Happy Git and GitHub for the useR

Jenny Bryan and the STAT 545 TAs

2016-06-22

Contents

What's going on here

Still from Heaven King video

THIS IS AN EXPERIMENT!

Marshalling everything I have re: Git/GitHub + R/RStudio/Rmd in bookdown format, in anticipation of the tutorial at useR! 2016:

- repo that makes this site: https://github.com/jennybc/happy-git-with-r
- session in the useR! schedule
- full description on the main site
- slides for a related talk I've given a couple times

Chapter 1

Why Git? Why GitHub?

1.1 Why Git?

Git is a **version control system**. Its original purpose was to help groups of developers work collaboratively on big software projects. Git manages the evolution of a set of files – called a **repository** – in a sane, highly structured way. If you have no idea what I'm talking about, think of it as the "Track Changes" features from Microsoft Word on steroids.

Git has been re-purposed by the data science community. In addition to using it for source code, we use it to manage the motley collection of files that make up typical data analytical projects, which often consist of data, figures, reports, and, yes, source code.

A solo data analyst, working on a single computer, will benefit from adopting version control. But not nearly enough to justify the pain of installation and workflow upheaval. There are much easier ways ways to get versioned back ups of your files, if that's all you're worried about.

In my opinion, **for new users**, the pros of Git only outweigh the cons when you factor in the overhead of communicating and collaborating with other people. Who among us does not need to do that? Your life is much easier if this is baked into your workflow, as opposed to being a separate process that you dread or neglect.

1.2 Why GitHub?

This is where hosting services like GitHub, Bitbucket, and GitLab come in. They provide a home for your Git-based projects on the internet. If you have no idea what I'm talking about, think of it as DropBox but much, much better. The remote host acts as a distribution channel or clearinghouse for your Git-managed project. It allows other people to see your stuff, sync up with you, and perhaps even make changes. These hosting providers improve upon traditional Unix Git servers with well-designed web-based interfaces.

We target GitHub – not Bitbucket or GitLab – for the sake of specificity. However, all the big-picture principles and even some mechanics will carry over to these alternative hosting platforms.

Don't get too caught up on public versus private at this point. There are many ways to get private repositories from the major providers for low or no cost. Just get started and figure out if and how Git/GitHub is going to work for you! If you outgrow this arrangement, you can throw some combination of technical savvy and money at the problem. You can either pay for a higher level of service or self-host one of these platforms.

1.3 Is it going to hurt?

You have to install Git, get local Git talking to GitHub, and make sure RStudio can talk to local Git (and, therefore, GitHub). This is one-time or once-per-computer pain.

For new or existing projects, you will:

- Dedicate a directory (a.k.a "folder") to it.
- Make it an RStudio Project.
- Make it a Git repository.
- Go about your usual business. But instead of only *saving* individual files, periodically you make a **commit**, which takes a multi-file snapshot of the entire project.
  - Have you ever versioned a file by adding your initials or the date? That is effectively a **commit**, albeit only for a single file: it is a version that is significant to you and that you might want to inspect or revert to later.
- Push commits to GitHub periodically.
  - This is like sharing a document with colleagues on DropBox or sending it out as an email attachment. It signals you're ready to make your work visible to others and invite comment or edits.

This is a change to your normal, daily workflow. It feels weird at first but quickly becomes second nature. FWIW, STAT 545 students are required to submit all coursework via GitHub. This is a major topic in class and office hours for the first two weeks. Then we practically never discuss it again.

The rest of this site is dedicated to walking you through the necessary setup and creating your first few Git projects. We conclude with prompts that guide you through some of the more advanced usage that makes all of this initial pain worthwhile.

1.4 What is the payoff?

If someone needs to see your work or if you want them to try out your code, they can easily get it from GitHub. If they use Git, then can clone or fork your repository. If they don't use Git, they can still browse your project on GitHub like a normal website and even grab everything by downloading a zip archive.

If you need to collaborate on data analysis or code development, then all involved should use Git. Use GitHub as your clearinghouse: individuals work independently, then send work back to GitHub for reconciliation and transmission to the rest of the team.

If you care deeply about someone else's project, such as an R package you use heavily, you can track its development on GitHub. You can watch the repository to get notified of major activity. You can fork it to keep your own copy. You can modify your fork to add features or fix bugs and send them back to the owner as a proposed change.

1.5 Who can do what?

A public repository is readable by the world. The owner can grant higher levels of permission to others, such as the ability to push commits.

A private repository is invisible to the world. The owner can grant read, write (push), or admin access to others.

1.6 Special features of GitHub

this is perhaps too detailed … full stop? or does it belong elsewhere?

In addition to a well-designed user interface, GitHub offers two especially important features:

- **Issues.** Remember how we're high-jacking software development tools? Well, this is the bug tracker. It's a list of things … bugs, feature requests, to do's, whatever.
  - Issues are tightly integrated with email and therefore allow you to copy/embed important conversations in the associated repo.
  - Issues can be assigned to people (e.g., to do's) and tagged ("bug" or "progress-report").
  - Issues are tightly integrated with commits and therefore allow you to record *that the changes in this commit solve that problem which was discussed in that issue.*
  - As a new user of GitHub, one of the most productive things you can do is to use GitHub issues to provide a clear bug report or feature request for a package you use.
- **Pull requests.** Git allows a project to have multiple, independent branches of development, with the notion that some should eventually be merged back into the main development branch. These are technical Git terms but hopefully also make sense on their own. A pull request is a formal proposal that says: "Here are some changes I would like to make." It might be linked to a specific issue: "Related to #14." or "Fixes #56". GitHub facilitates and preserves the discussion of the proposal, holistically and line-by-line.

1.7 What's special about using R and with Git and GitHub?

- the active R package development community on GitHub
- workflows for R scripts and R Markdown files that make it easy to share source and rendered results on GitHub
- Git- and GitHub-related features of the RStudio IDE

1.8 Audience and pre-reqs

The target audience for this site is someone who analyzes data, probably with R, though much of the content may be useful to analysts using other languages. While R package development with Git(Hub) is absolutely in scope, it's not an explicit focus or requirement.

