git merge and git rebase

(from <https://stackoverflow.com/questions/804115/when-do-you-use-git-rebase-instead-of-git-merge> )

CONTENTS

Introduction

git merge

git merge, fast-forward

git merge, three-way

git merge, three-way, automatic by Git

git merge, three-way, merge conflict, manual by a person

git rebase

Other notes

**Introduction**

Initially, Git searches for 3 commits

(1): the common ancestor commit

at this point in time, both branches had the same content

(and then evolved differently)

(2): latest commit on one branch

(3): latest commit on the other branch

Let’s say we have the main branch A and the new branch B

We want to merge the new branch onto the main branch

We can do the merge with git merge or got rebase.

# **git merge**

“git merge <branch B>”

Depending on the situation we do,

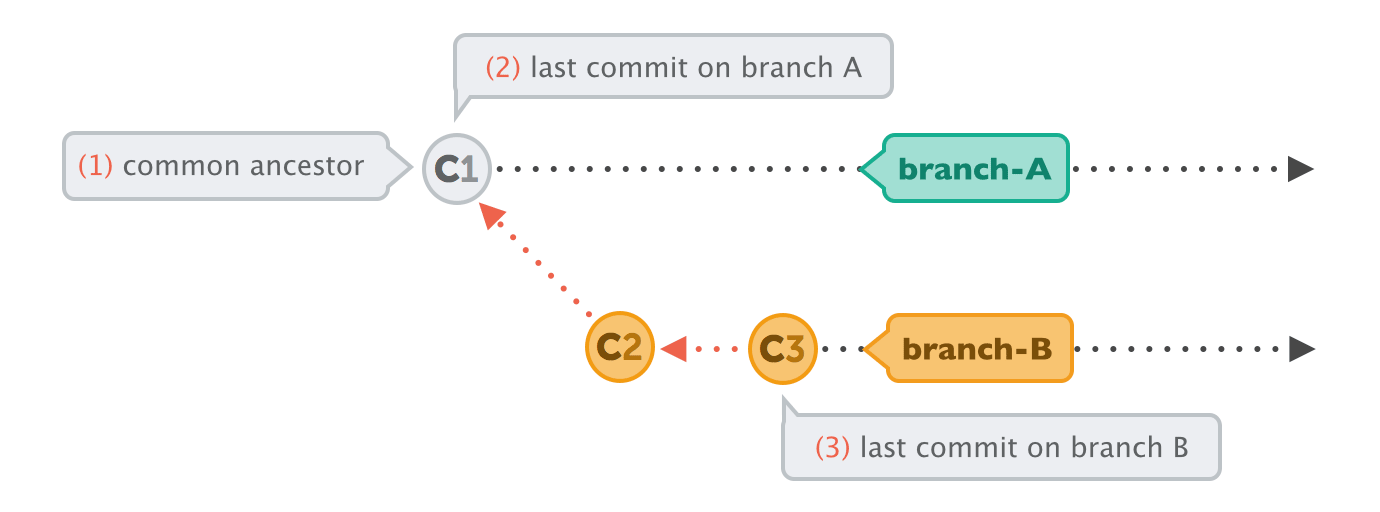
* git merge, fast-forward or
* git merge, three-way

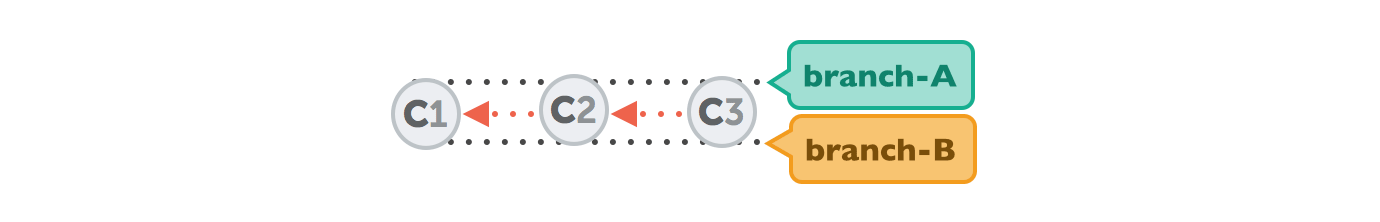
## git merge, fast-forward

situation: branch A does not have new commits since the branching happened

branch B has two new commits since the branching happened

we want to merge branch B onto branch A

Before the merge:

After the merge: 

