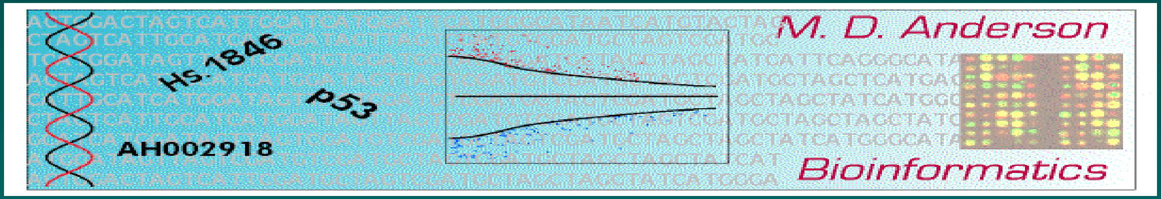
Reproducible Research with R and RStudio: Infrastructure

Keith A. Baggerly
Bioinformatics and Computational Biology
kabagg@gmail.com

FDA, Nov 16, 2018



What We Did First

What We Did First

Assume a mindset - we would make it possible for others to reproduce what we did

Adopt literate programming

Keep text describing what we intend to do next to code showing how we did it

Use Sweave

Adopt a common report structure

Assemble template reports for common tasks

Impose second person review

Things I'd Add/Change Today

Use Markdown/RMarkdown

Work publicly in spirit

Keep everything together in a folder (project, use here)

Use a consistent folder structure and workflow

Write a README for the project

Put raw data in/pull raw data from repositories

Use nice filenames

Remember prose and audience when reporting

Bundle scripts and templates in packages

Markdown/RMarkdown

Markdown

“Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).”

“The idea is that
a Markdown-formatted document should be publishable as-is, as plain text, without looking like it’s been marked up with tags or formatting instructions.”

– *John Gruber*, creator of Markdown

Markdown is Pretty Simple

Text based, like (and somewhat derived from) email

It lets you focus on writing, not formatting

Aside: working with HTML really brings home how much of printing deals with placement of items on a page.

Markdown can be filtered/processed to pretty painlessly produce output in several formats (e.g., html, pdf, docx)

RMarkdown

Lets us include code chunks which will be evaluated in-place (mostly R, but also allows for other languages, e.g. Python)

Lets us include figures

Is built into RStudio along with pandoc

Is *much* easier than LaTeX-based *Sweave* for a novice

This last was the “killer app” aspect that made us push it strongly in our department

Let’s test it out...

RMarkdown Demo Concepts

It's much easier to show most of this live.
I'm listing topics to make sure I hit them.

Please test along with me!

RStudio's Markdown help

The YAML header

Code in YAML (e.g., dates)

Options I use a lot:
toc: true,
number_sections: true,
code_folding: hide

Section headings, subheadings, etc

Demo Concepts Contd

Links and URLs

Including figures

Emphasis, italics, and bold

Inline code

Code font

Block quotes

Bullet Lists

Numbered Lists

Code Chunk Concepts

Name your code chunks! (see 04_parse_geo_data.Rmd)

Some commonly used chunk options:

eval=FALSE, warning=FALSE, message=FALSE

echo=FALSE

fig.width, fig.height

cache=TRUE

auto-completion!

Different Output Formats

html (default) - html_document

pdf (requires LaTeX) - pdf_document

docx - word_document

md (markdown) - md_document, github_document

Slides (beamer)

PowerPoint (requires recent version of pandoc)

Note relative file sizes...

Interleaving R and Python

This exploits the reticulate package

Assign in python: x = 5
Access in R: py$x

Assign in R: y <- 3
Access in python: print(r.y)

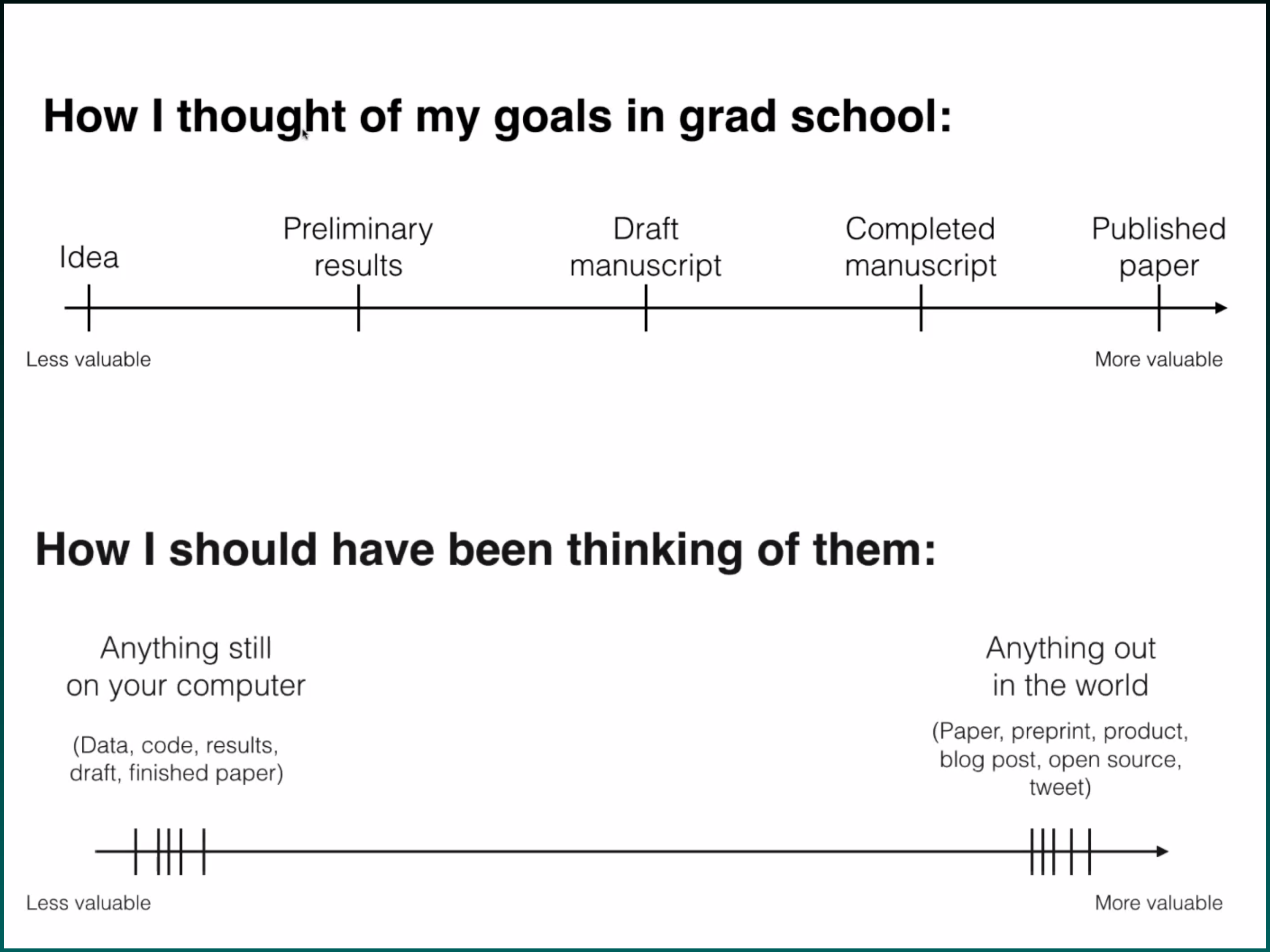
There's much more!

Going from one to the other isn't too hard.

