Building a package that lasts

Colin FAY - eRum 2018

What are we going to talk about today?

- 13h30- 14h00: Introduction & Package init
- 14h00 - 14h30: Functions and documentation
- 14h30 - 15h00: Dependencies

Coffee Break: 15h - 15h30

- 15h30 - 16h00: Optimisation
- 16h00 - 16h30: Testing
- 16h30 - 16h45: Continuous integration
- 16h45 - 17h00: Conclusion

Tweet that!

- Hashtag: #erum2018
- @_ColinFay
- @thinkr_fr
- @erum2018

Internet connexion:

- CEU Gest
- Budapest1991

$whoami

Colin Fay - ThinkR

French agency of R experts, focused on everything related to R.

- Training
- Dev & Infrastructure
- Consulting

$whoami

Vincent Guyader

Codeur Fou, formateur et expert logiciel R

Sébastien Rochette

Modélisateur, Formateur R, Joueur de cartographies

Diane Beldame

Dompteuse de ~~dragons~~ données, formatrice logiciel R

Colin Fay

Data Analyst, Formateur R, expert Social Media

Packages, a quick refresher

What is a package?

In R, the fundamental unit of portable code is called a package.

A package is a more or less a combination of:

- code
- data
- documentation
- tests

You can put other content, but we won't cover it today.

Package structure

- `DESCRIPTION` : the metadata of your package
- `NAMESPACE` : how your package interacts with R and with other packages
- `R/` : the code
- `man/` : the documentation
- `inst/` : content that will be put in your package folder after installation
- `data/` : data
- `data-raw/` : a folder with content that will be ignored on build
- `tests` : the tests
- `vignettes` : the vignettes
- `.Rbuildignore` : a description of what will be ignored when the package is built

...

About *.Rbuildignore*

The *.Rbuildignore* file is used to tell R what to ignore when building the package.

The name of the content to ignore can be written in full, or match a regex.

For example:

```
^.*\.Rproj$
^\.Rproj\.user$
^README\.Rmd$
^README-.*\.png$
.travis.yml
^CONDUCT\.md$
^data-raw$
^cran-comments\.md$
paper\..*
^revdep$
^docs$
```

What is a "package that lasts"?

Reproducible and automated package

- Make a package **you will be able to develop**
- Make a package **you will be able to maintain**

The UX of a package: taking a user first approach

- Make a package **people will use**
- Make a package **people will effectively use**

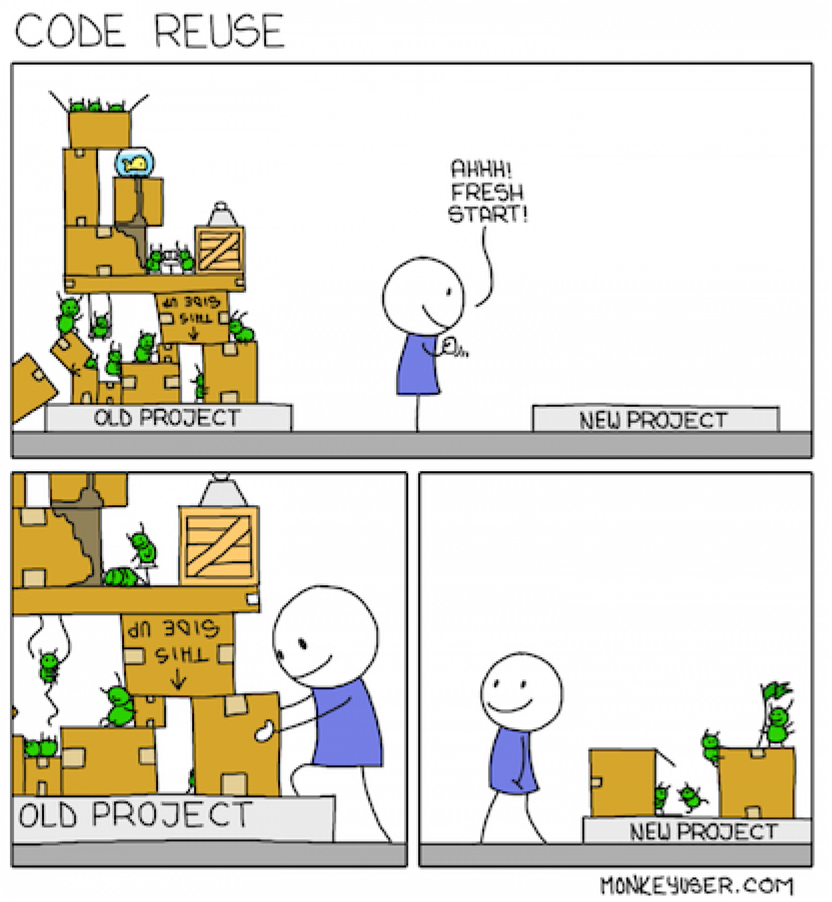
Make a package you will be able to develop

