"Build reproducible and shareable data analyses using R packages

Conférence AFH 2022

Sébastien Rochette, Florence Mounier

Material of this course is on Github: statnmap/teach-package-dev-rmdfirst

One step package building

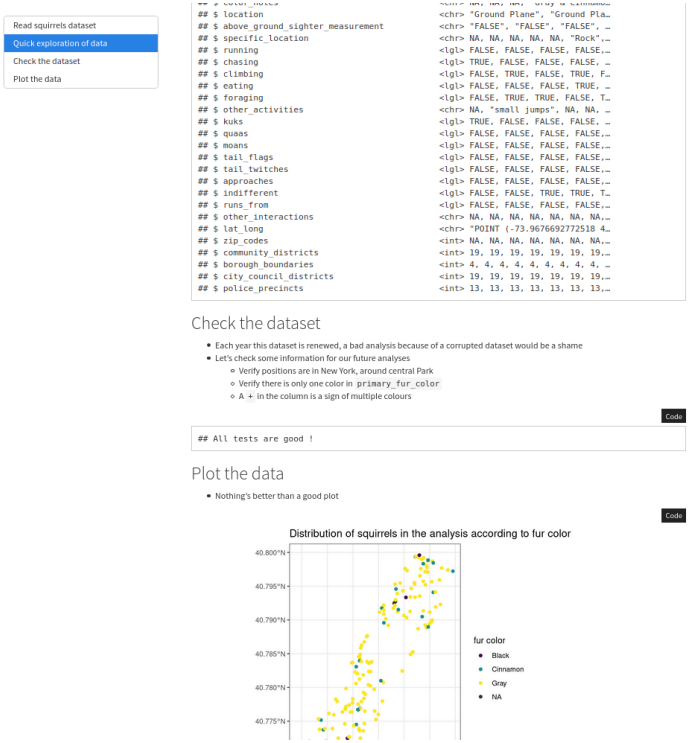
From Rmd to package

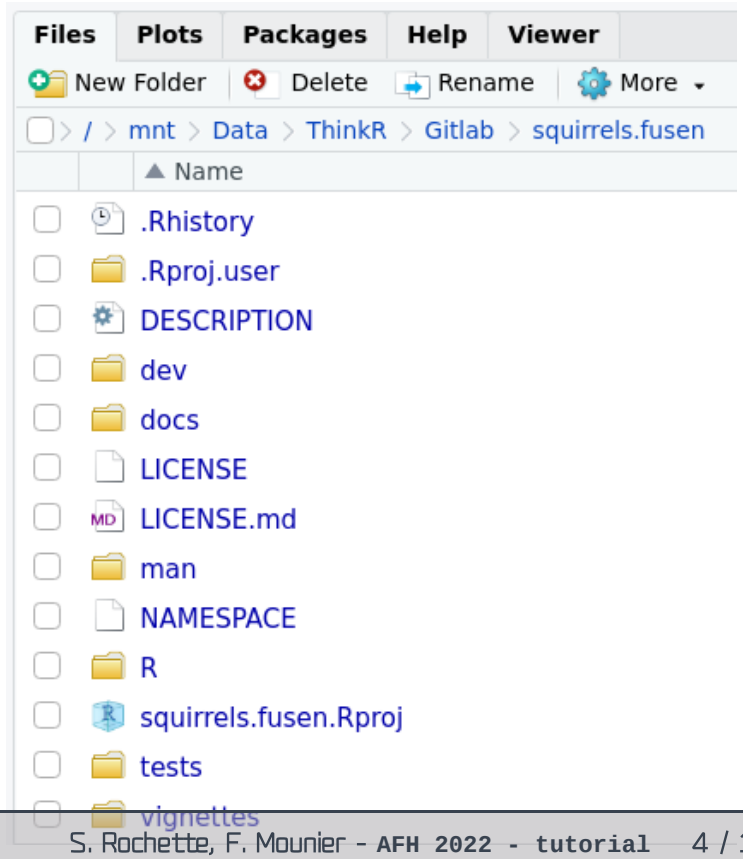
Learning goals

- Understand the methodology proposed by the "Rmd First" method
- Be able to refactor a code into correctly formatted functions
- Understand the structure of a package
- Be able to build a documented and tested R package

Building a package in a few steps

- Start with a Rmd
- Build your functions inside
- Document your functions
- Inflate as a Package





How this tutorial will be held

- E-learning platform: https://bakacode.io
- Instructors speak on slides
- Quizz where every attendees will be able to participate
- Direct questions where attendees are asked to participate
- Hands-on parts, in breakout rooms, where attendees are asked to share their screens for instructors to help
- A 5 minutes break every hour
- We stop 15 minutes before the end to say goodbye

Code of Conduct

All conference participants agree to:

- Be considerate in language and actions, and respect the boundaries of fellow participants.
- Refrain from demeaning, discriminatory, or harassing behaviour and language.
- Alert a member of the ThinkR team if you notice someone in distress, or observe violations of this code of conduct, even if they seem inconsequential.

What is Bakacode?

The ThinkR e-learning platform

Take a tour!

BakACode is the home-made ThinkR e-learning platform

- Everything you need during the course is available in one place
- The server has all dependencies required for your tutorial
- You can communicate directly with your instructors

Please take this tour to get used to the platform

Take the tour and use it to answer the quiz at the end of the chapter

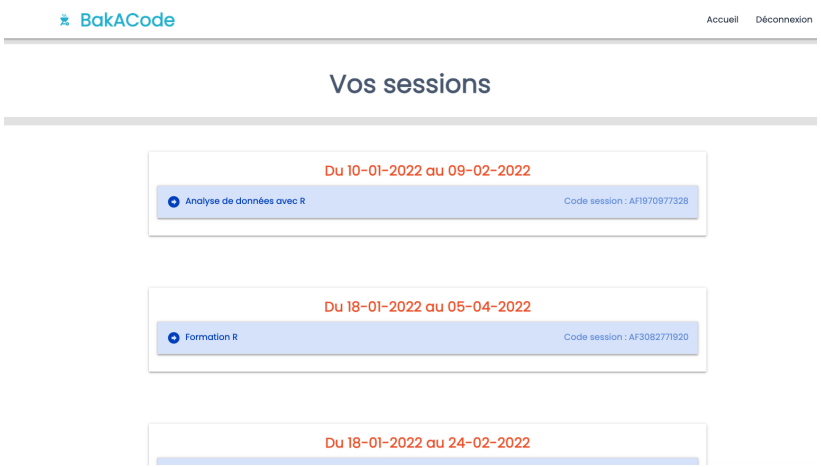
- The instructor creates a file named "hello.Rmd" in the "shared/" directory
  - You will need it for the quiz

Now we let you 3 minutes to read the following slides and answer the quiz at the end of this chapter

Connection

- Use your credentials to connect with username and password

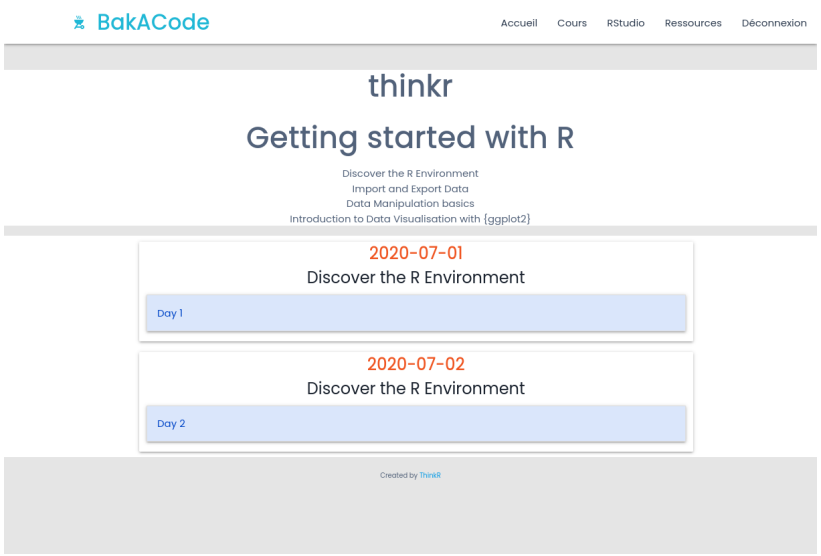
Connect to https://bakacode.io



Sessions overview

- View your current and past training sessions

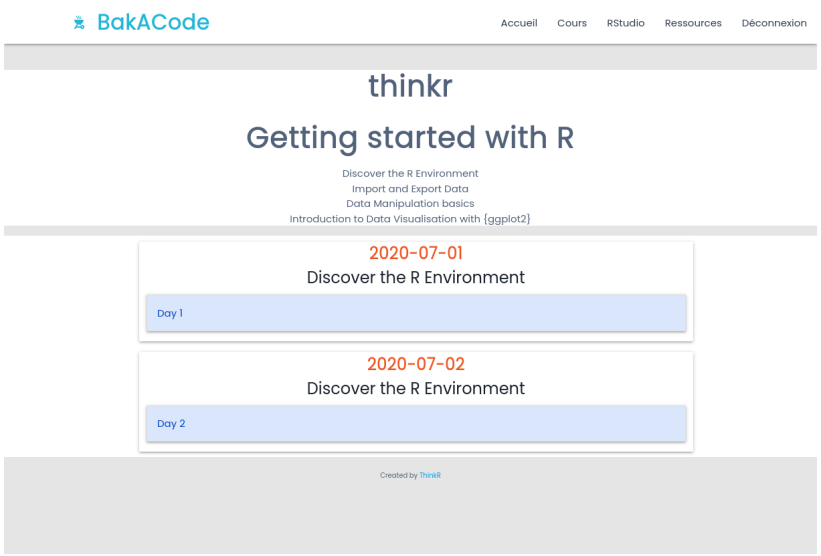
Home page



Learning objectives

- Learning objectives of the tutorial

Home page



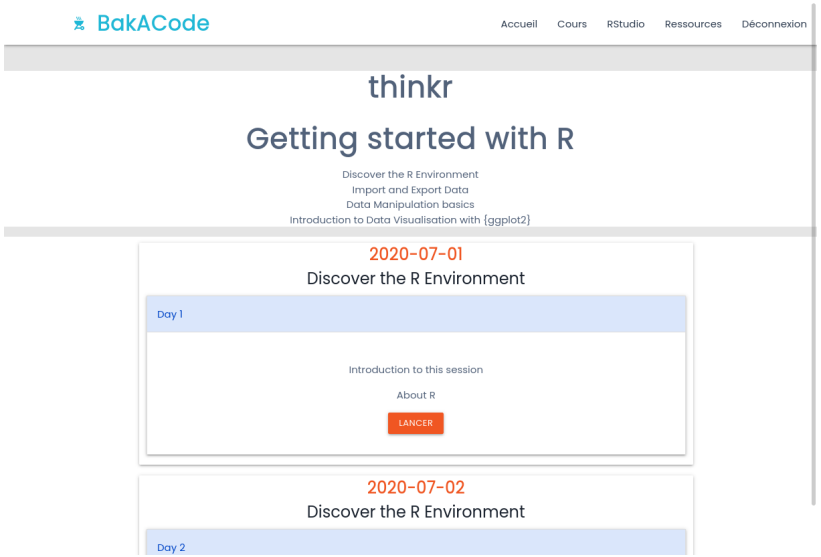
Learning objectives

- Learning objectives of the tutorial

Sections

- A box for each section
- Date and description of the section
- Click on the box to show chapters presented

