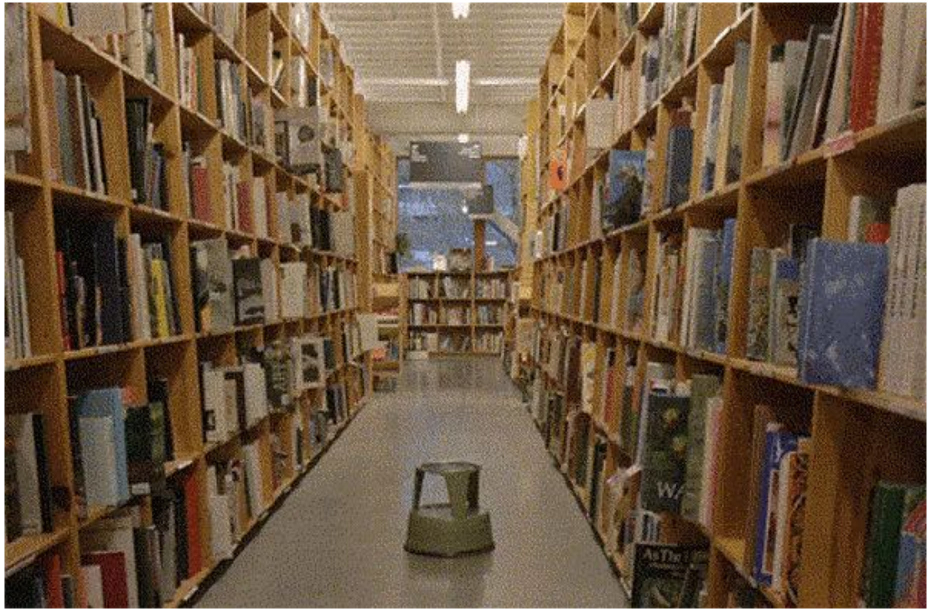
DevOps

Versioning with Git

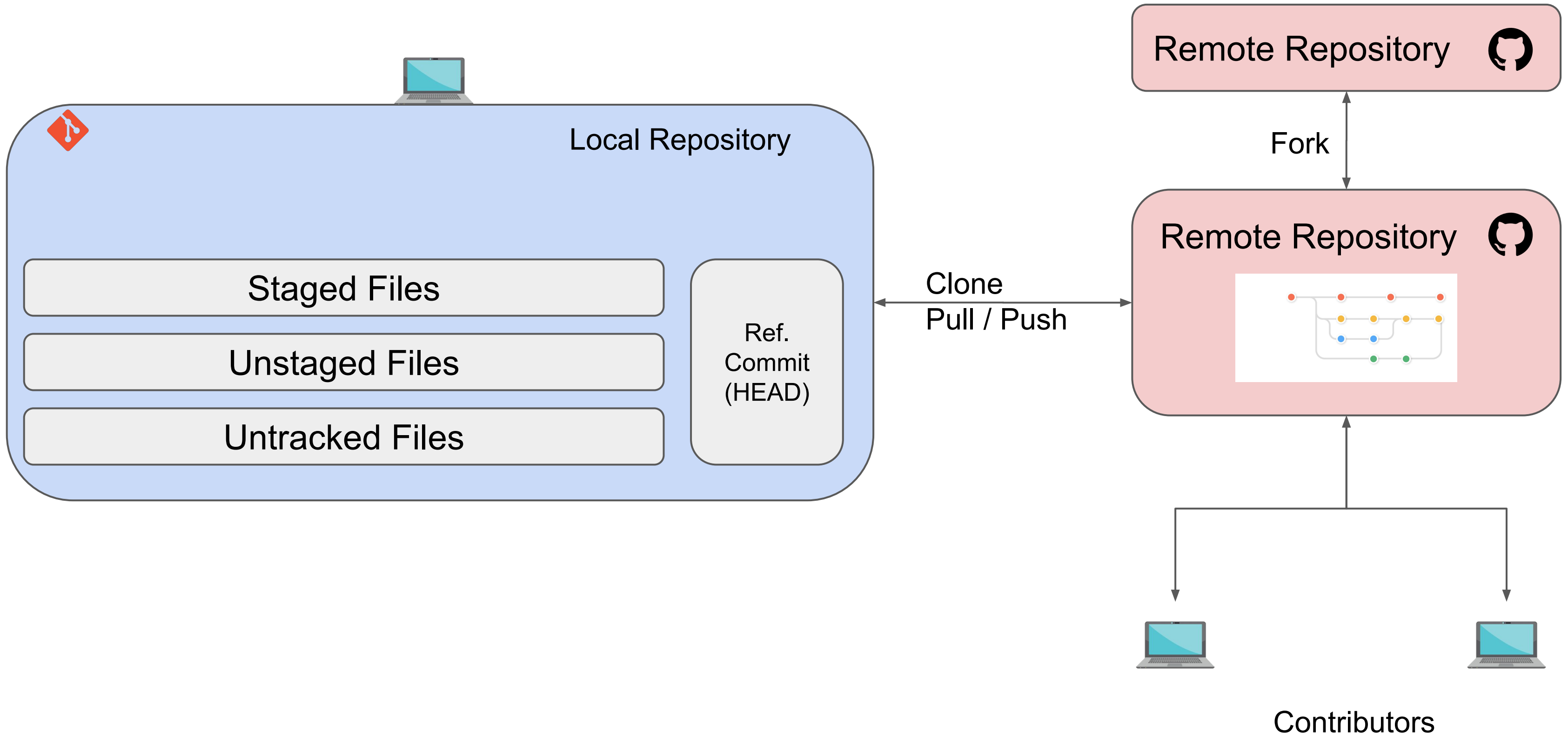
Why use Version Control Systems ?



