Building a package that lasts

Part 2: functions

A "package-ready" R function

In the R folder are what we can call "nice" functions, i.e. functions that :

- Do not use `library()` or `require()`: dependencies are managed in the NAMESPACE file.
- Do not change user `options()` or `par()`.
- Do not use `source()` to call code.
- Do not play with `setwd()`.
- Do not silently write in any other place than a temp file.

Arranging your R folder

You can organize your functions in as many .R files as you want.

There is no perfect number, but consider that **each .R contains one "big" function (and its methods if necessary) or one family of functions**.

You can your .R in family, with an accronyme at the beginning (family-*.R).

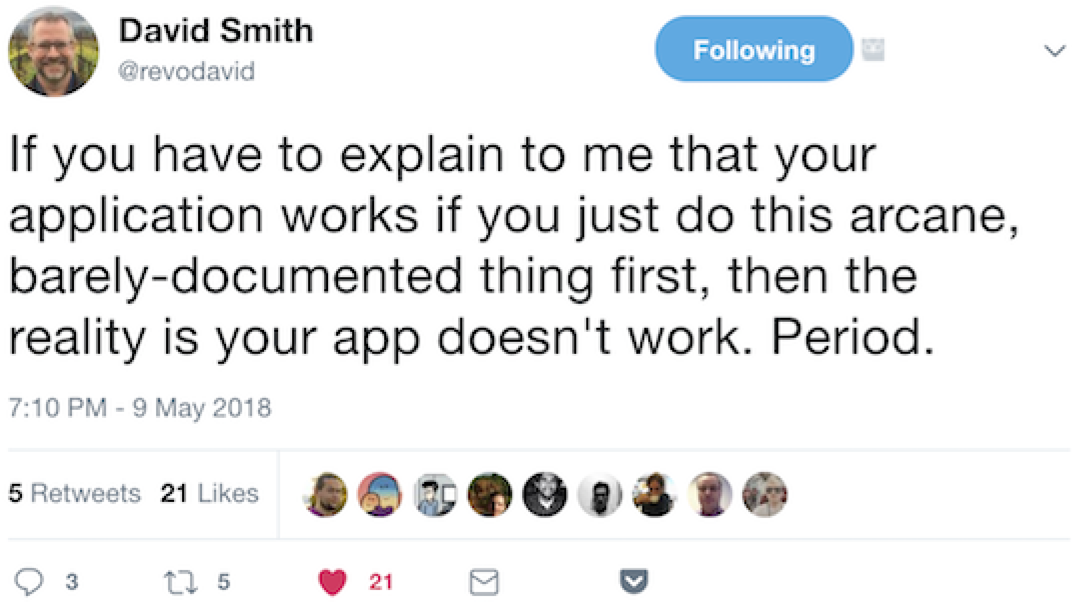
Group the "utility" functions in a utils.R file.

Beware of capital letters and non alphanumeric characters in file name, which can give surprising results if you change operating system.

Note that the R/ folder must not contain subfolders.

Package UX is about as important as the code behind.

Why document?


David Smith
@revodavid

Following

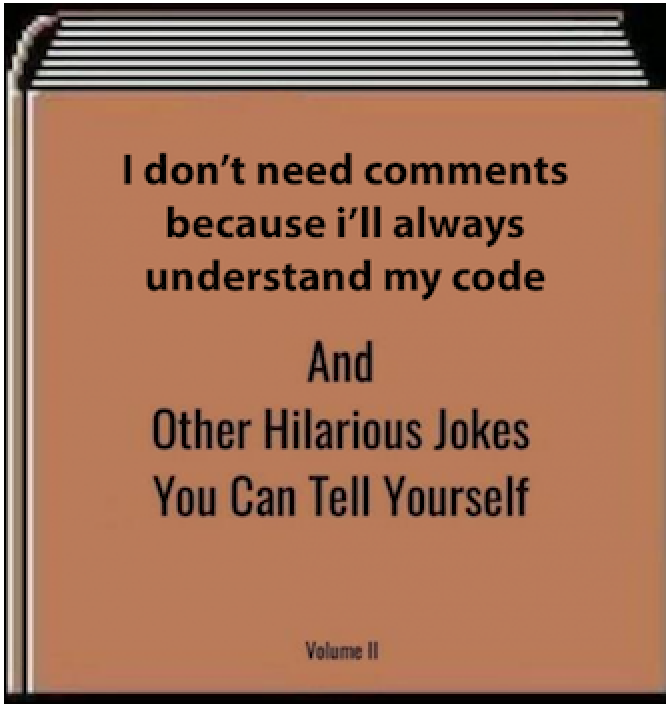
If you have to explain to me that your application works if you just do this arcane, barely-documented thing first, then the reality is your app doesn't work. Period.

