Paquetes de rOpenSci: Desarrollo, mantenimiento y revisión por pares

Equipo editorial para la revisión de software de rOpenSci (actuales y pasados): Brooke A

Tabla de contenidos

Guía de desarrollo de rOpenSci

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 3.0 Estados Unidos de América. Consulta el DOI de Zenodo de la version original y el DOI de la traducción para saber como citarlas.

Ejemplo:

```
rOpenSci editorial team (2024). Paquetes rOpenSci: Desarrollo, mantenimiento y revisión por p
```

También puedes leer la versión PDF de este libro.

Prefacio

¡Te damos la bienvenida! Este libro es una guía para quienes desarrollan y mantienen paquetes y quienes revisan y editan envíos de paquetes a rOpenSci.

La primera sección del libro contiene nuestras recomendaciones para crear y *testear* los paquetes de R.

La segunda sección está dedicada al proceso de revisión por pares de software de rOpenSci: en qué consiste, nuestras políticas y las guías específicas para orientar a quienes crean, mantienen, revisan y editan a lo largo del proceso. Para la *revisión de software estadístico*, consulta la página web del proyecto y los recursos asociados.

La tercera y última sección presenta nuestras buenas prácticas para hacer crecer tu paquete una vez que ha sido incorporado a rOpenSci: cómo colaborar con otras personas, cómo documentar cada versión, cómo promover tu paquete y cómo aprovechar GitHub como plataforma de desarrollo. La tercera sección también incluye un capítulo para quienes desean comenzar a contribuir a los paquetes de rOpenSci.

Esperamos que esta guía te resulte útil y clara, y agradecemos tus sugerencias en las issues del libro. ¡Mucho éxito desarrollando paquetes de R!

El equipo editorial de rOpenSci.

Este libro es un documento vivo. Puedes ver las actualizaciones de nuestras buenas prácticas y políticas a través de las notas de cada versión.
Puedes citar este libro utilizando sus metadatos de Zenodo y su DOI.

Em Markowitz (NOAA) · Sam Albers · Toph Allen · Kaique dos S. Alves · Brooke Anderson · Alison Appling · Zebulun Arendsee · Taylor Arnold · Al-Ahmadgaid B. Asaad · Dean Attali · Mara Averick · Suzan Baert · James Balamuta · Vikram Baliga · David Bapst · Joëlle Barido-Sottani · Allison Barner · Cale Basaraba · John Baumgartner · Marcus Beck · Gabriel Becker · Jason Becker · Salvador Jesus Fernandez Bejarano · Dom Bennett · Ken Benoit · Aaron Berdanier · Fred Boehm · Carl Boettiger · Will Bolton · Ben Bond-Lamberty · Anne-Sophie Bonnet-Lebrun · Alison Boyer · Abby Bratt · François Briatte · Eric Brown · Julien Brun · Jenny Bryan · Lukas Burk · Lorenzo Busetto · Maria Paula Caldas · Mario Gavidia Calderón · Carlos Cámara-Menoyo · Brad Cannell · Joaquin Cavieres · Kevin Cazelles · Scott Chamberlain · Cathy Chamberlin · Jennifer Chang · Pierre Chausse · Jorge Cimentada · Nicholas Clark · Chase Clark · Jon Clayden · Dena Jane Clink · Will Cornwell · Nic Crane · Enrico Crema · Verónica Cruz-Alonso · Ildiko Czeller · Tad Dallas · Kauê de Sousa · Christophe Dervieux · Amanda Dobbyn · Jasmine Dumas · Dewey Dunnington · Remko Duursma · Mark Edmondson · Paul Egeler

· Evan Eskew · Harry Eslick · Denisse Fierro-Arcos · Alexander Fischer · Kim Fitter · Robert M Flight · Sydney Foks · Stephen Formel · Zachary Stephen Longiaru Foster · Auriel Fournier · Kaija Gahm · Zach Gajewski · Carl Ganz · Duncan Garmonsway · Jan Laurens Geffert · Sharla Gelfand · Monica Gerber · Duncan Gillespie · David Gohel · A. Cagri gokcek · Guadalupe Gonzalez · Rohit Goswami · Laura Graham · Charles Gray · Matthias Grenié · Corinna Gries · Hugo Gruson · Ernest Guevarra · W Kyle Hamilton · Ivan Hanigan · Jeffrey Hanson · Liz Hare · Jon Harmon · Rayna Harris · Ted Hart · Nujcharee Haswell · Verena Haunschmid · Stephanie Hazlitt · Andrew Heiss · Max Held · Anna Hepworth · Bea Hernandez · Jim Hester · Peter Hickey · Roel Hogervorst · Kelly Hondula · Allison Horst · Sean Hughes · James Hunter · Brandon Hurr · Ger Inberg · Christopher Jackson · Najko Jahn · Tamora D James · Veronica Jimenez-Jacinto · Mike Johnson · Will Jones · Max Joseph · Megha Joshi · Krunoslav Juraic · Soumya Kalra · Zhian N. Kamvar · Michael Kane · Andee Kaplan · Tinula Kariyawasam · Hazel Kavılı · Jonathan Keane · Christopher T. Kenny · Os Keyes · Eunseop Kim · Aaron A. King · Michael Koontz · Bianca Kramer · Will Landau · Sam Lapp · Erin LeDell · Thomas Leeper · Sam Levin · Lisa Levinson · Stephanie Locke · Marion Louveaux · Robin Lovelace · Julia Stewart Lowndes · Tim Lucas · Muralidhar, M.A. · Andrew MacDonald · Jesse Maegan · Mike Mahoney · Tristan Mahr · Yohann Mansiaux · Paula Andrea Martinez · Anthony Martinez · Joao Martins · Ben Marwick · Claire Mason · Miles McBain · Lucy D'Agostino McGowan · Amelia McNamara · Elaine McVey · Bryce Mecum · Nolwenn Le Meur · François Michonneau · Mario Miguel · Helen Miller · Beatriz Milz · Jessica Minnier · Priscilla Minotti · Nichole Monhait · Kelsey Montgomery · Ronny A. Hernández Mora · Paula Moraga · Natalia Morandeira · George Moroz · Ross Mounce · Athanasia Monika Mowinckel · Lincoln Mullen · Matt Mulvahill · Maria Victoria Munafó · David Neuzerling · Dillon Niederhut · Joel Nitta · Rory Nolan · Kari Norman · Jakub Nowosad · Matt Nunes · Daniel Nüst · Lauren O'Brien · Joseph O'Brien · Paul Oldham · Samantha Oliver · Dan Olner · Jeroen Ooms · Victor Ordu · Luis Osorio · Philipp Ottolinger · Mark Padgham · Marina Papadopoulou · Edzer Pebesma · Thomas Lin Pedersen · Antonio J. Pérez-Luque · Marcelo S. Perlin · Rafael Pilliard-Hellwig · Rodrigo Neto Pires · Lindsay Platt · Nicholas Potter · Joanne Potts · Josep Pueyo-Ros · Etienne Racine · Manuel Ramon · Nistara Randhawa · David Ranzolin · Quentin Read · Neal Richardson · tyler rinker · Emily Robinson · David Robinson · Alec Robitaille · Francisco Rodriguez-Sanchez · Sam Rogers · Julia Romanowska · Xavier Rotllan-Puig · Bob Rudis · Edgar Ruiz · Kent Russel · Michael Sachs · Sheila M. Saia · Chitra M Saraswati · Alicia Schep · Klaus Schliep · Clemens Schmid · Patrick Schratz · Collin Schwantes · Marco Sciaini · Eric Scott · Heidi Seibold · David Selby · Julia Silge · Margaret Siple · Peter Slaughter · Mike Smith · Tuija Sonkkila · Øystein Sørensen · Jemma Stachelek · Aymeric Stamm · Christine Stawitz · Irene Steves · Kelly Street · Matt Strimas-Mackey · Alex Stringer · Michael Sumner · Chung-Kai Sun · Sarah Supp · Emi Tanaka · Jason Taylor · Filipe Teixeira · Andy Teucher · Jennifer Thompson · Joe Thorley · Nicholas Tierney · Tiffany Timbers · Tan Tran · Tim Trice · Utku Turk · Zoë Turner · Kyle Ueyama · Ted Underwood · Adithi R. Upadhya · Kevin Ushey · Josef Uyeda · Frans van Dunné · Mauricio Vargas · Remi Vergnon · Jake Wagner · Ben Ward · Elin Waring · Rachel Warnock · Leah Wasser · David Watkins · Lukas Weber · Marc Weber · Karissa Whiting · Stefan Widgren · Anna Willoughby · Saras Windecker · Luke Winslow · David Winter · Sebastian Wójcik · Witold Wolski · Kara Woo · Marvin N. Wright · Jacob Wujciak-Jens · Bruna Wundervald · Lauren Yamane · Emily Zabor · Taras Zakharko · Sherry Zhang · Hao Zhu · Chava Zibman · Naupaka Zimmerman · Jake Zwart · Felipe · santikka · kasselhingee · Bri · Flury · Vincent · eholmes · Pachá · Rich · Claudia · Jasmine · Zack · Lluís · becarioprecario · gaurav

Si quieres contribuir sugerencias o correcciones a este libro, visita el repositorio de GitHub y, en particular, la guía de contribución. ¡Gracias!

Agradecemos a quienes crean, mantienen, revisan y editan paquetes por ayudarnos a mejorar el sistema y esta guía a lo largo de los años. Gracias también a las siguientes personas que han contribuido a esta guía y a sus versiones anteriores: Katrin Leinweber, John Baumgartner, François Michonneau, Christophe Dervieux, Lorenzo Busetto, Ben Marwick, Nicholas Horton, Chris Kennedy, Mark Padgham, Jeroen Ooms, Sean Hughes, Jan Gorecki, Joseph Stachelek, Dean Attali, Julia Gustavsen, Nicholas Tierney, Rich FitzJohn, Tiffany Timbers, Hilmar Lapp, Miles McBain, Bryce Mecum, Jonathan Carroll, Carl Boettiger, Florian Privé, Stefanie Butland, Daniel Possenriede, Hadley Wickham, Mauro Lepore, Matthew Fidler, Luke McGuinness, Aaron Wolen, Indrajeet Patil, Kevin Wright, Will Landau, Hugo Gruson„ Hao Ye„ Sébastien Rochette„ Edward Wallace„ Alexander Fischer„ Maxime Jaunatre„ Thomas Zwagerman.. Por favor, avísanos si nos olvidamos de reconocer tu contribución.

Parte I

Construyendo tu paquete

1 Guía de empaquetado

rOpenSci acepta paquetes que cumplen con nuestras recomendaciones a través de un proceso fluido de Revisión por pares de software. Para garantizar un estilo consistente entre todas nuestras herramientas, hemos escrito este capítulo en el que se destacan recomendaciones para el desarrollo de paquetes. Por favor, también lee y aplica nuestro capítulo sobre integración continua (CI). En la tercera sección de este libro, que comienza con un capítulo sobre la colaboración, ofrecemos una guía para continuar luego del proceso de revisión.

Recomendamos a quienes desarrollen paquetes que lean el libro de Hadley Wickham y Jenny Bryan sobre el desarrollo de paquetes. Es exhaustivo y está disponible gratis en línea (pero en inglés). Algunas partes de nuestra guía son redundantes con respecto a otros recursos, pero destaca las recomendaciones propias de rOpenSci.

Para leer por qué vale la pena enviar un paquete a rOpenSci siguiendo las recomendaciones, echa un vistazo a razones para enviar un paquete.

1.1 Nombre del paquete y metadatos

1.1.1 Cómo elegir el nombre de tu paquete

- Recomendamos fuertemente elegir nombres cortos y descriptivos en minúsculas. Si tu paquete se conecta con uno o más servicios comerciales, asegúrate de que el nombre no infringe las recomendaciones de la marca. Puedes comprobar si el nombre de tu paquete está disponible, es informativo y no es ofensivo (en inglés) utilizando la función `pak::pkg_name_check()`; también utiliza un motor de búsqueda porque asi verías si es ofensivo en otras idiomas que inglés. En particular, *no* elijas un nombre de paquete que no esté disponible en CRAN o en Bioconductor.
- Existe un equilibrio entre las ventajas de un nombre de paquete único y un nombre de paquete menos original.
  - Un nombre poco común tiene la ventaja de que su uso es más fácil de detectar (habrá menos falsos positivos al buscar el nombre en la búsqueda de código de GitHub para que tú o rOpenSci evalúen su uso) y de buscar (para que quienes lo usen puedan buscar “cómo usar el paquete <nombre_paquete>?” en internet).

  - Por otro lado, un nombre *demasiado* único puede hacer que el paquete sea más difícil de descubrir (es decir, difícil de que aparezca como resultado de la búsqueda "cómo hago esto en R"). Es un argumento para que el nombre de tu paquete sea algo muy cercano al tema, como geojson.

- Este artículo de Nick Tierney (en inglés) lista otras consideraciones interesantes a tener en cuenta al nombrar paquetes de R. En caso de que cambies de opinión sobre el nombre de tu paquete, este segundo artículo de Nick explica cómo cambiar el nombre de tu paquete.

1.1.2 Crear metadatos para tu paquete

Recomendamos usar el paquete `codemetar` para crear y actualizar un archivo JSON CodeMeta para tu paquete usando `codemetar::write_codemeta()`. Incluirá automáticamente toda la información útil, incluyendo temas de GitHub. CodeMeta utiliza términos de Schema.org por lo que, a medida que gane popularidad, los metadatos JSON de tu paquete podrían ser utilizados por servicios de terceros, incluso por motores de búsqueda.

1.2 Plataformas

- Los paquetes deben funcionar en todas las plataformas principales (Windows, macOS, Linux). Puede haber excepciones para paquetes que interactúen con funciones específicas del sistema, o que adapten utilidades que sólo funcionan en plataformas limitadas, pero se debe hacer todo lo posible para garantizar la compatibilidad entre plataformas, incluyendo la compilación específica en cada sistema, o la contenerización de utilidades externas.

1.3 API del paquete

1.3.1 Nombres de funciones y argumentos

- Los nombres de las funciones y los argumentos deben elegirse para que funcionen juntos y formen una API de programación común y lógica que sea fácil de leer y autocompletar.

  - Considera un esquema de nomenclatura `objeto_verbo()` para las funciones de tu paquete que toman un tipo de datos común o interactúan con una API común. `objeto` se refiere a los datos/API y `verbo` a la acción principal. Este esquema ayuda a evitar conflictos de nombres con paquetes que pueden tener verbos similares, y hace que el código sea legible y fácil de autocompletar. Por ejemplo, en **stringi** las funciones que empiezan con `stri_` manipulan cadenas de texto (`stri_join()`, `stri_sort()` y en **googlesheets**

las funciones que empiezan con `gs_` son llamadas a la API de Google Sheets (`gs_auth()`, `gs_user()`, `gs_download()`).

- Para las funciones que manipulan un objeto o dato y devuelven un objeto o dato del mismo tipo, haz que el objeto o dato sea el primer argumento de la función para mejorar la compatibilidad con el operador pipe (`|>` de R base, `%>%` del paquete magrittr).
- Recomendamos fuertemente usar `snake_case` por sobre todos los otros estilos, a menos que estés usando funciones de un paquete que ya se utilice ampliamente.
- Evita los conflictos de nombres de funciones con los paquetes de R base u otros paquetes populares (por ejemplo `ggplot2`, `dplyr`, `magrittr`, `data.table`)
- Los nombres y el orden de los argumentos deben ser coherentes entre las funciones que utilizan inputs similares.
- Las funciones del paquete que importan datos no deben importar los datos al entorno global, sino que deben devolver objetos. En general, se debe evitar asignar objectos al entorno global.

1.3.2 Mensajes de la consola

- Utiliza el paquete cli o las herramientas de R base (`message()` y `warning()`) para comunicarte con las personas que usen tus funciones.
- Lo más destacado del paquete cli incluye: empaquetado automático, respeto de la convención NO_COLOR muchos elementos semánticos y amplia documentación. Más información en este artículo.
- No utilices `print()` o `cat()` a menos que sea para un método `print.*()` o `str.*()` ya que estos métodos de impresión de mensajes son más difíciles de silenciar.
- Proporciona una forma para suprimir la verbosidad, preferiblemente a nivel de paquete: haz que la creación de mensajes dependa de una variable u opción de entorno (como "usethis.quiet" en el paquete usethis), en lugar de en un parámetro de la función. El control de los mensajes podría ser a varios niveles ("ninguno,"informar", "depurar") en lugar de lógico (ningún mensaje / todos los mensajes). El control de la verbosidad es útil para usuarios/as finales, pero también en las pruebas. Puedes encontrar más comentarios interesantes en este issue de la guía de diseño de tidyverse.

1.3.3 Interfaces interactivas o gráficas

Si proporcionas una interfaz gráfica de usuario (GUI) (como una aplicación Shiny), para facilitar el flujo de trabajo incluye un mecanismo para reproducir automáticamente los pasos realizados en la GUI. Esto puede incluir la generación automática del código necesario para reproducir los mismos resultados, la generación de valores intermedios producidos en la herramienta interactiva, o simplemente un mapeo claro y bien documentado entre las acciones de la GUI y las funciones usadas. (Ver también "Tests" más abajo).

El paquete `tabulizer` por ejemplo, tiene un flujo de trabajo interactivo para extraer tablas, pero también puede extraer sólo coordenadas, por lo que se pueden volver a ejecutar los mismos pasos como un script. Otros dos ejemplos de aplicaciones shiny que generan código son https://gdancik.shinyapps.io/shinyGEO/y https://github.com/wallaceEcoMod/wallace/.

1.3.4 Comprobación de entradas

Recomendamos que tu paquete utilice un método coherente de tu elección para comprobar las entradas – ya sea R base, un paquete de R o ayudantes personalizados.

1.3.5 Paquetes que envuelven recursos web (clientes API)

Si tu paquete accede a una API web o a otro recurso web,

- asegúrate de que las peticiones envían un agente de usuario es decir, una forma de identificar qué (tu paquete) o quién envió la solicitud. Se debe poder anular el agente de usuario por defecto del paquete. Lo ideal sería que el agente de usuario fuera diferente en los servicios de integración continua y en los de desarrollo (basándose, por ejemplo, en los nombres de usuario de GitHub de quienes desarrollan).
- Puede que elijas valores por defecto diferentes (mejores) que los de la API, en cuyo caso deberías documentarlos.
- Tu paquete debería ayudar con la paginación, permitiendo que quienes lo usen no se preocupen en absoluto de ella, ya que tu paquete realiza todas las peticiones necesarias.
- Tu paquete debe ayudar a limitar la tasa de acuerdo con las normas de la API.
- Tu paquete debe reproducir los errores de la API, y posiblemente explicarlos en mensajes de error informativos.
- Tu paquete podría exportar funciones de alto nivel y funciones de bajo nivel, estas últimas permitiendo a las personas que lo usen llamar directamente a los puntos finales de la API con más control (como `gh::gh()`).

Para más información, consulta el artículo del blog Por qué deberías (o no) crear un cliente API.

1.4 Estilo del código

- Para más información sobre cómo dar estilo a tu código, nombrar funciones y scripts de R dentro de la carpeta `R` te recomendamos que leas el capítulo "Code" (Código) del libro R Packages. Recomendamos el paquete `styler` para automatizar parte del estilo del código. También te sugerimos que leas la Guía de estilo de Tidyverse (en inglés).
- Puedes optar por utilizar = en lugar de <- siempre que seas coherente con esa elección dentro de tu paquete. Recomendamos evitar el uso de -> para la asignación dentro del paquete. Si utilizas <- en todo tu paquete, y también utilizas `R6` en ese paquete, debrás utilizar = para la asignación dentro de la generación de la clase `R6Class` - esto no se considera una incoherencia porque no puedes usar <- en este caso.

1.5 Archivo CITATION

- Si tu paquete aún no tiene un archivo CITATION, puedes crear uno con `usethis::use_citation()` y llenarlo con los valores generados por la función `citation()`.
- CRAN exige que los archivos CITATION se declaren como items `bibentry` y no en la forma aceptada previamente de `citEntry()`.
- Si archivas cada versión de tu repositorio de GitHub en Zenodo, añade el DOI principal de Zenodo al archivo CITATION.
- Si un día **luego de** la revisión en rOpenSci publicas un artículo sobre tu paquete, añádelo al archivo CITATION.
- Menos relacionado con tu paquete en sí mismo, pero con lo que lo soporta: si tu paquete incluye un recurso concreto, como una fuente de datos o, por ejemplo, un algoritmo estadístico, recuérdale a quien lo use cómo citar ese recurso mediante, por ejemplo, `citHeader()`. Quizás incluso añade la referencia a ese recurso.

Como ejemplo, revisa el archivo CITATION de dynamite donde se refiere al manual, así como a otras publicaciones asociadas.

```
citHeader("To cite dynamite in publications use:")

bibentry(
  key = "dynamitepaper",
  bibtype  = "Misc",
  doi = "10.48550/ARXIV.2302.01607",
  url = "https://arxiv.org/abs/2302.01607",
  author = c(person("Santtu", "Tikka"), person("Jouni", "Helske")),
  title = "dynamite: An R Package for Dynamic Multivariate Panel Models",
  publisher = "arXiv",
```

```
  year = "2023"
)
bibentry(
  key = "dmpmpaper",
  bibtype  = "Misc",
  title    = "Estimating Causal Effects from Panel Data with Dynamic
    Multivariate Panel Models",
  author = c(person("Santtu", "Tikka"), person("Jouni", "Helske")),
  publisher = "SocArxiv",
  year     = "2022",
  url      = "https://osf.io/preprints/socarxiv/mdwu5/"
)

bibentry(
  key = "dynamite",
  bibtype  = "Manual",
  title    = "Bayesian Modeling and Causal Inference for Multivariate
    Longitudinal Data",
  author = c(person("Santtu", "Tikka"), person("Jouni", "Helske")),
  note  = "R package version 1.0.0",
  year     = "2022",
  url      = "https://github.com/ropensci/dynamite"
)
```

- También puedes crear y almacenar un archivo `CITATION.cff` gracias al paquete cffr que también proporciona un flujo de trabajo con GitHub Actions para mantener el archivo `CITATION.cff` actualizado.

1.6 README

- Todos los paquetes deben tener un archivo *README*, llamado `README.md` en la raíz del repositorio. El *README* debe incluir, en este orden:
  - El nombre del paquete.
  - Etiquetas de integración continua y la cobertura de tests, la etiqueta de revisión por pares de rOpenSci una vez que haya comenzado (ver más abajo), una etiqueta de repostatus.org, y cualquier otra (por ejemplo R-universe).
  - Una breve descripción de los objetivos del paquete (¿qué hace? ¿por qué seria interesante usarlo?) con enlaces descriptivos a todas las viñetas (renderizadas, es decir, legibles,

por ejemplo el sitio web de documentación) a menos que el paquete sea pequeño y sólo haya una viñeta que repita el *README*. Asegúrate también de que las viñetas estén renderizadas y sean legibles, consulta la sección "sitio web de documentación").
  - Instrucciones de instalación utilizando, por ejemplo, el paquete remotes, pak o R-universe.
  - Cualquier configuración adicional necesaria (tokens de autenticación, etc).
  - Una breve demostración de uso.
  - Si aplica, cómo se compara el paquete con otros paquetes similares y/o cómo se relaciona con otros paquetes.
  - Información sobre cómo citar el paquete. Es decir, indica el formato de cita preferida en el *README* con el texto "así es como se puede citar mi paquete". Véase, por ejemplo el *README* de ecmwfr.

Si utilizas otra etiqueta de estado del repo, como el ciclo de vida, por favor añade también una etiqueta de repostatus.org . Ejemplo de un *README* de un repo con dos insignias de estado.

- Una vez que hayas enviado el paquete y haya completado la revisión editorial, añade la insignia de revisión por pares como

```
[![](https://badges.ropensci.org/<issue_id>_status.svg)](https://github.com/ropensci/software
```

donde issue_id es el número del *issue* en el repositorio donde se hizo la revisión del software. Por ejemplo, la etiqueta de revisión de `rtimicropem` utiliza el número 126, ya que es el número de *issue* de revisión. La etiqueta indicará primero "en revisión" y luego "revisado" una vez que tu paquete haya sido incorporado (*issue* etiquetado como "aprobado" y cerrado), y estará vinculado con el *issue* de la revisión.

- Si tu *README* tiene muchas etiquetas, considera la posibilidad de ordenarlas en una tabla HTML para que sea más fácil ver la información de un vistazo. Consulta los ejemplos en el repo de `drake` y en el repo de `qualtRics`. Las secciones posibles son:

  - Desarrollo (estados de CI, ver el capítulo CI, canal de Slack para la discusión, repostatus)
  - Edición o publicación (etiquetas de la versión de CRAN y de la fecha de publicación de METACRAN, etiqueta de la API de chequeos de CRAN, etiqueta de Zenodo)
  - Estadísticas o uso (descargas, por ejemplo etiquetas de descarga de r-hub/cranlogs)

  La tabla debe ser más ancha que larga para enmascarar el resto del *README*.

- Si tu paquete se conecta a una fuente de datos o a un servicio en línea, o utiliza otro software, ten en cuenta que el *README* de tu paquete puede ser el punto de entrada para alguien que lo usa por primera vez. Debe proporcionar suficiente información para poder entender la naturaleza de los datos, el servicio o el software, y proporcionar enlaces a otros datos y documentación relevantes. Por ejemplo un *README* no debe limitarse a decir: "Proporciona

acceso a GooberDB", sino que también debe incluir "…, un repositorio online de avistamientos de Goober en Sudamérica. Se puede encontrar más información sobre GooberDB, y la documentación de la estructura de la base de datos y metadatos en *este enlace*".

- Recomendamos no crear el `README.md` directamente, sino a partir de un archivo `README.Rmd` (un archivo R Markdown) si incluye código de ejemplo. La ventaja del archivo `.Rmd` es que puedes combinar el texto con el código que puede actualizarse fácilmente cada vez que se actualice tu paquete.
- Considera utilizar `usethis::use_readme_rmd()` para generar una plantilla para el archivo `README.Rmd` y para configurar automáticamente un chequeo que garantice que `README.md` sea siempre más reciente que `README.Rmd` antes de hacer un commit.
- Los ejemplos largos deben incluirse sólo en las viñetas. Si quieres que las viñetas sean más accesibles antes de instalar el paquete, te sugerimos crear un sitio web para tu paquete.
- Añade un código de conducta y una guía de contribución.
- Consulta el *README* de `gistr` para ver un buen ejemplo de *README* de un paquete pequeño, y el *README* de `bowerbird` para un buen ejemplo de *README* de un paquete más grande.

1.7 Documentación

1.7.1 General

- Todas las funciones exportadas del paquete deben estar completamente documentadas con ejemplos.
- Si existe un posible solapamiento o confusión con otros paquetes que ofrezcan una funcionalidad similar o tengan un nombre parecido, añade una nota en el *README*, en la viñeta principal y, potencialmente, en el campo descripción de archivo DESCRIPTION. Por ejemlo, el *README* de rebird README o del paquete slurmR (que no es parte de rOpensci).
- El paquete debe contener la documentación general del paquete para `?paquete` (o ?`paquete-package``` si hay un conflicto de nombres). Opcionalmente, puedes utilizar tanto `?paquete` como?paquete-package``` para el archivo del manual del paquete utilizando la etiqueta@aliasesde roxygen. [usethis::use_package_doc lib.org/reference/use_package_doc.html) añade la plantilla para generar la documentación general.
- El paquete debe contener al menos una viñeta en formato **HTML** que cubra una parte importante de las funciones del paquete, ilustrando casos de uso realistas y cómo se supone que las funciones interactúen entre ellas. Si el paquete es pequeño, la viñeta y el *README* pueden tener un contenido muy similar.

- Al igual que el *README*, la documentación general o las viñetas puede ser el punto de entrada para alguien que lo usa por primera vez. Si tu paquete se conecta a una fuente de datos o a un servicio en línea, o utiliza otro software, debe proporcionar suficiente información para poder entender la naturaleza de los datos, el servicio o el software, y proporcionar enlaces a otros datos y documentación relevantes. Por ejemplo, la introducción de una viñeta o la documentación no debería limitarse a decir: "Proporciona acceso a GooberDB", sino incluir también: "…, un repositorio online de avistamientos de Goober en Sudamérica. Puedes encontrar más información sobre GooberDB y la documentación de la estructura de la base de datos y los metadatos en *este enlace*". Cualquier viñeta debe incluir que conocimientos previos son necesarios para poder entenderla.

La viñeta general debe presentar una serie de ejemplos que progresen en complejidad desde el uso básico al avanzado.

