Introduction to Building R Packages

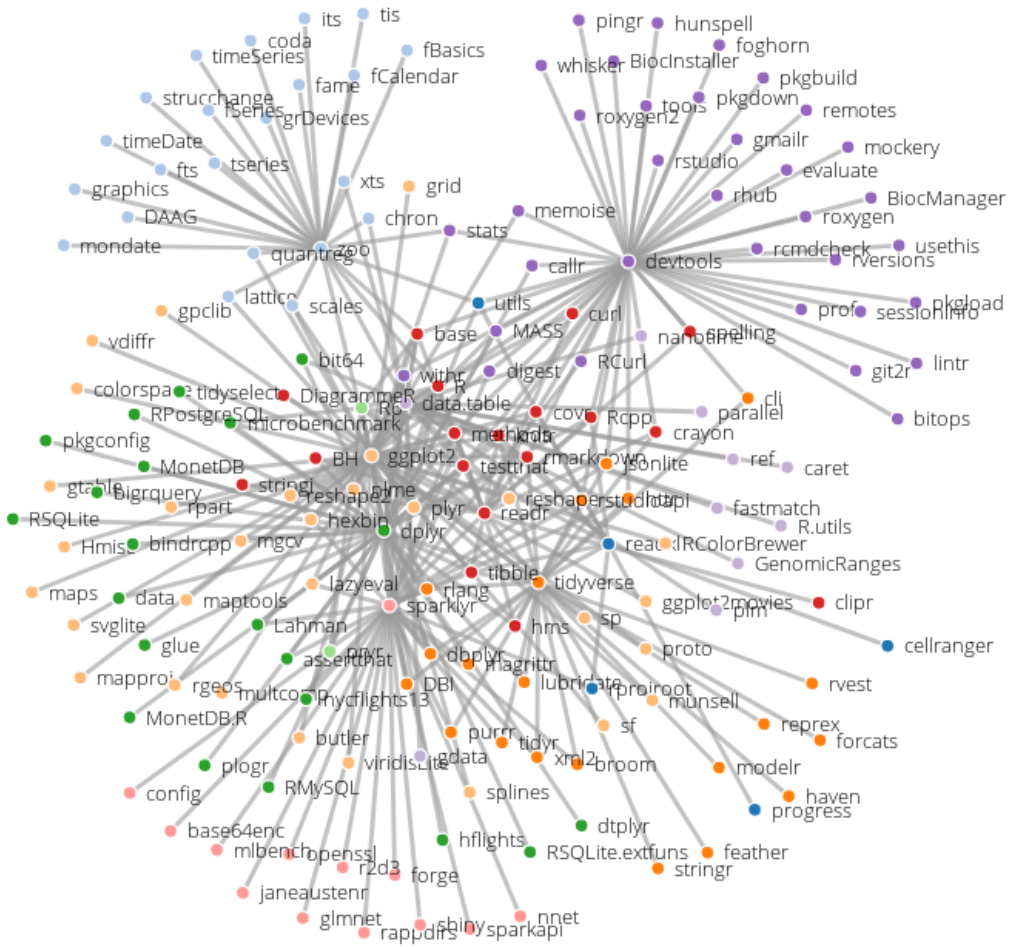
Juliane Manitz – r@manitz.org

R Ladies Boston

January 30, 2019

Motivation

CRAN Repository: Top packages dependency graph



Total number of packages on CRAN: 16,631

Source: `https://www.rdocumentation.org/trends`; January 25, 2019

Install R Packages

- Most common/popular repositories for R packages include:
  - CRAN: official repository maintained by maaany servers world-wide

```
install.packages("package")
```

  - Bioconductor: specific packages for bioinformatics

```
source("https://bioconductor.org/biocLite.R")
biocLite() # core packages
biocLite("package")
```

  - github: no review process

```
devtools::install_github()
```

- Some house-keeping tools

```
installed.packages()        # check all installed packages
update.packages()           # update package
remove.packages("package") # remove package
```

Motivation

Load package

```
library(package)
require(package)                  # no error if not installed

package::function()               # execute specific function only

detach("package", unload=TRUE) # unload package
```

Help files

```
?function
?package::function

# package overview
help(package = "packagename")

# vignette/tutorials
vignette(package = "packagename")
vignette("vignettename")
```

Motivation

Why Building R Packages?

- platform-independent distribution of R code
  - alpha/beta versions on R-forge or github
  - finished projects on CRAN or Bioconductor
- archiving R code for a specific project and software documentation
- reproducible research: distribute data and software accompanying a publication
- maintainance of dependencies, and automated loading of required external code
- CRAN uses `R CMD check` to test package on various platforms; packages are tested daily

Table of Contents

File Structure

A basic (but good) R package has the following structure:

`DESCRIPTION` what does the package? who can use it (license)? who is responsible (maintainer)?

