aeet
ASOCIACIÓN ESPAÑOLA DE ECOLOGÍA TERRESTRE

Ecosistemas 29(1): 1948 [Enero-Abril 2020]
https://doi.org/10.7818/ECOS.1948

ecosistemas
REVISTA CIENTÍFICA DE ECOLOGÍA Y MEDIO AMBIENTE

ISSN 1697-2473

disponible en www.revistaecosistemas.net

COMUNICACIÓN BREVE

Cómo crear paquetes de R

D. García-Callejas[1,2,*], M. de la Cruz Rot[3]

(1) Estación Biológica de Doñana (CSIC). Américo Vespucio 26, 41092 Sevilla.
(2) Departamento de Biología, Instituto Universitario de Investigación Marina (INMAR). Universidad de Cádiz, Puerto Real.
(3) Departamento de Biología y Geología, Física y Química Inorgánica. ESCET. Universidad Rey Juan Carlos. Móstoles.

* Autor de correspondencia: D. García-Callejas [david.garcia.callejas@gmail.com]

> Recibido el 21 de febrero de 2020 - Aceptado el 24 de febrero de 2020

García-Callejas, D., De la Cruz, M. 2019. Cómo crear paquetes de R. *Ecosistemas* 28(3): 1948. https://doi.org/10.7818/ECOS.1948

¿Por qué crear un paquete de R?

En el lenguaje de programación R (R Core Team 2019), los paquetes son la mejor manera de organizar, mantener y distribuir código y documentación. La motivación más directa para crear un paquete es la facilidad con que permiten compartir código con otros usuarios, pero de igual manera resulta extremadamente útil crear paquetes "de consumo propio", por ejemplo en cuanto tengamos funciones propias que utilizamos en diferentes proyectos (véase De la Cruz (2019) para la creación de funciones en R). Incluir un paquete de R, bien documentado, en el material suplementario de un artículo o de un informe científico garantiza la reproducibilidad de los resultados y promueve la difusión de los mismos.

Existen numerosos recursos en internet con consejos y recetas para crear paquetes de R. La referencia fundamental es el manual oficial "Writing R Extensions", que acompaña cada instalación de R, centrado en la construcción manual, paso a paso, de los paquetes. Además, Wickham (2015) (disponible en http://r-pkgs.had.co.nz/; segunda edición en camino disponible en https://r-pkgs.org/) proporciona una visión alternativa basada en la automatización de las tareas necesarias para la construcción y, sobre todo, el mantenimiento de paquetes. En esta sucinta nota, proporcionaremos una visión muy básica de la creación de paquetes usando la funcionalidad de los paquetes `devtools` (Wickham et al. 2019) y `roxygen2` (Wickham et al. 2018). Con la instrucción:

```
install.packages("devtools", dependencies=TRUE)
```

nos aseguramos de tener instalados ambos paquetes.

Estructura básica

El primer paso para crear un paquete es la creación de la estructura de directorios (carpetas) asociada al mismo. Para nuestro pequeño ejemplo, crearemos un paquete llamado `paqueteR`. Independientemente del nombre tan poco original escogido para nuestro ejemplo, es importante elegir nombres explicativos y fáciles de recordar (véase http://r-pkgs.had.co.nz/package.html#naming). Una vez tengamos un nombre interesante, lo primero que necesitamos es crear una carpeta llamada igual que nuestro paquete y, dentro de ella, un archivo llamado `DESCRIPTION` (tal cual, sin extensión) que contenga, como mínimo, el siguiente texto:

```
Package: paqueteR
Version: 0.1
encoding: UTF-8
```

Estos pasos se pueden automatizar usando la función `create_package` del paquete `usethis`, o desde Rstudio, creando un proyecto nuevo en forma de paquete. El archivo `DESCRIPTION` suele contener, además de los campos mencionados, multitud de información, incluyendo un pequeño resumen de la funcionalidad, los autores y, en su caso, las dependencias del paquete, es decir, los paquetes auxiliares que utilicemos en las funciones de nuestro propio paquete. En el manual oficial mencionado anteriormente se puede consultar en detalle el contenido de este importante archivo.

Añadir y documentar código

El siguiente paso es añadir el código que queremos incluir en nuestro paquete. Siguiendo con nuestro ejemplo, queremos que nuestro `paqueteR` incluya dos funciones:

```
sqrtfun1 <- function(x) {sqrt(x)}
sqrtfun2 <- function(x) {x^(1/2)}
```

En general, es recomendable guardar cada función en un archivo independiente que tenga el mismo nombre que la función. Esto facilita la modularidad del código y evita potenciales duplicidades. En nuestro caso, por tanto, tendríamos un archivo `sqrtfun1.R` y otro `sqrtfun2.R`. Para incluir ambas funciones en nuestro paquete, hemos de crear una carpeta llamada `R` dentro del directorio principal, y mover allí nuestras funciones.

