R programming: packages

2025-03-21

M1 MIDS/MFA/LOGOS
Université Paris Cité
Année 2024
Course Homepage
Moodle

This lab is just an incentive to dig into R Packages (2e) by Wickham and Bryan

Introduction

Sharing/Storing/Organizing

Packages come with conventions

Packaging can be helpful to perform data analysis.

Setup

Automating package development is facilitated by a collection of packages.

```
stopifnot(
  require("devtools"),
  require("usethis"),
  require("testthat"),
  require("styler"),
  require("roxygen2")
)
```

`devtools` is the cornerstone of the collections

- with `rstudio` (hands in hands)
- with `vs code`
- with `positron`

Package organization

Make sure you understand the different states of a package : *source*, *bundled*, *binary*, *installed*, *in-memory*.

- At first, we deal with a *source* package. Where is it hosted?
- How is the source organized?
- What are metadata for?
- What is the purpose of file `.Rbuildignore`?
- Why do we bundle packages?

-

Structure and states of a package

To initialize a package development directory.

```
packpath <- "hmw2.ma7by020"

if (!fs::dir_exists(packpath)){

  fs::dir_create(packpath)
}

usethis::create_package(packpath)
```

Within directory `hmw2.ma7by020`, you should have the next organization

```
hmw2.ma7by020/
   DESCRIPTION
   .gitignore
   hmw2.ma7by020.Rproj
   NAMESPACE
   R
   .Rbuildignore

2 directories, 5 files
```

> What is the difference between *loading* and *attaching* a package?

You will have to

- populate the `R` subdirectory
- update `DESCRIPTION`

Reusing scripts

Styling

Testing

The workhorse of the development process is

```
devtools::load_all()
```

1. Code and/or fix bugs
2. Load the code
3. Test the code

Checking Package state

```
devtools::check()
```

`NAMESPACE` and dependencies

Versioning (git)

References

Writing R extensions