The site is aimed at intermediate to advanced R users, who are comfortable writing R scripts and managing R projects. You should have a good grasp of files and directories and be generally knowledgeable about where things live on your computer.

Although we will show alternatives for most Git operations, we will inevitably spend some time in the shell and we assume some prior experience. For example, you should know how to open up a shell, navigate to a certain directory, and list the files there. You should be comfortable using shell commands to view/move/rename files and to work with your command history.

1.9 What this is NOT

We aim to teach novices about Git on a strict "need to know" basis. Git was built to manage development of the Linux kernel, which is probably very different from what you do. Most people need a small subset of Git's functionality and that will be our focus. If you want a full-blown exposition of Git as a directed acyclic graph or a treatise on the Git-Flow branching strategy, you will be sad.

Chapter 2

Contributors

Jenny Bryan (twitter, GitHub) is a professor at the University of British Columbia. She's been using and teaching R (or S!) for 20 years, most recently in STAT 545 and Software Carpentry. Other aspects of her R life include work with rOpenSci, development of the `googlesheets` and `gapminder` packages, and being academic director for UBC's Master of Data Science.

The development and delivery of this material has benefited greatly from contributions by:

- Dean Attali (blog, GitHub, twitter) who recently earned his M.Sc. Bioinformatics at UBC. He's the developer of the shinyjs package and much more.
- Bernhard Konrad (GitHub, twitter) recently earned his Ph.D. in Applied Math at UBC, completed an Insight Data Science Fellowship, and is a software engineer at Google.
- Shaun Jackman (GitHub, twitter, sjackman.ca) is a Ph.D. in bioinformatics at UBC, the lead developer of Linuxbrew and a developer of Homebrew-Science and the genome sequence assembly software ABySS
- The STAT 545 TA group

Chapter 3

Workshops

These materials can be used for independent study, but I also use them to support

- in-person workshops (see below)
- STAT 545 at UBC
- UBC Master of Data Science

3.1 useR! 2016 Stanford

3.1.1 Logistics

Monday, June 27, 2016
8:15am - 9:00 TAs available to help complete system set up
9:00am - 12:00pm Workshop, with coffee break 10:15am - 10:30am
Lyons & Lodato 326 Galvez Street Stanford, CA 94305-6105 Google Maps
Sessions in the useR! schedule website: part 1, part 2
Full description on the main useR! website

3.1.2 Pre-tutorial set-up

It is vital that you attempt to set up your system in advance. You cannot show up at 9am with no preparation and keep up!

These are battle-tested instructions, so most will succeed. We believe in you! If you have trouble, you can open an issue here and we *might* be able to help in the days leading up to useR! (no promises). We will have TAs in the room starting at 8:15am and throughout the workshop.

Try this. Give it about 1 - 2 hours. If you hit a wall, we will help:

- Register a free GitHub account (chapter 5).
- Install or update R and RStudio (chapter 6).
- Install Git (chapter 7).
- Introduce yourself to Git (chapter 8).
- Prove local Git can talk to GitHub (chapter 10).
- Cache your username and password (chapter 11) or set up SSH keys (chapter 12) so you don't need to authenticate yourself to GitHub interactively *ad nauseum*.
- Prove RStudio can find local Git and, therefore, can talk to GitHub (chapter 13).

  - FYI: this is where our hands-on activities will start. We walk through a similar activity together, with narrative, and build from there.

Troubleshooting:

- If RStudio is having a hard time finding Git, see chapter 14.
- For all manner of problems, both installation and usage related, see chapter 15.

Optional reading on "big picture stuff":

- Read "Why Git? Why GitHub?" in chapter 1.

3.1.3 What we can do together

- Verify and complete the most difficult part: installation and configuration!
- Create a Git repository and connect the local repo to a GitHub remote, for new and existing projects.
- Run R code, via R Markdown or a script, and share a presentable report via GitHub.
- The intersection of GitHub and the R world:
  - R packages developed on Github and how to make use of Issues
  - METACRAN read-only mirror of all of CRAN + R-specific searching tips.
- Propose a change to someone else's project, i.e. "make a pull request".
- Discuss daily workflow:
  - How often should I commit? Which files should I commit?
  - Data files and the dilemmas they present.
  - Most common Git predicaments. How to avoid and recover.
  - How do groups of 1, 5, or 10 people structure their work with Git(Hub)?

3.2 CSAMA 2016: Statistical Data Analysis for Genome Biology

3.2.1 Logistics

http://www.huber.embl.de/csama2016/
July 10 - 15, 2016, Bressanone-Brixen, Italy
Monday July 11, Lab 2, 15:30 - 17:00 Reproducible research and R authoring with markdown and knitr
Thursday July 14, Lab 4, 14:00 - 15:30 Use of Git and GitHub with R, RStudio, and R Markdown

Part I

Installation

Chapter 4

Installation pain

Getting all the necessary software installed, configured, and playing nicely together is honestly half the battle here. Brace yourself for some pain. The upside is that you can give yourself a pat on the back once you get through this. And you WILL get through this.

You will find far more resources for how to *use Git* than for installation and configuration. Why? The experts …

- Have been doing this for years. It's simply not hard for them anymore.
- Probably use some flavor of Unix. They may secretly (or not so secretly) take pride in neither using nor knowing Windows.
- Get more satisfaction and reward for thinking and writing about Git concepts and workflows than Git installation.

In their defense, it's hard to write installation instructions. Failures can be specific to an individual OS or even individual computer. If you have some new problem and, especially, the corresponding solution, we'd love to hear from you!

4.1 Success and operating systems

Our installation instructions have been forged in the fires of STAT 545, STAT 540, and assorted workshops, over several years. We regularly hear from grateful souls on the internet who also have success.

