R package development tutorial

a gentle introduction

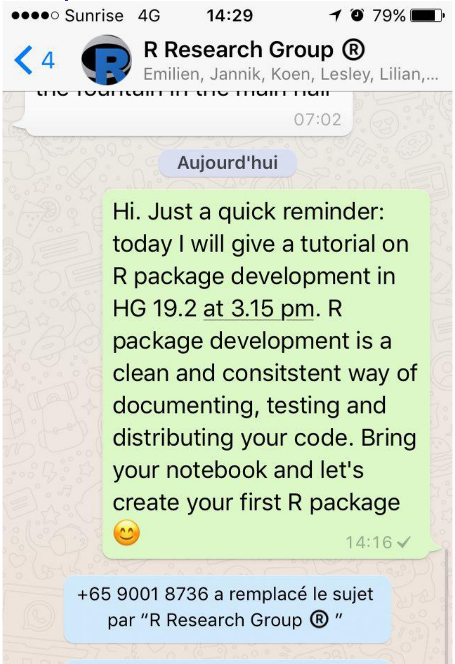
Lorenz Walthert (@lorenzwalthert)

March 7, 2018

Today's plan

- first 60 minutes: Theory + group practice.
- last 30 minutes: (optional participation): Training problems, general questions.

The impact of this tutorial


●●●●○ Sunrise 4G 14:29 79%

R Research Group ®

Emilien, Jannik, Koen, Lesley, Lilian,...

07:02

Aujourd'hui

Hi. Just a quick reminder: today I will give a tutorial on R package development in HG 19.2 at 3.15 pm. R package development is a clean and consitstent way of documenting, testing and distributing your code. Bring your notebook and let's create your first R package 😊 14:16 ✓

+65 9001 8736 a remplacé le sujet par "R Research Group ® "

What is R package development?

Process of

- writing code in a standardized framework.
- writing code for generic use.

Why should I develop an R Package?

Tell me why :-)

Why should I develop an R Package?

Standard way of

- documenting
- testing
- distributing

code

What is a package (states)?

- source code (when you create it).
- built package (what you save to your disk with `install.pacakges()`).

What is a package (source code)?

A directory with a few files / folders:

- DESCRIPTION (meta info)
- NAMESPACE (import / export)
- R/ (your functions)
- man/ (your documentation)
- vignettes/ (long form documentation)
- tests/ (tests)
- data/ (your data)

Creating a package (skeleton)

Let's do it together.

- Open RStudio.
- New RStudio Project -> Package

Done :-)

A few basic things about using Rstudio

- go to file/function.
- file browser.
- break points and `browser()`
- Command history (arrow + recursive search).

Clean-up

- make NAMESPACE ready for roxygen2 by overwriting NAMESPACE contents with `# Generated by roxygen2: do not edit by hand`.
- remove the documentation file for `hello-world` (how?) and the code file `hello-world.R` (how).
- enable markdown support `Roxygen: list(markdown = TRUE)`. This will allow you to use markdown syntax.
- adapt DESCRIPION to your specific needs.

Intermezzo: Markdown syntax

- `[label](https://...)` for web-links
- `[fun()]` for cross-references
- ‘code‘ for code font.
- stars /numbers with dots for enumerations.
- ...

Framework for package development

We are going to use the devtools framework with uses the following R packages:

- `roxygen2` for in line documentation.
- `testthat` for unit tests.
- `usethis` for common non-coding, manipulating, editing tasks.

roxygen

- Special comments `#'` we use in source code are called roxygen comments.
- `roxygen2` extracts those and creates documentation in `man/` for you. The file format of the documentation is `.Rd`.
- Alternative: learn latex-like language and write plain `.Rd`.

Adding a function to your package

- `>usethis::use_r("my-file-name")` to create a new file `my-file-name.R` in `R/`.
- Create a function. I suggest `add_two_numbers <- function(x, y) {`

  `sum(x, y)`

`}`

- Suggest to name files by context, not by function.
- Can have multiple function declarations in one file.
- `R/` is only for declarations, no top-level function calls.
- Exception: Obviously in function declarations, you can have calls.

Adding a function to your package

- How can I use this function interactively? -> Workflow

Intermezzo: Namespaces

- Every object in R is defined in a “context”.
- The order of the namespaces on the search path defines where `R` will look for an object first.
- What does `search()`?
- You probably used the global namespace so far.
- What happens when you call `library()`?
- Every package a name space.
- That’s why you can redefine `var()` and `sd()` is unaffected (Why?).

Workflow

- Edit a file.
- Run `devtools::load_all()`. Just updates the package namespace.
- Don't put anything in the global name space, e.g. via `source`.
- Run `devtools::document()` to generate `.Rd` from roxygen comments.

devtools::load_all()

- This updates the namespace in your current R session.
- You don't change the package in the place where you have all your other packages stored.
- For this reason: (i) it is much faster than building the package with `devtools::build()` and (ii) it is not available in other R sessions.

Adding a function to your package

- How can I use this function as a “usual” R package in any R session?
- Need to build the package.

NAMESPACE stuff

- decide whether you want function to be exported (=available for the end user) (roxygen tag `@export`).
- if you need functions from other packages: `usethis::use_package("pkgname")`. This will add package to DESCRIPTION.
- then, refer to the function from the package with `pkgname::fun()`.
- Alternative: Refer to package with `fun()` and add roxygen tag `@importFrom packagename fun`. This will add package and function to NAMESPACE.
- You can often use a roxygen comment to refer to multiple "things". I.e. `@importFrom pkgname fun1 fun2 fun3`.

