Using Git in a team: a cheatsheet

December 2016

I don't go anywhere near the majority of what Git can do. Nevertheless, over the years I've arrived at a workflow which seems to have stuck. It's by no means revolutionary, but it gets the job done.

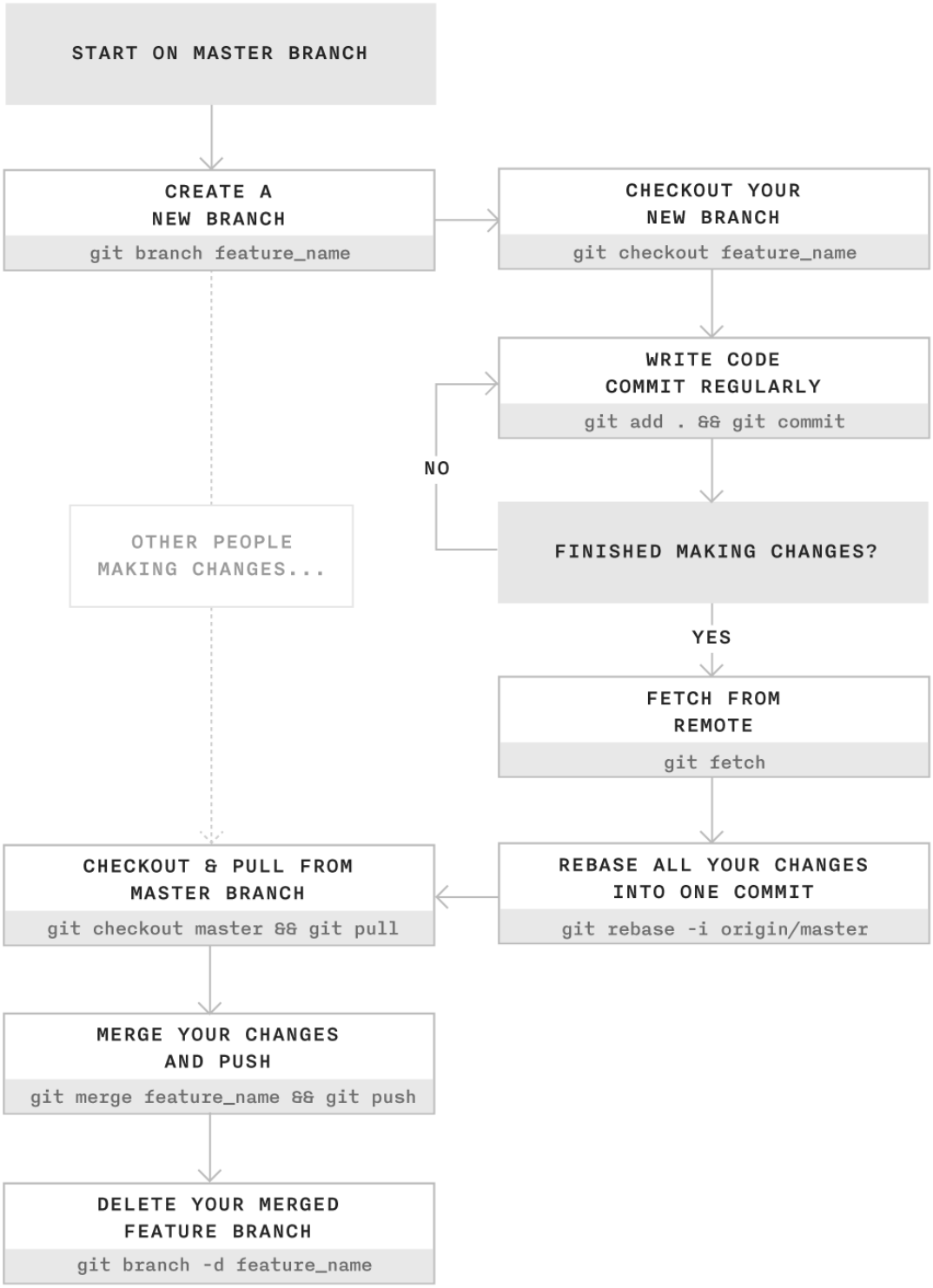
About half the time I use Git on projects only I will ever see, and the rest of the time I work collaboratively with a handful of people in my team. This workflow is heavily influenced by the concepts covered in the Git course on Upcase. If you're looking to improve your skills as a developer, check it out.

