Pacotes rOpenSci: Desenvolvimento, manutenção e revisão por pares

Equipe editorial para a revisão de software rOpenSci (atual e anterior)

Índice

rOpenSci - Guia para desenvolvedores **1**

Prefácio **3**

I Construindo seu pacote **5**

1 Guia de desenvolvimento de pacotes **7**

1.1 Nome do pacote e metadados 7

1.1.1 Como nomear seu pacote 7

1.2 Plataformas 8

1.3 API do pacote 8

1.3.1 Nomeando funções e argumentos 8

1.3.2 Mensagens do console 9

1.3.3 Interfaces interativas/gráficas 10

1.3.4 Verificação de entrada 10

1.3.5 Pacotes que envolvem recursos da Web (clientes de API) . . 10

1.3.6 Pacotes que envolvem software externo 11

1.4 Estilo de código e práticas recomendadas 11

1.5 Arquivo CITATION 11

1.6 README 13

1.7 Documentação 15

1.7.1 Geral 15

1.7.2 roxygen2 use 16

1.7.3 Exemplos de conjuntos de dados 18
1.7.4 URLs na documentação 18
1.8 Site de documentação 18
1.8.1 Implementação automática do site de documentação . . . 18
1.8.2 Idioma 19
1.8.3 Agrupamento de funções no índice 19
1.8.4 Marca de autoria 20
1.8.5 Ajustando a barra de navegação 20
1.8.6 Renderização matemática 20
1.8.7 Logotipo do pacote 20
1.9 Autoria 20
1.9.1 Autoria do código incluído no pacote 21
1.10 Licença 21
1.11 Testes 22
1.12 Exemplos 23
1.13 Dependências de pacotes 24
1.14 Estruturas recomendadas 26
1.15 Controle de versão 26
1.16 Problemas diversos do CRAN 27
1.16.1 Verificações do CRAN 28
1.17 Problemas do Bioconductor 28
1.18 Orientações adicionais 28
1.18.1 Aprendendo sobre o desenvolvimento de pacotes 28

2 Práticas Recomendadas de Integração Contínua 31
2.1 O que é a integração contínua? 31
2.2 Por que usar a integração contínua (CI)? 31
2.3 Qual(is) serviço(s) de integração contínua? 32
2.3.1 Travis CI (Linux e Mac OSX) 33
2.3.2 AppVeyor CI (Windows) 33
2.3.3 Circle CI (Linux e Mac OSX) 34

2.4 Cobertura de testes 34
2.5 Ainda mais CI: OpenCPU 34
2.6 Ainda mais CI: documentos da rOpenSci 35

3 Práticas de segurança recomendadas no desenvolvimento de pacotes 37
3.1 Diversos 37
3.2 Segurança no acesso ao GitHub 37
3.3 https 38
3.4 Segredos em pacotes 38
3.4.1 Credenciais em pacotes e proteção do(a) usuário(a) 38
3.4.2 Credenciais em pacotes e desenvolvimento 39
3.4.3 Credenciais e CRAN 40
3.5 Leitura adicional 40

II Revisão por pares de software de pacotes 41

4 Revisão de software por pares, por quê? O que é? 43
4.1 O que é a revisão de software por pares da rOpenSci? 43
4.2 Por que enviar o seu pacote para a rOpenSci? 44
4.3 Por que revisar pacotes para a rOpenSci? 45
4.4 Por que as avaliações são abertas? 45
4.5 Como os(as) usuários(as) saberão que um pacote foi revisado? . . . 46
4.6 Editores(as) e revisores(as) 46
4.6.1 Editor(a)-Chefe 47
4.6.2 Editores(as) associados(as) 47
4.6.3 Revisores(as) e editores(as) passados(as) 48

5 Políticas de revisão de software por pares 51
5.1 Processo de revisão 51
5.1.1 Publicando em outros locais 52
5.1.2 Conflito de interesses para a equipe de revisão e edição . . 52
5.2 Objetivos e Escopo 53

5.2.1 Categorias de pacotes 54
5.2.2 Outras considerações sobre o escopo 56
5.2.3 Sobreposição de pacotes 56
5.3 Propriedade e manutenção de pacotes 57
5.3.1 Função da equipe da rOpenSci 57
5.3.2 Responsividade de mantenedores 57
5.3.3 Compromisso com a qualidade 58
5.3.4 Remoção de pacotes 59
5.4 Ética, Privacidade de Dados e Pesquisa com Seres Humanos 59
5.4.1 Recursos 61
5.5 Código de Conduta 62

6 Guia para Autores 63
6.1 Planejando uma Submissão (ou uma Consulta de Pré-Submissão) . 63
6.1.1 Escopo 63
6.1.2 Ciclo de vida 63
6.1.3 Documentação 64
6.2 Preparando para Submissão 65
6.2.1 Solicitação de ajuda 65
6.2.2 Diretrizes 65
6.2.3 Verificações automáticas 65
6.2.4 Manuscrito de acompanhamento (opcional) 65
6.3 O Processo de Submissão 66
6.4 O Processo de Revisão 67

7 Guia para revisores 69
7.1 Se voluntariando como revisor(a) 69
7.2 Preparando a sua revisão 70
7.2.1 Diretrizes gerais 70
7.2.2 Interações feitas fora dos canais oficiais 72
7.2.3 Experiências de revisores(as) anteriores 72
7.2.4 Pacote de ajuda para revisores 72

7.2.5 Comentários sobre o processo 73
7.3 Envio da revisão 73
7.4 Acompanhamento da revisão 73

8 Guia para a Equipe Editorial 75
8.1 Responsabilidades do papel de Editoria-Chefe (EiC) 76
8.1.1 Deveres gerais do papel de EiC 76
8.1.2 Deveres de EiC para cada submissão inicial 76
8.1.3 O Painel Editorial da rOpenSci 79
8.1.4 Convidando uma pessoa para a tarefa de edição 80
8.2 Responsabilidades gerais dos(as) editores 81
8.3 Responsabilidades de edição de um pacote 81
8.3.1 Após a submissão 82
8.3.2 Busque e atribua duas pessoas para revisar o pacote 83
8.3.3 Durante a revisão 86
8.3.4 Após a revisão 88

9 Gerenciamento editorial 89
9.1 Recrutamento de novos editores 89
9.2 Convidando um(a) novo(a) editor(a) 89
9.3 Integrando um(a) novo(a) editor(a) ao time 90
9.4 Desvincular um(a) editor(a) 91
9.5 Colocando o sistema em pausa 91
9.6 Gerenciando o lançamento de um guia de desenvolvimento 92
9.6.1 Governança do guia de desenvolvimento 92
9.6.2 Postagem no blog sobre um lançamento 92

III Mantendo pacotes 95

10 Folha de dicas de manutenção do pacote rOpenSci 97
10.1 Você precisa de ajuda? 97
10.2 Acesso ao repositório do GitHub 97

10.3 Outros tópicos do GitHub 98
10.4 Documentação do pkgdown 98
10.5 Acesso ao espaço de trabalho do rOpenSci no Slack 98
10.6 Publicações no blog sobre pacotes 98
10.7 Promoção de problemas de pacotes 98
10.8 Promoção de casos de uso de pacotes 98

11 Guia de colaboração 99

11.1 Torne a contribuição e a colaboração do seu repositório amigáveis . 99
11.1.1 Código de conduta 99
11.1.2 Guia de contribuição 100
11.1.3 Gerenciamento de *issues* 101
11.1.4 Comunicação com as pessoas usuárias 102
11.2 Trabalhando com colaboradores 102
11.2.1 Integração de pessoas colaboradoras 102
11.2.2 Trabalhando com colaboradores (incluindo você) 103
11.2.3 Seja generoso(a) com as atribuições 103
11.2.4 Dando as boas-vindas aos colaboradores da rOpenSci . . . 104
11.3 Outros recursos 104

12 Mudando os(as) mantenedores(as) de um pacote 105

12.1 Você quer desistir da manutenção do seu pacote? 105
12.2 Você quer assumir a manutenção de um pacote? 105
12.3 Assumir a manutenção de um pacote 106
12.3.1 Perguntas frequentes para novos(as) mantenedores(as) . . 106
12.4 Tarefas da equipe do rOpenSci 107

13 Publicação de um pacote 109

13.1 Controle de versão 109
13.2 Publicação 109
13.3 Arquivo de notícias 110

14 Marketing do seu pacote 111

15 Preparação do GitHub 113

15.1 Torne seu repositório mais detectável 113

15.1.1 Áreas de repositório do GitHub 113

15.1.2 GitHub linguist 113

15.2 Comercialize sua própria conta 114

16 Evolução do pacote - alteração de itens em seu pacote 115

16.1 Filosofia das alterações 115

16.2 O pacote lifecycle 115

16.3 Parâmetros: alteração dos nomes dos parâmetros 116

16.4 Funções: alteração de nomes de funções 116

16.5 Dados: descontinuar 117

16.5.1 Testando funções descontinuadas 120

16.6 Renomeando pacotes 120

16.7 Arquivamento de pacotes 121

17 Política de curadoria de pacotes 123

17.1 O registro de pacotes 124

17.2 Pacotes mantidos pela equipe 124

17.3 Pacotes revisados por pares 125

17.4 Pacotes legado que foram adquiridos 126

17.5 Pacotes de incubadora 126

17.5.1 Pacotes de incubadora que não sejam pacotes de R 127

17.6 Livros 127

18 Guia de contribuição 129

IV Apêndice 131

19 NEWS 133

19.1 1.0.0 133

19.2 0.9.0 135

19.3 0.8.0 136
19.4 0.7.0 137
19.5 0.6.0 138
19.6 0.5.0 139
19.7 0.4.0 139
19.8 0.3.0 141
19.9 0.2.0 142
19.10 0.1.5 143
19.11 First release 0.1.0 143
19.12 place-holder 0.0.1 143

20 Modelo de revisão 145

20.1 Revisão do pacote 145
20.1.1 Comentários da revisão 146

21 Modelo para o(a) editor(a) 147

21.0.1 Checks do editor: 147
21.1 [ ] **Gerenciamento do projeto**: O monitoramento de problemas (*issues*) e PRs (*pull requests*) está em bom estado, e.g. existem bugs muito críticos, está claro quando um pedido de *feature* está planejado para ser tratado? 148

22 Modelo de solicitação de revisão 149

23 Modelo de comentário de aprovação do(a) revisor(a) 151

23.1 Resposta do(a) revisor(a) 151

24 Modelo de notícias 153

25 Orientação para o lançamento de livros 155

25.1 Versão de lançamento do livro 155
25.1.1 Manutenção do repositório entre lançamentos 155
25.1.2 1 mês antes do lançamento 155
25.1.3 2 semanas antes do lançamento 156
25.1.4 Lançamento 156

26 Como definir um redirecionamento 157

26.1 Site que não seja de páginas do Github Pages (por exemplo, Netlify) 157

26.2 Páginas do GitHub 157

27 Comandos do bot 159

27.1 Para todos 159

27.1.1 Veja a lista de comandos disponíveis para você 159

27.1.2 Veja o código de conduta 159

27.2 Para autores 159

27.2.1 Verificar o pacote com o pkgcheck 159

27.2.2 Enviar resposta aos revisores 160

27.2.3 Finalizar a transferência do repositório 160

27.2.4 Obter um novo convite após a aprovação 160

27.3 Para o editor-chefe 160

27.3.1 Atribua um (a) editor (a) 160

27.3.2 Colocar o envio em espera 160

27.3.3 Indique que o envio está fora do escopo 161

27.4 Para o editor designado 161

27.4.1 Colocar o envio em espera 161

27.4.2 Verificar o pacote com o pkgcheck 161

27.4.3 Verificar padrões estatísticos 161

27.4.4 Verifique se o README tem o selo de revisão de software . . 161

27.4.5 Indique que você está procurando revisores 162

27.4.6 Atribuir um (a) revisor (a) 162

27.4.7 Remover um (a) revisor (a) 162

27.4.8 Ajustar a data de vencimento da revisão 162

27.4.9 Registre que uma revisão foi enviada 162

27.4.10 Aprovar o pacote 162

rOpenSci - Guia para desenvolvedores

Este trabalho está licenciado com uma licença Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License. Consulte o DOI do Zenodo da versão original e o DOI da tradução para saber como citá-las.

Exemplo:

```
Equipe editorial da rOpenSci (2026). Pacotes rOpenSci: Desenvolvimento, manutenção e revisão por
Correia, António Pedro) Zenodo. https://zenodo.org/doi/10.5281/zenodo.18833361 (Trabalho original
```

Você também pode ler a versão PDF deste livro.

Prefácio

Boas vindas! Este livro é um guia para autores, mantenedores, revisores e editores da rOpenSci.

A primeira seção do livro contém as nossas diretrizes para criar e testar pacotes do R.

A segunda seção é dedicada ao processo de revisão por pares de software da rOpenSci: o que é esse processo, quais são as nossas políticas e guias específicos para autores, editores e revisores durante todo o processo. Para *revisão de software estatístico*, consulte a página da Web e os recursos do projeto.

A terceira e última seção apresenta as nossas práticas recomendadas para você cuidar do seu pacote depois que ele tiver sido integrado: como colaborar com outros desenvolvedores, como documentar lançamentos, como promover o seu pacote e como aproveitar o GitHub como uma plataforma de desenvolvimento. A terceira seção também apresenta um capítulo para quem deseja começar a contribuir com os pacotes do rOpenSci.

Esperamos que você ache o guia útil e claro, e agradecemos suas sugestões no *issue tracker* do livro. Feliz embalagem R!

A equipe editorial da rOpenSci.

Este livro é um documento vivo. Você pode ver as atualizações das nossas práticas recomendadas e políticas nas notas de versão.
Você pode citar este livro usando os metadados Zenodo e DOI.

Se você quiser contribuir com este livro (sugestões, correções), consulte o repositório do GitHub em particular as diretrizes de contribuição. Obrigado!

Agradecemos a todos os autores, revisores e editores convidados por nos ajudarem a aprimorar o sistema e este guia ao longo dos anos. Agradecemos também às seguintes pessoas que fizeram contribuições para este guia e suas versões 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,, Jemma 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. Informe-nos se esquecemos de reconhecer a sua contribuição!

Parte I

Construindo seu pacote

Capítulo 1

Guia de desenvolvimento de pacotes

A rOpenSci aceita pacotes que atendam às nossas diretrizes por meio de um processo simplificado de Revisão por Pares de Software. Para garantir um estilo consistente em todas as nossas ferramentas, escrevemos este capítulo destacando nossas diretrizes para o desenvolvimento de pacotes. Leia e aplique também nosso capítulo sobre integração contínua (CI). Outras orientações para depois do processo de revisão são fornecidas na terceira seção deste livro, começando com um capítulo sobre colaboração.

Recomendamos que as pessoas desenvolvedoras de pacotes leiam o livro completo de Hadley Wickham e Jenny Bryan sobre desenvolvimento de pacotes, que está disponível gratuitamente on-line (em inglês). Nosso guia é parcialmente redundante em relação a outros recursos, mas destaca as diretrizes da rOpenSci.

Para saber por que vale a pena enviar um pacote para a rOpenSci para atender às diretrizes, dê uma olhada em motivos para submeter.

1.1 Nome do pacote e metadados

1.1.1 Como nomear seu pacote

- Recomendamos fortemente nomes curtos e descritivos em letras minúsculas. Se o seu pacote tratar de um ou mais serviços comerciais, verifique se o nome não viola as diretrizes de marca. Você pode verificar se o nome do seu pacote está disponível, é informativo e não é ofensivo usando a função `pak::pkg_name_check()`; use também um mecanismo de pesquisa para ver

se é ofensivo em um idioma diferente do inglês. Em particular, *não* escolha um nome de pacote que já esteja sendo usado no CRAN ou no Bioconductor.

- Existe um equilíbrio entre as vantagens de um nome de pacote exclusivo e um nome de pacote menos original.
  - Um nome de pacote mais exclusivo pode ser mais fácil de rastrear (para você e nós avaliarmos o uso do pacote, por exemplo, menos falsos positivos ao digitar seu nome na pesquisa de código do GitHub) e pesquisar (para quando as pessoas usuárias perguntarem "como usar o pacote blah" em um mecanismo de pesquisa).
  - Por outro lado, um nome único *demais* pode fazer com que o pacote seja menos detectável (ou seja, não seja possível encontrá-lo ao pesquisar "como fazer isso em R"). Isso pode ser um argumento para nomear seu pacote com algo muito próximo ao tópico, como geojson).
- Encontre outros aspectos interessantes sobre como nomear seu pacote neste texto do blog do Nick Tierney (em inglês) e, caso você mude de ideia, descubra como renomear seu pacote nesta outra postagem do blog do Nick.

1.2 Plataformas

- Os pacotes devem funcionar em todas as principais plataformas (Windows, macOS, Linux). Pode haver exceções para pacotes que interajam com funções específicas do sistema ou que adaptem utilitários que só funcionam em plataformas limitadas, mas deve-se fazer todo o possível para garantir a compatibilidade entre plataformas, incluindo a compilação específica em cada sistema ou a containerização de utilitários externos.

1.3 API do pacote

1.3.1 Nomeando funções e argumentos

- A nomenclatura das funções e dos argumentos deve ser escolhida de modo a trabalhar em conjunto para formar uma API de programação comum e lógica que seja fácil de ler e de autocompletar.
  - Considere um esquema de nomenclatura `objeto_verbo()` para as funções do seu pacote que usam um tipo de dados comum ou interagem com uma API comum. `objeto` refere-se aos dados/API e `verbo` a ação principal. Esse esquema ajuda a evitar conflitos de nome com pacotes que podem ter verbos semelhantes e torna o código legível e fácil de preencher automaticamente. Por exemplo, em **stringi** as funções que

começam com `stri_` manipulam strings (`stri_join()`, `stri_sort()`, e em **googlesheets** funções que começam com `gs_` são chamadas para a API do Google Sheets (`gs_auth()`, `gs_user()`, `gs_download()`).

- Para funções que manipulam um objeto/dado e retornam um objeto/dado do mesmo tipo, faça com que o objeto/dado seja o primeiro argumento da função para aumentar a compatibilidade com o operador pipe (|> do R base, %>% do pacote magrittr).
- Recomendamos fortemente usar `snake_case` em vez de todos os outros estilos, a menos que esteja fazendo a portabilidade de um pacote que já esteja sendo amplamente utilizado.
- Evite conflitos de nomes de funções com pacotes básicos ou outros pacotes populares (por exemplo `ggplot2`, `dplyr`, `magrittr`, `data.table`)
- A nomenclatura e a ordem dos argumentos devem ser consistentes entre as funções que usam entradas semelhantes.
- As funções do pacote que importam dados não devem importar dados para o ambiente global, mas, em vez disso, devem retornar objetos. As atribuições ao ambiente global devem ser evitadas em geral.

1.3.2 Mensagens do console

- Use o pacote cli ou as ferramentas do R básico (`message()` e `warning()`) para se comunicar com as pessoas que usam suas funções.
- Os destaques do pacote cli incluem: empacotamento automático, respeito a convenção NO_COLOR, muitos elementos semânticos e ampla documentação. Mais informações neste texto em inglês.
- Por favor, não use `print()` ou `cat()` a menos que seja para um `print.*()` ou `str.*()`, pois esses métodos de impressão de mensagens são mais difíceis de serem silenciados.
- Forneça uma maneira de suprimir a verbosidade, de preferência em nível de pacote: torne a criação de mensagens dependente de uma variável ou opção de ambiente (como "usethis.quiet" no pacote usethis), em vez de um parâmetro da função. O controle das mensagens poderia ser feito em vários níveis ("nenhum", "informar", "debugar") em vez de ser lógico (nenhuma mensagem / todas as mensagens). O controle da verbosidade é útil para quem utiliza a função, mas também em testes. Você pode encontrar mais comentários interessantes nesta issue do guia de design do tidyverse
- Você pode fornecer traduções para as mensagens do seu pacote. O pacote potools pode te ajudar com essa tarefa.

1.3.3 Interfaces interativas/gráficas

Se estiver fornecendo uma interface gráfica de usuário (GUI) (como um aplicativo Shiny), para facilitar o fluxo de trabalho, inclua um mecanismo para reproduzir automaticamente as etapas realizadas na GUI. Isso pode incluir a geração automática de código para reproduzir os mesmos resultados, a saída de valores intermediários produzidos na ferramenta interativa ou simplesmente um mapeamento claro e bem documentado entre as ações da GUI e as funções usadas. (Consulte também a seção "Testes" abaixo).

O pacote `tabulizer` por exemplo, tem um fluxo de trabalho interativo para extrair tabelas, mas também pode extrair apenas coordenadas para que seja possível executar novamente como um script. Além disso, dois exemplos de aplicativos brilhantes que geram código são https://gdancik.shinyapps.io/shinyGEO/ e https://github.com/wallaceEcoMod/wallace/.

1.3.4 Verificação de entrada

Recomendamos que seu pacote use um método consistente de sua escolha para verificação de entradas (*inputs*) – seja o R básico, um pacote em R ou ajudantes personalizados.

1.3.5 Pacotes que envolvem recursos da Web (clientes de API)

Se o seu pacote acessar uma API da Web ou outro recurso da Web,

- Certifique-se de que as solicitações enviem um agente de usuário, ou seja, uma maneira de identificar o que (seu pacote) ou quem enviou a solicitação. Deve ser possível substituir o agente de usuário padrão do pacote. Idealmente, o agente de usuário deve ser diferente nos serviços de integração contínua e no desenvolvimento (com base, por exemplo, nos nomes de usuário do GitHub das pessoas desenvolvedoras).
- Você pode escolher padrões diferentes (melhores) do que os da API e, nesse caso, deve documentá-los.
- Seu pacote deve ajudar com a paginação, permitindo que os usuários não se preocupem com isso, pois o pacote faz todas as solicitações necessárias.
- Seu pacote deve ajudar a limitar a taxa de acordo com as regras da API.
- Seu pacote deve reproduzir erros de API e possivelmente explicá-los em mensagens de erro informativas.
- Seu pacote pode exportar funções de alto nível e funções de baixo nível, sendo que estas últimas permitem que os usuários chamem os pontos de acesso (*endpoints*) da API diretamente com mais controle (como `gh::gh()`).

Para obter mais informações, consulte a postagem do blog: Por que você deve (ou não deve) criar um cliente de API (em inglês).

1.3.6 Pacotes que envolvem software externo

- Documente claramente como instalar o pacote, incluindo todos os pacotes ou bibliotecas externas necessários, incluindo, quando aplicável, etapas explícitas em sistemas operacionais comuns.
- Forneça uma função de relatório de situação (sitrep) para verificar se o software foi instalado, com dicas caso algo esteja faltando. Exemplo do pacote greta.
- Se possível, forneça uma função que ajude na instalação. Exemplo no pacote hugodown.

1.4 Estilo de código e práticas recomendadas

- Para obter mais informações sobre como estilizar seu código, nomear funções e scripts R dentro da seção `R/` recomendamos a leitura do capítulo *R Code* do livro *R Packages* (em inglês). Recomendamos o uso do Air ou do pacote styler para automatizar parte do estilo do código. Também sugerimos a leitura do Guia de estilo do Tidyverse (em inglês).

- Você pode optar por usar = ao invés de <− desde que seja consistente com uma escolha em seu pacote. Recomendamos evitar o uso de −> para atribuição em um pacote. Se você usar <− em seu pacote e também usar `R6` nesse pacote, você será forçado a usar = para atribuição em seu `R6Class` - isso não é considerado uma inconsistência porque você não pode usar <− nesse caso.

- Você pode usar o pacote lintr para identificar algumas possíveis áreas de melhoria. Exemplo de fluxo de trabalho.

1.5 Arquivo CITATION

- Se o seu pacote ainda não tiver um arquivo CITATION, você poderá criar um com `usethis::use_citation()` e preenchê-lo com os valores gerados pela função `citation()`.

- O CRAN exige que os arquivos CITATION sejam declarados como itens `bibentry` e não na forma previamente aceita de `citEntry()`.

- Se você arquivar cada versão do seu repositório do GitHub no Zenodo, adicione o DOI *concept* do Zenodo ao arquivo CITATION.

- Se um dia **depois de** revisão na rOpenSci você publicar um artigo sobre seu pacote, adicione-o ao arquivo CITATION.

- Menos relacionado ao seu pacote em si, mas ao que o apoia: se o seu pacote envolve um recurso específico, como uma fonte de dados ou, digamos, um algoritmo estatístico, lembre as pessoas que utilizam o software sobre como citar esse recurso por meio de, por exemplo, `citHeader()`. Talvez até mesmo adicione a referência do recurso.

Como exemplo, veja o arquivo CITATION do dynamite que faz referência ao manual do R, bem como a outras publicações associadas.

```
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"
)
```

- Você também pode criar e armazenar um `CITATION.cff` graças ao pacote

cffr. Ele também fornece um fluxo de trabalho do GitHub Action para manter o `CITATION.cff` atualizado.

1.6 README

- Todos os pacotes devem ter um arquivo README, denominado `README.md` na raiz do repositório. O README deve incluir, de cima para baixo:
  - O nome do pacote.
  - Selos (*badges*) para integração contínua e cobertura de testes, o selo para revisão por pares da rOpenSci assim que ele for iniciado (veja abaixo), um selo do repostatus.org e quaisquer outros selos (por exemplo do R-universe).
  - Uma breve descrição dos objetivos do pacote (o que ele faz? por que seria interessante usá-lo?), com links descritivos para todas as vinhetas (*vignettes*), a menos que o pacote seja pequeno e haja apenas uma vinheta repetindo o README. Certifique-se também de que as vinhetas sejam renderizadas e legíveis, consulte a seção "site de documentação").
  - Instruções de instalação usando, por exemplo, o pacote remotes, pacote pak ou R-universe.
  - Qualquer configuração adicional necessária (tokens de autenticação, etc.).
  - Breve demonstração de uso.
  - Se aplicável, como o pacote se compara a outros pacotes semelhantes e/ou como ele se relaciona com outros pacotes.
  - Informações de citação, ou seja, direcione a forma de citação preferida no README adicionando o texto padrão "aqui está como citar meu pacote". Veja, por exemplo o README do pacote ecmwfr.

Se você usar outro selo de status de repositório, como o ciclo de vida, adicione também um selo do repostatus.org. Exemplo de um README de repositório com dois selos de status de repositório.

- Depois de enviar um pacote e ele ter sido aprovado na verificação editorial, adicione um selo de revisão por pares por meio do

```
[![rOpenSci software peer-review](https://badges.ropensci.org/<issue_id>_status.svg)](https://git
review/issues/<issue_id>)
```

onde issue_id é o número da *Issue* no repositório `software-review`. Por exemplo, o selo para `rtimicropem` usa o número 126, pois é o número da Issue de revisão. O selo indicará primeiro "under review" (em revisão) e depois "peer-reviewed" (revisado por pares) assim que o seu pacote tiver sido integrado (Issue marcada como "approved" (aprovada) e fechada), e será vinculado à Issue de revisão.

- Se o seu README tiver muitos selos, considere ordená-los em uma tabela HTML para facilitar a obtenção de informações. Veja exemplos no repositóriondrake e no repositório `qualtRics`. As seções possíveis são:

  - Desenvolvimento (status de integração contínua (CI), ver o Capítulo sobre CI, canal do Slack para discussão, repostatus)
  - Edição ou Publicação (selos de versão do CRAN e data de lançamento do METACRAN, selo de checagem da API do CRAN, selo do Zenodo)
  - Estatísticas/Uso (downloads, por exemplo: selos de quantidade de downloads do r-hub/cranlogs) A tabela deve ser mais larga do que longa para não mascarar o restante do README.

- Se o seu pacote se conectar a uma fonte de dados ou serviço on-line, ou envolver outro software, considere que o README do seu pacote pode ser o ponto de partida de quem usa o utiliza pela primeira vez. Ele deve fornecer informações suficientes para entender a natureza dos dados, do serviço ou do software e fornecer links para outros dados e documentação relevantes. Por exemplo, um README não deve se limitar a dizer: "Fornece acesso ao GooberDB", mas também incluir, "..., um repositório on-line de avistamentos de Goober na América do Sul. Mais informações sobre o GooberDB e a documentação da estrutura e dos metadados do banco de dados podem ser encontradas em *link*".

- Recomendamos não criar o `README.md` diretamente, mas a partir de um `README.Rmd` (um arquivo R Markdown) se você tiver algum código de demonstração. A vantagem do `.Rmd` é que você pode combinar texto com código que pode ser facilmente atualizado sempre que seu pacote for atualizado.