7:10 PM - 9 May 2018

5 Retweets 21 Likes

3 5 21

Comment your functions - for the dev


I don't need comments because i'll always understand my code
And
Other Hilarious Jokes
You Can Tell Yourself
Volume II

Looking at your own code written more than 6 months ago is like looking at someone else code.

Comment your functions - for the dev

```
// The User class.
public class User {
  ...
}
```

Function documentation

Documenting your functions - the UX

- Doc is not about what it is
- Doc is about what is does

Documenting your functions - the UX

Documenting your package is essential: both for your "future you", and for other people who will use your package.

The documentation is what you will read in the Help tab once the package is installed, or via `?function` (or by pressing F1).

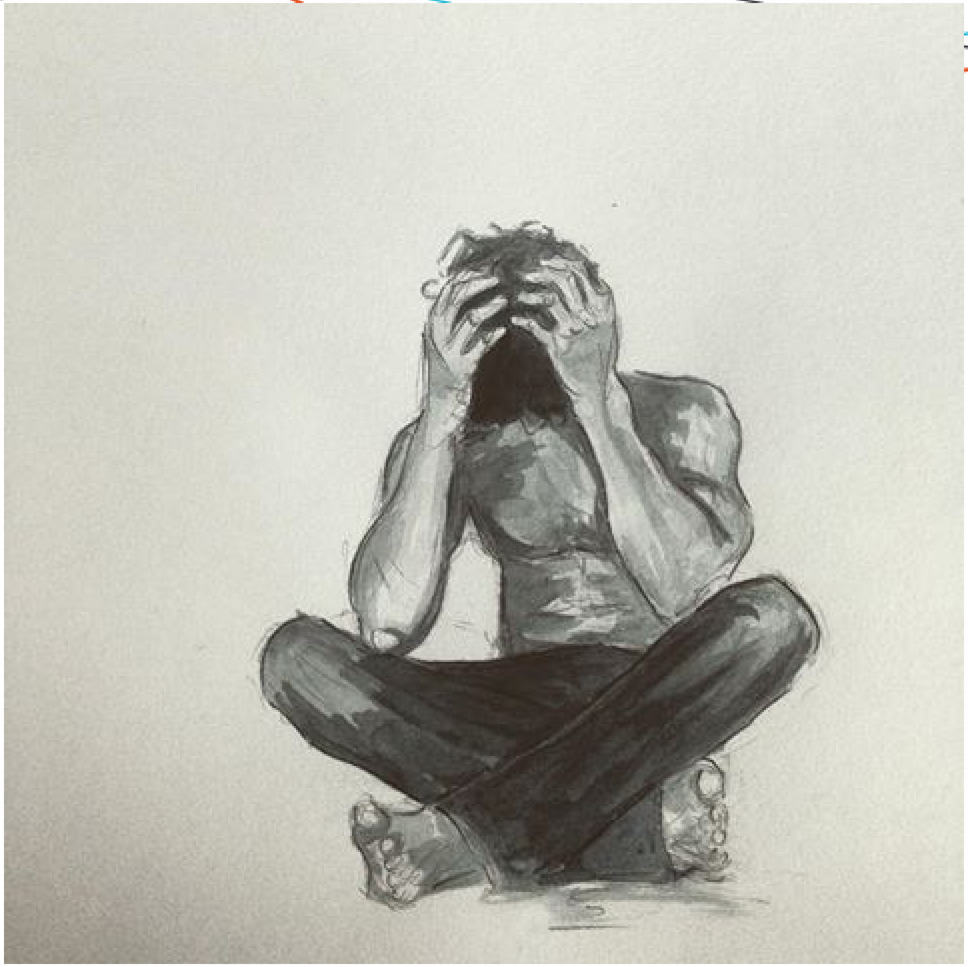
Enable roxygen2

In Build> Configure build tools, make sure to check "Generate documentation with roxygen" and all the boxes in the options window.