Here's some recent data on the subset of STAT 545 students for which we recorded the operating system: half Mac, just under half Windows (various flavours), and a dash of Linux.

	2014	2015
Mac	16 (41%)	38 (52%)
Windows 7	9 (23%)	13 (18%)
Windows 8	12 (31%)	9 (12%)
Windows 10	0 (0%)	8 (11%)
Linux	2 (5%)	5 (7%)

Chapter 5

Register a GitHub account

Register an account with GitHub. It's free!

- https://github.com

5.1 Username advice

You will be able upgrade to a paid level of service, apply discounts, join organizations, etc. in the future, so don't fret about any of that now. **Except your username. You might want to give that some thought.**

A few tips, which sadly tend to contradict each other:

- Incorporate your actual name! People like to know who they're dealing with. Also makes your username easier for people to guess or remember.
- Reuse your username from other contexts, e.g., Twitter or Slack. But, of course, someone with no GitHub activity will probably be squatting on that.
- Pick a username you will be comfortable revealing to your future boss.
- Shorter is better than longer.
- Be as unique as possible in as few characters as possible. In some settings GitHub auto-completes or suggests usernames.
- Make it timeless. Don't highlight your current university, employer, or place of residence.
- Avoid words laden with special meaning in programming. In my first inept efforts to script around the GitHub API, I assigned lots of issues to the guy with username `NA` because my vector of GitHub usernames contained missing values. A variant of Little Bobby Tables.

You can change your username later, but better to get this right the first time.

- https://help.github.com/articles/changing-your-github-username/
- https://help.github.com/articles/what-happens-when-i-change-my-username/

5.2 Free private repos

GitHub offers free unlimited private repositories for users and organizations in education, academic research, nonprofits, and charities.

Go ahead and register your free acount NOW and then pursue any special offer that applies to you:

- Students, faculty, and educational/research staff: GitHub Education.
  - GitHub "Organizations" can be extremely useful for courses or research/lab groups, where you need some coordination across a set of repos and users.
- Official nonprofit organizations and charities: GitHub for Good

5.3 Pay for private repos

Everyone else can pay for some private repos. A personal plan with unlimited private repos is $7 / month at the time of writing. See the current plans and pricing here:

- https://github.com/pricing

Chapter 6

Install or upgrade R and RStudio

Install pre-compiled binary of R for your OS: https://cloud.r-project.org

Install Preview version RStudio Desktop: https://www.rstudio.com/products/rstudio/download/preview/

Update your R packages: `update.packages(ask = FALSE, checkBuilt = TRUE)`

Read on for more detail or hand-holding.

6.1 More about R and RStudio

Get current, people. You don't want to adopt new things on day one. But at some point, running old versions of software adds unnecessary difficulty.

In live workshops, there is a limit to how much we can help with ancient versions of R or RStudio. Also, frankly, there is a limit to our motivation. By definition, these problems are going away and we'd rather focus on edge cases with current versions, which affect lots of people.

For more guidance on installing or upgrading R and RStudio, go here:

STAT 545 page on Installing R and RStudio

Chapter 7

Install Git

You need Git, so you can use it at the command line and so RStudio can call it.

If there's any chance it's installed already, verify that, rejoice, and skip this step.

Otherwise, find installation instructions below for your operating system.

7.1 Git already installed?

Go to the shell (chapter A) and enter `which git` and `git --version`:

```
jenny@2015-mbp happy-git-with-r $ which git
/usr/bin/git

jenny@2015-mbp happy-git-with-r $ git --version
git version 2.7.4 (Apple Git-66)
```

If Git reports a path to an executable and a version, that's great! You have Git already. No need to install! Move on.

If, instead, you see something more like `git: command not found`, keep reading.

Mac OS users might get an immediate offer to install command line developer tools. Yes, you should accept! Click "Install" and read more below.

7.2 Windows

Option 1 (*recommended*): Install Git for Windows, previously known as `msysgit` or "Git Bash", to get Git in addition to some other useful tools, such as the Bash shell. Yes, all those names are totally confusing.

- This approach leaves the Git executable in a conventional location, which will help you and other programs, e.g. RStudio, find it and use it. This also supports a transition to more expert use, because the Bash shell will be useful as you venture outside of R/RStudio.
- This also leaves you with a Git client, though not a very good one. So check out Git clients we recommend (chapter 9).

Option 2 (*NOT recommended*): The GitHub hosting site offers GitHub Desktop for Windows that provides Git itself, a client, and smooth integration with GitHub.

- Their Windows set-up instructions recommend this method of Git installation.
- Why don't we like it? We've seen GitHub Desktop for Windows lead to Git installation in suboptimal locations, such as `~/AppData/Local`, and in other places we could never find. If you were **only** going to interact with GitHub via this app, maybe that's OK, but that does not apply to you. Therefore, we recommend option 1 instead.

7.3 Mac OS

Option 1 (*highly recommended*): Install the Xcode command line tools (not all of Xcode), which includes Git.

Go to the shell and enter one of these commands to elicit an offer to install developer command line tools:

```
git --version
git config
```

Accept the offer! Click on "Install".

Another way to request this is via `xcode-select --install`. We just happen to find this Git-based trigger apropos.

Option 2 (*recommended*): Install Git from here: http://git-scm.com/downloads.

- This arguably sets you up the best for the future. It will certainly get you the latest version of Git of all approaches described here.
- The GitHub home for this project is here: https://github.com/timcharper/git_osx_installer.
  - At that link, there is a list of maintained builds for various combinations of Git and Mac OS version. If you're running 10.7 Lion and struggling, we've had success in September 2015 with binaries found here: https://www.wandisco.com/git/download.

Option 3 (*recommended*): If you anticipate getting heavily into scientific computing, you're going to be installing and updating lots of software. You should check out homebrew, "the missing package manager for OS X". Among many other things, it can install Git for you. Once you have Homebrew installed, do this in the shell:

```
brew install git
```

Option 4 (*NOT recommended*): The GitHub hosting site offers GitHub Desktop for Mac that provides *the option* to install Git itself, a client, and smooth integration with GitHub..