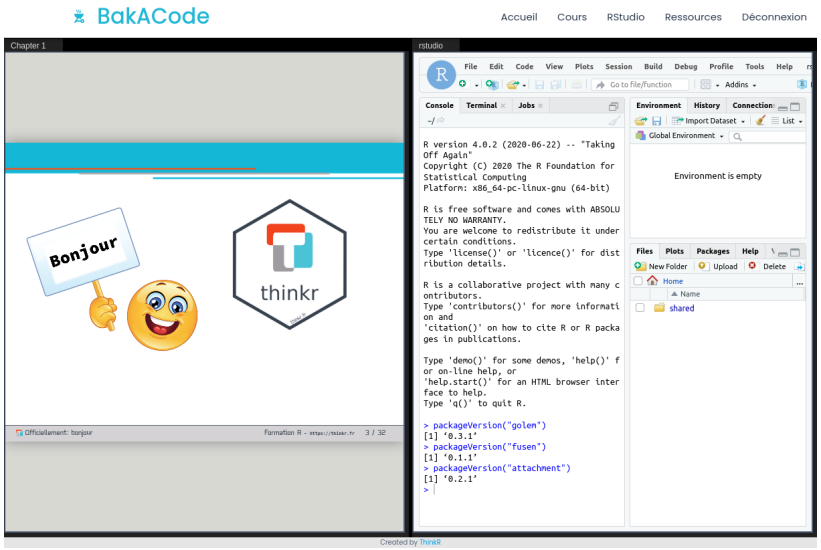
Home page - launch



Launch tutorial

- Click on the Launch ("Lancer") button to start the tutorial

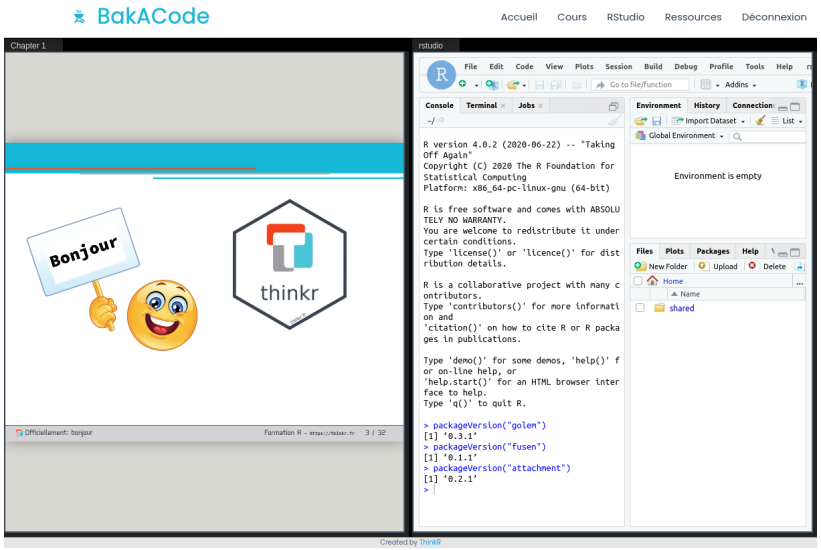
Pratice - presentation



Vertical separator

- Maintain left click on the vertical bar and use the mouse to change left/right windows ratio

Pratice - presentation



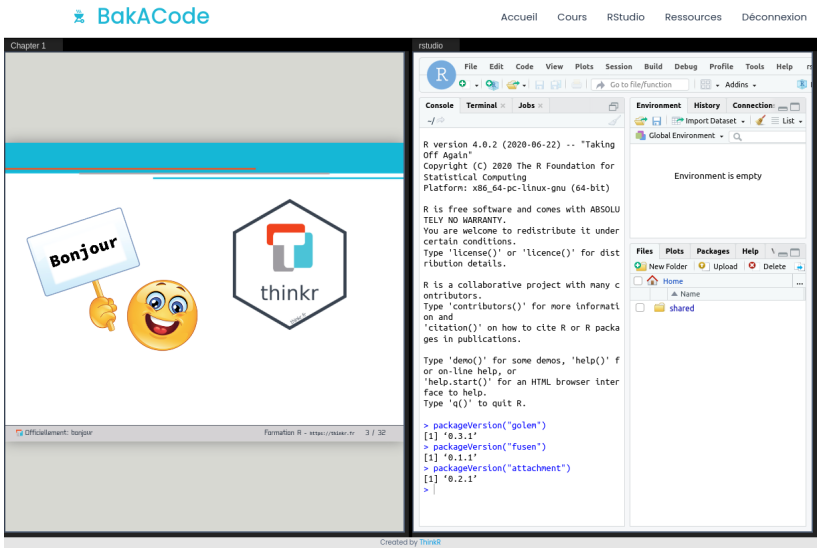
Vertical separator

- Maintain left click on the vertical bar and use the mouse to change left/right windows ratio

Slides

- Use arrows of your keyboard to go up and down in the slides
- You can copy-paste code parts

Pratice - presentation



Vertical separator

- Maintain left click on the vertical bar and use the mouse to change left/right windows ratio

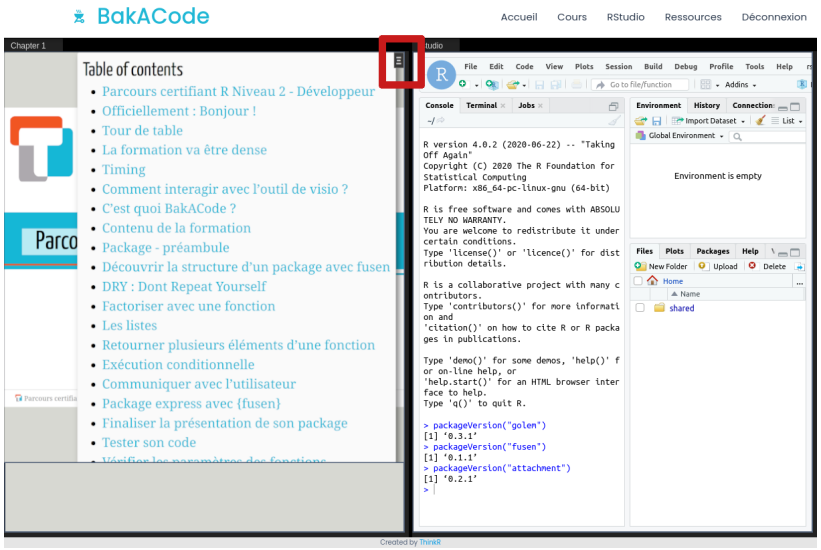
Slides

- Use arrows of your keyboard to go up and down
- You can copy-paste code parts

RStudio server

- RStudio project opened with your data and Rmd files for exercises
- Be sure to work in a project!

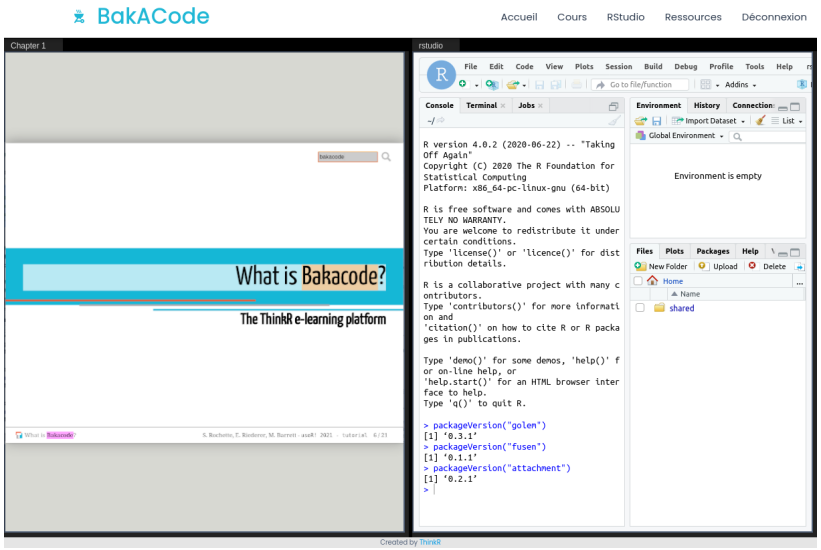
Pratice - search



Search chapter

- Click on the slides
- Click on the black box in the top right corner to open the table of content
- Choose your chapter
- Click on the black box to quit the table of content and navigate

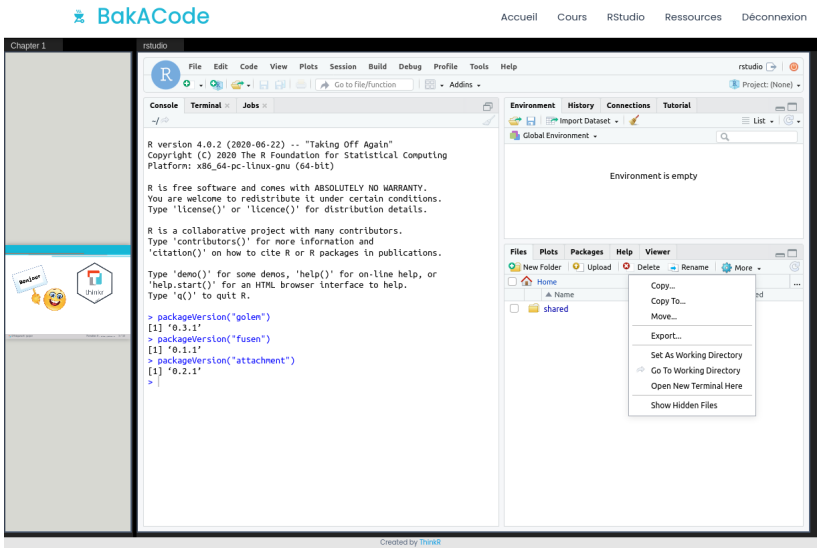
Pratice - search



Search words

- Click on the slides
- Use CTRL + F to open the slides search bar or click on the magnifier icon
- Write your word
- Press Enter to find the next occurence of your word
- Use ESC / ECH to quit the search bar and navigate

Pratice - export

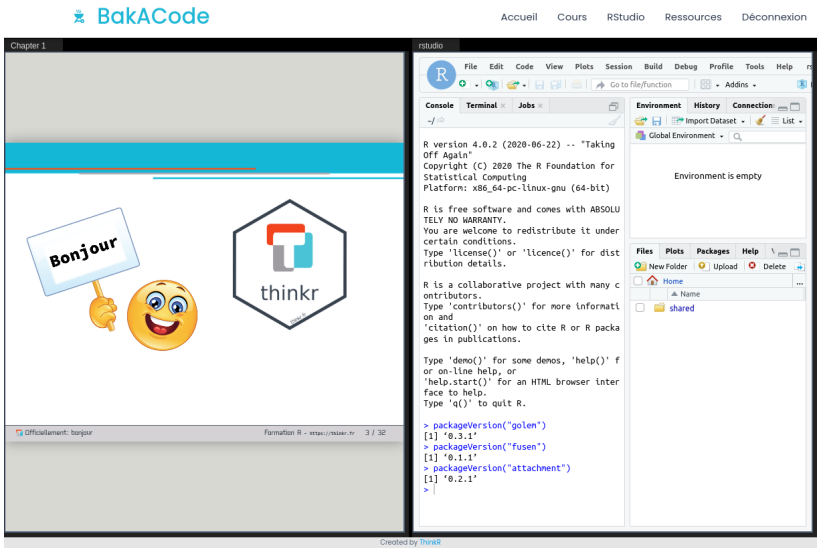


Download your files

- Download files on your computer
- Select one or more files in the file Pane using checkboxes
- Open the 'More' menu
- Choose 'Export'
- 'Download'

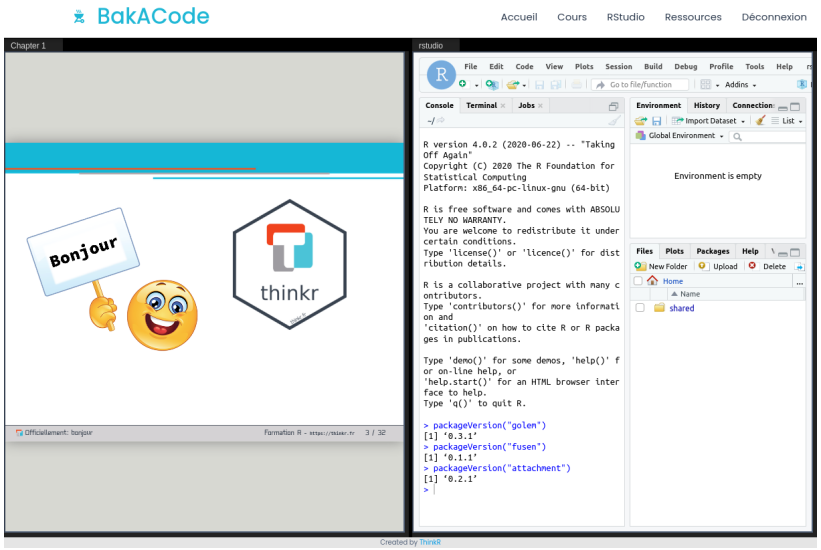
> We recommend that you export your full project at the end of the course

