Creating a simple R Package

Gaston Sanchez

October 19, 2016

Creating a minimalist R package

More information can be found in Hadley Wickham's **R Packages** book:

http://r-pkgs.had.co.nz/intro.html

Required Packages

In order to create a package make sure you have the following packages:

```
install.packages(c("devtools", "roxygen2", "testthat", "knitr"))
```

Creating a package as an R Project

Here's the list of suggested steps to create an R package from scratch

- Open RStudio
- Go to `File` in the menu bar
- Select `New Project`
- Choose `New Directory`
- Choose `R Package`
- Specify a name for the package (e.g. `oski`), and choose directory
- Click on button `Create Project`

What's in the package structure?

The series of previous steps should initialize an "R project" with the following contents:

```
.Rbuildignore
oski.Rproj
DESCRIPTION
NAMESPACE
R/
man/
.Rproj.user/
```

- Open the `DESCRIPTION` file and customize its fields with your information
- Don't touch `NAMESPACE` (this will be updated via `devtools`)
- Don't modify the contents in `man/`
- Add R code (functions) in the directory `R/`

Adding code and Roxygen comments

- Go to the `R/` directory and add your own R scripts (containing the functions).
- Document your code using Roxygen comments
- Add examples to the main functions

Here's an example of code for a function `coin()` (in file `coin.R`)

```
#' @title Coin
#' @description Creates an object of class \code{"coin"}
#' @param sides vector of coin sides
#' @param prob vector of side probabilities
#' @return an object of class coin
#' @export
#' @examples
#' # default
#' coin1 <- coin()
#'
#' # another coin
#' coin2 <- coin(c('h', 't'))
#'
#' # us cent
#' cent1 <- coin(c('lincoln', 'shield'))
#'
#' # loaded coin
#' loaded <- coin(prob = c(0.7, 0.3))
#'
coin <- function(sides = c("heads", "tails"), prob = c(0.5, 0.5)) {
  object <- list(
    sides = sides,
    prob = prob)
  class(object) <- "coin"
  object
}
```

Workflow with `"devtools"`

The typical package development workflow with `"devtools"` consists of:

- generating the documentation: `.Rd` files in `man/` directory
- checking that the documentation is OK
- build the bundled package: creates the compressed file (a.k.a. *tarball* `.tar.gz` file); this file ca be shared and installed in any platform.
- install the package: installs the bundled package.

We are ssuming that you are using an R project to work in your package. In this way, R's working directory is actually the directory of your project. Here's the `devtools` commands you should use:

```r
library(devtools)

# creating documentation (i.e. the Rd files in man/)
devtools::document()

# checking documentation
devtools::check_man()

# building tarball (e.g. oski_0.1.tar.gz)
devtools::build()

# checking install
devtools::install()
```

Including tests for your functions

There are various packages in R that allow you to include unit tests. One of them is the package `"testthat"`.

R package `"testthat"`

`"testthat"` is one of the packages in R that helps you write tests for your functions. One of the main references is the paper *testthat: Get Started with Testing* by Hadley Wickham (see link below). This paper clearly describes the philisoply and workflow of `"testthat"`. But keep in mind that since the introduction of the package, many more functions haven been added to it, and some have been deprecated.

https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

More technical information on tests for a package is here:

http://r-pkgs.had.co.nz/tests.html

About `"testthat"`

- `"testthat"` provides a testing framework for R that is easy to learn and use
- `"testthat"` has a hierarchical structure made up of:
  - expectations
  - tests
  - contexts
- A **context** involves **tests** formed by groups of **expectations**
- Each structure has associated functions:
  - `expect_that()` for expectations (see table below)
  - `test_that()` for groups of tests
  - `context()` for contexts

List of common expectation functions

Function	Description
`expect_true(x)`	expects that `x` is `TRUE`
`expect_false(x)`	expects that `x` is `FALSE`
`expect_null(x)`	expects that `x` is `NULL`
`expect_type(x)`	expects that `x` is of type `y`
`expect_is(x, y)`	expects that `x` is of class `y`
`expect_length(x, y)`	expects that `x` is of length `y`
`expect_equal(x, y)`	expects that `x` is equal to `y`
`expect_equivalent(x, y)`	expects that `x` is equivalent to `y`
`expect_identical(x, y)`	expects that `x` is identical to `y`
`expect_lt(x, y)`	expects that `x` is less than `y`
`expect_gt(x, y)`	expects that `x` is greater than `y`
`expect_lte(x, y)`	expects that `x` is less than or equal to `y`
`expect_gte(x, y)`	expects that `x` is greater than or equal `y`
`expect_named(x)`	expects that `x` has names `y`
`expect_matches(x, y)`	expects that `x` matches `y` (regex)
`expect_message(x, y)`	expects that `x` gives message `y`
`expect_warning(x, y)`	expects that `x` gives warning `y`
`expect_error(x, y)`	expects that `x` throws error `y`

