

Using Git with GitHub (Part 2)

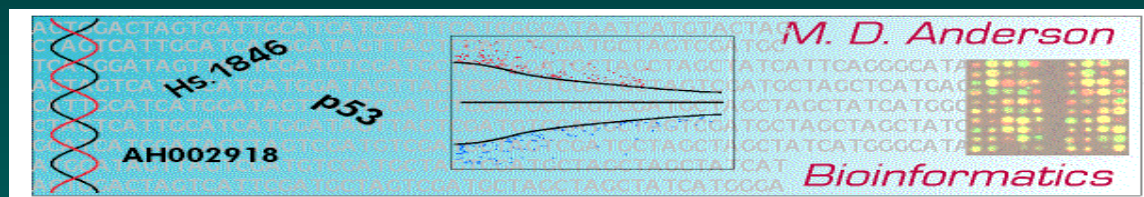
Keith A. Baggerly

Bioinformatics and Computational Biology

UT M. D. Anderson Cancer Center

kabagg@mdanderson.org

SISBID, July 19, 2016



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.

But, as we've noted, the real power of version control is how it can help with collaborative efforts. GitHub is an excellent example of this, but we're going to take a roundabout path to get there.

Collaboration involves working with “remote” 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.

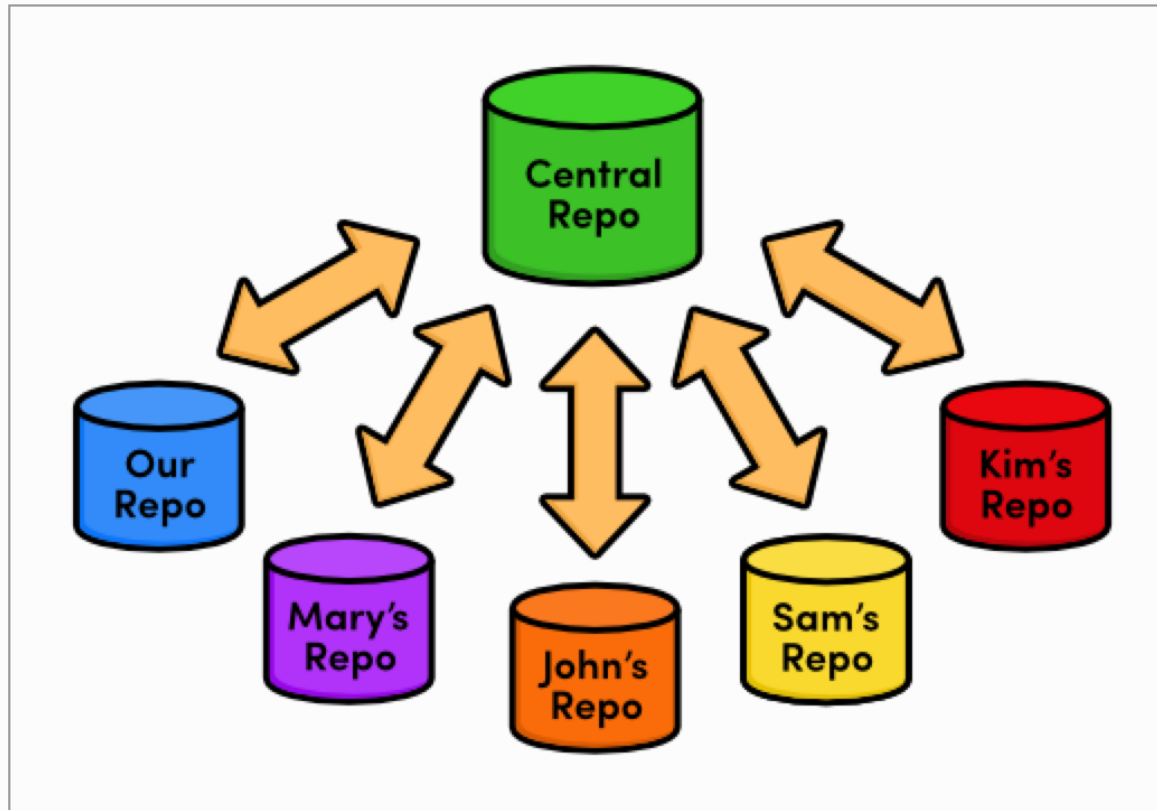
Sharing Politely - Bare Repos

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

Other Ways of Sharing: SSH

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

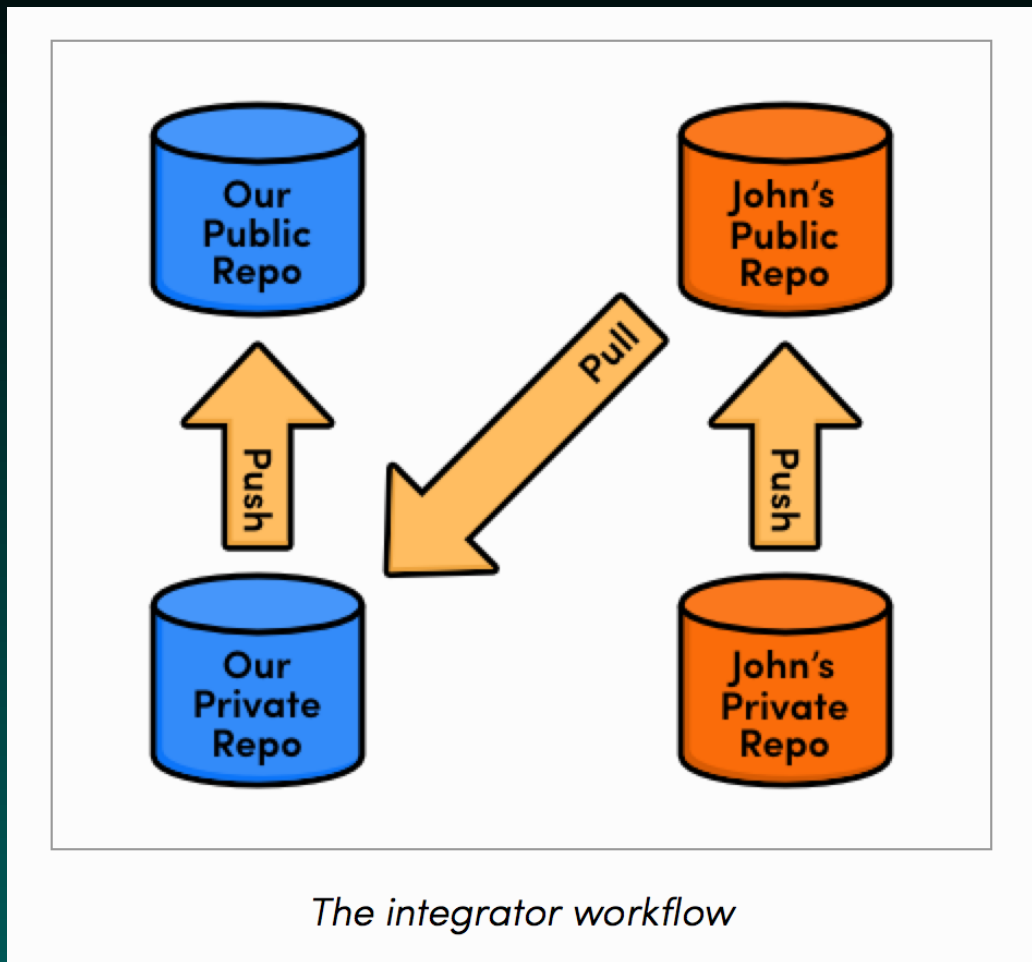
```
$ git remote add myServer \
  ssh://kabaggerly@mdadqscfs01.mdanderson.edu/\
    home/kabaggerly/TestGit/bareRepo.git
$ git push myServer master
Password:
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 531 bytes | 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

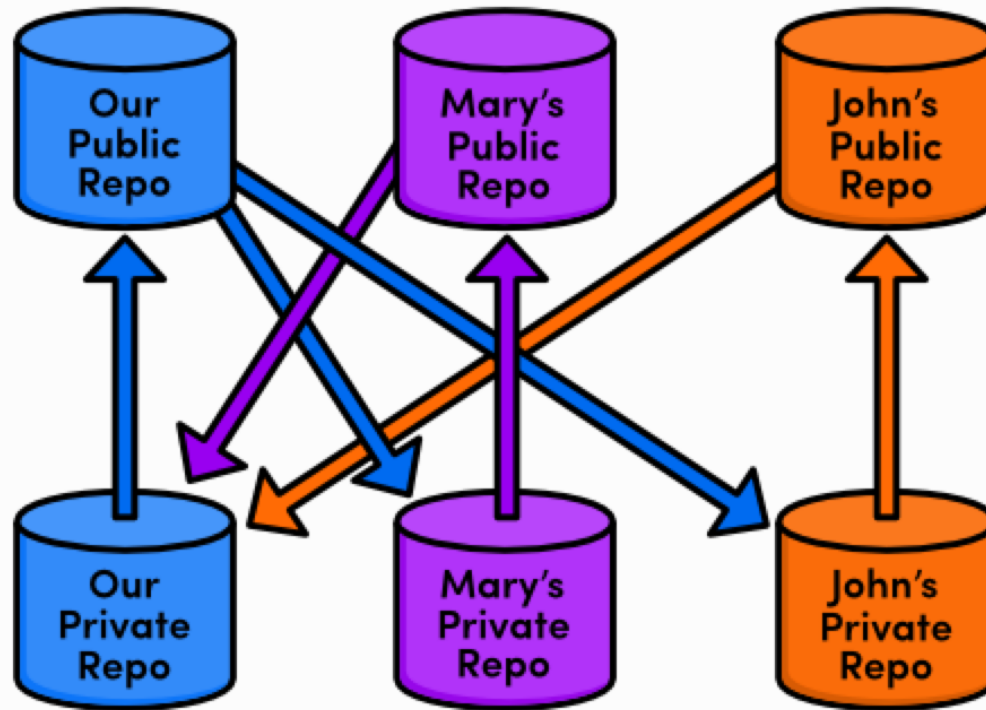
