



7 Méthodes secrètes des informaticiens pour mieux programmer

Rencontres R Avignon 2023



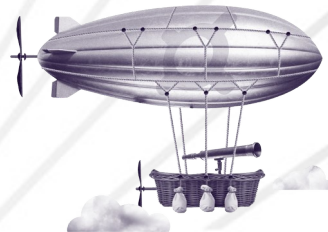
**MAKINA
CORPUS**

regilero



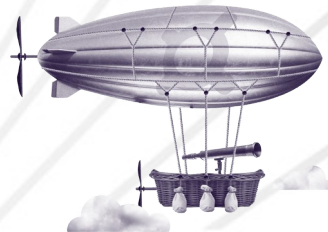
Bonjour

- Régis Leroy (regilero)
- DevOp, Architecture Web
- Hacking Web
- Bases de Données
- HTTP, PHP, Python, Bash, SQL et R
- En tant qu'informaticien je possède une qualité que beaucoup considèrent comme un défaut...



La Paresse

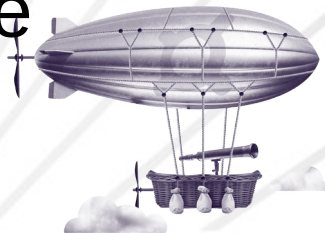
- Éviter de perdre du temps
- Investir son temps au bon endroit
- Mémoriser le moins possible



À qui s'adresse cette présentation ?

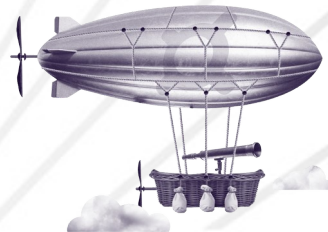
Scientifiques, experts, utilisateurs de R, mais l'informatique n'était pas votre première compétence

- Les gens qui gèrent des programmes qui ressemblent à ce que j'ai pu auditer
- Les gens qui ont hérité de la maintenance de ces programmes
- Les gens qui vont écrire des programmes dans le futur



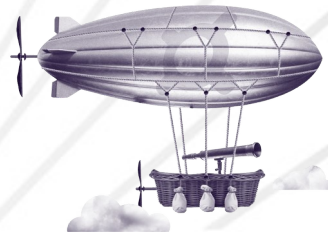
Des méthodes secrètes ?

- On ne comprends pas toujours l'importance de ces concepts au moment de notre formation
- C'est une **culture** transmise par les pairs...
- .. ou bien apprise dans la douleur



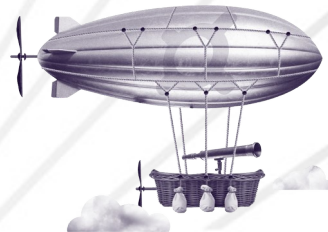
Objectifs

- Le programme devra vivre longtemps
 - R va changer, les librairies vont changer
 - Vous oublierez des choses
- Il sera peut être partagé, transmis, réutilisé
- Il devra sans doute tourner sur une autre machine un jour
- Il devra sans doute être amendé, corrigé, complété, etc.
- Il devrait être correct et robuste



1 - Documenter

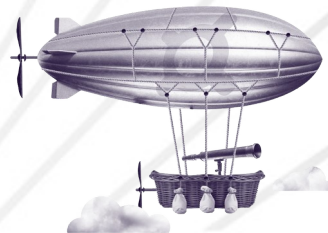
- Ne comptez ni sur votre mémoire ni sur les évidences



1- Documenter

- un **README** est **obligatoire**
 - Visitez github, tous les projets ont un README, et le README sert de page d'introduction.

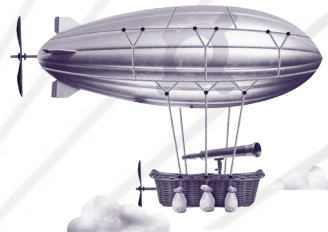
Il y a beaucoup d'éléments critiques à documenter. Mais quel est le plus important ?



README : INSTALLATION

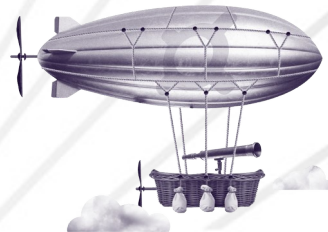
Le plus important

- Comment Installer le programme
 - Si on arrive pas à faire tourner le programme, c'est comme s'il n'existait pas
 - Ce sera toujours votre premier contact avec un utilisateur futur
 - Ce sera aussi un aide mémoire pour vous-même quand vous devrez réinstaller le programme



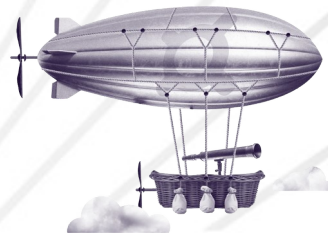
README : INSTALLATION

- Commencer par décrire votre propre installation
 - Rstudio ?
 - Renv ? Packrat? Conda ?
 - **Version de R**
- Plus tard vous pourrez vous attaquer à documenter d'autres installations réussies



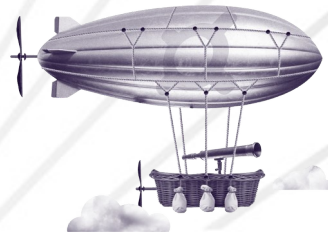
README : Usage

- C'est une extension du point précédent
- Vous même dans 4 ans aurez oublié comment lancer certaines commandes, quels fichiers sont à adapter, etc.



