Reproducible Research via R, LaTeX, and knitr

James Balamuta

Department of Statistics
University of Illinois at Urbana-Champaign
balamut2@illinois.edu

June 15, 2016
Department of Statistics

On the Agenda

1. Reproducible Research
   - Definition
   - Workflow
2. Tools of Reproducible Research
   - Overview
   - Setting up the Workspace
3. Dynamic Documents
   - .Rmd documents
   - Features of `knitr`
   - Dynamic Generation
4. Extra
   - Explanation of Git Commands
   - Helpful Collaboration Tips

Ready?

On the Agenda

What is Reproducible research?

Reproducible Research is the idea that the experiment's collected data, data analysis code, and derived principal results are assembled in a way so that another body is able to re-create all of the results (e.g., data formatting, parameter estimates, figures, tables, and so on).

In essence, reproducible research seeks to satisfy a very minimal portion of how to obtain *replicable* results championed by the scientific theory.

Reproducible vs. Replicable

In general, there are lots of papers that debate what the definitions of Reproducible and Replicable are.

For our purpose, we will consider the viewpoint of Prof. Roger Peng of the Journal of Biostatistics - held as the Journal's standard - and echoed by Prof. David Banks, former editor of JASA.

Reproducible if there is a specific set of computational functions/analyses (usually specified in terms of code) that exactly reproduce all of the numbers in a published paper from raw data.

Replicable if you perform the exact same experiment (at least) twice, collect data in the same way both times, perform the same data analysis, and arrive at the same conclusions.

Why Practice Reproducible Research?

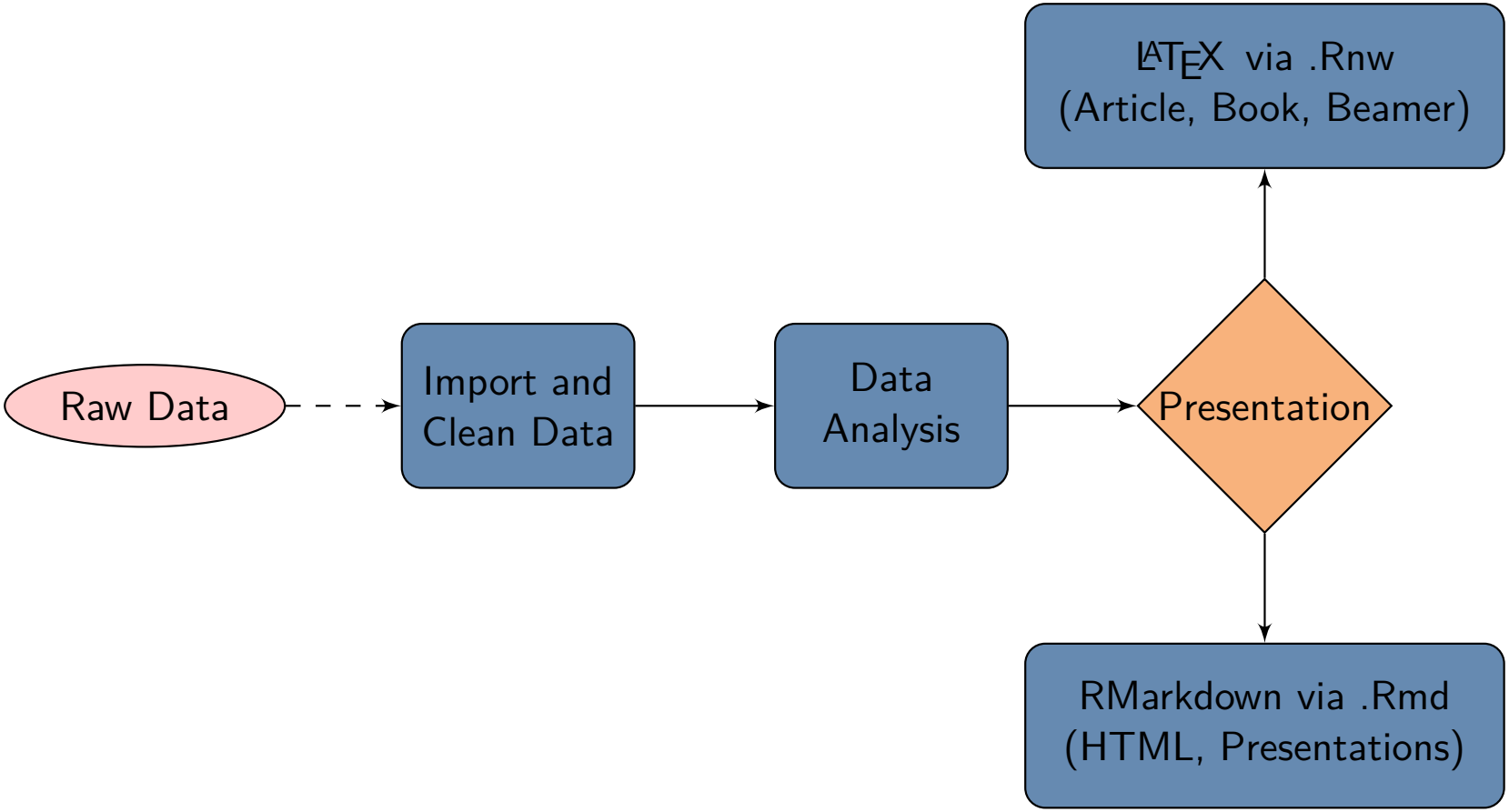
Many issues have arisen over the recent years regarding the validity of the results published in Journals.

By structuring research so that it is reproducible, not only is the work more useful and acceptable to Journals but also the overload on the researcher is reduced.

The overload is reduced since the hope of reproducible research is to put an end to the practice of copying and pasting results into documents, asymmetric data modifications in excel, and undocumented code.

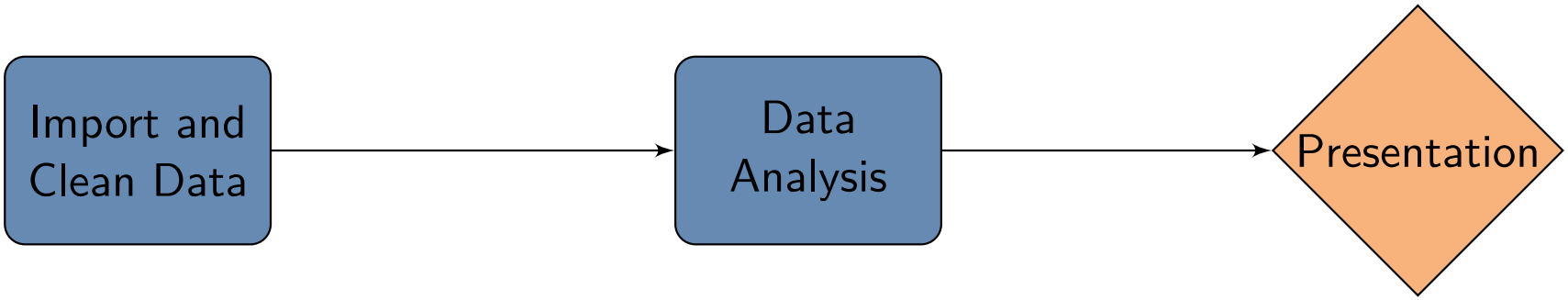
As they say...
If you do something *by hand* once, you'll end up doing it at least 20 times.

Ideal Work Flow



Only raw data exists outside of the ecosystem.
All blue boxes are done with a script to ensure reproducibility.

The Power of Scripts



- Modifications are documented
- Uniformly applied cleaning methods
- Resiliency to wrong data version

- Perform analysis like normal, but...
- No need to export figures or tables
- Code is **reusable** between projects

- Figures and tables are already created!
- Analysis changed? **Auto-updates!**
- Results are shareable and customizable

The Best Reason to Practice Reproducible Research...

Chances are your closest collaborator is yourself from three months ago, who conveniently does not reply to emails.

On the Agenda

Software of Reproducible Research

The following software programs are key to Reproducible Research:

- R - Programming Language
- R Studio - Integrated Developer Environment for R
- LaTeX - Your Favoriate Distribution of LaTeX
- pandoc - Swiss Army Knife of Document Conversion
- git - Version Control System (VCS)

R Packages used in Reproducible Research

Within R, there are several packages oriented at enabling Reproducible Research. The list below is not extensive. For more options, see the CRAN Task View on Reproducible Research.

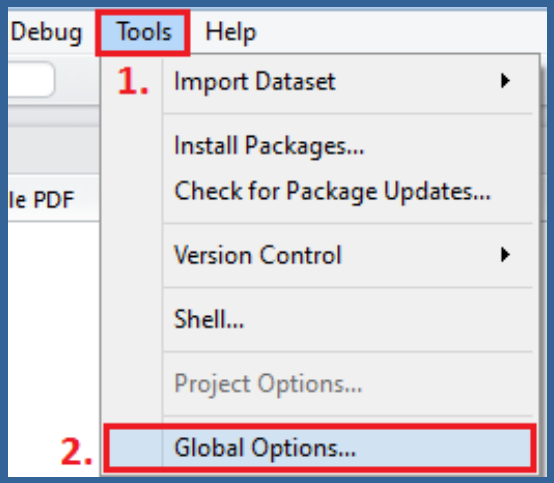
For the presentation, I will focus on the following R packages:

- knitr - Dynamic Report Generation
- rmarkdown - R interface to Pandoc

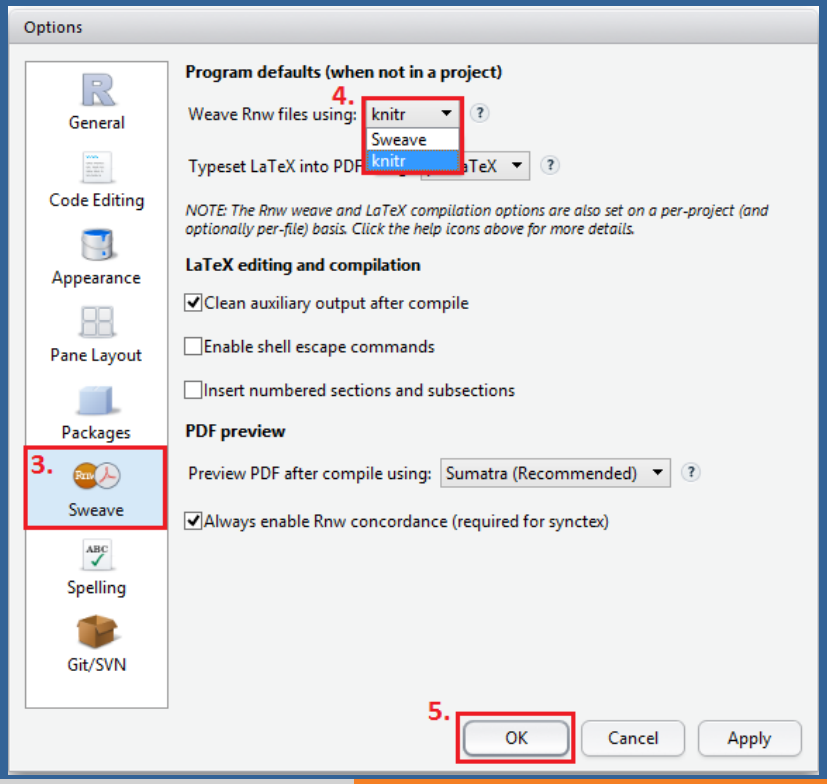
Configuring RStudio to build `knitr` documents

Make sure that `knitr` is install before continuing (install.packages("knitr"))

Click 'Tools'
Select 'Global Options'



Click 'Sweave'
Select 'knitr' from drop down menu



Initializing the Workspace

The Checklist:

1. Make sure everything is up-to-date (R, R Studio, and R Packages)
   - Depending on how sensitive the analysis is, consider using packrat to create project based libraries for r packages.
2. Create a repository on GitHub
3. Setup an R project in R Studio based on the repository.
4. Place project directory in either Dropbox or BoxSync folder.
   - Make sure to turn off new file notification...

