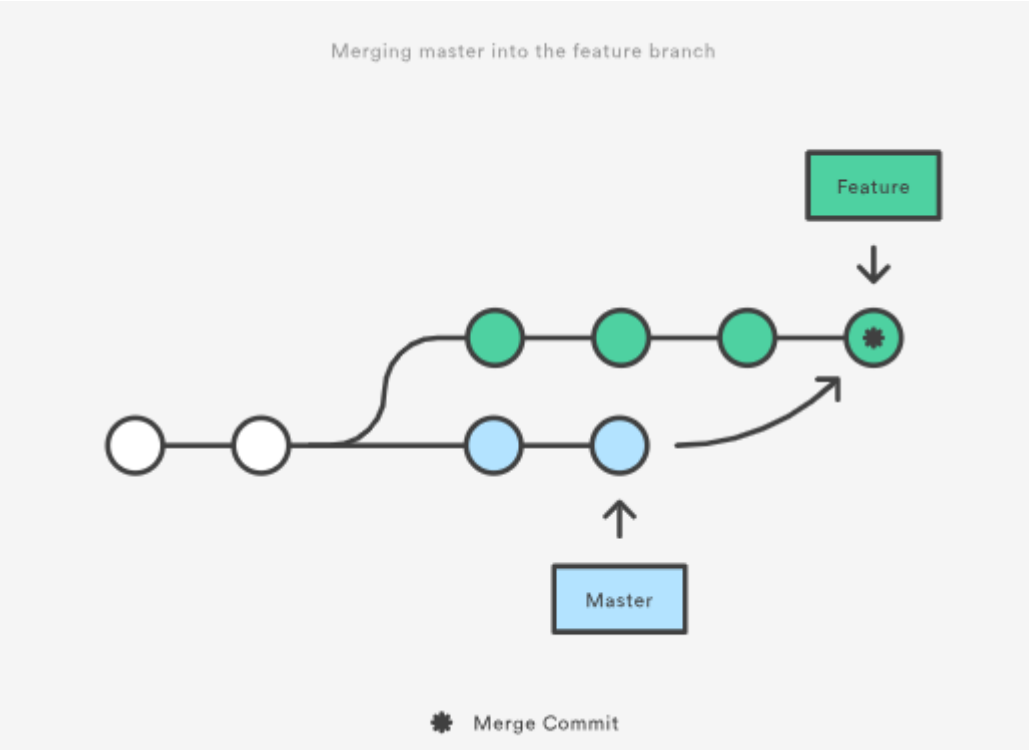
Advance Git Commands

Git Merge vs Rebase

In Git, there are two ways to integrate changes from one branch into another: the merge and rebase.

Merge:

The easiest option is to merge the master branch into the feature branch. This creates a new *"merge commit"* in the feature branch that ties together the histories of both branches, giving you a branch structure that looks like this:



merge branch
git checkout mastergit merge branch// git command to merge the branch(feature branch - add-header)
Ex: git merge add-header

Merge is always a forward-moving change record. Merging is nice because it's a *non-destructive* operation. The existing branches are not changed in any way. This avoids all of the potential pitfalls of rebasing (discussed below).

Rebase:

Merge is nice but on the other hand, this also means that the feature branch will have an extraneous merge commit every time you need to incorporate upstream changes.

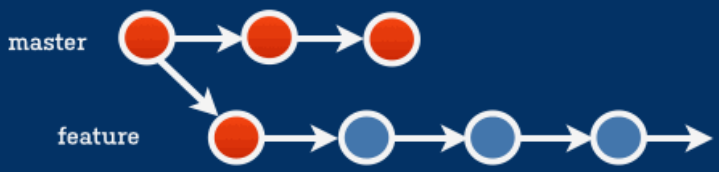
The first thing to understand about Git rebase is that it solved the same problem as git merge. Both of these commands are designed to integrate changes from one branch into another branch-they just do it in very different ways. Rebasing is the process of ***moving or combining a sequence of commits to a new base commit***. The major benefit of rebasing is that you get a much cleaner project history. First, it eliminates the ***unnecessary merge commits required by git merge***.