- Considere o uso da função `usethis::use_readme_rmd()` para obter um modelo para o arquivo `README.Rmd` e para configurar automaticamente uma verificação para garantir que o arquivo `README.md` seja sempre mais recente que o `README.Rmd` antes de fazer um commit.

- Exemplos extensos devem ser mantidos em uma vinheta. Se você quiser tornar as vinhetas mais acessíveis antes de instalar o pacote, sugerimos criar um site para seu pacote.

- Adicionar um código de conduta e as diretrizes de contribuição.

- Veja o README do pacote `gistr` para um bom exemplo de README a ser seguido em um pacote pequeno, e o README do pacote `bowerbird` para um bom exemplo de README para um pacote maior.

1.7 Documentação

1.7.1 Geral

- Todas as funções exportadas de pacote devem ser totalmente documentadas com exemplos.

- Se houver possível sobreposição ou confusão com outros pacotes que forneçam funcionalidade semelhante ou que tenham um nome semelhante, adicione uma nota no README, na vinheta principal e, potencialmente, no campo Descrição do DESCRIPTION. Exemplos em README do rtweet, README do rebird e o pacote slurmR (que não é parte da rOpenSci).

- O pacote deve conter documentação geral para `?pacote`(ou `?`pacote-package` se houver um conflito de nomes). Opcionalmente, você pode usar ambos `?pacote` e ``?pacote-package`` `para o arquivo de manual do pacote, usando a etiqueta@aliasesdo roxygen.` [usethis::use_package_doc()‘](https://usethis.r-lib.org/reference/use_package_doc.html) adiciona o modelo para a documentação geral.

- O pacote deve conter pelo menos um **HTML** que ofereça uma cobertura substancial das funções do pacote, ilustrando casos de uso realistas e como as funções devem interagir. Se o pacote for pequeno, a vinheta e o README poderão ter conteúdo muito semelhante.

- Como no caso de um README, a documentação geral ou as vinhetas podem ser o primeiro ponto de entrada para quem usa o pacote. Se o seu pacote se conectar a uma fonte de dados ou a um serviço on-line, ou envolver outro software, ele deverá fornecer informações suficientes para entender a natureza dos dados, do serviço ou do software e fornecer links para outros dados e documentação relevantes. Por exemplo, a introdução ou a documentação de uma vinheta não deve se limitar a dizer: “Fornece acesso ao GooberDB”, mas também incluir: “…, um repositório on-line de avistamentos de Goober na América do Sul. Mais informações sobre o GooberDB e a documentação da estrutura e dos metadados do banco de dados podem ser encontradas no *link*”. Qualquer vinheta deve descrever o conhecimento necessário para que seja possível entender a vinheta antecipadamente.

A vinheta geral deve apresentar uma série de exemplos que progridam em complexidade, do uso básico ao avançado.

- A funcionalidade que provavelmente será usada apenas para desenvolvimento mais avançado pode ser melhor colocada em uma vinheta separada (por exemplo, a programação usando NSE (*non-standard evaluation*) com dplyr).

- O README, a documentação geral do pacote, as vinhetas, os sites etc. devem ter informações suficientes no início para obter uma visão geral de alto nível do pacote e dos serviços/dados aos quais ele se conecta e fornecer navegação para outras partes relevantes da documentação. Isso é para seguir o princípio de *vários pontos de entrada* ou seja, levar em conta o fato de que qualquer parte da documentação pode ser o primeiro encontro que alguém tem com o pacote e/ou com a ferramenta/dados que ele envolve.

- A(s) vinheta(s) deve(m) incluir citações de software e documentos, quando apropriado.

- Se o seu pacote fornecer acesso a uma fonte de dados, exigimos que o arquivo DESCRIPTION contenha (1) uma breve identificação e/ou descrição da organização responsável pela emissão dos dados; e (2) o URL com link para uma página pública que forneça, descreva ou permita o acesso aos dados (que muitas vezes pode ser diferente do URL que leva diretamente à fonte de dados).

- Use mensagens de inicialização de pacote somente quando necessário (mascaramento de função, por exemplo). Evite mensagens de inicialização de pacotes como "Esse é o pacote 2.4-0" ou orientação de citação, pois elas podem ser irritantes para quem o utiliza. Confie na documentação para obter essa orientação.

- Você pode optar por ter uma seção README sobre casos de uso do seu pacote (outros pacotes, publicações em blogs etc.), exemplo.

1.7.2 roxygen2 use

- Solicitamos que todos os envios usem o roxygen2 para a documentação. O roxygen2 é um pacote R que compila automaticamente os arquivos `.Rd` para a pasta `man` em seu pacote a partir de etiquetas escritas acima de cada função. O roxygen2 tem suporte à sintaxe Markdown. Uma das principais vantagens de usar o roxygen2 é que seu `NAMESPACE` sempre será gerado automaticamente e estará atualizado.

- Mais informações sobre o uso da documentação do roxygen2 estão disponíveis no capítulo sobre documentação de funções do livro *R Packages* (em inglês) e no próprio site do roxygen2.

- Se você estivesse escrevendo o `.Rd` diretamente sem o roxygen2, o Rd2roxygen contém funções para converter o `.Rd` em documentação do roxygen.

- Todas as funções devem documentar o tipo de objeto retornado com a etiqueta `@return`.

- O valor padrão de cada parâmetro deve ser claramente documentado. Por exemplo, em vez de escrever "Um valor lógico que determina se ...", você deve

escrever "Um valor lógico (por padrão `TRUE`) que determina se ...". Também é uma boa prática indicar os valores padrão diretamente na definição da função:

```
f <- function(a = TRUE) {
  # código da função
}
```

- A documentação deve dar suporte à navegação, incluindo links cruzados úteis entre funções relacionadas e documentando funções relacionadas em grupos ou em páginas de ajuda comuns. Recomendamos o uso da etiqueta `@family` que cria automaticamente links do tipo *"See also"* e podem ajudar a agrupar funções em sites com pkgdown. Veja o capítulo sobre documentação de funções do livro *R Packages* (em inglês) e a seção "agrupamento de funções" do presente capítulo para obter mais detalhes.
- Você pode reutilizar partes da documentação (por exemplo, detalhes sobre autenticação, pacotes relacionados) nas páginas de vinhetas, README e de documentação. Consulte a vinheta do roxygen2 sobre reutilização de documentação.
- Para incluir exemplos, você pode usar o clássico `@examples` (no plural *"examples"*), mas também a tag `@example <path>` (no singular *"example"*) para armazenar o código de exemplo em um script R separado (de preferência na pasta `man/`), e a tag `@exampleIf` para executar exemplos condicionalmente e evitar falhas na verificação do R CMD. Consulte a documentação do roxygen2 sobre exemplos.
- Adicionar `#' @keywords internal` para marcar uma função como interna e, ao mesmo tempo, gerar documentação para ela. Se você não quiser que nenhuma documentação de função seja gerada, use `#' noRd`. Consulte a documentação do `roxygen2` sobre tags para indexação e referência cruzada e tags para funções de documentação. Para fins de desenvolvimento, você pode se interessar pelo pacote experimental devtag para obter páginas de manual locais ao usar o `#' @noRd`.
- A partir da versão 7.0.0 do roxygen2, as classes `R6` são oficialmente suportadas. Consulte a documentação do roxygen2 para obter detalhes sobre como documentar classes `R6`.
- Ainda não há suporte para o fornecimento de páginas de manual em diferentes idiomas, mas há um progresso interessante no projeto de pacote em R rhelpi18n.

1.7.3 Exemplos de conjuntos de dados

Para documentar a interface do seu pacote, talvez seja necessário usar conjuntos de dados de exemplo. Você pode usar conjuntos de dados básicos do R (no pacote

datasets), como `penguins`, ou redistribuir e documentar os dados com as devidas atribuições. Tenha o cuidado de escolher dados que estejam em conformidade com o código de conduta da rOpenSci e que, em geral, não sejam prejudiciais ou alienantes para ninguém.

1.7.4 URLs na documentação

Esta subseção é particularmente relevante para quem deseja enviar seu pacote para o CRAN. O CRAN verificará os URLs em sua documentação e não permite códigos de status de redirecionamento, como 301. Você pode usar o pacote urlchecker para reproduzir essas verificações e, em particular, substituir os URLs pelos URLs para os quais eles redirecionam. Outras pessoas já usaram a opção para escapar de alguns URLs (alterar `<https://ropensci.org/>` para `https://ropensci.org/`, ou `\url{https://ropensci.org/}` para `https://ropensci.org/`.), mas se você fizer isso, precisará implementar algum tipo de verificação de URL para evitar que eles sejam quebrados sem que você perceba. Além disso, os links não poderão ser clicados nos documentos locais.

1.8 Site de documentação

Recomendamos a criação de um site de documentação para seu pacote usando o pacote `pkgdown`. O livro *R packages* (em inglês) apresenta um capítulo sobre pkgdown e também o `pkgdown` tem seu próprio site de documentação.

Há alguns elementos que gostaríamos de destacar aqui.

1.8.1 Implementação automática do site de documentação

Você só precisa se preocupar com a implementação automática (*automatic deployment*) do seu site até a aprovação e a transferência do repositório do seu pacote para a organização ropensci; de fato, depois disso, um site pkgdown será criado para o seu pacote após cada *push* para o repositório do GitHub. Você pode encontrar o status dessas compilações em `https://dev.ropensci.org/job/package_name` por exemplo para `magick`; e o site em `https://docs.ropensci.org/package_name` por exemplo para `magick`. A construção do site usará seu arquivo de configuração pkgdown, se você tiver um, exceto para o estilo que usará o arquivo de modelo do pacote`rotemplate`. O site resultante terá uma barra de pesquisa local. Pedimos que informe erros, perguntas e solicitações de recursos sobre a implementação automática em https://github.com/ropensci/docs/ e sobre o modelo em https://github.com/ropensci/rotemplate/.

Se as vinhetas do seu pacote precisarem de credenciais (chaves de API, tokens, etc.) para serem ativadas, talvez você queira renderizar as vinhetas previamente ou armazenar as respostas em cache, já que as credenciais não podem ser usadas no servidor que renderiza a documentação.

Antes do envio e da transferência do pacote, você pode usar a abordagem documentada por `pkgdown` ou o pacote `tic` para a implantação automática do site do pacote. Isso evitaria o incômodo de executar (e lembrar de executar) `pkgdown::build_site()` toda vez que o site precisar ser atualizado. Primeiro, consulte nosso capítulo sobre integração contínua se você não estiver familiarizado com a integração contínua. De qualquer forma, não se esqueça de atualizar todas as ocorrências do URL do site após a transferência para a organização ropensci.

1.8.2 Idioma

Se a documentação do seu pacote estiver escrita em um idioma diferente do inglês (mas suportado pelo sistema de revisão por pares do software rOpenSci), você poderá declarar esse idioma para que o site do pkgdown seja localizado.

Porém, ainda não é possível obter um site pkgdown multilíngue diretamente.

1.8.3 Agrupamento de funções no índice

Quando seu pacote tiver muitas funções, é conveniente que apareçam agrupadas no índice da documentação, o que pode ser feito de forma mais ou menos automática.

Se você usa o roxygen2 acima da versão 6.1.1, deve usar a tag `@family` na documentação de suas funções para indicar o agrupamento. Isso lhe dará links entre as funções na documentação local do pacote instalado (seção *"See also"*) *e* permitirá que você use a função `has_concept` do pacote `pkgdown` no arquivo de configuração do seu site. Exemplo não relacionado a rOpenSci, cortesia de `optiRum`: tag family, arquivo de configuração do `pkgdown` e seção no índice resultante. Para personalizar o texto do título da referência cruzada criada pelo roxygen2 (`Other {family}:`), consulte a documentação do roxygen2 sobre como fornecer uma lista `rd_family_title` no arquivo `man/roxygen/meta.R`.

De forma menos automática, veja o exemplo do website do pacote `drake` e arquivo de configuração associado.

1.8.4 Marca de autoria

Você pode tornar os nomes de (algumas) das pessoas autoras clicáveis, adicionando um URL, e pode até mesmo substituir os nomes por um logotipo (pense na rOpenSci… ou na sua organização/empresa!). Veja a documentação do `pkgdown`.

1.8.5 Ajustando a barra de navegação

Você pode tornar o conteúdo do seu site mais fácil de navegar ajustando a barra de navegação, consulte a documentação do `pkgdown`. Em particular, observe que, se você nomear a vinheta principal de seu pacote como "pkg-name.Rmd", ela poderá ser acessada na barra de navegação como `Para começar` (*Get started*) em vez de via `Artigos > Título da vinheta` (*Articles > Vignette Title*).

1.8.6 Renderização matemática

Consulte a documentação do pkgdown. Nosso modelo é compatível com essa configuração.

1.8.7 Logotipo do pacote

Para usar o logotipo de seu pacote na página inicial do pkgdown, consulte `usethis::use_logo()`. Se o seu pacote não tiver um logotipo, o construtor de documentos da rOpenSci usará o logotipo da rOpenSci em seu lugar.

1.9 Autoria

O arquivo DESCRIPTION de um pacote deve listar as pessoas que participaram da autoria e que colaboraram com o pacote, usando o parâmetro `Authors@R` para indicar suas funções (*author*/*creator*/*contributor*, etc.) e usando o campo de comentário para indicar o ID do ORCID de cada pessoa e o ID ROR de cada organização, se houver. Veja esta seção de "Escrevendo extensões R" para obter detalhes.
Se você achar que as pessoas que revisaram fizeram uma contribuição substancial para o desenvolvimento do seu pacote, poderá listá-los na seção `Authors@R` com o tipo de contribuição `"rev"`, da seguinte forma:

```
    person("Bea", "Hernández", role = "rev",
    comment = "Bea revisou o pacote (v. X.X.XX) para rOpenSci, veja <https://github.co
review/issues/116>"),
```

Somente inclua revisores(as) depois de pedir seu consentimento. Leia mais nesta postagem do blog *Thanking Your Reviewers: Gratitude through Semantic Metadata* ("Agradecendo as revisões: Gratidão por meio de metadados semânticos"). Por favor, não liste pessoas editoras como colaboradoras. Sua participação e contribuição para a rOpenSci já são agradecimentos suficientes!

1.9.1 Autoria do código incluído no pacote

Muitos pacotes incluem códigos de outros softwares. Se arquivos inteiros ou funções individuais forem incluídos de outros pacotes, os pacotes rOpenSci devem seguir a *Política de Repositório* do CRAN:

> A propriedade dos direitos autorais e de propriedade intelectual de todos os componentes do pacote deve ser clara e inequívoca (inclusive a partir da especificação de autoria no arquivo DESCRIPTION). Quando o código for copiado (ou derivado) do trabalho de outros (inclusive do próprio R), deve-se tomar cuidado para que quaisquer declarações de direitos autorais/licenças sejam preservadas e a autoria não seja deturpada.
>
> De preferência, um campo 'Authors@R' seria usado com funções 'ctb' para quem tem a autoria deste código. Como alternativa, o campo "Autor" deve listar essas pessoas como colaboradoras.
>
> Quando os direitos autorais forem detidos por uma entidade que não seja as pessoas autoras do pacote, isso deve ser indicado preferencialmente por meio das funções 'cph' no campo 'Authors@R' ou usando um campo 'Copyright' (se necessário, referindo-se a um arquivo inst/COPYRIGHTS).
>
> As marcas registradas devem ser respeitadas.

1.10 Licença

O pacote precisa ter uma licença aceita pelo CRAN ou OSI. O livro *R packages* (em inglês) inclui uma seção útil sobre licenças.

Se o seu pacote agrupar código de outras fontes, você também precisará reconhecer a autoria do código original no seu arquivo DESCRIPTION, geralmente com uma função de detentor de direitos autorais: `role = "cph"`. Para saber como atualizar seu arquivo DESCRIPTION, consulte o livro *R packages* (em inglês).

1.11 Testes

- Todos os pacotes devem passar nas verificações do `R CMD check`/`devtools::check()` em todas as principais plataformas.
- Todos os pacotes devem ter um conjunto de testes que abranja a funcionalidade principal do pacote. Os testes também devem abranger o comportamento do pacote em caso de erros.

- É uma boa prática escrever testes unitários para todas as funções e para todo o código do pacote em geral, garantindo que a funcionalidade principal seja coberta. Se a cobertura de testes em seu pacote está abaixo de 75%, provavelmente exigirá testes adicionais ou explicações antes de ser enviado para revisão.
- Recomendamos o uso do pacote testthat para escrever testes. Uma alternativa é o tinytest.
- Se esforce para escrever testes ao escrever cada nova função. Isso atende à necessidade óbvia de ter um teste adequado para o pacote, mas permite que você pense sobre as várias maneiras pelas quais uma função pode falhar e programe *defensivamente* contra essas falhas. Mais informações sobre testes.
- Os testes devem ser fáceis de entender e ser tão autocontidos quanto possível. Ao usar o testthat, evite usar código fora do `test_that()` (como etapas de pré-processamento). Recomendamos a leitura da seção *"high-level principles for testing"* (princípios de alto nível para testes) no *livro R Packages*.
- Os pacotes com aplicativos Shiny devem usar uma estrutura de testes unitários, como `shinytest2` ou `shinytest` para testar se as interfaces interativas se comportam conforme o esperado.
- Para testar as funções que criam gráficos, sugerimos usar o vdiffr, uma extensão do pacote testthat que se baseia em testes com snapshots do testthat.
- Se o seu pacote interagir com recursos da Web (APIs da Web e outras fontes de dados na Web), você poderá achar o livro *HTTP testing in R*, de Scott Chamberlain e Maëlle Salmon relevante. Alguns pacotes que ajudam nos testes de HTTP (e seus clientes HTTP correspondentes) são:
  - httptest2 (httr2);
  - httptest (httr);
  - vcr (httr, crul);
  - webfakes (httr, httr2, crul, curl).
- O pacote testthat tem uma função `skip_on_cran()` que você pode usar para não executar testes no CRAN. Recomendamos usar isso em todas as funções que são chamadas de API, pois é muito provável que elas falhem no CRAN. Esses testes ainda devem ser executados na integração contínua. Observe que a partir do testthat 3.1.2 `skip_if_offline()` chama automaticamente `skip_on_cran()`. Mais informações sobre em *CRAN preparedness for API wrappers* (Preparação do CRAN para utilização de APIs).
- Se o seu pacote interagir com um banco de dados, você poderá achar o pacote dittodb útil.
- Depois de configurar a integração contínua (CI) use o relatório de cobertura de código do seu pacote (veja esta seção do nosso livro) para identificar linhas não testadas e adicionar mais testes.

- Mesmo que você use a integração contínua, recomendamos que você execute testes localmente antes de enviar seu pacote (talvez seja necessário definir `Sys.setenv(NOT_CRAN="true")`).

1.12 Exemplos

- Inclua exemplos abrangentes na documentação. Além de demonstrar como usar o pacote, eles podem funcionar como uma maneira fácil de testar a funcionalidade do pacote antes de haver testes adequados. No entanto, lembre-se de que exigimos testes em pacotes contribuídos.

- Você pode executar exemplos com `devtools::run_examples()`. Observe que quando você executa o R CMD CHECK ou equivalente (por exemplo, `devtools::check()`), seus exemplos que não estão incluídos no `\dontrun{}` ou `\donttest{}` são executados. Consulte a seção tabela de resumo na documentação do roxygen2.

- Para evitar que os exemplos sejam executados no CRAN (por exemplo, se requerem autenticação), você precisa usar `\dontrun{}`. No entanto, para uma primeira submissão, o CRAN não permitirá que você pule todos os exemplos. Nesse caso, você pode adicionar alguns pequenos exemplos de brinquedo, ou envolver o código de exemplo com `try()`. Consulte também a etiqueta `@exampleIf` do roxygen2.

- Além de executar exemplos localmente em seu próprio computador, é altamente recomendável que você execute exemplos em um dos sistemas de integração contínua. Mais uma vez, os exemplos que não estão incluídos em `\dontrun{}` ou `\donttest{}` serão executados, mas para aqueles que estão, você pode configurar suas compilações de integração contínua para executá-los por meio dos argumentos de verificação do R CMD `--run-dontrun` e/ou `--run-donttest`.

1.13 Dependências de pacotes

- Em geral, é melhor ter menos dependências.

- Considere as vantagens e desvantagens envolvidas no fato de depender de um pacote. Por um lado, o uso de dependências reduz o esforço de codificação e pode se basear em funcionalidades úteis desenvolvidas por outras pessoas, especialmente se a dependência executar tarefas complexas e tiver alto desempenho, e/ou for bem avaliada e testada. Por outro lado, ter muitas dependências sobrecarrega a pessoa mantenedora ao ter que acompanhar as alterações nesses pacotes, arriscando a sustentabilidade de longo prazo do seu

pacote. Isso também aumenta o tempo e o tamanho da instalação, o que leva em consideração principalmente o seu ciclo de desenvolvimento e o de outras pessoas, bem como os sistemas de compilação automatizados. Pacotes "pesados" - aqueles com muitas dependências e aqueles com grandes quantidades de código compilado - aumentam esse custo.

- As abordagens para reduzir as dependências incluem:
  - Se você usar apenas algumas funções de uma dependência grande ou pesada, poderá copiá-las para seu próprio pacote. (Consulte a seção *Autoria* acima para saber como reconhecer a autoria original do código copiado). Por outro lado, funções complexas com muitos casos especiais (por exemplo, analisadores sintáticos) exigem testes e verificações consideráveis.
    * Um exemplo comum disso é o retorno de "tibbles" no estilo tidyverse em funções do pacote que fornecem dados. É possível evitar o uso do pacote **tibble** retornando um tibble criado pela modificação de um *data.frame* da seguinte forma:

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

      (Observe que essa abordagem deve ser usada e testada com muito cuidado, especialmente porque pode quebrar o comportamento esperado de objetos reclassificados).
  - Certifique-se de que esteja usando o pacote em que a função está definida, e não aquele em que ela é reexportada. Por exemplo, muitas funções do **devtools** podem ser encontradas em pacotes especializados menores, como **sessioninfo**. A função %>% deve ser importada do pacote **magrittr** onde ela é definida, em vez do mais pesado **dplyr** , que a reexporta.
  - Algumas dependências são preferidas porque fornecem uma interpretação mais fácil de nomes de funções e sintaxe mais fáceis do que as soluções básicas do R. Se esse for o principal motivo para usar uma função em uma dependência pesada, considere a possibilidade de envolver a abordagem do R básico em uma função interna bem nomeada em seu pacote. Veja, por exemplo, o script em R do rlang que fornece funções com uma sintaxe semelhante às funções purrr.
  - Se as dependências tiverem funcionalidades sobrepostas, verifique se você pode confiar em apenas uma delas.
  - Mais dicas de gerenciamento de dependências podem ser encontradas no capítulo *"Dependencies: Mindset and Background"* do livro *R packages* (em inglês) e em um post do Scott Chamberlain.
- Usar `Imports` em vez de `Depends` para pacotes que fornecem funções de outros pacotes. Certifique-se de listar os pacotes usados para o teste (`testthat`) e a documentação (`knitr`, roxygen2) em seu `Suggests` das dependências

do pacote (se você usar `usethis` para adicionar a infraestrutura de teste via `usethis::use_testthat()` ou uma vinheta via usethis::use_vignette(), os pacotes necessários serão adicionados ao DESCRIPTION). Se você usar algum pacote nos exemplos ou testes do seu pacote, certifique-se de listá-lo em `Suggests` se ainda não estiver listado em `Imports`.

- Verifique o status de desenvolvimento de todas as dependências que você adicionar. Especialmente para pacotes hospedados no GitHub, é muito útil verificar se eles são mantidos ativamente e se não foram arquivados.

- Se o seu pacote (não do Bioconductor) depender de pacotes do Bioconductor, certifique-se de que as instruções de instalação no README e na vinheta sejam claras o suficiente, mesmo para uma pessoa não esteja familiarizada com o ciclo de publicação do Bioconductor.

  - É necessário usar o `BiocManager` (recomendado)? Documente isso.
  - A instalação automática de pacotes do Bioconductor usando `install.packages()` é suficiente? Nesse caso, mencione que é necessário executar `setRepositories()` se ainda não tiver definido os repositórios necessários do Bioconductor.
  - Se o seu pacote depender do Bioconductor após uma determinada versão, mencione isso na DESCRIPTION e nas instruções de instalação.

- Especificar dependências mínimas (por exemplo `glue (>= 1.3.0)` em vez de apenas `glue`) deve ser uma escolha consciente. Se tiver certeza de que seu pacote quebrará abaixo de uma determinada versão de dependência, especifique-a explicitamente. Mas se não souber, então não há necessidade de especificar uma dependência mínima. Nesse caso, quando um usuário relatar um bug que esteja explicitamente relacionado a uma versão mais antiga de uma dependência, resolva-o. Um exemplo de prática ruim seria, ao desenvolver o pacote, considerar as versões atuais de suas dependências como sendo a versão mínima. Isso forçaria desnecessariamente todos a atualizar (causando problemas com outros pacotes) quando não há um bom motivo por trás dessa escolha de versão.

- Na maioria dos casos em que é necessário expor as funções das dependências, você deve importar e reexportar essas funções individuais em vez de listá-las no campo `Depends`. Por exemplo, se as funções do seu pacote produzem objetos do tipo `raster`, você pode reexportar do pacote **raster** apenas as funções de impressão e plotagem.

- Se seu pacote usar uma dependência de *sistema*, você deve

  - Indicá-la no DESCRIPTION;
  - Verifique se ele está listado por `sysreqsdb` para permitir que ferramentas automáticas o instalem, ou envie uma contribuição caso contrário;

  - Verificar se está listado em um script `configure` (exemplo) e que fornecerá uma mensagem de erro útil caso não seja encontrado (exemplo). Os scripts `configure` podem ser desafiadores, pois geralmente exigem soluções improvisadas para fazer com que as diversas dependências do sistema funcionem em todos os sistemas. Use exemplos (mais aqui) como ponto de partida, mas observe que é comum encontrar bugs e casos extremos e, muitas vezes, violar as políticas do CRAN.

1.14 Estruturas recomendadas

- Para solicitações HTTP, recomendamos o uso dos pacotes httr2, httr, curl ou crul ao invés do RCurl. Se você gosta de clientes de baixo nível para HTTP, o curl é melhor, enquanto o httr2, o httr e o crul são melhores para acesso de alto nível.
- Para converter JSON (*parsing*) , use jsonlite em vez de rjson ou RJSONIO.
- Para converter, criar e manipular XML, recomendamos enfaticamente o pacote xml2 para a maioria dos casos. Você pode consultar as observações de Daniel Nüst sobre a migração de XML para xml2 (em inglês).
- Para dados espaciais, o pacote sp deve ser considerado obsoleto em favor do pacote sf e os pacotes rgdal, maptools e rgeos foram aposentados em 2023. Recomendamos o uso do conjunto de ferramentas espaciais desenvolvidas pelas comunidades r-spatial e rspatial. Veja esta edição do GitHub para discussões relevantes.

1.15 Controle de versão

- Os arquivos de origem do seu pacote devem estar sob controle de versão, mais especificamente rastreados com Git. Você pode achar o pacote gert relevante, bem como algumas das funções do pacote usethis relacionadas ao Git/GitHub; no entanto, você pode usar o git como quiser.
- O nome da ramificação (*branch*) padrão não deve ser `master`, pois isso pode ser ofensivo para algumas pessoas. Consulte a seção do projeto Git e da Software Freedom Conservancy para obter mais contexto. É uma prática geral nomear uma ramificação padrão `main`, embora outros nomes também possam ser usados. Consulte a postagem do blog do tidyverse "Renomeando a ramificação padrão" para saber mais sobre como usar essa funcionalidade para ajudar a renomear as ramificações padrão.
- Certifique-se de listar arquivos desnecessários, como `.DS_Store`, no arquivo .gitignore. Você pode achar a função `usethis::git_vaccinate()` e o pacote gitignore relevantes.

- Uma seção posterior deste livro contém algumas dicas de fluxo de trabalho com git.

1.16 Problemas diversos do CRAN

Esta é uma coleção de problemas do CRAN que vale a pena evitar desde o início.

