Christopher Gandrud

Reproducible Research with R and RStudio Second Edition

2

Getting Started with Reproducible Research

Researchers often start thinking about making their work reproducible near the end of the research process when they write up their results or maybe even later when a journal requires their data and code be made available for publication. Or maybe even later when another researcher asks if they can use the data from a published article to reproduce the findings. By then there may be numerous versions of the data set and records of the analyses stored across multiple folders on the researcher's computers. It can be difficult and time consuming to sift through these files to create an accurate account of how the results were reached. Waiting until near the end of the research process to start thinking about reproducibility can lead to incomplete documentation that does not give an accurate account of how findings were made. Focusing on reproducibility from the beginning of the process and continuing to follow a few simple guidelines throughout your research can help you avoid these problems. Remember "reproducibility is not an afterthought–it is something that must be built-into the project from the beginning" (Donoho, 2010, 386).

This chapter first gives you a brief overview of the reproducible research process: a workflow for reproducible research. Then it covers some of the key guidelines that can help make your research more reproducible.

2.1 The Big Picture: A Workflow for Reproducible Research

The three basic stages of a typical computational empirical research project are:

- data gathering,
- data analysis,
- results presentation.

Each stage is part of the reproducible research workflow covered in this book. Tools for reproducibly gathering data are covered in Part II. Part III teaches tools for tying the data we gathered to our statistical analyses and presenting the results with tables and figures. Part IV discusses how to tie these findings into a variety of documents you can use to advertise your findings.

Instead of starting to use the individual tools of reproducible research as soon as you learn them, I recommend briefly stepping back and considering how the stages of reproducible research *tie* together overall. This will make your workflow more coherent from the beginning and save you a lot of backtracking later on. Figure 2.1 illustrates the workflow. Notice that most of the arrows connecting the workflow's parts point in both directions, indicating that you should always be thinking about how to make it easier to go backwards through your research, i.e. reproduce it, as well as forwards.

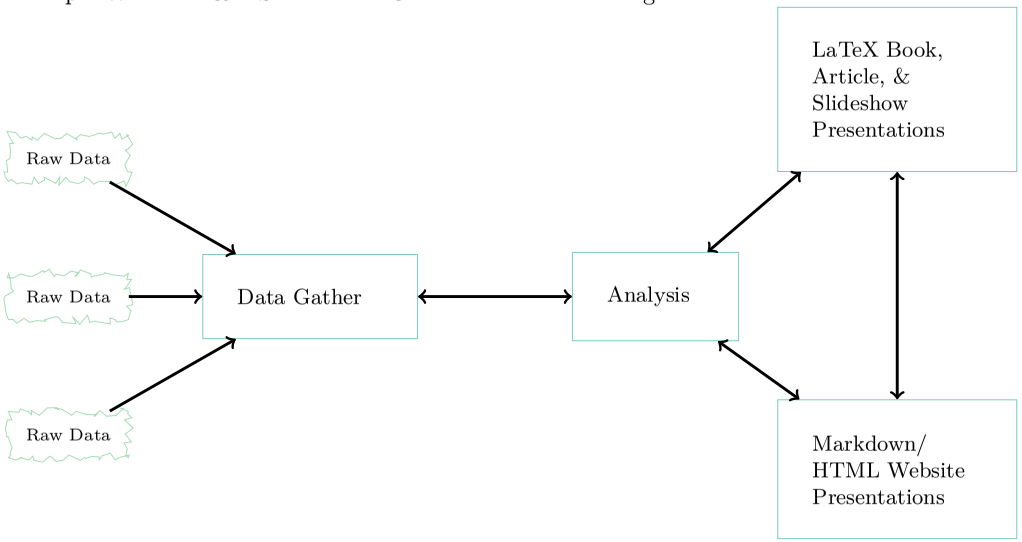
Around the edges of the figure are some of the commands you will learn to make it easier to go forwards and backwards through the process. These commands tie your research together. For example, you can use API-based R packages to gather data from the internet. You can use R's `merge` command to combine data gathered from different sources into one data set. The `getURL` function from R's *RCurl* package (Temple Lang, 2015) and the `read.table` function can be used to bring this data set into your statistical analyses. The *knitr* or *rmarkdown* package then ties your analyses into your presentation documents. This includes the code you used, the figures you created, and, with the help of tools such as the `kable` function in the *knitr* package, tables of results. You can even tie multiple presentation documents together. For example, you can access the same figure for use in a LaTeX article and a Markdown-created website with the `includegraphics` and `![]()` commands, respectively. This helps you maintain a consistent presentation of results across multiple document types. We'll cover these commands in detail throughout the book. See Table 2.1 for a brief but more complete overview of the main *tie commands*.