Con estos sencillos pasos ya tenemos la estructura básica de un paquete, pero antes de "construirlo" es necesario documentar nuestro código, para facilitar su distribución y uso por parte de otros usuarios (o de nosotros mismos en un futuro). Generar documentación a través de `roxygen2` es muy sencillo. Basta con añadir, al comienzo de nuestros archivos .R, una serie de campos que expliquen el uso de la función en cuestión (el comando 'Insert Roxygen Skeleton' en RStudio puede ayudar generando una plantilla). En nuestras funciones, los archivos documentados tendrían la siguiente forma:

```
#' Raíz cuadrada 1
#'
#' Función para calcular la raíz cuadrada.
#'
#' @param x Un vector, o array, numérico o complejo
#'
#' @return Raíz cuadrada de x
#' @export
#'
#' @examples sqrt(10)
sqrtfun1 <- function(x) {sqrt(x)}
```

Como se puede observar, la parte dedicada a la documentación viene distinguida por dos caracteres especiales al comienzo de cada línea (`#'`). La primera línea indica el título de la función, y las siguientes pueden llevar una descripción más detallada de la misma. Cada argumento de la función viene documentado por un campo `@param`, y el tipo de dato que devuelve la función viene explicado en el campo `@return`. El campo `@export` indica que la función estará disponible para los usuarios del paquete, y en `@examples` podemos incluir ejemplos de uso de nuestra función. Estos campos corresponden exactamente con los que podemos ver en la ayuda de cualquier función de R; existen varios campos más que, por concreción, no detallamos aquí pero pueden ser consultados en la ayuda o en las viñetas de `roxygen2`. Una vez nuestras funciones están apropiadamente documentadas, y con el directorio de trabajo en el directorio raiz de nuestro paquete, podemos procesar la documentación con

```
devtools::document()
```

Si no ha habido ningún problema, esta acción habrá generado un archivo nuevo en nuestro directorio llamado `NAMESPACE`, así como un archivo .Rd en la subcarpeta `man` por cada función de nuestro paquete. El archivo `NAMESPACE` es una guía que indica a R 1) si una función del paquete está disponible para los usuarios (una función exportable) o no (una función interna) y 2) dónde se encuentran las funciones de otros paquetes que se emplean dentro del código de nuestras funciones. Cada uno de los archivos .Rd contiene la documentación que hemos creado y servirá para generar las páginas de ayuda y el manual del paquete cuando realicemos la instalación.

Creación del paquete

Con esta estructura básica, ya tenemos todos los requisitos para construir nuestro `paqueteR`. Para este último paso basta con teclear en la terminal de R (esta instrucción y la siguiente, todavía desde el directorio raiz del paquete)

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

Esta función generará un archivo comprimido .tar.gz con nuestro flamante paquete listo para ser compartido. Para instalar el paquete, usamos el comando

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

Podemos comprobar si todo ha ido bien cargando nuestro paquete como cualquier otro e invocando directamente las funciones o la ayuda sobre las mismas.

```
library(paqueteR)
sqrtfun1(25)
?sqrtfun1
```

Frente a la facilidad para crear un paquete sencillo como este, la única pega que le podemos poner a `devtools` es que no siempre respeta los caracteres propios del español (por ejemplo, la **ñ** o las vocales con tilde) en los ficheros de ayuda. Teniendo en cuenta que lo habitual es escribir la documentación de los paquetes de R en inglés, esto no suele suponer un problema (que por otra parte se puede solucionar o bien evitando dichos caracteres o bien adoptando la aproximación ortodoxa promovida en el manual oficial y aprovechando la posibilidad de codificación alternativa).

Además de lo esbozado en esta nota, los paquetes de R pueden incluir conjuntos de datos, y son estructuras ideales para trabajar con plataformas de control de versiones como git y el servidor github. En internet se pueden encontrar multitud de tutoriales específicos, entre los cuales recomendamos el desarrollado por Karl Broman. El código necesario para reproducir este documento y el `paqueteR` usado como ejemplo se puede consultar en GitHub.

Agradecimientos

Al grupo de EcoInformática de la AEET por sus comentarios y sugerencias. MCR agradece la financiación del proyecto REMEDINAL TE-CM (S2018/EMT-4338). DGC agradece la financiación del proyecto SIMPLEX (MINECO PRPCGL2017-92436-EXP).

Referencias

De la Cruz, M. 2019. Como escribir funciones en R. *Ecosistemas* 28(3): 213-216. https://doi.org/10.7818/ECOS.1880

R Core Team 2019. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Wickham, H. 2015. *R Packages: Organize, Test, Document, and Share Your Code.* O'Reilly media Inc., Champaign, IL 61820, Estados Unidos.

Wickham, H., Danenberg, P., Eugster, M. 2018. *roxygen2: In-Line Documentation for R*. R Studio. Boston, MA, Estados Unidos. Disponible en: https://roxygen2.r-lib.org/

Wickham, H., Hester, J., Chang, W. 2019. *devtools: Tools to Make Developing R Packages Easier*. R Studio. Boston, MA, Estados Unidos. Disponible en: https://devtools.r-lib.org/