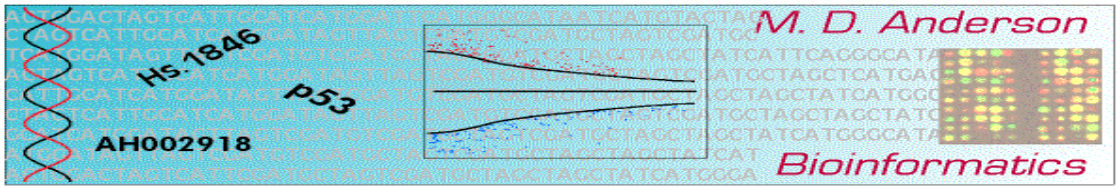
Git/GitHub 2: Branching and Merging

Keith A. Baggerly
Bioinformatics and Computational Biology
UT M. D. Anderson Cancer Center
kabagg@mdanderson.org

SISBID, July 17, 2018



Branching

So far, our Git and GitHub repositories have grown “linearly” - modifications have followed one another in strict sequence.

However, this doesn’t exploit the full power of Git.
It doesn’t emphasize that several people can be working to extend the same repo concurrently.

We can have multiple development “branches” emerging from a single commit, being used for modifications, and then being “merged” back into the main master branch.

RStudio helps with this, but some branch manipulations (merging and pruning) require command line interaction.

An Example

The first example that comes to mind involves tweaking something that's stable, when you're not completely sure if this will help or hurt.

A common approach:
treat the main master branch as a reference
use a development branch for modification and testing
when things are working,
merge the changes back into master and
prune the development branch.

I'm going to show the basics with local branches first.
After that's running smoothly we'll expand to GitHub.

Our First Branch

In the upper right of RStudio's Git pane, there's a "new branch" button (a diamond with squares above and to the right), and the pulldown immediately to the right shows the branches that currently exist.

As we begin, there's just "master" on both the local and remote (GitHub) repos.

Let's add a new branch for "dev1", setting the "Remote" option to "(None)".

This pops up a "Git Branch" window noting

```
>>> git checkout -B dev1
Switched to a new branch 'dev1'
```

Under the Hood

There are actually a few common git commands being used here.

git branch by itself simply lists the branches that exist, and indicates which one is currently active.

git branch branchname creates “branchname” as a new branch.

git checkout branchname makes branchname the active branch.

git branch -B branchname both creates the branch and selects it as active.

From the Command Line

```
kabaggerly$ git branch
* dev1
  master
kabaggerly$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
kabaggerly$ git branch
  dev1
* master
```

When dev1 is first created, the files in it are the same as in master.

Adding a New File

Let's create a new Rmd file, generate a corresponding md file, add, and commit.

Now look at the Git changes/history popup, which lets us toggle between the histories of different branches.

Looking at the master history, we see indicators for the location of the last commit applied to both "master" and "origin/master".

Looking at the dev1 history, we see one more commit, showing we've moved beyond where master was, and an indicator for "HEAD" indicating dev1 is currently active.

Check it Out!

Showing the pulldown for which branches exist is equivalent to invoking git branch.

Selecting a branch from the pulldown (e.g., “master”) is equivalent to git checkout.

Keep an eye on the files pane as we alternate checking out master and dev1.

The files in the folder change depending on which branch we’re in.

Git is either hiding files away or pulling them out of the repo’s internals (.git/) as needed.

Merging Back Into Master

Let's say our addition to dev1 satisfies us, and we'd like to move the change over into the master branch.

Checkout master.

git merge branchname tries to merge the contents of branchname with those of the active branch (whose name is implicit).

git merge is the first command for which there isn't (as far as I know) a neat RStudio GUI shortcut.

Here, we want to try
git checkout master and then
git merge dev1.

Cleaning Up

In this case there was a straight progression from the last commit in master to the last commit in dev1.

git “flattens out” the tree structure using a “fast-forward” (ff) merge. An ff merge doesn’t need a new commit message.

Once we’re done with a branch (we’ve merged it), it’s good practice to prune it so it doesn’t clutter up the workspace.

git branch -d branchname gets rid of the branch if it can.

This is another task there isn’t a shortcut for (yet).

git branch -d dev1

Taking it Slow

Let's repeat the above process (tweaking a file) with a new branch, "dev2".

