Building R Packages

Emil Hvitfeldt

2019-3-28

Slides and material

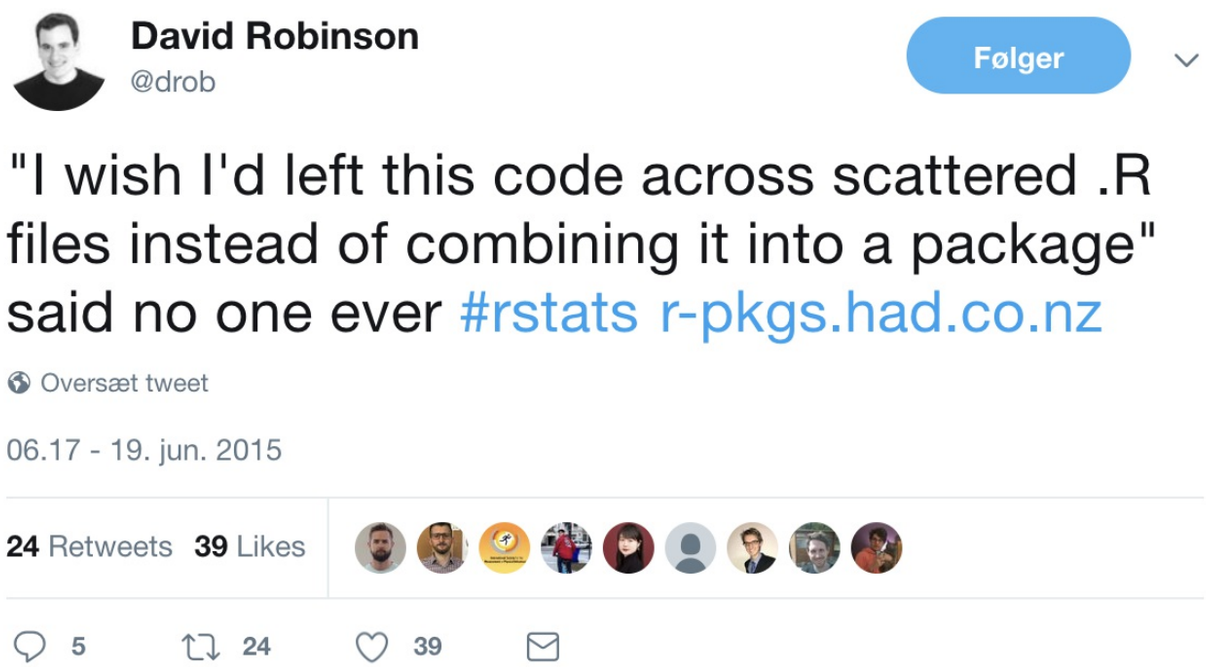
Slides are up now
https://github.com/USCbiostats/software-dev

Chapter will be up by end of week
https://github.com/USCbiostats/handbook

Overview

- Motivation
- Minimal R Package
- Not so minimal R package
- Live demo

Motivation


David Robinson
@drob
Følger
"I wish I'd left this code across scattered .R files instead of combining it into a package" said no one ever #rstats r-pkgs.had.co.nz
Oversæt tweet
06.17 - 19. jun. 2015
24 Retweets 39 Likes
5
24
39

What is a R package?

- Packages are collections of R functions, data, and compiled code in a well-defined format.

R comes pre-installed with some packages (base packages)

```
rownames(installed.packages(priority = "base"))
```

```
##  [1] "base"      "compiler"  "datasets"  "graphics"  "grDevices"
##  [6] "grid"      "methods"   "parallel"  "splines"   "stats"
## [11] "stats4"    "tcltk"     "tools"     "utils"
```

And some are loaded automatically

```
getOption("defaultPackages")
```

```
## [1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
```

Why create a R package?