- La funciones de uso avanzado o que sean usadas sólo para desarrollo pueden incluirse en una viñeta aparte (por ejemplo, la programación usando evaluación no estándar con dplyr).
- El *README*, la documentación del paquete de nivel superior, las viñetas, los sitios web, etc., deben tener suficiente información al principio para obtener una visión general del paquete y de los servicios o datos a los que se conecta, y proporcionar navegación a otras partes de la documentación relevante. Esto es para seguir el principio de *múltiples puntos de entrada*; es decir, tener en cuenta el hecho de que cualquier pieza de documentación puede ser el primer encuentro de la persona con el paquete y/o la herramienta o los datos que accede.
- La(s) viñeta(s) debe(n) incluir citas a software y de artículos cuando corresponda.
- Si tu paquete proporciona acceso a una fuente de datos, requerimos que la DESCRIPCIÓN contenga (1) nna breve identificación y/o descripción de la organización responsable de generar datos; y (2) un URL con la página pública que proporciona, describe o permite el acceso a los datos (que a menudo puede diferir de la URL que conduce directamente a la fuente de datos).
- Utiliza mensajes de inicio en el paquete sólo cuando sea necesario (cuando algunas funciones son enmascaradas, por ejemplo). Evita los mensajes de inicio del paquete tales como "Esto es paquete 2.4-0" o la guía de cómo citarlo porque pueden ser molestos quien lo usa. Utiliza la documentación para brindar ese tipo de información.
- Puedes optar por tener una sección en el *README* sobre casos de uso de tu paquete (otros paquetes, artículos de blog, etc.). Por ejemplo, esta sección en el *README* del paquete vcr).

1.7.2 Uso de roxygen2

- Pedimos que todos los paquetes que se presenten para revisión utilicen roxygen2 para generar la documentación. roxygen2 es un paquete de R que compila automáticamente archivos

`.Rd` en la carpeta `man` del paquete utilizando etiquetas incluidas antes de cada función. roxygen2 tiene soporte para la sintaxis Markdown. Una ventaja clave de utilizar roxygen2 es que tu archivo `NAMESPACE` siempre se generará automáticamente y estará actualizado.

- Más información sobre el uso de roxygen2 para generar documentación está disponible en el libro *R Packages* y en el sitio web de roxygen2.
- Si escribiste los archivos .Rd directamente sin roxygen2, el paquete Rd2roxygen contiene funciones para convertir la documentación de Rd a documentación de roxygen2.
- Todas las funciones deben documentar el tipo de objeto devuelto bajo el encabezado `@return`.
- El valor por defecto de cada parámetro debe estar claramente documentado. Por ejemplo, en lugar de escribir "Un valor lógico que determina si ..." deberías escribir "Un valor lógico (por defecto `TRUE`) que determina si ...". También es una buena práctica indicar los valores por defecto directamente en la definición de tu función:

```
f <- function(a = TRUE) {
  # código de la función
}
```

- La documentación debe ayudar a la navegación incluyendo links entre funciones relacionadas y agrupando la documentación de funciones relacionadas en página de ayuda comunes. Para esto recomendamos la etiqueta `@seealso`, que crea automáticamente enlaces *"See also"* (ver también, en inglés) y puede agrupan las funciones en sitios web generados con pkgdown. Consulta la sección "manual" del libro *R Packages* y la sección "Agrupación de funciones" de este capítulo para más detalles.
- Puedes reutilizar partes de la documentación (por ejemplo, detalles sobre la autenticación, paquetes relacionados) en las viñetas, *README* y páginas de manual. Consulta la viñeta de roxygen2 sobre la reutilización de documentación.
- Para incluir ejemplos, puedes utilizar el clásico `@examples` (en plural *"examples"*), pero también la etiqueta `@example <path>` (en singular *"example"*) para almacenar el código de ejemplo en un script R independiente (idealmente en `man/`), y la etiqueta `@exampleIf` para ejecutar ejemplos condicionalmente y evitar fallos de `R CMD check`. Consulta la documentación de roxygen2 sobre ejemplos.
- Añade `#' @noRd` a las funciones internas. Quizá te interese el paquete experimental devtag para obtener páginas de manual locales al utilizar `#' @noRd`.
- A partir de la versión 7.0.0 de roxygen2, las clases `R6` son oficialmente compatibles. Consulta la documentación de roxygen2 para saber cómo documentar las clases `R6`.

1.7.3 URLs en la documentación

Esta subsección es especialmente relevante para quienes deseen enviar su paquete a CRAN. CRAN comprobará las URLs incluidas en la documentación y no permitirá páginas que redirijan a códigos 301. Puedes utilizar el paquete urlchecker para reproducir los checks de CRAN y, en particular, sustituir las URLs originales por las URLs a las que redirigen. Otra opción es mostrar algunas URLs de manera explícita sin link (cambiar `<https://ropensci.org/>` por `https://ropensci.org/` o `\url{https://ropensci.org/}` por `https://ropensci.org/`), pero si lo haces, tendrás que implementar algún tipo de comprobación de URL para evitar que se rompan sin que te des cuenta. Además, no se podrá hacer click en los enlaces desde la documentación local.

1.8 Sitio web de documentación

Te recomendamos que crees un sitio web con la documentación de tu paquete utilizando `pkgdown`. Hay un capítulo sobre pkgdown (en inglés) en libro "R packages", y, cómo no es de extrañar, el paquete `pkgdown` tiene su propio sitio web de documentación.

Hay algunos elementos que nos gustaría destacar aquí.

1.8.1 Construcción automática del sitio web de documentación

Sólo tendrás que preocuparte por la construcción automática de tu sitio web cuando se apruebe y se transfiera el repositorio de tu paquete a la organización ropensci; de hecho, después de eso se construirá un sitio web con pkgdown para tu paquete luego de cada *push* al repositorio de GitHub. Puedes encontrar el estado de estas acciones en `https://dev.ropensci.org/job/nombre_paquete` (por ejemplo para `magick`) y el sitio web en `https://docs.ropensci.org/nombre_paquete` (por ejemplo para `magick`). La construcción del sitio web utilizará el archivo de configuración de pkgdown si tienes uno, excepto para el estilo, ya que utilizará el paquete `rotemplate`. El sitio web resultante tendrá una barra de búsqueda local. Por favor, informa de los errores, y haz preguntas o pedidos de nuevas características sobre la construcción del sitio centralizada en https://github.com/ropensci/docs/ y sobre la plantilla en https://github.com/ropensci/rotemplate/.

Si las viñetas de tus paquetes necesitan credenciales (claves de API, tokens, etc.) para generarse, es posible que quieras pregenerar las viñetas ya que las credenciales no se pueden utilizar en el servidor que genera la documentación.

Antes de presentar y transferir tu paquete, puedes utilizar el enfoque documentado por `pkgdown` o el paquete `tic` para construir el sitio web de tu paquete automáticamente. Esto te ahorrará el trabajo de ejecutar (y acordarte de ejecutar) `pkgdown::build_site()` cada vez que haya que actualizar el sitio. Consulta nuestro capítulo sobre integración continua si ésto no te resulta familiar. En cualquier caso, no olvides actualizar la URL del sitio web en todos los lados donde aparezca después de hacer la transferencia a la organización ropensci.

1.8.2 Agrupar funciones en el índice

Cuando tu paquete tenga muchas funciones, es conveniente que aparezcan agrupadas en el índice de la documentación, lo cual se puede hacer de forma más o menos automática.

Si utilizas roxygen2 versión 6.1.1 o mayor, puedes utilizar la etiqueta `@family` en la documentación de tus funciones para indicar un grupo al que pertenecen. Esto generará enlaces las funciones en la documentación local del paquete instalado (en la sección "See also") *y* te permitirá utilizar la función `has_concept` de `pkgdown` en el archivo de configuración de tu sitio web. Puedes ver un ejemplo (de un paquete no perteneciente a rOpenSci) cortesía de `optiRum`: etiqueta family, archivo de configuración de `pkgdown` y la sección en el índice resultante. Para personalizar el texto del título de la referencia cruzada creada por roxygen2 (`Other {family}:`), puedes consultar la documentación de roxygen2 sobre cómo proporcionar una lista `rd_family_title` en `man/roxygen/meta.R`.

Menos automáticamente, puedes ver el sitio web de `drake` y el archivo de configuración asociado como ejemplo.

1.8.3 Marca de autoría

Puedes hacer que los nombres de (algunas) personas autoras del paquete aparezcan como links añadiendo su URL, e incluso puedes sustituir sus nombres por un logo (por ejemplo rOpenSci… ¡o tu organización/empresa!). Ver la documentación de `pkgdown`.

1.8.4 Configurar la barra de navegación

Puedes hacer que el contenido de tu sitio web sea más fácil de navegar modificando la barra de navegación, consulta `pkgdown` documentación. En particular, ten en cuenta que si el nombre de la viñeta principal de tu paquete es "pkg-name.Rmd", ésta será accesible desde la barra de navegación en la sección `Para empezar` en vez de en `Artículos > Título de la Viñeta`.

1.8.5 Mathjax

Una vez que tu paquete sea transferido y obtenga un sitio web utilizando nuestra plantilla de `pkgdown`, si quieres utilizar Mathjax tendrás que especificarlo en el archivo de configuración de `pkgdown` de la siguiente manera

```
template:
  params:
    mathjax: true
```

1.8.6 Logo del paquete

Para utilizar el logo de tu paquete en la página de inicio de pkgdown, consulta `usethis::use_logo()`. Si tu paquete no tiene ningún logotipo, el generador de documentación de rOpenSci utilizará el logo de rOpenSci en su lugar.

1.9 Autoría

El archivo `DESCRIPTION` de un paquete debe enumerar las personas que participaron de la autoría y quienes colaboraron en el paquete, utilizando la sintaxis `Authors@R` para indicar sus funciones (*author*/*creator*/*contributor*, etc.) si hay más de una persona listada. Utiliza el campo de comentarios para indicar el ID ORCID de cada persona, si tuviera (ver este post). Revisa esta sección de "Writing R Extensions" (Escribiendo extensiones de R) para más detalles. Si crees que quienes revisaron tu paquete han hecho una contribución sustancial al desarrollo de tu paquete, puedes agregar sus nombres en el campo `Authors@R` usando el tipo de contribución `"rev"`, por ejemplo:

```
person("Bea", "Hernández", role = "rev",
       comment = "Bea revisó el paquete (v. X.X.XX) para rOpenSci, ver <https://github.c
```

No incluyas a nadie en tu paquete sin antes pedir su consentimiento. Lee más en este artículo de blog *Thanking Your Reviewers: Gratitude through Semantic Metadata* (Agradeciendo las revisiones: La gratitud a través de los metadatos semánticos). Por favor, no agregues al equipo editorial. ¡Su participación y contribución a rOpenSci es suficiente agradecimiento!

1.9.1 Autoría del código incluido en el paquete

Muchos paquetes incluyen código de otro software. Tanto si se incluyen archivos enteros como funciones individuales de otros paquetes, los paquetes de rOpenSci deben seguir la *Política del Repositorio CRAN*:

> La propiedad de los derechos de autor y de propiedad intelectual de todos los componentes del paquete debe ser clara e inequívoca (incluso en la especificación de autoría en el archivo DESCRIPTION). Cuando el código se copie (o se derive) del trabajo de otras personas (incluso del propio R), hay que cuidar de conservar las declaraciones de derechos de autor/licencia y no se tergiverse la autoría.
>
> Preferiblemente, se utilizaría un campo "Autors@R" con el rol "ctb" para quienes tiene la autoría de dicho código. Como alternativa, el campo "Autor" puede listarlas como contribuyentes.

Cuando los derechos de autor los tenga una entidad distinta de quienes mantienen la autoría del paquete, se indicará preferentemente mediante el rol “cph” en el campo “Autors@R”, o utilizando un campo “Copyright” (si es necesario, redirigiendo a un archivo inst/COPYRIGHTS).

Deben respetarse las marcas comerciales.

1.10 Licencia

El paquete debe tener una licencia aceptada por CRAN u OSI. Para más detalles sobre las licencias, consulta el libro R packages.

1.11 Testeo

- Todos los paquetes deben aprobar `R CMD check`/`devtools::check()` en las plataformas principales.
- Todos los paquetes deben tener un conjunto de *tests* que cubran la funcionalidad principal del paquete. Los *tests* deben cubrir también el comportamiento del paquete en caso de error.
- Es una buena práctica escribir *tests* unitarios para todas las funciones, y para todo el código del paquete en general, asegurando que se cubra las funcionalidades clave. Si la cobertura de los *tests* en tu paquete está por debajo del 75% probablemente requerirá *tests* adicionales o una explicación de la baja cobertura antes de ser enviado para su revisión.
- Recomendamos utilizar testthat para escribir los *tests*. Intenta escribir *tests* a medida que escribas cada nueva función. Esto responde a la necesidad obvia de tener *tests* adecuados para el paquete, pero te permite pensar en varias formas en las que una función puede fallar, y programar *defensivamente* contra esas fallas. Más información sobre *tests*.
- Los *tests* deben ser fáciles de entender. Te sugerimos que leas el artículo de blog *Why Good Developers Write Bad Unit Tests* (Por qué las personas que son buenas desarrollando escriben malos tests) de Michael Lynch.
- Los paquetes con aplicaciones Shiny deberán generar *tests* unitarios usando `shinytest2` o `shinytest` para comprobar que las interfaces interactivas funcionan como es esperado.
- Para testear las funciones que crean gráficos, sugerimos utilizar vdiffr, una extensión del paquete testthat que esta basada en tests con instantáneas de testthat.

- Si tu paquete interactúa con recursos web (APIs web y otras fuentes de datos en la web) el libro *HTTP testing in R* (Testeando HTTP en R) de Scott Chamberlain y Maëlle Salmon puede resultarte relevante. Algunos paquetes que ayudan a realizar *tests* HTTP (y sus clientes HTTP correspondientes) son:
  - httptest2 (httr2);
  - httptest (httr);
  - vcr (httr, crul);
  - webfakes (httr, httr2, crul, curl).
- El paquete testthat tiene una función `skip_on_cran()` que puedes utilizar para que algunos *tests* no se ejecuten en CRAN. Recomendamos utilizarla en todas las funciones que tengan llamadas a APIs web, ya que es muy probable que fallen en CRAN. Estos *tests* deberán ejecutarse en la integración continua. Ten en cuenta que a partir de testthat 3.1.2, `skip_if_offline()` llama automáticamente a `skip_on_cran()`. Más información en *CRAN preparedness for API wrappers* (Preparación en CRAN para utilización de APIs).
- Si tu paquete interactúa con una base de datos, dittodb puede resultarte útil.
- Una vez que hayas configurado la integración continua (CI), utiliza el informe de cobertura de tu paquete (revisa esta sección de nuestro libro) para identificar las líneas no testeadas y añadir más *tests*.
- Dado que a menudo algunos tests se omiten en la integración continua, te recomendamos asegurarte de que todos se ejecuten antes de enviar tu paquete ejecutándolos localmente (puede que tengas que establecer `Sys.setenv(NOT_CRAN="true")`).

1.12 Ejemplos

- Incluye ejemplos extensos en la documentación. Además de demostrar cómo se utiliza el paquete, pueden ser una forma fácil de testear la funcionalidad del paquete antes de que haya *tests* adecuados. Sin embargo, ten en cuenta que exigimos *tests* en los paquetes contribuidos.
- Puedes ejecutar los ejemplos con `devtools::run_examples()`. Ten en cuenta que los ejemplos que no estén rodeados de `\dontrun{}` o `\donttest{}` serán ejecutados cuando ejecutes R CMD CHECK o su equivalente (por ejemplo, `devtools::check()`). Consulta la tabla de resumen en la documentación de roxygen2.
- Para evitar que los ejemplos se ejecuten en CRAN (por ejemplo, si requieren autenticación), tienes que utilizar `\dontrun{}`. Sin embargo, para un primer envío, CRAN no te permitirá saltearte todos los ejemplos. En este caso puedes añadir algunos pequeños ejemplos de juguete, o encapsular el código de los ejemplos en `try()`. Consulta también la etiqueta `@exampleIf`, que al momento de escribir este artículo, se encuentra en la versión de desarrollo de roxygen2.

- Además de ejecutar los ejemplos localmente en tu propia computadora, te aconsejamos fuertemente que ejecutes los ejemplos en uno de los sistemas de integración continua. De nuevo, se ejecutarán los ejemplos que no estén incluidos en `\dontrun{}` o `\donttest{}`. Puedes configurar la integración continua para que éstos se ejecuten a través de los argumentos de R CMD `--run-dontrun` y/o `--run-donttest`.

1.13 Dependencias del paquete

- Pensa en las ventajas y desventajas de depender de un paquete. Por un lado, las dependencias reducen el esfuerzo de desarrollo, y permite construir en base a funcionalidades útiles desarrolladas por otras personas, especialmente si la dependencia realiza tareas complejas, es de alto rendimiento y/o está bien revisada y probada. Por otro lado, tener muchas dependencias supone una carga de mantenimiento ya que requiere estar al día con los cambios en esos paquetes, con riesgo para la sostenibilidad de tu paquete a largo plazo. También aumenta el tiempo y el tamaño de la instalación, lo que supone principalmente una consideración en el ciclo de desarrollo tuyo y del resto, y en los sistemas de compilación automatizados. Los paquetes "pesados" -los que tienen muchas dependencias, y los que tienen grandes cantidades de código compilado- aumentan este costo. He aquí algunos enfoques para reducir las dependencias:
  - Si sólo utilizas unas pocas funciones de una dependencia grande o pesada, puedes copiarlas en tu propio paquete. (Ver la sección *Autoría* para saber cómo reconocer la autoría del código copiado). Por otro lado, las funciones complejas con muchos casos especiales (por ejemplo, los analizadores sintácticos) requieren considerable testeo y revisión.
    * Un ejemplo común de esto es devolver "tibbles" usadas por el tidyverse en las funciones del paquete que proporcionan datos. Se puede evitar el uso del paquete **tibble** devolviendo un tibble creado modificando un data.frame de la siguiente manera

      ```
      class(df) <- c("tbl_df", "tbl", "data.frame")
      ```

      (Ten en cuenta que este enfoque no está universalmente aceptado.)
  - Asegúrate de que utilizas la función del paquete donde está definida originalmente y no de un paquete que la re-exporta. Por ejemplo, muchas funciones de **devtools** pueden encontrarse en paquetes especializados más pequeños, como **sessioninfo**. La función %>% debe importarse de **magrittr** donde está definida, en lugar de **dplyr** que la reexporta y es mucho más pesado.
  - Algunas dependencias proporcionan nombres de funciones y sintaxis más fáciles de interpretar que los nombres de las funciones y la sintaxis de R base. Si ésta es la razón principal para usar una función en una dependencia pesada, considera incluir el código

de R base en una función interna bien nombrada en tu paquete. Consulta, por ejemplo, el script de R de rlang que proporciona funciones con una sintaxis similar a las funciones de purrr.

  - Si las dependencias tienen funcionalidades redundantes, considera depender de una sola en lo posible.
  - Puedes encontrar más consejos sobre la gestión de dependencias en [el capítulo *"Dependencies: Mindset and Background"* (Dependencias: Mentalidad y Contexto) del libro *"R packages"* y en este artículo de Scott Chamberlain (en Inglés).

- Utiliza la sección `Imports` en lugar de `Depends` para listar los paquetes cuyas funciones usas en tu paquete. Utiliza sección `Suggests` para listar los paquetes que usas en los *tests* (`testthat`), y para generar la documentación (`knitr`, roxygen2) (si utilizas `usethis` para añadir la infraestructura de *tests* con `usethis::use_testthat()` o una viñeta mediante usethis::use_vignette() los paquetes necesarios se añadirán a `DESCRIPTION`). Si utilizas algún paquete en los ejemplos o *tests* de tu paquete, asegúrate de listarlo en `Suggests` si no aparece ya en `Imports`.
- Si tu paquete no está en Bioconductor pero depende de paquetes de Bioconductor, asegúrate de que las instrucciones de instalación en el *README* y la viñeta sean lo suficientemente claras incluso para una persona que no esté familiarizada con el ciclo de publicación de Bioconductor.
  - ¿Hay que usar `BiocManager` (recomendado)? Documenta esto.
  - ¿La instalación automática de paquetes de Bioconductor con `install.packages()` es suficiente? En ese caso, menciona que se debe ejecutar `setRepositories()` si aún no han configurado los repositorios de Bioconductor necesarios.
  - Si tu paquete depende de Bioconductor a partir de una determinada versión, menciónalo en el archivo DESCRIPTION y en las instrucciones de instalación.
- Especificar las dependencias mínimas (por ejemplo `glue (>= 1.3.0)` en lugar de sólo `glue`) debería ser una elección deliberada. Si sabes con certeza que tu paquete se romperá con una dependencia debajo de una determinada versión, especifícalo explícitamente. Pero si no lo sabes, no es necesario especificar una dependencia mínima. En ese caso, puedes agregarlo si una persona informa de un fallo que está explícitamente relacionado con una versión antigua de una dependencia. Considerar las versiones de paquetes locales como la versiones mínimas necesarias para las dependencias es una mala práctica. Eso obligaría a todo el mundo a actualizar las dependencias innecesariamente (causando problemas con otros paquetes) cuando no hay una buena razón detrás de esa elección de versiones.
- En la mayoría de los casos en los que debes exponer funciones que vienen de otros paquetes, debes importar y reexportar esas funciones individuales en lugar de listarlos en el campo `Depends`. Por ejemplo, si las funciones de tu paquete producen objetos `raster` puedes reexportar, del paquete raster, sólo las funciones de impresión y graficado.

- Si tu paquete utiliza una dependencia de *sistema*, debes
  - Indicarla en el archivo DESCRIPTION;
  - Comprobar que aparece en la lista de `sysreqsdb` para permitir que las herramientas automáticas lo instalen, o enviar una contribución si no es así;
  - Comprobar que está instalada usando un script `configure` (ejemplo) y devolver un mensaje de error útil si no se la encuentra (ejemplo). Los scripts `configure` pueden ser difíciles de escribir, ya que a menudo requieren soluciones rebuscadas para asegurarse de que dependencias del sistema muy distintas funcionen en todos los sistemas. Utiliza ejemplos como punto de partida (más aquí), pero ten en cuenta que es habitual encontrar errores y casos límite y que a menudo violan las políticas de CRAN. No dudes en pedir ayuda en nuestro foro.

1.14 Andamiaje recomendado

- Para las consultas HTTP recomendamos utilizar httr2, httr, curl, crul, en vez de RCurl. El paquete curl es mejor si prefieres los clientes de bajo nivel de HTTP, mientras que httr2, httr o crul son mejores para un control de nivel más alto.
- Para parsear JSON, utiliza jsonlite en lugar de rjson o RJSONIO.
- Para parsear, crear y manipular XML, recomendamos fuertemente xml2 en la mayoría de los casos. Puedes consultar las notas de Daniel Nüst sobre la migración de XML a xml2 (en Inglés).
- Para datos espaciales, el paquete sp debe considerarse obsoleto en favor de sf. Los paquetes rgdal, maptools y rgeos se retirarán a finales de 2023. Recomendamos el uso de las suites espaciales desarrolladas por las comunidades r-spatial y rspatial. Consulta este issue de GitHub para ver las discusiones pertinentes.

1.15 Control de versiones

- Los archivos fuente de tu paquete tienen que estar bajo control de versiones, más concretamente versionados con Git. Puede que el paquete gert te resulte útil, así como algunas de las funciones de usethis relacionadas con Git/GitHub; sin embargo, puedes utilizar git como quieras.
- Asegúrate de listar archivos innecesarios, como `.DS_Store`, en .gitignore. La función `usethis::git_vaccinate()`, y el paquete gitignore pueden resultarte útil para esto.
- Una sección posterior de este libro incluye algunos consejos sobre el flujo de trabajo con git.

1.16 Problemas comunes en CRAN

Esta es una colección de problemas en CRAN que vale la pena evitar desde el principio.

- Asegúrate de que las palabras del título de tu paquete comiencen con mayúsculas (lo que en inglés se denomina *Title Case*).
- No pongas un punto al final de tu título.
- No pongas "en R" o "con R" en tu título, ya que esto es obvio en los paquetes alojados en CRAN. Si a pesar de todo quieres que esta información aparezca en tu sitio web, revisa la documentación de `pkgdown` para saber cómo anular esta restricción.
- Evita empezar la descripción con el nombre del paquete o "Este paquete…".
- Asegúrate de incluir enlaces a sitios web si utilizas una API, obtienes datos de una página web, etc. en el campo `Description` de tu archivo `DESCRIPTION`. Las URL deben ir entre simbolos <>, por ejemplo `<https://www.r-project.org>`.
- Tanto en el campo `Title` como en `Description` los nombres de los paquetes u otro software externo deben ir entre comillas simples (por ejemplo *Integración de 'Rcpp' para la Biblioteca de Álgebra Lineal Armadillo*).
- Evita tests y ejemplos que tarden en correr. Considera usar `testthat::skip_on_cran()` en los test que toman mucho tiempo para que se omitan en CRAN pero sigan corriendo localmente y en la integración continua.
- Incluye archivos extra ubicados en la raíz del proyecto, como `paper.md` y archivos de configuración de integración continua, en tu archivo `.Rbuildignore`.

Para más consejos, consulta la lista colaborativa mantenida por ThinkR, "Prepárate para CRAN".

1.16.1 Comprobaciones en CRAN

Una vez que tu paquete esté en CRAN, será comprobado regularmente en diferentes plataformas. Los fallos de estas comprobaciones, cuando no son falsos positivos, pueden hacer que el equipo de CRAN se ponga en contacto contigo. Puedes revisar el estado de las comprobaciones de CRAN con:

- el paquete `foghorn`.
- las etiquetas de estado "CRAN checks Badges".

1.17 Guía para Bioconductor

Si deseas que tu paquete se envíe a Bioconductor, o si tu paquete está en Bioconductor, consulta las Directrices de paquetes de Bioconductor y el libro de desarrollo actualizado.

1.18 Otras recomendaciones

- Si envías un paquete a rOpenSci a través del repo de software-review puedes dirigir tus preguntas al equipo de rOpenSci a través de issues o en nuestro foro de discusión.
- Lee la guía para quienes crean paquetes.
- Lee, incorpora y actúa según los consejos del capítulo *Guía de Colaboración*.

1.18.1 Aprender sobre el desarrollo de paquetes

1.18.1.1 Libros

- *R Packages* (Paquetes de R) de Hadley Wickham y Jenny Bryan es un recurso excelente y fácil de leer sobre el desarrollo de paquetes que está disponible gratis en línea (y se puede comprar impreso).
- *Writing R Extensions* (Escribiendo extensiones de R) es la referencia canónica y normalmente más actualizada, para crear paquetes de R.
- *Mastering Software Development in R* (Dominar el desarrollo de software en R) por Roger D. Peng, Sean Kross y Brooke Anderson.
- *Advanced R* (R avanzado) de Hadley Wickham
- *Tidyverse style guide* (guía de estilo del equipo Tidyverse)
- *Tidyverse design guide* (guía de diseño del equipo tidyverse) (trabajo en elaboración) con su boletín de noticias.

1.18.1.2 Tutoriales

- "Your first R package in 1 hour" (tu primer paquete de R en una hora) de Shannon Pileggi.
- Esta descripción del flujo de trabajo por Emil Hvitfeldt.
- Esta ilustración de Matthew J Denny.

1.18.1.3 Blogs

- Blog de R-hub.
- Algunos posts del blog de rOpenSci por ejemplo *"How to precompute package vignettes or pkgdown articles"* (Cómo pregenerar viñetas de paquetes o artículos de pkgdown).
- Sección *El rincón del desarrollo de paquetes* del boletín de rOpenSci.
- Algunos posts del blog de tidyverse por ejemplo *"Upgrading to testthat edition 3"* (Actualizando a la 3ra edición de testthat).

1.18.1.4 MOOCs

Existe una especialización en Coursera correspondiente al libro de Roger Peng, Sean Kross y Brooke Anderson con un curso específico sobre paquetes de R.

2 Buenas prácticas de integración continua

Este capítulo explica qué es la integración continua (CI, por sus siglas en inglés) y luego resume nuestras recomendaciones sobre cómo usarla.

Junto con el capítulo anterior, forma parte de nuestras directrices para la revisión por pares de software.

2.1 ¿Qué es la integración continua?

La integración continua ejecuta tests sobre el software automáticamente. En el caso de rOpenSci, CI significa, en la práctica, que un conjunto de test se ejecutará automáticamente a través de GitHub, cada vez que hagas un *commit* o *pull request* a GitHub.

La CI automatiza la ejecución de tests globales de los paquetes, como `R CMD check` (ver tests). Es posible configurar la CI antes de escribir tus tests, entonces CI los ejecutará a medida que los envíes al repositorio.

2.2 ¿Por qué utilizar la integración continua (CI)?

Todos los paquetes de rOpenSci deben utilizar alguna forma de integración continua. Esto asegura que todos los *commits*, *pull requests* y nuevas *branches* pasan por `R CMD check`. Los resultados de todos los tests se muestran en la página del *pull request* en GitHub, lo cual proporciona información sobre los problemas y proteje de aceptar cambios que rompan tu paquete. La integración continua de los paquetes de rOpenSci también debe estar vinculada a un servicio de cobertura de código que indique cuántas líneas son chequeadas por los tests unitarios.

Tanto el estado de los tests como el porcentaje de cobertura del código deben informarse mediante etiquetas en el *README* de tu paquete.

Los paquetes de R deben tener CI para todos los sistemas operativos (Linux, Mac OSX, Windows) si contienen:

- Código compilado
- Dependencias de Java

- Dependencias de otros lenguajes
- Paquetes con llamadas al sistema
- Procesamiento de texto, por ejemplo obtener nombres de personas (para encontrar problemas de codificación).
- Cualquier cosa que incluya llamadas al sistema de archivos / rutas de acceso.

Ante la duda de si estos criterios se aplican a tu paquete, es mejor añadir CI para todos los sistemas operativos. La mayoría de los servicios de CI para paquetes de R lo permiten sin mucha complicación utilizando la configuración estándar.