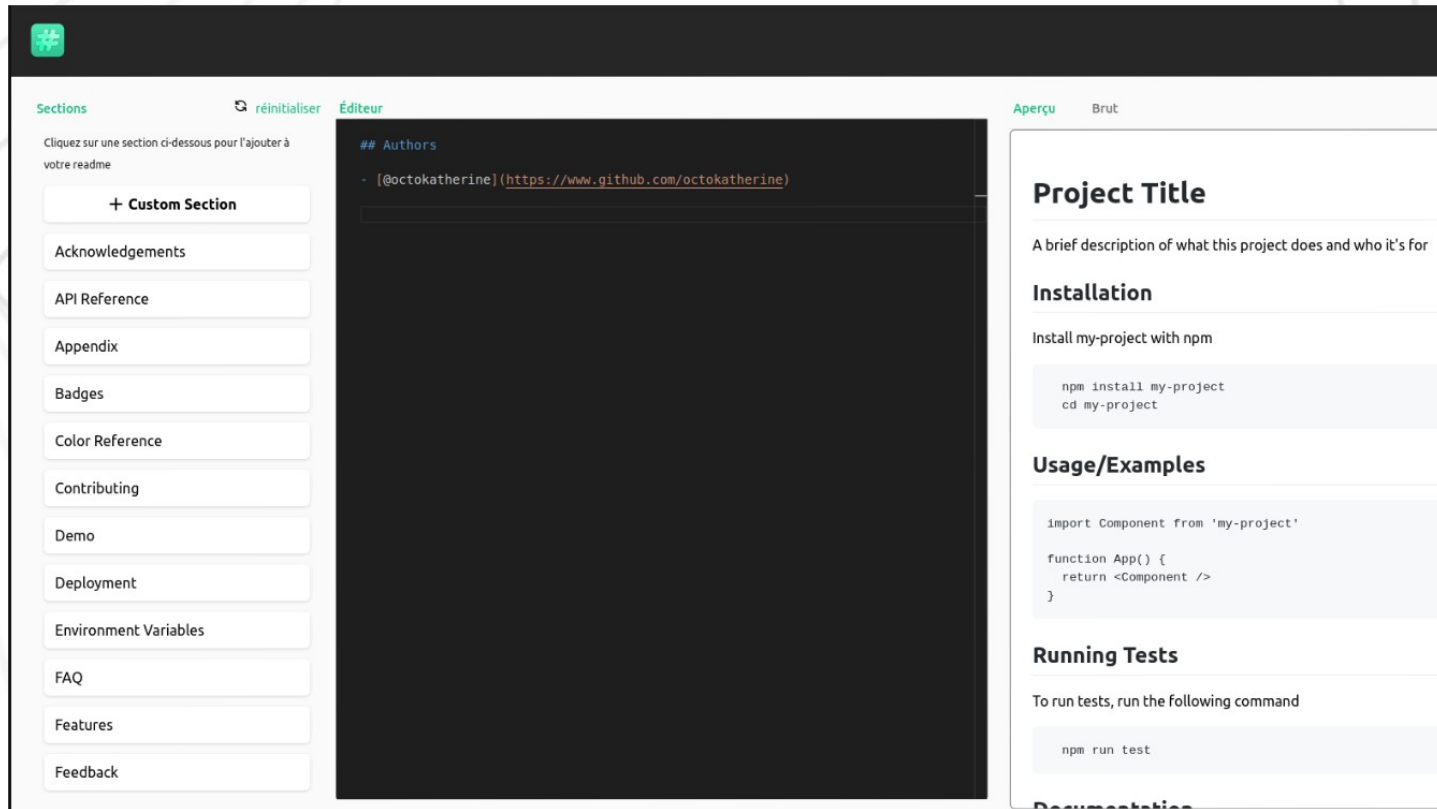
README

- Le format est libre, mais n'hésitez pas à utiliser le **markdown** ou **rmarkdown** (un markdown boosté pour le R, avec effet « notebook »).
- Documentez ce qui n'est **plus** dans le code :
 - Vos objectifs initiaux
 - Vos expérimentations
 - Les pistes abandonnées



README

- Un bon site **readme.so**



The screenshot shows the README.so web editor. On the left is a sidebar with a list of sections to add to the README, including Acknowledgements, API Reference, Appendix, Badges, Color Reference, Contributing, Demo, Deployment, Environment Variables, FAQ, Features, and Feedback. The main area is a dark-themed code editor with a light blue header bar containing 'réinitialiser' and 'Éditeur'. The code in the editor starts with '## Authors' followed by a list item for '@octokatherine' with a GitHub link. On the right is a preview pane with a light gray header bar containing 'Aperçu' and 'Brut'. The preview shows the rendered README with sections for 'Project Title', 'Installation', 'Usage/Examples', and 'Running Tests', each with placeholder text and code blocks.

Sections réinitialiser Éditeur

Cliquez sur une section ci-dessous pour l'ajouter à votre readme

+ Custom Section

Acknowledgements

API Reference

Appendix

Badges

Color Reference

Contributing

Demo

Deployment

Environment Variables

FAQ

Features

Feedback

Authors

- [[@octokatherine](https://www.github.com/octokatherine)](<https://www.github.com/octokatherine>)

Aperçu **Brut**

Project Title

A brief description of what this project does and who it's for

Installation

Install my-project with npm

```
npm install my-project
cd my-project
```

Usage/Examples

```
import Component from 'my-project'

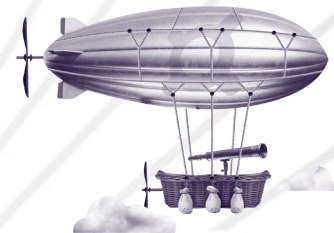
function App() {
  return <Component />
}
```

Running Tests

To run tests, run the following command

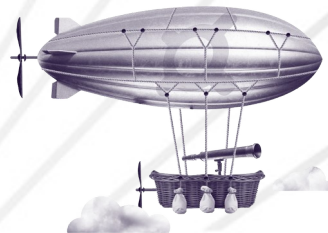
```
npm run test
```

Documentation



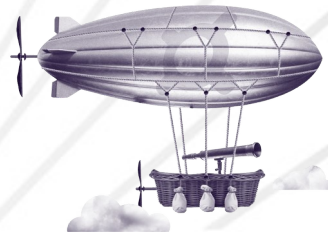
2 - Configuration

- **Paramétrages et spécificités** liées à **l'environnement** doivent absolument être gérés à part.
- La répétition dans le code de chemins absolus du type « C:\User\Marco\Program Files » est un travers très présent



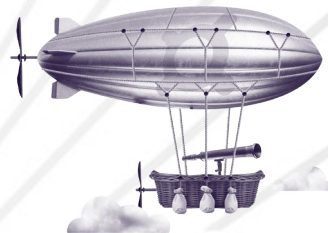
2 - Configuration

- Si vous ne voyez pas le bénéfice immédiat d'un fichier de configuration faites moi confiance, il y en aura.
- Premier bénéfice : On ne touche plus au code pour changer des éléments de réglages



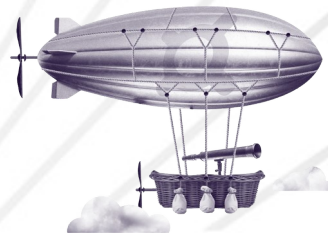
2 - Configuration

- Spécificités liés à l'environnement
 - Des chemins vers des fichiers ou dossiers
 - Des urls externes
 - Des identifiants, clefs, etc.
- Paramétrages
 - Oui ! Facilitez vos modifications de facteurs, coefficients, et autres matrices, dissociez-les du code !