Saving yourself time

Sharing

Organization

Collaboration

Credit

Deployment

Personal

It never have to leave your computer

Open-source it

Put it on Github/Gitlab

CRAN / Bioconductor

Package Manuals

Light

https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/

Medium

https://r-pkgs.org/

Heavy

https://cran.r-project.org/doc/manuals/R-exts.html

Not So Standard Deviations

A statistics (etc.) blog by Hilary Parker

Search

About Me

Contact

Posted on **April 29, 2014**

← Previous Next →

Writing an R package from scratch

As I have worked on various projects at Etsy, I have accumulated a suite of functions that help me quickly produce tables and charts that I find useful. Because of the nature of iterative development, it often happens that I reuse the functions many times, mostly through the shameful method of copying the functions into the project directory. I have been a fan of the idea of personal R packages for a while, but it always seemed like A Project That I Should Do Someday and someday never came. Until…

Etsy has an amazing week called "hack week" where we all get the opportunity to work on fun projects instead of our regular jobs. I sat down yesterday as part of Etsy's hack week and decided "I am finally going to make that package I keep saying I am going to make." It took me such little time that I was hit with that familiar feeling of the joy of optimization combined with the regret of past inefficiencies (joygret?). I wish I could go back in time and create the package the first moment I thought about it, and then use all the saved time to watch cat videos because that really would have been more productive.

This tutorial is not about making a beautiful, perfect R package. This tutorial is about creating a bare-minimum R package so that you don't have to keep thinking to yourself, "I really should just make an R package with these functions so I don't have to keep copy/pasting them like a goddamn luddite." Seriously, it doesn't have to be about sharing your code (although that is an added benefit!). It is about saving yourself time. (n.b. this is my attitude about all reproducibility.)

(For more details, I recommend this chapter in Hadley Wickham's *Advanced R Programming* book.)

Step 0: Packages you will need

The packages you will need to create a package are `devtools` and `roxygen2`. I am having you download the development version of the `roxygen2` package.

```
install.packages("devtools")
library("devtools")
devtools::install_github("klutometis/roxygen")
```

R Packages

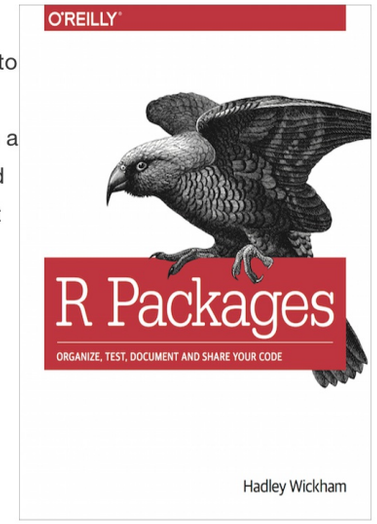
Hadley Wickham

Jenny Bryan

R packages

Packages are the fundamental units of reproducible R code. They include reusable R functions, the documentation that describes how to use them, and sample data. In this book you'll learn how to turn your code into packages that others can easily download and use. Writing a package can seem overwhelming at first. So start with the basics and improve it over time. It doesn't matter if your first version isn't perfect as long as the next version is better.

This edition is a work in progress. If you're looking for the electronic version of the 1st edition, you can find it online at http://r-pkgs.had.co.nz/.



1.2.4 Using C++11 code

R can be built without a C++ compiler although one is available (but not necessarily installed) on all known R platforms. For full portability across platforms, all that can be assumed is approximate support for the C++98 standard (the widely used `g++` deviates considerably from the standard). Some compilers have a concept of 'C++03' ('essentially a bug fix') or 'C++ Technical Report 1' (TR1), an optional addition to the 'C++03' revision which was published in 2007. A revised standard was published in 2011 and compilers with pretty much complete implementations are available. C++11 added all of the C99 features which are not otherwise implemented in C++, and C++ compilers commonly accept C99 extensions to C++98. A minor update[38] to C++11 (C++14) was published in December 2014. The latest standard (C++17) was published in December 2017, and a further revision ('C++20') is in preparation.

What standard a C++ compiler aims to support can be hard to determine: the value[39] of `__cplusplus` may help but some compilers use it to denote a standard which is partially supported and some the latest standard which is (almost) fully supported. As from version 6, `g++` defaults to C++14 (with GNU extensions): earlier versions aim to support C++03 with many extensions (including support for TR1) with version 5 having fairly complete C++14 support enabled by flag `-std=gnu++14`. `clang` with its native[40] `libc++` headers and library has since version 3.4 included almost all C++14 features, but does not support TR1. As from version 6.0.0, `clang` defaults to C++14.