Git - Introduction

- Versioning software created in 2005 by Linus Torvalds
- Distributed by design
- Many available hosting services (Github / Bitbucket / Gitlab)

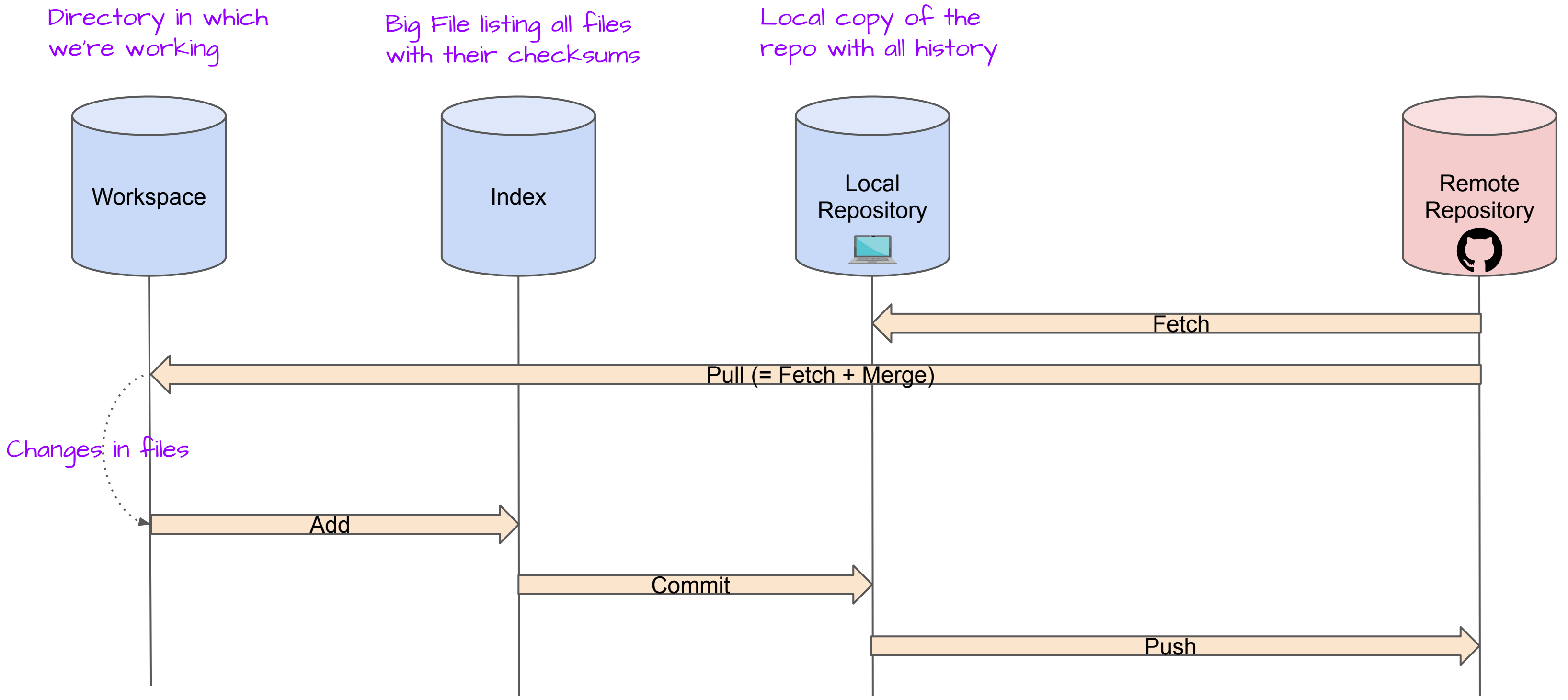
Repository


Local Repository
Staged Files
Unstaged Files
Untracked Files
Ref.
Commit
(HEAD)
Clone
Pull / Push
Remote Repository
Fork
Remote Repository
Contributors

Repository

- Creating a git repository
- Cloning an existing repository
- Public vs Private repositories
- Multiple Remotes
- Mono-repo vs Multi-repos

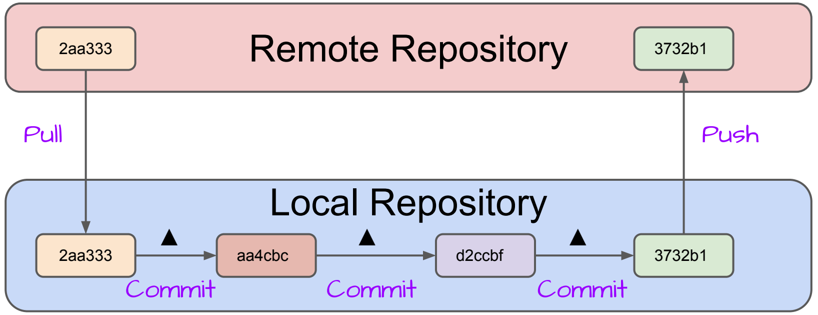
Working with Git


Directory in which we're working
Big File listing all files with their checksums