2.3 ¿Qué servicio/s de integración continua usar?

Existen muchos servicios de integración continua. Algunos son servicios independientes (CircleCI, AppVeyor), mientras que otros están integrados a servicios de alojamiento del código o relacionados (GitHub Actions, GitLab, AWS Code Pipeline). Distintos servicios permiten distintas configuraciones de sistemas operativos.

GitHub Actions es una opción conveniente para quienes desarrollan paquetes de R y ya utilizan GitHub, ya que está integrado en la plataforma y es compatible con todos los sistemas operativos necesarios.
Hay acciones compatibles con el ecosistema R así como soporte de primera clase con el paquete usethis. Todos los paquetes enviados a rOpenSci para su revisión por pares son comprobados por nuestro propio sistema, `pkgcheck`, el cual se describe con más detalle en la Guía de autoría. Estos tests también están disponibles como acción de GitHub en el repositorio `ropensci-review-tools/pkgcheck-action`. Es recomendable utilizar esta acción para confirmar que el paquete pasa todos los tests antes de enviarlo. Consulta nuestro blog para más información.

El paquete usethis se puede usar con otros sistemas de CI, aunque estas funciones están siendo deprecadas. rOpenSci también provee el paquete circle, el cual ayuda a configurar cadenas de CircleCI, y el paquete tic para la construcción de cadenas de CI más complicadas.

2.3.0.1 Tests con diferentes versiones de R

Requerimos que los paquetes de rOpenSci no sólo se testeen usando la versión más reciente de R, sino también con la anterior y con la versión en desarrollo. Esto garantiza compatibilidad con R base, tanto con versiones futuras como con versiones pasadas.

Los detalles de cómo ejecutar tests utilizando diferentes versiones de R localmente se pueden encontrar en la viñeta de R-hub sobre la ejecución de Checks locales en Linux con Docker.

Puedes definir los detalles de los checks para cada una de las versiones utilizando una matriz de tests.

Si desarrollas un paquete que depende o está destinado para Bioconductor, puede que el paquete biocthis te resulte relevante.

2.3.0.2 Minimizar los tiempos de construcción en CI

Puedes utilizar estos consejos para minimizar los tiempos de construcción en CI:

- Guardar los paquetes instalados en una caché. La acción r-lib/actions lo hace por defecto.

2.3.0.3 Dependencias del sistema

Puede que encuentres útil el post de Hugo Gruson Dependencias del sistema en paquetes de R y pruebas automáticas.

2.3.1 Travis CI (Linux y Mac OSX)

Recomendamos dejar de utilizar Travis.

2.3.2 AppVeyor CI (Windows)

Para la integración continua en Windows, consulta R + AppVeyor. Configúralo con `usethis::use_appveyor()`.

Aquí tienes consejos para minimizar el tiempo de compilación de AppVeyor:

- Guarda los paquetes instalados en una caché. Mira, por ejemplo, este archivo de configuración. AppVeyor usará la caché automáticamente si lo configuras con `usethis::use_appveyor()`.
- Activa los *rolling builds*.

Ya no transferimos los proyectos de AppVeyor a la cuenta de AppVeyor de rOpenSci, así que después de transferir tu repo a la organización de GitHub "ropensci", la etiqueta será `[![AppVeyor Build Status](https://ci.appveyor.com/api/projects/status/github/ropensci/pkgname?branch=master&sv`

2.3.3 Circle CI (Linux y Mac OSX)

Circle CI es utilizado, por ejemplo, por el paquete de rOpenSci `bomrang` como servicio de integración continua.

2.4 Cobertura de tests

La integración continua también debe incluir información sobre la cobertura de los tests a través de un servicio de testing como Codecov o Coveralls.

Recomendamos utilizar Codecov. Para activar Codecov para tu repo, ejecuta `usethis::use_github_action("test-` el cual crea un archivo `.github/workflows/test-coverage.yaml`. También tienes que dar acceso a Codecov a tu repositorio de GitHub, ver la guía de inicio rápido de Codecov (en inglés) para saber cómo hacerlo. Luego añade una etiqueta de estado de Codecov en la parte superior de tu README.md, puedes consultar *Status Badges* (etiquetas de estado) en la documentación de Codecov.

Actualmente, Codecov tiene acceso a todos los repositorios de GitHub de la organización ropensci por defecto. Cuando tu repositorio sea aceptado y transferido a ropensci, el acceso de Codecov debería transferirse automáticamente. Tendrás que actualizar la URL de la etiqueta para que apunte al repositorio alojado en ropensci.

Para más detalles, consulta el README del paquete **covr**, así como la documentación de `usethis::use_coverage()` y `usethis::use_github_action()`.

Si ejecutas la cobertura en varios servicios de CI los resultados se unirán.

2.5 Más servicios de CI: OpenCPU

Después de transferir el repo a la organización GitHub "ropensci" de rOpenSci, cada *push* se chequeará en OpenCPU y la persona que hace el *commit* recibirá una notificación por correo electrónico. Este es un servicio adicional de CI para que permite correr funciones en paquetes de R de forma remota a través de https://ropensci.ocpu.io/ utilizando la API de opencpu. Para más detalles sobre este servicio, consulta la página de ayuda de OpenCPU que también indica dónde hacer preguntas.

2.6 Aún más servicios de CI: rOpenSci docs

Después de la transferencia a la organización GitHub "ropensci" de rOpenSci, cada *push* al repo de GitHub disparará la construcción (o actualización) del sitio web de tu paquete usando pkgdown. Puedes encontrar el estado de este proceso en la URL `https://ropensci.r-universe.dev/ui#packages` y en el estado del commit. El sitio web estará en `https://docs.ropensci.org/package_name` (por ejemplo para `magick`). Si tu paquete tiene un archivo de configuración de pkgdown, rOpenSci docs lo usará para crear el sitio web, excepto para el tema, que se utilizará `rotemplate` paquete.

Por favor, informa sobre errores, haz preguntas y solicita nuevas funcionalidades sobre este servicio y sobre la plantilla en https://github.com/ropensci-org/rotemplate/.

3 Buenas prácticas de seguridad en el desarrollo de paquetes

Este es un capítulo en construcción que incluye consejos sobre la gestión de información confidencial en los paquetes y enlaces a lecturas adicionales.

3.1 Miscelaneos

Recomendamos el artículo *Ten quick tips for staying safe online* (Diez consejos rápidos sobre seguridad en Internet) de Danielle Smalls y Greg Wilson.

3.2 Seguridad al acceder a GitHub

- Te recomendamos proteger tu cuenta de GitHub con autenticación de dos factores (2FA). Esto es *obligatorio* para miembros de la organización ropensci en GitHub y quienes colaboren de manera externa, así que asegúrate de habilitarla antes de que se apruebe tu paquete.
- También te recomendamos que compruebes regularmente quién tiene acceso al repositorio de tu paquete, y que elimines cualquier acceso no utilizado (como el de personas que hayan dejado de colaborar).

3.3 https

- Si el servicio web al que se conecta tu paquete tiene opciones de https y http, usa https.

3.4 Credenciales usadas en paquetes

Esta sección incluye consejos para cuando desarrolles un paquete que interactúa con un recurso web que requiere credenciales (claves API, tokens, etc.). Consulta también la viñeta de `httr` sobre compartir credenciales.

3.4.1 Credenciales de quien usa el paquete

Supongamos que tu paquete necesita una clave para hacer consultas a una API en nombre de quien usa el paquete.

- En la documentación de tu paquete, brinda consejos para que la clave de la API no se almacene en el archivo .Rhistory o en los scripts.
  - Fomenta el uso de variables de entorno para almacenar la clave de la API (o incluso elimina la posibilidad de pasarla como argumento a las funciones). Puedes mencionar esta introducción a los archivos de arranque y la función `usethis::edit_r_environ()`.
  - Otra posibilidad es usar o fomentar el uso del paquete `keyring` para almacenar variables en los sistemas de almacenamiento de credenciales del sistema operativo (que son más seguros que guardarlas en el archivo .Renviron). Para esto tu paquete tendría una función para guardar la clave en el keyring y una para recuperarla; alternativamente quien use tu paquete podría escribir `Sys.setenv(CLAVESUPERSECRETA = keyring::key_get("miservicio"))` al principio de su script.
  - No imprimas la clave de la API en ningún mensaje, advertencia o error, ni siquiera en modo verboso.
- La plantilla de issues en GitHub debe advertir de no compartir ninguna credencial. Si alguien comparte accidentalmente las credenciales en un issue, asegúrate de avisarle para que revoque la clave (es decir, pregúntale explícitamente si se ha dado cuenta de que ha compartido su clave).

3.4.2 Credenciales en paquetes y desarrollo

Durante el desarrollo del paquete tendrás que proteger tus credenciales igual que proteges las credenciales de quienes usan tu paquete, pero hay más cosas a considerar y tener en cuenta.

3.4.2.1 Credenciales y consultas guardadas en los tests

Si utilizas `vcr` o `httptest` en los test para almacenar las respuestas de la API en caché, tienes que asegurarte de que las consultas grabadas o fixtures no contengan credenciales. Consulta la guía de seguridad de `vcr` y la guía *Redacting and Modifying Recorded Requests* (Censurando y modificando consultas grabadas) de `httptest` e inspecciona tus consultas grabadas y *fixtures* antes de hacer un primer *commit* para asegurarte de que tienes la configuración correcta.

Como `vcr` es un paquete de rOpenSci, puedes hacer cualquier pregunta que tengas en el foro de rOpenSci.

3.4.2.2 Compartir credenciales con servicios de integración contínua

Si utilizas un servicio de integración continua (CI por sus siglas en inglés), es posible que necesites compartir credenciales con el mismo.

Puedes almacenar las claves de la API como variables de entorno o secretos consultando la documentación del servicio de CI.

Para más detalles y consejos sobre el flujo de trabajo, consulta al artículo de gargle *Managing tokens securely* (Gestionando claves de forma segura) y el capítulo de seguridad del libro *HTTP testing in R*.

Documenta los pasos que has seguido en CONTRIBUTING.md para que tú, o alguien en el futuro, pueda recordar cómo hacerlo la próxima vez.

3.4.2.3 Credenciales y colaboraciones

¿Qué pasa con los pull requests de contribuciones externas? En GitHub, por ejemplo, los "secretos" sólo están disponibles para las acciones de GitHub para los *pull requests* iniciados desde el propio repositorio, no desde el *fork*. Los test que utilicen tus credenciales fallarán a menos que utilices algún tipo de respuesta simulada/en caché, por lo que es posible que quieras omitirlos dependiendo del contexto. Por ejemplo, en tu cuenta CI podrías crear una variable de entorno llamada `ESTE_SOY_YO` y luego omitir los test en función de la presencia de esta variable. Obviamente, esto significa que los tests del PR por parte del servicio de CI no serán exhaustivos, por lo que tendrás que comprobar el PR localmente para ejecutar todos los tests.

Documenta el comportamiento de tu paquete frente a PRs externos en CONTRIBUTING.md por el bien de la gente que hace PRs y de la gente que los revisa (tú dentro de unas semanas, y otras personas que mantengan del paquete).

3.4.3 Credenciales y CRAN

En CRAN, omite las pruebas y los ejemplos que requieran credenciales utilizando `skip_on_cran()` y `dontrun` respectivamente.

También omite las viñetas que requieran credenciales.

3.5 Lecturas adicionales

Recursos útiles sobre seguridad:

- Encuentro de la comunidad rOpenSci sobre "Seguridad para R" (grabación, diapositivas, etc.). Ver en particular la lista de recursos;
- los proyectos relacionados con la seguridad de la unconf18;
- el artículo de `gargle` *Managing tokens securely* (Gestionando claves de forma segura)

Parte II

Revisión por pares de software de paquetes

4 Revisión por pares de software, ¿por qué? ¿qué es?

Este capítulo contiene una introducción general a nuestro sistema de revisión por pares de software para paquetes, razones para enviar un paquete, razones para ofrecerse para revisar, por qué nuestras revisiones son abiertas, y agradecimientos a quienes participan del sistema.

Nuestro sistema se ha ampliado recientemente a la revisión por pares de software estadístico.

Si utilizas nuestros estándares, listas de tareas, etc. al revisar software en otro lugar, cuenta que estos materiales proceden de rOpenSci a quienes reciben la revisión (por ejemplo, quien edita la revista, estudiantes, colegas durante una revisión interna de código), y también cuéntanoslo en nuestro foro público, o por correo electrónico si prefieres comentarlo en forma privada.

4.1 ¿Qué es la revisión por pares de software de rOpenSci?

La suite de paquetes de rOpenSci recibe aportes tanto de personal de la organización, como por miembros de la comunidad, lo que significa que se nutre de una gran diversidad de habilidades y experiencias. Pero, ¿cómo asegurar un buen nivel de calidad? Ahí es donde entra en juego la revisión por pares de software: los paquetes aportados por la comunidad se someten a un proceso de revisión transparente, constructivo, no conflictivo y abierto. El mismo se sustenta principalmente en el trabajo voluntario: desde el rol de edición asociada, gestionando el flujo de información y garantizando que los envíos progresen; personas que crean, envían y mejoran su paquete; personas que revisan paquetes (dos por envío), examinan el código y la experiencia de usuario. Este artículo del blog, escrito por miembros del equipo editorial de rOpenSci es una buena introducción a la revisión por pares de software de rOpenSci. Puedes encontrar más artículos en el blog de rOpenSci sobre el proceso y los paquetes que pasaron por el mismo en la etiqueta "software-peer-review".

Puedes reconocer los paquetes de rOpenSci que han sido revisados por pares mediante una etiqueta verde que dice "peer-reviewed" en su *README* (ej. el paquete restez) y por el icono azul en su descripción en la página de paquetes de rOpenSci. Ambos contienen links a la revisión.

En cuanto a la parte técnica, le sacamos el jugo a GitHub: cada proceso de revisión es un *issue* en el repositorio ropensci/software-review. Por ejemplo, haz click aquí para leer el hilo de revisión del paquete `ropenaq`. El proceso es una conversación activa hasta que el paquete es aceptado, con dos revisiones externas como momentos importantes. Además, utilizamos funcionalidades de GitHub

como las plantillas de *issues* (como plantilla de envío), y el etiquetado que utilizamos para seguir el progreso de los envíos (desde las revisiones iniciales hasta la aprobación).

4.2 ¿Por qué enviar tu paquete a rOpenSci?

- En primer lugar, y sobre todo, esperamos que envíes tu paquete para su revisión **porque valoras la devolución**. Nuestro objetivo es proporcionar una devolución útil a las personas que crearon el paquetes y que nuestro proceso de revisión sea abierto, no conflictivo y centrado en la mejora de la calidad del software.
- Una vez a aceptado, tu paquete seguirá recibiendo **apoyo de los miembros de rOpenSci**. Mantendrás la propiedad y el control de tu paquete, pero podemos ayudarte con los problemas de mantenimiento continuo, como los relacionados con las actualizaciones de R y las dependencias, y las políticas de CRAN.
- rOpenSci **promoverá tu paquete** a través de nuestra página web, blog y redes sociales (como Mastodon y Twitter). Los paquetes de nuestra suite también tienen un sitio web de documentación que se construye y despliega automáticamente después de cada *push*.
- Los paquete de rOpenSci **se pueden incluir** en otros repositorios como CRAN y BioConductor.
- Los paquetes de rOpenSci que son relevantes para el Journal of Open-Source Software y añaden un artículo corto de acompañamiento pueden beneficiarse, a discreción del equipo editorial de JOSS, de un proceso de revisión acelerado.
- Si escribes un ***gitbook*** **relacionado a tu paquete**, rOpenSci lo difundirá: se puede transferir su código fuente a la organización de GitHub `ropensci-books` para ser listado en books.ropensci.org.

4.3 ¿Por qué revisar paquetes para rOpenSci?

- Como en cualquier proceso de revisión por pares, esperamos que elijas revisar **para contribuir a rOpenSci y a las comunidades científicas**. Nuestra misión de ampliar el acceso a los datos científicos y promover una cultura de investigación reproducible sólo es posible gracias a los esfuerzos voluntarios de miembros de la comunidad como tú.
- La revisión es una conversación bidireccional. Al revisar los paquetes, tendrás la oportunidad de **seguir aprendiendo buenas prácticas de desarrollo de quienes crean y revisan los paquetes**.
- La naturaleza abierta de nuestro proceso de revisión te permite **establecer redes y conocer colegas y personas con quien colaborar**. Nuestra comunidad es amigable y tiene muchos miembros con ganas de ayudar y conocimiento en el desarrollo en R y en muchas otras áreas de la ciencia y la informática científica.
- Para ofrecerte para revisar paquetes, completa este breve formulario con tu información de contacto y tus áreas de conocimiento. Siempre buscamos más personas con experiencia en

el desarrollo de paquetes en general y con experiencia en los campos en los que se utilizan los paquetes.

4.4 ¿Por qué las revisiones son abiertas?

Nuestros hilos de revisión son públicos. Todas las personas involucradas (en roles de autoría, revisión y edición) conocen la identidad del resto y la comunidad en general puede ver o incluso participar en la conversación a medida que se produce. Esto proporciona un incentivo para realizar una revisión minuciosa y proporcionar revisiones constructivas y no conflictivas. Tanto quienes envían paquete como quienes los revisan dicen disfrutar y aprender más de este intercambio abierto y directo. También tiene el beneficio de construir una comunidad, ya que quienes participan tienen la oportunidad de interactuar de manera significativa con nuevas personas. Nuevas colaboraciones han nacido gracias a las ideas surgidas durante el proceso de revisión.

Somos conscientes de que los sistemas abiertos pueden tener inconvenientes. Por ejemplo, en la revisión académica tradicional la revisión por pares con doble ciego puede aumentar la representación de autoras femeninas, lo que sugiere un sesgo en las revisiones abiertas. También es posible que quienes hacen la revisión sean menos críticos en la revisión abierta. Sin embargo, proponemos que la apertura de la conversación de la revisión proporciona un control de la calidad de la revisión y del sesgo; es más difícil hacer comentarios no fundamentados o subjetivos en público sin la cobertura del anonimato. En definitiva, creemos que la comunicación directa y pública quienes desarrollaron el paquete y quienes lo revisan mejora la calidad y la imparcialidad de las revisiones.

Además, tanto quienes envían paquete y quienes los revisan tienen la posibilidad de contactar al equipo editorial por privado si tienen alguna duda o pregunta.

4.5 ¿Cómo se distingue un paquete que fue revisado?

- El *README* de tu paquete incluirá una etiqueta de revisión por pares que enlaza con el hilo de revisión.
- Tu paquete tendrá un sitio web de documentación en `docs.ropensci.org` que podrás enlazar en el archivo *DESCRIPTION*.
- El repositorio de tu paquete será transferido a la organización rOpenSci.
- Si quienes realizaron la revisión aceptan aparecer en el archivo *DESCRIPTION*, sus metadatos mencionarán la revisión.

4.6 Personas responsables de edición y revisión

4.6.1 Equipo editorial asociado

El proceso de revisión por pares del software de rOpenSci está dirigido por

- Noam Ross, EcoHealth Alliance
- Karthik Ram, rOpenSci
- Maëlle Salmon, rOpenSci
- Mark Padgham, rOpenSci
- Anna Krystalli, University of Sheffield RSE
- Melina Vidoni, RMIT University (School of Science)
- Mauro Lepore, 2 Degrees Investing Initiative
- Laura DeCicco, USGS
- Julia Gustavsen, Agroscope
- Emily Riederer, Capital One
- Adam Sparks, Department of Primary Industries and Regional Development
- Jeff Hollister, US Environmental Protection Agency

4.6.2 Equipo de revisión

Agradecemos a las siguientes personas que han ofrecido su tiempo y experiencia para revisar los paquetes enviados a rOpenSci:

Em Markowitz (NOAA) · Sam Albers · Toph Allen · Kaique dos S. Alves · Brooke Anderson · Alison Appling · Zebulun Arendsee · Taylor Arnold · Al-Ahmadgaid B. Asaad · Dean Attali · Mara Averick · Suzan Baert · James Balamuta · Vikram Baliga · David Bapst · Joëlle Barido-Sottani · Allison Barner · Cale Basaraba · John Baumgartner · Marcus Beck · Gabriel Becker · Jason Becker · Salvador Jesus Fernandez Bejarano · Dom Bennett · Ken Benoit · Aaron Berdanier · Fred Boehm · Carl Boettiger · Will Bolton · Ben Bond-Lamberty · Anne-Sophie Bonnet-Lebrun · Alison Boyer · Abby Bratt · François Briatte · Eric Brown · Julien Brun · Jenny Bryan · Lukas Burk · Lorenzo Busetto · Maria Paula Caldas · Mario Gavidia Calderón · Carlos Cámara-Menoyo · Brad Cannell · Joaquin Cavieres · Kevin Cazelles · Scott Chamberlain · Cathy Chamberlin · Jennifer Chang · Pierre Chausse · Jorge Cimentada · Nicholas Clark · Chase Clark · Jon Clayden · Dena Jane Clink · Will Cornwell · Nic Crane · Enrico Crema · Verónica Cruz-Alonso · Ildiko Czeller · Tad Dallas · Kauê de Sousa · Christophe Dervieux · Amanda Dobbyn · Jasmine Dumas · Dewey Dunnington · Remko Duursma · Mark Edmondson · Paul Egeler · Evan Eskew · Harry Eslick · Denisse Fierro-Arcos · Alexander Fischer · Kim Fitter · Robert M Flight · Sydney Foks · Stephen Formel · Zachary Stephen Longiaru Foster · Auriel Fournier · Kaija Gahm · Zach Gajewski · Carl Ganz · Duncan Garmonsway · Jan Laurens Geffert · Sharla Gelfand · Monica Gerber · Duncan Gillespie · David Gohel · A. Cagri gokcek · Guadalupe Gonzalez · Rohit Goswami · Laura Graham · Charles Gray · Matthias Grenié · Corinna Gries · Hugo Gruson · Ernest Guevarra · W Kyle Hamilton · Ivan Hanigan · Jeffrey Hanson · Liz Hare · Jon Harmon · Rayna Harris · Ted Hart ·

Nujcharee Haswell · Verena Haunschmid · Stephanie Hazlitt · Andrew Heiss · Max Held · Anna Hepworth · Bea Hernandez · Jim Hester · Peter Hickey · Roel Hogervorst · Kelly Hondula · Allison Horst · Sean Hughes · James Hunter · Brandon Hurr · Ger Inberg · Christopher Jackson · Najko Jahn · Tamora D James · Veronica Jimenez-Jacinto · Mike Johnson · Will Jones · Max Joseph · Megha Joshi · Krunoslav Juraic · Soumya Kalra · Zhian N. Kamvar · Michael Kane · Andee Kaplan · Tinula Kariyawasam · Hazel Kavılı · Jonathan Keane · Christopher T. Kenny · Os Keyes · Eunseop Kim · Aaron A. King · Michael Koontz · Bianca Kramer · Will Landau · Sam Lapp · Erin LeDell · Thomas Leeper · Sam Levin · Lisa Levinson · Stephanie Locke · Marion Louveaux · Robin Lovelace · Julia Stewart Lowndes · Tim Lucas · Muralidhar, M.A. · Andrew MacDonald · Jesse Maegan · Mike Mahoney · Tristan Mahr · Yohann Mansiaux · Paula Andrea Martinez · Anthony Martinez · Joao Martins · Ben Marwick · Claire Mason · Miles McBain · Lucy D'Agostino McGowan · Amelia McNamara · Elaine McVey · Bryce Mecum · Nolwenn Le Meur · François Michonneau · Mario Miguel · Helen Miller · Beatriz Milz · Jessica Minnier · Priscilla Minotti · Nichole Monhait · Kelsey Montgomery · Ronny A. Hernández Mora · Paula Moraga · Natalia Morandeira · George Moroz · Ross Mounce · Athanasia Monika Mowinckel · Lincoln Mullen · Matt Mulvahill · Maria Victoria Munafó · David Neuzerling · Dillon Niederhut · Joel Nitta · Rory Nolan · Kari Norman · Jakub Nowosad · Matt Nunes · Daniel Nüst · Lauren O'Brien · Joseph O'Brien · Paul Oldham · Samantha Oliver · Dan Olner · Jeroen Ooms · Victor Ordu · Luis Osorio · Philipp Ottolinger · Mark Padgham · Marina Papadopoulou · Edzer Pebesma · Thomas Lin Pedersen · Antonio J. Pérez-Luque · Marcelo S. Perlin · Rafael Pilliard-Hellwig · Rodrigo Neto Pires · Lindsay Platt · Nicholas Potter · Joanne Potts · Josep Pueyo-Ros · Etienne Racine · Manuel Ramon · Nistara Randhawa · David Ranzolin · Quentin Read · Neal Richardson · tyler rinker · Emily Robinson · David Robinson · Alec Robitaille · Francisco Rodriguez-Sanchez · Sam Rogers · Julia Romanowska · Xavier Rotllan-Puig · Bob Rudis · Edgar Ruiz · Kent Russel · Michael Sachs · Sheila M. Saia · Chitra M Saraswati · Alicia Schep · Klaus Schliep · Clemens Schmid · Patrick Schratz · Collin Schwantes · Marco Sciaini · Eric Scott · Heidi Seibold · David Selby · Julia Silge · Margaret Siple · Peter Slaughter · Mike Smith · Tuija Sonkkila · Øystein Sørensen · Jemma Stachelek · Aymeric Stamm · Christine Stawitz · Irene Steves · Kelly Street · Matt Strimas-Mackey · Alex Stringer · Michael Sumner · Chung-Kai Sun · Sarah Supp · Emi Tanaka · Jason Taylor · Filipe Teixeira · Andy Teucher · Jennifer Thompson · Joe Thorley · Nicholas Tierney · Tiffany Timbers · Tan Tran · Tim Trice · Utku Turk · Zoë Turner · Kyle Ueyama · Ted Underwood · Adithi R. Upadhya · Kevin Ushey · Josef Uyeda · Frans van Dunné · Mauricio Vargas · Remi Vergnon · Jake Wagner · Ben Ward · Elin Waring · Rachel Warnock · Leah Wasser · David Watkins · Lukas Weber · Marc Weber · Karissa Whiting · Stefan Widgren · Anna Willoughby · Saras Windecker · Luke Winslow · David Winter · Sebastian Wójcik · Witold Wolski · Kara Woo · Marvin N. Wright · Jacob Wujciak-Jens · Bruna Wundervald · Lauren Yamane · Emily Zabor · Taras Zakharko · Sherry Zhang · Hao Zhu · Chava Zibman · Naupaka Zimmerman · Jake Zwart · Felipe · santikka · kasselhingee · Bri · Flury · Vincent · eholmes · Pachá · Rich · Claudia · Jasmine · Zack · Lluís · becarioprecario · gaurav

También agradecemos a las siguientes personas que han tomado el rol de edición de forma invitada:

Ana Laura Diedrichs · Francisco Rodriguez-Sanchez · Hao Zhu

5 Políticas de la Revisión por Pares de Software

Este capítulo contiene las políticas de la Revisión por Pares de Software de rOpenSci.

En particular, encontrarás nuestras políticas sobre a la revisión por pares de software en sí misma: el proceso de presentación de revisiones (incluyendo nuestras políticas sobre conflictos de intereses) y los objetivos y alcance del sistema de revisión por pares de software. Este capítulo también presenta nuestras políticas sobre propiedad y mantenimiento de los paquetes.

Por último, pero no menos importante, encontrarás el código de conducta de la Revisión por Pares de Software de rOpenSci.

5.1 Proceso de revisión

- Para que un paquete sea considerado para ingresar a rOpenSci, quienes crearon el paquete deben iniciar una solicitud en el repositorio ropensci/software-review.
- Los paquetes se revisan en su calidad, adecuación, documentación y claridad. El proceso de revisión es bastante similar al de la revisión de un artículo científico (véase nuestra guía de empaquetado y guía de revisión para más detalles). A diferencia de la revisión de un artículo científico, este proceso será una conversación continua.
- Una vez resueltas todas las cuestiones y preguntas importantes que puedan solucionarse con un esfuerzo razonable, la persona responsable de la edición tomará una decisión (aceptar, esperar o rechazar). Los rechazos suelen hacerse pronto (antes de que comience el proceso de revisión; ver la sección de objetivos y alcance), pero en raras ocasiones un paquete puede no ser aceptado después de la revisión. En última instancia, es decisión de quien hace la edición rechazar o no el paquete en función de cómo se aborden las revisiones.
- La comunicación entre quienes enviaron el paquete, quienes lo revisan y editan tendrá lugar principalmente en GitHub, aunque puedes optar por ponerte en contacto con quien hace la edición por correo electrónico o por Slack para algunas cuestiones. Cuando envíes un paquete, asegúrate de que tu configuración de notificaciones de GitHub sea correcta para que no te pierdas un comentario.
- Quien envía el paquete puede elegir que su envío se ponga en espera (en cuyo caso se le aplica la etiqueta *holding*). El estado de espera se revisará cada 3 meses, y después de un año el *issue* se cerrará.

