

Package Development

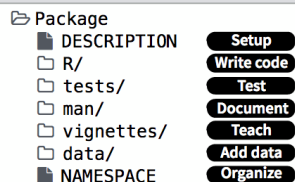
with devtools Cheat Sheet



Package Structure

A package is a convention for organizing files into directories.

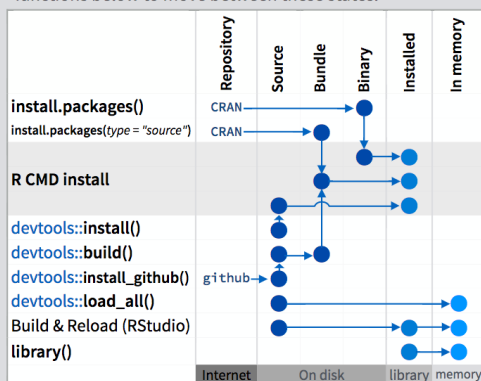
This sheet shows how to work with the 7 most common parts of an R package:



The contents of a package can be stored on disk as a:

- source** - a directory with sub-directories (as above)
- bundle** - a single compressed file (.tar.gz)
- binary** - a single compressed file optimized for a specific OS

Or installed into an R library (loaded into memory during an R session) or archived online in a repository. Use the functions below to move between these states.



devtools::add_build_ignore("file")

Adds file to .Rbuildignore, a list of files that will not be included when package is built.

RStudio® is a trademark of RStudio, Inc. • RStudio • 844-448-1212 •

Setup (DESCRIPTION)

The DESCRIPTION file describes your work and sets up how your package will work with other packages.

- ☒ You must have a DESCRIPTION file
- ☒ Add the packages that yours relies on with **devtools::use_package()**
Adds a package to the Imports field (or Suggests field (if second argument is "Suggests")).

CC0	MIT	GPL-2
No strings attached.	MIT license applies to your code if re-shared.	GPL-2 license applies to your code, and all code anyone bundles with it, if re-shared.

```
Package: mypackage
Title: Title of Package
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
  "", role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.1.0)
License: GPL-2
LazyData: true
Imports:
  dplyr (>= 0.4.0),
  ggvis (>= 0.2)
Suggests:
  knitr (>= 0.1.0)
```

Import packages that your package must have to work. R will install them when it installs your package.

Suggest packages that are not very essential to yours. Users can install them manually, or not, as they like.

Write code (R/)

All of the R code in your package goes in R/. A package with just an R/ directory is still a very useful package.

- ☒ Create a new package project with **devtools::create("path/to/name")**
Create a template to develop into a package.
- ☒ Save your code in R/ as scripts (extension .R)

Workflow

1. Edit your code.
2. Load your code with one of **devtools::load_all()**
Re-loads all saved files in R/ into memory.
Ctrl/Cmd + Shift + L (keyboard shortcut)
Saves all open files then calls load_all().
3. Experiment in the console.
4. Repeat.

- Use consistent style with r-pkgs.had.co.nz/r.html#style
- Click on a function and press F2 to open its definition
- Search for a function with Ctrl + .

Visit <http://r-pkgs.had.co.nz> for more

Learn more at <http://r-pkgs.had.co.nz> • devtools 1.6.1 • Updated: 1/15
RStudio® is a trademark of RStudio, Inc. • All rights reserved
• 844-448-1212 •

Test (tests/)

Use tests/ to store unit tests that will inform you if your code ever breaks.

- ☒ Add a tests/ directory and import **testthat** with **devtools::use_testthat()**
Sets up package to use automated tests with testthat
- ☒ Write tests with **context()**, **test()**, and expectations
- ☒ Save your tests as .R files in tests/testthat/

Workflow

1. Modify your code or tests.
2. Test your code with one of **devtools::test()**
Runs all tests saved in tests/.
Ctrl/Cmd + Shift + T (keyboard shortcut)
3. Repeat until all tests pass

Example test

```
context("Arithmetic")

test_that("Math works", {
  expect_equal(1 + 1, 2)
  expect_equal(1 + 2, 3)
  expect_equal(1 + 3, 4)
})
```

expect_equal()	is equal within small numerical tolerance?
expect_identical()	is exactly equal?
expect_match()	matches specified string or regular expression?
expect_output()	prints specified output?
expect_message()	displays specified message?
expect_warning()	displays specified warning?
expect_error()	throws specified error?
expect_is()	output inherits from certain class?
expect_false()	returns FALSE?
expect_true()	returns TRUE?

