

# Developing an R package: a tutorial

The essentials to write your package

---

Ghislain Durif (<https://gdurif.perso.math.cnrs.fr/>)

July 2022

CNRS – LMBC (Lyon, France)

All contents are available under [CC-BY-4.0](#) license.

## Getting started

---

# R packages to help you create R packages

- `usethis`: to automate package and project creation/configuration/setup
- `devtools`: complete collection of development tools
- `roxygen2`: to document your code and generate help pages
- `lintr`: to review your code (“adherence to a given style, syntax errors and possible semantic issues”)

- If you are not working from Rstudio, you will not benefit from all its functionality but it is possible to do everything from the R command line.
- `pkg::fun()` refers to the function `fun()` defined in the package `pkg` in R.

# Setup your environment

- install R packages providing development tools<sup>1</sup>: `devtools`, `usethis`, `roxygen2`, `lintr`

```
install.packages(c("devtools", "usethis", "roxygen2", "lintr"))
```

- for a more complete setup: see <https://r-pkgs.org/setup.html>

---

<sup>1</sup>`devtools` may require to install additional system libraries depending on your OS, a quick search on the web will help you if you encounter any error.

# Create a package

- Initialize a package template:

```
usethis::create_package("mypkg")
```

- Directly from Rstudio (equivalent): File - New Project - New directory - R package
- **Attention:** if you want to initialize an R package without initializing an Rstudio project<sup>2</sup>, use:

```
usethis::create_package("mypkg1", rstudio = FALSE, open = FALSE)
```

---

<sup>2</sup>e.g. because you want to create your package in an existing Rstudio project, or you don't use Rstudio

# Rstudio project

- Project specific configuration, workspace, history
- Isolated R environment for the project
- RStudio project management feature (e.g. `git` management)
- More information regarding Rstudio project at <https://r-pkgs.org/workflows101.html#projects>



# The “old-fashion” built-in R function to create package

(for more advanced users)

```
## two functions and two "data sets" :
```

```
f <- function(x, y) x+y
```

```
g <- function(x, y) x-y
```

```
d <- data.frame(a = 1, b = 2)
```

```
e <- rnorm(1000)
```

```
## automatically "fill" the package
```

```
package.skeleton(list = c("f","g","d","e"), name = "mypkg2")
```

**Attention:** using `package.skeleton()` creates a package that is not ready “out-of-the-box”, you will have to edit and fix the help pages (e.g. by using `roxygen2`, c.f. later).

# Naming your package

- Three formal requirements:
  - “The name can only consist of letters, numbers, and periods, i.e., ..”
  - “It must start with a letter.”
  - “It cannot end with a period.”
- **Advice:** use a catchy name or acronym with a link to your package functionality
- Check if the name you chose is not already used to name a package with the `available` package
- More details on naming convention at <https://r-pkgs.org/workflows101.html#naming>

# Always choose a license!

- It governs the possibility to **use, modify** or **redistribute** a software
- It helps to identify clear **authorship/copyright**<sup>3</sup>
- Without a license: fuzzy and unclear (generally “all rights reserved” but you are never sure<sup>4</sup>)

---

<sup>3</sup>depending on legal consideration, varying from one country to another

<sup>4</sup>“Was it forgotten or a deliberate choice?”

# Different types of license

- Use a **software-specific license** for software and a content-specific license for data<sup>5</sup>
- **Recommandation:** favor free<sup>6</sup> and open-source licenses (versus proprietary or closed licenses), either **permissive** or **with copyleft**

---

<sup>5</sup>e.g. [Creative Commons license](#) are for contents and not for software

<sup>6</sup>as in "*libre*" and not as in "*gratis*" (proprietary software can be gratis)

# How to choose a license?

- **Introduction**<sup>7</sup> to software license and free/open source software
- `usethis` **doc**

`usethis::use_mit_license()` or `usethis::use_gpl_license()` etc.

Additional resources on software license:

- <https://r-pkgs.org/license.html>
- <https://choosealicense.com>
- <https://opensource.org/licenses>
- <https://www.gnu.org/licenses/license-list.en.html>

---

<sup>7</sup>full disclaimer: I wrote this

# R package structure

---

# Files and sub-directories (1)

Empty package:

```
mypkg
├── DESCRIPTION
├── NAMESPACE
└── R
    └── (empty)
```

More complete package:

```
mypkg2
├── data
│   ├── d.rda
│   └── e.rda
├── DESCRIPTION
├── man
│   ├── d.Rd
│   ├── e.Rd
│   ├── f.Rd
│   ├── g.Rd
│   └── mypkg2-package.Rd
├── NAMESPACE
└── R
    ├── f.R
    └── g.R
```