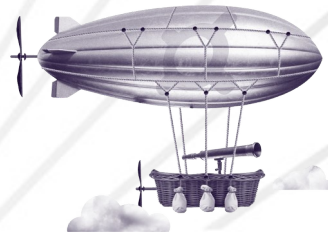
2 - Configuration

- Pourquoi ?
 - Pour pouvoir s'installer ailleurs
 - Pour centraliser en un seul endroit tous ces éléments éditables
 - Pour les tests (on en reparlera)
 - Pour le style (on en reparlera)
 - Pour sauver des configurations qui marchent



2 - Configuration

- Comment ?
 - Package **config**
 - Fichier config.yml (YAML)
 - Multi environnement
 - Possibilité d'expressions R



default:

```
debug: false
workdir: "src/"
dataDir: "../data"
current_data: "dpt_2000_2021_csv"
all_departments_numbers: !expr c("01", "02", "03", "04", "05", "06", "07", | "08", "09", seq(10, 95), seq(971, 974), "976")
coef_aggregation: 0.025
```

dev:

```
debug: true
workdir: "/home/rle/src/R/Zorglub/src"
coef_aggregation: 0.045
```

production:

```
debug: !expr Sys.getenv("ENABLE_DEBUG")
```

test1:

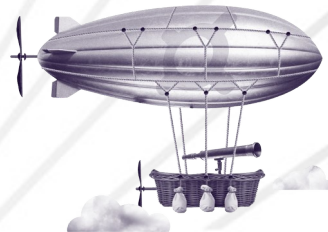
```
dataDir: "../tests/data/A"
current_data: "tcsv"
all_departments_numbers: !expr c("01", "976")
coef_aggregation: 0.5
```

test2:

```
dataDir: "../tests/data/A"
current_data: "tcsv"
all_departments_numbers: !expr c("01", "976")
```

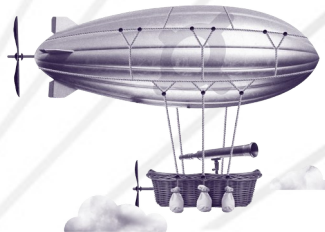
3 – Gestion des dépendances

- Si le programme ne tourne que sur une seule machine, et qu'on arrive plus à le déplacer, il disparaîtra avec cette machine.



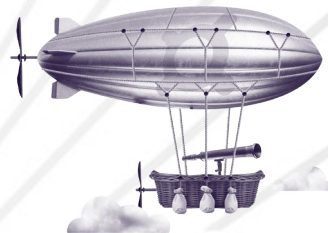
3 – Gestion des dépendances

- Le principal frein au redéploiement d'un programme R dans un laboratoire scientifique
 - « on a pas réussi à réinstaller R avec les packages nécessaires »
- C'est un problème **très** sérieux



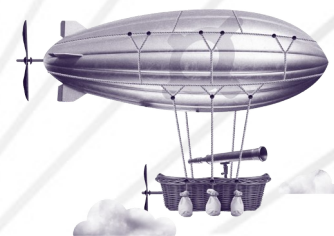
3 – Gestion des dépendances

- D'autres langages disposent d'outils plus avancés que R :
 - composer.json / composer.lock en PHP
 - package.lock / package.json en Nodejs
 - un fichier **déclaratif** (humain), un fichier de résultat (calculé)
- Déclarations avancés (version ≥ 8.05 < 10.3)
- Recherche Opérationnelle (graphes) sur la résolution de l'arbre de dépendances



3 – Gestion des dépendances

- Il existe des outils avancés en R, destinés aux mainteneurs de packages
- Pour l'utilisateur 'classique', il faut connaître l'un des deux outils
 - **Packrat**
 - **Renv** (plus récent)



- On les retrouve de base dans RStudio (en fonction de la version)

é sans AUCUNE GARANTIE.
sous certaines conditions
e()'

avec
plus
le le

instra
pour c

New Project

Back

Create New Project

Directory name:
projet_42

Create project as subdirectory of:
~/src/R

☒ Create a git repository

☒ Use packrat with this project

New Project Wizard

Back

Create New Project

Directory name:
Zorglub

Create project as subdirectory of:
~/src/R

☒ Create a git repository

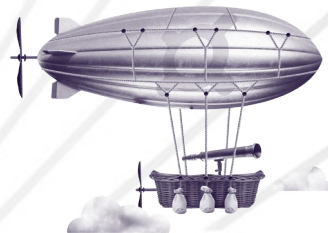
☒ Use renv with this project



3 – Gestion des dépendances

```
'install.packages("remotes")  
library(remotes)  
install_version("MASS", "7.3.57")  
install_version("lattice", "0.20-44")  
renv::install("ggplot2")
```

- L'idée principale :
 - Installez vos packages
 - Résolvez vous mêmes vos conflits de dépendances quand ils arrivent
 - Photographiez le résultat (sauvé dans le « .lock »)
 - Partagez cette photo en partageant ce fichier de lock
 - Faites évoluer le fichier lock quand de nouveaux problèmes arrivent



3 – Gestion des dépendances

- Exemple de restauration

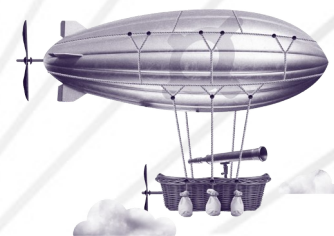
```
* Project '~/src/R/Zorglub' loaded. [renv 0.17.3]
* One or more packages recorded in the lockfile are not installed.
* Use `renv::status()` for more details.
[Sauvegarde de la session précédente restaurée]
```

```
> renv::status()
The following packages are recorded in the lockfile, but not installed:
```

vroom	[1.6.3]
tzdb	[0.4.0]
ggplot2	[3.4.2]
readr	[2.1.4]
labeling	[0.4.2]

Use `renv::restore()` to restore the packages recorded in the lockfile.

```
> renv::restore()
```

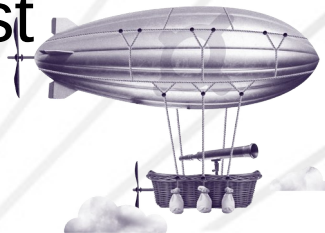


3 – Gestion des dépendances

- La deuxième idée
 - Les packages sont installés **pour ce projet uniquement**, dans ou sous-dossier

Ce qui nous amène au deuxième aspect problème:

- **ISOLEZ** vos dépendances
 - Gérer des dépendances sur plusieurs projets est intenable

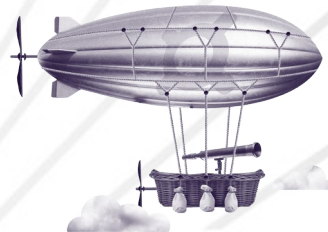


3 – Gestion des dépendances

Environnement Virtuel

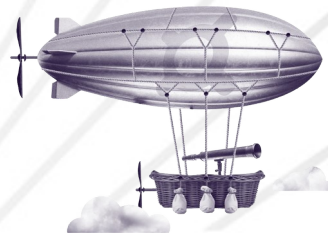
En **alternative** ou **complément** à Renv/packrat

- Utilisez des environnements virtuels pour isoler les packages du système
- Testez facilement plusieurs versions de R
 - **conda**
 - **anaconda**