`NAMESPACE` which function should be seen by the user? which are internal?

`R/` R functions

`man/` documentation, help files with syntax similar to LaTeX

`data/` example data files

Additional (optional) files in R packages:

`src/` C, C++, FORTRAN source code

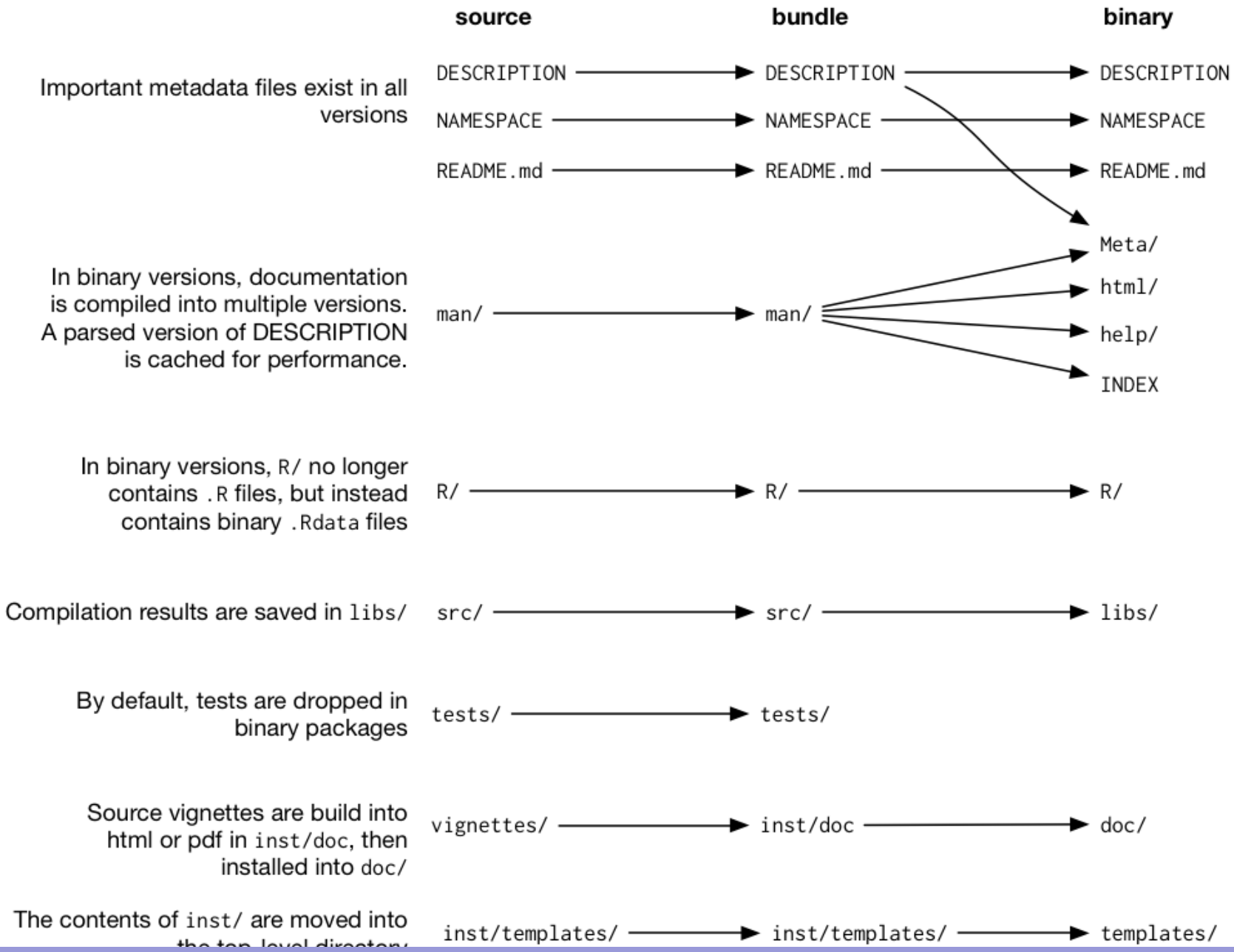
`tests/` tests

`vignettes/` vignette

`inst/CITATION` how should the user cite the package?

...

