paradigm

Showing your work

Martin Frigaard

This book is for sale at http://leanpub.com/showingyourwork

This version was published on 2019-06-10

Contents

Getting discovered in the information age

> "*Your work should speak for itself..*" - unknown

You have entered graduate school in a fantastic time. Most people are walking around with more computational power in their pockets than previous generations ever imagined possible. An internet browser now gives you access to nearly all the accumulated knowledge of the human species (and an unreasonable number of cat pictures). As an example to how much has changed and how fast it's happened, consider a 2000 paper in Nature[1] by Steve Lawrence and C. Lee Giles titled, "Accessibility of information on the web." The authors open with the jaw-dropping statistic that the internet is "800 million pages, encompassing about six terabytes of text data on about 3 million servers." Fast forward to 2016, and Google claims to be aware of 130 trillion pages[2] across the web.

Put simply; we've never had more access to information than we do right now. And it's unlikely that there will be less available information in the future.

I know how the internet works–why are you telling me this?

Because you need your work to be discoverable **on the internet**, you want collaborators, future employers, other graduate students, and (most importantly) future 'you' to be able to find a catalog of what you've been doing all those late nights in front of your computer.

Isn't that what my thesis/dissertation is for?

Be honest–*how many theses/dissertations have you read?* Ask your most bibliophilic friends what their favorite thesis is, or what dissertation they think everyone *must* read? These documents aren't a waste of time–they serve a different purpose (and it's not to make sure all of your hard work reaches a broad audience).

Good thing I have peer-reviewed publications!

Peer-reviewed articles are usually written for a very niche audience, and rarely in a way that makes the contents interesting beyond only those researchers closest to the subject matter. These papers are still essential to advancing science (and your career as a scientist), but they represent the end of a long process in which you've demonstrated many different skills (reading and summarizing previous research, designing a study, data analysis, and communication). This sentiment is summarized well in the quote from Jonathan Buckheit and David Donoho at Stanford,

> "*An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete*

[1] https://www.nature.com/articles/21987
[2] https://searchengineland.com/googles-search-indexes-hits-130-trillion-pages-documents-263378

> *software development environment and the complete set of instructions which generated the figures.*"

And like most advertising, peer-reviewed papers leave many essential details out. For example, journal articles can't document how you ended up at the particular research question or hypothesis you tested (and why you had to change it). Neither the thesis/dissertation nor the peer-reviewed paper can document what you did in graduate school.

- coming up with an idea,
- turning that idea into a research question,
- convincing people of that idea (or some version of it),
- collecting your data,
- teaching labs/lectures,
- recruiting volunteers,
- entering your data,
- managing your committee member's expectations (and egos),
- cleaning your data,
- persuading someone to read early drafts,
- politely reminding your committee to give you constructive criticism (promptly so you can graduate), and
- solving a million other little day-to-day problems.

A paper and set of slides won't capture most of these experiences, so if you want your work to count, it needs to documented somewhere else.

Showing your work

This book will show you how to make your work more discoverable. We'll introduce you to the technologies, methods, and places used by scientists who have successfully communicated their work. These scientists have used the internet as a tool to reach broader audiences, document some of their daily struggles/successes, and share more about what it means to conduct research.

Won't people find my work boring–who would want to read this?

The best science writers capture their audience by weaving science into a compelling narrative. Carl Sagan, Mary Roach, Freeman Dyson, Jared Diamond–all of these authors have a unique talent for making complicated, intricate scientific topics enjoyable by engaging us with the people in the story. By entering graduate school and doing your research, you're now one of the people in that story. No one is born with an ability to write well–it takes a lot of practice and feedback. The more you communicate with different audiences about your research, the better you'll get at finding an ability to convey its importance.

Lastly, science is a method, not a product. That process of how you found what you found is the most critical part of your research because it's the part that tells us 1) why we can trust what you published, and 2) how we can try to reproduce your findings.

So when you think about it, all those early drafts of your proposal, the lines of code you've used to wrangle your data, the graphs, tables, and figures you've used to learn about the distributions and relationships between your variables, the random notes, and epiphanies, etc…*these artifacts are the real 'science'*.

To quote Roger Peng[3] from the Department of Biostatistics, in the Johns Hopkins Bloomberg School of Public Health,

> "the data and code used to make a finding are available and they are sufficient for an independent researcher to recreate the finding."

When we were kids in math class, the teacher would ask us to "Show our work." Teachers gave these instructions so they could follow our thought process through a problem. Communicating our work should be the goal of everyone doing research. If you think about it, our job isn't done as researchers when we defend our thesis/dissertation and get the degree; it isn't done when our research has been submitted and accepted to a conference or high-impact journal; it isn't even done when someone reads the article or attends the talk.

As researchers, our jobs are done when someone has understood our research and the impact it will have on the world.

---

[3]https://science.sciencemag.org/content/334/6060/1226

Getting set up

All of the tools in this book are available completely free. The reason we recommend using open-source software is the communities behind these tools. You'll find a massive network of support on Stackoverflow[4], RStudio Community[5], and Google Groups[6].

To follow along with this book, you'll need to download and install R, RStudio, and Git. I will cover this process assuming you have none of these on your machine. This section will include downloading installing R and RStudio, and basic commands in Terminal.

Using R & RStudio

What is R?

R[7] is a free statistical modeling software application and language.

What is RStudio?

RStudio[8] is an integrated development environment (IDE) for using R.

RStudio is a free and open-source integrated development environment[9] (IDE) for R. You should explore different IDE's on your own–you'll see there are many options, both paid and unpaid. These applications typically come with a code editor (with syntax highlighting), a graphical/drag-and-drop tools, and some debugging display.