The documentation is managed by the .rd files contained in the man/ folder, generated in LaTeX, and are available in HTLM, text or pdf. Thanks to {roxygen2}, you dont have to type the LaTeX code by hand.

Using {roxygen2} has several advantages. Notably:

- You don't have to write LaTeX
- The documentation is in the same place as the object
- The documentation adapts to the type of object (function, data...)



"Writing Package Doc in LaTeX" - pre-roxygen2 period.

Documentation

Example:

```r
#' Title
#'
#' description
#' @param x param
#' @param y param
#' @return what the function returns
#' @examples
#' myfun(4,5)
#' myfun(5,9)
#' @export

myfun <- function(x, y){
  return(x+y)
}
```

Documentation

Each roxygen comment is preceded by `#'`. Then, we open with `@`, then the name of the filed to fill. RStudio offers autocompletion after the `@`.

If they are no `@`, the first two lines are understood as the title and description. Then, the most common fields are:

- `@details` : details on the function
- `@param x` a function param, with a description
- `@return` : what the function returns
- `@examples` : examples of how to use the function
- `@export` : **needed if you want the function to be accessible outside the package**
- `@import` and `@importFrom` : functions or packages to import
- `@source` and `@references` : references on the function
- `@section my_section` : add a custom section to the documentation

To know the list of parameters: `browseVignettes("roxygen2")`.

Documentation

Some roxygen fields make it easier to navigate in packages:

- `@seealso` : points to other resources, on the web or in the package
- `@family` : makes the function belong to a family of functions
- `@aliases` : gives "nicknames" to the function, to allow it to be found with `?my_alias`.

Documentation

Do not forget to create a help for the package. It can be called by the user by doing ?pkgname.

```
#' What the package does
#'
#' the description of the package, with details
#'
#' @name pkgname-package
#' @aliases pkgname-package pkgname
#' @docType package
#' @author colin <colin@@thinkr.fr>
NULL
```

Document several functions

There are several methods to document multiple functions "at once".

- Document a function, and comment with `@rdname`. Then, each new function with the same documentation must contain the same `@rdname`. This method generates a single .rd document for all functions with the same `@rdname`.
- use `@describeIn` : in a new function, we indicate where the documentation is located.
- inherit parameters with `@inheritParams` : in this case, you "only" inherit the specified parameters, not all the function documentation.

Misc

Prevent an example from being executed

If we put an example in \dontrun{}, it won't be evaluated when the package is built.

Document two parameters

#' @param x,y a numerical vector

Generate the doc

Once the documentation has been written, launch the roxygenise().

```
roxygen2::roxygenise()
```

Pkg documentation

A "UX first" approach

Building a package that lasts means that once your package is stable, you'll need **people to (be able) use it**.

"Be able" means: documentation should easily guide them.

Consider:

> "An R Package using the R6 paradigm to create an object oriented API designed to interactively and programmatically write Docker setup files inside an R session or script."

VS

> "Easy Dockerfile Creation with R6"

The truth is 90% of your end users don't care about technical stuffs.

Writing a README

```r
usethis::use_readme_rmd()
```

Creates a template of a README file, perfect for GitHub and for any other sharing platform.

This files should include:

- A quick explanation of what the package does
- How to install it
- Basic use of the package
- Where to fill issues / ask questions

Knit it to create a GitHub markdown

> Tips: a well written README can easily be used as an introduction Vignette.

Vignettes

A vignette is an html/pdf file that accompanies a package and is more developed than help pages, as the form is free (you can include images, tables, links, html...).

```
usethis::use_vignette("mypackage")
```

...creates a "mypackage.Rmd" file in a vignettes/ folder at the root of the package, and adds the appropriate dependency in your DESCRIPTION.