- Si quien envió el paquete no ha solicitado una etiqueta de espera, pero no responde a las consultas, podremos cerrar el *issue* en el plazo de un mes desde el último intento de contacto. Este intento incluirá un comentario etiquetando a la persona, pero también un correo electrónico utilizando la dirección de correo electrónico que aparece en el archivo *DESCRIPTION* del paquete. Este es uno de los raros casos en los que quien hace la edición intentará ponerse en contacto con el autor por correo electrónico.
- Para volver a enviar un paquete cuyo envío fue cerrado hay que iniciar un nuevo envío. Si el paquete sigue siendo relevante, habrá que responder a las revisiones iniciales antes de que la persona encargada de la edición empiece a buscar nuevas personas para la revisarlo.

5.1.1 Publicar en otros lugares

- Te recomendamos fuertemente que envíes tu paquete para su revisión *antes de* publicarlo en CRAN o enviar un artículo de software que describa el paquete a una revista. Las devoluciones de las revisiones pueden dar pie a importantes mejoras y actualizaciones de tu paquete, incluso cambiar el nombre o modificaciones incompatibles con versiones previas. Consideramos que la publicación previa en CRAN o en otros lugares no es razón suficiente para rechazar las recomendaciones surgidas del proceso de revisión.
- No envíes tu paquete para su revisión mientras éste o un artículo científico asociado también esté siendo revisado en otro lugar, ya que esto puede dar lugar a recomendaciones incompatibles.

5.1.2 Conflicto de intereses de quienes revisan o editan

Los siguientes criterios pretenden ser una guía de lo que constituye un conflicto de intereses para quien hace la edición o revisión. Existe un conflicto de intereses si:

- quien potencialmente revisará o editará pertenece a la institución o a una componente institucional (por ejemplo, un departamento) de quienes tienen un papel importante en el paquete;
- quien potencialmente revisará o editará ha colaborado o ha tenido cualquier otra relación profesional con cualquier persona que tenga un papel importante en el paquete en los últimos tres años;
- quien potencialmente revisará o editará es miembro del consejo asesor del proyecto que se está revisando;
- quien potencialmente revisará o editará recibiría un beneficio económico directo o indirecto si se acepta el paquete;
- quien potencialmente revisará o editará ha contribuido significativamente a un proyecto que se considera competencia;

- también hay un conflicto de interés de por vida para familiares, personas con relaciones comerciales y estudiantes o personas que asesoran o mentorean.

En el caso de que ninguno de los miembros del equipo editorial pueda realizar la edición, se invitará a una persona externa para que la realice.

5.2 Objetivos y alcance

rOpenSci tiene como objetivo apoyar paquetes que permiten la investigación reproducible y la gestión del ciclo de vida de los datos para quienes hacen ciencia. Los paquetes enviados a rOpenSci deben encajar en una o más de las categorías indicadas a continuación. El software estadístico también puede ser enviado para su revisión por pares, para lo cual tenemos un conjunto de directrices y normas. Mientras que el resto de este capítulo se aplica a ambos tipos de software, las categorías que aparecen a continuación son para el software general, y no para el estadístico. Si te queda claro si tu paquete encaja en una de las categorías generales o estadísticas, abre un *issue* como consulta previa a la presentación (**Ejemplos**).

Como se trata de un documento vivo, estas categorías pueden cambiar con el tiempo y no todos los paquetes incluidos anteriormente estarían en el ámbito de aplicación en la actualidad. Por ejemplo, los paquetes de visualización de datos ya no están incluidos. Aunque nos esforzamos por ser coherentes, evaluamos los paquetes caso por caso y podemos hacer excepciones.

Ten en cuenta que no todos los proyectos y paquetes de rOpenSci están incluidos en el ámbito de aplicación o pasan por una revisión por pares.\

Los proyectos desarrollados por el personal o en conferencias pueden ser experimentales, exploratorios, abordar las prioridades de la infraestructura básica y, por tanto, no entrar en estas categorías. Busca la etiqueta de revisión por pares (ver más abajo) para identificar los paquetes revisados por pares en el repositorio de rOpenSci.

Figura 5.1: Ejemplo de insignia verde de revisión por pares

5.2.1 Categorías de paquetes

- **obtención de datos**: Paquetes para acceder y descargar datos de fuentes en línea con usos científicos. Nuestra definición de “usos científicos” es amplia. Incluye servicios de almacenamiento de datos, revistas y otros servidores remotos, ya que existe una gran variedad de fuentes de datos que pueden ser de interés para quienes hacen ciencia. Sin embargo, los paquetes de obtención de datos deben centrarse en las *fuentes* o *temas* de los datos y no en

servicios. Por ejemplo, un cliente general para el almacenamiento de datos de *Amazon Web Services* no estaría en nuestro ámbito. (Ejemplos: **rotl**, **gutenbergr**)

- **extracción de datos**: Paquetes que ayudan a extraer datos de fuentes no estructuradas, como texto, imágenes y PDFs, así como a leer datos en formatos científicos y salidas de equipamiento científico. Las bibliotecas estadísticas o de *machine learning* para el modelado o la predicción no suelen incluirse en esta categoría, como tampoco los analizadores de código. Los modelos entrenados que actúan como utilidades (por ejemplo, para el reconocimiento óptico de caracteres) podrían calificar. (Ejemplos: **tabulador** para extraer tablas de documentos PDF, **genbankr** para segmentar archivos de GenBank, **treeio** para la lectura filogenética en archivos de árboles filogenéticos, **lightr** para segmentar archivos de instrumentos espectroscópicos)
- **manipulación de datos**: Paquetes para procesar datos obtenidos de los formatos anteriores. Esta área no incluye las herramientas de manipulación de datos en general, como **reshape2** o **tidyr** o herramientas para extraer datos del propio código de R. Más bien, se centra en herramientas para manipular datos en formatos científicos específicos generados a partir de flujos de trabajo científicos o exportados desde instrumentos científicos. (Ejemplos: **plateR** para leer datos estructurados como mapas de placas de instrumentos científicos, o **phonfieldwork** para procesar archivos de audio anotados para la investigación fonética)
- **depósito de datos**: Paquetes que apoyan el depósito de datos en repositorios de investigación, incluyendo el proceso de dar formato a los datos y la generación de metadatos. (Ejemplo: **EML**)
- **validación y comprobación de datos**: Herramientas que permiten la validación y comprobación automatizada de la calidad e integridad de los datos como parte de flujos de trabajo científico. (Ejemplo: **assertr**)
- **automatización del flujo de trabajo**: Herramientas que automatizan y encadenan flujos de trabajo, como los sistemas de construcción y las herramientas para gestionar la integración continua. No incluye las herramientas generales para la programación literiaria (por ejemplo, las extensiones de R markdown que no están en los temas anteriores). (Ejemplo: **drake**)
- **control de versiones**: Herramientas que facilitan el uso del control de versiones en los flujos de trabajo científico. Ten en cuenta que esto no incluye todas las herramientas que interactúan con los servicios de control de versiones en línea (por ejemplo, GitHub), a menos que encajen en otra categoría. (Ejemplo: **git2rdata**)
- **gestión de citas y bibliometría**: Herramientas que facilitan la gestión de las referencias, como por ejemplo para escribir publicaciones científicas, crear CVs o atribuir de otra manera las contribuciones científicas, o acceder, manipular o trabajar de otra manera con datos bibliométricos. (Ejemplo: **RefManageR**)
- **capa de interfaz de software científico**: Paquetes que permiten correr aplicaciones usadas en la investigación científica desde R. Estos programas deben ser específicos de los campos de investigación, no utilidades informáticas generales. Las capas deben ser no triviales, en el

sentido de que debe haber un valor añadido significativo por encima del uso de `system()` o de vincular el programa, ya sea en darle formato a las entradas entradas y salidas, en el manejo de datos, etc. La mejora del proceso de instalación, o la ampliación de la compatibilidad a más plataformas, puede constituir un valor añadido si la instalación es compleja. Esto no incluye las capas de interfaz de otros paquetes de R o bibliotecas de C/C++ que puedan incluirse en los paquetes de R. Tampoco se incluyen los paquetes que son clientes de API web, que deben pertenecer a una de las otras categorías. Animamos encarecidamente crear capas de interfaz a utilidades de código abierto y de licencia abierta; las excepciones se evaluarán caso por caso, teniendo en cuenta si existen opciones de código abierto. (Ejemplos: **babette**, **nlrx**)

- **herramientas de reproducibilidad de campo y laboratorio**: Paquetes que mejoran la reproducibilidad de los flujos de trabajo del mundo real mediante la estandarización y la automatización de los protocolos de campo y de laboratorio. Por ejemplo, el seguimiento y el etiquetado de las muestras, la generación de formularios y hojas de datos, la interconexión con los equipos de laboratorio o los sistemas de información, y la ejecución de diseños experimentales. (Ejemplo: **baRcodeR**)
- **enlaces a software de bases de datos**: Enlaces y capas de interfaz para las API genéricas de bases de datos (Ejemplo: **rrlite**)

Además, tenemos algunos *temas especializados* con un alcance algo más amplio.

- **datos geoespaciales**: Aceptamos paquetes centrados en el acceso, manipulación y conversión entre formatos de datos geoespaciales. (Ejemplos: **osmplotr**, **tidync**).
- **traducción**: Como parte de nuestro trabajo en publicación multilingüe tenemos especial interés en paquetes que faciliten la traducción y publicación de recursos científicos y de programación a múltiples lenguages (humanos) para que sean accesibles a públicos más amplios y diversos. Podría tratarse de interfaces para programas de traducción automática, infraestructura para gestionar la documentación en varios lenguages o programas que accedan a recursos lingüísticos especializados. Se trata de un ámbito nuevo y experimental, por lo que te rogamos que abras un issue para preguntar si el software está dentro del alcance de rOpenSci si te interesa presentar un paquete en esta categoría.

5.2.2 Otras consideraciones sobre el ámbito de aplicación

Los paquetes deben ser *generales* en el sentido de que deben resolver un problema de la forma más amplia posible, manteniendo una interfaz de usuario y una base de código coherentes. Por ejemplo, si varias fuentes de datos utilizan una API idéntica, preferimos un paquete que proporcione acceso a todas las fuentes de datos, en lugar de una sola.

Los paquetes que incluyan herramientas interactivas para facilitar los flujos de trabajo científico (por ejemplo, las shiny apps) deben tener un mecanismo para que el flujo de trabajo interactivo sea reproducible, como la generación de código o una API que permita la creación de *scripts*.

Para los paquetes que no están en el ámbito de rOpenSci, animamos a presentarlos en CRAN, BioConductor, así como en otras iniciativas de desarrollo de paquetes de R (por ejemplo cloudyr), y a las revistas de software como JOSS, JSS o la revista R, dependiendo del alcance actual de esas revistas.

Ten en cuenta que los paquetes desarrollados internamente por rOpenSci, a través de nuestros eventos o mediante colaboraciones, no están todos en el ámbito de nuestro proceso de revisión por pares de software.

5.2.3 Superposición de paquetes

rOpenSci fomenta la competencia entre paquetes, la bifurcación y la reimplementación, dado que éstas mejoran las opciones de quienes los usan en general. Sin embargo, como queremos que los paquetes del universo de rOpenSci sean nuestras principales recomendaciones para las tareas que realizan, pretendemos evitar la duplicación de la funcionalidad de los paquetes de R existentes en cualquier repo sin mejoras significativas. Un paquete que replica la funcionalidad de un paquete existente puede ser considerado para su inclusión en rOpenSci si mejora significativamente las alternativas en cualquier repositorio (RO, CRAN, BioC) al ser

- más abierto en cuanto a licencias o prácticas de desarrollo;
- más amplio en funcionalidad (por ejemplo, proporcionando acceso a más conjuntos de datos, proporcionando un mayor conjunto de funciones), pero no sólo duplicando paquetes adicionales;
- mejor en cuanto a usabilidad y rendimiento;
- activamente mantenido si las alternativas tienen poco o ningún mantenimiento activo.

Estos factores deben considerarse *en su conjunto* para determinar si el paquete es una mejora significativa. Un nuevo paquete no cumpliría esta norma sólo por seguir nuestras directrices sobre paquetes mientras otros no lo hacen, a menos que esto suponga una diferencia significativa en las áreas mencionadas.

Recomendamos que los paquetes destaquen en su *README* y/o en sus viñetas las diferencias con respecto a los paquetes que se solapan y las mejoras que introducen.

Animamos a quienes desarrollaron paquetes que no son aceptados debido al solapamiento a que sigan considerando su envío a otros repositorios o revistas.

5.3 Propiedad y mantenimiento de los paquetes

5.3.1 Rol del equipo de rOpenSci

Quienes crearon los paquetes mantienen esencialmente la misma propiedad que tenían antes de que su paquete se uniera al conjunto paquetes de rOpenSci. Estas personas seguirán manteniendo y desarrollando su software luego de su aceptación en rOpenSci. A menos que se les añada explícitamente con rol de colaboración, el equipo de rOpenSci no interferirá mucho en las actividades cotidianas. Sin embargo, el equipo puede intervenir con correcciones de errores críticos, o abordar cuestiones urgentes si quienes son responsables de los paquetes no responden de manera oportuna (ver la sección sobre la capacidad de respuesta de quienes mantienen los paquetes).

5.3.2 Capacidad de respuesta de quienes mantienen los paquetes

Si quienes matienen los paquetes no responden a tiempo a las solicitudes de corrección de de CRAN o de rOpenSci, enviaremos recordatorios varias veces, pero después de 3 meses (o un plazo más corto, dependiendo de lo crítica que sea la corrección) haremos los cambios.

Lo anterior es un poco impreciso, por lo que a continuación se presentan algunos ejemplos a considerar.

- Ejemplos en los que querríamos actuar con rapidez:
  - El paquete `foo` es importado por uno o más paquetes en CRAN, y `foo` tiene problemas, y por tanto generará problemas en los paquetes que dependan de `foo`.
  - El paquete `bar` puede no ser usado por otros paquetes que se encuentran en CRAN, pero es muy utilizado, por lo que arreglar rápidamente los problemas es muy importante.
- Ejemplos en los que podemos esperar más:
  - El paquete `hello` no está en CRAN, o está en CRAN, pero no tiene dependencias inversas (no hay paquetes que dependan de de `hello`).
  - El paquete `world` necesita algunas correcciones. La persona responsable ha respondido, pero simplemente está muy ocupada con un nuevo trabajo, u otra razón, y atenderá el problema pronto.

Instamos a quienes mantienen paquetes a asegurarse de que reciben las notificaciones de GitHub, y de que los correos electrónicos del equipo de rOpenSci y de CRAN no van a su bandeja de correo no deseado. Invitaremos al Slack de rOpenSci a quienes hayan desarrollado paquetes que son incorporados a rOpenSci para charlar con el equipo y la comunidad de rOpenSci en general. Cualquier persona puede sumarse a la conversación con la comunidad rOpenSci en el foro de discusión de rOpenSci.

Si quienes desarrollaron un paquete abandonan su mantenimiento y se trata de un paquete utilizado activamente, consideraremos la posibilidad de solicitar a CRAN que transfiera el estatus de mantenedor del paquete a rOpenSci.

5.3.3 Compromiso de calidad

rOpenSci se esfuerza por desarrollar y promover software de investigación de alta calidad. Para asegurarnos de que tu software cumple nuestros criterios, revisamos todos los envíos como parte del proceso de revisión por pares del software, e incluso después de la aceptación seguiremos interviniendo con mejoras y correcciones de errores.

A pesar de nuestros esfuerzos por apoyar el software contribuido, los errores son responsabilidad de quien mantiene el paquete. El software con errores y sin mantenimiento puede ser eliminado de nuestra suite de paquetes en cualquier momento.

5.3.4 Eliminación de paquetes

En el improbable caso de que quien colabora con de un paquete solicite que su paquete sea retirado de la suite, nos reservamos el derecho de mantener una versión del paquete en nuestra suite con fines de archivo.

5.4 Ética, privacidad de datos e investigación con sujetos humanos

Los paquetes de rOpenSci y otras herramientas se utilizan para diversos fines, pero nuestro foco está en las herramientas para la investigación. Esperamos que las herramientas permitan un uso ético por parte de quienes hacen investigación, quienes tienen la obligación de seguir códigos éticos como la Declaración de Helsinki y el Informe Belmont. Quienes investigan son responsables del uso que hacen del software, pero quienes desarrollan el software deben considerar el uso ético de sus productos y deben adherir a los códigos éticos para profesionales de la informática, como los expresados por el IEEE y ACM. Quienes colaboran en rOpenSci a menudo desempeñan el papel de investigación y desarrollo de software.

Pedimos que quienes desarrollan software se pongan en el papel de quienes investigan y consideren los requisitos de un flujo de trabajo ético utilizando el software. Dada la variación y la velocidad de los cambios en los enfoques éticos para los análisis basados en Internet, es necesario evaluar cada caso en lugar de elaborar recetas. La Guía de Ética de la Asociación de Investigadores de Internet proporciona un marco sólido y animamos tanto a quienes mantienen paquetes como a quienes los revisan y editan, a utilizarlo a la hora de evaluar su trabajo. En general, la adhesión a los requisitos legales o reglamentarios mínimos puede no ser suficiente, aunque éstos (p. ej, GDPR), pueden ser

relevantes. Quienes crean los paquetes deben dirigir a quienes los usan a los recursos pertinentes para el uso ético del software.

Algunos paquetes pueden ser sujetos a un mayor escrutinio debido a la naturaleza de los datos que manejan. En estos casos, quienes hacen la edición pueden exigir funcionalidad adicional (o reducida) y una sólida documentación, valores por defecto y advertencias para dirigir a quienes usan el paquete a las prácticas éticas pertinentes. Los siguientes temas pueden merecer un mayor escrutinio:

- ***Poblaciones vulnerables***: Quienes crean paquetes y flujos de trabajo que tratan con información relacionada con poblaciones vulnerables tienen la responsabilidad de protegerlas de posibles daños.
- ***Datos personales o sensibles***: La divulgación de datos identificables o sensibles es potencialmente perjudicial. Esto incluye los datos "razonablemente identificables", que una persona motivada podría rastrear hasta la persona propietaria o creadora, incluso si los datos son anónimos. Esto incluye tanto los casos en los que los datos identificadores (por ejemplo, nombre, fecha de nacimiento) están disponibles como parte de los datos, y también si los seudónimos o nombres de usuario están vinculados a mensajes de texto completo, a través de los cuales se puede puede vincular a las personas individuales mediante referencias cruzadas con otros conjuntos de datos.

Aunque la mejor respuesta a los problemas éticos dependerá del contexto, se deben seguir estas directrices generales cuando se presenten los desafíos anteriores:

- Los paquetes deben adherirse a las condiciones de uso de la fuente de datos, expresadas en los términos y condiciones del sitio web, archivos "robots.txt", políticas de privacidad y otras otras restricciones relevantes, y enlazarlas de forma destacada en la documentación del paquete. Los paquetes deben proporcionar o documentar la funcionalidad para cumplir restricciones (por ejemplo, el *scrapeado* sólo de los servicios permitidos, el uso de limitación de velocidad adecuada en el código, los ejemplos o las viñetas). Ten en cuenta que aunque los Términos y Condiciones, las políticas de privacidad, etc., pueden no ofrecer límites suficientes para un uso ético, pueden proporcionar límites mínimos.
- Una herramienta clave para abordar los riesgos que plantea el estudio de poblaciones vulnerables o utilizar datos de identificación personal es ***el consentimiento informado***. Quienes crean los paquetes deben permitir la obtención del consentimiento informado cuando sea pertinente. Esto puede incluir proveer enlaces al método preferido por la fuente de datos para adquirir el consentimiento, información de contacto de quienes proveen los datos (por ejemplo, quienes moderan un foro), documentación de los protocolos de consentimiento informado, u obtener la aprobación previa para los usos generales de un paquete.

  Ten en cuenta que el consentimiento no se concede implícitamente sólo porque los datos sean accesibles. Los datos accesibles no son necesariamente públicos, ya que diferentes personas y contextos tienen diferentes expectativas normativas de privacidad (véase el trabajo de *Social Data Lab*).

- Los paquetes que acceden a información personal identificable deben tener especial cuidado en seguir las buenas prácticas de seguridad (por ejemplo, el uso exclusivo de protocolos de Internet seguros, mecanismos fuertes para almacenar credenciales, etc.).
- Los paquetes que acceden o manejan datos personales identificables o sensibles deben habilitar, documentar y demostrar los flujos de trabajo para la desidentificación, el almacenamiento seguro, y otras buenas prácticas para minimizar el riesgo de daños.

A medida que las normas de privacidad de los datos y la investigación siguen evolucionando, agradeceremos los aportes de quienes desarrollan paquetes sobre las consideraciones específicas de su software y la documentación complementaria, como la aprobación de los comités de revisión ética de las universidades. Estos pueden adjuntarse a los *issues* de los envíos de paquetes o a las consultas previas al envío, o transmitirse directamente a quienes se encargan de la edición si es necesario. En general, sugerencias pueden enviarse como *issue* en el repositorio de este libro.

5.4.1 Recursos

Los siguientes recursos pueden ser útiles para quienes investigan, desarrollan paquetes, hacen la edición o revisión a la hora de abordar cuestiones éticas relacionadas con la privacidad y el software de investigación.

- El sitio web de la Declaración de Helsinki y el Informe Belmont proporcionan principios fundamentales para la práctica ética de quienes hacen investigación.
- Varias organizaciones ofrecen orientación sobre cómo trasladar estos principios en el contexto de la investigación en Internet. Entre ellas se encuentra la Guía de Ética de la Asociación de Investigadores de Internety la Guía de la NESH sobre Eica de la Investigación de Internet, y Directrices éticas de BPS para la investigación a través de Internet.
- Anabo et al (2019) proporcionan una visión general útil de estas guías.
- El Laboratorio de Redes Sociales ofrece una revisión general de alto nivel con datos sobre las expectativas de privacidad y uso en los foros sociales.
- Bechmann A., Kim J.Y. (2019) Big Data: A Focus on Social Media Research Dilemas. En: Iphofen R. (eds) Handbook of Research Ethics and Scientific Integrity. https://doi.org/10.1007/978-3-319-76040-7_18-1
- Chu, K.-H., Colditz, J., Sidani, J., Zimmer, M., y Primack, B. (2021). Re-evaluating standards of human subjects protection for sensitive health data in social media networks. Social Networks, 67, 41-46. https://dx.doi.org/10.1016/j.socnet.2019.10.010
- Lomborg, S., y Bechmann, A. (2014). Using APIs for Data Collection on Social Media. The Information Society, 30(4), 256–265. https://dx.doi.org/10.1080/01972243.2014.915276
- Flick, C. (2016). Informed consent and the Facebook emotional manipulation study. Research Ethics, 12(1), 14–28. https://doi.org/10.1177/1747016115599568
- Sugiura, L., Wiles, R., y Pope, C. (2017). Ethical challenges in online research: Public/private perceptions. Research Ethics, 13(3–4), 184–199. https://doi.org/10.1177/1747016116650720

- Taylor, J., y Pagliari, C. (2018).Mining social media data: How are research sponsors and researchers addressing the ethical challenges? Research Ethics, 14(2), 1–39. https://doi.org/10.1177/1747016117738559
- Zimmer, M. (2010). “But the data is already public”: on the ethics of research in Facebook. Ethics and Information Technology, 12(4), 313–325. https://dx.doi.org/10.1007/s10676-010-9227-5

5.5 Código de conducta

La comunidad de rOpenSci es nuestro mejor recurso. Tanto si colaboras habitualmente como si recién llegas, nos esforzamos para que éste lugar sea seguro para ti y te apoyamos en lo que necesites. Tenemos un Código de Conducta que se aplica a todas las personas que participan en la comunidad de rOpenSci, incluidos el equipo y la dirección de rOpenSci y a todos las formas de interacción en línea o en persona. El Código de Conducta se mantiene en el sitio web de rOpenSci.

6 Guía para quienes crean paquetes

Esta guía condensa el proceso de revisión por pares desde el punto de vista de quienes crean paquetes.

6.1 Planificar un envío (o una consulta previa al envío)

- ¿Esperas mantener tu paquete durante al menos 2 años, o poder encontrar a otra persona para mantenerlo?
- Consulta nuestras políticas para evaluar si tu paquete cumple los criterios para ser incluido en nuestro conjunto de paquetes y no se superpone con otros.
  - Si no sabes si un paquete cumple con nuestros criterios, no dudes en abrir un issue para consultarlo.
  - Ejemplo de respuesta sobre solapamiento. Considera también añadir información sobre paquetes similares a tu documentación de paquetes.
- Por favor, considera cuál es el mejor momento en el desarrollo de tu paquete para presentarlo. Tu paquete debe estar lo suficientemente maduro como para que quienes lo revisen puedan evaluar todos los aspectos esenciales, pero ten en cuenta que la revisión puede resultar en cambios importantes.
- Te recomendamos fuertemente que envíes tu paquete para su revisión *antes de* publicarlo en CRAN o enviar un artículo de software que describa el paquete a una revista. Las devoluciones de las revisiones pueden dar pie a importantes mejoras y actualizaciones de tu paquete, incluso cambiar el nombre o modificaciones incompatibles con versiones previas.
- No envíes tu paquete para su revisión mientras éste o un artículo científico asociado también esté siendo revisado en otro lugar, ya que esto puede dar lugar a recomendaciones incompatibles.
- Ten en cuenta también el tiempo y el esfuerzo necesarios para responder a las revisiones: piensa en tu disponibilidad o en la de quienes colaboran con tu paquete en las semanas y meses siguientes al envío. Ten en cuenta que las revisiones son realizadas por personas voluntarias, y te pedimos que respetes su tiempo y esfuerzo respondiendo puntual y respetuosamente.
- Si utilizas etiquetas de repostatus.org (que recomendamos), envía tu paquete cuando esté listo para obtener la etiqueta *Active* en vez de *WIP*. Si utilizas etiquetas de ciclo de vida, el envío debe producirse cuando el paquete esté al menos en el estado *Maturing*.

- Para cualquier envío o consulta previa al envío, el *README* de tu paquete debería proporcionar suficiente información sobre el mismo (objetivos, uso, paquetes similares) para que quienes revisan el paquete puedan evaluar su alcance sin tener que instalarlo. Mejor aún, crea un sitio web de pkgdown para que puedan evaluar las funcionalidads detalladamente en línea.
- En la fase de envío, todas las funciones principales deben ser lo suficientemente estables como para estar completamente documentadas y testeadas. El *README* tiene que dar una buena impresión de tu paquete.
- El archivo *README* debe esforzarse por explicar las funcionalidades y los objetivos de tu paquete asumiendo poco o ningún conocimiento del dominio. Además, debe aclarar todos los temas técnicos, incluidas las referencias a otros software.
- Tu paquete seguirá evolucionando después de la revisión, el capítulo sobre *Evolución de paquetes* proporciona orientación sobre el tema.

6.2 Preparación para el envío

- Lee y sigue nuestra guía de estilo para paquetes y nuestra guía para la revisión para asegurarte de que tu paquete cumple nuestros criterios de estilo y calidad.
- No dudes en hacer cualquier pregunta sobre el proceso en general, o sobre tu paquete en particular en nuestro foro de discusión.
- Todos los envíos son revisados automáticamente por nuestro propio sistema para garantizar que los paquetes sigan nuestras directrices. Se espera que hayas corrido la función `pkgcheck` localmente para confirmar que el paquete está listo para ser enviado. Otra forma aún más fácil de asegurarse de que un paquete está listo para su envío es utilizar la acción de GitHub `pkgcheck` para ejecutar `pkgcheck` como una Acción de GitHub, como se describe en esta publicación en nuestro blog.
- Si tu paquete requiere dependencias inusuales del sistema (ver *Guía de Empaquetado* para que nuestra Acción de GitHub pase, por favor envíe un *pull request* añadiéndolas a nuestro *Dockerfile* base.
- Si hay algún test de `pkgcheck` que tu paquete no pueda aprobar, explica los motivos en la plantilla de envío.
- Si crees que tu paquete es relevante para el Journal of Open Source Software (JOSS), no lo sometas a consideración de JOSS hasta que haya finalizado el proceso de revisión de rOpenSci: si el equipo de edición de JOSS considera que tu paquete está dentro de su ámbito de aplicación, sólo se revisará el breve artículo que lo acompaña (pero no el software que habrá sido revisado por rOpenSci en ese momento). No todos los paquetes de rOpenSci pueden aplicar a JOSS.

6.3 El proceso de envío

- Para presentar tu paquete a revisión tienes que abrir un nuevo *issue* en el repositorio de revisión de software y completar la plantilla.
- La plantilla comienza con una sección que incluye varias variables de estilo HTML (`<!---variable--->`). Éstas son utilizadas por nuestro bot (`ropensci-review-bot`) y no deben modificarse. Los valores de las variables deben completarse ente los puntos de inicio y fin, así:

```
<!---variable--->insertar valor aquí<!---end-variable>
```

- La comunicación entre quines envian el paquete, quienes lo revisen y quienes hagan la edición se dará principalmente en GitHub, para que el *issue* de revisión sirva de registro completo de la misma. Puedes contactarte con quien se encarga de la edición por correo electrónico o Slack si es necesario realizar una consulta privada (por ejemplo, preguntar cómo responder a una pregunta de quien está haciendo la revisión). No te pongas en contacto con quienes revisan tu paquete fuera del *issue* sin antes preguntarles, dentro del mismo, si están de acuerdo con ello.
- *Cuando envíes un paquete, por favor asegúrate de tener configuradas las notificaciones de GitHub para que no te pierdas ningún comentario.*
- Los paquetes se checkean automáticamente al ser enviados con nuestro sistema `pkgcheck`, el cual confirmará si está listo para ser revisado o no.
- Los paquetes enviados pueden estar en la rama *main/default*, o en cualquier otra rama no predeterminada. En este último caso, es recomendable, aunque no obligatorio, enviar el paquete a través de una rama dedicada llamada `ropensci-software-review`.