STATS@UIUC Analytical Environment

Starting in the **Spring of 2016**, the Department of Statistics at the University of Illinois has had an online analytical environment. The primary purpose of this analytical environment is to try out various approaches to working with technology in Statistics.

`https://rstudio.stat.illinois.edu`

To login, please use your NetID and Active Directory Password or what you use to log into Compass2g.

The instructions given next are for users who opt to use their own computer instead of the platform.

Obtaining git on Windows

Unfortunately, unlike macOS and Linux, Windows does not currently have native support for git. This will change when `bash` is added to the command line.

Until then, to have `git` on your system you must download it from http://git-scm.com/downloads.

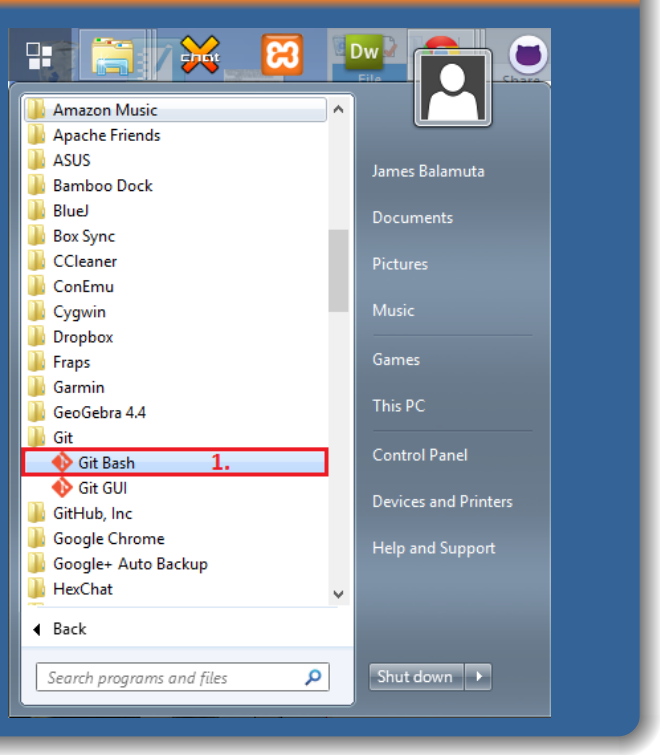
Furthermore, during the installation, **make sure you select the option that places git.exe on your system's PATH variable.**

Both of these steps are omitted here.

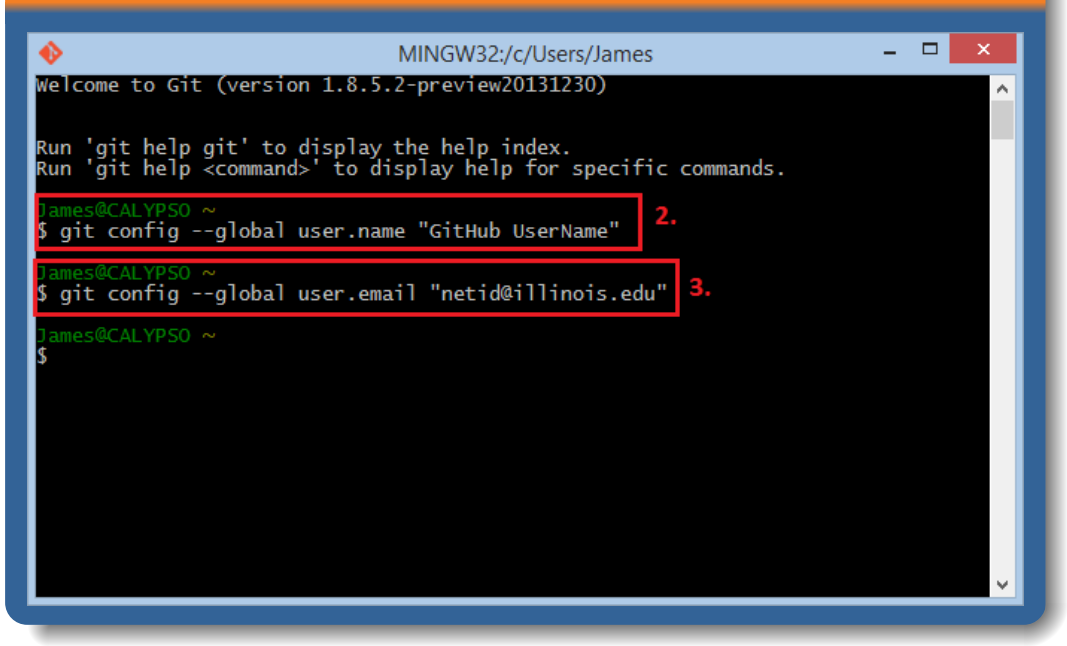
Configuring git for the first time

Once git is installed, you will have to provide an initial configuration scheme.

Open Git Bash from Start Menu

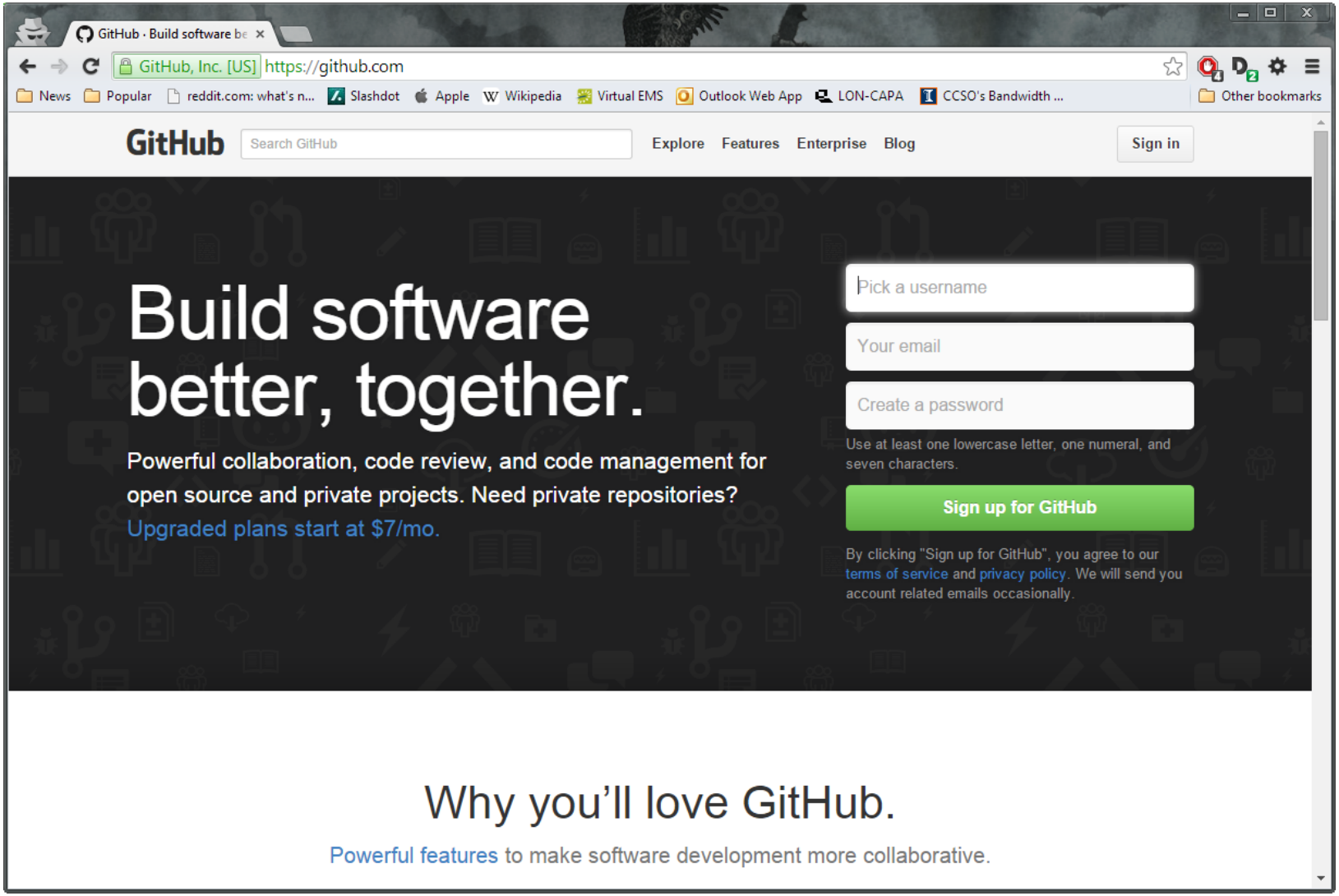


Enter the following two commands using your information



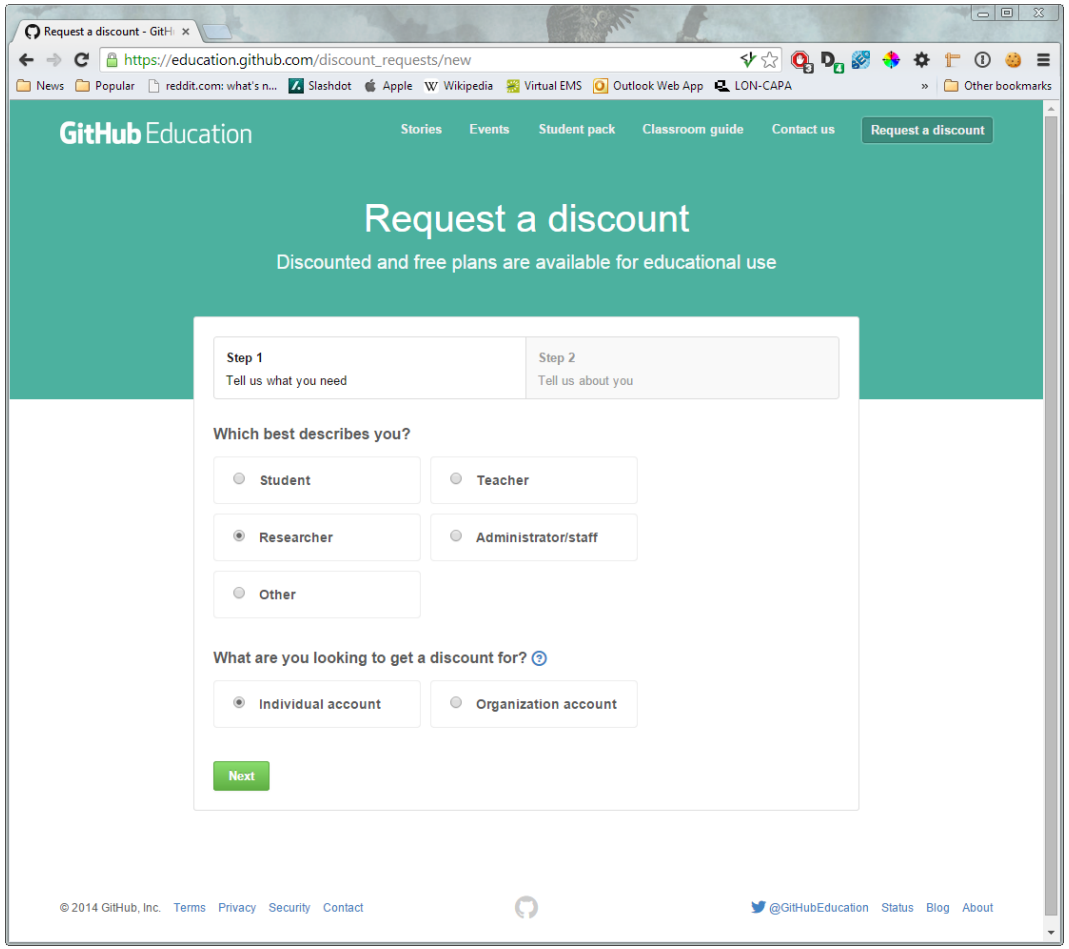
Create a GitHub Account

Register an account on GitHub using your @illinois.edu e-mail address.