The vignette is a simple RMarkdown page.

Vignettes

```
---
title: "Vignette Title"
author: "Vignette Author"
date: `2018-05-14''
output: rmarkdown::html_vignette
thumbnail: >
  %\VignetteIndexEntry{Vignette
Title}
  %
ThumbnailEngine{knitr::rmarkdown}
  \usepackage[utf8]{inputenc}
---
```

Describes your vignette

We replace the title, the author, and the "VignetteIndexEntry".

Then, you can write the vignette as a classic Markdown file.

To preview the vignette rendering, press the knit button, or Ctrl/Cmd + Shift + K.

How many vignettes?

There are no right answers to this question, that depends of the size of your package and of what you have implemented.

Build Vignettes

Use `devtools::build_vignettes()` to render the vignette (in /inst/doc).

The end user can see the vignettes with the following instruction:

```r
browseVignettes("myPackage")
```

{pkgdown}

If you want to publish a "mini-site" on your package, you can use the {pkgdown} package.

```
devtools::install_github("hadley/pkgdown")
# or
install.packages("pkgdown")
```

Then

```
pkgdown::build_site()
```

This command will add a docs/ folder, in which you will find all the necessary elements to a mini-site built from README, your vignettes, and the documentation of your functions.

You can customize this site as a classic site (CSS...). The home of the site is on `index.html`.

{pkgdown} online

As these are plain html files, you can upload them to GitHub or to any other server.

{pkgdown} in your package

You can "install" the site in any folder:

```r
pkgdown::build_site(path ="inst/site")
```

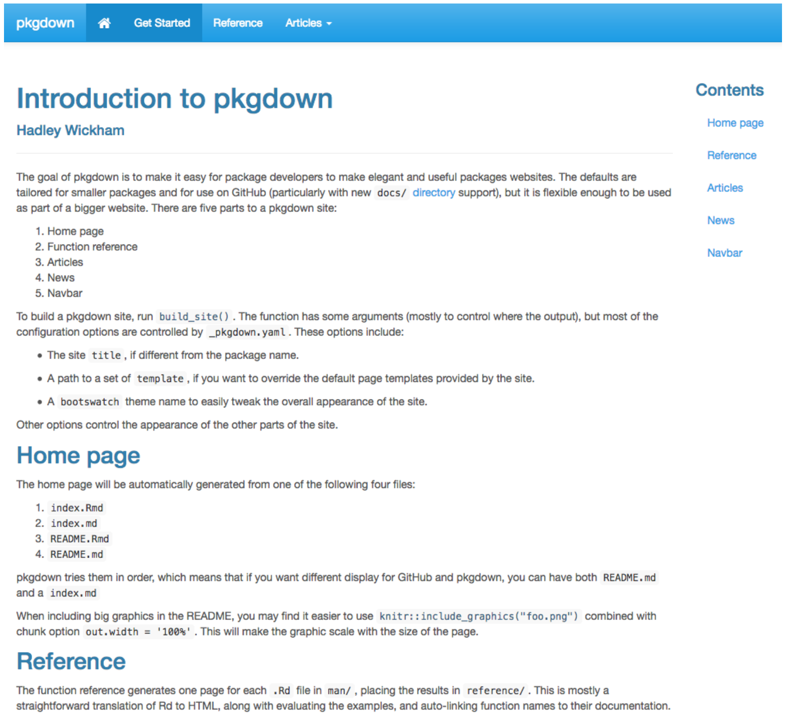
Reminder: all the elements contained in inst/ are moved at the root of the package folder once installed. It can be accessed with:

```r
path <- system.file("index.html",package ="mypkg")
browseURL(path)
```

We can therefore create a function:

```r
launch_help <- function() {
  browseURL(system.file("index.html",package ="mypkg"))
}
```

{pkgdown}



Introduction to pkgdown

Hadley Wickham

The goal of pkgdown is to make it easy for package developers to make elegant and useful packages websites. The defaults are tailored for smaller packages and for use on GitHub (particularly with new `docs/` directory support), but it is flexible enough to be used as part of a bigger website. There are five parts to a pkgdown site:

1. Home page
2. Function reference
3. Articles
4. News
5. Navbar

To build a pkgdown site, run `build_site()`. The function has some arguments (mostly to control where the output), but most of the configuration options are controlled by `_pkgdown.yaml`. These options include:

- The site `title`, if different from the package name.
- A path to a set of `template`, if you want to override the default page templates provided by the site.
- A `bootswatch` theme name to easily tweak the overall appearance of the site.