```
$ git clone \
    ssh://kabaggerly@mdadqscfs01.mdanderson.edu/\
    home/kabaggerly/TestGit/bareRepo.git
Cloning into 'bareRepo'...
Password:
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (6/6),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

Working with Others



from [Ry's Git Tutorial, Distributed:](#)
Fork, Clone, Edit, Push, Wave, Fetch or Pull

Suggestions from Others



The integrator workflow with many developers

more from [Ry's Git Tutorial, Distributed](#): Everyone can share!

GitHub is Just Such a Case

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

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

Now let's share our package with others.

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!

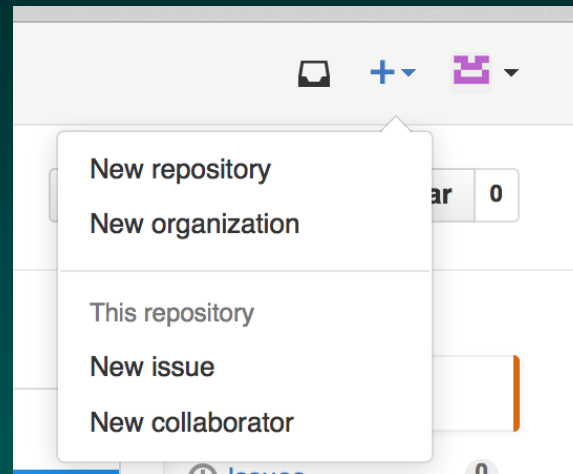
You do need to supply your name, email, and a password.

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.

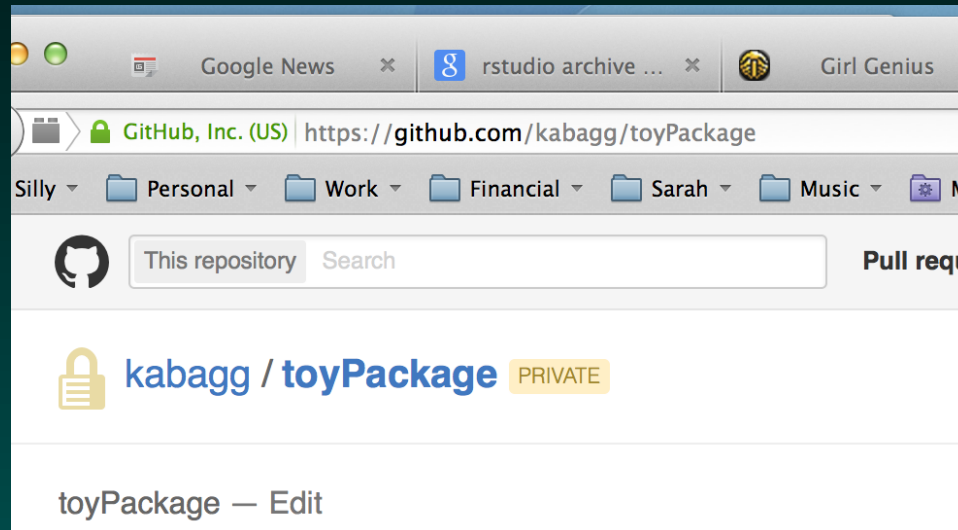
In the upper right of every GitHub page there's a "+" pulldown menu which will let you create a "New Repository".



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.