Top menu



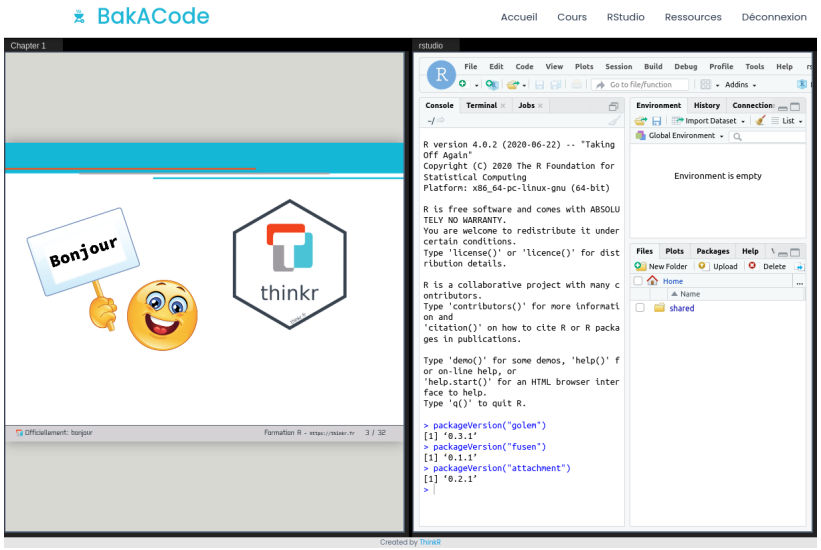
Home

- Go back to home page "Accueil"



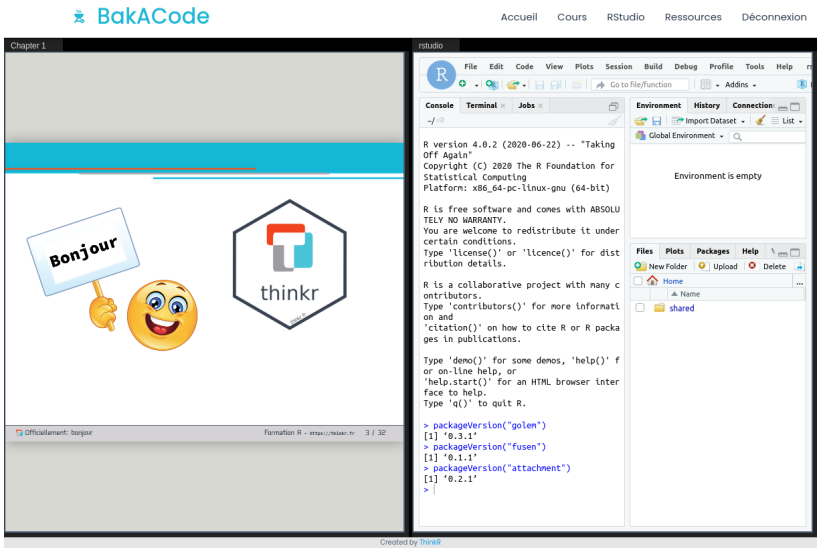
Courses

- Navigate through the list of courses sections available



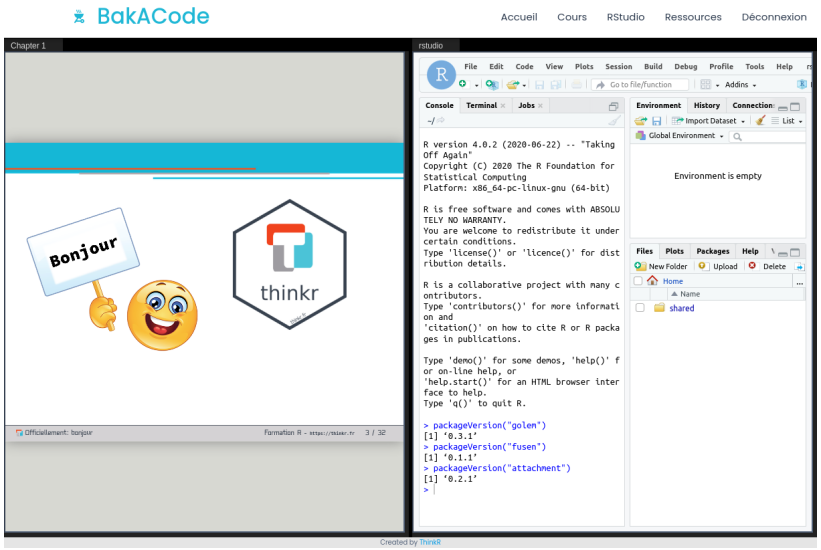
RStudio

- Opens your current RStudio session as full screen in a new tab
- Note that this will close the RStudio session in the "Practice" interface



Resources

- Opens a new tab with R cheatsheets as PDF
- You can open or download them



Disconnect

- Disconnect from BakACode

Quiz

- Can you download the pdf of the courses?
  - It is stored in the "pdf/" folder, in the Home directory
  - If you loose the connection, at least you have the slides
- Did you find the "shared/hello.Rmd" file ?
  - You can open it and write your name in it.
  - Save it quickly as it is shared with other attendees!
- Can you use the table of content or the search bar to retrieve my email somewhere at the end of the slides?
  - Write it in the chat in private, and keep it for yourself, just in case

Start with the documentation

What are vignettes?

Do you know package {attachment}?

Let's say you don't!

How to discover what does a package?

How to discover what does a package?

My questions as a user

- What does it do?
- How to install it with its dependencies?
- What are its function?
- How to fill parameters of this function?
- Can I have an example on how to use this function?
- Can I have an example on how to use the package as a whole?
- Will it work with the last version of R and dependencies?

How to discover what does a package?

What does it do?

- CRAN page: https://cran.r-project.org/web/packages/attachment/index.html

`attachment: Deal with Dependencies`

Tools to help manage dependencies during package development. This can retrieve all dependencies that are used in R files in the "R" directory, in Rmd files in "vignettes" directory and in 'roxygen2' documentation of functions. There is a function to update the Description file of your package and a function to create a file with the R commands to install all dependencies of your package. All functions to retrieve dependencies of R scripts and Rmd files can be used independently of a package development.

Version:	0.2.1
Depends:	R (≥ 3.4)
Imports:	desc (≥ 1.2.0), glue (≥ 1.3.0), knitr (≥ 1.20), magrittr (≥ 1.5), rmarkdown (≥ 1.10), roxygen2, stats, stringr (≥ 1.3.1), utils
Suggests:	testthat (≥ 2.1.0)
Published:	2021-01-21
Author:	Sébastien Rochette [cre, aut], Vincent Guyader [aut] (previous maintainer), ThinkR [cph, fnd]
Maintainer:	Sébastien Rochette <sebastien at thinkr.fr>
BugReports:	https://github.com/Thinkr-open/attachment/issues
License:	GPL-3
URL:	https://thinkr-open.github.io/attachment/, https://github.com/Thinkr-open/attachment
NeedsCompilation:	no
Materials:	README NEWS
CRAN checks:	attachment results

`Downloads:`

Reference manual:	attachment.pdf
Vignettes:	Extract scripts dependencies and generate your Description file
Package source:	attachment_0.2.1.tar.gz
Windows binaries:	r-devel: attachment_0.2.1.zip, r-release: attachment_0.2.1.zip, r-oldrel: attachment_0.2.1.zip
macOS binaries:	r-release (arm64): attachment_0.2.1.tgz, r-release (x86_64): attachment_0.2.1.tgz, r-oldrel: attachment_0.2.1.tgz
Old sources:	attachment archive

`Linking:`

Please use the canonical form `https://CRAN.R-project.org/package=attachment` to link to this page.

How to discover what does a package?

How to install it with its dependencies?

- `install.packages('attachment')`

How to discover what does a package?

What are its functions?

- Index of the package

Deal with Dependencies 

Documentation for package ‘attachment’ version 0.2.1

- DESCRIPTION file.

Help Pages

attachment-deprecated	Deprecated functions
att_amend_desc	Amend DESCRIPTION with dependencies read from package code parsing
att_from_description	Return all package dependencies from current package
att_from_namespace	return package dependencies from NAMESPACE file
att_from_rmd	Get all dependencies from a Rmd file
att_from_rmds	Get all packages called in vignettes folder
att_from_rscript	Look for functions called with '::' and library/requires in one script
att_from_rscripts	Look for functions called with '::' and library/requires in folder of scripts
att_to_description	Deprecated functions
att_to_desc_from_is	Amend DESCRIPTION with dependencies from imports and suggests package list
att_to_desc_from_pkg	Amend DESCRIPTION with dependencies read from package code parsing
create_dependencies_file	Create a dependencies.R in the 'inst' folder
extract_pkg_info	Internal. Core of find_remotes separated for unit tests
find_remotes	Proposes values for Remotes field for DESCRIPTION file based on your installation
install_from_description	Install missing package from DESCRIPTION
install_if_missing	install packages if missing

How to discover what does a package?

How to fill parameters of this function?

- `?att_amend_desc`

att_amend_desc {attachment} R Documentation

Amend DESCRIPTION with dependencies read from package code parsing

Description

Amend package DESCRIPTION file with the list of dependencies extracted from R, tests, vignettes files.
att_to_desc_from_pkg() is an alias of att_amend_desc(), for the correspondence with att_to_desc_from_is.

Usage

```
att_amend_desc(
  path = ".",
  path.n = "NAMESPACE",
  path.d = "DESCRIPTION",
  dir.r = "R",
  dir.v = "vignettes",
  dir.t = "tests",
  extra.suggests = NULL,
  pkg_ignore = NULL,
  document = TRUE,
  normalize = TRUE,
  inside_rmd = FALSE
)

att_to_desc_from_pkg(
  path = ".",
  path.n = "NAMESPACE",
  path.d = "DESCRIPTION",
  dir.r = "R",
  dir.v = "vignettes",
  dir.t = "tests",
  extra.suggests = NULL,
  pkg_ignore = NULL,
  document = TRUE,
```

How to discover what does a package?

Can I have an example on how to use this function?

- `?att_amend_desc` => Examples

`document`	Run function roxygenise of roxygen2 package
`normalize`	Logical. Whether to normalize the DESCRIPTION file. See `desc_normalize`
`inside_rmd`	Logical. Whether function is run inside a Rmd, in case this must be executed in an external R session

Value

Update DESCRIPTION file.

Examples

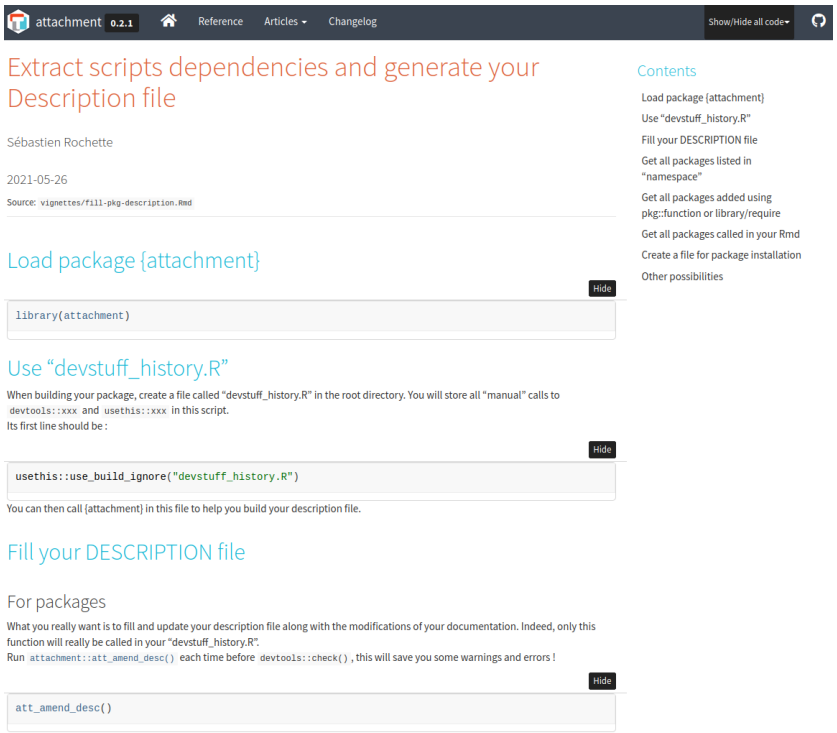
```
tmpdir <- tempdir()
file.copy(system.file("dummypackage",package = "attachment"), tmpdir,
 recursive = TRUE)
dummypackage <- file.path(tmpdir, "dummypackage")
# browseURL(dummypackage)
att_amend_desc(path = dummypackage)
```

[Package *attachment* version 0.2.1 Index]

How to discover what does a package?

