# Common Commands

git branch # To remind myself what features I'm working on.

git checkout <name\_of\_branch> # To switch to whatever I want to work on.

git checkout -b <name\_of\_new\_feature> # To start work on a new branch

git add <name\_of\_file> # To add it to the list of tracked files.

git commit -m <commit\_message> # To checkpoint my work.

git merge <name\_of\_branch> # To integrate changes back to trunk.

git branch -d <name\_of\_branch> # To delete a branch after it has been merged.

git fetch origin  
git reset --hard origin/master # To set your branch to exactly match the remote branch

# Example Workflows

[Great article](http://nvie.com/posts/a-successful-git-branching-model/) on how we should operate.

You might benefit from the workflow Scott Chacon describes in [Pro Git](http://progit.org/). In this workflow, you have two branches that always exist, *master* and *develop*. *master* represents the most stable version of your project and you only ever deploy to production from this branch. *develop* contains changes that are in progress and may not necessarily be ready for production.

From the *develop* branch, you create topic branches to work on individual features and fixes. Once your feature/fix is ready to go, you merge it into *develop*, at which point you can test how it interacts with other topic branches that your coworkers have merged in. Once *develop* is in a stable state, merge it into *master*. It should always be safe to deploy to production from *master*.

Scott describes these long-running branches as "silos" of code, where code in a less stable branch will eventually "graduate" to one considered more stable after testing and general approval by your team.

Step by step, your workflow under this model might look like this:

1. You need to fix a bug.
2. Create a branch called *myfix* that is based on the *develop* branch.
3. Work on the bug in this topic branch until it is fixed.
4. Merge *myfix* into *develop*. Run tests.
5. You discover your fix conflicts with another topic branch *hisfix* that your coworker merged into*develop* while you were working on your fix.
6. Make more changes in the *myfix* branch to deal with these conflicts.
7. Merge *myfix* into *develop* and run tests again.
8. Everything works fine. Merge *develop* into *master*.
9. Deploy to production from *master* any time, because you know it's stable.

For more details on this workflow, check out the [Branching Workflows](http://progit.org/book/ch3-4.html) chapter in Pro Git.

# When to Branch

* You are about to make a major or disruptive change
* You are about to make some changes that might not be used
* You want to experiment on something that you are not sure it will work
* When you are told to branch out, others might have something they need to do in master