# Développer ses applications et partager ses résultats avec R et GitLab

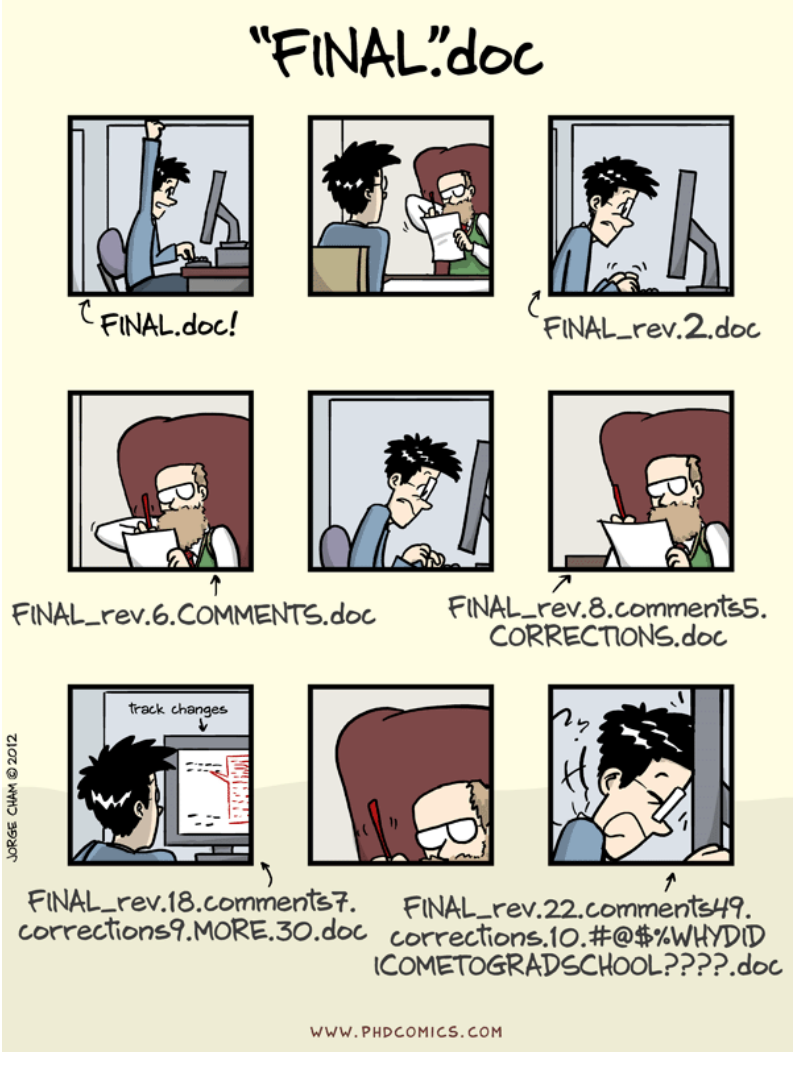
## Retours sur les solutions disponibles actuellement

Cédric Midoux — INRAE, PROSE & MaIAGE

Café des sciences BioSP — 2022-02-07

# Versionner & Déployer

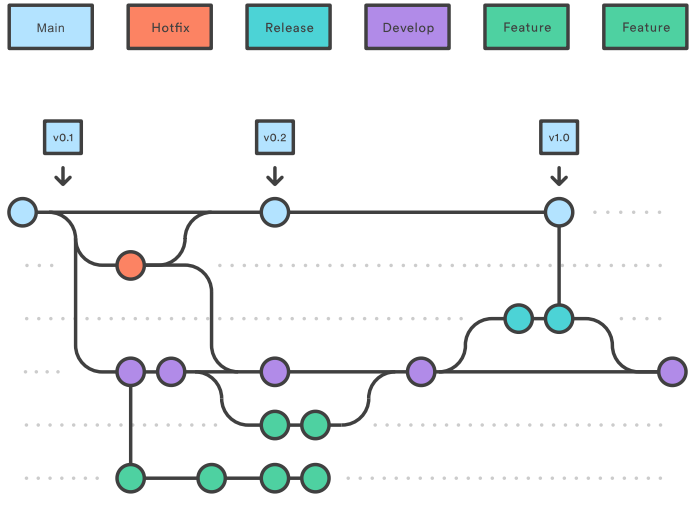
# Pourquoi versionner ses projets ?




→ Suivre l'évolution des fichiers

→ Faciliter le développement collaboratif

→ Revenir à une version précédente



# Git : concepts et vocabulaire (1/2)


- **repository** : dépôt. Un dossier contenant tous les fichiers d'un projet ainsi que l'historique de révision. Un dépôt peut être personnel ou partagé. Il peut être public ou privé. Il peut être local ou distant (remote).
- **commit** : révision. Modification d'un ou plusieurs fichier du dépôt. Lorsque vous effectuez un commit pour enregistrer votre travail, Git crée un identifiant unique (`SHA`) qui vous permet de garder une trace des modifications spécifiques réalisées ainsi que de leur auteur et de leur date. On associe généralement un court descriptif à un commit.
- **branch** : enchaînement de commit. Les branches permettent de faire des développements parallèles qui n'affectent pas les autres branches.
- **main** : branche de développement par défaut. Les projets historiques utilisent le terme `master`.
- **HEAD** : commit le plus récent, à la tête de la branche.

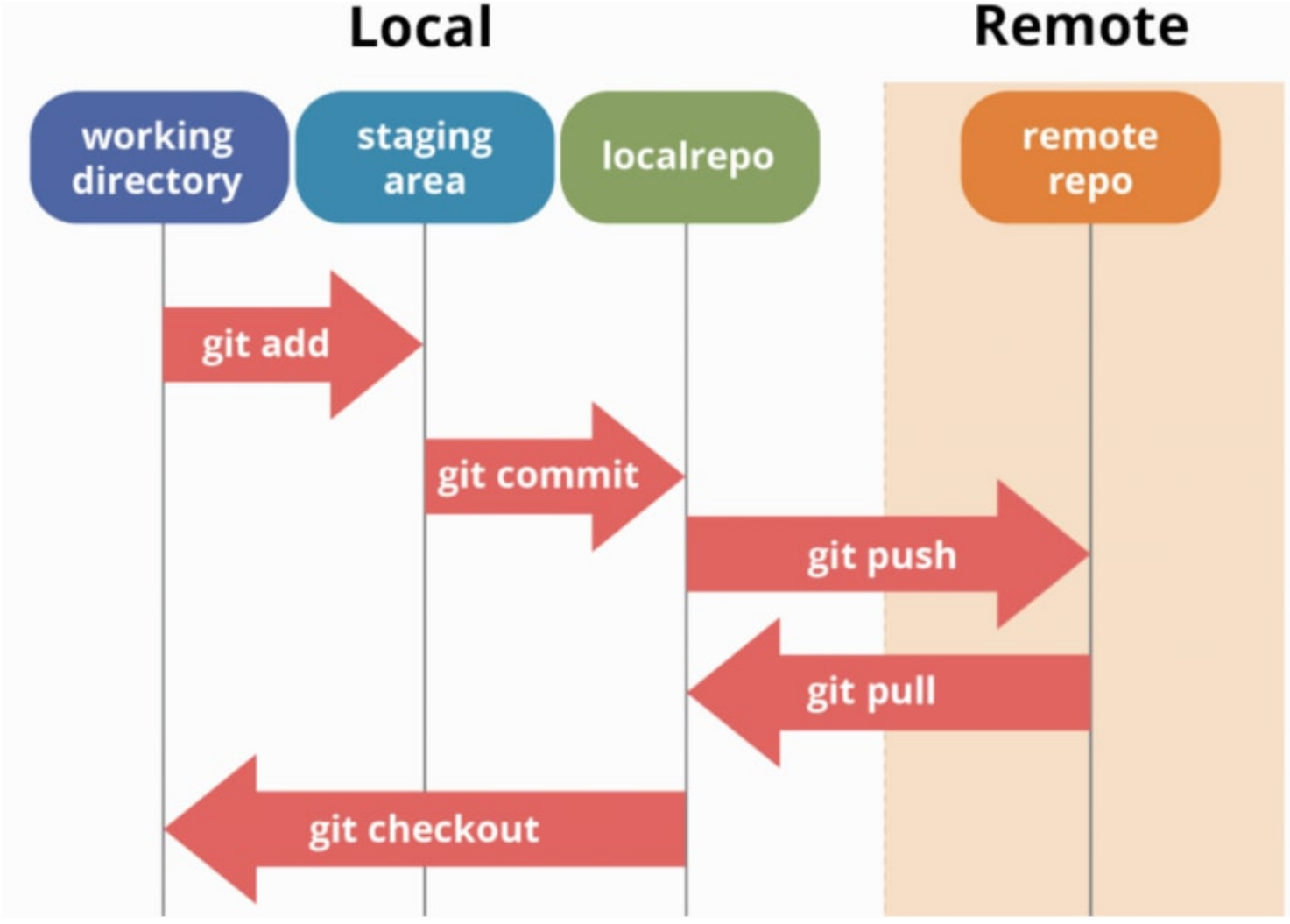
# Git : concepts et vocabulaire (2/2)