- Their Mac set-up instructions recommend this method of Git installation.
- We don't like GitHub Desktop as a Git client, so this is a very cumbersome way to install Git. Consider this option a last resort.

7.4 Linux

Install Git via your distro's package manager.

Ubuntu or Debian Linux:

```
sudo apt-get install git
```

Fedora or RedHat Linux:

```
sudo yum install git
```

A comprehensive list for various Linux and Unix package managers:
https://git-scm.com/download/linux

Chapter 8

Introduce yourself to Git

In the shell (chapter A):

```
git config --global user.name 'Jennifer Bryan'
git config --global user.email 'jenny@stat.ubc.ca'
git config --global --list
```

substituting your name and **the email associated with your GitHub account**.

8.1 More about `git config`

From RStudio, go to *Tools* > *Shell* and tell `git` your name and **GitHub email** by typing (use your own name and email):

- `git config --global user.name 'Jennifer Bryan'`
  - This does **NOT** have to be your GitHub username, although it can be. Another good option is your actual first name and last name. Your commits will be labelled with this name, so this should be informative to potential collaborators.
- `git config --global user.email 'jenny@stat.ubc.ca'`
  - This **must** be the email that you used to sign up for GitHub.

These commands return nothing. You can check that `git` understood what you typed by looking at the output of `git config --global --list`.

Chapter 9

Install a Git client

This is optional but **highly recommended**.

Unless specified, it is not required for live workshops and will not be explicitly taught, though you might see us using one of these clients.

9.1 What and why

Learning to use version control can be rough at first. I found the use of a GUI – as opposed to the command line – extremely helpful when I was getting started. I call this sort of helper application a Git client. It's really a Git(Hub) client because they also help you interact with GitHub or other remotes.

Git and your Git client are not the same thing, just like R and RStudio are not the same thing. A Git client and the RStudio IDE are not necessary to use to Git or R, respectively, but they make the experience more pleasant because they reduce the amount of command line bullshittery.

RStudio offers a very basic Git client. I use this often for simple operations, but you probably want another, more powerful one as well.

Fair warning: for some things, you will have to use the command line. But the more powerful your Git client is, the less often this happens.

Fantastic news: because all of the clients are just forming and executing Git commands on your behalf, you don't have to pick one. You can literally do one operation from the command line, do another from RStudio, and another from SourceTree, one after the other, and it just works. *Very rarely, both clients will scan the repo at the same time and you'll get an error message about* `.git/index.lock`. *Try the operation again at least once before doing any further troubleshooting.*

9.2 Recommended Git clients

- SourceTree is a free, powerful Git(Hub) client that I highly recommend. It was my first Git client and is still my favorite for nontrivial Git tasks. Available for Mac and Windows. If I'm teaching you in a course or workshop, you might see me using this.
- GitKraken is quite new on the scene and is free. I would probably start here if I was starting today. Why? Because it works across all three OSes my students use: Windows, Mac, and Linux. I hear very good reviews, especially from long-suffering Linux users who haven't had any great options until now.

- GitUp is a free, open source client for Mac OS. I've heard really good things about it and like what I read on the website. However, I tried to make myself use it for a day and went sort of nuts, possibly because I'm so used to SourceTree. YMMV.
- GitHub also offers a free Git(Hub) client for Windows and Mac. We do NOT recommend it for Windows and have serious reservations even for Mac OS. What do we object to?
  - The degree of hand-holding offered by GitHub's clients borders on hand-*cuffs*.
  - The Windows client sometimes leaves the Git executable so well hidden that we can't find it. This is a deal-killer, because that means RStudio can't find it either.
  - These clients wrap Git functionality so thoroughly that we've had students make some destructive mistakes. For example, we've seen a "sync" operation that resulted in the loss of local uncommitted changes. Exactly which Git operations, in what order, are implied by "sync", is not entirely clear. We prefer clients that expose Git more explicitly.
  - We've heard similar negative reviews from other instructors. It's not just us.
- Others that I have heard positive reviews for:
  - SmartGit
  - git-cola
  - magit, for Emacs nerds
- Browse even more Git(Hub) clients.

Part II

Connect Git, GitHub, RStudio

Chapter 10

Connect to GitHub

Objective: make sure that you can pull from and push to GitHub from your computer.

I do not explain all the shell and Git commands in detail. This is a black box diagnostic / configuration exercise. In later chapters and in live workshops, we revisit these operations with much more narrative.

10.1 Make a repo on GitHub

Go to https://github.com and make sure you are logged in.

Click green "New repository" button. Or, if you are on your own profile page, click on "Repositories", then click the green "New" button.

Repository name: `myrepo` (or whatever you wish, we will delete this)
Public
YES Initialize this repository with a README

Click big green button "Create repository."

Copy the HTTPS clone URL to your clipboard via the green "Clone or Download" button.

10.2 Clone the repo to your local computer

Go to the shell.

Take charge of – or at least notice! – what directory you're in. `pwd` to display working directory. `cd` to move around. Personally, I would do this sort of thing in `~/tmp`.

Clone `myrepo` from GitHub to your computer. This URL should have **your GitHub username** and the name of **your practice repo**. If your shell cooperates, you should be able to paste the whole `https://...` bit that we copied above. But some shells are not (immediately) clipboard aware. In that sad case, you must type it. **Accurately.**

```
git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git
```

This should look something like this:

```
jenny@2015-mbp tmp $ git clone https://github.com/jennybc/myrepo.git
Cloning into 'myrepo'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

Make this new repo your working directory, list its files, display the README, and get some information on its connection to GitHub:

```
cd myrepo
ls
less README.md
git remote show origin
```

This should look something like this:

```
jenny@2015-mbp ~ $ cd myrepo

jenny@2015-mbp myrepo $ ls
README.md

jenny@2015-mbp myrepo $ less README.md
# myrepo
tutorial development

jenny@2015-mbp myrepo $ git remote show origin
* remote origin
  Fetch URL: https://github.com/jennybc/myrepo.git
  Push  URL: https://github.com/jennybc/myrepo.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

10.3 Make a local change, commit, and push

Add a line to README and verify that Git notices the change:

```
echo "A line I wrote on my local computer" >> README.md
git status
```

This should look something like this:

```
jenny@2015-mbp myrepo $ echo "A line I wrote on my local computer" >> README.md
jenny@2015-mbp myrepo $ git status
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Commit this change and push to your remote repo on GitHub. If you're a new GitHub user, you will be challenged for your GitHub username and password. Provide them!

