R packages development training

Mathieu Depetris

Table of contents

Welcome!

Welcome to the online support to learn how to organise your R code through the creation of an R package. You will find that this process is easier than you can imagine and with a little bit of pratiques you will be able to create your own package. In addition, you will discover that this way open you the field to develop reproducible code, documented faster and simply your processes at many levels and learn how other people can download and use it.

All this website and material associated if free to use, licenced under the CC BY-NC-ND 4.0 licence. At least you will find below several training sessions available if you want a guided explanation for R package creation.

To conclude, this support was developed according to the book *R Packages (2e)* [1] written by Hadley Wickham and Jennifer Bryan. An online version is available through this link, don't hesitate to take a look.

Introduction

To do

Part I

Training framework and requirements

1 - Operating system and working environment associated

Before going into the fun of R package development, take a moment to setup our working environment and overview some interesting software which can simplify our life.

First, all the training and test that we will do among this book will be executed with Windows Operating System (OS). When you develop your package, one of the aims could be to share or propose your work to other peoples and you have to take into account about the variability of the processes among OS, software for example R dependencies. We will put in place several processes to support us in our task but keep in your mind that processes that work on an OS could not work in the same pattern in another one. Futermore, one of our tasks among the lifecycle of our package will be to maintain a maximum of compatibility, for example through integration of update or evolution in our code.

Developing R packages assume that you have a little experience with the R software and not start from scratch with it. Even if you see that R package development is not so hard that it looks like, you will find a lot of resources on the internet to support you if you want to start with R and develop your skills in it.

1.1 Requirements software

For the training we will use the *R* version 4.5.0 for Windows available through the web page of The Comprehensive R Archive Network. You will find also precompiled binary distributions for Linux and macOS.

In addition, we will use the Integrated Development Environment (IDE) *RStudio*. Today this software is one of the most used in the R communities and includes a lot of things like a console, syntax-highlighting editor that supports direct code execution, and tools for plotting, history, debugging, and workspace management. Furthermore, recent evolution like the creation of Posit compagny make the software ecosystem more friendly connected with R and Python. For the training we will use the RStudio version 2025.05.0+496 "Mariposa Orchid" for Windows OS. You can download the one for your configuration throught the Posit download webpage.

Under Windows OS, it's necessary to install the RTools software. This is a toolchains for building R and R packages from source on Windows. And don't need it on macOS and Linux because theses OS have usually pre-installed compilation tools. You can download it from the offical link.Be aware that you need to take the correct version according to our R version installated (for this training it would be the RTools 4.5).

At least, we will using Git, for version control system, to host our R package and apply several processes of Continuous Integration (CI), most of them already prepare by the R communities. Today, this is a very precious tool that helps developers track changes in their code, collaborate efficiently, and manage different versions of a project. During this training we will use one of them. You can download the software on the offical website throught this link.

1.2 Additional software

Several software are not necessary to create your own R package but bring you a lot of improvement and at leave simplify you life.

The first one is Visual Studio Code (or VS Code) which is a free, lightweight, extensible code editor developed by Microsoft. He supports a wide variety of programming languages such as R, JavaScript, Python, C++, Java, HTML/CSS. In addition is cross-platform compatibility, support Git and other version control systems and one of his best advantages is to provide extensions to add functionality and customize the environment. If you want you can download it from this offical website and if it's possible according to your configuration prefer a system installer version. For this training I will use the version 1.100.2 on Windows OS.

According to the use of Git, we will use one of them call GitHub. You will see that a loot of processes were developed for work natively with GitHub and facilitate your R package creation. If you want to use another one, like a GitLab hosted by your company for example, you can of course but you will have to modify and adapted by yourself several processes propose during this training. In addition we will use here a Graphical User Interface (GUI) for managing Git repositories called GitHub Desktop. This is a free, open-source application which simplifies current Git operations, making it easier for developers to clone repositories, commit changes, create branches, and push updates without using the command line. Like its name suggests is working natively with GitHub repositories but you can also use it for connection with other Git like GitLab. Officially is available for Windows and MacOS but you can also find adapted version for Linux on the internet. For download it goes through the following link.

2 - Overview of R packages used

This section aim to introduce brively which R packages environment we will use. Since several years, the R communities evolve in many interesting ways and today if you want to write your own package there are a lot of support that you can find.

First of all, it’s difficult to talk about package whithout talk about the tidyverse approach. Tidyverse is a collection of R packages designed for data science. They share an underlying design philosophy, grammar, and data structures. If you use R, it’s allmost sure that you have allready use this kind of packages, maybe even without knowing them. Even if there are none mandatary for R use, theses packages bring a lot of benefices and at the end make data science easier, faster, and more fun (yes, you will see).

So far we don’t go deeper into all the useful R packages of the tidyverse collection, but only focusing on theses related to the R packages developement.

The first one is called devtools [2]. The aim of this package is to make R package development easier by providing R functions that simplify and expedite common tasks. In practical, if you use the Rstudio IDE you will not interact directly with this package because the majority of common processes will be integrated automatically in the Rstudio environment and become “click button” processes.