File Structure



R code R/

"There are only two hard things in Computer Science: cache invalidation and naming things." – Phil Karlton

Naming R Packages:

- can contain letters and numbers, but start with a letter
- avoid self-invented abbreviations, captital letters, ...
- should be identifiable in online search
- R package `available`

`R/` **directory** contains all R code:

- each function in a separate file (good for small packages)
- everything in one file (ok for small packages)
- group related functions in a file with meaningful names (best solution for larger projects)

Example: Create R code and structure for myutils

The standard structure can be obtained automatically using utils::package.skeleton():

```
# define some basic functions
add <- function(x, y){ x + y }
plusone <- function(x){ x + 1 }
# prepare some example data
dat <- data.frame(id=1:10, x=rpois(10, 5), y=rpois(10, 5))

# test your source code
add(10, 1)
plusone(4)
dat

# create standard structure
fdlist <- c("add","plusone","dat")
package.skeleton("myutils", fdlist)
```

DESCRIPTION file

```
Package: myutils
Type: Package
Title: What the package does (short line)
Version: 1.0
Date: 2019-01-24
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does (maybe more than one line)
License: What license is it under?
```

DESCRIPTION file

```
Package: ggplot2
Version: 3.1.0
Title: Create Elegant Data Visualisations Using the Grammar of
       Graphics
Description: A system for 'declaratively' creating graphics,
             based on "The Grammar of Graphics". [...]
Depends: R (>= 3.1)
Imports: digest, grid, gtable (>= 0.1.1), lazyeval, MASS, mgcv,
         plyr (>= 1.7.1), reshape2, rlang (>= 0.2.1), scales,[...]
Enhances: sp
License: GPL-2 | file LICENSE
URL: http://ggplot2.tidyverse.org
BugReports: https://github.com/tidyverse/ggplot2/issues
Collate: 'ggproto.r' 'ggplot-global.R' 'aaa-.r' .....
VignetteBuilder: knitr
RoxygenNote: 6.1.0
NeedsCompilation: no
Author: Hadley Wickham [aut, cre], Winston Chang [aut], [...]
Maintainer: Hadley Wickham <hadley@rstudio.com>
Date/Publication: 2018-10-25 04:30:25 UTC
Built: R 3.5.2; ; 2019-01-07 06:31:07 UTC; unix
```

DESCRIPTION file

`Package:` name of the package

`Title:` description of the package (one line, < 65 characters)

`Description:` detailed description (one paragraph, multiple sentences)

`Version:` version number formatwise `major.minor-patchlevel` or `major.minor.patchlevel`.

`Maintainer:` name und e-mail of a person who wants to take over the responsibility

`License:` abbreviation of a software licence (`GPL-2`, `BSD`, `MIT`, ...)

`Depends`, `Suggests`, `Imports`, `Enhances` package dependencies

`URL:` for website of a package

`Collate:` order R files are loaded (default: alphabetically)

.Rd manual file

- R documentation format is very LaTeX-like output (LaTeX installation required)

```
\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)
}
```

Package `roxygen2`

There are three steps in the transformation from roxygen comments in your source file to human readable documentation:

1. add roxygen comments to your source file
2. `roxygen2::roxygenise()` or `devtools::document()` converts roxygen comments to `.Rd` files
3. `R CMD check` converts `.Rd` files to human readable documentation

`roxygen2`: `http://cran.r-project.org/web/packages/roxygen2/vignettes/rd.html`

Package roxygen2

- roxygen comments start with #'
- tags like @param, @return, @author define parts in .Rd file
- tags like @includes, @export, @importFrom generate NAMESPACE und Collate
- tags like @method for OOP documentation

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

Package roxygen2

add {rvest} R Documentation

Add together two numbers

Description

Add together two numbers

Usage

```
add(x, y)
```

Arguments

x	A number
y	A number

Value

The sum of x and y

Examples

```
add(1, 1)
add(10, 1)
```

R CMD check

For building a R package `pkg` run the following commands in your console:

`R CMD SHLIB pkg` compiles C/C++/Fortran code in `pkg/src`

`R CMD build pkg` generates package bundle `pkg.tar.gz` or `pkg.zip`

`R CMD INSTALL pkg.tar.gz` installs package

`R CMD check pkg.tar.gz` runs CRAN validity checks (is `pkg` valid?)

In windows, installation of `Rtools` is required:

```
# On windows:
R CMD INSTALL --build pkg
```

Example: Build and Check `myutils`

```
jmanitz@Rladies$ R CMD build myutils_complete
* checking for file ‘myutils_complete/DESCRIPTION’ ... OK
* preparing ‘myutils’:
* checking DESCRIPTION meta-information ... OK
* installing the package to process help pages
* [...]
* building ‘myutils_1.0.tar.gz’

jmanitz@Rladies$ R CMD check myutils_1.0.tar.gz
* using log directory ‘/home/Rladies/example/myutils.Rcheck’
* using R version 3.5.2 (2018-12-20)
* using platform: x86_64-pc-linux-gnu (64-bit)
* checking for file ‘myutils/DESCRIPTION’ ... OK
* [...]
* checking PDF version of manual ... OK
* DONE

Status: OK
```

Resources

- Hadley Wickham (2015). *R packages*. O'Reilly Media. Available online: `http://r-pkgs.had.co.nz/`
- R-project manual: Writing R Extensions. Available online: `https://cran.r-project.org/doc/manuals/r-release/R-exts.html`
- Friedrich Leisch. *Creating R Packages: A Tutorial*: `http://cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf`
- Internet search, R-help, other R packages, ...