You can see more in the webinar, with slides on github

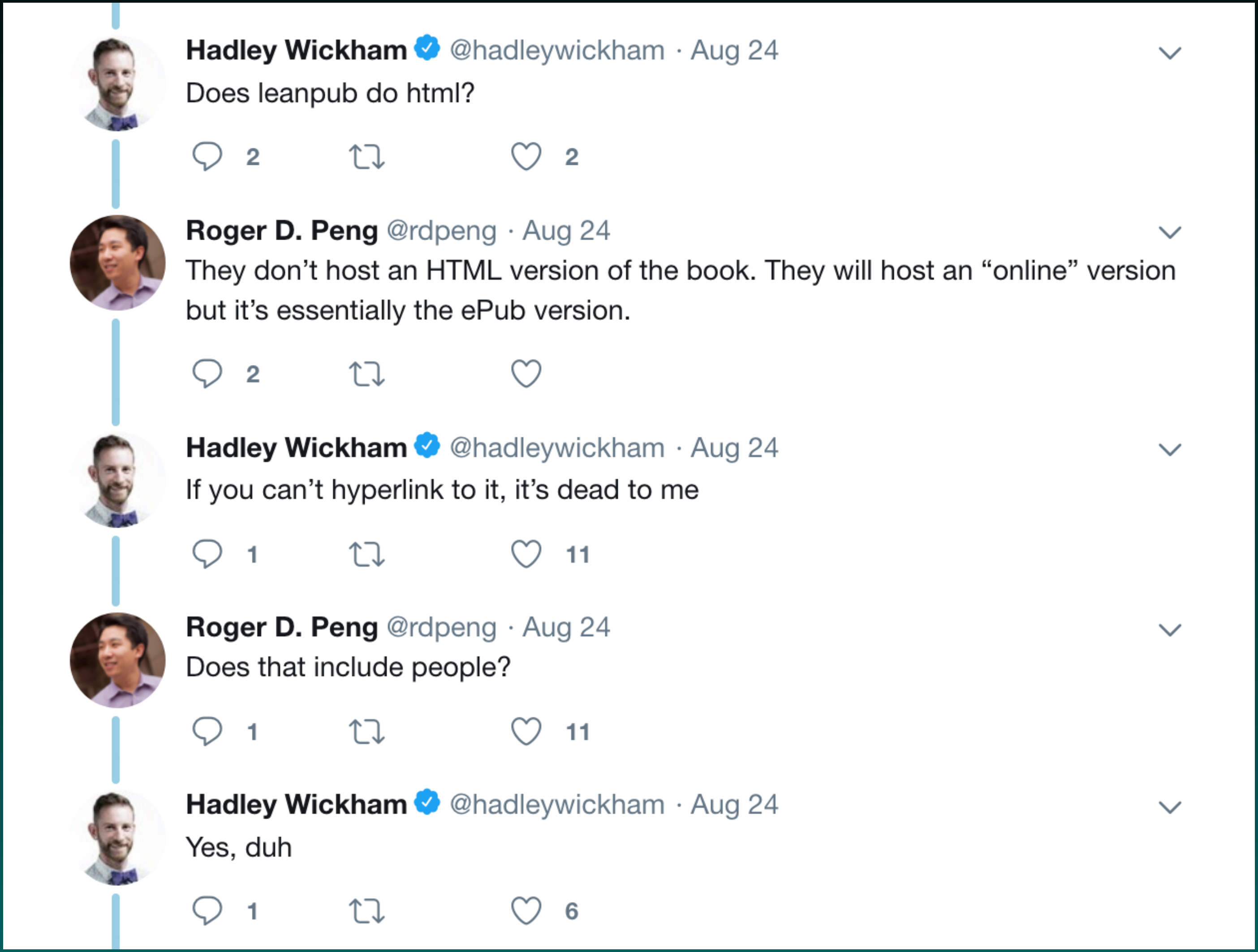
Work Publicly In Spirit

Work 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)

Similar Points from Hadley and Roger



Hadley Wickham @hadleywickham · Aug 24
Does leanpub do html?
2 2

Roger D. Peng @rdpeng · Aug 24
They don't host an HTML version of the book. They will host an "online" version but it's essentially the ePub version.
2

Hadley Wickham @hadleywickham · Aug 24
If you can't hyperlink to it, it's dead to me
1 11

Roger D. Peng @rdpeng · Aug 24
Does that include people?
1 11

Hadley Wickham @hadleywickham · Aug 24
Yes, duh
1 6

Discussing publishing

Make Key Steps Shareable Early

Post of the web or put things on a shared filesystem you can ideally send collaborators links to.

For someone else to look at what I've done, they need to be able to find it (and I prefer not emailing lots of attachments)

Planning to share reinforces the mindset

Sanity self-check: If I start from just what I've posted, could I get what I want?

Would someone else find this clear?

This can lead to early discussion of where things should be found

Why "In Spirit"?

If you're working with certain types of data (e.g., patient data with HIPAA constraints),
there may be limits on what can be placed out in the wild.

It's often possible to set up instantiations of locally shareable repositories (e.g., gitlab) within a department or institution which can allow for greater access control and authentication.

The mindset: There is a place, outside my computer and potentially accessible by others, where the main part of my analysis will be consistently findable. Once I've shared this with others, I won't modify it in an unrecoverable way.

How Do I Share?

First, I define a GitHub repository to store my analyses. This will generate a url.

Then, I create a new RStudio project on my local machine, and specify the project should be drawn from GitHub, using the url provided.

At this point, RStudio makes getting my analyses up on the web pretty trivial.

Jenny Bryan: Excuse me...

Jenny Bryan: Happy Git and GitHub for the useR

The project name should be clear, even years later

Using Git and GitHub (Live)

If you have an account on GitHub (free), and you create a git version controlled repository (a project), RStudio can set up a new project using a link to the repository.

This means the simplest interactions with GitHub (pushing to upload to, and pulling to download from) are button clicks.

GitHub “automatically” renders Markdown as html

GitHub will automatically render a README.md as (part of) the default home page for the repository.

GitHub and Findability

Include the GitHub repo URL in the intro of reports you write.
Your collaborators will be able to find it later.

At least as important, if they come back asking for modifications a few years(!) later, you can find it.

Every journal article does essentially this
by including the name of the journal, page numbers, and other information at the top or bottom of every page.

Why Use Projects?

Looking at a New R Project

Note the presence of “.Rproj” and “.gitignore” files

The .Rproj file defines the top (root) folder of the project.
All other files and folders can be positioned relative to .Rproj.

This type of “anchoring” is what git uses (targeting .git) to position files, and can be used in R with the “rprojroot” and “here” packages (the latter has a simpler syntax)

Anchoring lets us make the entire project “modular”.

We never have to refer to where things are relative to our own (unique) filesystem.

We want to AVOID using getwd and setwd.

Absolute, Relative, Absolutely Relative

An absolute path specifies a file’s location relative to the root of that filesystem. If that filesystem is your machine, and I try to run your code, that’ll break.

A relative path specifies a file’s location relative to where you’re working now. If you lose track of which directory you’re in, this can get frustrating.

Anchors let you have the peristence of absolute positioning relative to a contained (and shared) root.

This is how R Packages, Perl Modules, etc work.

Using here

Inside an R project, invoke

```
library(here)
here()
```

To save my_stuff.RData to results/ from an R script,

```
save(my_stuff,
  file = here("results", "my_stuff.RData"))
```

positions things correctly if your working directory is any subfolder of the project. Don't nest projects!

Loading works similarly

Jenny Bryan: here here!

Use a Consistent File Structure

Use a Consistent Folder Structure

Good Enough Guidelines

Who among us has never had a folder filled with a plethora of apparently unrelated files?

I still have plenty...

Something that'll help you and others understand the workflow (and potentially automate it) is using a consistent set of rules for putting files in common places.

This is not news.

But keep the rules simple and minimal if possible.

What I Try to Use

R/ (for .R and .Rmd files)
data/ (for raw data and metadata)
results/ (for processed data, md files)
reports/ (for shared html, docx, and pdf)
README.Rmd, .md

This is minimal, but I have a pretty clear idea where to go.

I may also include:
figures/,
LICENSE.Rmd,
TODO.Rmd,
CITATION.Rmd,
etc.

What Would You Use?

My list is not sacred.

Pick a structure you're comfortable with, but use it consistently.

For each folder name in your starting list, create that folder,
in the given position relative to the project root,
if it doesn't already exist.

Repeat this process for files you want to create *a priori*.

Avoid Complete Uniqueness

Particularly when collaborating,
it’s easier to share if we can build on existing conventions.

MS Windows

Driving on the right

Value added from interoperability

We’re not the only ones to notice this, and certainly not first.

Project Template
is a neat R package that implements a few different layouts

workflowr is another recent implementation

Write a README First

Write a README First

GitHub’s treatment enforces the idea that
starting a project with a README is good practice.

At least in outline,
what do I think the goal of the project is?
how do I intend to pursue it?
what data do I have?
what would constitute “success” or “failure”?
what don’t I understand yet?

As the project proceeds, I can add questions, links, and workflow

Sharing your README with collaborators before beginning analyses can save time!

It’s Worth Spending Time On This

For me, writing out my own understanding of the project
in a way I think my collaborators will understand
generally clarifies where the gaps in my understanding are.

If I don’t understand it well enough to describe it,
I don’t understand it well enough to analyze it.

This can also help your collaborators clarify their thinking, and prevent you spending time on analyses they don’t care about.

Some Example READMEs

A pre-assembled project of mine

Overview

Brief Results, with links

Workflow

Libraries

A simple report from Jenny Bryan

workflowr

Let’s Write a README Now

Using RMarkdown, of course!

Given the Rmd, we want to produce md for GitHub

Edit the YAML to indicate output will be a “github_document”. (md_document can work too.)

In addition to the md, this will produce a “preview version” html so you can see what your file will look like when posted.

As it happens, md is an intermediate step in producing many other types of output. Thus, we can also set “keep_md: true” for other document types in the YAML.

Gathering Data

Gather the Raw Data

We'll begin in R/ with

01_gather_raw_data.R

Ideally, this will pull the raw data from a formal repository (which is unlikely to move).

This can make your project and analysis more "lightweight", in that you don't need to include the raw data in what gets posted to GitHub or your other shared site.

This also discourages keeping many different copies of the raw data in different places.

Many copies can be frustrating, especially if files differ slightly and you're not sure which are most current.

If the Data's Not in a Repo, Change That

There should be one canonical copy.
With trackable provenance.

If I worry about the stability of the place I got it from, or I'm only getting the data from back and forth emails, one goal is to establish an agreed upon place for everyone to look for it.

I did this for one of our datasets by posting the data (and a description) to figshare. The first 20G is free...

Don't go overboard here.

If your entire dataset is fairly small (say under 1Mb), go ahead and post it, possibly compressed.

I use “downloader”

As with other steps, I really try to avoid pointing and clicking.

This means that while I’ll often record the base URL where the data can be found, I’ll also spend some time trying to find the right modification which lets me script the file download.