Can I have an overview on how to use the package as a whole?

- Vignettes
- GitHub: https://thinkr-open.github.io/attachment/articles/fill-pkg-description.html



How to discover what does a package?

Will it work with the last version of R and dependencies?

- README Check, https://github.com/ThinkR-open/attachment
- Unit tests, code coverage, Continuous Integration

README.md

R-CMD-check passing | codecov 80% | CRAN 0.2.1 | downloads 1332/month

attachment

The goal of attachment is to help to deal with package dependencies during package development. It also gives useful tools to install or list missing packages used inside Rscripts or Rmds.

When building a package, we have to add `@importFrom` in our documentation or `pkg::fun` in the R code. The most important is not to forget to add the list of dependencies in the "Imports" or "Suggests" package lists in the DESCRIPTION file.

Why do you have to repeat twice the same thing ?
And what happens when you remove a dependency for one of your functions ? Do you really want to run a "Find in files" to verify that you do not need this package anymore ?

Let {attachment} help you ! This reads your NAMESPACE, your functions in R directory and your vignettes, then update the DESCRIPTION file accordingly. Are you ready to be lazy ?

See full documentation realized using {pkgdown} at https://thinkr-open.github.io/attachment/

How to discover what does a package?

My answers as a user

Questions	Answers
What does it do?	CRAN page
How to install it with its dependencies?	`install.packages('attachment')`
What are its functions?	`?attachment` => Index
How to fill parameters of this function?	`?att_amend_desc`
Can I have an example on how to use this function?	`?att_amend_desc` => Examples
Can I have an overview on how to use the package as a whole?	Vignettes, GitHub
Will it work with the last version of R and dependencies?	README Check

There is a dedicated website that gathers all these answers: https://thinkr-open.github.io/attachment/

How to discover what does a package?

Developers point of view

- All these sources of information are adressed by developers

If you start with the documentation, from the vignette, you will not have to think too much about all of these

Let's talk again about vignettes

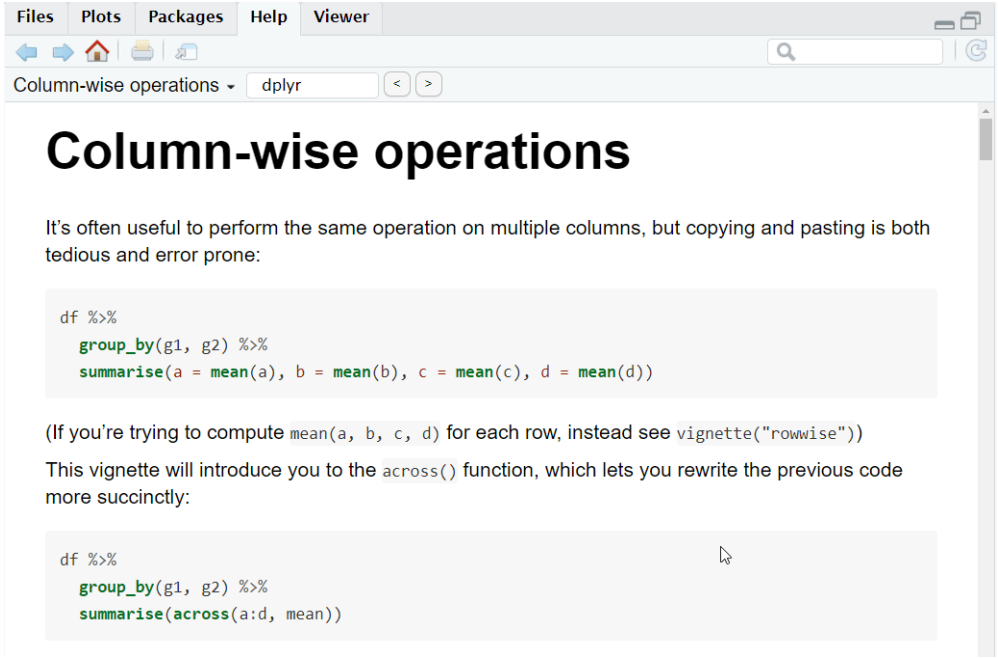
- HTML pages telling the stories of the package
- List vignettes using: `vignette(package = "thepackage")`.

```
Vignettes in package ‘dplyr’:

colwise                 colwise (source, html)
compatibility           dplyr compatibility (source, html)
base                    From base R to dplyr (source, html)
grouping                Grouped data (source, html)
dplyr                   Introduction to dplyr (source, html)
programming             Programming with dplyr (source, html)
rowwise                 rowwise (source, html)
two-table               Two-table verbs (source, html)
window-functions        Window functions (source, html)
```

Let's talk again about vignettes

Content of a vignette

```
vignette(topic = "colwise", package = "dplyr")
```



Let's talk again about vignettes

Vignette = Rmarkdown (usually)

- What is the story of the package
- What is the function for
- How to use it with or without parameters
- Reproducible examples
- Different outputs depending on chosen parameters

Let's start building our package from its story !

Quizz: What is a vignette?

- A: A website somewhere on Internet
- B: A Sticker you can add on your laptop
- C: A R script with the help of function accessible in the installed package
- D: A html page built from a Rmd file, accessible in the installed package

Quizz: Have you already built a package with all these?

- A: Yes, everything. Functions, examples, tests, vignettes
- B: Only part of documentation. Functions, examples, maybe vignettes
- C: Only functions in a R/ directory
- D: No. I never built a package from scratch

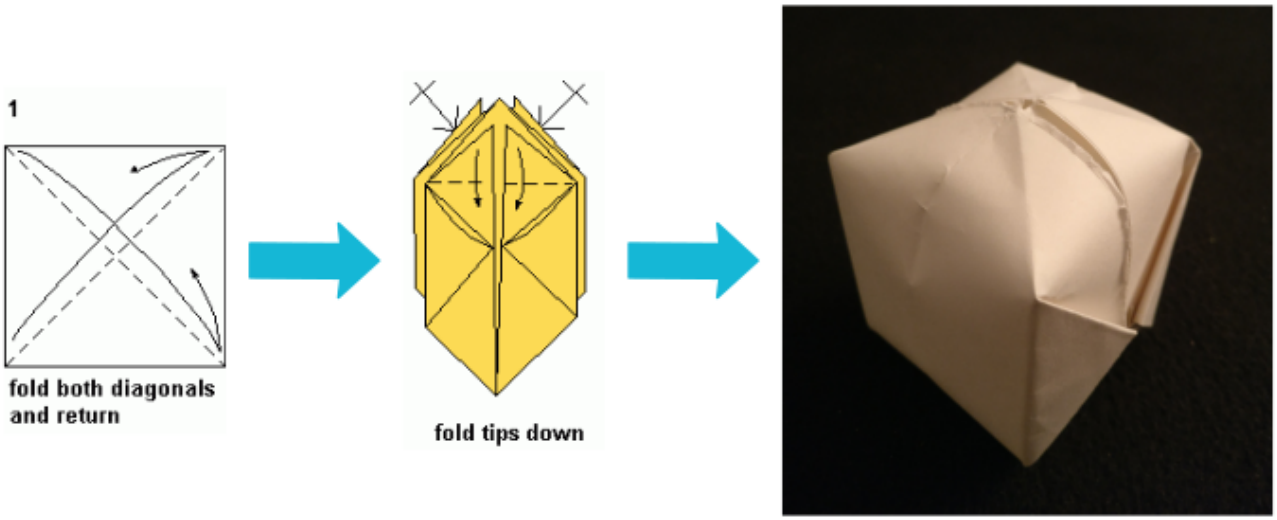
Discover the structure of a package with {fusen}

Where does all of this come from?

{fusen} : adopt "Rmd-first" method

- Start with documentation
- Develop everything in a familiar place: the RMarkdown
- {fusen} inflates the package for you

> What if there was a package that could take an Rmd file, kind of like a sheet of paper, and if you follow the right folding, you can blow it up like a package?



[Short version] - Preamble

- What are the principal components of a package?
- Where do I have to write what?
- How does {fusen} make my life easier?

We suggest you to :

- Watch your trainers create a package {mytools} by following these short steps, without practicing yourself
- Redo this package {mytools} on your own in a new project

We won't explain everything here, but you will see the components of a package and where they fit. For the details, that's the goal of your complete training!

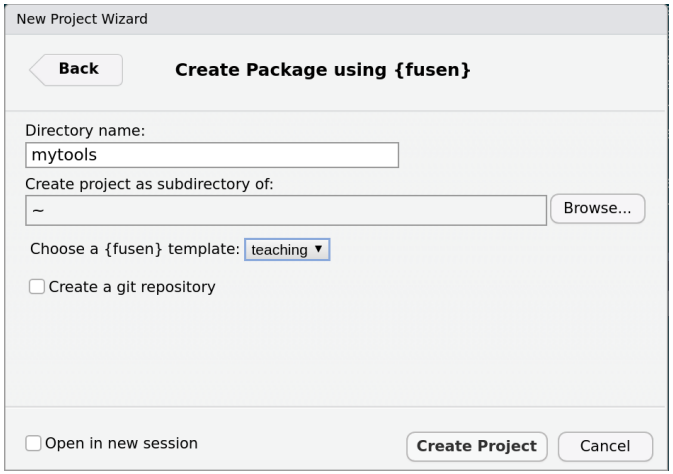
[Short] 1: Create a new {fusen} project

In Rstudio :

- File > New project > New directory > Package using {fusen}
- Choose the name of the package (explicit, in lower case)
  - Name of the package: "mytools"

> *no capital letters, underscores, spaces or special characters*

- Choose the {fusen} template in the dropdown menu: "teaching"
- Choose the directory where to save the project
- Create the project



[Short] 2: Open the {fusen} Rmd flat template

- The project opens up on a flat template file: "flat_teaching.Rmd"
- Here are the main components of a package, in a unique Rmd file
  - Note the name of the chunks that are required
  - The present file is pre-filled with examples

````
<!--
# Description of your package

This will fill the description of your package.
Fill and run the content of this chunk, before anything else.

Note: when you will use other flat templates, this part will be in a separate file. Do not be surprised!
-->

```{r description, eval=FALSE}
# Describe your package
fusen::fill_description(
  pkg = here::here(),
  fields = list(
    Title = "Learn how to build A Package From Rmarkdown file",
    Description = "A Set of tools to understand packages structure. Use Rmarkdown First method to build a package from a defined template. Start your package with documentation. Everything can be set from a Rmarkdown file in your project.",
    `Authors@R` = c(
      person("Sebastien", "Rochette", email = "sebastien@thinkr.fr", role = c("aut", "cre")),
      person(given = "ThinkR", role = "cph")
    )
  )
)
# Define License with use_*_license()
usethis::use_mit_license("Sébastien Rochette")
```
````