Rebasing works by ***transferring each local commit to the updated master branch one at a time***. This means that you catch merge conflicts on a commit-by-commit basis rather than resolving all of them in one massive merge commit. In turn, this makes easier to figure out where bugs were introduced and if necessary to roll back changes with minimal impact on the project.

git checkout branchgit rebase mastergit checkout mastergit merge branch// git command to merge the branch(feature branch - add-header)
Ex: git merge add-header

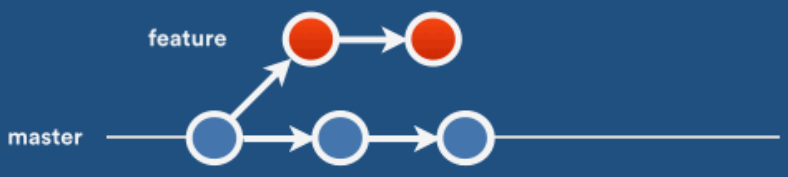
What is a **rebase**?

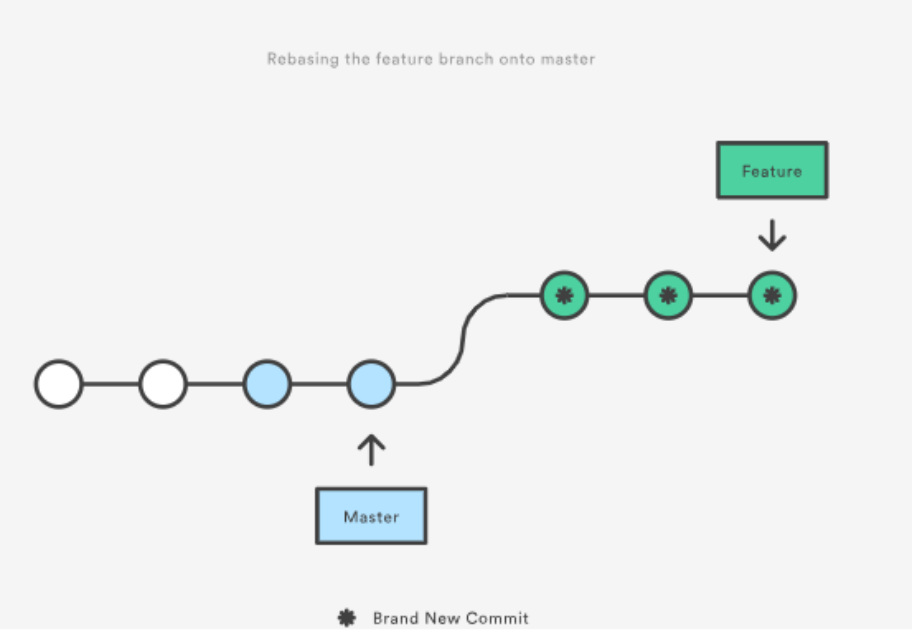
preserving the order
of change-sets



What is a rebase?