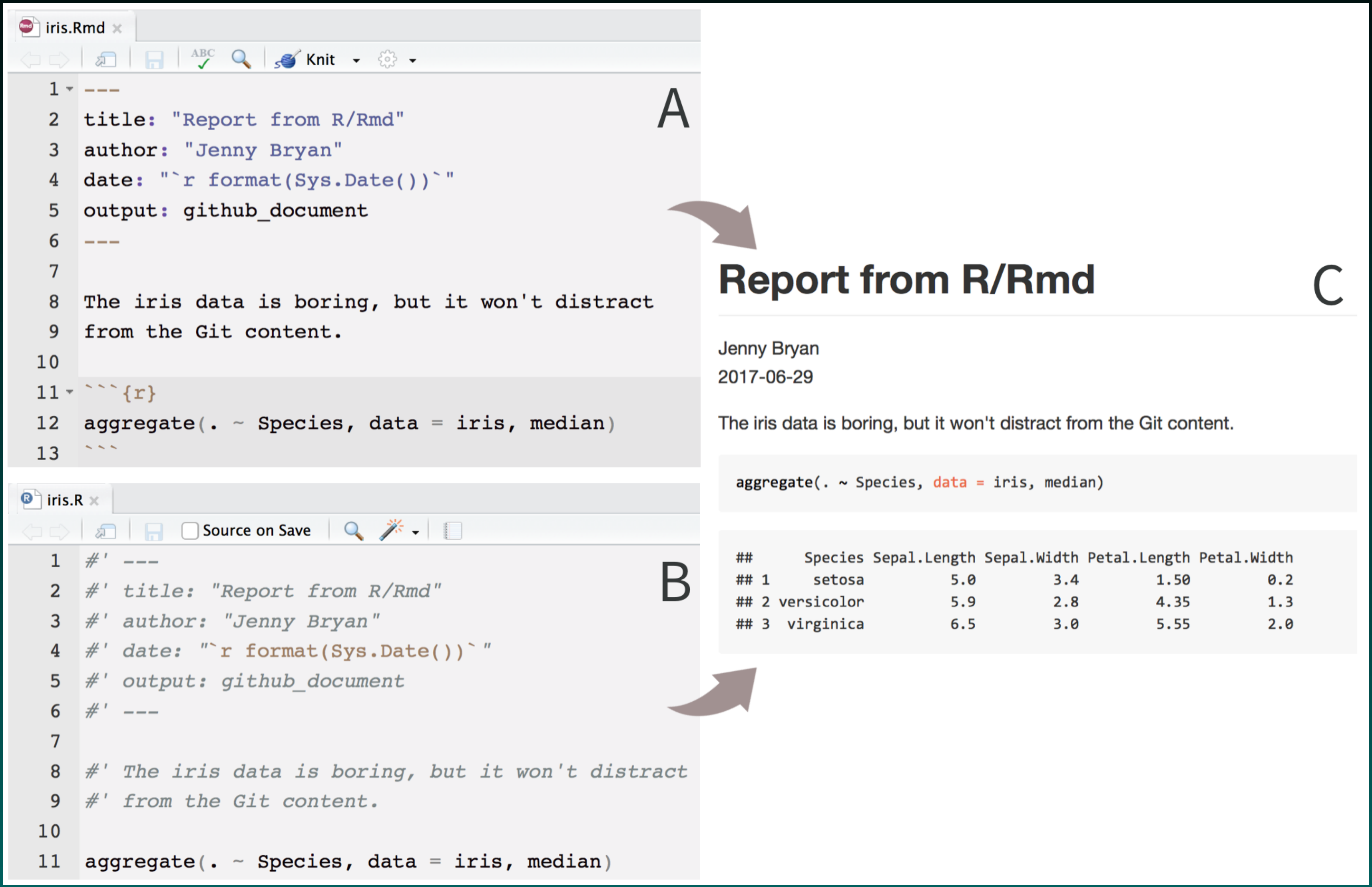
This occasionally involves poking around at the html of the top page with “View Source” in my browser’s tools.

I position the downloaded file with here.

```
download(potti_url,
  destfile = here("data", potti_data_file),
  mode = "wb")
```

But what about markdown?

R or Rmd? Use Either!



~~~
---
title: "Report from R/Rmd"
author: "Jenny Bryan"
date: "`r format(Sys.Date())`"
output: github_document
---

The iris data is boring, but it won't distract
from the Git content.

```{r}
aggregate(. ~ Species, data = iris, median)
```
~~~

```
#' ---
#' title: "Report from R/Rmd"
#' author: "Jenny Bryan"
#' date: "`r format(Sys.Date())`"
#' output: github_document
#' ---

#' The iris data is boring, but it won't distract
#' from the Git content.

aggregate(. ~ Species, data = iris, median)
```

Report from R/Rmd

Jenny Bryan

2017-06-29

The iris data is boring, but it won't distract from the Git content.

```
aggregate(. ~ Species, data = iris, median)
```

```
##      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     setosa          5.0         3.4         1.50         0.2
## 2 versicolor          5.9         2.8         4.35         1.3
## 3  virginica          6.5         3.0         5.55         2.0
```

Jenny Bryan, Excuse Me, Fig 4

Is Code or Text Primary?

We can produce md files from either R or Rmd (given YAML)!

The two types of analysis files can be used interchangeably. This is why I use one folder and one numbering scheme.

You need different comment characters, but the one for R scripts will show up again later today.

Most of the time, I try to describe what I'm doing in words at least as often as I write code, so I default to Rmd files.

If I'm writing more than a few stand-alone functions (un-numbered .R files) I try to put these in an R package.

Don’t Repeat Yourself (DRY) and Looping

My first version of “gather data” had a lot commented out.

Most of these were my initial attempts to download data files one at a time until I got things to work (worth doing!)

But, as I did it, it became apparent
I was using essentially the same code more than twice.

That’s a sign I should be trying to condense things by looping or writing a function.

Condensing makes the final code more consistent and maintainable.

Conventions

On Naming Files

From Jenny Bryan's Speakerdeck slides on Naming Things

Three principles:

machine readable (parseable with regular expressions)

human readable (we can guess what it does)

plays well with default ordering
(if we run all files in a folder in sequence, will we be ok?)

Numbers at left (with zero padding!) for sequencing

Dates use YYYY-MM-DD format

On Writing Code

Adopt specific coding conventions.

Style is important. It will affect how readily people will be able to read and understand your code, and their willingness to try.

As with directory structures, picking one set of conventions and using it consistently is more important than which one you choose.

That said, in the R space, there's now one set that has a good deal of momentum.

Tidyverse Conventions

There is a tidyverse style guide.

The tidyverse is popular. Following these rules expands the set of people who'll find it easier to read your code.

There are also two R packages, styler and lintr, which will automatically check selected blocks of code for conformity with the style guide.

Style covers things like indentation, spacing, line wrapping, variable naming, etc.

Some Ways this Helps

How do we split words in filenames?

read_table vs read.table

R's history is somewhat mixed, so it used to be you had to guess a fair amount. R evolved via academic anarchy, and might benefit from more top-down structure

Another example: Piping and command chaining

```
tidy_df <-
  tidy_df %>%
  operator(other_args)
```

It makes code easier to read

It allows for greater code reuse

Foo. Foo? Foo! (no Bar)

A programmer meets a children's story...

```
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)

bop(scoop(hop(foo_foo, through = forest),
          up = field_mice), on = head)

foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mice) %>%
  bop(on = head)
```

From R4DS!

Workflows, Make, and R

Make All

As we start assembling R and Rmd files, we want to be able to run all of them in our preferred order.

Explicitly specifying the order clarifies the workflow, and (ideally) the dependencies involved.

Pulling files together in a specified order is an old problem. One tool for doing this in great generality is Gnu Make.

But.

Make has a somewhat arcane syntax. If we're just using R, we can just code the most common options.

I implement one in 99_make_all.R.

rmarkdown::render

As with downloading files, we want to get away from button clicking as we produce reports (even if the knitr icon is cute).

We do this by invoking “render” and looping over the files to be run.

```
rmarkdown::render(
  here("R", files_in_r_to_run[i1]),
  output_format =
    github_document(html_preview = TRUE,
      toc = TRUE),
  envir = new.env())
```

Again, we control locations with here. The github_document function is also in rmarkdown.

Make Clean

Aside from make all, the other most common option is make_clean.

95_make_clean.R gets rid of intermediate or derived files, both to clean up directories and to ensure that when you next run 99_make_all.R, there isn't some undocumented "carryover".

Given the directory structure I'm using, I'm emptying out data/ and results/, but leaving R/ alone.

At intermediate stages, I may not clean everything every time I add one file. For example, I often don't clean out data/ if I'm sure the download step works.

Parsing and Cleaning Data

Loading.

Visually Skimming.

Organizing/Parsing/Arranging.

Basic EDA.

Describing the Data

I'm now shifting to 02_parsing_potti_data.Rmd.

Some repositories contain metadata about the raw data (e.g., data dictionaries, sample run dates, array types, etc).

If these exist as separate files (ideal), these should also be downloaded into data/.

If the metadata do not exist in such form, they should be briefly summarized in a separate note in this Rmd file.

A data dictionary by any other name...

Loading and Viewing the Data

I also use this step to make comments about the data I see in the files supplied, before doing any processing.

If the data are tables, I often use View.

Do I understand what the measurements mean?
Do I see clearly what samples are to be contrasted?
Do the dimensions make sense?

Simply staring at the numbers for a few minutes often highlights additional questions I want to ask.

Are there ties? Missing values? Thresholded values?

Cleaning the Raw Data

There's too much to this to cover it in complete generality (sorry).

That said, we want to define what constitutes clean data for our project.

Then we want to load and process the raw data to put it into this clean form.

As we do this, there are basically 3 rules.

The 3 Rules

1. treat “data/” as read-only (Jenny Bryan)
2. don’t copy and paste
3. script everything (Karl Broman)

These rules are definitely somewhat redundant.

They’re there because there should be one canonical set of raw data, and preprocessing often has as much of an effect as later analysis choices.

Save the Cleaned Data in results/

Putting these files in results/ as opposed to data/ emphasizes that these are derived quantities.

There are two things I waffle about.

1. I've saved the processed data in RData form. I might want to save the data in some type of plain text format in case someone else wants to analyze it differently, but I haven't.

2. I haven't numbered the saved files to indicate which report generated them. This is something I may change later; I'm still mulling this one. Thoughts?

Making Data Tidy?

Hadley Wickham has made a good case for assembling the data to be analyzed in a consistent, tidy, form.