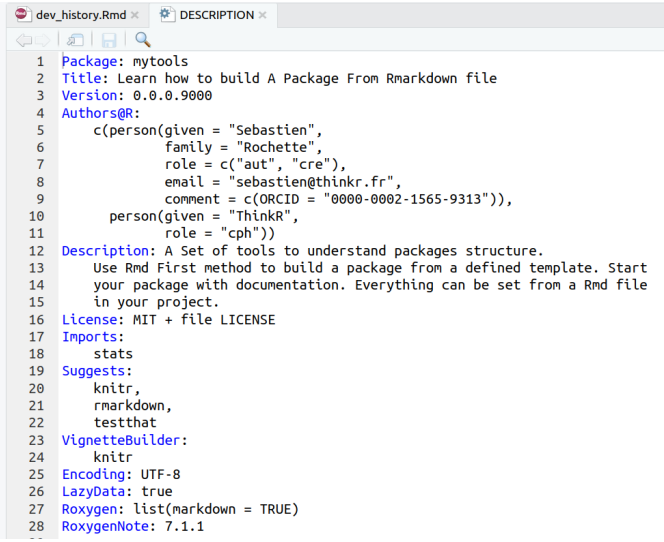
````
# Add one to any value

This is the first tool of our wonderful package.
You can add `1` to any `value` using function `add_one()`.

<!--
This first section shows:

- the three parts necessary for a package: 'function', 'examples' and 'tests'.
  + Note that the three following chunks have names accordingly.

-->

```{r function-add_one}
#' Add one to any value
#'
#' @param value A numeric value
#'
#' @return Numeric. value + 1
#' @export
add_one <- function(value) {
  result <- value + 1
  return(result)
}
```

```{r examples-add_one}
add_one(12)
add_one(56)
```

```{r tests-add_one}
test_that("add_one works", {
  expect_equal(add_one(12), 13)
  expect_equal(add_one(56), 57)
})
```
````

[Short] 3 : Description

Describe your future package:

- Change the name with yours in the `description` chunk
- Execute the complete content of the `description` chunk
  - "CTRL + SHIFT + ENTER" should be good
  - There are two functions to execute here
- A "DESCRIPTION" file appears with the same information

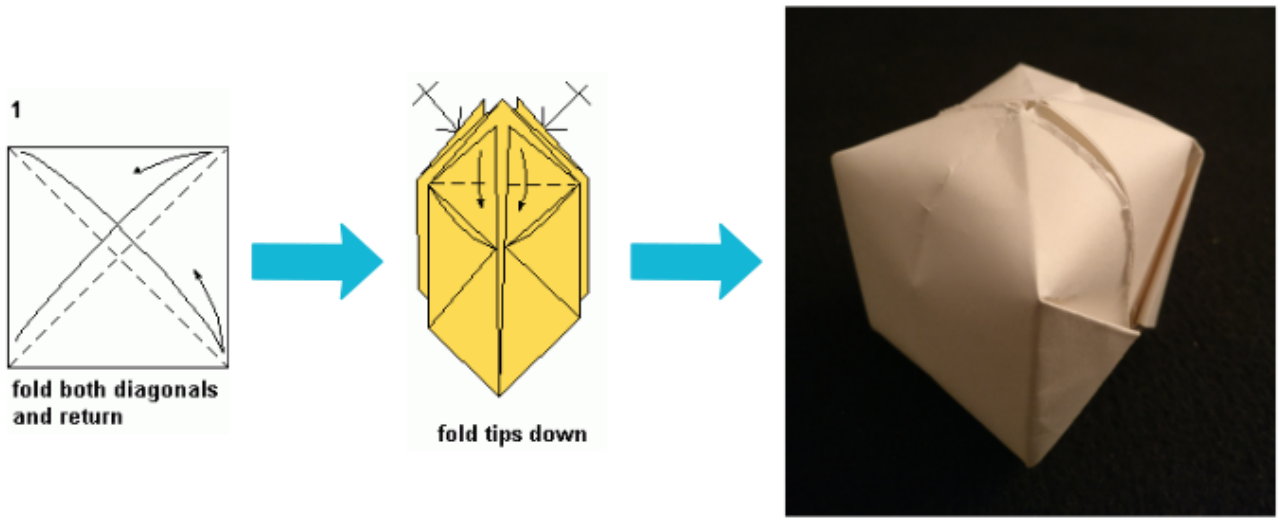
DESCRIPTION is the first source of documentation for your package



[Short] 4 : Inflate the package

- Go down the Rmd file and inflate

```
fusen::inflate(flat_file = "dev/flat_teaching.Rmd")
```



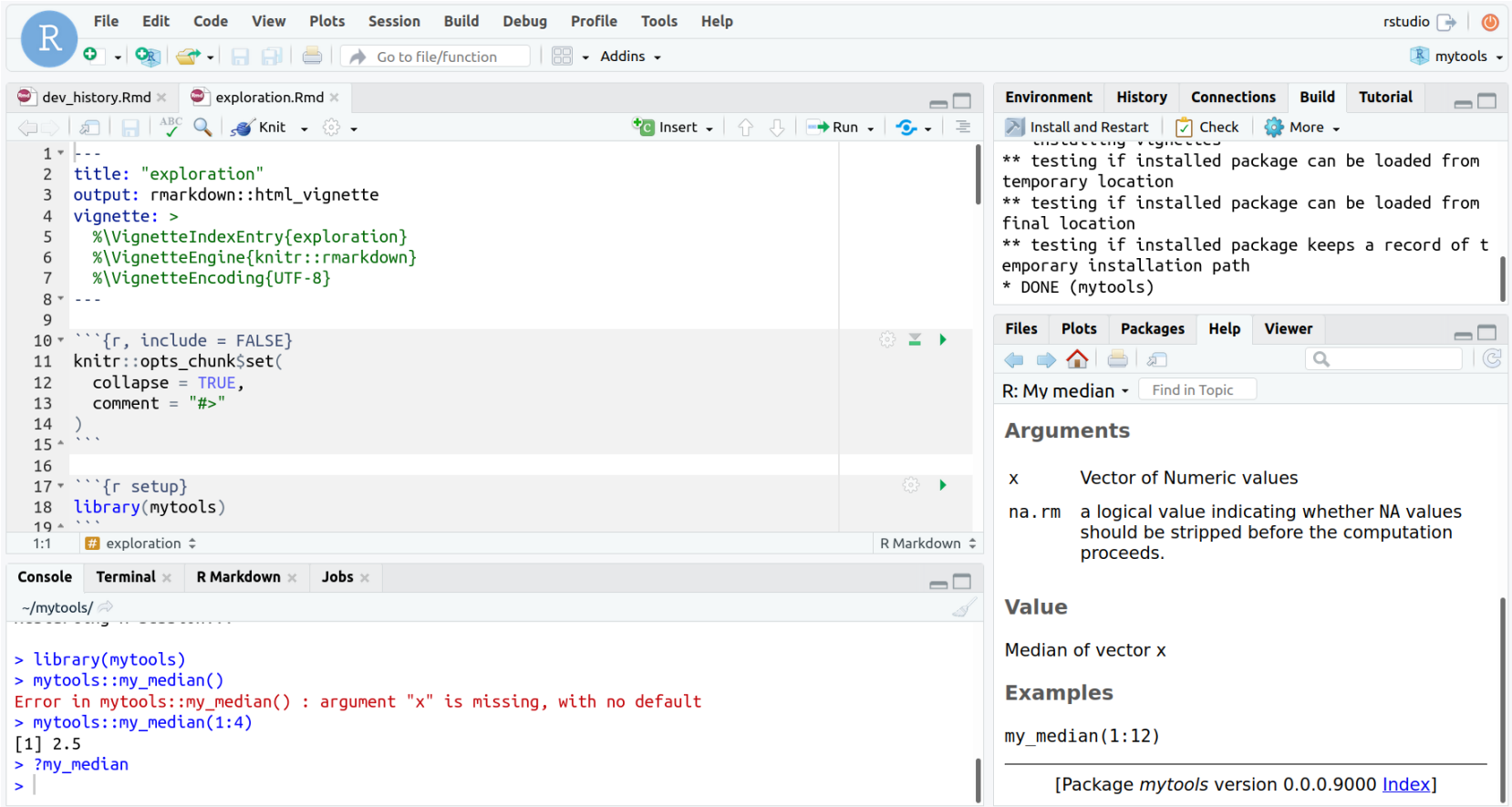
You built a package!

[Short] 5 : Does it work ?

- The 'Build' tab should already appear in RStudio
  - Otherwise, restart your RStudio session
- Install the package
  - Panel Build > Install and Restart
- Test the package directly in the console
  - `mytools::add_one(value = 56)`
- Test the knit of the *vignette*
  - The vignette is opened "get-started.Rmd"
  - Hit the "Knit" button
- Check that the help for your function appears
  - `?add_one`
  - Run the reproducible example from help

[Short] 5 : Does it work ?

If you verified everything listed above, your RStudio should look like this



Your turn

Go back to section "[Short] 1" and execute the steps yourself until this slide

Package express with {fusen}

An Rmd is a package

Preparation: Tools

To create a package, we will use:

- RStudio.
- {fusen} package
- the packages {pkgbuild}, {devtools}, {usethis} and {attachment} to save time.
- the {roxygen2} package to generate the documentation.
- the {testthat} package to generate unit tests.
- Rtools.exe (optional and under windows only).

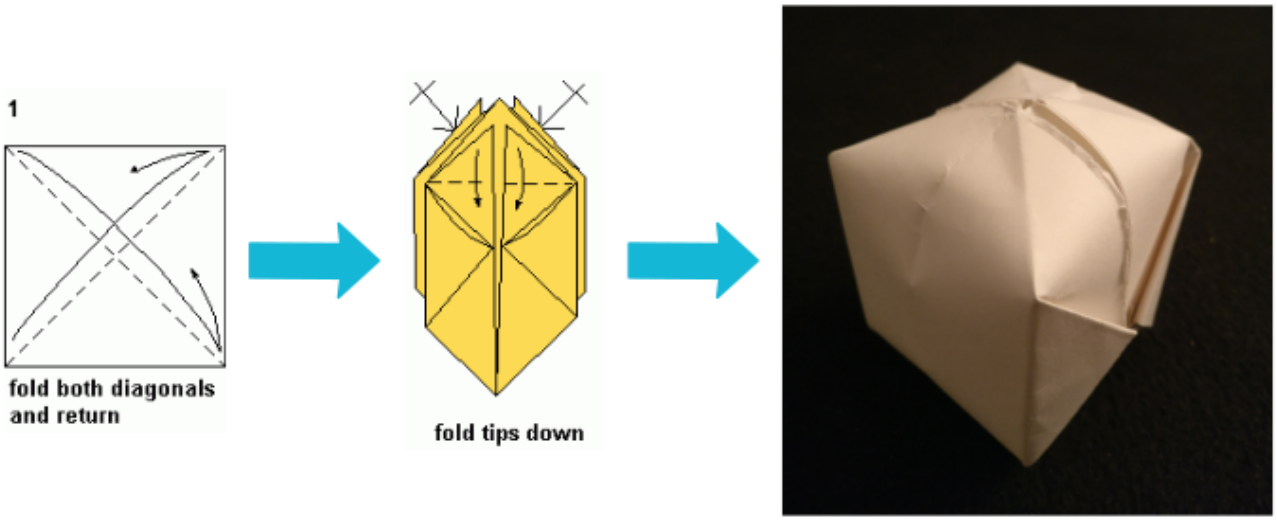
```
install.packages(c("fusen", "devtools", "usethis", "pkgbuild", "roxygen2",
"attachment", "testthat"))
```

Rtools is available here: https://cran.r-project.org/bin/windows/Rtools/
Once (properly) installed `pkgbuild::has_rtools()` should return `TRUE`. Rtools installs everything needed to compile c++ etc.

