



A git workflow for DMWM

Andrew Melo

Several better written articles exist. Spend some time to understand

<http://git-scm.com/book/en/Git-Internals-Git-Objects>

If you want to get deeper idea of what's happening.

- The main branch is called *master*, which is roughly *trunk* in SVN
- Each self-contained bit of work (usually answering a ticket) belongs on a *feature branch* started by a developer writing the code
 - Name the branch something descriptive
- Developers write code in feature branches
- Maintainers push them into *master*

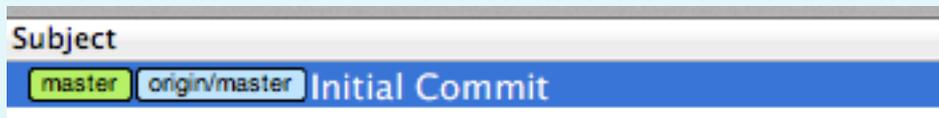


Tutorial Setup



VANDERBILT

```
$ mkdir git-tutorial
$ git init local-repo
Initialized empty Git repository in ./local-repo/.git/
$ git init --bare remote-repo
Initialized empty Git repository in ./remote-repo/.git/
$ cd local-repo/
$ git remote add origin ../remote-repo/
$ echo "Initial Commit" > README
$ git add README
$ git commit -am "Initial Commit"
$ git log --pretty=oneline
56f19de1bede08cf2c0b75afbf801745f45d6d45 Initial Commit
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 229 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
To ../remote-repo/
 * [new branch]    master -> master
```





Tutorial: Make a Feature Branch



VANDERBILT

```

$ git checkout master # Choose where you want to fork off of
$ git checkout -b "feature/tutorial-branch1" # Make a new branch and check it out
Switched to a new branch 'feature/tutorial-branch1'
# HACK HACK HACK
$ echo "READMEs are for chumps" > README
$ git commit -am "Tutorial Commit"
[feature/tutorial-branch1 1613f75] Tutorial Commit
1 file changed, 1 insertion(+), 1 deletion(-)
# Done hacking, now let's send the branch to GH where it can be pulled
$ git push origin feature/tutorial-branch1
Counting objects: 5, done.
Writing objects: 100% (3/3), 268 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
To ../remote-repo/
* [new branch] feature/tutorial-branch1 -> feature/tutorial-branch1

```

Subject	Author
 feature/tutorial-branch1 origin/feature/tutorial-branch1 Tutorial...	Andrew Melo
 master origin/master Initial Commit	Andrew Melo



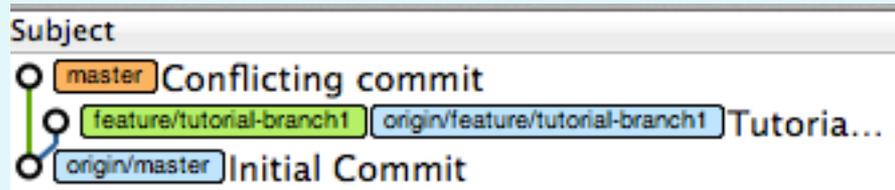
Tutorial: Syncing with upstream



VANDERBILT

What happens if the remote branch advances?

```
$ git checkout master
$ echo "Conflict" > README
$ git commit -am "Conflicting commit"
```



Let's see what happens if we try to merge master into the feature branch

NOTE: "git pull" is the same as "git fetch ; git merge"

```
$ git merge --no-commit --no-ff master # basically means --dry-run
```

Auto-merging README

CONFLICT (content): Merge conflict in README

Automatic merge failed; fix conflicts and then commit the result.

Crap, things broke. Git will put the conflicts side-by-side in the files mentioned above

```
$ cat README
```

```
<<<<<< HEAD # Everything from here to ===== is from the feature branch
```

READMEs are for chumps

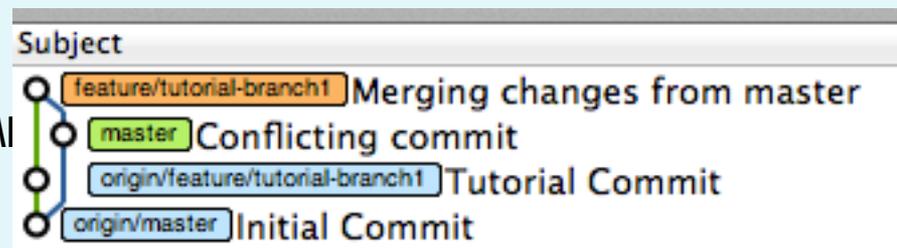
```
===== # Everything from here to >>>> master is from the master branch
```