The second one is the sub-package you are most likely to interact with directly. It be called usethis [3] and is aim is to automates repetitive tasks that arise during project setup and development. The idea is to, if it's possible, use functions available by the package, rather that modify directely the code, to reduce error and be sure to have the last version of process strucutre.

There are two more important packages related to design of the documentation: roxygen2 [4] and pkgdown [5]. With the first one throught the use of tags, the code and the documentation are adjacent and dynamically inspects the objects that it's documenting. We will go back to them into the following sections but keep in your mind that theses two tools simplify and automate the documentation creation and evolution.

The last one is called testthat [6] and focusing on the field of testing your code, which can be painful and tedious, but it greatly increases the quality of your code.

In addition we will use or talk about several other packages, for example lintr [7] or renv [8], but it's more a case-by-case utilisation and we will discuss them when the time comes.

3 - Before go down the rabbit hole

Before we start to developing our first R package, let's talk briefly about the aim to organise your code into a package and in fact why we consume time to develop this kind of structure.

When you use R, a package is the fundamental unit of shareable codes and brings a lot of advantage, like regroup in one place all the stuff (code, data, documentation for example), improve the standardisation of your process and the sharing them easier. In fact, build an R package is nothing more than a way of organising things that you have already done by another way and facilitate at the end the life cycle of your processes.

Furthermore, you will see quickly that you can use a lot of processes already developed ,and update in time, by the R communauties. There are package that make package, processes that allow to automated tasks and the tidyverse philosophy explain in the previous section help a lot.

To summarise:

- don't worry, you will see that all this process is painless and you will be able to create very great thing in brief time,
- in addition, you will save time! you will follow a template and automate a lot of processes with the rule "anything that can be automated, should be automated",
- to conclude, you will use standardised conventions that open to you the path a several free and evolving standardised tools.

Part II

First part - R package architecture

4 - Before create anything

Before starting playing with functions and create the global structure of your package, the best thing to do is to think a little bit about what you want to create. The idea is not to spend a very long time to do that, but if you take this time (especially is it's the first time) you will see that at the end you will gain. The majority of the package component can be modified after definition, but some update could be heavy and painful and sometimes it's simple to avoid that by taking a moment of reflection.

4.1 First reflexion, the perimeter

The aim here is to answer the question, what I want to put into my package or what I want my package will do? You don't have to be exhaustive (if you can it's better) but try to explore several fields like:

- If I have several functions already developed (even partially), it's logical to put all of them into a single structure?
- If my package will be used by other users, does the process of use if relevant and logical for users?
- If I think bigger, do you think my package could be placed in specific environment and maybe interact with other packages? A good example of that is if a package interact with other packages and each one has a specific purpose (for example one dedicates to the controls and another dedicates to the graphic display).

You can image other cases and again it's normal if your perimeter evolve with time regarding the maturity of your work. A good solution could be to design flux diagrams for example explain (even for you) the process and maybe associated one. In addition, these kinds of figures could be useful for the documentation process. A quick research on the internet will be displaying several software, free and paid, for doing that.

4.2 Second reflexion, the name

Now you know globally the purpose of my package. The next step is to define is named. It's an important moment because the name is one of the first things that you and the others will see. At a time where the referencing (like the SEO or Search Engine Optimisation) and visibility is important items regarding the life of your work, it could be bad to lose points at this moment just because the name is not very clear and friendly.

Globably, I advise you to follow several good praticals:

- the name has to be **simple** and **clearness**. What's better than a name that's easy to understand and remember and ideally allows a reading to understand what it does.
- avoid **special caracters**, like the accents or dot, and prefer use convention like Snake Case (each word is separated by and underscore character) or at least Camel Case (start with the first word with lowercase and capitalize the first letter of each word that follows withou space).
- insure the **compatibilty** of the name with package respositories like the CRAN (Comprehensive R Archive Network) or Bioconductor.

To support you in this step, you can use functions of two R packages, pak [9] and available [10].

5 - Creation of the package structure

5.1 Initiation the global architecture

Now you have an working environnement correctly configured, the first step is to initiate your new package from an active R session. You don't have to take care about your working directory or the configuration of your local session because the function associated will be create a clean new project for your package.

References

1. Wickham H, Bryan J (2023) R packages: Organize, test, document, and share your code (2nd edition). O'Reilly Media

2. Wickham H, Hester J, Chang W, Bryan J (2022) Devtools: Tools to make developing r packages easier

3. Wickham H, Bryan J, Barrett M, Teucher A (2024) Usethis: Automate package and project setup

4. Wickham H, Danenberg P, Csárdi G, Eugster M (2024) roxygen2: In-line documentation for r

5. Wickham H, Hesselberth J, Salmon M, Roy O, Brüggemann S (2025) Pkgdown: Make static HTML documentation for a package

6. Wickham H (2011) Testthat: Get started with testing. The R Journal 3:5–10

7. Hester J, Angly F, Hyde R, et al (2025) Lintr: A 'linter' for r code

8. Ushey K, Wickham H (2025) Renv: Project environments

9. Csárdi G, Hester J (2025) Pak: Another approach to package installation

10. Ganz C, Csárdi G, Hester J, Lewis M, Tatman R (2022) Available: Check if the title of a package is available, appropriate and interesting

A Not define yet

Appendice a