- **clone** : Copie local d'un dépôt. Le dépôt cloné est toujours connecté à la version distante de sorte qu'il soit possible de synchroniser facilement (push/pull/...).
- **pull** : récupérer les commits distants et les fusionner avec la version locale.
- **push** : pousser ses commits vers un dépôt distant, afin que d'autres puissent y accéder. Permet également d'assurer une sauvegarde distante.
- **merge** : fusion. Appliquer les commits d'une branche à une autre branche. Si il y a des conflits, il faut les résoudre avant la fusion.
- **checkout** : changer de branche de travail actuelle. Permet de changer de branche ou bien de revenir à une version antérieure.
- **staging area** : index. Espace de travail qui servira de base pour le prochain commit.

> **Happy Git and GitHub for the useR**

> GitHub Docs
> GitHub cheat-sheet
> Git cheat-sheet (ndpsoftware)

# Git flow


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout

# Démo

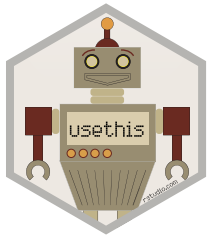
1. Création d'un dépôt avec GitLab

2. Clone avec RStudio

3. Modification et commit

4. git pull / git push

# Conseils pour utiliser git avec R (et RStudio)




- Cloner un projet :
  `File` > `New Project` > `Version Control` > `Git` > URL (SSH ou HTTPS)
- Initier un projet

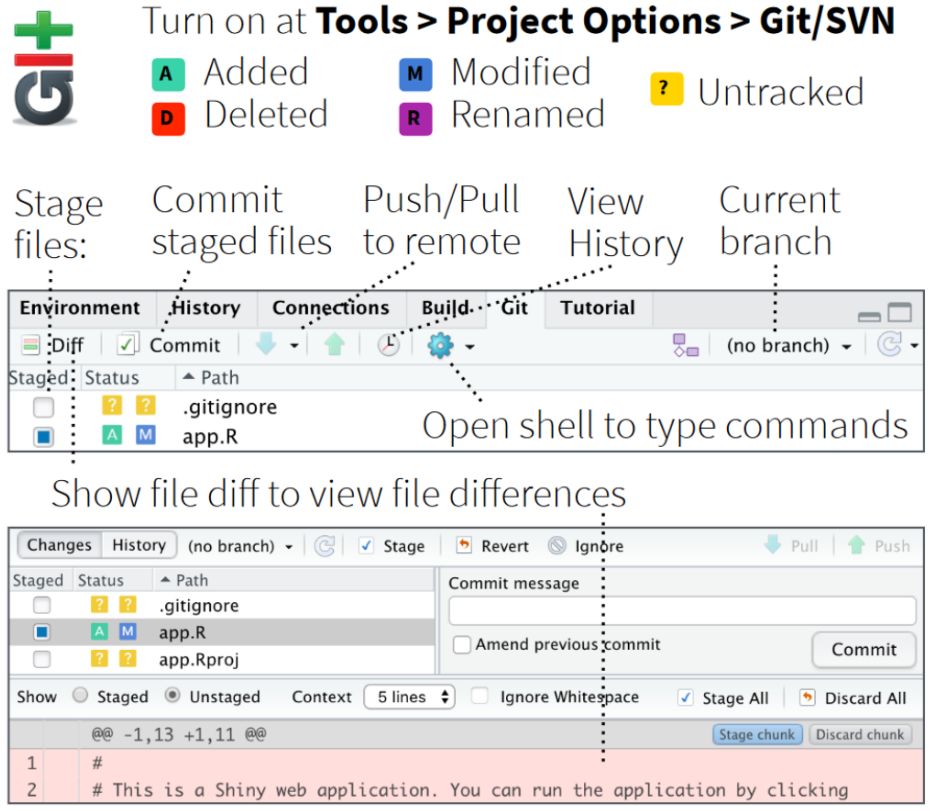
```
usethis::use_git()
```

- Personaliser son `git config`

```
usethis::use_git_config(user.name = "Jane", u
```

- Générer un `.gitignore`

```
usethis::git_vaccinate()
```



# Rédiger un document avec R Markdown

`rmarkdown` permet de regrouper de créer dynamiquement des documents comprenant du **code**, des **résultats** (tel que des plots) et de la **prose** (les commentaires et explications).

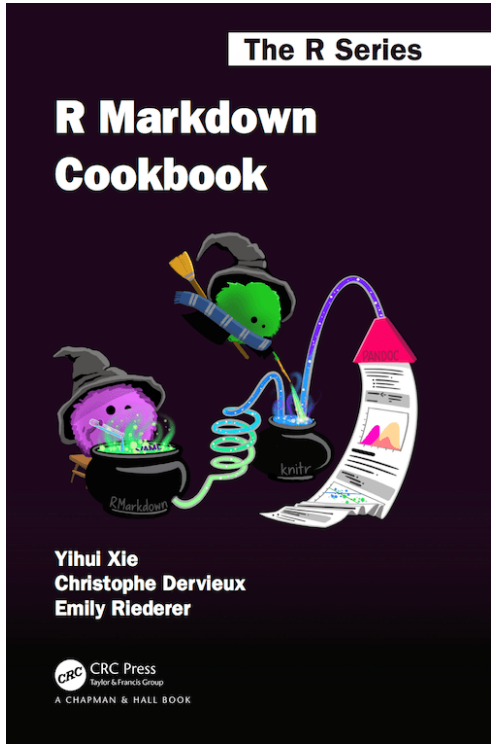
Ceci est utile pour :

