

# R package building workshop

(for neuroscientists)

Dominik Krzemiński



Software  
Sustainability  
Institute

# Background

- R is an open source (originally) statistical software, which was the answer to the commercial S (created at Bell Labs)
- The first official release came in 1995.
- Can be easily extended by making new packages.
- The Comprehensive R Archive Network (CRAN) was officially announced 23 April 1997 with 3 mirrors and 12 contributed packages.

source : [en.wikipedia.org](https://en.wikipedia.org)

# Background



## What is package?

In R, a package is the fundamental unit of shareable code. It bundles together code, data, documentation, and tests, and is easy to share with others. As of June 2019, there were over 14,000 packages available on CRAN.

## What is CRAN?

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

Source: <https://cran.r-project.org/index.html>

# Background

`install.packages(packageName)` - command for installing packages

`library(packageName)` - command for loading a package

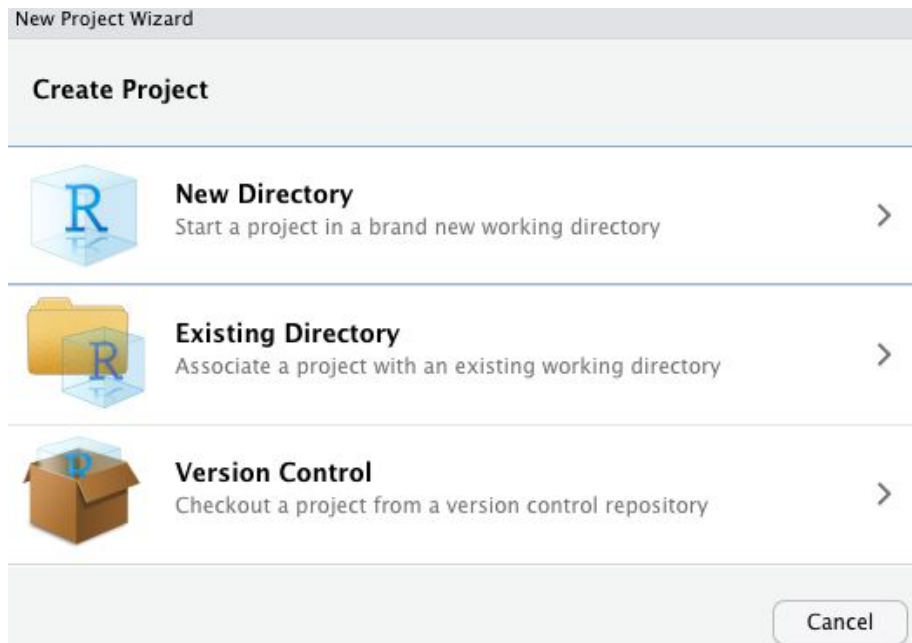
`installed.packages()` - lists all installed packages

`packageVersion("nat")` - checks package version

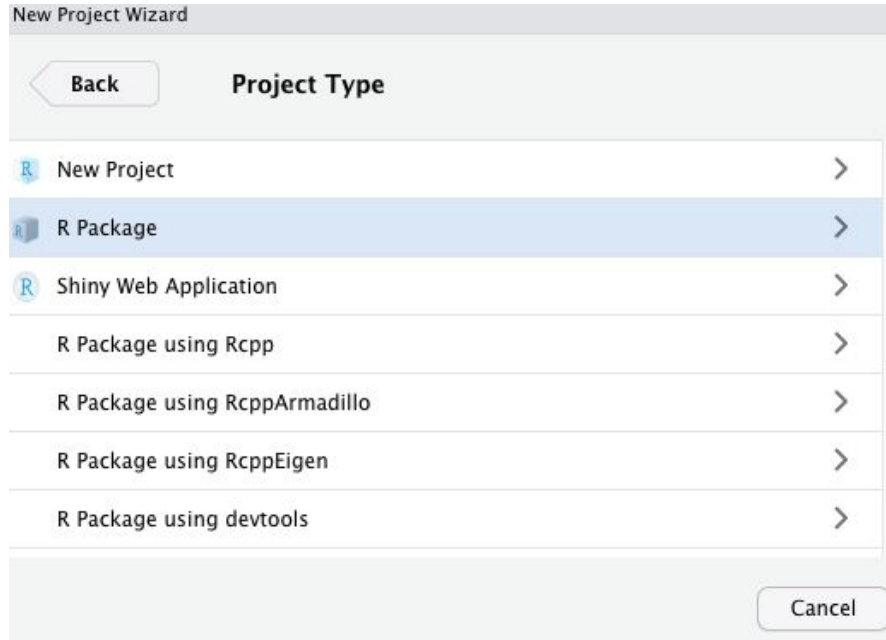
`sessionInfo()` - lists packages loaded to namespace