6.4 El proceso de revisión

- Una persona realizará la edición y revisará tu envío en un plazo de 5 días laborables y te responderá con los siguientes pasos a seguir. Puede asignar el paquete a personas para que lo revisen, solicitar que el paquete se actualice para cumplir los criterios mínimos antes de la revisión, o rechazar el paquete porque el mismo no encaja en rOpenSci o porque se solapa con uno ya existente.
- Si tu paquete cumple con los criterios mínimos, se le asignará de 1 a 3 personas para hacer la revisión, a quienes se les pedirá proporcionar revisiones en forma de comentarios sobre tu *issue* en un plazo máximo de 3 semanas.
- Te pedimos que respondas a estos comentarios en un plazo máximo de 2 semanas desde la última revisión presentada, pero puedes actualizar tu paquete o responder en cualquier momento. Tu respuesta debe incluir un enlace a la actualización del archivo *NEWS.md* de tu paquete. Aquí tienes un ejemplo de respuesta. Fomentamos las conversaciones continuas entre quienes envían el paquete y quienes lo revisan. Consulta la guía de revisión para más detalles.

- Si algún cambio en el paquete puede modificar los resultados de `pkgcheck`, se puede solicitar un nuevo chequeo con el comando `@ropensci-review-bot check package`.
- Por favor, notifícanos inmediatamente si ya no puedes mantener tu paquete o responder a las revisiones. En ese caso, se espera que retractes el envío o que encuentres responsables alternativos para mantener del paquete. También puedes discutir los problemas de mantenimiento en el Slack de rOpenSci.
- Una vez que tu paquete sea aceptado, te proporcionaremos más instrucciones sobre la transferencia de tu repositorio al repositorio de rOpenSci.

Nuestro código de conducta es obligatorio para la participación en nuestro proceso de revisión.

7 Guía para quienes hacen una revisión

¡Gracias por aceptar revisar un paquete para rOpenSci! Este capítulo incluye nuestras recomendaciones para preparar, enviar y realizar un seguimiento de tu revisión.

Puedes ponerte en contacto con la persona encargada de la edición del paquete si tienes dudas sobre el proceso en general o tu revisión en particular.

Trata de completar tu revisión en un plazo de 3 semanas desde aceptar participar de la misma. Intentaremos recordar a quienes hacen una revisión las fechas de vencimiento próximas y pasadas. La persona encargada de la edición puede asignar personas adicionales o alternativas si una revisión se retrasa excesivamente.

La comunidad de rOpenSci es lo más importante. Nuestro objetivo es que las revisiones sean abiertas, no conflictivas y con el objetivo de mejorar la calidad del software. ¡Sé amable! y comportate con respeto. Consulta nuestra guía para quienes realizan una revisión y el [código de conducta] (https://ropensci.org/code-of-conduct/) para más información.

Si utilizas nuestros estándares, listas de tareas, etc. al revisar software en otro lugar, cuenta que estos materiales proceden de rOpenSci a quienes reciben la revisión (por ejemplo, quien edita la revista, estudiantes, colegas durante una revisión interna de código), y también cuéntanoslo en nuestro foro público, o por correo electrónico si prefieres comentarlo en forma privada.

7.1 Voluntariarte para revisar

Gracias por tu deseo de participar en la revisión de software de rOpenSci revisando paquetes.

Por favor, completa nuestro formulario de voluntariado.

Si ves una presentación vigente que te sea particularmente interesante, por favor envía un correo electrónico a `info@ropensci.org`, incluyendo el nombre del paquete, la URL del *issue* de presentación y el nombre de la persona responsable de la edición. Sin embargo, ten en cuenta que las invitaciones a revisar quedan a discreción quien se encuentra a cargo de la edición, y es muy posible que ya haya realizado invitaciones. Por favor, no te ofrezcas para revisar en todos los *issues* y no te presentes a través de la interfaz de GitHub.

Para otras formas de contribuir, consulta la Guía de contribución de rOpenSci.

7.2 Preparar tu revisión

Las revisiones deben basarse en la versión actual de GitHub en la rama por defecto, a menos que se indique lo contrario. Todos los envíos generan un informe detallado sobre la estructura y funcionalidad del paquete, generado por nuestro paquete `pkgcheck`. Si el paquete ha cambiado sustancialmente desde las últimas revisiones, puedes solicitar una nueva revisión con el comando `@ropensci-review-bot check package`. Ten en cuenta que, al instalar el paquete para revisarlo, debes asegurarte de que tienes todas las dependencias necesarias (por ejemplo usa `pak::pak()`).

7.2.1 Directrices generales

Para revisar un paquete, empieza por copiar nuestra plantilla de revisión (o nuestra plantilla de revisión en español) y utilízala como lista de verificación general. Además de marcar los criterios mínimos, te pedimos que hagas comentarios generales sobre lo siguiente:

- ¿Cumple el código con los principios generales de la Guía de revisión de Mozilla?
- ¿Cumple el paquete con la guía de empaquetado de rOpenSci?
- ¿Puede mejorarse el estilo del código?
- ¿Puede reducirse el código duplicado, de haberlo?
- ¿Podría mejorarse la interfaz de usuario?
- ¿Podría mejorarse el rendimiento?
- ¿La documentación (instrucciones de instalación, viñetas, ejemplos, demos) es clara y suficiente? ¿Utiliza el principio de *múltiples puntos de entrada*? Es decir, ¿tiene en cuenta que cualquier parte de la documentación puede ser la primera exposición de una persona con el paquete y/o la herramienta o datos que utiliza?
- ¿Los nombres de funciones y argumentos forman una API de programación común y lógica que sea fácil de leer y que se beneficie de herramientas de automcompletado?
- Si tienes datos propios relevantes o un problema relacionado, trabaja con el paquete. Puede que encuentres asperezas y/o casos de uso no contemplados.

Por favor, en tus revisiones dirígete con respeto y amabilidad hacia las personas que presentaron el paquete. Nuestro código de conducta es obligatorio para quienes participan en nuestro proceso de revisión. Esperamos que envíes tu revisión en un plazo de 3 semanas, según el plazo establecido por la persona a cargo de la edición. Por favor, ponte en contacto con esta persona directamente o mediante el *issue* de envío para informarle de posibles retrasos.

Te recomendamos usar herramientas automatizadas para facilitar tu revisión. Éstas incluyen

- Comprobar el informe inicial del paquete generado por nuestro bot `@ropensci-review-bot`.
- Comprobar los registros del paquete en sus servicios de integración continua (GitHub Actions, Codecov, etc.)

- Ejecutar `devtools::check()` y `devtools::test()` en el paquete para encontrar cualquier error que haya pasado desapercibido en la computadora de quien desarrolló el paquete.
- Revisar si los tests salteados son justificados (por ejemplo usar `skip_on_cran()` en tests que usan una API versus saltearse todos los tests en un determinado sistema operativo).
- Si el paquete no se envía a través de la *branch* principal o por defecto, recuerda cambiar a la *branch* enviada para revisar antes de comenzar tu revisión. En este caso, tendrás que usar la búsqueda local para revisar el código, ya que la búsqueda en GitHub está limitada a la *branch* por defecto. Además, la documentación alojada en un sitio web de `pkgdown` no va a estar necesariamente actualizada, por lo que recomendamos revisar la documentación del paquete localmente ejecutando `pkgdown::build_site()`.

Puedes volver a correr la comprobación del paquete de `@ropensci-review-bot` en cualquier momento enviando un comentario en el *issue* de revisión con el contenido "`@ropensci-review-bot check package`".

7.2.2 Interacciones fuera del hilo

Si interactúas con quienes desarrollaron el paquete y hablas de la revisión fuera de un *issue* de revisión (en chats, DMs, en persona, issues en el repositorio del proyecto), asegúrate de que tu revisión refleje y/o enlace los elementos de estas conversaciones que sean relevantes para el proceso.

7.2.3 Experiencia de revisiones anteriores

Si es la primera vez que revisas un paquete, puede resultarte útil leer algunas revisiones anteriores. En general, puedes encontrar *issues* de presentación de paquetes aceptados aquí. Estas son algunas revisiones seleccionadas en inglés (ten en cuenta que tus reseñas no tienen que ser tan largas como estos ejemplos):

- `rtika` revisión 1 y revisión 2
- `NLMR` revisión 1 y revisión 2
- `bowerbird` comentario previo a la revisión, revisión 1, revisión 2.
- `rusda` revisión (previo a que tuviéramos una plantilla de revisión)

Puedes leer los artículos del blog sobre experiencias al revisar paquetes a través de este enlace. En particular esta publicación de Mara Averick (en inglés) habla como puedes proporcionar una devolución útil aún careciendo de experticia en el tema o en la implementación del paquete adoptando el rol de "persona ingenua" y preguntándote *"¿Qué creo que hace esta cosa? ¿lo hace? ¿qué cosas me producen rechazo?"* En otro artículo de blog Verena Haunschmid explica cómo alternó entre usar el paquete y revisar el código.

Como antiguo revisor y autor de paquetes, Adam Sparks escribió lo siguiente "[escribe] una buena crítica de la estructura del paquete y de las mejores prácticas de escritura de código. Si sabes cómo hacer algo mejor, dímelo. Como desarrollador es fácil pasar por alto oportunidades de documentación, pero como al hacer la revisión, tienes una visión diferente. Eres una persona que puede dar devolución. ¿Qué no está claro en el paquete? ¿Cómo puede hacerse más claro? Si lo utilizas por primera vez, ¿es fácil? ¿Conoces otro paquete R que quizás debería utilizar? ¿O hay alguno que esté utilizando y que quizás no debería? Si puedes contribuir al paquete, ofrécete".

7.2.4 Paquete de ayuda para quienes hacen una revisión

Si trabajas en RStudio, puedes agilizar tu flujo de trabajo de revisión utilizando el paquete `pkgreviewr` creado por la editora asociada Anna Krystalli. Digamos que aceptas revisar el paquete `refnet`, escribirías

```
remotes::install_github("ropensci-org/pkgreviewr")
library(pkgreviewr)
pkgreview_create(pkg_repo = "embruna/refnet",
                 review_parent = "~/Documents/workflows/rOpenSci/reviews/")
```

Esto

- clonará el repositorio de GitHub del paquete `refnet`,
- creará un proyecto de revisión, que contendrá una notebook para que rellenes, y la plantilla de revisión.

Ten en cuenta que si el paquete no se envía a través de la *branch* principal o por defecto, tienes que cambiar a la *branch* usada para enviar antes de comenzar tu revisión.

7.2.5 Devoluciones sobre el proceso

Te animamos a que preguntes y opines sobre el proceso de revisión en nuestro foro.

7.3 Enviar la revisión

- Cuando tu revisión esté completa, agrégala como comentario en el *issue* de revisión del paquete.
- Comentarios adicionales son bienvenidos en el mismo *issue*. Aspiramos a que las revisiones de paquetes funcionen como una conversación continua entre las partes en lugar de una única ronda de revisiones típica de los manuscritos académicos.

- También puedes crear *issues* o *pull requests* directamente al repositorio del paquete si lo deseas, pero si lo haces, por favor, menciónalo y vincúlalo en el hilo de revisión de software para que tengamos un registro y un texto centralizados de tu revisión.
- Por favor, al terminar incluye una estimación de cuántas horas le has dedicado a tu revisión.

7.4 Seguimiento de la revisión

Las personas que enviaron el paquete a revisón deben responder en un plazo de 2 semanas con sus cambios en el paquete en respuesta a tu revisión. En esta fase, te pedimos que respondas si los cambios abordan suficientemente las cuestiones planteadas en tu revisión. Fomentamos el debate continuo entre quienes enviaron el paquete y quienes lo revisan, y también puedes pedir que quienes están a cargo de la edición aclaren cosas en el mismo hilo.

- Utilizarás la plantilla de aprobación.

8 Guía para el equipo editorial

La revisión por pares de software en rOpenSci es gestionada por un equipo editorial. Estamos probando un sistema donde el rol de Líder Editorial (LE) va rotando.

Este capítulo presenta las responsabilidades quien lidera el equipo editorial, de cualquiera a cargo de editar un envío, cómo responder a un envío fuera del alcance de rOpenSci y cómo gestionar la publicación de una nueva versión de la guía de desarrollo.

Si estás en el rol de edición de forma invitada, ¡gracias por tu ayuda! Si tienes alguna duda, ponte en contacto con la persona que te invitó a encargarte del proceso de revisión de un paquete.

> Asume siempre que quienes participan en el sistema de revisión de software (tanto en la edición, como la revisión y la autoría de paquetes) están dando su mayor esfuerzo, y comunícate con amabilidad, especialmente cuando preguntes por qué hay un retraso.

8.1 Responsabilidades del rol de edición

- Además de ocuparse de los paquetes (unos 4 al año), quienes realizan la edición intervienen en las decisiones editoriales del equipo (como si un paquete está dentro del ámbito de aplicación) y determinan las actualizaciones de nuestras políticas. Generalmente lo hacemos a través de Slack, que esperamos que los miembros del equipo puedan consultar con regularidad.
- También rotamos Responsabilidades del Líder Editorial (decisiones de alcance en primera instancia y asignación de roles editoriales) entre los miembros del equipo aproximadamente cada tres meses.
- No tienes que hacer un seguimiento de otros envíos, pero si observas un problema con un paquete que está siendo gestionado por otra persona, no dudes en plantear ese problema directamente a quien está a cargo de la edición, o publica la preocupación en el canal exclusivo para el equipo editorial en Slack. Por ejemplo:
  - Sabes de un paquete que se solapa, que aún no se ha mencionado en el proceso.
  - Ves una pregunta para la que tienes una respuesta experta que no se ha dado al cabo de unos días (por ejemplo, sabes de un artículo en el blog que aborda cómo añadir imágenes a los documentos del paquete).

- Las preocupaciones relacionadas, por ejemplo, con la rapidez del proceso, deberían ser abordadas por quien sea Líder Editorial, así que es a esa persona a quien te dirigirías para tales preguntas.

8.2 Lista de tareas para la edición de un paquete

8.2.1 En el momento del envío:

- Si eres LE o eres la primera persona que responde, asigna a una persona para la edición con un comentario de `@ropensci-review-bot assign @username as editor`. Esto también añadirá la etiqueta `1/editor-checks` al issue.
- Para los envíos de paquetes estadísticos (identificables como "Submission Type: Stats" en la plantilla del issue), añade la etiqueta "stats" a la edición.
- El envío generará automáticamente una salida de comprobación del paquete por parte de ropensci-review-bot, que deberá examinarse en busca de cualquier problema pendiente (la mayoría de las excepciones deberán ser justificadas por quien presenta el paquete en el contexto particular de su paquete). Si quieres volver a realizar comprobaciones después de cualquier cambio en el paquete, envía un comentario con el texto "`@ropensci-review-bot check package`".
- Después de publicar las comprobaciones automáticas, utiliza la plantilla editorial para guiar las comprobaciones iniciales y registrar tu respuesta sobre el envío. También puedes agilizar las comprobaciones de edición utilizando el paquete `pkgreviewr` creado por la editora asociada Anna Krystalli. Procura terminar las comprobaciones y empezar a buscar personas para hacer la revisión en un plazo de 5 días laborables.
- Comprueba que la plantilla haya sido completada correctamente.
- Comprueba las políticas de alcance y solapamiento. Inicia un debate a través del canal Slack #software-review si es necesario en casos especiales que no hayan sido detectados por comprobaciones previas. Si se rechaza el paquete, ver esta sección sobre cómo responder.
- Comprueba que las partes obligatorias de la plantilla estén completas. Si así no fuera, orienta a quienes presentaron el paquete hacia las instrucciones adecuadas.
- Para paquetes que necesiten integración continua en varias plataformas (ver criterios en la sección del capítulo sobre CI) asegúrate de que el paquete se pruebe en varias plataformas (por ejemplo, haciendo que el paquete se compruebe en varios sistemas operativos a través de GitHub Actions).
- Siempre que sea posible, cuando pidas cambios, sugiere herramientas automáticas como `usethis` y `styler` y recursos en línea (secciones de esta guía, secciones del Libro *R Packages*) para facilitar tu devolución. Ejemplo de comprobaciones de una editora.
- Lo ideal es que las observaciones que hagas se aborden antes de que las personas a cargo de la revisión comiencen su trabajo.

- Si los chequeos iniciales muestran falencias importantes, solicita cambios antes de asignar personas para la revisión. Si quien envió el paquete menciona que los cambios pueden llevar tiempo, usa la etiqueta de espera escribiendo `@ropensci-review-bot put on hold`. Recibirás un recordatorio cada 90 días (en el *issue*) para que te pongas en contacto con las personas a cargo del paquete.
- Si el paquete plantea un problema inesperado relacionado con la política de rOpenSci, inicia una conversación en Slack o abre un debate en el foro rOpenSci para discutirlo con el equipo editorial (ejemplo de discusión sobre política).

8.2.2 Busca y asigna dos personas para revisar el paquete

8.2.2.1 Tareas

- Comenta con `@ropensci-review-bot seeking reviewers`.
- Utiliza la plantilla de correo electrónico si es necesario para invitar a las personas a participar en la revisión
  - Cuando envíes la invitación, incluye algo como "si no tengo noticias tuyas en una semana, asumiré que no puedes revisar", para dar un plazo claro en el que empezarás a buscar a alguien más.
- Para asignar a alguien para la revisión, usa `@ropensci-review-bot assign @username as reviewer`. `add` también puede utilizarse en lugar de `assign` y `to reviewers` (plural) en lugar de `as reviewer` (singular). Por lo tanto, lo siguiente también es válido: `@ropensci-review-bot add @username to reviewers`. Debe generarse una orden para cada persona. Si es necesario sacar a alguien de la revisión, usa `@ropensci-review-bot remove @username from reviewers`.
- Si quieres cambiar la fecha de vencimiento de una revisión utiliza `@ropensci-review-bot set due date for @username to YYYY-MM-DD`.

8.2.2.2 Cómo buscar personas para hacer una revisión

8.2.2.2.1 ¿Dónde buscarlas?

Como responsable de la edición, utiliza

- las posibles sugerencias realizadas por quienes presentaron el paquete (aunque éstas pueden tener una visión limitada de los tipos de conocimientos necesarios; sugerimos no utilizar más de una de las personas sugeridas);
- la base de datos *Airtable* de revisión y voluntariado (ver siguiente subsección);
- y personas con paquetes de rOpenSci.

Cuando estas fuentes de información no sean suficientes,

- pide ideas al equipo editorial en Slack,
- busca personas que usen el paquete o de la fuente de datos o servicio al que se conecta el paquete (a través de la apertura de *issues* en el repositorio, destacándolo, citándolo en artículos, hablando de él en redes sociales).
- También puedes buscar personas con paquetes relacionados en r-pkg.org.
- R-Ladies tiene un directorio en el que se especifican las aptitudes e intereses de las personas incluidas en la lista.
- Puedes publicar la búsqueda en los canales #general y/o #software-review del Slack de rOpenSci, o en las redes sociales.

8.2.2.2.2 Consejos para la búsqueda en Airtable

Puedes utilizar filtros, clasificación y búsqueda para identificar personas para la revisión con una experiencia concreta:

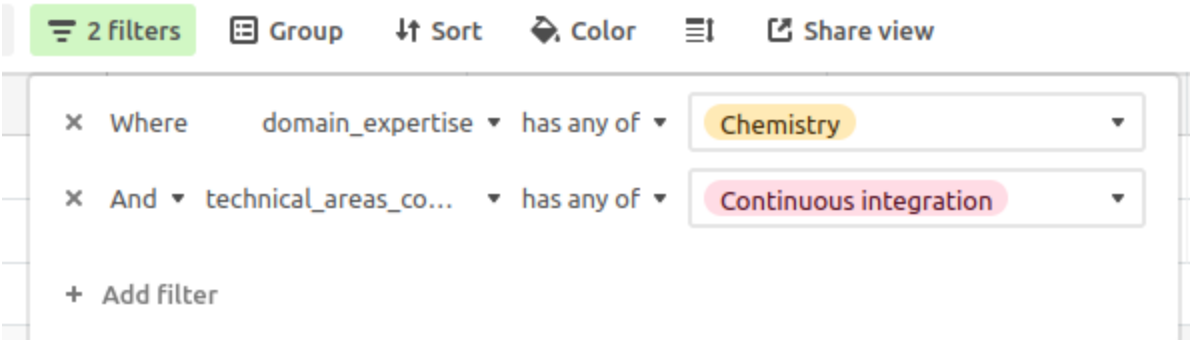

Figura 8.1: Captura de pantalla de la interfaz de filtros de Airtable con un filtro sobre experiencia en disciplinas que debe incluir química y en conocimientos técnicos que tienen que incluir integración continua

Comprueba la revisión más reciente de la persona y evita a cualquiera que haya revisado en los últimos seis meses. Asimismo, comprueba si una persona que es nueva revisando ha indicado que requiere tutoría en el campo`require_mentorship`. En caso afirmativo, utiliza la parte de tutoría de la plantilla de correo electrónico y prepárate para proporcionar orientación adicional.

8.2.2.2.3 Criterios para elegir a las personas que harán la revisión

Estos son los criterios que debes tener en cuenta a la hora de elegir a alguien para realizar la revisión. Puede que tengas que reunir esta información buscando en CRAN y en su página GitHub, así como en su presencia general en Internet (sitio web personal, Twitter).

- No ha revisado ningún paquete para rOpenSci en los últimos 6 meses.

- Alguna experiencia en el desarrollo de paquetes.
- Alguna experiencia de dominio en el campo del paquete o fuente de datos.
- No hay conflictos de intereses.
- Intenta equilibrar lo que sabes sobre su experiencia con la complejidad del paquete.
- Diversidad: las dos personas que revisen el paquete no deberían ser hombres cis blancos.
- Alguna prueba de que que le interesa el software libre o las actividades de la comunidad de R, aunque enviar un correo electrónico sin esta información está bien.

Cada envío debe ser revisado por *dos* personas. Aunque está bien que una de ellas tenga menos experiencia en el desarrollo de paquetes y más conocimientos del dominio, la revisión no debe dividirse en dos. Ambas deben revisar el paquete de forma exhaustiva, aunque desde su perspectiva particular. En general, al menos una debe tener experiencia previa en revisiones y, por supuesto, invitar gente nueva amplía nuestro grupo de revisión.

8.2.3 Durante la revisión:

- Consulta de vez en cuando a tanto a quienes están revisando el paquete como a quienes lo enviaron. Ofrece aclaraciones y ayuda cuando sea necesario.
- En general, espera 3 semanas para la revisión, 2 semanas para cambios posteriores, y 1 semana para la aprobación de los cambios por parte de quien hace la revisión.
- Una vez enviada cada revisión,
  - Escribe un comentario agradeciendo a quien hizo la revisión con tus palabras;
  - Registra la reseña escribiendo un nuevo comentario `@ropensci-review-bot submit review <review-url> time <time in hours>`. Por ejemplo, para la reseña https://github.com/ropensci/software-review/issues/329#issuecomment-809783937 el comentario sería `@ropensci-review-bot submit review https://github.com/ropensci/software-time 4`.
- Si quien presentó el paquete deja de responder, consulta las políticas y/o notifica al resto del equipo editorial en el canal Slack para discutirlo. Es importante que, si se asignó una persona para hacer la revisión a un issue que se cerró, te pongas en contacto con esa persona al cerrar el issue para explicarle la decisión, agradecerle una vez más su trabajo y tomar nota en nuestra base de datos para asignarle la próxima vez un envío con altas posibilidades de que la revisión del software se realice sin problemas (por ejemplo, una persona que ya nos haya enviado paquetes).
- Una vez realizados los cambios, cambia la etiqueta de estado de revisión a `5/awaiting-reviewer-response` y pide a las personas a cargo de la revisión que indiquen su aprobación con la [plantilla de aprobación de revisión]{#approval2template}.

8.2.4 Después de la revisión:

- `@ropensci-review-bot approve <package-name>`
- *Si quien tiene la propiedad original del repositorio no quiere transferirlo a rOpenSci, añade una línea con su dirección a esta lista de repositorios para garantizar que el paquete se incluya en el registro de paquetes de rOpenSci.*
- Nomina el paquete para que aparezca en un artículo del blog de rOpenSci o en una nota técnica si crees que puede ser de gran interés. Por favor, anota en el *issue* de revisión una o dos cosas que destacables, y etiqueta a `@ropensci/blog-editors` para su seguimiento.
- Si el paquete tiene un gitbook asociado (aunque sólo sea parcialmente), ponte en contacto con alguien del equipo de rOpenSci para que se ponga en contacto con quienes lo mantienen y hable sobre su transferencia a la página `ropensci-books` organización de GitHub.

8.2.5 Promoción de paquetes:

- Dirige quien presentó el paquete a los capítulos de la guía sobre publicación de paquetes, marketing y preparación para GitHub.

8.3 Responsabilidades del rol de LE

Quien ocupa el rol de LE desempeña sus funciones durante 3 meses o el tiempo que acuerden todos los miembros del equipo editorial. Tiene derecho a tomar decisiones de alcance y solapamiento con la mayor independencia posible (pero puede solicitar ayuda/asesoramiento). En detalle, el rol de LE implica las siguientes funciones

- Vigila todos los *issues* publicados en el repositorio de revisión del software (se suscribe a las notificaciones del repositorio en GitHub, o vigila la página `#software-peer-review-feed` en Slack).
- Etiqueta el *issue* con `0/editorial-team-prep`
- Usa `@ropensci-review-bot check srr` sobre consultas previas a la presentación de software estadístico. Revisar los capítulos correspondientes a la *Guía de desarrollo de estadística* para más detalles.
- Asigna los envíos de paquetes a otros miembros del equipo editorial, incluyéndose, para que se encarguen de ellos. En la mayoría de los casos, esta responsabilidad rota entre los miembros del equipo editorial a menos que quien sea LE considere que un miembro en particular es especialmente adecuado para algún paquete, o que alguien rechace encargarse del envío por no tener tiempo o por tener un conflicto de intereses.

  `@ropensci-review-bot assign @username as editor`

- Supervisa el ritmo del proceso de revisión y recuerda a otros miembros del equipo editorial que hagan avanzar los paquetes según sea necesario.
- Al asumir el rol de LE, revisa el estado de las revisiones abiertas actuales y recuerda a quienes las editan que respondan o actualicen el estado según sea necesario.
- Responde a los *issues* publicados en el repositorio software-review-meta
- Toma decisiones sobre el alcance y solapamiento de las consultas previas a la presentación, las recomendaciones desde JOSS u otros socios de publicación, y las presentaciones si ven un caso ambiguo (este último caso también puede ser realizado por personas que están a cargo de la edición de un paquete (ver más abajo)). Para iniciar el debate, debe publicar en el canal exclusivo para el equipo editorial del Slack de rOpenSci junto con un pequeño resumen de lo que trata la presentación (pre)enviada/remitida, y sus dudas, es decir, digerir un poco la información. Si al cabo de uno o dos días considera que no ha recibido suficientes respuestas, puede enviar una notificación a todo el equipo editorial.
  - Cualquier miembro del equipo editorial debería sentirse libre de intervenir en estos casos. Véase esta sección sobre cómo responder a los (pre) envíos fuera de nuestro alcance temático.
  - Tras explicar la decisión de fuera de alcance, escribe un comentario con "`@ropensci-review-bot out-of-scope`" en el *issue*.
- Solicita a alguien más para cubrir el rol de LE cuando finalice su rotación (establece un recordatorio en el calendario antes de la fecha prevista de finalización y pide reemplazos en el canal de Slack del equipo editorial).

8.3.1 Pide más detalles

En algunos casos, la documentación en línea es escasa. Un *README* mínimo y la ausencia de un sitio web de *pkgdown* dificultan la evaluación. En ese caso, por favor, pide más detalles: aunque el paquete se considere fuera de alcance, la documentación del paquete habrá mejorado, así que no nos molesta pedir estos esfuerzos.

Ejemplo de texto

```
Hola <nombre> y muchas gracias por su envío.

Estamos discutiendo si el paquete está dentro del ámbito de aplicación y necesitamos un poco

¿Le importaría añadir más detalles y contexto al *README*?
Después de leerlo alguien con poco conocimiento del dominio debería tener suficiente informa

<opcional>
```

```
Si un paquete tiene funcionalidad que se solapa con otros paquetes, requerimos que demuestre
</opcional>
```

8.3.2 Invitar a una persona para la tarea de edición

Tras debatirlo con otros miembros del equipo, quien sea LE puede invitar a una persona externa para que se encargue de un envío (por ejemplo, si el volumen de envíos es grande, si todos los miembros del equipo de edición tienen un conflicto de intereses, si se necesitan conocimientos específicos, o como prueba antes de enviar una invitación a formar parte del equipo de edición).

Cuando invites a alguien para realizar una edición de forma invitada:

- Pregunta sobre los conflictos de intereses utilizando el mismo texto que para quienes revisan paquetes,
- Proporciona un enlace a la guía para quines se encargan de la edición.

