Package Development: : CHEAT SHEET

Package Structure

A package is a convention for organizing files into directories.

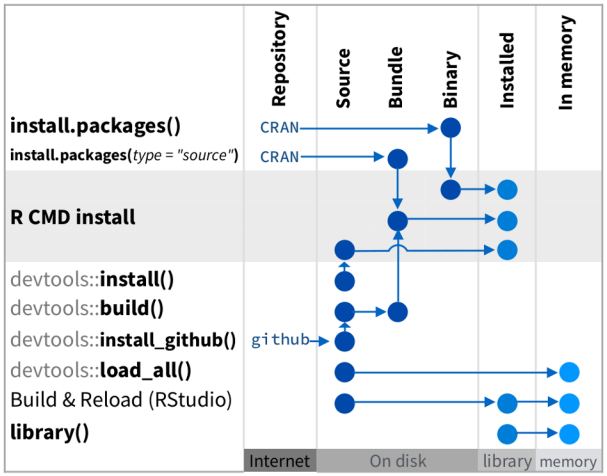
This sheet shows how to work with the 7 most common parts of an R package:

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.



devtools::**use_build_ignore**(*"file"*)

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

Setup (DESCRIPTION)

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

- ☑ You must have a DESCRIPTION file
- ☑ Add the packages that yours relies on with
  devtools::**use_package()**
  Adds a package to the Imports or Suggests field

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 =
    "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/. 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 **r-pkgs.had.co.nz** to learn much more about writing and publishing packages for R

Test (tests/)

Use tests/ to store tests that will alert you if your code breaks.

- ☑ Add a **tests/** directory
- ☑ Import **testthat** with devtools::**use_testthat()**, which sets up package to use automated tests with testthat
- ☑ Write tests with **context()**, **test()**, and expect statements
- ☑ 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 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 statement	Tests
expect_equal()	is equal within small numerical tolerance?
expect_identical()	is exactly equal?
expect_match()	matches specified string or regular
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