Git Commands

Command	**Source Files**	**Index**	**Commit Chain**	**References**
git add	Unchanged	Updated with new file	Unchanged	Unchanged
git rm	File removed	File removed	Unchanged	Unchanged
git mv	File moved/renamed	Updates file name/location	Unchanged	Unchanged
git commit	Unchanged	Unchanged	A new commit object is created from the index and added to the top of the commit chain	HEAD in the current branch points to new commit object
git tag	Unchanged	Unchanged	Unchanged	A new tag is created
git revert	Changed to reflect reversion	Uncommitted changes discarded	New commit created; no actual commits removed	HEAD of current branch points to new commit
git reset	Unchanged	Discard uncommitted changes	Unchanged	Unchanged (unless form as used below; then HEAD of current

				branch moves to a prior commit)
git branch	Unchanged	Unchanged	Unchanged	A new branch is created in .git/refs/heads HEAD for the new branch points to HEAD of the current branch; the current branch is set to the new branch
git checkout	Modified to match commit tree specified by branch or commit ID; untracked files not deleted	Unchanged	Unchanged	Current branch reset to that checked out; HEAD (in .git/HEAD) now refers to last commit in branch
git rebase	Unchanged	Unchanged	Parent branch commit moved to a different commit	Unchanged
git am	Modified by patch	Updated to reflect patch	New commit object created and added to top of commit chain	HEAD; points to new commit object

git apply	Modified by patch (unless --check option specified)	Unchanged (unless --index option specified)	Unchanged	Unchanged

As your project grows through a series of commits, your repository may grow large in size. You can optimize and compact your repository by issuing **git gc**.

Rebasing

There are potential problems associated with rebasing, some of which arise from its Orwellian nature. When you do a rebase, you change the history of commits, because the changes are temporarily removed and then all put back in. This leads to at least several problems:

- Presumably, you have been testing your work as you have been progressing it, and the code branch you were testing against came from the earlier branch point. There is no guarantee that just because things worked before when you tested they still will. Indeed, problems that arise may be very subtle.
- You may have done your work in a series of small commits, which is a good practice when trying to locate where problems may have been introduced, for instance when using bisection. But now you have collapsed matters into a larger commit.
- If anyone else has been using your work, has been pulling changes from your tree, you have just pulled the rug out from under their feet. In this case, many developers consider rebasing to be a terrible sin.

If you are doing merging rather than rebasing, other problems can arise, especially if you do it often or not at major stable development points. Once again, history can be confusing in your project. Whatever strategy you choose, think things through and only do merging or rebasing at good, well-defined stages of development to minimize problems.