- Certifique-se de que as palavras do título do seu pacote comecem com letra maiúscula (o que em inglês é chamado de *Title Case*.
- Não coloque um ponto final no final do título.
- Não coloque "no R" ou "com R" em seu título, pois isso é óbvio nos pacotes hospedados no CRAN. Se, mesmo assim, quiser que essas informações sejam exibidas em seu site, verifique a documentação do `pkgdown` para saber como substituir isso.
- Evite iniciar a descrição com o nome do pacote ou "Este pacote ...".
- Certifique-se de incluir links para sites se você envolver uma API da Web, extrair dados de um site etc. na seção `Description` do seu arquivo DESCRIPTION. Os URLs devem ser colocados entre colchetes angulares (<>), por exemplo `<https://www.r-project.org>`.
- Em ambos os `Title` e `Description`, os nomes de pacotes ou outros softwares externos devem ser colocados entre aspas simples (por exemplo, *Integração do 'Rcpp' para a biblioteca de álgebra linear com modelo 'Armadillo'*).
- Evite testes e exemplos que sejam demorados. Considere usar `testthat::skip_on_cran` nos testes para pular coisas que demoram muito, mas ainda assim testá-las localmente e em integração contínua.
- Inclua arquivos de nível superior, como `paper.md` e arquivos de configuração de integração contínua, no arquivo `.Rbuildignore`.

Para obter mais dicas, consulte a lista colaborativa mantida pelo ThinkR, "Prepare-se para o CRAN".

1.16.1 Verificações do CRAN

Quando seu pacote estiver no CRAN, ele será checado regularmente em diferentes plataformas. As falhas nessas verificações, quando não são falsos positivos, podem levar a equipe do CRAN a entrar em contato com você. Você pode monitorar o estado das verificações do CRAN por meio

- do `foghorn` pacote.
- das etiquetas de checagem do CRAN (*"CRAN checks Badges"*).

1.17 Problemas do Bioconductor

Se você pretende que seu pacote seja enviado para o Bioconductor ou se o pacote estiver no Bioconductor, consulte as Diretrizes de empacotamento do Bioconductor e o livro de desenvolvimento atualizado.

1.18 Orientações adicionais

- Se você estiver enviando um pacote para a rOpenSci por meio do repositório de revisão de software, poderá encaminhar outras perguntas à equipe da rOpenSci usando as *issues*.
- Leia o guia de autoria.
- Leia, incorpore e aja de acordo com os conselhos do capítulo *Guia de Colaboração*.

1.18.1 Aprendendo sobre o desenvolvimento de pacotes

1.18.1.1 Livros

- O livro *R packages* (Pacotes em R) escrito por Hadley Wickham e Jenny Bryan é um recurso excelente e de fácil leitura sobre o desenvolvimento de pacotes, está disponível gratuitamente on-line (e pode ser comprado impresso).
- *Writing R Extensions* (Escrevendo extensões do R) (em inglês) é a referência canônica, geralmente a mais atualizada, para a criação de pacotes em R.
- O livro *Mastering Software Development in R* (Dominando o desenvolvimento de software em R) por Roger D. Peng, Sean Kross e Brooke Anderson.
- *Advanced R* (R avançado) por Hadley Wickham.
- *Tidyverse style guide* (Guia de estilo do Tidyverse).
- *Tidyverse design guide* (Guia de design do Tidyverse) (em elaboração) e o boletim de notícias (*newsletter*) que o acompanha.

1.18.1.2 Tutoriais

- *Your first R package in 1 hour* (Seu primeiro pacote R em 1 hora) por Shannon Pileggi.
- essa descrição de fluxo de trabalho por Emil Hvitfeldt.
- Esta ilustração de Matthew J Denny.

1.18.1.3 Blogs

- Blog do R-hub.
- Algumas postagens do blog da rOpenSci, por exemplo *How to precompute package vignettes or pkgdown articles* ("Como pré-computar vinhetas de pacotes ou artigos pkgdown").
- Seção *Package Development Corner* ("Espaço de desenvolvimento de pacotes") do boletim informativo da rOpenSci.
- Algumas postagens do blog do tidyverse, por exemplo *Upgrading to testthat edition 3* ("Atualizando para o testthat edition 3").

1.18.1.4 MOOCs

Existe um especialização do Coursera correspondente ao livro escrito por Roger Peng, Sean Kross e Brooke Anderson, com um curso específico sobre pacotes R.

Capítulo 2

Práticas Recomendadas de Integração Contínua

Este capítulo resume as nossas diretrizes sobre a integração contínua, depois de explicar o que o termo integração contínua significa.

Juntamente com o [capítulo anterior] (#construção), ele forma as nossas diretrizes para a revisão de software por pares.

2.1 O que é a integração contínua?

A integração contínua (do inglês, CI) se refere a execução de testes automáticos em software. No caso da rOpenSci, a CI significa praticamente que um conjunto de testes será executado automaticamente por meio do GitHub, sempre que você fizer um *commit* ou um *pull request* ao GitHub.

A CI automatiza a execução de verificações gerais de pacotes, como `R CMD check`; Veja testando. É possível configurar a CI antes que os testes sejam escritos, assim a CI executará os testes quando você os enviar para o repositório por meio de *commits*.

2.2 Por que usar a integração contínua (CI)?

Todos os pacotes da rOpenSci devem usar uma forma de integração contínua. Isso garante que todos os commits, pull requests e novas ramificações sejam executados pelo `R CMD check`. Os resultados de todos os testes são exibidos na página de *pull requests* no GitHub, fornecendo outra camada de informações sobre os problemas e a proteção contra a quebra do seu pacote antes de fazer a fusão das alterações.

A integração contínua dos pacotes da rOpenSci também deve ser vinculada a um serviço de cobertura de código, indicando quantas linhas são cobertas por testes de unidade.

Tanto o status do teste quanto a cobertura do código devem ser relatados por meio de distintivos no README do seu pacote.

Os pacotes R devem ter CI para todos os sistemas operacionais (Linux, Mac OSX, Windows) quando contiverem:

- Código compilado
- Dependências em Java
- Dependências em outras linguagens
- Pacotes com chamadas de sistema
- Processamento de texto, como obter os nomes das pessoas (para encontrar problemas de codificação)
- Qualquer coisa com sistema de arquivos/chamadas de diretório

Em caso de dúvida sobre a aplicabilidade desses critérios ao seu pacote, é melhor adicionar CI para todos os sistemas operacionais. A maioria das configurações de padrões de serviços de CI para pacotes R permite que isso seja feito sem muito trabalho.

2.3 Qual(is) serviço(s) de integração contínua?

Há vários serviços de integração contínua, incluindo serviços autônomos (CircleCI, AppVeyor) e outros integrados à hospedagem de código ou a serviços relacionados (GitHub Actions, GitLab, AWS Code Pipeline). Diferentes serviços oferecem suporte a diferentes configurações de sistema operacional.

Ações do GitHub é uma opção conveniente para muitas pessoas desenvolvedoras de R que já usam o GitHub, pois está integrada à plataforma e oferece suporte a todos os sistemas operacionais necessários. Existem ações compatíveis com o ecossistema R bem como suporte de primeira classe no pacote {usethis}. Todos os pacotes enviados à rOpenSci para revisão por pares são verificados por nosso sistema `pkgcheck`, descrito mais detalhadamente na seção Guia para Autores. Essas verificações também são fornecidas como uma ação do GitHub no repositório `ropensci-review-tools/pkgcheck-action`. Os autores e as autoras de pacotes são incentivados a usar essa ação para confirmar, antes do envio, que um pacote passa em todas as nossas verificações. Consulte nossa publicação no blog para obter mais informações.

usethis oferece suporte a configuração de CI para outros sistemas embora essas funções estejam levemente obsoletas. A rOpenSci também oferece suporte ao pacote círculo, que auxilia na configuração de pipelines CircleCI, e ao pacote tic para criar pipelines de CI mais complicadas.

2.3.0.1 Testes usando diferentes versões do R

Exigimos que os pacotes da rOpenSci sejam testados nas versões mais recentes, porém também nas versões anteriores e de desenvolvimento do R, para garantir a compatibilidade retroativa e progressiva com o R básico.

Detalhes sobre como executar testes/verificações usando diferentes versões do R localmente podem ser encontrados na vinheta do R-hub ao executar Verificações locais do Linux com Docker.

Você pode ajustar a implementação de testes com cada versão usando uma matriz de testes.

Se você desenvolver um pacote que dependa ou seja destinado ao Bioconductor, esta informação biocthis pode ser relevante.

2.3.0.2 Minimizando o tempo de compilação na CI

Você pode usar estas dicas para minimizar o tempo de compilação na CI:

- Instalar os pacotes em um cache que possa ser reutilizado no processo de CI quando necessário (*cache installation of packages*). O padrão faz isso r-lib/actions workflows.

2.3.0.3 Dependências do sistema

Você pode achar a postagem de Hugo Gruson útil Dependências do sistema em pacotes R e testes automáticos.

2.3.1 Travis CI (Linux e Mac OSX)

Recomendamos que você afaste-se de Travis.

2.3.2 AppVeyor CI (Windows)

Para a integração contínua no Windows, consulte R + AppVeyor. Configure-o usando `usethis::use_appveyor()`.

Aqui estão algumas dicas para você minimizar o tempo de compilação do AppVeyor:

- Instale os seus pacotes em algum tipo de cache. Exemplo de um arquivo de configuração. Ele já estará no arquivo de configuração se você configurar o AppVeyor CI usando `usethis::use_appveyor()`.

- Ativar compilações contínuas.

Não transferimos mais projetos AppVeyor para a conta "ropensci" no AppVeyor, portanto, após a transferência do seu repositório para a conta "ropensci" no GitHub, o distintivo será `[![AppVeyor Build Status](https://ci.appveyor.com/api/projects/status/githu`

2.3.3 Circle CI (Linux e Mac OSX)

Circle CI é usado, por exemplo, pelo pacote `bomrang` da rOpenSci como serviço de integração contínua.

2.4 Cobertura de testes

A integração contínua também deve incluir relatórios de cobertura de teste por meio de um serviço de teste, como o Codecov ou Coveralls.

Recomendamos que você use Codecov. Para ativar Codecov em seu repositório, execute `usethis::use_github_action("test-coverage")` para criar um arquivo `.github/workflows/test-coverage.yaml`. Você também precisa dar ao Codecov acesso ao seu repositório do GitHub, consulte Guia de início rápido do Codecov para saber como configurar o acesso. Em seguida, adicione um distintivo de status do Codecov à parte superior do seu README.md, consulte Distintivos de status do Codecov.

Se o seu repositório for transferido para a organização GitHub ropensci, o acesso ao Codecov deverá ser transferido automaticamente. Você precisará atualizar o URL do distintivo para apontar ao repositório hospedado na rOpenSci.

Para mais detalhes e instruções, consulte a seção README do pacote **covr**, bem como `usethis::use_coverage()` e `usethis::use_github_action()`.

Se você executar a cobertura em vários serviços de CI os resultados serão fundidos.

2.5 Ainda mais CI: OpenCPU

Após a transferência para a organização "ropensci" no GitHub pertencente a rOpenSci, cada envio para o repositório será contruido no OpenCPU e a pessoa que fizer o *commit* receberá um e-mail de notificação. Esse é um serviço de CI adicional para autores e autoras de pacotes que permite que as funções do R em pacotes sejam chamadas remotamente via https://ropensci.ocpu.io/, usando o API opencpu. Para obter mais detalhes sobre esse serviço, consulte a página de ajuda do OpenCPU que também indica onde você pode fazer perguntas.

2.6 Ainda mais CI: documentos da rOpenSci

Após a transferência para a organização "ropensci" no GitHub pertencente a rOpenSci, um site pkgdown será criado para o seu pacote:

- Para cada novo commit no ramo padrão (verificado aproximadamente uma vez por hora).
- Quando qualquer uma das dependências fortes no mesmo universo atualiza o número da versão.
- Uma vez por mês.

Você pode encontrar o status dessas compilações em `https://ropensci.r-universe.dev/ui#packages` e na seção status do commit. A compilação do site usará o seu arquivo config do pkgdown, se você tiver um, exceto para o estilo que usará o pacote `rotemplate`. Se sua documentação incluir código que dependa, por exemplo, de credenciais, veja aqui como garantir que os documentos pkgdown sejam renderizados da melhor maneira possível.

- Para exemplos de funções, use a tag `examplesIf` do roxygen2 com a variável `IN_PKGDOWN`, por exemplo, `#' @examplesIf identical(Sys.getenv("IN_PKGDOWN"), "true")`. Exemplo: gtexr (consulte também a documentação da tag do roxygen2).
- Para vinhetas, pré-compile se forem necessárias ferramentas/dados/credenciais especiais que não estão disponíveis em servidores de compilação genéricos (consulte https://ropensci.org/blog/2019/12/08/precompute-vignettes/) ou use a variável `IN_PKGDOWN` com a opção knitr eval. Exemplo: gtextr ou:

```
knitr::opts_chunk$set(
  collapse = TRUE,
  comment = "#>",
  eval = Sys.getenv("IN_PKGDOWN") == "true"
)
```

- No caso de vinhetas/artigos que executam requisições HTTP, você pode usar pacotes específicos do R, como o vcr para armazenar as respostas em cache. Exemplo: nettskjemar.

Por favor, informe bugs, faça perguntas e solicitações de recursos sobre as compilações centrais e sobre o modelo em https://github.com/ropensci-org/rotemplate/.

Capítulo 3

Práticas de segurança recomendadas no desenvolvimento de pacotes

Este capítulo em desenvolvimento inclui [orientações sobre o gerenciamento de credenciais em pacotes] (#pkgsecrets), além de [links para leituras complementares] (#furthersecreading).

3.1 Diversos

Recomendamos a leitura do artigo Dez dicas rápidas para você se manter seguro(a) on-line, de Danielle Smalls e Greg Wilson.

3.2 Segurança no acesso ao GitHub

- Recomendamos que você proteja a sua conta do GitHub com uma autenticação 2FA (autenticação de dois fatores). Essa medida é *obrigatória* para todos os membros da organização ropensci e colaboradores externos que usam o GitHub, portanto, certifique-se de ativá-la antes que o seu pacote seja aprovado.
- Também recomendamos que você verifique regularmente quem tem acesso ao repositório do seu pacote e remova qualquer acesso não utilizado (como os de ex-colaboradores e ex-colaboradoras).

3.3 https

- Se o serviço Web que o seu pacote utiliza oferecer tanto https quanto http, opte por https.

3.4 Segredos em pacotes

Esta seção oferece orientações para quando você desenvolve pacotes que interagem com recursos da Web que exigem credenciais (como chaves de API, tokens, etc.). Consulte também a vinheta do pacote `httr` sobre o compartilhamento de credenciais.

3.4.1 Credenciais em pacotes e proteção do(a) usuário(a)

Digamos que o seu pacote precise de uma chave de API para fazer as solicitações em nome dos(as) usuários(as) do seu pacote.

- Na documentação do seu pacote, oriente o(a) usuário(a) para que a chave de API não seja registrada no arquivo .Rhistory ou no arquivo de código dos(as) usuários(as) do seu pacote.
  - Incentive o uso de variáveis de ambiente para armazenar a chave de API (ou até mesmo remova a possibilidade de passá-la como um argumento para as funções). Você pode, na documentação, fazer referência a introdução aos arquivos de inicialização e a função `usethis::edit_r_environ()`.
- Ou o seu pacote pode depender ou incentivar o uso de `keyring` para ajudar o(a) usuário(a) a armazenar variáveis nos gerenciadores de credenciais específicos do sistema operacional (mais seguro do que .Renviron), ou seja, você criaria uma função para definir a chave e teria outra para recuperá-la; ou escreveria `Sys.setenv(SUPERSECRETKEY = keyring::key_get("myservice"))` no início do seu arquivo de código.
  - Não imprima a chave de API, nem mesmo no modo verboso, em qualquer mensagem, aviso ou erro.
- No modelo de *issue* do GitHub, deve ser declarado que nenhuma credencial deve ser compartilhada. Se um(a) usuário(a) do seu pacote compartilhar acidentalmente as credenciais em uma issue, certifique-se de que ele(a) esteja ciente disso para que possa revogar a chave (ou seja, pergunte explicitamente, em uma resposta, se a pessoa percebeu que compartilhou a chave).

3.4.2 Credenciais em pacotes e desenvolvimento

Você precisará proteger as suas credenciais da mesma forma que protege as credenciais dos(as) usuários(as), mas há mais aspectos a serem considerados e mantidos em mente.

3.4.2.1 Credenciais e solicitações registradas em testes

Se você utiliza `vcr` ou `httptest` em testes para armazenar as respostas da API em cache, é importante garantir que as requisições ou configurações registradas não contenham credenciais. Consulte o guia de segurança do pacote `vcr` e o guia do pacote `httptest` "Redigindo e modificando requisições registradas". Além disso, inspecione as suas requisições ou configurações registradas antes de realizar o primeiro commit para garantir que você fez a configuração correta.

3.4.2.2 Compartilhe credenciais com os serviços de CI

Agora, você pode precisar compartilhar credenciais com os serviços de integração contínua.

Você pode armazenar as chaves de API como variáveis de ambiente ou credenciais, consultando a documentação do serviço de CI.

Para obter mais detalhes e orientações sobre o fluxo de trabalho, consulte o artigo do pacote gargle - "Gerenciando tokens com segurança" e o capítulo sobre segurança do livro HTTP testing in R.

Documente as etapas em CONTRIBUTING.md para que você, ou um(a) novo(a) mantenedor(a), possa se lembrar como proceder da próxima vez.

3.4.2.3 Credenciais e colaborações

E quanto a pull requests de colaboradores(as) externos(as)? No GitHub, por exemplo, as credenciais só estão disponíveis para GitHub Actions em pull requests iniciados a partir do próprio repositório, e não a partir de um fork. Os testes que usam as suas credenciais falharão, ao menos que você use algum tipo de resposta simulada ou em cache, portanto, pode ser interessante ignorá-los dependendo do contexto. Por exemplo, na sua conta de CI, você poderia criar uma variável de ambiente chamada `THIS_IS_ME` e, então, ignorar os testes com base na presença dessa variável. Isso significa, portanto, que as verificações de PR feitas pela CI não serão exaustivas e, como consequência, você precisará verificar o PR localmente para executar todos os testes.

Documente o comportamento do seu pacote em relação a PRs externos no arquivo CONTRIBUTING.md. Isso será útil tanto para quem faz PRs quanto para quem os revisa, seja você no futuro ou outras pessoas mantenedoras do pacote.

3.4.3 Credenciais e CRAN

No CRAN, ignore quaisquer testes (`skip_on_cran()`) e exemplos (`dontrun`) que exijam credenciais.

Gere previamente as vinhetas que requerem credenciais, ou use o pacote vcr.

3.5 Leitura adicional

Materiais úteis sobre segurança:

- a sessão da comunidade rOpenSci "Segurança para R" (veja a gravação, os slides, etc. e, em particular, a lista de recursos);
- os projetos relacionados à segurança do unconf18;
- o artigo do pacote `gargle` "Gerenciando tokens de forma segura"

Parte II

Revisão por pares de software de pacotes

Capítulo 4

Revisão de software por pares, por quê? O que é?

Este capítulo contém uma introdução geral ao nosso sistema de revisão de software por pares para pacotes, razões para submeter um pacote, razões para se voluntariar como revisor(a), por que as nossas revisões são abertas e agradecimentos às pessoas que participam do sistema de revisão.

Nosso sistema foi recentemente ampliado para a revisão de software estatístico por pares.

_Se você usar os nossos padrões/checklists/etc ao revisar um software em outro lugar, informe as pessoas destinatárias (por exemplo, editores(as) de periódicos, estudantes, revisores(as) internos(as) de código) que eles vieram da rOpenSci e nos informe em nosso fórum público ou em particular por e-mail.

4.1 O que é a revisão de software por pares da rOpenSci?

A coleção de pacotes da rOpenSci é resultado parcialmente das contribuições de membros(as) da equipe e parcialmente das contribuições de membros(as) da comunidade, o que significa que a coleção provém de uma grande diversidade de habilidades e de experiência de pessoas desenvolvedoras. Como garantir a qualidade de toda a coleção? É aí que entra a revisão de software por pares: os pacotes contribuídos pela comunidade passam por um processo de revisão transparente, construtivo, não adversarial e aberto. Para esse processo, que depende principalmente de trabalho voluntário, os(as) editores(as) associados(as) gerenciam o fluxo de submissões e garantem o andamento dos envios; os(as) autores(as) criam, submetem e

aprimoram os seus pacotes; os(as) revisores(as), duas pessoas por submissão, examinam o código e a experiência do(a) usuário(a). Esta publicação no blog, escrita por editores(as) da rOpenSci, é uma boa introdução à revisão de software por pares da rOpenSci. Outras postagens sobre a própria revisão e sobre os pacotes revisados podem ser encontradas através da tag "software-peer-review" no blog da rOpenSci.

Você pode reconhecer os pacotes da rOpenSci que foram revisados por pares por meio de um selo (*badge*) verde com o texto "peer-reviewed" em seu README, com os links para as suas revisões (por exemplo); e por meio de um botão azul com o texto *"Peer-reviewed"* próximo à sua descrição na página de pacotes da rOpenSci com os links para as revisões.

Tecnicamente, aproveitamos ao máximo o GitHub: cada processo de revisão de pacote é uma *issue* no repositório `ropensci/software-review` no GitHub. Por exemplo, você pode ler o tópico de revisão do pacote `ropenaq`: o processo é uma conversa contínua até a aceitação do pacote, com duas revisões externas como marcos importantes. Além disso, usamos os recursos do GitHub, como o uso de *issue templates* (como templates de submissão) e uso de *labels* (etiquetas) para acompanhar o progresso dos envios (desde as verificações do(a) editor(a) até a aprovação).

4.2 Por que enviar o seu pacote para a rOpenSci?

- Em primeiro lugar, e acima de tudo, esperamos que você envie o seu pacote para análise **porque você valoriza o feedback**. Nosso objetivo é fornecer feedbacks úteis para os(as) autores(as) de pacotes e fazer com que o nosso processo de revisão seja aberto, não contraditório e focado na melhoria da qualidade do software.
- Ao fazer parte da coleção de pacotes da rOpenSci, o seu pacote continuará a receber **suporte da equipe da rOpenSci**. Você manterá a propriedade e o controle do seu pacote, mas podemos ajudar com as questões de manutenção contínua, como as associadas a atualizações do R, dependências e políticas do CRAN.
- A rOpenSci irá **divulgar o seu pacote** através da nossa página web, blog e mídias sociais (como Mastodon e LinkedIn). Os pacotes em nossa coleção também recebem um site de documentação que é renderizado e publicado automaticamente após cada envio.
- Os pacotes da rOpenSci **podem ser incluídos em uma lista cruzada** com outros repositórios, como o CRAN e o BioConductor.
- Os pacotes da rOpenSci que estão no escopo do Journal of Open-Source Software e que tenham um artigo curto relacionado, poderão, a critério dos(as) editores(as) do JOSS, beneficiar-se de um processo de revisão acelerado (*fast-track*).
- Se você escrever um, a rOpenSci irá **divulgar os livros relacionados ao seu pacote**: você pode transferir o código-fonte de tais livros para a organização

`ropensci-books` no GitHub para que os livros sejam listados em books.ropensci.org.

4.3 Por que revisar pacotes para a rOpenSci?

- Como em qualquer processo de revisão por pares, esperamos que você opte por revisar **para que você retribua à rOpenSci e às comunidades científicas.** Nossa missão de expandir o acesso a dados científicos e promover uma cultura de pesquisa reprodutível só é possível por meio dos esforços voluntários de membros(as) da comunidade como você.
- A revisão é uma conversa de mão dupla. Ao revisar os pacotes, você terá a chance de **continuar aprendendo as práticas de desenvolvimento com os(as) autores(as) e outros(as) revisores(as)**.
- A natureza aberta do nosso processo de revisão permite que você **faça contatos e conheça colegas e colaboradores(as)** durante o processo de revisão. Nossa comunidade é acolhedora e composta por pessoas prestativas, especialistas em desenvolvimento em R e em diversas outras áreas da ciência e da computação científica.
- Para se voluntariar para ser um(a) de nossos(as) revisores(as), preencha este breve formulário fornecendo as suas informações de contato e áreas de especialização. Estamos sempre procurando mais revisores(as) com experiência geral em desenvolvimento de pacotes e com conhecimento especializado nas áreas em que os pacotes são usados.

4.4 Por que as avaliações são abertas?

Nossos tópicos (*issues*) de revisão são públicos. As pessoas autoras, revisoras e editoras conhecem as identidades umas das outras. A comunidade mais ampla pode visualizar ou até participar da conversa conforme ela acontece. Isso incentiva a atenção aos detalhes e o fornecimento de avaliações construtivas e não adversariais. Tanto autores(as) quanto revisores(as) relatam que apreciam e aprendem mais com essa troca aberta e direta. Também traz o benefício de fortalecer a comunidade. Participantes têm a oportunidade de estabelecer conexões significativas com colegas, e novas colaborações surgiram a partir de ideias geradas durante o processo de revisão.

Estamos cientes de que sistemas abertos podem ter desvantagens. Por exemplo, na revisão acadêmica tradicional, a revisão por pares duplo-cega pode aumentar a representação de autoras do gênero feminino, sugerindo viés em revisões não-cegas. Também é possível que revisores(as) sejam menos críticos em revisões abertas. No entanto, sustentamos que a abertura da conversa de revisão funciona como um controle de qualidade e viés; é mais difícil inserir comentários subjetivos ou sem embasamento em público e sem o manto do anonimato. Em última análise, acreditamos

que a comunicação direta e pública entre autores(as) e revisores(as) melhora a qualidade e a equidade das revisões.

Além disso, os(as) autores(as) e revisores(as) podem entrar em contato com os(as) editores(as) em particular se tiverem alguma dúvida ou questão.

4.5 Como os(as) usuários(as) saberão que um pacote foi revisado?

- O README do seu pacote apresentará um selo de revisão por pares com um link para o tópico de revisão do software.
- Seu pacote terá um site de documentação em `docs.ropensci.org` que você pode referenciar no DESCRIPTION.
- Se você quiser, o repositório do seu pacote poderá ser transferido para a organização rOpenSci no GitHub.
- Se os(as) revisores(as) concordarem em ser listados(as) no DESCRIPTION, os metadados do pacote mencionarão a revisão.

4.6 Editores(as) e revisores(as)

O processo de revisão de software por pares da rOpenSci é conduzido pela nossa equipe dedicada de editores(as) e revisore(as). Informações sobre a equipe atual e o andamento da revisão de software por pares podem ser consultadas em nosso painel interativo.

```
Warning in value[[jj]][ri] <- if (is.factor(xij)) as.vector(xij) else xij:
number of items to replace is not a multiple of replacement length
```

```
Warning in names(value[[jj]])[ri] <- nm: number of items to replace is not a
multiple of replacement length
```

```
Warning in value[[jj]][ri] <- if (is.factor(xij)) as.vector(xij) else xij:
number of items to replace is not a multiple of replacement length
```

```
Warning in names(value[[jj]])[ri] <- nm: number of items to replace is not a
multiple of replacement length
```

```
Warning in value[[jj]][ri] <- if (is.factor(xij)) as.vector(xij) else xij:
number of items to replace is not a multiple of replacement length
```

```
Warning in names(value[[jj]])[ri] <- nm: number of items to replace is not a
multiple of replacement length

Warning in value[[jj]][ri] <- if (is.factor(xij)) as.vector(xij) else xij:
number of items to replace is not a multiple of replacement length

Warning in names(value[[jj]])[ri] <- nm: number of items to replace is not a
multiple of replacement length

Warning in value[[jj]][ri] <- if (is.factor(xij)) as.vector(xij) else xij:
number of items to replace is not a multiple of replacement length

Warning in names(value[[jj]])[ri] <- nm: number of items to replace is not a
multiple of replacement length

Warning in value[[jj]][ri] <- if (is.factor(xij)) as.vector(xij) else xij:
number of items to replace is not a multiple of replacement length

Warning in names(value[[jj]])[ri] <- nm: number of items to replace is not a
multiple of replacement length
```

4.6.1 Editor(a)-Chefe

Nós alternamos nosso(a) editor(a)-chefe, geralmente a cada três meses. Nosso(a) editor(a)-chefe atual éEmily Riederer.

4.6.2 Editores(as) associados(as)

O processo de revisão de software por pares da rOpenSci é conduzido por:

- Laura DeCicco;
- Jouni Helske, University of Jyväskylä, Finland;
- Toby Hocking, Northern Arizona University, USA;
- Jeff Hollister, US Environmental Protection Agency;
- Rebecca Killick, Lancaster University, U.K.;
- Anna Krystalli;
- Mauro Lepore, 2 Degrees Investing Initiative;
- Beatriz Milz, University of Sao Paulo - Institute of Energy and Environment;
- Mark Padgham;
- Francisco Rodriguez-Sanchez, Universidad de Sevilla, Spain;
- Noam Ross, rOpenSci and EcoHealth Alliance;
- Maëlle Salmon, rOpenSci;

- Margaret Siple, National Oceanic and Atmospheric Administration;
- Adam Sparks, Curtin University;
- Emi Tanaka, Australian National University;
- Sebastian Wójcik, Statistics Poland, University of Rzeszów (Institute of Mathematics).