{fusen}: adopt "Rmd-first" method

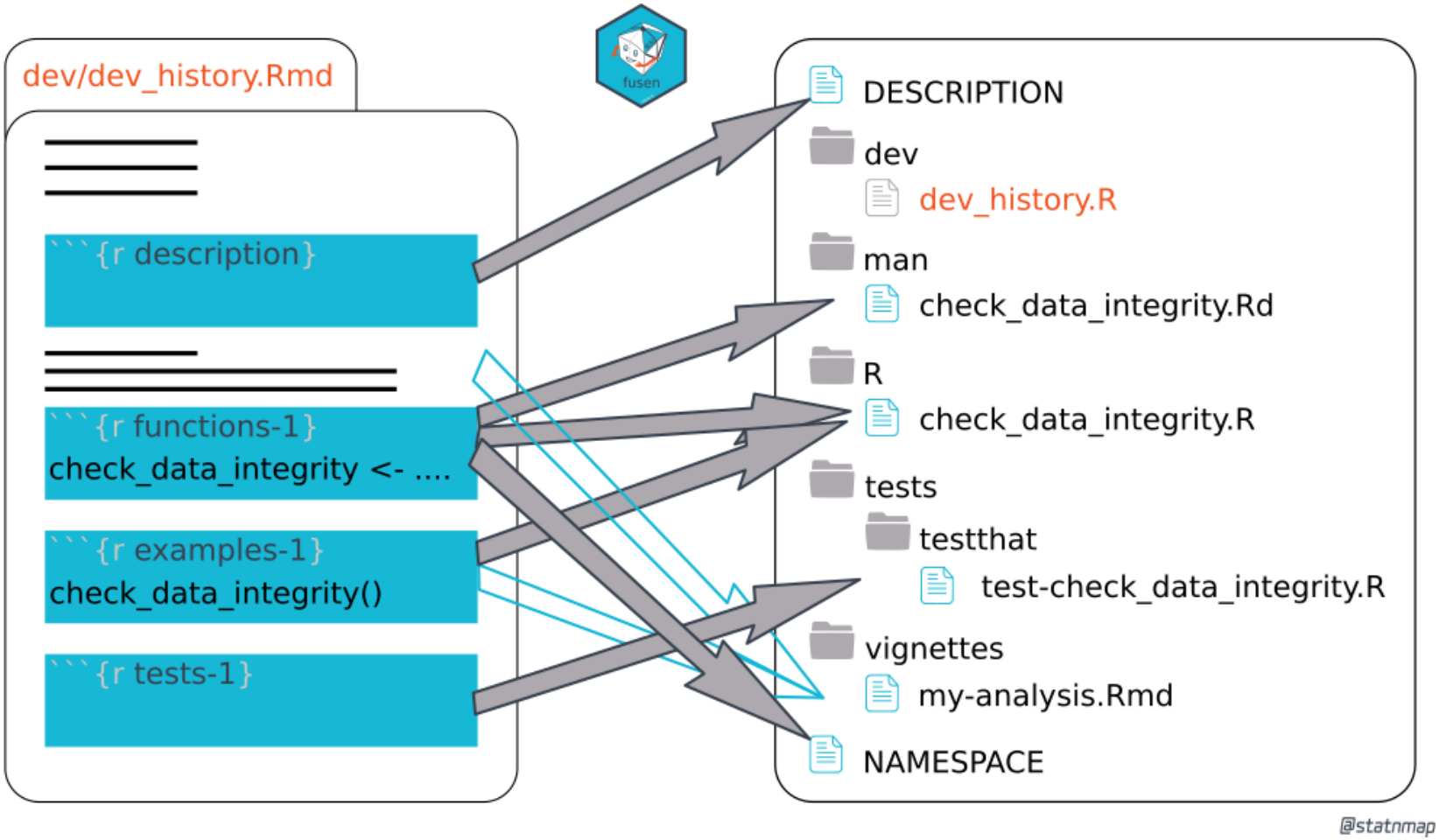
- Start with documentation
- Develop everything in a familiar place: the RMarkdown
- {fusen} inflates the package for you

> What if there was a package that could take an Rmd file, kind of like a sheet of paper, and if you follow the right folding, you can blow it up like a package?



How {fusen} works?

- {fusen} copy-pastes in the right place



Longer version - Preamble

This procedure contains **10 steps**, to be done in order. Some points are not explained in detail to allow you to obtain a functional R package quickly.

We suggest you to:

- Watch your trainers create a package by following these steps, without practicing yourself
- Do this package {hello} on your own in a new project

> Procedure is split in two parts to let you practice. You will thus have 2 times: "Watch - Do"

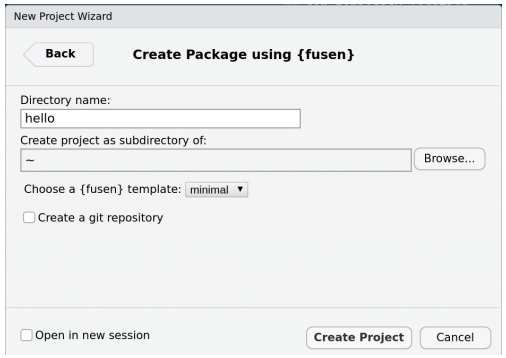
Step 1: Create a new {fusen} project

In Rstudio:

- File > New project > New directory > Package using {fusen}
- Choose the name of the package (explicit, in lower case)
  - Name of the package: "hello"

> *no capital letters, underscores, spaces or special characters*

- Choose the {fusen} template in the dropdown menu: "minimal"
- Choose the directory where to save the project
- Create the project

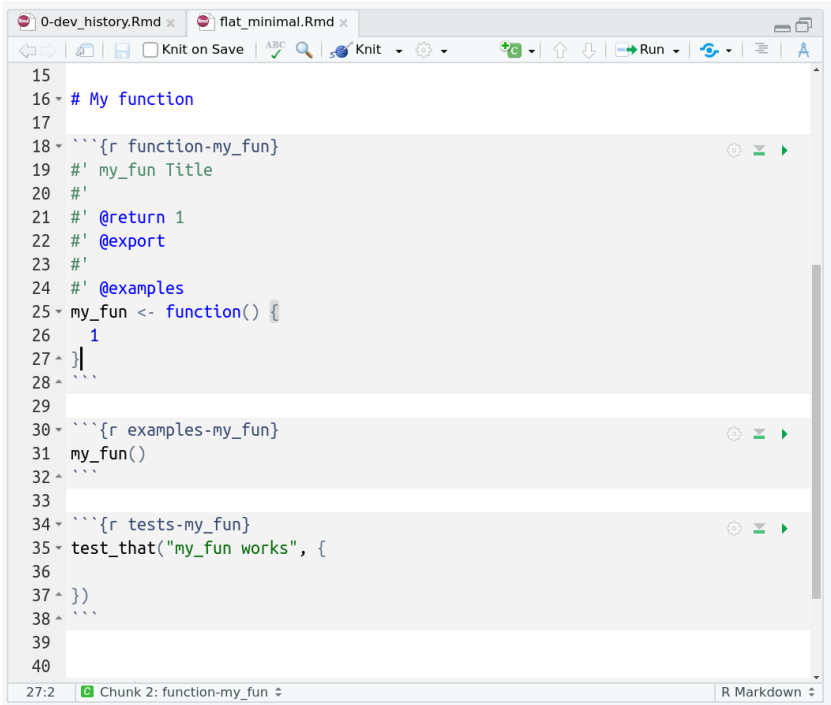


> You could also directly run: `fusen::create_fusen(path = "~/hello", template = "minimal")`

Step 2: Open the {fusen} Rmd templates

This time the template is divided into two flat Rmd files

- "0-dev_history.Rmd" with the general development commands. In particular the `description` part.
- "flat_minimal.Rmd" with the empty skeleton for creating a function like in the previous "teaching" template



Step 3: Description

The description of the package takes place in the first `description` chunk of the "dev_history.Rmd" file

The first fields to fill in:

- Title: "Quick Description of the Goal of your Package" (Title Case, no dot at the end)
- Description: "Long description. Sentence case with a dot at the end of the sentences."
- `Authors@R`: vector of one or more `person()`
  - `person("Sébastien", "Rochette", email = "sebastien@thinkr.fr", role = c("aut", "cre"))`
- License: The choice of the license

Let's run the content of chunk `description`

> Observe the content of file "DESCRIPTION" created

Step 4: Documentation of the package

A package is created to automate some operations. Starting with the documentation forces you to think about the structure of the package and the logical sequence of operations.

- Say what you do in the Rmd text part
- Do what you said in the `function` chunk

Open the "flat_minimal.Rmd" template

Let's run the content of chunk `development`

Step 4: Documentation of the package

Let's see the process of writing a function

````
## Say hello to someone

You can say hello to someone in
particular using `say_hello()`.

```{r development}
library(glue) # On top with others

message("Hello someone")

someone <- "Seb"
message(glue("Hello {someone}"))
```
````

1. Describe in words the first operation that the package will have to solve
2. Define the input data, the possible modifiable parameters, the output result
3. Write the R code to perform these operations in a `development` chunk
4. Call necessary packages in a `development` chunk only, at the very beginning, with other `library()` calls

Step 5: Embed in a function

- Move code in the `function` chunk as soon as you transformed it as a function
- Add examples of use in the `examples` chunk

````
## Say hello to someone

You can say hello to someone in particular using `say_hello()`.

```{r function}
say_hello <- function(someone) {
  message(glue("Hello {someone}"))
}
```

```{r examples}
say_hello(someone = "Seb")
```
````

- Each chunk has a specific task
- Run the chunk `function` to make it available
- Run the function in the `examples` chunk of the Rmd to try it

Step 5: Embed in a function

The created function can now be documented

- Add the doc in {roxygen2} format
  - `@param` to present the content of inputs
  - `@export` for the function to be accessible to the users
  - `@importFrom package function` for functions coming from other packages
  - `@return` to describe the object that comes out of the function
- Use RStudio menu: Code > Insert Roxygen Skeleton

> You must declare the dependencies for the package with `@importFrom`. The calls to `library()` are only there for development, like a classic Rmd, but are not used by the package when inflated

Step 5: Embed in a function

- Your "flat_minimal.Rmd" file should look like this:

````
## Say hello to someone

You can say hello to someone in particular using `say_hello()`.

```{r function}
#' Show a message in the console to say Hello to someone
#'
#' @param someone Character. Name of the person to say hello to
#' @importFrom glue glue
#' @return Used for side effect. Outputs a message in the console
#' @examples
#' @export

say_hello <- function(someone) {
  message(glue("Hello {someone}"))
}
```

```{r examples}
say_hello(someone = "Seb")
```
````

Your turn!

- We are in the half way point of the process
- Got back to Step 1 of this Longer version
- Follow the 5 steps until this slide

Step 6: Write a unit test

- What do you look for when you run your example?
- What makes you say "it works"?

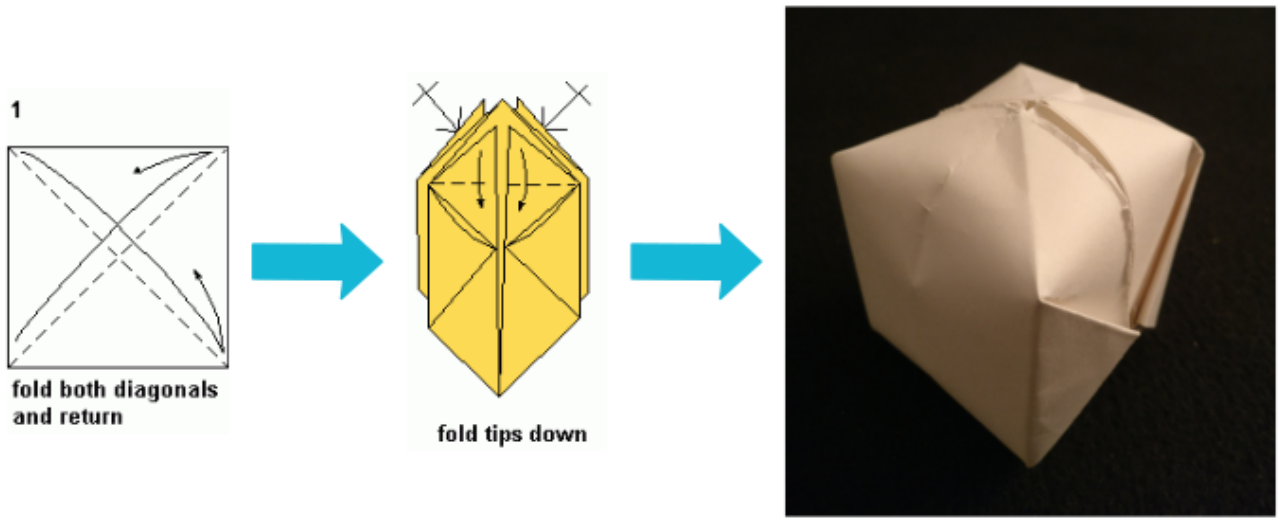
Write your thoughts into code

````
```{r tests}
test_that("say_hello works", {
  expect_message(say_hello(someone = "Seb"), "Hello Seb")
})
```
````

