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

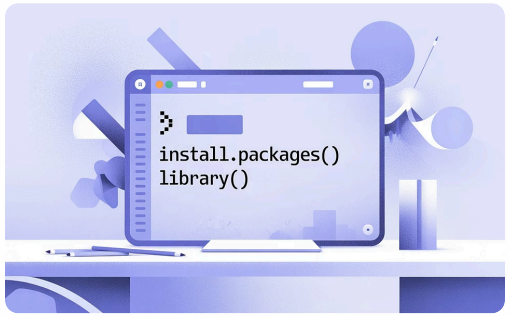
R Package: The fundamental unit of shareable code

Complete bundle

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

CRAN ecosystem

19,000 packages available as of 2023 on CRAN (Comprehensive R Archive Network) - a key reason for R's success.



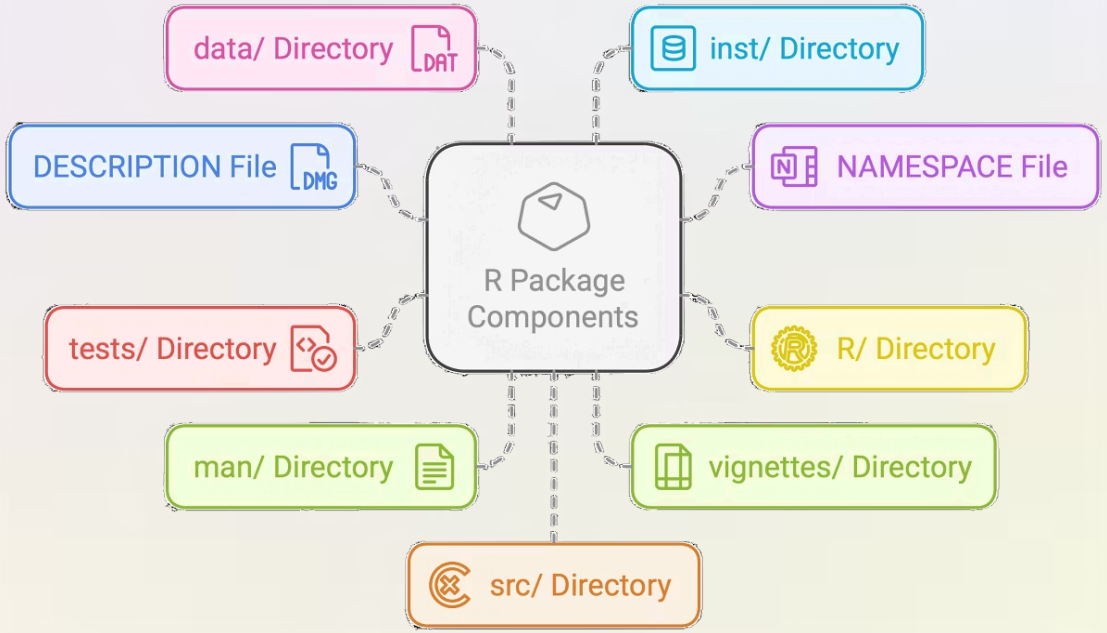
Simple usage

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/.