4.6.3 Revisores(as) e editores(as) passados(as)

Agradecemos as seguintes pessoas que dedicaram o seu tempo e conhecimento para revisar os pacotes submetidos à rOpenSci.

Em Markowitz (NOAA) · Lorena Abad · Sam Albers · Toph Allen · Kaique dos S. Alves · 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 · Tanguy Barthelemy · 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 · Kyle F Butts · Maria Paula Caldas · Mario Gavidia Calderón · Carlos Cámara-Menoyo · Brad Cannell · Paul CARTERON · Joaquin Cavieres · Kevin Cazelles · Cathy Chamberlin · Jennifer Chang · Pierre Chausse · Jorge Cimentada · Nicholas Clark · Chase Clark · Jon Clayden · Dena Jane Clink · Will Cornwell · Pao Corrales · Nic Crane · Enrico Crema · Verónica Cruz-Alonso · Ildiko Czeller · Tad Dallas · Kauê de Sousa · Christophe Dervieux · Dylan Dijk · Amanda Dobbyn · Jasmine Dumas · Christophe Dutang · Remko Duursma · Mark Edmondson · Paul Egeler · Evan Eskew · Harry Eslick · Denisse Fierro-Arcos · Alexander Fischer · Kim Fitter · Robert M Flight · Sydney Foks · Air Forbes · Stephen Formel · Zachary Stephen Longiaru Foster · Auriel Fournier · Kaija Gahm · Zach Gajewski · Carl Ganz · Duncan Garmonsway · Jan Laurens Geffert · Sharla Gelfand · Monica Gerber · Alex Gibberd · Duncan Gillespie · David Gohel · A. Cagri gokcek · Guadalupe Gonzalez · Rohit Goswami · Laura Graham · João Granja-Correia · Charles Gray · Matthias Grenié · Corinna Gries · Hugo Gruson · Hugo Gruson · Ernest Guevarra · W Kyle Hamilton · Ivan Hanigan · Jeffrey Hanson · Sebastian Hanß · 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 · Tan Ho · 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 · Nerea Piñeiro Juncal · Krunoslav Juraic · Soumya Kalra · Zhian N. Kamvar · Michael Kane · Andee Kaplan · Tinula Kariyawasam · Hazel Kavılı · Ella Kaye · Jonathan Keane · Christopher T. Kenny · Os Keyes · Eunseop Kim · Aaron A. King · Michael Koontz · Alexandros Kouretsis · Bianca Kramer · Robert Kubinec · Will Landau · Sam Lapp · Erin LeDell · Thomas Leeper · Alex Leith · Sam Levin · Lisa Levinson · Stephanie Locke · Marion Louveaux · Robin Lovelace · Julia Stewart Lowndes · Adam Loy · Tim Lucas · Muralidhar, M.A. · Andrew MacDonald · Jesse Maegan ·

Mike Mahoney · Tristan Mahr · Christina Maimone · Yohann Mansiaux · Paula Andrea Martinez · Anthony Martinez · Joao Martins · Ben Marwick · Claire Mason · Joan Maspons · Tom Matthews · Miles McBain · Lucy D'Agostino McGowan · Amelia McNamara · Elaine McVey · Bryce Mecum · Nolwenn Le Meur · François Michonneau · Mario Miguel · David L Miller · Helen Miller · Jessica Minnier · Priscilla Minotti · Nichole Monhait · Kelsey Montgomery · Ronny A. Hernández Mora · Natalia Morandeira · George Moroz · Ross Mounce · Athanasia Monika Mowinckel · Lincoln Mullen · Matt Mulvahill · Maria Victoria Munafó · David Neuzerling · Dillon Niederhut · Joel Nitta · Rory Nolan · Simon Nolte · 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 · Marina Papadopoulou · Edzer Pebesma · Thomas Lin Pedersen · Antonio J. Pérez-Luque · Marcelo S. Perlin · Rafael Pilliard-Hellwig · July Pilowsky · Rodrigo Pires · Lindsay Platt · Nicholas Potter · Joanne Potts · Josep Pueyo-Ros · Etienne Racine · Manuel Ramon · Nistara Randhawa · David Ranzolin · Quentin Read · Nicola Rennie · Neal Richardson · Emily Riederer · tyler rinker · Emily Robinson · David Robinson · Alec Robitaille · 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 · 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 · phanikumar s tata · Jason Taylor · Simon Taylor · Filipe Teixeira · Christian Testa · Andy Teucher · Jennifer Thompson · Joe Thorley · Nicholas Tierney · Tiffany Timbers · Tan Tran · Tim Trice · Sunny Tseng · Anatolii Tsyplenkov · 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 · Daniel 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 · 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 · brock · kasselhingee · Bri · Flury · Vincent · eholmes · Pachá · Rich · Claudia · Jasmine · Zack · Lluís · becarioprecario · gaurav

Também agradecemos aos seguintes editores(as) que atuaram anteriormente.

- Brooke Anderson;
- Scott Chamberlain, Fred Hutch Cancer Center;
- Julia Gustavsen, Agroscope;
- Paula Moraga, King Abdullah University of Science and Technology (KAUST), Saudi Arabia;
- Karthik Ram, University of California, Berkeley, rOpenSci;
- Melina Vidoni.

E aos(as) seguintes que atuaram como editores(as) convidados(as).

- Ana Laura Diedrichs;
- Julia Gustavsen;
- Emily Riederer;
- Hao Zhu.

Capítulo 5

Políticas de revisão de software por pares

Este capítulo contém as políticas de revisão de software por pares da rOpenSci.

Em particular, você deverá ler nossas políticas relativas à revisão de software por pares por conta própria: o processo de submissão de revisões, incluindo nossas políticas de conflito de interesses e os objetivos e escopo do sistema de Revisão de Software por Pares. Esse capítulo também apresenta nossas políticas relacionadas à propriedade e manutenção de pacotes.

Por último, mas não menos importante, você encontrará o código de conduta da Revisão de Software por Pares da rOpenSci.

5.1 Processo de revisão

- Para que um pacote seja considerado para a coleção da rOpenSci, os(as) autores(as) do pacote devem iniciar uma solicitação no repositório ropensci/software-review.
- Os pacotes são revisados quanto à qualidade, adequação, documentação e clareza, e o processo de revisão é bastante semelhante ao de um manuscrito (consulte nosso guia de desenvolvimento de pacotes e guia de revisão para obter mais detalhes). Diferentemente de uma revisão de manuscrito, esse processo se configura como uma conversa contínua.
- Quando todos os principais problemas e dúvidas, incluindo aqueles que podem ser abordados com esforço razoável, forem resolvidos, o(a) editor(a) designado para o pacote tomará uma decisão (aceitar, esperar ou rejeitar). Rejeições geralmente são feitas com antecedência (antes do início do processo de revisão; consulte a seção de objetivos e escopo). Em casos raros, um pacote

também pode não ser aceito após a análise e a revisão. Em última análise, a decisão de rejeitar ou não o pacote é do(a) editor(a), com base em como as revisões são tratadas.

- A comunicação entre as pessoas envolvidas na autoria, revisão e edição ocorrerá principalmente pelo GitHub, embora você possa optar por entrar em contato com a pessoa responsável pela edição por e-mail ou Slack para alguns problemas. Ao submeter um pacote, certifique-se de que suas configurações de notificação do GitHub estejam ajustadas para que seja improvável você perder algum comentário.
- O(a) autor(a) pode optar por ter seu envio colocado em espera (o(a) editor(a) aplica a etiqueta (*label*) de espera). O status de espera será revisado a cada 3 meses e, após 1 ano, a *issue* será encerrada.
- Se o(a) autor(a) não tiver solicitado uma etiqueta de espera, mas simplesmente não estiver respondendo, devemos fechar a edição dentro de 1 mês após a última ocasião de contato. Esta ocasião incluirá um comentário marcando o(a) autor(a), mas também um e-mail usando o endereço de e-mail listado no *DESCRIPTION* do pacote, que é um dos raros casos no qual o(a) editor(a) tentará entrar em contato com o(a) autor(a) por e-mail.
- Se uma submissão for encerrada e o(a) autor(a) desejar reenviá-la, ele(a) terá que iniciar uma nova submissão. Se o pacote ainda estiver no escopo, o(a) autor(a) terá que responder às revisões iniciais antes que o(a) editor(a) comece a procurar novos(as) revisores(as).

5.1.1 Publicando em outros locais

- Sugerimos fortemente que você envie seu pacote para revisão *antes de* publicá-lo no CRAN ou enviar um artigo de software descrevendo o pacote em um periódico. O feedback da revisão pode resultar em grandes aprimoramentos e atualizações do seu pacote, incluindo renomeações e alterações críticas de funções. Não consideramos a publicação anterior no CRAN ou em outros locais como razão suficiente para não adotar as recomendações de revisores(as) ou editores(as).
- Não envie seu pacote para revisão se ele ou um manuscrito associado estiver sendo revisado em outro local, pois isso pode resultar em solicitações de alterações conflitantes.

5.1.2 Conflito de interesses para a equipe de revisão e edição

Os critérios a seguir devem servir de guia para o que constitui um conflito de interesses envolvendo quem faz a edição ou revisão. Existe um conflito de interesses se:

- Quem potencialmente revisará ou editará for da mesma instituição ou componente institucional (por exemplo, departamento) que qualquer autor(a) que

tenha um papel importante no pacote.
- Quem potencialmente revisará ou editará colaborou ou teve outros laços profissionais com pelo menos uma pessoa que tenha um papel importante no pacote nos últimos 3 anos.
- Quem potencialmente revisará ou editará atua ou atuou membro(a) do conselho consultivo do projeto em análise.
- Quem potencialmente revisará ou editará receberia um benefício financeiro direto ou indireto se o pacote fosse aceito.
- Quem potencialmente revisará ou editará contribuiu significativamente para um projeto concorrente.
- Também existe um conflito de interesses vitalício para familiares, relações comerciais/negócios e pessoas envolvidas em relações de orientação, sejam estudantes ou orientação/mentoria.

No caso em que nenhuma das pessoas da equipe editorial possa realizar a edição, uma pessoa externa será convidada para realizá-la.

5.2 Objetivos e Escopo

O objetivo da rOpenSci é oferecer suporte a pacotes que possibilitem replicabilidade na pesquisa, assim como gerenciamento do ciclo de vida de dados para cientistas. Os pacotes enviados à rOpenSci devem se enquadrar em uma ou mais das categorias descritas abaixo. Software estatístico também pode ser enviado para revisão por pares, para o qual temos uma seção separada de diretrizes e padrões. As categorias abaixo são para software em geral, e não estatístico, enquanto o restante deste capítulo se aplica a ambos os tipos de software. Se você não tiver certeza se o seu pacote se encaixa em uma das categorias citadas, abra uma *issue* como uma consulta de pré-submissão (**Exemplos**).

Como este é um documento dinâmico, essas categorias podem mudar com o tempo e nem todos os pacotes previamente integrados podem estar no escopo atual. Por exemplo, os pacotes de visualização de dados não estão mais no escopo. Embora nos esforcemos para ser consistentes, avaliamos os pacotes caso a caso e podemos abrir exceções.

Observe que nem todos os projetos e pacotes da rOpenSci estão dentro do escopo ou passam por revisão por pares. Projetos desenvolvidos pela equipe da rOpenSci ou em conferências podem ser experimentais, exploratórios, tratar de prioridades essenciais de infraestrutura e, portanto, não se enquadram nessas categorias. Procure o selo (*badge*) de revisão por pares – veja abaixo – para identificar pacotes revisados por pares no repositório da rOpenSci.

Figura 5.1: exemplo de um selo (*badge*) verde de revisão por pares

5.2.1 Categorias de pacotes

- **recuperação de dados**: Pacotes para acessar e fazer download de dados de fontes online com aplicações científicas. Nossa definição de aplicações científicas é ampla, incluindo serviços de armazenamento de dados, periódicos e outros servidores remotos, já que muitas fontes de dados podem ser interessantes para pesquisadores(as). Entretanto, os pacotes de recuperação devem se concentrar em *fontes de dados* ou *tópicos* em vez de *serviços*, e deve fazer mais do que apenas baixar dados. Por exemplo, um cliente geral para armazenamento de dados da Amazon Web Services não estaria no escopo, nem um pacote que oferecesse apenas a funcionalidade de download sem qualquer pré-processamento ou pré-filtragem. Exemplos de revisão de pacotes de recuperação de dados dentro do escopo incluem **rotl**, que oferece um ampla gama de funções; e **gutenbergr**, que pré-processa um enorme conjunto de metadados para facilitar muito a interface com um banco de dados enorme e variado.

- **extração de dados**: Pacotes que auxiliam na obtenção de dados de fontes não estruturadas, como texto, imagens e PDFs, bem como na análise de tipos de dados científicos e *outputs* (saídas) de equipamentos científicos. Bibliotecas estatísticas ou de aprendizado automático para modelagem ou previsão normalmente não são incluídas nesta categoria, nem os analisadores de código (*code parsers*). Os modelos treinados que atuam como utilitários (por exemplo, para reconhecimento óptico de caracteres) podem se qualificar. (Exemplos: **tabulizer** para extrair tabelas de documentos PDF, **genbankr** para analisar arquivos do GenBank, **treeio** para leitura de arquivos de árvores filogenéticas, **lightr** para analisar arquivos de instrumentos espectroscópicos)

- **processamento de dados**: Pacotes para processamento de dados dos formatos acima. Essa área não inclui ferramentas de manipulação de dados amplas, como **reshape2** ou **tidyr**, ou ferramentas para extração de dados do próprio código R. Em vez disso, ele se concentra em ferramentas para lidar com dados em formatos científicos específicos, gerados a partir de fluxos de trabalho científicos ou exportados de instrumentos científicos. (Exemplos: **plateR** para leitura de dados estruturados como mapas de placas para instrumentos científicos, ou **phonfieldwork** para processar arquivos de áudio anotados para pesquisa fonética)

- **depósito de dados**: Pacotes que possibilitam o depósito de dados em repositórios de pesquisa, incluindo a formatação de dados e a geração de metadados. (Exemplo: **EML**)

- **validação e teste de dados**: Ferramentas que permitem a validação e a verificação automatizadas da qualidade e da integridade dos dados como parte dos fluxos de trabalho científicos (Exemplo: **assertr**)

- **automação de fluxo de trabalho**: Ferramentas que automatizam e vinculam fluxos de trabalho, como sistemas de compilação e ferramentas para geren-

ciar a integração contínua. Não inclui ferramentas gerais para programação letrada (*literate programming*) (por exemplo, extensões de R Markdown não incluídas nos tópicos anteriores) (Exemplo: **drake**)

- **controle de versão**: Ferramentas que facilitam o uso de controle de versões em fluxos de trabalho científicos. Observe que isso não inclui todas as ferramentas que interagem com serviços de controle de versão online (por exemplo, GitHub), a menos que elas se enquadrem em outra categoria (Exemplo: **git2rdata**)

- **gerenciamento de citações e bibliometria**: Ferramentas que facilitam o gerenciamento de referências, como para escrever manuscritos, criar currículos ou atribuir contribuições científicas, ou acessar, manipular ou trabalhar com dados bibliométricos (Exemplo: **RefManageR**)

- ***wrappers* de software científico**: Pacotes que envelopam (*wrap*) programas utilitários fora do ambiente R, usados para pesquisa científica. Esses programas devem ser específicos para campos de pesquisa, e não utilitários gerais de computação. Os *wrappers* devem ser não triviais, ou seja, devem ter um valor agregado significativo em relação ao simples uso de chamadas ou vinculações do `system()`, seja na análise de argumentos (*inputs*) e resultados (*outputs*), no manuseio de dados, etc. Um processo de instalação aprimorado ou a extensão da compatibilidade para mais plataformas pode constituir um valor agregado se a instalação for complexa. Isso não inclui *wrappers* de outros pacotes do R ou bibliotecas de C/C++ que podem ser incluídas nos pacotes do R. Isso também não inclui pacotes que são clientes para APIs na internet, que devem se enquadrar em uma das outras categorias. Recomendamos enfaticamente que você inclua utilitários de código aberto e de licença aberta. Exceções serão avaliadas caso a caso, considerando se opções de código aberto existem (Exemplos: **babette**, **nlrx**)

- **ferramentas de reprodutibilidade de campo e laboratório**: Pacotes que melhoram a reprodutibilidade de fluxos de trabalho do mundo real por meio da padronização e automação de protocolos de campo e laboratório, como rastreamento e marcação de amostras, geração de formulários e planilhas de dados, interface com equipamentos de laboratório ou sistemas de informação e execução de desenhos experimentais (Exemplo: **baRcodeR**)

- **vinculações de software de banco de dados**: Vinculações (*bindings*) e *wrappers* para APIs de bancos de dados genéricos (Exemplo: **rrlite**)

Além disso, temos alguns *tópicos especializados* com um escopo um pouco mais amplo.

- **dados geoespaciais**: Aceitamos pacotes focados no acesso, na manipulação e na conversão entre formatos de dados geoespaciais (Exemplos: **osmplotr**, **tidync**)

- **tradução**: Como parte de nosso trabalho em publicações multilíngue, temos um interesse especial em pacotes que facilitem a tradução e a publicação de recursos científicos e de programação em vários idiomas (humanos) para que sejam acessíveis ao público em maior alcance e diversidade. Isso pode incluir interfaces para programas de tradução automática, estruturas para gerenciar documentação em vários idiomas ou programas que acessem recursos linguísticos especializados. Este é um escopo novo e experimental, portanto, crie uma consulta de pré-submissão se você tiver interesse em enviar um pacote nesta categoria.
- **Ferramentas internas da rOpenSci** pacotes criados e/ou usados pela equipe da rOpenSci para apoiar a revisão por pares de software e outras iniciativas relacionadas.

5.2.2 Outras considerações sobre o escopo

Os pacotes devem ser **gerais** no sentido de que devem resolver um problema da forma mais ampla possível, enquanto mantém uma interface de usuário(a) e uma base de código coerentes. Por exemplo, se várias fontes de dados usam uma API idêntica, preferimos um pacote que forneça acesso a todas estas fontes de dados, em vez de acesso a apenas uma.

Os pacotes que incluem ferramentas interativas para facilitar os fluxos de trabalho de pesquisadores(as) (por exemplo, aplicativos em *shiny*) devem ter um mecanismo para tornar o fluxo de trabalho interativo reprodutível, como a geração de código ou uma API com script.

Para pacotes que não estão no escopo da rOpenSci, recomendamos que você os envie para o CRAN, Bioconductor, bem como para outras iniciativas de desenvolvimento de pacotes do R (por exemplo, cloudyr) e periódicos de software, como JOSS, JSS ou o R journal, dependendo do escopo atual desses periódicos.

Observe que os pacotes desenvolvidos internamente pela rOpenSci, por meio de nossos eventos ou colaborações, não estão necessariamente no escopo do nosso processo de revisão de software por pares.

5.2.3 Sobreposição de pacotes

A rOpenSci incentiva a competição entre pacotes, como *forking* e reimplementação, pois isto melhora as opções dos usuários em geral. No entanto, como queremos que os pacotes da coleção rOpenSci tenham nossas principais recomendações para as tarefas que realizam, nosso objetivo é evitar a duplicação da funcionalidade de pacotes R existentes em qualquer repositório, se estes não apresentam melhorias significativas. Um pacote R que replica a funcionalidade de um pacote R já existente pode ser considerado a inclusão no conjunto rOpenSci, se ele melhorar significativamente as alternativas em qualquer repositório (RO, CRAN, BioC) por ser:

- Mais aberto nas práticas de licenciamento ou desenvolvimento
- Mais amplo em termos de funcionalidade (por exemplo, fornecendo acesso a mais conjuntos de dados ou um conjunto maior de funções), mas não apenas duplicando pacotes adicionais
- Melhor em termos de usabilidade e desempenho
- Mantido mais ativamente, enquanto as alternativas são pouco ou não são mais mantidas

Esses fatores devem ser considerados **como um todo** para determinar se o pacote representa uma melhoria significativa. Um novo pacote não atenderia a esse padrão apenas por seguir nossas diretrizes, enquanto outros não o fazem, a menos que isso leve a uma diferença significativa como mencionado acima.

Recomendamos que os pacotes destaquem as diferenças e os aprimoramentos quanto aos pacotes aos quais se sobrepõem em seu README e/ou *vignettes*.

Incentivamos as pessoas desenvolvedoras cujos pacotes não forem aceitos devido à sobreposição com outros pacotes a considerarem submissões para outros repositórios ou periódicos.

5.3 Propriedade e manutenção de pacotes

5.3.1 Função da equipe da rOpenSci

Os(as) autores(as) de pacotes contribuídos a rOpenSci mantêm essencialmente a mesma propriedade que tinham antes de seu pacote entrar no conjunto rOpenSci. Os(as) autores(as) de pacotes continuarão a manter e desenvolver seu *software* após a aceitação na rOpenSci. A menos que sejam explicitamente adicionados como colaboradores, a equipe da rOpenSci não interferirá muito nas operações corriqueiras. No entanto, essa equipe poderá intervir com correções de *bugs* críticos ou resolver problemas urgentes se os(as) autores(as) dos pacotes não responderem em tempo hábil (consulte abaixo a seção sobre responsividade de mantenedores).

5.3.2 Responsividade de mantenedores

Se as pessoas que mantém o pacote não responderem em tempo hábil às solicitações de correções de pacotes feitas pelo CRAN ou por nós, lembraremos algumas vezes, mas depois de 3 meses (ou em um período mais curto, dependendo da importância da correção) a equipe da rOpenSci fará as alterações.

O que foi dito acima é um pouco vago, portanto, a seguir estão alguns aspectos a serem considerados.

- Exemplos em que gostaríamos de agir rapidamente:

  - Pacote `foo` é importado por um ou mais pacotes no CRAN. `foo` está quebrado e, portanto, quebraria suas dependências reversas
  - Pacote `bar` pode não ter dependências reversas no CRAN, mas é amplamente usado e, portanto, a correção rápida de problemas que ele apresenta é de grande importância

- Exemplos em que podemos esperar um pouco mais:

  - Pacote `hello` está ou não está no CRAN, porém não tem dependências reversas
  - Pacote `world` precisa de algumas correções. A pessoa que mantém o pacote respondeu, mas simplesmente está muito ocupada, porém atenderá a solicitação em breve

Pedimos aos(as) mantenedores(as) de pacotes que se certifiquem de que estão recebendo notificações do GitHub, bem como que os e-mails da equipe da rOpenSci e dos mantenedores do CRAN não estejam indo para a caixa de *spam*. Os(as) autores(as) de pacotes integrados à rOpenSci serão convidados ao Slack da rOpenSci para poder conversar com a equipe da rOpenSci e com a comunidade em geral.

Se os(as) autores(as) abandonarem a manutenção de um pacote usado ativamente na rOpenSci, consideraremos a possibilidade de solicitar ao CRAN a transferência do status de mantenedor do pacote para a rOpenSci.

5.3.3 Compromisso com a qualidade

A rOpenSci se esforça para desenvolver e promover software de pesquisa de alta qualidade. Para garantir que o seu software atende aos nossos critérios, analisamos todas as nossas submissões dentro do processo de Revisão de Software por Pares. Mesmo após a aceitação do pacote, continuaremos a contribuir com melhorias e correções de *bugs*.

Apesar de nossos melhores esforços em oferecer suporte ao software contribuído, erros são de responsabilidade dos(as) mantenedores(as) individuais. Os softwares com muitos *bugs* e sem manutenção adequada podem ser removidos da nossa coleção a qualquer momento.

5.3.4 Remoção de pacotes

No caso improvável de um(a) colaborador(a) de um pacote solicitar a remoção de seu pacote da coleção da rOpenSci, temos o direito de manter uma versão do pacote em nossa coleção para fins de arquivamento.

5.4 Ética, Privacidade de Dados e Pesquisa com Seres Humanos

Os pacotes da rOpenSci e outras ferramentas são usados para uma variedade de finalidades, mas nosso foco são ferramentas para pesquisa. Esperamos que as ferramentas possibilitem o uso ético por profissionais da pesquisa, que são obrigados(as) a aderir a códigos de ética, como a Declaração de Helsinque e o Relatório Belmont. Os(as) pesquisadores(as) são responsáveis pelo uso de software, mas desenvolvedores(as) de software devem considerar o uso ético de seus produtos. Desenvolvedores(as) de software também aderem a códigos de ética direcionados a profissionais de computação, como aqueles mencionados pela IEEE e ACM. Os(as) contribuidores(as) da rOpenSci frequentemente desempenham tanto o papel de pesquisador(a) como desenvolvedor(a).

Pedimos que os(as) desenvolvedores(as) de software se coloquem no papel de pesquisadores(as) e considerem os requisitos de um fluxo de trabalho ético usando o software dos(as) autores(as). Dada a variação e o grau de fluxo das abordagens éticas para análises baseadas na Internet, são necessários julgamentos em vez de receitas. As Diretrizes Éticas da Associação de Pesquisadores da Internet fornece uma estrutura robusta, a qual incentivamos autores(as), editores(as) e revisores(as) usar para que avaliem seus trabalhos. Em geral, a adesão a normas legais ou requerimentos mínimos regulamentares (por exemplo, GDPR) pode não ser suficiente, embora relevantes. Os(as) autores(as) de pacotes devem direcionar os(as) usuários(as) a recursos relevantes para o uso ético do software. *(Nota de tradução. GDPR é a sigla para* General Data Protection Regulation*, a lei de proteção de dados pessoais vigente na Europa. É similar à LGPD - Lei Geral de Proteção de Dados Pessoais, existente no Brasil.)*

Alguns pacotes, devido à natureza dos dados que manipulam, podem ser considerados pelos(as) editores(as) como necessitando um exame mais minucioso. Para esses pacotes, os(as) editores(as) podem exigir funcionalidades adicionais (ou reduzidas), uma documentação robusta, padrões e avisos para direcionar os(as) usuários(as) a práticas éticas relevantes. Os tópicos a seguir podem merecer um exame minucioso:

- ***Populações vulneráveis***. Autores(as) de pacotes e fluxos de trabalho que lidam com informações relacionadas a populações vulneráveis têm a responsabilidade de proteger essas populações de possíveis danos.

- ***Dados sensíveis ou de identificação pessoal***: A liberação de dados sensíveis ou de identificação pessoal é potencialmente prejudicial. Isso inclui dados "razoavelmente reidentificáveis", onde um indivíduo poderia rastrear determinada pessoa proprietária ou criadora, mesmo que os dados sejam anônimos. Isso inclui ambos casos em que identificadores (por exemplo, nome, data de nascimento) estão disponíveis como parte dos dados ou se pseudônimos/*nicknames* exclusivos estiverem vinculados a postagens de texto com-

pleto, por meio das quais alguém pode vincular indivíduos por meio de referências cruzadas com outros conjuntos de dados.

Embora a melhor resposta às preocupações éticas seja específica para cada contexto, essas diretrizes gerais devem ser seguidas pelos pacotes quando os desafios acima surgirem:

- Os pacotes devem aderir aos termos de uso da fonte de dados, conforme expresso em Termos e condições do site, arquivos "robots.txt", políticas de privacidade e e outras restrições relevantes, e coloque um link para eles de forma destacada na documentação do pacote. Os pacotes devem fornecer ou documentar a funcionalidade para aderir a a essas restrições (por exemplo, raspar somente de *endpoints* permitidos, usar o limite de taxa (*rate limiting*) apropriado nos códigos, exemplos ou vinhetas). Observe que, embora os Termos e Condições, Políticas de Privacidade, etc., não forneçam limites suficientes para a uso ético, eles podem fornecer um limite externo.

- Uma ferramenta fundamental para lidar com os riscos apresentados ao estudar populações vulneráveis ou no uso de dados pessoalmente identificáveis é o ***consentimento informado***. Autores(as) do pacote devem apoiar a obtenção do consentimento informado dos usuários, quando relevante. Isso pode incluir o fornecimento de links para o método preferido da fonte de dados para a obtenção do consentimento, informações de contato dos provedores de dados (por exemplo, moderadores do fórum), documentação de protocolos de consentimento informado ou a obtenção de pré-aprovação para usos gerais de um pacote.