`.libPaths()` - this is where your packages are located

# RStudio package building start




# RStudio package building start



# RStudio package building start

New Project Wizard

[Back](#) **Create R Package**



Type: Package Package name:

Create package based on source files:

[Add...](#)  
[Remove](#)

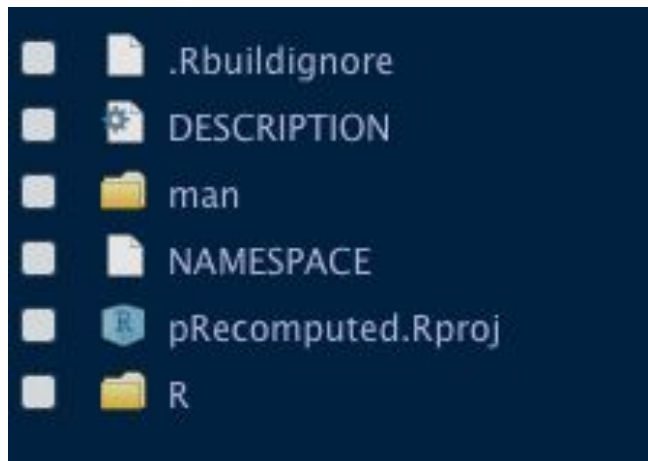
Create project as subdirectory of:

[Browse...](#)

☐ Create a git repository ☐ Use renv with this project

☐ Open in new session [Create Project](#) [Cancel](#)

# Package structure





# Package structure

- R: contains R code files
- data: contains data files
- man: contains documentation files (.Rd)
- src: contains C, C++, or FORTRAN source.
- tests: with tests
- inst: all extra data with subdirectories copied recursively to the installation directory.

## The two most important files

- DESCRIPTION: manual file for the users.
- NAMESPACE

## Less important files

- LICENSE
- Readme

# DESCRIPTION

```
1 Package: pRecomputed
2 Type: Package
3 Title: What the Package Does (Title Case)
4 Version: 0.1.0
5 Author: Who wrote it
6 Maintainer: The package maintainer <yourself@somewhere.net>
7 Description: More about what it does (maybe more than one line)
8     Use four spaces when indenting paragraphs within the Description.
9 License: What license is it under?
10 Encoding: UTF-8
11 LazyData: true
12 |
```

- Title contains no more than 65 characters
- Version must be always a sequence of integers
- Author can be string, but multiple authors have a specific format Author@R

# DESCRIPTION

```
Package: nat
Type: Package
Title: NeuroAnatomy Toolbox for Analysis of 3D Image Data
Version: 1.10.2.9000
Authors@R: c(
  person("Gregory", "Jefferis", email="jefferis@gmail.com", role = c("aut", "cre"), comment = c(ORCID =
"0000-0002-0587-9355")),
  ...
  person("Dominik", "Krzeminski", role = c("ctb"), comment = c(ORCID = "0000-0003-4568-0583"))
)
URL: https://github.com/natverse/nat, https://natverse.org/
BugReports: https://github.com/natverse/nat/issues
Description: NeuroAnatomy Toolbox (nat) enables analysis and visualisation of 3D
  biological image data....
Depends:
  R (>= 2.15.1),
  rgl (>= 0.98.1)
Imports:
  nabor,
  igraph (>= 0.7.1),
  methods,
  filehash (>= 2.3), ...
Suggests:
  spelling,
  testthat,
  Htttr, ...
License: GPL-3
LazyData: yes
RoxygenNote: 7.1.1
Encoding: UTF-8
VignetteBuilder: knitr
Language: en-GB
```

# NAMESPACE

```
1  # Generated by roxygen2: do not edit by hand
2
3  export(build_markdown_report)
4  export(extract_markers_to_md)
5  export(todor)
6  export(todor_file)
7  export(todor_file_addin)
8  export(todor_package)
9  export(todor_package_addin)
10 export(todor_project_addin)
11 import(rex)
12 import(rstudioapi)
13 import(utils)
```