Local copy of the repo with all history
Workspace
Index
Local Repository
Remote Repository
Fetch
Pull (= Fetch + Merge)
Changes in files
Add
Commit
Push

Commiting

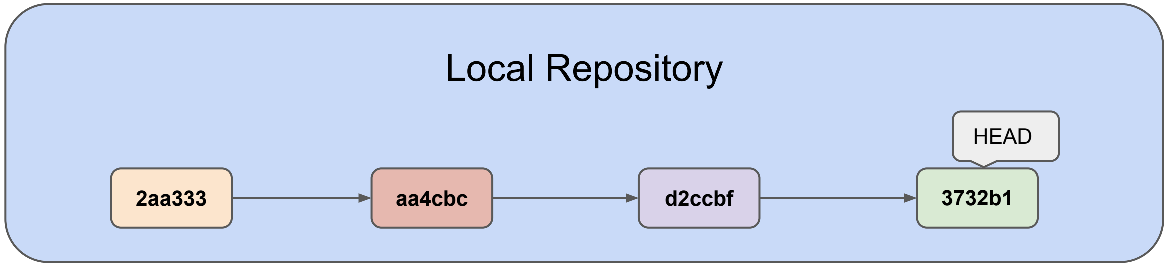
- Commiting in Git allows to save a state (snapshot) with some metadata (message, time, user).

- A unique identifier is created for each commit so we can refer to it.



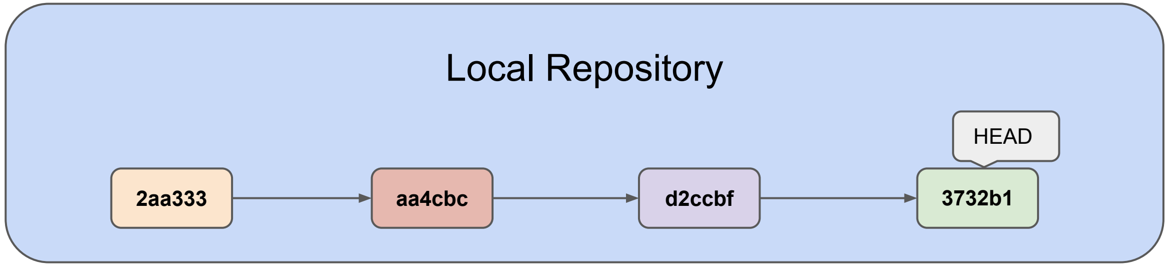
Commiting

- The latest commit (in a branch) is called the "HEAD".
- Git provides the ability to move the pointer between commits.
- This allows us to reload an older commit and apply changes to it. We can also remove (cancel) commits.