  Observe que o consentimento não é concedido implicitamente apenas pelo fato de os dados estarem acessíveis. Dados acessíveis não são necessariamente públicos, já que diferentes pessoas e contextos têm diferentes expectativas normativas de privacidade (consulte o trabalho do Social Data Lab).

- Os pacotes que acessam informações pessoais identificáveis devem ter um cuidado especial para seguir as práticas recomendadas de segurança (por exemplo, uso exclusivo de protocolos de internet seguros, mecanismos fortes para armazenamento de credenciais, etc.)

- Pacotes que acessam ou manipulam dados pessoais identificáveis ou dados sensíveis devem permitir, documentar e demonstrar fluxos de trabalho para desidentificação, armazenamento seguro e outras práticas recomendadas para minimizar o risco de danos.

À medida que os padrões de privacidade de dados e pesquisa continuam a evoluir, damos boas-vindas as contribuições dos(as) autores(as) sobre considerações específicas de seu software e documentações suplementares, como aprovação de comitês de ética universitária. Essas podem ser anexadas à *issue* de submissão de pacote ou consulta de pré-submissão, ou transmitidas diretamente aos(as) editores(as), se

necessário. Sugestões genéricas podem ser registradas como *issues* no repositório deste livro.

5.4.1 Recursos

Os recursos a seguir podem ser úteis para pesquisadores(as), autores(as) de pacotes, editores(as) e revisores(as) na abordagem de questões éticas relacionadas à privacidade e ao software de pesquisa.

- Os Declaração de Helsinque e o Relatório Belmont fornecem princípios fundamentais para a prática ética de pesquisadores(as).
- Várias organizações fornecem orientações sobre como traduzir esses princípios para o contexto da pesquisa na internet. Entre elas estão a Diretrizes Éticas da Associação de Pesquisadores da Internet, o Guia NESH para a Ética de Pesquisa na Internet e as Diretrizes de Ética para pesquisas mediadas pela Internet da BPS. Anabo et al (2019) fornece uma visão geral útil sobre isso.
- O *Social Data Science Lab* fornece uma visão geral com dados sobre expectativas normativas de privacidade e uso em fóruns sociais.
- Bechmann A., Kim J.Y. (2019) Big Data: A Focus on Social Media Research Dilemmas. Em: Iphofen R. (org.) 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., & 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., & 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., & 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., & 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 Conduta

A comunidade da rOpenSci é o nosso melhor patrimônio. Seja você uma pessoa colaboradora assídua ou recém-chegada, nós nos preocupamos em fazer deste um lugar seguro para você, por isso te apoiamos. Temos um Código de Conduta que se aplica a todas as pessoas que participam da comunidade rOpenSci, incluindo a equipe e a liderança da rOpenSci, e a todos os modos de interação online ou pessoalmente. O Código de Conduta é mantido no site da rOpenSci.

Capítulo 6

Guia para Autores

Este guia conciso apresenta o processo de revisão de software por pares, para você como autor de um pacote.

6.1 Planejando uma Submissão (ou uma Consulta de Pré-Submissão)

6.1.1 Escopo

- Consulte nossas políticas de uso para ver se o seu pacote atende aos nossos critérios e se encaixa em nossa coleção, não se sobrepondo a outros pacotes já existentes.
  - Se você não tiver certeza de que um pacote atende aos nossos critérios, sinta-se à vontade para abrir um *issue* no GitHub como uma consulta de pré-submissão para perguntar se o pacote é apropriado.
  - Exemplo de resposta a sobreposição. Também considere adicionar alguns pontos sobre pacotes semelhantes ao seu na sua documentação do pacote.

6.1.2 Ciclo de vida

- Não envie vários pacotes ao mesmo tempo: solicitamos que você espere até que um pacote seja aprovado antes de enviar outro.
- Você pretende manter o seu pacote por pelo menos 2 anos ou ser capaz de identificar uma nova pessoa para mantê-lo?

- Considere o melhor momento de desenvolvimento do seu pacote para enviar sua submissão. Seu pacote deve estar suficientemente maduro para que os revisores possam analisar todos os aspectos essenciais, mas tenha em mente que revisões podem resultar em grandes alterações.

  - Sugerimos enfaticamente que você envie seu pacote para análise *antes de* publicá-lo no CRAN ou antes de enviá-lo para publicação como artigo em um periódico. O *feedback* da revisão pode resultar em grandes aprimoramentos e atualizações do seu pacote, incluindo renomeações e alterações de funções.
  - Não envie seu pacote para revisão enquanto este ou o manuscrito associado também estiver sendo revisado em outro local, pois isso pode resultar em solicitações conflitantes de alterações.

- Considere também o tempo e o esforço necessários para responder às revisões: pense na sua disponibilidade ou na de seus colaboradores nas próximas semanas e meses após o envio da submissão. Observe que os revisores são voluntários e pedimos que você respeite o tempo e o esforço deles, respondendo de maneira oportuna e respeitosa.
- Se você usa distintivos do repostatus.org (o que recomendamos), envie uma submissão quando você estiver pronto para receber um distintivo tipo *Active* em vez de *WIP*. Da mesma forma, se você usa distintivos tipo *lifecycle* o envio da submissão deverá ocorrer quando o pacote for *Stable*.
- Seu pacote continuará a evoluir após a revisão. O capítulo sobre *Evolução do pacote* fornece mais orientações sobre este tópico.

6.1.3 Documentação

- Para qualquer envio ou consulta de pré-submissão, o README do seu pacote deve fornecer informações suficientes sobre o pacote (objetivos, uso, pacotes semelhantes) para que os editores avaliem seu escopo sem precisar instalar o pacote. Melhor ainda, crie um website pkgdown para permitir uma avaliação mais detalhada da funcionalidade online.

  - No estágio de envio da submissão, todas as principais funções devem ser estáveis o suficiente para serem totalmente documentadas e testadas; o README deve apresentar uma base segura para o pacote.
  - Seu arquivo README deve assegurar-se em explicar a funcionalidade e os objetivos do seu pacote, presumindo que os leitores tenham pouco ou nenhum conhecimento do domínio. Todos os termos técnicos, inclusive as referências a outros softwares, devem ser esclarecidos.

- Seu pacote continuará a evoluir após a revisão. O capítulo sobre *Evolução do pacote* fornece mais orientações sobre este tópico.

6.2 Preparando para Submissão

6.2.1 Solicitação de ajuda

- Fique à vontade para fazer perguntas sobre o processo ou sobre seu pacote em específico no https://github.com/ropensci/software-review-meta/issues.

6.2.2 Diretrizes

- Leia e siga nosso guia de estilo de pacotes e nosso guia de revisão, para garantir que seu pacote atenda aos nossos critérios de estilo e qualidade.

6.2.3 Verificações automáticas

- Todas as submissões são verificadas automaticamente pelo nosso `pkgcheck` para garantir que os pacotes sigam as nossas diretrizes. Espera-se que todas as pessoas autoras tenham executado a principal função do `pkgcheck` localmente para confirmar que o pacote está pronto para ser submetido. Como alternativa, uma maneira ainda mais fácil de garantir que um pacote está pronto para ser submetido é usando a função `pkgcheck` do GitHub Action, conforme descrito em nossa postagem no blog.
- Se o seu pacote exigir dependências incomuns de sistema (consulte *Guia de pacotes*) para que a *GitHub Action* seja aprovada, envie um *pull request* adicionando-as ao nosso arquivo Dockerfile. Consulte esta vinheta `pkgcheck` para obter detalhes sobre o nosso ambiente de verificação e como modificá-lo para ajudar o seu pacote a passar nas verificações.
- Se houver algum aspecto do `pkgcheck` no qual seu o pacote não possa ser aprovado, explique os motivos no seu modelo de submissão.

6.2.4 Manuscrito de acompanhamento (opcional)

Se você pretende enviar um manuscrito de acompanhamento para seu o pacote, a rOpenSci tem uma parceria de colaboração com os periódicos [Journal of Open-Source Software] (https://joss.theoj.org/) e [Methods in Ecology and Evolution] (https://besjournals.onlinelibrary.wiley.com/journal/2041210X):

- Para enviar um pacote ao Journal of Open-Source Software (JOSS), não o envie para consideração do JOSS até que o processo de revisão do rOpenSci tenha terminado: se o seu pacote for considerado dentro do escopo pelos editores do JOSS, apenas o artigo curto que o acompanha será revisado (não o software que terá sido revisado extensivamente pelo rOpenSci até aquele momento). Nem todos os pacotes da rOpenSci atenderão aos critérios do JOSS.

- Para uma submissão ao Methods in Ecology and Evolution (MEE), envie-a ao MEE somente depois que as revisoras e revisores da rOpenSci tiverem enviado suas revisões, antes ou depois de o pacote ter sido aceito. A colaboração de revisão com a MEE foi apresentada em uma postagem de blog. O tipo de artigo relevante para a MEE é *Applications* para obter mais detalhes.

6.3 O Processo de Submissão

- Um software é enviado/submetido para revisão através da abertura de uma nova *issue* no repositório de revisão do software, sendo preenchido o modelo sugerido.
- O modelo sugerido começa com uma seção que inclui diversas variáveis no estilo HTML (`<!---variável--->`). Elas são usadas pelo nosso `ropensci-review-bot` e devem ser deixadas nos seus respectivos lugares, com valores preenchidos entre os pontos de início e fim indicados, assim:

```
<!---variável--->insira valor aqui<!---variável-fim>
```

- A comunicação entre autores, revisores e editores ocorrerá primeiramente no GitHub para que o tópico de revisão possa servir como um registro completo da revisão. Você pode optar por entrar em contato com o editor por e-mail ou Slack se for necessária uma consulta particular (por exemplo, perguntar como responder a uma pergunta de um revisor). Não entre em contato com os revisores fora do tópico (*thread* do GitHub) sem perguntar a eles de antemão se eles concordam com isso.
- Ao submeter um pacote, certifique-se de que suas notificações do GitHub estão ativadas para que você não perca qualquer comentário relacionado a sua submissão.
- Os pacotes são verificados automaticamente no momento de submissão pelo nosso `pkgcheck`, que confirma se um pacote está ou não pronto para ser revisado.
- Os pacotes submetidos podem ser hospedados na ramificação principal/padrão ou em qualquer outra ramificação não padrão. Neste último caso, é recomendável, mas não obrigatório, enviar o pacote por meio de uma ramificação dedicada tipo `ropensci-software-review`.
- Para envios em ramificações (*branches*) que não sejam a padrão, o URL indicado em "Repository" no modelo de envio deve ser o URL completo da ramificação de revisão, como por exemplo: `https://github.com/my/repo/tree/ropensci-software-review`.

6.4 O Processo de Revisão

- Um editor editor analisará sua submissão em até 5 dias úteis e responderá com as próximas etapas. O editor poderá atribuir o pacote a revisores, solicitar que o pacote seja atualizado para atender aos critérios mínimos antes da revisão ou rejeitar o pacote devido à falta de adequação ou sobreposição.
- Se o seu pacote atender aos critérios mínimos, o editor designará de 1 a 3 revisores. Eles serão solicitados a fornecer revisões como comentários sobre a sua *issue* (submissão) dentro de 3 semanas.
- Pedimos que você responda aos comentários dos revisores em até 2 semanas após a última revisão enviada, mas você pode fazer atualizações no seu pacote ou responder a qualquer momento. Sua resposta deve incluir um link para a versão atualizada da sua NEWS.md do seu pacote. Aqui está um exemplo de resposta de autor. Incentivamos conversas contínuas entre autores e revisores. Consulte a seção guia de revisão para obter mais detalhes.
- Frequentemente, mudanças no pacote podem alterar os resultados automatizados das verificações `pkgcheck`. Para avaliar isso, autores podem solicitar uma nova verificação do pacote com o comando `@ropensci-review-bot check package`.
- Notifique-nos imediatamente se você não puder mais manter o seu pacote ou responder às revisões. Nestes casos, se espera que você retire a submissão ou que encontre mantenedores alternativos para o pacote. Você também pode discutir questões de manutenção na área de trabalho da rOpenSci no Slack.
- Assim que seu pacote for aprovado, forneceremos mais instruções sobre a transferência do seu repositório para o repositório da rOpenSci.

Nosso código de conduta é obrigatório para todos os envolvidos em nosso processo de revisão.

Capítulo 7

Guia para revisores

Obrigado por aceitar revisar um pacote para a rOpenSci! Este capítulo consiste em nossas diretrizes para preparar, enviar e acompanhar a sua revisão.

Você pode entrar em contato com o(a) editor(a) responsável pela submissão para esclarecer qualquer dúvida que tenha sobre o processo ou sobre a sua revisão.

Tente ao máximo concluir sua avaliação em até 3 semanas após o aceite da solicitação de avaliação. Nós vamos lembrar os(as) revisores(as) das datas de vencimento, tanto futuras quanto passadas. Os(as) editores(as) podem designar revisores(as) adicionais ou alternativos se uma revisão estiver muito atrasada.

A comunidade da rOpenSci é o nosso melhor patrimônio. Nosso objetivo é que as revisões sejam abertas, colaborativas e focadas na melhoria da qualidade do software. Seja respeitoso e gentil! Consulte o nosso guia para revisores e também o [código de conduta] (https://ropensci.org/code-of-conduct/) para obter mais informações.

Se você usar nossos processos, guias, listas de verificação, etc., ao revisar um software em outra plataforma, por favor, notifique os envolvidos (e.g., editores(as) de periódicos, estudantes ou equipes de revisão interna de código) que esses processos, padrões e materiais são provenientes da rOpenSci, e nos informe em nosso [fórum público] (https://ropensci.org/usecases) ou em [particular por e-mail] (https://ropensci.org/contact/).

7.1 Se voluntariando como revisor(a)

Obrigado pelo seu desejo de participar como revisor(a) da revisão por pares de software da rOpenSci!

Por favor, preencha o nosso formulário de voluntariado.

Se você encontrar alguma submissão que seja particularmente relevante para seus interesses, envie um e-mail para `info@ropensci.org`, incluindo o nome do pacote, a URL da *issue* que corresponde a este envio e o nome do(a) editor(a) responsável. No entanto, lembre-se de que o(a) editor(a) é responsável por indicar os(as) potenciais revisores(as), sendo possível que outras pessoas já tenham sido convidadas. Pedimos que não se candidate a todas as submissões e também evite se voluntariar diretamente pela interface do GitHub.

Para saber outras maneiras de contribuir, consulte o Guia de contribuição da rOpenSci.

7.2 Preparando a sua revisão

As revisões devem ser baseadas na versão mais recente da *branch* `main` no GitHub, a menos que os(as) autores(as) do pacote indiquem o contrário. Todos os envios criam um relatório detalhado, gerado pelo nosso pacote `pkgcheck`, sobre a estrutura e a funcionalidade do pacote enviado. Se o pacote tiver sido alterado substancialmente desde as últimas verificações, você pode solicitar uma nova verificação com o comando `@ropensci-review-bot check package`. Observe que, ao instalar o pacote para revisá-lo, você deve se certificar de que todas as dependências estão disponíveis em sua máquina (por exemplo, você pode executar `pak::pak()` para isso).

7.2.1 Diretrizes gerais

Para revisar um pacote, comece copiando o nosso modelo de revisão e use ele como uma lista de verificações que precisam ser feitas. Além de marcar os critérios mínimos, pedimos que você forneça comentários gerais abordando o seguinte:

- O pacote está de acordo com os requisitos do Guia de pacotes da rOpenSci?
- Há melhorias que poderiam ser feitas no estilo e nos padrões de código? Por exemplo, as funções precisam ser divididas em funções auxiliares menores e o papel de cada função auxiliar está claro?
- Há duplicação de código no pacote que deveria ser reduzida?
- Existem funções no R básico ou na dependência leve que fornecem a mesma interface que algumas funções auxiliares no pacote?
- Há melhorias na interface do usuário que poderiam ser feitas?
- Existem melhorias de desempenho que podem ser feitas?
- A documentação (instruções de instalação, *vignettes*, exemplos e demonstrações) é clara e suficiente? O princípio de *vários pontos de entrada* está sendo usado? Ou seja, a documentação leva em conta o fato de que qualquer parte

da documentação pode ser o primeiro encontro do usuário com o pacote e/ou com as ferramentas e os dados que ele contém?
- As funções e os argumentos foram nomeados para trabalharem juntos e formarem uma API de programação comum e lógica que seja fácil de ler e de autocompletar?
- Se você tiver seus próprios dados/problemas relevantes, analise-os com o pacote. Você pode acabar encontrando casos de uso extremos nos quais o autor não pensou.

Seja respeitoso e gentil com os(as) autores(as) em suas avaliações. Nosso código de conduta é obrigatório para todos os envolvidos em nosso processo de avaliação. Esperamos que você envie sua avaliação em até 3 (três) semanas, de acordo com o prazo estabelecido pelo(a) editor(a). Entre em contato com o(a) editor(a) diretamente ou no tópico de envio para informá-lo sobre possíveis atrasos.

Incentivamos você a usar ferramentas automatizadas para facilitar a revisão. Isso inclui:

- Verificar o relatório inicial do pacote gerado pelo nosso `@ropensci-review-bot`.
- Verificar os registros (logs) nos serviços de integração contínua utilizados pelo pacote (GitHub Actions, Codecov, etc.)
- Executar `devtools::check()` e `devtools::test()` no pacote para localizar quaisquer erros que possam ter passado despercebidos pelo(a) autor(a).
- Verificar se a omissão de testes é justificada (e.g., o uso de `skip_on_cran()` em testes que fazem solicitações reais de API vs. ignorar todos os testes em um sistema operacional).
- Se o pacote não for enviado por meio da *branch* padrão/principal, lembre-se de mudar para a *branch* de revisão que foi submetida antes de iniciar sua revisão. Nesse caso, você também terá que pesquisar o pacote localmente, dado que a pesquisa no GitHub é limitada à *branch* padrão. Além disso, documentações hospedadas em um website tipo `pkgdown` não são necessariamente atualizadas, por isso recomendamos que você inspecione a documentação do pacote localmente, executando `pkgdown::build_site()`.

Os(as) revisores(as) também podem gerar novamente os resultados da verificação de pacotes a partir de `@ropensci-review-bot` a qualquer momento, emitindo um único comentário em um *issue* de revisão: `@ropensci-review-bot check package`.

Alguns elementos de nossa lista de verificação são inspirados no [guia de revisão de código da Mozilla] (https://mozillascience.github.io/codeReview/review.html).

7.2.2 Interações feitas fora dos canais oficiais

Se você interagiu com os(as) autores(as) do pacote e falou sobre a revisão fora de um tópico de revisão (como em *chats*, mensagens diretas, pessoalmente ou *issues* no repositório do projeto), certifique-se de que sua revisão capture e/ou referencie elementos dessas conversas que sejam relevantes para o processo.

7.2.3 Experiências de revisores(as) anteriores

Revisores(as) iniciantes podem achar útil ler sobre algumas revisões anteriores. Em geral, você pode encontrar os tópicos de envio de pacotes que já foram integrados. Aqui, estão alguns exemplos de revisões (observe que suas revisões não precisam ser tão longas quanto esses exemplos):

- revisão 1 e revisão 2 de `rtika`
- `NLMR` revisão 1 e revisão2
- `bowerbird` comentário pré-revisão, revisão 1, revisão 2.
- `rusda` revisão (de antes de termos um modelo de revisão)

Você pode ler algumas publicações escritas por avaliadores(as) sobre as suas experiências através deste link. Em especial, nesta postagem de Mara Averick, você pode ler sobre o papel de "usuário ingênuo" que um(a) revisor(a) pode assumir para fornecer um feedback útil, mesmo sem ser especialista no tópico ou na implementação do pacote, ao se perguntar: *"O que eu achava que seria feito? Realmente, faz isso? Quais são os detalhes que me assustam?"*. Em outra postagem, Verena Haunschmid compartilha sua experiência de alternar entre usar o pacote e revisar seu código.

O ex-revisor e autor de pacotes Adam Sparks escreveu o seguinte "[escreva] uma boa crítica da estrutura do pacote e das práticas recomendadas de codificação. Se você souber como fazer algo melhor, diga-me. É fácil perder oportunidades de documentação como desenvolvedor(a); porém, como revisor(a), você tem uma visão diferente. Você é um usuário que pode dar um *feedback*. O que não está claro no pacote? Como você poderia deixar isso mais claro? Se você estiver usando-o pela primeira vez, é fácil? Você conhece algum outro pacote R que talvez eu devesse usar? Ou há algum que eu esteja usando e que talvez não devesse usar? Se você puder contribuir com o pacote, então contribua."

7.2.4 Pacote de ajuda para revisores

Se estiver trabalhando no RStudio, você pode otimizar o fluxo de trabalho de revisão usando o pacote `pkgreviewr`, criado pela editora associada Anna Krystalli. Digamos que você tenha aceitado revisar o pacote `refnet`, então você escreveria o seguinte:

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

Como resultado:

- o repositório do GitHub do pacote `refnet` será clonado.
- um projeto de revisão será criado, contendo um bloco de notas para você preencher, já usando o modelo de revisão recomendado.
- observe que, se o pacote não for enviado por meio da *branch* padrão/principal, você precisará mudar para a *branch* enviada antes de iniciar a revisão.

7.2.5 Comentários sobre o processo

Incentivamos você a fazer perguntas e fornecer feedbacks sobre o processo de revisão em https://github.com/ropensci/software-review-meta/issues

7.3 Envio da revisão

- Quando sua revisão estiver concluída, cole-a como um comentário na *issue* onde a revisão do pacote está sendo feita/tratada.
- Comentários adicionais são bem-vindos na mesma issue. Esperamos que as revisões de pacotes funcionem como uma conversa contínua com os(as) autores(as), em vez de uma única rodada de revisões típica de manuscritos acadêmicos.
- Você também pode enviar *issues* ou *Pull Requests* diretamente para o repositório dos pacotes, se preferir, mas, se o fizer, comente sobre eles e coloque um link para eles dentro do tópico onde a revisão do pacote está sendo tratada, para que tenhamos um registro centralizado da sua revisão.
- Inclua uma estimativa de quantas horas você gastou na revisão.

7.4 Acompanhamento da revisão

Os(as) autores(as) devem responder dentro de 2 (duas) semanas com as alterações feitas no pacote em resposta à sua avaliação. Nesse estágio, pedimos que você avalie se as alterações são suficientes para resolver as questões levantadas em sua revisão. Incentivamos a discussão contínua entre autores(as) de pacotes e revisores(as), e você também pode pedir aos editores que esclareçam os problemas no tópico de revisão.

Você usará o modelo de aprovação.

Capítulo 8

Guia para a Equipe Editorial

A revisão por pares de software na rOpenSci é gerenciada por uma equipe editorial. A função de Editor(a)-chefe (do inglês EiC) é alternada trimestralmente entre os(as) membros(as) experientes do nosso conselho editorial. As informações sobre o status atual de todas as pessoas da equipe editorial estão disponíveis em nosso [*Painel Editorial*] (#eic-dashboard).

Este capítulo apresenta as responsabilidades do(a) Editor(a)-chefe e dos(as) editores(as) responsáveis pelas submissões. Ele também descreve como responder a uma submissão fora do escopo e fornece orientação sobre responder às perguntas dos(as) revisores(as).

Se você for um editor convidado, obrigado por ajudar! Entre em contato com o editor que o convidou para lidar com uma submissão para esclarecer qualquer dúvida que você possa ter.

A comunidade da rOpenSci é o nosso melhor patrimônio. Nosso objetivo é que as revisões sejam abertas, não conflituosas e focadas na melhoria da qualidade do software. Seja respeitoso(a) e gentil! Consulte o nosso guia para revisores e também o [código de conduta] (https://ropensci.org/code-of-conduct/) para obter mais informações.

Este capítulo está estruturado para refletir a progressão típica de uma revisão de software na rOpenSci. Todas as submissões são inicialmente consideradas pelo(a) Editor(a)-chefe, que toma a decisão inicial sobre a adequação do pacote ao escopo. Em caso positivo, é designado um(a) editor(a) responsável por guiar o processo de revisão propriamente dito.

8.1 Responsabilidades do papel de Editoria-Chefe (EiC)

As pessoas que exercem o papel de Editoria-Chefe rotativo (EiCs) geralmente servem por 3 meses ou por um período acordado por toda a equipe do conselho editorial. Quem ocupa o papel de EiC é responsável pelo gerenciamento geral do processo de revisão e pelos estágios iniciais de todas as submissões.

8.1.1 Deveres gerais do papel de EiC

A pessoa que exerce o papel de EiC gerencia todas as *issues* de revisão de software com a ajuda do nosso painel editorial conforme descrito a seguir na subseção sobre o painel. As responsabilidades de EiC incluem as seguintes tarefas gerais:

- No início de um rodízio, deve revisar o status das revisões abertas atuais no painel editorial e emitir lembretes para outros(as) editores(as) ou autores(as) de pacotes, conforme necessário.
- Acompanhar todas as novas *issues* postados no repositório de revisão de software (`software-review`), para os quais deve ativar as notificações de repositório no GitHub, e também acompanhar o canal `#editors-only` no Slack.
- Monitorar regularmente (por exemplo, semanalmente) o ritmo de todas as revisões abertas, observando o *Painel de controle* e lembrar outros(as) editores(as) de mover pacotes conforme o necessário.

8.1.2 Deveres de EiC para cada submissão inicial

A pessoa que exerce o papel de EiC é responsável pelo processamento inicial de todas as novas submissões. Suas principais funções são:

1. Decidir se uma submissão deve ou não ser considerado no escopo e prosseguir com a análise e, se for o caso,
2. Prosseguir com a submissão completa e designar um(a) editor(a) responsável.

8.1.2.1 Decidir sobre o escopo e a sobreposição

A pessoa que exerce o papel de EiC tem o direito de tomar decisões sobre o escopo e a sobreposição da forma mais independente possível, mas é recomendado solicitar opiniões no canal `#editors-only` no Slack. As decisões de escopo para software estatístico geralmente são mais fáceis do que para submissões gerais (não estatísticas), como descrito abaixo. Para cada nova pré-submissão ou submissão, a pessoa que exerce o papel de EiC deve:

- Consultar as categorias descritas na seção *Objetivos e escopo* para tomar decisões sobre o escopo e a sobreposição para consultas de pré-submissão, indicações do JOSS ou de outros parceiros de publicação e submissões.
  - Inicie discussões no canal `#editors-only` do Slack da rOpenSci, escrevendo um resumo do software (pré-)submetido, juntamente com quaisquer preocupações que possa ter.
  - Se achar que não recebeu respostas suficientes após um ou dois dias, pode enviar uma mensagem no Slack mencionando toda a equipe editorial.
  - Deve buscar outras opiniões sobre as submissões que estejam além de suas próprias áreas de especialização.
  - O software estatístico deve ser considerado dentro do escopo, desde que cumpra pelo menos metade de todas as normas aplicáveis (gerais e pelo menos uma categoria).
- Se uma pré-submissão ou submissão for considerada fora do escopo, a pessoa que exerce o papel de EiC deverá agradecer as pessoas que submeteram o pacote, explicar os motivos da decisão e direcionar a outros locais de publicação, se for o caso. Quando for relevante, use o texto de *Objetivos e escopo* com relação à evolução do escopo ao longo do tempo.
  - Exemplos de submissões e respostas fora do escopo.
  - Depois de explicar uma decisão fora do escopo, comente na *issue*: `@ropensci-review-bot out-of-scope`.
- Se uma consulta de pré-submissão for considerada dentro do escopo, a pessoa que exerce o papel de EiC *pode* realizar verificações preliminares. O *modelo de editores* pode ser usado para isso. Para auxiliar as respostas dos(as) autores(as) aos comentários editoriais, é útil usar uma notação inequívoca para cada comentário, como:

  ```
  Meus comentários estão etiquetados com "EIC" e uma sequência numérica. Por favor, use esta n

  **EIC01** Por favor, melhore o README
  ```

  Também pode ser útil distinguir requisitos de recomendações, por exemplo, formatando os requisitos como caixas de seleção (`- [ ] **EIC01**`). É claro que você pode usar os prefixos que quiser, inclusive suas próprias iniciais, como neste exemplo de um de nossos editores, Mauro Lepore.

