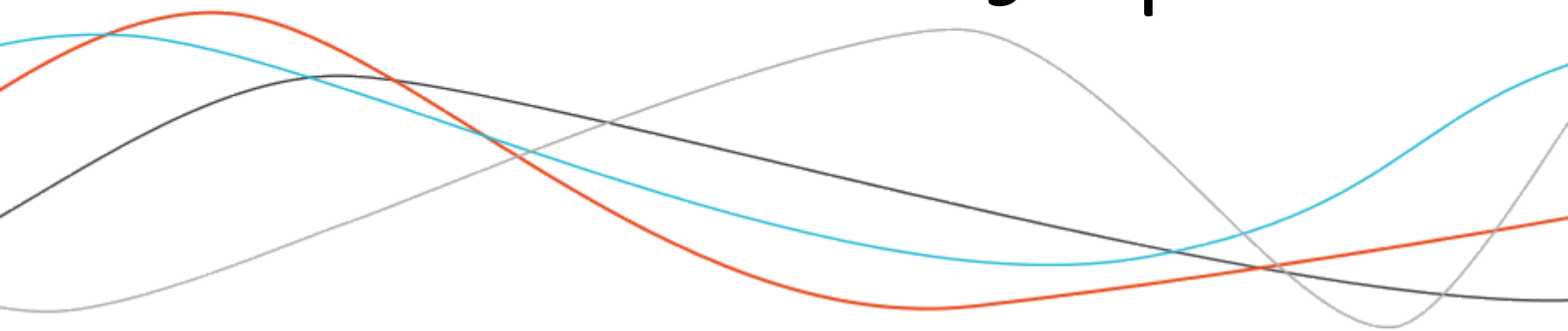
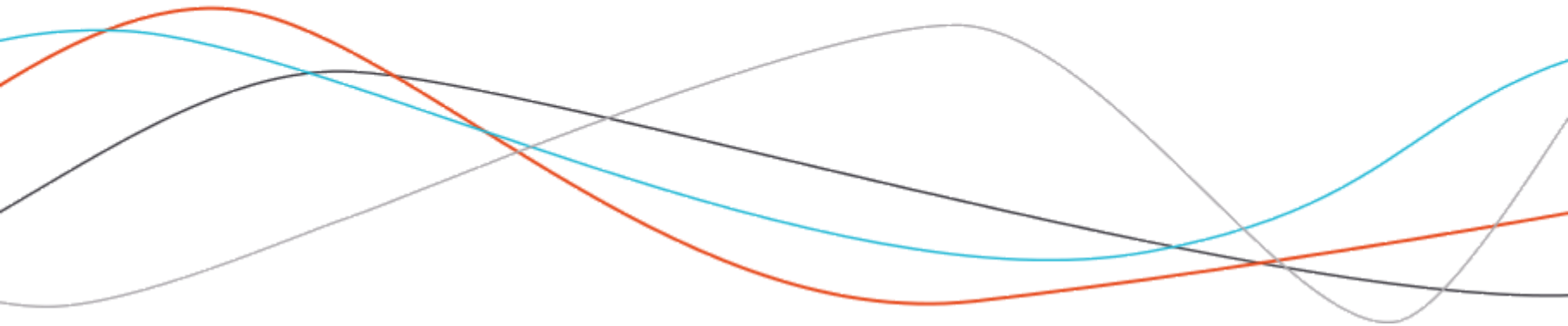
Building a package that lasts

Part 3: dependencies



Understanding namespace



About the NAMESPACE

The NAMESPACE file is one of the most important files of your package. It's also the one you should not edit by hand.

This file describes **how your package interacts with R, and with other packages**. This is where, among other things, the dependencies are managed.

This file also lists the functions that are exported.

The namespace allows the package to work. This file is managed by {roxygen2}, via the tags `@export`, `@import` and `@importFrom`.

Understanding searchPath

When R looks for an object, it will go up the `searchPath` until it finds this object (keep in mind that a function is an object).

Here is an example of a search path:

```
search()
```

```
#>  [1] ".GlobalEnv"        "package:attempt"   "package:forcats"
#>  [4] "package:stringr"   "package:dplyr"     "package:purrr"
#>  [7] "package:readr"     "package:tidyr"     "package:tibble"
#> [10] "package:ggplot2"   "package:tidyverse" "package:rhub"
#> [13] "package:testthat"  "package:bindrcpp"  "tools:rstudio"
#> [16] "package:stats"     "package:graphics"  "package:grDevices"
#> [19] "package:utils"     "package:datasets"  "package:methods"
#> [22] "Autoloads"         "package:base"
```

Understanding searchPath

Each time I load a new package, R moves it to the "top of the list".

```
library(attempt)
search()
```

```
#>  [1] ".GlobalEnv"        "package:attempt"   "package:forcats"
#>  [4] "package:stringr"   "package:dplyr"     "package:purrr"
#>  [7] "package:readr"     "package:tidyr"     "package:tibble"
#> [10] "package:ggplot2"   "package:tidyverse" "package:rhub"
#> [13] "package:testthat"  "package:bindrcpp"  "tools:rstudio"
#> [16] "package:stats"     "package:graphics"  "package:grDevices"
#> [19] "package:utils"     "package:datasets"  "package:methods"
#> [22] "Autoloads"         "package:base"
```

Understanding searchPath

You can give any name to a function:

```
rnorm <- function(x) x + 1
rnorm(1)
```

```
#> [1] 2
```

```
rnorm
```

```
#> function(x) x + 1
```

You can specify the namespace of the function with `::`

```
stats::rnorm(1)
```

```
#> [1] -2.449438
```

```
stats::rnorm
```

```
#> function (n, mean = 0, sd = 1)
#> .Call(C_rnorm, n, mean, sd)
#> <bytecode: 0x102e9c4a0>
#> <environment: namespace:stats>
```

The notation `namespace:stats` indicates in which namespace the function is located.

