Hacking RStudio

useR! 2019

Colin Fay - ThinkR

What are we going to talk about today?

- 09h00 - 10h00: Addin & `{rstudioapi}`
- 10h00 - 10h30: Customising RStudio with CSS & Snippets

Coffee Break: 10h30 - 11h00

- 11h00 - 11h45: Building Templates
- 11h45 - 12h30: Connections

Tweet that!

- Hashtag: #useR2019
- @_ColinFay
- @thinkr_fr
- @UseR2019_Conf

Internet connexion:

- USER
- useR!2019

Find these slides

https://github.com/ColinFay/user2019workshop

$ whoami

Colin FAY

Data Scientist & R-Hacker at ThinkR, a french company focused on Data Science & R.

Hyperactive open source developer.

- https://thinkr.fr
- https://rtask.thinkr.fr
- https://colinfay.me
- https://twitter.com/_colinfay
- https://github.com/colinfay

ThinkR

Data Science engineering, focused on R.

- Training
- Software Engineering
- R in production
- Consulting

$whoarewe

Vincent Guyader

Codeur Fou, formateur et expert logiciel R

Diane Beldame

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

Colin Fay

Data scientist et R hacker

Sébastien Rochette

Modélisateur, Formateur R, Joueur de cartographies

Cervan Girard

Le nouveau

Hacking RStudio

Part 1 Addin & `{rstudioapi}`

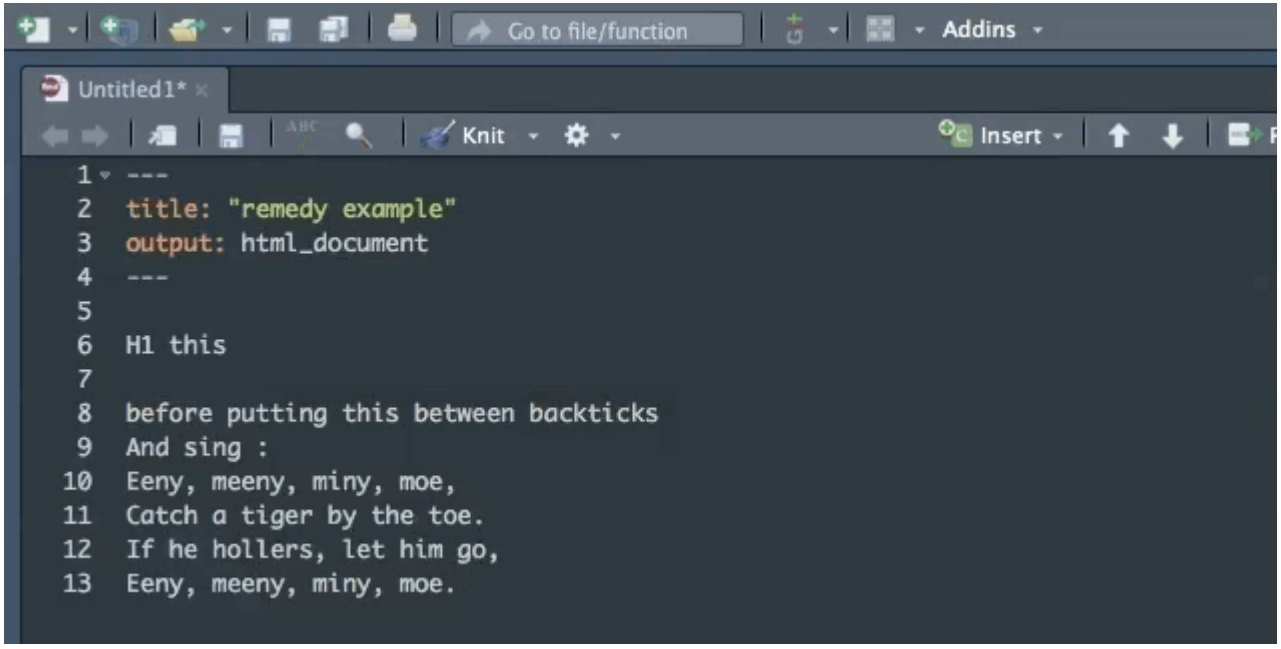
Why?