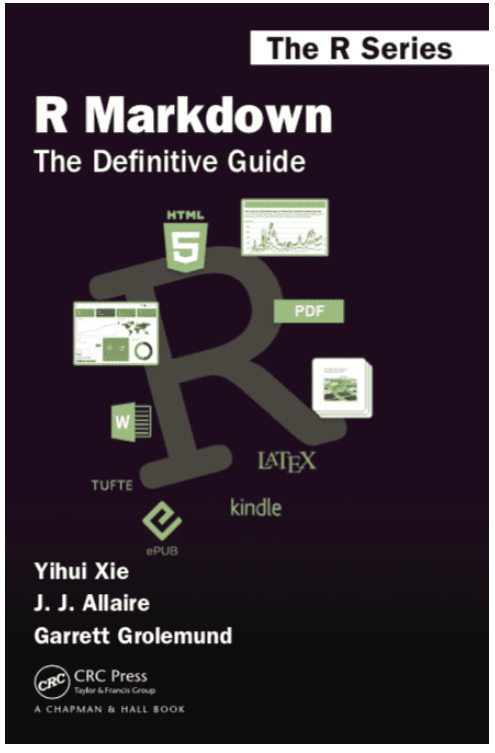
- Faire des analyses de DataScience de manière interactive (avec RStudio par exemple)
- Reproduire ses analyses
- Collaborer et partager son code
- Communiquer ses résultats à des collègues pas nécessairement à l'aise avec R.

Les métadonnées du documents sont regroupées dans un en-tête YAML.

Les documents R Markdown peuvent être compilés en HTML, PDF, ...

On utilise l'extension `.Rmd` ou `.rmd`.




> R Markdown Cookbook

> R Markdown: The Definitive Guide

# Démo

1. Création d'un document (dans notre dépôt Git)

2. RStudio visual editor

3. *Code chunks*

4. Ajout de tables avec `kable` et `DT`

5. Ajout d'une table des matières (`toc: true`)

6. Bloc de code en python (nécéssite des packages dédiés)

7. `Knit` pour exporter en HTML

# Partager un HTML grace aux GitLab Pages


Avec GitLab Pages, on peut publier facilement des sites web statiques directement à partir d'un dépôt GitLab.

Il suffit d'ajouter un template d'intégration continue (`CI/CD pipelines`).

Pour chaque modification poussée sur le dépôt, GitLab CI/CD exécute le pipeline qui publie immédiatement les modifications sur le site.

Possibilité d'ajouter un domaine dédié en `.inrae.fr` (ou autre)

⚠️ Si `<user>` comprend un point `.`, celui-ci sera considéré comme un sous domaine double. N'ayant pas de certificat dédié, le navigateur demandera une exception de sécurité. (par exemple `cedric.midoux`)

## Démo

**Sur GitLab** :

1. Ajouter un nouveau fichier `Add(+)`
2. Sélectionner un template `.gitlab-ci.yml`
3. Appliquer le template `Pages - HTML`
4. Commit

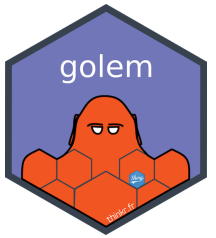
Les paramètres généraux sont disponibles dans `Settings > Pages`

Les paramètres de visibilité sont à détailler dans `Settings > General > Visibility`

Les fichiers sont accessibles à https://<user>.pages.mia.inra.fr/<repos>/<file>.html

# Packager une application Shiny

# golem - Concepts

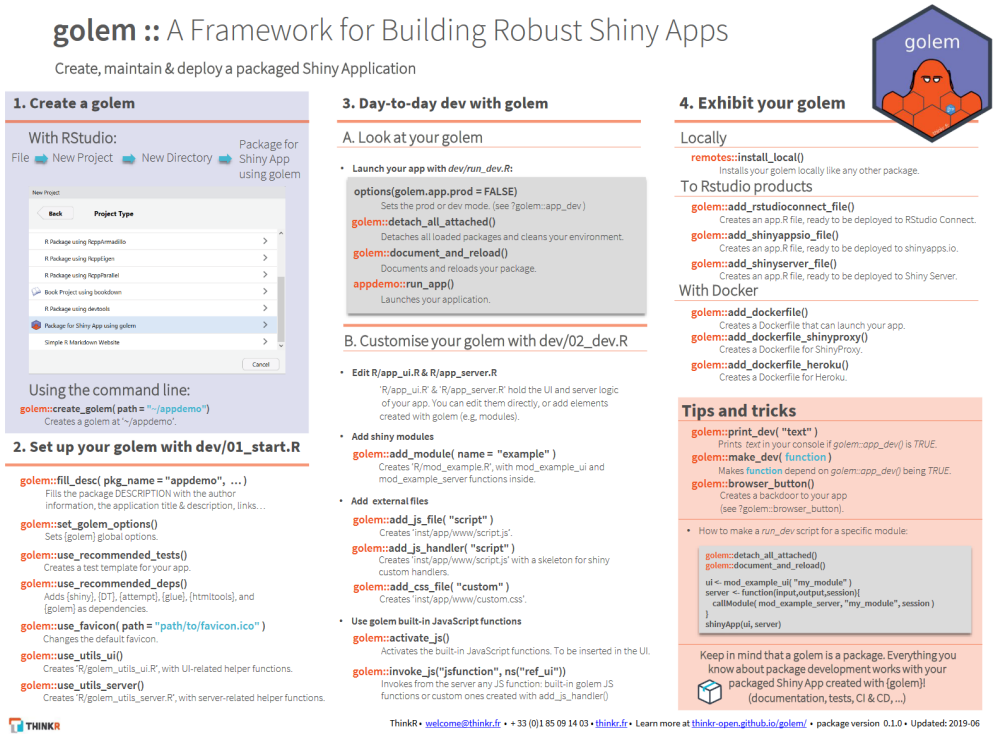



- golem est un framework qui permet de créer, maintenir et déployer facilement une application Shiny.

- Packager une appli dans un package pour :
  - Automatiser ce qui est automatisable
  - Travailler avec des outils fiables
  - Gagner du temps de développement
  - Simplifier le déploiement
  - Standardiser le travail en équipe

→ **Application Shiny dans un package**

- Application découpée en fonctions / modules
- Application testée
- Application avec des dépendances explicites
- Application documentée
- Application disposant de métadonnées


golem :: A Framework for Building Robust Shiny Apps

Create, maintain & deploy a packaged Shiny Application

golem

1. Create a golem

With RStudio:

File → New Project → New Directory → Package for Shiny App using golem

Using the command line:

golem::create_golem( path = "~/appdemo")
Creates a golem at '~/appdemo'.

2. Set up your golem with dev/01_start.R

golem::fill_desc( pkg_name = "appdemo", ... )
Fills the package DESCRIPTION with the author information, the application title & description, links...

golem::set_golem_options()
Sets {golem} global options.

golem::use_recommended_tests()
Creates a test template for your app.

golem::use_recommended_deps()
Adds {shiny}, {DT}, {attempt}, {glue}, {htmltools}, and {golem} as dependencies.

golem::use_favicon( path = "path/to/favicon.ico" )
Changes the default favicon.

golem::use_utils_ui()
Creates 'R/golem_utils_ui.R', with UI-related helper functions.

golem::use_utils_server()
Creates 'R/golem_utils_server.R', with server-related helper functions.

