Créer son premier package R

Gérer les versions de votre package

Juliette ENGELAERE-LEFEBVRE - Maël THEULIERE

Qu'est ce qu'une version ?

- Une *version* d'un logiciel correspond à un état donné de l'évolution d'un produit logiciel utilisant le *versionnage*
- Le versionnage est le mécanisme qui consiste à conserver la version d'une entité logicielle quelconque, de façon à pouvoir la retrouver facilement, même après l'apparition et la mise en place de versions plus récentes.
- Il est souvent associé à une numérotation spécifique de votre logiciel, parfois couplée à un nommage ad hoc en complément (R 4.0.3 "Bunny-Wunnies Freak Out", Ubuntu 21.04 "Hirsute Hippo")

`

source : wikipedia

Les différents types de version d'un logiciel

Une version de développement est une version non stable du logiciel, car en cour de développement.

Une version patch de notre logiciel est une version corrective, elle ne doit ajouter que des corrections de bug ou éventuellement des fonctionnalités mineures.

Une version mineure peut contenir des corrections de bugs, des nouvelles fonctionnalités, et quelques changements ayant des impacts de rétrocompatibilité.

Les versions majeures sont réservées :

- soit à l'annonce d'une version stabilisée de vos fonctionnalités (1.0.0) ;
- soit à l'annonce, pour les suivantes, de ruptures majeures dans vos packages (générant souvent des problèmes de rétrocompatibilité importants).

`

Pourquoi gérer les versions d'un package R ?

Les packages R sont versionnés automatiquement. Quand vous créez un premier package, il a un numéro de version par défaut à 0.0.0.9000.

Versionner permet :

- de faire évoluer votre package en augmentant/modifiant/supprimant des fonctionnalités ;
- en documentant ces évolutions ;
- le tout sans rompre des chaînes de traitements de vos utilisateurs, qui pourront toujours utiliser une version antérieure donnée.

`

Exemple : la vie de dplyr

dplyr (development version)

- `coalesce()` accepts 1-D arrays (#5557).

dplyr 1.0.7

- `across()` uses the formula environment when inlining them (#5886).
- `summarise.rowwise_df()` is quiet when the result is ungrouped (#5875).
- `c_across()` and `across()` key deparsing not confused by long calls (#5883).
- `across()` handles named selections (#5207).

dplyr 1.0.6

- `add_count()` is now generic (#5837).
- `if_any()` and `if_all()` abort when a predicate is mistakingly used as `.cols=` (#5732).

Gérer vos versions : le workflow

Tracer vos évolutions dans un fichier NEWS.md

Au début de vos développements, nous n'avez pas encore publié de version.
Commencez déjà à tracer ce que vous ajoutez à votre package.

- Créer un fichier `NEWS.md` dans votre package pour détailler les évolutions au fur et à mesure de votre développement. Comme toujours, {`usethis`} est là pour vous faciliter la vie : `usethis::use_news_md()`
- Tracez-y toutes les modifications des fonctionnalités : ajout de fonctions, modifications de fonctions, ajouts de tables.
- Regroupez les suivant une typologie qui vous paraît cohérente.

Exemple :

- Nouvelles fonctionnalités
- Fonctionnalités expérimentales
- Améliorations mineures et corrections de bugs
- Evolutions susceptibles de 'casser' le code de vos utilisateurs (*breaking changes*)

`

Accordez-vous sur système de codification de vos versions

La recommandation dans l'écosystème R est d'avoir une numérotation du type `majeur.mineur.patch.dev`. Par exemple : `v1.0.0.0` ou `v0.0.0.9000`.

Vous avez le droit de lui attribuer aussi des noms français ou anglais, souvent on se base sur thème précis. Par exemple, les femmes célèbres : "v1.0.0 Rosa Parks", "v1.1.0 Marie Curie", ...)

En général, le numéro associé à *dev* doit être très grand (exemple, 9000), afin de bien montrer qu'on est ici sur une version non stable de notre logiciel.

`

Publier une version de votre package

La montée de votre package vers une nouvelle version doit être indiquée à 3 endroits différents :

Dans le fichier DESCRIPTION

C'est ce qui est compris par R, c'est le numéro de version que vous retrouvez dans le panneau "packages" de l'interface Rstudio.

Dans le fichier NEWS.md

C'est ce fichier qui sera compréhensible par vos utilisateurs, cela permet de les informer de la vie de votre package.

Dans github ou gitlab

Ce qui permettra à utilisateur de pouvoir installer une version précise de votre package.