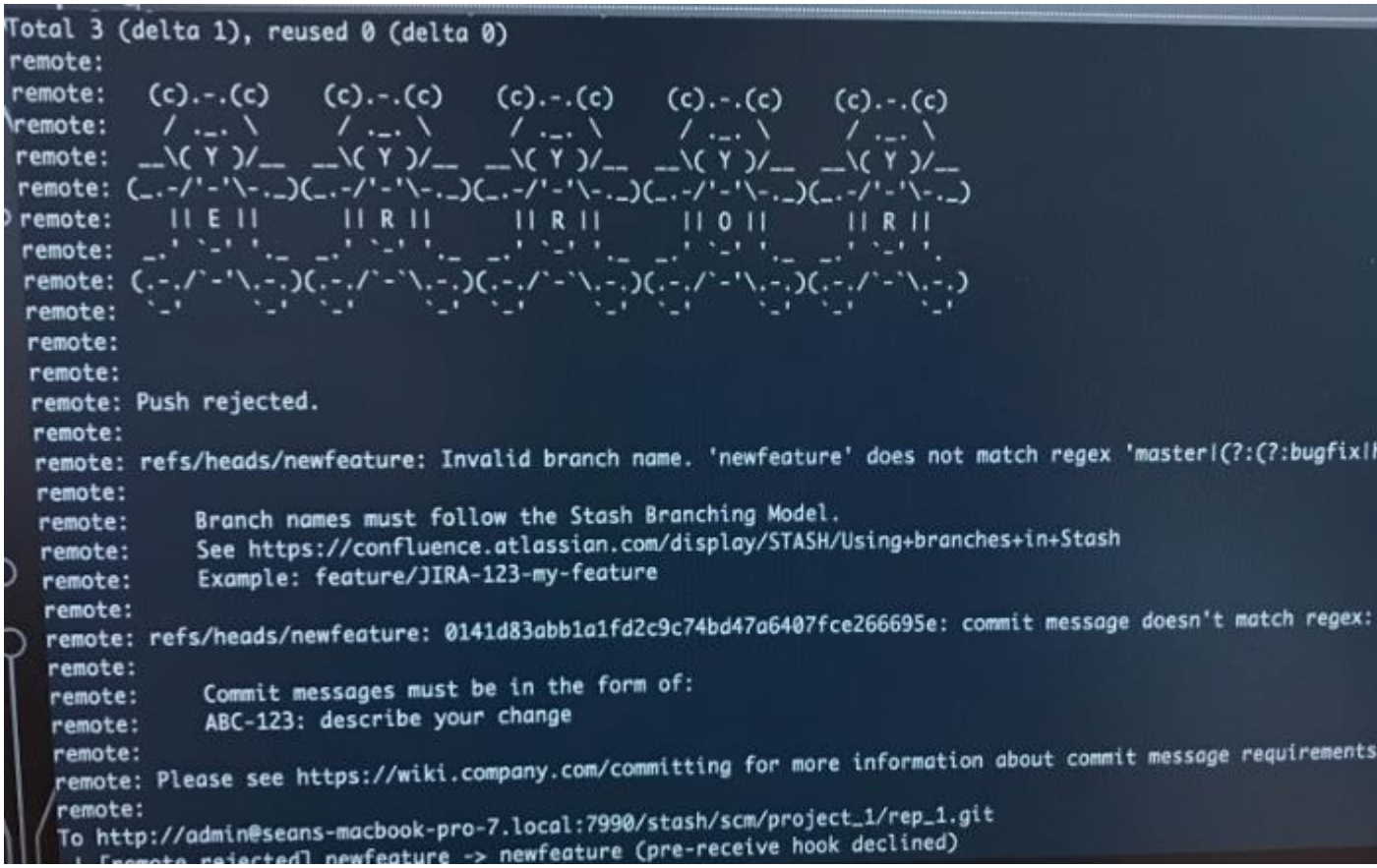
It's a way to replay commits,
one by one, on top of a branch