Since version 3.1.0, R has provided support for C++11 in packages in addition to C++98. This support is not uniform across platforms as it depends on the capabilities of the compiler (see below). When R is configured, it will determine whether the C++ compiler supports C++11 and which compiler flags, if any, are required to enable C++11 support. For example, recent versions of `g++` or `clang++` accept the compiler flag `-std=c++11`, and earlier versions support a flag `-std=c++0x`, but the latter only provided partial support for the C++11 standard (it later became a deprecated synonym for `-std=c++11`).

In order to use C++11 code in a package, the package's `Makevars` file (or `Makevars.win` on Windows) should include the line

```
CXX_STD = CXX11
```

Compilation and linking will then be done with the C++11 compiler.

Packages without a `src/Makevars` or `src/Makefile` file may specify that they require C++11 for code in the `src` directory by including '`C++11`' in the '`SystemRequirements`' field of the `DESCRIPTION` file, e.g.

```
SystemRequirements: C++11
```

If a package does have a `src/Makevars[.win]` file then setting the make variable '`CXX_STD`' is preferred, as it allows `R CMD SHLIB` to work correctly in the package's `src` directory.

Conversely, to ensure that the C++98 standard is assumed even when this is not the compiler default, use

```
SystemRequirements: C++98
```

or

```
CXX_STD = CXX98
```

The C++11 compiler will be used systematically by R for all C++ code if the environment variable `USE_CXX11` is defined (with any value). Hence this environment variable should be defined when invoking `R CMD SHLIB` in the absence of a `Makevars` file (or `Makevars.win` on Windows) if a C++11 compiler is required.

Further control over compilation of C++11 code can be obtained by specifying the macros '`CXX11`' and '`CXX11STD`' when R is configured[41], or in a personal or site `Makevars` file. See Customizing package compilation in *R Installation and Administration*. If C++11 support is not available then these macros are both empty; if it is available by default, '`CXX11`' defaults to '`CXX`' and '`CXX11STD`' is empty . Otherwise, '`CXX11`' defaults to the same value as the C++ compiler '`CXX`' and the flag '`CXX11STD`' defaults to `-std=c++11` or similar. It is possible to specify '`CXX11`' to be a distinct compiler just for C++11–using packages, e.g. `g++` on Solaris. Note however that different C++ compilers (and even different versions of the same compiler) often differ in their ABI so their outputs can rarely be mixed. By setting '`CXX11STD`' it is also possible to choose a different dialect of the standard such as `-std=c++11`.

As noted above, support for C++11 varies across platforms: on some platforms, it may be possible or necessary to select a different compiler for C++11, *via* personal or site `Makevars` files.

There is no guarantee that C++11 can be used in a package in combination with any other compiled language (even C), as the C++11 compiler may be incompatible with the native compilers for the platform. (There are known problems mixing C++11 with Fortran.)

Getting started

You will need the following packages

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

Developmental version of **devtools** and **usethis**.

```
devtools::install_github("r-lib/devtools")
devtools::install_github("r-lib/usethis")
```

On Windows, download and install Rtools
On Mac, download and install XCode
On Linux, download and install the R development tools

Creating a minimal R package - Name

Must be Googleable and avaliable.
https://github.com/ropenscilabs/available

Can be

- Informative name, (goodpractice, microbenchmark)
- A play on words, (dplyr, strapgod, wibble)
- An Abriviation, (ergm, mcmc)
- Add an extra R, (lintr, purrr)
- No connection (shiny)

Create a minimal R package

```
> library(usethis)
> create_package("~/path/to/my/package")
✔ Setting active project to '/Users/emilhvitfeldthansen/path/to/my/package'
✔ Creating 'R/'
✔ Creating 'man/'
✔ Writing 'DESCRIPTION'
✔ Writing 'NAMESPACE'
✔ Writing 'package.Rproj'
✔ Adding '.Rproj.user' to '.gitignore'
✔ Adding '^package\\.Rproj$', '^\\.Rproj\\.user$' to '.Rbuildignore'
✔ Opening new project 'package' in RStudio
✔ Setting active project to '/Users/emilhvitfeldthansen/Desktop/mini'
>
```


..
.gitignore
.Rbuildignore
.Rproj.user
DESCRIPTION
man
NAMESPACE
package.Rproj
R

We are done!