As decisões sobre o escopo podem exigir informações adicionais de quem submeteu o pacote. A pessoa que ocupa o papel de EiC deve garantir, no mínimo, que a documentação seja suficiente para julgar o escopo, incluindo um site que a acompanhe. Se não for o caso, solicite mais detalhes; mesmo que um pacote seja considerado fora do escopo, solicitar melhorias na documentação do pacote só ajudará outras pessoas a entenderem o pacote. Este é um exemplo de solicitação:

```
Olá <username>, agradecemos muito por sua submissão.

Estamos discutindo se o pacote está dentro do escopo e precisamos de um pouco mais de i

Você poderia adicionar mais detalhes e contexto ao README?
Após a leitura do README, alguém com pouco conhecimento específico sobre o tema deve se

<opcional>
Se um pacote possui funcionalidades que se sobrepõem às de outros pacotes, solicitamos
Você poderia adicionar ao README uma comparação mais detalhada em relação aos pacotes r
</opcional>
```

8.1.2.2 Deveres iniciais de EiC para submissões completas

- Quando a pessoa que ocupa o papel de EiC estiver convencida de que um pacote pode prosseguir para uma submissão completa, convide as pessoas que submeteram o pacote a abrir uma *issue* de submissão completa e, em seguida, feche a *issue* de consulta de pré-submissão.
- Inicialmente, marque cada novo envio completo com `0/editorial-team-prep`
- Encontre alguém da equipe editorial para que fique responsável pelo envio (incluindo você, potencialmente). As pessoas da equipe editorial atualmente disponíveis estão indicadas no *Painel editorial* As cargas de trabalho editoriais devem ser distribuídas da maneira mais uniforme possível, consultando os gráficos de cargas editoriais recentes no *Painel editorial*. A pessoa que ocupa o papel de EiC também pode recrutar alguém como editora convidada para acompanhar alguma submissão, conforme descrito na subseção abaixo.
- Atribua uma pessoa para realizar a edição, escrevendo um comentário na *issue* de revisão:

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

  Isso também adicionará na *issue* a etiqueta `1/editor-checks`.

8.1.2.3 Submissões de software estatístico

O software estatístico deve ser considerado no escopo, desde que esteja em conformidade com > 50% de todas as normas aplicáveis. Se uma consulta pré-submissão indicar que os padrões já foram cumpridos, a pessoa que ocupa o papel de EiC deverá:

- Use `@ropensci-review-bot check srr` para confirmar que as normas foram cumpridas.
- Considere se o pacote é melhor colocado na categoria estatística indicada ou se categorias alternativas ou adicionais podem ser apropriadas.

Se as normas ainda não tiverem sido cumpridas, a pessoa que ocupa o papel de EiC deve pedir para os(as) autores(as) que avaliem se acham que o pacote será capaz de cumprir pelo menos metade de todos os normas gerais e específicas da categoria. Isso pode exigir uma discussão sobre a categoria ou categorias apropriadas. Se o(a) autor(a) não tiver cumprido as normas, mas concordar em fazê-lo, geralmente deverá ser aplicado uma etiqueta de "Holding" na *issue*, até que o `@ropensci-review-bot check srr` dê uma aprovação (☒). Os detalhes completos sobre o acompanhamento de submissões de software estatístico pelo(a) EiC são fornecidos no capítulo correspondente do *Guia de desenvolvimento de pacotes estatísticos*.

8.1.3 O Painel Editorial da rOpenSci

O *Painel Editorial da rOpenSci* é atualizado diariamente, principalmente pela extração de informações sobre todas as *issues* de revisão de software no GitHub, juntamente com informações adicionais do Slack e do nosso banco de dados no Airtable. O painel fornece uma visão geral atualizada da nossa equipe editorial, suas responsabilidades recentes e o estado atual de todas as *issues* de revisão de software. Quem ocupa o papel de EiC (ou qualquer pessoa interessada) pode obter uma visão geral do status, da disponibilidade e das cargas de trabalho recentes da equipe editorial na página sobre a *equipe editorial*. Ela deve ser usada para encontrar e designar editores(as) para novas edições de revisão de software. Uma visão geral de todas as revisões de software atuais está disponível na página de *Revisão de Software*, com linhas coloridas por "urgência", resumida na tabela na parte inferior da página.

As tarefas para revisões nos estágios específicos de revisão incluem:

- Revise os envios em `0\/presubmission` e `1\/editorial-team-prep` para verificar se alguma ação precisa ser tomada (como sondar a equipe editorial, tomar decisões, colocar edições em espera, mencionar pessoas para solicitar atualizações ou encontrar e designar pessoas editoras).

- Revise os envios em `2\/seeking-reviewer(s)` para garantir que as coisas estejam progredindo rapidamente. Se a busca de revisores(as) estiver demorando muito (cor vermelha), verifique se o envio está em espera, leia a conversa da *issue* para obter o contexto e entre em contato com a pessoa editora responsável no privado para pedir mais informações.

- Revise os envios em `3\/reviewer(s)-assigned`. Se ainda faltarem avaliações depois de um tempo muito longo (cor vermelha), verifique se o envio está

em espera, leia a conversa da *issue* para obter o contexto e entre em contato com a pessoa editora responsável no privado para pedir mais informações.

- Revise os envios em `4\/review(s)-in-awaiting-changes`. Se alguns ainda não tiverem uma resposta do(a) autor(a) depois de um tempo excepcionalmente longo (cor vermelha), verifique se o envio está em espera, leia a conversa da *issue* e entre em contato com a pessoa editora responsável no privado para pedir mais informações.

8.1.4 Convidando uma pessoa para a tarefa de edição

Depois de discutir com a equipe editorial, o(a) EiC pode convidar uma pessoa externa para acompanhar uma submissão. Pode ser desejado ou necessário convidar pessoas para a tarefa de edição se houver necessidade de especialização em um domínio específico, se o volume atual de envios for grande, se potenciais editores(as) tiverem conflitos de interesse ou como um teste antes de convidar uma pessoa para fazer parte da equipe editorial.

Para decidir sobre convidar alguém para a tarefa de edição de uma submissão:

- Crie um canal privado do Slack chamado #guest-editor-.
- Convide a equipe editorial para este canal.
- Busque os canais arquivados semelhantes do Slack para obter possíveis recomendações.
- Depois que uma decisão for tomada e a pessoa convidada tenha aceitado, arquive o canal.

Quando você convidar uma pessoa externa para a edição,

- Pergunte sobre conflitos de interesse usando o mesmo texto usado para revisores(as).
- Forneça o link para o guia para editores.

Se a pessoa aceitar o convite (eba!),

- Certifique-se de que a pessoa tenha habilitado a autenticação 2FA em sua conta do GitHub,
- Convide-a para a equipe `ropensci/editors` e para a organização ropensci no GitHub,
- Depois que ela aceitar o convite do repositório, atribua a *issue* a ela,
- Certifique-se de que ela (já) esteja convidada para o espaço de trabalho do Slack da rOpenSci,
- Peça a alguém da equipe para adicionar o nome da pessoa convidada à tabela de editores(as) convidados(as) do Airtable (para que seus nomes apareçam neste livro e no README da revisão do software).

Depois que o processo de revisão for concluído (pacote aprovado, *issue* fechada),

- Agradeça novamente à pessoa convidada,
- Remova-o da equipe `ropensci/editors` no GitHub (mas não da organização ropensci).

8.2 Responsabilidades gerais dos(as) editores

- Além de lidar com as submissões, conforme descrito na seção a seguir, os(as) editores opinam sobre as decisões editoriais do grupo, como, por exemplo, se as submissões estão dentro do escopo, e orientam as atualizações de nossas políticas. Geralmente, fazemos isso por meio do Slack, que esperamos que os(as) editores possam verificar regularmente.
- Você só precisa acompanhar suas próprias submissões, mas se notar um problema com um pacote que está sendo acompanhado por outra pessoa, sinta-se à vontade para levantar esse problema diretamente com a pessoa responsável pela edição, ou escrever uma mensagem com a preocupação no canal exclusivo para a equipe editorial no Slack (`editors-only`). Por exemplo:
  - Você sabe de um pacote que gere uma sobreposição, e que ainda não foi mencionado no processo.
  - Você vê uma pergunta para a qual tem uma resposta especializada que não foi dada depois de alguns dias (como um link para uma publicação de blog que pode responder a uma pergunta).

8.3 Responsabilidades de edição de um pacote

As pessoas designadas como editoras de um pacote (*handling editors*) são responsáveis por orientar cada submissão designada durante todo o processo, desde o envio inicial até o aceite final. Cada pessoa da equipe editorial não deve ser designada para mais de uma edição por trimestre, ou no máximo quatro por ano.

8.3.1 Após a submissão

8.3.1.1 Checagens automáticas

- A submissão gerará automaticamente o resultado da checagem de pacotes do robô de revisão da rOpenSci (`ropensci-review-bot`). Todas as falhas de checagem (☒) devem ser corrigidas antes de prosseguir, embora exceções possam ser justificadas a critério da pessoa responsável pela edição.

Incentive a pessoa que submeteu o pacote a acionar as checagens chamando `@ropensci-review-bot check package` para confirmar que todas as verificações foram aprovadas (☒).
- Examine todos os itens marcados com ☒ e solicite mais informações ou ações da pessoa que submeteu o pacote, quando apropriado.
- Para submissões estatísticas (identificáveis como "Submission Type: Stats" no modelo da edição), adicione a etiqueta "stats" à *issue* de edição (se ainda não tiver sido adicionada).
  - Se a pessoa que submeteu o pacote apontar que "incorporou a documentação de normas… por meio do pacote srr", então chame `@ropensci-review-bot check srr` para confirmar que pelo menos 50% de todas as normas foram cumpridas.
- Verifique se o modelo de *issue* foi preenchido corretamente. Os erros e omissões mais comuns devem ser detectados e anotados pelo robô, mas uma verificação manual sempre ajuda. As pessoas editoras podem editar os modelos diretamente para fazer pequenas correções ou podem orientar os(as) autores(as) a preencher os campos obrigatórios do modelo que possam estar faltando.
- O sistema de checagem passa por um rebuild toda terça-feira, às 00:01 UTC, e pode levar algumas horas. Se as checagens automáticas falharem nesse horário, aguarde algumas horas e tente novamente.

As chegagens automáticas também incluem uma seção expansível de *Propriedades estatísticas* que fornece detalhes. A seção expansível sinaliza quaisquer propriedades "notáveis", definidas como aspectos que se encontram no quinto percentil superior ou inferior em comparação com os valores de todos os pacotes atuais do CRAN. Preste atenção especial a:

- Valores extremamente grandes ou pequenos para linhas de código ("loc").
- Valores extremamente grandes ou pequenos para números de funções ("n_fns_…").

O relatório automatizado também incluirá um link para uma página de *visualização de rede interativa de chamadas entre objetos no pacote*. Essa visualização pode fornecer informações úteis sobre como as funções em um pacote são estruturadas. Use todas essas informações para avaliar se:

- Um pacote tem muito poucas funções ou linhas de código e, nesse caso, pode não estar suficientemente desenvolvido para a revisão por pares.
- Um pacote é extremamente grande e, nesse caso, pode acabar sobrecarregando os(as) revisores. Por exemplo, essa checagem inicial revelou um pacote com mais de 8.500 linhas de código R (correspondendo a 97,6% de todos os pacotes) e 251 funções em R exportadas.

8.3.1.2 Comentários editoriais iniciais

- Depois que as checagens automáticas forem postadas, use o modelo de edição para orientar as verificações iniciais (se ainda não estiverem cobertas pelo(a) EiC) e registre sua resposta na *issue* de submissão. Você também pode simplificar as verificações de editor(a) usando o pacote `pkgreviewr`, criado pela ex-editora Anna Krystalli. Por favor, tente concluir as verificações e começar a buscar revisores(as) dentro de 5 dias úteis.
- Verifique as políticas em busca de possíveis ajustes e sobreposições. Se necessário, inicie uma discussão por meio do canal #software-review do Slack para casos extremos que não tenham sido detectados por verificações anteriores do EiC. Se houver rejeições, consulte esta seção sobre como responder.
- Para pacotes que precisam de integração contínua em várias plataformas (veja os critérios nesta seção do capítulo sobre CI), certifique-se de que o pacote seja testado em várias plataformas (tendo o pacote criado em vários sistemas operacionais por meio do GitHub Actions, por exemplo).
- Sempre que possível, ao solicitar alterações, sugira ferramentas automáticas, como usethis, Air, Jarl, flir, e para recursos on-line (como seções deste guia, seções do livro *R Packages*) para facilitar o uso do seu feedback. Veja este exemplo de verificações de uma editora.
- O ideal é que você faça observações e estas sejam resolvidas antes que revisores comecem a revisar.
- Se as verificações iniciais mostrarem grandes lacunas, solicite alterações antes de designar revisores. Se o(a) autor(a) mencionar que as alterações podem levar tempo, aplique a etiqueta de espera escrevendo `@ropensci-review-bot put on hold`. Você receberá um lembrete a cada 90 dias na *issue* de edição para verificar com os(as) autores(as) do pacote.
- Se o pacote levantar uma nova questão em relação às políticas da rOpenSci, inicie uma conversa no Slack.

8.3.2 Busque e atribua duas pessoas para revisar o pacote

8.3.2.1 Encontrar revisores(as)

- Escreva na *issue* `@ropensci-review-bot seeking reviewers`.
- Use o modelo de e-mail, se necessário, para convidar pessoas para a revisão
  - Ao convidar revisores(as), inclua algo como "se eu não tiver notícias suas em uma semana, presumirei que você não poderá revisar", para dar um prazo claro para que você procure outra pessoa.
  - Você pode enviar vários convites ao mesmo tempo, especialmente porque isso geralmente ajuda a encontrar revisores(as) mais rapidamente. Você sempre pode responder dizendo que uma pessoa já foi encontrada para fazer a revisão, e agradecer às pessoas por se oferecerem.

As fontes de informação para encontrar revisores(as) incluem:

- Sugestões possíveis feitas por quem submeteu o pacote, (embora possam ter uma visão restrita dos tipos de conhecimentos necessários; sugerimos que você não use mais que um dos nomes indicados);
- Nosso banco de dados Airtable de revisores(as) e voluntários(as) (consulte a próxima subseção);
- Autores de pacotes rOpenSci semelhantes.

Quando essas fontes de informação não forem suficientes,

- peça ideias para a equipe editorial no Slack.
- Busque pessoas que usam o pacote ou o serviço/fonte de dados ao qual o pacote se conecta (verificando quem abriu *issues* no repositório, marcou o repositório com estrela, citou em artigos ou mencionou o pacote nas redes sociais).
- Você também pode procurar autores(as) de pacotes relacionados em r-universe.dev.
- A R-Ladies tem um diretório especificando as habilidades e os interesses das pessoas listadas.
- Você pode publicar uma busca por revisores(as) nos canais #general e/ou #software-review no Slack da rOpenSci ou nas redes sociais.

8.3.2.1.1 Dicas para a pesquisa de revisores(as) no Airtable

Você pode usar filtros, classificações e pesquisas para identificar revisores(as) com experiência específica:

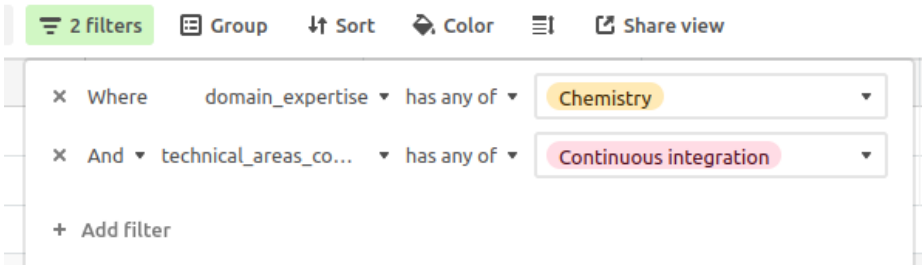

Figura 8.1: Captura de tela da interface em inglês de filtros do Airtable com um filtro de experiência de domínio que deve incluir química e áreas técnicas que devem incluir integração contínua

Por favor, verifique a avaliação mais recente do revisor(a) e evite qualquer pessoa que tenha avaliado alguém nos últimos seis meses. Além disso, verifique se um revisor(a) iniciante indicou que precisou de mentoria (`require_mentorship`). Nestes casos, use a parte de orientação do modelo de e-mail e esteja preparado para fornecer orientações adicionais.

8.3.2.1.2 Critérios para escolher um(a) revisor(a)

Aqui estão os critérios que você deve ter em mente ao escolher alguém para revisar um pacote. Talvez você precise reunir essas informações pesquisando no *r-universe*, no perfil do GitHub do(a) possível revisor(a) e na presença on-line em geral (site pessoal, redes sociais).

- Não revisou um pacote para nós nos últimos 6 meses.
- Tem alguma experiência em desenvolvimento de pacotes.
- Tem alguma experiência de domínio no campo do pacote ou da fonte de dados.
- Não tem conflitos de interesse.
- Tente equilibrar sua percepção da experiência do(a) possível revisor(a) com a complexidade do pacote.
- Diversidade - com duas pessoas revisoras, ambas não devem ser homens brancos cis.
- Alguma evidência de que a pessoa tenha interesse em práticas abertas ou em atividades da comunidade de R, embora o contato direto via e-mail seja válido.

Cada submissão deve ser analisada por *duas* pessoas revisoras de pacotes. Embora seja bom que um(a) deles tenha menos experiência em desenvolvimento de pacotes e mais conhecimento do domínio, a revisão não deve ser dividida em dois. Ambos(as) os(as) revisores(as) precisam revisar o pacote de forma abrangente, embora sob suas perspectivas específicas. Em geral, pelo menos um(a) revisor(a) deve ter experiência prévia em revisão e, é claro, convidar uma nova pessoa amplia nosso grupo de revisores(as).

8.3.2.2 Atribuir revisores(as)

- Atribua a cada revisor(a) com o comando `@ropensci-review-bot assign @username as reviewer`. Um comando deve ser emitido para cada revisor(a). Se necessário remover alguém da revisão, use o comando `@ropensci-review-bot remove @username from reviewers`.
- As datas de vencimento das revisões são definidas por padrão como 21 dias (3 semanas) após a data da atribuição.
- Se você quiser alterar a data de vencimento de uma revisão, use `@ropensci-review-bot set due date for @username to YYYY-MM-DD`.

8.3.3 Durante a revisão

- Consulte ocasionalmente com as pessoas revisoras e autoras do pacote. Ofereça esclarecimentos e ajuda conforme necessário.

- Em geral, tente estabelecer um prazo de 3 semanas para a revisão, 2 semanas para os ajustes solicitados, e 1 semana para que os(as) revisores(as) aprovem as mudanças.
- Após o envio de cada revisão,
  - Escreva um comentário agradecendo o(a) revisor(a), com suas próprias palavras.
  - Registre a revisão escrevendo um novo comentário `@ropensci-review-bot submit review <review-url> time <time in hours>` como neste exemplo.
- Quando as alterações forem feitas, altere a etiqueta de status da revisão para `5/awaiting-reviewer-response` e solicite que os(as) revisores(as) indiquem a aprovação usando o modelo de aprovação de revisão.
- Se as pessoas autoras tiverem a intenção de enviar um manuscrito de acompanhamento do tipo *Applications* no periódico Methods in Ecology and Evolution, indique que o envio do manuscrito pode ser feito após a conclusão da revisão.

8.3.3.1 Respondendo às perguntas dos(as) revisores(as)

Tanto os(as) autores(as) quanto os(as) revisores(as) podem solicitar feedback sobre aspectos de um processo de revisão, incluindo, por exemplo, o tom ou a substância das revisões ou das respostas dos(as) autores(as). Embora este guia se esforce para fornecer orientação suficiente para a maioria dos casos, outras pessoas da equipe editorial estão sempre à disposição para esclarecer dúvidas. Se você notar algo que possa ser editado ou adicionado de forma produtiva neste *Guia de desenvolvimento*, por favor, abra uma *issue* ou um *pull request*. Além desses princípios gerais, essas revisões fornecem exemplos úteis:

- Um exemplo difícil, mas construtivo, que sugere uma reescrita da *vignette*: ropensci/software-review#191 (comentário).
- O pacote `slopes`, que acabou sendo fundamentalmente redesenhado em resposta às revisões. Todas as revisões foram sempre totalmente construtivas, o que parece ter desempenhado um papel importante na motivação dos(as) autores(as) para embarcar em uma revisão tão grande. Comentários como, *"este pacote não …"* ou *"não tem …"* eram invariavelmente seguidos de sugestões construtivas sobre o que poderia ser feito (há, por exemplo, várias sugestões em uma das primeiras revisões).
- As revisões do pacote tic apresentaram algumas ressalvas educadamente: https://github.com/ropensci/software-review/issues/305#issuecomment-504762517 e https://github.com/ropensci/software-review/issues/305#issuecomment-508271766
- O pacote bowerbird recebeu uma "pré-revisão" útil, que resultou na divisão do pacote antes das revisões propriamente ditas.

Outros desafios que podem surgir durante a revisão incluem:

- **Se o(a) autor(a) parar de responder** consulte as políticas e/ou envie uma mensagem para a equipe editorial no canal do Slack para discussão. É importante ressaltar que, se um(a) revisor(a) foi atribuído(a) a uma *issue* fechada, entre em contato com ele(a) quando fechar a *issue* para explicar a decisão e agradecer mais uma vez pelo trabalho. Informe a equipe editorial no canal Slack para que eles(as) sejam considerados revisores(as) de um pacote no futuro, com grandes chances de uma revisão de software tranquila.
- **Se um(a) revisor(a) estiver em atraso com a revisão ou parar de responder**, envie um lembrete após uma semana e novamente após duas semanas. O primeiro lembrete pode ser uma menção (`@tag`) no GitHub. Depois disso, use o e-mail ou outra forma de comunicação direta. Se após três semanas ainda não houver resposta, determine a melhor maneira de seguir em frente sem essas pessoas:
  - Se o(a) revisor(a) já tiver enviado sua revisão primária e outro(a) revisor(a) estiver ativo(a) e fornecendo feedback substancial, o(a) editor(a) poderá prosseguir com o processo de revisão e deverá assumir o papel do(a) revisor(a) ausente para determinar se as alterações dos(as) autores(as) são suficientes.
  - Se o(a) revisor(a) ausente não tiver enviado sua revisão, o(a) editor(a) deverá tentar encontrar uma nova pessoa para fazer a revisão e prosseguir com o processo quando houver duas revisões. Nesse ponto, o(a) editor(a) deve priorizar a busca de revisores(as) experientes que possam se comprometer com uma resposta rápida. Não se esqueça de enviar mensagens para a equipe editorial no Slack.
    - A seu critério, o(a) editor(a) *pode* optar por realizar a segunda revisão, mas só deve fazê-lo após várias tentativas fracassadas de encontrar um(a) novo(a) revisor(a) e se o(a) editor(a) tiver experiência suficiente para fazê-lo. Não recomendamos que os(as) editores(as) façam isso com frequência, pois isso aumenta a carga de trabalho e reduz a diversidade de pontos de vista trazidos à comunidade pelos(as) revisores(as).
    - Em qualquer caso, escreva um comentário agradecendo a pessoa revisora original e a remova com o comando `@ropensci-review-bot remove @username from reviewers`.

8.3.4 Após a revisão

- Aprove o pacote com `@ropensci-review-bot approve <package-name>`
- Consulte a seção a seguir para saber como permitir que os(as) autores(as) mantenham o pacote em sua própria organização do GitHub se preferirem não transferir para a rOpenSci.

- Indique o pacote para que seja apresentado em uma publicação ou nota técnica do blog da rOpenSci, se você achar que ele pode ser de grande interesse. Anote na *issue* de revisão do software uma ou duas coisas que o(a) autor(a) poderia destacar e marque `@ropensci/blog-editors` para acompanhamento.
- Se os(as) autores(as) mantiverem um livro on-line que seja, pelo menos em parte, sobre o pacote, entre em contato com alguém da equipe da rOpenSci para que possam entrar em contato com os(as) autores(as) sobre a transferência para a organização do GitHub `ropensci-books`.

8.3.4.1 Pacotes que permanecem nas organizações originais do GitHub

Para autores(as) de pacotes que desejam manter seus repositórios em suas organizações originais do GitHub, em vez de transferi-los para github.com/ropensci, os editores devem:

- Pedir aos autores de pacotes que façam um pull request para o arquivo JSON que lista todos os repositórios que não foram transferidos. Exemplo de commit.
- Pedir aos autores do pacote que substituam o conteúdo do código de conduta atual do repositório pelo conteúdo do código de conduta padrão da organização rOpenSci no GitHub.

8.3.4.2 Divulgação de pacotes

- Direcione o(a) autor(a) para os capítulos do guia sobre lançamento de pacotes, marketing e como melhorar a performance no GitHub.

Capítulo 9

Gerenciamento editorial

Orientações para gerenciar a equipe editorial e também sobre [como gerenciar o lançamento de um guia de desenvolvimento] (#bookrelease).

9.1 Recrutamento de novos editores

O recrutamento de novos editores e a manutenção de um conselho editorial suficiente e equilibrado é responsabilidade da Líder de Revisão de Software com o apoio e a orientação do conselho editorial.

Etapas:

- Inicie um canal privado para discussão (para que você não tenha um histórico no canal de editores no qual futuros editores entrarão, o que pode ser incômodo).
- Marque os editores na conversa para garantir que eles recebam uma notificação, pois este é um tópico importante.
- Espere que a maioria dos editores se manifeste antes de convidar alguém. Dê a eles uma semana para responder.

9.2 Convidando um(a) novo(a) editor(a)

- Um(a) candidato(a) pode começar como um(a) editor(a) convidado(a).
- Envie um e-mail.

```
Gostaríamos de convidá-lo a fazer parte do conselho editorial da rOpenSci como membro e
Acreditamos que você seria um excelente incremento à equipe.

[SE FOR EDITOR CONVIDADO -> Você está familiarizado(a) com a função do editor, pois já
Pedimos que editores assumam um compromisso informal de servir por dois anos, reavaliar
Em um curto prazo, qualquer editor pode se recusar a lidar com um pacote ou argumentar

Além de lidar com pacotes, os editores participam das decisões editoriais do grupo, com
Geralmente, fazemos esse trabalho por meio do Slack, que esperamos que os editores poss
Também, fazemos chamadas com o conselho editorial anualmente.
Ademais, alternamos as responsabilidades do editor-chefe (decisões de escopo de primeir
Você terá a oportunidade de participar desse rodízio depois de fazer parte do conselho
Alguns de nós também assumem projetos maiores para aprimorar o processo de revisão por

Esperamos que você faça parte do conselho!
Este é um momento empolgante para a revisão por pares na rOpenSci.

Por favor, reflita sobre a nossa oferta e faça perguntas se algo não estiver claro. Nos
se a equipe rOpenSci.

Atenciosamente,
[EDITOR], em nome do Conselho Editorial da rOpenSci
```

9.3 Integrando um(a) novo(a) editor(a) ao time

- Adicionar ao arquivo "team.json" do site com `"editor": true` e também `"stats": true` para editores de estatísticas. Isso adicionará automaticamente novos editores à equipe editorial mostrada na página do site da rOpenSci.

- Informe o(a) gerente da comunidade rOpenSci para que você possa preparar uma postagem introdutória no blog.

- Se eles ainda não fizeram este passo como editores convidados, solicite que ativem a autenticação de dois fatores (2FA) para o GitHub.

- Convide-os para a organização rOpenSci no GitHub como membro da equipe `editors` e da equipe de editores de estatísticas `stats-editors`. Isso dará a eles as permissões apropriadas e garantirá que apareçam no painel de controle dos editores.

- Convide-os para o banco de dados AirTable de análise de software (vinculado na descrição do canal, somente para editores, no Slack). Certifique-se de que o convite seja "Somente leitura".

- Convide-os para o canal privado “editors-only” no espaço de trabalho Slack da rOpenSci (e para o espaço de trabalho Slack em geral, se eles ainda não estiverem lá).
- Quando eles estiverem no canal “editors-only”, publique uma mensagem de boas-vindas informando todos os editores.

9.4 Desvincular um(a) editor(a)

- Agradeça o(a) editor(a) por seu trabalho!
- Informe o(a) gerente da comunidade rOpenSci ou outros membros da equipe, para garantir que nossa newsletter anuncie sua saída, e para agradecer a essa pessoa.
- Remova-os do canal “editors-only” e da equipe de editores do Slack.
- Remova este(a) editor(a) de https://github.com/orgs/ropensci/teams/editors/members e da subequipe.
- Atualizar o “team.json” no site substituindo “roles” por “past_roles” e adicionando um novo campo, `alumnus: "true"`.
- Remova o acesso do(a) editor(a) ao espaço de trabalho do Airtable.
  - Na exibição “interface”, abra o menu principal no canto superior esquerdo e digite “View Data” (Exibir dados)
  - Clique no botão “Share” (Compartilhar) no canto superior direito e, em seguida, em “People with access” (Pessoas com acesso)
  - Clique na caixa de seleção à esquerda do(a) editor(a) a ser removido(a) e em “Remove 1 collaborator” (Remover 1 colaborador/a).
- As listas de editores (anteriores) em ambos os capítulos do dev_guide que apresenta a revisão de software e do software-review README são preenchidos automaticamente a partir dos dados do AirTable. As atualizações são executadas diariamente, portanto, verifique um dia após as atualizações do AirTable para garantir que ambos tenham sido atualizados.

