

Using Git, Part 2: GitHub

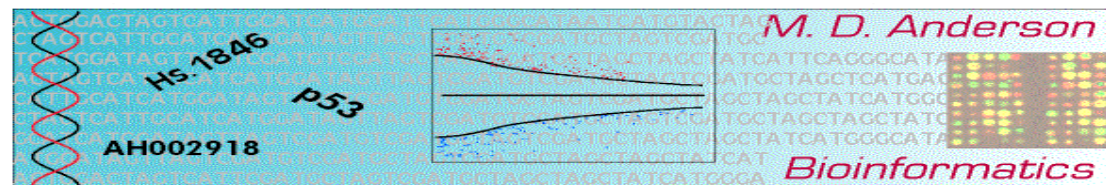
Keith A. Baggerly

Bioinformatics and Computational Biology

UT M. D. Anderson Cancer Center

`kabagg@mdanderson.org`

SISBID, July 18, 2017



Version Control and Sharing

In Part 1, we looked at how you could apply git in your own projects, including tracking, branching, and project flow.

The real power of version control is how it can help you share your work and collaborate. GitHub is an excellent example of this, but we're going to take a roundabout path to get to collaboration.

The boundaries between this part and the next are blurry.

Collaboration involves working with “remote” repositories, “bare” repositories, and requires a few more steps, but the basics are similar.

Cloning a Repo (same Filesystem)

The repos we've constructed thus far are essentially Working Directories with special “.git” subdirectories.

If you're working with someone else in your group (at your institution, in your department, etc) and you've got your work under version control, it's often useful to share repos - each of you can work on your own parts of a project, and then these parts can be merged.

This typically begins by “cloning” the first repo

```
git clone /path/to/initial-repo \  
    /path/to/cloned-repo
```

Checking the Cloned Repo

```
$ cd cloned-repo/
$ git remote
origin
$ git branch
* master
$ git remote -v
origin /Users/kabaggerly/Repro/TestGit/\
    initial-repo (fetch)
origin /Users/kabaggerly/Repro/TestGit/\
    initial-repo (push)
$ git branch -r
origin/HEAD -> origin/master
origin/master
```

Linked Local and Remote

You might guess the cloned repo would be the same as the initial one, and you'd be *almost* right.

The “cloned” repo is linked to a “remote” repo (the one we started from) which is accessed under the name “origin”.

We can access the contents of origin as if they were part of a “remote branch” (listed by the call to `branch -r`).

Adding a Remote Repo

We can set up the converse mapping as well

```
$ cd ../initial-repo  
$ git remote add myClone ../cloned-repo  
$ git remote -v  
myClone ../cloned-repo/ (fetch)  
myClone ../cloned-repo/ (push)
```

So what if things change?

Keeping in Sync

Let's head back to the clone and add a file.

```
$ git add newFile.txt
```

```
$ git commit -m "add newFile"
```

```
[master 84fd1c4] add newFile
```

```
1 file changed, 1 insertion(+)
```

```
$ git status
```

On branch master

Your branch is ahead of 'origin/master' by 1 commit.

(use "git push" to publish your local commits)

nothing to commit, working directory clean

Fetching from the Clone

```
$ git fetch myClone
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From ../cloned-repo
 * [new branch]      master      -> myClone/master
$ git branch -r
    myClone/master
$ git log master..myClone/master
commit 84fd1c4c07942fa8c34cc408ae797fe5b411dc66
Author: Baggerly, Keith A <kabagg@mdanderson.org>
Date:   Fri Jul 17 18:23:12 2015 -0500
    add newFile
```

Merging Changes

```
$ ls
README.txt
$ git branch
* master
$ git branch -r
  myClone/master
$ git merge myClone/master
Updating a69fad4..84fd1c4
Fast-forward
 newFile.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 newFile.txt
$ ls
README.txt newFile.txt
```

What else? Let's Push

```
$ git branch illustrationBranch
$ git push myClone illustrationBranch
Total 0 (delta 0), reused 0 (delta 0)
To ../cloned-repo/
 * [new branch]          illustrationBranch -> illustrati
$ cd ../cloned-repo/
$ ls
README.txt newFile.txt
$ git branch
    illustrationBranch
* master
```

this may be impolite.