This is what the workflow looks like:

> Check out a new branch
>
> Work on that branch, making several commits
>
> Merge changes back into the master branch in a single, curated commit

The idea is that a single large commit keeps the master branch neater than lots of small commits. Let's go through each step in more detail.

The cheatsheet



Step 1: Create a new branch to work on

```
# Create a new feature branch
```

```
git branch jc_new_feature
# Checkout your new feature branch
git checkout jc_new_feature
```

It's good practice to start feature branch names with your initials, then a description of the feature itself (e.g. `jc_feature_name`). This helps others working on a project to see who's been doing what.

Step 2: Write code, commit regularly

```
# Add all files to the stage
git add .
# Commit files
git commit -m "Description of this commit"
# Optional (but recommended) push local branch to remote
git push origin jc_feature_name
```

Commit regularly. I've never regretted making a commit, but have regretted not making one. You'll have a chance to change your commit messages before merging back into master.

Step 3: Fetch when you're done

When you're ready to merge your features back into the master branch, run `git fetch`.

```
git fetch
```

Fetching makes sure you're up to date when merging changes back into master. This doesn't actually merge the code with the code on your machine (git pull does that), but rather updates references to any remote changes which may have happened while you've been working locally. It's the groundwork for the next stage.

Step 4: Squash your commits and get ready to merge

Next, you'll rebase your changes into the master branch. This effectively condenses down all your feature branch commits (`jc_feature_name`) into a single commit. We'll merge this single commit back into the `master` branch.

```
# Open the interactive rebase tool
git rebase -i origin/master
```

This will open an interactive rebase tool, which shows all the commits you've made on your branch. You'll then "squash" all your changes into one commit.

```
# Interactive rebase tool
pick bf27dcd Initial change
pick cc570e1 Second change
pick ca0ec55 Third change
```

Squash your commits by replacing pick with s for all but the top commit. s is shorthand for squash - imagine all the changes you've made being "squashed" up into the top commit.

```
# 'Squashing' the second and third commit into the first
pick bf27dcd Initial change
s cc570e1 Second change
s ca0ec55 Third change
```

When you close this window you'll see all your existing commits, which you can edit to be more concise. Exit the rebase tool and you're ready to merge.

Step 5: Merge your changes

Prepare to merge your changes by switching to the master branch. Remember that you're merging changes *into* the branch you're currently

on.

```
# Checkout the master branch
git checkout master
# Merge jc_feature_name INTO master
git merge jc_feature_name
```

The changes from the `jc_feature_name` branch are now merged into your `master` branch. Happy days. Now's a good time to push your changes to the remote master branch.

```
# Push your local master branch to remote
git push origin master
```

Step 6: Cleanup

With your changes merged into the master branch, you can safely delete your feature branches.

```
# Delete remote feature branch (the colon is important!)
git push origin :jc_feature_name
# Delete local branch
git branch -d jc_feature_name
```

And with that, you're done.

Making it work

It's easy to feel insecure about using Git. The spectre of lost work looms large. The key to making this system work is to go through the process a lot, regularly branching and merging. This gets you comfortable with the workflow and has the added bonus of ensuring one branch never

gets too far out of line with another.

Good evening. I'm James.

If you'd like to join fellow bootstrappers getting my twice-monthly newsletter about building and growing self-funded SaaS products, enter your email below:

Enter your email

Subscribe