

What are R Packages?

R Packages

- 1. Sets of Functions
- 2. Set of Functions + Documentation
- 3. Set of Functions + Documentation + Data
- 4. Set of Functions + Documentation + Data + Vignet
- 5. Set of Functions + Documentation + Data + Vignett + Versions
- 6. Set of Functions + Documentation + Data + Vignets + Versions + Dependencies



The resource with All the information

https://cran.r-project.org/doc/manuals/r-release/R-exts.html

Writing R Extensions

Table of Contents

Acknowledgements

1 Creating R packages

1.1 Package structur

1.1.1 The DESCRIPTION file

1.1.2 Licensing

1.1.3 Package Dependencies

1.1.3.1 Suggested package

.1.4 The INDEX file

1.5 Package subdirecto

1.1.6 Data in packages

1.1.7 Nov. Dominion in contra

1.1.7 Non-R scripts in packages

1.1.8 Specifying URL

.2 Configure and cleanup

<u>Using Makevars</u>

Z.T.T Openivir suppo

.2.1.2 Using pthreads

1.2.1.3 Compiling in sub-directoric

1.2.2 Configure example

1.2.3 Using F9x code

1.2.4 Using C++11 code

1.2.5 Using C++14 coc

1.2.0 Osing C++17 code

1.3.1 Checking package

1.3.2 Building package tarbal

1.3.3 Building binary packages

What's in a Name?

Package Names:

- must start with letter
- no underscores
- periods allowable or use CamelCase
- $\cdot\;$ can have numbers
- should be Google-able

What's in a Name?

If you have a package name in mind before you start, use the available package:

install.packages("available")
available::available("packageName")

Checks CRAN/Bioconductor/GitHub for potentially similar package names.

Starting Up

 This will use GitHub for packages, sign up for an account if you don't have one.

Use RStudio and the devtools and usethis packages. It's easier.

install.packages("devtools")
install.packages("usethis")

In RStudio, File -> New Project -> New Directory -> R Package using devtools (scroll down).

Trying our first build

In $\mathsf{RStudio}$, click $\mathsf{Build} o \mathsf{Clean}$ and $\mathsf{Rebuild}$.

Boom! You have a package installed.

In RStudio, click Build → Check Package.

This checks our package.

Setting Up RStudio

Go to Build -> Configure Build Tools

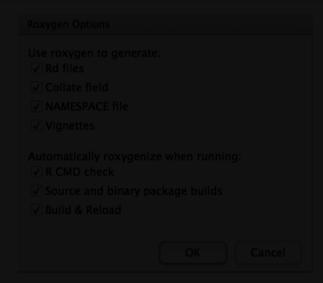
Add ––as–cran to "Check Package" (may not be necessary anymore)



Setting Up: Documentation

In RStudio, click Build → Configure Build Tools →
Generate Documentation with Roxygen. If that is
gray, install roxygen2.

install.packages("roxygen2")

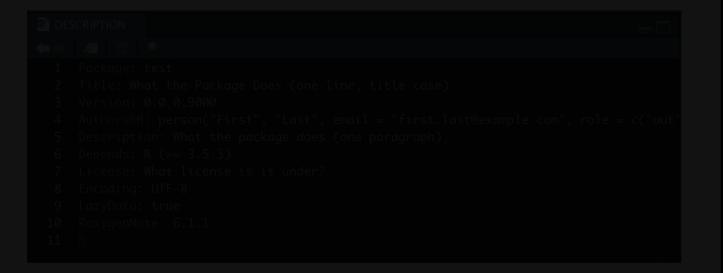


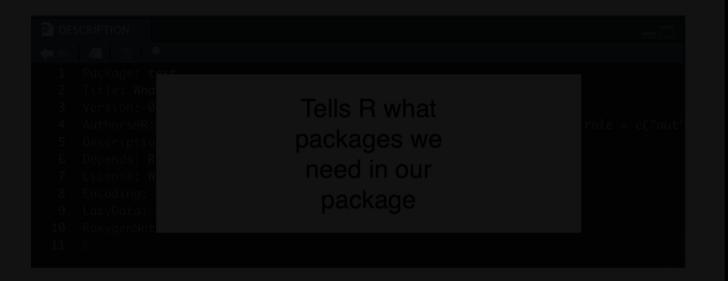
DESCRIPTION file

In the RStudio project, go to "Go to file/function" search bar on the menu bar. Also Ctrl + . on Mac.