2.1.1 Reproducible theory

An important part of the research process that I do not discuss in this book is theoretical stage. Ideally, if you are using a deductive research design, the bulk of this work will precede and guide the data gathering and analysis stages. Just because I don't cover this stage of the research process doesn't mean that theory building can't and shouldn't be reproducible. It can in fact be "the easiest part to make reproducible" (Vandewalle et al., 2007, 1254). Quotes and paraphrases from previous works in the literature obviously need to be fully cited so that others can verify that they accurately reflect the source material. For mathematically based theory, clear and complete descriptions of the proofs should be given.

Though I don't actively cover theory replication in depth in this book, I do touch on some of the ways to incorporate proofs and citations into your presentation documents. These tools are covered in Part IV.

FIGURE 2.1
Example Workflow & a Selection of Commands to Tie It Together



`Make`
`download.file`
`source_data`
`source_DropboxData`
`read.table`
`merge`
`getURL`
API-based
packages

`source_data`
`source_DropboxData`
`read.table`
`getURL`

knitr
rmarkdown
`source`
`source_url`
`kable`
`print(xtable())`
`texreg`

knitr
`input`
`include`
`includegraphics`
Pandoc
`![]()`

2.2 Practical Tips for Reproducible Research

Before we start learning the details of the reproducible research workflow with R and RStudio, it's useful to cover a few broad tips that will help you organize your research process and put these skills in perspective. The tips are:

1. Document everything!
2. Everything is a (text) file.
3. All files should be human readable.
4. Explicitly tie your files together.
5. Have a plan to organize, store, and make your files available.

Using these tips will help make your computational research really reproducible.

2.2.1 Document everything!

In order to reproduce your research, others must be able to know what you did. You have to tell them what you did by documenting as much of your research process as possible. Ideally, you should tell your readers how you gathered your data, analyzed it, and presented the results. Documenting everything is the key to reproducible research and lies behind all of the other tips in this chapter and tools you will learn throughout the book.

Document your R session info

Before discussing the other tips it's important to learn a key part of documenting with R. You should *record your session info*. Many things in R have stayed the same since it was introduced in the early 1990s. This makes it easy for future researchers to recreate what was done in the past. However, things can change from one version of R to another and especially from one version of an R package to another. Also, the way R functions and how R packages are handled can vary across different operating systems, so it's important to note what system you used. Finally, you may have R set to load packages by default (see Section 3.1.8 for information about packages). These packages might be necessary to run your code, but other people might not know what packages and what versions of the packages were loaded from just looking at your source code. The `sessionInfo` command in R prints a record of all of these things. The information from the session I used to create this book is:

```
# Print R session info
sessionInfo()
```

```
## R version 3.2.0 (2015-04-16)
## Platform: x86_64-apple-darwin14.3.0 (64-bit)
## Running under: OS X 10.10.3 (Yosemite)
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] ZeligBayesian_0.1   MCMCpack_1.3-3      coda_0.17-1
##  [4] Zelig_4.2-1         sandwich_2.3-3      MASS_7.3-40
##  [7] boot_1.3-16         xtable_1.7-4        WDI_2.4
## [10] tidyr_0.2.0         texreg_1.35         survival_2.38-1
## [13] stargazer_5.1       shiny_0.12.0        rvest_0.2.0
## [16] RJSONIO_1.3-0       rmarkdown_0.6.1     repmis_0.4.2
## [19] RCurl_1.95-4.6      bitops_1.0-6        quantmod_0.4-4
## [22] TTR_0.22-0          xts_0.9-7           zoo_1.7-12
## [25] packrat_0.4.3       openair_1.5         maps_2.3-9
## [28] lazyeval_0.1.10     markdown_0.7.7      magrittr_1.5
## [31] knitcitations_1.0.5 httr_0.6.1          htmlwidgets_0.4
## [34] highlight_0.4.7     googleVis_0.5.8     ggplot2_1.0.1
## [37] formatR_1.2         extrafont_0.17      dplyr_0.4.1
## [40] digest_0.6.8        devtools_1.8.0      data.table_1.9.4
## [43] countrycode_0.18    brew_1.0-6          animation_2.3
## [46] knitr_1.10.5
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-120        lubridate_1.3.3     bit64_0.9-4
##  [4] RColorBrewer_1.1-2  R.cache_0.10.0      tools_3.2.0
##  [7] R6_2.0.1            DBI_0.3.1           mgcv_1.8-6
## [10] colorspace_1.2-6    bit_1.1-12          git2r_0.10.1
## [13] chron_2.3-45        extrafontdb_1.0     scales_0.2.4
## [16] hexbin_1.27.0       stringr_1.0.0       R.utils_2.0.2
## [19] htmltools_0.2.6     bibtex_0.4.0        highr_0.5
## [22] R.oo_1.19.0         Matrix_1.2-0        Rcpp_0.11.6
## [25] munsell_0.4.2       proto_0.3-10        RefManageR_0.8.45
## [28] R.methodsS3_1.7.0   stringi_0.4-1       plyr_1.8.2
## [31] grid_3.2.0          parallel_3.2.0      lattice_0.20-31
## [34] splines_3.2.0       mapproj_1.2-2       rjson_0.2.15
## [37] reshape2_1.4.1      XML_3.98-1.1        evaluate_0.7
## [40] latticeExtra_0.6-26 mapdata_2.2-3       png_0.1-7
## [43] httpuv_1.3.2        RgoogleMaps_1.2.0.7 Rttf2pt1_1.3.3
## [46] twitteR_1.1.8       gtable_0.1.2        assertthat_0.1
## [49] mime_0.3            memoise_0.2.1       rversions_1.0.0
## [52] cluster_2.0.1
```