Si la persona acepta (¡bien!),

- asegúrate de que haya activado la autenticación de 2 factores en su cuenta de GitHub;
- invítala al equipo `ropensci/editors` y a la organización ropensci;
- una vez que haya aceptado esta invitación al repositorio, asígnale el *issue*;
- asegúrate de que esté invitada al espacio de trabajo en Slack de rOpenSci;
- añade su nombre a la tabla de *guest-editor* de Airtable (para que su nombre aparezcan en este libro y en el *README* de la revisión del software).

Una vez finalizado el proceso de revisión (paquete aprobado, issue cerrado),

- vuelve a dar las gracias a la persona invitada;
- elimínala del equipo `ropensci/editors` (pero no de la organización ropensci).

8.4 Responder a las presentaciones fuera del ámbito de aplicación

Agradece a quienes presentaron el paquete por el envío, explica los motivos de la decisión y propone otros lugares de publicación, si corresponde, y al foro de debate de rOpenSci. Utiliza el texto de Objetivos y alcance en particular en lo que respecta a la evolución del alcance a lo largo del tiempo, y al solapamiento y las diferencias entre el desarrollo de unconf/personal/revisión de software.

Ejemplos de envíos y respuestas fuera del ámbito de aplicación.

8.5 Responder a las preguntas de quienes revisan

Las personas que hacen revisiones pueden pedir devoluciones sobre, por ejemplo, el tono de su revisión. Además de darles las orientaciones generales de esta guía, pregunta a los miembros del equipo de edición o abre un issue cuando falten indicaciones. Aquí tienes algunos ejemplos de reseñas que pueden ser útiles:

- Ejemplo duro pero constructivo: la parte de esta revisión que sugiere una reescritura de la viñeta: ropensci/software-review#191 (comentario).
- El paquete `slopes` que acabó siendo rediseñado fundamentalmente en respuesta a las revisiones. Todas las revisiones fueron en todo momento totalmente constructivas, lo que parece haber desempeñado un papel importante a la hora de motivar a quienes desarrollaron el paquete a embarcarse en una revisión tan importante. Comentarios como, *"este paquete no…"* o *"no tiene …"* iban seguidas invariablemente de sugerencias constructivas sobre lo que se podría hacer (hay, por ejemplo, varias en una de las primeras revisiones).
- La revisión al paquete `tic` expresaban educadamente sus reservas: https://github.com/ropensci/software-review/issues/305#issuecomment-504762517 y https://github.com/ropensci/software-review/issues/305#issuecomment-508271766
- El paquete `bowerbird` recibió una útil "revisión previa" que dio lugar a una división del paquete antes de las revisiones propiamente dichas.

8.6 Gestión de la publicación de la guía de desarrollo

Si te encargas de gestionar una versión de este mismo libro que estás leyendo, utiliza la guía para la publicación de libros como plantilla de issue para publicar en el gestor de *issues* de la guía de desarrollo y no dudes en hacer preguntas al resto del equipo de edición.

8.6.1 Gobernanza de la guía de desarrollo

Para modificaciones muy pequeñas de la guía de desarrollo, no es necesaria la revisión del PR. Para enmiendas mayores, solicita la revisión de al menos varios miembros del equipo de edición (si ninguno participó en la discusión relacionada con la enmienda, solicita una revisión de todos en GitHub, y en ausencia de cualquier reacción aprueba el PR después de una semana).

Dos semanas antes del lanzamiento de una nueva versión de la guía de desarrollo, una vez que el PR de dev a master **y la publicación en el blog** estén listos para su revisión, todos los miembros del equipo de edición deben recibir una notificación de GitHub ("solicitud de revisión" en el PR de dev a main) y Slack, pero no es necesario que todos aprueben explícitamente la publicación.

8.6.2 Artículo de blog sobre una publicación

El artículo del blog sobre una nueva edición será revisado por el equipo de edición y por alguien de `@ropensci/blog-editors`.

8.6.2.1 Contenido

Consulta la guía general para blogs rOpenSci y las orientaciones más específicas a continuación.

Primer ejemplo de un artículo de este tipo; segundo ejemplo.

El artículo del blog debe mencionar todos los elementos importantes del registro de cambios organizados en (sub)secciones: por ejemplo, una sección sobre el gran cambio A, otra sobre el gran cambio B, y otra sobre cambios menores agrupados. Menciona primero los cambios más importantes.

Por cada cambio realizado por una persona que colabora de manera externa, agradéceselo explícitamente utilizando la información del registro de cambios. Por ejemplo `[Matt Fidler](https://github.com/mattfidler/) arregló nuestra sección sobre mensajes en la Consola [ropensci/dev_guide#178](https://github.com/ropensci/dev_guide/pull/178).`.

Al final del post, menciona los próximos cambios enlazando a los *issues* abiertos en el repositorio, e invita a quienes leen a contribuir a la guía de desarrollo abriendo *issues* y participando en los debates abiertos. Plantilla de conclusiones:

```
En este post resumimos los cambios incorporados a nuestro libro ["rOpenSci Packages: Develop
Agradecemos todas las personas cuyas contribuciones han hecho posible esta versión.
Ya estamos trabajando en actualizaciones para nuestra próxima versión, como *ISSUE1*, *ISSUE
Echa un vistazo a [la lista de *issues*](https://github.com/ropensci/dev_guide/issues/) si qu
```

8.6.2.2 Autoría

Quien escribe el post lidera la lista de autoría, el resto del equipo de edición aparecen en orden alfabético.

9 Gestión editorial

Guía para gestionar el equipo editorial.

9.1 Reclutar nuevas personas para la edición

Reclutar nuevas personas y mantener un equipo editorial funcional y equilibrado es responsabilidad de quien Lidere la Revisión de Software con el apoyo y asesoramiento del comité editorial.

Estos son los pasos a seguir:

- Crea un canal privado para discutir la invitación; de este modo, el historial de conversación no queda disponible en el canal al que luego se unirán los nuevos miembros del equipo, lo que podría ser incómodo.
- Haz un *ping* a las personas del equipo editorial para asegurarte de que reciban una notificación, ya que se trata de un tema importante.
- Espera a que la mayoría de del equipo intervenga antes de realizar la invitación. Dales una semana para que respondan.

9.2 Invitar a una nueva persona al equipo

- Los posibles nuevos miembros pueden empezar realizando una edición de forma invitada. Haz la invitación como invitarías a cualquier otra persona por otros motivos.
- Si el posible nuevo miembro realiza una edición invitada, evalúa cómo ha sido el proceso una vez abordada la presentación. Vuelve a pedir consejo a otras personas del equipo.
- Envía un correo electrónico:

```
Nos gustaría invitarte a formar parte del consejo editorial de rOpenSci como miembro en plen
Creemos que serías una magnífica incorporación al equipo.

[SI REALIZÓ EDICIÓN DE FORMA INVITADA: Ya conoces el rol de edición ya que has sido editora/
Pedimos a los miembros del comité editorial que se comprometan de manera informal a prestar
```

```
A corto plazo, cualquier persona puede rechazar encargarse de un paquete o decir: "Tengo mi a

Además de encargarse de los paquetes, los miembros del equipo editorial opinan sobre las deci
Generalmente usamos Slack, donde esperamos que los miembros del equipo participen regularment
Tenemos reuniones anuales del equipo editorial.
También rotamos las responsabilidades de Lider Editorial (quien toma decisiones de alcance en
Tendrás la oportunidad de entrar en esta rotación cuando lleves un tiempo en el equipo, norma
Algunos mimebros del equipo también asumimos proyectos más grandes para mejorar el proceso de

¡Esperamos que te unas al equipo!
Es un momento emocionante para la revisión por pares en rOpenSci.

Por favor, reflexiónalo, pregúntanos cualquier duda que tengas y haznos saber si puedes unirt

Te deseamos lo mejor,
[NOMBRE], en nombre del equipo editorial de rOpenSci.
```

9.3 Incorporar un nuevo miembro al equipo

- Pide a quien gestiona la comunidad de rOpenSci que
  - agregue al nuevo miembro del equipo editorial al sitio web de rOpenSci;
  - prepare una artículo de presentación en el blog.
- Solicita al nuevo miembro que active la autenticación de dos factores (2FA) en GitHub si aún no lo han hecho.
- Invítalo a la organización GitHub de rOpenSci y a los equipos `editors` y `data-pkg-editors` o `stats-board`, según corresponda. Esto le dará los permisos adecuados y le permitirá recibir notificaciones específicas del equipo.
- Dale acceso a la base de datos de revisión de software de AirTable.
- Dale acceso al canal privado del equipo editorial en el espacio de trabajo de Slack de rOpenSci (y al espacio de trabajo de Slack en general si no lo tenía previamente, en ese caso pregunta a quien maneja la comunidad de rOpenSci).
- Publica un mensaje de bienvenida en el canal, haciendo ping a `@editors`.
- Agrégalo al equipo `editors` de Slack para que `@editors` también le envíe una notificación.
- Añade su nombre a:
  - la lista de personas autoras de dev_guide;

  - el capítulo dev_guide que introduce la revisión de software (en dos lugares de este archivo: como miembros del equipo editorial y un poco más abajo para eliminar su nombre de la lista de personas que revisan paquetes);
  - el *README* de revisión de software (también en dos lugares de este archivo!).

Tanto la `dev_guide` como el *REAMDE* de revisión de software se renderizan automáticamente con integración continua.

9.4 Dar de baja a un miembro del equipo editorial

- Agradece su trabajo.
- Elimínalo del canal del equipo editorial y del equipo `@editors` de Slack.
- Elimínalo del equipo `editors` y del sub-equipo correspondiente en GitHub.
- Informa a quien maneja la comunidad de rOpenSci o a algún otro miembro del personal para moverlo a la sección de ex-integrantes en el sitio web.
- Elimina su acceso al espacio de trabajo de Airtable.
- Elimínalo de:
  - el capítulo dev_guide que introduce la revisión de software (en dos lugares de este archivo: como miembro del equipo editorial y un poco más abajo para eliminar su nombre de la lista de personas que revisan paquetes);
  - el *README* de revisión de software (también en dos lugares de este archivo!).

Tanto la `dev_guide` como el *REAMDE* de revisión de software se renderizan automáticamente con integración continua.

Parte III

Manteniendo paquetes

10 Guía rápida para mantener paquetes de rOpenSci

Un recordatorio de la infraestructura y los canales de contacto para quienes mantienen paquetes de rOpenSci.

10.1 ¿Necesitas ayuda?

Si necesitas ayuda (por ejemplo una revisión de un PR o resolver problemas de CI), o asistencia para encontrar personas que ayuden o tomen la responsabilidad de mantener el paquete, o si necesitas que retiremos tu paquete, menciónanos en GitHub con `@ropensci/admin` o envía un correo electrónico a `info@ropensci.org`.

También puedes utilizar el canal de slack #package-maintenance.

¡Nunca dudes en pedir ayuda!

10.2 Acceso al repositorio GitHub

Deberías tener acceso "admin" al repositorio GitHub de tu paquete. Si no es el caso (por ejemplo, porqué falló el proceso automatizado o porque perdiste acceso después de haber tenido que desactivar temporalmente la autenticación de dos factores), por favor contáctanos enviando un correo electrónico a `info@ropensci.org`.

10.3 Otros temas de GitHub

Si tienes una pregunta o un pedido sobre GitHub, como por ejemplo, añadir una nueva persona que colabora a la organización GitHub, puedes utilizar un canal público en el Slack de rOpenSci o mencionar a `@ropensci/admin` en GitHub.

10.4 Documentación creada con pkgdown

Consulta rOpenSci docs.

10.5 Acceso al Slack de rOpenSci

Quienes mantienen o desarrollan paquetes de rOpenSci deberían tener acceso al Slack de rOpenSci. Si no recibiste la invitación o no la aceptaste a tiempo, o si quieres que alguien que ahora contribuye regularmente a tu paquete reciba una invitación, por favor envía un correo electrónico a `info@ropensci.org` indicando el correo electrónico de la persona que quiere unirse al espacio de trabajo.

El canal #package-maintenance es relevante para hacer y responder consultas, así como para dar y recibir apoyo emocional cuando sea necesario.

10.6 Artículos para el blog de rOpenSci sobre tu paquete

Consulta nuestra guía de blog.

10.7 Promoción de issues de tu paquete {#package-issues-promotion}*issues* de tu paquete

Si etiquetas *issues* con *"help wanted"* ("se busca ayuda", en inglés), las difundiremos en la comunidad.

10.8 Promoción de casos de uso del paquete

Puedes notificar ejemplos y casos de uso de tu paquete o animar a las personas que usan tu paquete a compartirlos en nuestro foro, para que sean publicados en nuestro sitio web y en nuestro boletín de noticias.

11 Guía de colaboración

Colaborar con otras personas mejorará tu paquete, y si les das a algunas de ellas rol de autoría y permisos de escritura en el repositorio, tu paquete se desarrollará de forma más sostenible. Además, ¡trabajar en equipo puede ser muy agradable!

Los consejos para colaborar en este capítulo están divididos en una sección sobre cómo hacer que la infraestructura de tu repo sea accessible para la contribución y la colaboración (código de conducta, directrices de contribución, etiquetas de *issues*) y una sección sobre cómo colaborar con nuevas personas que quieran contribuir, en particular en el contexto de la organización "ropensci" de rOpenSci en GitHub.

Además de estos consejos, en su mayoría técnicos, es importante recordar que hay que ser amable y tener en cuenta la perspectiva de las otras personas, especialmente cuando sus prioridades difieren de las tuyas.

11.1 Haz que tu repositorio sea accesible a las contribuciones y la colaboración

11.1.1 Código de conducta

Tras el traslado a nuestra organización de GitHub, el Código de Conducta de rOpenSci se aplicará a tu proyecto. Por favor, añade este texto al *README*

```
Ten en cuenta que este paquete se publica con un [Código de Conducta para contribuciones](ht
Al contribuir a este proyecto, te comprometes a cumplir sus condiciones.
```

Y eliminar el código de conducta actual del paquete, si lo hubiera.

11.1.2 Guía de contribución

Puedes utilizar guías de *issues*, *pull request* y de contribución. Tener un archivo de contribución como `.github/CONTRIBUTING.md` o `docs/CONTRIBUTING.md` es obligatorio. Una forma fácil de insertar una plantilla para una guía de contribución es la función `use_tidy_contributing()` del

paquete `usethis` que inserta esta plantilla como `.github/CONTRIBUTING.md`. Un ejemplo más extenso es esta plantilla de Peter Desmet o las completas páginas wiki de GitHub para el paquete mlr3. Estas y otras plantillas generalmente tendrán que ser modificadas para su uso con un paquete de rOpenSci, en particular haciendo referencia y enlazando con nuestro Código de Conducta como se describe en otra sección de este libro. Modificar una guía de contribución genérica para añadir un toque propio también tiende a hacer que parezca menos impersonal y más sincera. Las preferencias personales en una guía de contribución incluyen

- Preferencias de estilo. Sin embargo, puedes preferir que el estilo sea uno automático (a través de lintr o styler) o bien arreglar el estilo de código por tu cuenta especialmente si no utilizas un estilo de código popular como el estilo de código tidyverse.
- Infraestructura, como roxygen2.
- Preferencias de flujo de trabajo. Por ejemplo, abrir un *issue* antes de hacer un *pull request*.
- Una declaración de alcance, como en el paquete skimr.
- Creación de una cuenta de prueba, tests simulados, enlace a documentación externa.

rOpenSci recomienda que las guías de contribución incluyan una declaración del ciclo de vida, que aclare la visión y las expectativas para el desarrollo futuro del paquete, como en este ejemplo. Los paquetes estadísticos deben tener una declaración de ciclo de vida, como se especifica en Normas Generales de Estadística G1.2. Ese enlace proporciona una plantilla para una declaración de ciclo de vida sencilla. Los archivos CONTRIBUTING.md también pueden describir cómo se reconocen las contribuciones (ver esta sección).

Te animamos a que dirijas las consultas y comentarios que no sean informes de errores o pedidos de nueva funcionalidad al foro de rOpenSci después de asegurarte de que verás esas preguntas en el foro. El foro puede usarse para hacer preguntas sobre cómo usar el paquete e informar de sus casos de uso, y puedes suscribirte a categorías y etiquetas individuales para recibir notificaciones sobre tu paquete. No dudes en mencionar esto en los documentos de tu paquete, ya sea en la guía de contribución o en la plantilla de *issue*. Pide etiquetar las publicaciones con el nombre del paquete.

Una vez que un *pull request* está más cerca de ser aceptado, puedes utilizar un flujo de trabajo de *GitHub Actions PR* para aplicar el estilo al código con styler.

11.1.3 Gestión de issues

Utilizando las funciones de GitHub en torno a los *issues* puedes hacer que sean más encontrables y que tu hoja de ruta sea pública.

11.1.3.1 Plantillas de issue

Puedes utilizar una o varias plantilla(s) de *issue* para que los informes de errores o pedido de funcionalidad sean más fáciles de completar. Cuando hay varias plantillas de *issue*, éstas se muestran en un menú al hacer click en el botón de "nuevo *issue*"

Incluso puedes configurar una de las opciones para que apunte a algún lugar fuera de tu repositorio (por ejemplo, un foro de discusión).

Consulta la documentación de GitHub (en inglés).

11.1.3.2 Etiquetado de issues

Puedes utilizar etiquetas como "se necesita ayuda" y "buen primer *issue*" para que seas más fáciles de encontrar, incluso por gente que llega por primera vez a tu repo. Consulta este artículo de GitHub (en inglés). También puedes utilizar la etiqueta "principiante". Consulta ejemplos de issues para principiantes en todos los repos de ropensci.

11.1.3.3 Resaltado de issues

Puedes resaltar hasta 3 issues por repositorio que aparecerán en la parte superior de tu gestor de *issues* como bonitas tarjetas. Puede ayudar a anunciar cuáles son tus prioridades.

11.1.3.4 Hitos

Puedes crear hitos y asignarles *issues*, lo que ayuda a ver lo que planeas para la próxima versión de tu paquete, por ejemplo.

11.1.4 Comunicación

Puedes dirigir a quienes usan tu paquete al foro de rOpenSci, si lo supervisas, o habilitar Discusiones de GitHub para el repositorio de tu paquete. Las discusiones de GitHub pueden convertirse en *issues* si fuera necesario (y al revés, los *issues* pueden convertirse en discusiones).

11.2 Trabajar en equipo

Lo primero lo primero: ¡estate al tanto de tu repositorio de GitHub!

- no te olvides de **vigilar tu repositorio de GitHub** para que te notifiquen los *issues* o *pull requests* (si trabajas por períodos, puedes informarlo en la guía de contribución).
- no olvides hacer *push* de las actualizaciones que tengas localmente.
- desactiva los test que fallan si no puedes arreglarlos, ya que crean ruido en los *pull requests* que pueden desconcertar quienes comienzan a colaborar.

11.2.1 Incorporación de miembros al equipo

No hay una regla general en rOpenSci sobre cómo debes incorporar a quienes colaboran en tu paquete al equipo. Deberías ir dándoles más derechos en el repositorio a medida que ganes confianza, y definitivamente deberías reconocer las contribuciones (ver esta sección).

Puedes pedir a las nuevas personas que se incorporen que hagan PRs (ver la siguiente sección para evaluar un PR localmente, es decir, más allá de las comprobaciones de CI) a *dev/main* y evaluarlos antes de aceptarlos, y después de un tiempo dejar que hagan *push* a *main*. También puede que quieras mantener un sistema de revisión de PRs… ¡incluso para ti una vez que tengas miembros de equipo!

Jim Hester ofrece un posible modelo de incorporación de miembros de equipo en su repositorio de `lintr`.

Si tu problema es *reclutar* nuevas personas, puedes publicar una convocatoria abierta como la de Jim Hester en Twitter, GitHub y, como responsable de un paquete de rOpenSci, puedes pedir ayuda en el Slack de rOpenSci y solicitar al equipo de rOpenSci ideas para reclutar ayuda.

11.2.2 Trabajar con otras personas (Incluyéndote a tí en el futuro)

Las *branches* son baratas. Utilízalas mucho cuando desarrolles funciones, pruebes nuevas ideas o arregles problemas.

Si sigues el desarrollo troncal, "agregarás pequeñas y frecuentes actualizaciones" a tu *branch* principal. Ver también la documentación del Flujo de GitHub y el flujo de GitLab. Es posible que quieras incrementar los números de versión (en `DESCRIPTION`) frecuentemente. Un aspecto particular del trabajo con otras personas es la revisión de *pull requests*. Pueden ver algunas guías útiles en:

- *The Art of Giving and Receiving Code Reviews (Gracefully)* (El arte de dar y recibir revisiones de código (con elegancia)), por Alex Hill
- Documentación de GitHub sobre revisiones de PR

Puedes modificar la configuración de tu repositorio de GitHub, por ejemplo requeriendo revisiones de *pull requests* antes de aceptarlas e incorporarlas. También puedes consultar la documentación acerca de propiedad sobre código.

Para hacer y revisar *pull requests* recomendamos explorar las funciones del paquete usethis.

Para tu configuración "git remote" consulta *Happy Git with R*. También es útil la sección Useful Git patterns for real life en el mismo libro.

11.2.3 Atribuye con generosidad

Si una persona contribuye a tu repositorio, considera añadirla en el archivo DESCRIPTION, con el rol de contribución ("ctb") para pequeñas contribuciones o autoría ("aut") para contribuciones mayores. Tradicionalmente, cuando se cita un paquete en una publicación científica, sólo se enumeran a quienes están como "aut", no quienes están como "ctb". Los sitios web creados con `pkgdown` listan en la página de inicio sólo a las personas con rol de autoría ("aut"). El resto están listadas en una página dedicada.

Como mínimo, considera la posibilidad de añadir el nombre de quien contribuyó cerca de la línea de la función o corrección de errores en el archivo NEWS.md.

Te recomendamos reconozcas estas contribuciones generosamente, ya que es algo agradable y porque hará que sea más probable que la gente vuelva a contribuir a tu paquete o a otros repos de la organización.

Como recordatorio de nuestra guía de empaquetado si tu paquete fue revisado y crees que quienes lo hicieron han hecho una contribución sustancial al desarrollo de tu paquete, puedes usar el campo `Authors@R` con el rol de revisión (`"rev"`), así

```
persona("Bea", "Hernández", rol = "rev",
comment = "Bea revisó el paquete (v. X.X.XX) para rOpenSci, véase <https://github.com/ro
```

Sólo incluye a quienes hicieron la revisión con su consentimiento. Lee más en esta entrada del blog *Thanking Your Reviewers: Gratitude through Semantic Metadata* (Agradeciendo a quienes revisaron tu paquete: La gratitud a través de los metadatos semánticos). Ten en cuenta que 'rev' generará una NOTA de CRAN a menos que el paquete esté construido con R v3.5. Asegúrate de que utilizas la ultima versión de `roxygen2` en CRAN.

Por favor, no incluyas a quienes se encargaron de la edición. ¡Tu participación y contribución a rOpenSci es suficiente agradecimiento!

11.2.4 Bienvenida a nuevas personas en rOpenSci

Si das a alguien permisos de escritura en el repositorio

- por favor, ponte en contacto con un miembro del quipo para **invitar a la organización GitHub "ropensci" de rOpenSci** a este nuevo miembro del equipo (en lugar de una colaboración externa)
- ponte en contacto con quien gestiona la comunidad de rOpenSci o con otro miembro del equipo para que el nuevo miembro pueda ser **invitado al espacio de trabajo de Slack de rOpenSci**.

11.3 Otros recursos

- Llamada comunitaria de rOpenSci *Set Up Your Package to Foster a Community* (Configura tu paquete para fomentar una comunidad).
- Para reutilizar respuestas amables y habituales, considera las respuestas guardadas.

12 Cambio de quienes mantienen paquetes

Este capítulo presenta nuestra guía para asumir el mantenimiento de un paquete.

12.1 ¿Quieres dejar de mantener tu paquete?

Nuestro newsletter quincenal tiene una sección de pedidos de colaboración denominada *"Call for Contributors"* (*Pedidos de contribuciones*, en inglés). En la misma se destacan paquetes que buscan nuevas personas para encargarse del mantenimiento. Si quieres dejar de mantener tu paquete, contáctanos para que lo incluyamos en esta sección.

12.2 ¿Quieres encargarte del mantenimiento de un paquete?

Nuestro newsletter quincenal tiene una sección de pedidos de colaboración denominada *Pedidos de contribuciones*. En la misma se destacan paquetes que buscan nuevas personas para encargarse del mantenimiento. Si aún lo hiciste, es una buena idea subscribirte al newsletter para recibir notificaciones cuando hay un paquete buscando ayuda.

12.3 Asumir el mantenimiento de un paquete

- Agrégate en el archivo DESCRIPTION, con rol `role = c("aut", "cre")` y cambia el rol de la persona anterior a `role = c("aut")`.
- Asegúrate reemplazar el nombre anterior en cualquier otra parte del paquete por el tuyo, manteniendo a la persona anterior como autora (por ejemplo, en el archivo de manual a nivel de paquete, CONTRIBUTING.md, CITATION, etc.)
- La Guía de Colaboración tiene consejos sobre cómo añadir nuevos miembros al equipo de mantenimiento y colaboración.
- Para paquetes que han sido archivados o han quedado huérfanos en CRAN, no es necesario el consentimiento de quien lo mantenía anteriormente para hacerse responsable del mismo en CRAN. En estos casos, contáctanos para que podamos ofrecerte la ayuda que necesites.

- Si el paquete no ha sido archivado por CRAN y cambia quien lo mantiene, haz que la persona responsable anterior envíe un correo electrónico a CRAN y ponga por escrito quién se hará cargo del mantenimiento de ahí en adelante. Asegúrate de mencionar que se envió ese correo cuando envíes la próxima versión a CRAN. Si quien mantenía al paquete no es localizable o no envía ese correo electrónico, ponte en contacto con el personal de rOpenSci.
- Si puedes contactarte con quien te está transfiriendo el paquete, puedes organizar una reunión para que te explique cómo funciona.

12.3.1 Preguntas frecuentes

- Hay algunos *issues* no resueltos del paquete que no sé cómo arreglar. ¿A quién puedo pedir ayuda?

  Depende; esto es lo que hay que hacer en diferentes situaciones:

  - Si se puede contactar con la persona anterior, contáctate con ella y pídele ayuda.
  - Consulta el canal #package-maintenance en el Slack de rOpenSci. Este es un buen lugar para obtener ayuda sobre problemas específicos o generales.
  - Siéntete libre de hacer cualquier pregunta en el foro de discusión de rOpenSci.
  - No dudes en contactarte con cualquier integrande del equipo de rOpenSci por correo electrónico o enviando *issues* a GitHub; nos encantará ayudarte.
  - Por supuesto, también hay ayuda general de R en el Foro de la comunidad de Posit, StackOverflow, Mastodon #rstats, etc. si eso se ajusta a tus necesidades.

- ¿Cuánto puedes o debes cambiar en el paquete?

  Para obtener ayuda general sobre cómo cambiar el código de un paquete, consulta la sección Evolución de paquetes.

  Al pensar en esto, hay varias cosas a considerar.

  ¿Cuánto tiempo tienes para dedicarle al paquete? Si tu tiempo es muy limitado, lo mejor será que te centres en las tareas más críticas, sean las que sean para el paquete en cuestión. Si tienes mucho tiempo, tus objetivos pueden ser más ambiciosos.

  ¿Qué grado de madurez tiene el paquete? Si el paquete es maduro, es probable que mucha gente tenga código que dependa de su API (es decir, las funciones exportadas y sus parámetros). Además, si hay paquetes en CRAN que dependen de tu paquete, entonces tienes que comprobar que cualquier cambio que hagas no rompa esos paquetes. Cuanto más maduro sea el paquete, más cuidado deberás tener a la hora de hacer cambios, especialmente con los nombres de las funciones exportadas, sus parámetros y la estructura exacta de lo que devuelven las funciones exportadas. Es más fácil hacer cambios que sólo afecten a los aspectos internos del paquete.

12.4 Tareas del personal de rOpenSci

Como organización, a rOpenSci le interesa asegurarse de que los paquetes de nuestra suite estén disponibles mientras sean útiles para la comunidad. Si un paquete se queda sin personas que lo mantengan, en la mayoría de los casos intentaremos conseguir nuevas personas que se encarguen del mismo. Para ello, las siguientes tareas son responsabilidad del personal de rOpenSci.

- Ten en cuenta que el hecho que un repositorio no haya ninguna actividad (*commits*, *issues*, *pull requests*) en mucho tiempo puede significar simplemente que se trata de un paquete maduro con poca necesidad de cambios.
- Si no hay respuestas a reportes de problemas o *pull requests* en muchos meses, ya sea por email, *issues* en GitHub o mensajes de Slack:
  - Ponte en contacto para averiguar cuál es la situación. Puede que la persona encargada diga que le gustaría dejar de mantener el paquete, en cuyo caso busca un reemplazo.
- Si no recibes respuesta o no puedes establecer el contacto:
  - Si esto ocurre, insistiremos con el intento de contacto durante un mes como máximo. Sin embargo, si la actualización del paquete es urgente, podemos utilizar nuestro acceso de administrador para realizar los cambios en su nombre.
- Haz un llamado en la sección “Pedidos de contribuciones” del boletín rOpenSci para buscar reemplazo - abre un *issue* en el repo del boletín.

