Git/GitHub 1: Sharing and RR
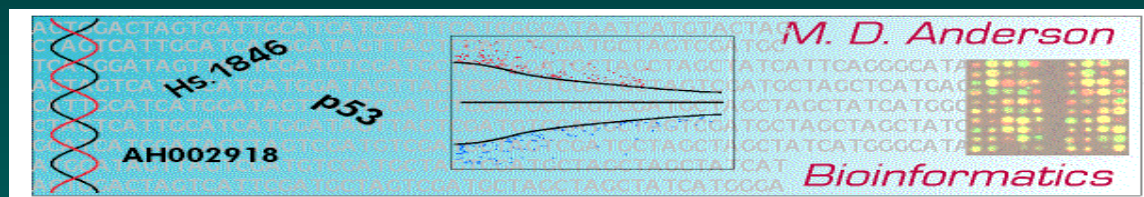
Keith A. Baggerly
Bioinformatics and Computational Biology
UT M. D. Anderson Cancer Center
kabagg@mdanderson.org

SISBID, July 17, 2018

Credits

Much of what follows is strongly influenced by Jenny Bryan's:

Happy Git and GitHub for the useR

Excuse me, do you have a moment to talk about version control? from the American Statistician's Data Science issue

What They Forgot to Teach You About R
(a short course from RStudio 2018)

What Are Git and GitHub?

Git is a distributed version control system (VCS).

A VCS lets us track the evolution of files over time, so we can avoid keeping multiple copies of files with suffixes like
V2,
V3,
V4Final,
V5FinalReally,
and so on.

A collection of files (a folder) under version control with git is a repository (a repo).

Git History and Development

Git was initially developed as a command-line tool for the programming community by Linus Torvalds (of Linux fame).

The Linux kernel, which has been updated and contributed to by a large group of programmers over several years, is a prominent example of the successful use of Git.

Using git requires some familiarity with commands like
git init,
git add,
git commit,
and so on.

Installing Git

We're not going to spend time on this. There are good instructions out there. For example:

```
https://git-scm.com/book/en/v2/
Getting-Started-Installing-Git
```

Once git is installed, there are two things to set up right away:

```
git config --global user.name "Keith Baggerly"
git config --global user.email kabagg@gmail.com
```

Please check that your version of git is fairly recent.
I'm working with 2.15.2 (the latest is 2.18.0)

What Is GitHub?

GitHub is an aggregator of Git repos (a repo of repos) which makes them easier to find, access, and share.
GitHub is the largest and best known such aggregator.

Public hosting of open source repos is free.
Restricted access (private) repos can involve fees.

GitHub also provides a great deal of value added.

Markdown is automatically rendered as html.
README.md is rendered as the repo welcome page.

GUI versions of several common commands (e.g. diff), and views are also provided.