In a tidy dataset, each row is an observation, each column is a variable, and each cell is a value. In the tidyverse, such data frames have been reformatted (slightly) as tibbles with cleaner display syntax and no rownames.

Being able to assume a common data format lets code be written far more generalizably, and allows for “piping” using the magrittr package, which can make code more readable.

I definitely recommend reading R for Data Science, which desribes this in more detail.

I Made These Data Tidy (This Time)

I didn't in other short courses

I have no objection to doing so.

I'd welcome any recasting of the preprocessing and analyses that improved clarity further!

In this case, however, I found it useful to envision the array data in gene-by-sample matrix form.

The non-tidy arrangement let me spot something stupid which I might've missed had I tidied everything.

I'm shifting over time, but I want to better understand how to preserve my ability to check for stupidity.

Analyzing Data

Explain What You’re Trying to Do and Why

I find it far easier to read code if I’m regularly reminded of what the goal is.

Avoid long blocks (say more than half a page) of uninterrupted code.

Each block of code is performing some set of operations.

Why are you performing them?
What do you expect to see as a result?
Looking at the resulting data/figure, do you see it?
What interpretation do you give to what you see?

Who is This For?

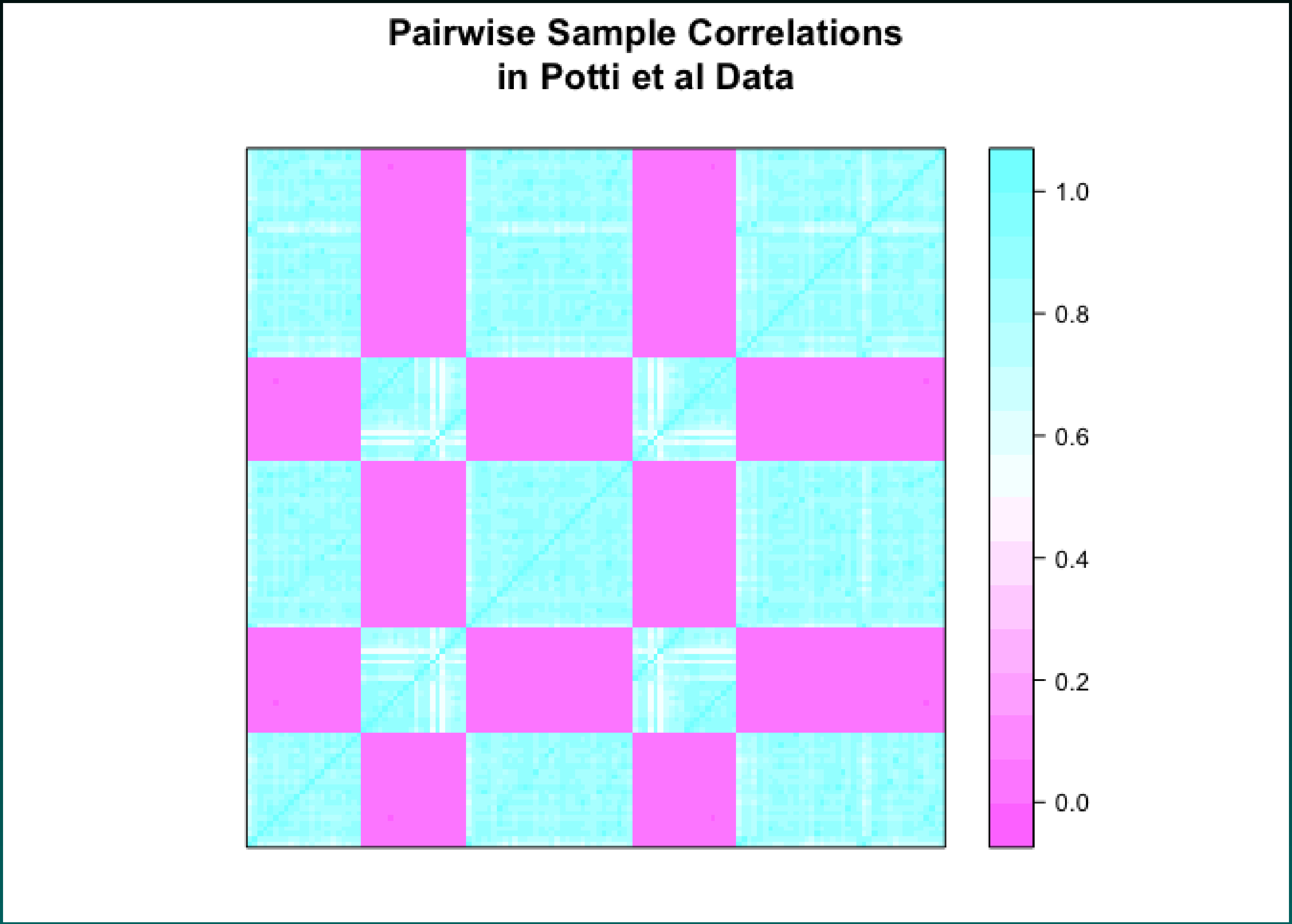
“Your worst collaborator is you from 6 months ago, and you don’t answer emails.”

– various, but I heard it from Karl Broman

Hadley Wickham describes this as writing for “future you”.

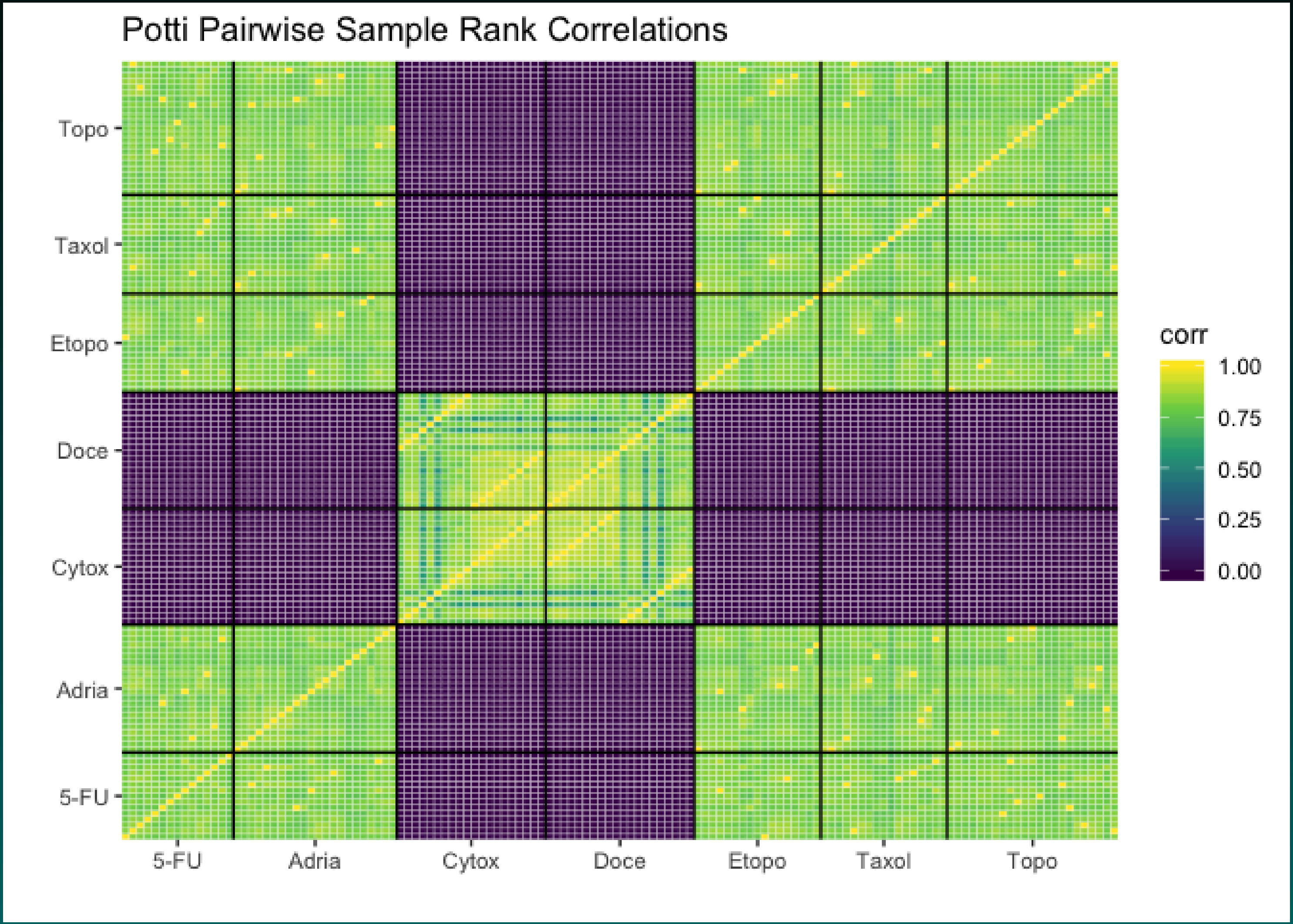
These practices, even if they take a while to become habits, can repay the time spent surprisingly quickly if you ever revisit projects.

