Mastering R Packaging: A Comprehensive Guide

A step-by-step guide to creating, documenting, testing, and sharing R packages using the *devtools* workflow.

National Workshop on R Package Development for Statistical Models in Biomedical Research organized by Division of Cancer Epidemiology & Biostatistics, Regional Cancer Centre, Thiruvananthapuram, Kerala
May 15 - May 17, 2025

by Rakesh Poduval

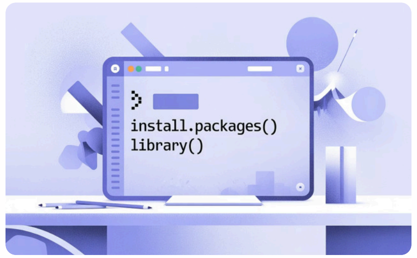
R Package: The fundamental unit of shareable code

Complete bundle

Bundles together code, data, documentation and tests and is easy to share with others.

CRAN ecosystem

22,430 packages available as of May 2025 on CRAN (Comprehensive R Archive Network) - a key reason for R's success.



Simple usage

Work with install.packages(x), library(x), help(package=x) to incorporate packages into your workflow.

Organized structure

Package makes your life easier because it comes with conventions, e.g., you put R code in R/, put tests in tests/, and data in data/.

The objective is to show you how to develop packages so that you can share your code for others to use.

cran.r-project.org

CRAN: Contributed Packages

Currently, the CRAN package repository features 22430 available packages.

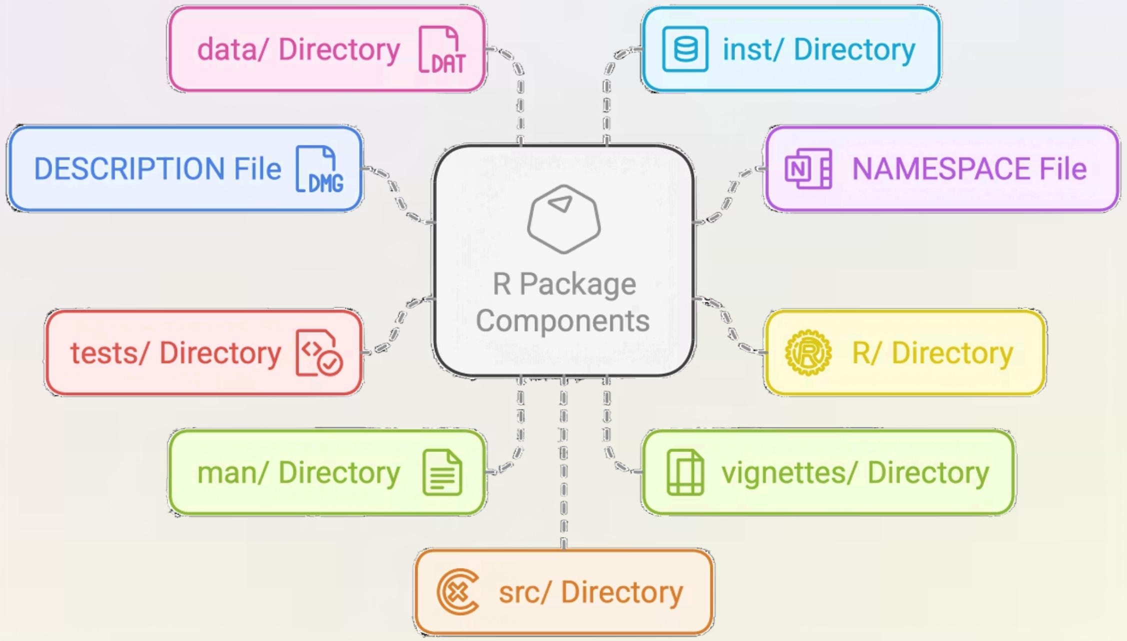
Popular R packages

TIBBLE
config
RIO
tidymodels
renv
plumber
kableExtra
tidyverse
Shiny
forecast
stringr
R6
reticulate
lubridate
devtools
covr
dplyr
data.table
test that
ggplot2
esquisse
golem

packagemanager.posit.co

Posit Package Manager

📦 Key components of R package



- packageDescription("dplyr")
- packageVersion("dplyr")
- packageDate("plyr")
- help(package = "dplyr")
- vignette(topic = "in-packages", package = "tidyr")
- data(package = "nycflights13")
- news(package = "plyr")
- ...

style.tidyverse.org

Tidyverse style guide

Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread...

📦 Key Components of an R Package

1 DESCRIPTION

Contains metadata about the package (title, version, dependencies, authors). This file is mandatory. See: desc.r-lib.org

2 NAMESPACE

Defines which functions are exposed to users. Typically auto-generated using roxygen2 with special tags (@export, etc.).

3 man/

Stores documentation files (*.Rd) for each function. Generated via roxygen2 comments in your R scripts.

4 data/

Holds external data sets (as .rda files). These are accessible with data() once the package is installed.

5 R/

Contains all R function definitions and internal objects. This is the core logic of your package.

6 tests/

Includes test code to validate functionality. Use frameworks like testthat to ensure your code works as intended.

7 inst/

For non-code files that should be installed with the package (e.g., raw data, docs). Files in inst/ appear in the top level of the installed package.