3. Day-to-day dev with golem

A. Look at your golem

- Launch your app with *dev/run_dev.R*:

options(golem.app.prod = FALSE)
Sets the prod or dev mode. (see ?golem::app_dev)
golem::detach_all_attached()
Detaches all loaded packages and cleans your environment.
golem::document_and_reload()
Documents and reloads your package.
appdemo::run_app()
Launches your application.

B. Customise your golem with dev/02_dev.R

- Edit R/app_ui.R & R/app_server.R
  'R/app_ui.R' & 'R/app_server.R' hold the UI and server logic of your app. You can edit them directly, or add elements created with golem (e.g. modules).
- Add shiny modules
  golem::add_module( name = "example" )
  Creates 'R/mod_example.R', with mod_example_ui and mod_example_server functions inside.
- Add external files
  golem::add_js_file( "script" )
  Creates 'inst/app/www/script.js'.
  golem::add_js_handler( "script" )
  Creates 'inst/app/www/script.js' with a skeleton for shiny custom handlers.
  golem::add_css_file( "custom" )
  Creates 'inst/app/www/custom.css'.
- Use golem built-in JavaScript functions
  golem::activate_js()
  Activates the built-in JavaScript functions. To be inserted in the UI.
  golem::invoke_js("jsfunction", ns("ref_ui"))
  Invokes from the server any JS function: built-in golem JS functions or custom ones created with add_js_handler()

4. Exhibit your golem

Locally

remotes::install_local()
Installs your golem locally like any other package.

To Rstudio products

golem::add_rstudioconnect_file()
Creates an app.R file, ready to be deployed to RStudio Connect.
golem::add_shinyappsio_file()
Creates an app.R file, ready to be deployed to shinyapps.io.
golem::add_shinyserver_file()
Creates an app.R file, ready to be deployed to Shiny Server.

With Docker

golem::add_dockerfile()
Creates a Dockerfile that can launch your app.
golem::add_dockerfile_shinyproxy()
Creates a Dockerfile for ShinyProxy.
golem::add_dockerfile_heroku()
Creates a Dockerfile for Heroku.

Tips and tricks

golem::print_dev( "text" )
Prints *text* in your console if *golem::app_dev()* is *TRUE*.
golem::make_dev( function )
Makes function depend on *golem::app_dev()* being *TRUE*.
golem::browser_button()
Creates a backdoor to your app (see ?golem::browser_button).

- How to make a *run_dev* script for a specific module:

```
golem::detach_all_attached()
golem::document_and_reload()

ui <- mod_example_ui( "my_module" )
server <- function(input,output,session){
  callModule( mod_example_server, "my_module", session )
}
shinyApp(ui, server)
```

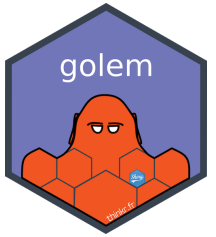
Keep in mind that a golem is a package. Everything you know about package development works with your packaged Shiny App created with {golem}! (documentation, tests, CI & CD, ...)

THINKR

ThinkR • welcome@thinkr.fr • + 33 (0)1 85 09 14 03 • thinkr.fr • Learn more at thinkr-open.github.io/golem/ • package version 0.1.0 • Updated: 2019-06

> golem, YouTube et Cheatsheet

# golem - Structure




```
golem::create_golem(path = "~/appdemo")
```

```
.
├── DESCRIPTION                 # Métadonnées
├── NAMESPACE                   # Fonctions exportées et celles importées
├── R                           # Fonctions
│   ├── app_config.R            #
│   ├── app_server.R            # Remplace la partie server de l'appli
│   ├── app_ui.R                # Remplace la partie ui de l'appli
│   └── run_app.R               # Lance l'application
├── appdemo.Rproj               # R Projet
├── dev                         # Dossier de développement
│   ├── 01_start.R              # Script d'initialisation
│   ├── 02_dev.R                # Script de développement
│   ├── 03_deploy.R             # Script de déploiement
│   └── run_dev.R               # Exécuté par golem::run_dev() pour visualiser l'appli
├── inst
│   ├── app
```

# Démo

1. Création d'un projet golem

```
golem::create_golem(path = "~/appdemo")
```

2. `golem::run_dev()` lance l'appli

3. Édition de l'interface dans `app_ui.R`

4. *Install and Restart*

5. Notre appli est un package qui lance l'appli avec `appdemo::run_app()`

# `dev/01_start.R`: initialisation de l'appli


## Fonctions à lancer (ou pas) début du projet

- `golem::fill_desc()`: remplir le fichier `DESCRIPTION`
- `golem::set_golem_options()`: mise à jour de `inst/golem-config.yml`
- `golem::use_recommended_tests()`: mise en place de tests unitaires
- `golem::use_recommended_deps()`: ajout de dépendances

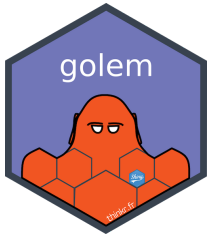
## Fonctions issues de `usethis`:

- `usethis::use_mit_license( "Golem User" )`, `usethis::use_gpl3_()`, ...
- `usethis::use_readme_rmd( open = FALSE )`
- `usethis::use_git()`
- ...

## Autres fonctions à utiliser en connaissance de cause

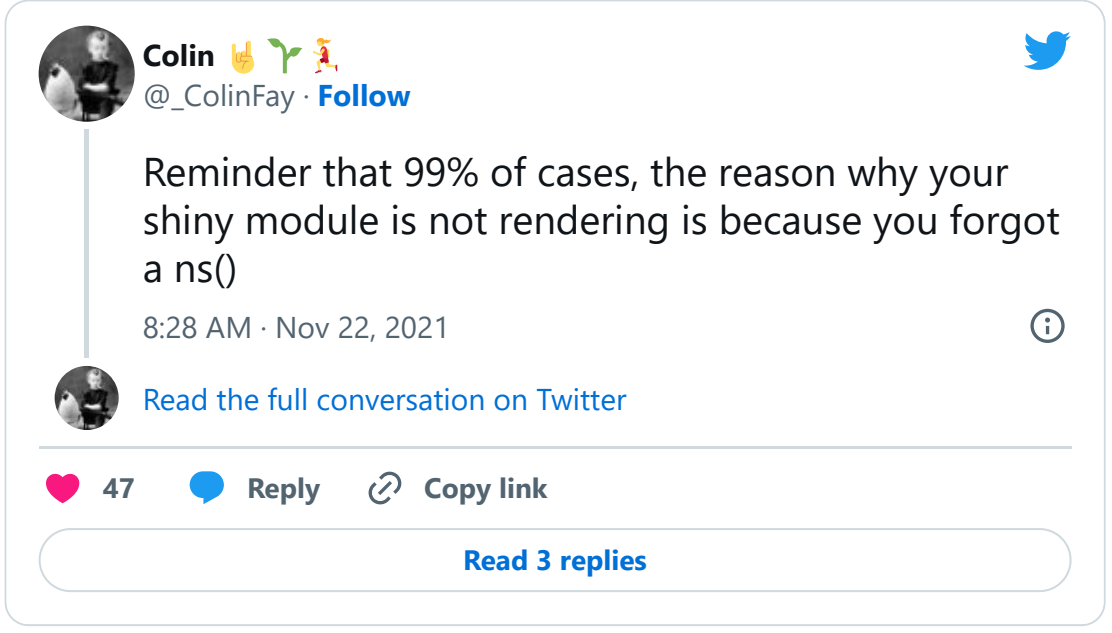
- `golem::use_utils_ui()` et `golem::use_utils_server()`: fonctions utilitaires d'aide au développement