Chapter 4 gives specific details about how to create files with dynamically included session information. If you use non-R tools you should also record what versions of these tools you used.

2.2.2 Everything is a (text) file

Your documentation is stored in files that include data, analysis code, the write-up of results, and explanations of these files (e.g. data set codebooks, session info files, and so on). Ideally, you should use the simplest file format possible to store this information. Usually the simplest file format is the humble, but versatile, text file.[1]

Text files are extremely nimble. They can hold your data in, for example, comma-separated values (`.csv`) format. They can contain your analysis code in `.R` files. And they can be the basis for your presentations as markup documents like `.tex` or `.md`, for LaTeX and Markdown files, respectively. All of these files can be opened by any program that can read text files.

One reason reproducible research is best stored in text files is that this helps *future-proof* your research. Other file formats, like those used by Microsoft Word (`.docx`) or Excel (`.xlsx`), change regularly and may not be compatible with future versions of these programs. Text files, on the other hand, can be opened by a very wide range of currently existing programs and, more likely than not, future ones as well. Even if future researchers do not have R or a LaTeX distribution, they will still be able to open your text files and, aided by frequent comments (see below), be able to understand how you conducted your research (Bowers, 2011, 3).

Text files are also very easy to search and manipulate with a wide range of programs–such as R and RStudio–that can find and replace text characters as well as merge and separate files. Finally, text files are easy to version and changes can be tracked using programs such as Git (see Chapter 5).

2.2.3 All files should be human readable

Treat all of your research files as if someone who has not worked on the project will, in the future, try to understand them. Computer code is a way of communicating with the computer. It is 'machine readable' in that the computer is able to use it to understand what you want to do.[2] However, there is a very good chance that other people (or you six months in the future) will not understand what you were telling the computer. So, you need to make all of your files 'human readable'. To make them human readable, you should comment on your code with the goal of communicating its design and purpose (Wilson et al., 2012). With this in mind it is a good idea to *comment frequently* (Bowers, 2011, 3) and *format your code using a style guide* (Nagler, 1995). For especially important pieces of code you should use *literate programming*–where the source code and the presentation text describing its design and purpose

[1] Plain text files are usually given the file extension `.txt`. Depending on the size of your data set it may not be feasible to store it as a text file. Nonetheless, text files can still be used for analysis code and presentation files.

[2] Of course, if the computer does not understand it will usually give an error message.

appear in the same document. Doing this will make it very clear to others how you accomplished a piece of research.

Commenting

In R, everything on a line after a hash character–`#`–(also known as number, pound, or sharp) is ignored by R, but is readable to people who open the file. The hash character is a comment declaration character. You can use the `#` to place comments telling other people what you are doing. Here are some examples:

```
# A complete comment line
2 + 2 # A comment after R code
```

```
## [1] 4
```

On the first line the `#` (hash) is placed at the very beginning, so the entire line is treated as a comment. On the second line the `#` is placed after the simple equation `2 + 2`. R runs the equation and finds the answer `4`, but it ignores all of the words after the hash.

Different languages have different comment declaration characters. In LaTeX everything after the `%` percent sign is treated as a comment, and in Markdown/HTML comments are placed inside of `<!- ->`. The hash character is used for comment declaration in command-line shell scripts.

Nagler (1995, 491) gives some advice on when and how to use comments:

- write a comment before a block of code describing what the code does,
- comment on any line of code that is ambiguous.

In this book I follow these guidelines when displaying code. Nagler also suggests that all of your source code files should begin with a comment header. *At the least* the header should include:

- a description of what the file does,
- the date it was last updated,
- the name of the file's creator and any contributors.

You may also want to include other information in the header such as what files it depends on, what output files it produces, what version of the programming language you are using, sources that may have influenced the code, and how the code is licensed. Here is an example of a minimal file header for an R source code file that creates the third figure in an article titled 'My Article':

```
##################
# R Source code file used to create Figure 3 in My 'Article'
# Created by Christopher Gandrud
# MIT License
##################
```

Feel free to use things like the long series of hash marks above and below the header, white space, and indentations to make your comments more readable.

