O'REILLY®

R Packages

ORGANIZE, TEST, DOCUMENT, AND SHARE YOUR CODE

Hadley Wickham

O'REILLY®

R Packages

Turn your R code into packages that others can easily download and use. This practical book shows you how to bundle reusable R functions, sample data, and documentation together by applying author Hadley Wickham's package development philosophy. In the process, you'll work with devtools, roxygen, and testthat, a set of R packages that automates common development tasks. Devtools encapsulates best practices that Hadley has learned from years of working with this programming language.

Ideal for developers, data scientists, and programmers with various backgrounds, this book starts with the basics and shows you how to improve your package writing over time. You'll learn to focus on what you want your package to do, rather than think about package structure.

- Learn about the most useful components of an R package, including vignettes and unit tests
- Take advantage of devtools to automate anything you can
- Get tips on good style, such as organizing functions into files
- Streamline your development process with devtools
- Discover the best way to submit your package to the Comprehensive R Archive Network (CRAN)
- Learn from a well-respected member of the R community who created 30 R packages, including ggplot2, dplyr, and tidyr

Hadley Wickham is Chief Scientist at RStudio. He's a well-respected member of the R community who has written and contributed to over 30 R packages. Hadley won the John Chambers Award for Statistical Computing for his work developing tools for data reshaping and visualization.

"This book is a practical, hands-on guide for building high-quality software in R. Any R programmer looking to 'reach the next level' would do well to give this a read."

—**Wes McKinney**
creator of pandas

DATA/DATA SCIENCE

US $39.99 **CAN $45.99**

ISBN: 978-1-491-91059-7

Twitter: @oreillymedia
facebook.com/oreilly

R Packages

Hadley Wickham

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

R Packages
by Hadley Wickham

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Ann Spencer and Marie Beaugureau
Production Editor: Kara Ebrahim
Copyeditor: Jasmine Kwityn
Proofreader: Kim Cofer
Indexer: Wendy Catalano
Interior Designer: David Futato
Cover Designer: Ellie Volckhausen
Illustrator: Rebecca Demarest

April 2015: First Edition

Revision History for the First Edition
2015-03-20: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491910597* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *R Packages,* the cover image of a kaka, and related trade dress are trademarks of O'Reilly Media, Inc.

978-1-491-91059-7

[LSI]

Table of Contents

Part II. Package Components

Preface

In This Book

This book will guide you from being a user of R packages to being a creator of R packages. In Chapter 1, Introduction, you'll learn why mastering this skill is so important, and why it's easier than you think. Next, you'll learn about the basic structure of a package, and the forms it can take, in Chapter 2, Package Structure. The subsequent chapters go into more detail about each component. They're roughly organized in order of importance:

Chapter 3, R code
: The most important directory is *R/*, where your R code lives. A package with just this directory is still a useful package. (And indeed, if you stop reading the book after this chapter, you'll have still learned some useful new skills.)

Chapter 4, Package Metadata
: The *DESCRIPTION* lets you describe what your package needs to work. If you're sharing your package, you'll also use the *DESCRIPTION* to describe what it does, who can use it (the license), and who to contact if things go wrong.

Chapter 5, Object Documentation
: If you want other people (including "future you"!) to understand how to use the functions in your package, you'll need to document them. I'll show you how to use roxygen2 to document your functions. I recommend roxygen2 because it lets you write code and documentation together while continuing to produce R's standard documentation format.

Chapter 6, Vignettes: Long-Form Documentation
: Function documentation describes the nitpicky details of every function in your package. Vignettes give the big picture. They're long-form documents that show how to combine multiple parts of your package to solve real problems. I'll show

you how to use Rmarkdown and knitr to create vignettes with a minimum of fuss.

Chapter 7, Testing
: To ensure your package works as designed (and continues to work as you make changes), it's essential to write unit tests that define correct behavior, and alert you when functions break. In this chapter, I'll teach you how to use the testthat package to convert the informal interactive tests that you're already doing to formal, automated tests.

Chapter 8, Namespace
: To play nicely with others, your package needs to define what functions it makes available to other packages and what functions it requires from other packages. This is the job of the *NAMESPACE* file and I'll show you how to use roxygen2 to generate it for you. *NAMESPACE* is one of the more challenging parts of developing an R package, but it's critical to master if you want your package to work reliably.

Chapter 9, External Data
: The *data/* directory allows you to include data with your package. You might do this to bundle data in a way that's easy for R users to access, or just to provide compelling examples in your documentation.

Chapter 10, Compiled Code
: R code is designed for human efficiency, not computer efficiency, so it's useful to have a tool in your back pocket that allows you to write fast code. The *src/* directory allows you to include speedy compiled C and C++ code to solve performance bottlenecks in your package.