## Files and sub-directories (2)

- Meta-data files: **DESCRIPTION** and **NAMESPACE** (c.f. later)
- **R** sub-directory: for the **R source files** implementing the function included in your package
- **man** sub-directory: for the the **mandatory help pages**
- **src** sub-directory (optional): for the **code to be compiled** (written in other languages, not in R) included in your package
- **data** sub-directory (optional): for the **data files** attached to your package



# R source code (1)

The **R** sub-directory:

- Write your code as **functions**
- Save your code implementing functions in **R source code files**<sup>8</sup>
- Group related functions in the same file
- Create and edit source code files manually or with `usethis::use_r("name")`
- See **R code formatting convention**
- Check your code formatting with the `lintr` package: `lintr::lint_package()`

---

<sup>8</sup>with `.R` extension

### Debugging-friendly advice:

- Avoid very long functions (split long functions into several shorter ones)
- Factorize re-used code into specific functions (avoid copying-pasting chunk of codes several time)

# From R scripts to R functions

Script:

```
# data
a = 7
b = 3
# intermediate operations
tmp1 = 2 * a
tmp2 = b / 6
# final computations
c = tmp1 + tmp2
```

**Objective:** simplify your code by “hiding” intermediate steps into a function

Implementing functions in your package:

```
myFun <- function(x, y) {
  tmp1 = 2 * x
  tmp2 = y / 6
  return(tmp1 + tmp2)
}
```

In a script using your package:

```
library(mypkg)
# data
a = 7
b = 3
# computations
c = myFun(a,b)
d = myFun(10, 3)
```

# Meta-data files

- **DESCRIPTION**: a structured text file giving **information about your package** (title, **description, authors, license, dependencies**, etc.)
- **NAMESPACE**: a text file indicating names of R objects<sup>9</sup> (functions, datasets) that are imported in your package (from other packages), and/or exported by your package (to be usable when installing your package)

---

<sup>9</sup>and/or the name of the dynamic library related to compiled codes used in your package if relevant

## DESCRIPTION file (1)

- Can be edited manually, or created and modified with `usethis::use_description()` and other `usethis::use_XXX()`
- Setup your package requirements and dependencies (c.f. later)
- More details at <https://r-pkgs.org/description.html>
- Important: see [package versioning convention](#) (e.g. `1.0.1`)

## DESCRIPTION file (2)

Example:

```
Package: mypkg
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
  person(given = "First",
         family = "Last",
         role = c("aut", "cre"),
         email = "first.last@example.com",
         comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a
  license
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
```

# NAMESPACE file

Can be created with `usethis::use_namespace()`, or edited manually, or (better) **automatically updated** thanks to `roxygen2` **inline documentation** (c.f. later)

Example:

```
# Generated by roxygen2: do not edit by hand
```

```
importFrom(stats, runif)
export(my_function)
```

## Data in your package

- Binary R data file (`.Rda` or `.Rdata` file) can be stored in the `data` sub-directory
- Raw data can be stored in the `inst` sub-directory and found after installation with the `system.file()` function<sup>10</sup> (**important**: you do not need to search for the `inst` sub-directory, just anything in it)

```
system.file(filename, package = "mypkg")  
system.file(dirname, package = "mypkg")  
system.file(package = "mypkg") # package install root directory
```

More at <https://r-pkgs.org/data.html>

---

<sup>10</sup>any other file or sub-directory shipped with a package can be found likewise



# help/man pages

- Available with the R commands `?function_name` (e.g. `?rnorm`) or `help(function_name)`
- **Content:**
  - usage description and functioning details
  - input arguments and return value description
  - function authorship
  - link to related functions
  - bibliographic reference
  - minimum working examples
- Encoded in `.Rd` files in the `man` sub-directory: structured text files with a specific syntax

## Document your code and generate help page at once

- Avoid creating and editing `.Rd` file manually (laborious)
- **Good practice: DOCUMENT YOUR CODE** (with inline comments) for others and YOUR FUTURE SELF
- Inline code documentation with `roxygen2` based on tags identified with `@`

# Document your code and your package with roxygen2 (1)

Inline code documentation (identified with `#'`):

```
#' Add together two numbers
#'  
#' @param x numeric value  
#' @param y numeric value  
#' @return The sum of \code{x} and \code{y}  
#' @author Anonymous  
#' @examples  
#' add(1, 1)  
#' add(10, 1)  
#' @export  
add <- function(x, y) {  
  x + y  
}
```