9.5 Colocando o sistema em pausa

Se você quiser colocar o sistema em uma pausa, por exemplo, durante as férias, antes de sair:

- Adicione uma mensagem de férias ao campo `about` dos templates de issues. Exemplo de PR.

- Adicione uma mensagem de férias à resposta de boas-vindas padrão do bot. Exemplo de PR.

Ao retomar as atividades:

- Remover a mensagem de férias dos templates de issues. Exemplo de PR.
- Remover a mensagem de férias da resposta de boas-vindas padrão do bot. Exemplo de commit.

9.6 Gerenciando o lançamento de um guia de desenvolvimento

Se você estiver encarregado de gerenciar uma versão do livro que está lendo, use o guia de lançamento de livros como um modelo de *issue* a ser publicado no rastreador de problemas do guia de desenvolvimento. Não hesite em fazer perguntas a outros editores.

9.6.1 Governança do guia de desenvolvimento

Para alterações muito pequenas no guia de desenvolvimento, não é necessária fazer revisão de *pull request* (PR). Para alterações maiores, solicite a revisão de pelo menos alguns editores (se nenhum deles participou da discussão relacionada à alteração, solicite uma revisão de todos eles no GitHub e, na ausência de qualquer reação, faça o *merge* após 1 semana).

Duas semanas antes do lançamento de um guia de desenvolvimento, uma vez que o PR do dev para o *master* **e a postagem do blog de lançamento** estiverem prontos para revisão, todos os editores devem receber um *ping* no GitHub ("solicitação de revisão" no PR de dev para *master*) e do Slack, mas o lançamento não precisa que todos eles aprovem explicitamente o lançamento.

9.6.2 Postagem no blog sobre um lançamento

A postagem no blog sobre um lançamento será revisada por editores e um dos `@ropensci/blog-editors`.

9.6.2.1 Conteúdo

Consulte a orientação geral sobre blogs da rOpenSci e as orientações mais específicas abaixo.

Primeiro exemplo de uma postagem desse tipo; segundo exemplo.

A postagem no blog deve mencionar todos os itens importantes do changelog, organizados em (sub)seções: e.g., uma seção sobre uma grande mudança A, outra sobre uma grande mudança B e uma sobre as mudanças menores agrupadas. Mencione as alterações mais importantes primeiro.

Para cada alteração feita por uma pessoa colaboradora externa, agradeça-a explicitamente usando as informações do *changelog*. Por exemplo, `[Matt Fidler](https://github.com/mattfidler/) modificou nossa seção sobre mensagens de console [ropensci/dev_guide#178](https://github.com/ropensci/dev_guide/pull/17`

No final da postagem, mencione as próximas alterações vinculando-as a problemas abertos no rastreador de problemas e convide as pessoas leitoras a contribuir com o guia de desenvolvimento abrindo *issues* e participando de discussões abertas. Modelo de conclusão:

```
Nesta postagem, resumimos as alterações incorporadas em nosso livro ["rOpenSci Packages: Developm
Somos gratos por todas as contribuições que tornaram possível esse lançamento.
Já estamos trabalhando em atualizações para a nossa próxima versão, como _issue1_, _issue2_.
Confira o [issue tracker] (https://github.com/ropensci/dev_guide/issues/) se você quiser contribu
```

9.6.2.2 Autoria

O editor que está escrevendo a postagem é o primeiro autor, os outros editores estão listados em ordem alfabética.

Parte III

Mantendo pacotes

Capítulo 10

Folha de dicas de manutenção do pacote rOpenSci

Um lembrete da infraestrutura e dos canais de contato para mantenedores de pacotes rOpenSci.

10.1 Você precisa de ajuda?

Se você precisar de ajuda pontual (por exemplo, uma revisão de PR; ou alguma solução de problemas de CI), ou ajuda para procurar co-mantenedores ou um novo mantenedor, ou se precisar que retiremos seu pacote, envie-nos um ping no GitHub via `@ropensci/admin` ou envie um e-mail para você `info@ropensci.org`. Você também pode usar nosso canal de manutenção de pacotes no Slack.

Nunca hesite em pedir ajuda.

10.2 Acesso ao repositório do GitHub

Você deve ter acesso administrativo ao repositório do GitHub do seu pacote. Se esse não for mais o caso (por exemplo, o processo automatizado falhou ou você perdeu o acesso depois de ter que desativar temporariamente a autenticação de dois fatores), entre em contato conosco via `info@ropensci.org`.

10.3 Outros tópicos do GitHub

Se tiver alguma pergunta ou solicitação sobre o GitHub (por exemplo, adicionar um colaborador à organização do GitHub), você pode usar um canal público do espaço de trabalho slack do rOpenSci ou enviar um ping para `@ropensci/admin` no GitHub.

10.4 Documentação do pkgdown

Veja Documentos do rOpenSci.

10.5 Acesso ao espaço de trabalho do rOpenSci no Slack

Os mantenedores e desenvolvedores de pacotes devem ter acesso a Slack do rOpenSci. Se você não recebeu o convite ou não o aceitou a tempo, ou se você quiser que um novo colaborador regular receba um convite, envie um e-mail para `info@ropensci.org`, indicando para qual endereço de e-mail você deseja receber o convite.

Você pode achar que o canal #package-maintenance é relevante para perguntas e respostas, bem como para uma comiseração amigável quando necessário.

10.6 Publicações no blog sobre pacotes

Consulte nosso guia do blog.

10.7 Promoção de problemas de pacotes

Rotular as edições com “procura-se ajuda” para obtê-las para que sejam transmitidos à comunidade.

10.8 Promoção de casos de uso de pacotes

Você pode relatar casos de uso do seu pacote ou incentivar os usuários a relatá-los por meio do nosso fórum para obtê-los publicados em nosso site e em nosso boletim informativo.

Capítulo 11

Guia de colaboração

Ter pessoas colaborando vai trazer melhorias para o seu pacote e, se você integrar algumas delas como autoras do pacotes com permissões de escrita no repositório, seu pacote será desenvolvido de forma mais sustentável. Também pode ser muito agradável trabalhar em equipe!

Este capítulo contém nossa orientação para colaboração, em uma seção sobre como tornar o seu repositório amigável para contribuições e colaborações por infraestrutura (código de conduta, diretrizes de contribuição, rótulos de problemas); e uma seção sobre como colaborar com novos colaboradores, em particular no contexto da organização da rOpenSci no GitHub.

Além dessas dicas técnicas, é importante lembrar-se de ser gentil e levar em conta a perspectiva das outras pessoas, especialmente quando as prioridades delas forem diferentes das suas.

11.1 Torne a contribuição e a colaboração do seu repositório amigáveis

Tornar seu repositório fácil de colaborar é parte do que pode ajudar a garantir que ele perdure para além do seu trabalho.

11.1.1 Código de conduta

Depois que o seu pacote for listado na rOpenSci registry (a lista oficial de pacotes da rOpenSci), o Código de Conduta da rOpenSci será aplicado ao seu projeto. Você deve adicionar este texto ao README:

```
Please note that this package is released with a [Contributor
Code of Conduct](https://ropensci.org/code-of-conduct/).
By
contributing to this project, you agree to abide by its terms.
```

E

- Se o seu pacote foi transferido para a organização da rOpenSci no GitHub, exclua o código de conduta atual do repositório, se houver um, pois o código de conduta padrão da organização GitHub será exibido.
- Se o seu pacote não for transferido para a organização da rOpenSci no GitHub, substitua o conteúdo do código de conduta atual do repositório pelo conteúdo do [código de conduta padrão da organização rOpenSci no GitHub](https://github.com/ropensci/.github/blob/main/CODE_OF_CONDUCT.md).

11.1.2 Guia de contribuição

Você pode usar a *issue*, a solicitação de *pull request* e as diretrizes de contribuição. Ter um arquivo sobre contribuição como `.github/CONTRIBUTING.md` ou `docs/CONTRIBUTING.md` é obrigatório. Uma maneira fácil de inserir um modelo para um guia de contribuição é usar a função `use_tidy_contributing()` do pacote `usethis` que insere este modelo como `.github/CONTRIBUTING.md`. Um exemplo mais completo é este modelo de Peter Desmet ou as abrangentes páginas da wiki do pacote `mlr3` no GitHub. Em geral, esses e outros modelos precisarão ser modificados para serem usados com um pacote da rOpenSci, principalmente por meio de referências e links para o nosso Código de Conduta conforme descrito em um outro lugar neste livro. Modificar um guia de contribuição genérico para adicionar um toque pessoal também tende a fazer com que ele pareça menos genérico e mais sincero. As preferências pessoais em um guia de contribuição incluem:

- Preferências de estilo? No entanto, você pode preferir tornar o estilo uma configuração (de Air, styler, lintr,) ou para corrigir você mesmo o estilo de código especialmente se você não usar um estilo de código popular como o estilo de código do tidyverse.
- Infraestrutura como a do `roxygen2`?
- Preferências de fluxo de trabalho? *Issue* antes de um PR?
- Uma declaração de escopo, como no pacote skimr?
- Criação de contas de sandbox? *Mocking* em testes? Links para documentos externos?

A rOpenSci também incentiva os guias de contribuição a incluírem uma declaração de ciclo de vida que esclareça as visões e expectativas para o desenvolvimento futuro do seu pacote, como neste exemplo. É necessário que os pacotes estatísticos tenham uma declaração de ciclo de vida, conforme especificado em *Padrões estatísticos gerais* G1.2. Esses links fornecem um modelo para uma declaração de ciclo de vida simples. Os arquivos `CONTRIBUTING.md` também podem descrever como você reconhece as contribuições (consulte esta seção).

Recomendamos que você envie comentários que não sejam um relatório de bug ou uma solicitação de *feature* para as ferramentas de discussão que sua plataforma Git oferece, como Discussões do GitHub para o GitHub.

Quando uma pull request estiver mais perto de ser mesclada, você poderá estilizar o código com Air ou styler.

11.1.3 Gerenciamento de *issues*

Ao usar os recursos do GitHub em torno dos *issues*, você pode ajudar os possíveis colaboradores a encontrá-los e tornar público o seu roteiro.

11.1.3.1 Modelos de *issues*

Você pode usar um ou vários modelo(s) de *issue(s)* para ajudar as pessoas usuárias a preencherem os relatórios de bugs ou as solicitações de *feature*. Quando há vários modelos de *issues*, os usuários que clicam em abrir uma nova *issue* veem um menu que orienta as suas escolhas.

Você pode até configurar uma das opções para apontar para algum lugar fora do seu repositório (por exemplo, um fórum de discussão).

Consulte a Documentação do GitHub.

11.1.3.2 Rotulagem de *issues*

Você pode usar rótulos como “help wanted” (procura-se ajuda) e “good first issue” (bom primeiro problema) para ajudar colaboradores em potencial, inclusive novatos, a encontrar o seu repositório. Consulte o artigo do GitHub. Você também pode usar o rótulo “Beginner” (Iniciante). Você pode ver exemplos de problemas para iniciantes em todos os repositórios da rOpenSci.

11.1.3.3 Fixando *issues*

Você pode fixar até 3 *issues* por repositório que aparecerão na parte superior do seu rastreador de *issues* como cartões de *issues* bonitos. Isso pode ajudar a divulgar quais são suas prioridades.

11.1.3.4 Marcos

Você pode criar marcos e atribuir problemas a eles, o que ajuda outras pessoas a verem o que você planeja para a próxima versão do seu pacote, por exemplo.

11.1.4 Comunicação com as pessoas usuárias

Você pode indicar para as pessoas usuárias o fórum da rOpenSci se você monitorar ou ativar as Discussões no GitHub para o repositório do seu pacote. Cada discussão do GitHub pode ser convertida em um *issue*, se necessário (e vice-versa, os *issues* podem ser convertidos em discussões).

11.2 Trabalhando com colaboradores

Em primeiro lugar: mantenha contato com o seu repositório do GitHub!

- Não se esqueça de **observar o seu repositório do GitHub** para receber notificações sobre *issues* ou *pull requests* (como alternativa, se você trabalha de forma concentrada em certas épocas, talvez adicione essas informações ao guia de contribuição).
- não se esqueça de enviar as atualizações que você tem localmente.
- desative os testes com falha se você não puder corrigí-los, pois eles criam ruído nos PRs que podem confundir colaboradores iniciantes.

11.2.1 Integração de pessoas colaboradoras

Não existe uma regra geral da rOpenSci sobre como você deve integrar pessoas colaboradoras. Você deve aumentar os direitos delas sobre o repositório à medida que ganhar confiança e, sem dúvida, deve reconhecer as contribuições que fizerem (consulte esta seção).

Você pode pedir a um novo colaborador que crie PRs (consulte a seção a seguir para avaliar um PR localmente, ou seja, além das verificações de CI) para dev/main e avalie-os antes de mesclá-los e, depois de algum tempo, deixe-os enviar para o main, embora você possa querer manter um sistema de revisões de PRs… mesmo para si mesmo quando tiver colegas de equipe!

Um modelo possível para a integração de pessoas colaboradoras é fornecido por Jim Hester em seu `lintr` repo.

Se o seu problema for o recrutamento de pessoas colaboradoras, você pode publicar uma chamada aberta como a de Jim Hester no Twitter, ou no GitHub e, como

autor(a) de um pacote da rOpenSci, você pode pedir ajuda no slack da rOpenSci e pedir à equipe da rOpenSci por ideias para recrutar novas pessoas colaboradoras.

11.2.2 Trabalhando com colaboradores (incluindo você)

Branchs (ou ramificações) são baratas. Use-as extensivamente ao desenvolver recursos, testar novas ideias e corrigir problemas.

Uma das *branches* é a *branch* `main` (que é a ramificação padrão/principal), na qual, se você seguir desenvolvimento baseado no tronco você "faz *merge* de atualizações pequenas e frequentes". Veja também as documentações do Fluxo do GitHub e do fluxo do GitLab. Talvez você também queira incrementar frequentemente os números da versão de seu pacote (em `DESCRIPTION`). Um aspecto específico do trabalho com colaboradores é a revisão de *Pull requests*, com algumas orientações úteis em:

- The Art of Giving and Receiving Code Reviews (Gracefully) (A arte de dar e receber revisões de código (graciosamente)), de Alex Hill;
- Documentação do GitHub sobre revisões de PR.

Você pode querer mexer nas configurações do seu repositório do GitHub para, por exemplo, exigir revisões de *pull request* antes de fazer o *merge*. Consulte também os documentos do GitHub sobre "proprietários de código".

Para fazer e revisar *pull requests*, recomendamos que você que você explore essas funções.

Para que você configure o "git remote", consulte Happy Git and GitHub for the useR. Veja também a seção Useful Git patterns for real life no mesmo livro.

11.2.3 Seja generoso(a) com as atribuições

Se alguém contribuir para o seu repositório, considere adicioná-la em `DESCRIPTION`, como contribuinte ("ctb") para pequenas contribuições, e autor ("aut") para contribuições maiores. Tradicionalmente, ao citar um pacote em uma publicação científica, apenas as pessoas autoras "aut" são listados, e não as contribuidores "ctb"; e em `pkgdown` e, nos sites, somente os nomes "aut" são listados na página inicial, e todas as pessoas autoras são listados na página de autores.

No mínimo, considere adicionar o nome das pessoas colaboradoras próximo à linha de correção de *feature*/*bugs* em NEWS.md.

Você também pode usar o pacote R allcontributors para agradecer a todos os colaboradores no arquivo README.

Recomendamos que você seja generoso(a) com esses agradecimentos, porque é uma coisa boa de se fazer e porque isso aumentará a probabilidade de as pessoas contribuírem novamente com o seu pacote ou com outros repositórios da organização.

Como um lembrete de nossas diretrizes de empacotamento, se o seu pacote foi revisado e você acha que as pessoas revisoras fizeram uma contribuição substancial para o desenvolvimento dele, você pode listá-las na seção `Authors@R` com um tipo de contribuinte "Revisor(a)" (`"rev"`), da seguinte forma:

```
    person("Bea", "Hernández", role = "rev",
    comment = "Bea reviewed the package (v. X.X.XX) for rOpenSci, see <https://github.c
review/issues/116>"),
```

Inclua as pessoas revisoras somente após solicitar o consentimento delas. Leia mais nesta postagem do blog "Agradecendo seus revisores: Gratidão por meio de metadados semânticos". Observe que 'rev' gerará uma CRAN NOTE, a menos que o pacote seja criado usando o R v3.5. Certifique-se de que você use `roxygen2` na versão mais recente disponível no CRAN.

Por favor, não liste os editores como colaboradores. Sua participação e contribuição para a rOpenSci são agradecimentos suficientes!

11.2.4 Dando as boas-vindas aos colaboradores da rOpenSci

Se você conceder a alguém permissões de escrita no repositório,

- entre em contato com um membro da equipe para que esse novo colaborador possa receber **um convite para a organização da rOpenSci no GitHub** (em vez de ser um colaborador externo)
- entre em contato com a equipe da rOpenSci ou um outro membro da equipe para que esse novo colaborador possa receber **um convite para o *workspace* do Slack da rOpenSci**.

11.3 Outros recursos

- Chamada da comunidade rOpenSci: Configure seu pacote para promover uma comunidade.
- Para reutilizar respostas gentis e usuais, considere a funcionalidade de respostas salvas do GitHub.

Capítulo 12

Mudando os(as) mantenedores(as) de um pacote

Este capítulo apresenta um guia sobre como assumir a manutenção de um pacote.

12.1 Você quer desistir da manutenção do seu pacote?

Temos uma seção em nosso boletim informativo para chamadas de novos(as) colaboradores(as) que é publicado a cada duas semanas. A seção se chama *Chamada para colaboradores*. Nessa seção, destacamos os pacotes que estão à procura de novos(as) mantenedores(as). Se você deseja desistir da função de mantenedor(a) do seu pacote, entre em contato conosco e poderemos destacar o seu pacote em nosso boletim informativo.

12.2 Você quer assumir a manutenção de um pacote?

Temos uma seção em nosso boletim informativo para chamadas de novos(as) colaboradores(as) que é publicado a cada duas semanas. A seção se chama *Chamada para colaboradores*. Nessa seção, destacamos os pacotes que estão à procura de novos(as) mantenedores(as). Se você ainda não está inscrito no boletim informativo, é uma boa ideia assinar para que você seja notificado quando houver um pacote procurando por um novo(a) mantenedor(a).

12.3 Assumir a manutenção de um pacote

- Adicione você como o(a) novo(a) mantenedor(a) no arquivo DESCRIPTION, com `role = c("aut", "cre")`, e, altere o(a) antigo(a) mantenedor(a) para o papel `aut` apenas;
- Certifique-se de alterar o(a) mantenedor(a) para o seu nome em qualquer outro lugar do pacote, mantendo o(a) antigo(a) mantenedor(a) como autor (por exemplo, em manuais do pacote, CONTRIBUTING.md, CITATION etc.);
- Os Guia de Colaboração tem orientações sobre como adicionar novos(as) mantenedores(as) e colaboradores(as);
- Pacotes que foram arquivados ou órfãos no CRAN não precisam da permissão do(a) mantenedor(a) anterior para serem retomados no CRAN. Nesses casos, entre em contato conosco para que possamos oferecer a ajuda necessária;
- Se o pacote não tiver sido arquivado pelo CRAN e houver uma mudança de mantenedor(a), peça ao(à) mantenedor(a) antigo(a) que envie um e-mail ao CRAN e informe por escrito quem é o(a) novo(a) mantenedor(a). Certifique-se de mencionar esse e-mail sobre a mudança de mantenedor(a) quando você enviar a primeira nova versão ao CRAN. Se o(a) antigo(a) mantenedor(a) estiver inacessível ou não enviar esse e-mail, entre em contato com a equipe do rOpenSci;
- Se o(a) mantenedor(a) anterior estiver acessível, agendar uma reunião ajudará você a conhecer melhor a situação atual do pacote;

12.3.1 Perguntas frequentes para novos(as) mantenedores(as)

- Existem alguns problemas no pacote que ainda não foram resolvidos, e que eu não sei como corrigir. A quem posso pedir ajuda?

  Depende; aqui está o que você deve fazer em diferentes cenários:

  - Se for possível entrar em contato com o(a) antigo(a) mantenedor(a): entre em contato com ele(a) e peça ajuda;
  - Slack do rOpenSci: bom para obter ajuda em problemas específicos ou gerais, consulte o canal #package-maintenance;
  - Equipe do rOpenSci sinta-se livre para entrar em contato com um de nós, seja por e-mail ou nos marcando em um problema no GitHub, ficaremos felizes em ajudar.

- O quanto você pode/deve alterar no pacote?

  Para obter ajuda geral sobre como alterar o código em um pacote, consulte a seção Evolução do pacote de pacotes.

  Quando você pensa nisso, há muitas considerações a se fazer.

  Quanto tempo você tem para dedicar ao pacote? Se você tiver um tempo muito limitado, seria melhor se concentrar nas tarefas mais importantes,

sejam elas quais forem para o pacote em questão. Se você tiver bastante tempo, suas metas poderão ter um escopo maior.

Qual é o grau de maturidade do pacote? Se o pacote for maduro, muitas pessoas provavelmente têm códigos que dependem da API do pacote (ou seja, as funções exportadas e seus parâmetros). Além disso, se houver outros pacotes que dependam do seu pacote no CRAN, você precisa verificar se as suas alterações vão quebrar ou não esses outros pacotes. Quanto mais maduro for o pacote, mais cuidadoso(a) você precisará ser ao fazer alterações, especialmente com os nomes das funções exportadas, seus parâmetros e a estrutura exata do que as funções exportadas retornam. É mais fácil fazer alterações que afetem apenas os aspectos internos do pacote.

12.4 Tarefas da equipe do rOpenSci

Como organização, a rOpenSci está interessada em garantir que os pacotes de nossa suíte estejam disponíveis enquanto forem úteis para a comunidade. Como os(as) mantenedores(as) precisam seguir em frente, na maioria dos casos tentaremos conseguir um(a) novo(a) mantenedor(a) para cada pacote. Para isso, as tarefas a seguir são de responsabilidade da equipe da rOpenSci.

- Se um repositório não tiver nenhuma atividade (commits, issues, pull requests) há muito tempo, ele pode ser simplesmente um pacote maduro com pouca necessidade de alterações/etc., portanto, leve isso em consideração.
- O(a) mantenedor(a) atual não responde a problemas/solicitações pull há muitos meses, por meio de e-mails, problemas no GitHub ou mensagens no Slack:
  - Entre em contato e veja qual é a situação. Ele(a) pode dizer que gostaria de deixar o cargo de mantenedor(a) e, nesse caso, procure um(a) novo(a) mantenedor(a)
- O(a) mantenedor(a) atual está completamente ausente/não está respondendo
  - Se isso acontecer, tentaremos entrar em contato com o(a) mantenedor(a) por até um mês. No entanto, se a atualização do pacote for urgente, poderemos usar nosso acesso de administrador para fazer alterações em seu nome.
- Faça uma chamada na seção “Call for Contributors” do boletim informativo da rOpenSci para um(a) novo(a) mantenedor(a) - abra um problema no repositório do boletim informativo.

Capítulo 13

Publicação de um pacote

Seu pacote deve ter versões diferentes ao longo do tempo: *snapshots* de um estado do pacote que você pode publicar no CRAN, por exemplo. Essas versões devem ser devidamente *numeradas*, *publicadas* e *descritas em um arquivo NEWS*. Mais detalhes abaixo.

Observe que você pode simplificar o processo de atualização do NEWS e do controle de versão do seu pacote usando [o pacote `fledge`] (https://github.com/cynkra/fledge).

13.1 Controle de versão

- Recomendamos enfaticamente que os pacotes rOpenSci usem controle de versão semântico. Uma explicação detalhada está disponível no capítulo de ciclo de vida do livro *R Packages*.

13.2 Publicação

- Usando `usethis::use_release_issue()` e `devtools::release()` ajudará você a se lembrar de mais verificações.
- Marque cada versão no Git após cada envio ao CRAN. Mais informações
- O CRAN não gosta de atualizações muito frequentes. Dito isso, se você notar um grande problema uma semana após o lançamento do CRAN, explique-o no cran-comments.md e tente lançar uma versão mais recente.

13.3 Arquivo de notícias

Um arquivo NEWS descrevendo as alterações associadas a cada versão facilita a visualização do que está sendo alterado no pacote e como isso pode afetar o fluxo de trabalho. Você deve adicionar um para o seu pacote e torná-lo de fácil leitura.

- É obrigatório usar um arquivo `NEWS` ou `NEWS.md` na raiz do seu pacote. Recomendamos que você use o arquivo `NEWS.md` para tornar o arquivo mais navegável.
- Você pode usar nosso exemplo arquivo NEWS como modelo. Você pode encontrar um bom arquivo NEWS em uso real [no repositório do pacote `targets`] (https://github.com/ropensci/targets/blob/main/NEWS.md)), por exemplo.
- Se você usar `NEWS` adicione-o a `.Rbuildignore` mas não se você usar `NEWS.md`
- Atualize o arquivo de notícias antes de cada envio para o CRAN, com uma seção com o nome do pacote, a versão e a data de lançamento, (como visto em nosso exemplo arquivo NEWS):

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

- Sob esse cabeçalho, coloque as seções conforme necessário, incluindo: `NEW FEATURES`, `MINOR IMPROVEMENTS`, `BUG FIXES`, `DEPRECATED AND DEFUNCT`, `DOCUMENTATION FIXES` e qualquer título especial que agrupe um grande número de alterações. Em cada cabeçalho, liste os itens conforme necessário (como visto em nosso exemplo Arquivo NEWS). Para cada item, forneça uma descrição do novo recurso, melhoria, correção de bug ou função/característica obsoleta. Link para qualquer problema relacionado no GitHub, como `(#12)`. O problema `(#12)` será resolvido no GitHub em Releases para um link para esse problema no repositório.
- Depois que você tiver adicionado um `git tag` e enviado para o GitHub, adicione os itens de notícias para essa versão marcada às notas de versão de uma versão no seu repositório do GitHub com um título como `pkgname v0.1.0`. Você pode ver Documentos do GitHub sobre a criação de uma versão.
- Novos lançamentos do CRAN serão escritos sobre em nosso boletim informativo mas veja próximo capítulo sobre marketing sobre como informar mais usuários em potencial sobre o lançamento.
- Para obter mais orientações sobre o arquivo NEWS, sugerimos que você leia o documento guia de estilo do tidyverse NEWS.

Capítulo 14

Marketing do seu pacote

Consulte também a postagem do blog *Marketing Ideas For Your Package*.

Ajudaremos você a promover o seu pacote, mas aqui estão mais algumas coisas que você deve ter em mente.

- Se você souber de um caso de uso de seu pacote, incentive o autor a publicar o link em nosso site, para um post no Mastodon da rOpenSci e possível inclusão no boletim quinzenal da rOpenSci. Também recomendamos que você adicione um link para o caso de uso em uma seção "use cases in the wild" do seu README.
- Quando você liberar uma nova versão do seu pacote ou você lançá-lo pela primeira vez no CRAN,
  - Faça um *pull request* para a R Weekly com uma linha sobre essa nova versão na seção "New Releases" (ou "New Packages" para a primeira versão do GitHub/CRAN).
  - Publique sobre seu pacote nas mídias sociais (Mastodon, Bluesky, LinkedIn…) usando a hashtag "#rstats" e marque a rOpenSci se ela estiver presente nessa plataforma! Isso pode ajudar no engajamento de pessoas colaboradoras e usuárias. Exemplo.
  - Considere a possibilidade de enviar um breve post sobre o lançamento para as Notas técnicas da rOpenSci. Entre em contato com o/a gerente da comunidade do rOpenSci (por exemplo, via Slack ou info@ropensci.org). Consulte as diretrizes sobre como contribuir com um *blog post*).
  - Envie seu pacote para listas de pacotes, como a Visão de tarefas do CRAN.
- Se você optar por divulgar o seu pacote dando uma palestra sobre ele em um encontro ou conferência (excelente ideia!) leia este artigo de Jenny Bryan e Mara Averick.