Documentation

- Example: `style_pkg()` from the `styler` package.
- Compare (i) rendered .Rd with `styler::style_pkg()` with (ii) the source at https://github.com/r-lib/styler/blob/master/R/ui.R#L6

Structure:

- Title
- two lines blank
- description
- @param x,y Vectors.
- @export
- @examples

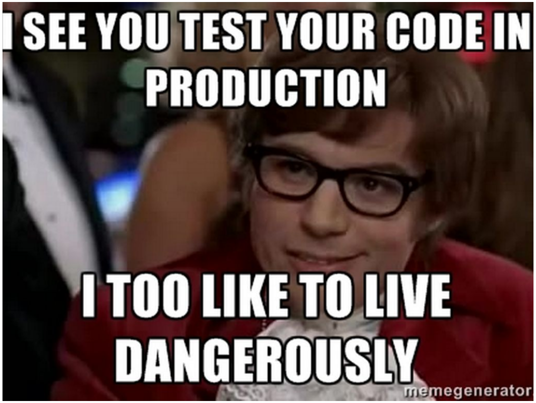
Documentation

Other useful roxygen tags:

- @inheritParams
- @seeAlso
- @family
- ...

Start at this vignette (has a some non-markdown syntax in there like `\code{x}`, just use it for the tags).

Testing (basic idea)


I SEE YOU TEST YOUR CODE IN PRODUCTION
I TOO LIKE TO LIVE DANGEROUSLY
memegenerator

Testing (basic idea)

Testing (basic idea)

- Why? Complexity grows exponentially with internal dependencies.
- test whether your function returns what you expect.
- I.e. match result of function call with reference.

Testing (basic idea)

- Example with `add_two_numbers()`?
- unit testing -> test (small) units.
- fail fast.
- high coverage (-> `covr` package)

Testing (Setting up)

- tests life in `tests/`
- we use the package testthat for unit testing.
- `>usethis::use_testthat()` to add infrastructure

Testing (adding a test)

- `usethis::use_test("testname")` adds a test file in `tests/testthat`
- all testthat test file names start with `test-`.

Testing (writing tests)

- `context()`: Useful when running tests to see which one fail / succeed.
- ‘test_that(“x y z returns correct number of rows”, [code]).
- Form sentences for easy understanding: “Test that x y z returns correct number of rows”

Testing (testing functions)

- most testthat functions start with `expect_`.
- `expect_silent()`
- `expect_s3class()`
- `expect_equal()`
- `expect_true()`
- `expect_equal_to_reference()` (use git ! or `update = FALSE`)
- ...
- `skip_*()` to skip platform dependent.

How to run tests

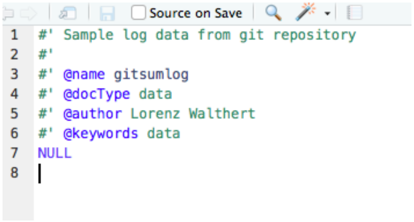
- Cmd+Shift+T or build tab.
- Note that default working directory of for tests is not root directory of package.
- Using the `rprojroot` package to find these files. Also useful for .Rmd files (since their default working directory is where the file is located).

```
testthat_file <- function(...) {
file.path(rprojroot::find_testthat_root_file(), ...)
}
```

- Then, in tests, refer to files relative to testtthat directory as follows:
- `expect_equal_to_references(my_object, test_that_file("reference-objects/ref-1"))`

Adding data

- `>usethis::use_data(dataset)` adds data with the name `dataset`.



```r
#' Sample log data from git repository
#'
#' @name gitsumlog
#' @docType data
#' @author Lorenz Walthert
#' @keywords data
NULL

```

Putting it all together: R CMD CHECK

Asserts a whole range of things, in particular:

- documentation matches code (all arguments documented).
- all unit tests pass.
- package can be installed
- ...

Run `devtools::check()` (or simply Ctrl+Shift+E)

Let's make your package pass R CMD CHECK

Iterative process of

- running `devtools::check()`.
- fixing errors, warnings, notes.

Using continuous integration

Can run R CMD check locally, but also on fresh copy of

- Linux / mac (with Travis)
- Windows (with AppVeyor)

More add ons

- `covr` for code coverage reports.
- slack notifications etc.

An example: www.github.com/r-lib/styler

Vignettes

- Long for documentation.
- `>usethis::use_vignette("vignette-title")`
- can be found via package index.
- build vignettes into `inst/` with `devtools::build_vignettes()`

pkgdown

- Package to create nice html documentation from your source code.
- can be hosted on GitHub.

An example: www.github.com/r-lib/styler

Releases

- Why releases?
- Suggested form: `major.minor.patch`, e.g. `0.1.2`
- Change in `DESCRIPTION`

Putting your package on GitHub

Last weeks course:

1. initialize GitHub repo locally.
2. Create repo on GitHub
3. Go back to command line and push

Installing from GitHub

`remotes::install_github("[username]/[repo]")`, e.g. `remotes::install_github("tidyverse/dplyr")`.

Anouncement

- We are looking for R package developers in Google Summer of code for styler
- 3 months 1:1 code review from me and Kirill Müller (2nd most CRAN downloads after Hadley Wickham)
- CHF 6600 from Google
- and (probably) also ~6 credits from ETH for applied area.
- Talk to me afterwards or go to this GitHub repo.

Closing

- Go here for more details: http://r-pkgs.had.co.nz
- Thanks for your attention.