Corresponding `.Rd` file:

```
% Generated by roxygen2 (3.2.0): do not edit by hand  
\name{add}  
\alias{add}  
\title{Add together two numbers}  
\usage{  
  add(x, y)  
}  
\arguments{  
  \item{x}{A number}  
  
  \item{y}{A number}  
}  
\value{  
  The sum of \code{x} and \code{y}  
}  
\description{  
  Add together two numbers  
}  
\examples{  
  add(1, 1)  
  add(10, 1)  
}
```

## Document your code and your package with roxygen2 (2)

- Generate the man pages (and update `NAMESPACE` file) with `devtools::document()` or in Rstudio interface (Build panel - More - Document<sup>11</sup>)
- Identify **exported functions**<sup>12</sup> with the tag `@export` (automatically added to the `NAMESPACE` file)
- Identify **imported functions**<sup>13</sup> with the tag `@importFrom package function` (automatically added to the `NAMESPACE` file)

---

<sup>11</sup>keyboard shortcut: `CTRL + SHIFT + D`

<sup>12</sup>your functions that will be available to users

<sup>13</sup>functions from other packages that you use

# Document your code and your package with roxygen2 (4)

## A more complete example

```
#' A function to do some stuff
#' @description
#' Do some stuff
#' @details
#' I do the stuff in a complicated way.
#' @param x numeric value
#' @param y numeric value
#' @return what the function is returning
#' @author Someone
#' @importFrom stats rnorm
#' @seealso [mypkg::my_other_fun()]
#' @examples
#' add(1, 1)
#' add(10, 1)
#' @export
my_fun <- function(x, y) {
  tmp = rnorm(7)
  ...
}
```

## Tips:

- add `Roxygen: list(markdown = TRUE)` to the `DESCRIPTION` file to use markdown syntax in documentation chunks or run `usethis::use_roxygen_md()` (possible conversion from existing standard `roxygen2` syntax with `roxygen2md` package)
- **internal functions** (only used by other functions in your package, and not to be available for users) can be tagged with `@keywords internal` and (and without `@export` tag)

## Document your code and your package with roxygen2 (5)

### References:

- More details at <https://r-pkgs.org/man.html>
- `roxygen2` [cheat sheet](#)
- `roxygen2` help pages about [tags](#) and [documentation chunk format](#)

## Manage your dependencies (1)

Several fields in the `DESCRIPTION` file (example):

`Depends: R (>= 3.1.0)`

`LinkingTo:`

`Rcpp (>= 1.0.1),`

`RcppEigen (>= 0.3.3.5)`

`Imports:`

`Rcpp (>= 1.0.1),`

`openssl`

`Suggests:`

`testthat (>= 2.1.0)`

`SystemRequirements: C++11`

## Manage your dependencies (2)

- the **minimal R version** required by your package: `Depends: R (>= 3.1.0)`
- **required packages** (with optional minimal versions) forr your package to work:  
`Imports: ...`
- (optional) additional packages (with optional minimal versions) that are not necessary for your package to work but that would improve the user experience with your package: `Suggests: ...`



## Manage your dependencies (3)

- Additional (optional) fields:
  - **LinkingTo** (useful if external codes needs to be compiled and linked against external library)
  - **OS\_type: unix** (to specify which OS are supported<sup>14</sup>)
  - **SystemRequirements: C++11** (to specify additional external system requirements<sup>15</sup>)

---

<sup>14</sup>here it means that Windows is not supported

<sup>15</sup>here a C++ compiler compatible with C++11 standard

## Manage your dependencies (3)

- `usethis::use_package()` to update `Imports` or `Suggests` fields
- **Note:** not updated by `roxygen2`
- More details at <https://r-pkgs.org/description.html#dependencies>

## Manage your dependencies (4)

Imported objects (functions, dataset) should be declared in the `NAMESPACE` file

→ automatic management thanks to `roxygen2`

## Other (optional) files and sub-directories (1)

- `README`<sup>16</sup>, `LICENSE` (depending on the license your choose), `COPYRIGHT` (to detail authorship, copyright associated to the package content) files
- `src` directory: source codes to be compiled (c.f. later)
- `inst` directory to store additional files (e.g. required for tests, vignettes, etc.), raw data, etc.
- `tests` directory to write automatic tests (c.f. later)

---

<sup>16</sup>possible format: `.md` or `.Rmd` (Rmarkdown), see `usethis::use_readme_md()` or `usethis::use_readme_rmd()`

## Other (optional) files and sub-directories (2)

- More details at <https://r-pkgs.org/misc.html>
- Non-standard files can be present in your project but not shipped in your package but creating and filling a **.Rbuildignore** file