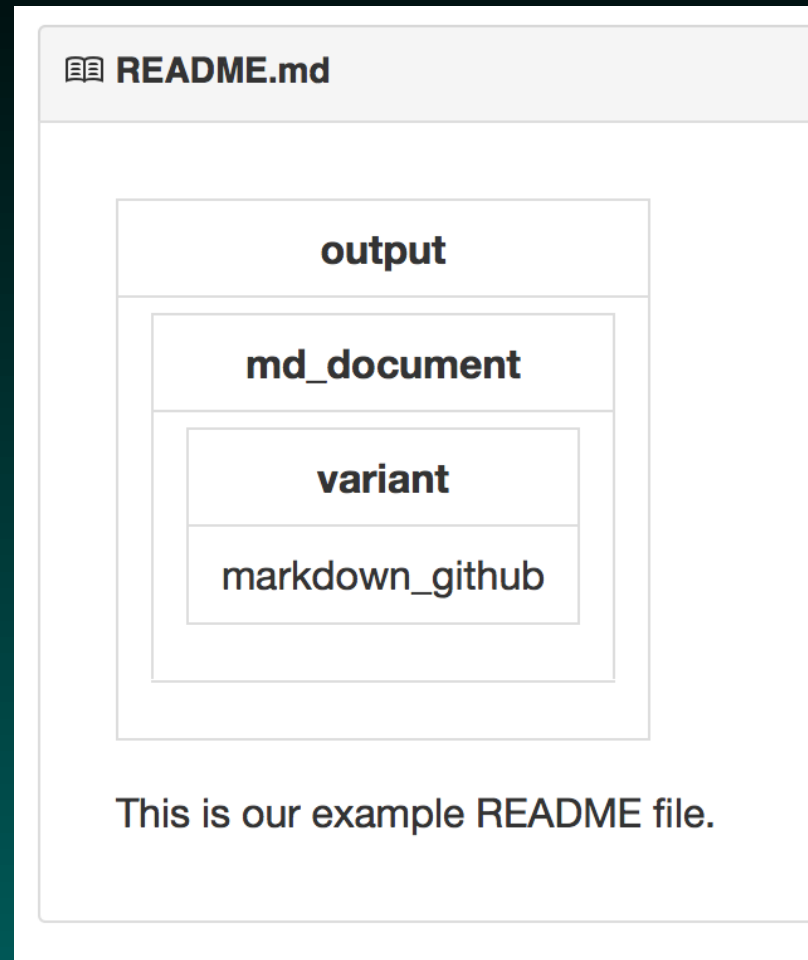
Pushing to GitHub

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, done.
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.
```

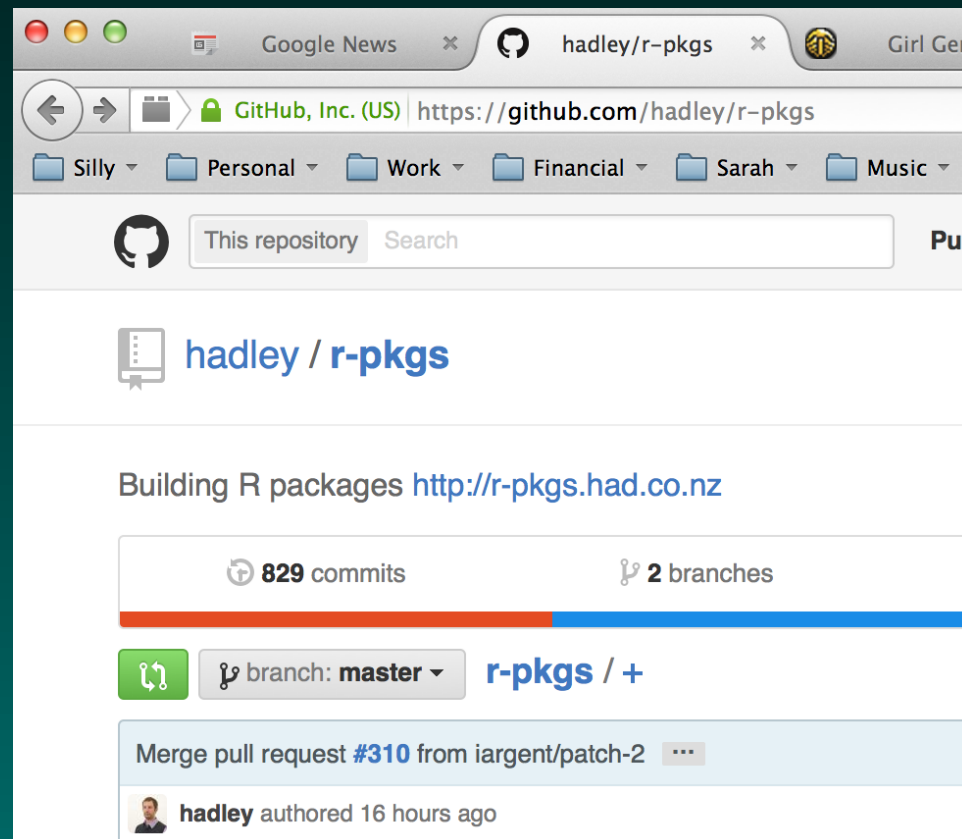
Looking at our README.md on GitHub



We have a webpage for free!

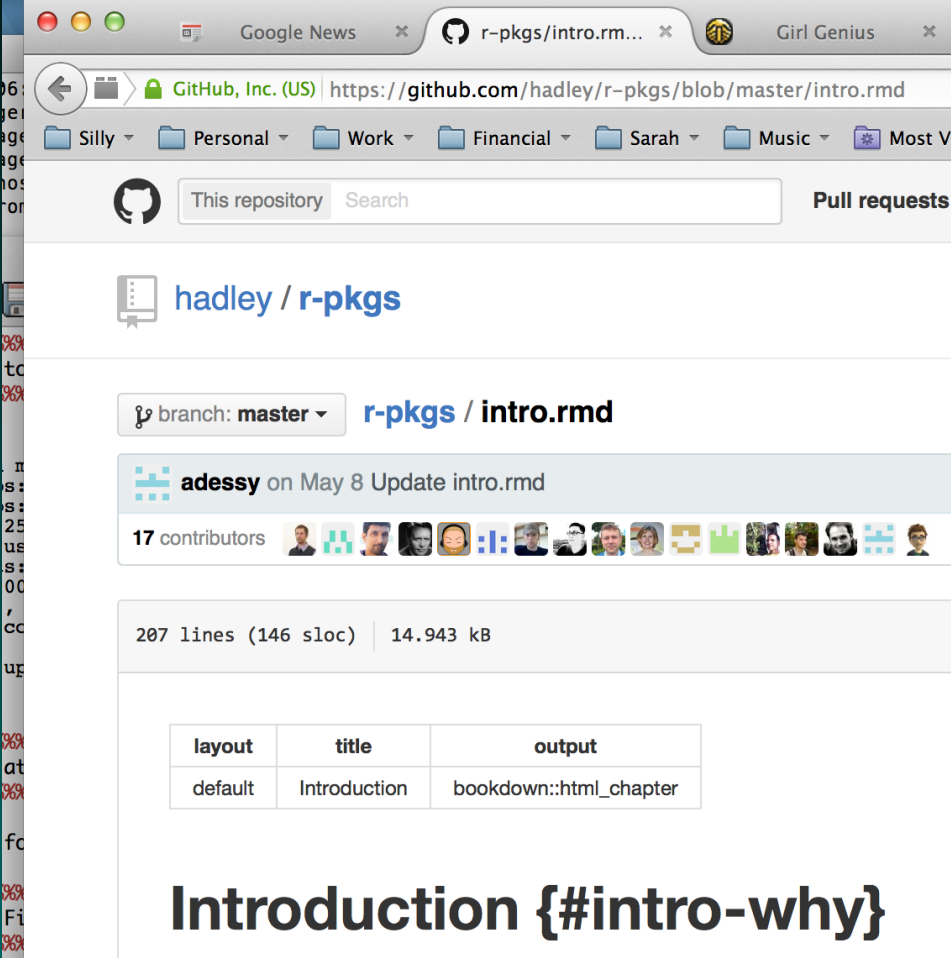
Others' Files on GitHub

While we're here, let's look around. Exploring others' repos is a great way to learn. Here're some snapshots from Hadley's *R Packages* book repo



An Introduction

Zooming in on specific files shows the contents



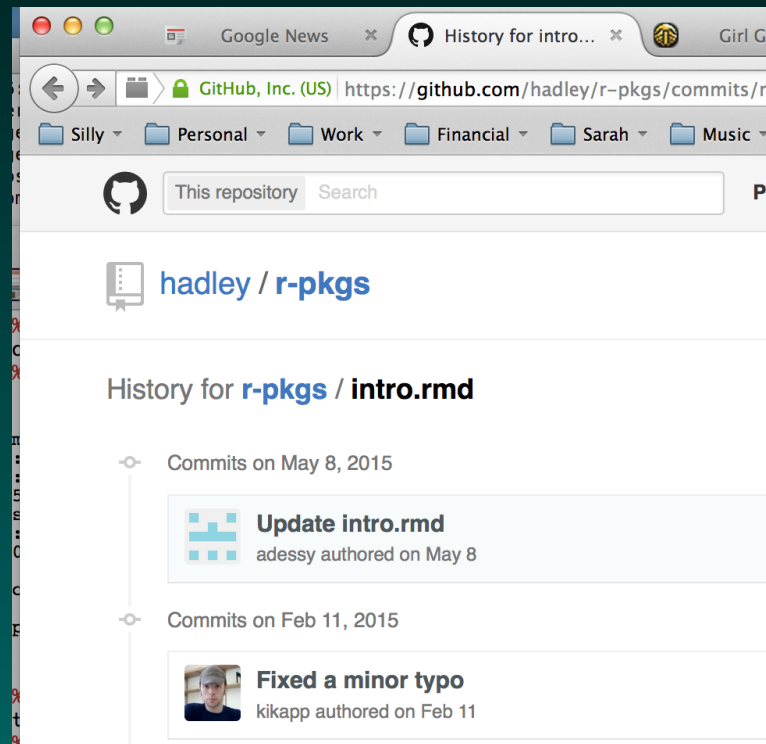
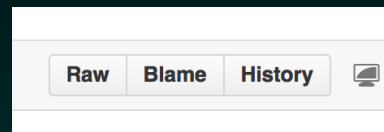
The screenshot shows a web browser window with the GitHub repository page for 'hadley / r-pks'. The URL is <https://github.com/hadley/r-pkgs/blob/master/intro.rmd>. The page displays the file 'intro.rmd' on the 'master' branch. A commit by 'adessy' on May 8 is shown, updating 'intro.rmd'. Below the commit, there are 17 contributors. The file statistics show 207 lines (146 sloc) and 14.943 kB. A table at the bottom of the file view shows the layout, title, and output for the file.

layout	title	output
default	Introduction	bookdown::html_chapter