Other examples of IDEs are DataGrip[10] for relational data, Spyder IDE[11] for Python, or Stata[12]. *These are not free.*

Getting set up with R/RStudio

1. Download and install R from CRAN[13]
2. Download and install RStudio[14], the integrated development environment (IDE) for R

[4]https://stackoverflow.com/questions/tagged/r
[5]https://community.rstudio.com/
[6]https://groups.google.com/forum/#!forum/r-help-archive
[7]https://www.r-project.org/
[8]https://www.rstudio.com/products/RStudio/
[9]https://en.wikipedia.org/wiki/Integrated_development_environment
[10]https://www.jetbrains.com/datagrip/
[11]https://www.spyder-ide.org/
[12]https://www.stata.com/why-use-stata/
[13]https://cran.r-project.org/
[14]https://www.rstudio.com/products/rstudio/download/

3. An alternative to downloading and installing RStudio is using RStudio.Cloud[15] which operates entirely in your browser. You'll need to sign up for RStudio.cloud for free using your Google account or email address, but we recommend using a Github account. You can create a Github account here[16]

In the next section, we'll cover using the command line, version control, and syncing RStudio with Github.

Notes

- The scientific journal industry is not looking out for your best interests. They have a clearly unethical business model[17], even prominent universities can't afford their prices[18] (which means fewer people reading your work), and they won't compensate[19] you for your efforts.
- The metrics previously used to measure success in academic publishing are unreliable and susceptible to being gamed[20]. You don't want to have these be your sole measure of productivity.
- Read more about the reproducibility crisis in science here[21], and why it's so important.

[15]https://rstudio.cloud/
[16]https://github.com/join
[17]https://www.theguardian.com/commentisfree/2011/aug/29/academic-publishers-murdoch-socialist
[18]https://www.theguardian.com/science/2012/apr/24/harvard-university-journal-publishers-prices
[19]https://whyevolutionistrue.wordpress.com/2011/09/01/the-racket-of-academic-publishing/
[20]https://academic.oup.com/gigascience/article/8/6/giz053/5506490
[21]https://www.nature.com/news/1.19970

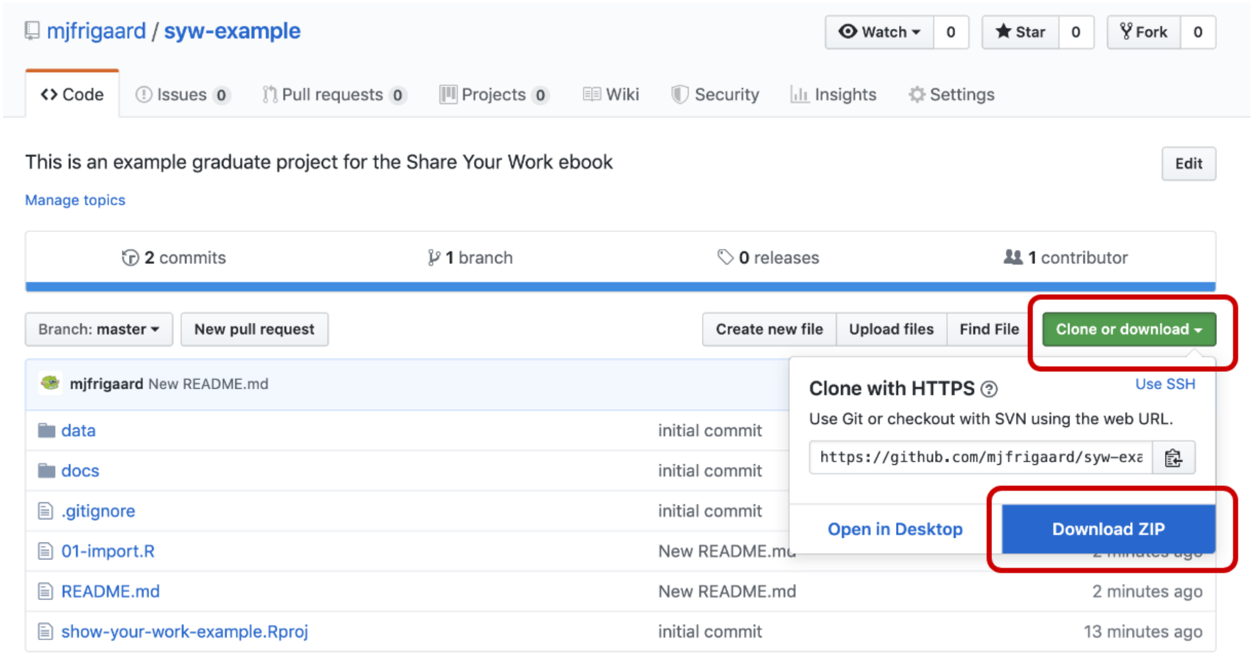
Taking command of the command line

This chapter covers some essential computer science topics, terminology, and how they've changed modern research methods.

An example project

To help guide you through learning these technologies, we've made a code repository of some files typically found in a research project. The files in this repository were used to create this master's thesis[22] and this peer-reviewed publication[23].

Download these files by clicking on the green icon and downloading the zip file.



This is an example graduate project for the Share Your Work ebook

Use Git or checkout with SVN using the web URL.

`https://github.com/mjfrigaard/syw-exa`

mjfrigaard New README.md

data	initial commit	
docs	initial commit	
.gitignore	initial commit	
01-import.R	New README.md	
README.md	New README.md	2 minutes ago
show-your-work-example.Rproj	initial commit	13 minutes ago

Put the zipped file in a recognizable place (like the `Documents` folder or on your `Desktop`). Unzip the folder and examine its contents. We'll be using these files throughout the text.

[22] http://csuchico-dspace.calstate.edu/handle/10211.3/10211.4_387
[23] https://journals.sagepub.com/doi/abs/10.1177/1941406412470719

The language barrier

"*You must learn to talk clearly. The jargon of scientific terminology which rolls off your tongues is mental garbage.*" - Martin H. Fischer

The most substantial barrier to understanding new disciplines or technologies is getting a handle on the jargon. Because this book sits at the intersection of computer science, statistics, and web technologies, the vocabulary can often seem like learning a foreign language.

Wherever possible, I'll do my best to clear up or define any terms related to computer science, data management system, web technology, or statistics. To maximize the power of the tools in this text, it will help to know a little about their history, so we'll also cover some background.

