for $7.5 billion
- offers unlimited private repositories to all plans, including free accounts
- the largest host of source code in the world