Reverting and Resetting Commits

coursera.org/learn/git-distributed-development/supplement/AnQFC/reverting-and-resetting-commits

From time to time, you may realize that you have committed changes unwisely. Either you may have made changes you never should have, or you committed prematurely.

You can back out a particular commit with:

```
1

$ git revert commit_name
```

where **commit_name** can be specified in a number of ways and need not be the most recent.

Commits can be delineated with:

- **HEAD** : most recent commit
- **HEAD~** : previous commit (the parent of **HEAD**)
- HEAD~~ or
- HEAD~2 : the grandparent commit of **HEAD**
- **{hash number}** : specific commit by full or partial **sha1** hash number
- **{tag name}** : a name for a commit.

Note that **git revert** puts in a new commit set of changes, i.e. a reversed patch as an additional commit. This is the appropriate thing to do if someone else has downloaded a tree containing the changes that have been reversed. This will change your working copy of source files.

In short, **git revert** builds and adds a new commit object, sets **HEAD** to it and updates the working directory (changes induced by **git revert**):

Command	Source Files	Index	Commit Chain	References

Command	Source Files	Index	Commit Chain	References
git revert	Changed to reflect reversion	Uncommitted changes discarded	New commit created; no actual commits removed	**HEAD** of current branch points to new commit

In the case where you are the only one that has seen the updated repository, the **git reset** command is more appropriate. For example, if you want to pull out the last two commits, you can do:

```
1

$ git reset HEAD~2
```

This will ***not*** change your working copy of source files; it just reverses the commits and makes the index match the specified commit (changes induced by **git reset**):

Command	Source Files	Index	Commit Chain	References
git reset	Unchanged	Discard uncommitted changes	Unchanged	Unchanged (unless form as used below; then **HEAD** of current branch moves to a prior commit)

If using any one of the options **--soft**, **--mixed**, **--hard**, **--merge** or **--keep**, the behavior is different. In this case, the **HEAD** reference in the current branch is also set to the specified commit, optionally modifying the index and the source files to match:

- **--soft**: just moves the current branch to prior commit object (index unchanged in this case)
- **--mixed** (default): also updates the index to match new head (un-stages everything)
- **--hard**: same as **--mixed**, but also updates working directory to match new head (un-edits your files).

Suppose you have been furiously coding and realize that your last three commits are still a work in progress, but everything before that should be made available to others. Then, you should create a new working branch while resetting the main branch back three commits, as in:

```
3

$ git checkout work
```

which restores the main branch to its previous state, while leaving the speculative work in work where you will continue to play. We will discuss branches in detail later.