Computers and science

Just about every field of science also has a 'computational' area or journal to accompany it. Archaeologists[24] use computers to study geographical information systems (GIS) data and simulate human behavior. Chemists[25] use data and simulation to determine the arrangements and features of molecules and particles, or to estimate binding affinities for drug molecules on a given receptor or target. Biologists[26] use computers to build models and simulate biological, ecological, behavioral, and social systems. The list goes on and on…

- Economics[27]
- History[28]
- Finance[29]
- Linguistics[30]
- Law[31]
- Sociology[32]

Computers and their peripheral technologies help researchers in almost every discipline research, record, analyze, review, and publish their work. So odds are if you're doing research in graduate school, you'll be using your computer a lot.

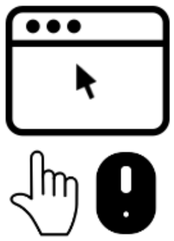
I already know how to use my computer–I do it everyday…

[24]https://en.wikipedia.org/wiki/Computational_archaeology
[25]https://en.wikipedia.org/wiki/Computational_chemistry
[26]https://en.wikipedia.org/wiki/Computational_biology
[27]https://en.wikipedia.org/wiki/Computational_economics
[28]https://en.wikipedia.org/wiki/Computational_history
[29]https://en.wikipedia.org/wiki/Computational_finance
[30]https://en.wikipedia.org/wiki/Computational_linguistics
[31]https://en.wikipedia.org/wiki/Computational_law
[32]https://en.wikipedia.org/wiki/Computational_sociology

Most people interact with their computers using a graphical user interface[33] or GUI (pronounced ‘gooey’). GUI’s are quick and easy to learn because the operating system or software application environment usually mimics an actual physical space (i.e., desktops, folders, documents). If a new task is needed, an additional software application gets installed in this virtual environment to perform that specific function. Below is a list of standard computer tasks, and the associated software GUIs (point-and-click):

- Browsing the internet: Chrome, Safari, and Firefox
- Word processing (articles & reports): Microsoft Word, Apple iWork Papers, and Google Docs
- Composing emails: Microsoft Outlook or Apple Mail
- Building presentations: Microsoft PowerPoint or Apple iWork Keynote
- Creating spreadsheets for numerical calculations to organize data: Microsoft Excel, Apple iWork Numbers, or Googlesheets

In a command line interface, lines of text are used to communicate intentions to the computers



In a graphical user interfaces (GUI), a mouse, trackpad, or finger is used to communicate intentions to the computers.

Users interact with a GUI using a mouse, track-pad, or touchscreen. These devices serve as digital appendages for transmitting intentions to their computers, whether this means opening an application by clicking on it, deleting a file by dragging it into a virtual trash bin, pinching fingers together to zoom in on an image, etc.

Having a user-centered design[34] has made software applications (and other technologies) available to a broader range of people, and reduced many of the frustrating experiences many of us had in the early days of computing.

But all the benefits of GUIs come with a cost. Creating applications and operating systems that encourage clicking around until you can figure out how to get things done sounds harmless, but it also presents challenges for automation and keeping track of everything the user does in a GUI. Furthermore, most GUIs come with a limited collection of possible operations a user can choose from (all of which were selected by the designer of the software).

[33]https://en.wikipedia.org/wiki/Graphical_user_interface
[34]https://en.wikipedia.org/wiki/User-centered_design

The command-line interface is the predecessor to a GUI, and there is On the other hand, a command line interface[35] is a text-based screen where users interact with their computer's programs, files, and operating system using a combination of commands and parameters.

Don't worry–we're not going to advise you start only interacting with your computer via the command line. There are plenty of tasks that are better suited for a GUI (*imagine how fun it would be if you had to play angry birds on a command line*).

But as someone who'll be using a computer to document and communicate their research, you do need to understand the technologies that are used to store, manipulate, and analyze data.

Hadley Wickham made this point in an excellent talk aptly titled," You can't do data science in a GUI[36] "

> "*The gooey is the easiest type of approach where you point and click, and everything is laid out in front of you. All of the options are laid out in front of you, which is great because you can see everything you can do. But it's also terrible because you have constraints–you can only do what the inventors of (SAS or Excel) wanted. Whereas with R–or other programming languages–is the opposite. All you get is this blinking cursor, and it's just telling you can do literally anything, but it's not gonna give you much…*"

> "*So I think an important thing about programming languages–like R or Python–is they give you a language to express your ideas, they give you very few constraints, which makes life tough for your learning or doing data science things occasionally, but the payoff for investing in a programming language is you get this whole this new language, and what you can express with them.*"

WARNING–command-lines are frustrating. Most of the technologies we interact with daily don't behave in ways that are easy to understand (that's why GUIs exist). Switching from a GUI to a CLI seems like a step backward at first, but the initial headaches pay off because of the gains you'll have in control, flexibility, automation, and reproducibility.

The Terminal

The Terminal is a command line interface application for Mac users. Terminal is available as an application (on Mac go to *Applications* > *Utilities* > *Terminal*) or as a pane in RStudio.

Here is a quick list of commonly used Terminal commands.