- This searches through the files in the package.
 - Also searches for function names and can go to that function in the file

Type description and open that file.





DESCRIPTION

- "Title What the Package Does (Title Case)
- "Author: YOURNAME"
- "Maintainer: YOURNAME <your@email.com>"
- "Description: Use paragraph prose here. Don't start with word package" Use four spaces when indenting paragraphs within the Description.
- "License:", one of GPL-2 GPL-3 LGPL-2 LGPL-2.1 LGPL-3 AGPL-3 Artistic-2.0 BSD_2_clause BSD_3_clause MIT

DESCRIPTION: additional fields

- Imports: package1, package2
 - packages with functions we need in code
- Depends: package3, package5
 - ALL functions loaded from package, but loaded in user library before your package
 - Similar to library (package3);
 library (yourpackage) not recommended
- Suggests: package4, package6: used in examples or vignettes

DESCRIPTION: not so fun default

In the RStudio build, it may add something like:

Depends: R (>= 3.5.3)

This is not great because anyone with a lower ${\tt R}$ version (like 3.4) cannot install your package. Unless you need the newest ${\tt R}$ functionality, delete this line.

Additional Changes to DESCRIPTION

Authors

ladd this to the DESCRIPTION file:

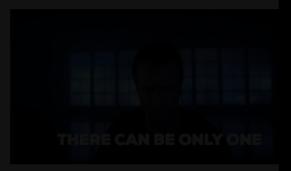
```
Authors@R: person(given = "John",
    family = "Muschelli",
    role = c("aut", "cre"),
    email = "muschellij2@gmail.com",
    comment = c(ORCID = "0000-0001-6469-1750"))
```

Authors

I use Authors@R even if there is only one author because of the desc package issue and that package is relied upon for site building and stuff later.

Roles, see ?person

- cre creator/maintainer, can only have one
- aut = author, can have multiple



Maintainer

If you use Authors@R, you should be able to remove the Maintainer field, but I don't. I am explicit about Maintainer because some people use that field as some parsers use Author and not Authors@R.

Maintainer: John Muschelli <muschellij2@gmail.com>

License

luse License: GPL-3 or GPL-2.

For GPL-3:

usethis::use gpl3 license()

https://www.r-project.org/Licenses/

https://kbroman.org/pkg_primer/pages/licenses.html

DESCRIPTION: Package Title

Change the Title so that it's Title Case and make sure to put single quotes around weird words (like science-specific).

Title: How to Analyze Data and Images

· do **not** start with R package or Package

DESCRIPTION: Description

Change the Description so that it's a sentence (prose) and it ends with a period.

Also I keep putting single quotes around weird words (like science-specific). Make sure to put links in angle brackets (<http...>).

Use DOIs if you can (<doi...>). If you go too long on a line, indent it with 4 spaces "" and go to the next.

Look at some of favorite package descriptions

Roxygen2

Roxygen allows for functions and documentation in the same file. Let's make a function:

```
top = function(x, n) {
    xx = x[1:n, 1:n]
    hist(xx)
    print(xx)
}
```

Save this to top.R file in R/ (where R functions are). Delete hello.R file if present.

Function names should likely be verbs

Adding a package to DESCRIPTION field

Say you want to use dplyr functions for your code:

usethis::use_package("dplyr")

but what if you did this instead? (Function masking)

usethis::use_package("dplyr", type = "Depends")

Roxygen2

Highlight the following code:

```
top = function(x, n) {
```

Go to Code -> Insert Roxygen Skeleton

Roxygen header (what turns into a help file):

```
#' Title
#'
#' @param x
#' @param n
#'
#' @return
#' @export
#'
#' @examples
```

- Oparam stands for a parameter/argument for that function.
- @return denotes what the function returns. This is required.

- @export when people install your package, can they use this function
 - non-exported functions are usually helpers, really small, or not fully formed yet
 - @examples code to show how the function works. Wrap functions in \dontrun{} if not wanted to run and \donttest{} for not testing
 - make sure \dontrun{} not \dontrun {}
 (spaces fail)

```
#' Print the top of a matrix
#' @param x a \code{matrix}
#' @param n Number of rows and columns to display of the matrix
#'
#' @return A \code{NULL}
#' @export
#'
#' @examples
#' mat = matrix(rnorm(100), nrow = 10)
#' top(mat, n = 4)
#' \dontrun{
#' top(mat, n = 10)
#' }
```

Instead of Title, you can use separate @title and @description tags if you want them to be different. For example:

```
#' @title Print the top of a matrix
```