Introduction {#intro-why}

Checking the History


Checking the history shows when edits were made





Checking an Edit




Checking an edit shows the specific changes


Fixed a minor typo


 master (#251)

 **kikapp** authored on Feb 11

 Showing **1 changed file** with **1 addition** and **1 deletion**.

2    intro.rmd

			@@ -91,7 +91,7 @@ You'll start by learning about
91	91		The final three chapters describe general best p
92	92		
93	93		* [Git and github](#git): mastering a version co
94		-	- colloborate with others, and is useful even fo
	94	+	+ collaborate with others, and is useful even fo
95	95		you to easily undo mistakes. In this chapter y
96	96		popular git and GitHub combo with RStudio.
97	97		



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?

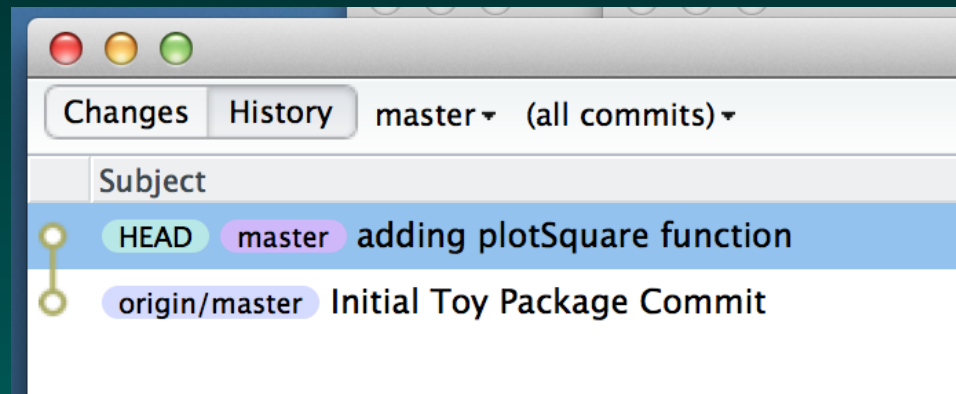
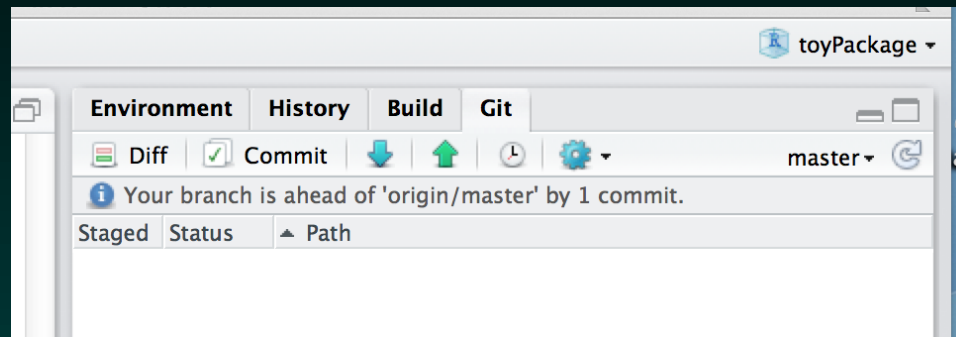
The above edits should change

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?

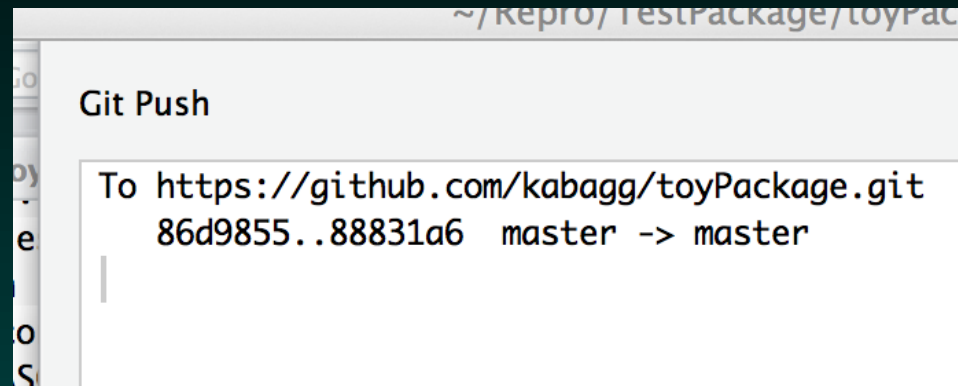
We're now *ahead* of GitHub



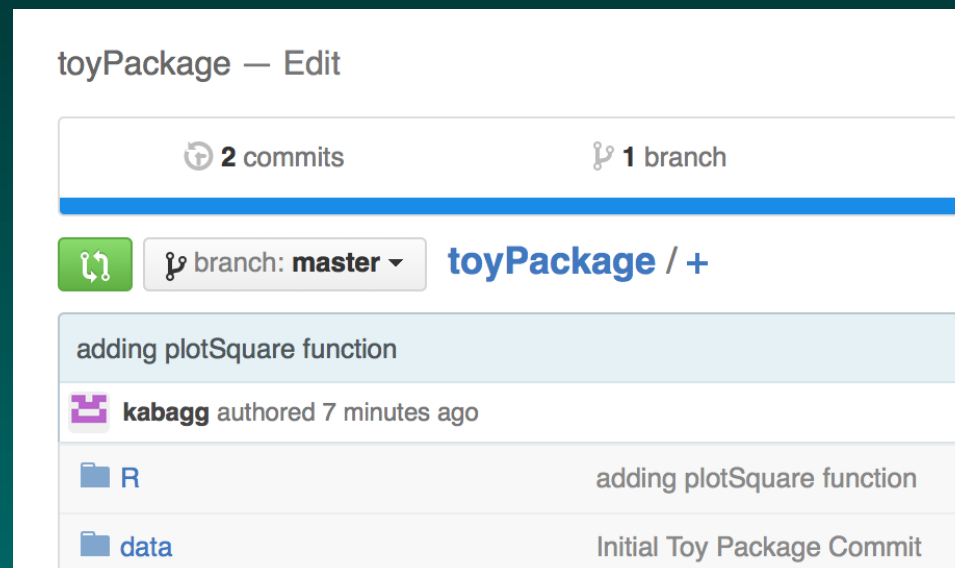
The commit history shows a chain

Pushing the Changes to GitHub

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