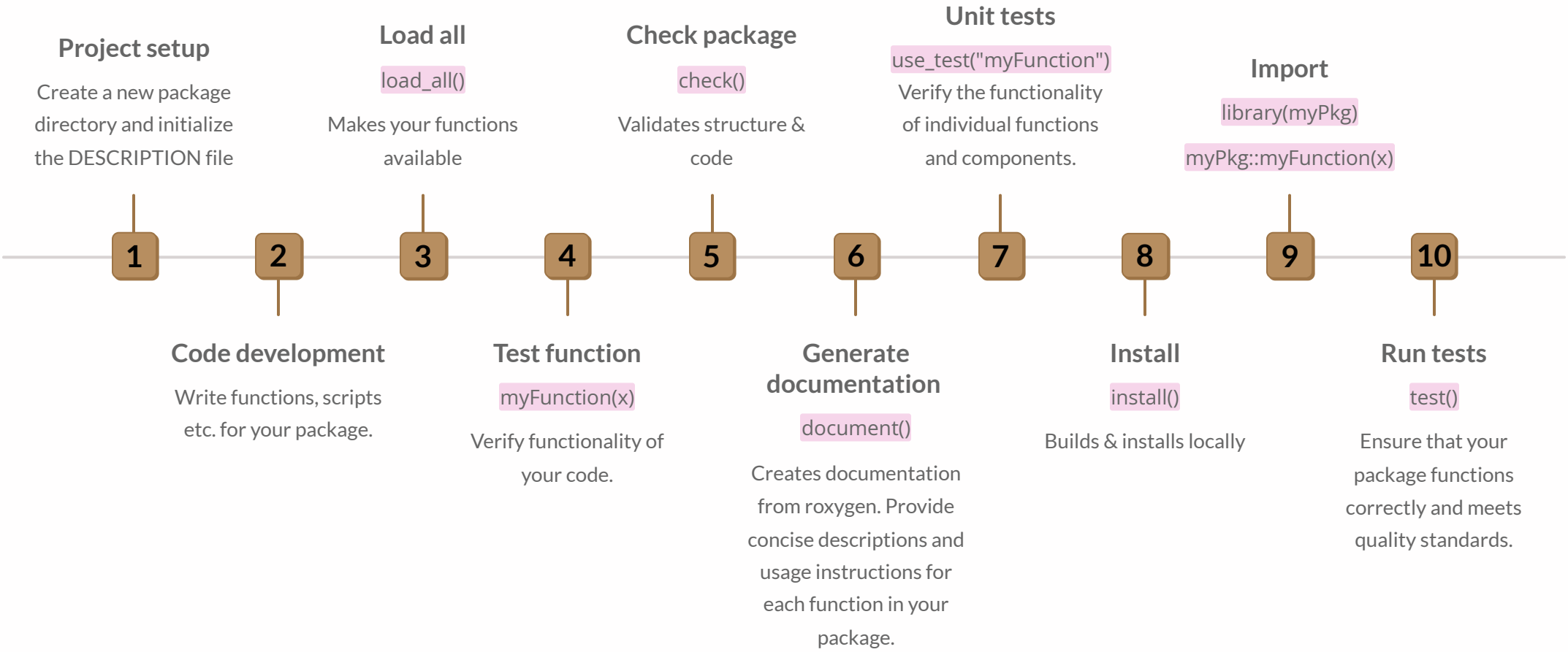
The objective is to show you how to develop packages so that you can share your code for others to use.



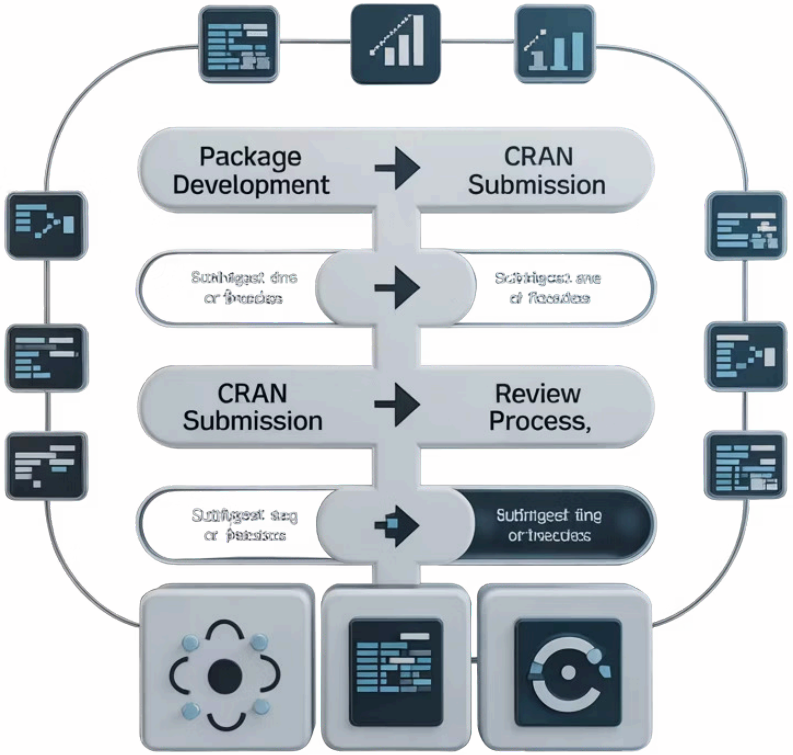
Core elements of R package

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

The packaging workflow



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.

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

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!