8 vignettes/

Provides long-form guides or workflows. Ideal for tutorials that demonstrate how to use the package start to finish.

The packaging workflow

1. **Project setup**

   Create a new package directory and initialize the DESCRIPTION file

2. **Code development**

   Write functions, scripts etc. for your package. `load_all()` makes your functions available in the session

3. **Run tests**

   `test()`

   Ensure that your package functions correctly and meets quality standards.

4. **Generate documentation**

   `document()` creates documentation using `roxygen2`. Provide concise descriptions and usage instructions for each function in your package.

5. **Check package**

   `check()` validates structure & code. It looks for all possible problems for submitting it to CRAN.

6. **Install**

   Use `build()` & `install()` to build & install the package locally

7. **Import & use**

   `library(myPkg)`

   `myPkg::myFunc(x)`

Advanced packaging techniques

Modular package structure

- Organize code into smaller, focused functions across multiple files in the R/ directory
- Group related functions in logical subfolders (e.g., R/data_utils/, R/plotting/) for better maintainability

Internal and External data

- Store **external data** in data/ directory (.rda files) with comprehensive `@format` documentation
- Manage **internal data** through data-raw/ directory and hide using `usethis::use_data(..., internal = TRUE)`

Object-oriented systems

- Choose **S3** for simplicity and flexibility, **S4** or **R6** for formal structure and encapsulation
- Implement generic methods (print, plot, summary) for more extendable and consistent APIs

C++ Integration via Rcpp

- Optimize performance-critical code sections with C++ through Rcpp integration
- Set up C++ integration seamlessly using `usethis::use_rcpp()` for proper configuration

Advanced roxygen2 documentation

- Control dependencies precisely with `@importFrom` to import only necessary functions
- Organize documentation using `@family` and `@concept` tags to group related functions

Namespace management

Define explicit exports and imports to control function visibility and create a clean, professional package interface

Package dependencies

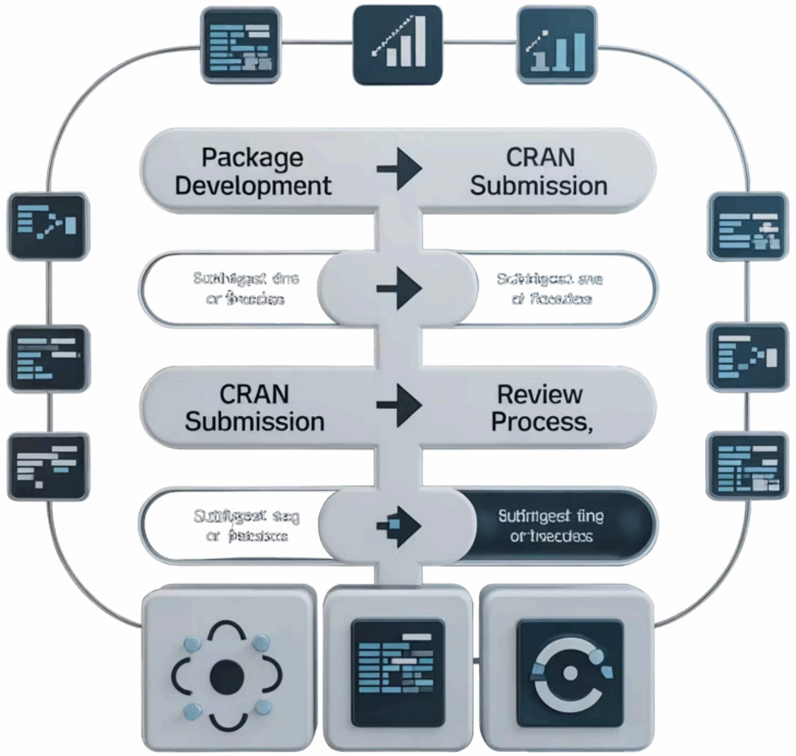
Carefully manage dependencies in DESCRIPTION file, distinguishing between Imports, Suggests, and Depends fields

Comprehensive testing

- Develop modular, mockable tests with `testthat` for robust quality assurance
- Implement tests for edge cases, error handling, and integration scenarios

Version control

Implement Git workflows to track changes, manage collaborations, and integrate with continuous integration platforms



Submitting to CRAN

1

CRAN Policies

Review CRAN's submission guidelines and policies to ensure compliance.

2

Package Preparation

Organize your package files, documentation, and tests.

3

Submission Process

Submit your package to CRAN using the 'R CMD submit' command.

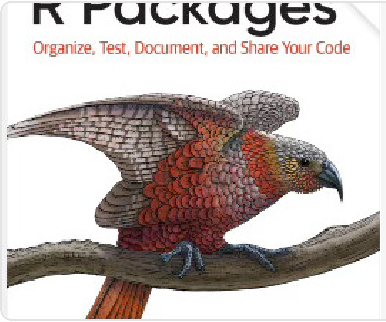
4

Review and Publication

CRAN maintainers will review your package and potentially request revisions.

Conclusion and next steps

Mastering R packaging empowers you to create and share your own code with the world. By following this guide, you can develop robust, well-documented, and easily distributable packages that contribute to the R community. Remember to keep learning, explore advanced techniques, and participate in open-source projects to further enhance your R packaging skills. Happy coding!



https://r-pkgs.org

R Packages (2e)

Learn how to create a package, the fundamental unit of shareable, reusable, and...