Try to Use Figures



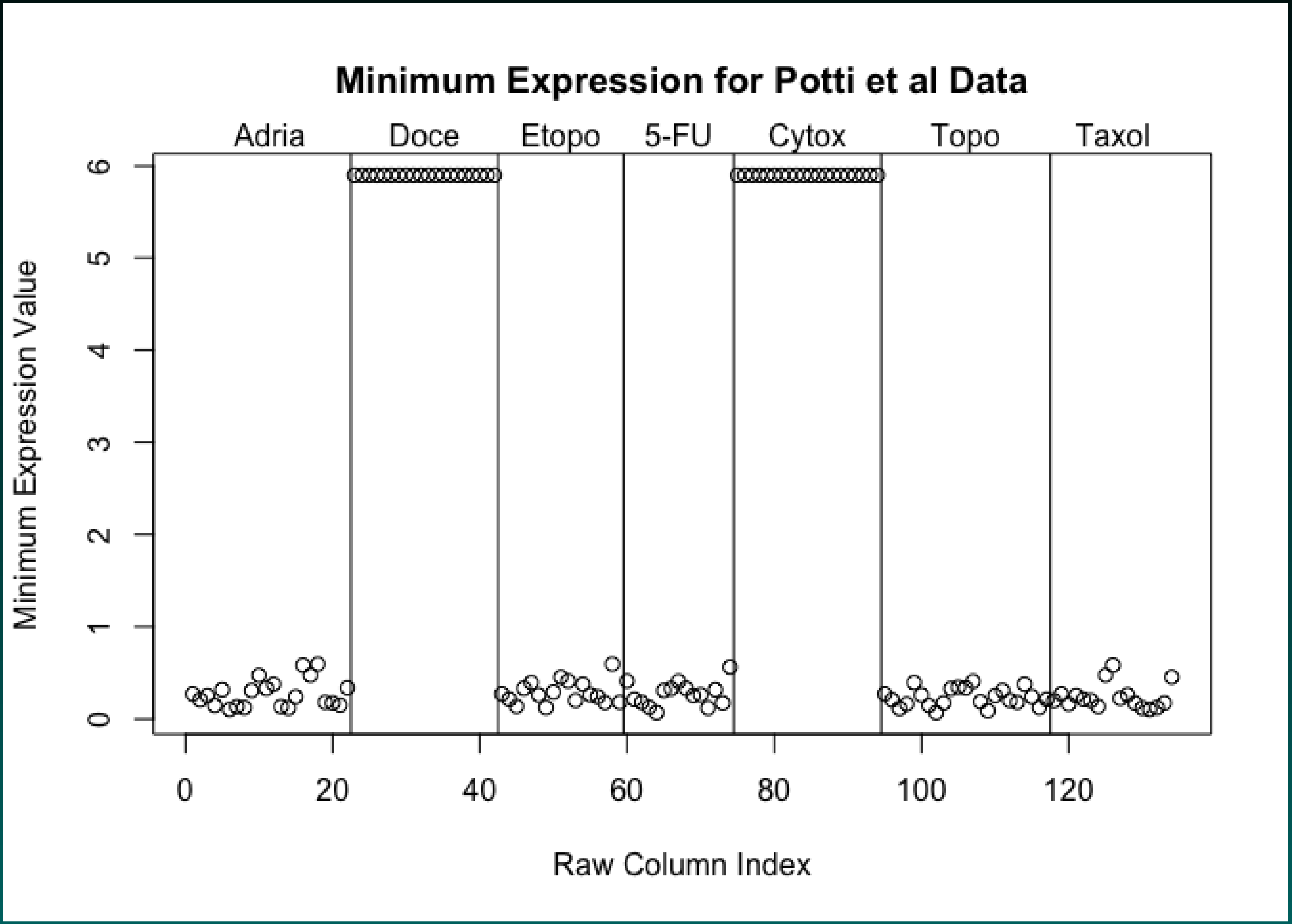
Label them. Explain what you see.

Try to Use Figures (2)



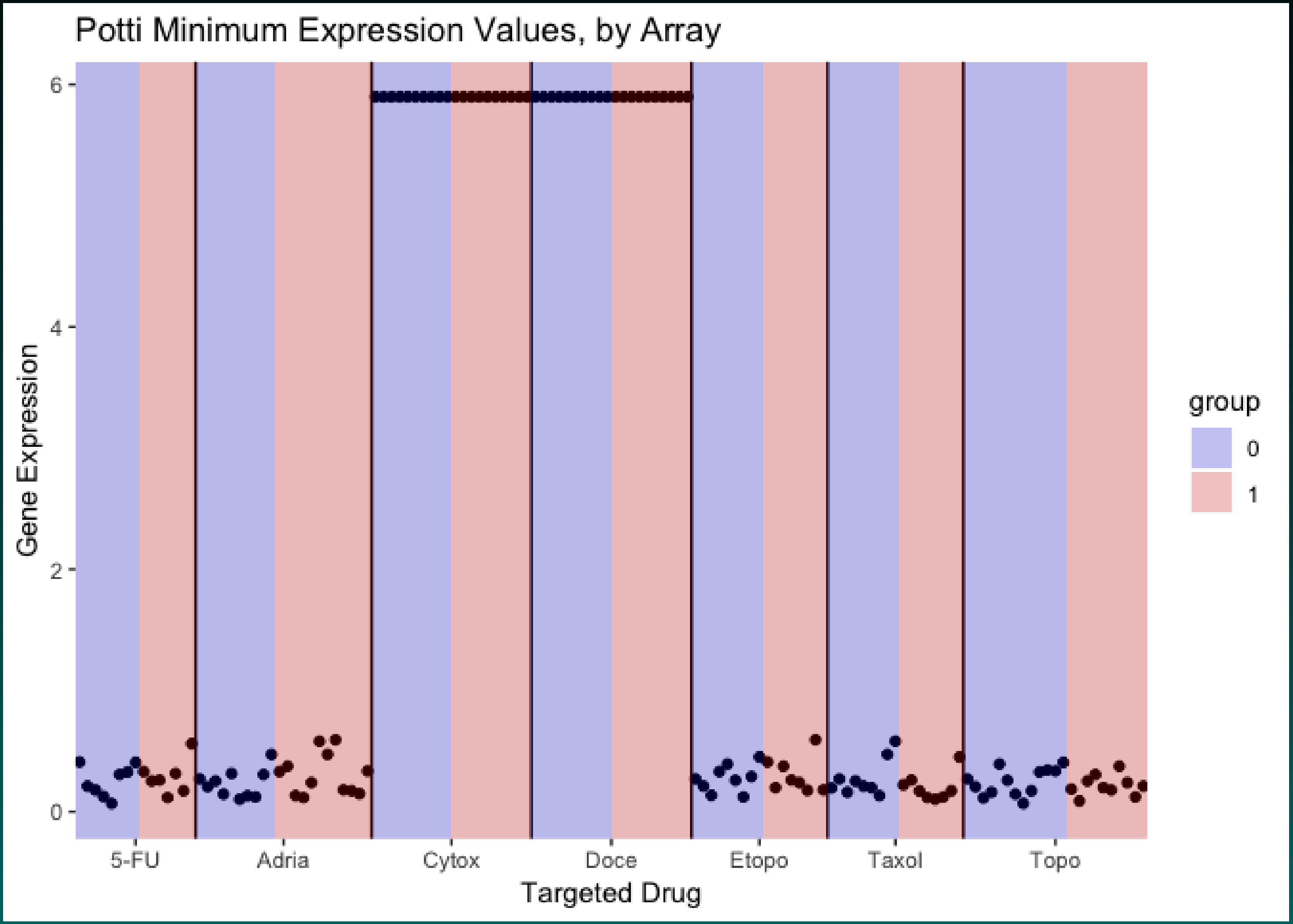
Do labels help?

Try to Use Figures (3)



Remember your audience as you explain.

Try to Use Figures (4)



This can sometimes take a while, but it’s worth it.

Figures Will Get Reused

Your pictorial summaries are the bits of your analyses that most people will focus on.

You may be asked (especially for publications) to modify your figures “just a little bit”.

If you come back to a figure more than once, consider splitting off the code generating that figure into its own standalone R script.

Reporting Analyses

What A Report Means to Me

This is a summary that will go out to my collaborators directly.

While I'm happy to make my underlying scripts and component pieces available,
this is where my audience changes a bit.

As a practical matter, this also means I often produce reports in several different output formats (html, pdf, docx)

I include a link to where things are (the GitHub URL), and to subsidiary reports as needed.

Writing Good Reports Enhances Reproducibility

I treat report writing as a separate topic in part based on negative feedback we got.

When we first implemented *Sweave* reporting in our group, our collaborators hated it.

They had no objections to reproducibility, but there was now too much emphasis on code.
They couldn't quickly find the results they cared about.

So, we restructured to meet both goals.

What Should A Report Contain?

At the outset, a report should clearly state

The underlying question we hope to address.

What experiments were performed, and how these are expected to provide an answer to the question.

What analyses are being performed in the report at hand.

The results of these analyses.

The conclusions we draw from these results, and an indication of where we should go next.

It should then present the analysis itself.

The Executive Summary: Structure

Introduction
– Background and Rationale
– Objectives
Data & Methods
– Data Description
– Statistical/Analytical Methods
Results
Conclusions

This what our clinicians are used to, and find easy to digest.

This summary should be in prose, ideally ≤ 2 pages.

Strive for Parallelism

If there are multiple objectives listed in the introduction,
address them in the same order
when discussing the conclusions.

Similarly, if tests A, B, and C are described in the methods, present the results in the same order.

Parallelism improves clarity and makes it easier to “check off” whether everything has been done.

Clarity, clarity, clarity

Questions I find myself asking report writers regularly:

Do all team members share an understanding of the goals?

Common changes I request (beyond spelling and grammar):

What do + and - mean?
Is a high value good or bad for the patient?

What do you infer from the plot you’re showing?
Describe what lets lets you do this.
What would the plot look like if there’s no structure?

What is this chunk of code meant to do?

What Sanity Checks Have You Employed?

What changes did you expect to see? Did you?

What changes shouldn’t be there if this is working?

Have you plotted the p-values from all contrasts?

Have you plotted low dimensional summaries of the data? (e.g., PCA?)

How have you plotted the data?

Do the results make sense?