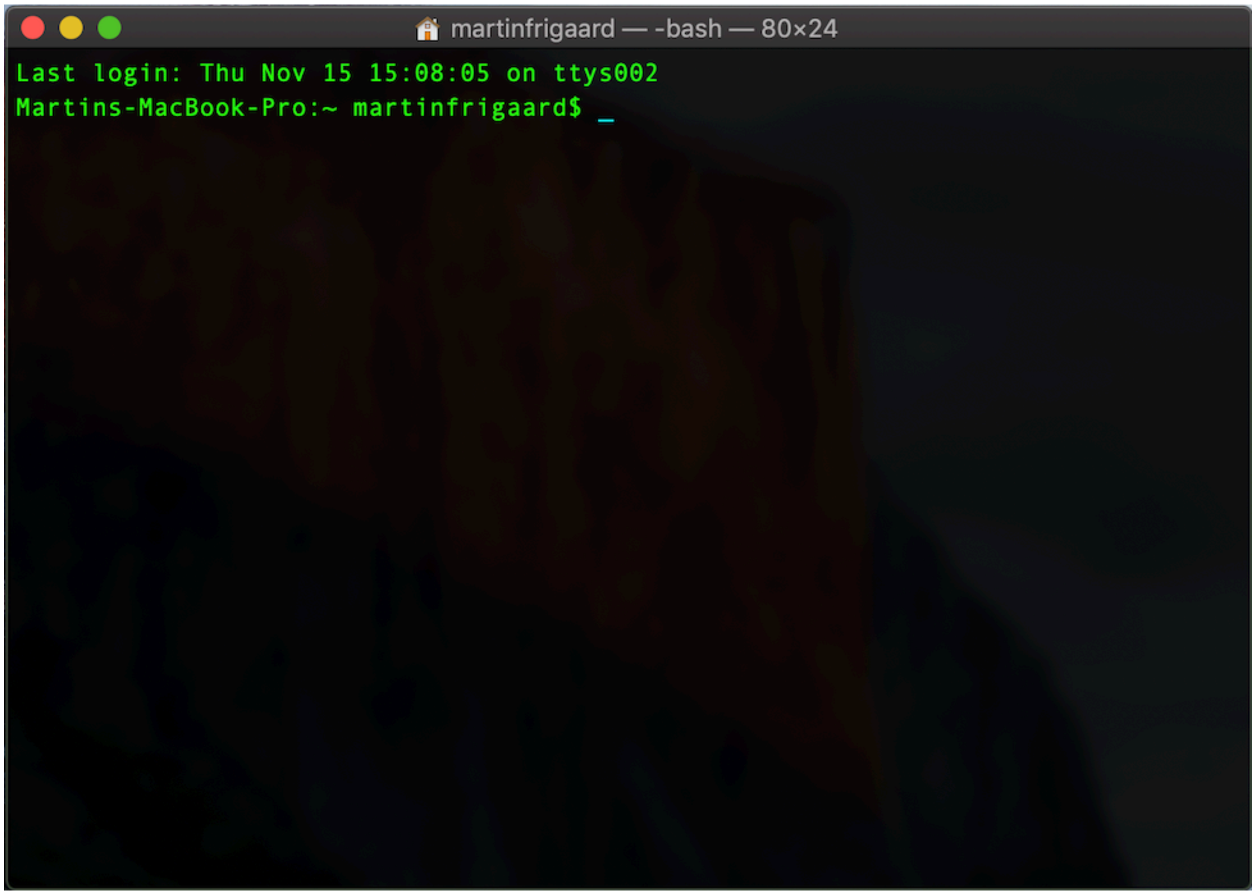
- `pwd` - Print working directory
- `cd` - change directories

[35]https://en.wikipedia.org/wiki/Command-line_interface
[36]https://www.youtube.com/watch?v=cpbtcsGE0OA

- `cp` - copy files from one directory to another
- `ls` - list all files
- `ls -la` - list all files, including hidden ones
- `mkdir` - make directory
- `cat` - display a text file in Terminal screen
- `echo` - outputs text as arguments, prints to Terminal screen, file, or in a pipeline
- `touch` - create a few files
- `grep` - “globally search a regular expression and print”
- `>>` and `>` - redirect output of program to a file (don’t display on Terminal screen)
- `sudo` and `sudo -s` (BE CAREFUL!!) perform commands as `root` user

The Terminal application

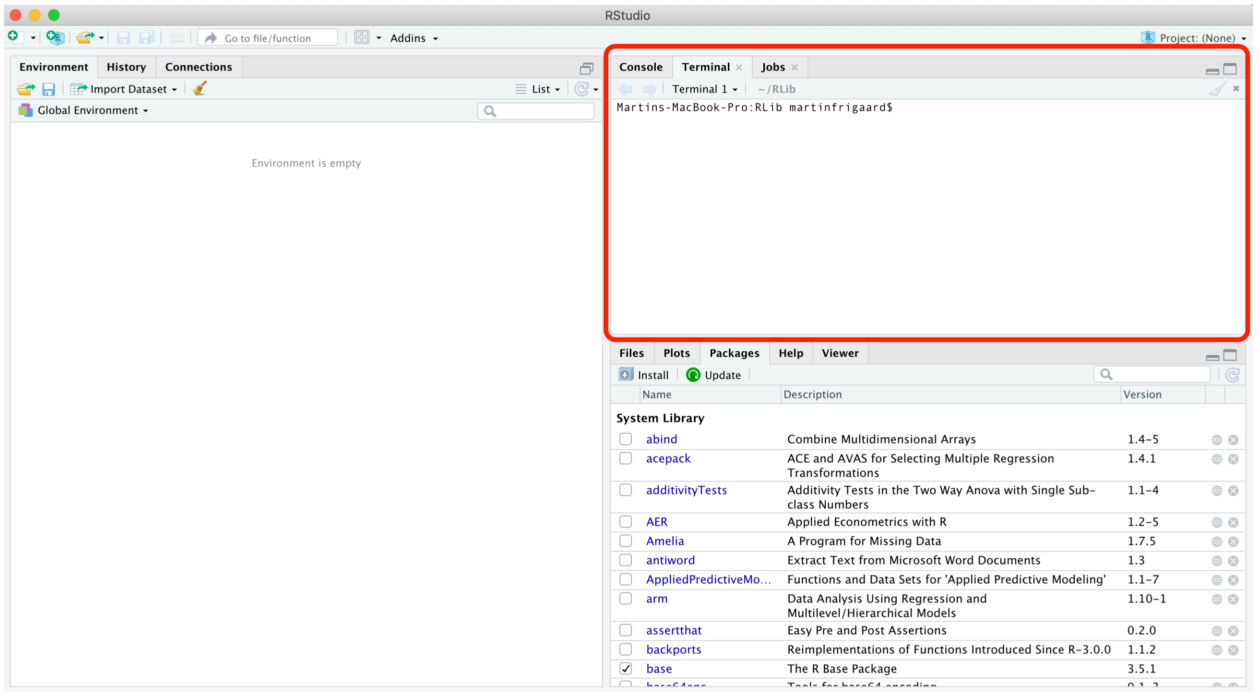
Below is an image of what the terminal application looks like on macOS with Homebrew syntax highlighting.



```
Last login: Thu Nov 15 15:08:05 on ttys002
Martins-MacBook-Pro:~ martinfrigaard$ _
```

