Lecture 14: Continue on R package intro

Irina Gaynanova

10/8/2019

Before we start

- Open R project associated with empty R package skeleton we created last time
- If you don't have it, follow

```
library(usethis)

# Specify path where you want to store the package
path <- getwd() # current working directory
# or explicit path
# path <- "/Users/Irina/Documents/Teaching/TAMU/Computing"

# Create full path
fullpath <- file.path(path, "MyPackageTest1")

# Create package skeleton
create_package(fullpath)
```

DESCRIPTION - Author

```
person("Irina", "Gaynanova",
    email = "irinag@stat.tamu.edu",
    role = c("aut", "cre"))
```

```
## [1] "Irina Gaynanova <irinag@stat.tamu.edu> [aut, cre]"
```

Tell that both author (*aut*) and maintainer (*cre*) is Irina Gaynanova.

From H. Wickham

> *The full list of roles is extremely comprehensive. Should your package have a woodcutter ("wdc"), lyricist ("lyr") or costume designer ("cst"), rest comfortably that you can correctly describe their role in creating your package*

DESCRIPTION - Multiple authors

```
c(person("Yunfeng", "Zhang",
    email = "some email",
    role = c("aut", "cre")),
    person("Irina", "Gaynanova",
        email = "irinag@stat.tamu.edu",
        role = "aut"))
```

```
## [1] "Yunfeng Zhang <some email> [aut, cre]"
## [2] "Irina Gaynanova <irinag@stat.tamu.edu> [aut]"
```

LICENSE

The most confusing part.

- **GPL 2** or **GPL-3** - open source, most commonly used to my knowledge, anyone who does derivatives of your work has to make the code available under the same license.
- **MIT** - also open source, the license should always be distributed with the code.

See R packages for more.

Version

From Hadley again

> *A released version number consists of three numbers, <major>.<minor>.<patch>. For version number 1.9.2, 1 is the major number, 9 is the minor number, and 2 is the patch number. Never use versions like 1.0, instead always spell out the three components, 1.0.0.*

The fourth number always indicated the package in development, i.e. 9000 (cause many development may happen before the release)

R package building and checks

Run checks. What does it complain about?

Can Install and Restart.

Can build binary and source files (the latter are needed for CRAN submission)

Example function/documentation

Suppose you want to have a soft-thresholding function available within your package.

```
soft <- function(a, lambda){
  sign(a) * max(abs(a) - lambda, 0)
}
```

Save it as soft.R file within **R/** package folder.

Try to build a package using Build -> More -> Build Source Package.

Install and restart (from Build menu). Try using function soft. What happens?

Where is function soft?

The package by default **masks** function soft because it has **not been exported**

You can still see that it's there by typing

```
getAnywhere(soft)
```

getAnywhere() is a very useful function to check other packages code for functions that are **masked** from the user, that is have not been exported/have no documentation

Go to **NAMESPACE** file. What do you see?

Basics of documentation

!!! You may need to first install roxygen2 using

```
install.packages("roxygen2")
```

To make functions usable, they have to be documented (at least barely).

While one can write documentation completely manually, it is **strongly recommended** and **required for your project** to use Roxygen.

Go to your Build Toold (Build $->$ More $->$ Configure Build Tools). Make sure you have check mark next to Roxygen documentation. Also click on Configure, and add check mark to everything.

Documentation with roxygen

Go to soft function you created, place a cursor anywhere, and then go to Code− >Insert Roxygen Skeleton

- Roxygen comments star with #', and come before the function
- 1st sentence is a user-readable function title
- **@param** automaticall catch the input of the function. You will need to eventually provide description for each input
- **@return** here you should explain the function output
- **@examples** here you should provide example of the function use (**required** by CRAN for all functions that are made available to the user)

Documentation with roxygen

Try to reinstall you new package (without messing with roxygen).
Do you see function soft now?

Documentation with roxygen

Try to reinstall you new package (without messing with roxygen). Do you see function soft now?

Go to **man** folder. Now you can see **soft.Rd** file. Open that file in Rstudio and hit Preview.

Go to **NAMESPACE** file. What do you see?

Why roxygen?

Karl Broman:

> *With Roxygen2, you write specially-structured comments preceding each function definition. These are processed to produce the .Rd files that R wants, and it also creates that painful NAMESPACE file for you.*

Documentation with roxygen

Let's give some simple description to soft function within roxygen documentation.

Documentation with roxygen

You can (and often should) add more to the documentation than required by the skeleton.

See **R packages** for some examples and also **R package primer**

Basics of testing

The **testthat** package can be used to create an automatic testing directory for any of your packages.

usethis makes it easy to add testing to your package

```
library(usethis)
use_testthat()
use_test()
```

What happened?

Basics of testing - creating a test

```
library(usethis)
use_testthat()
use_test()
```

- The first command created a testing infrastructure.
- The second command created a particular test skeleton for our soft function.

Modify the tests to create 2 test for soft-thresholding function.

Run the tests using Build− >More− > Test package

Package dependencies

If you package needs other packages to work property, i.e. **glmnet**, **ggplot2**, you should use DESCRIPTION file

Imports: glmnet, ggplot2

Some older packages have **Depends** but according to Hadley one should not need this unless one really knows why.

Specifying dependancies in DESCRIPTION via Imports will automatically result in installation of those packages with yours.

Package dependencies

The packages listed in Imports will be **installed**, but not necessarily **attached**.

What this means? Use glmnet::glmnet(),
i.e. pck_name::function_name() when using outside functions

Getting it on GITHUB - reminder

You may want to make your package available to the public via GitHub.

devtools R package will allow others to seemlessly install any R package that is on GitHub.

Reminder: From local to GitHub

- Commit the work we have done, if not already.
- Setup a new repository on GitHub **without readme**
- Use

```
git remote add origin [URL path to created repo on GitHub]
git push --set-upstream origin master
```