Pour un namespace tout en souplesse

Swann Floc'hlay*

Résumé

Si l'on vous demande *"Comment est votre namespace?"*, que répondez-vous ?

- *"vide, je ne programme qu'en `Base-R`"*
- *"automatique, c'est `{roxygen2}` qui s'en charge"*
- *"nébuleux, j'en suis à ma 30ème dépendance"*
- *"mon namespace ?"*

Quelle que soit votre réponse, ajouter des dépendances à son package est un passage quasi obligé lorsqu'on développe. Cela permet de ne pas avoir à réinventer la roue et de bénéficier des projets open source disponibles autour de soi. Pour garder le fil, il est possible de lister ces dépendances dans un fichier `NAMESPACE` à la racine de son package.

Cependant, toutes les dépendances n'ont pas forcément le même public. Par exemple, les fonctions de `{testthat}` seront utiles aux développeurs, mais pas aux utilisateurs. Les dépendances n'auront aussi pas la même importance si elles sont systématiquement appelées, ou propres à une petite gamme de paramètres. Chaque dépendance vient aussi avec son lot plus ou moins conséquent de sous-dépendances, et ses évolutions aux fils des montées de version.

Avec tout ça en tête, comment bien ficeler son `NAMESPACE`, en s'assurant de ne rien oublier, sans trop en rajouter ?

Mots-clefs : Package - Reproductibilité - Integration Continue - Dépendances

Développement

Dans cette présentation, je vous propose de passer un `NAMESPACE` à la loupe grâce à l'intégration continue (CI), puis de réorganiser les dépendances à travers une catégorisation entre `Suggests` et `Imports`.

Le fichier `NAMESPACE` contient à la fois les fonctions à importer depuis les dépendances, et les fonctions du package à exporter dans l'environnement de l'utilisateur. Ce fichier peut être rempli manuellement, ou à l'aide de `{roxygen2}` et des balises de documentation (*e.g.* `@importFrom`, `@export`). L'absence d'une référence dans ce fichier peut causer une erreur du type `could not find function "x"`.

Le `R CMD CHECK` est un outil utile pour corriger ces erreurs. Il s'assure que toutes les dépendances sont installées et que les exemples se lancent correctement. Il y a toutefois quelques trous dans la raquette, car il ne vérifie pas si nos fonctions débordent hors du namespace, et utilisent des packages non référencés. À moins de désinstaller un à un nos packages, difficile à dire.

C'est ici que le `CI` entre en jeu ! L'utilisation de l'intégration continue permet de valider les besoins en packages externes à partir d'une coquille basique de R, sans aucune pré-installation. On peut y détecter les dépendances non référencées manquantes, qui cette fois seront inaccessibles lors de l'appel au `R CMD CHECK`.

*ThinkR, swann@thinkr.fr

Une fois notre liste de dépendance complétée, il faut être vigilant à ne pas tomber dans l'excès de zèle et référencer tout ça dans le même panier. La catégorisation des dépendances entre `Imports` et `Suggests` hiérarchise les besoins pour passer sous la limite de dépendances surnuméraires du `R CMD CHECK`. Cette refonte finale sera balisée par des appels à `requirenamespace()`, pour garantir une réutilisation du package sans retour de bâton.