Branches

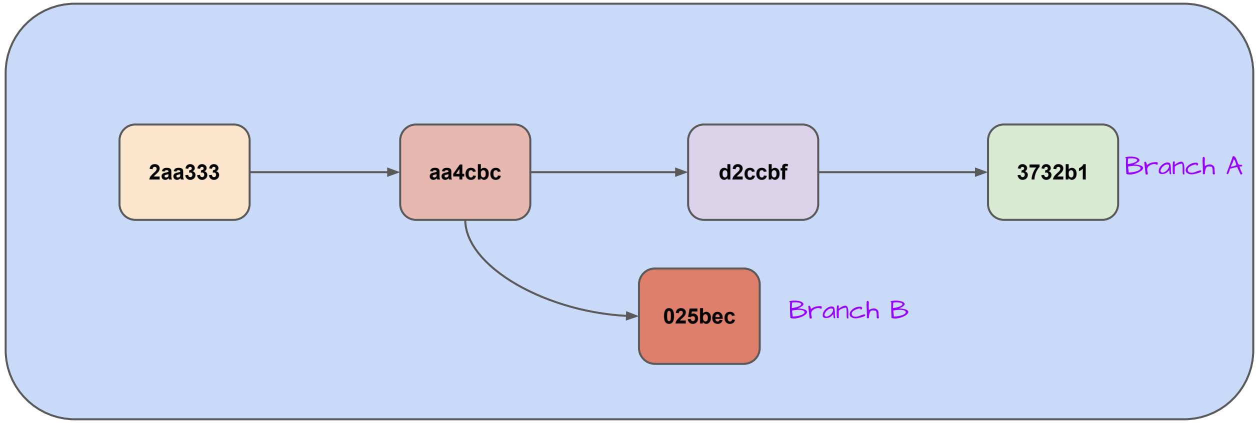
- Imagine we have deployed the commit “aa4cbc” in production and we’ve started commiting new features.



- A bug or an urgent feature is requested for Production and we don’t have time to validate the two last commits.

Branches

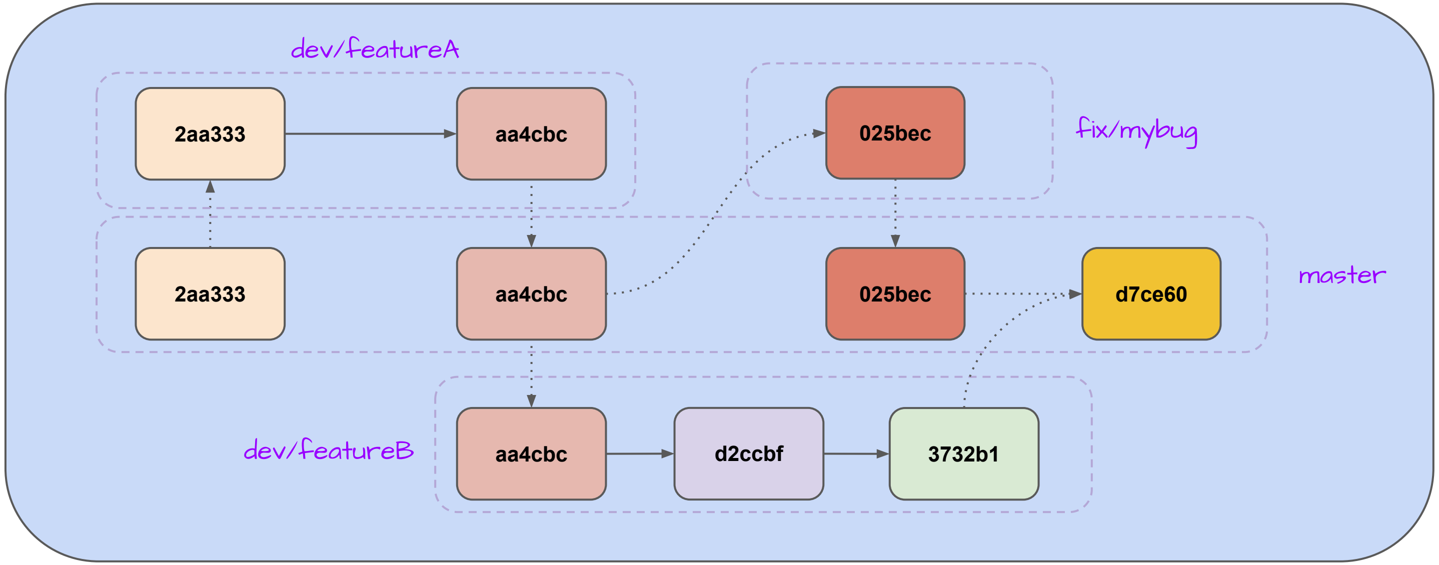
- Parallel version of your repository
- "Checkout" (or reset) to the older commit, create a new branch and solve the issue.