RStudio, Git, and GitHub

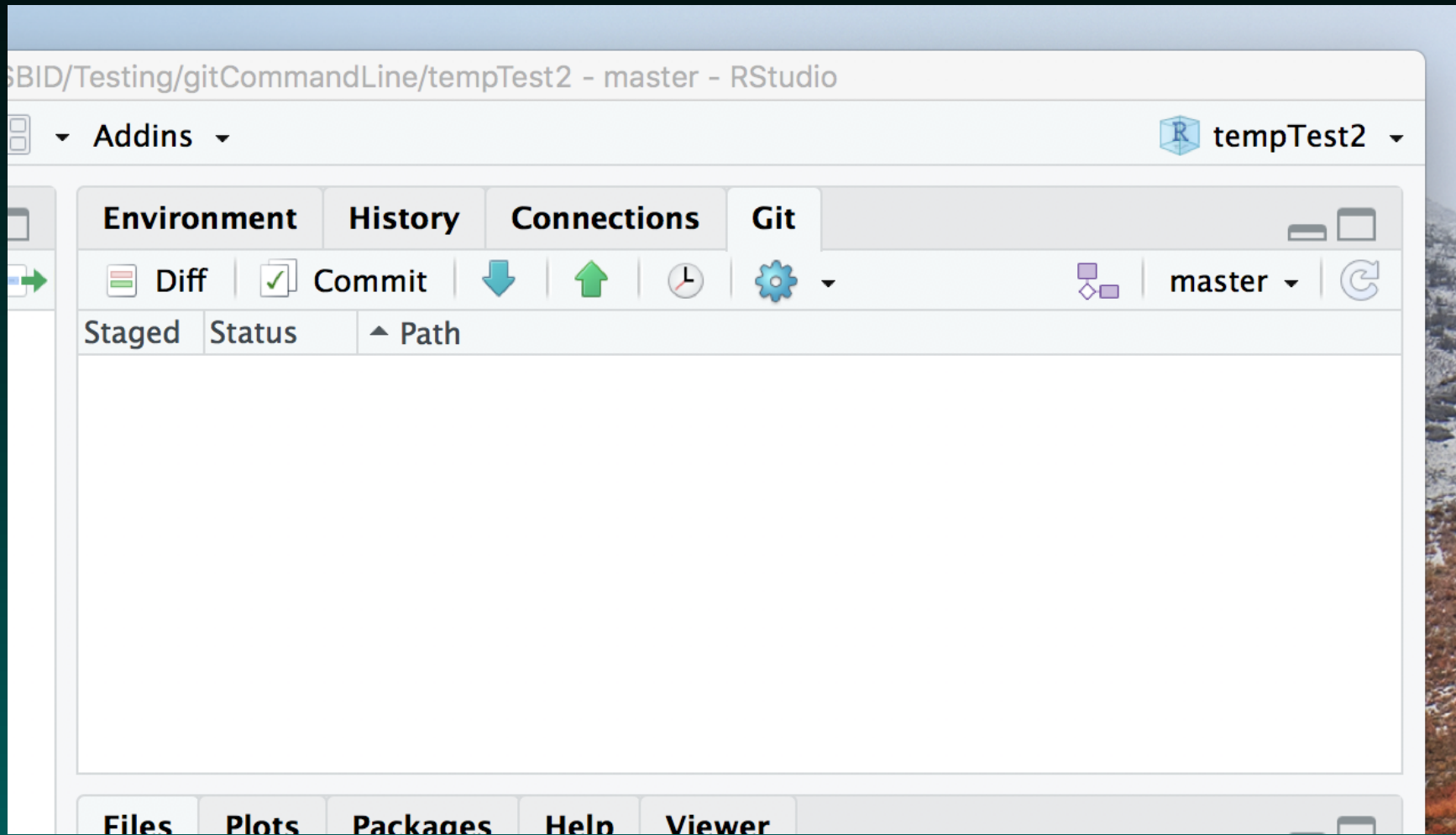RStudio has been designed to work well with Git and GitHub.

RStudio incorporates several displays for dealing with the
mechanics of Git, so the most common operations have
button click shortcuts. These include
the Git Pane, and
the Git History popup window.

Syncing between local repos on your machine and matched
repos on GitHub is pretty straightforward.

These tools let us work publicly in spirit.

The Git Pane

The Git History

Working Publicly in Spirit

How I thought of my goals in grad school:

Idea — Preliminary results — Draft manuscript — Completed manuscript — Published paper

Less valuable → More valuable

How I should have been thinking of them:

Anything still on your computer
(Data, code, results, draft, finished paper)

Anything out in the world
(Paper, preprint, product, blog post, open source, tweet)

Less valuable → More valuable

From David Robinson's keynote at eUSR 2017 (27m01s)

The Plan

I'm going to demo how I use the RStudio/Git/GitHub combo in new analyses. (Ok, how I try to use it.)

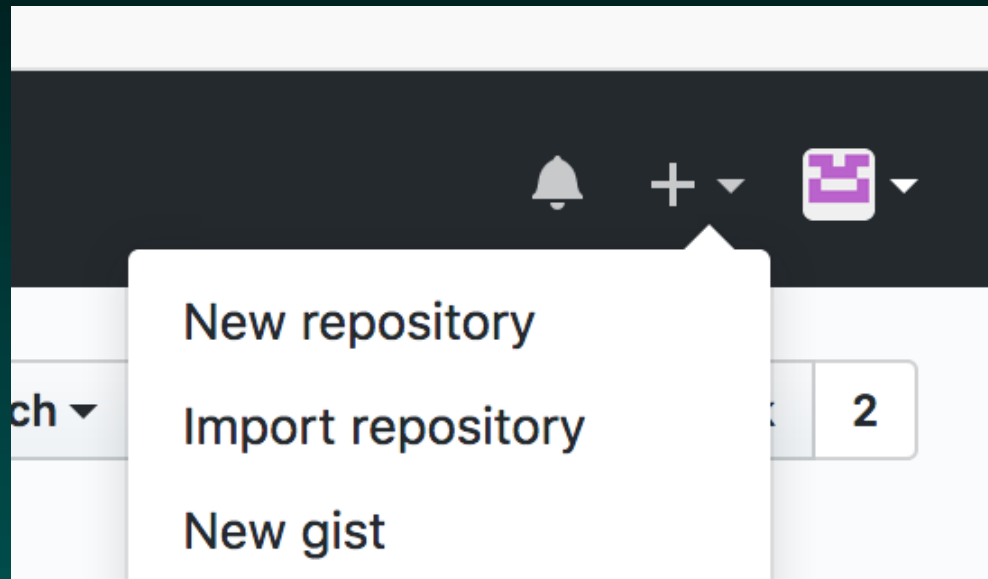Next, we'll "look under the hood" at the command line. This will (hopefully) let us modify things if desired.

