Créer son premier package R

Juliette ENGELAERE-LEFEBVRE - Maël THEULIERE

Objectif de cette séquence

Après cet atelier vous saurez :

- comprendre ce qu'est un package
- comprendre la structure d'un package
- créer votre premier package

`

Qu'est ce qu'un package ?

Qu'est ce qu'un package ?

En R, un package est le moyen le plus adapté pour étendre les fonctionnalités de R.

Un package est une collection de fonctions, de données, de documentations et de tests.

Un package est facilement partageable avec d'autres.

Un package est le plus souvent versionné, c'est à dire qu'il peut exister plusieurs versions d'un même package pas forcément compatibles entre elles.

`

Qu'est ce qu'un package ?

Un package peut être hébergé sur le CRAN ou sur une forge logiciel comme github ou gitlab.

Quand il est sur le cran, la fonction `install.packages()` permet de l'installer.

```
install.packages("mon_package")
```

Quand il est sur une forge, on utilise les fonctions du package `{remotes}` pour les installer. Dans ce cas, il faut lui spécifier la forge et l'adresse relative de ce package.

Exemples :

```
remotes::install_github("auteur_du_package/son_package")
```

```
remotes::install_gitlab("auteur_du_package/son_package")
```

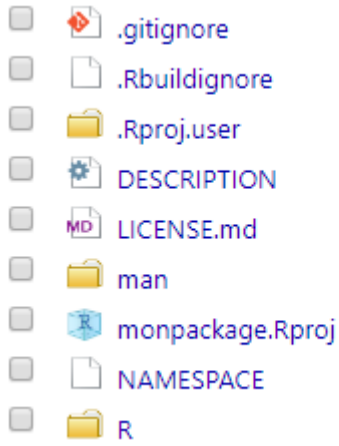
`

Structure d'un package

La structure minimale d'un package

La structure d'un package va différer suivant le contenu de celui-ci. La structure minimale consiste en :

- un fichier `DESCRIPTION` qui contient la description d'un package et ses dépendances, c'est à dire la liste des packages qui seront nécessaires pour faire fonctionner ce package
- un répertoire `R/` qui contient les fichiers sources des fonctions du package
- un fichier `NAMESPACE` qui contient les fonctions exportées par le package et la liste des fonctions importées d'autres packages. Ce fichier est généré automatiquement
- un répertoire `man/` qui contient la documentation des fonctions du package. Son contenu est généré automatiquement