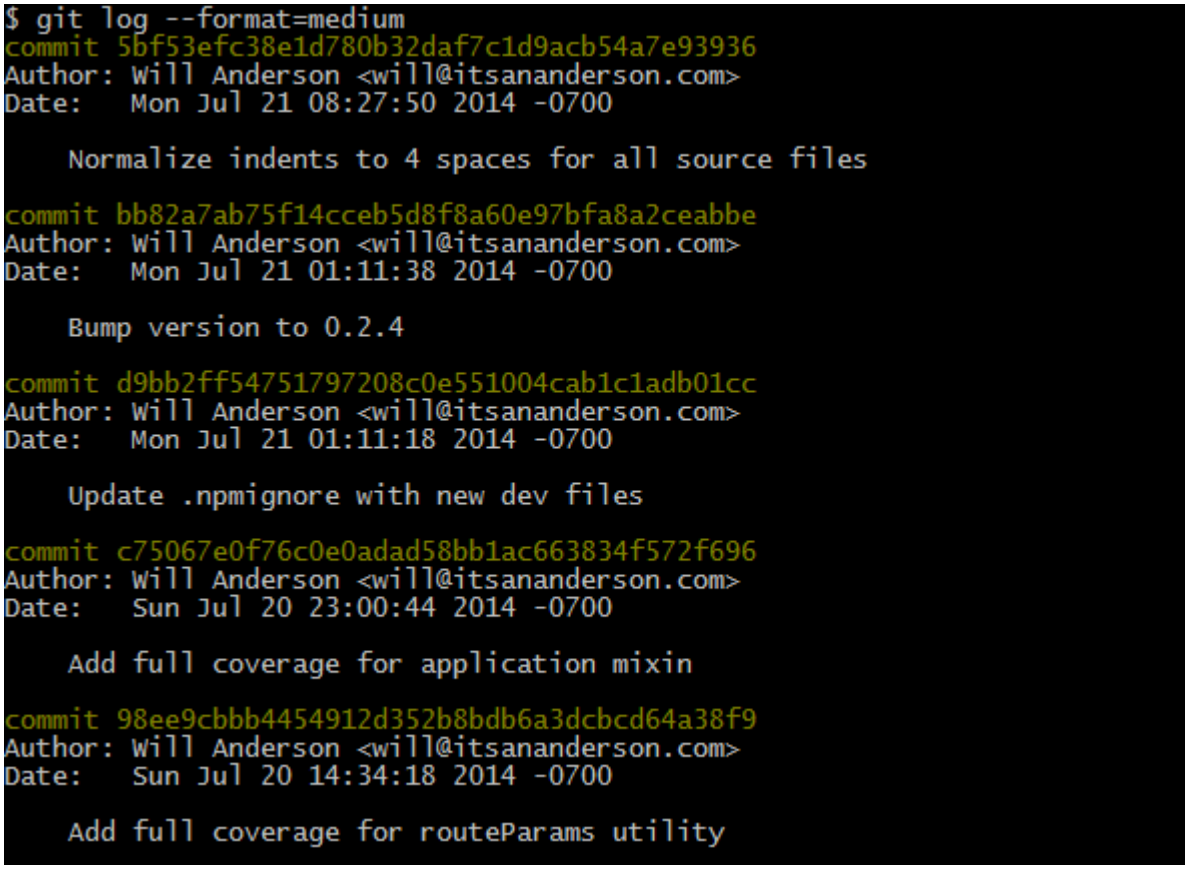
7.Reset vs Revert vs Checkout

If you are using JIRA it requires you to mention the story number when committing the changes then push the changes but sometimes we tend to forget to mention the story number and it causes you to not to push the changes to the repository. You might have seen this kind of error in git bash window.



git push rejected

so, in this kind of situation, you can reset or revert or check out the changes. Before doing this do git log to know in which commit you missed to mention the JIRA story number.