Posting and Sharing

What Should You Post?

What did I put in .gitignore?

```
.Rproj.user
.Rhistory
.RData
.Ruserdata
.html
data/*
results/*.html
results/*.RData
```

I post all .R, .Rmd, and .md files by default

Minimalist

Minimally modified text files take little space.
They'll be parseable with almost any text editor.

What's derived? Is it big?

If I need to post data, have I compressed it?

Most R file reading functions can handle compressed files without modification.

I occasionally make exceptions for reports (stuff I send to collaborators) and save other output versions (e.g., docx).
I still check file sizes.

I'm ok with figures!
I may try to choose a smaller output format (e.g., png).

What’s Enough?

I’ve posted enough if
I’d be comfortable removing the folder from my own computer
because I could regenerate everything fairly quickly from
what I posted.

For the project on GitHub, rerunning the entire analysis takes
about 1.5 minutes on my laptop.

A big chunk of that is downloading the raw data from the
repositories.

Git and GitHub

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)

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)

RStudio, Git, and GitHub

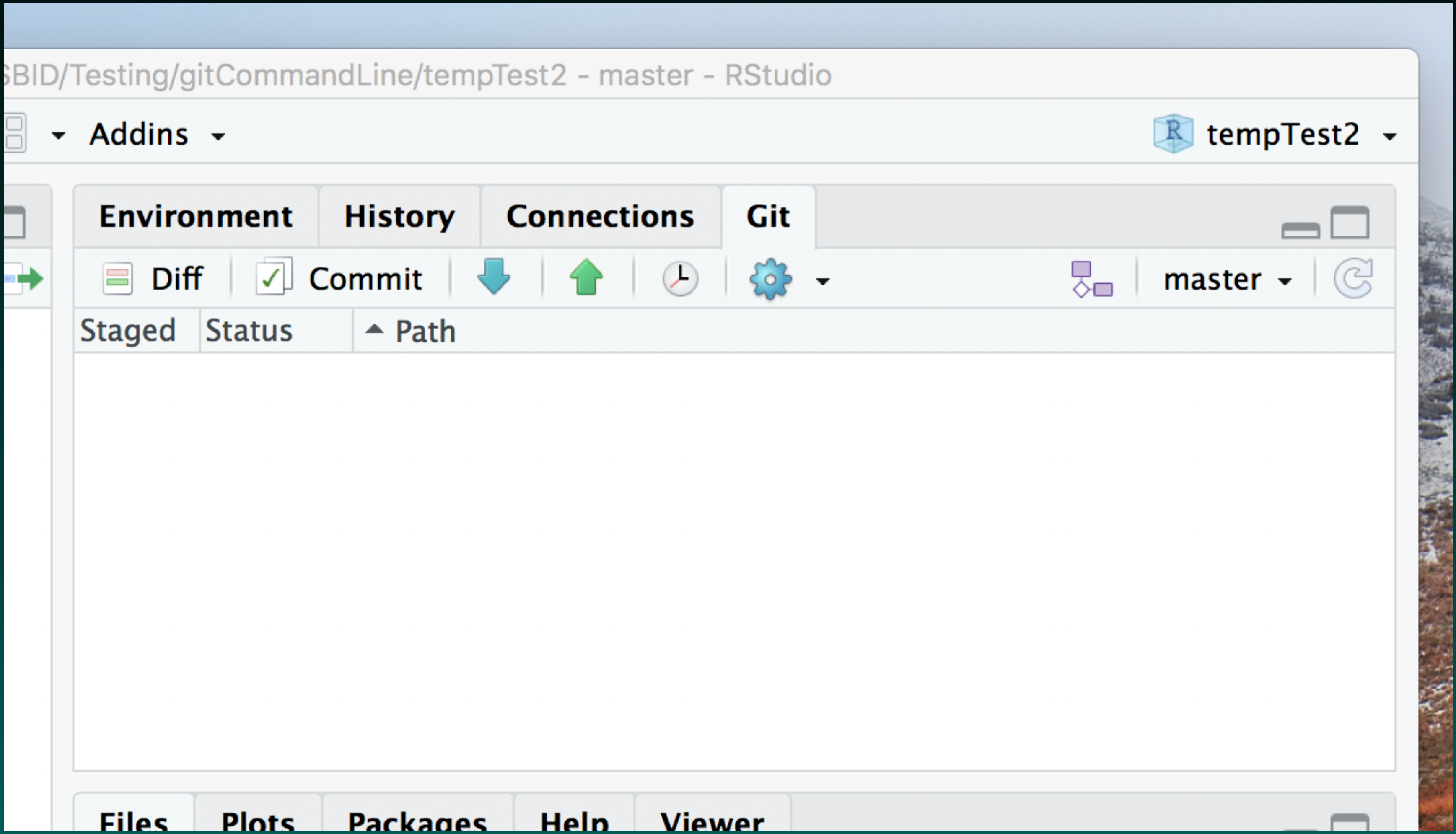
RStudio has been designed to work well with Git and GitHub.

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.

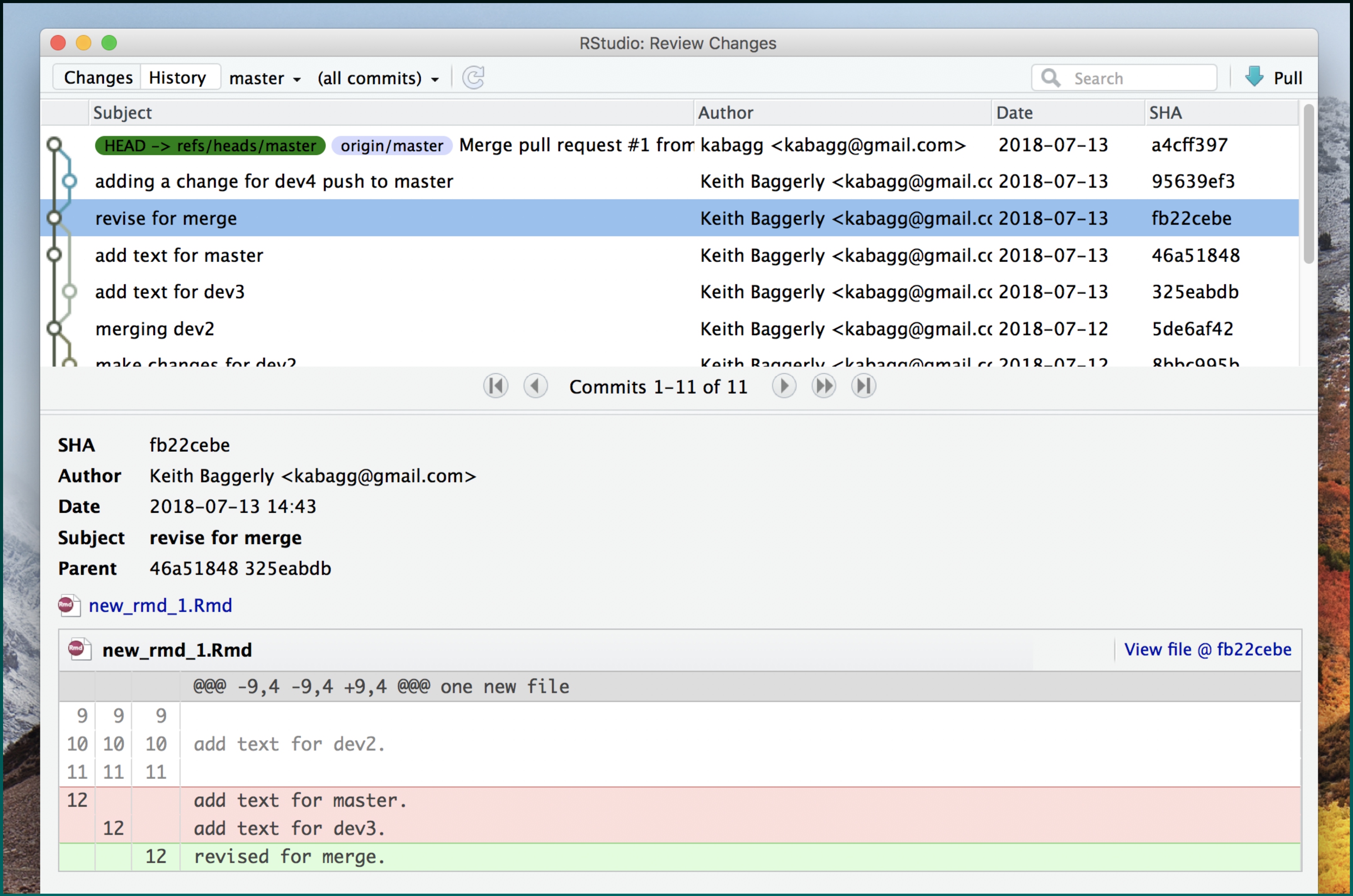
Syncing between local repos on your machine and matched repos on GitHub is pretty straightforward.

These tools let us work publicly in spirit.

