FORMATION R-DÉVELOPPEMENT DE PACKAGE

RÉSEAU MBS ET LA KIM DATA & LIFE SCIENCE

MATHIEU DEPETRIS, MATHIEU.DEPETRIS@IRD.FR

19/05/2021

PROGRAMME DE LA FORMATION

- Formation flash sur le développement de package sous R
  - Présentation générale de la construction d'un package
    - Pourquoi faire un package ? Comment et avec quelles bonnes pratiques ?
    - Vous permettre de vous lancer, savoir ou chercher et surtout avoir les références pour aller plus loin
  - Si on a un peu de temps, construire un exemple de a à z
- Structure de la formation en lien avec un tutoriel existant (https://kbroman.org/pkg_primer/) et un livre (https://r-pkgs.org/)
- Durant cette session nous utiliserons l'environnement Windows avec R version 4.0.4
  - Plusieurs logiciels tiers « intéressants » à utiliser pour notamment se simplifier la vie
    - Rstudio : https://www.rstudio.com/products/rstudio/
    - Rtools (uniquement pour Windows) : https://cran.r-project.org/bin/windows/Rtools/
    - Visual Studio Code : https://code.visualstudio.com/
    - Github Desktop : https://desktop.github.com/

PARTIE I

POURQUOI ÉCRIRE DES PACKAGES R ?

PARTIE 1 : POURQUOI ÉCRIRE DES PACKAGES R

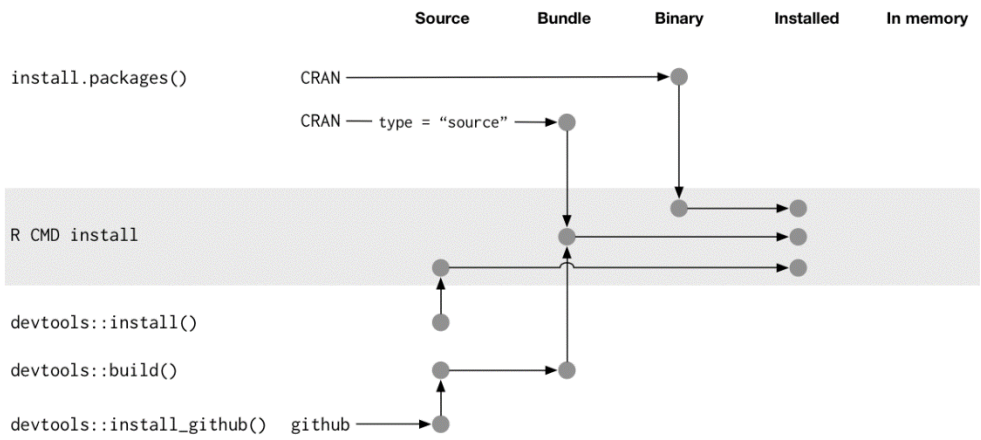
- Les packages R et le Comprehensive R Archive Network (CRAN) sont des fonctionnalités extrêmement importantes de R
  - Moyen simple de distribuer du code et la documentation associée
    - Réutilisation du code, distribution de données et autres pour des articles, …
  - Standardisation du code
  - Les packages publiés sur le CRAN sont pratiquement garantis d'être fonctionnels
    - Régulièrement compilés, installés et testés sur différents systèmes
  - Très simple à créer et de nombreux outils existent pour se simplifier la vie
    - Package roxygen2 pour la documentation (https://roxygen2.r-lib.org/)
    - Packages usethis (https://usethis.r-lib.org/) et devtools (https://devtools.r-lib.org/)
    - L'environnement de Rstudio directement
  - Les packages R peuvent être grands et importants, mais ce ne sont ni plus ni moins que des « fonctions » structurées d'une certaine façon

PARTIE 2

STRUCTURE D'UN PACKAGE R

PARTIE 2 : STRUCTURE D'UN PACKAGE R

- Un package peut être sous différentes formes
  - « source », un exemple ici https://github.com/tidyverse/readxl
  - « bundled », compressé en 1 seul fichier (par convention des extensions .tar ou .gz)
  - « binary », comme précèdent 1 seul fichier mais avec à l'intérieur des outils de développement spécifiques à une plateforme
  - « installed », package décompressé dans une librairie