Chapter 11, Installed Files
: You can include arbitrary extra files in the *inst/* directory. This is most commonly used for extra information about how to cite your package, and to provide more details about copyrights and licenses.

Chapter 12, Other Components
: This chapter documents the handful of other components that are rarely needed: *demo/*, *exec/*, *po/*, and *tools/*.

The final three chapters describe general best practices not specifically tied to one directory:

Chapter 13, Git and GitHub
: Mastering a version control system is vital for collaborating with others, and is useful even for solo work because it allows you to easily undo mistakes. In this chapter, you'll learn how to use the popular Git and GitHub combo with RStudio.

Chapter 14, Automated Checking
: R provides useful automated quality checks in the form of `R CMD check`. Running them regularly is a great way to avoid many common mistakes. The results can sometimes be a bit cryptic, so I provide a comprehensive cheat sheet to help you convert warnings to actionable insight.

Chapter 15, Releasing a Package
: The life cycle of a package culminates with release to the public. This chapter compares the two main options (CRAN and GitHub) and offers general advice on managing the process.

This is a lot to learn, but don't feel overwhelmed. Start with a minimal subset of useful features (e.g., just an *R/* directory!) and build up over time. To paraphrase the Zen monk Shunryū Suzuki: "Each package is perfect the way it is—and it can use a little improvement."

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *http://r-pkgs.had.co.nz/*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*R Packages* by Hadley Wickham (O'Reilly). Copyright 2015 Hadley Wickham, 978-1-491-91059-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders,

McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/r-packages*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

The tools in this book wouldn't be possible without many open source contributors. Winston Chang, my coauthor on devtools, spent hours debugging painful S4 and compiled code problems so that devtools can quickly reload code for the vast majority of packages. Kirill Müller contributed great patches to many of my package development packages including devtools, testthat, and roxygen2. Kevin Ushey, JJ Allaire, and Dirk Eddelbuettel tirelessly answered all my basic C, C++, and Rcpp questions. Peter Danenburg and Manuel Eugster wrote the first version of roxygen2 during a Google Summer of Code. Craig Citro wrote much of the code to allow travis to work with R packages.

Often the only way I learn how to do it the right way is by doing it the wrong way first. For suffering many package development errors, I'd like to thank all the CRAN maintainers, especially Brian Ripley, Uwe Ligges, and Kurt Hornik.

This book was written in the open and it is truly a community effort: many people read drafts, fixed typos, suggested improvements, and contributed content. Without those contributors, the book wouldn't be nearly as good as it is, and I'm deeply grateful for their help. A special thanks goes to Peter Li, who read the book from cover to cover and provided many fixes. I also deeply appreciate the time the reviewers (Duncan Murdoch, Karthik Ram, Vitalie Spinu, and Ramnath Vaidyanathan) spent reading the book and giving me thorough feedback.

Thanks go to all contributors who submitted improvements via GitHub (in alphabetical order): @aaronwolen, @adessy, Adrien Todeschini, Andrea Cantieni, Andy Visser, @apomatix, Ben Bond-Lamberty, Ben Marwick, Brett K, Brett Klamer, @contravariant, Craig Citro, David Robinson, David Smith, @davidkane9, Dean Attali, Eduardo Ariño de la Rubia, Federico Marini, Gerhard Nachtmann, Gerrit-Jan Schutten, Hadley Wickham, Henrik Bengtsson, @heogden, Ian Gow, @jacobbien, Jennifer (Jenny) Bryan, Jim Hester, @jmarshallnz, Jo-Anne Tan, Joanna Zhao, Joe Cainey, John Blischak, @jowalski, Justin Alford, Karl Broman, Karthik Ram, Kevin Ushey, Kun Ren, @kwenzig, @kylelundstedt, @lancelote, Lech Madeyski, @lindbrook, @maiermarco, Manuel Reif, Michael Buckley, @MikeLeonard, Nick Carchedi, Oliver Keyes, Patrick Kimes, Paul Blischak, Peter Meissner, @PeterDee, Po Su, R. Mark Sharp, Richard M. Smith, @rmar073, @rmsharp, Robert Krzyzanowski, @ryanatanner, Sascha Holzhauer, @scharne, Sean Wilkinson, @SimonPBiggs, Stefan Widgren, Stephen Frank, Stephen Rushe, Tony Breyal, Tony Fischetti, @urmils, Vlad Petyuk, Winston Chang, @winterschlaefer, @wrathematics, and @zhaoy.