- The default branch is called the **master** branch. By convention, we only write production code in the master branch and we create separate branches for Features & Bug fixes

Branches

- In practice, we would have used branches from the project's start.



- Note that a "merge" operation may be necessary to create the "d7ce60" commit. Writing commits to the master branch should be controlled by "Pull Requests".

Branches

Some reasons why you should use branches:

- Isolate your work on a specific feature / bug fix.
- Try new things without affecting a “stable” code.
- Collaborate with others.

Synchronizing Branches

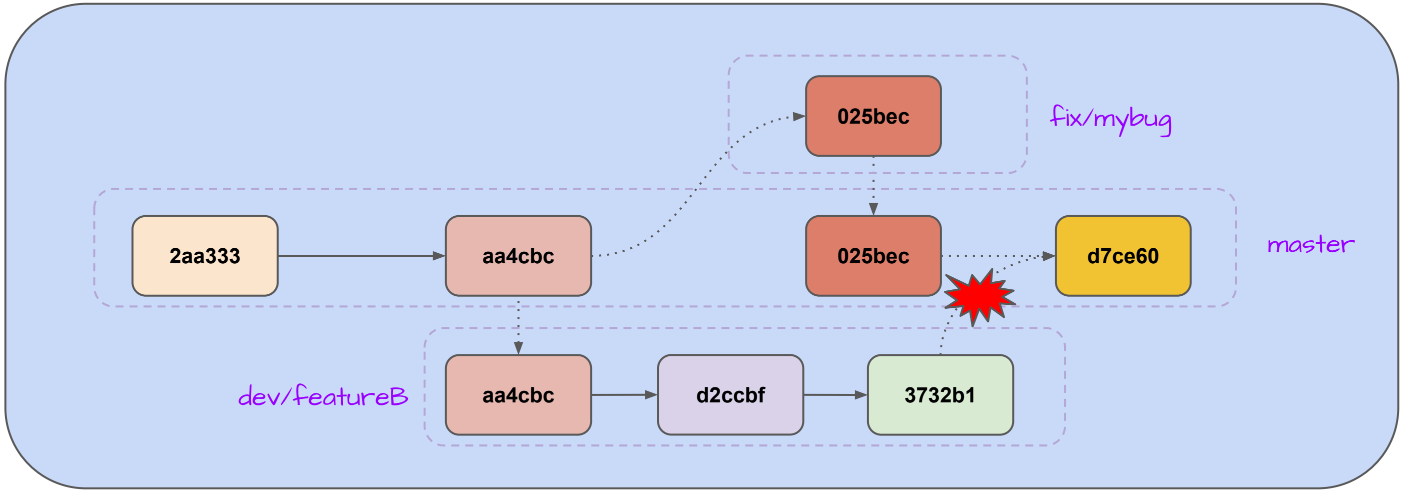
- Applying some branch’s changes to another is done via a **Merge** operation.

- The merge operation consists on taking all modifications done on each branch (they must have a common root commit identifier) and applying them together in a new Commit.

- What happens if a same block of code (ie: same function) was modified with different implementations on two separate branches ?

