# Git Cheatsheet II

**NOTE: This is an incomplete reference. Check the man pages for more features.**

[**Git Cheatsheet II**](#_2z0cj7i7we51) **1**

[How to Use](#_fzamcqgw3nk7) 1

[Rebase](#_2og5bly6wf75) 1

[Reset](#_bkbprbi6eyfo) 2

[Revert](#_ealkcjkcov2g) 3

[Bisect](#_hdfttq3on5cy) 3

[Stash](#_7wst67jbg861) 4

[Cherry-pick](#_k4mvni8jyfge) 5

## How to Use

Values in square brackets are **[optional]**.

Items in ***italics*** are names that you specify.

**[...]** means "and any additional ones like the previous".

A ***commitish*** is a commit hash, branch, tag, or other method of referencing a particular commit.

## Rebase

**MOST IMPORTANT RULE**: ***Never rebase any commits that have been duplicated in any clone of a repo, i.e. anything that's been pushed. It will make people very unhappy and make you very unpopular.***

Takes commits from one branch and reapplies them after another branch, effectively moving the commits. Interactively, it allows you to consolidate and clean up commit history.

***"Does anyone else in the universe, including bare repoes, have a clone of the commits I am about to rebase?" If the answer is "yes", DON'T REBASE. There be dragons.***

**git fetch; git rebase**

Fetch origin/master then rebase all local commits onto origin/master. Equivalent to git pull --rebase.

**git rebase *otherbranch***

Apply all commits on the current branch onto the *otherbranch*, beginning where the branches diverged.

**git rebase *otherbranch* *abranch***

Apply all commits from *abranch* onto *otherbranch*, beginning where the branches diverged.

**git rebase -i *branchname* [*specifiedbranch*]**

Interactively rebase current (or *specifiedbranch*) branch onto *branchname*. This allows you to drop or include individual commits.

**git rebase -i HEAD~2**

Interactively rebase to a prior commit, e.g. two commits prior to HEAD.

**git pull --rebase**

Run git rebase instead of git merge after the git fetch.

**git pull --rebase=interactive [remote] [branch]**

Perform an interactive rebase after a git fetch.

**git rebase --continue**

Continue the rebasing process after conflict resolution.

**git rebase --abort**

Abort the rebase process (e.g. after conflicts you don't want to deal with).

**git rebase --onto *destbranch* *divergebranch* *thisbranch***

Rebase *thisbranch* onto *destbranch* starting from the commit where *thisbranch* diverged from *divergebranch*.

## Reset

With commits, moves HEAD and branches to a commit, optionally reset index and/or working tree. With files, unstages files or copies files from a commit to staging.

**git reset --soft *commitish***

Moves HEAD and current branch to point to *commitish*. Staged files and working tree files are unchanged.

**git reset [--mixed] *commitish***

Moves HEAD and current branch to point to *commitish*, updates staged files to match new HEAD. Working tree files are unchanged.

**git reset --hard *commitish***

Moves HEAD and current branch to point to *commitish*, updates staged files to match new HEAD, updates working tree files to match new HEAD.

**git reset file**

Copies file from current HEAD to staging. Effectively unstages the file.

**git reset *commitish* file**

Copies file from *commitish* to staging.

## Revert

Reverts specific commits, applying the reversion to the working tree, making new commits. The opposite of git cherry-pick.

**git revert commitish**

Revert a particular commit.

**git revert -n commitish1 commitish2 [...]**

Revert several commits, but do not make a new commit. Follow with git revert --continue.

**git add file**

Necessary after resolving a conflict, just like with a normal merge.

**git revert --continue**

Continue the revert process after git revert -n or conflict resolution.

**git revert --abort**

Abandon the current reversion process.

## Bisect

Binary search through commits looking for the one that broke things.

**git bisect start**

Start the bisect process.

**git bisect bad**

Mark this commit as bad.

**git bisect good *commitish***

Mark a particular commit as good.

**git bisect good**

Mark this commit as good.

**git bisect bad *commitsh***

Mark a particular commit as bad.

**git bisect reset**

Get back to the original starting point and stop bisecting.

**git bisect reset bisect/bad**

Check out the first bad commit and stop bisecting.

## Stash

Stores working tree (and optionally index) safely off to the side while you do other work.

**git stash [push]**

Save the current index (staging) and working tree onto stash stack.

**git stash [push] --keep-index**

Save the current working tree onto stash stack, leaving the index (staging) intact.

**git stash [push] --include-untracked**

Same as git stash but also saves untracked files.

**git stash pop [stash@{0}]**

Restore the working tree from the top of the stash stack.

**git stash pop --index**

Restore the current index (staging) and working tree from the top of the stash stack. This might conflict if you have things in staging when you run this command.

**git stash pop stash@{2}**

Restore the working tree from the 2nd-from-the-top of the stash stack.

**git stash apply**

Like git stash pop, except leaves the stash stack intact.

**git stash apply stash@{2}**

Apply a specific stash (e.g. 2nd-from-the-top) from the stack.

**git stash apply --index**

Like git stash pop --index, except leaves the stash stack intact.

**git stash drop [stash@{0}]**

Drop a stash from the stash stack, doing nothing to index or working tree.

**git stash list**

Show the current stash stack.

**git stash clear**

Destroy the current stash stack, doing nothing to index or working tree.

**git stash branch *branchname* [stash@{0}]**

Create a branch at the last commit prior to the given stash, pop the given stash on that new branch.

## Cherry-pick

Applies specific commits to the working tree, making new commits. The opposite of git revert.

**git cherry-pick *commitish* [*commitish* ...]**

Apply the specified commit after the current commit.

**git cherry-pick -x *commitish***

Automatically add "(cherry picked from commit ...)" to the commit message.

**git cherry-pick -ff *commitish***

If the commit is a direct child of HEAD, just fast-forward HEAD to that commit instead of making a new one.

**git cherry-pick -n *commitish* [*commitish* ...]**

Don't make a new commit; useful when you want to take a bunch of commits and make them into a single commit.

**git add file**

Necessary after resolving a conflict, just like with a normal merge.

**git cherry-pick --continue**

Continue the cherry picking process after resolving a conflict.

**git cherry-pick --abort**

Abandon the cherry picking process and pretend you never started it.