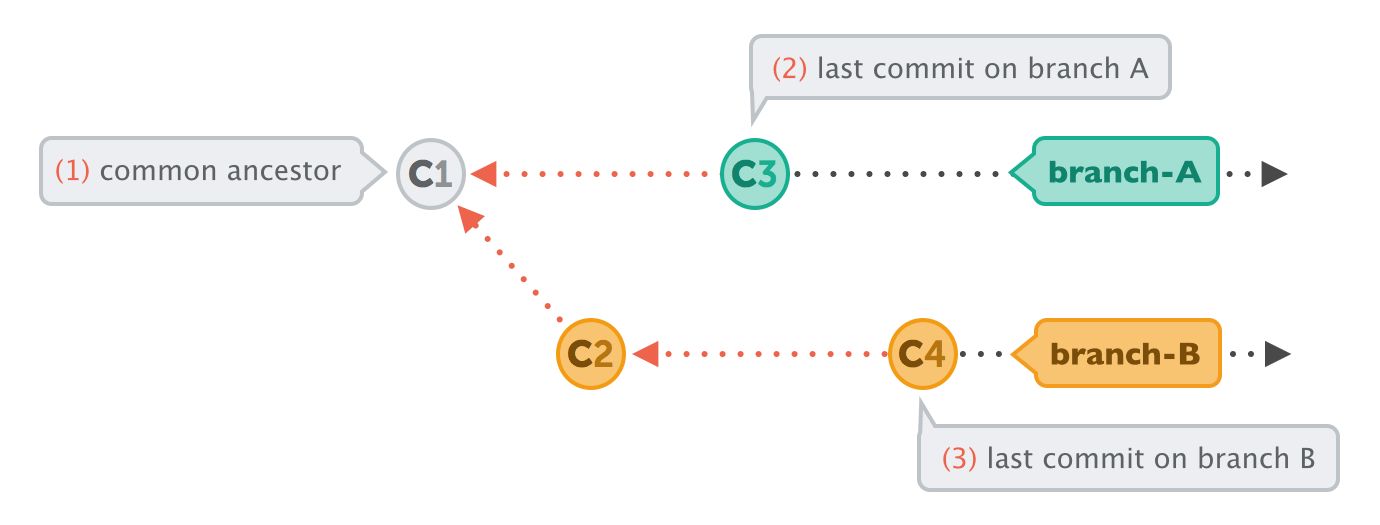
Finally, we have the branch AB, history of commits is: C1, C2, C3

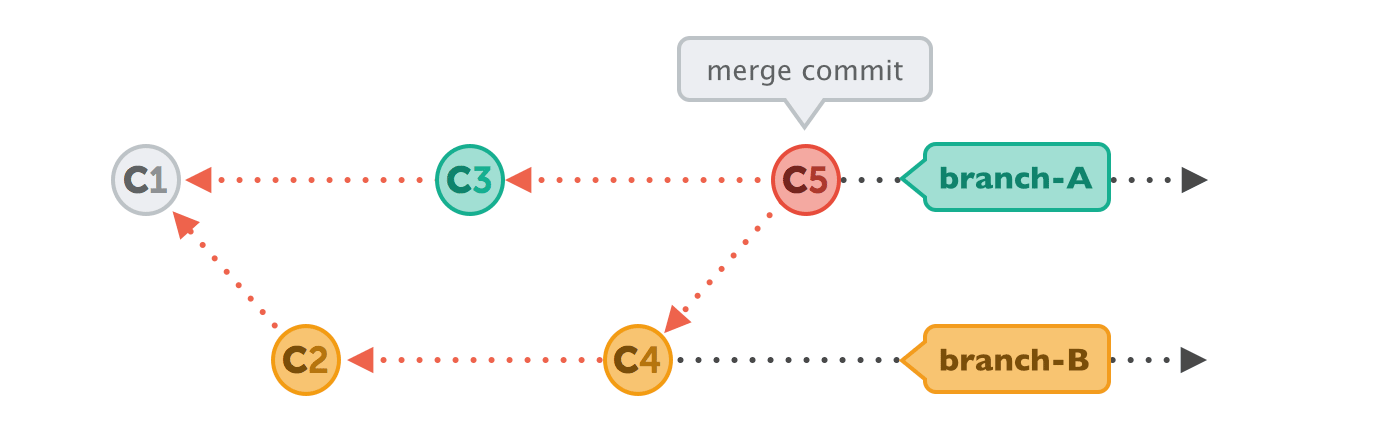
## git merge, three-way

### git merge, three-way, automatic by Git

situation: both branches have new commits since branching happened

we want to merge branch B onto branch A

Before the merge: 

After the merge:

Finally, we have the branch AB, history of commits is: C1, C2, C3, C4, C5

C5 is a new commit! It is called Merge Commit.

It contains the differences between previous commits

C5 gets created automatically by Git. And instead of wrapping a set of related changes, its purpose is to connect two branches, just like a knot. If you want to understand a merge operation later, you need to take a look at the history of both branches and the corresponding commit graph.