Synchronizing Branches

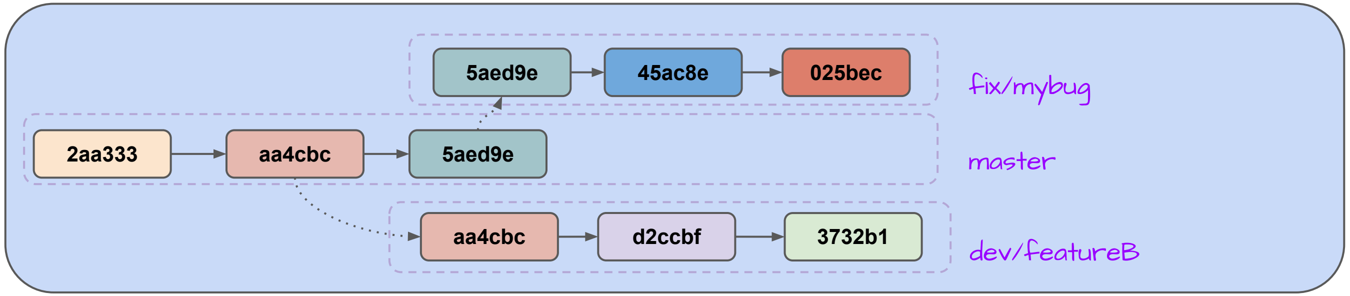
- We end up with a **Conflict** which must be solved before the Merge operation finishes.



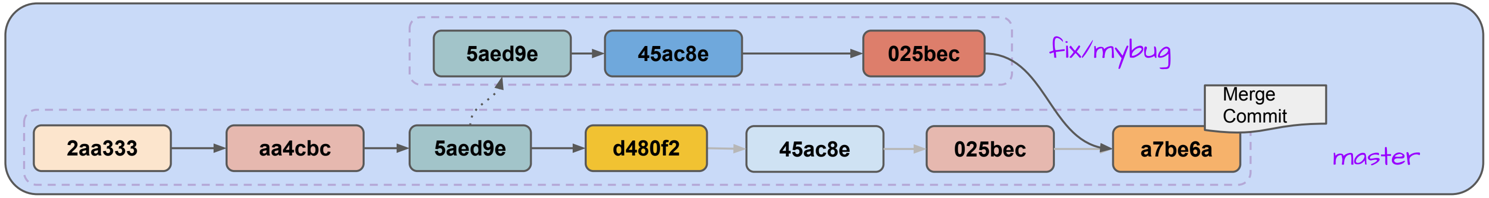
- Technically, Git will list the files automatically merged and the ones presenting conflicts. Git will also add markers in the files to show which block is in conflict with both versions so we can fix it.

Synchronizing Branches

- Say we have the three following branches and operations



- We want to merge the branch fix/mybug to master. We can either perform a merge in fix/mybug by pulling master OR pull fix/mybug from the master branch.

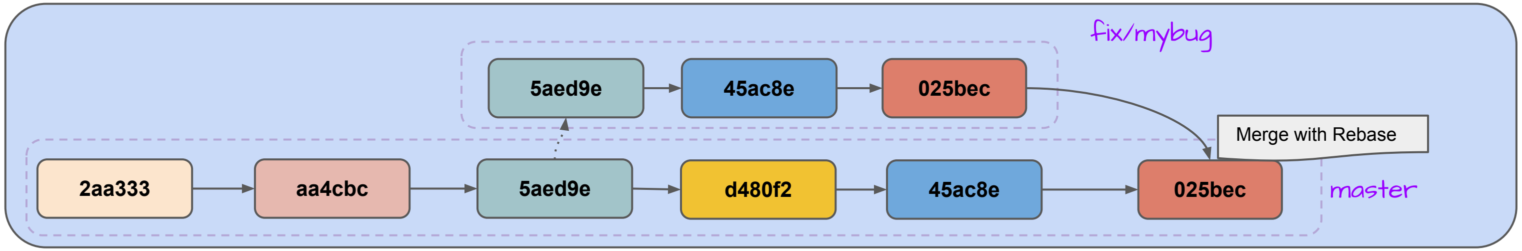


Synchronizing Branches

- An extra “merge commit” is created if Git can’t simply fast-forward the commits (there is no direct linear path from the target branch to the source branch).