```
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

Working with Someone Else

There's also a *down* arrow (for “pulls”).

If we're doing our edits locally, where are these changes coming from?

Other people!

If the changes are from one of your collaborators (someone else with permission to directly push changes to the repository), then simply pulling down the changes will synchronize your repo with GitHub's (there's no problem with pulling the version you already have).

But others can suggest changes as well.

How Other Changes Work

Let's say you're looking at someone else's repo because it relates to your work, and you notice a bug. If you see a fix, you can suggest this as follows.

“fork” the repo you're looking at. This creates a copy of the repo in your GitHub account which you own and can edit.

“clone” your repo copy from GitHub to your local machine.

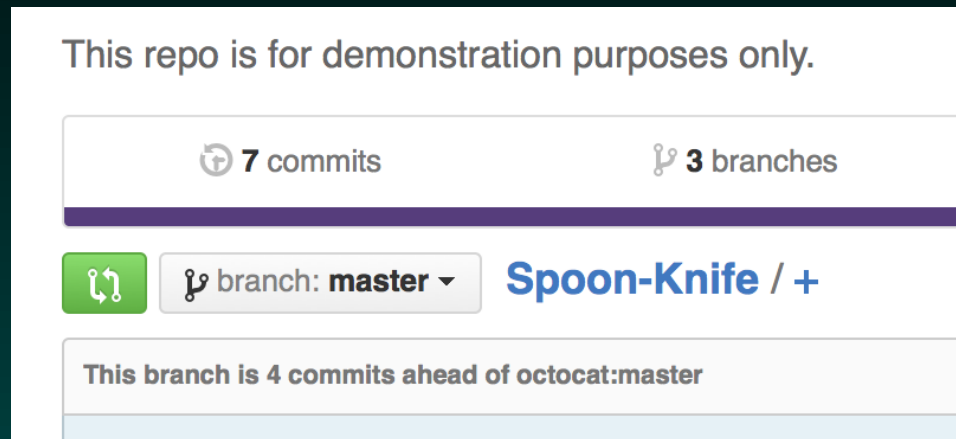
Make your edits.

Push your edited version of the repo back to GitHub.

Send the original owner a “pull request”, asking them to pull the edits you've made into their repo.

Fork and Clone

Clicking the + sign next to the repo name will fork it for you.



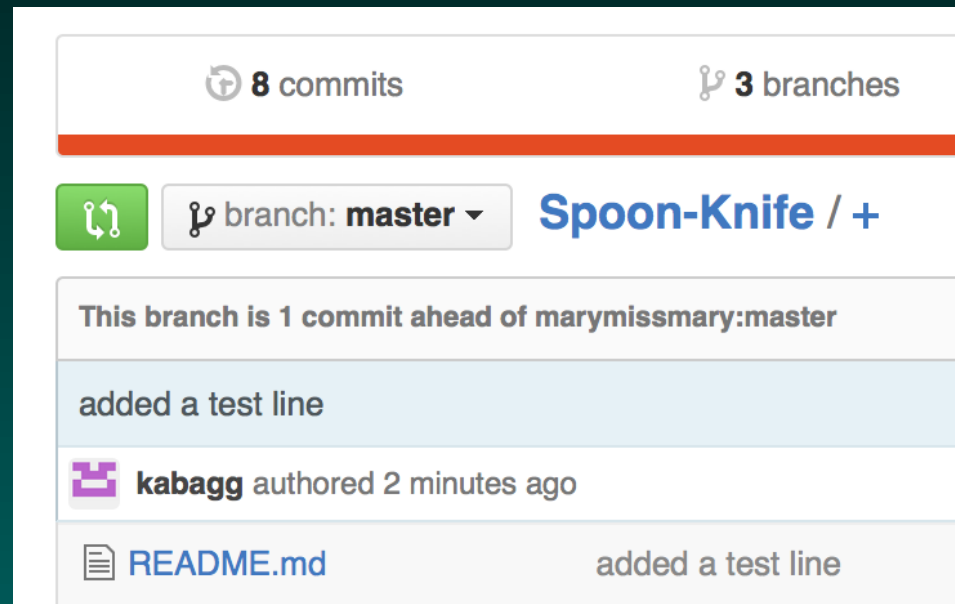
Now, within Rstudio, go to
File/New Project/Version Control/Project Setup
and give it the git url.