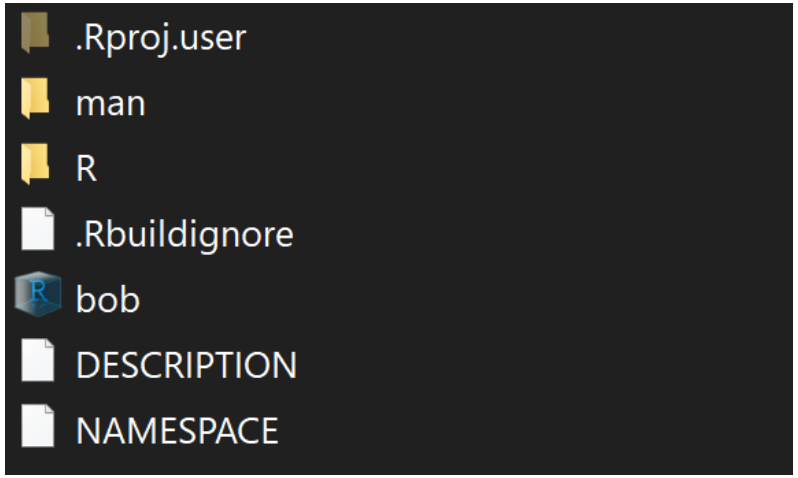


PARTIE 2 : STRUCTURE D'UN PACKAGE R

- Il y a plusieurs façons de construire un package R
  - Manuellement en créant chaque dossier et fichier (dans un format spécifique)
  - Via des commandes spécifiques
    - usethis::create_package(path = "path\\nom_package")
    - Directement depuis l'interface de Rstudio
      - File – New Project – New directory – R package
- Il n'y a pas de bonne ou mauvaise méthode, juste celle qui vous convient
- Pratique
  - Essayez de créer le squelette d'un package en l'appelant "mon_premier_package"
  - Maintenant, ressayez en utilisant un nom plus « cohérent »

PARTIE 2 : STRUCTURE D'UN PACKAGE R

- Petite parenthèse sur le nom de votre package
  - Plus compliqué qu'il n'y parait
  - Allez jeter un coup d'œil ici https://r-pkgs.org/workflows101.html#naming
  - Vous aider d'un package available (https://cran.r-project.org/web/packages/available/index.html)
- Une fois que l'on a le nom, allons voir le répertoire de notre package
  - Voilà la plus petite architecture d'un package R
- Ne pas utiliser la fonction de base de R package.skeleton() !
  - Processus de développement différent des précédents
  - Génère des erreurs qu'il faut corriger



PARTIE 2 : STRUCTURE D'UN PACKAGE R

- Intérêt de passer par un projet Rstudio
  - Allez faire un tour ici : https://r-pkgs.org/workflows101.html#projects
  - Environnement de travail disponible immédiatement ("launch-able »)
  - Isolation de l'environnement de travail par rapport aux autres
  - Accès directement à des boutons d'accès rapide facilitant le développement
  - Accès à des options globales de notre projet (par exemple en rapport avec la documentation)
- Pratique
  - Allons voir cela de plus près

PARTIE 3

DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Dossier r
  - Contient tous les codes R et fonctions, extension en .R
  - Pas de règle fixe, mais une certaine convention dans l'organisation des scripts
    - Allez faire un tour sur https://style.tidyverse.org/files.html
    - Quelques exemples
      - Les noms des fonctions doivent signifier quelque chose
      - Pas de caractères spéciaux
      - Séparer le code par des sections
      - Pas d'espaces en plus
      - La vie est déjà assez compliquée, utiliser au maximum les outils de Rstudio (comme les indentations)
  - Regrouper ce qui peut être regroupé, mais ne pas chercher à avoir le minimum de fichier
  - Utiliser les fonctions comme lintr::lint_package() pour vérifier notre code au fur et à mesure