The Git Pane


BID/Testing/gitCommandLine/tempTest2 - master - RStudio
Addins
tempTest2
Environment
History
Connections
Git
Diff
Commit
master
Staged
Status
Path
Files
Plots
Packages
Help
Viewer

The Git History



Subject	Author	Date	SHA
HEAD -> refs/heads/master origin/master Merge pull request #1 from	kabagg <kabagg@gmail.com>	2018-07-13	a4cff397
adding a change for dev4 push to master	Keith Baggerly <kabagg@gmail.cc	2018-07-13	95639ef3
revise for merge	Keith Baggerly <kabagg@gmail.cc	2018-07-13	fb22cebe
add text for master	Keith Baggerly <kabagg@gmail.cc	2018-07-13	46a51848
add text for dev3	Keith Baggerly <kabagg@gmail.cc	2018-07-13	325eabdb
merging dev2	Keith Baggerly <kabagg@gmail.cc	2018-07-12	5de6af42
make changes for dev2	Keith Baggerly <kabagg@gmail.cc	2018-07-12	8bbc995b

Commits 1–11 of 11

SHA	fb22cebe
Author	Keith Baggerly <kabagg@gmail.com>
Date	2018-07-13 14:43
Subject	**revise for merge**
Parent	46a51848 325eabdb

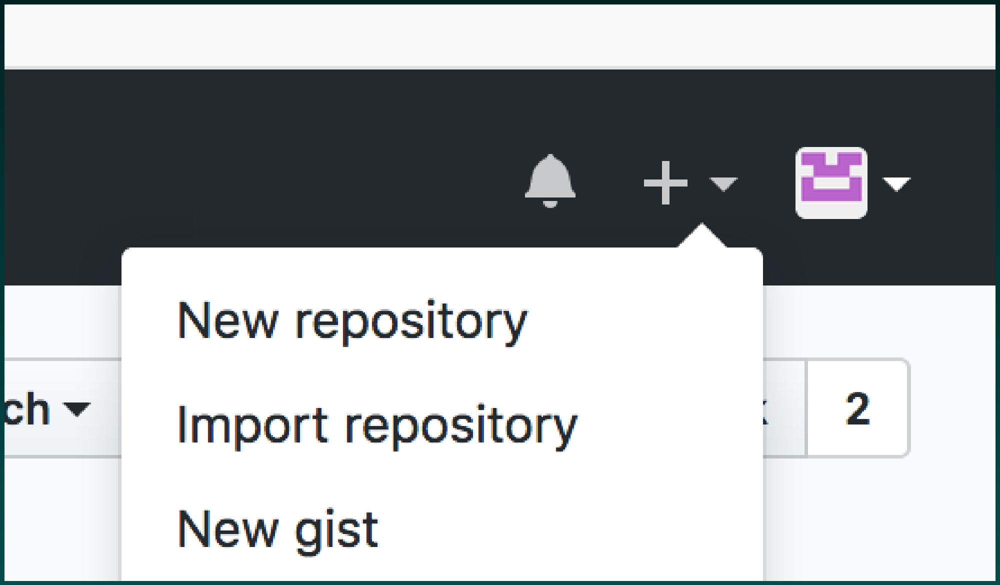
new_rmd_1.Rmd

new_rmd_1.Rmd View file @ fb22cebe

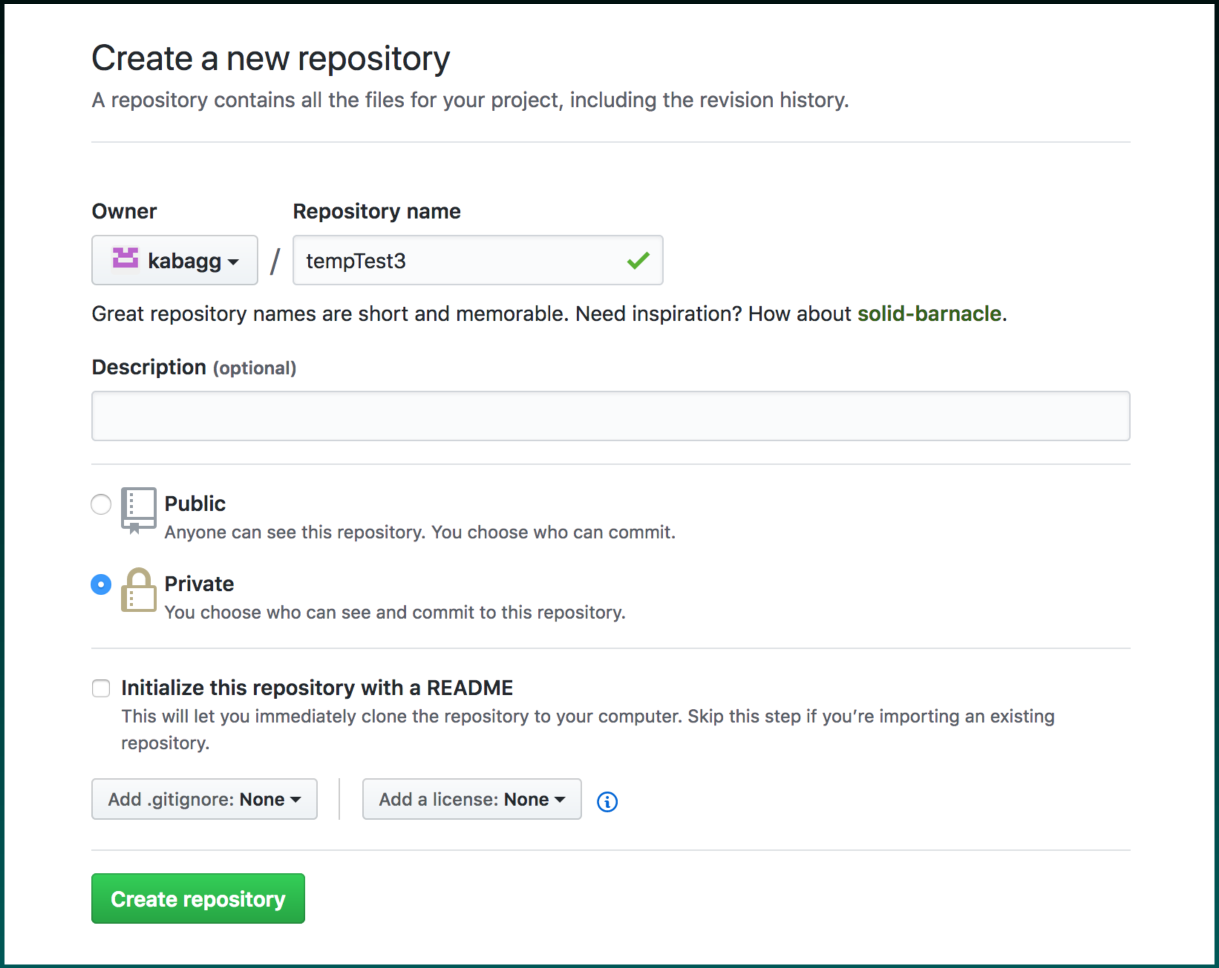
```
         @@@ -9,4 -9,4 +9,4 @@@ one new file
 9   9   9
10  10  10   add text for dev2.
11  11  11
12           add text for master.
    12       add text for dev3.
        12   revised for merge.
```

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



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: kabagg / Repository name: tempTest3 ✓

Great repository names are short and memorable. Need inspiration? How about **solid-barnacle**.

Description (optional)

○ **Public**
Anyone can see this repository. You choose who can commit.

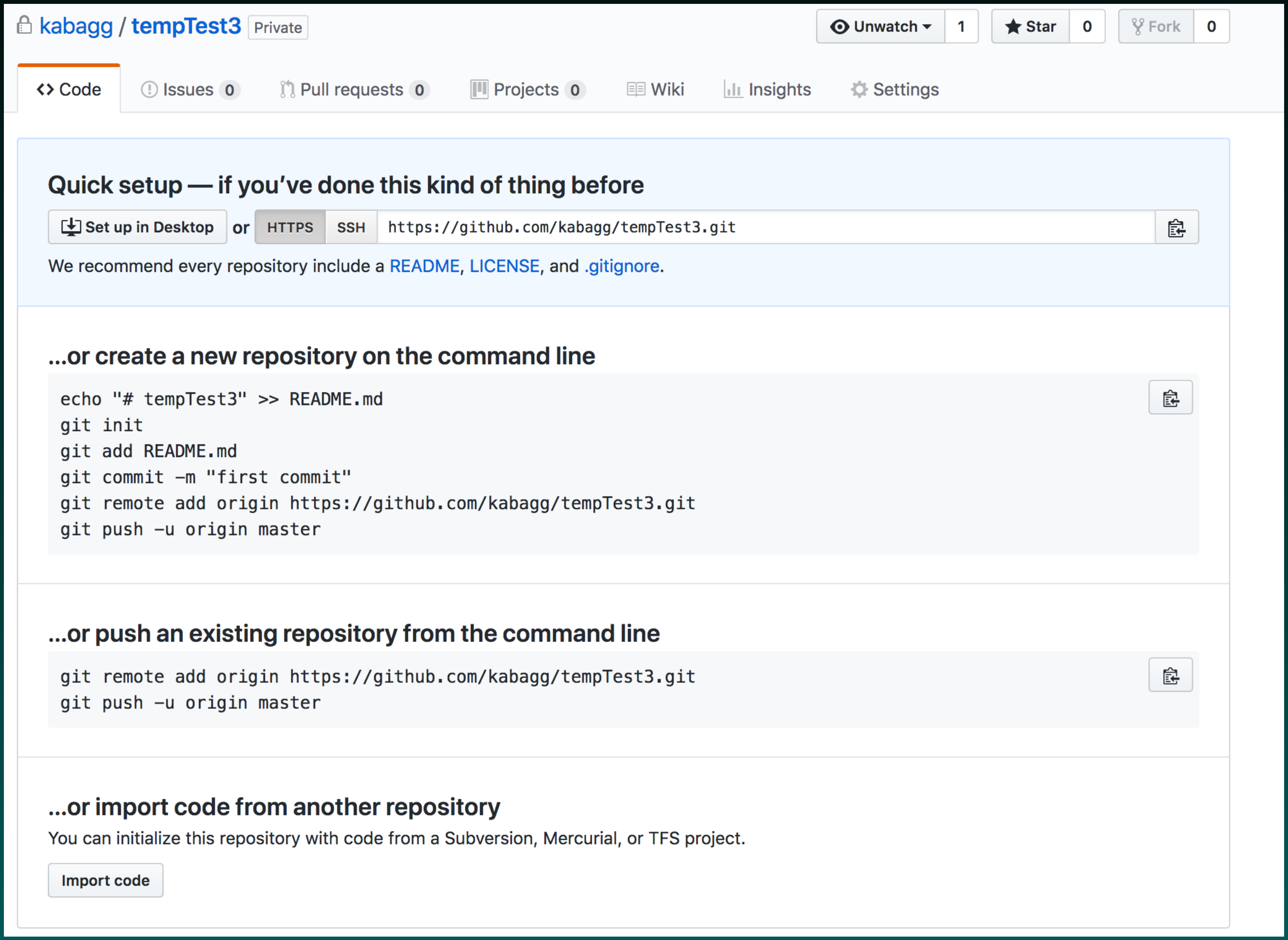
● **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

GitHub recommends quickly setting up README, .gitignore, and LICENSE.