Tests for an R package

- In the directory of the package, create a folder `"tests"`
- Inside the folder `tests/` create another folder `"testthat"`; this is where you include R scripts containing the unit tests.
- All the script files inside `testthat/` should start with tha name **test** e.g. `test-coin.R`, `test-flip.R`, `test-toss.R`, etc.
- Inside the folder `testthat/`, create an R script `test-check-coin.R`

Your Turn

Let's modify the code of `coin()`, in the file `coin.R`, by including a couple of auxiliary checking functions: `check_sides()` and `check_prob()`:

```
coin <- function(sides = c("heads", "tails"), prob = c(0.5, 0.5)) {
  check_sides(sides)
  check_prob(prob)

  object <- list(
    sides = sides,
    prob = prob)
```

```
  class(object) <- "coin"
  object
}


# private function to check vector of sides
check_sides <- function(sides) {
  if (length(sides) != 2) {
    stop("\n'prob' must be a vector of length 2")
  }
  if (!is.numeric(sides) & !is.character(sides)) {
    stop("\n'sides' must be a character or numeric vector")
  }
  TRUE
}


# private funciton to check vector of probabilities
check_prob <- function(prob) {
  if (length(prob) != 2 | !is.numeric(prob)) {
    stop("\n'prob' must be a numeric vector of length 2")
  }
  if (any(is.na(prob))) {
    stop("\n'prob' cannot contain missing values")
  }
  if (any(prob < 0) | any(prob > 1)) {
    stop("\n'prob' values must be between 0 and 1")
  }
  if (sum(prob) != 1) {
    stop("\nelements in 'prob' must add up to 1")
  }
  TRUE
}
```

Tests

The idea is to write tests for `check_sides()`, `check_prob()`, and `coin()`.

- Go to the `tests/testthat/` folder and open the `test-check-coin.R` file
- use `context()` to describe what the test are about
- to run the tests from the R console, use the function `test_file()`
- Give a `context()` to to describe what the test are about.
- You typically group various related tests in one context; e.g. a context for *coin arguments*
- In a given context, you form groups of expectations inside calls to `test_that()`
- For example, you can expect that `check_sides()` returns `TRUE` if the value of `sides` is of length 2.

- Likewise, you can expect an error from `check_sides()` if the values of `sides` is of length different than 2:

```
context("Coin arguments")

test_that("check_sides with ok vectors", {

  expect_true(check_sides(c('heads', 'tails')))
})


test_that("check_sides fails with invalid lengths", {

  expect_error(check_sides(c('one', 'two', 'three')))
  expect_error(check_sides(c('one')))
})
```

To run the tests, you use the `"devtools"` function `test()`. This means that your typical packaging workflow will look like this:

```
library(devtools)

# creating documentation (i.e. the Rd files in man/)
devtools::document()

# checking documentation
devtools::check_man()

# running tests
devtools::test()

# building tarball (e.g. oski_0.1.tar.gz)
devtools::build()

# checking install
devtools::install()
```

Vignettes

Another element to consider adding to a package is a vignette. A vignette is supporting material—usually in the form of a tutorial—that describes how to use the main functions of the package.

http://r-pkgs.had.co.nz/vignettes.html

To create your first vignette with `"devtools"`, run:

```
devtools::use_vignette("my-vignette")
```

This will:

1. Create a `vignettes/` directory.
2. Add the necessary dependencies to `DESCRIPTION` (i.e. it adds `knitr` to the `Suggests` and `VignetteBuilder` fields).
3. Make a draft vignette, `vignettes/my-vignette.Rmd`.

Once you have this file, you need to modify the vignette. After editing the vignette, your expanded workflow will be like this:

```
library(devtools)

# creating documentation (i.e. the Rd files in man/)
devtools::document()

# checking documentation
devtools::check_man()

# run tests
devtools::test()

# checking documentation
devtools::build_vignettes()

# building tarball (e.g. oski_0.1.tar.gz)
devtools::build()

# checking install
devtools::install()
```