Git Cheat Sheet

August 2, 2017

1 Basics

Creates a new Git repository:

```
$ git init
```

Inspects the contents of the working directory and staging area:

```
$ git status
```

Adds files from the working directory to the staging area:

```
$ git add
```

Shows the difference between the working directory and the staging area:

```
$ git diff
```

Permanently stores file changes from the staging area in the repository:

```
$ git commit
```

Shows a list of all previous commits:

```
$ git log
```

1.1 Undo a commit and redo

```
$ git commit -m "Something misguided" (1)
$ git reset HEAD~                     (2)
<< edit files as necessary >>         (3)
$ git add ...                         (4)
$ git commit -c ORIG_HEAD             (5)
```

1. This is what you want to undo.
2. This leaves your working tree unchanged but undoes the commit and leaves the changes you committed unstaged (so they'll appear as "Changes not staged for commit" in git status and you'll need to add them again before committing).1
3. Make corrections to working tree files.
4. *git add* anything that you want to include in your new commit.
5. Commit the changes, reusing the old commit message.2

1 If you only want to add more changes to the previous commit, or change the commit message[1], you could use git reset –soft HEAD instead, which is like git reset HEAD but leaves your existing changes staged.

2 reset copied the old head to .git/ORIG_HEAD; commit with -c ORIG_HEAD will open an editor, which initially contains the log message from the old commit and allows you to edit it. If you do not need to edit the message, you could use the -C option. `https://stackoverflow.com/questions/927358/how-to-undo-the-last-commits-in-git`

2 Backtracking

```
$ git show HEAD
```

Discards changes in the working directory:

```
$ git checkout HEAD filename
```

Unstages file changes in the staging area:

```
$ git reset HEAD filename
```

Removes staged and working directory changes:

```
$ git reset --hard
```

Can be used to reset to a previous commit in your commit history:

```
$ git reset SHA
```

3 Branching

Lists all a Git project's branches:

```
$ git branch
```

Creates a new branch:

[1]Note, however, that you don't need to reset to an earlier commit if you just made a mistake in your commit message. The easier option is to git reset (to unstage any changes you've made since) and then git commit –amend, which will open your default commit message editor pre-populated with the last commit message.

```
$ git branch branch_name
```

Used to switch from one branch to another:

```
$ git checkout branch_name
```

Used to join file changes from one branch to another:

```
$ git merge branch_name
```

Deletes the branch specified:

```
$ git branch -d branch_name
```

4 GIT Teamwork

Creates a local copy of a remote:

```
$ git clone remote_location clone_name
```

Lists a Git project's remotes:

```
$ git remote -v
```

Fetches work from the remote into the local copy:

```
$ git fetch
```

Merges origin/master into your local branch:

```
$ git merge origin/master
```

Pushes a local branch to the origin remote:

```
$ git push origin <branch_name>
```

5 Stashing

Saves changes internally and cleans working directory:

```
$ git stash
```

Outputs list of stashes:

```
$ git stash list
```

Deletes all git stashes at once:

```
$ git stash clear
```

Re-applies stashed changes (most recent snapshot):

```
$ git stash pop
```