***Context:*** *As I was learning to use git with Angular repositories, I noticed that everyone on the team seemed to do things a little differently. This is my attempt to capture a “recipe” for new git users working on Angular. It is not gospel. If you have your own recipe that’s working, no need to change what you’re doing. I’m sharing this in case it’s useful to others. -naomi*

## Things to never ever do (or at least try to avoid)

*Especially if you have commit access to an Angular repository*

* don't make changes to master, always start a new branch.
* don’t merge. It messes up the commit history.
* don’t pull upstream without a rebase (see above). git fetch and then rebase instead (or equivalently, git pull upstream master --rebase).
* don’t use git commit -a. You could silently commit something regrettable. Use -p instead.

**Setting up your repository for work on angular.dart**

1. [Install and configure git](http://git-scm.com/book/en/Getting-Started-Installing-Git) on your local machine. (I found the Mac GUI client a waste of time, YMMV. If you are comfy with the command line, you are probably going to want to skip the GUI.
2. Install node.js (you’ll need this for npm)
3. Create a fork of angular/angular.dart.git (or angularjs, or whichever)
4. Clone your fork: git clone [https://github.com/*username*/angular.dart.git](https://github.com/username/angular.dart.git)
5. In your local repository, set up a remote for upstream:  
   $ git remote add upstream git@github.com:angular/angular.dart.git
6. Create ssh keys: ssh-keygen -t rsa -C "[*youremail@somedomain.com*](mailto:youremail@somedomain.com)"
7. ssh-add ~/.ssh/id\_rsa
8. pbcopy < ~/.ssh/id\_rsa.pub
9. Go paste your keys into Github, under SSH Keys for your account.
10. Install Dart packages: pub get

## A git recipe for Angular repositories

For a change with a specific scope, e.g. updates to the doc generation script:

1. Set up a git client where origin is a fork of the repository (e.g. naomiblack/angular.dart), and upstream is the real deal (e.g. angular/angular.dart)
2. Before creating a branch to work in, first make sure you’re on your local master branch

git checkout master

1. Next, make sure that master is in sync with the upstream source of truth:

git fetch upstream

git merge --ff-only upstream/master

Or, if you prefer

git pull upstream master --rebase

*Note: if there’s a conflicting commit in the history of your master branch, you can* ***destroy your branch*** *and replace it with a fresh copy using the command* git checkout -B master upstream/master.

1. Now create a new branch

git checkout -b *doc-script-changes*

1. On the new branch, make edits to the files.   
   *Note: Time passes, stuff changes in the upstream repo....*
2. Commit your changes with git commit -p, or git commit and individually add files with git add Set a commit message that conforms to the contribute guidelines.
3. To sync your changes with what's upstream, git fetch.
4. To make sure your commit goes in at the top of everything else on the upstream repo, rebase:

git rebase upstream/master

If there are conflicts, open the file and look for the diff markers, resolve, and continue.

1. Send your changes to your forked copy of the repo in the appropriate branch:

git push -f origin *doc-script-changes*.

1. In the web client, go to your fork of the repo, and initiate a pull request by pushing the Pull Request button. Submit the pull request!
2. While the pull request is out for consideration:
   * Any new changes unrelated to this one should be on a brand new branch (git checkout -b some-new-thing). Don't forget to check out the master branch first, otherwise you'll branch off of the current PR branch
   * If you want to make changes to your earlier commit in response to comments on the pull request, you change back to the branch that you submitted it from (git checkout doc-script-changes), make any changes, then commit and push them (steps 6-9). These get automatically added to your pull request since they're in the same branch.
     1. If your changes are small fixes, they should not be a new commit. Instead, use git add and then git commit --amend to fix up your original commit.
   * If you decide to abandon a pull request, you can CLOSE the issue it created and ignore it.
   * If the pull request is good, there's nothing else for you to do, besides wait for someone to accept it.
3. Once the pull request is accepted (or closed) you can delete your branch from the client. Or, you can wait until you have collected a bunch of them and delete all of the obsolete ones in one go.

Also keep in mind that `git branch -D my-branch` deletes branches only locally, to delete them from the remote repo you have to do `git push origin :my-branch`

**Moving a change between branches**

Sometimes you make a change on the wrong branch. You can move it to the right branch with git stash. From the branch where you made the changes:

git stash

git checkout *branch-you-want-it-on*

git stash pop

## Some useful tips and tricks

### Modify your github client to pull in all the PRs and work on them

You can set up your github client to make it easy to work with submitted pull requests.

Edit your .git/config and add the two fetch lines shown below under remote “upstream”:

[remote "upstream"]

url = git@github.com:angular/angular.git

fetch = +refs/heads/\*:refs/remotes/upstream/\*

fetch = +refs/pull/\*/head:refs/remotes/upstream/pr/\*

Now, when you fetch upstream, you’ll get references to a bunch of PRs.