Conflict

```
>>>>>> master
```

```
$ echo 'READMEs are for chumps (merged)' > README
```

```
$ git commit -am 'Merging changes from master'
```

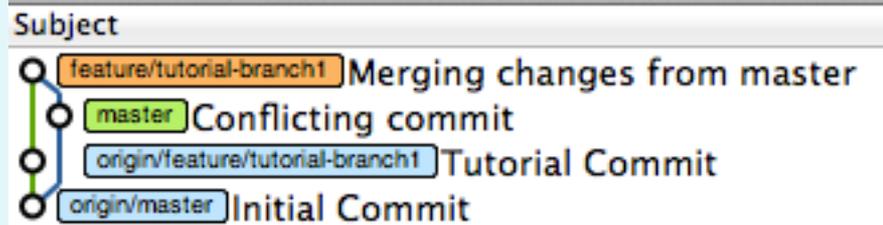




Tutorial: A better way to sync w/upstream



Doing “git merge” leaves us with a “merging changes from master” commit for every time we try to sync with master. This is really noisy for maintainers and anyone else trying to read the history:



PerilousApricot and others added some commits

4 months ago

- PerilousAp... removing other file [2e85a5c](#)
 - PerilousAp... Squashed commits that update the new StageIn code [058a771](#)
 - PerilousAp... commit got jumbled up somehow [a673468](#)
 - PerilousAp... Merge remote-tracking branch 'upstream/master' into fixstageout-squas... [fabb57d](#)
- ...hed2

The “Merge remote-tracking” commit is truly bad, it has completely unrelated diffs from Seangchan (his code was changes for monitoring, mine was for StageOut, none of the modified files were the same). If you’ve ever seen a pull request with 100s of additions/deletions and tons of modified files, you’ve hit the same problem.



Tutorial: git rebase



VANDERBILT

Because of the issues in the previous slide, it's a better idea to do “git pull --rebase” to pull down commits from upstream. This rewrites the commits so instead of the upstream commits going AFTER your commits, they show up BEFORE them, chronologically. See <http://linux.die.net/man/1/git-rebase> for the gory details.