This will set up a local project repo for you (with a .Rproj file)

Edit and Push

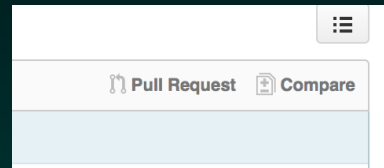
After making your edits, use the Rstudio commit popup to ignore the Rproj file and commit your desired changes.

Then push the changes over to GitHub




Submit the Pull Request

Assemble the pull request (PR), along with a brief comment about what you're trying to do.




Open a pull request

Create a new pull request by comparing changes across two branches.

 base fork: **marymissmary/Spoon-Knife** ▾ base: **master** ▾ ..

✓ **Able to merge.** These branches can be automatically merged.



added a test line

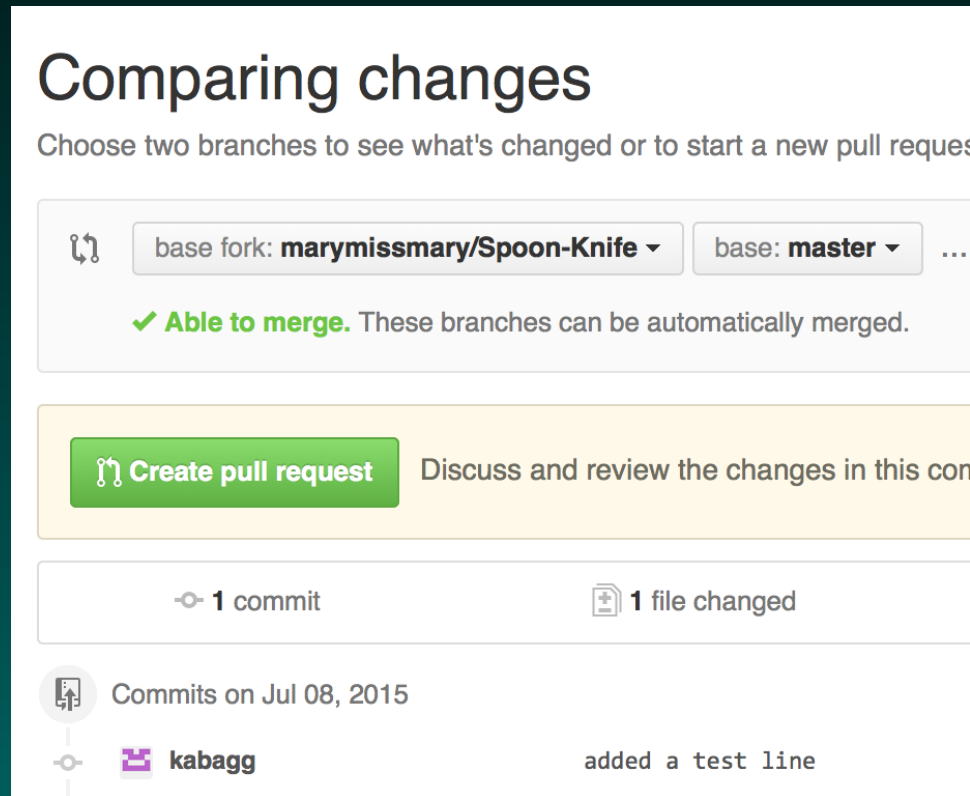
Write

Preview

Testing out pull requests with another person

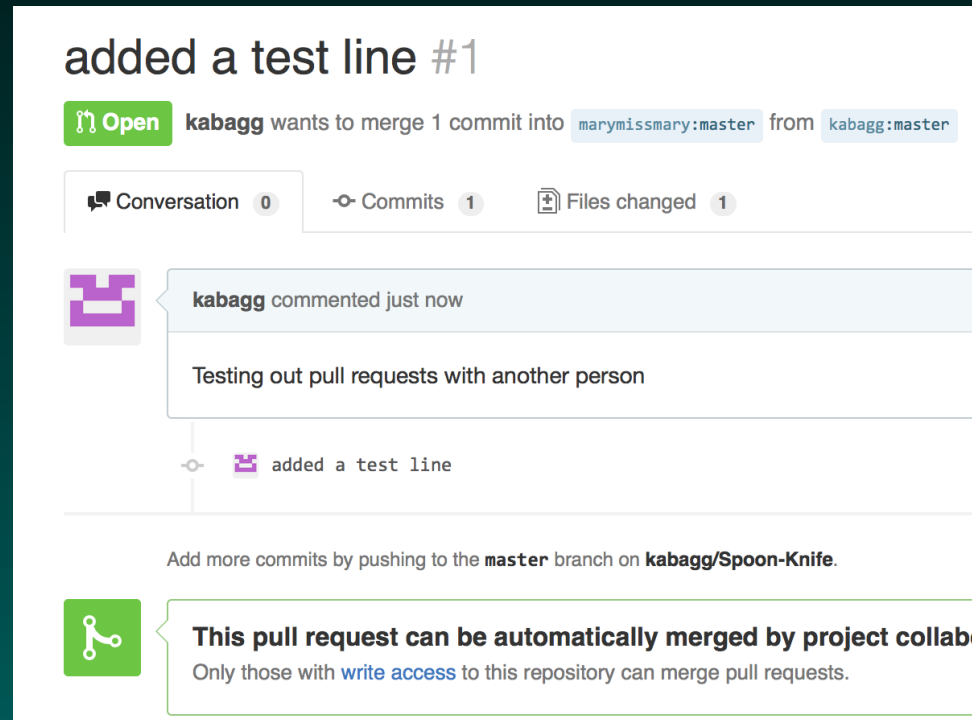
You Get Some GitHub Feedback

Even before you submit the PR, GitHub will indicate whether including the suggested changes is easy.