### git merge, three-way, merge conflict, manual by a person

exactly like above, but a merge conflict appears and Git cannot resolve it.

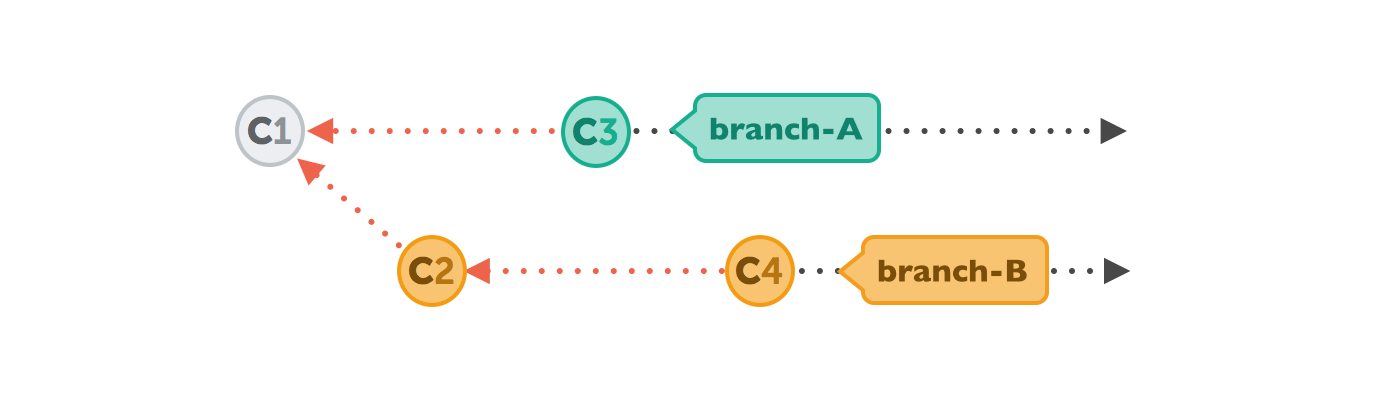
We resolve it manually.

I have no pictures here☹

# **git rebase**

situation: both branches have new commits since branching happened

we want to merge branch B onto branch A

Before merging:

THREE STEPS FOR GIT REBASE

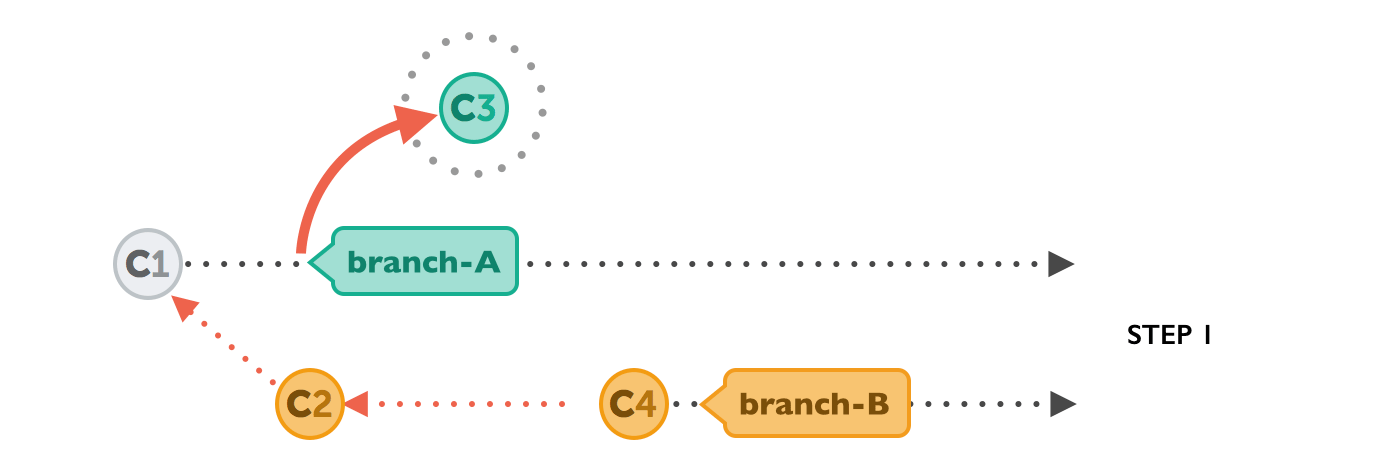
git rebase branch-A: Branch A is the branch we want to set as the new base.

git rebase <branchname> : changes the base of the current branch to be <branchname>

Git will "undo" all commits on branch-A that happened after the lines began to branch out (after the common ancestor commit).