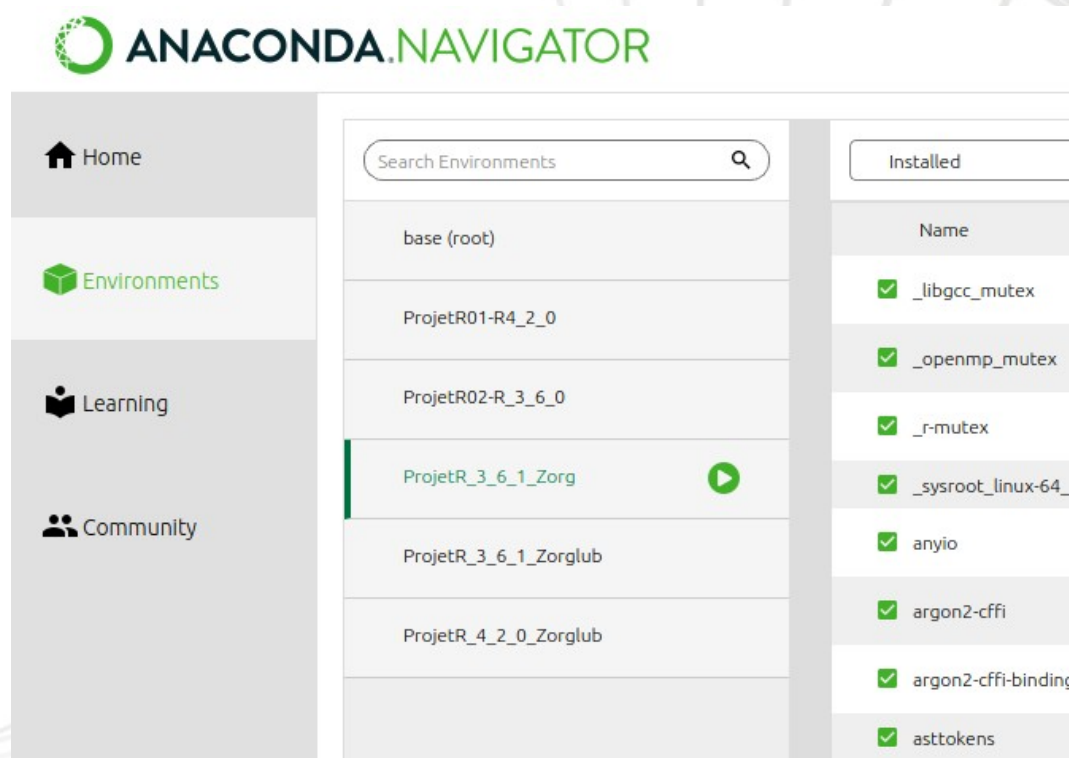
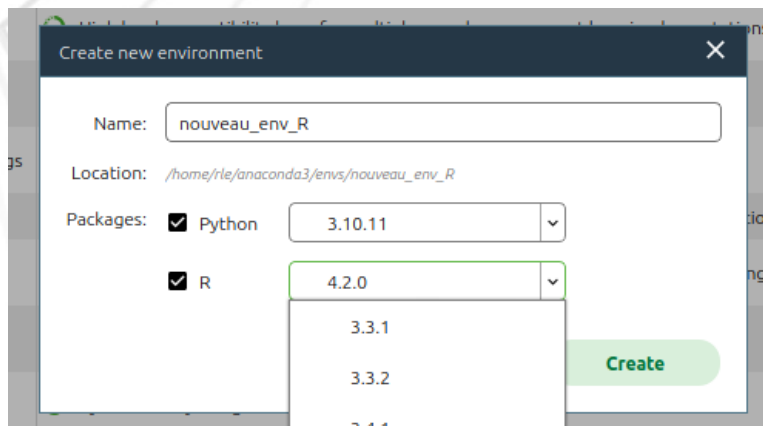
Environnement Virtuel

- **Anaconda** : très utilisé dans le domaine de l'intelligence artificielle
- Navigateur graphique
- Intégration avec Jupyter NotebookR, DataSpell, Spyder, etc.
- R Studio est figé dans une ancienne version (v11,456) et Rstudio ne 'supporte' pas Anaconda



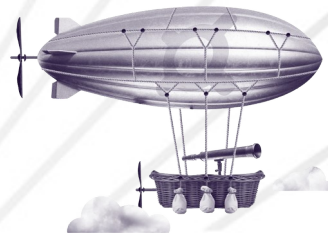
Environnement Virtuel

- Derrière vous avez plusieurs environnements « conda »



Environnement Virtuel

- Vous pouvez à la place gérer vos environnements conda « à la main »
- Testez que votre projet se réinstalle sans attendre de changer de PC.
- Passer à docker c'est déjà beaucoup plus compliqué, gardez cela pour plus tard



(base) rle@nii4:~\$ conda search R

Loading channels: done

#	Name	Version	Build	Channel
r		3.3.1	r3.3.1_0	pkgs/r
r		3.3.1	r3.3.1_1	pkgs/r
r		3.3.2	r3.3.2_0	pkgs/r
r		3.4.1	r3.4.1_0	pkgs/r
r		3.4.2	h65d9972_0	pkgs/r
r		3.4.3	mro343_0	pkgs/r
r		3.4.3	r343_0	pkgs/r
r		3.5.0	mro350_0	pkgs/r
r		3.5.0	r350_0	pkgs/r
r		3.5.1	mro351_0	pkgs/r
r		3.5.1	r351_0	pkgs/r
r		3.6.0	r36_0	pkgs/r

(base) rle@nii4:~\$ conda create --name testR3-5-1 r=3.5.1


```
#  
# To activate this environment, use  
#  
#     $ conda activate testR3-5-1  
#  
# To deactivate an active environment, use  
#  
#     $ conda deactivate
```

```
(base) rle@nii4:~$ conda activate testR3-5-1  
(testR3-5-1) rle@nii4:~$ R --version  
R version 3.5.1 (2018-07-02) -- "Feather Spray"  
Copyright (C) 2018 The R Foundation for Statistical Computing  
Platform: x86_64-conda_cos6-linux-gnu (64-bit)
```

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
<http://www.gnu.org/licenses/>.

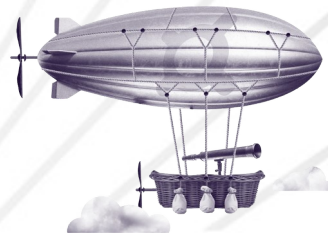


Conda Jupyter

- Testez par exemple un Jupyter notebook

```
(testR3-5-1) rle@nii4:~$ conda install r-essentials r-base jupyterlab
```

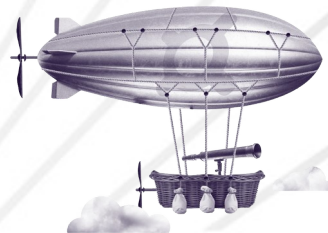
- Maîtriser votre version de R en ligne de commande sera aisée
- Du côté de R Studio c'est plus compliqué... peut être moins avec les versions payantes.