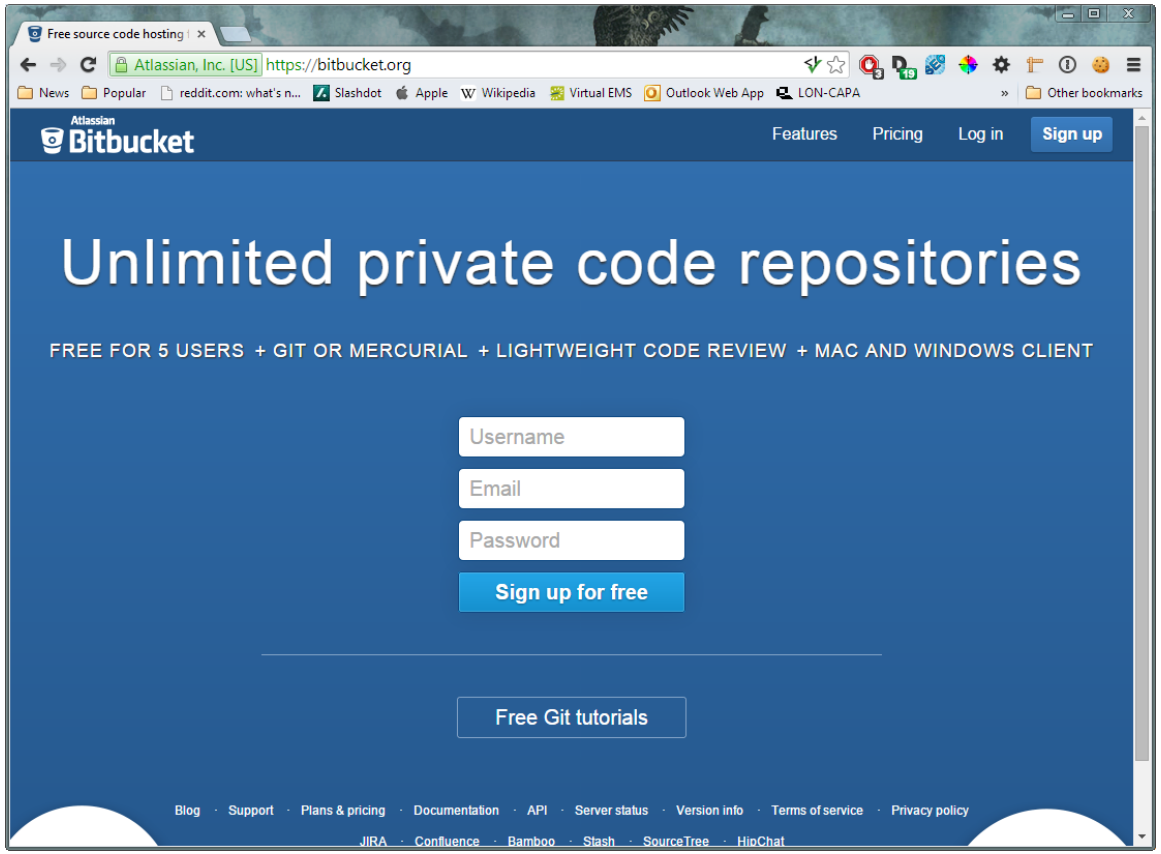
Request a Student Account

Place a request using GitHub Education to obtain **unlimited** free private repositories. Also, consider forming an organization, which is able to receive **unlimited** free private repositories with fine permission control.

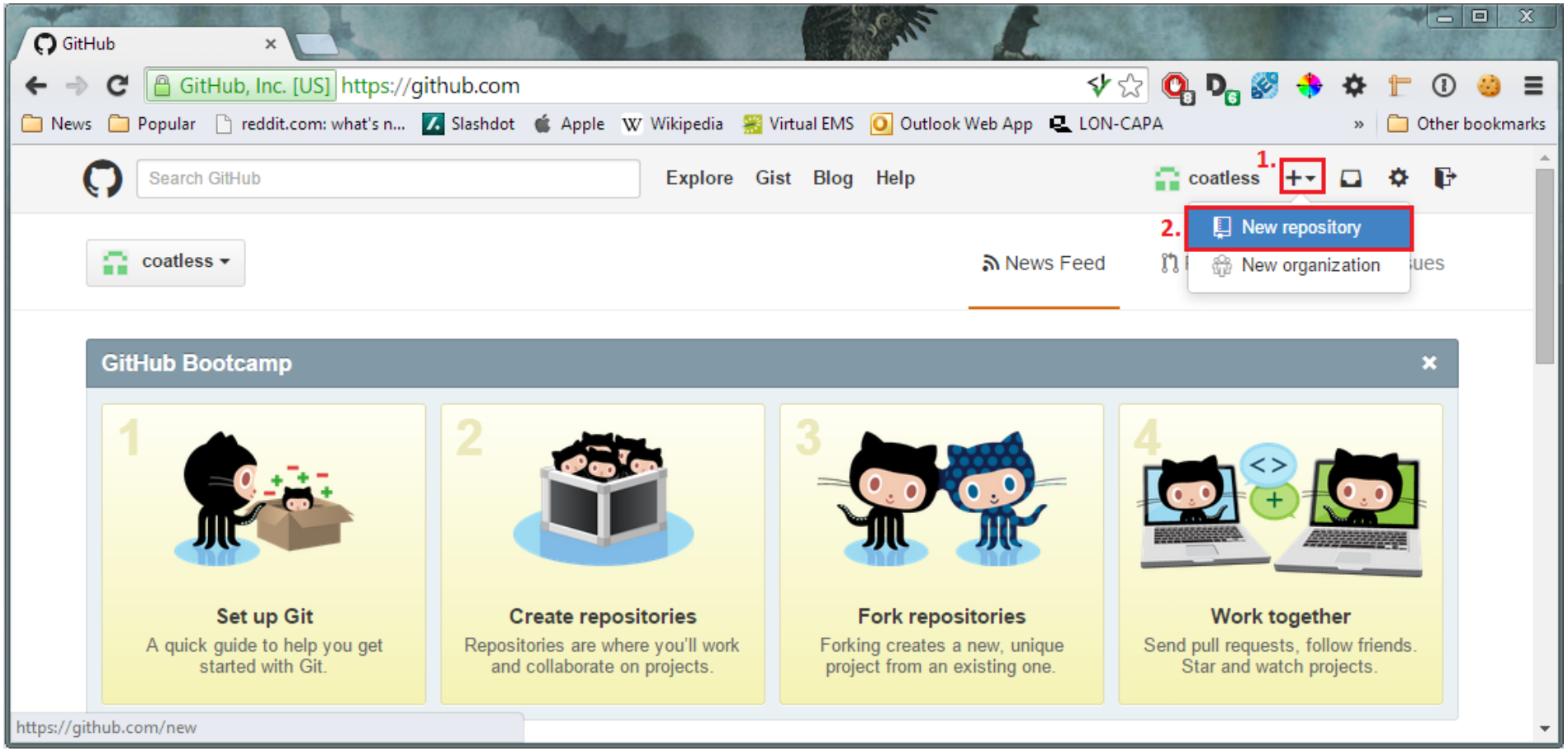


BitBucket: An alternative to GitHub

Dislike GitHub? Consider using BitBucket. The repositories are generally more closed source than GitHub. However, for researchers and students, BitBucket provides unlimited private repositories with unlimited contributors.