* check one out with git checkout upstream/pr/3328
* from detached head mode, create a branch with git branch BRANCHNAME
* fetch upstream, rebase against master, test things out.
* push to your branch to verify that CI tests are green for these changes.

When you’re ready to commit the PR you’ve validated:

* run gulp enforce-format and gulp lint to verify that you haven’t introduced any syntax/style issues.
* look at git log and double-check git status to be sure everything is in the top commit(s)
* when everything is green and looks legit:   
  git push upstream BRANCHNAME:master

### Pull in a specific PR for testing

*Thanks to Pete Bacon Darwin for this one*

Suppose you want to pull in someone else’s PR (#1234) for testing, on angular.js. Try:

curl https://github.com/angular/angular.js/pull/1234.patch | git am -3

This will download it into the patch and merge it into the current branch for you.

### Drop commits from a PR

In the branch where you created the commits you want to drop:

**$** git rebase -i upstream/master

This opens an editor. Delete the lines you don’t want. Then:

**$** git push -f origin *branchname*

### Pretty colors and branch name in your command line prompt

Add something like this to your .bashrc

source ~/.bash\_colors

# Git completion

source $PATH\_TO\_GIT\_CORE/git-completion.bash

source $PATH\_TO\_GIT\_CORE/git-prompt.sh

# Prompt

export PS1="\[$Green\]\t\[$Red\]:\[$Yellow\]\W\[\033[m\]\[$Blue\]\$(\_\_git\_ps1)\[$White\]\$ "

And create a file .bash\_colors. [Here’s one you could copy.](https://gist.github.com/jeffrafter/4708244)

## For committers to angular.dart

If you have commit access on angular.dart, and it is time to merge in a change from your branch, here’s one way to do it. Note that this process is a little different because we use a presubmit queue to test before submitting changes to master.

1. Start in a local branch *branchname* that has the change that you want to add.
2. If there’s a Pull Request, use git commit -c HEAD --amend to modify the commit message. Add a newline for space, and then reference the Pull Request issue # as “Closes #123”
3. Check that you’re seeing the right SHA with git log -n1.
4. Make sure you’re on top of the latest changes from upstream:

git fetch upstream

git rebase upstream/master

1. Now push your changes upstream. Important! Use the prefix *presubmit-* in naming the upstream branch so that it is routed through Travis CI.

git push upstream *branchname*:presubmit-*myname*

1. Look up the builds for angular.dart on Travis (<https://travis-ci.org/angular/angular.dart/builds>). You should see your commit to presubmit-*myname* at the top of the list, just after a completed build against master. If you don’t, alas and bad news, someone else was pushing a change at the same time you’ll have to check again later to be sure that your commit made it in.
2. Wait for Travis to complete running the presubmit tests. If everything passes, the change will be pushed for you, this time against master. When that completes, rejoice, the change is now part of angular.dart.

### How to pull a PR into your local repository and submit it

First, you’ll need to make a change to the .git/config file for the repository. Add this line under remote “orgin”

fetch = +refs/pull/\*/head:refs/remotes/upstream/pr/\*

Like so...

[remote "origin"]

url = git@github.com:angular/angular.dart.tutorial.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

fetch = +refs/pull/\*/head:refs/remotes/upstream/pr/\*

Once that’s done, you can pull individual pull requests in as follows:

1. First, make sure you’re current with what’s in the repository:

git fetch

1. Then check out the PR you want. For example, PR 123, and create a fresh branch to work in:

git checkout upstream/pr/123

git checkout -b *mybranchname*

1. Make some changes, and rebase on top of master: git rebase master
2. Then commit your change (git commit)
3. And push it back to github. E.g. with and without presubmit queues...

git push upstream *mybranchname*:presubmit-*myname*

or

git push upstream *mybranchname*:master

### How to resubmit your commit after a collision during Travis CI tests

If your change collided with someone else’s change on its way through CI tests, you’ll have to fix the order of the commits and submit it again:

1. Get the latest changes onto your branch:

git fetch upstream

git rebase upstream/master

1. Want to double-check what’s in that commit you’re about to push? git show.
2. Let’s double check that the top SHA from the main angular.dart repository is indeed in our log: git log | grep *SHA-of-top-commit-in-github*
3. If it is, then we can try to push the change to the presubmit queue again:

git push upstream HEAD:presubmit-*myname* -f

Note the “-f”, this is needed because you’re taking a commit that’s out of stream for master and adding it back in. If you forget the -f, you’ll get an error:

! [rejected] HEAD -> presubmit-naomi (non-fast-forward)

error: failed to push some refs to '[git@github.com](mailto:git@github.com):angular/angular.dart.git'