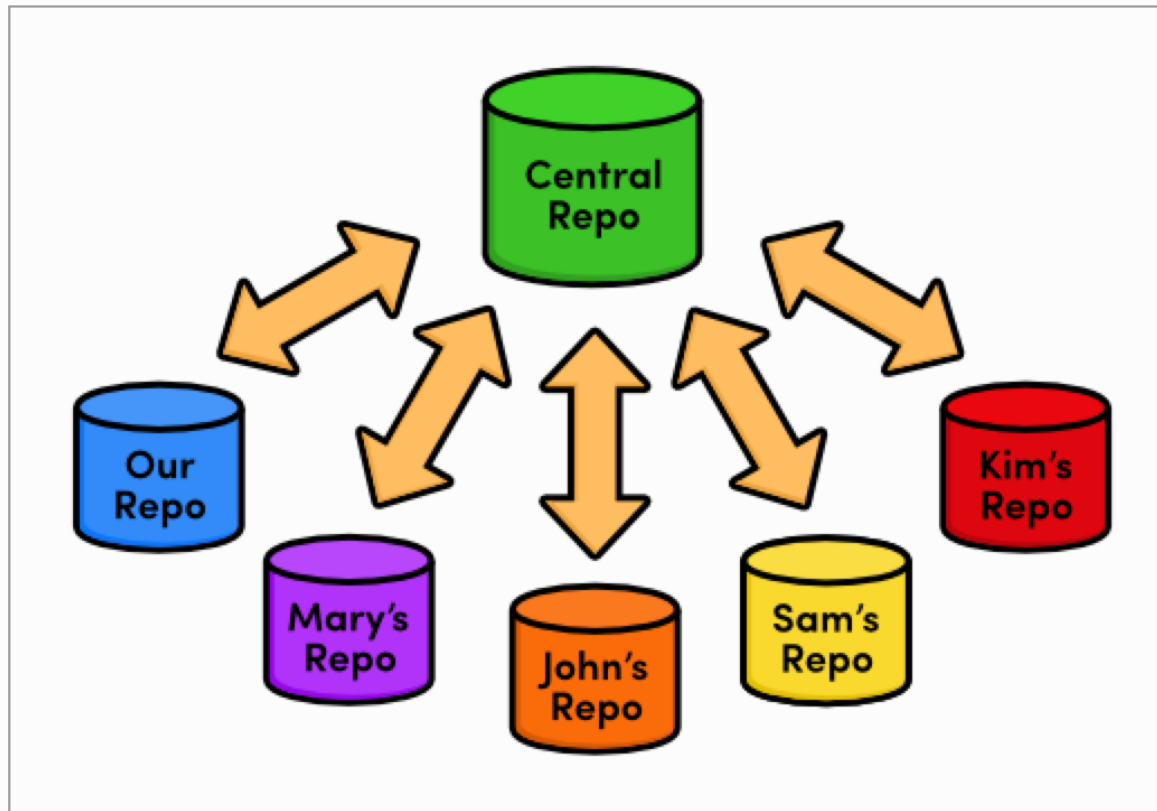
Sharing Politely - Bare Repos

In many cases, it's nice to separate development, which may have uncommitted files, from sharing.

For this, we use a “bare” repository, whose name (by convention) should end in .git. Before our repo was a working directory with a .git subfolder. A bare repo is, in essence, just the .git subfolder. Files here are commits from somewhere else.

```
$ cd ..  
$ ls  
cloned-repo  initial-repo  
$ git init --bare laBare.git  
Initialized empty Git repository in  
  /Users/kabaggerly/Repro/TestGit/laBare.git/
```

Central (bare) repos



The centralized workflow with many developers

from Ry's Git Tutorial, Centralized Workflows
Bare repos make good archives.

Other Ways of Sharing: SSH

Repo on a server? Treat it as a remote! From “initial-repo”:

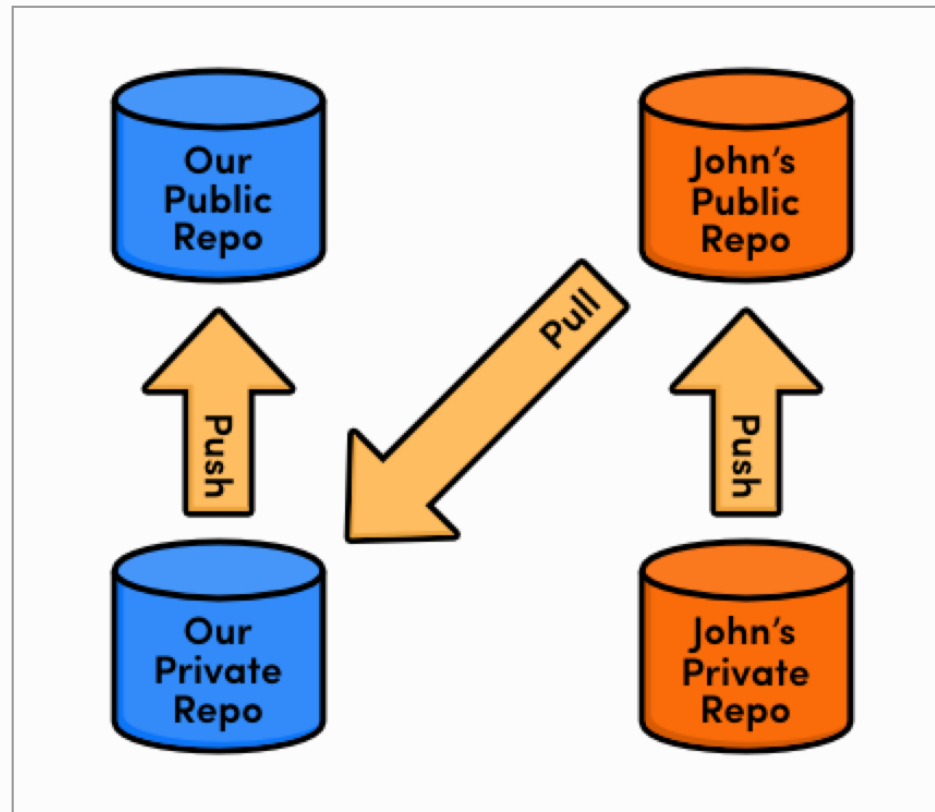
```
$ git remote add myServer \
    ssh://kabaggerly@mdadqscfs01.mdanderson.edu/\
    home/kabaggerly/TestGit/bareRepo.git
$ git push myServer master
Password:
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 531 bytes | 0 bytes/s
Total 6 (delta 0), reused 0 (delta 0)
To ssh://kabaggerly@mdadqscfs01.mdanderson.edu/home/kabaggerly/TestGit/bareRepo.git
 * [new branch]      master -> master
```

Cloning via SSH

and now if we want our own copy

```
$ git clone \
    ssh://kabaggerly@mdadqscfs01.mdanderson.edu/\
    home/kabaggerly/TestGit/bareRepo.git
Cloning into 'bareRepo' ...
Password:
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (6/6), done.
Checking connectivity... done.
```

Sharing with Others



The integrator workflow

from Ry's Git Tutorial, Distributed:
Fork, Clone, Edit, Push, Wave, Fetch or Pull

GitHub is Findable Owned Bare Repos

GitHub lets us share pretty much anything with anyone, in a way that allows for security.

Anyone can see what you want to share, but they can't edit without you.

In many ways, this is like *publication*, and can be used as an *archive of record*.

We'll introduce this in the context of modifying our `toyPackage`.

We'll also look more at how Rstudio's graphical interface highlights some stuff we know how to do.

Going to GitHub

Say we want to share our package with others worldwide.

The easiest way to do this is to post it on GitHub as a public repository.

If there're only a few people you want to see the repository, then you can either pay a small fee to maintain some private repositories, or ask at your institution about whether local repositories are available.

A common variant of this is GitLab.

For now, let's work with GitHub.

Registering at GitHub

Registration and a few public repositories can be had for free!

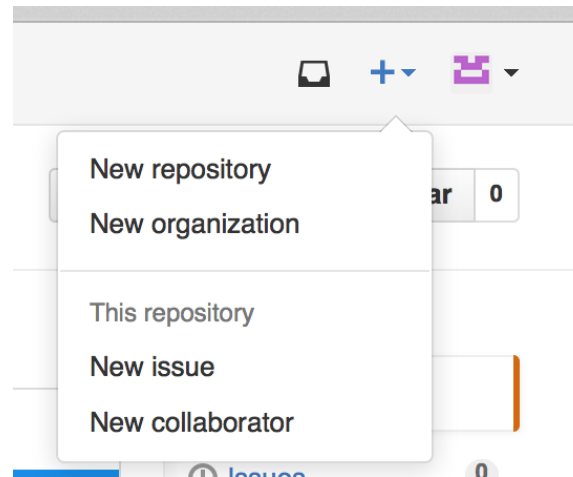
You do need to supply your name, email, and a password.