Learn more at <http://r-pkgs.had.co.nz> • devtools 1.6.1 • Updated: 1/15

Document (man/)

man/ contains the documentation for your functions, the help pages in your package.

- ✓ Use roxygen comments to document each function beside its definition
- ✓ Document the name of each exported data set
- ✓ Include helpful examples for each function

Workflow

1. Add roxygen comments in your .R files
2. Convert roxygen comments into documentation with one of

devtools::document()

Converts roxygen comments to .Rd files and places them in man/. Builds NAMESPACE.

Ctrl/Cmd + Shift + D (Keyboard Shortcut)

3. Open help pages with ? to preview documentation
4. Repeat

.Rd formatting tags

\email{name@foo.com}	\href{uri}{display}
\emph{italic text}	\url{uri}
\strong{bold text}	
\code{function(args)}	\link[=dest]{display}
\pkg{package}	\linkS4class{class}
	\code{\link{function}}
\dontrun{code}	\code{\link[package]{function}}
\dontshow{code}	
\donttest{code}	\tabular{lr}{
	left \tab centered \tab right \cr
\deqn{a + b (block)}	cell \tab cell \tab cell \cr
\eqn{a + b (inline)}	}

The roxygen package

roxygen lets you write documentation inline in your .R files with a shorthand syntax.

- Add roxygen documentation as comment lines that begin with #.
- Place comment lines directly above the code that defines the object documented.
- Place a roxygen @ tag (right) after # to supply a specific section of documentation.
- Untagged lines will be used to generate a title, description, and details section (in that order)

```
#' Add together two numbers.
#
#' @param x A number.
#' @param y A number.
#' @return The sum of \code{x} and \code{y}.
#' @examples
#' add(1, 1)
#' @export
add <- function(x, y) {
  x + y
}
```

Common roxygen tags

@aliases	@inheritParams	@seealso
@concepts	@keywords	@format
@describeIn	@param	@source data
@examples	@rdname	@include
@export	@return	@slot S4
@family	@section	@field RC

Teach (vignettes/)

vignettes/ holds documents that teach your users how to solve real problems with your tools.

- ✓ Create a vignettes/ directory and a template vignette with devtools::use_vignette()
Adds template vignette as vignettes/my-vignette.Rmd.
- ✓ Append YAML headers to your vignettes (like right)
- ✓ Write the body of your vignettes in R Markdown

```
---
title: "Vignette Title"
author: "Vignette Author"
date: "r Sys.Date()"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{Vignette Title}
  %\VignetteEngine{knitr::rmarkdown}
  \usepackage[utf8]{inputenc}
---
```

Add data (data/)

The data/ directory allows you to include data with your package.

- ✓ Store data in one of data/, R/Sysdata.rda, inst/extdata
- ✓ Always use LazyData: true in your DESCRIPTION file.
- ✓ Save data as .Rdata files (suggested)

devtools::use_data()

Adds a data object to data/
(R/Sysdata.rda if internal = TRUE)

devtools::use_data_raw()

Adds an R Script used to clean a data set to data-raw/. Includes data-raw/ on .Rbuildignore.

Store data in

- data/ to make data available to package users
- R/sysdata.rda to keep data internal for use by your functions.
- inst/extdata to make raw data available for loading and parsing examples. Access this data with system.file()

Organize (NAMESPACE)

The NAMESPACE file helps you make your package self-contained: it won't interfere with other packages, and other packages won't interfere with it.

- ✓ Export functions for users by placing @export in their roxygen comments
- ✓ Import objects from other packages with package::object (recommended) or @import, @importFrom, @importClassesFrom, @importMethodsFrom (not always recommended)

Workflow

1. Modify your code or tests.
2. Document your package (devtools::document())
3. Check NAMESPACE
4. Repeat until NAMESPACE is correct

Submit your package