- **Automate everything you can automate**: on the long run, it will prevent mistakes.
- Don't lose your breathe on what can be automated.

Make a package you will be able to maintain

- Create a package you can come back to in two years without having to start everything over.
- Prevent your package from failing when it is released.

Make your code reusable



Make a package people will use

- Take a UX-first approach: **it should be as easy as possible to start using your package**.
- The **simpler and clearer your package is for the user**, the better.

Make a package people will effectively use

- Use **meaningful package and function names**.
- Create **useful, easy to understand, and complete documentation**. That will prevent from "issues overload".

Some of the tools we'll use

- RStudio
- {devtools} - https://github.com/r-lib/devtools
- {desc} - https://github.com/r-lib/desc
- {usethis} - https://github.com/r-lib/usethis
- {roxygen2} - https://github.com/klutometis/roxygen
- {testthat} - https://github.com/r-lib/testthat
- Rtools.exe (if you're on window) : https://cran.r-project.org/bin/windows/Rtools/
- r-base-dev on Unix

Building a package that lasts

Part 1: init

A package is like a house: it won't last without solid fundations.

Before anything... find a good name!

Some tips & conventions :

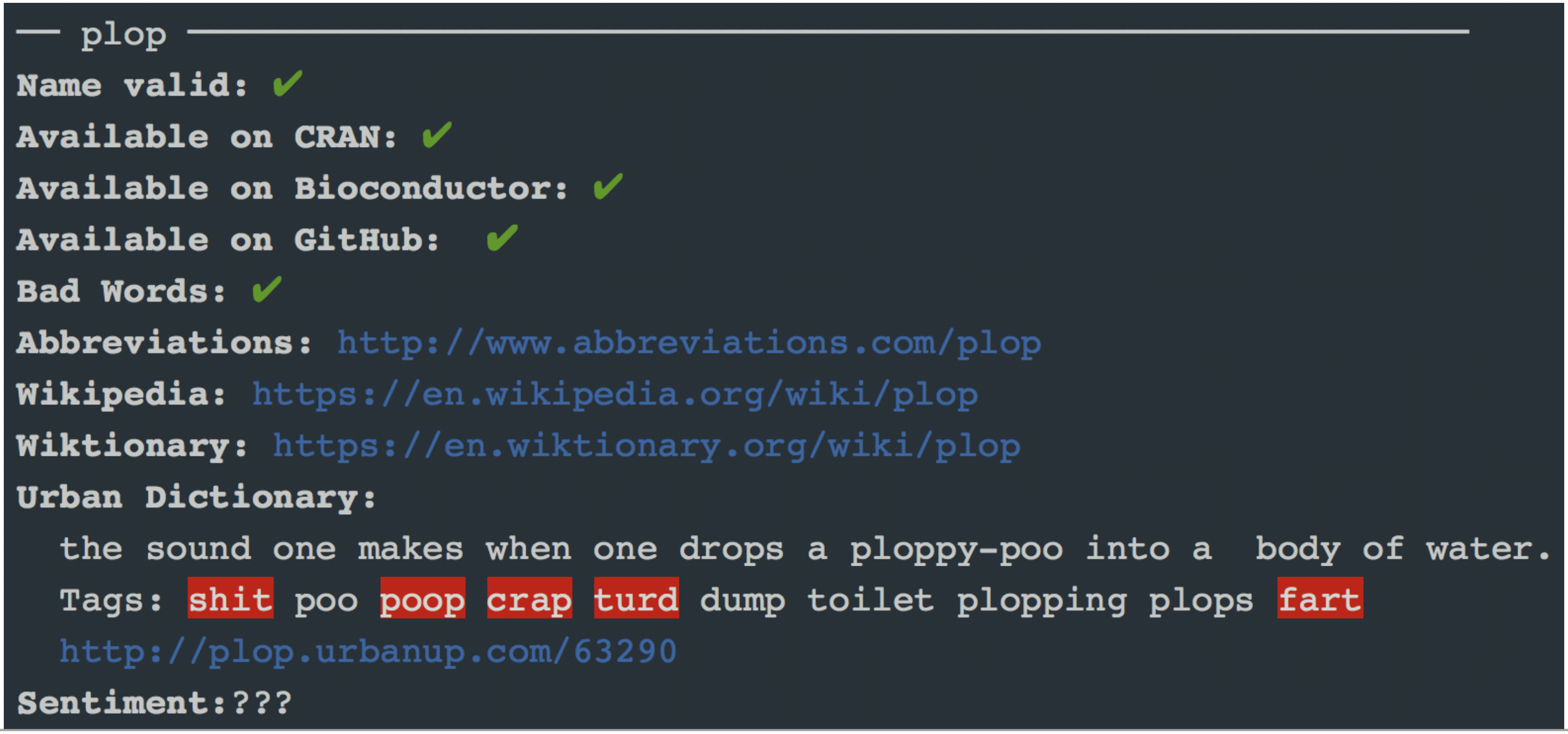
- If Open Source, prefer a name that is easy to find on Google.
- Find a name that is unique, and that describes well what the package does: for example, {testthat} allows to "test that X", and with {usetthis} we "use this X".
- Placing an r on a word can make a good package name: for example {stringr} allows to manipulate strings in R.
- The name can only contain letters, numbers, and dots
- The name must begin with a letter, and must not end with a period.

Good practices :

- Avoid capitalisation, to facilitate memorisation and typing
- Avoid dots, to prevent confusion with S3 methods

Find a name

```r
available::available("plop")
```



```
── plop ──────────────────────────────────────────────────────
Name valid: ✔
Available on CRAN: ✔
Available on Bioconductor: ✔
Available on GitHub:  ✔
Bad Words: ✔
Abbreviations: http://www.abbreviations.com/plop
Wikipedia: https://en.wikipedia.org/wiki/plop
Wiktionary: https://en.wiktionary.org/wiki/plop
Urban Dictionary:
  the sound one makes when one drops a ploppy-poo into a  body of water.
  Tags: shit poo poop crap turd dump toilet plopping plops fart
  http://plop.urbanup.com/63290
Sentiment:???
```

Automation, automation, automation

> "Anything that can be automated, should be automated. Do as little as possible by hand".