Most of this will be live, with toy examples.

Some bits will involve poking around in more complex pre-assembled examples.

We begin by assuming we've just met for initial discussions about the project.

Step 1: Set up a GitHub Repo

Your GitHub account page has a button in the upper right for creating a new repo.

We'll Leave the Repo Empty
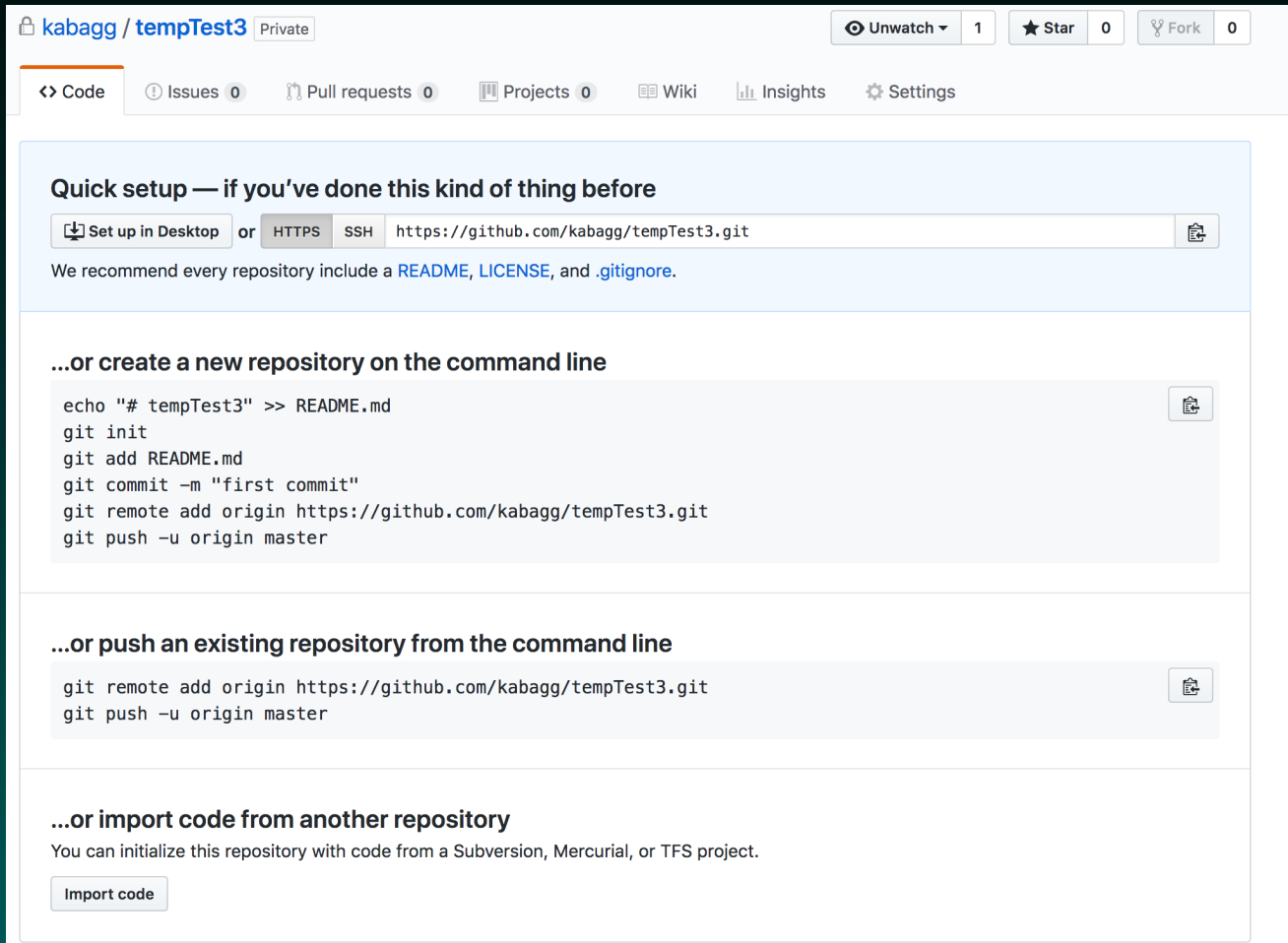
GitHub recommends quickly setting up
README, .gitignore, and LICENSE.