git log — list of git commits

Reset

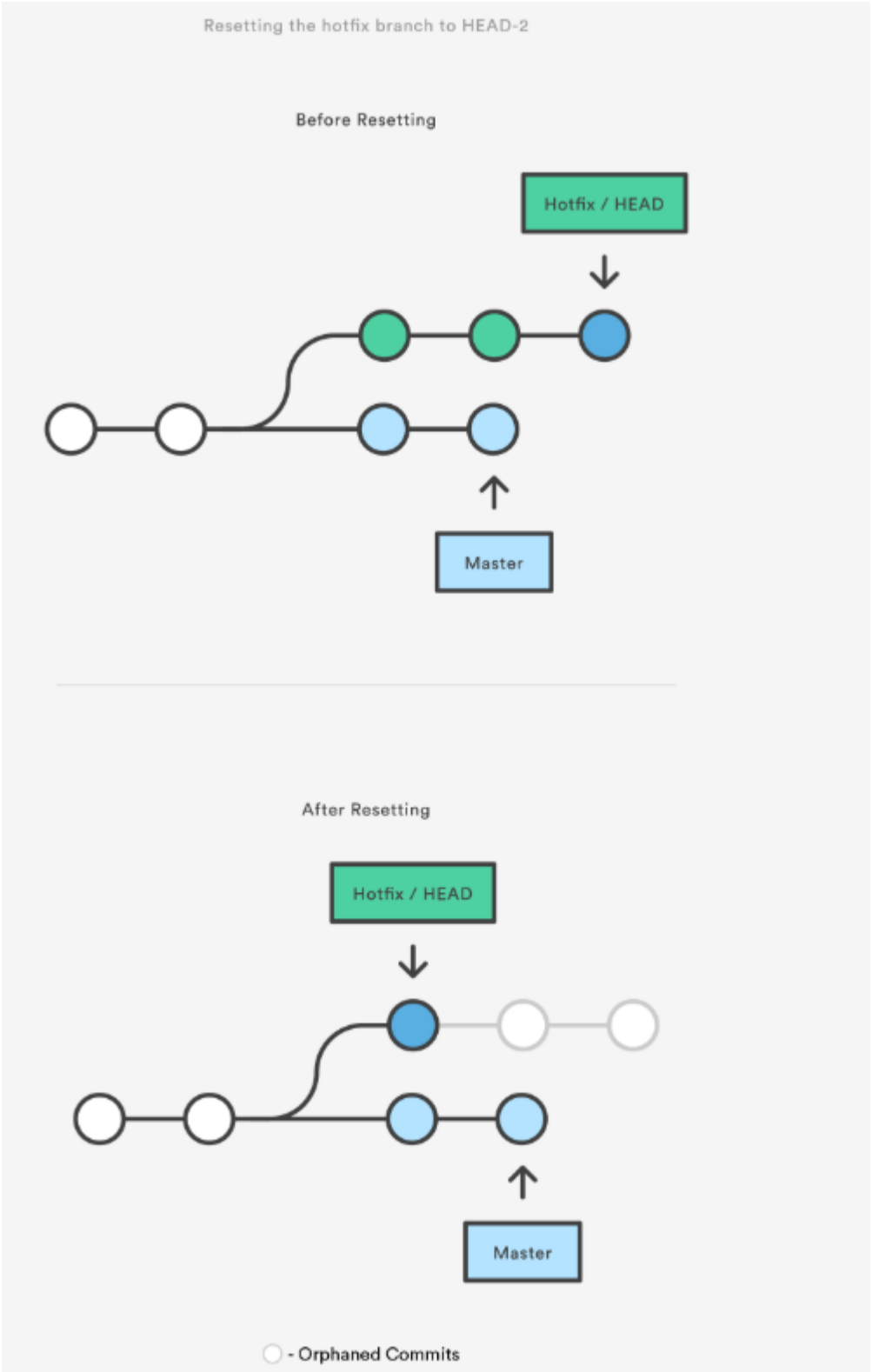
A reset is an operation that takes a specified commit and resets the “three trees” to match the state of the repository at that specified commit. A reset can be invoked in three different modes which correspond to the three trees.

git reset can be done in file level and commit level as well.

In addition to moving the current branch, you can also get git reset to alter the staged snapshot and/or the working directory by passing it one of the following flags:

- --soft – The staged snapshot and working directory are not altered in any way.

- --mixed – The staged snapshot is updated to match the specified commit, but the working directory is not affected. This is the default option.
- --hard – The staged snapshot and the working directory are both updated to match the specified commit.