4 - Pas de modifications manuelles

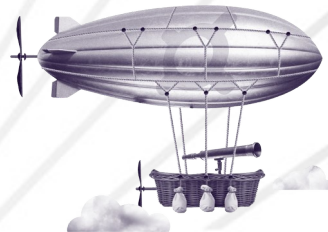
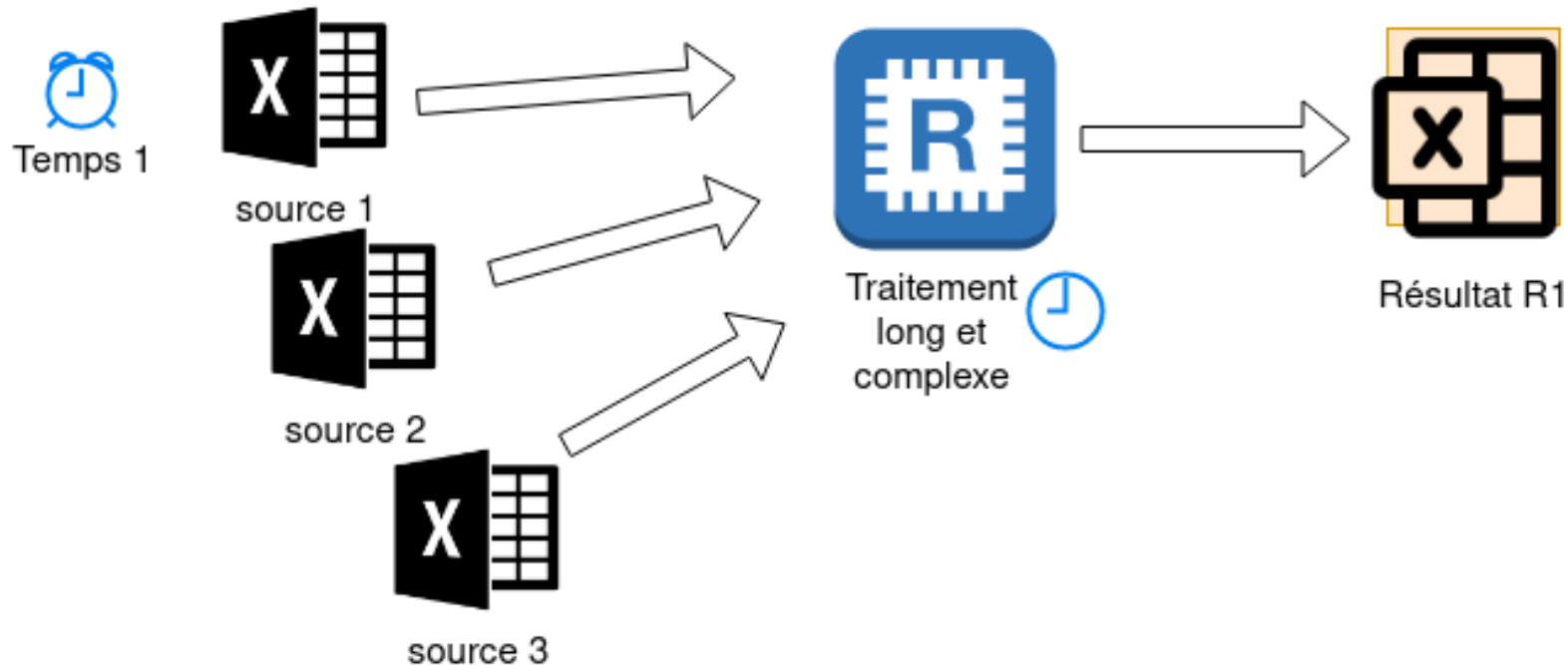
- Ce « truc » est en fait très très simple

Ne modifiez pas manuellement le résultat du programme.



4 - Pas de modifications manuelles

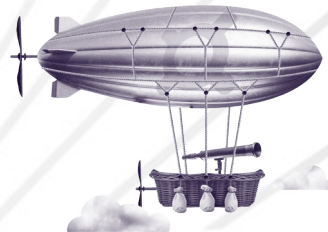
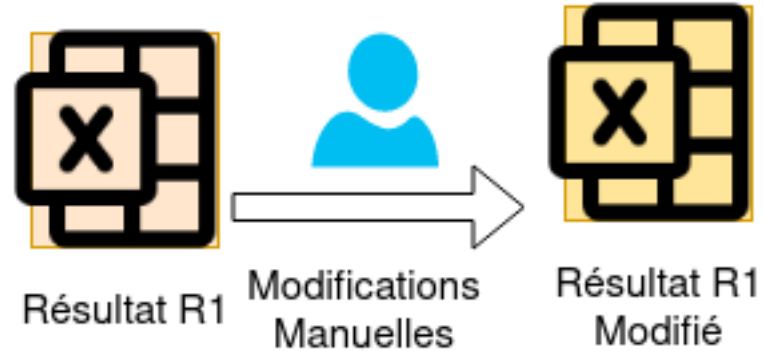
- Cas classique :