Welcome to the Conversation

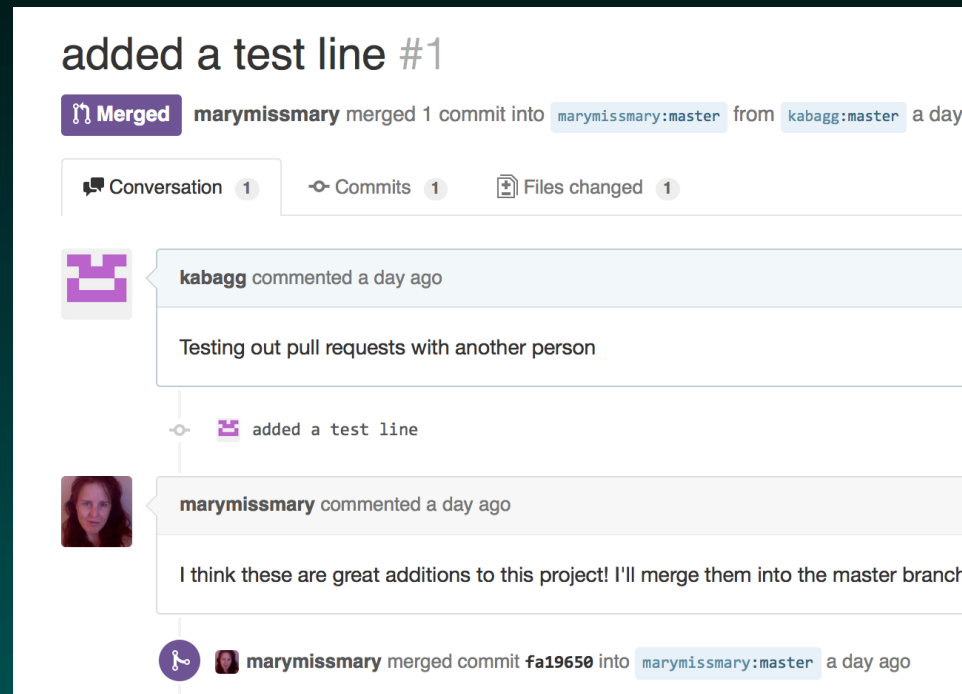
Submitting the PR starts a “conversation” with the owner (initially by email)



The screenshot displays a GitHub Pull Request (PR) interface. At the top, the title "added a test line #1" is shown. Below the title, a green "Open" button is visible, followed by the text "kabagg wants to merge 1 commit into marymismary:master from kabagg:master". A navigation bar includes tabs for "Conversation" (0), "Commits" (1), and "Files changed" (1). The "Conversation" tab is active, showing a comment from "kabagg" stating "kabagg commented just now" and "Testing out pull requests with another person". Below the comment, a commit is listed: "added a test line". At the bottom, a green box contains the text: "This pull request can be automatically merged by project collaborator. Only those with write access to this repository can merge pull requests."

Welcome to the Conversation (2)

The owner can choose to include your suggestion (or not)



Including Suggestions from Others

Of course, they can comment on your code as well ;).

The screenshot displays a GitHub pull request titled "Embellished description of plotCircle #1". At the top, a green "Open" button is followed by the text "marymismary wants to merge 2 commits into kabagg:master from marymismary:master". Below this, a navigation bar shows "Conversation 0", "Commits 2", and "Files changed 1". A comment from "marymismary" is shown, stating: "Expanded description in plotCircle to briefly describe what the function does." Below the comment, a commit history section shows two commits by "Rohrdanz, Mary A" and others: "Embellished description of plotCircle ..." (hash 5bd4c15) and "making a merge conflict ..." (hash fe03da). A message below the commits reads: "Add more commits by pushing to the master branch on marymismary/toyPackage." A green box contains the text: "This pull request can be automatically merged. You can also merge branches on the command line." with a "Merge pull request" button. At the bottom, there is a "Write" and "Preview" section with a "Markdown supported" indicator and an "Edit in fullscreen" link.

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

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.
```

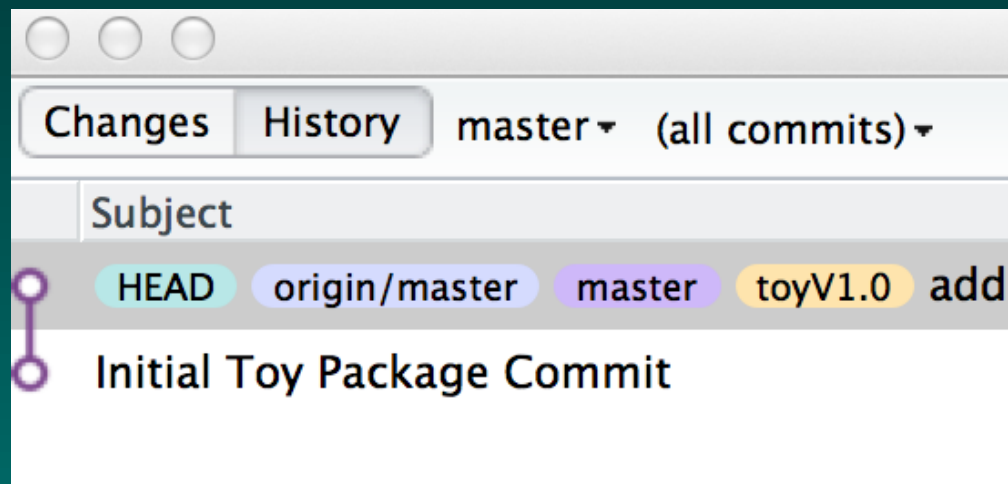
Every commit is “named” with a secure hash (SHA). Any unique part of the SHA can id the commit.

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
```

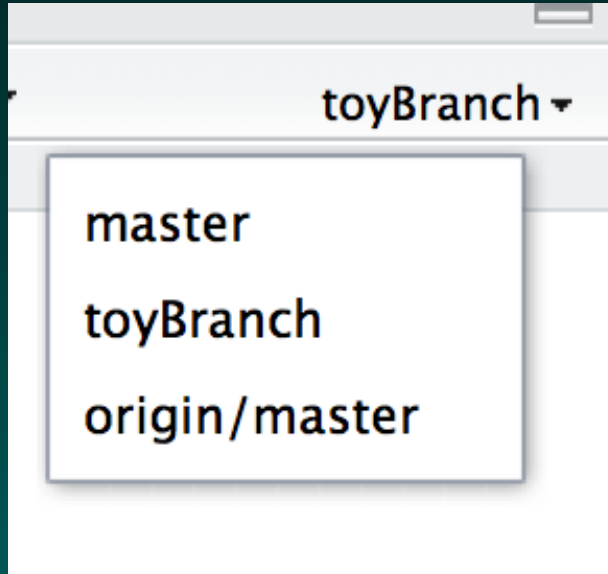
This tag also appears in Rstudio's commit page