My note: it is like we are removing commit history of branch A to create space for the commit history of branch B. Then we add again the commit history of branch A, AFTER the commit history of branch B

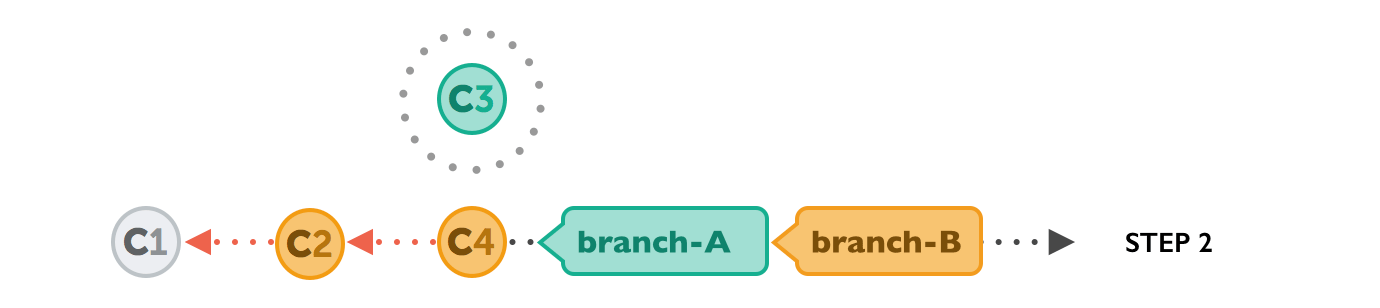
*This will work automatically if the changes are made in different parts of the files, but will require manual intervention if the changes were made in other files*



git checkout branch-A: Switch to branch-A

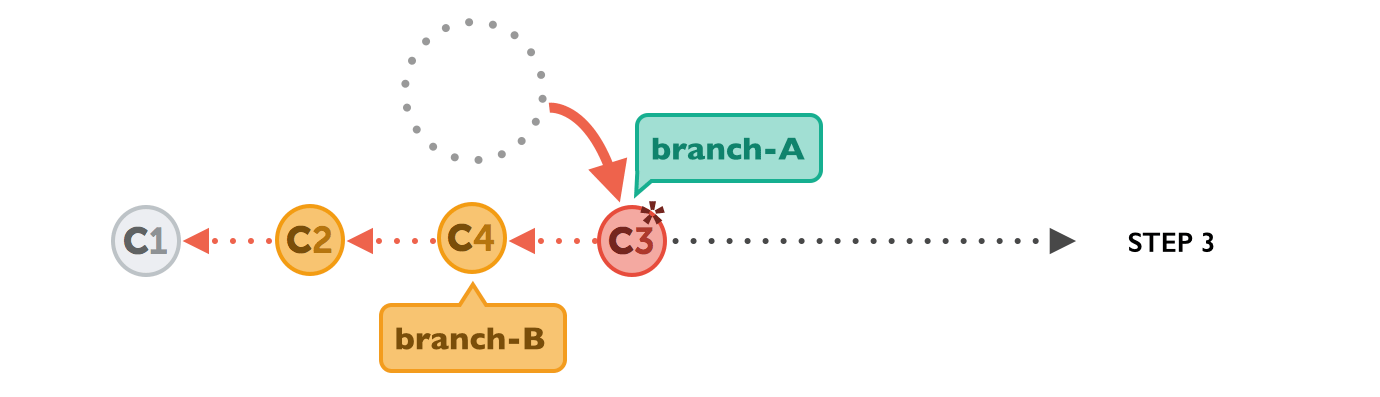
applies the commits from branch-B that we want to integrate.

At this point, both branches look exactly the same.



git merge branch-B: Merge/take the changes from branch-B to branch-A

the new commits on branch-A are now reapplied - but on a new position, on top of the integrated commits from branch-B (they are re-based).



# Other notes

git-merge Join two or more development histories together

git-rebase Reapply commits on top of another base tip

git merge and git rebase, both integrate changes from one branch into another

git merge and git rebase do the same thing (merge two branches) but with a different way

git merge adds a new commit, preserving the history.

git rebase “moves” a new branch A into a main branch B (see Step 3 above)

git merge preserves history whereas git rebase rewrites it.

you should use rebase only for cleaning up your local work - but never to rebase commits that have already been published.

**ADVANTAGES of git merge**

Simple and familiar.

Preserves complete history and chronological order.

Maintains the context of the branch.

**ADVANTAGES of git rebase**

Streamlines a potentially complex history.

Avoids merge commit “noise” in busy repos with busy branches.

Cleans intermediate commits by making them a single commit, which can be helpful for DevOps teams.

Produces linear commit history

**When to Use Merge vs. Rebase**

If you're working with a big team, use merge.

If you're working alone or on a small team, use rebase.