Prompts for Linking the New Repo


kabagg / tempTest3 Private
Unwatch 1 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/kabagg/tempTest3.git`

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# tempTest3" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/kabagg/tempTest3.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/kabagg/tempTest3.git
git push -u origin master
```

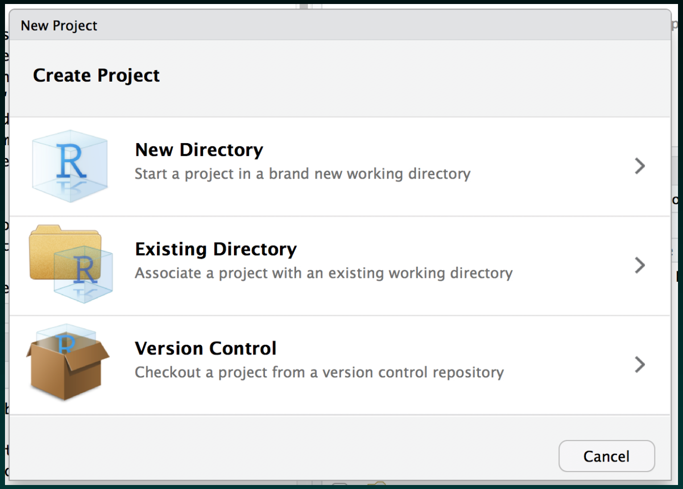
...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

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

Step 2: Start an RStudio Project from GitHub



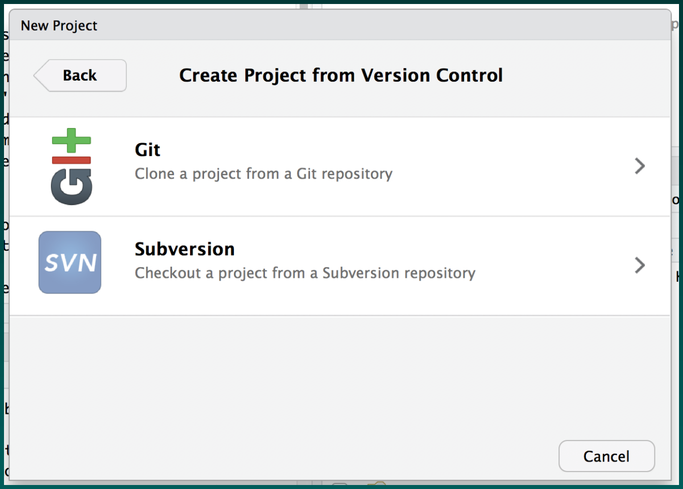
New Project

Create Project

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository



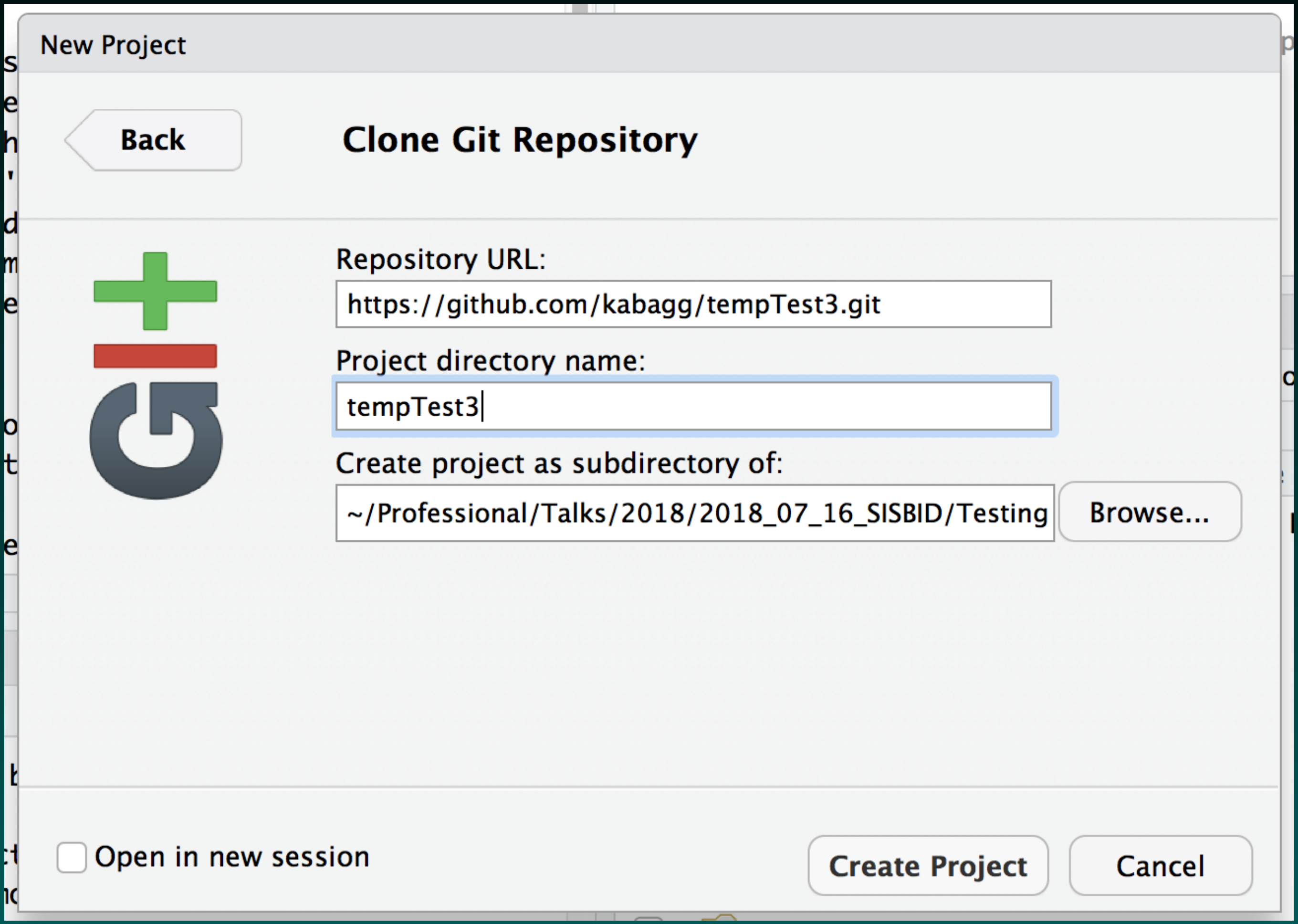
New Project

Create Project from Version Control

Git
Clone a project from a Git repository

Subversion
Checkout a project from a Subversion repository

Link the Project to GitHub



Clone Git Repository

Repository URL: https://github.com/kabagg/tempTest3.git

Project directory name: tempTest3

Create project as subdirectory of: ~/Professional/Talks/2018/2018_07_16_SISBID/Testing

☐ Open in new session

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

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.

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.

Kludges, Workarounds, and Warnings

Not everything works the way I want it to...

Software Versions and RStudio

Most times, I list the packages I'm using along with their version numbers in the README.

```
`r packageVersion("magrittr")`
`r version[["version.string"]]`
RStudio.Version()[["version"]]
```

At present, I can't do this with RStudio itself. The last line above doesn't work. It only works in an active RStudio session, and knitr typically runs things in background.

This issue has been raised, but there isn't a workaround yet.

I currently hardcode this value.

github_document and toc

I like the github_document output format.

I also like to include tables of contents.

Unfortunately, the github_document toc option doesn't work.

This issue has been raised, but there isn't a workaround yet. This seems to have broken with a move to "pandoc2.0" which parses text under the hood.

I currently run things twice - once as an md_document, to get the toc, which I cut and paste, and then as a github_document.

rmarkdown::render and output_dir

I like being able to automatically specify where my output should go, using relative positioning.

rmarkdown::render has an “output_dir” option which I happily started using. But. This replaced all my nice relative paths with absolute ones, so they broke on other machines.

This issue has been raised, but Yihui’s recommendation is:

Don’t use output_dir. Produce the output files in R/Rmd file folder, and move them once they’re produced.

99_make_all.R follows this approach.

rmarkdown::render and new.env()

I like using rmarkdown::render as opposed to clicking the knitr button (cute though the latter is). Render lets me script things more completely.

The problem is their default behaviors aren't quite the same. If you click knit, the Rmd file will be run in a "new session", implicitly testing whether things are self contained. rmarkdown::render evaluates in the global environment by default. This can cause leaks and odd behavior.

Tell rmarkdown::render to use "envir = new.env()" to keep things distinct.

99_make_all.R does this.

Other Topics

Stuff for further investigation...

Other Topics

drake

continuous integration (CI), e.g. travis

bookdown / blogdown

packrat / docker / sessionInfo

reprex

notebooks vs Rmd files

finding projects later

report sizes