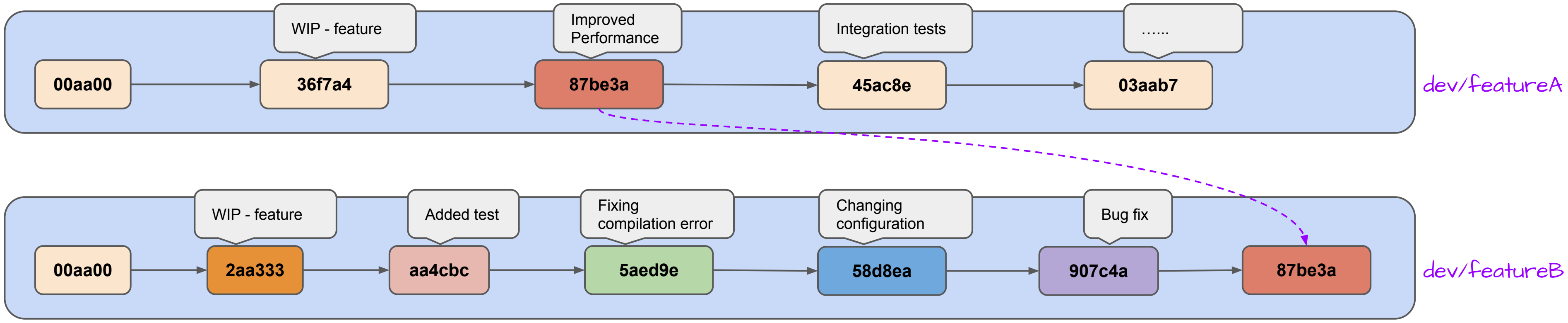
- What if we don’t want to create that extra “merge commit”? This is possible by using the **rebase** mechanism.

- Rebase in git helps to re-apply commits in order. This way, if we have no conflict between two branches, there is no need to create the “extra merge commit”.



Synchronizing Branches

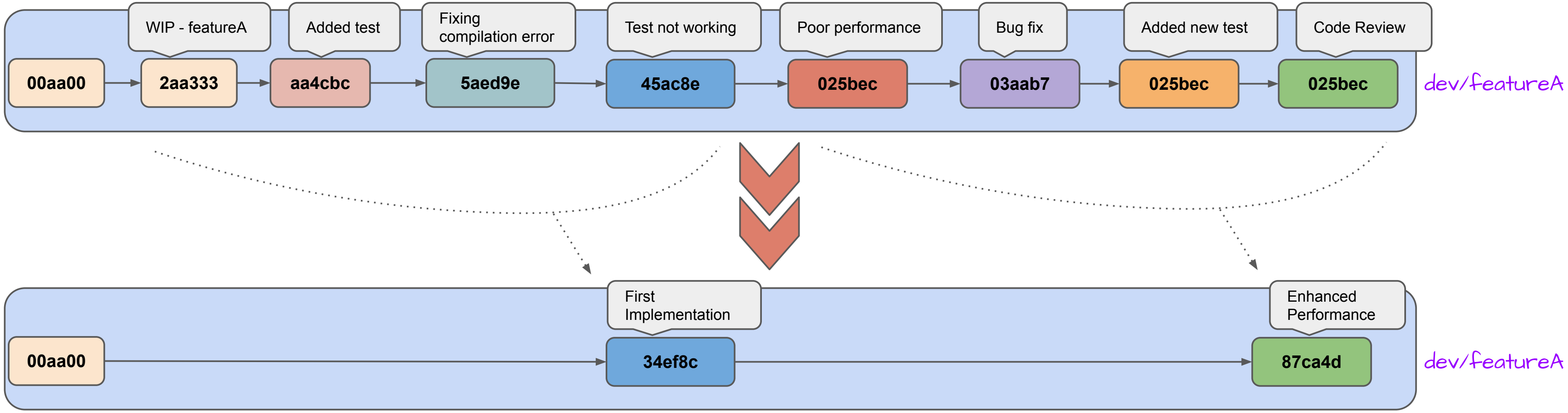
- Git also allows to pick a specific commit from another branch and apply it to your current workspace. This feature is called “cherry-pick”.