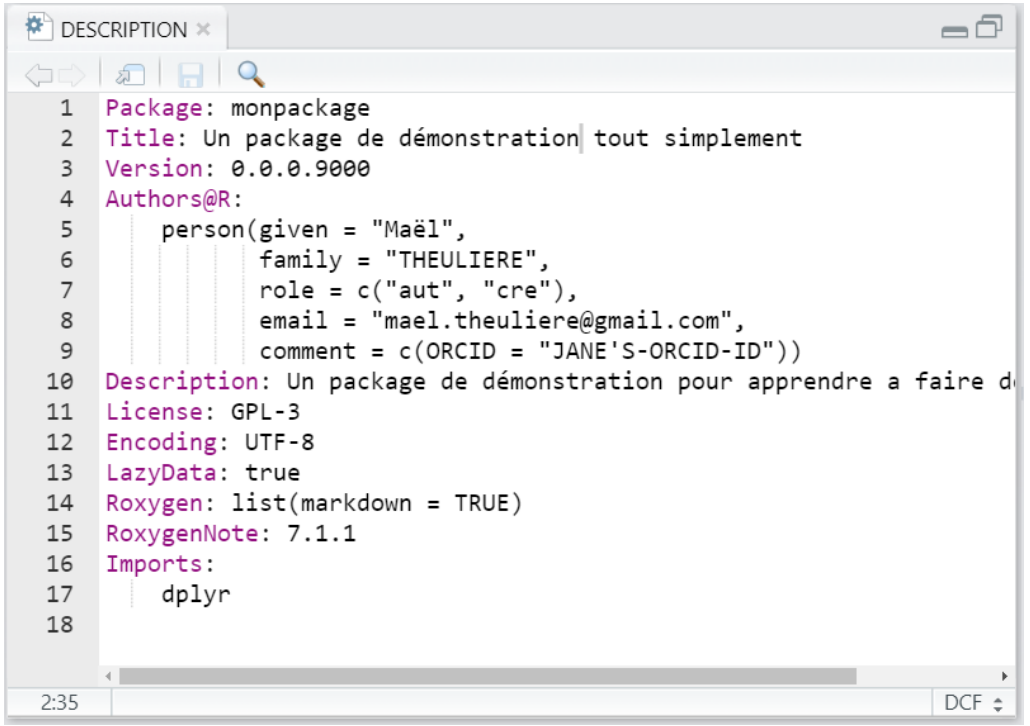


Structure d'un package

Le fichier DESCRIPTION

Le fichier `DESCRIPTION` contient :

- la description du package (nom, auteur, licence, version...)
- la liste des packages qui seront nécessaires pour faire fonctionner ce package (rubrique `Imports`)



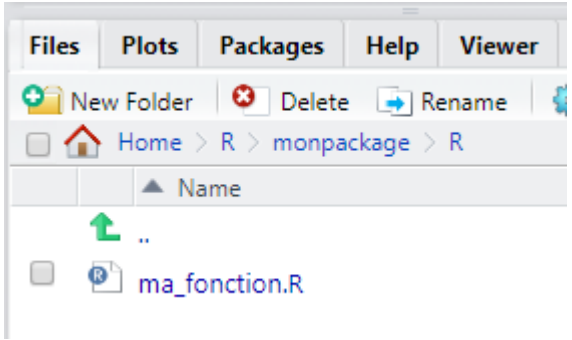
`

Structure d'un package

Le répertoire R/

Le répertoire R/ contient les fichiers sources des fonctions du package.

Conseil pour bien commencer : créer un fichier R pour chaque fonction nommé de la même façon que la fonction.



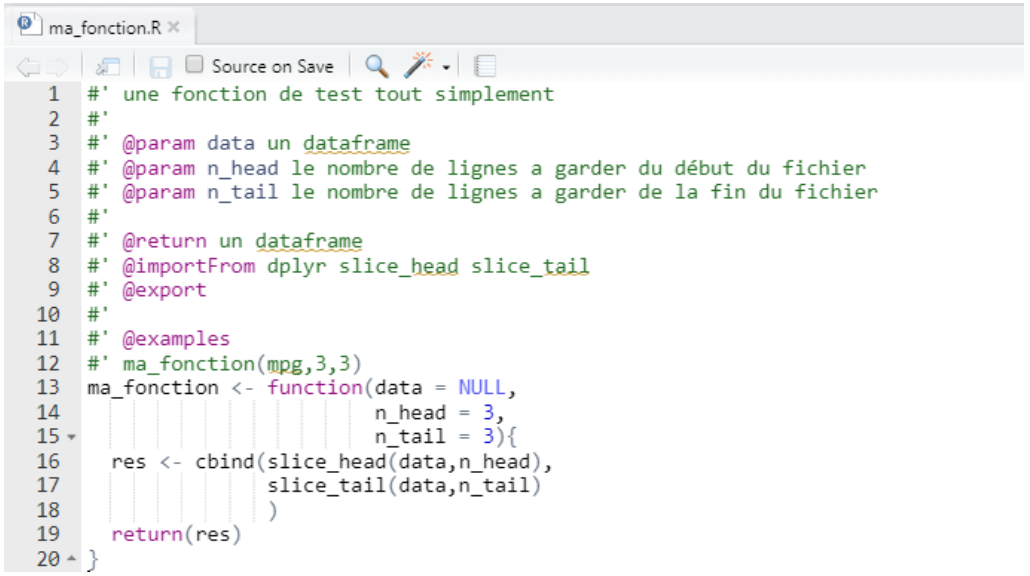
`

Structure d'un package

Le répertoire R/

Chaque fichier R contient :

- la définition de la fonction en R
- le code roxygen qui permettent de construire la documentation et de définir les dépendances de la fonction. Le code roxygen commence toujours par `#'`