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Fichier DESCRIPTION
  - Stockage des métadonnées de notre package, allez voir ici https://r-pkgs.org/description.html#description)
  - Attention à l'indentation
  - Peut être remplie à la main ou via des fonctions (allez voir https://usethis.r-lib.org/reference/use_description.html)
    - usethis::use_description()
  - C'est dans ce fichier qu'a lieu la gestion des dépendances
    - Concrètement ce sont les packages nécessaires au fonctionnement du vôtre
    - Deux types de dépendances majeurs :
      - Imports : nécessaire pour que votre package fonctionne
      - Suggests : pas nécessaire pour que votre package fonctionne, mais peut être utile par exemple pour des jeux de données tests
      - Utiliser la fonction usethis::use_package(), attention à bien mettre une version quand vous spécifiez le package
  - Pour la licence aller voir ici https://r-pkgs.org/license.html, bien réfléchir avant de le remplir
    - Utiliser les fonctions comme use_gpl_license()
  - Attention au champ version il est important, allez voir ici https://semver.org/

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Fichier NAMESPACE
  - Peut paraître inutile, mais permet au contraire d'encapsuler le package et d'éviter qu'il n'interfère (ou inversement) avec d'autres packages
  - Vital si l'on veut publier sur le CRAN
  - Permet d'expliciter exactement les fonctions utilisées et surtout le package associé à chacune
    - Éviter notamment d'avoir des erreurs dans la sélection des fonctions (lancer la commande library(dplyr))
    - Différence entre « Loading » (chargé, mais non ajouté dans le « search path ») et « Attaching » (chargé et ajouté dans le « search path »)
  - Permet aussi de rendre les fonctions de notre package accessibles directement ou non (utilisation :: ou :::)
  - Peut (et doit) être rempli par roxygen2
    - Utiliser les champs @importFrom et @export quand vous générez la documentation de vos fonctions (voir partie 3 sur la documentation)

@

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Dossier man
  - Va contenir la documentation de notre package, extension en . Rd
  - Peut, encore, s'écrire de manière automatique avec roxygen2. Allez voir ici https://r-pkgs.org/man.html et ici https://roxygen2.r-lib.org/index.html
  - Plusieurs « niveaux » de documentation
    - Directement dans notre code en utilisant des tags (avec @)
      - Permet notamment de documenter les paramètres de fonction, le nom ou encore le type de sortie
    - Via des vignettes : documentation plus approfondie et détaillée que précédemment (allez voir ici https://r-pkgs.org/vignettes.html)
      - Utiliser usethis::use_vignette("my-vignette") pour initialiser le dossier vignettes et créer la première
      - Elles sont écrites en R Markdown (allez voir ici https://rmarkdown.rstudio.com/)
    - En compilant les deux précédents niveaux pour avoir une documentation en ligne via les github pages
      - Utilisation du package pkgdown (https://pkgdown.r-lib.org/)
      - Voir la partie 5

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Dossier inst
  - Permet de stocker des données externes
    - Jeux de données tests
    - Données internes précompilées (par exemple une carte du monde en carrée de 1°)
    - Données pour les vignettes
    - Données brutes
  - On peut créer des sous-dossiers pour une meilleure organisation
  - On peut aussi y accéder une fois le package installé
    - Utiliser des fonctions comme load(file = system.file(« nom_sous_dossier", « nom_fichier_avec_extension", package = « package"))

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Dossier tests
  - Partie vitale du développement
  - Permet de vérifier de notre code fait bien ce pour quoi il est programmé
  - On parle de test unitaire
    - Attention c'est différent des tests de cohérences ou du contrôle qualité (même si la frontière est parfois floue)
  - Permet d'avoir un package robuste
    - L'ajout de nouvelles fonctionnalités ne perturbe par le fonctionnement du package
  - Pour initialiser le processus, lancez la commande usethis::use_testthat()
    - Allez faire un tour ici https://testthat.r-lib.org/index.html
    - Pour lancer les tests, utilisez la commande Test Package de Rstudio