# `dev/02_dev.R`: le développement quotidien




## Modularisation de l'appli: *shiny modules*

- Permettent de découper l'application en petites unités
- Composés d'une partie `server` et d'une partie `ui`
- Permettent de rappeler ces fonctions à plusieurs endroits de l'application
- Ils s'assurent de l'unicité des ID, les identifiants doivent être spécifiés dans des `ns()`



# Démo : module "lancer *1 à 10* dés 6"


## Création du module: `golem::add_module( name = "dice" )`

```
mod_dice_ui <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(inputId = ns("nb"),
      label = "Combien de dés ?",
      min = 1, max = 10, value = 2, step = 1
    ),
    verbatimTextOutput(ns("tirage"))
  )
}
```

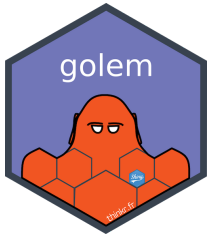
```
mod_dice_server <- function(id) {
  moduleServer(id, function(input, output, se
    ns <- session$ns

    output$tirage <- renderPrint({
      sample(6, size = input$nb, replace = TR
    })
  })
}
```

```
## To be copied in the UI
# mod_dice_ui("dice_1")

## To be copied in the server
# mod_dice_server("dice_1")
```

# dev/03_deploy.R: le déploiement




- Les tests

```
devtools::check()
```

- Construction du package `.tar.gz`

```
devtools::build()
```

## En local

```
remotes::install_local()
```

> golem, YouTube et Cheatsheet
> Engineering Production-Grade Shiny Apps

## Avec RStudio

```
golem::add_rstudioconnect_file()
golem::add_shinyappsio_file()
golem::add_shinyserver_file()
```

## Avec Docker

```
golem::add_dockerfile()
golem::add_dockerfile_shinyproxy()
```

# Tour d'horizon de packages R à (re)découvrir

# `tidyverse`, l'incontournable


Le `tidyverse` est une collection de packages R conçus pour la **DataScience**. Tous les packages partagent une philosophie de conception, une grammaire et des structures de données sous-jacentes.

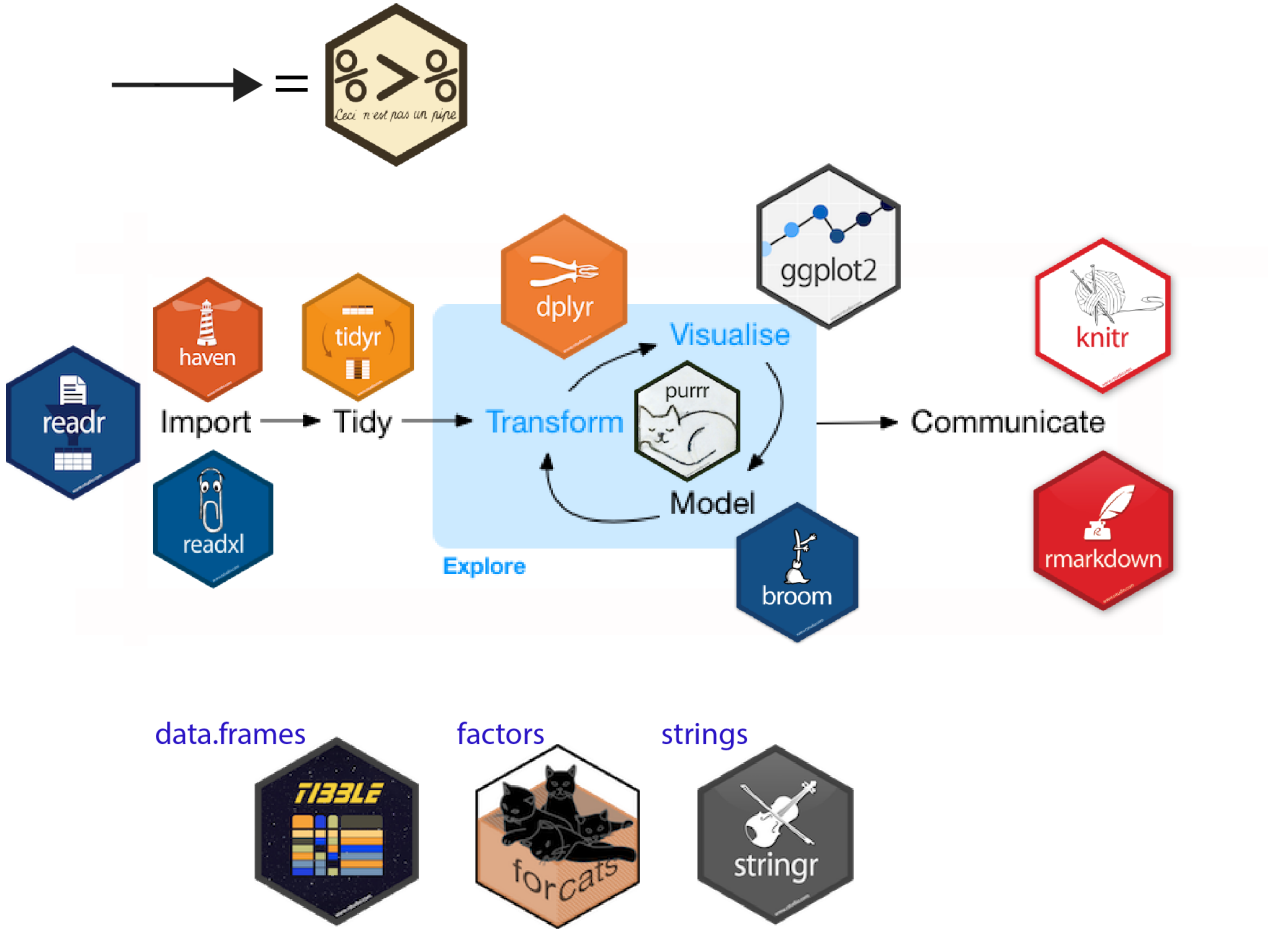
Core tidyverse:

- `ggplot2` : visualiser les données
- `dplyr` : manipuler les données
- `tidyr` : mettre en forme les données
- `readr` : importer les données
- `purrr` : itérer des fonctions sur les données
- `tibble` : version moderne des data-frames
- `stringr` : manipuler de chaînes de caractères
- `forcats` : manipuler des facteurs

Le tidyverse comprend également de nombreux autres packages avec une utilisation plus spécialisée.


> Tidyverse et R4DS

# tidyverse, l'incontournable



# `fusen`, le développeur

`fusen` permet de créer un package à partir de fichiers RMarkdown avec une philosophie «**Rmd-first**».

