Most common git screwups/questions and solutions

This is a fork of Nava Whiteford's article Most common git screwups/questions and solutions. I've made this fork as a base to add my own notes. Home of this fork is my git-info-collection.

The original author was looking to learn a bit more about the parts of git he hasn't ventured into yet. What better way that looking the most common ways people screw them up and how to fix the resulting problems! Here's a short list, compiled from his own experience and issues he's come across on the Internet.

I wrote the wrong thing in a commit message

If the commit hasn't been push you can do the following, which will allow you to edit the message on the most recent commit:

```
git commit --amend
```

How do I add a forgotten file to an commit?

Sooner or later you will commit to fast and forgott a file. To avoid a cluttered history, it is possible to add the file to the previous commit object.

```
git commit -m 'committed to fast and forgotten a file'
git add forgotten_file
git commit --amend
```

I've worked on the wrong branch. How to move unstaged files?

In the case you have changed some files and forgotten to switch to the right branch, it is possible to *move the changes out of the way* by using the stash. After stashing the changes, the workspace is on par with the last commit of the branch.

Note: untracked files are still there and can be ignored.

```
$ git stash
$ git checkout correct-branch
$ git stash pop
```

After the `stash pop`, all previously made changes available and ready for a commit.

I've worked on the wrong branch. How to branch unstaged files?

In the case you have changed some files and forgotten to create a new branch, it is possible to *move the changes out of the way* by using the stash. After stashing the changes, create a new branch and apply the stashed changes.

Note: untracked files are still there.

```
$ git stash
$ git checkout -b new-branch
$ git stash pop
```

After the `stash pop`, all previously made changes available and ready for a commit.

How to unstage a staged file?

```
$ git add .
$ git status
...
$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M   CONTRIBUTING.md
$ git status
```

How can I undo the last commit?

You can use git reset e.g.:

```
git reset --hard HEAD~1
```

`HEAD\~1` means HEAD-1 commit. It should be noted that this is the nuclear option, and any changes you made will be **discarded**. If you want to keep your changes in the working tree use:

```
git reset --soft HEAD~1
```

If you've already published your commits, you should use revert. This is create new commits undoing the last change:

```
git revert HEAD~1..HEAD
git revert commitid
```

Note: the revert command is a safer alternative to the reset command, since the commit objects are still available!

Rename a tag

To rename a tag, a new tag must be added and the old one must be deleted.

```
git tag new-name old-name
git tag -d old-name
```

If the tag is pushed to a remote repository, you have to clear the old name there too. The colon in the push command removes the tag from the remote repository. If you don't do this, git will create the old tag on your machine when you pull.

```
git push origin :old-name
git push --tags
```

Note: On every client with clones of the repository, users must remove the deleted tag too. This could be easily done with the following command:

```
git pull --prune --tags
```

Delete a Git branch remotely

Git knows two ways to delete a branch inside the remote repository only. The shorter version is available since git v1.5.0.

```
git push origin --delete branchname
git push origin :branchname
```

What are the differences between `git pull` and `git fetch`?

git pull, is git fetch followed by git merge. git fetch gets the remote changes, they get kept under `refs/remotes/`

How do I undo a `git add` before committing

You did a `git add filename` accidentally and want to undo it before committing your change. You can simply do:

```
git reset filename
```

To unstage your changes to that file.

How do I deal with merge conflicts

Use `git mergetool` which gives a handy interface for solving merge conflicts.

Remove all local untracked files (and directories) from your local clone

Careful! You might want to take a backup before doing this:

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

Clone all remote branches

You probably already have, they're just hiding! Use the following to see all the branches:

```
git branch -a
```

You can then use `git checkout origin/branchname` to take a look at the branch. Or `git checkout -b branchname origin/branchname`. To create a local tracking branch.

Rename local branch?

```
git branch -m oldname newname
```

Revert to a previous Git commit

You can use reset as above to revert to a previous commit, this assumes you mean go back to where you were before permanently rather than just take a look (for that you want to checkout an old version). The commit ID, should be as shown in the output of `git log`.

```
git reset --hard commitid
```

Again this will discard all changes in your working tree, so make sure this is really what you want to do! Or look at using –soft rather than –hard.

Remove a git submodule

Creating a submodule is pretty straight-forward, but deleting them less so the commands you need are:

```
git submodule deinit submodulename
git rm submodulename
git rm --cached submodulename
rm -rf .git/modules/submodulename
```

Over-write local files when doing a git pull

Git reset is your friend again:

```
git fetch --all
git reset --hard origin/master
```

How can I add an empty directory to my repository?

You can't! git doesn't support this, but there's a hack. You can create a `.gitignore` file in the directory with the following contents:

```
# Ignore everything in this directory
*
# Except this file
!.gitignore
```

I don't believe it actually matters what's in the .gitignore (and this .gitignore will ignore all files).

git export, exporting the source code as with `svn export`

Use git archive e.g:

```
git archive --format zip --output /full/path/to/zipfile.zip master
```

Discard all my unchecked in changes

```
git checkout -- .
```

Discard the changes made to a single file

```
git checkout -- filename
```

Create a new remote branch from the current local one

```
git config --global push.default current
git push -u
```

Restore a deleted file

First find the commit when the file last existed:

```
git rev-list -n 1 HEAD -- filename
```

Then checkout that file

```
git checkout deletingcommitid^ -- filename
```

Revert a single file to a specific previous commit

Similar to the above, but a bit simpler:

```
git checkout commitid filename
```

Make git ignore permissions/filemode changes

```
git config core.fileMode false
```

Get git to remember my password when checking out with https

It'll only remember your password for 15mins:

```
git config --global credential.helper cache
```

You can make it remember your password longer with:

```
git config --global credential.helper "cache --timeout=XXXX"
```

Take a quick look at an old revision of a single file

```
git show commitid:filename
```

Tips find in the comments

you want to undo the last rebase you completed

```
git reset --soft ORIG_HEAD
```

(or –hard if you're confident/adventurous)

I lost my commit somehow (or made a bad amend)

```
git reflog
```

then use one of the commands above that take a commitid.