DESCRIPTION file

```
Package: package
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person(given = "Emil",
           family = "Hvitfeldt",
           role = c("aut", "cre"),
           email = "emilhhvitfeldt@gmail.com")
Description: What the package does (one paragraph).
License: MIT + file LICENSE
Encoding: UTF-8
LazyData: true
Imports:
    dplyr
Suggests:
    ggplot2
```

R/ - adding functions

Create `fibonacci.R` file with `use_r("fibonacci")`

```
fibonacci <- function(n) {
  if(n == 0) return(0)
  if(n == 1) return(1)

  fibonacci(n - 1) + fibonacci(n - 2)
}
```

R/ - adding documentations

```r
#' Calculates the fibonacci numbers
#'
#' This function takes in a number and returns the corresponding
#' fibonacci number.
#'
#' @param n A number
#'
#' @return A number
#' @export
#'
#' @examples
#' fibonacci(10)
fibonacci <- function(n) {
  if(n == 0) return(0)
  if(n == 1) return(1)

  fibonacci(n - 1) + fibonacci(n - 2)
}
```

```r
devtools::document()
```

```
==> devtools::document(roclets = c('rd', 'collate', 'namespace'))

Updating package documentation
Updating roxygen version in /Users/emilhvitfeldthansen/path/to/my/package/DESCRIPTION
Writing NAMESPACE
Loading package
Writing fibonacci.Rd
Writing NAMESPACE
Documentation completed
```

Automatically populate the `man/` folder

R/ - adding documentations

fibonacci {package} R Documentation

Calculates the fibonacci numbers

Description

This function takes in a number and returns the corresponding fibonacci number.

Usage

```
fibonacci(n)
```

Arguments

n	A number

Value

A number

Examples

```
fibonacci(10)
```

Namespace

⚠ This document is read only.

```
# Generated by roxygen2: do not edit by hand

export(fibonacci)

```

Dependencies

```
Package: package
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person(given = "Emil",
           family = "Hvitfeldt",
           role = c("aut", "cre"),
           email = "emilhhvitfeldt@gmail.com")
Description: What the package does (one paragraph).
License: MIT + file LICENSE
Encoding: UTF-8
LazyData: true
Imports:
    dplyr
Suggests:
    ggplot2
```

Dependencies

- `Imports` are packages that your package to work.
- `Suggests` are packages that your package can use but doesn't need.
- `LinkingTo` packages listed here rely on C or C++ code in another package.

How to distribute

- Self-use

```
devtools::install() # Installs locally
devtools::build()   # Creates package.tar.gz file
```

- Github + git

```
usethis::use_github()
```

- CRAN

```
devtools::release()
```

Additional components

- Package name
- README
- Documentation
- Documentation website
- Authorship
- License
- Data
- Tests
- Vignettes
- Continuous integration
- News
- codecov
- Dependencies
- Examples

Checking

`R CMD check`

Run a bunch of sanity checks.

run

```
devtools::check()
```

or Ctrl/Cmd + Shift + E in RStudio

Readme

README.md

yardstick

build passing | coverage 96% | CRAN 0.0.3 | downloads 4284/month | lifecycle maturing

Overview

`yardstick` is a package to estimate how well models are working using tidy data principles. See the package webpage for more information.

Installation

To install the package:

```
install.packages("yardstick")

# Development version:
devtools::install_github("tidymodels/yardstick")
```

Two class metric

For example, suppose you create a classification model and predict on a new data set. You might have data that looks like this:

```
library(yardstick)
library(dplyr)

head(two_class_example)
#>    truth  Class1   Class2 predicted
#> 1 Class2 0.00359 0.996411    Class2
#> 2 Class1 0.67862 0.321379    Class1
#> 3 Class2 0.11089 0.889106    Class2
```

Readme

Will generally be first point of contact for developers.

Contains:

- The package name
- Badges for continuous integration and test coverage
- Short description of goals of package
- Installation instructions
- Brief demonstration usage
- Citation information

Readme - creation

Simply type

```
use_readme_rmd()
```

Workflow

- Modify
- Knit (crtl/cmd + k)
- repeat

Licenses

Officially Authorized

- The “GNU Affero General Public License” version 3
- The “Artistic License” version 2.0
- The “BSD 2-clause License”
- The “BSD 3-clause License”
- The “GNU General Public License” version 2
- The “GNU General Public License” version 3
- The “GNU Library General Public License” version 2
- The “GNU Lesser General Public License” version 2.1
- The “GNU Lesser General Public License” version 3
- The “MIT License”
- The “Creative Commons Attribution-ShareAlike International License” version 4.0

Licenses

(I'm not a lawyer!)