The Terminal pane

The Terminal pane is also available in RStudio under *Tools* > *Terminal* > *New Terminal.*



```
Martins-MacBook-Pro:RLib martinfrigaard$
```

System Library

	Name	Description	Version
☐	abind	Combine Multidimensional Arrays	1.4–5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
☐	additivityTests	Additivity Tests in the Two Way Anova with Single Sub–class Numbers	1.1–4
☐	AER	Applied Econometrics with R	1.2–5
☐	Amelia	A Program for Missing Data	1.7.5
☐	antiword	Extract Text from Microsoft Word Documents	1.3
☐	AppliedPredictiveMo...	Functions and Data Sets for 'Applied Predictive Modeling'	1.1–7
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.10–1
☐	assertthat	Easy Pre and Post Assertions	0.2.0
☐	backports	Reimplementations of Functions Introduced Since R–3.0.0	1.1.2
✓	base	The R Base Package	3.5.1

Operating systems

In 2007, Apple released its Leopard[37] operating system that was the first to adhere to the Single Unix Specification[38]. I only introduce this bit of history to help keep the terminology straight. macOS and Linux are both Unix systems, so they have a similar underlying architecture (and philosophy). You can use most Linux commands on a Mac.

Windows has a command line tool called Powershell, but this is not the same as the Unix shells discussed above. The differences between these tools reflect the differences in design between the two operating systems. However, if you're a Windows 10 user, you can install a bash shell command-line tool[39].

Terminal applications

Strictly speaking, the Terminal application is not a shell, but rather it *gives the user access to the shell*. Other terminal emulator options exist, depending on your operating system and age of your machine. Terminal.app is the default application installed on macOS, but you can download other options (see iTerm2[40]). For example, the GNOME[41] is a desktop environment based on Linux which also has a Terminal emulator, but this gives users access to the Unix shell.

[37] https://en.wikipedia.org/wiki/MacOS_version_history#Version_10.5:_%22Leopard%22
[38] https://en.wikipedia.org/wiki/Single_UNIX_Specification
[39] https://www.windowscentral.com/how-install-bash-shell-command-line-windows-10
[40] https://www.iterm2.com/
[41] https://en.wikipedia.org/wiki/GNOME

Shells

On Macs, the Terminal application runs a bash shell[42]. This is the most commonly used shell, but there are other options too (see Zsh[43], tcsh[44], and sh[45]). *in fact, bash is a pun for Bourne-again shell.*

In the next section, We are going to introduce version control with Git and linking your local machine to Github.

[42]https://en.wikipedia.org/wiki/Bash_(Unix_shell)
[43]http://zsh.sourceforge.net/
[44]https://en.wikipedia.org/wiki/Tcsh
[45]https://en.wikipedia.org/wiki/Bourne_shell

Keeping track of changes to your work

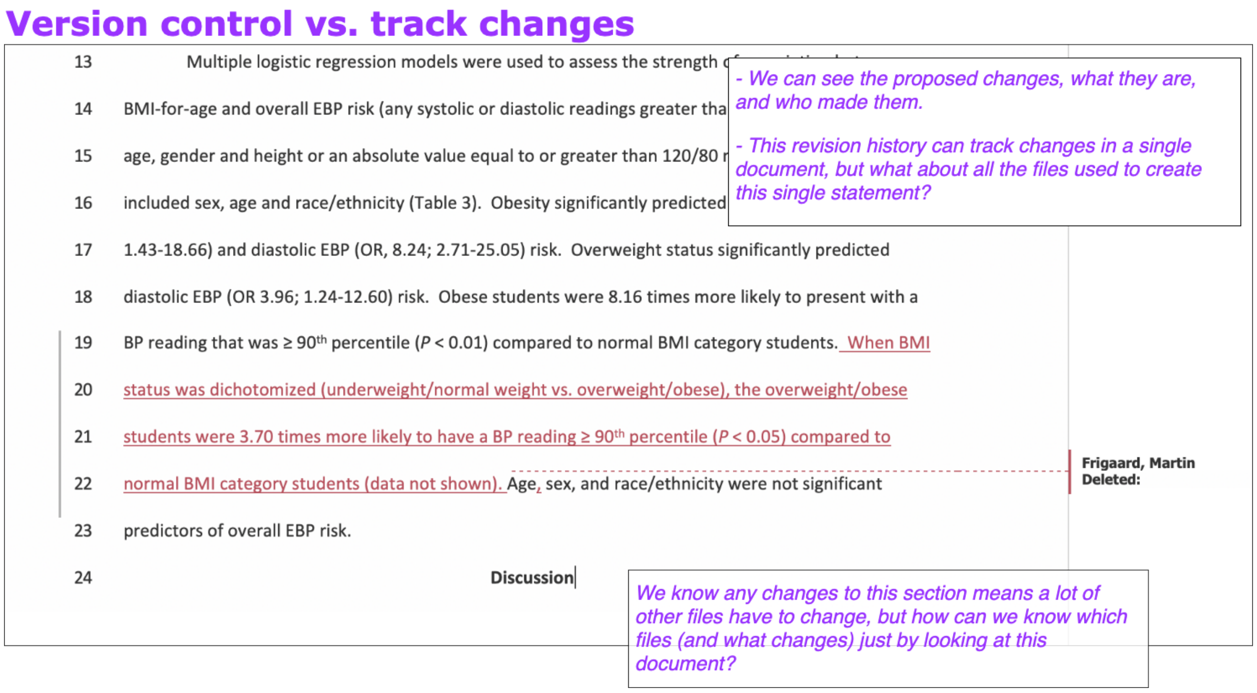
In the previous section, we covered R and RStudio, the command line, Shells, and the Terminal. If you're unfamiliar with these topics, please start there. This section will cover tracking changes, plain text files, version control with Git in RStudio.