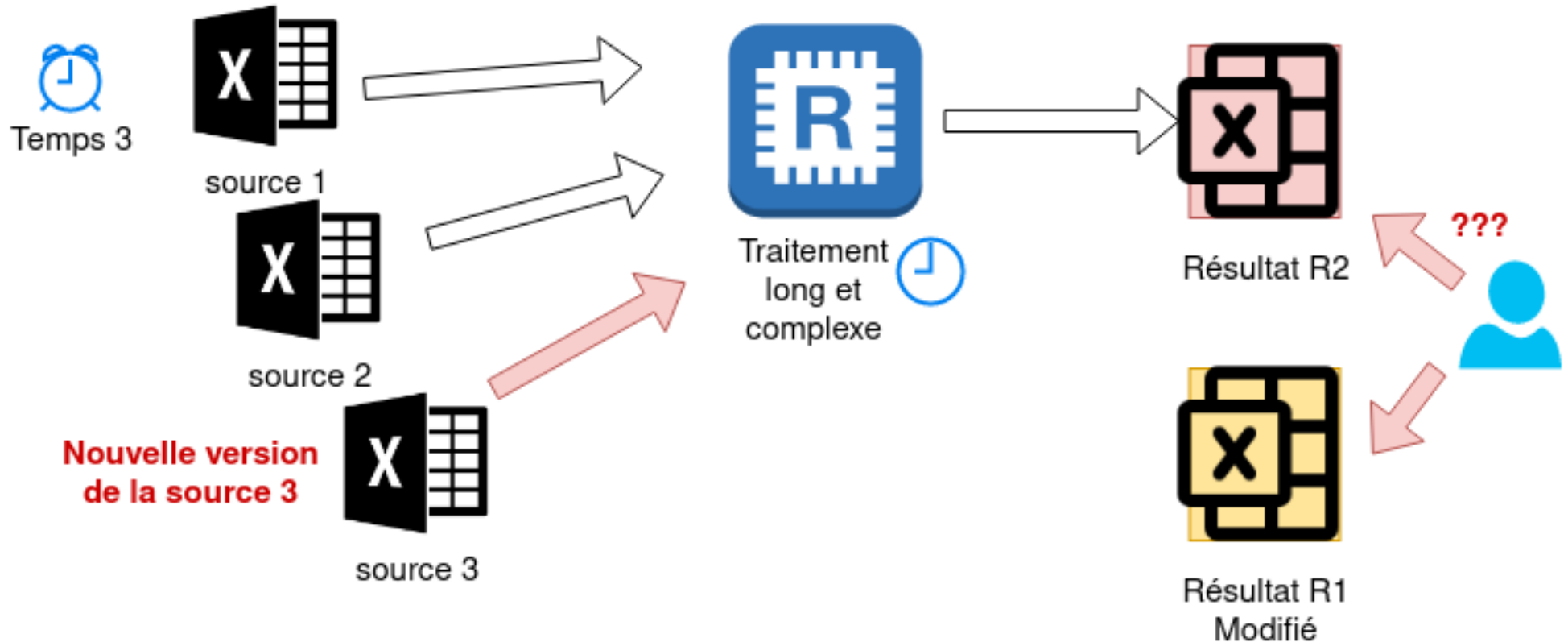
4 - Pas de modifications manuelles

- Cas classique :

NON

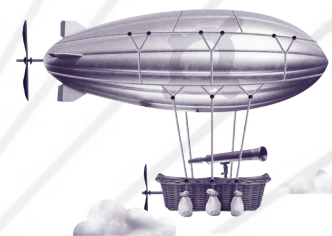
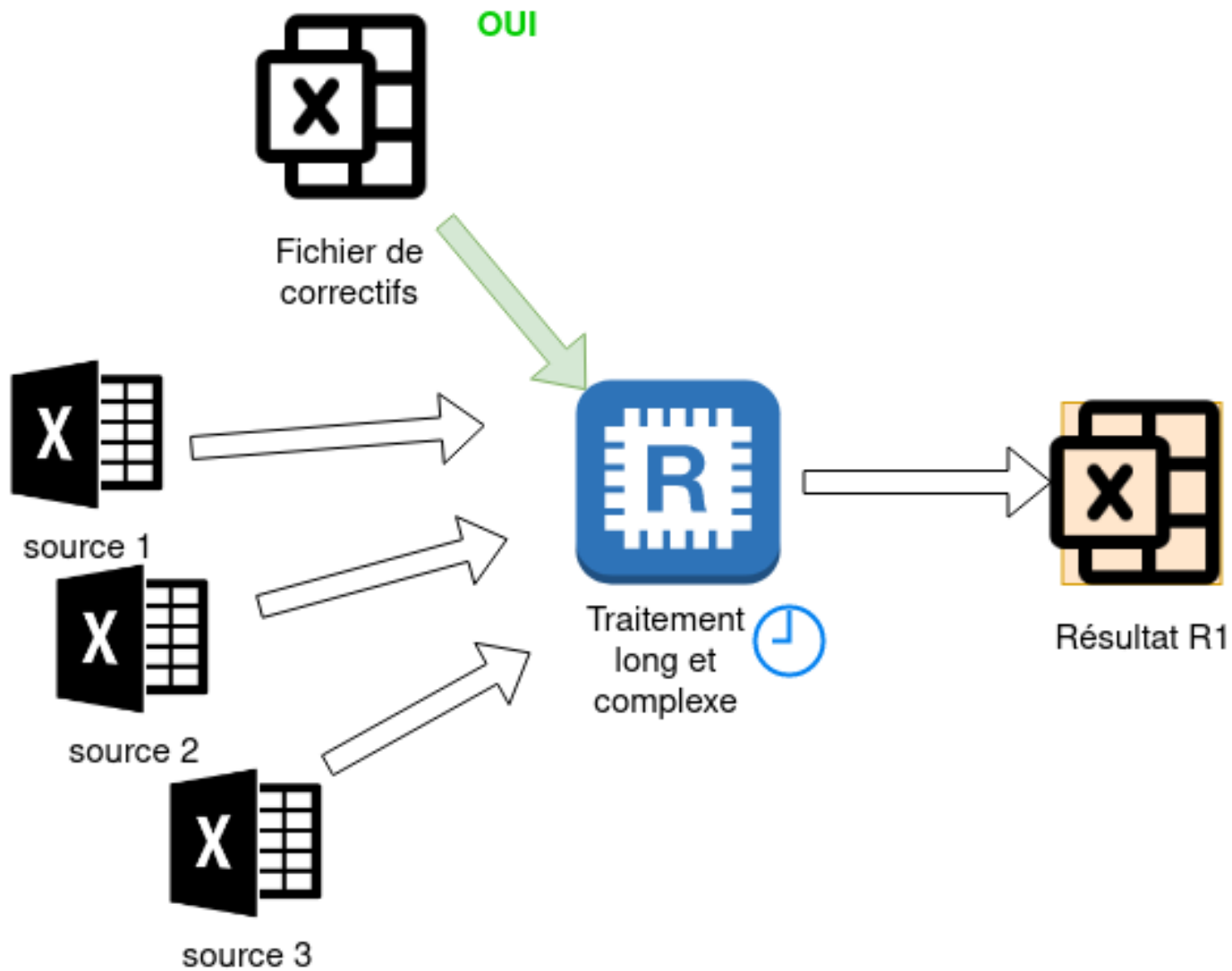


4 - Pas de modifications manuelles



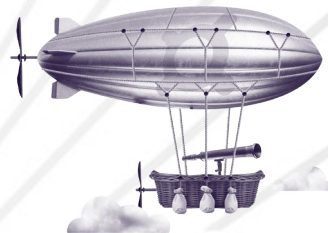


Temps 2



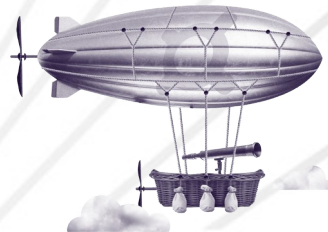
4 - Pas de modifications manuelles

- Les modifications font partie des entrées
 - On peut commencer simple : lignes de données complètes
 - On peut penser à un fichier d'ordres avec un langage d'ordre plus ou moins simple
 - On peut étendre ce principe (génération de rapport markdown, etc.)