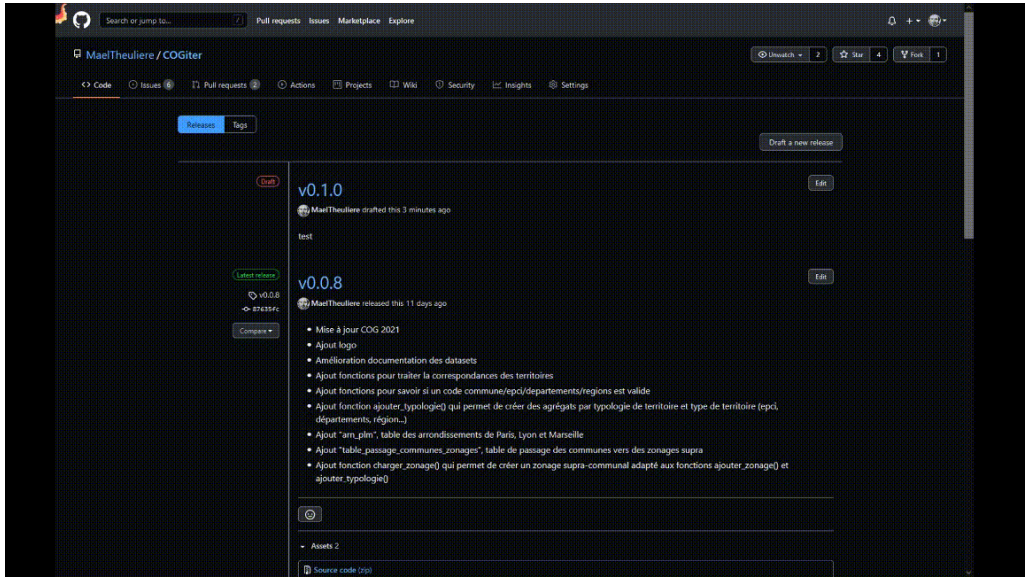
`

Publier une version de votre package

Une fois encore, {`usethis`} à la rescousse 🦸

`usethis::use_version()` vous permet de compléter le fichier DESCRIPTION, le NEWS.md et de commiter cette modification immédiatement. Cette fonction prend en paramètre le saut de version que l'on veut faire (paramètre `which` qui peut prendre les valeurs "major", "minor", "patch", "dev").

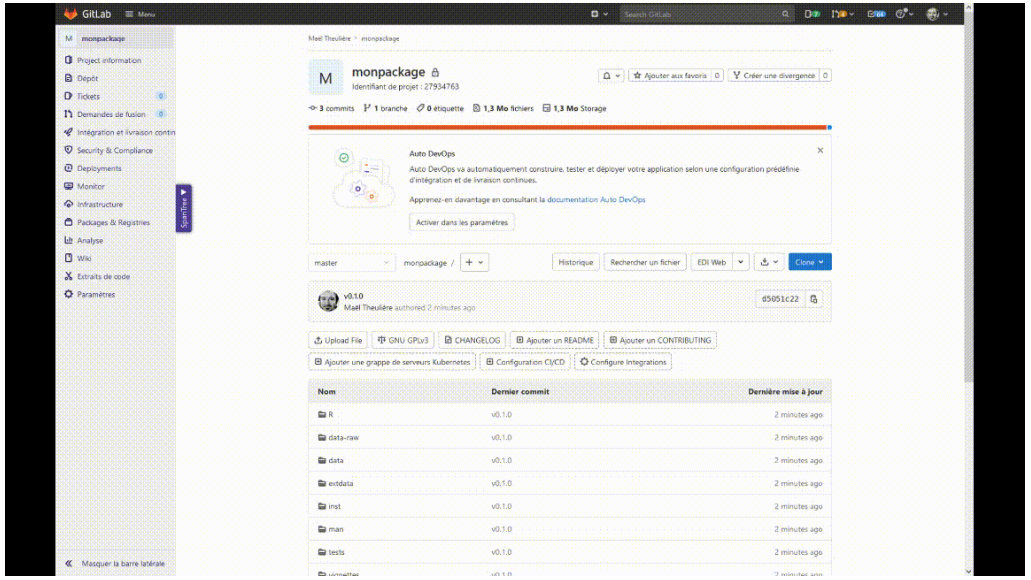
`usethis::use_github_release()` vous permet de créer un pré-publication de votre version active sur github. Il vous reste ensuite manuellement à la publier depuis l'interface de github.



`

Publier une version de votre package

Sur gitlab, l'opération de publication de la version est à réaliser manuellement depuis l'interface.



`

En pratique : la check list synthétique ✅

A partir de la branche de dev contenant les nouvelles fonctionnalités ou autres nouveautés/amélioration :

1. Check de la branche à `0 error, 0 warnings, 0 notes`.
2. Incrémenter le n° version sur DESCRIPTION.
3. Dans News.md, sous le nouveau n° de version, proposer une présentation structurée des nouveautés.
4. Faire un commit de release, toujours dans la branche de dev, puis un push de la branche.
5. Fusionner la branche de dev dans master/main.
6. Utiliser la fonction release de github (`usethis::use_github_release()`) ou gitlab.