^{#&#}x27; @description \code{top} is a small function to not just present the first rows

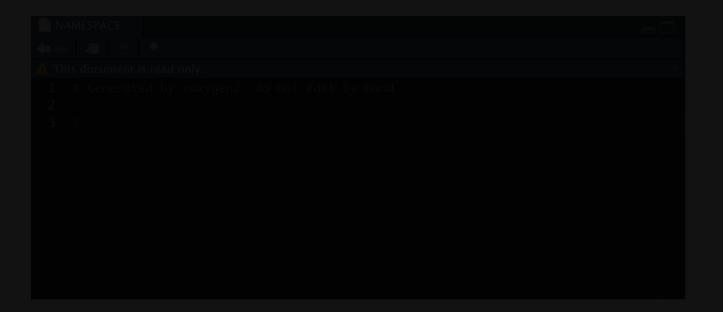
Roxygen and Markdown

If you want to use markdown with roxygen2, then you can use

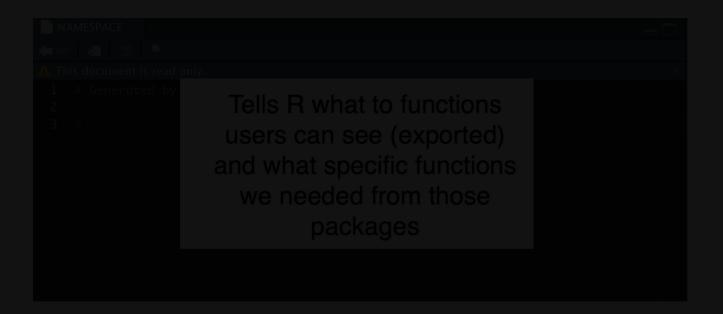
```
usethis::use_roxygen_md()
```

so instead of \code{top} you can write `top` from the previous slide.

NAMESPACE



NAMESPACE



NAMESPACE

In Roxygen:

- \cdot <code>@export</code> <code>adds</code> this to the <code>NAMESPACE</code> file
 - when package is installed, users can call this function

NAMESPACE

In Roxygen:

- @import imports ALL functions from that package
 - if package is listed under Depends in DESCRIPTION, then the whole package is loaded when you load your package
 - otherwise it simply exposes them for your package to use them, but not the user, users still have to do library (PACKAGENAME)

You do not have library() functions in R package code

Check

Go to Build → Check Package. You should see something like:

```
cop: no visible global function definition for 'hist'
Indefined global functions or variables:
  hist
Consider adding
  importFrom("graphics", "hist")
to your NAMESPACE file.
```

The Goal is to have NO Errors/Warnings/Notes on R CMD Check

Why the Message?

R sees hist from graphics, but we never told it we needed it

You should import anything explicitly other than from the base package, includying anything from stats (e.g. quantile) or graphics (e.g. hist). Don't import base.

Importing Packages or Functions from Packages

- @importFrom if you want to import a function, you say @import PACKAGENAME func1 func2
 - only imports these functions (preferred to @import)
 - if pkgA has function A and pkgB has functions A and B, if @import pkgA A, @import pkgB B, then if you call A(), R knows it's from pkgA
 - But should use pkgA::A() in practice

Fixing the Message

Add @importFrom graphics hist to your top.R file (anywhere in the header)

RStudio: Build → Check Package again

Importing Package Functions

- You only have to import a whole package or a package function, don't mix and match
- You only have to import a function in one file.
- For example, @importFrom graphics hist can go in any .R file.
- Any package you want to import, make sure you use use package, so we'd use:

usethis::use package("graphics")

Build and Reload Again

- RStudio: Go to Build → Build and Reload the package
 - First time you may see some warnings (no NAMESPACE file!)
 - Rerunning may get rid of these
- Then try Build → Check Package

Troubleshooting

In case things are not working or seem off

 Delete the NAMESPACE file. If building fails, add ar empty file with

Generated by roxygen2: do not edit by hand

at the top and rerun. - Delete the man folder - roxygen2 will create the documentation automatically

Including Data

 Data should be small, but good for running examples or vignettes.

Run use_data_raw() if you have code to generate data (R data):

```
usethis::use data raw()
```

This will create a data-raw folder.

Including Data

Let's say you want to make a data set mydata_set:

Create data-raw/mydata_set.R and at the end, run

```
usethis::use_data(mydata_set, compress = "xz")
```

And this will make a data/mydata_set.rda file.