Improved workflow & Better user experience

Everything that can be (safely) automated should be automated

- Automate the boring stuff
- Avoid copying and pasting
- Create shortcuts for common behaviours
- Better user experience for a package

One example


Go to file/function
Addins
Untitled1*
Knit
Insert
---
title: "remedy example"
output: html_document

H1 this
before putting this between backticks
And sing :
Eeny, meeny, miny, moe,
Catch a tiger by the toe.
If he hollers, let him go,
Eeny, meeny, miny, moe.

One example

- `{remedy}` is a 📦 that tries to bring RMarkdown writing closer to a "word-processor experience"
- Mimics what you would find in things like Open Office (e.g: select a portion of text, and bold it with a keypress)
- This package uses the `{rstudioapi}` and RStudio addin template

Hacking RStudio

The `{rstudioapi}` 

`{rstudioapi}`

The `{rstudioapi}` 📦 is designed to manipulate RStudio (when available) through Command line. You can:

- Manipulate documents (edit, save, open...) and projects
- Generate dialog boxes
- Interact with RStudio terminals & the current R Session
- Launch jobs
- Open new tabs

{rstudioapi}

The versions used for this workshop is:

```
packageVersion("rstudioapi")
```

```
#> [1] '0.10'
```

```
rstudioapi::versionInfo()$version
```

```
#> [1] '1.2.1335'
```

Manipulate documents w/ `{rstudioapi}`

Creating elements

- `documentNew()` & `documentClose()`
- `initializeProject()`

Navigation

- `navigateToFile(path, line, column)`
- `openProject(path)`

Saving files

- `documentSave()`, `documentSaveAll()`

Manipulate documents w/ `{rstudioapi}`

- `document_range()` & `modifyRange()`

A range is a set of `document_position` in an RStudio document.

A position is defined by a row and a column in a document. A range is two positions, one for the beginning, one for the end of the range.

- `insertText(range, text, id)` & `setDocumentContents(text, id = NULL)`

`insertText()` adds a text at a specific range inside a given document (passed to the `id` argument. If `id` is NULL, the content will be passed to the currently open or last focused document).

`setDocumentContents()` takes a text and the id of a document, and set the content of this doc to `text`.

> If range == `Inf`, the content is added at the end of the document.

Manipulate documents w/ `{rstudioapi}`

```
enclose <- function(prefix, postfix = prefix) {
  # Get the context of the Editor
  a <- rstudioapi::getSourceEditorContext()
  # a$selection is a list refering to the selected text
  for (s in a$selection) {
    rstudioapi::insertText(
      location = s$range,
      text = sprintf(
        "%s%s%s",
        prefix,
        s$text,
        postfix
      )
    )
  }
}
italicsr <- function() enclose("_")
```

> https://github.com/ThinkR-open/remedy

Access RStudio interface elements

- `getActiveDocumentContext()`

```
# get console editor id
context <- rstudioapi::getActiveDocumentContext()
id <- context$id
id
```

```
#> [1] "#console"
```

- `getActiveProject()`
- `getConsoleEditorContext()`
- `getSourceEditorContext()`

Manipulate R session(s) w/ `{rstudioapi}`

- `restartSession()`

This function restarts the current R Process.