5 – Un peu de style

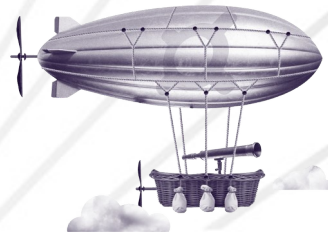
- Quelques petites règles :
 - Pour avoir du style
 - Pour ne pas découvrir trop tard des choses que des milliers d'informaticiens ont mis du temps à découvrir



5 – Un peu de Style

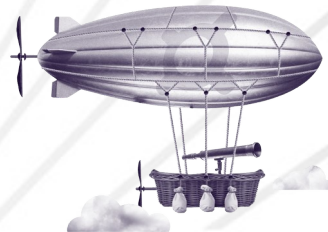
Fausse pistes

- Il y a des choses qui vous donneront faussement l'impression d'être un bon informaticien
 - Trouver des noms de variables concis (a, b, b1, b2, cx, crt)
 - Écrire la ligne la plus longue
 - Coder à deux sur le même clavier



5 – Un peu de style

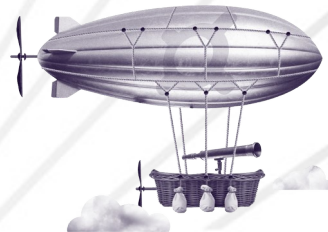
- Lisez les « Style Guides »
 - <https://style.tidyverse.org/>
 - <https://google.github.io/styleguide/Rguide.html>
- Cela améliore la **lisibilité**
- Cela prévient beaucoup de **bugs**



5 – Un peu de style

- Linter : forcer un style
 - Package lintr
 - Fichier .lintr

```
lintr: linters_with_defaults(  
  line_length_linter(100),  
  commented_code_linter = NULL  
)|  
exclusions: list(  
  "src/extracteur_orig.r"  
)  
encoding: "UTF-8"
```



5 – Un peu de style

- Linter : exemple `lintr::lint_dir()`

src/extracteur.r:97:39: style: [function_left_parentheses_linter] Remove spaces before the left parenthesis in a function call.

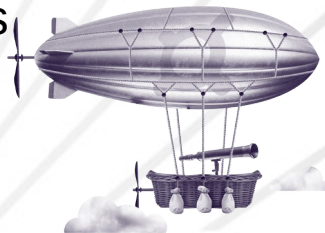
```
df_both <- df_global %>% inner_join (df_dept, by=c("name" = "name"))
                                   ^
```

src/extracteur.r:97:52: style: [infix_spaces_linter] Put spaces around all infix operators.

```
df_both <- df_global %>% inner_join (df_dept, by=c("name" = "name"))
                                   ^
```

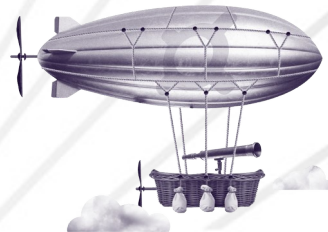
src/extracteur.r:100:64: style: [trailing_whitespace_linter] Trailing whitespace is superfluous.

```
df_both$ecart_local > config$filter_percent_diff_global &
```



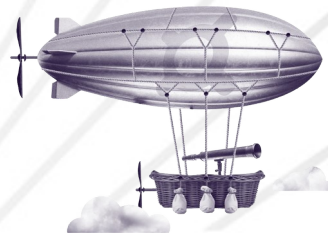
5 – Un peu de style

- Bien Nommer ses variables
 - Perdre du temps à réfléchir aux noms est utile
 - C'est un art difficile
 - Lisibilité et maintenabilité
- Ce qui se conçoit bien s'exprime clairement
- Astuce : oralisez



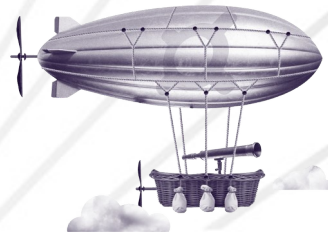
5 – Un peu de style

- **Règle de 3** pour les copier/coller
- Elle marche autant si vous êtes trop « informaticien » (tendance à factoriser et à reconnaître les schémas) ou pas assez
 - copier/coller plusieurs fois n'est pas rédhibitoire
 - Après 3 copier/coller/adaptation il est temps de penser à factoriser
 - faire une fonction avec des paramètres?
 - Réfléchir à une solution architecturale ?



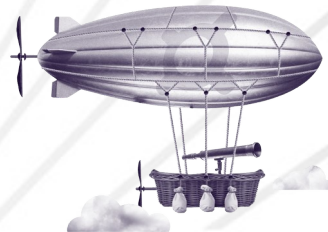
5 – Un peu de style

- **KISS** : Keep It Stupid Simple
 - Décomposez en fonctions simples (et **courtes**!)
 - Décomposez en fichiers différents
 - Chacun sa tâche
- Plus facile à faire évoluer, combiner et tester