13 Publicar un paquete

Tu paquete probablemente va a tener diferentes versiones a lo largo del tiempo: instantáneas de un estado del paquete que puedes publicar en CRAN, por ejemplo. Estas versiones deben estar debidamente *numeradas*, *publicadas* y *descriptas en un archivo NEWS*. Más detalles a continuación.

Ten en cuenta que el proceso de actualización de *NEWS* y de versionado de tu paquete se puede hacer más fluido utilizando el paquete `fledge`.

13.1 Versionar

- Recomendamos encarecidamente que los paquetes de rOpenSci utilicen el versionado semántico. Hay una explicación detallada en el capitulo “lifecyle” del libro “R packages” (en inglés).

13.2 Publicar

- Utiliza las funciones `usethis::use_release_issue()` y `devtools::release()` para recordar mejor los pasos a seguir.
- Etiqueta cada versión en git después de cada envío a CRAN. Más información.
- A CRAN no le gustan las actualizaciones demasiado frecuentes. Dicho esto, si observas un problema importante una semana después de una publicación en CRAN, explícalo en el archivo cran-comments.md e intenta publicar una versión más reciente.

13.3 Archivo de noticias

Un archivo *NEWS* que describa los cambios asociados a cada versión facilita ver qué cambia en el paquete y cómo puede afectar los flujos de trabajo. Debes añadir uno para tu paquete, y hacerlo fácil de leer.

- Es obligatorio utilizar un `NEWS` o `NEWS.md` en la raíz de tu paquete. Recomendamos utilizar `NEWS.md` para que el archivo sea más navegable.
- Utiliza nuestro archivo *NEWS* de ejemplo como modelo. Puedes encontrar un buen archivo *NEWS* real en el repositorio del paquete `taxize` por ejemplo.
- Si utilizas `NEWS`, añádelo a `.Rbuildignore`, pero no si usas `NEWS.md`.
- Actualiza el archivo de noticias antes de cada envío a CRAN, con una sección con el nombre del paquete, la versión y la fecha de envío, como (como se ve en nuestro archivo *NEWS* de ejemplo):

```
foobar 0.2.0 (2016-04-01)
=========================
```

- Debajo de esa cabecera, agrega las secciones que sean necesarias, incluyendo "Nueva funcionalidad", "Mejoras menores", "Bugs resueltos", "Funciones obsoletas y caducas", "Mejoras en la documentación" y cualquier epígrafe especial que agrupe un gran número de cambios. Debajo de cada encabezado enumera los elementos según sea necesario (como se ve en nuestro archivo *NEWS* de ejemplo). Para cada elemento, proporciona una descripción de la nueva característica, mejora, error corregido o función/característica obsoleta. Enlaza a cualquier *issue* de GitHub relacionado como `(#12)`. El `(#12)` se reemplazará en GitHub con un enlace a ese *issue* en el repo.
- Después de haber añadido un `git tag` y lo hayas enviado a GitHub, añade las noticias de esa versión etiquetada a las notas de publicación de una versión en tu repo de GitHub con un título como `pkgname v0.1.0`. Consulta la documentación de GitHub sobre la creación de una publicación.
- Nuestro boletín de noticias describirá los paquetes con nuevas versiones en CRAN. Ver el siguiente capítulo sobre marketing para más ideas sobre cómo informar a más personas que podrían usar tu paquete sobre el lanzamiento.
- Para obtener más orientación sobre el archivo *NEWS*, te sugerimos que leas la guía de estilo de tidyverse.

14 Promocionando tu paquete

Te ayudaremos a promocionar tu paquete, pero aquí hay algunas cosas más para tener en cuenta.

- Si te enteras de un caso de uso de tu paquete, anima a quien lo escribió a publicar el enlace en nuestro foro de debate en la categoría Casos de uso, para recibir un toot (mensaje en Mastodon) de rOpenSci y su posible inclusión en el boletín mensual de rOpenSci. También te recomendamos que añadas un enlace al caso de uso en la sección "casos de uso reales" de tu archivo *README*.
- Cuando publiques una nueva versión de tu paquete o lo publiques en CRAN por primera vez,
- Comparte sobre el paquete en Mastodon utilizando el hashtag "#rstats" y etiqueta a rOpenSci! Esto puede ayudar llegar a gente que pueda colaborar o usar el paquete. Ejemplos enviados desde rOpenSci mismo: "A package a day" (un paquete al dia), Help wanted (se necesita ayuda), Use case (caso de uso), Welcome post (mensaje de bienvenida).
  - haz un *pull request* a R Weekly con una mensaje sobre la versión en la sección "*New Releases*" (o "*New Packages*" para la primera versión de GitHub/CRAN).
  - Comparte en tus redes sociales.
  - Considera enviar un breve artículo sobre el lanzamiento a rOpenSci tech notes. Ponte en contacto con la persona encargada de gestionar la comunidad de rOpenSci, (por ejemplo, a través de Slack o info@ropensci.org). Consulta las recomendaciones sobre cómo contribuir con un artículo en el blog).
  - Envía tu paquete a listas de paquetes como *CRAN Task Views*.
- Si decides promocionar tu paquete dando una charla sobre él en una reunión o conferencia (¡excelente idea!) lee en este artículo de Jenny Bryan y Mara Averick (en inglés).

15 Gestión de GitHub

Actualmente la gran mayoría de los paquetes de rOpenSci se desarrollan en GitHub. Aquí tienes algunos consejos para aprovechar la plataforma en una sección sobre hacer que tu repo sea más fácil de descubrir y una sección sobre publicitar tu propia cuenta de GitHub después de pasar por la revisión por pares.

15.1 Haz que tu repositorio sea fácil de descubrir

15.1.1 Áreas de repositorio en GitHub

Las áreas del repositorio de GitHub ayudan a navegar y buscar en los repositorios de GitHub, son usadas por R-universe en paginas de paquetes y para resultados de busquedas. También son usadas por `codemetar` en las palabras clave del registro de rOpenSci.

Recomendamos:

- Añadir “r”, “r-package” y “rstats” como áreas al repositorio de tus paquetes.
- Añadir cualquier otra área relevante al repositorio de tus paquetes.

Es posible que te hagamos sugerencias una vez que tu paquete esté incorporado.

15.1.2 Lingüista de GitHub

El lingüista de GitHub asignará un lenguaje a tu repo en función de los archivos que contenga. Algunos paquetes que contienen mucho código en C++ pueden ser clasificados como paquetes de C++ en lugar de R, lo cual está bien y muestra la necesidad de las áreas “r”, “r-package” y “rstats”.

Recomendamos anular el lingüista de GitHub añadiendo o modificando un archivo .gitattributes a tu repositorio en dos casos:

- Si almacenas archivos html en lugares no estándar (fuera en docs/, por ejemplo, en vignettes/) omite los archivos de documentación. Añade `*.html linguist-documentation=true` a .gitattributes (como en este ejemplo).

- Si tu repo contiene código que no es de tu autoría, por ejemplo, código JavaScript, añade `inst/js/* linguist-vendored` a .gitattributes (como en este ejemplo).

De este modo, la clasificación lingüística y las estadísticas de tu repositorio reflejarán mejor el código fuente que contiene, además de hacerlo más descubrible. En particular, si el lingüista no reconoce correctamente que tu repositorio contiene principalmente código R, tu paquete no aparecerá en los resultados de búsqueda con el filtro `language:R`. Del mismo modo, tu repositorio no podrá aparecer entre los repos de R populares.

Más información sobre las anulaciones del lingüista de GitHub aquí.

15.2 Publicita tu propia cuenta

- Como responsable de un paquete incorporado, ahora eres miembro de la organización GitHub "ropensci" de rOpenSci. Por defecto, la pertenencia a la organización es privada; consulta cómo hacerla pública en la documentación de GitHub.
- Incluso después de que el repositorio de tu paquete haya sido transferido a rOpenSci, puedes destacarlo en tu propia cuenta.
- En general, recomendamos añadir al menos una imagen de perfil (¡que no tiene por qué ser tu cara!) y tu nombre a tu perfil de GitHub.

16 Evolución de paquetes - cambiando cosas en tu paquete

Este capítulo presenta nuestra guía para realizar modificaciones a tu paquete: cambiar los nombres de los parámetros, cambiar los nombres de las funciones, eliminar funciones, e incluso retirar y archivar paquetes.

Este capítulo fue aportado inicialmente como nota técnica en el sitio web de rOpenSci por Scott Chamberlain; puedes leer la versión original en inglés aquí.

16.1 Filosofía de los cambios

Cada persona es libre de tener su propia opinión sobre cuán libremente se pueden cambiar los parámetros, funciones, etc. en un paquete; ni CRAN ni ningún otro sitio impone normas sobre esto. En general, a medida que un paquete madura los cambios en los métodos orientados al uso (es decir, las funciones exportadas en un paquete R) deberían ser muy infrecuentes. Los paquetes de los cuales dependen muchos otros paquetes probablemente sean más cuidadosos con los cambios, y deberían serlo.

16.2 El paquete lifecycle

En este capítulo se presentan soluciones que no requieren el paquete lifecycle, pero que pueden resultarte útiles. Te recomendamos leer la documentación de lifecycle.

16.3 Parámetros: cambiando los nombres de los parámetros

A veces hay que cambiar los nombres de los parámetros, ya sea en pos de mayor claridad u otras razones.

Un posible enfoque es comprobar si los argumentos obsoletos no faltan, y llamar a `stop()` con un mensaje significativo.

```
foo_bar <- function(x, y) {
    if (!missing(x)) {
        stop("utiliza 'y' en lugar de 'x'")
    }
    y^2
}

foo_bar(x = 5)
#> Error en foo_bar(x = 5) : utiliza 'y' en lugar de 'x'
```

Si quieres que sea más útil, podrías emitir una advertencia pero tomar automáticamente la acción necesaria

```
foo_bar <- function(x, y) {
    if (!missing(x)) {
        warning("utiliza 'y' en lugar de 'x'")
        y <- x
    }
    y^2
}

foo_bar(x = 5)
#> Warning message:
#> In foo_bar(x = 5) : utiliza 'y' en lugar de 'x'
#> 25
```

Ten cuidado con el parámetro `...`. Si tu función tiene `...` y ya has eliminado un parámetro (llamémoslo `z`), alguien puede tener un código más antiguo que utilice `z`. Cuando usen `z`, como no es un parámetro en la definición de la función, es probable que sea capturado por `...` y se ignore silenciosamente – no es lo que quieres. En su lugar, deja el argumento `z`, emitiendo un error si se utiliza.

16.4 Funciones: cambiando los nombres de las funciones

Si tienes que cambiar el nombre de una función, hazlo gradualmente, como con cualquier otro cambio en tu paquete.

Digamos que tienes una función `foo`.

```
foo <- function(x) x + 1
```

Pero quieres cambiar el nombre de la función a `bar`.

En lugar de simplemente cambiar el nombre de la función y que `foo` deje de existir directamente, en la primera versión del paquete donde `bar` aparezca, haz un alias como

```
#' foo - añadir 1 a un input
#' @export
foo <- function(x) x + 1

#' @export
#' @rdname foo
bar <- foo
```

Con la solución anterior, se pueden usar tanto `foo()` como `bar()` – cualquiera de los dos hará lo mismo, ya que son la misma función.

También es útil tener un mensaje, pero entonces sólo querrás emitirlo cuando se use la función que va a desaparecer, por ejemplo

```
#' foo - añadir 1 a un input
#' @export
foo <- function(x) {
    warning("por favor, utiliza bar() en lugar de foo()", call. = FALSE)
    bar(x)
}

#' @export
#' @rdname foo
bar <- function(x) x + 1
```

Después de que la versión del paquete con `foo` y `bar` haya sido usada durante un tiempo, en la siguiente versión puedes eliminar el antiguo nombre de la función (`foo`), y tener sólo `bar`.

```
#' bar - añadir 1 a un input
#' @export
bar <- function(x) x + 1
```

16.5 Funciones: obsoletas y caducas

Para eliminar una función de un paquete (digamos que el nombre de tu paquete es `holamundo`), puedes utilizar el siguiente protocolo

- Marca la función como obsoleta en la versión del paquete `x` (por ejemplo `v0.2.0`)

En la propia función, utiliza `.Deprecated()` para indicar cual es la nueva función que substituye la anterior.

```
foo <- function() {
    .Deprecated("bar")
}
```

Hay opciones en `.Deprecated` para especificar un nuevo nombre de función, así como un nuevo nombre de paquete, lo que tiene sentido cuando se mueven funciones a paquetes diferentes.

El mensaje que da `.Deprecated` es una advertencia, por lo que los usuarios pueden suprimirla con `suppressWarnings()` si lo desean.

Haz una página de manual para las funciones obsoletas por ejemplo:

```
#' Funciones obsoletas en holamundo
#'
#' Estas funciones aún funcionan pero serán eliminadas (obsoletas) en la próxima versión.
#'
#' \itemize {
#'   \item \code{\link{foo}}: Esta función está obsoleta y se eliminará en la
#'    próxima versión de este paquete.
#' }
#'
#' @name holamundo-deprecated
NULL
```

Esto crea una página de manual a la que los usuarios pueden acceder como ‘?holamundo-deprecated“‘ y verán en el índice de la documentación. Añade cualquier función a esta página según sea necesario, y quítala cuando una función caduque (ver más abajo).

- En la próxima versión (`v0.3.0`) puedes caducar la función (es decir, que desaparezca completamente del paquete, excepto por una página *man* con una nota sobre ella).

En la propia función, utiliza `.Defunct()` de esta manera:

```
foo <- function() {
    .Defunct("bar")
}
```

Observa que el mensaje en `.Defunct` es un error para que la función se detenga mientras que `.Deprecated` utiliza una advertencia que permite que la función continúe.

Además, es bueno añadir `...` a todas las funciones caducas para que si los usuarios pasan algún parámetro obtengan el mismo mensaje de caducidad en lugar de un error de `argumento no utilizado` de esta forma:

```
foo <- function(...) {
    .Defunct("bar")
}
```

Sin `...` da:

```
foo(x = 5)
#> Error en foo(x = 5) : argumento no utilizado (x = 5)
```

Y con `...` da:

```
foo(x = 5)
#> Error: 'foo' ha sido eliminada de este paquete
```

Haz una página de manual para las funciones caducas por ejemplo:

```
#' Funciones caducas en holamundo
#'
#' Estas funciones han sido removidas, ya no están disponibles.
#'
#' \itemize {
#'   \item \code{\link{foo}}:  Esta función ha sido removida
#' }
#'
#' @name holamundo-defunct
NULL
```

Esto crea una página de manual que se accede como ‘?holamundo-defunct“‘ y que aparece en el índice de documentación. Añade a esta página las funciones que necesites. Es probable que quieras mantener esta página indefinidamente.

16.5.1 Testeando las funciones obsoletas

No es necesario que cambies los test de las funciones obsoletas hasta que caduquen.

- Ten en cuenta cualquier cambio realizado en una función obsoleta. Además de utilizar `.Deprecated` dentro de la función, ¿has cambiado los parámetros en la función obsoleta, o has creado una nueva función que sustituye a la función obsoleta, etc.? Hay que testear esos cambios, si se han hecho.
- En relación con lo anterior, si a la función obsoleta se le cambia simplemente el nombre, tal vez se pueda testar que la función antigua y la nueva devuelven resultados idénticos.
- `suppressWarnings()` podría utilizarse para suprimir la advertencia emitida desde `.Deprecated`. Pero como las pruebas no están orientadas al uso, no es tan malo que las pruebas emitan advertencias, y la advertencia podría incluso utilizarse como recordatorio para quien mantiene el paquete.

Una vez que se caduca una función, sus tests se eliminan sin más.

16.6 Archivando paquetes

Por lo general, el software tiene una vida útil finita, y es posible que los paquetes deban ser archivados en algún momento. Los paquetes son archivados y transladados a una organización dedicada de GitHub, ropensci-archive. Antes de archivar, el contenido del archivo *README* debe trasladarse a una ubicación alternativa (como "README-OLD.md"), y sustituirse por un contenido mínimo que incluya algo como lo siguiente

```
# <nombre del paquete>

[![Estado del proyecto: sin soporte](https://www.repostatus.org/badges/latest/unsupported.svg
[![Etiqueta de revisión por pares](https://badges.ropensci.org/<issue_number>_status.svg)](ht

Este paquete ha sido archivado. El antiguo README está ahora en [README-OLD](<link-a-README-
```

La etiqueta del estado del repositorio debe ser "*unsupported*" (sin soporte) para los paquetes anteriormente publicados, o "*abandoned*" (abandonado) para los ex paquetes conceptuales o en proceso, en cuyo caso el código de la etiqueta anterior debe sustituirse por:

```
[![Estado del proyecto: abandonado](https://www.repostatus.org/badges/latest/abandoned.svg)]
```

Un ejemplo de un archivo *README* mínimo en un paquete archivado está en ropensci-archive/monkeylearn. Una vez que el *README* se ha copiado en otro lugar y se ha reducido a su forma mínima, hay que seguir los siguientes pasos

- ☐ Cierra los issues con una frase que explique la situación y enlace a esta guía.
- ☐ Archiva el repositorio en GitHub (la opción se encuentra en la configuración del repositorio).
- ☐ Transfiere el repositorio a ropensci-archive o solicita a un miembro del equipo de rOpenSci que lo haga (puedes enviar un correo electrónico a `info@ropensci.org`).

Los paquetes archivados pueden ser desarchivados si quien lo mantenía, o una nueva persona, deciden reanudar el mantenimiento. Para ello, ponte en contacto con rOpenSci y se transferirá el repo a la organización ropenscilabs.

17 Política de gestión de paquetes

Este capítulo resume la política de gestión propuesta para el mantenimiento continuo de los paquetes desarrollados como parte de las actividades de rOpenSci y/o bajo la organización ropensci en GitHub. Esta política de gestión pretende apoyar estos objetivos:

- Garantizar que los paquetes proporcionados por rOpenSci estén actualizados y sean de alta calidad.
- Proporcionar información clara sobre el estado de desarrollo y revisión de cualquier software en los repositorios de rOpenSci.
- Gestionar el esfuerzo de mantenimiento para el equipo de rOpenSci, quienes mantienen paquetes y quienes contribuyen voluntariamente.
- Proporcionar un mecanismo para el retiro gradual de paquetes, manteniendo al mismo tiempo las etiquetas de revisión por pares

Los elementos de infraestructura necesarios para la aplicación de esta política están, en algunos casos, parcialmente construidos y en otros casos aún no se han iniciado. Nuestro objetivo es adoptar esta política en parte para priorizar el trabajo en estos componentes.

17.1 El registro de paquetes

- El registro de paquetes rOpenSci es una lista centralizada de paquetes de R mantenidos o revisados por rOpenSci. Contiene metadatos esenciales de los paquetes, incluyendo el estado de desarrollo y revisión, y es la fuente de los datos mostrados en sitios web, etiquetas, etc. Esto permite mantener este listado independientemente de la infraestructura o plataforma de alojamiento de los paquetes.

17.2 Paquetes mantenidos por el equipo

Los paquetes mantenidos por el equipo son desarrollados y mantenidos por el equipo de rOpenSci como parte de los proyectos de rOpenSci. Estos paquetes pueden ser revisados por pares o no. Muchos son paquetes de infraestructura que están fuera del alcance de la revisión por pares.

- Los paquetes mantenidos por el equipo aparecerán en el registro con la etiqueta "staff_maintained" y aparecerán en la página web de paquetes de rOpenSci o en lugares similares con la etiqueta "staff-maintained".
- Estos paquetes se almacenarán en la organización de GitHub ropensci.
- Los paquetes mantenidos por el equipo serán comprobados, y su documentación sera generada por el sistema de rOpenSci. Este sistema no envía notificaciones pero guarda los resultados en forma de estado de *commit* en GitHub (marca de verificación roja o cruz roja).
- Cuando las comprobaciones de un paquete fallen, el equipo de rOpenSci se esforzará en corregir los problemas, priorizando los paquetes en función de la base de las personas que lo usan (descargas), las dependencias inversas o los objetivos estratégicos.
- Cada uno o dos años, rOpenSci revisará todos los paquetes que lleven más de un mes con comprobaciones fallidas para determinar si se transfieren a la organización "ropensci-archive".
- Los paquetes que fallan consistentemente y que no tienen un plan para volver al mantenimiento activo pasarán al estado de "archivados". Estos paquetes serán transferidos a la organización "ropensci-archive" y se les asignará la etiqueta "archived" en el registro. No se comprobarán en el sistema de rOpenSci.
- Los paquetes archivados no se mostrarán por defecto en la página web de paquetes. Una pestaña especial de páginas de paquetes los mostrará con `"type": "archived"`, hayan sido revisados por pares o mantenidos por el personal.
- Los paquetes archivados pueden desarchivarse cuando una nueva persona, o quien lo mantenía anteriormente, quiera revivirlo y solucionar los problemas. Para ello, ponte en contacto con rOpenSci. Se transferirán a la organización ropenscilabs.

17.3 Paquetes revisados por pares

Los paquetes revisados por pares son aquellos aportados a rOpenSci por la comunidad y que han pasado por la revisión por pares. Tienen que estar en el ámbito de aplicación en el momento del envío para ser revisados.

- Una vez aceptados, estos paquetes revisados por pares se transfieren desde el GitHub original a la organización de GitHub ropensci.
- Los paquetes revisados por pares estarán en el registro etiquetados como "peer-reviewed" y tendrán una etiqueta de revisado por pares en su archivo *README*.
- Los paquetes revisados por pares aparecerán en la página web de rOpenSci o lugares similares con la etiqueta "peer-reviewed".

- Los paquetes revisados por pares serán comprobados, y su documentación sera generada por el sistema de rOpenSci. Este sistema no envía notificaciones pero guarda los resultados en forma de estado de *commit* en GitHub (marca de verificación roja o cruz roja).
- Si los paquetes están en CRAN, quienes lo mantienen pueden subscribirse a la API de notificaciones de comprobaciones de CRAN.
- Cada uno o dos años, el equipo de rOpenSci revisará los paquetes que estén fallando o que han estado fallando durante largos períodos y contactará a quienes los mantienen para determinar los planes de mantenimiento y las actualizaciones previstas. En base a esto, rOpenSci puede optar por mantener el estado actual del paquete con la expectativa de actualizaciones, contribuir a los arreglos, ayudar a alguien más para mantenerlo, o transferir el paquete al estado "archived".
- En función de la base de personas que usan el paquete (medida por las descargas), las dependencias inversas u objetivos estratégicos de rOpenSci, el equipo de rOpenSci puede asistir a los paquetes que fallan a través de PRs revisados por quienes mantienen el paquete, o cambios directos (si quienes lo mantienen no responden por aproximadamente un mes). rOpenSci también proporcionará apoyo a pedido, tanto del equipo como de personas voluntarias de la comunidad, en función del tiempo disponible.
- Si quien mantienen un paquete no responde consultas por un mes, o si ésta persona lo pide, rOpenSci puede buscar alguien más para mantener determinados paquetes revisados por pares que considere que tienen un alto valor para la comunidad en función de la base de personas que lo usan (descargas), dependencias inversas o los objetivos estratégicos de rOpenSci.
- Cuando se archiven, estos paquetes pasarán de la organización "ropensci" de GitHub a la organización "ropensci-archive" (o a las cuentas de GitHub quienes los mantienen si así lo desean), siguiendo la guía de transferencia. Obtendrán la etiqueta "archived" en el registro. Conservarán las etiquetas e insignias de "peer-reviewed". No se comprobará en el sistema rOpenSci.
- Los paquetes archivados no se mostrarán por defecto en la página web de paquetes. Una pestaña especial de páginas de paquetes los mostrará con `"type": "archived"`, hayan sido revisados por pares o mantenidos por el equipo.

17.4 Paquetes heredados

Los paquetes "heredados" son paquetes ni creados ni mantenidos por el equipo de rOpenSci ni revisados por pares, pero que están bajo la organización de rOpenSci GitHub por razones históricas. (Antes de establecer el proceso de revisión por pares y su alcance, rOpenSci absorbía paquetes de la comunidad sin criterios bien definidos).

- rOpenSci transferirá los paquetes heredados de nuevo a las organizaciones y repositorios de quienes los mantienen, si lo desean. Caso contrario, serán transferidos al repositorio "ropensci-archive" siguiendo la guía de transferencia. Si los paquetes están en el ámbito de aplicación, rOpenSci les preguntará si desean someterlos al proceso de revisión del software.
- Los paquetes heredados no se incluirán en el registro de paquetes.
- Se pueden hacer excepciones para los paquetes que son partes vitales del ecosistema de paquetes de R y/o rOpenSci que son supervisados activamente por el equipo.

17.5 Paquetes incubados

Los paquetes "incubados" son paquetes en desarrollo creados por el equipo o miembros de la comunidad como parte de proyectos comunitarios, como los creados en las desconferencias

- Los paquetes incubados vivirán en la organización ropenscilabs.
- Los paquetes incubados aparecerán en el registro de paquetes con la etiqueta "incubator".
- Los paquetes incubados no aparecerán por defecto en el sitio web, pero las páginas de paquetes incluirán una pestaña de "paquetes experimentales".
- Los paquetes incubados serán comprobados, y su documentación sera generada por el sistema de rOpenSci. Este sistema no envía notificaciones pero guarda los resultados en forma de estado de *commit* en GitHub (marca de verificación roja o cruz roja). La documentación indicará claramente que el paquete es experimental.
- Semestral o anualmente, rOpenSci se pondrá en contacto con quienes mantienen los paquetes incubados con repositorios de al menos tres meses de antigüedad para consultar por el estado de desarrollo y las preferencias para la migración a revisión por pares, ropensci-archive, o a sus organizaciones. En base a la respuesta, el paquete se migrará inmediatamente, se iniciará la revisión por pares, o la migración se aplazará hasta la próxima revisión. Los paquetes incubados se migrarán a ropensci-archive por defecto después de un año, siguiendo la guía de transferencia.
- Los paquetes incubados archivados ganarán la etiqueta "archived".

17.5.1 Paquetes incubados que no son de R

- La organización incubator también incluirá proyectos que no sean paquetes de R.
- Estos proyectos no aparecerán en el registro ni en una página web y no se comprobarán automáticamente.

- La política de migración será la misma que para los paquetes R, con ubicaciones de migración adecuadas (por ejemplo, “ropensci-books”).
- Si se archivan, los paquetes que no son de R se trasladarán a “ropensci-archive” siguiendo la guía de transferencia.

17.6 Libros

Los libros de rOpenSci son documentación larga, a menudo con formato de libro, relacionada con los paquetes, proyectos o temas de rOpenSci, creados por el personal de rOpenSci o por miembros de la comunidad.

- Los libros vivirán en la organización ropensci-books.
- Los libros se alojarán en books.ropensci.org.
- Los libros pueden estar maduros o en desarrollo, pero deben tener un mínimo de contenido antes de migrar a ropensci-books (por ejemplo desde ropenscilabs).
- La autoría y el estado de desarrollo de un libro deben estar claramente descritos en su página de inicio y en el archivo README.
- rOpenSci puede proporcionar etiquetas o plantillas (por ejemplo, “En desarrollo”, “Mantenido por la comunidad”) para utilizarse en las páginas de inicio de los libros en el futuro.

18 Guía de contribución

Este capítulo describe nuestra Guía de contribución, la cual indica cómo puedes contribuir al proyecto rOpenSci, ya sea con código o no.

¿Quieres contribuir a rOpenSci? ¡Fantástico! Hemos desarrollado la Guía de Contribución a la Comunidad rOpenSci para darte la bienvenida a rOpenSci y animarte a colaborar. Te ayudará a averiguar lo que podrías ganar aportando tu tiempo, tus conocimientos y tu experiencia, a hacer que tus necesidades coincidan con las cosas que ayudarán a la misión de rOpenSci y a ponerte en contacto con los recursos que te ayudarán en el camino.

Nuestro equipo y comunidad fomentan activamente un entorno acogedor donde personas de diferentes orígenes y niveles de habilidad aprenden, comparten ideas e innovan juntos abiertamente a través de normas y software compartidos, ya sea que usen o que desarrollen software. La participación en todas las actividades de rOpenSci está respaldada por nuestro Código de Conducta.

Aceptamos contribuciones de código y de otros tipos de personas con o sin experiencia, en cualquier etapa de su carrera, y en cualquier sector. ¡No es necesario que desarrolles software! Tal vez quieras dedicar **30 minutos** compartiendo el caso de uso de tu paquete en nuestro foro público o informando de un error, **una hora** aprendiendo en una Llamada Comunitaria, **cinco horas** revisando un paquete de R enviado para una revisión abierta por pares, **o tal vez quieras comprometerte de forma continuada** para ayudar a mantener un paquete.

¿Cuáles son algunos de los beneficios de contribuir?

- Conectar con una comunidad que comparte tu interés por hacer la ciencia más abierta.
- Aprender de personas fuera de tu área de trabajo que enfrentan desafíos similares a los tuyos con R.
- Plantear y responder a nuevas preguntas de investigación aprendiendo sobre nuevas herramientas de software.
- Ganar confianza y recibir apoyo en tus esfuerzos por escribir código y desarrollar software.
- Ganar visibilidad con tu trabajo de código abierto.
- Mejorar el software que utilizas o construyes.
- Mejorar tus conocimientos de R y ayudar a otras personas a mejorar los suyos.
- Mejorar tus habilidades de escritura.
- Conseguir una mayor exposición para tu paquete.