Step 7: Inflate the package

- Let {fusen} inflate the Rmd into a documented and tested package

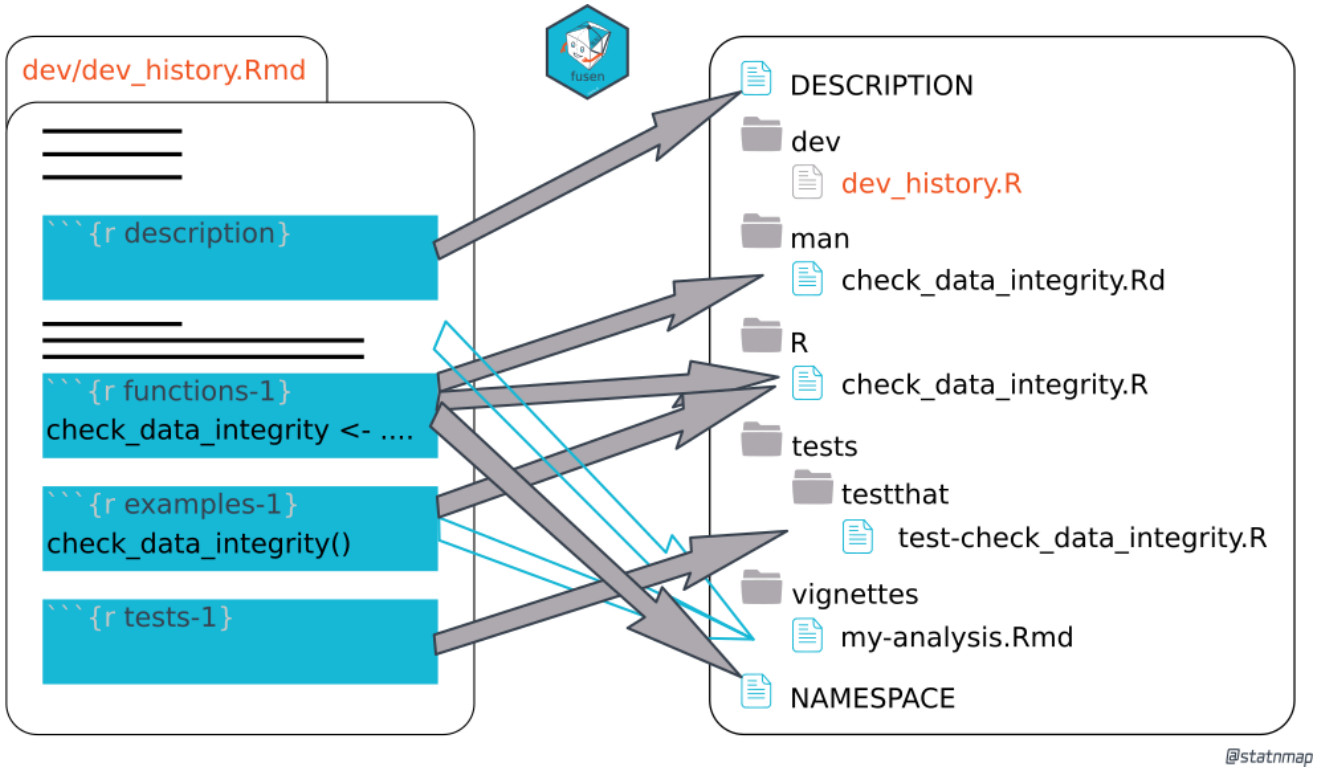
```
fusen::inflate(flat_file = "dev/flat_minimal.Rmd", vignette_name = "Say Hello!")
```



> If there are any errors or warnings, read them carrefuly, address them in the "flat_minimal.Rmd" and inflate the package again.

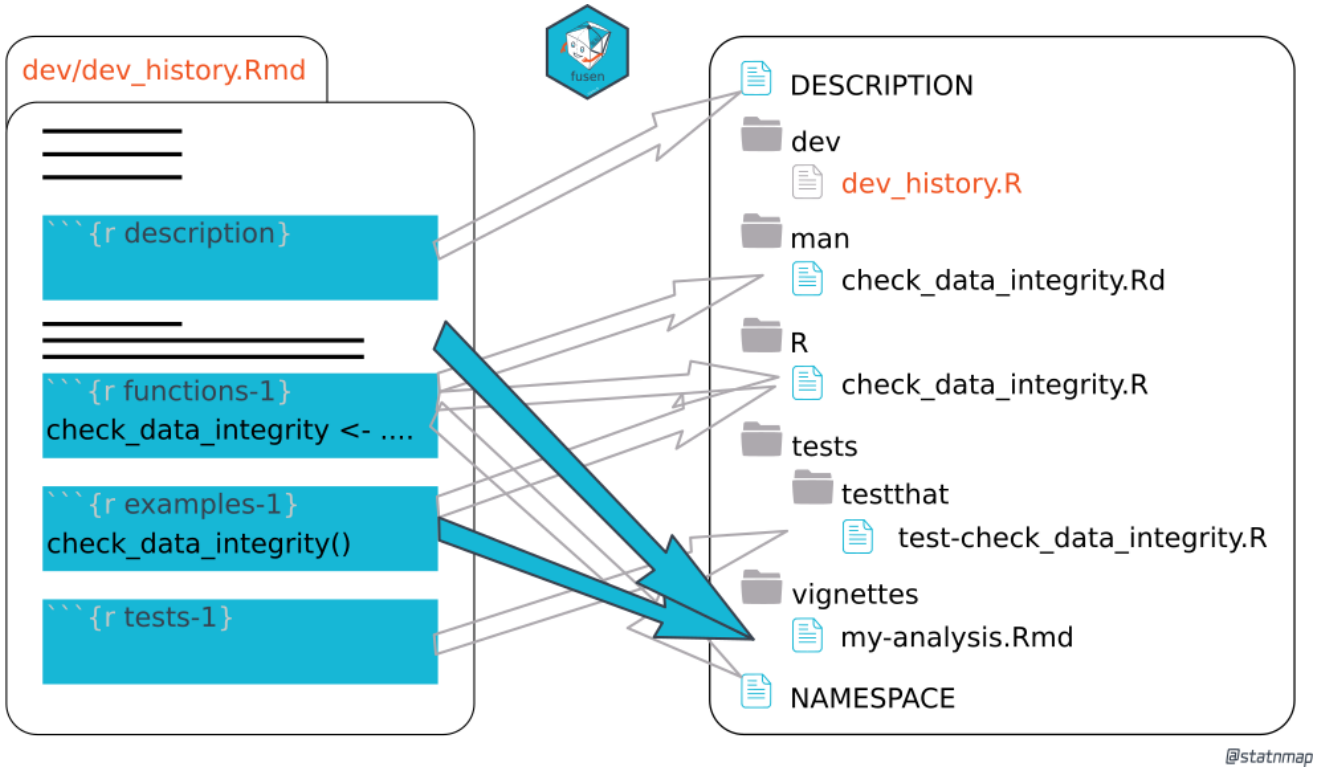
Step 8: Explore created folders and files

- "DESCRIPTION" with dependencies
- "R/" with the function
- "man/" with LateX documentation
- "tests/testthat/" with unit tests
-



Step 8: Explore created folders and files

- "DESCRIPTION" with dependencies
- "R/" with the function
- "man/" with LateX documentation
- "tests/testthat/" with unit tests
- "vignettes/" with documentation



Quiz: Find good definitions

Link files to their description:

Files and folders

1. DESCRIPTION
2. dev_history / flat_minimal
3. vignettes
4. script with roxygen
5. testthat

Documentation

A. Development process for developers

B. Present all the functions of the package and its story

C. How to use each function (for the user)

D. Testing the functions for the developers

E. Content and objectives of the package for all

Possible answers:

- a: ACBDE
- b: EDCBA
- c: EABCD
- d: BAECD

Step 9: Verify again the package

Generate documentation

- `attachment::att_amend_desc()`

Check that the package follows the packages rules

- `devtools::check()`
- Solve potential problems in the "flat_minimal.Rmd"
- Re-inflate the package if necessary
- Reach **0 Error, 0 Warnings, 0 Notes**

> Store this commands in the "0-dev_history.Rmd" file

> Note that `fusen::inflate()` already launches this two commands, but who knows!

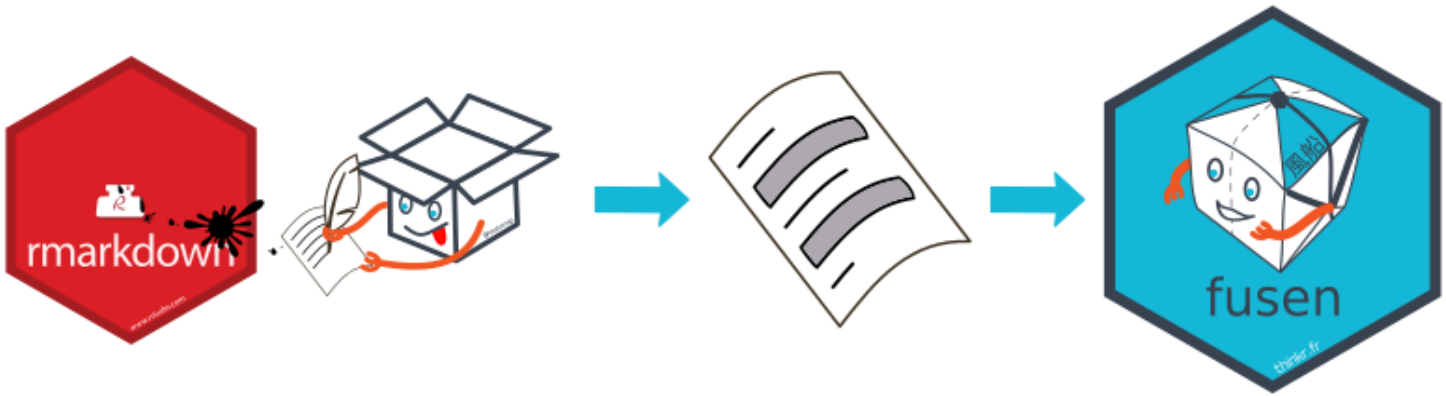
Step 10: Install and use your package

- The 'Build' tab should already appear in RStudio
  - Otherwise, restart your RStudio session
- Note that you can "check" in the 'Build' panel
  - Panel "Build" > "Check"
- Install the package
  - Panel "Build" > "Install and Restart
- Test the package directly in the console
  - `hello::say_hello("Toto")`
- Test the knit of the vignette
- Check that the help for your function appears
  - `?say_hello`
  - Run the reproducible example from help

Your turn

- Go back to step 6 of this Longer version
- Continue development and inflate your package

> Edit "flat_minimal.Rmd" and re-execute `inflate()` as many times as necessary until everything runs smoothly.



> Bonus: If you are motivated, you can start again from the beginning of the 10 steps procedure with a new package name {hello2}

How to add new functionnalities?

This would require to start over from 'Step 4' to:

- Upgrade your existing function
- Add a new function in the current flat template
  - New entitled section with `function`, `examples`, `tests`
  - Inflate
  - Install
- There is a RStudio *Addin* to "Add {fusen} chunks"
- Or create a new flat template using `fusen::add_flat_template("add")` for a new family of functions, thus new vignette
- There is a RStudio *Addin* to "Add {fusen} flat template"
- And `inflate()` this new "flat_additional.Rmd"

And the classical way without {fusen}?

- You can use the 'Rmd first ' approach writing your code in a Rmarkdown file
  - this can directly be your vignette
- Then, you'll need to fill / copy the different files yourself
  - "R/"
  - "tests/"
  - "vignettes/"

And the classical way without {fusen}?