PARTIE 3 : DESCRIPTION DU CONTENU D'UN RÉPERTOIRE SOURCE

- Il existe aussi d'autres répertoires et fichiers particuliers
  - Regarder ici https://r-pkgs.org/misc.html
  - Un fichier README.Rmd qui génère un README.md utilisé sur un git (voir partie 4)
  - Des dossiers et fichiers en rapport avec l'utilisation d'un git (dossiers .github ou encore des fichiers .gitignore, voir partie 4)
  - Un dossier pkgdown servant à la documentation en ligne (voir partie 5)
  - …

PARTIE 4

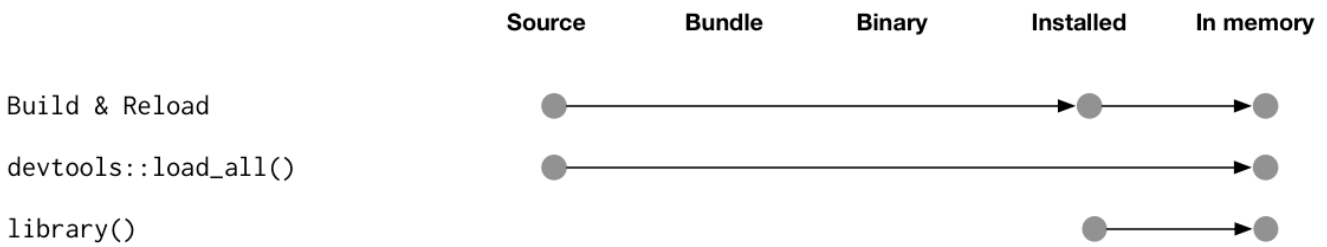
CYCLE DE DÉVELOPPEMENT

PARTIE 4 : CYCLE DE DÉVELOPPEMENT

- De manière générale et très simplifiée, le développement d'un package est la répétition d'un cycle

"lather, rinse, repeat"

- Deux fonctions à connaître
  - devtools::load_all()
  - Build & Reload (Install and Resart et Clean and Rebuild)



