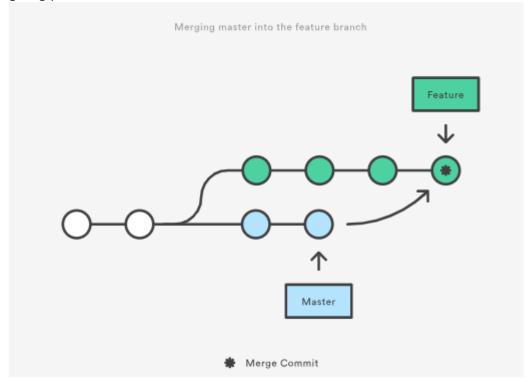
Advance Git Commands

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 - addheader)

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.

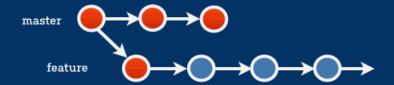
git checkout branchgit rebase mastergit checkout mastergit merge branch// git command to merge the branch(feature branch - add-header)

Ex: git merge add-header

Op.

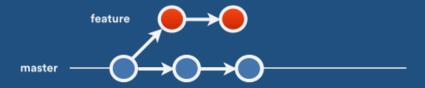
What is a rebase?

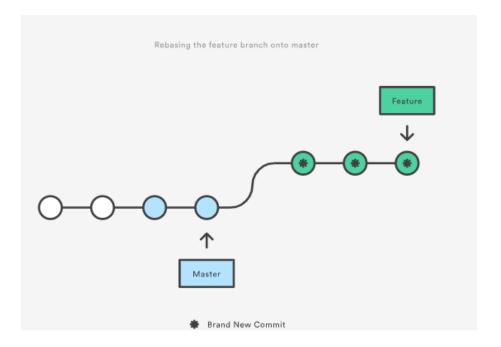
preserving the order of change-sets



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 <u>JIRA</u> 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.

```
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote:
           (c).-.(c)
                        (c).-.(c)
                                     (c).-.(c)
                                                  (c).-.(c)
                                    _\(\(\frac{1}{2}\)\__
remote:
                                                                / ._. \
          _/( Y )/_
                                                 __\( Y )/__
                       __/( Y )/__
remote:
                                                              _/( Y )/_
remote: (_.-/'-'\-._)(_.-/'-'\-._)(_.-/'-'\-._)(_.-/'-'\-._)
            ILE II TIRII TIRII TIOTI TIRII
 remote:
 remote: (.-./`-'\.-.)(.-./`-'\.-.)(.-./`-'\.-.)(.-./`-'\.-.)
 remote:
 remote:
 remote:
  remote: Push rejected.
  remote:
  remote: refs/heads/newfeature: Invalid branch name. 'newfeature' does not match regex 'masterl(?:(?:bugfix|
  remote:
              Branch names must follow the Stash Branching Model.
  remote:
              See https://confluence.atlassian.com/display/STASH/Using+branches+in+Stash
  remote:
              Example: feature/JIRA-123-my-feature
  remote:
  remote: refs/heads/newfeature: 0141d83abb1a1fd2c9c74bd47a6407fce266695e: commit message doesn't match regex:
              Commit messages must be in the form of:
   remote:
              ABC-123: describe your change
   remote:
   remote: Please see https://wiki.company.com/committing for more information about commit message requirements
   To http://admin@seans-macbook-pro-7.local:7990/stash/scm/project_1/rep_1.git
                 iected  newfeature -> newfeature (pre-receive hook declined)
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 --format=medium
commit 5bf53efc38e1d780b32daf7c1d9acb54a7e93936
Author: Will Anderson <will@itsananderson.com>
           Mon Jul 21 08:27:50 2014 -0700
     Normalize indents to 4 spaces for all source files
commit bb82a7ab75f14cceb5d8f8a60e97bfa8a2ceabbe
Author: Will Anderson <will@itsananderson.com>
           Mon Jul 21 01:11:38 2014 -0700
Date:
     Bump version to 0.2.4
commit d9bb2ff54751797208c0e551004cab1c1adb01cc
Author: Will Anderson <will@itsananderson.com>
           Mon Jul 21 01:11:18 2014 -0700
Date:
     Update .npmignore with new dev files
commit c75067e0f76c0e0adad58bb1ac663834f572f696
Author: Will Anderson <will@itsananderson.com>
Date: Sun Jul 20 23:00:44 2014 -0700
     Add full coverage for application mixin
 commit 98ee9cbbb4454912d352b8bdb6a3dcbcd64a38f9
Author: Will Anderson <will@itsananderson.com>
           Sun Jul 20 14:34:18 2014 -0700
      Add full coverage for routeParams utility
```

git log — list of git commits

Reset

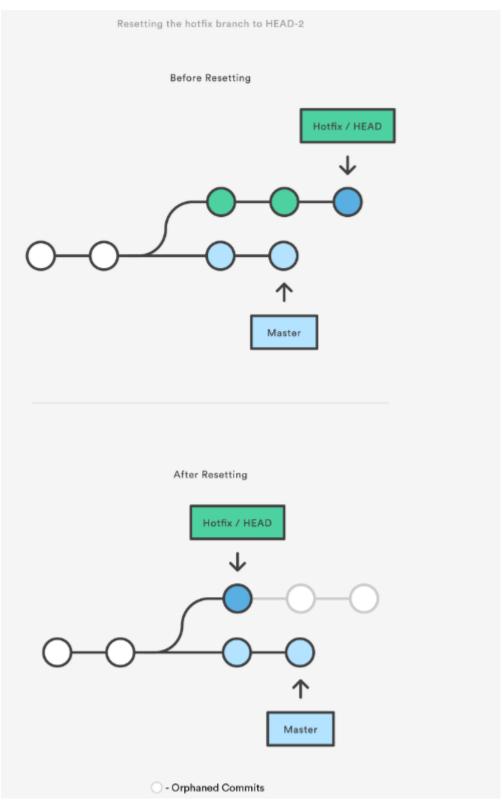
A reset is an operation that takes a specified commit and resets the "three trees" to match the state of the repository at that specified commit. A reset can be invoked in three different modes which correspond to the three trees.

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:

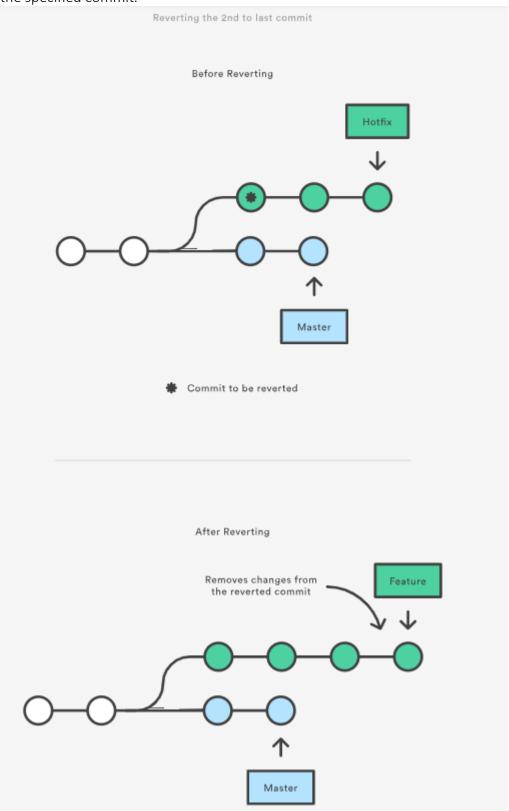
• --soft – The staged snapshot and working directory are not altered in any way.

- --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 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)