To keep things simple, please use the same values for these as you used when configuring git on your own machine.

Creating a GitHub Repository

Since sharing repos is the main reason for GitHub's existence, they try their darnedest to make this easy.

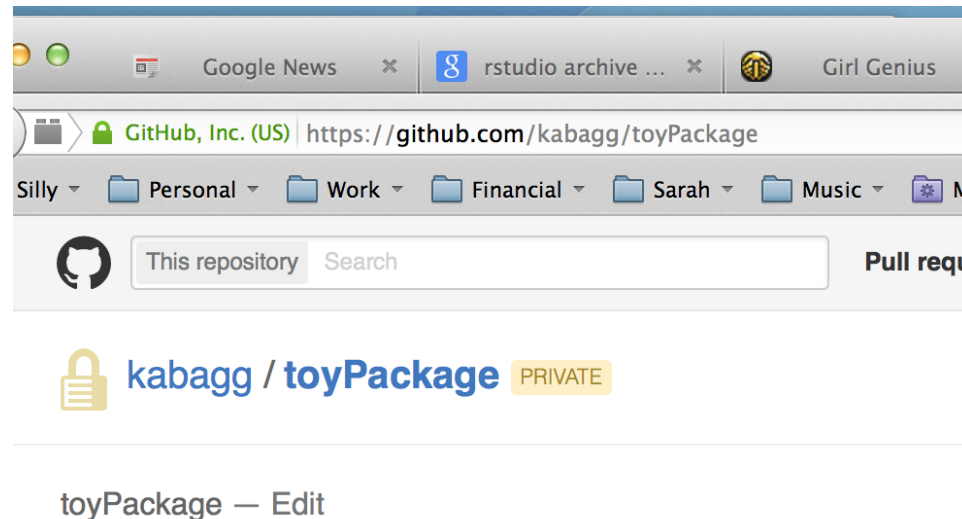
In the upper right of every GitHub page there's a "+" pulldown menu which will let you create a "New Repository".



When you do, it'll ask for a name and a short description; let's use the package name for both (i.e., "toyPackage")

Congratulations, it's a Repo!

This repo not only exists, it has a web page! The url is `https://github.com/yourUsername/yourPackageName`



Now, GitHub really wants *stuff* in its repos. It'll encourage you to put some files (such as README.md) in your repo right away. We won't, because we're going to fill the repo with material from our local machine using “git push”.

“Push” Setup

In order to “push” our package to GitHub, we need to leave Rstudio and go to a shell for a few direct invocations.

Using the “Tools” option from the Rstudio panel will invoke a shell for us, and (if we invoke this with our `toyPackage` open) will shift folder locations to where we want.

Within the shell, we now need to do two things:

- (1) tell Git where we want to put our remote repository, and
- (2) actually push it there.

Telling Git Where to Go

First, we supply git with location of the remote repo

```
git remote add origin \  
https://github.com/kabagg/toyPackage.git
```

This will tweak your git configuration (the “config” file in your .git/ folder) by adding info:

```
[remote "origin"]  
  url = https://github.com/kabagg/toyPackage.git  
  fetch = +refs/heads/*:refs/remotes/origin/*
```

This is where git keeps track of such settings.

What if you Get it Wrong?

I ask because I did the first time I tried this ;).

Don't edit the config file directly; that's a bad habit to get into.

Rather, try

```
git remote rm origin
```

This will properly remove (rm) the mistaken entry for origin from the config files, and let you supply the correct one.

