Using Git Rebase to Squash Commits - Revisited

Posted At : August 31, 2010 11:04 AM | Posted By : Bob Silverberg
Related Categories: Git

I wrote a blog post awhile back about **using Git rebase to squash a bunch of small commits into a single commit.** I really like this approach as it allows me to commit frequently without having to litter my public repo with lots of meaningless commits. I've recently been researching Git workflows, trying to come up with a workflow for contributors to **ValidateThis**, and noticed that there's an easier way to start a rebase for all commits in a branch, which is generally what I need to rebase.

When I start working on a new feature, or even a bug fix, I generally start a new branch for it:

```
git checkout -b newBranch
```

I then make my changes, committing frequently. When I'm ready to merge my changes back into the *master* branch I don't want all of those small, meaningless commits showing up, so I use *get rebase* to squash them. In the pervious article I discussed how I'd use *git log* to determine how many commits I wanted to rebase and then use *git rebase -i HEAD~n*, where *n* is the number of commits that I want to rebase. This is a useful feature if your commits are already in your *master* branch, but if all you want to do is rebase all of the commits in the current branch (*newBranch* in this example), then you can simply do:

```
git rebase -i master
```

Which tells Git to rebase all of the commits in the current branch that are not already in *master*. After issuing that command you can follow the steps as described in the **previous article** to choose which ones to squash and then to create a new commit message.