Prompts for Linking the New Repo

Most of these prompts assume we're working on the command line.

Step 2: Start an RStudio Project from GitHub
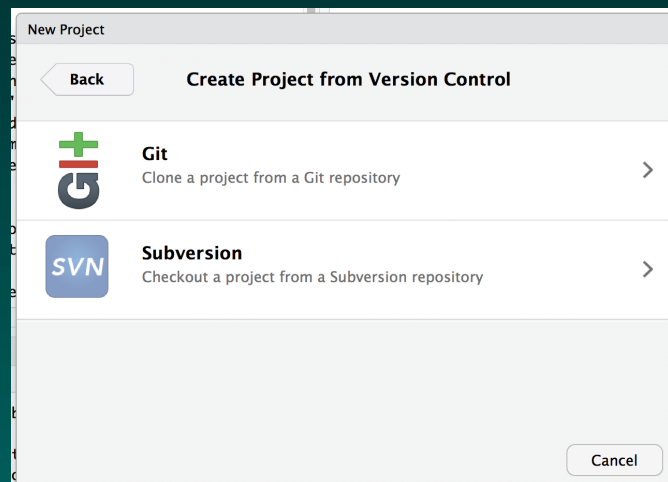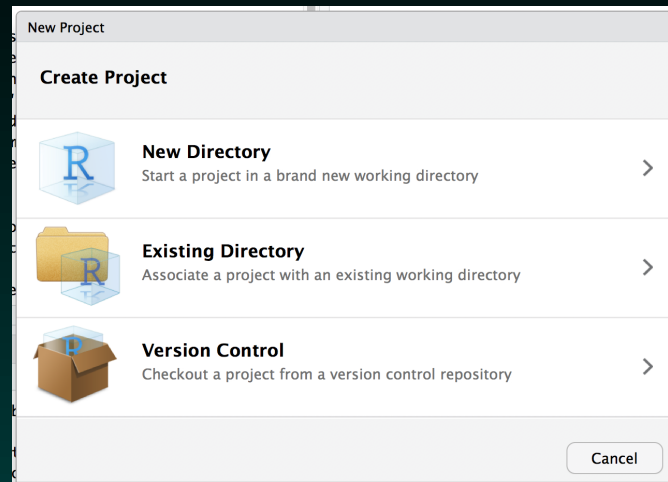
Link the Project to GitHub

Use the URL GitHub just supplied

What's Here?

The Git Pane

Files (with ??)

.gitignore

.git/ (viewable from Terminal with ls -a)

Step 3: Add and Commit

the commit message

the initial commit and the establishment of master

local master is ahead of github by one commit

the empty git pane

Step 4: Add a README

Work in RMarkdown

set output to "github‗document"

outline the nature of the project

add other text relevant to a README (we'll come back to this)

knit to generate markdown output

Step 5: Push Your Local Repo to GitHub

refresh the GitHub page

Does the README describe what you're trying to do?

local and remote are in sync

send the URL to other folks involved so they can confirm the accuracy of the description

invite "collaborators" from GitHub as desired
(giving them permission to add to the repo directly)

What Have We Done?

We've established a web presence we control for our project.

This web page can serve as a persistent, findable (searchable) location for reports going forward.

The web page is easy to update from our local machine.

The version controlled nature means others can see how the project evolved.

We've started the project with the intent of producing something shareable. This is the right mindset.

(Pull,) Add, Commit, Push, Iterate

For many small projects, the above process may suffice.

As we introduce new files or edit old ones, we repeatedly iterate through this cycle:

add new/updated files to the repo
commit the changes
push the changes up to GitHub

If you're working with others, you may want to add
pull at the beginning of the cycle to make sure you're working with the latest version of master!