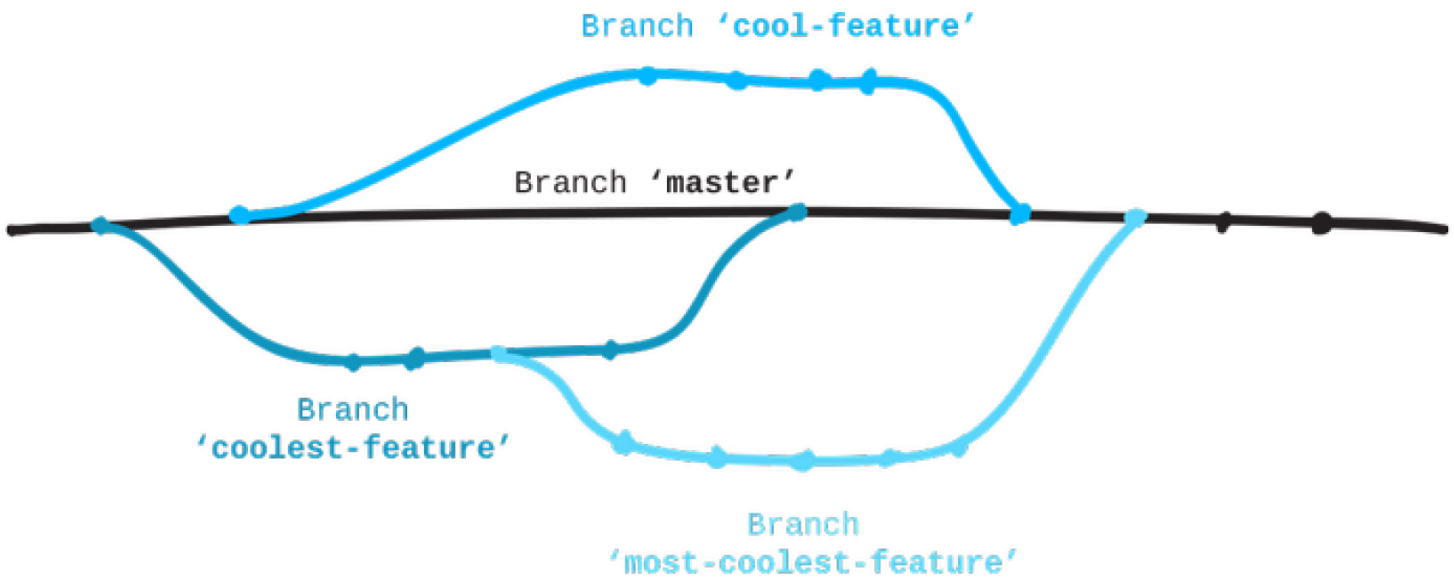
PARTIE 4 : CYCLE DE DÉVELOPPEMENT

- Hébergement sur un git
  - Intéressant d'héberger les fichiers sources du package sur un git (même après la publication sur le CRAN)
  - Git c'est quoi ?
    - Git est un système de contrôle de version open source gratuit (on parle aussi de forge ou de logiciel de gestion de versions décentralisées)
    - Allez voir ici https://git-scm.com/docs/git-submodule et https://r-pkgs.org/git.html
    - Il est existe différentes forges, payantes ou non de manière partielle ou totale, qui présentent des avantages et des inconvénients
      - Gitlab https://about.gitlab.com/
      - Github https://github.com/
      - …
    - Dans votre choix, il est important de tenir compte de vos besoins, surtout en termes de visibilité
    - Permet de conserver les fichiers sources en conservant la chronologie de toutes les modifications effectuées dessus
  - Outil presque indispensable pour la gestion et la coordination du travail en équipe

PARTIE 4 : CYCLE DE DÉVELOPPEMENT

- Exemple avec GitHub https://github.com/
  - Version gratuite bridée, mais possible d'avoir une version une version « pro » gratuite sous certaines conditions (par exemple si institut de recherche)
  - Seul point négatif le service est propriété de Microsoft
  - Incomparable en termes de rayonnance et de visibilité des projets
  - Accès au Github pages (https://pages.github.com/), c'est-à-dire un hébergement de site internet gratuit (avec des restrictions)
  - Système de tickets pour la gestion des beugs, futurs développements ou requêtes
    - Conversation de l'historique de tous les tickets, ce qui fait office de foire aux questions
  - Utilisation de branches

PARTIE 4 : CYCLE DE DÉVELOPPEMENT



PARTIE 4 : CYCLE DE DÉVELOPPEMENT

- Vérification du package via R CMD check
  - Part importante du processus de développement qui permet de détecter les problèmes
  - Obligatoire si vous voulez publier sur le CRAN (vous ne devez avoir aucune erreur)
  - Allez faire un tour ici https://r-pkgs.org/r-cmd-check.html
- Ne pas le lancer directement après la création de l'architecture du package, car vous aurez des erreurs (par exemple manque de la licence)
- Pour le lancer, utilisez la commande devtools::check() ou cliquez sur le bouton Check Package de Rstudio

PARTIE 5

PETIT BONUS, UTILISATION DU PACKAGE PKGDOWN VIA LES GITHUB PAGES

PARTIE 5 : PETIT BONUS, UTILISATION DU PACKAGE PKGDOWN VIA LES GITHUB PAGES

- Package pkgdown (https://pkgdown.r-lib.org/)
- Permet de créer simplement et rapidement un site internet en lien avec votre package
  - Vitrine de votre package
  - Utilise la documentation déjà générée dans vos scripts et vos vignettes
  - Meilleure lisibilité que de regarder l'aide la console de R
- Nécessite une synchronisation avec GitHub
  - Allez voir ici https://usethis.r-lib.org/articles/articles/git-credentials.html
  - Utilisez la commande usethis::create_github_token()
- Utilisez la commande usethis::use_pkgdown() pour initialiser le dossier et les fichiers nécessaires
- Notion d'intégration continue via les githubs actions
  - Utilisez la commande usethis::use_github_action("pkgdown")
  - Allez faire un tour ici https://github.com/r-lib/actions/tree/master/examples