Tracing your steps

'Showing your' work also means showing how your work has changed over time. In the example project, the `doc` folder contains a file titled, `2012-10-62-ican-manuscript-revision-v02.docx`.

The .docx is an earlier version of the manuscript, and there are some suggested changes to the results section below.

The good thing about using tracked changes in .docx files is that we can see 1) what the differences are, 2) who suggested them, 3) the time/date of the proposed change, and 4) any comments about the change.



Version control vs. track changes

Multiple logistic regression models were used to assess the strength o
BMI-for-age and overall EBP risk (any systolic or diastolic readings greater tha
age, gender and height or an absolute value equal to or greater than 120/80 r
included sex, age and race/ethnicity (Table 3). Obesity significantly predicted
1.43-18.66) and diastolic EBP (OR, 8.24; 2.71-25.05) risk. Overweight status significantly predicted
diastolic EBP (OR 3.96; 1.24-12.60) risk. Obese students were 8.16 times more likely to present with a
BP reading that was ≥ 90th percentile (P < 0.01) compared to normal BMI category students. When BMI
status was dichotomized (underweight/normal weight vs. overweight/obese), the overweight/obese
students were 3.70 times more likely to have a BP reading ≥ 90th percentile (P < 0.05) compared to
normal BMI category students (data not shown). Age, sex, and race/ethnicity were not significant
predictors of overall EBP risk.

Discussion

Frigaard, Martin
Deleted:

- We can see the proposed changes, what they are, and who made them.

- This revision history can track changes in a single document, but what about all the files used to create this single statement?

We know any changes to this section means a lot of other files have to change, but how can we know which files (and what changes) just by looking at this document?

Unfortunately, this only applies to a single document. When you're writing by committee (which is quite often in science), you know asking someone to change a single sentence can result in changes to dozens of files. Fortunately, this change is suggesting a deletion, so this is unlikely to result in generating additional analyses, tables, write-ups, etc.

We recommend you keep track of your changes, notes, and any pertinent documentation about your project in plain text `README` files. The reasons for this will become more apparent as we move through the example, but I wanted to outline a few here:

- plain text lasts forever (files written 40 years ago are still readable today)
- plain text can be *converted* to any other kind of document
- plain text is text searchable (ctrl+F or cmd+F allows us to find keywords or phrases)

These all sound great, but you might still be wondering what makes a file 'plain text,' so we'll define this below.

Plain text vs. most all file types

In the classic text The Pragmatic Programmer[46], authors Hunt and Thomas advise *'Keep[ing] Knowledge in Plain Text'*. This sentiment has been repeated here[47], here[48], and here[49].

This chapter will cover what people mean by 'plain text,' and why you might want to consider switching over to a plain text editor if you're currently using something like, Google Docs, Apple Papers, or Microsoft Word.

What *isn't* plain text

Non-plain text files are usually binary files (the `0/1` kind of binary). Binary files (i.e., files with binary-level compatibility) need special software to run on your computer. The language below is a handy way to think about these files:

> "Binary files are *computer-readable but not human-readable*[50]"

What *is* plain text

So if binary files aren't written in plain text, what's a plain text file? The language from the Wikipedia[51] description is helpful here:

[46]https://www.amazon.com/Pragmatic-Programmer-Journeyman-Master/dp/020161622X
[47]https://simplystatistics.org/2017/06/13/the-future-of-education-is-plain-text/
[48]https://richardlent.github.io/post/the-plain-text-workflow/
[49]http://plain-text.co/index.html#introduction
[50]https://www.webopedia.com/TERM/B/binary_file.html
[51]https://en.wikipedia.org/wiki/Text_file

> *When opened in a text editor, plain text files display computer and human-readable content.*

And here is the most crucial distinction–**human-readable vs. computer-readable**. I'll be sure to point out which files are binary and which are plain text as we go through the example, but generally speaking, a plain text file can be opened using a text editor. Examples of text editors include Atom[52], Sublime Text[53], and Notepad++[54]

Why would I use one of these applications and not just keep using Word?

We get–change is difficult, and if you have a working ecosystem of software that keeps you productive, don't abandon it. However, you should be aware of these technologies and recognize that people using them will be adapting *their* workflows to collaborate with you.

As Prof. Kieran Healy acknowledges the benefits (and downsides) of having collaborators working in binary file formats in his text[55], "The Plain Person's Guide to Plain Text Social Science,"

> "…it is generally easier for you to use their software than vice versa, if only because you are likely have a copy of Word on your computer already. In these circumstances you might also collaborate using Google Docs or some other service that allows for simultaneously editing the master copy of a document. This may not be ideal, but it is better than not collaborating. There is little to be gained from plain-text dogmatism in a .docx world."

Git

Git is a version control system[56] (VCS), which is somewhat like the **Tracked Changes** in Microsoft Word or the **Version History** in Google Sheets, but extended to every file in a project. Git will help you keep track of your documents, datasets, code, images, and anything else you tell it to keep an eye on.

