Package Development: : CHEAT SHEET

Package Structure

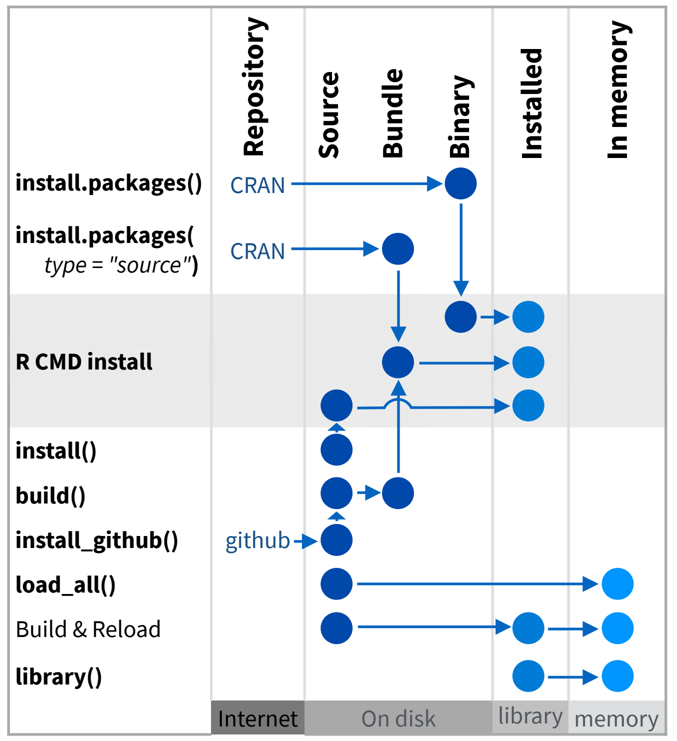
A package is a convention for organizing files into directories. This sheet shows how to work with the 7 most common parts of an R package:

Package	
DESCRIPTION	SETUP
R/	WRITE CODE
tests/	TEST
man/	DOCUMENT
vignettes/	TEACH
data/	ADD DATA
NAMESPACE	ORGANIZE

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.



Getting Started

The **devtools** package bundles together several packages, including **usethis**, which automates many steps of package development.

CREATE A PACKAGE

usethis::**create_package**(path) Create a new package at the specified location, creating a new directory if needed. Automatically create DESCRIPTION, NAMESPACE, and R/.

Create the remaining directories with:

devtools::**document()** Create man/

usethis::**use_testthat()** Create tests/

usethis::**use_vignettes**(name) Create vignettes/

usethis::**use_data(**...**)** Create data/

The **usethis** functions will automatically update DESCRIPTION and other package files as needed.

VERSION CONTROL

We strongly recommend Git/GitHub, for package development. Check out **happygitwithr.com**.

Basic Workflow

Once you've created your package, it's time to add your code! There are four main steps:

1. Write or edit code in the R/ directory. Use devtools::**load_all()** to make code available to test drive interactively.
2. Run checks with devtools::**check()**. Check often to catch errors early.
3. Add or edit documentation in your .R files. Run devtools::**document()** to update the documentation files in man/ and the NAMESPACE.
4. Add tests for your code to tests/. Run devtools::**test()** to run all tests.

Add longer instructional documents to vignettes/ or add data to your package in data/.

Setup (DESCRIPTION)

The DESCRIPTION file provides metadata about your package.