Consulta nuestra Guía de Contribución y explora la sección *What brings you here?* (¿Qué te trae por aquí?) para encontrar qué declaraciones *Quiero …* te representan mejor y elegir tu camino.
Para ayudar a que te reconozcas en estas declaraciones, las hemos agrupado en: Descubrir, Conectar, Aprender, Construir y Ayudar. Para cada categoría, enumeramos ejemplos de cómo podrían ser esas contribuciones y enlazamos a nuestros recursos para obtener los detalles que necesitas.

Parte IV

Apéndice

19 NEWS

19.1 0.9.0

- 2024-01-09, update roxygen2 wording (`@vincentvanhees`, #792).
- 2023-12-15, update roxygen2 advice, mainly linking to roxygen2 website (#750).
- 2023-09-15, add suggestions for API packages (#496).
- Translation to Spanish!
- 2023-07-17, Update Aims and Scope to include translation packages, remove experimental text-processing categories, and provide clarifications around API wrappers
- 2023-05-04, Added link to Bioconductor book (#663, `@llrs`).
- 2023-04-26, Changed suggested lifecycle stage in authors guide (#661, `@bart1`).
- 2023-04-25, changed the COI section to use parallel construction (#659, `@eliocamp`).
- 2022-07-04, Add resources around GitHub workflows (#479, `@maurolepore`).
- 2023-02-14, update instructions for CITATION to reflect new CRAN policies (#604, #609).
- 2023-02-14, add package maintainer cheatsheet (#608).
- 2023-01-25, add Mastodon as social media (#592, by @yabellini).
- 2023-01-25, add Mastodon as social media (#592, by `@yabellini`).
- 2023-01-20, fix small formating error (#590 by `@eliocamp`).
- 2022-11-22, mention shinytest2 near shinytest.
- 2022-09-20, add editor instruction to add “stats” label to stats submissions
- 2022-09-20, fixed link to reviewer approval template (#548), and rendering of editor’s template (#547)
- 2022-08-23, add recommendation to document argument default (`@Bisaloo`, #501)
- 2022-08-06, fix link to R Packages book (#498)
- 2022-07-21, mention GitHub Discussions and GitHub issue templates. (#482)

- 2022-07-21, highlight values for reviewing in more places (#481)
- 2022-07-20, Explanation of package submission via non-default branches (#485), added `@s3alfisc` to contributor list.
- 2022-07-20, add how to volunteer as a reviewer (#457).
- 2022-06-23, Expanded explanation of Codecov, added `@ewallace` to contributor list (#484)

19.2 0.8.0

- 2022-06-03, Remove former references to now-archived "rodev" package
- 2022-05-30, Advise that reviewers can also directly call `@ropensci-review-bot check package`
- 2022-05-27, Add Mark Padgham to list of authors
- 2022-05-27, Add `devguider::prerelease_checklist` item to pre-release template (#463)
- 2022-05-13, Align version number in DESCRIPTION file with actual version (#443)
- 2022-05-13, Update guidelines for CONTRIBUTING.md (#366, #462)
- 2022-05-09, Add section on authorship of included code, thanks to `@KlausVigo` (#388).
- 2022-05-09, Remove mention of 'rev' role requiring R v3.5
- 2022-05-05, Move all scripts from local `inst` directory to `ropensci-org/devguider` pkg.
- 2022-05-03, Update package archiving guidance to reduce README to minimal form.
- 2022-04-29, Advise that authors can directly call `@ropensci-review-bot check package`.
- 2022-04-29, Describe `pkgcheck-action` in CI section.
- 2022-04-29, Update scope in policies section to include statistical software.
- 2022-04-29, Add `prelease.R` script to open pre-release GitHub issue & ref in appendix.
- 2022-04-26, Add GitHub 2FA recommendation to package security.
- 2022-03-29, Remove references to Stef Butland, former community manager.
- 2022-03-28, Add comments on submission planning about time commitment.
- 2022-03-24, Remove approval comment template (coz it's automatically generated by the bot now).
- 2022-03-21, rephrase CITATION guidance to make it less strict. Also mentions CITATION.cff and the cffr package.

- 2022-03-08, add links to blogs related to package development (#389).
- 2022-02-17, update redirect instructions (`@peterdesmet`, #387).
- 2022-02-14, link to Michael Lynch's post Why Good Developers Write Bad Unit Tests.
- 2022-02-14, mention more packages for testing like dittodb, vcr, httptest, httptest2, webfakes.
- 2022-01-10, make review templates R Markdown files (`@Bisaloo`, #340).
- 2022-01-14, update guidance on CI services (#377)
- 2022-01-11, update guidance around branches, with resources suggested by `@ha0ye` and `@statnmap`.
- 2022-01-10, divide author's guide into sub-sections, and add extra info including `pkgcheck`.
- 2021-11-30, adds links to examples of reviews, especially tough but constructive ones (with help from `@noamross`, `@mpadge`, #363).
- 2021-11-19, add recommended spatial packages to scaffolding section (software-review-meta#47)
- 2021-11-18, update advice on grouping functions for pkgdown output (#361)

19.3 0.7.0

- 2021-11-04, add mentions of stat software review to software review intro and to the first book page (#342).
- 2021-11-04, mention pkgcheck in the author guide (@mpadge, #343).
- 2021-11-04, add editors' responsibilities including Editor etiquette for commenting on packages on which you aren't handling/reviewing (@jhollist, #354).
- 2021-11-04, give precise examples of tools for installation instructions (remotes, pak, R-universe).
- 2021-11-04, add more bot guidance (less work for editors).
- 2021-10-07, add guidance for editorial management (recruiting, inviting, onboarding, offboarding editors).
- 2021-09-14, add a requirement that there is at least one *HTML* vignette.
- 2021-09-03, add some recommendations around git. (@annakrystalli, #341)
- 2021-07-14, clarify the categories data extraction and munging by adding examples. (@noamross, #337)

- 2021-05-20, add guidance around setting up your package to foster a community, inspired by the recent rOpenSci community call. (with help from `@Bisaloo`, #289, #308)
- 2021-04-27, no longer ask reviewers to ask covr as it'll be done by automatic tools, but ask them to pay attention to tests skipped.
- 2021-04-02, add citation guidance.
- 2021-04-02, stop asking reviewers to run goodpractice as this is part of editorial checks.
- 2021-03-23, launched a new form for reviewer volunteering.
- 2021-02-24, add guidance around the use of `@ropensci-review-bot`.

19.4 0.6.0

- 2021-02-04, add guidance to enforce package versioning and tracking of changes through review (`@annakrystalli`, #305)
- 2021-01-25, add a translation of the review template in Spanish (`@Fvd`, `@maurolepore`, #303)
- 2021-01-25, the book has now better citation guidance in case you want to cite this very guide (`@Bisaloo`, #304).
- 2021-01-12, add some more guidance on escaping examples (#290).
- 2021-01-12, mention the lifecycle package in the chapter about package evolution (#287).
- 2021-01-12, require overlap information is put in documentation (#292).
- 2021-01-12, start using the bookdown::bs4_book() template.
- 2021-01-12, add a sentence about whether it is acceptable to push a new version of a package to CRAN within two weeks of the most recent version if you have just been made aware of, and fixed, a major bug (`@sckott`, #283)
- 2021-01-12, mention the HTTP testing in R book.
- 2021-01-12, mention testthat snapshot tests.
- 2021-01-12, remove mentions of Travis CI and link to Jeroen Ooms' blog post about moving away from Travis.
- 2021-01-12, update the package curation policy: mention a possible exception for legacy packages that are vital parts of the R and/or rOpenSci package ecosystem which are actively monitored by staff. (`@noamross`, #293)

19.5 0.5.0

- 2020-10-08, add help about link checking (@sckott, #281)
- 2020-10-08, update JOSS instructions (@karthik, #276)
- 2020-10-05, add links to licence resources (@annakrystalli, #279)
- 2020-10-05, update information about the contributing guide (@stefaniebutland, #280)
- 2020-09-11, make reviewer approval a separate template (@bisaloo, #264)
- 2020-09-22, add package curation policy (@noamross, #263)
- 2020-09-11, add more guidance and requirements for docs at submission (@annakrystalli, #261)
- 2020-09-14, add more guidance on describing data source in DESCRIPTION (@mpadge, #260)
- 2020-09-14, add more guidance about tests of deprecated functions (@sckott, #213)
- 2020-09-11, update the CI guidance (@bisaloo, @mcguinlu, #269)
- 2020-09-11, improve the redirect guidance (@jeroen, @mcguinlu, #269)

19.6 0.4.0

- 2020-04-02, give less confusing code of conduct guidance: the reviewed packages' COC is rOpenSci COC (@Bisaloo, @cboettig, #240)
- 2020-03-27, add section on Ethics, Data Privacy and Human Subjects Research to Policies chapter
- 2020-03-12, mention GitHub Actions as a CI provider.
- 2020-02-24, add guide for inviting a guest editor.
- 2020-02-14, add mentions of the ropensci-books GitHub organisation and associated subdomain.
- 2020-02-10, add field and laboratory reproducibility tools as a category in scope.
- 2020-02-10, add more guidance about secrets and package development in the security chapter.
- 2020-02-06, add guidance about Bioconductor dependencies (#246).
- 2020-02-06, add package logo guidance (#217).
- 2020-02-06, add one CRAN gotcha: single quoting software names(#245, @aaronwolen)

- 2020-02-06, improve guidance regarding the replacement of “older” pkgdown website links and source (#241, @cboettig)
- 2020-02-06, rephrase the EiC role (#244).
- 2020-02-06, remove the recommendation to add rOpenSci footer (https://github.com/ropensci/software-review-meta/issues/79).
- 2020-02-06, remove the recommendation to add a review mention to DESCRIPTION but recommends mentioning the package version when reviewers are added as “rev” authors.
- 2020-01-30, slightly changes the advice on documentation re-use: add a con; mention `@includeRmd` and `@example`; correct the location of Rmd fragments (#230).
- 2020-01-30, add more guidance for the editor in charge of a dev guide release (#196, #205).
- 2020-01-22, add guidance in the editor guide about not transferred repositories.
- 2020-01-22, clarify forum guidance (for use cases and in general).
- 2020-01-22, mention an approach for pre-computing vignettes so that the pkgdown website might get build on rOpenSci docs server.
- 2020-01-22, document the use of mathjax with rotemplate (@Bisaloo, #199).
- 2020-01-20, add guidance for off-thread interaction and COIs (@noamross, #197).
- 2020-01-20, add advice on specifying dependency minimum versions (@karthik, @annakrystalli, #185).
- 2020-01-09, start using GitHub actions instead of Travis for deployment.
- -2019-12-11, add note in Documentation sub-section of Packaging Guide section about referencing the new R6 support in roxygen2 (ropensci/dev_guide#189)
- 2019-12-11, add new CRAN gotcha about having ‘in R’ or ‘with R’ in your package title (@bisaloo, ropensci/dev_guide#221)

19.7 0.3.0

- 2019-10-03, include in the approval template that maintainers should include link to the docs.ropensci.org/pkg site (ropensci/dev_guide#191)
- 2019-09-26, add instructions for handling editors to nominate packages for blog posts (ropensci/dev_guide#180)
- 2019-09-26, add chapter on changing package maintainers (ropensci/dev_guide#128) (ropensci/dev_guide#194)

- 2019-09-26, update Slack room to use for editors (ropensci/dev_guide#193)
- 2019-09-11, update instructions in README for rendering the book locally (ropensci/dev_guide#192)
- 2019-08-05, update JOSS submission instructions (ropensci/dev_guide#187)
- 2019-07-22, break "reproducibility" category in policies into component parts. (ropensci/software-review-meta#81)
- 2019-06-18, add link to rOpenSci community call "Security for R" to security chapter.
- 2019-06-17, fix formatting of Appendices B-D in the pdf version of the book (bug report by `@IndrajeetPatil`, #179)
- 2019-06-17, add suggestion to use R Markdown hunks approach when the README and the vignette share content. (ropensci/dev_guide#161)
- 2019-06-17, add mention of central building of documentation websites.
- 2019-06-13, add explanations of CRAN checks. (ropensci/dev_guide#177)
- 2019-06-13, add mentions of the `rodev` helper functions where relevant.
- 2019-06-13, add recommendation about using `cat` for `str.*()` methods. RStudio assumes that `str` uses `cat`, if not when loading an R object the `str` prints to the console in RStudio and doesn't show the correct object structure in the properties. ([`@mattfidler`] (https://github.com/mattfidler/) #178)
- 2019-06-12, add more details about git flow.
- 2019-06-12, remove recommendation about `roxygen2` dev version since the latest stable version has what is needed. (`@bisaloo`, #165)
- 2019-06-11, add mention of usethis functions for adding testing or vignette infrastructure in the part about dependencies in the package building guide.
- 2019-06-10, use the new URL for the dev guide, https://devguide.ropensci.org/
- 2019-05-27, add more info about the importance of the repo being recognized as a R package by linguist (`@bisaloo`, #172)
- 2019-05-22, update all links eligible to HTTPS and update links to the latest versions of Hadley Wickham and Jenny Bryan's books (`@bisaloo`, #167)
- 2019-05-15, add book release guidance for editors. (ropensci/dev_guide#152)

19.8 0.2.0

- 2019-05-23, add CRAN gotcha: in the Description field of your DESCRIPTION file, enclose URLs in angle brackets.
- 2019-05-13, add more content to the chapter about contributing.
- 2019-05-13, add more precise instructions about blog posts to approval template for editors.
- 2019-05-13, add policies allowing using either <- or = within a package as long as the whole package is consistent.
- 2019-05-13, add request for people to tell us if they use our standards/checklists when reviewing software elsewhere.
- 2019-04-29, add requirement and advice on testing packages using `devel` and `oldrel` R versions on Travis.
- 2019-04-23, add a sentence about why being generous with attributions and more info about ctb vs aut.
- 2019-04-23, add link to Daniel Nüst's notes about migration from XML to xml2.
- 2019-04-22, add use of rOpenSci forum to maintenance section.
- 2019-04-22, ask reviewer for consent to be added to DESCRIPTION in review template.
- 2019-04-22, use a darker blue for links (feedback by `@kwstat`, #138).
- 2019-04-22, add book cover.
- 2019-04-08, improve formatting and link text in README (`@katrinleinweber`, #137)
- 2019-03-25, add favicon (`@wlandau`, #136).
- 2019-03-21, improve Travis CI guidance, including link to examples. (`@mpadge`, #135)
- 2019-02-07, simplify code examples in Package Evolution section (maintenance_evolution.Rmd file) (`@hadley`, #129).
- 2019-02-07, added a PDF file to export (request by `@IndrajeetPatil`, #131).

19.9 0.1.5

- 2019-02-01, created a .zenodo.json to explicitly set editors as authors.

19.10 First release 0.1.0

- 2019-01-23, add details about requirements for packages running on all major platforms and added new section to package categories.
- 2019-01-22, add details to the guide for authors about the development stage at which to submit a package.
- 2018-12-21, inclusion of an explicit policy for conflict of interest (for reviewers and editors).
- 2018-12-18, added more guidance for editor on how to look for reviewers.
- 2018-12-04, onboarding was renamed Software Peer Review.

19.11 place-holder 0.0.1

- Added a `NEWS.md` file to track changes to the book.

20 Plantilla de revisión

Puedes guardar esto como un archivo R Markdown, o eliminar el YAML y guardarlo como un archivo Markdown.

20.1 Package Review

Please check off boxes as applicable, and elaborate in comments below. Your review is not limited to these topics, as described in the reviewer guide

- **Briefly describe any working relationship you have (had) with the package authors.**

☐ As the reviewer I confirm that there are no conflicts of interest for me to review this work (if you are unsure whether you are in conflict, please speak to your editor *before* starting your review).

20.1.0.1 Documentation

The package includes all the following forms of documentation:

☐ **A statement of need:** clearly stating problems the software is designed to solve and its target audience in README
☐ **Installation instructions:** for the development version of package and any non-standard dependencies in README
☐ **Vignette(s):** demonstrating major functionality that runs successfully locally
☐ **Function Documentation:** for all exported functions
☐ **Examples:** (that run successfully locally) for all exported functions
☐ **Community guidelines:** including contribution guidelines in the README or CONTRIBUTING, and DESCRIPTION with `URL`, `BugReports` and `Maintainer` (which may be autogenerated via `Authors@R`).

20.1.0.2 Functionality

- ☐ **Installation:** Installation succeeds as documented.
- ☐ **Functionality:** Any functional claims of the software have been confirmed.
- ☐ **Performance:** Any performance claims of the software have been confirmed.
- ☐ **Automated tests:** Unit tests cover essential functions of the package and a reasonable range of inputs and conditions. All tests pass on the local machine.
- ☐ **Packaging guidelines**: The package conforms to the rOpenSci packaging guidelines.

Estimated hours spent reviewing:

- ☐ Should the author(s) deem it appropriate, I agree to be acknowledged as a package reviewer ("rev" role) in the package DESCRIPTION file.

20.1.1 Review Comments

21 Plantilla para quien hace la edición

21.0.1 Quien hace la edición debe comprobar:

- ☐ **Documentación:** El paquete tiene suficiente documentación disponible en línea (*README*, documentación de *pkgdown*) que permita una evaluación de la funcionalidad y el alcance sin instalar el paquete. En concreto,
  - ☐ ¿Está bien explicada la necesidad del paquete?
  - ☐ ¿La página con el índice de funciones es clara (agrupada por temas si es necesario)?
  - ☐ ¿Las viñetas son legibles, suficientemente detalladas y no superficiales?
- ☐ **Encaja:** El paquete cumple los criterios para ser incluido en una categoría y no hay solapamiento.
- ☐ **Instrucciones de instalación:** ¿Las instrucciones de instalación son lo suficientemente claras?
- ☐ **Tests:** Si el paquete provee herramientas interactivas, usa recursos HTTP, produce gráficos, etc., ¿los *tests* son consistentes con el estado del arte?
- ☐ **Información para contribuir:** ¿La documentación acerca de cómo contribuir es lo suficientemente clara, por ejemplo, los *tokens* para *tests*, espacios para pruebas?
- ☐ **Licencia:** El paquete tiene una licencia aceptada por CRAN u OSI.

•

21.1 [ ] Gestión del proyecto: ¿Los *issues* y *pull requests* se encuentran en buen estado? Por ejemplo, ¿hay errores pendientes, está claro cuándo se abordarán los pedidos de nueva funcionalidad?

21.1.0.1 Comentarios de edición

---

22 Modelo de pedido de revisión

Quienes estén a cargo de la edición pueden utilizar la siguiente plantilla de correo electrónico para convocar personas para realizar la revisión.

Estimado/a [REVISOR/A]

Hola, soy [EDITOR/A]. [COMENTARIO AMISTOSO]. Te escribo para preguntarte si te interesa revisar un paquete para rOpenSci. Como probablemente sepas, rOpenSci lleva a cabo la revisión por pares de los paquetes de R aportados a nuestra colección de forma similar a las revistas científicas.

El paquete, [PAQUETE] de [AUTOR(ES)], realiza [FUNCIÓN]. Puedes encontrarlo en GitHub aquí: [ENLACE REPO]. Hacemos nuestro proceso de revisión abierta a través de GitHub, aquí: [ISSUE DE REVISIÓN]

Si aceptas, ten en cuenta que te pediremos completes las revisiones en tres semanas. (Hemos comprobado que lleva un tiempo similar revisar un paquete que un artículo académico).

Nuestra guía de revisión detalla lo que buscamos en la revisión de un paquete, e incluye enlaces a ejemplos de revisiones. Nuestras normas se detallan en nuestra guía de empaquetado y proporcionamos una plantilla para que la utilices en revisión. Por favor, asegúrate de que no tienes un conflicto de intereses que te impida revisar este paquete. Si tienes alguna pregunta o comentario, no dudes en preguntarme o enviar un mensaje a la sección del foro de rOpenSci.

La comunidad de rOpenSci es nuestro mejor activo. Nuestro objetivo es que las revisiones sean abiertas, no conflictivas y centradas en mejorar la calidad del software. ¡Sé amable! Consulta nuestra guía de revisión y el código de conducta para más información.

[SI SE SOLICITÓ TUTORÍA: Has indicado en tu formulario que te gustaría tener tutoría para tu primera revisión.
Sientete libre de contactarme cuando lo necesites durante este proceso, lo que incluye hacer preguntas por correo electrónico y Slack (recibirás una invitación al Slack de rOpenSci), y compartir borradores de revisiones para recibir devoluciones antes de publicarlas. También me encantará hacer una breve videollamada para explicar el proceso. Por favor, házmelo saber en tu respuesta si quieres programar una.]

¿Puedes revisar? Si no puedes, siempre nos sirve si puedes sugerir a otras personas. Si no tengo noticias tuyas en el plazo de una semana, supondré que no puedes hacer una revisión en este momento.

Gracias por tu tiempo.

Atentamente,

[EDITOR/A]

23 Plantilla de comentarios de aprobación para revisión

23.1 Respuesta de quien hizo la revisión

23.1.0.1 Aprobación final (posterior a la revisión)

☐ **Quien mantiene el paquete ha respondido a mi revisión y ha realizado cambios satisfactorios. Recomiendo aprobar este paquete.**

Estimación de horas dedicadas a la revisión:

24 Plantilla de NEWS

```
foobar 0.2.0 (2016-04-01)
=========================

### NUEVA FUNCIONALIDAD

  * Nueva función `hacer_algo()` para hacer cosas. (#5)

### MEJORAS MENORES

  * Documentación mejorada para `cosas()`. (#4)

### ERRORES CORREGIDOS

  * Corrige un error en `algo()`. (#3)

### OBSOLETO Y CADUCADO

  * `hola_mundo()` ahora es una función obsoleta y se eliminará en un en una versión futura,

### CORRECCIONES EN LA DOCUMENTACIÓN

  * Se aclaró el papel de `hola_marte()` vs. `adios_marte()`.

### (algo especial: cualquier epígrafe que agrupe un gran número de cambios bajo un mismo ter

  * blablabla.

foobar 0.1.0 (2016-01-01)
=========================

### NUEVA FUNCIONALIDAD

  * publicado en CRAN.
```

25 Guía para publicar el libro

Antes de realizar una nueva publicación de este libro, se puede ejecutar el script `prelease.R` que se encuentra en la carpeta `inst` de este repositorio. Esto va a crear una *issue* en GitHub con una lista de todas las *issues* asignadas al *milestone* de la próxima versión junto con la siguiente lista de tareas. Antes de correr el script, por favor revistar la asignación de *issues* al *milestone*. Todo esto debería hacerse un mes antes de la fecha de publicación programada.

25.1 Publicación del libro, versión

25.1.1 Mantenimiento de repositorios entre versiones

- ☐ Mira los *issues* de la guía de desarrollo y del meta de revisión del software para revisar los cambios pendientes en la guía de desarrollo. Asigna los *issues* de la guía de desarrollo a los *milestones* correspondientes a cada versión, ya sea la siguiente o la versión que viene luego de esa, por ejemplo versión 0.3.0. Fomenta la creación de PR y haz que sean revisados.

25.1.2 1 mes antes de la publicación

- ☐ Recuerda al equipo de edición que abran *issues* o PRs sobre los temas que quieran incluir en la próxima versión.
- ☐ Ejecuta la función `devguide_prerelease()` del paquete `devguider`.
- ☐ Pide al equipo de edición cualquier devolución que necesites antes de la publicación.
- ☐ Por cada contribución o cambio, asegúrate de que se hayan actualizado las *NEWS* en Appendix.Rmd.
- ☐ Coordina una fecha para la publicación con quien sea Community Manager de rOpenSci, quien te dará una fecha para publicar un artículo en el blog o nota técnica.

25.1.3 2 semanas antes de publicar

- ☐ Redacta un artículo para el blog o nota técnica sobre el lanzamiento con suficiente antelación para que el equipo de edición, y luego quien sea Community Manager puedan revisarlo (2 semanas). Ejemplo, Instrucciones generales para artículos de blog, instrucciones específicas para articulos de lanzamiento.
- ☐ Haz un PR de la *branch* dev a la *branch* master, notifica al equipo de edición en GitHub y Slack. Menciona el borrador del artículo del blog en un comentario en ese PR.

25.1.4 Publicación

- ☐ Comprueba las URL utilizando la función `devguide_urls()` del paquete {devguider}.
- ☐ Comprueba la ortografía con la función `devguide_spelling()` del paquete {devguider}. Actualiza la LISTA DE PALABRAS según sea necesario.
- ☐ Unifica los *commits* (*squash*) y haz *merge* del PR de *dev* a *master*.
- ☐ Crea un *release* en GitHub, asegúrate de que se cree un *release* en Zenodo.
- ☐ Re-genera el libro (para actualizar los metadatos de Zenodo en el libro) o espera a la compilación diaria.
- ☐ Vuelve a crear la *branch* dev.
- ☐ Termina el PR de tu artículo para el blog o nota técnica. Indica los aspectos más importantes que deben destacarse en los tweets como parte de la discusión del PR.

26 Cómo configurar una redirección

26.1 Si el sitio no está en GitHub pages (p.e. Netlify)

Reemplaza el contenido del sitio web actual con sendos archivos `index.html` y `404.html`. Ambos deben contener:

```
<html>
<head>
<meta http-equiv="refresh" content="0;URL=https://docs.ropensci.org/<nombrepaquete>/">
</head>
</html>
```

26.2 Si el sitio está en GitHub pages

Puedes configurar la redirección desde tu repositorio principal de GitHub Pages.

- Crea un nuevo repositorio (si no tienes uno). `https://github.com/<usuario>/<usuario>.github.io`.
- En este repositorio, crea un directorio llamado `<nombrepaquete>` con dos archivos: `index.html` y `404.html`. Ambos deben redireccionar a la nueva ubicación según lo visto en la sección anterior.
- Comprueba que `https://<usuario>.github.io/<nombrepaquete>/index.html` redirecciona a la dirección correcta.

27 Comandos del bot

27.1 Para todo el mundo

Ten en cuenta que limpiamos los *issues* eliminando el contenido superfluo, por lo que la mayoría de las veces el registro de que has pedido ayuda a un bot se borrará u ocultará rápidamente.

27.1.1 Consulta la lista de comandos disponibles

Si necesitas un recordatorio rápido

```
@ropensci-review-bot help
```

27.1.2 Ver el código de conducta

```
@ropensci-review-bot code of conduct
```

27.2 Para las personas responsables del paquete

27.2.1 Comprueba el paquete con pkgcheck

Cuando tu paquete haya cambiado sustancialmente.

```
@ropensci-review-bot check package
```

27.2.2 Envía tu respuesta sobre la revisión

Para registrar tu respuesta al equipo revisor.

```
@ropensci-review-bot submit response <response-url>
```

donde `<response_url>` es el enlace al comentario de respuesta en el *issue*.

27.2.3 Finalizar la transferencia del repositorio

Una vez que hayas aceptado la invitación a la organización GitHub de rOpenSci y le hayas transferido tu repositorio GitHub, ejecuta este comando para recuperar el acceso de administración a tu repositorio.

```
@ropensci-review-bot finalize transfer of <package-name>
```

27.2.4 Obtener una nueva invitación tras la aprobación

Si se venció el plazo de una semana para aceptar la invitación a la organización ropensci en GitHub, ejecuta este comando para recibir una nueva.

```
@ropensci-review-bot invite me to ropensci/<package-name>
```

27.3 Para la persona encargada de la edición

27.3.1 Asignar la persona para editar esta revisión

```
@ropensci-review-bot assign @username as editor
```

27.3.2 Poner el envío en espera

Ver política editorial.

```
@ropensci-review-bot put on hold
```

27.3.3 Indicar que el envío está fuera de alcance

No olvides publicar primero un comentario explicando la decisión y agradeciendo a las personas responsables del paquete su envío.

```
@ropensci-review-bot out-of-scope
```

27.4 Para la persona asignada como responsable de la edición

27.4.1 Poner el envío en espera

Ver política editorial.

```
@ropensci-review-bot put on hold
```

27.4.2 Comprueba el paquete con pkgcheck

Generalmente se hace sólo en consultas previas al envío para revisión, o cuando las personas responsables del paquete indican que el paquete ha cambiado sustancialmente.

```
@ropensci-review-bot check package
```

27.4.3 Comprueba las normas estadísticas

Generalmente se hace sólo en consultas previas al envío para revisión, o cuando las personas responsables del paquete indican que el paquete ha cambiado sustancialmente.

```
@ropensci-review-bot check srr
```

27.4.4 Comprueba que el README tiene la etiqueta de revisión de software

Hacia el final del proceso de envío.

```
@ropensci-review-bot check readme
```

27.4.5 Indica que estás buscando personas para revisar

```
@ropensci-review-bot seeking reviewers
```

27.4.6 Asignar una persona al equipo revisor

```
@ropensci-review-bot assign @username as reviewer
```

o

```
@ropensci-review-bot add @username as reviewer
```

27.4.7 Eliminar una persona del equipo revisor

```
@ropensci-review-bot remove @username from reviewers
```

27.4.8 Ajustar la fecha límite de la revisión

```
@ropensci-review-bot set due date for @username to YYYY-MM-DD
```

27.4.9 Registra que se ha enviado una revisión

```
@ropensci-review-bot submit review <review-url> time <time in hours>
```

27.4.10 Aprobar un paquete

```
@ropensci-review-bot approve <package-name>
```