H. Wickham, "R Packages"

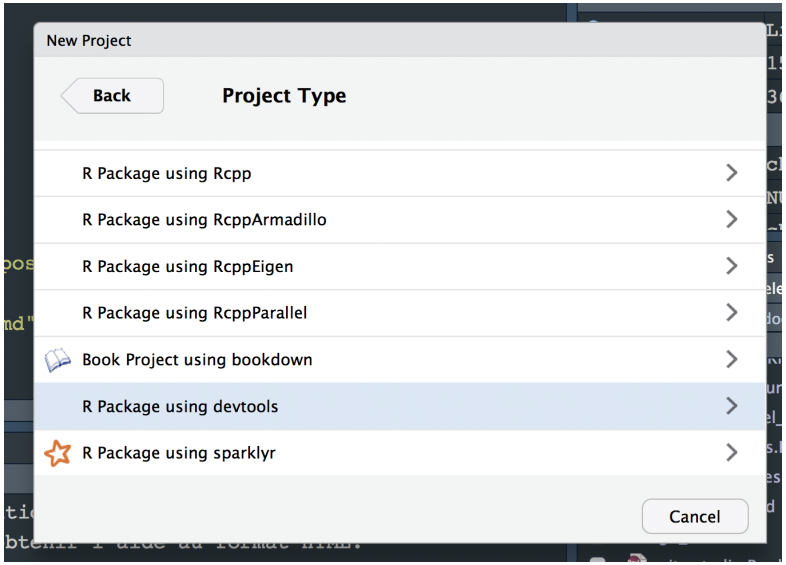
Package developers automate not out of laziness, but **out of security**.

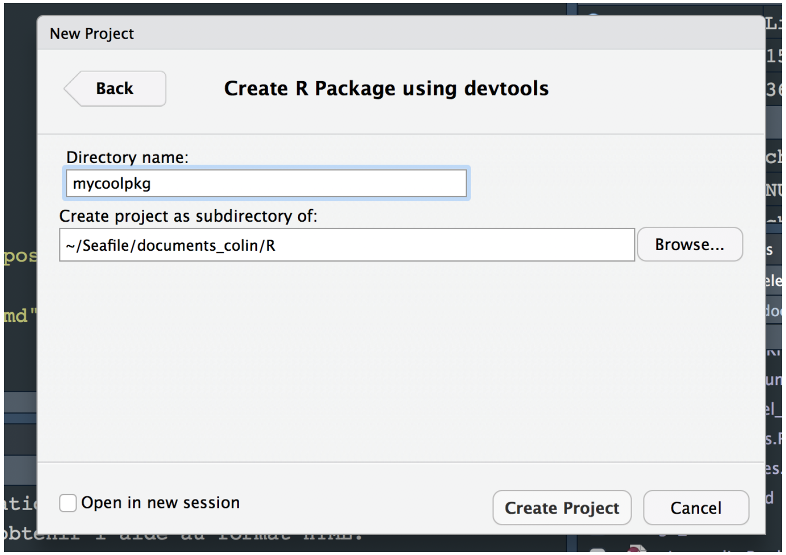
=> Once an automated process works, it will always work (well, in theory).

=> If you lose your computer in a train, you should be able to redo everything that you did.

Create a new package

From RStudio


New Project
Back
Project Type
R Package using Rcpp
R Package using RcppArmadillo
R Package using RcppEigen
R Package using RcppParallel
Book Project using bookdown
R Package using devtools
R Package using sparklyr
Cancel



Directory name: mycoolpkg
Create project as subdirectory of: ~/Seafile/documents_colin/R

Automate your package creation

Create your package with:

```
devtools::create("plop")
```

This will create a basic package skeleton.

You can create a list of options in `options("devtools.desc")` to prefill your DESCRIPTION.

You can also use the `{desc}` package, that we will see in a few slides.

Where do I put my dev script?

Using {usethis} and data-raw

```
library(usethis)
use_data_raw()
```

-> Creates a "data-raw" folder, and everything in there will be ignored at build.

```
file.create("data-raw/devstuffs.R")
```

-> Create a "devstuffs.R" (you can use any other name) to keep everything you did during the engineering process.