What Have We Done Under the Hood?

On the command line

git clone gives a local repo linked to the one on GitHub
git add stages files for addition to the repo
git commit updates the repo, and defines the master branch
the first time it's called
git push lets us shove our updates to GitHub

there are some variations:

git init establishes a repo without a link
git branch can link an unlinked local repo to our GitHub repo,
identifying the latter as a remote branch called origin
Let's walk through this variant...

After Setting Up a New GitHub Repo

```
git clone [GITHUB URL HERE], e.g. "temp1.git"
cd temp1
ls -a
.  ..  .git
```

Define a new RStudio project in temp1.

This adds
.gitignore and
temp1.RProj

Add and Commit

From the terminal window:

```
git add .gitignore
git add temp1.Rproj
git commit -m "initial commit"
```

Refreshing the Git pane after each step shows what's going on after the fact.

Each commit needs a message.

Add a README

Generate the Rmd file in RStudio, and compile it to md

```
git add README*
git commit -m "added basic README"
git push
```

Refreshing over at GitHub shows we've updated things.

Other Ways?

```
mkdir temp2
cd temp2
git init
git add stuff.txt
git commit -m "initial commit"
```

set up a new GitHub repo for temp2

```
git remote add origin [GITHUB URL]
```

Or, from within RStudio

```
usethis::use_github()
```

Other Implied Functionality

git status shows which files have been modified and/or staged for the next commit (the Git Pane shows this)

git log shows us the previous meassages and commit ids (secure hashes, or SHAs).
These are viewable from the Git History page.

git diff shows us what files have changed and how between commits. Again, these are visible from the Git History page.

We can also see
the tree structure,
who made the commits,
and when.

Improving Reproducibility

Set up your preferred directory structure
(e.g., R/, data/, results/).

Add, commit, push.
The GitHub repo will not add folders with no files!

Introduce your first few analysis scripts/files.

My first few are almost always

R/01_gather_data.R
R/95_make_clean.R
R/99_make_all.R

The latter two may be replaced with a Makefile.

Gathering Data

For me, this (hopefully) involves

specifying URLs where the raw data can be found
downloading data from the URLs to data/

Are the raw data in a public repository (e.g., GEO)?
Can we put the data in such a repository?
Can we put the data in an accessible site behind our firewall?
Does everyone agree this is the data we're working with?

In general, I don't put the raw data on GitHub.

Exceptions: small (100Kb?), publicly shareable datasets.

Rendering R files with RMarkdown

Preface text lines and YAML with roxygen2 comments, e.g.

```
#' ---
#' output: github_document
#' ---
```

rmarkdown::render works with either R or Rmd files

The Body of "make_all.R"

Loop through a vector of R and/or Rmd files in R/.

```r
for(i1 in 1:length(files_in_r_to_run)){

    rmarkdown::render(here("R",
         files_in_r_to_run[i1]),
      output_format =
        github_document(html_preview = TRUE,
          toc = TRUE),
      output_dir = here("results"))


}
```

Update .gitignore

Git and GitHub work best with files which are plain text with minimal markup: Rmd, md, .csv, .txt

They don't work well with binary files or files with lots of markup: docx, jpg, html

Add most of the latter to .gitignore.
Then they never get tracked or posted.

I may explicitly post a few files when they're especially useful (e.g., course_notes.docx).

pdfs and pngs are mixed - GitHub will now render these!

Update the README!

As new scripts/reports are added,

Add a Results section.

Insert brief (one or two line) descriptions of the main results of each to the README, along with a link to the md file output.

Keep a list (in alphabetic order) of R Packages used.

Add a Workflow section.

Point readers to the Makefile or corresponding script they need to run to get everything to work.

Some Example READMEs

Jenny Bryan's Packages Report Example

One from another short course I recently taught, GCC 2018

What are your favorites?

Check Reproducibility and Completeness

Create a new (temporary) R Project.

Link it to the same GitHub repo you've been using.

Pull down the files.

Follow the workflow described in the README, and run the files indicated.

Do you get the results reported and described?

Why Is This Test Meaningful?

How does the GitHub repo differ from your local repo?

Everything in the GitHub repo has been committed.

There's a separation between what you're doing now and what you did in previous sessions (e.g., loading libraries).

You can (intentionally!) position the new project so any accidentally included absolute paths will break.

You can check numbers for stability. (did you fix seeds?)

This approximates someone else reproducing your work.

Include the GitHub URL in reports you send out.