- `sendToConsole(code, execute, echo, focus)`

```
sendToConsole("library('golem')")
```

Dialogs w/ `{rstudioapi}`

- `selectFile()` & `selectDirectory()`
- `askForPassword()` & `askForSecret()`
- `showDialog()`, `showPrompt()` & `showQuestion()`

```
con <- DBI::dbConnect(RMySQL::MySQL(),
  host = "mydb",
  user = "colin",
  password = rstudioapi::askForPassword("password")
)
```

> https://db.rstudio.com/dplyr/

Dialogs w/ {rstudioapi}

```
file_info <- function(){
  path <- selectFile()
  file.info(path)
}
```

```
completed <- function(...){
  res <- force(...)
  showDialog("Done !", "Code has completed")
  return(res)
}
```

Playing with the terminals

- `terminalCreate()` & `terminalActivate()`
- `terminalExecute()`
- `terminalList()`
- `terminalVisible()`, `terminalBusy()` & `terminalRunning()`
- `terminalExitCode(termId)`
- `terminalBuffer(termId)`
- `terminalKill()`

Jobs

- `jobRunScript()`

sourceMarkers

Allow to display a custom `sourceMarkers` pane.

```
rstudioapi::sourceMarkers(
  "export",
  df
)
```

`df` must have:

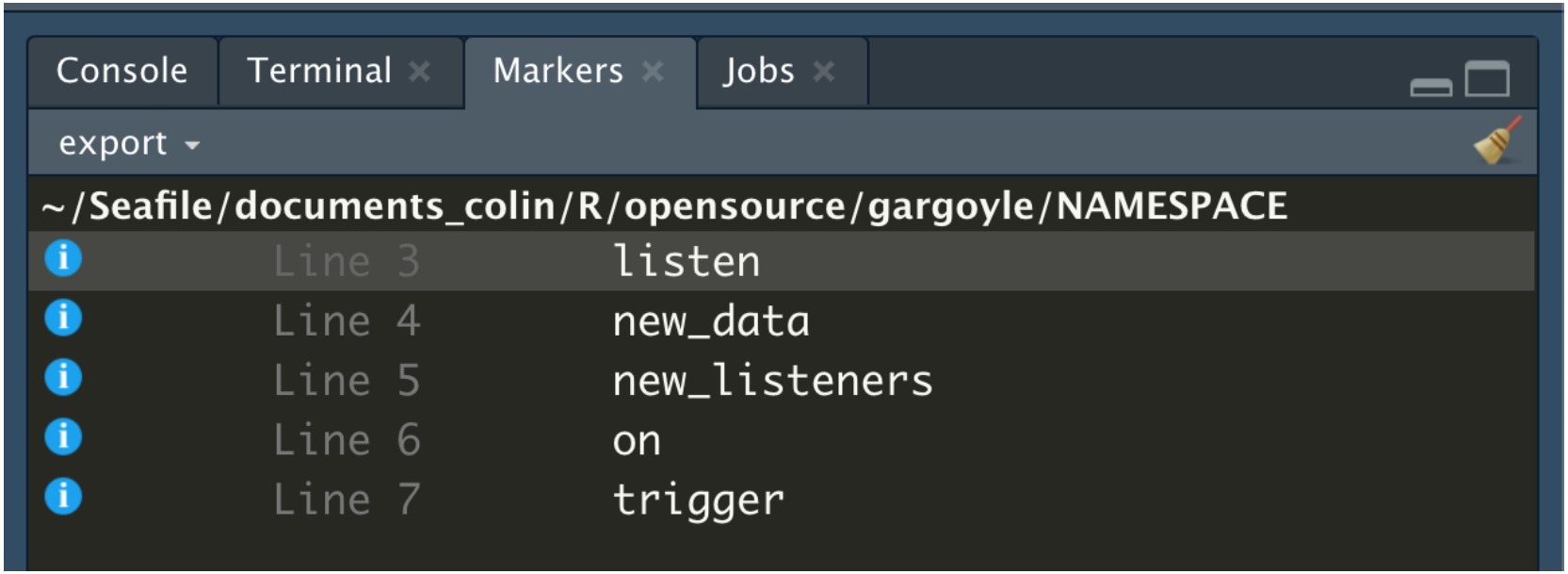
1. `type` Marker type ("error", "warning", "info", "style", or "usage")
2. `file` Path to source file
3. `line` Line number witin source file
4. `column` Column number within line
5. `message` Short descriptive message

sourceMarkers

```
a <- readLines("NAMESPACE")
l <- grepl("export", a)

df <- data.frame(
  type = "info",
  file = "NAMESPACE",
  line = which(l),
  column = 1,
  message = gsub(".*\\((.*)\\)", "\\1", a[l]),
  stringsAsFactors = FALSE
)
rstudioapi::sourceMarkers("export",  df)
```

sourceMarkers



`{rstudioapi}` dev pattern

=> Before running any `{rstudioapi}`-based function, check if RStudio is available.