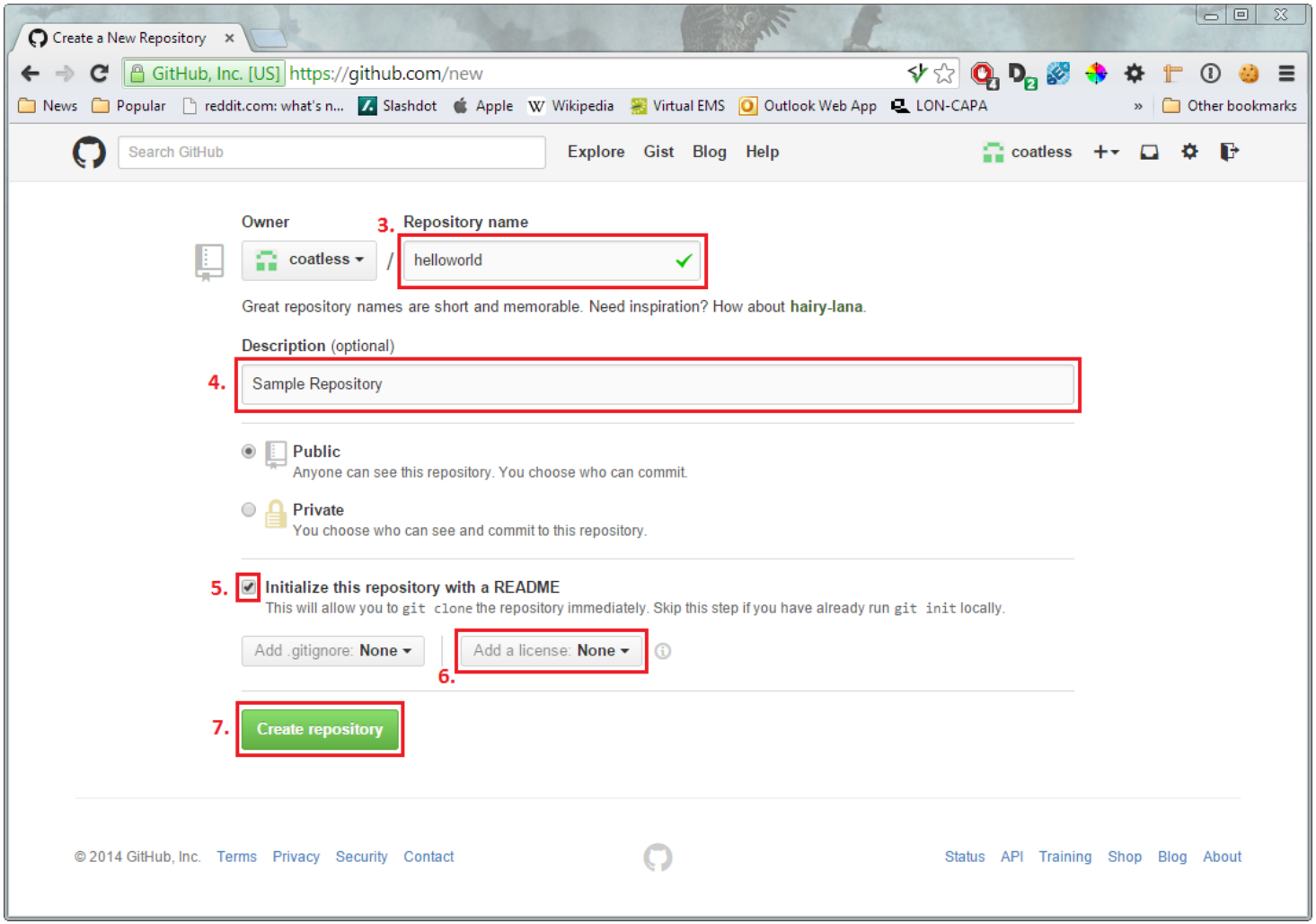


Create a GitHub Repository

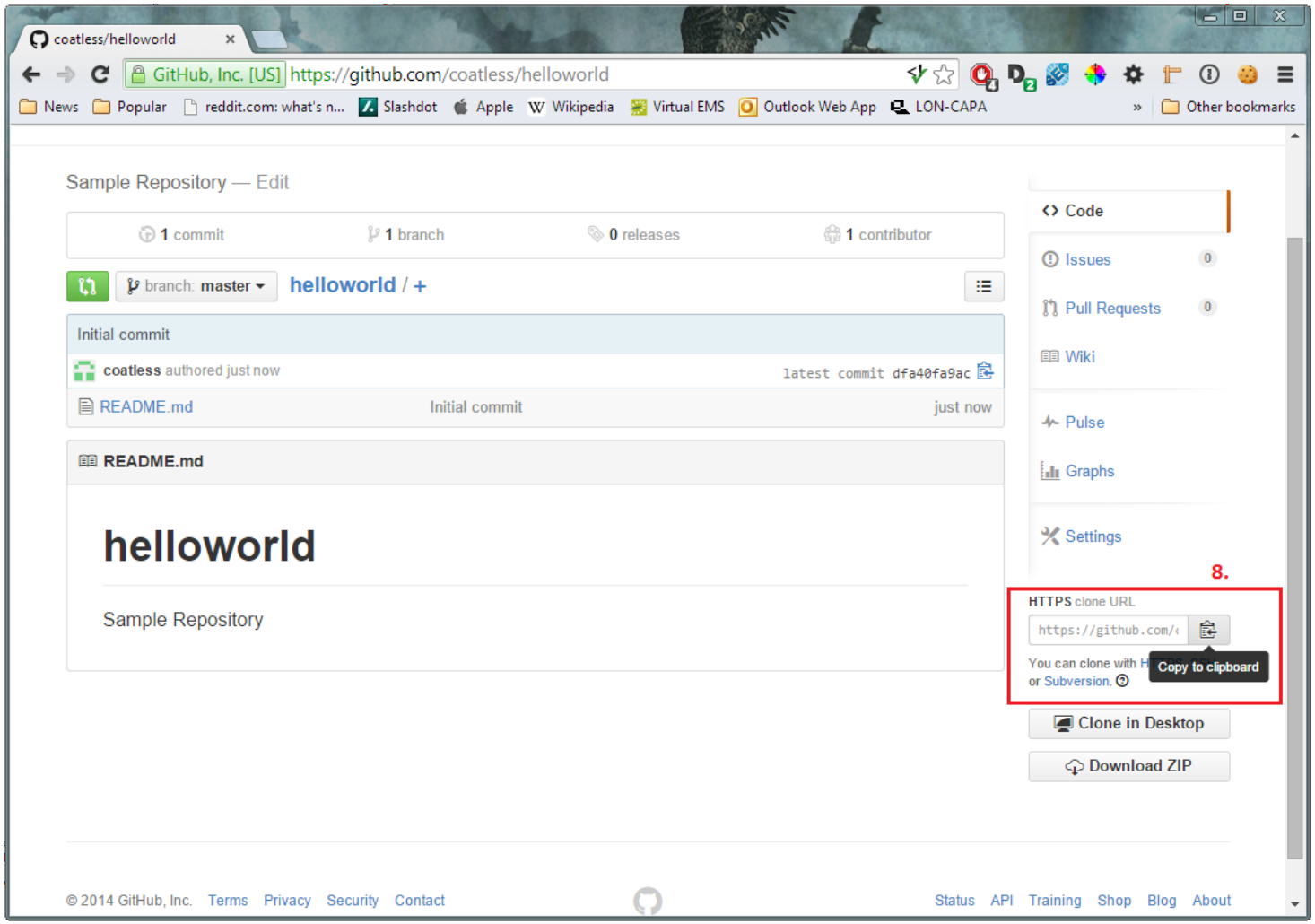


Create a GitHub Repository

When establishing the repository, consider adding a license to your code.

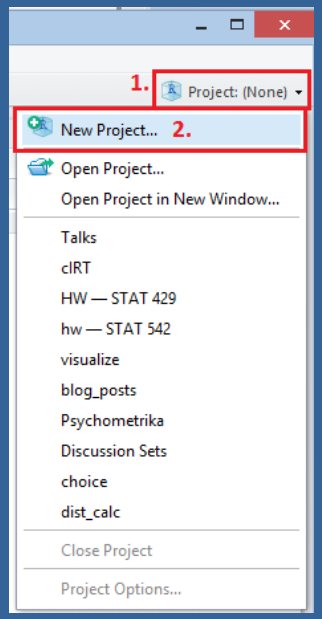


Obtain GitHub Repository Link

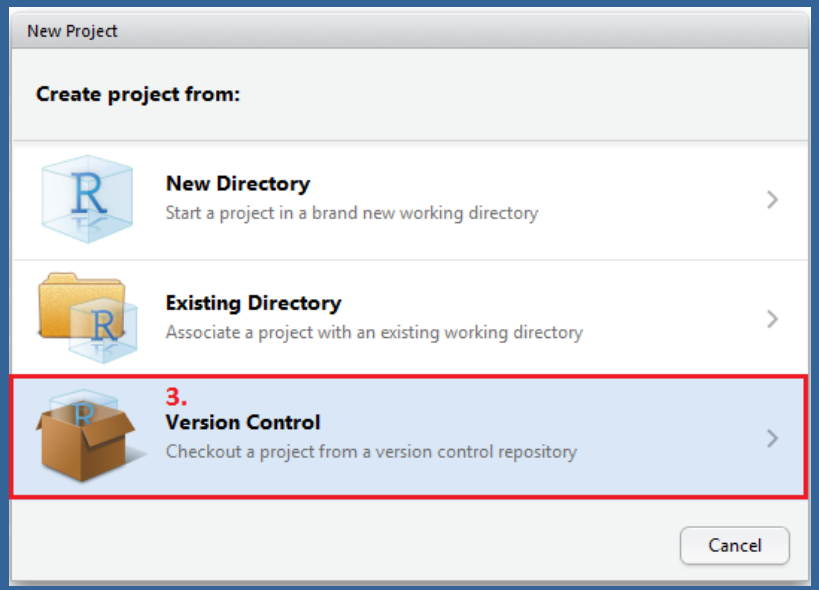


Create a new R Studio Project

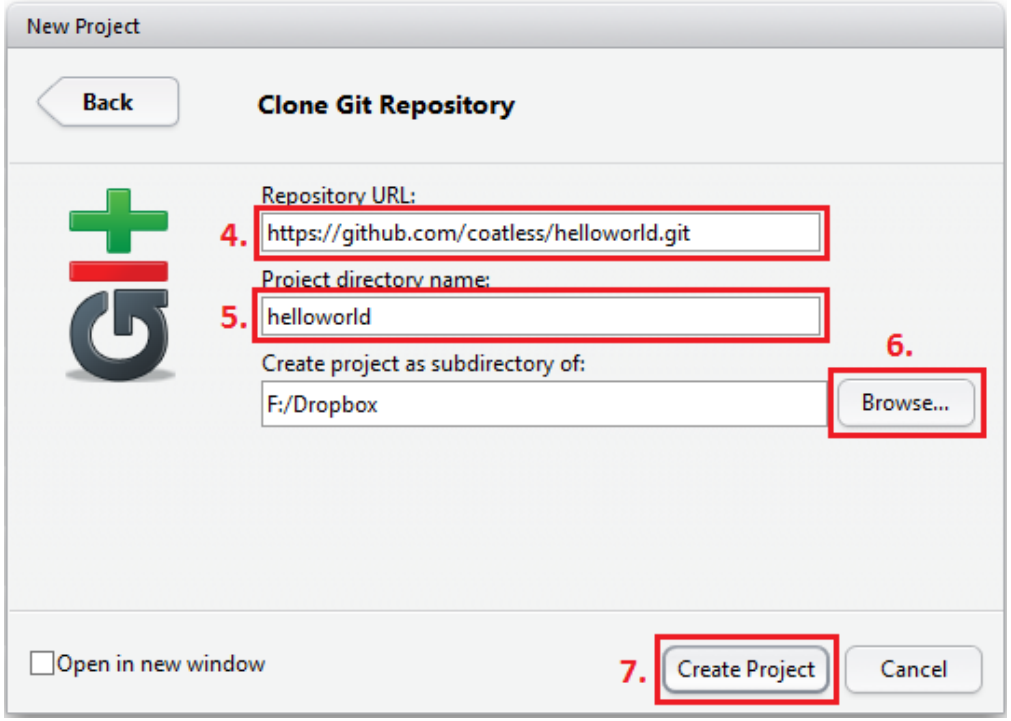
Open Project Menu Select 'New Project'