Pushing to GitHub

Now we push to GitHub (and get display arrows in Rstudio!)

```
git push -u origin master
```

```
Username for 'https://github.com': kabagg
```

```
Password for 'https://kabagg@github.com':
```

```
Counting objects: 25, done.
```

```
Delta compression using up to 8 threads.
```

```
Compressing objects: 100% (18/18), done.
```

```
Writing objects: 100% (25/25), 3.87 KiB | 0 bytes.
```

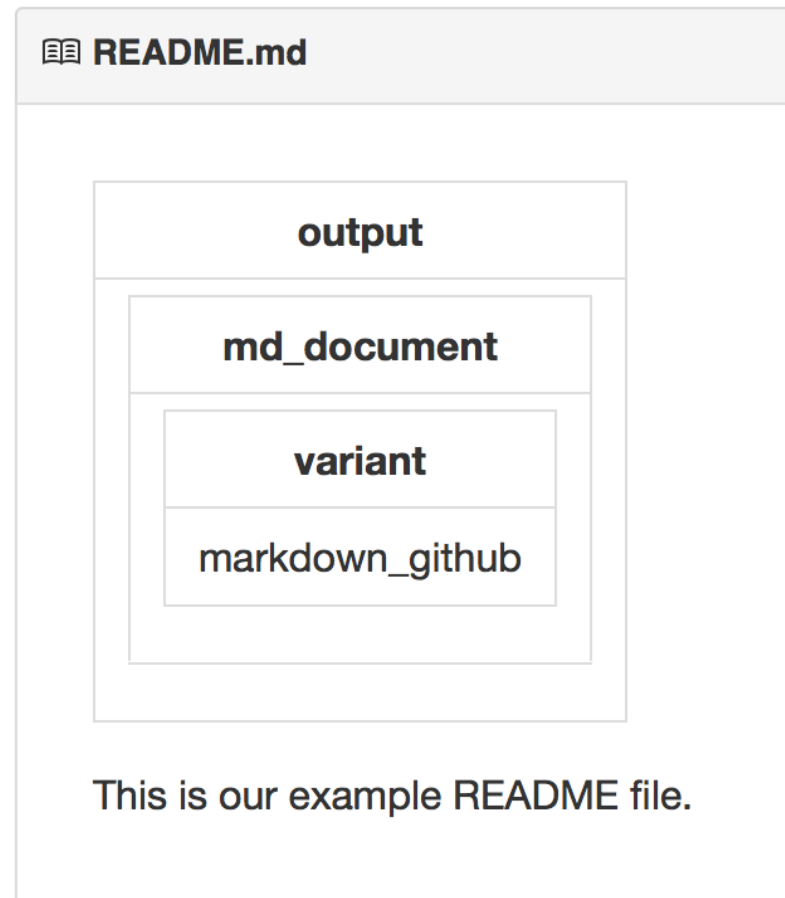
```
Total 25 (delta 2), reused 0 (delta 0)
```

```
To https://github.com/kabagg/toyPackage.git
```

```
 * [new branch]      master -> master
```

```
Branch master set up to track remote branch master
```

Looking at our README.md on GitHub



We have a webpage for free!

One Edit You Should Always Make

Once you establish a GitHub (or GitLab) repo for a project (or package), and have put the committed materials in it, you should update

DESCRIPTION files in packages to include the URL of the GitHub repo

Abstracts in reports to include the URL of the GitHub repo

This will dramatically increase findability.

If the GitHub repo is public, there's even a further increase, because Google can now find it and include links in response to search queries.

Editing our Toy Package

Now let's go back to our machine and change something.

In particular, let's add another function, "plotSquare.R" (I'll let you guess what this does).

Now we need to
edit the roxygen function comments
edit .Rbuildignore
and then, using devtools,

```
document()  
build()  
install()  
check()
```

Ready to Commit?