# Code organisation

## Organising

| principle                  | Source file                      | Comments  |
|----------------------------|----------------------------------|---|
| One function               | <code>tidyr/R/uncount.R</code>   | Defines exactly one function, <code>uncount()</code> , that's not particularly large, but doesn't fit naturally into any other <code>.R</code> file   |
| Main function plus helpers | <code>tidyr/R/separate.R</code>  | Defines the user-facing <code>separate()</code> (an S3 generic), a <code>data.frame</code> method, and private helpers  |
| Family of functions        | <code>tidyr/R/rectangle.R</code> | Defines a family of functions for "rectangling" nested lists ( <code>hoist()</code> and the <code>unnest()</code> functions), all documented together in a big help topic, plus private helpers |

Source: <https://r-pkgs.org/r.html>

## Example: fancr package

```
❯ ls -l R
839  4 Nov 11:46 cave.R
2301 15 Oct 15:20 cloudvolume.R
263  15 Oct 15:20 fancr-package.R
4602 15 Oct 15:20 ids.R
916  15 Oct 15:20 meshes.R
2161  4 Nov 11:46 partners.R
3548 15 Oct 15:20 urls.R
2930  4 Nov 12:12 xform.R
1709 15 Oct 15:20 zetta-api.R
136  15 Oct 15:20 zzz.R
```



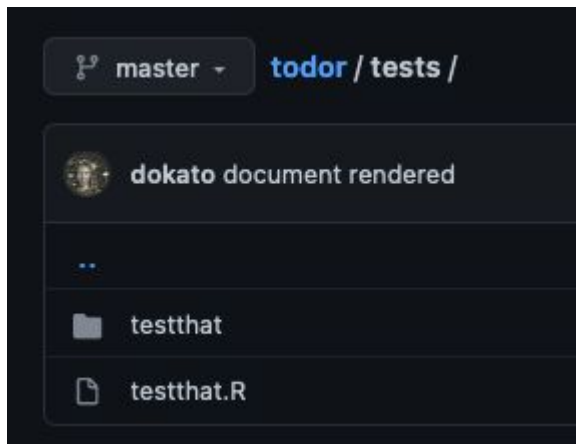
More: <https://roxygen2.r-lib.org/>

```
#' The length of a string
#'
#' Technically this returns the number of "code points", in a string. One
#' code point usually corresponds to one character, but not always. For example,
#' an u with a umlaut might be represented as a single character or as the
#' combination a u and an umlaut.
#'
#' @inheritParams str_detect
#' @return A numeric vector giving number of characters (code points) in each
#' element of the character vector. Missing string have missing length.
#' @seealso [stringi::stri_length()] which this function wraps.
#' @export
#' @examples
#' str_length(letters)
#' str_length(NA)
#' str_length(factor("abc"))
#' str_length(c("i", "like", "programming", NA))
str_length <- function(string) {
}
```



# Testing

Testthat makes testing as easy as possible.



```
1 context("internal functions")
2
3 test_that("test process_file function", {
4   to_detect <- c("BUG")
5   p <- process_file("demo.R", to_detect)
6   expect_equal(length(p), 1)
7   to_detect <- c("BUG", "TODO")
8   p <- process_file("demo.R", to_detect)
9   expect_equal(length(p), 2)
10 })
```

```
> devtools::: test()
```

# Devtools



The aim of devtools is to make package development easier by providing R functions that simplify and expedite common tasks.

Install devtools from CRAN

```
> install.packages("devtools")
```

Most useful command:

```
> devtools::document() # creates documentation and updates NAMESPACE file
> devtools::test() # runs tests with (typically) testthat
> devtools::check() # performs CRAN checks, check options for more advanced settings
```

<https://rawgit.com/rstudio/cheatsheets/main/package-development.pdf>

## Package Development: : CHEAT SHEET



### Package Structure

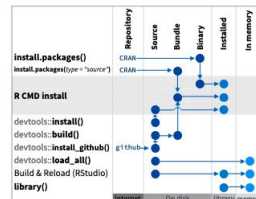
A package is a convention for organizing files into directories. This sheet shows how to work with the 7 most common parts of an R package:



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

- **source** - a directory with sub-directories (as above)
- **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 .buildignore, 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** No strings attached. MIT license applies to your code if re-shared. **GPL-2** GPL-2 license applies to your code, and all code anyone bundles with it, if re-shared.