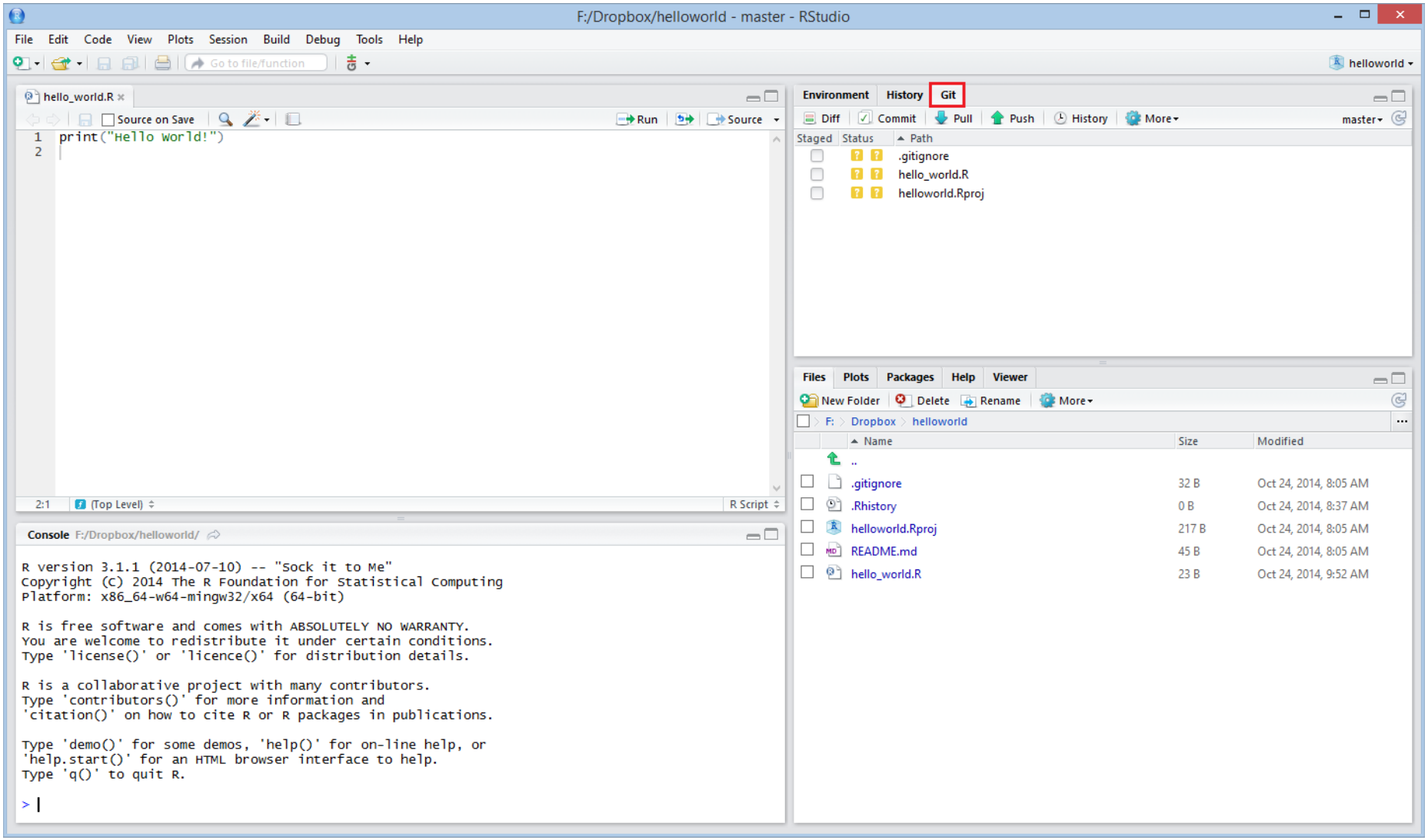
Select Version Control



Create a new R Studio Project

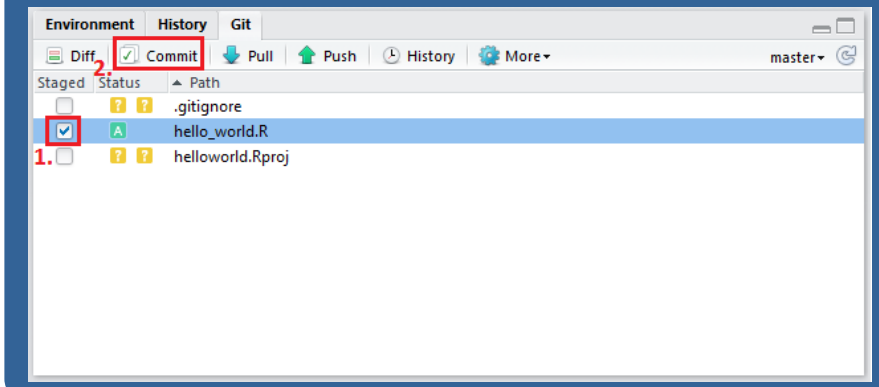


R Studio - Project View

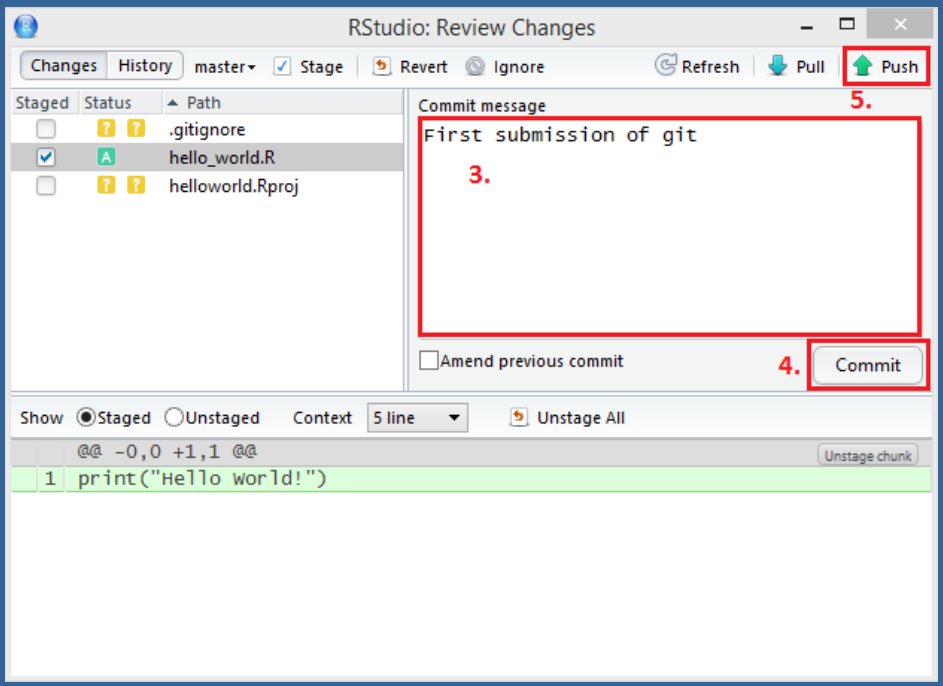


Using git via R Studio

Select file to be staged
Press 'Commit'

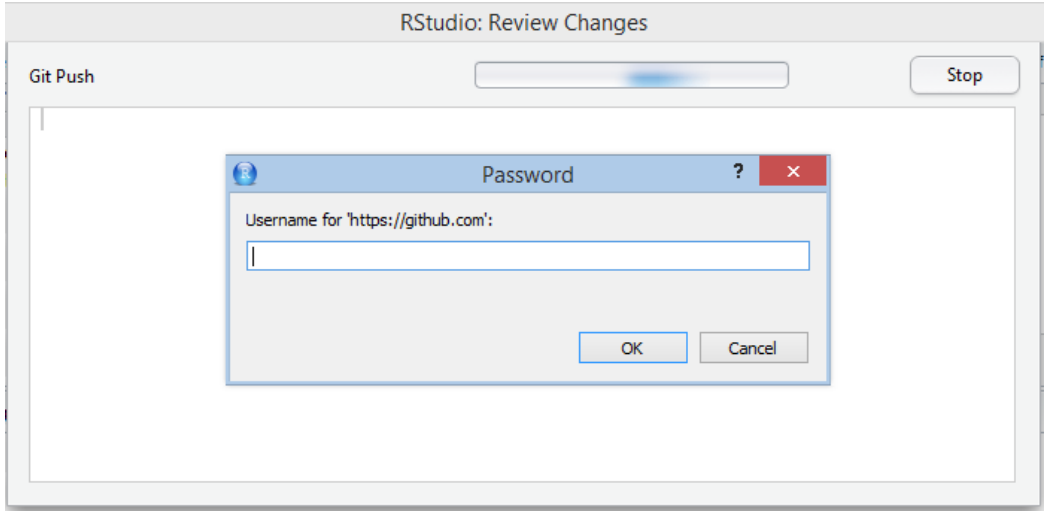


Enter commit message
Press 'Commit'
Press 'Push'



Authorizing...

One of the downsides of pushing to GitHub is the need to be authenticated...



Good Authorization via Public and Private SSH Keys

To simplify this process, we opt to authorize ourselves to GitHub using an encryption technique that uses a **public and private key scheme**.

In this case, the public key is a string of symbols that is out there for the world to see.

On the other hand, the private key is considered to be confidential and only viewable by its owner.

This scheme enables messages signed with the private key to only be viewable via its public key (and vice versa).

Good Authorizing - SSH Key Generation

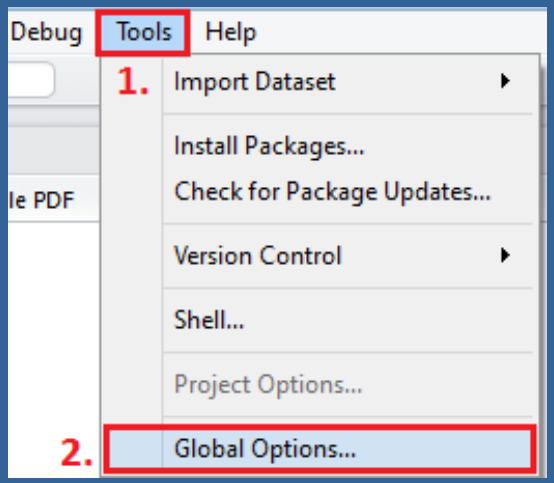
Instructions for generating an SSH key are given in two flavors:

1. Using RStudio's GUI
2. Using Shell/Terminal

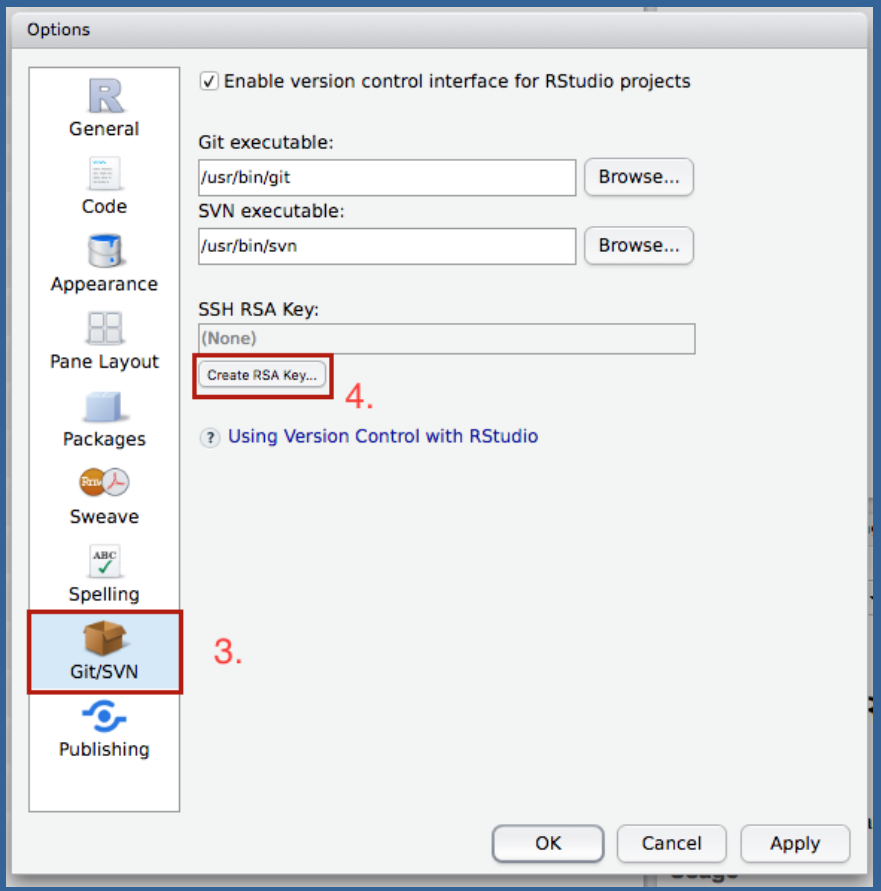
You only need to pick **one** route.

Good Authorizing - SSH Key Generation via RStudio

Click 'Tools'
Select 'Global Options'

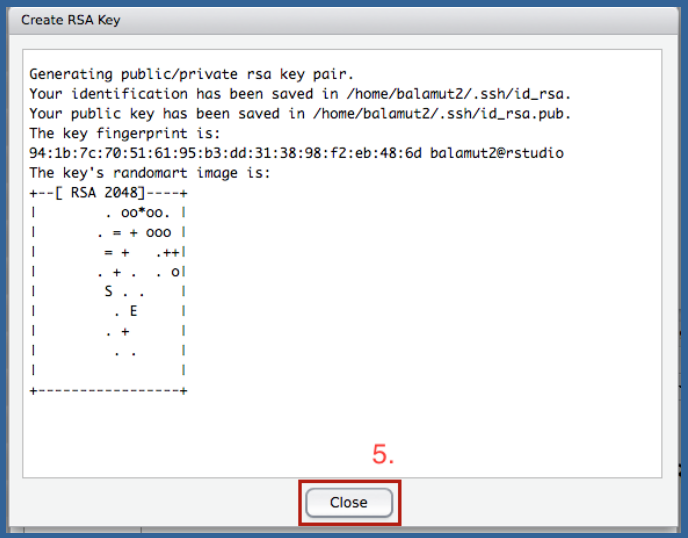


Click 'Git/SVN'
Click the 'Create RSA Key...'