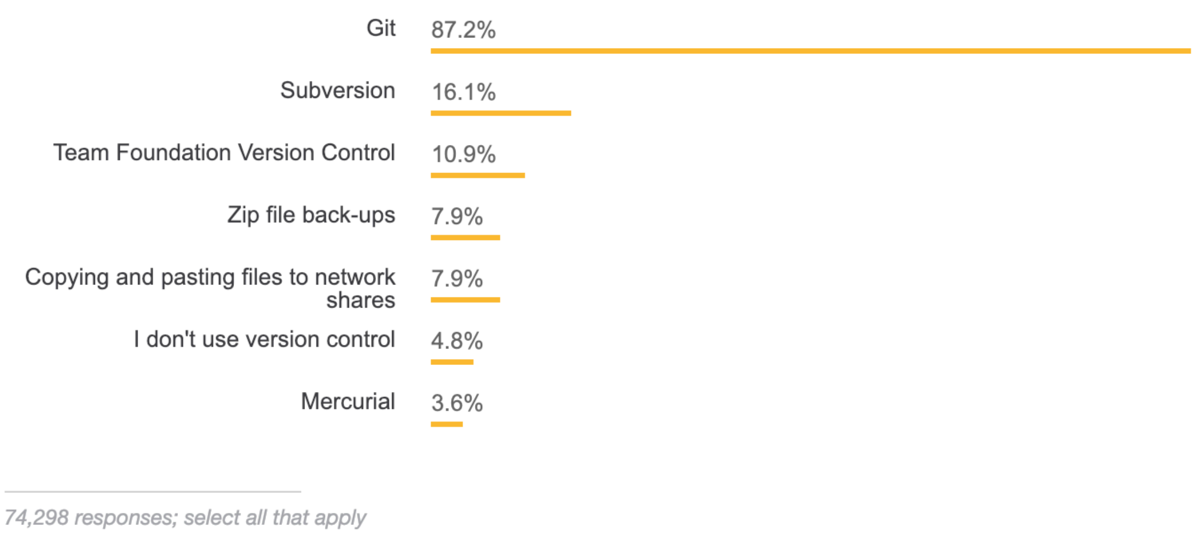
Plain text and Git

Git prefers plain text files because, until recently, software engineers and app developers were using programs like Git to track their source code (which they write in plain text)–another reason we recommend keeping your documentation and notes in a plain text file.

[52]https://atom.io/
[53]https://www.sublimetext.com/
[54]https://notepad-plus-plus.org/
[55]http://plain-text.co/
[56]https://en.wikipedia.org/wiki/Version_control

Why else should you use Git?

Git has become the most common version control system used by programmers[57].



Git is the dominant choice for version control for developers today, with almost 90% of developers checking in their code via Git.

source: StackOverflow Developer Survey Results[58]

Git is also a helpful way of thinking about the changes to your project. The terminology of Git is strange at first, but if you use Git long enough, you'll be thinking about your code in terms of ‘commits’, ‘pushes’, ‘forks’, and ‘repos’.

Installing Git

1. Download and install Git.[59]
2. Create a Github[60] account.

Configuring Git with `git config`

Git needs a little configuration before we can start using it and linking it to Github. There are three levels of configuration within Git, `system`, `user`, and `project`.

[57]https://insights.stackoverflow.com/survey/2018#work-version-control
[58]https://insights.stackoverflow.com/survey/2018#overview
[59]http://git-scm.org/
[60]https://github.com/

1) For **system** level configuration use:
`git config --system`

2) For **user** level configuration, use:
`git config --global`

3) For **Project** level configuration use:
`git config`

I'll set my Git `user.name` and `user.email` with `git config --global` so these are configured for all projects on this computer.

```
$ git config --global user.name "Martin Frigaard"
$ git config --global user.email "mjfrigaard@gmail.com"
```

I can check what I've configured with `git config --list`.

```
$ git config --list
```

At the bottom of the output, I can see the changes.

```
user.name=Martin Frigaard
user.email=mjfrigaard@gmail.com
```

These are also stored in a `.gitconfig` file I can view using:

```
$ cat .gitconfig
[user]
    name = Martin Frigaard
    email = mjfrigaard@gmail.com
```

Git terminology

Below are some commonly used terms/commands associated with Git and Github.

commits - commits are the staple in Git/Github the workflow. Commits are what Git uses to track the changes you've made to files or folders, so they can be considered nouns ("I'm creating a commit with these changes") or verbs (I am going to commit these changes to my project").

To quote David Demaree,

> "Semantically, each commit represents a complete snapshot of the state of your project at a given moment in time; its unique identifier serves to distinguish that state from the way the files in your project looked at any other moment in time."*

repository - this refers to the files and folders in your project and all the changes you make while working on them. On your local computer, a repository can exist in a folder you initialize a repository in (see below). On Github, a repository has the following structure: `https://github.com/<username>/<repository_-name>`.

init - the command `git init` is used to initialize a new git repository (it tells Git to start tracking changes in this directory).

status - whenever you wonder what you've done, what is happening, or if you're just generally confused, you can check the status of a git repository with `git status` (use this liberally).

clone - this command copies all files and changes into a new working directory from a remote, initializes (`init`) a new Git repository, and it adds a remote called `origin`.

diff - this is how Git shows differences between files. Read more about how changes are formatted/displayed here.[61]

Our example project

The example GitHub repository[62] has a the following files:

- a `README.md` file,
- two `.R` script files (`01-import.R` and `02-wrangle.R`),
- a `.csv` file (`IcanBP.csv`), and
- a revised manuscript document (`2012-10-62-ican-manuscript-revision-v02.docx`)