- MIT (simple and permissive)
- GPL-2 or GPL-3 (anti-capitalism)
- CC0 (freely be used for any purpose)

```
use_mit_license("My Name")
use_gpl3_license("My Name")
use_cc0_license("Ny Name")
```

Vignettes

Long form documentation.

Perfect for longer examples that doesn't fit in examples.

```
use_vignette("Awesome vignette")
```

News

More granular details regarding the changes to the package

```
use_news_md()
```

parsnip 0.0.2

Small release driven by changes in `sample()` in the current r-devel.

New Features

- A "null model" is now available that fits a predictor-free model (using the mean of the outcome for regression or the mode for classification).
- `fit_xy()` can take a single column data frame or matrix for `y` without error

Other Changes

- `varying_args()` now has a `full` argument to control whether the full set of possible varying arguments is returned (as opposed to only the arguments that are actually varying).
- `fit_control()` not returns an S3 method.
- For classification models, an error occurs if the outcome data are not encoded as factors (#115).
- The prediction modules (e.g. `predict_class`, `predict_numeric`, etc) were de-exported. These were internal functions that were not to be used by the users and the users were using them.
- An event time data set (`check_times`) was included that is the time (in seconds) to run `R CMD check` using the "r-devel-windows-ix86+x86_64` flavor. Packages that errored are censored.

Bug Fixes

- `varying_args()` now uses the version from the `generics` package. This means that the first argument, `x`, has been renamed to `object` to align with generics.
- For the recipes step method of `varying_args()`, there is now error checking to catch if a user tries to specify an argument that *cannot* be varying as varying (for example, the `id`) (#132).

Data

We want to include the data generation process.

```
use_data_raw()
```

```
## Next:

## • Add data creation scripts in 'data-raw/'
## • Use `usethis::use_data()` to add data to package
```

In `data-raw`

```
fib5 <- c(1, 1, 2, 3, 5, 8)
use_data(fib5)
```

Data - documentation

Create data R file

```
use_r("data")
```

Document as normal. Do not @export.

```
#' First 5 fibonacci numbers
#'
#' A vector with the first 5 fibonacci numbers.
#'
#' @format A vector of length 5
"fib5"
```

Compiled Code (c++)

Start by setting up package to work with compiled code

```
use_rcpp()
```

Create file: File -> New File -> C++ file

- Write code
- Document with Ctrl/Cmd + Shift + D
- Build & Reload Ctrl/Cmd + Shift + B
- repeat

Testing

Use a testing framework, we will use the testthat package

```
use_testthat()
```

add a test

```
use_test("fibonacci")
```

Testing

```r
context("test-fibonacci")


test_that("fibonacci work as intended", {
  expect_equal(fibonacci(6), 8)
})


test_that("fibonacci complains when n is not numeric scalar", {
  expect_error(fibonacci("six"))
  expect_warning(fibonacci(c(1, 2, 3, 4)))
})
```

Using Continuous Integration (CI)

> The idea behind continuous integration is that CI will automatically run R CMD check (along with your tests, etc.) every time you push a commit to GitHub.

```
use_travis()
```

Code coverage

```
use_coverage()
```

Badges

ggstatsplot: `ggplot2` Based Plots with Statistical Details

Package	Status	Usage	GitHub	References
CRAN 0.0.10 – 10 days ago	build passing	downloads 66/day	GitHub 0.0.10.9000	website ggstatsplot
CRAN Not OK	build passing	downloads 496/week	Github Pending PRs	rdocumentation 0.0.9
R>= 3.5.0	lifecycle maturing	downloads 1867/month	Github Issues	vignettes 0.0.10
code size 822 kB	coverage 100%	downloads 14K	Github 418	DOI 10.5281/zenodo.2074621
licence GPL-3	codecov 99%	hits 16281	last change 2019-03-24	last commit last monday
dependencies 29/188	covrpage Last Build 2019 01 31	Say Thanks !	repo status Active	contributions welcome
chat on gitter	dependencies ok		forks 49	

Live demo