- File > New Project > New directory > Package with devtools
- Fill DESCRIPTION file
- Run function for the desired license
  - `usethis::use_*_license()`
- Develop in a Rmd
  - Either in the sub-directory "dev/" + `usethis::use_build_ignore("dev/")`
  - Or, develop in a vignette directly
- Copy/Cut in the correct place
  - functions + examples => "R/"
    - `usethis::use_r("ma_fonction")`
  - tests => "tests/testthat/"
    - `usethis::use_testthat()`
    - `usethis::use_test("ma_fonction")`
  - vignettes => "vignettes/"
    - `usethis::use_vignette("Le titre de ma vignette")`
- Generate documentation
  - Either `attachment::att_amend_desc()`
  - Ou `roxygen2::roxygenise()` + Fill in DESCRIPTION (Suggests, Imports)
- Check the package
  - `devtools::check()` => `0 errors, 0 warnings, 0 notes`

And the classical way without {fusen}?

- You need to fill the different files yourself
  - "R/"
  - "tests/"
  - "vignettes/"
- While developing you could
  - Run an example of your function in "R/" directly with Ctrl + Enter
  - Run the unit test by clicking on "Run test"
  - Run the vignette if your package is installed
- Note that you can still do these actions using {fusen} after `inflate()`

> Be careful, when using {fusen}, if you want to modify some code, go back to the "flat_*.Rmd" and do `inflate()` again

Exercise: Take time to finish your drawings

Where were moved the pieces of code from chunks:

- `description`?
- `function`?
- `example`?
- `tests`?
- `development`?

> Verify and update your previous drawings. See each thing that {fusen} does for you!

Include datasets in your package

What about data?

Include datasets

In a package, it can be useful to include data.

- To demonstrate the use of a function with a relevant example
- To disseminate information

There are three types of datasets to include, thus three ways to include them into a package.

1. A raw data file (xlsx, csv or other), **NOT** available to the end user, but accessible to the developers only, stored in `data-raw/` folder

2. An example dataset in `rda` format, to be loaded as is, available to the user, stored in the `data/` folder (such as `iris` or `mtcars` for example)

3. A data file (xlsx, csv or other) not transformed into `rda`, available to the end user, stored in the `inst/` folder

1. Include datasets in `data-raw/`

Internal dataset, accessible to developers only

- Developers store raw datasets to keep the source of examples close to them
- Users can not access them, there are not installed

Steps:

- Create folder `data-raw/` at the root of the package using: `usethis::use_data_raw()`
- Insert raw datasets inside
- Use R/Rmd scripts to prepare a cleaner / smaller dataset for future examples

Information:

- `data-raw` is not installed with the package
  - It is not accessible to the user
- Example: https://github.com/ThinkR-open/prenoms/tree/master/data-raw

2. Include datasets in `data/`

Exported dataset, accessible to the user

- Developers includes a dataset in the package using: `usethis::use_data(my_dataset)`
- Users load the dataset named `my_dataset` using: `data(my_dataset)`

Steps:

- Create `data-raw/` to prepare your dataset using: `usethis::use_data_raw("my_dataset")`
- Prepare your dataset as needed for reproducible examples
- Store `my_dataset` as internal data

```
# Read some raw data
# my_data_to_clean <- readr::read_csv("my_raw_data.csv")

# Or use existing dataset like `diamonds`
my_dataset <- dplyr::slice_sample(diamonds, prop = 0.2)
usethis::use_data(my_dataset, overwrite = TRUE)
```

- `overwrite = TRUE` overwrite the dataset if already exists, as for an update
- Have a look at the content of newly created folder `data/`

2. Include datasets in `data/`

Exported dataset, accessible to the user

- Developers includes a dataset in the package using: `usethis::use_data(my_dataset)`
- Users load the dataset named `my_dataset` using: `data(my_dataset)`

Information:

- `my_dataset` is accessible to the user after package installation

```
library(mypackage)
data(my_dataset)
```

- `my_dataset` is stored as `.rda` file, only readable by R, similar to `.RData` files

2. Include datasets in `data/`

Exported dataset, accessible to the user

- Exported datasets need to be documented, like functions
- by convention the documentation will be written in a file of the form "R/doc_my_dataset.R".
- The following script will automatically build the documentation to be completed:

```
my_dataset <- dplyr::slice_sample(diamonds, prop = 0.2)
usethis::use_data(my_dataset, overwrite = TRUE)

cat(sinew::makeOxygen("my_dataset"),
    file = "R/doc_my_dataset.R")
rstudioapi::navigateToFile("R/doc_my_dataset.R")
```

à rajouter dans "data-raw/my_dataset.R"

2. Include datasets in `data/`

Exported dataset, accessible to the user

- Exported datasets need to be documented, like functions

Information:

2 main tags:

- `@format` (required) a summary of the data.
  - if `@format` is not specified, then roxygen will edit a standard one.
- `@source` : the origin of the data, the source.

We do not `@export` a dataset.

3. Include datasets in `inst/`

Raw dataset, accessible to the user

- Developers includes a dataset in the `inst/` folder as is directly
- Users find the dataset using `system.file("my_dataset.csv", package = "mypackage")`

Steps

- Create `inst/` folder using: `dir.create(here::here("inst"))`
- Add a dataset inside
- Install the package
- Users can access the path of the dataset and read the file as usual

```
# Store "my_dataset.csv" in "inst/" folder
the_data_path <- system.file("my_dataset.csv", package = "mypackage")
the_data <- readr::read_csv(the_data_path)
```

3. Include datasets in `inst/`

Raw dataset, accessible to the user

- Developers includes a dataset in the `inst/` folder as is directly
- Users find the dataset using `system.file("my_dataset.csv", package = "mypackage")`

Information

- Any type of data is allowed
- Use the path for your reproducible examples
- Organise the content of `inst/` as your want

Deal with datasets during development

- Datasets in `data/` or `inst/` are only available after package installation
- Simulate installation using `pkgload::load_all()` during development
  - `system.file()` temporarily returns development path (not installed path)

Steps

- Check your reproducible examples in the Rmd file with package data

```
# For development only
pkgload::load_all()

# Same code to add in your `examples` or `tests`
# Can be tested directly during development
the_data_path <- system.file("my_dataset.csv", package = "mypackage")
the_data <- readr::read_csv(the_data_path)
```

> Note that `pkgload::load_all()` also loads functions in development, as it simulates a real installation

Quizz

If I want to provide a dataset to the user in any format I want, where do I store it?

- A: `extdata/`
- B: `inst/`
- C: `data/`
- D: `data-raw/`

Your turn!

1. Look at the instructor demonstration with *Steps*
2. Do it yourself

Steps

- Create a new {fusen} project with `full` template
- Run the content of the `description` chunk in the "dev_history.Rmd"
- Uncomment `development` chunk in "Read data" section in the "flat_full.Rmd"
- Play with data in `inst/` during development

Bonus

- Create function `check_data_integrity()` that reads a dataset like `nyc_squirrels` and check its integrity
  - For instance, `primary_fur_color` columns should only contains a unique color, there should not be any `+` sign inside this column
  - Stop the function if the integrity is not good
  - Return a message if everything is ok
- Create a reproducible example using the dataset saved in `inst/` with `system.file()`
- Create unit tests with reproducible examples where function should fail

Versioning a {fusen} package with git

And the particular case of {fusen}?

Version a {fusen} package

2 possibilities:

- Your {fusen} package has not been created yet, and you want to set up the versioning before starting the developments
- Your {fusen} package has already been created, and you now want to version it

Versioning a {fusen} package

Case of a new {fusen} package

- Initialize the empty project on GitLab *(remembering to use the name of your future package as project name)*
- Get the `https` link to your project by clicking on the `clone` button
- In RStudio, create the new project File > New Project > Version Control > git and link to the Repository URL

Initiate your new {fusen} package:

```
fusen::create_fusen(path = ".",
                    template = "minimal",
                    overwrite = TRUE)
```

Versioning a {fusen} package

Case of a new {fusen} package

In the *Terminal*,

- Switch in a new *main* branch:

```
git switch -c main
```

- Select modified files in your project:

```
git add .
```

- Write a commit (with an explicit message):

```
git commit -m "Init fusen package"
```

- Push your changes back to the remote:

```
git push -u origin main
```

You can now begin your developments.

Versioning a {fusen} package

Case of an existing {fusen} package

> This procedure can be used for any type of R project, and therefore for any type of package (not only {fusen} packages)

- Initialize the empty project on GitLab *(remembering to use the name of your package as project name)*
- Retrieve the `https` link to your project by clicking on the `clone` button
- Open the RStudio project and run the command `usethis::use_git()` in the console *(the {usethis} package must be installed)*
- Answer yes to all the questions asked in the console, if relevant. A first "Initial Commit" is done for you.
- RStudio restarts and git is operational locally

Versioning a {fusen} package

Case of an existing {fusen} package

Now you have to link this project to your `remote`:

```
usethis::use_git_remote("origin",
                        url = "https://gitlab.com/my_name/mypackage.git",
                        overwrite = TRUE)
```

In the *Terminal*, type:

```
git push -u origin main
```

You just added a `remote` and made a first `push` on the `main` branch.

Exercise

Set up the *git* versioning for your `{hello}` package

What about data analyses in a package?

From Rmd to package to Rmd

Create shareable reports through packages

Create a package with your functions

Then, two possible ways to build your analysis reports:

- Classical package: Use external projects to use your package functions in Rmd reports
- Compendium-like: Include the Rmd reports inside the package development project

A Compendium is like a package

The compendium logic is to separate code from the report output:

- One place to store your R scripts in "R/"
- One place to store the Rmarkdown report in "analyses/"

This is similar to package logic:

- One place to store R scripts in "R/"
- One place to store Rmarkdown examples of use in "vignettes/"

Let's combine both worlds to benefit from robustness of packages

Combine {fusen} and Compendium structure

- Create a {fusen} project, versioned with git

```
# install.packages("fusen")
fusen::create_fusen(template = "full", with_git = TRUE)
```

- Add a "reports/" directory that will not interfer with the package

```
dir.create("reports")
usethis::use_build_ignore("reports")
```

Note: you can add the Compendium structure in the "reports/" sub-directory, with package {rcompendium}

```
# install.packages("rcompendium")
rcompendium::add_compendium("reports")
```

Combine {fusen} and Compendium structure

- Add a reduced dataset for the package in "inst/" or "data/"
  - Use it for examples, unit tests, documentation
- Build your functions in a "flat" file, like any {fusen} package
  - Write the content of the vignette as a reduced version of your future report
  - Create examples, unit tests, documentation as usual
  - Commit regularly
- Use the "reports/" directory to store your reports and outputs, while using the functions of your package
  - Create a Rmarkdown report using your functions loaded with `library(my.package)`
  - Do not forget to install your package prior to building your report

Why would I include my analysis inside my package?

Because of:

- {rim}: manage R versions
- {renv}: manage R packages versions
- and Docker: manage OS and system dependencies

Why would I include my analysis inside my package?

- Develop your project in a controled environment, with fixed versions of packages
  - Functions, tests and examples are valid for a specific set of packages versions with a specific R version
  - Use your functions for your analysis reports with the same working environment
  - Use `attachment::create_renv_for_dev()` (>=0.2.5) to build your "renv.lock" file
- Allow other users to create their analyses in the same environment, by cloning your repository
  - Share your work with the world, without bothering about their R installation
- You can build a Docker container to also fix system dependencies when using your package. {dockerfiler} can help you to set up the container.

Share your work

- Build and publish your {pkgdown} website to present how to use your package

```
# Have a proper Readme - Fill and knit
usethis::use_readme_rmd()

# Allow {pkgdown}
usethis::use_pkgdown()

# Try it locally
pkgdown::build_site()

# GitHub
# Add your credentials for GitHub
gitcreds::gitcreds_set()
# Send your project to a new GitHub project
usethis::use_github()

# Build and publish with GitHub Actions
usethis::use_github_action("pkgdown")

# Build and publish on GitLab
gitlabr::use_gitlab_ci()
```

Thanks for joining this tutorial!

Sébastien Rochette, Florence Mounier

sebastien@thinkr.fr, florence@thinkr.fr

Material of this course is on Github (with answers): statnmap/teach-package-dev-rmdfirst