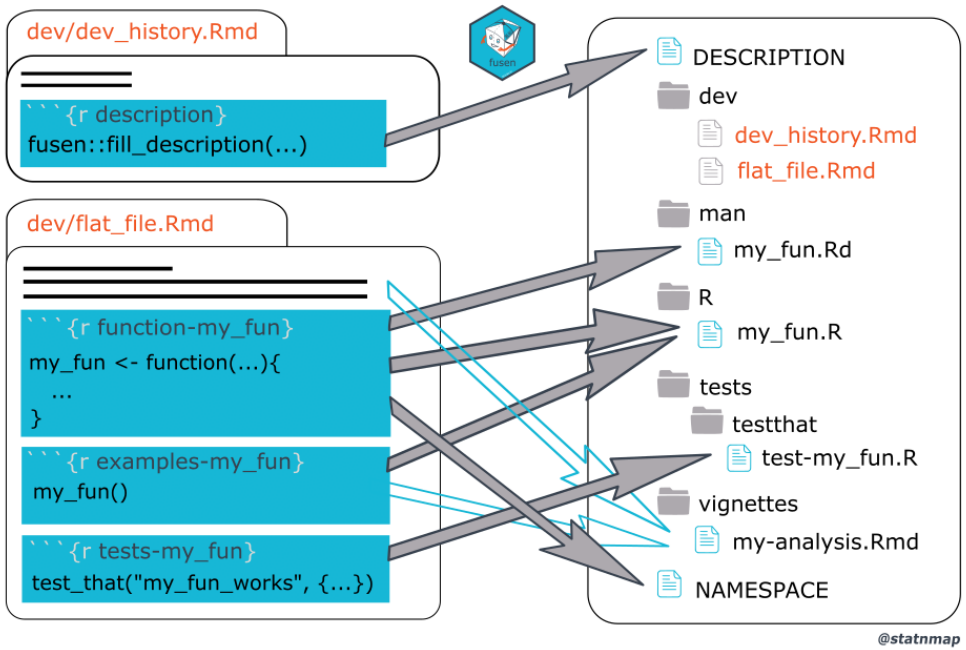
Dans le template rmarkdown on décrit la `description` puis pour chaque fonction :

- les chunks `function`
- les chunks `example`
- les chunks `tests`
- les chunks `development`

`fusen::inflate()` se charge du reste ...

Intégration des tests de développement (avec `testthat`), de la création de vignette, de la construction du site avec `pkgdown` et des outils de développement (`devtools` et `usethis`).

> fusen et blog



# `renv`, le manager de packages


`renv` permet une gestion fine des versions des packages au sein de chaque projet.

Les projets seront **isolés** (l'installation ou la mise à jour d'un package n'a pas d'effet sur les autres projets), **portables** (`renv` facilite l'installation des packages dont dépend votre projet) et **reproductibles** (`renv` enregistre les versions des packages dont le projet dépend).

Particulièrement adapté à une utilisation avec git.

Petit bonus, les packages ne sont pas installés dans le dossier de mon projet mais sont partagés avec un lien symbolique dans un dossier `renv/cache/` qui contient les différentes versions des packages.

> renv et slides

- Initialisation d'un nouvel environnement local pour le projet avec une bibliothèque privée

```
renv::init()
```

- Installation des packages comme d'habitude

```
install.packages(...)
```

- Sauvegarde de l'état de la bibliothèque privée du projet. Les packages **utilisés** et leur version sont détaillés dans le fichier `renv.lock`

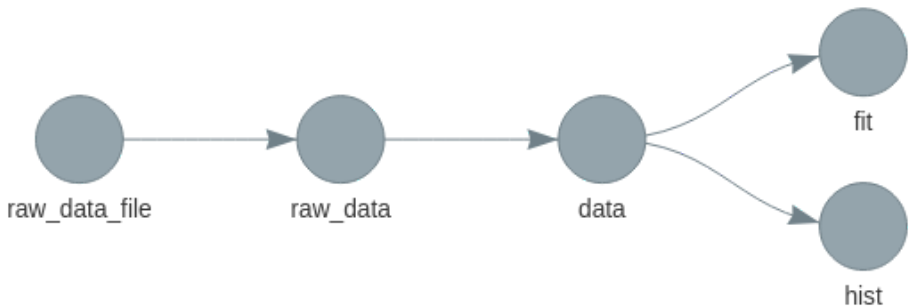
```
renv::snapshot()
```

- Si besoin, restaurer les versions tel-que détaillées le fichier de verrouillage

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

# `targets`, le ciblé


- `targets` est un gestionnaire de pipelines "make-like" pour les statistiques et DataScience.
- On définit *les cibles* et target déduit les relations entre les morceaux du projets. Il n'exécute que les calculs nécessaire.
- Visualisation :




> targets, book et YouTube

- `_targets.R` :

```
library(targets)
library(tarchetypes)
source("R/functions.R")
options(tidyverse.quiet = TRUE)
tar_option_set(packages = c("biglm", "dplyr",
list(
  tar_target(
    raw_data_file,
    "data/raw_data.csv",
    format = "file"
  ),
  tar_target(
    raw_data,
    read_csv(raw_data_file, col_types = cols(
  ),
  tar_target(
```

# `reprex`, l'exemplaire


Pour permettre à la communauté de nous aider il est souvent utile de fournir un **exemple reproductible**. Le code doit être le plus court possible et contenir toutes les dépendances. Si possible utiliser des jeux de données présents dans R (`iris`, `mtcars`, ...). De même, faciliter la lecture avec `styler`.
On peut ainsi reproduire le problème rencontré dans une nouvelle session R.

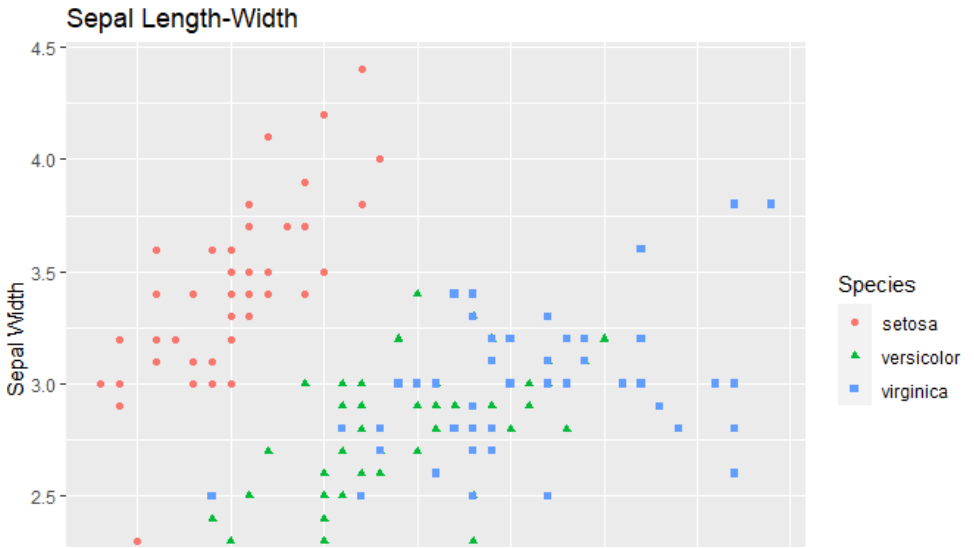
`reprex` génère un exemple reproductible à partir du code présent dans le clipboard pour vous permettre de demander facilement de l'aide à la communauté.

L'exemple est exécuté dans un rmarkdown et le résultat rendu dans le presse-papiers de l'utilisateur.

Divers formatages de sorties sont disponibles (R, Issues Github, Slack, Stack Overflow, ...) ainsi que l'ajout des informations de `session_info`.

> reprex

```
library(tidyverse)
summary(iris$Species)
#>     setosa versicolor  virginica
#>         50         50         50
ggplot(data=iris, aes(x = Sepal.Length, y = S
  geom_point(aes(color=Species, shape=Species
  xlab("Sepal Length") +  ylab("Sepal Width")
  ggtitle("Sepal Length-Width")
```



# pins, l'attachant


pins permet "d'accrocher" et de publier des données, des modèles et d'autres objets R. Ceci permet de **partager des objets R** entre projets et entre collaborateurs.