Adding a Branch

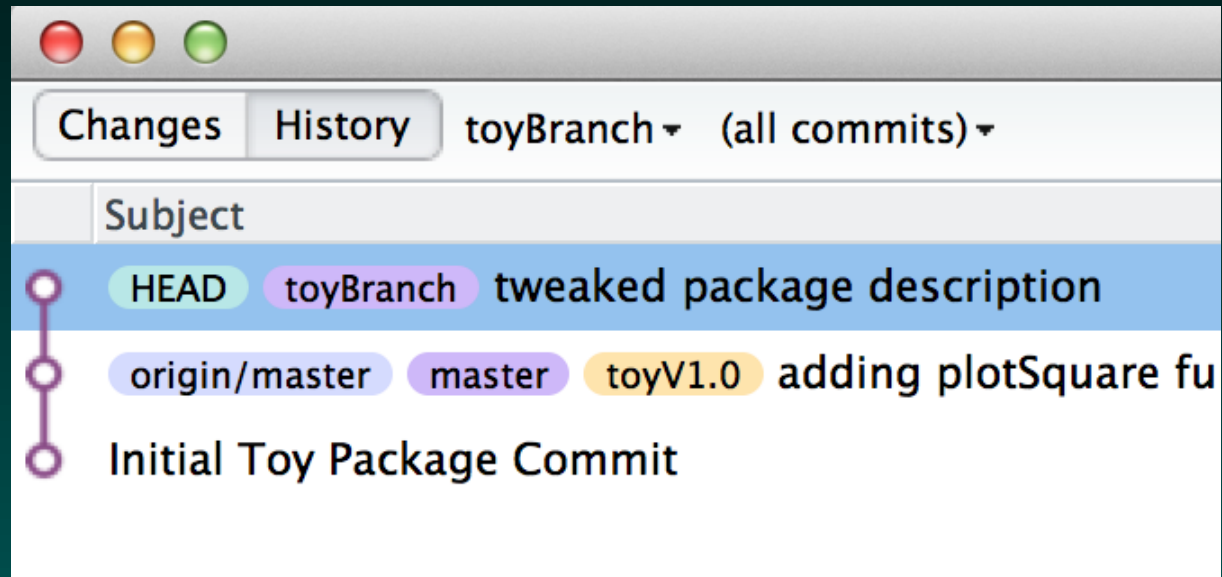
We can shift to a new named branch as follows:

```
> 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.



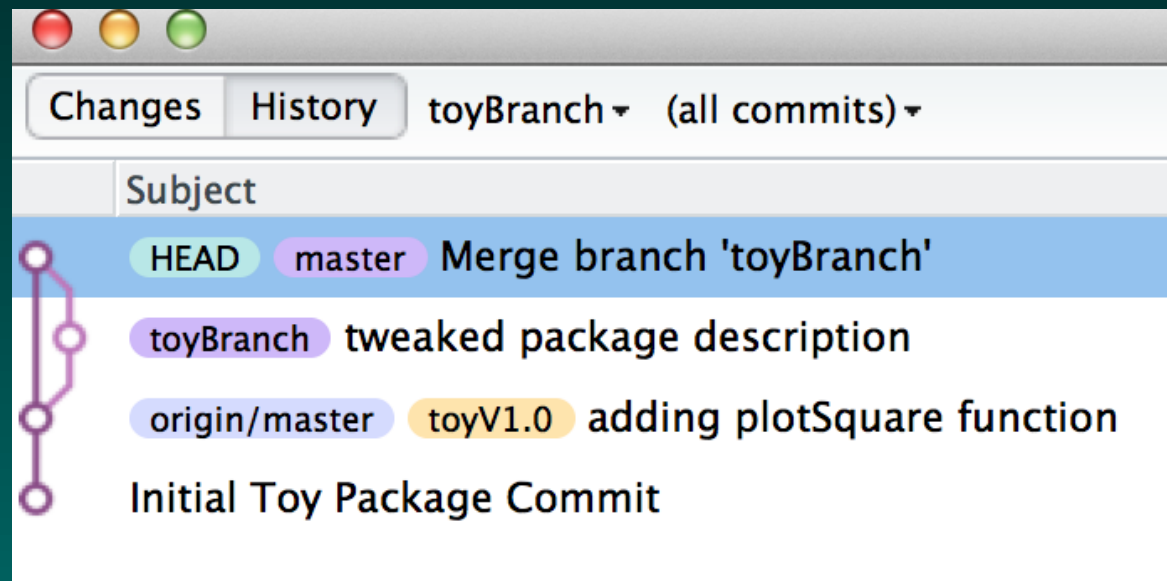
Merge in the Change

“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

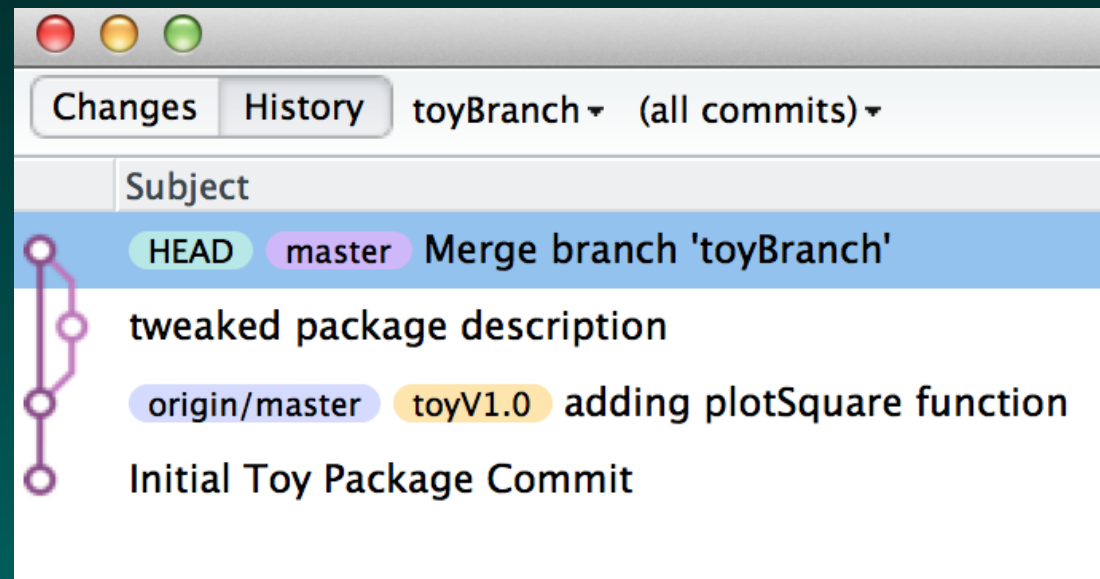


Prune the Branch

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).



Learning More

Git will require practice, and more examples than we've provided here. Fortunately, there are many good examples on the web.

Scott Chacon's [Pro Git](#), also in [book form](#)

[Ry's Git Tutorial](#), also in [e-book form](#)

Jon Loeliger, [Version Control with Git, 2e](#) (Amazon)

[GitHub Help](#)

How We Can Use Git

ok, we've introduced collaboration via GitHub, and we've tried to illustrate how Rstudio (or other graphical interfaces) can clarify or help with some steps.

So, what can you apply git to?

grant applications (e.g., references, multiple drafts)

homework submissions (precise timestamps!)

editing/commenting on reports

and lots of other stuff!