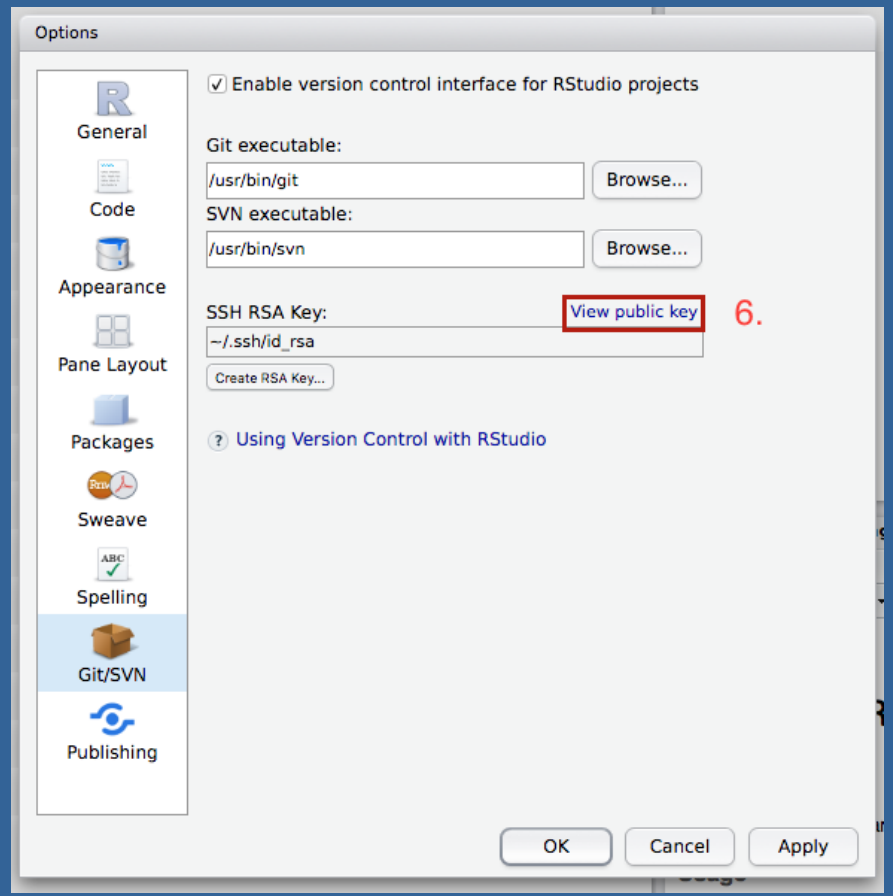


Good Authorizing - SSH Key Generation via RStudio

Click 'Close'

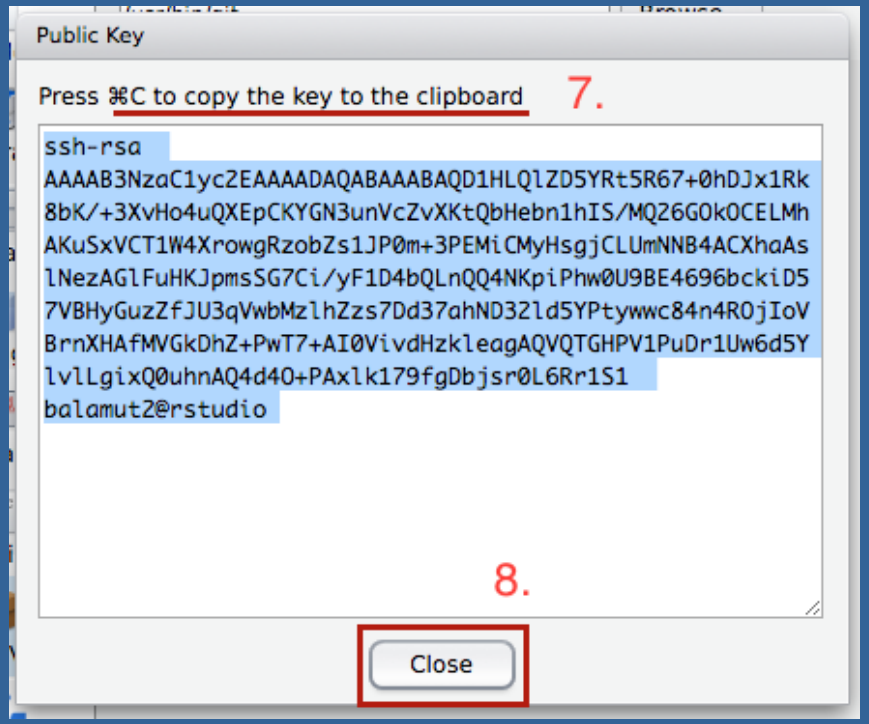


Click 'View Public Key'



Good Authorizing - SSH Key Generation via RStudio

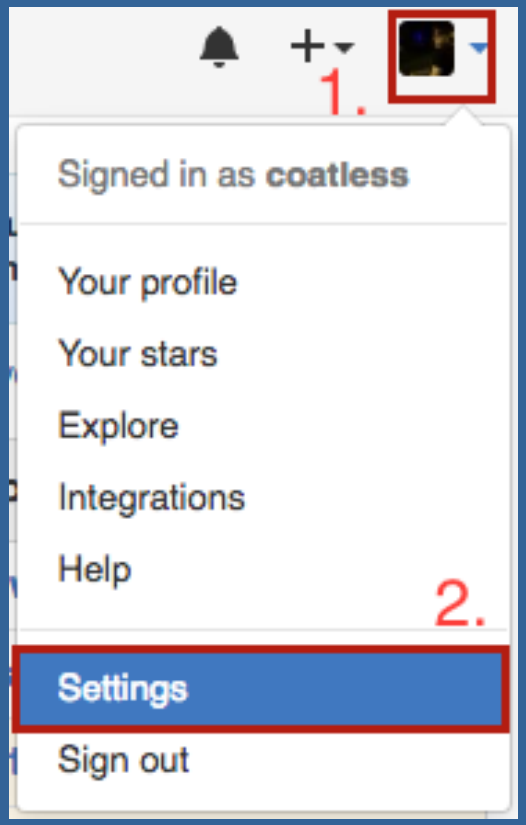
Copy the public key with either
macOS: `Command` + `C` or Windows: `Cntrl` + `C`



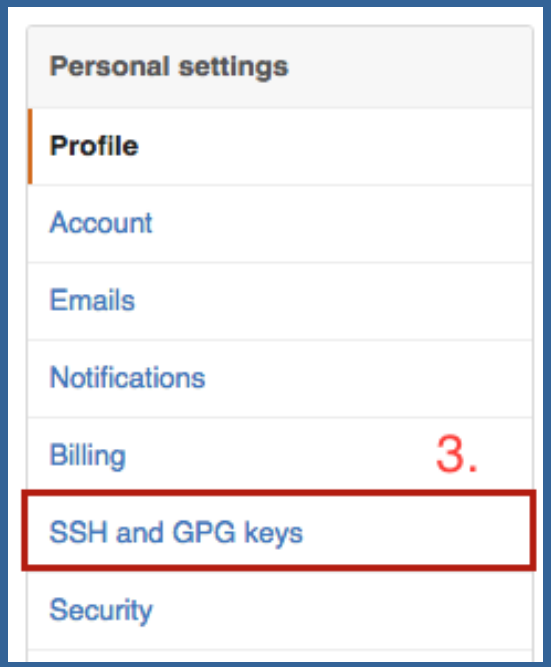
With the key being generated and on our clipboard, we can now add it to our GitHub account...

Good Authorizing - Add Public SSH Key to GitHub

Click on the picture in the upper right corner
Select ‘`Settings`’

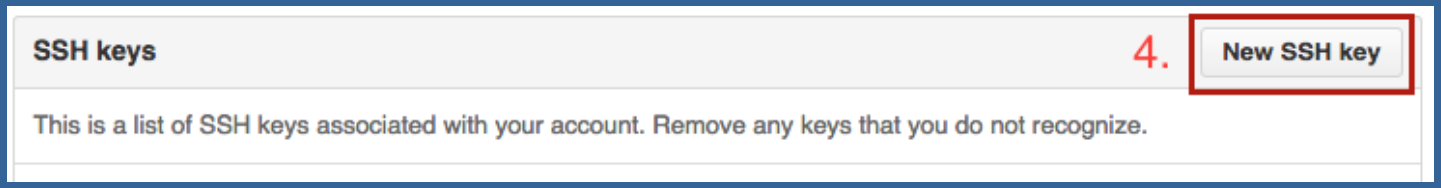


Click ‘`SSH and GPG Keys`’

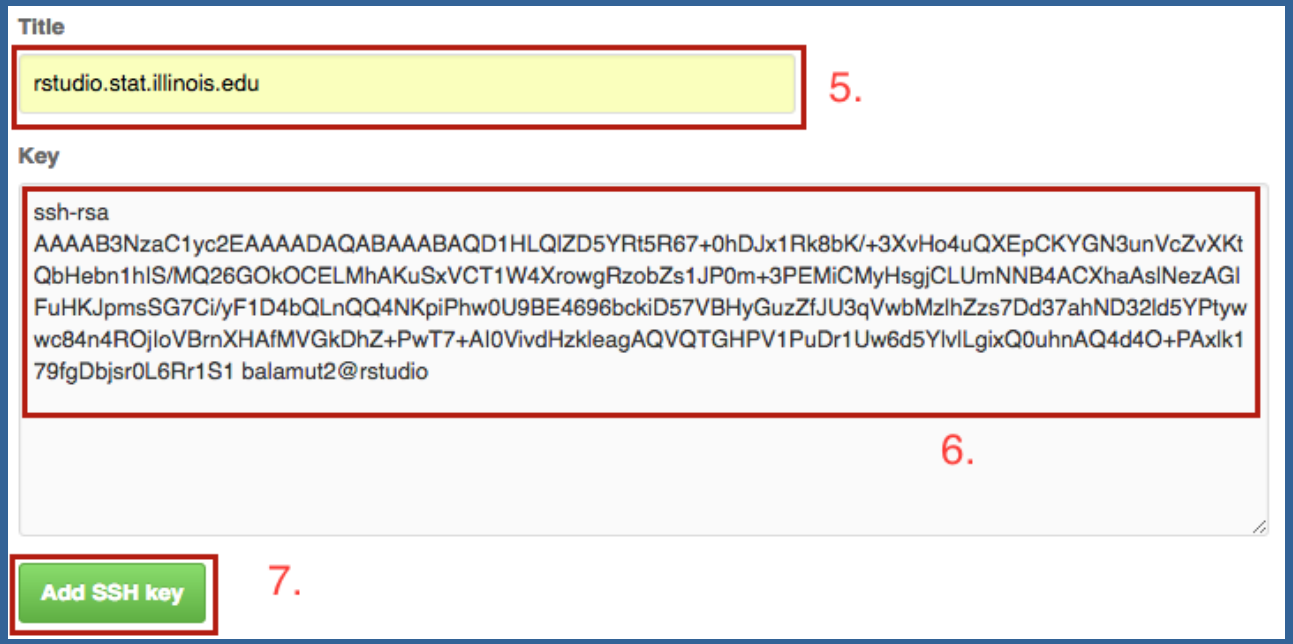


Good Authorizing - Add Public SSH Key to GitHub

Press the 'New SSH Key' button



Fill in the 'Title' input and paste the SSH into 'Key' with either macOS: Command + V or Windows: Cntrl + V



Good Authorizing - Add Public SSH Key to GitHub

Check that the key has been added

If you have made it to this step, well done!