Les étapes 2., 3. et 4. peuvent être initiées grâce à `usethis::use_version()`.

`

Détail sur l'étape 5. gestion des branches git

Deux options pour la fusion :

- On peut rassembler les différents commits de la branche de dev en un seul sur master/main :
    - oui si peu de développements : correction de bug, ajout d'un patch
    - non si implémentation de plusieurs nouvelles fonctionnalités :
      normalement chaque nouvelle fonctionnalité a été développée dans une branche adhoc et a été versée dans la branche de dev en un seul commit

- On peut supprimer la branche source (branche de dev) : non, sauf si à moyen/court terme les développements sont terminés.

`

Pour allez plus loin : rétro-compatibilité

La livraison d'une nouvelle versions majeure (en dehors de la v1.0.0) est réservée aux évolutions susceptibles de 'casser' le code des utilisateurs (*breaking changes*).

Même si on cherche au maximum à éviter ces désagréments, il ne faut pas avoir peur de changer de braquet quand on se rend compte d'une erreur de design, ou que maintenir la rétro-compatibilité devient de plus en plus difficile.

Les choses ne vont généralement pas aller en s'améliorant, et le plus tôt les breaking changes sont apportés est généralement le mieux. Il est toutefois recommandé de procéder par étape, en détaillant la position des fonctions, arguments vis à vis du cycle de vie du package.

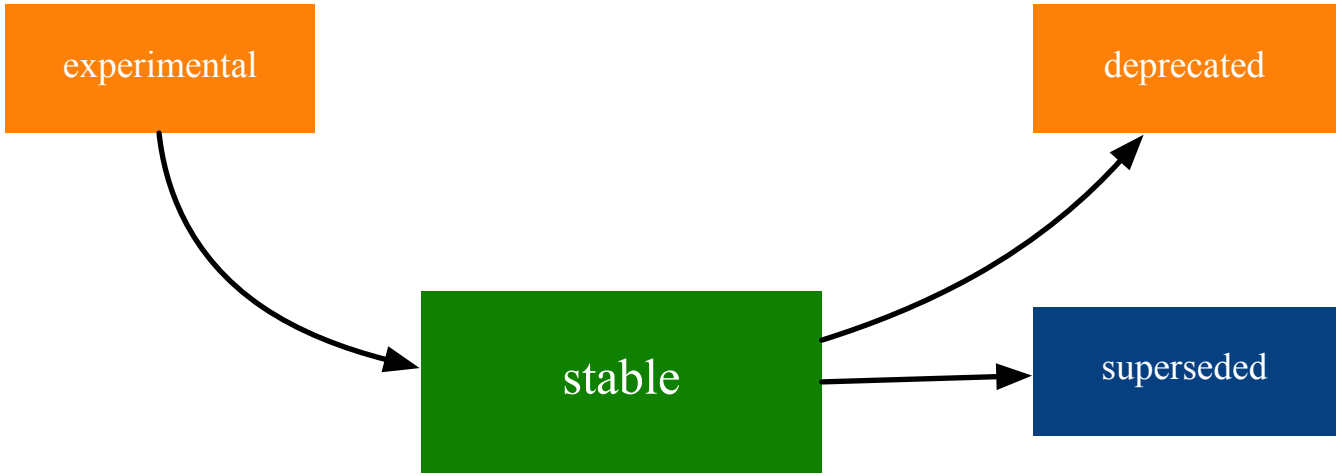
Pour en savoir plus : https://r-pkgs.org/release.html#compatibility

`

Pour allez plus loin : cycle de vie

Le package {`lifecycle`} nous aide à gérer les ruptures de rétro-compatibilité en proposant une série d'outils et de conventions pour gérer le cycle de vie des fonctions des packages.

Il distingue 4 états majeurs dans le cycle de vie : lifecycle stable, lifecycle deprecated (déprécié), lifecycle superseded (supplanté), and lifecycle experimental (expériemental).



Ces 4 états ou statuts peuvent s'appliquer aux packages, fonctions, arguments de fonction, et même valeurs spécifiques d'arguments de fonction.

`

Pour allez plus loin : cycle de vie

Pour en savoir plus :

- lire la vignette "stages" pour apprendre ce que signifient ces 4 statuts ;
- lire la vignette "manage" pour apprendre à gérer les changements d'états des fonctions qu'on utilise (càd prendre en compte les avertissements de dépréciation pour bannir des fonctions ou des manières de les utiliser...);
- lire la vignette "communicate" pour apprendre à communiquer sur les changement d'état de nos fonctions.

`