Other options control the appearance of the other parts of the site.

Home page

The home page will be automatically generated from one of the following four files:

1. `index.Rmd`
2. `index.md`
3. `README.Rmd`
4. `README.md`

pkgdown tries them in order, which means that if you want different display for GitHub and pkgdown, you can have both `README.md` and a `index.md`

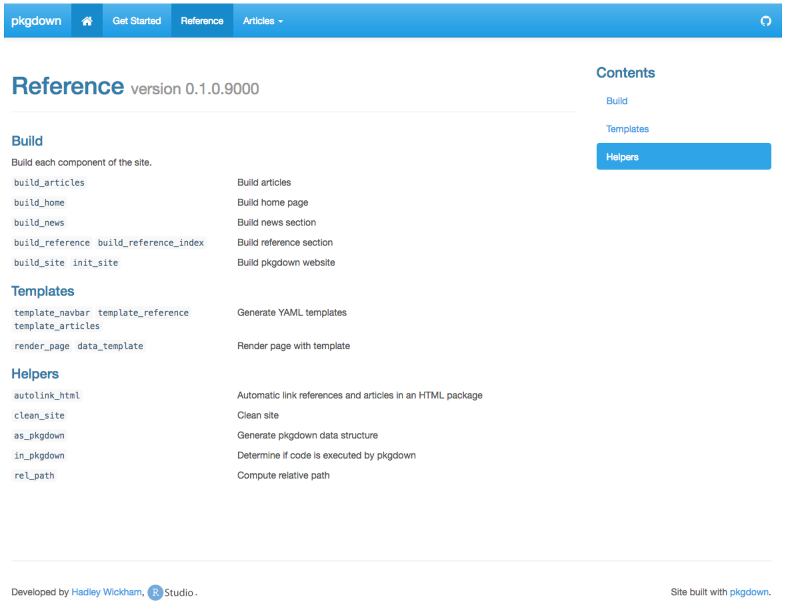
When including big graphics in the README, you may find it easier to use `knitr::include_graphics("foo.png")` combined with chunk option `out.width = '100%'`. This will make the graphic scale with the size of the page.

Reference

The function reference generates one page for each `.Rd` file in `man/`, placing the results in `reference/`. This is mostly a straightforward translation of Rd to HTML, along with evaluating the examples, and auto-linking function names to their documentation.

Contents

- Home page
- Reference
- Articles
- News
- Navbar



Reference version 0.1.0.9000

Build

Build each component of the site.

`build_articles`	Build articles
`build_home`	Build home page
`build_news`	Build news section
`build_reference` `build_reference_index`	Build reference section
`build_site` `init_site`	Build pkgdown website

Templates

`template_navbar` `template_reference` `template_articles`	Generate YAML templates
`render_page` `data_template`	Render page with template

Helpers

`autolink_html`	Automatic link references and articles in an HTML package
`clean_site`	Clean site
`as_pkgdown`	Generate pkgdown data structure
`in_pkgdown`	Determine if code is executed by pkgdown
`rel_path`	Compute relative path

Contents

- Build
- Templates
- Helpers

Developed by Hadley Wickham, RStudio.