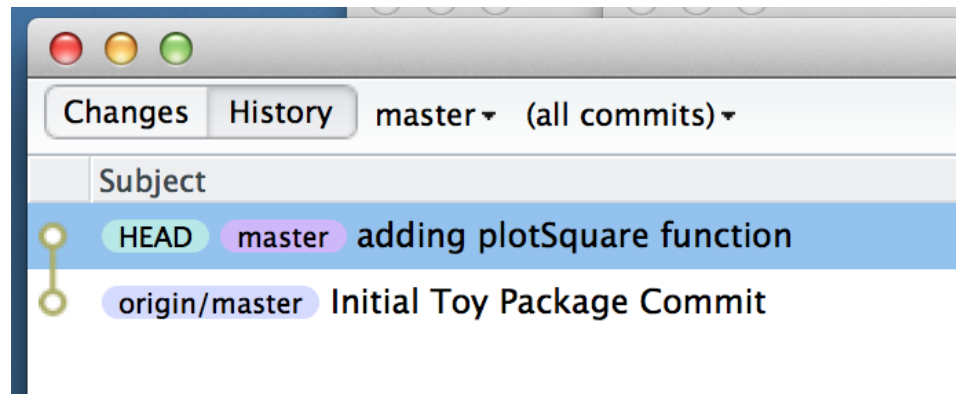
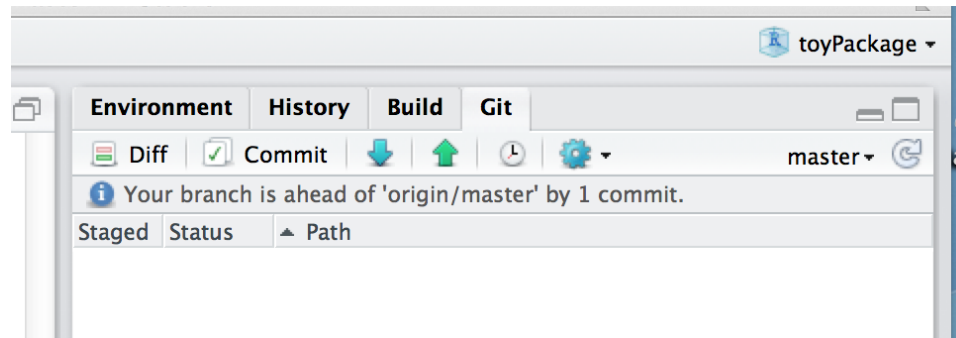
The above edits should change

R/plotSquare.R
man/plotSquare.Rd
NAMESPACE
.Rbuildignore

Once our revised package passes “check”, stage the changed files and commit the changes.

Where Are We Now?

We're now *ahead* of GitHub




The commit history shows a chain


Pushing the Changes to GitHub


The git pane “up arrow” will push changes to GitHub

```
~/kepro/testPackage/toyPack  
Git Push  
To https://github.com/kabagg/toyPackage.git  
86d9855..88831a6  master -> master
```


toyPackage — Edit


 2 commits


 1 branch

 branch: **master** ▼ **toyPackage** / +

adding plotSquare function

 **kabagg** authored 7 minutes ago

 R adding plotSquare function

 data Initial Toy Package Commit

pull = fetch + merge

When you pull down data from a remote repo, you're actually doing two things:

- (1) fetching the data (as a new branch), and
- (2) trying to merge that branch with what you've got in place.

Sometimes you may want to split these steps apart, so you can “fetch” the data and edit it before trying to merge it. This may let you avoid merge conflicts, or simply reduce their number.

Back to Rstudio

Some of these changes (pushing and pulling) are used often enough that Rstudio has added specific buttons to address them.

How does it represent some of the other stuff that's going on, e.g. branches?

When do pictures improve on the command line?

Let's revisit some stuff from before...

The Naming History

Search			Retresh	Pu
Author	Date	SHA		
Baggerly,Keith A <kabagg@mdanderson.org>	2015-07-08	88831a60		
Baggerly,Keith A <kabagg@mdanderson.org>	2015-07-07	86d98557		

```
BCBMC02L30BVFFT:toyPackage kabaggerly$ git log
commit 88831a608ec7119231101ad28828449e7013479f
Author: Baggerly,Keith A <kabagg@mdanderson.org>
Date:   Wed Jul 8 10:16:52 2015 -0500
```

adding plotSquare function

```
commit 86d98557c3cd3671b25d217eaafa2e93d1589b68
Author: Baggerly,Keith A <kabagg@mdanderson.org>
Date:   Mon Jul 6 21:57:01 2015 -0500
```

Initial Toy Package Commit

```
This is a minimal functioning R package.
BCBMC02L30BVFFT:toyPackage kabaggerly$ git log
```

