Guidelines for managing R packages

Cedric Arisdakessian

2022-07-28

This document is heavily inspired by: https://tinyheero.github.io/jekyll/update/2015/07/26/making-your-first-R-package.html

Pre-requisites

- GitHub access to repository
- Install `devtools` and `roxygen`
- (optional) install `usehat` if you want to build the vignette

Repository layout for an R package

With the `devtools` R library, you can build your own library directly from a github repository. However, `devtools` requires the repository to have a specific layout. More specifically, you need the following:

Mandatory:

- `R/` folder with your code. Only write functions or R objects (for more advanced R users)
- `DESCRIPTION` file, automatically generated if you initialized your package with `devtools::create("package_name")` . This file contains:
  - Metadata, such as its name, title, version, authors and other optional fields
  - Dependencies, either mandatory (in the "Imports" category) or optional (in the "Suggests" category).
- `NAMESPACE` file (automatically generated or updated each time `devtools::document()` is called. It contains all the function that should be available to the user when using the package.

Optional:

- `README.md` file
- External data in `inst/extdata` folder

Note that once you build the package, a `man/` folder (short for manual) is automatically generated with all the package documentation.

In the case of the labhuiofrank package, all of these files are present in the labhuiofrank github repository.

Install or update an existing R package from github

You can install or update an R package if it follows the previously described structure. Simply run:

```r
devtools::install_github("{username}/{respository_name}")

# example with the labhuiofrank.16S package
devtools::install_github("labhuiofrank/16S-data-analysis")
```

It is usually considered a good practice to have the same name for the github repository and the R package. It is however not the case for our package (16S-data-analysis vs labhuiofrank.16S)

Writing your own functions

If you want to add your own functions to an existing package, you will need to either include a file in the `R/` folder or modify an existing one. Let's see an example of how we can do that.

First, let's see an example of an R function that greets a person:

```r
greet <- function(first_name, last_name) {
  msg <- sprintf("Hello %s %s", first_name, last_name)
  return(msg)
}
```

Although we know how this function works, another person might not understand it. Thus, we need to provide documentation about what it does and how to use it. All the documentation is written before the function definition, and start with the 2 characters `#'` (pound + single quote). The documentation entries are split into different categories, such as the description, parameters, return values, etc.. Depending on the category, the syntax of a documentation entry can be (you will need to replace flag / name / descr with the appropriate value):

A. `#' text`
B. `#' @flag descr`
C. `#' @flag name descr`

In fact, you don't have to provide any documentation to your function. However, I would highly recommend you to provide the following ones:

1. Description of the function (syntax A). If the description is long, it's good to break it into multiple lines (each line still needs to start with `#'`)
2. Description of each parameter (syntax C, flag: `@param`): The flag needs to be followed by the parameter name and its description, both separated by a single space.
3. Return value (syntax B, flag: @return)
4. **Important**: If you want the function to be available in the package, you need to add a line with the flag `@export`
5. Examples (syntax B, flag: `@examples`). It is a good practice to provide the user some examples of how to use the function. Each example is written in a new line starting with `#'`

See below an example for the documentation of our `greet()` function:

```r
#' Great a person using their first and last name
#' @param first_name First name of the person to greet
#' @param last_name Last name of the person to greet (optional)
#' @return Greeting message
#' @export
#' @examples
#' greet("Kiana", "Frank")
#' greet("Cedric")
greet <- function(first_name, last_name="") {
  msg <- sprintf("Hello %s %s", first_name, last_name)
  return(msg)
}
```

For more examples, see the labhuiofrank github repository here

Once you have added your new functions, you will need to update the package. Go to root of the github repository folder, and run the following commands:

```r
# build the documentation and update the package's associated functions
devtools::document()
# Optional, build the vignette
usethat::use_vignette("introduction")
```

Include data in the package

In general, you should limit the amount of raw data you include in the package, especially if it's voluminous. GitHub might even prevent you to upload files larger than a few MB. However, it can be useful to include some small database or test data directly available upon loading the package.

To include data, you just need to add it to the `inst/extdata` folder in the GitHub repository. It will be available after you reinstall the package.

If you need to start a new package

Simply run:

```r
devtools::create("package_name")
```

And create a github repository for the folder created by `devtools`.