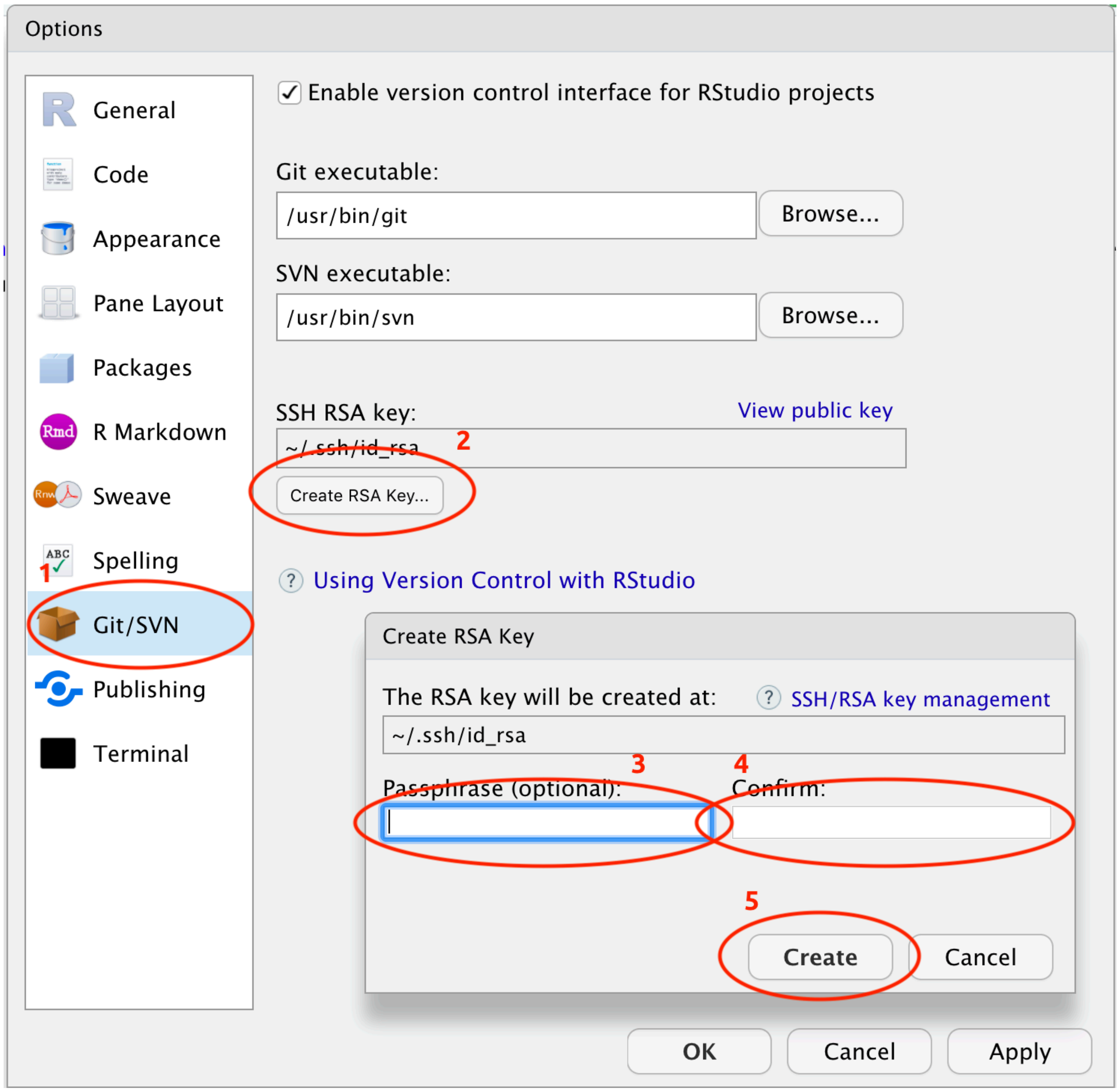
We will make some changes to the files in our local folder (i.e. repository), and then put everything back on Github. But first we need Git and RStudio to be able to talk to each other. The next section will cover this.

Synchronize RStudio and Git/Github

The online resource Happy Git and GitHub for the useR[63] has all in the information you will need for connecting RStudio and Git/Github. I repeat some of that information below (but add copious screenshots).

Go to *Tools* > *Global Options* > …

[61]https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/diffs
[62]https://github.com/mjfrigaard/syw-example
[63]http://happygitwithr.com/



[✓] Enable version control interface for RStudio projects

Git executable: /usr/bin/git

SVN executable: /usr/bin/svn

SSH RSA key: ~/.ssh/id_rsa

The RSA key will be created at: ~/.ssh/id_rsa

Passphrase (optional):

Confirm:

1. Click on *Git/SVN*
2. Then *Create RSA Key...*
3. In the dialog box, enter a passphrase (and store it in a safe place), then click *Create*.

The result should look something like this:

```
whoeveryouare ~ $ ssh-keygen -t rsa -b 2891 -C "USEFUL-COMMENT"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/username/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/username/.ssh/id_rsa.
Your public key has been saved in /Users/username/.ssh/id_rsa.pub.
The key fingerprint is:
SHA483:g!bB3r!sHg!bB3r!sHg!bB3r!sH USEFUL-COMMENT
The key's randomart image is:
+---[RSA 2891]----+
|      o+   . .   |
|     .=.o . +    |
|     ..= + +     |
|      .+* E      |
|     .= So =     |
|    .  +. = +    |
|   o.. = ..* .   |
|  o ++=.o =o.    |
| ..o.++o.=+.     |
+----[SHA483]-----+
```

Now I need to go back to Terminal and store this `SSH` from RStudio.

Adding a key SSH in Terminal

In the Terminal, I enter the following commands.

```
$ eval "$(ssh-agent -s)"
```

The response tells me I'm an `Agent`.

```
Agent pid 007
```

Now I want to add the *SSH RSA* to the keychain. There are three elements in this command: the `ssh-add`, the `-K`, and `~/.ssh/id_rsa`.

- The `ssh-add` is the command to add the *SSH RSA*
- The `-K` stores the passphrase I generated, and
- `~/.ssh/id_rsa` is the location of the *SSH RSA*.

```
$ ssh-add -K ~/.ssh/id_rsa
```

Enter the passphrase and then have it tell me the identity has been added.

```
Enter passphrase for /Users/username/.ssh/id_rsa:
Identity added: /Users/username/.ssh/id_rsa (username@Martins-MacBook-Pro.local)
```

Create the `~/.ssh/config` file

On macOS-Sierra 10.12.2 or higher requires a `config` file. I can do this using the Terminal commands above.

First, I move into the `~/.ssh/` directory.

```
$ cd ~/.ssh/
```

Then I create this `config` file with `touch`

```
$ touch config
# verify
$ ls
config          id_rsa          id_rsa.pub
```

I use `echo` to add the following text to the `config` file.

```
Host *
 AddKeysToAgent yes
 UseKeychain yes
```

Recall the `>>` will send the text to the `config` file.

```
# add text
$ echo "Host *
>  AddKeysToAgent yes
>  UseKeychain yes" >> config
```

Finally, I can check `config` with `cat`

```
# verify
$ cat config
Host *
 AddKeysToAgent yes
 UseKeychain yes
```

Great! Now I am all set up to use Git with RStudio. In the next section, we will move the contents of a local folder to Github.

Conclusion

This book is **done** and ready for *the world to see*, hooray!