```
git add -A
git commit -m "A commit from my local computer"
git push
```

This should look something like this:

```
jenny@2015-mbp myrepo $ git add -A

jenny@2015-mbp myrepo $ git commit -m "A commit from my local computer"
[master de669ba] A commit from my local computer
 1 file changed, 1 insertion(+)

jenny@2015-mbp myrepo $ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 311 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/jennybc/myrepo.git
   b4112c5..de669ba  master -> master
```

10.4 Confirm the local change propagated to the GitHub remote

Go back to the browser. I assume we're still viewing your new GitHub repo.

Refresh.

You should see the new "A line I wrote on my local computer" in the README.

If you click on "commits," you should see one with the message "A commit from my local computer."

If you have made it this far, you are ready to graduate to using Git and GitHub with RStudio (chapter 13). But first ...

10.5 Am I really going to type GitHub username and password on each push?

It is likely that your first push, above, leads to a challenge for your GitHub username and password.

This will drive you crazy in the long-run and make you reluctant to push. Do one of the follow to eliminate this annoyance:

- Credential caching for HTTPS access, chapter 11.
- Set up SSH keys, chapter 12.

Now is the perfect time to do this, since you have a functioning test repo.

10.6 Clean up

Local When you're ready to clean up, you delete the local repo any way you like. It's just a regular directory on your computer.

Here's how to do that in the shell, if current working directory is `myrepo`:

```
cd ..
rm -rf myrepo/
```

GitHub In the browser, go to your repo's landing page on GitHub. Click on "Settings".

Scroll down, click on "delete repository," and do as it asks.

Chapter 11

Cache credentials for HTTPS

If you plan to push/pull using HTTPS, you want Git to cache your credentials (username, password), so you don't need to enter them over and over again.

I do not explain all the shell and Git commands in detail. This is a black box diagnostic / configuration exercise.

11.1 Get a test repository

You need a functioning test Git repository. One that exists locally and remotely on GitHub, with the local repo tracking the remote.

If you have just verified that you can interact with GitHub (chapter 10) from your local computer, that test repo will be perfect.

If you have just verified that you can work with GitHub from RStudio (chapter 13), that test repo will also be perfect.

You may proceed when

- You have a test repo.
- You know where it lives on your local computer. Example:
  - `/home/jenny/tmp/myrepo`
- You know where it lives on GitHub. Example:
  - `https://github.com/jennybc/myrepo`
- You know local is tracking remote. In a shell with working directory set to the local Git repo, enter:

  ```
  git remote -v
  ```

  Output like this confirms that fetch and push are set to remote URLs that point to your GitHub repo:

  ```
  origin  https://github.com/jennybc/myrepo (fetch)
  origin  https://github.com/jennybc/myrepo (push)
  ```

  Now enter:

```
git branch -vv
```

Here we confirm that the local `master` branch has your GitHub master branch (`origin/master`) as upstream remote. Gibberish? Just check that your output looks similar to mine:

```
master b8e03e3 [origin/master] line added locally
```

11.2 Verify that your Git is new enough to have a credential helper

In a shell, do:

```
git --version
```

and verify your version is 1.7.10 or newer. If not, update Git (chapter 7) or use SSH keys (chapter 12).

11.3 Turn on the credential helper

11.3.0.1 Windows

In the shell, enter:

```
git config --global credential.helper wincred
```

11.3.0.2 Windows, plan B

If that doesn't seem to work, install an external credential helper.

- Download the git-credential-winstore.exe application.
- Run it! It should work if Git is in your `PATH` environment variable. If not, go to the directory where you downloaded the application and run the following:

  ```
  git-credential-winstore -i "C:\Program Files (x86)\Git\bin\git.exe"
  ```

11.3.0.3 Mac

Find out if the credential helper is already installed. In the shell, enter:

```
git credential-osxkeychain
```

And look for this output:

```
usage: git credential-osxkeychain <get|store|erase>
```

If you don't get this output, it means you need a more recent version of Git, either via command line developer tools or homebrew. Go back to the Mac section of chapter (7).

Once you've confirmed you have the credential helper, enter:

```
git config --global credential.helper osxkeychain
```

11.3.0.4 Linux

In the shell, enter:

```
git config --global credential.helper 'cache --timeout=10000000'
```

to store your password for ten million seconds or around 16 weeks, enough for a semester.

11.3.1 Trigger a username / password challenge

Change a file in your local repo and commit it. Do that however you wish. Here are shell commands that will work:

```
echo "adding a line" >> README.md
git add -A
git commit -m "A commit from my local computer"
```

Now push!

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

One last time you will be asked for your username and password, which hopefully will be cached.

Now push AGAIN.