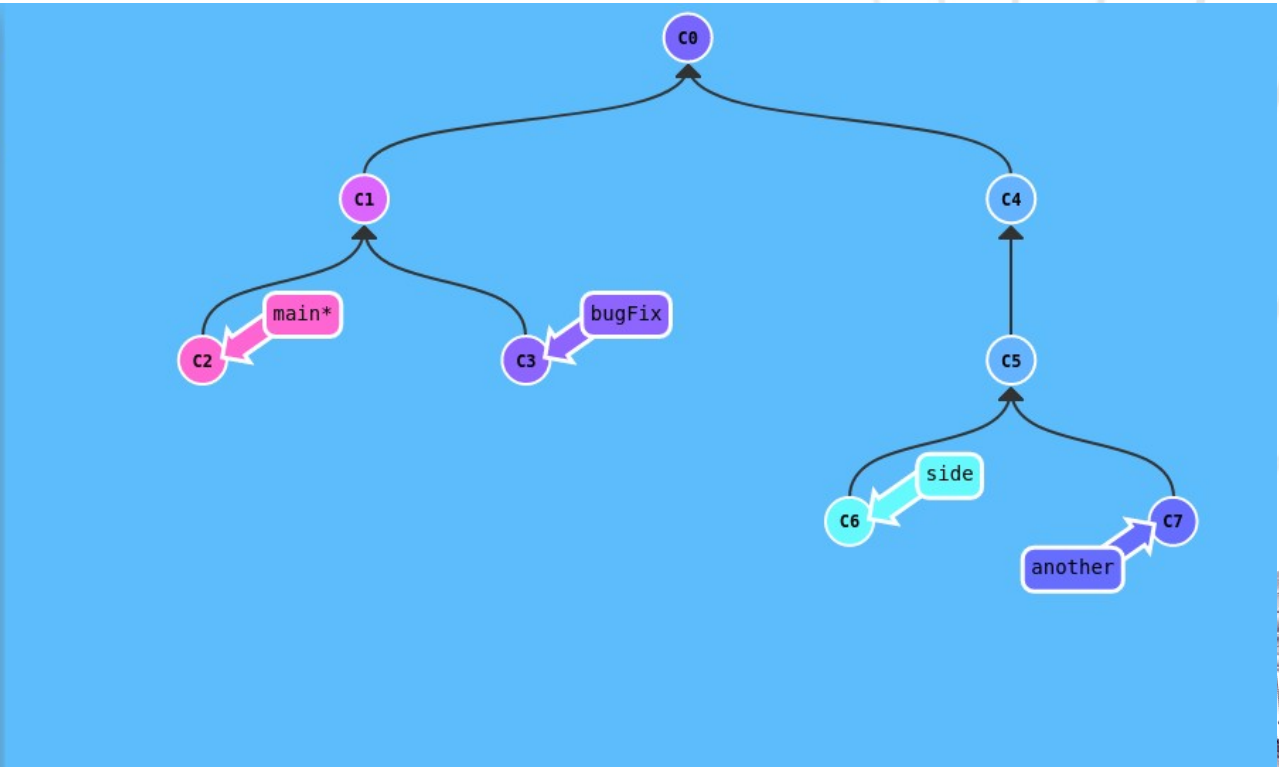
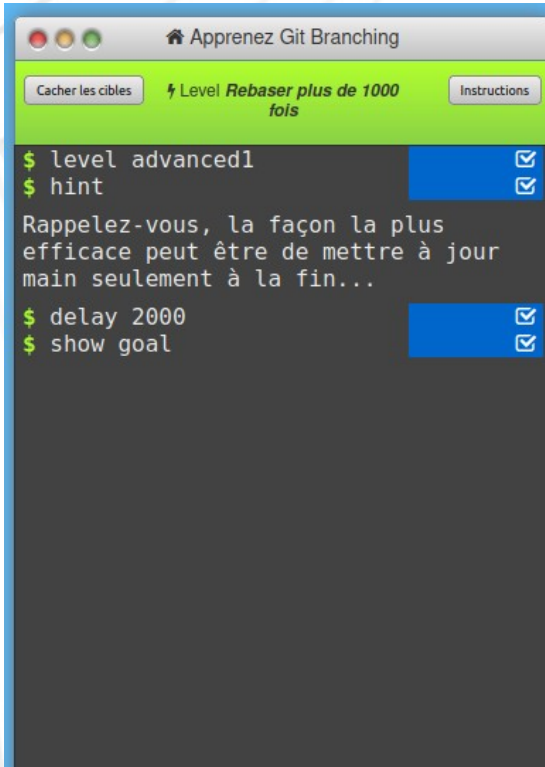
6 - GIT

- GIT c'est **bien**
- Perdre un peu de temps et d'énergie à comprendre GIT ce n'est jamais perdu



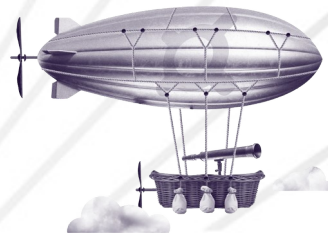
6 - GIT

- <https://learngitbranching.js.org/>



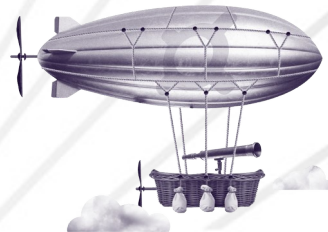
6 - GIT

- Rstudio propose git à la création d'un projet
- On peut ajouter git à n'importe quel moment
- « git init »
- **Pas besoin d'un dépôt distant** comme Github
- Git fonctionne en local
 - Le dépôt distant c'est mieux pour la sauvegarde et la collaboration mais c'est secondaire



6 - GIT

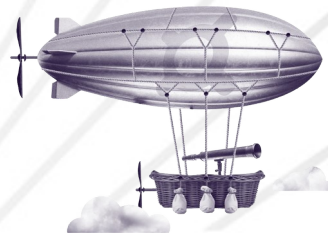
- Pourquoi ?
 - Libérer son esprit, tester des choses
 - Éviter de garder des longs morceaux de code 'historique' commenté
 - Relire et valider les modifications



6 - GIT

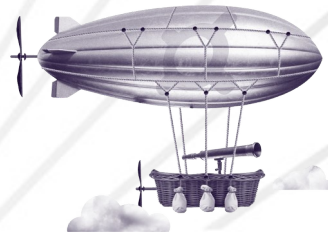
- Choses à apprendre pour commencer
 - .gitignore (vous ne voulez pas sauver vos datas)

```
.Rproj.user  
.Rhistory  
.RData  
.Ruserdata  
data/  
!data/README.txt  
!
```
 - commiter
 - git add -p (relecture de modifications)



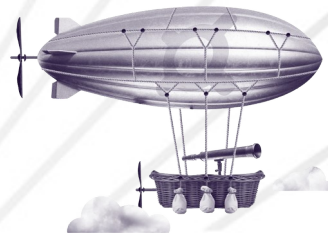
7 - Tester

- Tester ça paraît compliqué, inutile et long
- Si vous respectez les autres règles (fichiers de configuration et KISS surtout) c'est en fait assez simple



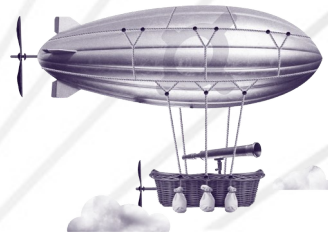
7 - Tester

- Tester apporte plusieurs choses
 - Cela documente votre code, ce qu'il a prévu (ou oublié)
 - Cela protège votre code des évolutions futures (qui devront valider les tests ou bien les adapter en connaissance de cause)
 - Cela permet la découverte de bugs et d'erreurs de conception AVANT le drame



7 - Tester

- Comment ?
 - Il y a des outils, des packages.
 - Mais ce n'est pas l'important
- Arrangez vous pour que les parties clefs de l'application puissent être testées
 - Avec des données en entrée
 - Et des résultats attendus



7 - Tester

- Tester « toto » ça marche, mais **tester que les choses simples marchent c'est pas du jeu**
- Soyez **malicieux**
 - Chaîne vide
 - Null
 - Zéro
 - Absence
 - Encodages
 - Répétitions
 - Dépassements de taille
 - Injection de grammaire
 - ...

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| " : ; , ? : @ = ` & / \ u 6 a 1 9 ' \ |
| " : ; , ? : @ = ` & / æ " ™ ' \
```



Pour aller plus loin

- **Makina Corpus** dispense diverses formations, le nouveau catalogue contient une formation nommée « **Bonnes pratiques de développement** » spécialement destinée aux universitaires ou data scientists
 - orientée sur Python et R
 - avec l'objectif de pratiquer les conseils indiqués ici
 - Et d'autres (packaging python, debugging, code reproductible, communautés, etc.)