Understanding namespaces

When programming, we are likely to name objects that have the same name as those in other packages.

And most of the time, you will need functions from other packages.

This is what the NAMESPACE is used for: to manage the interconnection between the different packages in the environment.

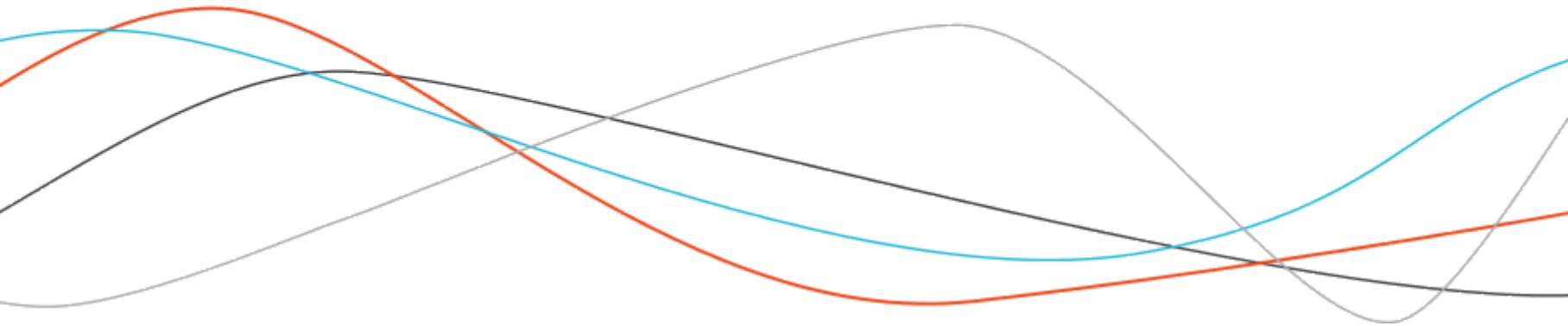
NAMESPACE conflicts

When two functions are called the same in two packages, and these two packages are launched, there is what is called a namespace conflict:

```
library(tidyverse)
tidyverse_conflicts()
```

```
#> ── Conflicts ─────────────────────────────────── tidyverse_conflicts() ──
#> ✖ dplyr::filter()    masks stats::filter()
#> ✖ attempt::if_else() masks dplyr::if_else()
#> ✖ purrr::is_null()   masks testthat::is_null()
#> ✖ dplyr::lag()       masks stats::lag()
#> ✖ dplyr::matches()   masks testthat::matches()
#> ✖ dplyr::vars()      masks ggplot2::vars()
```

Manage dependencies



What's a dependency?

To work, your package may need external functions, i.e. contained in other packages.

R has three types of dependencies that will be contained in the DESCRIPTION :

- `Depends` & `Imports`: the packages that will be `attach()` or `load()` respectively. In practice, always list in `Imports` : `attach` means that the package is attached to the search path (and remember that a good package should not touch to the user's environment).
- `Suggests` : suggests packages to use in addition to your package. Will not be attached or loaded.

These elements are filled automatically thanks to `{usethis}`.

Dependencies

The DESCRIPTION file contains the package dependencies.

A quick and easy way to specify it is the instruction:

```
usethis::use_package("attempt")
```

... adds `{attempt}` in the DESCRIPTION.

That's not enough: we will need to use a roxygen comment on each function of our package to specify the dependencies used.

To do that, we will use `@import` (a whole package) and `@importFrom` (a specific function).

Add dependencies to EACH function

```
#' @import magrittr
#' @importFrom  stats na.omit

moyenne <- function(x){
  x <- x %>% na.omit()
  sum(x)/length(x)
}
```

You can use `import` or `importFrom`.

The better is to use `importFrom`, for preventing namespace conflict.

Add to EACH function.

> It will take a lot of time, but it's better on the long run.

Strategies for naming function...

... to prevent NAMESPACE conflicts

- Use some letters at the beginning of each functions (as does `{stringr}`, for example)
- You can't search the whole CRAN for function names, but you can start with your own computer:

```
??my_fun
??bitShiftL
```

... to help UX

- Name the function after what it does: if the function is used to get X, call it `get_x`.

How many dependencies?

Tough question...

Just keep in mind that depending on another package means that :

- you will potentially have to recode your package if some breaking change happen in your dependencies.
- If one of your dependencies is removed from CRAN, you will be removed too.

In other words, beware the `{clipr}` effect.

Let's practice !