`

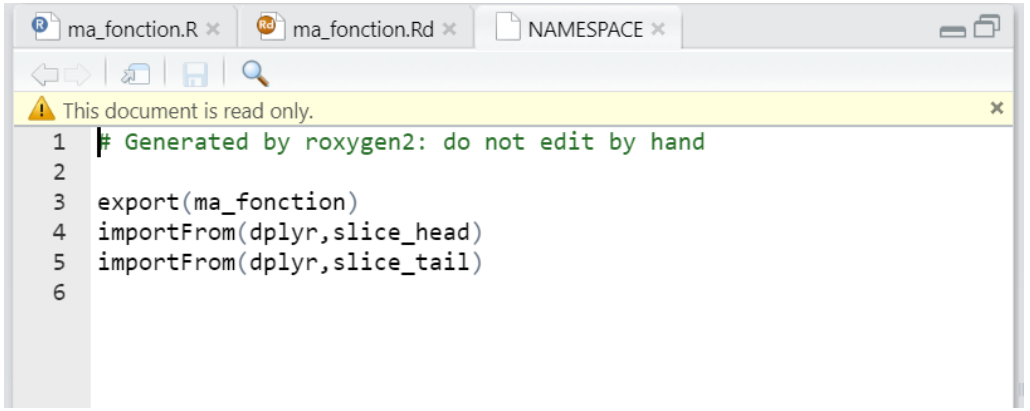
Structure d'un package

Le fichier NAMESPACE

Le fichier NAMESPACE contient :

- les fonctions exportées par le package
- la liste des fonctions importées d'autres packages utilisées dans notre package

Ce fichier est généré automatiquement grâce à Roxygen.



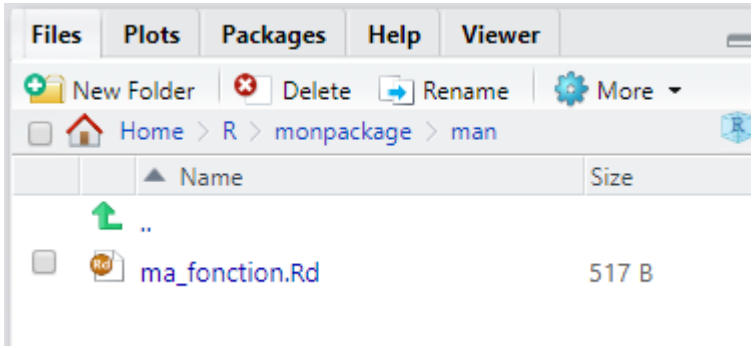
`

Le répertoire man

Le répertoire `man/` contient la documentation des fonctions du package.

Pour chaque fonction, il contient un fichier `.Rd` qui sera la documentation de la fonction consultable dans l'aide de R.

Son contenu est généré automatiquement grâce à Roxygen.