```
git push
```

You should NOT be asked for your username and password, instead you should see `Everything up-to-date`.

Rejoice and close the shell.

Chapter 12

Set up keys for SSH

instructions via RStudio and the shell to go here

In the meantime, here are good instructions:

- How to do via RStudio? see the end of the section on initial set up:
  - http://r-pkgs.had.co.nz/git.html#git-init
- How to do via shell?
  - https://help.github.com/articles/generating-an-ssh-key/

Chapter 13

Connect RStudio to Git and GitHub

Here we verify that RStudio can issue Git commands on your behalf. Assuming that you've gotten local Git to talk to GitHub, this means you'll also be able to pull from and push to GitHub from RStudio.

In later chapters and in live workshops~, we revisit these operations with much more explanation.

If you succeed here, your set up is DONE.

13.1 Prerequisites

We assume the following:

- You've registered a free GitHub account (chapter 5).
- You've installed/updated R and RStudio (chapter 6).
- You've installed Git (chapter 7).
- You've introduced yourself to Git (chapter 8).
- You've confirmed that you can push to / pull from GitHub from the command line (chapter 10).

13.2 Make a new repo on GitHub

Go to https://github.com and make sure you are logged in.

Click green "New repository" button. Or, if you are on your own profile page, click on "Repositories", then click the green "New" button.

Repository name: `myrepo` (or whatever you wish, we will delete this)
Public
YES Initialize this repository with a README

Click big green button "Create repository."

Copy the HTTPS clone URL to your clipboard via the green "Clone or Download" button. Or copy the SSH URL if you chose to set up SSH keys.

13.3 Clone the new GitHub repository to your computer via RStudio

In RStudio, start a new Project:

- *File > New Project > Version Control > Git.* In the "repository URL" paste the URL of your new GitHub repository. It will be something like this `https://github.com/jennybc/myrepo.git`.
  - Do you NOT see an option to get the Project from Version Control? Go to chapter 14 for tips on how to help RStudio find Git.
- Take charge of – or at least notice! – the local directory for the Project. A common rookie mistake is to have no idea where you are saving files or what your working directory is. Pay attention. Be intentional. Personally, I would do this in `~/tmp`.
- I suggest you check "Open in new session", as that's what you'll usually do in real life.
- Click "Create Project".

This should download the `README.md` file that we created on GitHub in the previous step. Look in RStudio's file browser pane for the `README.md` file.

13.4 Make local changes, save, commit

From RStudio, modify the `README.md` file, e.g., by adding the line "This is a line from RStudio". Save your changes.

Commit these changes to your local repo. How?

From RStudio:

- Click the "Git" tab in upper right pane.
- Check "Staged" box for `README.md`.
- If you're not already in the Git pop-up, click "Commit".
- Type a message in "Commit message", such as "Commit from RStudio".
- Click "Commit".

13.5 Push your local changes online to GitHub

Click the green "Push" button to send your local changes to GitHub. If you are challenged for username and password, provide them (but see below). You should see some message along these lines.

```
[master dc671f0] blah
 3 files changed, 22 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 myrepo.Rproj
```

13.6 Confirm the local change propagated to the GitHub remote

Go back to the browser. I assume we're still viewing your new GitHub repo.

Refresh.

You should see the new "This is a line from RStudio" in the README.

If you click on "commits," you should see one with the message "Commit from RStudio".

If you have made it this far, you are DONE with set up. But first ...

13.7 Were you challenged for GitHub username and password?

If you somehow haven't done so yet, now is the perfect time to make sure you don't need to keep providing username and password on each push.

Pick one:

- Credential caching for HTTPS access, chapter 11.
- Set up SSH keys, chapter 12.

Now is the perfect time to do this, since you have a functioning test repo.

13.8 Clean up

Local When you're ready to clean up, you delete the local repo any way you like. It's just a regular directory on your computer.

GitHub In the browser, go to your repo's landing page on GitHub. Click on "Settings".

Scroll down, click on "delete repository," and do as it asks.

Chapter 14

Detect Git from RStudio

If you want RStudio to help with your Git and GitHub work, it must be able to find the Git executable.

This often "just works", so this page is aimed at people who have reason to suspect they have a problem.

This is something you set up once-per-computer.

14.1 Do you have a problem?

Let's check if RStudio can find the Git executable.

- *File > New Project...* Do you see an option to create from Version Control? If yes, good.
- Select *New Directory > Empty Project*. Do you see a checkbox "Create a git repository"? If yes, good, CHECK IT.
- Give this disposable test project a name and click *Create Project*. Do you see a "Git" tab in the upper right pane, the same one that has "Environment" and "History"? If yes, good.

If all looks good, you can delete this project. Looks like RStudio and Git are talking to each other.

Keep reading if things don't go so well or you want to know more.

14.2 Find Git yourself

RStudio can only act as a GUI front-end for Git if Git has been successfully installed (chapter 7) **AND RStudio can find it**.

A basic test for successful installation of Git is to simply enter `git` in the shell. If you get a complaint about Git not being found, it means installation was unsuccessful or that it is not being found, i.e. it is not on your `PATH`.

If you are not sure where the Bit executable lives, try this in a shell:

- `which git` (Mac, Linux)
- `where git` (most versions of Windows)

14.3 Tell RStudio where to find Git

If Git appears to be installed and findable, launch RStudio. Quit and re-launch RStudio if there's **any doubt in your mind** about whether you opened RStudio before or after installing Git. Don't make me stop this car and restart RStudio for you in office hours. DO IT.

From RStudio, go to *Tools > Global Options > Git/SVN* and make sure that the box *Git executable* points to … the Git executable. It should read something like:

- `/usr/bin/git` (Mac, Linux)
- `C:/Program Files (x86)/Git/bin/git.exe` (Windows)

  Here is a screenshot of someone doing this on a Windows machine.

  - **WARNING**: On Windows, do **NOT** use `C:/Program Files (x86)/Git/cmd/git.exe`. `bin` is GOOD YES! `cmd` is BAD NO!
- At times, we've had trouble navigating to the necessary directory on Mac OS, once we've clicked "Browse" and are working with a Finder-type window. The keyboard shortcut "command + shift + g" will summon "Go To Folder", where you will be able type or paste any path you want.

Restart RStudio if you make any changes. Don't make me stop this car again and restart RStudio for you in office hours. DO IT.

Do the steps at the top of the page to see if RStudio and git are communicating now.

No joy?

- I've seen this help: With your Project open, go to `Tools > Project Options...`. If available, click on "Git/SVN" and select "Git" in the Version control system dropdown menu. Answer "yes" to the "Confirm New Git Repository" pop up. Answer "yes" to the "Confirm Restart RStudio" pop up.
- If you installed Git via GitHub for Windows, it is possible the Git executable is really well hidden. Get our help or install Git another way.
- Your `PATH` is probably not set up correctly and/or you should re-install Git and control/notice where it's going. Read more in 15.
- Get our help.

Chapter 15

RStudio, Git, GitHub Hell

Problems we have seen and possible solutions.

If you experience some new problem and, especially, find the corresponding solution, we'd love to hear from you!

15.1 I think I have installed Git but damn if I can find it

When you install Git, try to control or record where it is being installed! Make a mental or physical note of these things.

You may be able to find Git after the fact with these commands in the shell:

- `which git` (Mac, Linux, or anything running a bash shell)
- `where git` (Windows, when not in a bash shell)

It is not entirely crazy to just re-install Git, using a method that leaves it in a more conventional location, and to pay very close attention to where it's being installed. Live and learn.

15.2 Dysfunctional PATH

I'm pretty sure that most cases of RStudio *not* automatically detecting the Git executable stem from problems with `PATH`. This is the set of directories where your computer will look for executables, such as Git (today) or `make` (later in this course). Certain methods of Git installation, especially on Windows and/or older OSes, have a higher tendency to put Git in an unconventional location or to fail to add the relevant directory to `PATH`.

How to see your `PATH`?

In the shell:

```
echo $PATH
```

Take a good hard look at this. See the point above about finding your Git executable or re-installing it while you are **wide awake**. Is the host directory in your `PATH`? No? **Fix that.**

Go here for instructions on what to put in your `.bash_profile` in order to add a directory to `PATH`.

15.3 Push/Pull buttons greyed out in RStudio

Are you sure your local repository is tracking a remote repository, e.g. a GitHub repo? In a shell with working directory set to the local Git repo, enter these commands:

```
jenny@2015-mbp myrepo $ git remote -v
origin  https://github.com/jennybc/myrepo (fetch)
origin  https://github.com/jennybc/myrepo (push)

jenny@2015-mbp myrepo $ git branch -vv
* master b8e03e3 [origin/master] line added locally
```

We want to see that fetch and push are set to remote URLs that point to the remote repo. We also want to see that your local master branch has your GitHub master branch as upstream remote.

If you discover you still need to set a remote, go to the shell and get into the working directory of the RStudio Project and Git repo of interest.

- Initiate the "upstream" or "tracking" relationship by adding a remote. Substitute the HTTPS URL for **your GitHub repo**.

  ```
  git remote add origin https://github.com/jennybc/myrepo.git
  ```

- Download all the files from the online GitHub repository and deal with any conflicts.

  ```
  git pull origin master
  ```

- Cement the tracking relationship between your GitHub repository and the local repo by pushing and setting the "upstream" remote:

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

15.4 I have no idea if my local repo and my remote repo are connected.

See the above section on "Push/Pull buttons greyed out in RStudio."

15.5 Push fail at the RStudio level

Do you get this error in RStudio?

```
error: unable to read askpass response from 'rpostback-askpass'
```

Open the shell: *Tools > Shell*.

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

15.6 Push rejected, i.e. fail at the Git/GitHub level

You might have changes on the remote AND on your local repo. Just because you don't remember making any edits in the browser doesn't mean you didn't. Humor me.

Pull first. Resolve any conflicts. Then try your push again.

15.7 RStudio is not making certain files available for staging/commiting

Do you have a space in your directory or file names? A space in a file name is a space in your soul. Get rid of it.

Is your Git repo / RStudio Project inside a folder that … eventually rolls up to Google Drive, DropBox, Microsoft OneDrive, or a network drive? If yes, I recommend you move the repo / Project into a plain old directory that lives directly on your computer and that is not managed by, e.g., Google Drive.

If you cannot deal with the two root causes identified above, then it is possible that a more powerful Git client (chapter 9) will be able to cope with these situations. But I make no promises. You should also try Git operations from the command line.

15.8 I hear you have some Git repo inside your Git repo

Do not create a Git repository inside another Git repository. Just don't.

If you have a genuine need for this, which is really rare, the proper way to do is is via submodules.

In STAT 545, we certainly do not need to do this and when we've seen it, it's been a mistake. This has resulted in the unexpected and complete loss of the inner Git repository. To be sure, there was more going on here (cough, GitHub Desktop client), but non-standard usage of Git repos makes it much easier to make costly mistakes.

Part III

More

Chapter 16

More content

more stuff *coming soon* … really!!

placeholders and notes

16.1 New project

Revisit and expand "GitHub first, then RStudio" example presented here.

16.2 R Markdown + GitHub workflow

Test drive R Markdown but embrace the Git/GitHub aspects.

Push HTML and show how unsatisfactory that is. Push markdown and show how nice that is.

Optional: R script example and properly commented YAML.

16.3 Existing project

~~Create an RStudio project, then make it a Git repo, then make a GitHub repo, then connect them.~~ *oh, let's not*

Easier though less elegant: create repo on GitHub, clone via RStudio, then copy your stuff in.

What if it's an existing project that is also a Git repo with a history you care about? Then you have to do it properly. Using the "RStudio first, then GitHub" template from here: Test connection between RStudio and GitHub.

16.4 Clone a project

For when you just want a copy of something. But also want to track its evolution, which would not be true if you just, say, downloaded the ZIP archive.

I create a repo (or use one created above). They clone. I make a new commit. They pull to get it.

More practice: Do this once with … your favorite R package? The polygraphing film data (or something else that is not an R package)?

Need to explain when to do this versus fork.

How to keep a fork updated. The browser only method. The 2nd remote method.

16.5 The repeated amend

local only, i.e. never amend a commit that's already pushed

affecting a remote, explain why so dangerous and when it *might* be ok

16.6 Disaster recovery

Practice burn it all down.

Editing most recent commit or just its message.

Rebase avoidance techniques.

Headless state. Rebase hell.

http://stackoverflow.com/questions?sort=votes

16.7 Engage with R source on GitHub

Browsing

Searching

- My gist, re: the cran user: https://gist.github.com/jennybc/4a1bf4e9e1bb3a0a9b56
- Recent search for roxygen template usage in the wild: https://github.com/search?utf8= &q=man-roxygen+in:path&type=Code&ref=searchresults

Being a useful useR

- stay informed re: development
- use issues for bug reports, feature requests
- make pull requests

16.8 Workflow and psychology

Stress of working in the open

Workflows for group of 1, 2, 5, 10.

Appendix A

The shell

Even if you do most of your Git operations via a client, such as RStudio or GitKraken, you must sometimes work in the shell. As you get more comfortable with Git, you might prefer to do more and more via the command line.

A.1 What is the shell?

The `shell` (or `bash` or terminal) is a program on your computer whose job is to run other programs, rather than do calculations itself. The `shell` is a very old program and in a time before the mouse it was the only way to interact with a computer. It is still extremely popular among programmers because it is very fast, concise, and is particularly powerful at automating repetitive tasks.

Here we use the `shell` for quite modest goals: to navigate the file system, confirm the present working directory, configure Git, and configure Git remotes.

A.2 Starting the shell

In RStudio, go to *Tools > Shell*. This should take you to the shell (on Mac: Terminal, on Windows: GitBash or equivalent). It should be a simple blinking cursor, waiting for input and look similar to this (white text on black background, or black text on white background):

A.3 Using the shell

The most basic commands are listed below:

- `pwd` (**p**rint **w**orking **d**irectory). Shows directory or "folder" you are currently operating in. This is not necessarily the same as the `R` working directory you get from `getwd()`.
- `ls` (**l**i**s**t files). Shows the files in the current working directory. This is equivalent to looking at the files in your Finder/Explorer/File Manager. Use `ls -a` to also list hidden files, such as `.Rhistory` and `.git`.
- `cd` (**c**hange **d**irectory). Allows you to navigate through your directories by changing the shell's working directory. You can navigate like so:
- go to subdirectory `foo` of current working directory: `cd foo`
- go to parent of current working directory: `cd ..`