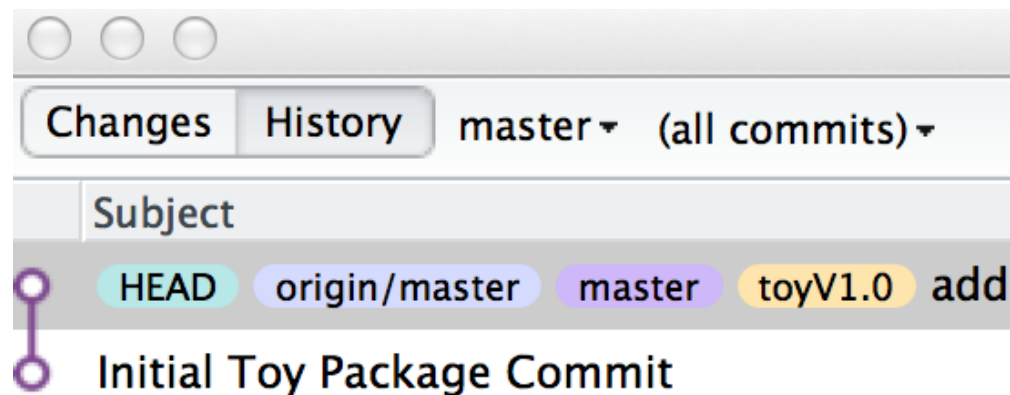
Every commit is “named” with a secure hash (SHA). Any unique part of the SHA can id the commit.

Tagging Commits

If we want to refer to a commit by something other than its SHA, we can assign a “tag” to that commit.

```
git tag toyV1.0 88831
git tag
toyV1.0
```

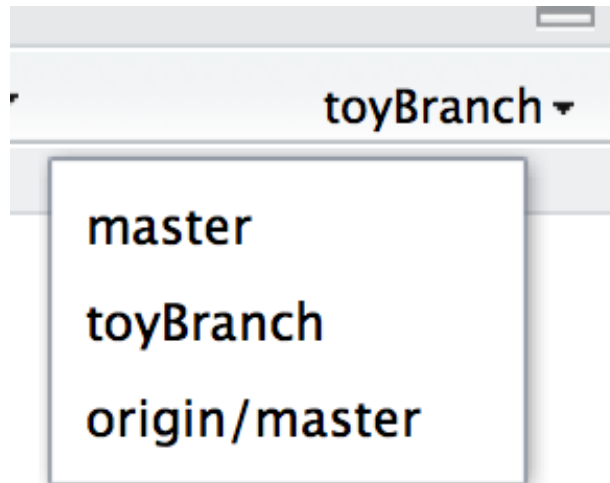
This tag also appears in Rstudio’s commit page



Adding a Branch

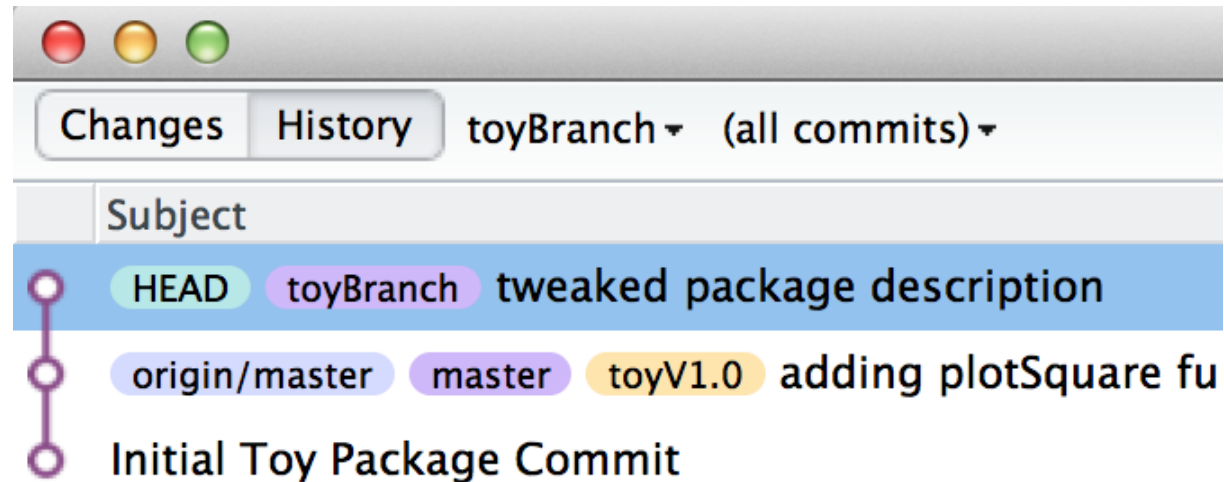
We can shift to a new named branch as follows:

```
> git checkout -b toyBranch master  
Switched to a new branch 'toyBranch'
```



Tweak Something

I added a sentence to the package documentation, saved, staged, and committed.