Les objets sont accrochés sur des *boards* qui peuvent être des dossiers (partagés ou non) ou des services tel que RSConnect, Amazon, Microsoft365, ...

Le suivi des versions des objets épinglés est facilité.

API *moderne* dédiée.

- Création d'un board :

```
board <- pins::board_temp()
# board <- pins::board_local() # share data a
# board <- pins::board_folder("~/Dropbox") #
# board <- pins::board_folder("Z:\\my-team\pi
```

> pins

```
board
```

```
## Pin board <pins_board_folder>
## Path: 'C:/Users/CEDRIC~1.MID/AppData/Local/Temp/Rt
## Cache size: 0
```

- Attacher un objet au board :

```
pins::pin_write(board, iris, name = "fleurs")
```

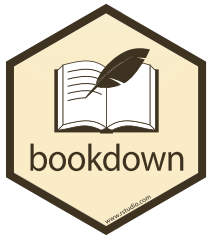
```
## Guessing `type = 'rds'`
## Creating new version '20220207T141419Z-f2bf1'
## Writing to pin 'fleurs'
```

- Lire un objet du board :

```
pins::pin_read(board, "fleurs") %>% identical
```

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

# bookdown, le rédacteur




bookdown facilite la **rédaction de livres**, de documentation d'outils et documents longs.

- Générer des livres et des ebooks prêts à imprimer
- Un langage de balisage plus facile à prendre en main que LaTeX
- Formats de sortie variés: PDF, LaTeX, HTML, EPUB ou Word
- Possibilité d'inclure des graphiques dynamiques et des applications interactives (Shiny)
- Prise en charge de langages autres que R, notamment C/C++, Python, SQL, ...
- Peut être publié sur GitHub, GitLab, bookdown.org

Intégration à RStudio : `File > New Project > New Directory > Book project using bookdown` puis `Build Book > bookdown::gitbook`

> Voir aussi pagedown pour créer des documents HTML paginés pour l'impression à partir de R Markdown.



> bookdown: Authoring Books and Technical Documents with R Markdown

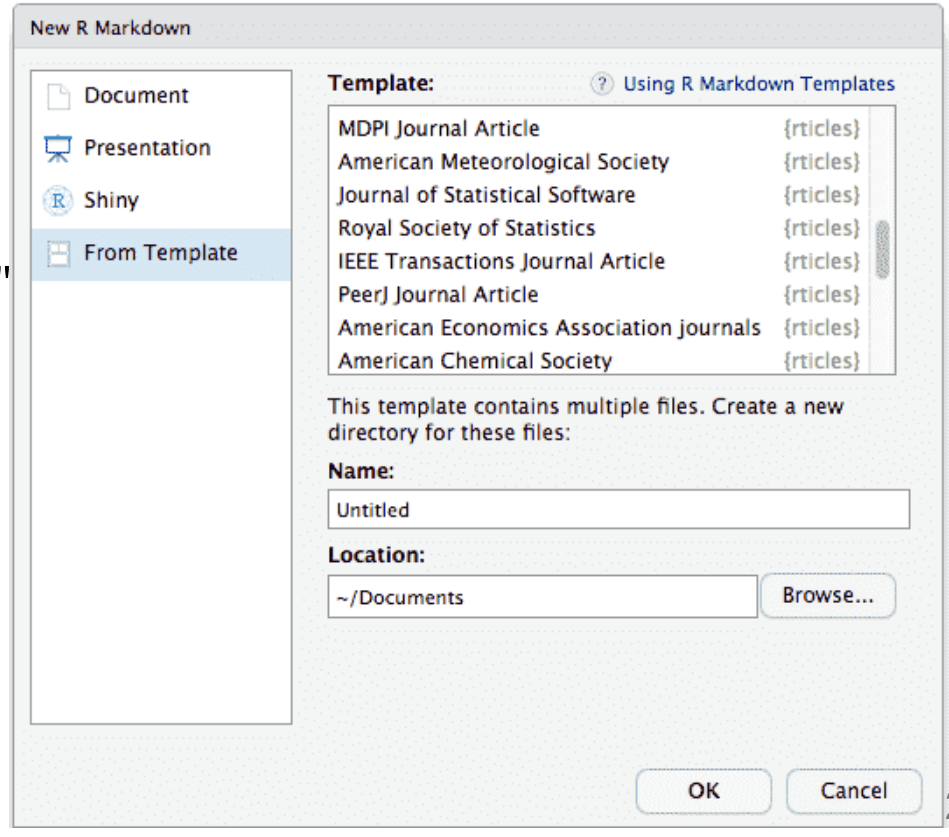
# `rticles`, l'auteur

`rticles` fourni des templates pour faciliter la rédaction d'**articles scientifiques** avec R Markdown et LaTeX.

```
rticles::journals()
```

```
##  [1] "acm"           "acs"           "aea"
##  [4] "agu"           "ajs"           "amq"
##  [7] "ams"           "arxiv"         "asa"
## [10] "bioinformatics" "biometrics"    "copernicus"
## [13] "ctex"          "elsevier"      "frontiers"
## [16] "glossa"        "ieee"          "ims"
## [19] "jasa"          "jedm"          "joss"
## [22] "jss"           "lipics"        "mdpi"
## [25] "mnras"         "oup_v0"        "oup_v1"
## [28] "peerj"         "pihph"         "plos"
## [31] "pnas"          "rjournal"      "rsos"
## [34] "rss"           "sage"          "sim"
## [37] "springer"      "tf"            "trb"
## [40] "wellcomeor"
```

Les templates sont disponibles avec `File > New File > R Markdown > From Template`




> rticles

# workflowr, l'organisé


`workflowr` permet d'**organiser les analyses** pour une gestion de projet efficace, une meilleure reproductibilité, une bonne collaboration et partage des résultats. Ce package promet :

- un modèle de projet avec une organisation en sous-dossiers
- des templates R Markdown dédiés
- une intégration du versionning avec git
- une exécution de chaque analyse dans une session R isolée
- un suivi des informations de session
- la création d'un site web dédié à chaque analyse pour présenter les résultats et hébergeable avec GitLab Pages.

> workflowr et démo et YouTube

```r
library("workflowr")

# Configure Git (only need to do once per com
wflow_git_config(user.name = "Full Name", use

# Start a new workflowr project
wflow_start("myproject")

# Build the site
wflow_build()

# Customize your site!
#   1. Edit the R Markdown files in analysis/
#   2. Edit the theme and layout in analysis/
#   3. Add new or copy existing R Markdown fi

# Preview your changes
wflow_build()

# Publish the site, i.e. version the source c
wflow_publish("analysis/*", "Start my new pro
```

# `xaringan`, le présentateur


`xaringan` permet de créer des **diaporamas** avec remark.js et R Markdown.

- remark.js gère la mise en forme des slides, la présentation et les raccourcis clavier.
- R Markdown (knitr) permet d'ajouter des calculs et résultats dynamiques à un simple markdown.

Intégration d'un addin `Infinite Moon Reader` pour une preview en direct des slides dans RStudio.