### How to get a change from someone’s repository into your repo before it’s pushed to master

1. Define a remote for their github repo, e.g.   
   git remote add *misko* https://github.com/*mhevery*/*angular.dart.git*
2. Now fetch their changes and rebase on top of the branch they have that change in:  
   git fetch *misko*  
   git rebase *misko*/*misko\_branchname*
3. If there are collisions, these files are removed from your git commit. Hunt them down by searching for seven > characters ‘>>>>>>>’ in your project. Resolve any conflicts in your editor. git status will also show the affected files (in red, since they’re not part of a commit)
4. Add the files back into tracking with git add .
5. Carry on with the rebase: git rebase --continue

### How to get rid of garbage files that shouldn’t be in git

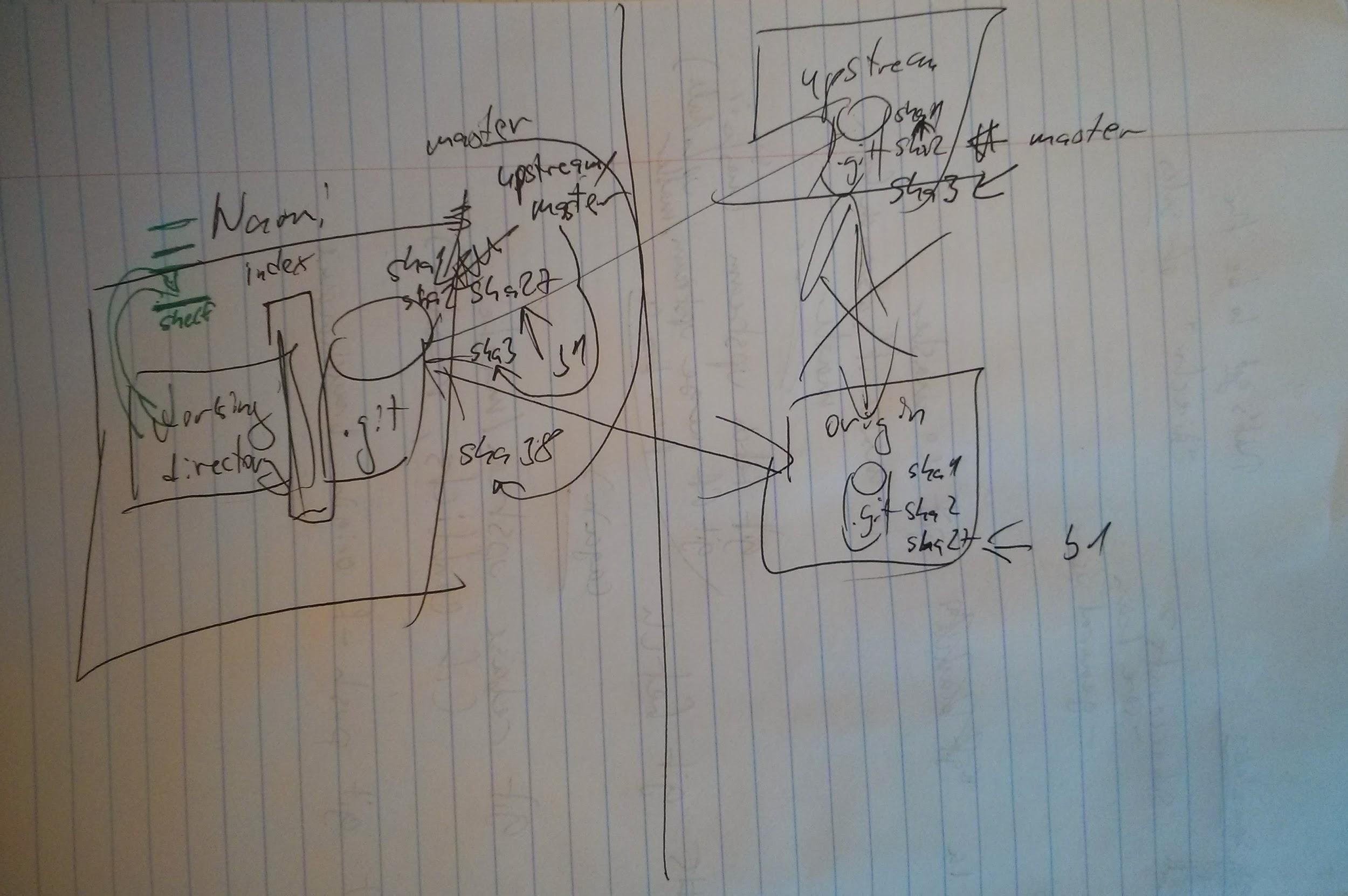
Sometimes my Mac makes .DS\_Store files in my git directories and I want to get rid of them:

**$** git status *check that you don’t have anything important that should be added first!*

**$** git clean . -f

Caution -- if you have any new files that aren’t under git control, this will remove all of them.

## Igor explains git for you with a simple pen and ink drawing



*There. That should be clear now, right?*