Style guides

In natural language writing you don't necessarily have to follow a style guide. People could probably figure out what you are trying to say, but it is a lot easier for your readers if you use consistent rules. The same is true when writing computer code. It's good to follow consistent rules for formatting your code so that it's easier for you and others to understand.

There are a number of R style guides. Most of them are similar to the Google R Style Guide.[3] Hadley Wickham also has a nicely presented R style guide.[4] You may want to use the *formatR* (Xie, 2015a) package to automatically reformat your code so that it is easier to read.

Literate programming

For particularly important pieces of research code it may be useful to not only comment on the source file, but also display code in presentation text. For example, you may want to include key parts of the code you used for your main statistical models and an explanation of this code in an appendix following your article. This is commonly referred to as literate programming (Knuth, 1992).

2.2.4 Explicitly tie your files together

If everything is just a text file, then research projects can be thought of as individual text files that have a relationship with one another. They are tied together. A data file is used as input for an analysis file. The results of an analysis are shown and discussed in a markup file that is used to create a PDF document. Researchers often do not explicitly document the relationships between files that they used in their research. For example, the results of an analysis–a table or figure–may be copied and pasted into a presentation document. It can be very difficult for future researchers to trace the table or figure back to a particular statistical model and a particular data set without

[3] See: `http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`.
[4] You can find it at `http://adv-r.had.co.nz/Style.html`.

clear documentation. Therefore, it is important to make the links between your files explicit.

Tie commands are the most dynamic way to explicitly link your files together. These commands instruct the computer program you are using to use information from another file. In Table 2.1 I have compiled a selection of key tie commands you will learn how to use in this book. We'll discuss many more, but these are some of the most important.

2.2.5 Have a plan to organize, store, and make your files available

Finally, in order for independent researchers to reproduce your work, they need to be able access the files that instruct them how to do this. Files also need to be organized so that independent researchers can figure out how they fit together. So, from the beginning of your research process you should have a plan for organizing your files and a way to make them accessible.

One rule of thumb for organizing your research in files is to limit the amount of content any one file has. Files that contain many different operations can be very difficult to navigate, even if they have detailed comments. For example, it would be very difficult to find any particular operation in a file that contained the code used to gather the data, run all of the statistical models, and create the results figures and tables. If you have a hard time finding things in a file you created, think of the difficulties independent researchers will have!

Because we have so many ways to link files together, there is really no need to lump many different operations into one file. So, we can make our files modular. One source code file should be used to complete one or just a few tasks. Breaking your operations into discrete parts will also make it easier for you and others to find errors (Nagler, 1995, 490).

Chapter 4 discusses file organization in much more detail. Chapter 5 teaches you a number of ways to make your files accessible through the cloud computing services Dropbox and GitHub.

TABLE 2.1
A Selection of Commands/Packages/Programs for Tying Together Your Research Files

Command/Package/ Program	Language	Description	Chapters Discussed
knitr	R	R package with commands for tying analysis code into presentation documents including those written in LaTeX and Markdown.	Throughout
rmarkdown	R	R package that builds on *knitr*. It allows you to use Markdown to output to HTML, PDFs compiled with LaTeX or Microsoft Word.	Throughout
`download.file`	R	Downloads a file from the internet.	6
`read.table`	R	Reads a table into R. You can use this to import a plain-text file formatted data into R.	6
`read.csv`	R	Same as `read.table` with default arguments set to import `.csv` formatted data files.	6
`source_data`	R	Reads a table stored on the internet into R. You can use it to import a plain-text formatted data file into R from secure (https) URLs.	6
`source_DropboxData`	R	Imports a plain-text data file stored in a Dropbox non-Public folder into R.	6
API-based packages	R	Various packages use APIs to gather data from the internet.	6
`merge`	R	Merges together data frames.	7
`source`	R	Runs an R source code file.	8
`source_url`	R	From the *devtools* package. Runs an R source code file from a secure (`https`) url like those used by GitHub.	8
`kable`	R	Creates tables from data frames that can be rendered using Markdown or LaTeX.	9
`toLaTeX`	R	Converts R objects to LaTeX.	2
`input`	LaTeX	Includes LaTeX files inside of other LaTeX files.	12
`include`	LaTeX	Similar to `input`, but puts page breaks on either side of the included text. Usually it is used for including chapters.	12
`includegraphics`	LaTeX	Inserts a figure into a LaTeX document.	10
`![]()`	Markdown	Inserts a figure into a Markdown document.	13
Pandoc	shell	A shell program for converting files from one markup language to another. Allows you to tie presentation documents together.	12 & 13
Make	shell	A shell program for automatically building many files at the same time.	6