`

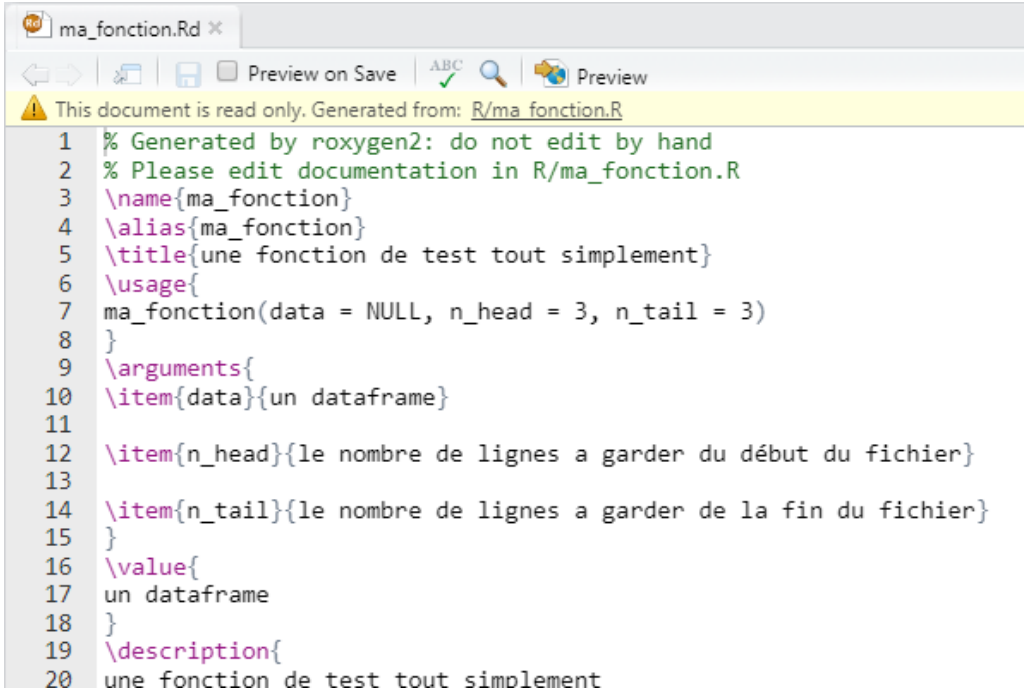
Structure d'un package

Le répertoire man

Le répertoire `man/` contient la documentation des fonctions du package.

Pour chaque fonction, il contient un fichier `.Rd` qui sera la documentation de la fonction consultable dans l'aide de R.

Son contenu est généré automatiquement grâce à Roxygen.



`

Créer son premier package

`

Vos compagnons pour créer et développer votre package

4 packages vous seront indispensables pour développer vos packages :

- `{devtools}` recense les fonctions qui seront utilisées pour compiler ou installer votre package.
- `{usethis}` contient un ensemble de fonctions qui vous permetront d'automatiser la plupart des tâches dont vous aurez besoin.
- `{roxygen2}` vous permettra de documenter vos fonctions (cf atelier 2).
- `{pkgdown}` vous permettra de créer un site de documentation de votre package.
- `{testthat}` vous permettra de tester vos fonctions. C'est à dire de définir un ensemble d'instructions qui vous permettront de vérifier que vos fonctions produisent bien le résultat attendu (cf atelier 2).

Pour installer ces packages :

```
install.packages(c('devtools','usethis','roxygen2','testthat', 'pkgdown'))
```

Créer son premier package

Configurer votre environnement

Avant de créer votre premier package, il va falloir réaliser quelques démarches initiales à faire une bonne fois pour toute pour configurer votre poste : configuration du proxy, de git...

Suivez le guide à cette adresse pour cela.

`

Créer son premier package

Mon premier package

Vous êtes maintenant prêt à créer votre premier package. Vous allez pouvoir utiliser `{usethis}` pour faciliter ce travail :

1 - création du package sur ma machine

```
usethis::create_package("monpremierpackage")
```

2 - utilisation de git comme gestionnaire de version

Une invite de commande s'affiche dans la console pour savoir si vous voulez réaliser un premier commit, répondez oui.

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

`

Mon premier package

3 - Créer un repo sur github ou gitlab

github gitlab

La fonction `use_github()` créer automatiquement le projet distant sur votre compte github.

```
usethis::use_github()
```

`

Créer son premier package

Mon premier package

Vous allez pouvoir utiliser `{usethis}` pour faciliter ce travail :

4 - création d'une fichier `dev_history.R` pour tracer nos développements

- Création du fichier

```
usethis::edit_file("dev_history.R")
```

- Ajout du fichier dans les fichiers à exclure lors de la compilation du package

```
usethis::use_build_ignore('dev_history.R')
```

5 - modification de la licence

```
usethis::use_gpl3_license()
```

`

Mon premier package

Vous allez pouvoir utiliser `{usethis}` pour faciliter ce travail :

6 - modification du fichier description

Modifier la description de votre package

```
usethis::edit_file('DESCRIPTION')
```

Et voilà, votre premier package est prêt ! Certes, il ne contient pour l'instant RIEN, mais il peut se compiler et s'installer

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

`

Et maintenant ?

Et maintenant ?

Dans l'atelier suivant nous allons voir comment rajouter une fonction dans votre package, la documenter et la tester. Vous verrez à travers cela le workflow type de travail pour développer votre package.

`