PART I

Getting Started

CHAPTER 1
Introduction

In R, the fundamental unit of shareable code is the package. A package bundles together code, data, documentation, and tests, and is easy to share with others. As of January 2015, there were over 6,000 packages available on the *C*omprehensive *R* *A*rchive *N*etwork, or *CRAN*, the public clearing house for R packages. This huge variety of packages is one of the reasons that R is so successful: chances are that someone has already solved a problem that you're working on, and you can benefit from their work by downloading their package.

If you're reading this book, you already know how to use packages:

- You install them from CRAN with `install.packages("x")`.
- You use them in R with `library(x)`.
- You get help on them with `package?x` and `help(package = "x")`.

The goal of this book is to teach you how to develop packages so that you can write your own, not just use other people's. Why write a package? One compelling reason is that you have code that you want to share with others. Bundling your code into a package makes it easy for other people to use it, because like you, they already know how to use packages. If your code is in a package, any R user can easily download it, install it, and learn how to use it.

But packages are useful even if you never share your code. As Hilary Parker says in her introduction to packages: "Seriously, it doesn't have to be about sharing your code (although that is an added benefit!). It is about saving yourself time." Organizing code in a package makes your life easier because packages come with conventions. For example, you put R code in *R/*, you put tests in *tests/*, and you put data in *data/*. These conventions are helpful because:

They save you time
: Instead of having to think about the best way to organize a project, you can just follow a template.

Standardized conventions lead to standardized tools
: If you buy into R's package conventions, you get many tools for free.

It's even possible to use packages to structure your data analyses, as Robert M. Flight discusses in a series of blog posts.

Philosophy

This book espouses my philosophy of package development: anything that can be automated should be automated. Do as little as possible by hand. Do as much as possible with functions. The goal is to spend your time thinking about what you want your package to do rather than thinking about the minutiae of package structure.

This philosophy is realized primarily through the devtools package, a suite of R functions that I wrote to automate common development tasks. The goal of devtools is to make package development as painless as possible. It does this by encapsulating all of the best practices of package development that I've learned over the years. Devtools protects you from many potential mistakes, so you can focus on the problem you're interested in, not on developing a package.

Devtools works hand in hand with RStudio, which I believe is the best development environment for most R users. The only real competitor is Emacs Speaks Statistics (ESS), which is a rewarding environment if you're willing to put in the time to learn Emacs and customize it to your needs. The history of ESS stretches back over 20 years (predating R!), but it's still actively developed and many of the workflows described in this book are also available there.

Together, devtools and RStudio insulate you from the low-level details of how packages are built. As you start to develop more packages, I highly recommend that you learn more about those details. The best resource for the official details of package development is always the official writing R extensions manual. However, this manual can be hard to understand if you're not already familiar with the basics of packages. It's also exhaustive, covering every possible package component, rather than focusing on the most common and useful components, as this book does. Writing R extensions is a useful resource once you've mastered the basics and want to learn what's going on under the hood.

Getting Started

To get started, make sure you have the latest version of R (at least 3.1.2, which is the version that the code in this book uses), then run the following code to get the packages you'll need:

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

Make sure you have a recent version of RStudio. You can check that you have the right version by running the following:

```
install.packages("rstudioapi")
rstudioapi::isAvailable("0.99.149")
```

If not, you may need to install the preview version. This gives you access to the latest and greatest features, and only slightly increases your chances of finding a bug.

If you want to keep up with the bleeding edge of devtools development, you can use the following code to access new functions as I develop them:

```
devtools::install_github("hadley/devtools")
```

You'll need a C compiler and a few command-line tools. If you're on Windows or Mac and you don't already have them, RStudio will install them for you. Otherwise:

- On Windows, download and install Rtools. *Nnote: this is not an R package!*
- On Mac, make sure you have either XCode (available for free in the App Store) or the "Command-Line Tools for Xcode". You'll need to have a (free) Apple ID.
- On Linux, make sure you've installed not only R, but also the R development tools. For example, on Ubuntu (and Debian) you need to install the Ubuntu `r-base-dev` package.

You can check that you have everything installed by running the following code:

```
library(devtools)
has_devel()
#> '/Library/Frameworks/R.framework/Resources/bin/R' --vanilla CMD SHLIB foo.c
#>
#> clang -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG
#>   -I/usr/local/include -I/usr/local/include/freetype2 -I/opt/X11/include
#>   -fPIC  -Wall -mtune=core2 -g -O2  -c foo.c -o foo.o
#> clang -dynamiclib -Wl,-headerpad_max_install_names -undefined dynamic_lookup
#>   -single_module -multiply_defined suppress -L/usr/local/lib -o foo.so foo.o
#>   -F/Library/Frameworks/R.framework/.. -framework R -Wl,-framework
#>   -Wl,CoreFoundation
[1] TRUE
```

This will print out some code that I use to help diagnose problems. If everything is OK, it will return `TRUE`. Otherwise, it will throw an error and you'll need to investigate the problem.