Documenting Data

· You don't export data.

```
' My Awesome Data Set
'
' A more detailed description
'
' @format A `data.frame` with 2 columns elements, which are:
' \describe{
' \item{x1}{column of x variables}
' \item{y2}{column of yvariables}
' }
mydata set"
```

Including Non-R data

If you want to include other types of data (such as images) put them into the <code>inst/folder</code> (or a subfolder of <code>inst</code>, I recommend <code>extdata</code>). You can access them using <code>system.file</code> when installed.

For example, if you have a file inst/extdata/myimage.nii.gz:

```
system.file("extdata", "myimage.nii.gz", package = "YOUR PACKAGE NAME")
```

will return the filename for you to access it in code.

Unit Tests

Unit testing packages: testthat and Runit.

To use testthat, run usethis::use_testthat(). All test files must be in testthat/tests/ and must start with the word test.

```
testthat::test_that("Description of test", {
# code for test
testthat::expect_equal(r, 3.5523334)
testthat::expect_true(x)
testthat::expect_is(my_df, "data.frame")
})
```

Good Rule of Thumb: Any issue created/bug found gets a test

If used in a paper, make sure tests check paper results

What to test

- Object output type (list/data.frame)
- Expected output value (3.233453)
- Multiple inputs give expected output
- Even graphs: https://github.com/r-lib/vdiffr

Creating Vignettes

usethis::use vignette("my-vignette")

title is on the document, VignetteIndexEntry is on the website:

95% of what you will be doing is now covered. Mostly it will be documenting and examples

Creating a README

usethis::use_readme_rmd()

This will create a README. Rmd file for your GitHub/Package.

- GitHub renders the README.
- Instructions on how to use the package (examples)
- Instructions on installing 3rd-party dependencies

Spell Checking

Believe it or not, CRAN will check some spelling.

devtools::spell check()

after you have fixed the issues, run

spelling::update wordlist()

to make a WORDLIST file. These are correctly spelled but likely jargon.

Creating NEWS.md

```
usethis::use news md()
```

This will create a NEWS.md file to update with new versions of the package, discussing changes (CRAN Requires).

```
package 0.2.0
- Added function `get_data`
- Fixed bug in `bad function` (fixes issue #52).
```

Continuous Integration

Continuous Integration: Testing on the Cloud

- Travis tests Linux/OSX and Appveyor tests Windows
- Integrate with GitHub, each push tested

```
usethis::use_git()
usethis::use_github() # must have GITHUB_PAT set up
usethis::use_appveyor()
usethis::use_travis()
```

appveyor.yml

Add the following lines:

```
environment:
   global:
    WARNINGS ARE ERRORS: 1
```

to appveyor.yml to make sure warnings are treated as errors.

Code Coverage

- Code coverage is the percentage of code (without whitespace) which is covered by automated tests.
- Calculated using covr package.
- I tend to check coverage by examples, vignettes and tests vs. just tests (default).
- ' You can run covr::package_coverage(type =
 "all") locally.

Code Coverage with Continuous Integration

- Run usethis::use_coverage(), with either coveralls or codecov.
 - Add lines to .travis.yml if necessary

Releasing to CRAN

- Push all changes to GitHub.
- usethis::use_cran_comments()
- Try your package on rhub with devtools::check_rhub()
- Run on CRAN Windows build devtools::check_win_devel()
- Run devtools::release(check = TRUE) and go through the steps
- Validate the email from CRAN for submission.

This is just a starting point

- \cdot A lot of additional things we can discuss individually
- https://cran.r-project.org/doc/manuals/r-release/R-exts.html, though huge, should cover "everything"
- Most of the work is documentation (80%) and vignettes and tests (20%)

Extra Slides

Making a Website

- \cdot $\,$ pkgdown makes nice doc pages for your $\mathsf{package}$
- Run usethis::use_pkgdown().Then rur
 pkgdown::build site().
- Push everything to GitHub.
- Go to GitHub repo → Settings → Scroll down to "GitHub Pages", set Source to master branch /docs folder
- \cdot Refresh your webpage.

No visible binding for global variable issue

If you get No visible binding for global variable, you're likely using Non-Standard Evaluation (NSE) or tidy evaluation

2 options, follow different approaches:

- 1. Use globalVariables
- 2. Let's say co15 is the global variable. At top of function write:

```
col5 = NULL # set to something
rm(list = "col5") # remove it
```

This will remove it but not be a "global" fix. Also, make sure col5 is not an argument in your function