usethis::**use_*_license()** Select a license and add the associated files. More at **r-pkgs.org/license.html**

usethis::**use_package(**package, type = "Imports", min_version = NULL**)** Add a package to Imports or Suggests. Use package functions in your code with **pkg::fun(...)** e.g. dplyr::summarise(...).

```
Package: mypackage
Title: Title of Package
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
    "hadley@me.com", 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.

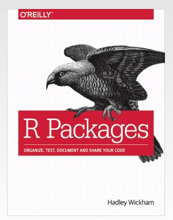
Write Code (R/)

All of the R code in your package goes in R/. Add .R files to R/ using usethis::**use_r(**name**)**.

WORKFLOW

1. Add .R files to R/ using usethis::**use_r(**name**)**.
2. Write or edit code.
3. Load code quickly with devtools::**load_all()** or **Ctrl/Cmd+Shift+L** to run code interactively.
4. Repeat.

For more on code style see the tidyverse style guide (**style.tidyverse.org**) or the **styler** package (**styler.r-lib.org**)



Visit **r-pkgs.org** to learn much more about writing and publishing packages for R

Test (**tests**/)

Import **testthat** and create a **tests/** directory with usethis::**use_testthat()**.

Example Test

```
# test-arithmetic.R
context("Arithmetic")

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

WORKFLOW

1. Create test files with usethis::**use_test(**name**)**.
2. Write tests using **context()** and **test_that()**.
3. Use devtools::**test()** or **Ctrl/Cmd+Shift+T** to run all tests.
4. Repeat.

Expect statement	Tests
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?

RStudio

Document (man/)

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

Use **roxygen comments** to document each function beside its definition. Also document exported data sets.

Use usethis::**use_pkgdown()** to create a website with **pkgdown** to build a website for your package. See **pkgdown.r-lib.org/**.

WORKFLOW

1. Add roxygen comments in your .R files. Generate a template in the RStudio IDE with **Code > Insert Roxygen Skeleton** or **Keyboard Shortcut?.**
2. Use devtools::**document()** or **Ctrl/Cmd+Shift+D** to create man/ if needed, convert roxygen comments to .Rd files and place them in man/, and automatically update NAMESPACE.
3. Open help pages with **?** to preview documentation.
4. Repeat.

.Rd FORMATTING TAGS

```
\emph{italic text}             \email{name@@foo.com}
\strong{bold text}             \href{url}{display}
\code{function(args)}          \url{url}
\pkg{package}
                               \link[=dest]{display}
\dontrun{code}                 \linkS4class{class}
\dontshow{code}                \code{\link{function}}
\donttest{code}                \code{\link[package]{function}}

\deqn{a + b (block)}           \tabular{lcr}{
\eqn{a + b (inline)}             left \tab centered \tab right \cr
                                 cell \tab cell      \tab cell  \cr
                               }
```

ROXYGEN2

The **roxygen2** package lets you write documentation inline in your .R files with a shorthand syntax. devtools implements roxygen2 to make documentation.

- 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}.
#' @export
#'
#' @examples
#' add(1, 1)
add <- function(x, y) {
  x + y
}
```

COMMON ROXYGEN TAGS

@aliases	@inheritParams	**@seealso**	
@concepts	@keywords	*@format*	used for data
@describeIn	**@param**	*@source*	
@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.

Use usethis::**use_vignette(**"my-vignette"**)** to create the vignettes/ directory and a template vignette, my-vignette.Rmd

Append YAML headers to your vignettes (like right)

Write the body of your vignettes in R Markdown (rmarkdown.rstudio.com)

```
---
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.

Save data as .Rdata files (suggested)

Always use **LazyData: true** in your DESCRIPTION file.

Store data in

- **data/** to make data available to package users. Use usethis::**use_data()** to create the directory and add data stored as an .rda file.
- **R/sysdata.rda** to keep data internal for use by your functions. Use usethis::**use_data(**internal = TRUE**)**.
- **inst/extdata** to make raw data available, for example for loading and parsing examples. Access this data with **system.file()**. Use usethis::**use_data_raw()** to add data to data-raw/ and include data-raw/ in .Rbuildignore.

Document data with roxygen in a separate .R file in R/.

```
#' Title
#' Description
#' @format
#' \describe{
#'   \item{name}{description}
#' }
#' @source \url{url}
"data_name"
```

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 with devtools::**document()**
3. Check NAMESPACE
4. Repeat until NAMESPACE is correct

RELEASE YOUR PACKAGE
See more at **r-pkgs.org/release.html**.