Conventions

Throughout this book I write `foo()` to refer to functions, `bar` to refer to variables and function parameters, and *baz/* to refer to paths. Larger code blocks intermingle input and output. Output is commented so that if you have an electronic version of the book (e.g., *http://r-pkgs.had.co.nz*), you can easily copy and paste examples into R. Output comments look like `#>` to distinguish them from regular comments.

Colophon

This book was written in Rmarkdown inside RStudio. knitr and pandoc converted the raw Rmarkdown to HTML and PDF. The website was made with jekyll, styled with bootstrap, and published to Amazon's S3 by travis-ci. The complete source is available from GitHub. This version of the book was built with:

```
library(roxygen2)
library(testthat)
devtools::session_info()
#> Session info -------------------------------------------------------------
#>  setting  value
#>  version  R version 3.1.2 (2014-10-31)
#>  system   x86_64, linux-gnu
#>  ui       X11
#>  language (EN)
#>  collate  en_US.UTF-8
#>  tz       <NA>
#> Packages -----------------------------------------------------------------
#>  package    * version    date       source
#>  bookdown     0.1        2015-02-12 Github (hadley/bookdown@fde0b07)
#>  devtools   * 1.7.0.9000 2015-02-12 Github (hadley/devtools@9415a8a)
#>  digest     * 0.6.8      2014-12-31 CRAN (R 3.1.2)
#>  evaluate   * 0.5.5      2014-04-29 CRAN (R 3.1.0)
#>  formatR    * 1.0        2014-08-25 CRAN (R 3.1.1)
#>  htmltools  * 0.2.6      2014-09-08 CRAN (R 3.1.2)
#>  knitr      * 1.9        2015-01-20 CRAN (R 3.1.2)
#>  Rcpp       * 0.11.4     2015-01-24 CRAN (R 3.1.2)
#>  rmarkdown    0.5.1      2015-02-12 Github (rstudio/rmarkdown@0f19584)
#>  roxygen2     4.1.0      2014-12-13 CRAN (R 3.1.2)
#>  rstudioapi * 0.2        2014-12-31 CRAN (R 3.1.2)
#>  stringr    * 0.6.2      2012-12-06 CRAN (R 3.0.0)
#>  testthat     0.9.1      2014-10-01 CRAN (R 3.1.1)
```

CHAPTER 2

Package Structure

This chapter will start you on the road to package development by showing you how to create your first package. You'll also learn about the various states a package can be in, including what happens when you install a package. Finally, you'll learn about the difference between a package and a library and why you should care.

Naming Your Package

> "There are only two hard things in computer science: cache invalidation and naming things."
>
> —Phil Karlton

Before you can create your first package, you need to come up with a name for it. I think this is the hardest part of creating a package! (Not least because devtools can't automate it for you.)

Requirements for a Name

There are three formal requirements: the name can only consist of letters, numbers, and periods (i.e., `.`); it must start with a letter; and it cannot end with a period. Unfortunately, this means you can't use either hyphens or underscores (i.e., `-` or `_`) in your package name. I recommend against using periods in package names because it has confusing connotations (i.e., file extension or S3 method).

Strategies for Creating a Name

If you're planning on releasing your package, I think it's worth spending a few minutes to come up with a good name. Here are some recommendations for how to go about it:

- Choose a unique name that can easily be Googled. This makes it easy for potential users to find your package (and associated resources) and for you to see who's using it. You can also check if a name is already used on CRAN by loading *http://cran.r-project.org/web/packages/[PACKAGE_NAME]*.
- Avoid using both upper- and lowercase letters: doing so makes the package name hard to type and even harder to remember. For example, I can never remember if it's Rgtk2 or RGTK2 or RGtk2.
- Find a word that evokes the problem and modify it so that it's unique:
  - plyr is a generalization of the apply family, and evokes pliers.
  - lubridate makes dates and times easier.
  - knitr (knit + r) is "neater" than sweave (s + weave).
  - testdat tests that data has the correct format.
- Use abbreviations:
  - Rcpp = R + C++ (plus plus).
  - lvplot = letter value plots.
- Add an extra R:
  - stringr provides string tools.
  - tourr implements grand tours (a visualization method).
  - gistr lets you programmatically create and modify GitHub gists.

If you're creating a package that talks to a commercial service, make sure you check the branding guidelines to avoid problems down the line. For example, rDrop isn't called rDropbox because Dropbox prohibits any applications from using the full trademarked name.

Creating a Package

Once you've decided on a name, there are two ways to create the package. You can use RStudio:

1. Click File → New Project.
2. Choose New Directory, as shown in Figure 2-1.