```
mars@marsmain ~ $ pwd
/home/mars
mars@marsmain ~ $ cd /usr/portage/app-shells/bash
mars@marsmain /usr/portage/app-shells/bash $ ls -al
total 130
drwxr-xr-x  3 portage portage  1024 Jul 25 10:06 .
drwxr-xr-x 33 portage portage  1024 Aug  7 22:39 ..
-rw-r--r--  1 root    root    35808 Jul 25 10:06 ChangeLog
-rw-r--r--  1 root    root    27002 Jul 25 10:06 Manifest
-rw-r--r--  1 portage portage  4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r--  1 portage portage  5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r--  1 portage portage  6151 Apr  5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r--  1 portage portage  5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r--  1 portage portage  5643 Apr  5 14:37 bash-4.0_p10-r1.ebuild
-rw-r--r--  1 portage portage  6230 Apr  5 14:37 bash-4.0_p10.ebuild
-rw-r--r--  1 portage portage  5648 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r--  1 portage portage  5532 Apr  8 10:21 bash-4.0_p17.ebuild
-rw-r--r--  1 portage portage  5660 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r--  1 root    root     5660 Jul 25 09:43 bash-4.0_p28.ebuild
drwxr-xr-x  2 portage portage  2048 May 30 03:35 files
-rw-r--r--  1 portage portage   468 Feb  9 04:35 metadata.xml
mars@marsmain /usr/portage/app-shells/bash $ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
<use>
  <flag name='bashlogger'>Log ALL commands typed into bash; should ONLY be
    used in restricted environments such as honeypots</flag>
  <flag name='net'>Enable /dev/tcp/host/port redirection</flag>
  <flag name='plugins'>Add support for loading builtins at runtime via
    'enable'</flag>
</use>
</pkgmetadata>
mars@marsmain /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
mars@marsmain /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esams.wikimedia.org (91.198.174.2) 56(84) bytes of data.

--- rr.esams.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.820/49.820/49.820/0.000 ms
mars@marsmain /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /boot
/dev/sda2               none
/dev/sda3               /
mars@marsmain /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
mars@marsmain /usr/portage/app-shells/bash $ lsmod
Module                  Size  Used by
rndis_wlan             23424  0
rndis_host              8696  1 rndis_wlan
cdc_ether               5672  1 rndis_host
usbnet                 18688  3 rndis_wlan,rndis_host,cdc_ether
parport_pc             38424  0
fglrx                2388128  20
parport                39648  1 parport_pc
iTCO_wdt               12272  0
i2c_i801                9380  0
mars@marsmain /usr/portage/app-shells/bash $
```

Figure A.1:

- go to your "home" directory: `cd ~` or simply `cd`
- go to directory using absolute path, works regardless of your current working directory: `cd /home/my_username/Desktop`. Windows uses a slightly different syntax with the slashes between the folder names reversed, `\`, e.g. `cd C:\Users\MY_USERNAME\Desktop`.
  - Pro tip 1: Dragging and dropping a file or folder into the terminal window will paste the absolute path into the window.
  - Pro tip 2: Use the `tab` key to autocomplete unambiguous directory and file names. Hit `tab` twice to see all ambiguous options.
- Use arrow-up and arrow-down to repeat previous commands. Or search for previous commands with `CTRL` + `r`.

A few Git commands:

- `git status` is the most used git command and informs you of your current branch, any changes or untracked files, and whether you are in sync with your remotes.
- `git remote -v` lists all remotes. Very useful for making sure `git` knows about your remote and that the remote address is correct.
- `git remote add origin GITHUB_URL` adds the remote `GITHUB_URL` with nickname `origin`.
- `git remote set-url origin GITHUB_URL` changes the remote url of `origin` to `GITHUB_URL`. This way you can fix typos in the remote url.
- *we should add more*

Appendix B

References

Bibliography