De nombreux thèmes et possibilité d'ajouter un CSS (*si vous arrivez à récupérer celui d'INRAE, ça m'intéresse*)

*xaringan a permis de faire ces slides (et elles sont mise à dispo grâce à GitLab CI/CD)*

> xaringan et slides

Particulièrement adapté aux formations R : DataTable

```
DT::datatable(iris, rownames = FALSE,
              options = list(scrollX = TRUE,
```

| Sepal.Length | Sepal.Width | Petal.Length | P |
|---|---|---|---|
| 5.1 | 3.5 | 1.4 | |
| 4.9 | 3 | 1.4 | |
| 4.7 | 3.2 | 1.3 | |
| 4.6 | 3.1 | 1.5 | |
| 5 | 3.6 | 1.4 | |

Previous 1 2 3 4 5 … 30 Next

# xaringan, le présentateur

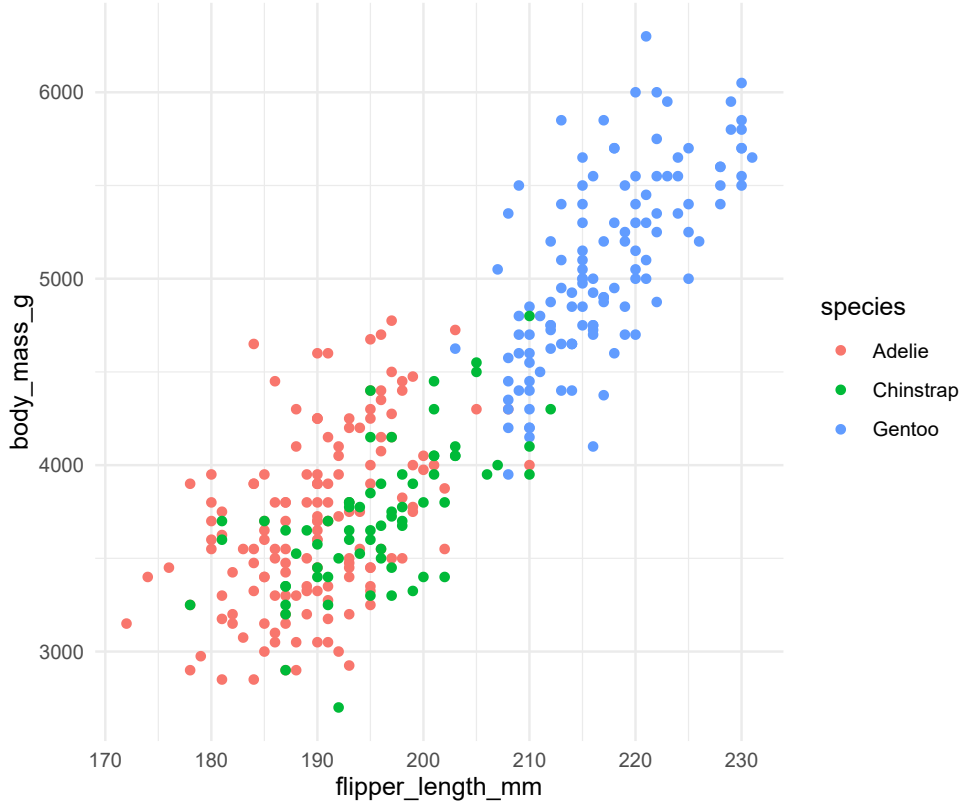

Particulièrement adapté aux formations R :

- Un plot dynamique

```
library(ggplot2)
library(ggiraph)
library(palmerpenguins)

gg_point <- ggplot(data = penguins) +
  geom_point_interactive(
    aes(
      x = flipper_length_mm,
      y = body_mass_g,
      color = species,
      tooltip = island
    )
  ) +
  theme_minimal()

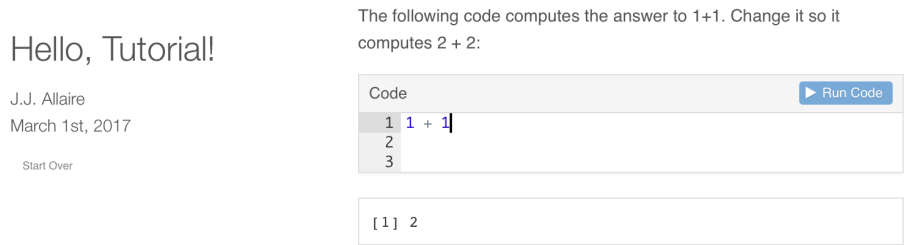
girafe(ggobj = gg_point)
```



# `learnr`, le pédagogue


`learnr` permet de rédiger un **tutoriel interactif** à partir de R Markdown. Les tutoriels peuvent comprendre :

- Du texte, des figures, des plots, des équations
- Des exercices de code (chunck R que l'utilisateur peur modifier et exécuter directement)
- Des quiz
- Des vidéos (hébergées sur YouTube)
- Des composants Shiny




```
---
title: "Hello, Tutorial!"
output: learnr::tutorial
runtime: shiny_prerendered
---
```{r setup, include=FALSE}
library(learnr)
```

The following code computes the answer
to 1+1. Change it so it computes 2+2:

```{r addition, exercise=TRUE}
1 + 1
```
```

> learnr

# `distill`, le blogueur


`distill` permet de créer des sites web à partir de R Markdown. Il a été optimisé pour les « communications scientifiques et techniques ».

- Initialisation du projet parmi deux structures disponibles : *website* et *blog* :

```
distill::create_website(dir = "mysite", title
```

*Création de `_site.yml` pour la configuration, `index.Rmd` pour la page principale, `_posts/welcome/welcome.Rmd` la première page du site.*

- Ajout de nouveau post:

```
distill::create_post("Un nouveau post")
```

*Les articles sont créés dans le dossier `_posts`.*

- Construction de l'article avec `Knit`.
- Construction du site avec l'onglet `Build Website`

```
rmarkdown::render_site()
```

- Mise à disposition, soit via un HTML standalone, soit via RPubs, soit avec GitLab Pages.

Comme les autres R Markdown, les pages prennent en charge textes, images, plots, tables, équations, bibliographie, footnotes, *toc*, blocs de code, thème CSS ...

> Voir aussi `blogdown` pour créer des sites web avec HUGO.


> `distill` for R Markdown

# Exemples de productions MIGALE

Slides de formations avec `xaringan`

Blog Tutorials avec `blogdown` et HUGO

Rapports d'analyses avec `rmarkdown` (mais pas de projets publics)

Tutoriels d'annotation de génomes du CATI BOOM avec `bookdown`

# Pour aller plus loin

• Les différents livres et sources présentés dans ces slides.

• Suivre des sujets tel que #RStats, #tidyverse, #R4DS, #TidyTuesday ou les grands noms de la communauté sur Twitter.

• Suivre les blogs de R Project, rOpenSci, ThinkR (vf), RStudio et Tidyverse.

• Aller aux différents évènements : The useR! Conference et Rencontres R.

• Conférence de Maëlle Salmon : S'il-vous-plaît... dessine-moi un projet (slides) et Projet PROPRE (PROcessus de Publications REproductibles) de Sébastien Rochette.

• Commencer par un petit projet, seul ou en équipe. Mettre en pratique petit à petit les différents éléments. Rester informé des nouveautés. Ne pas avoir peur de changer d'outils au fil du temps.

# Merci de votre attention !!

PROSE
PRocédés biOtechnologiques
au Service de l'Environnement

cedric.midoux@inrae.fr