Rewriting History

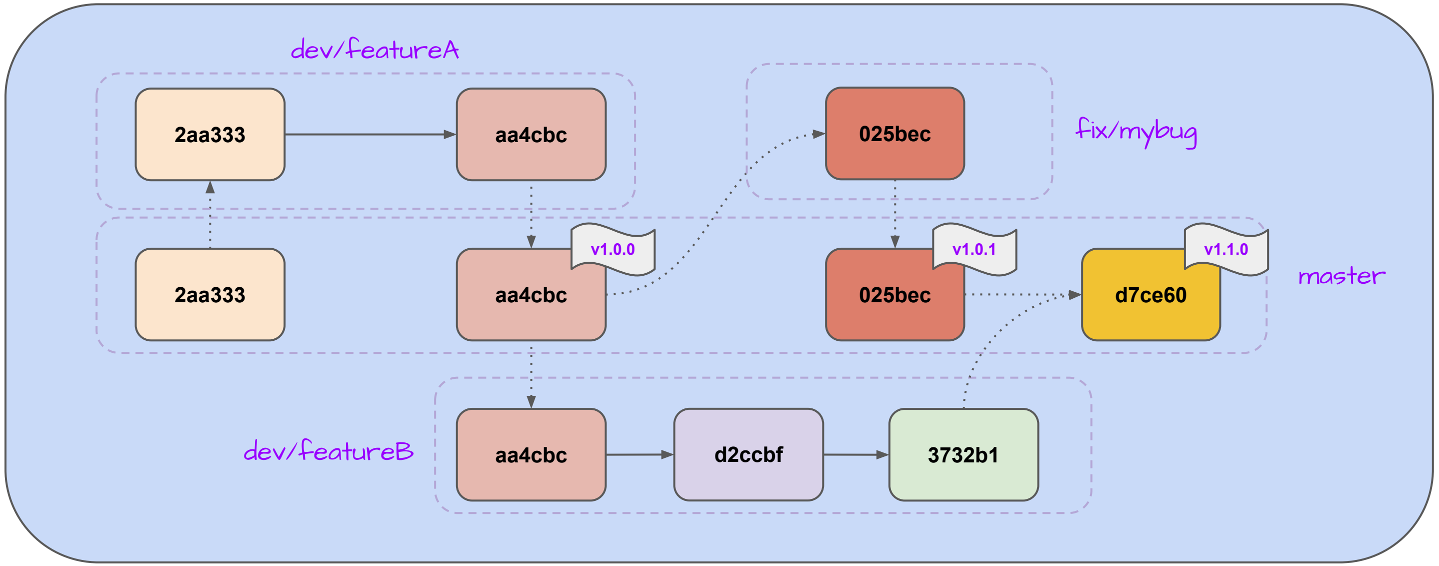
- Sometimes, when working on a complex feature, we may have many commits in the branch. Merging all the commits on master may result in a very complex commit graph, with unreadable history.

- The rebase command allows us to **Squash** commits to keep our versioning clean.



Tags

- It is possible to “tag” specific commits, for instance when releasing the application. It becomes easier to refer to these commits.

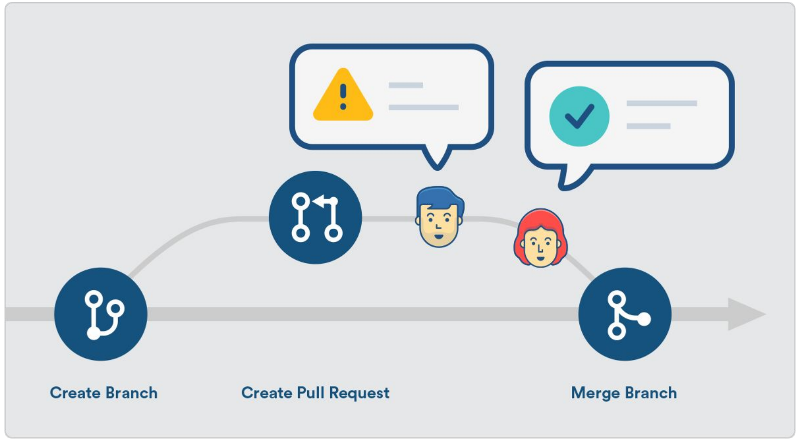


Pull Requests

- Let you share with others the changes you are willing to introduce in another branch / repository.

- Once opened, you can discuss and review each change with your pairs.

- Can be approved by pairs (to be released later) or rejected (with explanation).



Exercices