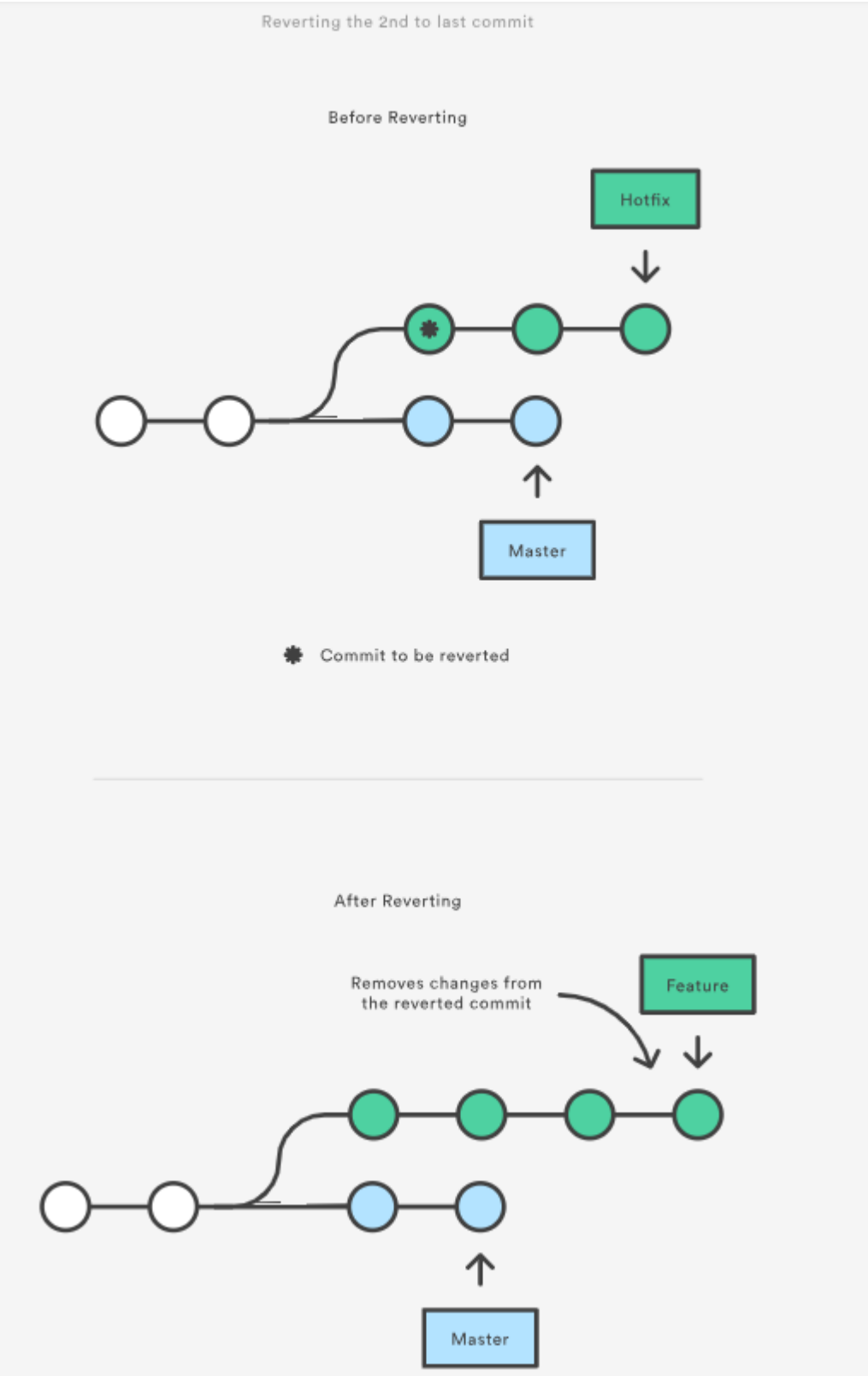
git reset

git reset f5c5cac0033439c17ebf905d4391dc0705dbd5f1//

f5c5cac0033439c17ebf905d4391dc0705dbd5f1 - commit you want to reset

Revert

Revert is an operation that takes a specified commit and creates a new commit which inverses the specified commit.



git revert
git revert f5c5cac0033439c17ebf905d4391dc0705dbd5f1//
f5c5cac0033439c17ebf905d4391dc0705dbd5f1 - commit you want to revert

checkout
git checkout f5c5cac0033439c17ebf905d4391dc0705dbd5f1//
f5c5cac0033439c17ebf905d4391dc0705dbd5f1 - commit you want to checkout

git revert *is the best command to use in case of an issue because it will not lose the commits whereas reset orphans the commits.*

Git Reset vs Revert vs Checkout reference

The table below sums up the most common use cases for all of these commands. Be sure to keep this reference handy, as you'll undoubtedly need to use at least some of them during your Git career.

Command	Scope	Common use cases
`git reset`	Commit-level	Discard commits in a private branch or throw away uncommited changes
`git reset`	File-level	Unstage a file
`git checkout`	Commit-level	Switch between branches or inspect old snapshots
`git checkout`	File-level	Discard changes in the working directory
`git revert`	Commit-level	Undo commits in a public branch
`git revert`	File-level	(N/A)