- `rstudioapi::isAvailable()` (returns a Boolean)

You can even check for a specific version:

- `rstudioapi::isAvailable(version_needed = "0.1.0")`

There is also `rstudioapi::verifyAvailable()`, which returns an error message if RStudio is not running (instead of a boolean).

```
rstudioapi::isAvailable()
```

```
#> [1] TRUE
```

`{rstudioapi}` dev pattern

=> Check if a function is available in the `{rstudioapi}` 📦

- `rstudioapi::hasFun()`

As `{rstudioapi}` relies on internal RStudio functions, the availability of `{rstudioapi}` is linked to the user version of RStudio.

For example, the `askForPassword()` function was added in version 0.99.853 of RStudio.

This function allows to run function only if they are available:

```
if (rstudioapi::hasFun("askForPassword")){
  rstudioapi::askForPassword()
}
```

Addins

- Execute R functions interactively from the IDE
- Can be used through the drop down menu
- Can be mapped to keyboard shortcuts

Two types

1. Text macros
2. Shiny Gadgets

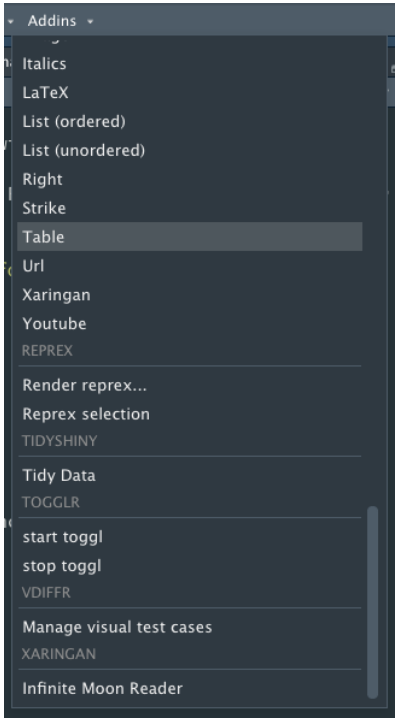
Addin examples

- `{datapasta}`: Reducing resistance associated with copying and pasting data to and from R
- `{giphyr}`: An R package for giphy API
- `{colourpicker}`: A colour picker tool for Shiny and for selecting colours in plots (in R)
- `{styler}`: Non-invasive pretty printing of R code
- `{esquisse}`: RStudio add-in to make plots with ggplot2
- `{todor}`: RStudio add-in for finding TODO, FIXME, CHANGED etc. comments in your code.

> See https://github.com/daattali/addinslist for more

Create an addin

- An addin is a package, so create a package
- Create the function that you want to be launched
- Run `usethis::use_addin()`
- Complete the `addins.dcf`
- Install the package
- And tadaa 🎂



Create an addin

`addins.dcf`

```
Name: New Addin Name
Description: New Addin Description
Binding: new_addin
Interactive: false
```

- Name of the addin
- Its description
- The function to bind to the addin
- Is the addin interactive (i.e does it launch a Shiny app)?

Let's practice !

Now it's your turn to create an addin

Pick an idea (or choose your own)

- Takes a selected word, and look for it on Wikipedia.
- Inserts a random cat picture in a markdown.
- Takes a selected word, and allows to turn to lower & uppercase.
- Opens a dialog that take a password, and looks if this password is anywhere in the current project. Optional: opens a sourceMarkers pane with the results.
- Takes a selected function, and add it into a `R/` folder.