Do this for reproducibility

{desc}

{desc} is a package that helps you create and configure your DESCRIPTION files

```
# Remove default DESC
unlink("DESCRIPTION")
# Create a new description object
my_desc <- desc::description$new("!new")

# Set your package name
my_desc$set("Package", "dockerfiler")

#Set your name
my_desc$set("Authors@R", "person('Colin', 'Fay', email =
'contact@colinfay.me', role = c('cre', 'aut'))")

# Remove some author fields
my_desc$del("Maintainer")
```

{desc}

```r
# Set the version
my_desc$set_version("0.0.0.9000")

# The title of your package
my_desc$set(Title = "Easy Dockerfile Creation from R")
# The description of your package
my_desc$set(Description = "Create a Dockerfile.")

# The urls
my_desc$set("URL", "https://github.com/ColinFay/dockerfiler")
my_desc$set("BugReports",
"https://github.com/ColinFay/dockerfiler/issues")

# Save everyting
my_desc$write(file = "DESCRIPTION")
```

{desc}

Other {desc} methods:

- `my_desc$add_author()`
- `my_desc$add_remotes()`
- `my_desc$add_to_collate()`
- `my_desc$bump_version()`
- `my_desc$del_*` (author, collate, dep, remote…)
- `my_desc$get_*` (author, deps, urls…)
- `my_desc$set_*` (authors, deps, urls…)
- `my_desc$normalize()`
- `my_desc$to_latex()`

About package number

Choosing your version number

The version number reads major.minor.patch, where major is a major release, minor a minor, and patch a bug fix.

Good practice

Until the first stable version of the package is released, the version number should be 0.0.0.9000.

This allows to increment 0001 at each new stage of the project with ease, without getting stuck, and also to clearly notify that the package is still in the development phase.

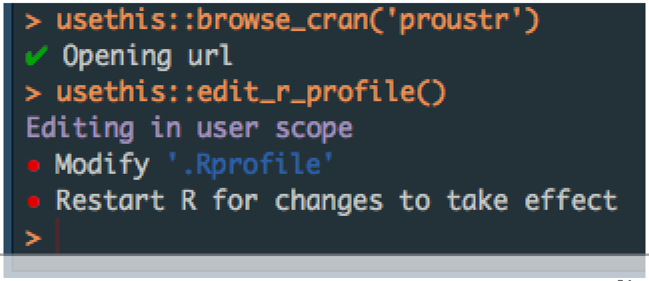
More about {usethis}

`{usethis}` is a package designed to automate the implementation of package elements. For example :

- The license
- Dependencies
- The README
- The connection to git
- The NEWS file
- The data-raw folder
- The use of external data

In the console, you will find messages with the following nomenclature :

- ✅: {usethis} did all the work
- 🔴: some tasks remain to be done



```
> usethis::browse_cran('proustr')
✔ Opening url
> usethis::edit_r_profile()
Editing in user scope
● Modify '.Rprofile'
● Restart R for changes to take effect
>
```

Development with {usethis}

All the functions that start with `use_` allow you to use a template and/or place the right thing in the right place.

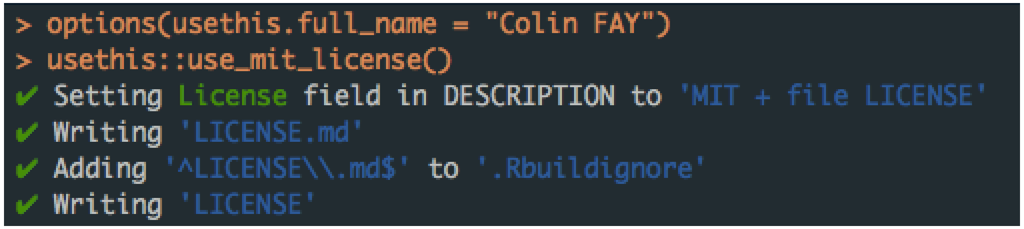
- `use_build_ignore(file)` : create a regular expression from a file name and add it to .Rbuildignore
- `use_data` and `use_data_raw` : the first transforms a data set into a .Rdata, then places it in the data/ folder. The second creates the `data-raw` folder.
- `use_description` : creates the DESCRIPTION file (used only if you don't use the RStudio package creation interface, the `{desc}` package or the `devtools::create` function).
- `use_package` : adds the package as Imports in the DESCRIPTION, `use_dev_package` adds a dependency in the Remote field.
- `use_git`, `use_github`, `use_github_labels`, `use_github_links`, `use_git_hook`, `use_git_ignore` : interaction with Git and Github.

