Fixing Mistakes

Amend

Oops. You *just* committed something with a bad message or the wrong files. Redo a recent commit's file changes or comment. Get the file system in the state you want it before issuing this command.

```
git commit --amend
```

or if you missed a few files that weren't in the last commit

```
git commit -a --amend
```

or if you need to remove a file that wasn't supposed to be in the commit

```
# Commit two files.
# Oops. Only meant to commit one.
git reset --soft HEAD^
# Puts files back into staged area.
# Edit files and/or remove one file from staging.
git reset HEAD file1.txt
# Recommit
git commit -a -c ORIG_HEAD
```

or if you merely need to add more files or changes to the previous commit (not removing any files from the set)

```
git commit -a --amend
```

Revert

Oops. You realize you made a bad commit yesterday, but it's already been shared across other Git repositories.

Revert to a previous commit (via a new commit), but don't commit it until you review what's going to be in this commit. This is similar to a banking transaction in which you fix the old error by a new journal entry, not deleting the mistaken one. This leaves an audit trail.

```
git revert -n
```

Revert is the better choice in the situation where the previous changes have been pushed to another repository. If they have, `amend` is a bad idea; you are re-writing history. Your remote repository may subsequently deny your amended push if the original was previously pushed.

Resetting to a Previous State

There are several modes of operation for reset for which the difference is quite striking. These include soft, hard, mixed, merge, and keep.

- Soft
- Does not touch the index or working directory
- Hard
- Discards index and working directory changes
- Mixed
- Changes the index, but not the working directory (the default behavior)
- Merge
- Resets index, updates working directory (clean files), keeps untracked changes.
- Keep
- 2 way merge

Discard any uncommitted changes

```
git reset --hard
```

Revert history to two commits ago (discarding them)

```
git reset --hard HEAD^^
```

Move staged files back to unstaged state

```
git reset
```

Restore a single file to its last committed state

```
git checkout -- SOMEFILE
```

Restore a file to a past specified state

```
git checkout TREEISH -- SOMEFILE
```

Cleaning

Remove untracked files (generally temp, generated, compiled)

```
git clean
```

But it will complain that you aren't asking it to actually clean (force, `-f`)

```
git clean -f
```

Or preview (`-n` or `--dry-run`) what will be done

```
git clean -n
git clean --dry-run
```

Or clean directories (`-d`) too (not just files)

```
git clean -f -d
```

Reflog

The `reflog` is the transactional journal of what's been performed on your repository, including `reset`s, `commit`s, `merge`s and `rebase`s. Can be used to identify a treeish to `reset` to (a known `HEAD@{X}` point).

```
git reflog
817c5e7 HEAD@{0}: commit (initial): Adding files
```

```
git reset HEAD@{0}
```

or

```
git merge HEAD@{0}
```

Reflog entries are kept for 90 days. Orphaned entries are kept for 30 days.