When it comes time to merge it back to master, however, let's tell it not to fast-forward, so we can track the branching structure more clearly. Doing things this way requires a new "merge commit" message.

```
git checkout master
git merge --no-ff dev2 -m "merging dev2"
git branch -d dev2
```

Messaging

I normally don’t use the –no-ff option, so I occasionally forget the message when I do.

Git really wants a message for each commit, so it will fire up a text editor to ask for one.

Unfortunately, the default text editor may not be one you’re all that familiar with! You can set this with

```
git config --global core.editor ``editorname''
```

Parallel Progression and Conflict

When you work with mutltiple branches, it's possible for things to get out of sync in incompatible ways.

You might do some work on dev and do other work on master before merging.

You might have multiple dev branches as you try to introduce different types of new functionality.

When merging tell gits to do two incompatible things at once, we get a merge conflict.

git will ask you to resolve the merge conflict
by editing the corresponding file. The revised file will then need to be added and committed to fix things.

Let’s Break Something

Let’s start a new development branch, dev3, and edit one of the files present by adding a single line.

Add and commit.

Now let’s checkout master and edit the same file by adding a different last line.

Add and commit.

Now try

git merge dev3

Git's Response

```
kabaggerly$ git merge dev3
Auto-merging new_rmd_1.Rmd
CONFLICT (content): Merge conflict in
  new_rmd_1.Rmd
Automatic merge failed; fix conflicts
  and then commit the result.
```

Inside the Problem File

This opens for me in the RStudio editor

```
<<<<<<< HEAD
add text for master.
=======
add text for dev3.
>>>>>>> dev3
```

I revise this to “revised for merge”.

Adding and committing at this point finishes the merge.

Cleanup again

The merge conflict is resolved.
We’re back to a clean end on master.

We’re done with our development branch, so let’s prune it.

```
git branch -d dev3
```

Avoiding Painful Conflicts

Fixing the conflict above was pretty easy, because there weren't many differences.

If there are a lot of differences, resolution can be time-consuming and frustrating.

We can avoid this (somewhat).

Make sure you're starting from a current version of master (pull changes).

Make commits small and focused.

Test for problems regularly.

Branches and GitHub

Let's try something different.

Let's create a new development branch, dev4, but this time let's have RStudio sync the branch to our remote repo (on GitHub) at the same time.

This produces a second "tracking" branch

```
>>> git checkout -B dev4
Switched to a new branch 'dev4'
>>> git push -u origin dev4
To https://github.com/kabagg/tempTest2.git
 * [new branch]      dev4 -> dev4
Branch 'dev4' set up to track remote branch
'dev4' from 'origin'.
```

Edit, Add, Commit, Push

Here we're going through the same process as before, with one key difference.

Once we've made a change and committed it to dev4, we're going to push dev4 instead of pushing master.

If we go to our GitHub page, we see that with the change present we're being prompted to Compare & pull request.

Pull Requests

Pull requests are GitHub's way of asking the owner of a repo to consider merging the proposed branch with one they have.

Here we own the repo, so we can approve changes immediately.

But.

The pull request let's us provide a more extensive description of the proposed change in a place where others can see it and comment on it before we implement it.

Getting comments on major changes may be a good idea if you're working on a shared project!

The Discussion

Written comments on GitHub allow for markdown formatting.
You can also add emoji if you want.

Each pull request (proposed change) is assigned a number, which can be linked to. For example, you can point out

“This issue was discussed earlier in #4.”

After comments, you can merge changes you like into master.

This “closes” the pull request.
You’re now invited to delete the branch.

“Insights/Network” on the GitHub repo page shows branching and merging graphically.

What Happened Locally?

Let’s go back to our machine.

Checkout master and pull changes from GitHub.

If we try “git merge dev4”, we’re told “Already up to date”.

So, let’s clean up.

git branch -d dev4

We’re not done yet!

If we go back to the branch pulldown, we’ll see that while our local version of dev4 is gone, the tracking branch is still listed. We want to remove the tracking branch too.

Remote Branches and Cleanup

Another way of seeing this is to invoke

```
git branch -r
```

This lists all of the remote branches
(this is called by RStudio's pulldown behind the scenes).

We can remove the tracking branch by supplying both the -d and -r options

```
git branch -d -r origin/dev4
```

This tidies things up!