You can now officially commit via SSH.

Good Authorizing - SSH Key Generation

As promised, next up are the instructions for SSH key generation within Terminal/Shell...

Do **NOT** repeat this if you opted for the RStudio approach.

Good Authorizing - SSH Authorization via Terminal

Use SSH Authorization by opening terminal/shell (in RStudio access it via Git tab's More $-$ > Shell)

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com" # Creates new SSH
Key

# After the key is generated, you will be prompted with
Enter a file in which to save the key (/Users/you/.ssh/id_rsa): [Press
enter]

# Create a password for the key (needs to be rememberable)
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]

# Start the SSH Agent
eval "$(ssh-agent -s)"

# Add the key
ssh-add ~/.ssh/id_rsa
```

Copy key to clipboard
Windows:

```
clip < ~/.ssh/id_rsa.pub
```

macOS:

```
pbcopy < ~/.ssh/id_rsa.pub
```

Then we add the key to GitHub using the previous steps...

Bad Authorizing - Use this approach when everything fails

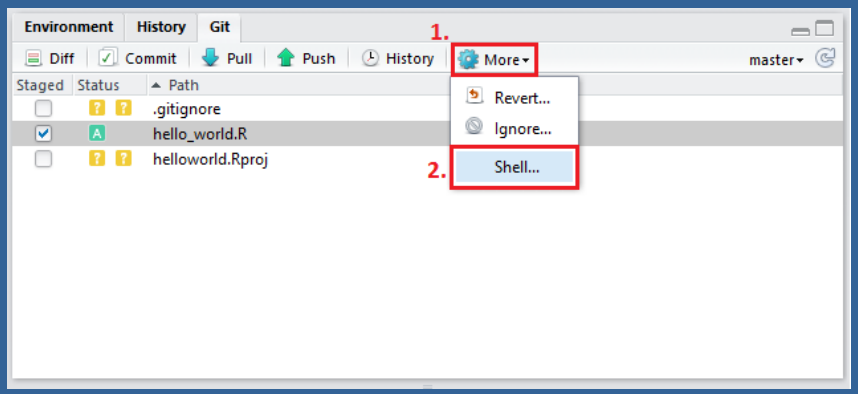
This is a very bad option since R Studio does NOT hash .proj files.

When you enter in your GitHub repository to create your R Project, append your username and password like so to the URL:

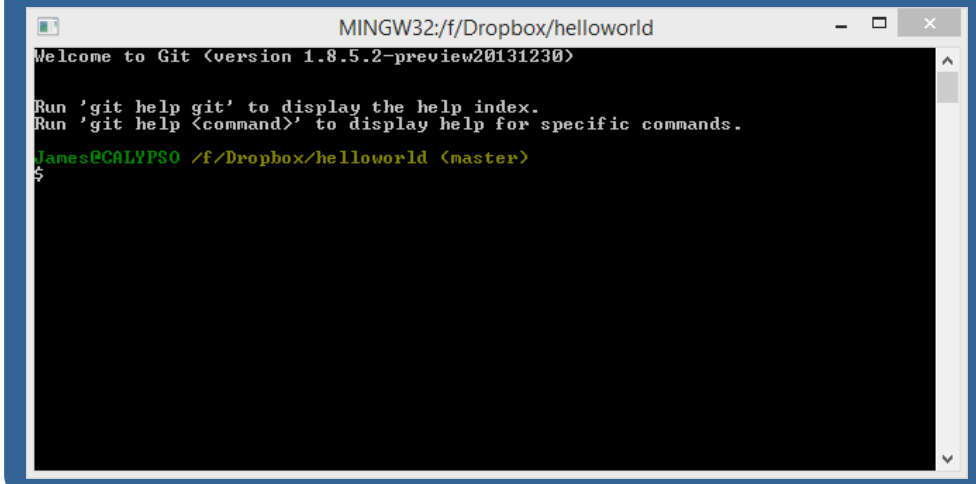
```
https://username:password@github.com/username/helloworld.git
```

Using git via shell

Select 'More'
Click 'Shell'



Type the commands on the next slide



Using git via shell

```
# Start a new feature
git checkout master      # Switch to master
git branch new_world     # Create a development branch
# or: git checkout -b new_world master

# Add a new file
git add hello_world.R
git commit -m "First submission of git"

# Merge in the new_world branch
git checkout master       # Switch to master
git merge new_world       # Merge in change
git branch -d new_world   # Remove development branch

# Push update
git push origin master
```

On the Agenda

Markdown

Markdown allows you to write a file format independent document using an easy-to-read and easy-to-write plain text format.

In essence, instead of marking up text similar to HTML:
e.g. <html><body>Name</body></html>

The goal is to mark down text to the simpliest form:
e.g. **Name**

As a result of documents being structured so loosely, any file format can really be applied to the rules using pandoc.

The downside is that there is less control over formatting.

R Markdown

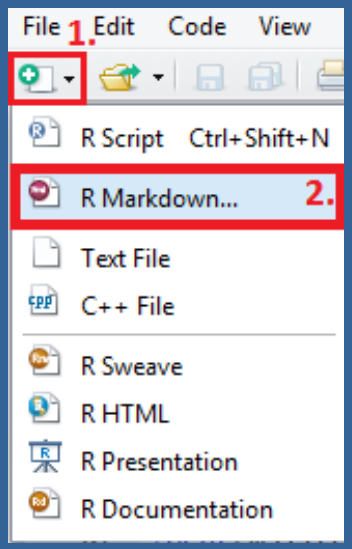
R Markdown developed by RStudio takes what Markdown has established and extends it 10x fold.

- Merge R code with Markdown
- Export options quadrupled
- R Markdown documents are fully reproducible
- So many more extras vs. the original Markdown

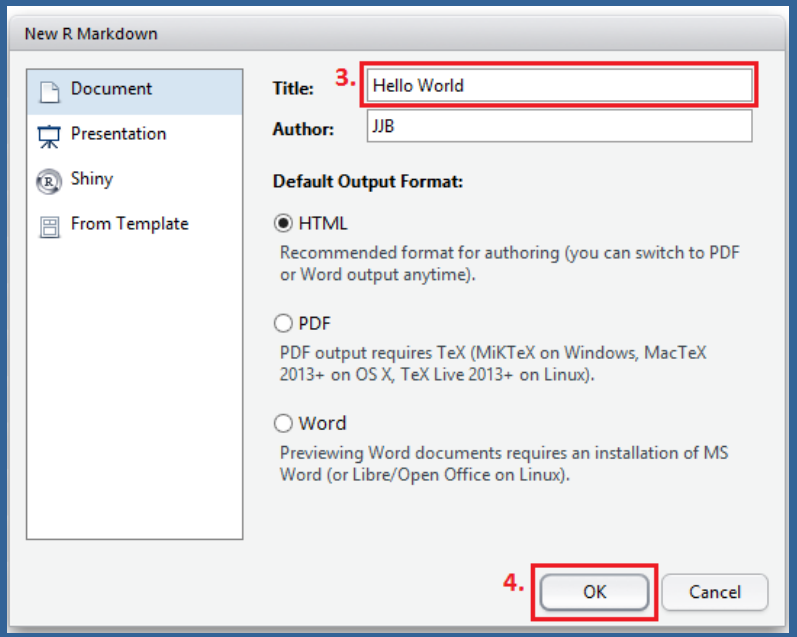
Creating a .Rmd Document

To create a .Rmd Document within R Studio:

Click the White Plus Select 'R Markdown'



Enter Document Title



.Rmd View

Untitled1

Knit HTML Run Chunks

```
---
title: "Hello World"
author: "JJB"
date: "Friday, October 24, 2014"
output: html_document
---

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r}
summary(cars)
```

You can also embed plots, for example:

```{r, echo=FALSE}
plot(cars)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
```

2:1 (Top Level) R Markdown

Text Features of RMarkdown

markdown_sample.Rmd

Knit HTML Run

```
# Header 1

This is an R Markdown document. Markdown is a simple formatting
syntax for authoring HTML, PDF, and MS Word documents. For more
details on using R Markdown see <http://rmarkdown.rstudio.com>.

Use an asterik mark, to provide emphasis such as *italics* and
**bold**.

Create unordered lists with a dash or star:

- Item 1
- Item 2

Created ordered lists with a number followed by a period:

1. Item 1
2. Item 2

You can write `inline` code with a back-tick.

```
Code blocks
for larger chunks
```

> Blockquotes are great
```

~/BoxSync/stat385/lectures/lec3/markdown_sample.html

markdown_sample.html Open in Browser Find Publish

Header 1

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

Use an asterik mark, to provide emphasis such as *italics* and **bold**.

Create unordered lists with a dash or star:

- Item 1
- Item 2
- Item 3

Created ordered lists with a number followed by a period:

1. Item 1
2. Item 2
3. Item 3

You can write `inline` code with a back-tick.

```
Code blocks
for larger chunks
```

> Blockquotes are great

Code Chunks, this time for .Rmd

To initiate a code chunk within .Rmd, all one needs to do is use:

```
```{r chunk_label}
# Code here
```
```

Example:

```
Here is a sample way of usings chunk options
```{r chunk_label, eval=FALSE}
opts_chunk$set(warning=FALSE, message=FALSE)
set.seed(1337)
```
More text...
```

Chunk options are the same between .Rnw and .Rmd

In-line R code for .Rmd

.Rmd has the ability to write inline using: ‘r expression_here‘

Example:

```
Did you know that there are `r dim(apples)[1]` observations
and `r dim(apples)[2]` variables contained within the apples
data set?
```

Note:

- All inline commands must be on the same line (no returns)!
- Helpful to have code chunk hidden before paragraph and use inline features to control output.

Output options

The key to .Rmd's strength is in the output statement at the front of the document. There are so many options to customize the final output.