### Write Code (R/)

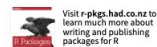
All of the R code in your package goes in **R/**. A package with just an **R/** directory is still a very useful package.

☒ Create a new package project with **devtools::create("path/to/home")**  
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 covered 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/htmlstyle**  
• Click on a function and press **F2** to open its definition  
• Search for a function with **Ctrl + .**



Package: mypackage  
Title: Title of Package  
Version: 0.1.0  
Author@R: person("Hadley", "Wickham", email = "h@adley@r.com", role = c("aut", "cre"))  
Description: What the package does (one paragraph)  
Depends: R (>= 3.1.0)  
License: GPL-2  
LazyData: true  
Tests: true  
dplyr (>= 0.4.0),  
ggplot2 (>= 0.1.0)  
Suggests: knitr (>= 0.1.0)

### 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?

# Building a package

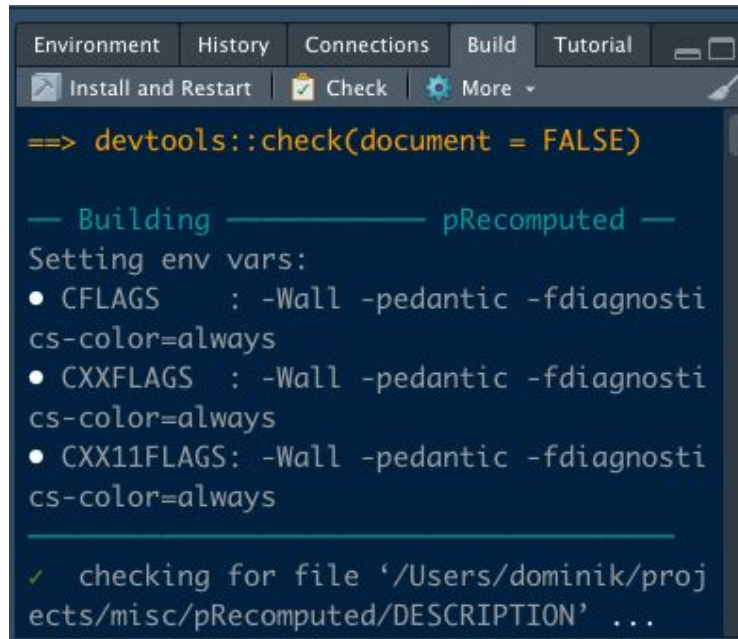
Command line:

```
$ R CMD build packageFolder
```

```
$ R CMD INSTALL
```

```
packageName_version.tar.gz
```

```
$ R CMD check packageName_version.tar.gz
```



```
Environment History Connections Build Tutorial
Install and Restart Check More

==> devtools::check(document = FALSE)

— Building ————— pRecomputed —
Setting env vars:
• CFLAGS      : -Wall -pedantic -fdiagnosti
cs-color=always
• CXXFLAGS    : -Wall -pedantic -fdiagnosti
cs-color=always
• CXX11FLAGS  : -Wall -pedantic -fdiagnosti
cs-color=always

✓ checking for file ‘/Users/dominik/proj
ects/misc/pRecomputed/DESCRIPTION’ ...
```

# Submit to CRAN

## Submit package to CRAN

### Step 1 (Upload)

Your name\*:

Your email\*:

Package\*:

Browse...

No file selected.

(\*tar.gz files only, max 100 MB size)

Optional comment:

\*: Required Fields

Before uploading please ensure the following:

- The package contains a DESCRIPTION file.
- DESCRIPTION file contains valid maintainer field "NAME <EMAIL>".
- You submit a tar.gz created with R CMD build.
- You are familiar with the rest of the [CRAN policies](#)

Upload package

### Step 2 (Submission)

### Step 3 (Confirmation)

# Links

<https://github.com/flyconnectome/pRecomputed>

<https://gist.github.com/dokato/3c1f5d322b4cfad8332836f10a675dac>

Book on package building: <https://r-pkgs.org/>

Package development cheatsheet: <https://rawgit.com/rstudio/cheatsheets/main/package-development.pdf>



Thank you!