Site built with pkgdown.

citation

There is a standardized form of quoting a package. We find it with `citation("package")`

```r
citation("purrr")
```

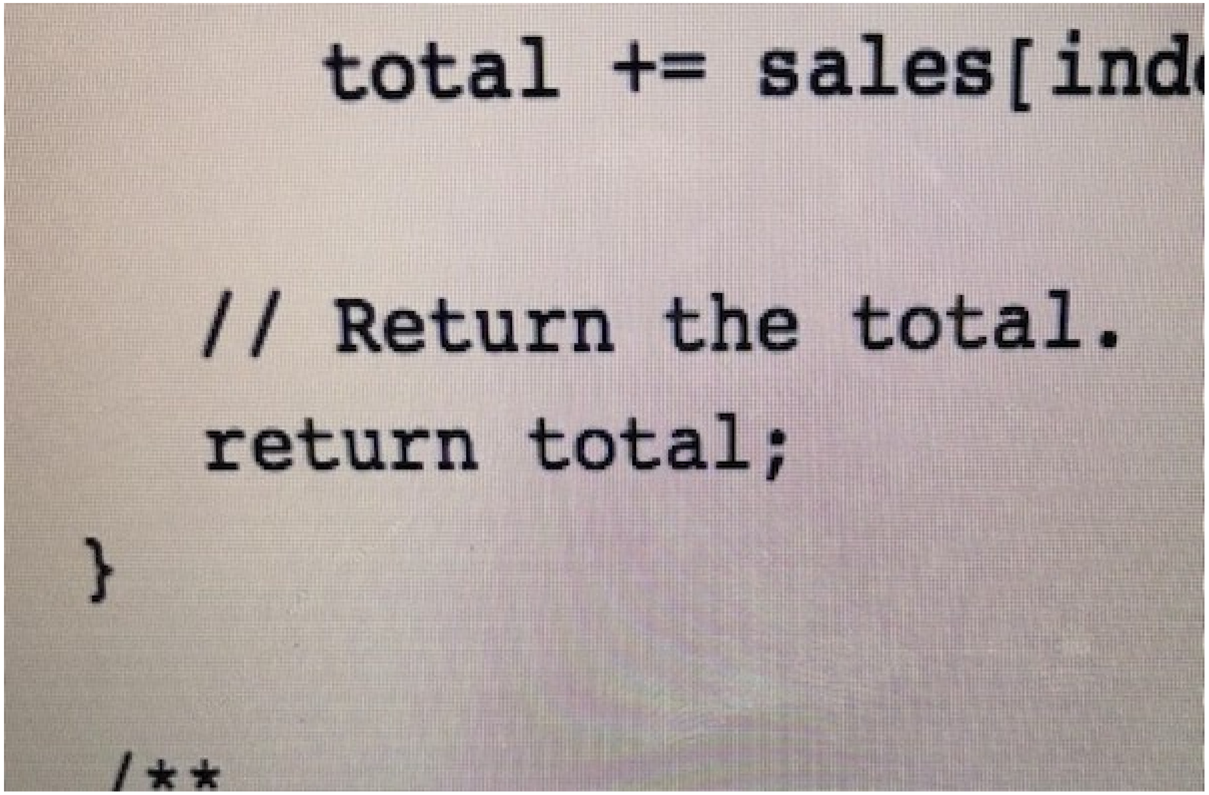
```
#>
#> To cite package 'purrr' in publications use:
#>
#>   Lionel Henry and Hadley Wickham (2017). purrr: Functional
#>   Programming Tools. R package version 0.2.4.
#>   https://CRAN.R-project.org/package=purrr
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {purrr: Functional Programming Tools},
#>     author = {Lionel Henry and Hadley Wickham},
#>     year = {2017},
#>     note = {R package version 0.2.4},
```

citation

It is possible to provide custom content for this function. In this case, we place a CITATION in the inst/ folder, which will have this form:

```
citHeader("Pour citer le lubrifiant dans les publications, utiliser :")
citEntry(entrée = "Article",
  title = "Dates and Times Made Easy with {lubridate}",
  author = personList(as.person("Garrett Grolemund"),
                   as.person("Hadley Wickham")),
  journal = "Journal of Statistical Software",
  année = "2011",
  volume = "40",
  nombre = "3",
  pages = "1--25",
  url = "http://www.jstatsoft.org/v40/i03/",
  textVersion = textVersion
  paste("Garrett Grolemund, Hadley Wickham (2011).".",
       "Dates and Times Made Easy with lubridate.",
       "Journal of Statistical Software, 40(3), 1-25.",
```

One more for the road



```
        total += sales[ind
    
    // Return the total.
    return total;
}

/**
```

Let's practice !