NOTE: NEVER (*) REBASE BRANCHES THAT ANYONE ELSE HAS PULLED LOCALLY

* okay, there are exceptions to that rule



Tutorial: Pulling with rebase



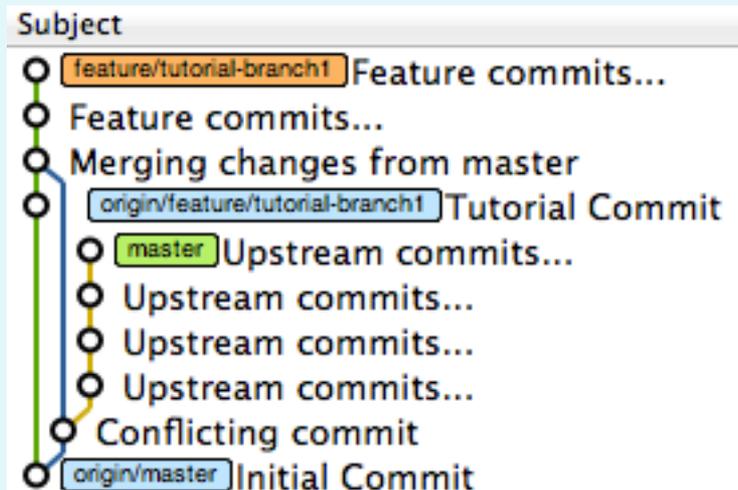
VANDERBILT

First, let's make some fake commits on master:

```
$ git checkout master
Switched to branch 'master'
$ echo `date` > README ; git commit -am "Upstream commits..."
[master 10ee7bb] Upstream commits...
 1 file changed, 1 insertion(+), 1 deletion(-)
$ echo `date` > README ; git commit -am "Upstream commits..."
[master e4980d1] Upstream commits...
 1 file changed, 1 insertion(+), 1 deletion(-)
$ echo `date` > README ; git commit -am "Upstream commits..."
[master 3b58f0b] Upstream commits...
 1 file changed, 1 insertion(+), 1 deletion(-)
$ echo `date` > README ; git commit -am "Upstream commits..."
[master c8abc0d] Upstream commits...
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Do the same on the feature branch...

We're left with this tree.
 Note master and our feature branch have diverged, and we'd like to update the feature branch with the commits from master

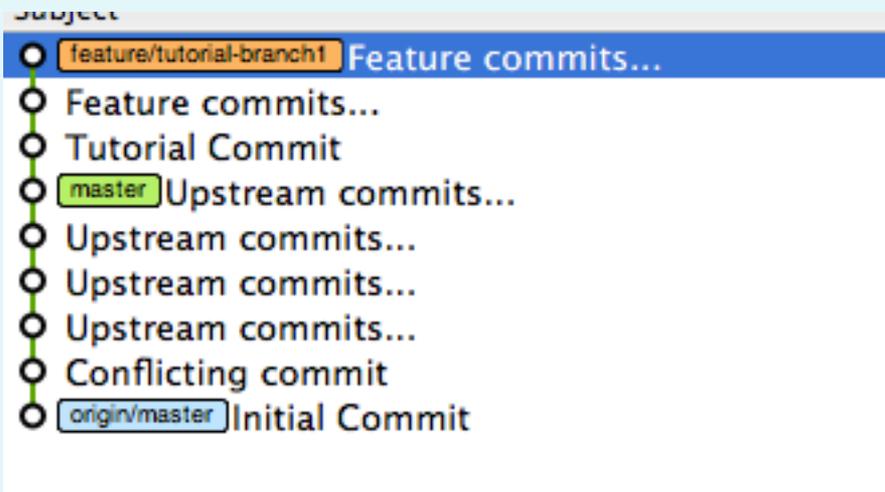


```
Andrew-Melos-MacBook-Pro:local-repo meloam$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Tutorial Commit
Using index info to reconstruct a base tree...
Falling back to patching base and 3-way merge...
Auto-merging README
CONFLICT (content): Merge conflict in README
Failed to merge in the changes.
Patch failed at 0001 Tutorial Commit
```

Fixing a conflicted rebase is like fixing a conflicted merge.
 Does anyone want an explanation for that?

When you have resolved this problem run "git rebase --continue".
 If you would prefer to skip this patch, instead run "git rebase --skip".
 To check out the original branch and stop rebasing run "git rebase --abort".

The end result is that instead of getting a spurious merge commit with all the changes from the master showing up in your branch and polluting history/adding a huge diff, you get:



The moral of the story is: When trying to merely sync a branch with commits made to another branch, and your local branch is unshared, use a `--rebase` instead of a merge

To keep the history clean, you can go one step further and “squash” multiple commits into one:

```
$ git rebase -i HEAD~3 # says, let's rewrite the last 3 commits
```

```
# your editor will show up with this:
```

```
pick 736f962 Tutorial Commit
```

```
pick 123e2e4 Feature commits...
```

```
pick 17ad036 Feature commits...
```

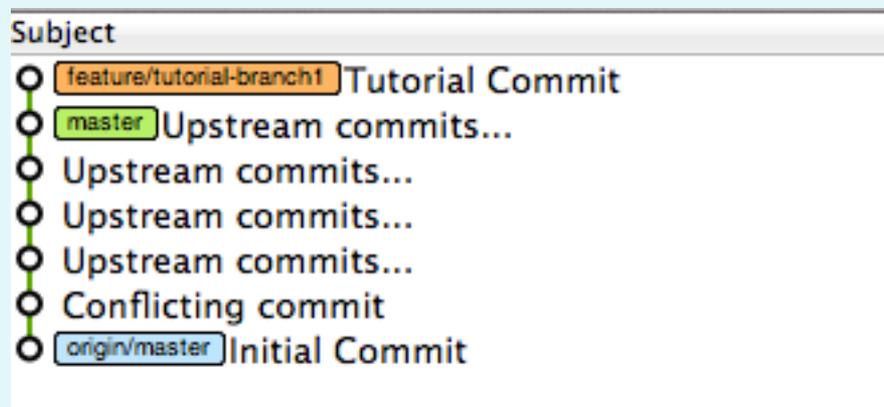
```
# Change it to say
```

```
pick 736f962 Tutorial Commit
```

```
squash 123e2e4 Feature commits...
```

```
squash 17ad036 Feature commits...
```

```
# and you end up with:
```



This is nice because it further keeps the history clean. If you find yourself with a lot of “checkpoint” commits, it’s helpful to tidy things up before you show others

- If anyone has specific questions about other topics, feel free to ask them. I think I covered the important ones for day-to-day developing.
- Finally, Github's UI is designed to work well with a certain commit message format: 1 subject line (50 char max), 1 blank line, N body lines (72 char max, excluding things like error messages). Please use that.

- Use the merge button on GitHub.
- If you don't use the merge button, be sure that you're not polluting the history with junk, while being double sure you're not rewriting history