```
---
title: "Sample Document"
author: "James Balamuta"
date: "June 16, 2016"
output:
  pdf_document:
    toc: true                # Table of Contents
    toc_depth: 2             # Nested 2 levels
    number_sections: true # Number Sections
    highlight: "zenburn"  # Code Highlighting
    keep_tex: true           # Retain .tex file used for .pdf
---
```

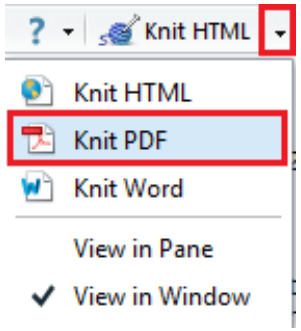
Outputing multiple files from a single .Rmd

```
---
title: "Sample Document"
author: "James Balamuta"
output:
  html_document:
    toc: true              # Table of contents
    theme: cerulean        # Bootstrap theme
  pdf_document:
    toc: true              # Table of contents
    keep_tex: true         # Retain .tex file used for .pdf
  word_document:
    fig_width: 5           # Set figure width
    fig_height: 5          # Set figure height
    fig_caption: true      # Output captions with figures
---
```

Compiling .Rmd to .*

Compile via R Studio to .html document: 

Switching to a different output format (pdf in this case):



Compiling .Rmd to .*

Compile .Rmd via r console...

Single line output declarations

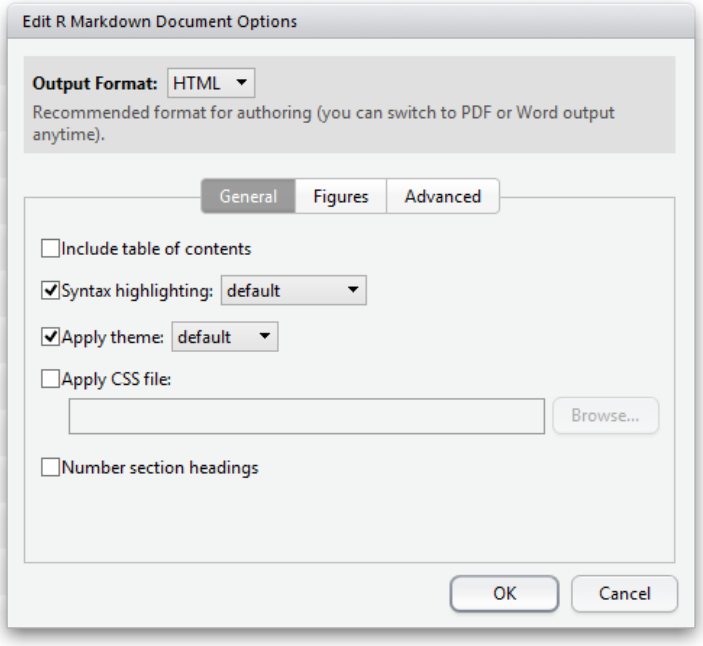
```
library(knitr)
knit2html('input.Rmd') # Creates .html file
knit2pdf('input.Rmd')  # Creates .pdf file
knit('input.Rmd')      # Creates .md file
```

Or use for multiline output declarations

```
library(rmarkdown)
render("input.Rmd", "pdf_document")
render("input.Rmd", "word_document")
render("input.Rmd", "md_document")
```

Options... Options... Options...

Some of .Rmd's output options can be configured via a GUI in R Studio:



To see all the options granted by .Rmd, check out the package website at: http://rmarkdown.rstudio.com/.

Fun Application of .Rmd

Create a new blog post on a Wordpress site using R:

```
# Check to see if RWordPress is install and install if necessary.
if (!require('RWordPress'))
  install.packages('RWordPress', repos = 'http://www.omegahat.org/R', type = 'source')

# Load Library
library(RWordPress)
# Pass user credentials
options(WordpressLogin = c(Username = 'password'),
        WordpressURL = 'http://publish.illinois.edu/<netid>/xmlrpc.php')

# Initialize knitr
library(knitr)

# Uploads r generated images to server. Can be switched to imgur, etc.
opts_knit$set(upload.fun = function(file){
  library(RWordPress)
  uploadFile(file)$url
})

# Push rmarkdown to wordpress site
knit2wp('file_name.Rmd', title = 'Test r post')
```

Setting Global Chunk Options

Instead of declaring options repetitively across multiple code chunks, it is better to create a global declaration in a "setup" chunk at the start of the document.

Even with a global declaration, values are able to be changed locally on code chunks on an as needed basis. The local values will not affect future code chunks.

To set global chunk settings use:

```
```{r set_global}
opts_chunk$set(eval = FALSE, comment=NA,
               fig.width=6, fig.height=6,
               fig.align='center')
```
```

Caching

Caching refers to storing data locally in order to speed up subsequent retrievals.

In essence, the code chunk will be run once, the resulting objects are then stored within a file, and the stored data is then displayed on subsequent runs.

This feature is very handy when embedding analysis on large data sets or time intensive computations.

Be Careful When Caching...

Simple yet problematic cache:

```
```{r cache_demo, cache = T}
x = rnorm(5)
```
```

```
## [1] -0.3407 -0.3130 -0.2468  0.4668  0.5254
```

The Problematic Cache

Suppose you have three chunks, A, B, and C, that have RNG and have been cached.

If chunk C is inserted between A and B, then B should be updated because RNG modifies .Random.seed as a side-effect, but the chunk B will *not* be updated

$\Rightarrow$ The reproducibility of B is bogus.

To guarantee reproducibility with RNG, .Random.seed needs to be associated with the cache for each chunk using:

```
# Set global chunk options
opts_chunk$set(cache.extra = rand_seed)
```

The Limiting Cache Scope

Sometimes the cache needs to be limited by R version, Session Info, or a time stamp.

```
# Set global chunk options
opts_chunk$set(cache.extra =
                 list(R.version,      # R version info w/ OS
                      rand_seed,      # .Random.seed
                      sessionInfo(), # Current Configuration
                      # Month timestamp
                      format(Sys.Date(), '%Y-%m'))
```

Limit Cache by File Info

Limit cache using the file stamp associated with data files being read into R for the analysis

```
#' @param files is a character vector containing filenames
#' @return time stamp
mtime = function(files){
  lapply(Sys.glob(files),function(x) file.info(x)$mtime)
}
```

```
```{r data_read_in, cache.extra=mtime("apple.csv")}
```
```

Returning to In-line R code for .Rmd

Previously, we executed code inline with an actual function call:

```
`r dim(apples)[1]`
```

In this case, it is highly preferred to move those calls to a code chunk *before* the text as this calculation can then be cached.

Code Chunk Example:

```
```{r data_inline_calc, cache = TRUE}
data("apples")       # Loads the apple data set.
obs  = nrow(apples) # Number of observations in the dataset
vars = ncol(apples) # Number of variables in the dataset
```
```

In document:

```
Did you know that there are `r obs` observations
and `r vars` variables contained within the apples
data set?
```

Compared to the previous approach, we now have:

- Cached computations
- Clearer function calls
- Less obtrusive explanations.

Code Externalization

When we first began using `knitr`, the R code was directly embedded into the .Rmd file.

This is not a good practice in general.

Why? The elements of the analysis are then merged with elements of the presentation.

If the code is externalized:

1. The code can be run without compiling the document.
2. Sharing the code with others is easier.
3. Switch between presentation slides and the code during the actual presentation.

Code Externalization in Action

To externalize code, the r code file must contain comments of the form:

---- label or ## @knitr label

, where label denotes a code chunk name.

Example Code File: analysis_code.R

```
# Label code as:

## @knitr external_code
(x = "Rawr?")

# or

## ---- external_code
(x = "Rawr?")
```

Example .Rmd File: analysis_writeup.Rmd

```
```{r external_code_load, echo=TRUE}
read_chunk('analysis_code.R') # Reads in file, parses comment
```
```

Activate Code Chunk in Document:

```
```{r external_code, echo=TRUE}
```
```

Result:

```
(x = "Rawr?")

## [1] "Rawr?"
```

Reusing Code Chunks

By labeling code chunks, `knitr` is able to call them in the future or embed them within other code chunks.

To reuse the code from <<chunk-name>>, call it from within ‘‘‘{r chunk-embedded}.

Reusing Code Chunks Example

Initial chunk:

```
```{r chunk-name}
test = "hello"
```
```

Embedded Chunk:

```
```{r chunk-embedded}
<<chunk-name>>
```
```

Result:

```
"hello"

## [1] "hello"
```

Linking to Content

To link content for Rmd files:

```
# Only the link
<http://stat385.thecoatlessprofessor.com>

# Create a link for specific text
[stat385 course material](http://stat385.thecoatlessprofessor.
```

Including Graphics

The inclusion of graphics generated outside of R follows a similar scheme as linking to external websites:

```
# On the Web
![alt text](http://example.com/logo.png)

# Locally via relative path
![alt text](figures/img.png)
```

BibTex

In R Studio, we are able to use BibTex to generate bibliographies. BibTex is a way to structure output of paper citations you can obtain from Google Scholar

Note: The biber backend for .bib files is not easily supported in R Studio.

Structuring a .bib file

Bibliography information is stored in .bib files.
Here is the mybiblib.bib used in the previous example:

```
@article{tv1981,
  title={The framing of decisions and the psychology of choice},
  author={Tversky, Amos and Kahneman, Daniel},
  journal={Science},
  volume={211},
  number={4481},
  pages={453--458},
  year={1981},
  publisher={American Association for the Advancement of Science}
}
@book{ho1987,
  title={Rational choice: Contrast between economics and psychology.},
  author={Hogarth, Robin M and Reder, Melvin W},
  year={1987},
  publisher={University of Chicago Press}
}
```

```
---
title: "Example Biblography heading"
output: html_document
bibliography: bibliography.bib
---
```

Reference in the document using:

- Authors + Year
  - "Choice is important" [@ho1987].
- Suppress author information and only have a year
  - Hogarth says, "Choice is important" [-@ho1987].

knitr's `kable()`

The main benefit to `kable()` is not having to worry about the mode `knitr` is currently in (e.g. html, latex, or markdown) and its ability to split tables over pages.

Make sure to set the chunk option `results = 'asis'`.

Demo of kable()

```
```{r results='asis'}
kable(head(iris,3),    # Data to be converted to table
      row.names=TRUE, # Show rownames
      align=c('l', 'c', 'r', 'l', 'r'), # Sets column alignmen
      digits=1         # Restrict amount of digits after decim
      )
```
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa

On the Agenda

Git Commands

Basic Git Commands:

git init:	Initializes a new git repository. (Must be run to setup repo, no repo = no commands working)
git config <option>:	Configure git options
git help <command>:	Provides information on how to use and configure a specific git command.
git status:	See what files are in the repository, what changes need to be committed, and what branch of the repository is active.

Git Commands

Feature Development Commands:

git add <file>:	Add a new or changed file or files (.) into a "staging" area.
git commit -m "Message here" :	Pushes change into the repository with message
git checkout:	A way to select which line of development you're working on. (e.g. Master or your own branch)
git branch <name>:	Build a new branch off of the active repository to make changes and file additions that are completely your own.
git merge:	Merge changes in your branch back to the master branch.

Git Commands

Syncing Git Commands:

- **git remote:** Create, view, and delete connections to other repositories.
- **git fetch:** Imports commits from a remote repository into your local repository. Helpful for reviewing changes before integrating them into the master branch.
- **git push:** Push the local changes to the repository up to the version control server.
- **git pull:** Pull the newest changes from the version control server to your local environment.

Sample git workflow

```
# Make sure repo is up to date
git checkout master
git fetch origin master
git pull --rebase origin

# Start a new feature
git checkout master
git branch AdvOfJames
# or: git checkout -b AdvOfJames master

# Add a new file
git add GiantPeach.r
git commit -m "New feature started"

# Update a file with changes
git add GiantPeach.r
git commit -m "New feature finished"

# Merge in the AdvOfJames branch
git checkout master      # Switch to master
git merge AdvOfJames     # Merge in change
git branch -d AdvOfJames # Remove development branch

# Push update
git push origin master
```

Collaboration Tips

1. Create a shared space to store all your materials
   - CITES Wiki
   - Dropbox (2.5 gigs) and BoxSync (50 gigs)
2. Group Document Editing
   - Google Drive (Unlimited Storage!!!)
   - ShareLaTeX (1 Collaborator + Supports Knitr)
   - writeLaTeX (1 gig + unlimited collaborators)
   - MS Word's Track Document Changes
3. Use a discussion board
   - Google Groups
   - Illinois Mailing Lists
4. Use a Versioning Tool
   - git
   - svn
5. Remote Communications Tools
   - Skype
   - Google Hangouts

Questions? Comments?

James Balamuta
balamut2@illinois.edu
thecoatlessprofessor.com