Namespace

- although the NAMESPACE file looks like R code, it is not processed as R code
- specifies which variables in the package should be exported to make them available to package users, and which variables should be imported from other packages

```
import(foo, bar)      # all functions from foo and bar imported
importFrom(foo, f, g) # selected functions f and g from foo
export(f, g)          # export functions f and g
```

- for packages with many variables to export it may be more convenient to specify the names to export with a regular expression

```
exportPattern("\^{}[\^{}\textbackslash\textbackslash.]")
```

Object-Oriented Programming (OOP)

- in OOP, computer programs are designed by making them out of objects that interact with one another
- a **class** defines the behaviour of **objects** by describing their attributes and their relationship to other classes.
- the class is also used when selecting **methods**, functions that behave differently depending on the class of their input.
- R has three OO systems: S3, S4, Reference classes (not covered by this lecture), and the system of base types

Picking a System

- majority of object-oriented code that I have written in R is S3
- S3 is sufficient for fairly simple objects and methods for pre-existing generic functions like `print()`, `summary()`, and `plot()`
- S4 may be more appropriate for more complicated systems of interrelated objects
- good example for S4 is the Matrix package by Douglas Bates and Martin Maechler

S3/S4 Object System Comparison

	S3	S4
defintion	not neccessary	`setClass('class_name', ...)`
generation of instances	`class(object) <- 'class_name'`	`new('class_name')`
inheritance	vector of class names (children before parents)	`contains='parental_cl'` in definition
test class	`inherits(object, 'class_name')`	`is(object, 'class_name')`
access slots	depends on base type: for lists $ or `[[]]`.	new operator: `@`
list methods	`methods()`	`showMethods()`

Conventions: S3/S4 Classes For S3 and S4, there are the following conventions

- constructor functions should be named like the class itself, e.g. `lm()`, with exception if a class is the return value of a number of functions
- standard methods, which are available supplied for many classes:

print	basic object information, also when using <RET> (S4 `show()`)
summary	more detailed description of the objects instance
plot	graphics

- every method should have the arguments of the correspondig generic (same order and defaults) and accept an arbitrary number of additional arguments (use `...`)

Namespace

- ensure that the generics are imported and register the methods using `S3method` directives
- the function `print.foo` does not need to be exported

```
# example myutils
export(add)
S3method(print, add)
export(plusone)
S3method(print, plusone)
.
.

# example ggplot2
S3method(autoplot,default)
export(autoplot)
import(plyr)
importFrom(MASS,cov.trob)
.
.
```

Namespace

- some additional steps are needed for packages which make use of S4 classes and methods
- package should depend on package `methods` (also `DESCRIPTION` file)
- you may need to import `graphics::plot` to make visible a function that can be converted into a implicit generic

```
exportPattern("^[[:alpha:]]+") # regular pattern

import("methods")              # S4
importFrom(graphics, "plot")   # S4 plot

# namespaces from dependencies
importFrom("utils", str, head, tail, assignInNamespace, capture.output)

# export methods and classes
exportMethods("cbind2", "rbind2", "plot", "show", "summary" )
exportClasses("denseMatrix", "sparseMatrix")
```

devtools

R functions from devtools that simplifies R packaging:

`load_all()` simulates installing and reloading your package

`document()` updates documentation, file collation and NAMESPACE.

`test()` reloads your code, then runs all testthat tests.

`run_examples()` will run all examples to make sure they work.

`check_doc()` runs most of the documentation checking components of `R CMD check`

`check()` updates the documentation, then builds and checks the package

`build()`, `build_win()` builds a package file from package sources (only one R version)

testthat

"A unit testing system designed to be fun, flexible and easy to set up." (Wickham)

- Provides functions that make it easy to describe what you expect a function to do, including catching errors, warnings and messages.
- Displays test progress visually, showing a pass, fail or error for every expectation. If you're using the terminal, it'll even colour the output.

```
library(testthat)
library(yourpackage)

test_check("yourpackage")
```

testthat

`expect_that` describes expected result of your code (value, class, correct error message, computation time, etc.)

`test_that` is grouping a number of `expectation`'s for one functions or a feature

`context()` is grouping a number of content-related tests

```
require(testthat)
test_that("trigonometric functions match identities", {
  expect_that(sin(pi / 4), equals(1 / sqrt(2)))
  expect_that(cos(pi / 4), equals(1 / sqrt(2)))
  expect_that(tan(pi / 4), equals(1))
})
```

`testthat`: Hadley Wickham (2011). *testthat: Get Started with Testing*. The R Journal 3(1).