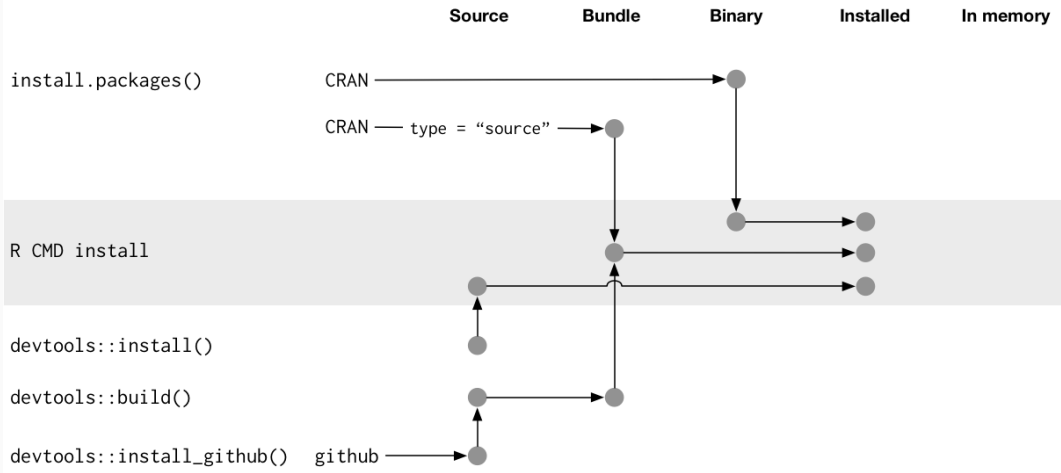
# Workflow

---

# Package state

- **source:** what you are writing
- **bundled:** a single-file compressed version containing your package sources (how package are shipped by the CRAN for installation)
- **binary:** a single-file binary version of your package containing compiled library (if relevant), mainly used by the CRAN to ship package for installation on Windows
- **installed:** available as a library on your system, i.e. the package files and sub-directories (along with library files if compilation was needed) have been copied somewhere on your computer
- **in-memory:** loaded and ready to use (after calling `library(mypkg)`)

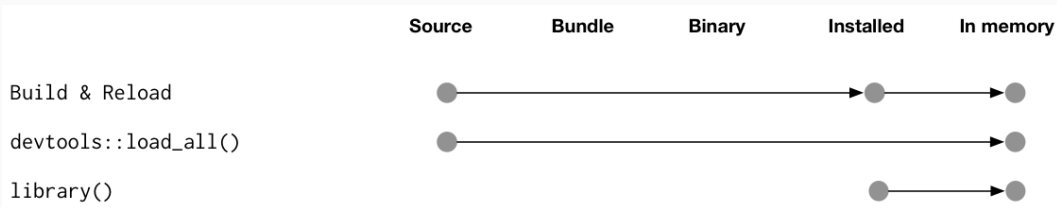
# Dev workflow (building and installing a package)



Ref: <https://r-pkgs.org/package-structure-state.html>



# Dev workflow (loading a package)



Ref: <https://r-pkgs.org/workflows101.html#load-all>

**Load your package** for a test drive (manual test) without building/installing it

- `devtools::load_all()`
- in Rstudio interface (Build panel - More - Load all<sup>17</sup>)

**Development cycle:** write code, test it, correct your code, test it, etc.

---

<sup>17</sup>keyboard shortcut: `CTRL + SHIFT + L`

Generate the man pages (and update `NAMESPACE` file)

- `devtools::document()`
- in Rstudio interface (Build panel More - Document<sup>18</sup>)

---

<sup>18</sup>keyboard shortcut: `CTRL + SHIFT + D`

## Prepare your package for installation (and distribution)

- `devtools::build()`
- in Rstudio interface (Build panel - More - “Build source package”)
- R built-in shell command line tool<sup>19</sup>: `R CMD build mypkg`
- Create a `.tar.gz` archive files containing the sources (or a `.zip` file if you use “Build binary package”) ready for installation

---

<sup>19</sup>`R.exe` on Windows

Verify that your **package is functional** and that your **package structure is correct**

- `devtools::check()`
- in Rstudio interface (Build panel - Check)
- R built-in shell command line tool<sup>20</sup>: `R CMD check mypkg_1.0.0.tag.gz`
- **Verbose output**: often clearly identify problems (and suggest fixes)
- More details at <https://r-pkgs.org/r-cmd-check.html>

---

<sup>20</sup>`R.exe` on Windows

## usethis (exhaustive tour)

See <https://usethis.r-lib.org/reference/index.html>

Possible to write every files manually for more advanced users.

## devtools (exhaustive tour)

See <https://devtools.r-lib.org/reference/index.html>

(**devtools** exports several functions from other development-oriented packages)

The end

---



Questions ?