Development with {usethis}

- `use_pipe` : import %>% from {magrittr}.
- `use_rcpp` : if your package uses Rcpp.
- `use_testthat` : creates the `testthat` folder.
- `use_vignette` : creates a Vignette template.
- `use_revdep` : creates documents for reverse dependencies.
- `use_appveyor`, `use_travis`: for continuous integration.
- `use_coverage`: for test coverage

Development with {usethis}

- `use_tidy_description`: puts the DESCRIPTION fields in a standard order and sorts the dependencies in alphabetical order.
- `use_tidy_eval`: functions for tidyeval.
- `use_tidy_versions`: adds to all dependencies the restriction to at least the version installed on the machine.
- `use_apl2_license`, `use_cc0_license`, `use_gpl3_license`, `use_mit_license`: licenses.



```
> options(usethis.full_name = "Colin FAY")
> usethis::use_mit_license()
✔ Setting License field in DESCRIPTION to 'MIT + file LICENSE'
✔ Writing 'LICENSE.md'
✔ Adding '^LICENSE\\.md$' to '.Rbuildignore'
✔ Writing 'LICENSE'
```

Development with {usethis}

- `use_code_of_conduct`: integrates a Code of Conducts file
- `use_cran_badge`, `use_depsy_badge`: create a CRAN badge with http://www.r-pkg.org, a Depsy badge with http://depsy.org
- `use_cran_comments`: create a comments file before submitting to CRAN
- `use_lifecycle_badge` : allows to indicate in the README the "state of development" of the package : Experimental, Maturing, Dormant, Stable, Questioning, Retired, Archived.
- `use_news_md` : create a NEWS file
- `use_pkgdown` : create a pkgdown
- `use_readme_rmd` and `use_readme_md` : create the README file in the corresponding format

Development with {usethis}

```r
use_news_md()
use_readme_rmd()
use_mit_license(name = "Colin FAY")
use_code_of_conduct()
use_lifecycle_badge("Experimental")
use_testthat()
use_test("R6")
use_package("attempt")
use_vignette("dockerfiler")
use_travis()
use_appveyor()
use_coverage()
use_tidy_description()
```

Automate startup

```r
init_data_raw <- function(name = "devstuffs"){
  stop_if_not(name, is.character, "Please use a character vector")
  use_data_raw()
  file.create(glue("data-raw/{name}.R"))
  file.edit("data-raw/devstuffs.R")
}

init_docs <- function(name = "Colin FAY"){
  stop_if_not(name, is.character, "Please use a character vector")
  use_mit_license(name)
  use_readme_rmd()
  use_news_md()
  use_testthat()
}
```

Automate startup

```r
fill_desc <- function(name, Title, Description, repo){
  unlink("DESCRIPTION")
  my_desc <- description$new("!new")
  my_desc$set("Package", name)
  my_desc$set("Authors@R", "person('Colin', 'Fay', email =
'contact@colinfay.me', role = c('cre', 'aut'))")
  my_desc$del("Maintainer")
  my_desc$set_version("0.0.0.9000")
  my_desc$set(Title = Title)
  my_desc$set(Description = Description)
  my_desc$set("URL", glue("https://github.com/ColinFay/{repo}"))
  my_desc$set("BugReports",
glue("https://github.com/ColinFay/{repo}/issues"))
  my_desc$write(file = "DESCRIPTION")
}
```

Let's practice !