Beautiful R

Caio Lente + Curso-R

2022-08-09

Table of contents

Preface

If you came here of your own volition, you're probably an R programmer. Maybe you're still a novice, maybe you've already had your fair share of `cannot coerce type 'closure'` errors; either way, you're trying to grapple with this (at the time of writing) almost thirty years-old programming language.

It's tough, I know. Take a deep breath. I've got you covered.

Beautiful R is a book about what R does right: its goal is to both help you succeed at programming R, and showcase this quirky little language's full potential. Think of it as the antithesis to *The R Inferno*.

The book is divided into three sections in increasing order of complexity:

1. **Structure** explains how to create and use R projects effectively. Learn how to setup your environment, use IDEs responsibly, and never lose a file again.
2. **Patterns** is full of tips on programming R well. Transform your projects into packages, avoid silly mistakes by creating unit tests, and learn how to write good R code.
3. **Style** covers the last step in the ladder: making your code clean, easy to understand, and beautiful. Once you are able to consistently write good code, it's time to start writing *great* code.

Introduction

R is wabi-sabi.

Part I

Structure

Empty.

1 Environment

Programming necessarily presupposes a programming *environment*. Just as one can't write text without choosing a font, one can't write code outside of an environment.

> **Note**
>
> Keep in mind that in R, "environment" usually has a very specific meaning. In this chapter this word refers to the state of your computer while you're programming, including the organization of your files, the version of R that you're using, etc.

The environment of an R script is like air is to us: you might forget that it exists, but it is always there. Then why write a whole chapter about this? Why write about air?

```
# Printing usually works like this
print(1:9)
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```
# What happened here?
print(1:9)
```

```
[1] 1 2 3
[4] 4 5 6
[7] 7 8 9
```

Breathing foul air can make you dizzy, and working in a broken environment might cause your script to function incorrectly; this is why we need to learn how to properly manage it. There is no trick in the code above, by the way. I haven't redefined the `print()` function, I just changed the `width` option in R's environment.

1.1 File system

In Chapter 2, we'll discuss how to organize an R project, but first we need some guidelines on how to organize files in general. Many people underestimate the usefulness organizing and naming their files consistently, but the truth is that having a system saves you valuable time when searching through your past projects.

First of all, I suggest you create a folder in your computer for all your programming needs. It's not that rare to create *a lot* of files when coding, and making sure they will all be in a single place really helps. Since scripts are able to write and delete files, this lowers the odds that an R program ruins your valuable personal files by accident.

Now for the files. Danielle Navarro has an amazing presentation about project structure where she outlines three main principles to go by when naming files:

1. Be nice to machines.
2. Be nice to humans.
3. Make sorting and searching easy.

1.1.1 Be nice to machines

Machines usually get confused by spaces, special characters (like `^.*?+|$"`), and accents. Some operating systems also treat uppercase and lowercase as the same. Our files should, therefore, always separate words with `_` or `-` (consistently), only use lowercase letters and numbers, and never use accented characters.

```
# Good
draft01_jalapeno_essay.docx

# Bad
Draft "Jalapeño" Essay(1).docx
```

An important exception here are file extensions: R scripts should be terminated by `.R`, always uppercase. This is a longstanding tradition, so it takes precedence over the other rules.

1.1.2 Be nice to humans

Humans need context, so short and vague file names are to be avoided. A good practice is to begin the file name with it's "type" (e.g. analysis, note, report, etc.) and then append a meaningful description.

```
# Good
analysis01_descriptive-statistics.R
notes02_tentative-write-up.docx

# Bad
01.R
notes2.docx
```

Note how, in the examples above, `-` is used to separate words that belong to the same "chunk", and `_` is used to separate different "chunks". I don't follow this suggestion personally, but many people like the idea.

1.1.3 Make sorting and searching easy

If your files names need to include dates, always follow the `YYYY-MM-DD` format and place them before the description so that the files are sorted automatically.

```
# Good
2012-01-01.R
2012-01-02.R
2012-04-01.R

# Bad
1-April-2012.R
1-Jan-2012.R
2-Jan-2012.R
```

If you need to order files by something other than date, use numbers as prefix, but remember to left pad with `0` so that all numbers have the same length. Also include keywords if you want to divide files by type.

```
# Good
reading01_shakespeare_romeo-and-juliet.docx
reading02_shakespeare_romeo-and-juliet.docx
...
reading11_shakespeare_romeo-and-juliet.docx
notes01_shakespeare_romeo-and-juliet.docx
...
```

1.2 Profile

Most users don't know about a little file called `.Rprofile` (the dot at the beginning makes the file invisible). Your `.Rprofile` contains R code that is run every time you open R, so this is a good place to set some options and configurations. It's also an amazing place to make your analyses impossible to reproduce.

If you've never used this feature of R, I suggest you stay this way until you are a seasoned veteran in the language. A good tip for using you `.Rprofile` correctly is to never put anything that you would write in a script there.

Here is a good example of an `.Rprofile`:

```
# Set repo
options(repos = c(CRAN = "https://cran.rstudio.org"))

# Change width (like in the beginning of the chapter)
if (interactive()) {
  options(width = 10)
}
```

If you *really* want to add something to your `.Rprofile`, you can install the usethis package and run `usethis::edit_r_profile()`.

1.3 Versions

To me, this is the most important part of the chapter. You can fix all of your file names and remove everything from your `.Rprofile`; if you don't update system, it's all for naught.

R is a relatively old language, so it has been updated many times since launch. Every two to three months, the R Core Team releases a new version with many improvements and bug fixes. These updates can be very minor, but sometimes there are truly exciting new features. in R 4.1.0, for example, the native pipe (`|>`) was introduced, pretty much retiring magrittr.

This means that, if you don't upgrade your R installation, you'll be missing out on the evolution of the language and leaving your system vulnerable to known bugs.

The same is true for packages, the life and blood of R. Packages change much more and much faster than R itself, since they're not always worried about backwards compatibility. If you leave your packages to rot, expect your scripts to stop working on other peoples machines, a harder time googling bugs, and to get stuck with old code.

My advice is then the following:

1. Every few weeks make sure to run `update.packages()` to update your packages.
2. Every few months you should also update the language. If you're on Windows you can use the installr package, otherwise you can go to their website and grab the newest binary.

It's evidently a bit harder to keep your environment up to date than to just keep it unchanged forever. Sometimes you'll have to rewrite something that stopped working from one version to the next, but this is much easier than letting your system grow more and more out of date until, all of the sudden, everything stops working all at once. If you keep up with every release, the adjustments will be small and frequent; if you don't, the adjustments will be major and you might not be able to make them in a reasonable amount of time.

As a last suggestion, also make sure to update your operating system. R doesn't support every version of Windows and Mac forever, and I'm not even mentioning the security patches you're missing when you choose to "update later".

2 Projects

Empty.

3 IDEs

Empty.

4 Git

Empty.

Part II

Patterns

Empty.

5 Packages

Empty.

6 Dependencies

Empty.

7 Data

Empty.

8 Testing

Empty.

9 Renv

Empty.

10 CI/CD

Empty.

Part III

Style

Empty.

11 Beauty

Empty.

12 Pipe

Empty.

13 Functions

Empty.

14 Linting

Empty.

15 Shiny

Empty.

References