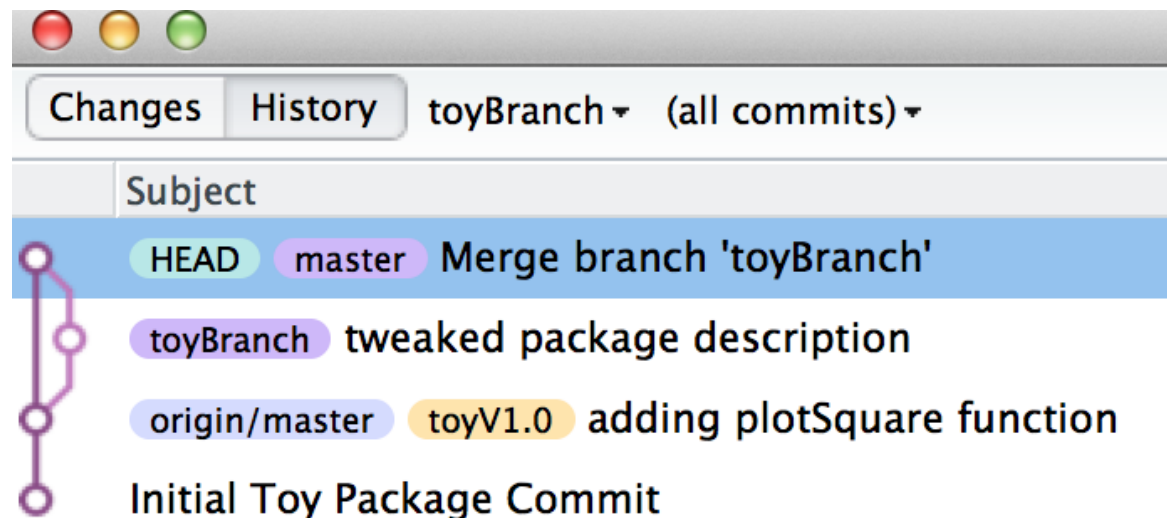
Merge in the Change

“Checkout” shifts us to the specified branch, here “master”

```
git checkout master
```

```
git merge --no-ff toyBranch -m "minor change"
```

The “--no-ff” option keeps the history around pictorially

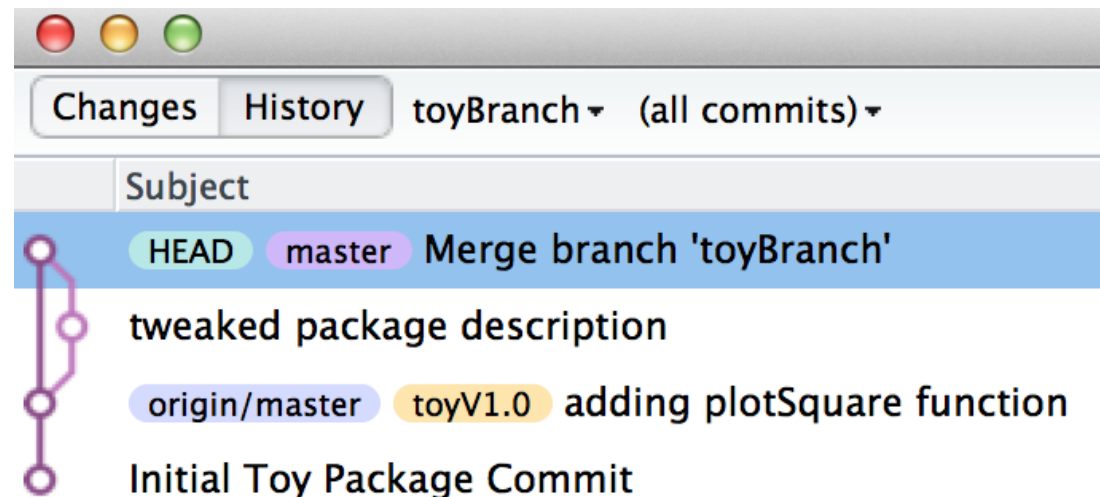


Prune the Branch

Since I'm not taking this further, I also delete the branch, and will work with a new branch from master if needed.

```
git branch -d toyBranch
```

Deleted branch toyBranch (was 0d8fcd9) .



How We Can Share With GitHub

posting R packages

posting analyses

including README descriptions in markdown

allowing for public searching

sharing with yourself later