Capítulo 15

Preparação do GitHub

Atualmente, os pacotes da rOpenSci são, em sua grande maioria, desenvolvidos no GitHub. Aqui, estão algumas dicas para aproveitar a plataforma em uma seção sobre tornar seu repositório mais detectável e uma seção sobre comercializar sua própria conta do GitHub após passar pela revisão por pares.

15.1 Torne seu repositório mais detectável

15.1.1 Áreas de repositório do GitHub

As áreas de repositório do GitHub ajudam a navegar e pesquisar repositórios do GitHub, são usadas pelo R-universe em páginas de pacotes e para resultados de pesquisa.

Recomendamos:

- Adicionar “r”, “r-package” e “rstats” como tópicos ao repositório de seu pacote.
- Adicionar quaisquer outros tópicos relevantes ao repositório do seu pacote.

Poderemos fazer sugestões a você depois que seu pacote for integrado.

15.1.2 GitHub linguist

O GitHub linguist atribuirá uma linguagem ao seu repositório com base nos arquivos que ele contém. Alguns pacotes que contêm muito código em C++ podem ser classificados como pacotes C++ em vez de pacotes R, o que é bom e mostra a necessidade de adicionar os tópicos “r”, “r-package” e “rstats”.

Recomendamos que você substitua o GitHub linguist adicionando ou modificando um .gitattributes ao seu repositório em dois casos:

- Se você armazenar arquivos html em locais diferentes do padrão (não em docs/, por exemplo, em vignettes/), use as substituições de documentação. Adicione `*.html linguist-documentation=true` ao arquivo .gitattributes (Exemplo em uso real)
- Se o seu repositório contiver código que você não criou, por exemplo, código JavaScript, adicione `inst/js/* linguist-vendored` a .gitattributes (Exemplo em uso real)

Dessa forma, a classificação da linguagem e as estatísticas do seu repositório refletirão melhor o código-fonte que ele contém, além de torná-lo mais detectável. Notavelmente, se o GitHub linguist não reconhecer corretamente que seu repositório contém principalmente código R, seu pacote não aparecerá nos resultados de pesquisa usando o filtro `language:R`. Da mesma forma, seu repositório não poderá ser listado entre os repositórios R em alta.

Mais informações sobre as substituições do GitHub linguist.

15.2 Comercialize sua própria conta

- Como autor de um pacote integrado, você agora é membro da organização "ropensci" da rOpenSci no GitHub. Por padrão, as participações da organização são privadas; consulte como torná-la pública na documentação do GitHub.
- Mesmo que o repositório do seu pacote tenha sido transferido para a rOpenSci, você pode fixá-lo em sua conta pessoal.
- Em geral, recomendamos que você adicione pelo menos um avatar (que não precisa ser seu rosto!) e seu nome no seu perfil do GitHub.

Capítulo 16

Evolução do pacote - alteração de itens em seu pacote

Este capítulo apresenta nossa orientação para alterar coisas em seu pacote: alterar nomes de parâmetros, alterar nomes de funções, descontinuar funções e até mesmo retirar e arquivar pacotes.

Este capítulo foi inicialmente contribuído como uma nota técnica no site da rOpenSci por Scott Chamberlain; você pode ler a versão original em inglês.

16.1 Filosofia das alterações

Todos são livres para ter sua própria opinião sobre a liberdade com que parâmetros/funções/etc. são alterados em uma biblioteca - as regras sobre alterações de pacotes não são impostas pelo CRAN ou de outra forma. Em geral, à medida que uma biblioteca se torna mais madura, as alterações nos métodos voltados para o usuário (ou seja, funções exportadas em um pacote R) devem se tornar muito raras. As bibliotecas que são dependências de muitas outras bibliotecas provavelmente serão mais cuidadosas com as alterações, e devem ser.

16.2 O pacote lifecycle

Este capítulo apresenta soluções que não requerem o pacote lifecycle, mas que você ainda pode considerar úteis. Recomendamos que você leia a documentação do lifecycle.

16.3 Parâmetros: alteração dos nomes dos parâmetros

Às vezes, os nomes dos parâmetros precisam ser alterados para maior clareza ou por algum outro motivo.

Uma abordagem possível é verificar se os argumentos descontinuados não estão faltando e parar de fornecer uma mensagem significativa.

```
foo_bar <- function(x, y) {
    if (!missing(x)) {
        stop("use 'y' instead of 'x'")
    }
    y^2
}

foo_bar(x = 5)
#> Error in foo_bar(x = 5) : use 'y' instead of 'x'
```

Se quiser que a função seja mais útil, você pode fazê-la emitir um aviso e tomar automaticamente a ação necessária:

```
foo_bar <- function(x, y) {
    if (!missing(x)) {
        warning("use 'y' instead of 'x'")
        y <- x
    }
    y^2
}

foo_bar(x = 5)
#> 25
```

Esteja ciente do parâmetro `...`. Se sua função tiver `...` e você já tiver removido um parâmetro (vamos chamá-lo de `z`), um usuário pode ter um código mais antigo que usa `z`. Quando você passa o parâmetro `z` ele não é um parâmetro na definição da função e provavelmente será ignorado silenciosamente – não é o que você deseja. Em vez disso, deixe o argumento presente, lançando um erro se ele for usado.

16.4 Funções: alteração de nomes de funções

Se você precisar alterar o nome de uma função, faça-o gradualmente, como em qualquer outra alteração em seu pacote.

Digamos que você tenha uma função `foo`.

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

No entanto, você deseja alterar o nome da função para `bar`.

Em vez de simplesmente alterar o nome da função e `foo` deixar de existir imediatamente, na primeira versão do pacote em que o `bar` aparecer, crie um alias como:

```
#' foo - add 1 to an input
#' @export
foo <- function(x) x + 1

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

Com a solução acima, o usuário pode usar `foo()` ou `bar()` – ambos farão a mesma coisa, pois são a mesma função.

Também é útil ter uma mensagem, mas você só vai querer lançar essa mensagem quando eles usarem a função antiga, por exemplo,

```
#' foo - add 1 to an input
#' @export
foo <- function(x) {
    warning("please use bar() instead of foo()", call. = FALSE)
    bar(x)
}

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

Depois que os usuários tiverem usado a versão do pacote por algum tempo (com ambos os `foo` e `bar`), na próxima versão você poderá remover o nome da função antiga (`foo`), e você terá apenas `bar`.

```
#' bar - add 1 to an input
#' @export
bar <- function(x) x + 1
```

16.5 Dados: descontinuar

Se você precisar descontinuar (*deprecate*) um conjunto de dados fornecido pelo seu pacote, leia a solução proposta por Matthijs Berends no Stack Overflow, com link

para [um método de três etapas na orientação do Bioconductor] (https://contributions.bioconductor.org/deprecation.html#deprecate-dataset). O ponto principal é o uso de `delayedAssign()` para criar uma *promise* que servirá tanto para emitir um aviso (*warning*) quanto para fornecer os dados. ## Funções: descontinuadas e removidas {#functions-deprecate-defunct}

Para remover uma função de um pacote (digamos que o nome do seu pacote seja `helloworld`), você pode usar o seguinte protocolo:

- Marque a função como descontinuada na versão do pacote `x` (por exemplo, `v0.2.0`)

Na própria função, use `.Deprecated()` para apontar para a função de substituição:

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

Há opções em `.Deprecated` para especificar um novo nome de função, bem como um novo nome de pacote, o que faz sentido quando você move funções para pacotes diferentes.

A mensagem que é dada por `.Deprecated` é um aviso, portanto, pode ser suprimida por usuários com `suppressWarnings()` se você desejar.

Crie uma página de manual para funções descontinuadas, como:

```
#' Deprecated functions in helloworld
#'
#' These functions still work but will be removed (defunct) in the next version.
#'
#' \itemize{
#'  \item \code{\link{foo}}: This function is deprecated, and will
#'  be removed in the next version of this package.
#' }
#'
#' @name helloworld-deprecated
NULL
```

Isso cria uma página de manual que os usuários podem acessar como `?` `helloworld-deprecated` e que você verá no índice da documentação. Adicione quaisquer funções a essa página conforme necessário e remova-as quando uma função se tornar "defunct" (veja abaixo).

- Na próxima versão (`v0.3.0`), você pode tornar a função "defunct" (ou seja, completamente removida do pacote, exceto por uma página de manual com uma nota sobre ela).

Na própria função, use `.Defunct()` como:

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

Observe que a mensagem em `.Defunct` é um erro para que a função pare, enquanto `.Deprecated` usa um aviso que permite que a função continue.

Além disso, é bom adicionar `...` a todas as funções removidas para que, se os usuários passarem algum parâmetro, recebam a mesma mensagem de "defunct" em vez de um `unused argument` assim, por exemplo:

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

Sem `...` o resultado é:

```
foo(x = 5)
#> Error in foo(x = 5) : unused argument (x = 5)
```

E com `...` o resultado é:

```
foo(x = 5)
#> Error: 'foo' has been removed from this package
```

Faça uma página de manual para funções "defunct", como:

```
#' Defunct functions in helloworld
#'
#' These functions are gone, no longer available.
#'
#' \itemize{
#'  \item \code{\link{foo}}: This function is defunct.
#' }
#'
#' @name helloworld-defunct
NULL
```

Isso cria uma página de manual que os usuários podem acessar como `?` `helloworld-defunct` e que você verá no índice da documentação. Você pode adicionar quaisquer funções a essa página, conforme necessário. Você provavelmente desejará manter essa página de manual indefinidamente.

16.5.1 Testando funções descontinuadas

Você não precisa alterar os testes de funções descontinuadas até que elas se tornem "defunct".

- Considere todas as alterações feitas em uma função descontinuada. Além de usar `.Deprecated` dentro da função, você alterou os parâmetros na função descontinuada ou criou uma nova função que substitui a função descontinuada, etc.? Essas alterações devem ser testadas, caso você as tenha feito.
- Em relação ao que foi dito acima, se a função descontinuada estiver apenas recebendo uma alteração de nome, talvez você possa testar se as funções antiga e nova retornam resultados idênticos.
- `suppressWarnings()` poderia ser usado para suprimir o aviso lançado pelo `.Deprecated`, mas os testes não são voltados para o usuário e, portanto, não é tão ruim se o aviso for lançado nos testes, e o aviso pode até ser usado como um lembrete para o mantenedor.

Quando uma função se torna "defunct", seus testes são simplesmente removidos.

16.6 Renomeando pacotes

Não há problema em renomear um pacote que está em desenvolvimento inicial. Pode ser a oportunidade, antes da revisão, de estar em conformidade com os nossos conselhos de nomenclatura.

Renomear um pacote que já foi amplamente adotado e/ou lançado no CRAN é problemático. O CRAN tem uma política que afirma que os nomes de pacotes no CRAN são persistentes e, em geral, não é permitido alterar o nome de um pacote. O pacote com seu nome antigo pode ser uma dependência de outros pacotes, scripts e recursos em documentações, publicações científicas, postagens em blogs, entre outros.

Ao mudar radicalmente a interface, é melhor começar um novo pacote do zero, como o httr2, que é a segunda geração do httr; ou criar edições de um pacote, como o testthat. Se você também mantiver o pacote antigo, poderá fazer uma depreciação suave com uma mensagem de inicialização, como no pacote httr. Isso permite que as pessoas usuárias e autores(as) de pacotes escolham quando/se devem atualizar sua base de código para o novo pacote ou edição. Se você copiar o código de outro pacote, certifique-se de reconhecer os(as) autores(as) do código que você reutiliza, listando seus nomes no arquivo DESCRIPTION com um comentário que declare que essas pessoas foram autoras do pacote original. Exemplo.

16.7 Arquivamento de pacotes

Software geralmente tem uma vida útil finita, e os pacotes podem precisar ser arquivados. Os pacotes arquivados são arquivados e movidos para uma organização dedicada no GitHub, ropensci-archive. Antes do arquivamento, o conteúdo do arquivo README deve ser movido para um local alternativo (como "README-OLD.md") e substituído por um conteúdo mínimo, incluindo algo como o seguinte:

```
# <package name>

[![Project Status: Unsupported](https://www.repostatus.org/badges/latest/unsupported.svg)](https:
[![Peer-review badge](https://badges.ropensci.org/<issue_number>_status.svg)](https://github.com/

This package has been archived. The former README is now in [README-old](<link-to-README-old>).
```

O *badge* de status do repositório deve estar como "unsupported" (sem suporte) para pacotes lançados anteriormente ou como "abandoned" (abandonado) para pacotes de conceito anterior ou WIP (trabalho em progresso), caso em que o código do *badge* acima deve ser substituído por:

```
[![Project Status: Abandoned](https://www.repostatus.org/badges/latest/abandoned.svg)](https://ww
```

Um exemplo de um README mínimo em um pacote arquivado está em ropensci-archive/monkeylearn. Depois que o README tiver sido copiado em outro lugar e reduzido à forma mínima, você deverá seguir as etapas a seguir:

- ☐ Encerre os *issues* com uma frase que explique a situação e faça um link para este guia.
- ☐ Arquive o repositório no GitHub (também nas configurações do repositório).
- ☐ Transfira o repositório para ropensci-archive ou solicite um membro da equipe do rOpenSci para transferi-lo (você pode enviar um e-mail para `info@ropensci.org`).

Os pacotes arquivados podem ser desarquivados se os autores ou uma nova pessoa optarem por retomar a manutenção. Para isso, entre em contato com a rOpenSci.

Capítulo 17

Política de curadoria de pacotes

Este capítulo resume uma proposta de política de curadoria para a manutenção contínua de pacotes desenvolvidos como parte das atividades da rOpenSci e/ou sob a organização da rOpenSci no GitHub. Essa política de curadoria visa apoiar os seguintes objetivos:

- Garantir que os pacotes fornecidos pela rOpenSci estejam atualizados e sejam de alta qualidade
- Fornecer clareza quanto ao status de desenvolvimento e revisão de qualquer software nos repositórios da rOpenSci
- Gerenciar o esforço de manutenção para a equipe da rOpenSci, para os(as) autores(as) de pacotes e para os(as) colaboradores(as) voluntários(as)
- Fornecer um mecanismo para que os pacotes sejam descontinuados de forma adequada, mantendo o selo de revisão por pares

Elementos de infraestrutura descritos abaixo necessários para a implementação da política foram, em alguns casos, parcialmente construídos e, em outros casos, ainda não foram iniciados. Nosso objetivo é adotar essa política em parte para priorizar o trabalho nesses componentes.

17.1 O registro de pacotes

- O pacote rOpenSci registry é uma lista centralizada dos pacotes R que são mantidos atualmente (ou que foram mantidos anteriormente) pela rOpenSci. Ele contém metadados essenciais sobre os pacotes, incluindo o status de desenvolvimento e de revisão, e será a fonte de dados para exibição em sites, *badges*, etc. Ele permite que essa lista seja mantida de forma independente do pacote ou das plataformas de hospedagem de infraestrutura.

17.2 Pacotes mantidos pela equipe

Os pacotes mantidos pela equipe são pacotes desenvolvidos e mantidos pela equipe da rOpenSci como parte dos projetos internos da rOpenSci. Esses pacotes também podem ser revisados por pares mas não são necessariamente revisados por pares. Muitos desses pacotes estão fora do escopo da revisão por pares.

- Os pacotes mantidos pela equipe serão listados no registro com a tag "staff_maintained" e listados na página da Web de pacotes da rOpenSci, ou em locais similares com a tag "staff-maintained" (mantido pela equipe)
- Esses pacotes serão armazenados no dentro da organização no GitHub chamada "ropensci"
- Os pacotes mantidos pela equipe e seus documentos serão criados pelo sistema da rOpenSci. Esse sistema não envia notificações, mas gera resultados como status de commit do GitHub (o *red check mark* ou o *red cross*).
- Quando os pacotes falham nas verificações, a equipe da rOpenSci se esforça para corrigir as alterações, priorizando os pacotes com base no volume de usuários (isto é, o volume de *downloads*), de dependências reversas ou de objetivos estratégicos.
- Em uma base semestral ou anual, a rOpenSci analisará todos os pacotes que estão falhando há mais de um mês para determinar se você deve transferi-los para a organização "ropensci-archive" no GitHub.
- Pacotes que falham consistentemente e sem um plano contínuo para retornar para uma manutenção ativa, vão passar para o status de "archive". Quando arquivados, os pacotes da equipe serão movidos para o diretório "ropensci-archive" (a ser criado) e ganharão o tipo "archived" no registro. Eles não serão construídos no sistema da rOpenSci.
- Os pacotes arquivados não serão exibidos por padrão na seção de pacotes da página da Web. Esses pacotes serão exibidos em uma guia especial das páginas de pacotes com `"type": "archived"` que foram revisados por pares ou que foram mantidos pela equipe.

- Os pacotes arquivados podem ser desarquivados quando o mantenedor antigo ou um novo mantenedor estiver disposto a resolver os problemas e quiser reviver o pacote. Para isso, você deve entrar em contato com a rOpenSci. Esses pacotes serão transferidos para a organização ropenscilabs.

17.3 Pacotes revisados por pares

Os pacotes revisados por pares são aqueles contribuídos para a rOpenSci pela comunidade e que passaram pela revisão por pares. Eles precisam estar dentro do escopo no momento em que eles são enviados para serem revisados.

- Após o aceite, esses pacotes revisados por pares são transferidos do GitHub do(a) autor(a) para dentro da organização "ropensci" no GitHub ou, alternativamente, passam a ser monitorados pela adição deles a um arquivo JSON.
- Os pacotes revisados por pares estarão marcados no registro como "peer-reviewed", e terão um selo de revisão por pares em seu README.
- Os pacotes revisados por pares serão listados na página da Web da rOpenSci, ou em locais semelhantes, com a tag "peer-reviewed" (revisado por pares)
- Os pacotes revisados por pares e seus documentos serão construídos pelo sistema da rOpenSci. Esse sistema não envia notificações mas gera resultados como o status de commit do GitHub (o *red check mark* ou o *red cross*).
- Anualmente ou semestralmente, a equipe da rOpenSci revisará os pacotes que estão em estado de falha ou que estão falhando já por longos períodos, e entrará em contato com os autores para determinar o status da manutenção e das atualizações esperadas. Com base nesse intercâmbio, a rOpenSci pode optar por manter o status atual do pacote com a expectativa de uma atualização, ou contribuir com algum suporte, ou ainda, buscar um novo mantenedor, ou transferir o pacote para o status "archived".
- Com base no volume de usuários (isto é, o volume de *downloads* do pacote), ou das dependências reversas, ou dos objetivos estratégicos da rOpenSci, a equipe da rOpenSci pode apoiar os pacotes que estiverem com problemas e falhas, por meio de PRs que são revisados pelos autores dos pacotes, ou ainda, com alterações diretas (se os autores não responderem por aproximadamente um mês). A rOpenSci também fornecerá suporte aos autores de pacotes mediante solicitação, tanto pela equipe interna, quanto por voluntários da comunidade, de acordo com o tempo disponível.
- A pedido do autor, ou se os autores não responderem às consultas por aproximadamente um mês, a rOpenSci poderá procurar um novo mantenedor para os pacotes selecionados, que sejam revisados por pares, e que a rOpenSci

considere ter alta valor para a comunidade, com base no volume de usuários/*downloads*, ou nas dependências reversas, ou nos objetivos estratégicos da rOpenSci.

- Quando arquivados, esses pacotes serão movidos da organização "ropensci" para a organização "ropensci-archive" no GitHub (ou para a conta do autor no GitHub, caso for de desejo do autor), seguindo as orientações de transferência. Elas ganharão o tipo "archived" no registro. Esses pacotes vão manter as tags "peer-reviewed" (revisado por pares) e e *badges*. Eles não serão construídos no sistema da rOpenSci.
- Os pacotes arquivados não serão exibidos por padrão na seção de pacotes da página da Web. Esses pacotes serão exibidos em uma guia especial das páginas de pacotes com `"type": "archived"` que foram revisados por pares, ou que foram mantidos pela equipe.

17.4 Pacotes legado que foram adquiridos

Os pacotes "legado" são pacotes que não foram criados ou mantidos pela rOpenSci e que também não são revisados por pares, mas que estão sob o controle da rOpenSci no GitHub devido a razões históricas. (Antes de estabelecer a organização, e o seu processo de revisão por pares e o seu escopo, a rOpenSci absorveu pacotes de vários desenvolvedores sem critérios bem definidos).

- A rOpenSci transferirá os pacotes legado de volta para as organizações e repositórios dos autores. Se os autores não tiverem interesse, transferiremos para o repositório "ropensci-archive", seguindo as regras das orientações de transferência. Se os pacotes estiverem no escopo, a rOpenSci perguntará se os autores gostariam de submetê-los ao processo de revisão de software.
- Os pacotes legado não serão listados no registro de pacotes.
- Exceções podem ser feitas para pacotes que sejam partes vitais do ecossistema de pacotes do R e/ou da rOpenSci, e que sejam ativamente monitorados pela equipe.

17.5 Pacotes de incubadora

Os pacotes de "incubadora" são pacotes em desenvolvimento criados pela equipe ou por membros da comunidade como parte de projetos comunitários, como os criados por em desconferências.

- Os pacotes de incubadora ficarão na organização "ropenscilabs" no GitHub.

- Os pacotes de incubadora aparecerão no registro de pacotes com a tag "incubator".
- Os pacotes de incubadora não serão exibidos no site por padrão, mas as páginas de pacotes incluem uma guia especial de "pacotes experimentais".
- Os pacotes da incubadora e seus documentos serão criados pelo sistema da rOpenSci. Esse sistema não envia notificações mas gera resultados como o status de commit do GitHub (o *red check mark* ou o *red cross*). Os documentos indicarão claramente que o pacote é experimental.
- Semestralmente ou anualmente, a rOpenSci entrará em contato com os mantenedores desses pacotes de incubadora sobre repositórios que tenham pelo menos três meses de idade, perguntando sobre o status de desenvolvimento e as preferências dos autores sobre uma migração para o processo de revisão por pares, ou para o "ropensci-archive", ou para uma organização dos autores. Baseado em nas respostas, o pacote será migrado imediatamente, e a revisão por pares será iniciada, ou a migração será adiada para a próxima revisão. Os pacotes de incubadora serão migrados para o "ropensci-archive" por padrão, seguindo as orientações de transferência.
- Os pacotes de incubadora arquivados ganharão o tipo "archived".

17.5.1 Pacotes de incubadora que não sejam pacotes de R

- A organização da "incubadora" também pode incluir pacotes que não sejam pacotes de R.
- Esses projetos não serão listados no registro, e não vão aparecer no site da rOpenSci, e também não serão construídos automaticamente.
- A política de migração para esses pacotes será a mesma dos pacotes de R, com locais de migração apropriados (por exemplo, "ropensci-books")
- Se um pacote que não for um pacote de R for arquivado, ele será movido para a organização "ropensci-archive", seguindo as orientações de transferência.

17.6 Livros

Os livros da rOpenSci são documentações longas, geralmente no formato `bookdown`, e estão relacionados a pacotes, projetos ou temas da rOpenSci, criados tanto pelos autores de pacotes, quanto pela equipe da rOpenSci, e também por membros da comunidade.

- Os livros ficarão dentro da organização "ropensci-books" no GitHub.

- Os livros serão hospedados no domínio books.ropensci.org
- Os livros podem estar maduros ou em desenvolvimento, mas devem ter um mínimo de esboços/conteúdo antes de serem migrados da organização "ropenscilabs" para dentro da organização "ropensci-books".
- A autoria e o status de desenvolvimento de um livro devem ser claramente descritos em sua página inicial e no README.
- A rOpenSci pode fornecer *badges* ou modelos (por exemplo, "Em desenvolvimento," "Mantido pela comunidade") para os autores usarem nas páginas iniciais de seus livros.

Capítulo 18

Guia de contribuição

Este capítulo apresenta o nosso Guia de Contribuição, que descreve como você pode fazer contribuições com código e sem código para a rOpenSci.

- Então você quer contribuir para a rOpenSci? Fantástico! Nós desenvolvemos o Guia de contribuição da comunidade rOpenSci para dar as boas-vindas a você na rOpenSci e te ajudar a se reconhecer como um(a) colaborador(a) em potencial. O guia ajudará você a descobrir o que você pode ganhar doando seu tempo, conhecimento e experiência, combinando suas necessidades com coisas que ajudarão a missão da rOpenSci, e conectando você a recursos que podem te ajudar ao longo do caminho.

Nossa equipe e comunidade promovem ativamente um ambiente acolhedor em que pessoas usuárias e desenvolvedoras de diferentes origens e níveis de habilidade aprendem, compartilham ideias e inovam juntas abertamente por meio de normas e software compartilhados. A participação em todas as atividades da rOpenSci é apoiada por nosso Código de Conduta.

Aceitamos contribuições com código e sem código de pessoas programadoras novas ou experientes em qualquer estágio da carreira e em qualquer setor. Você não precisa ser um(a) desenvolvedor(a)! Talvez você queira **passar 30 minutos** compartilhando o caso de uso do seu pacote em nosso fórum público ou relatando um bug, **uma hora** aprendendo e participando de uma chamada da comunidade, **cinco horas** revisando um pacote de R enviado para revisão aberta por pares, **ou talvez você queira assumir um compromisso contínuo** para ajudar a manter um pacote.

Quais são alguns dos benefícios de contribuir?

- Conectar-se com uma comunidade que compartilha do seu interesse em tornar a ciência mais aberta

- Aprenda com pessoas de fora de seu domínio que usam o R com desafios semelhantes aos seus
- Faça e responda a novas perguntas de pesquisa conhecendo novas ferramentas de software e aliados
- Sentir-se confiante e com apoio em seus esforços para escrever código e desenvolver software
- Ganhar visibilidade para seu trabalho de código aberto
- Melhorar o software que você usa ou constrói
- Aprimore suas habilidades em R e ajude outras pessoas a aprimorarem as delas
- Aumente o nível de suas habilidades de escrita
- Obter mais exposição para o seu pacote

Consulte nosso Guia de contribuição e navegue pela seção "O que traz você aqui?" para saber quais são os *Eu quero …* que se encaixam melhor em você e escolha o seu caminho! Para ajudar você a se reconhecer, nós as agrupamos em: Descobrir; Conectar; Aprender; Construir; Ajudar. Para cada categoria, listamos exemplos de como essas contribuições podem ser e colocamos links para os nossos recursos para obter os detalhes de que você precisa.

Parte IV

Apêndice

Capítulo 19

NEWS

19.1 1.0.0

- 2026-02-05, clarify the Zenodo DOI recommentation (#985, `@Rafnuss`).
- 2026-02-12, updates checklists for onboarding and offboarding editors (#989).
- 2026-02-09, huge improvements to editor chapter (#981).
- 2026-01-26, update scope for data retrieval packages (#978).
- 2025-12-15, remove codemeta.json requirement (#968).
- 2025-10-17, clarify the use of Rd tags for internal functions (#918, `@cforgaci`).
- 2025-10-09, add translation of the last chapter to Portuguese!
- 2025-09-23, Add section on challenges (non-responding reviewers). Also move text on non-responding authors to this section. (#955).
- 2025-07-11, document better when the pkgdown websites of rOpenSci packages are re-built (#919).
- 2025-07-11, add minimal mention of example datasets (#868).
- 2025-07-17, add category for rOpenSci internal and peer-review tools (#848).
- 2025-07-09, add more details on how to safeguard docs building for rOpenSci packages (#910, `@rmgpanw`)
- 2025-07-11, add mentions of tools useful for translation and localization (#812).
- 2025-07-09, add mention of tinytest. (#904)

- 2025-07-09, add mention of the allcontributors package. (#899)
- 2025-07-09, stop recommending the use of the rOpenSci forum for package discussions. (#898)
- 2025-07-09, add a mention of ROR IDs (#909).
- 2025-07-09, remove the upper-case from the NEWS.md template and update the real example link. (#896)
- 2025-07-01, many typo fixes and English language improvements (#912, `@Moohan`)
- 2025-03-11, document drawbacks of renaming widely used package (#831)
- 2025-03-13, add a note on how to deprecate *data* (#649)
- 2025-04-10, add link to pkgcheck vignette on our testing environment (#589)
- 2025-04-10, replace the link to the Mozilla Code Review guide with explicit items (#835)
- 2025-04-03, document how to put the system on vacation(#865)
- 2025-04-03, add details about MEE process and structure the author guide a bit more (`@robitalec`, #862)
- 2025-03-11, add note in the packaging guide about checking maintenance status of dependencies (#881)
- 2025-03-11, add item about "top level code" in packaging guide (#879)
- 2025-03-11, explicitly mention need to acknowledge authors of bundled code (#873)
- 2025-03-27, add guidance for packages wrapping external software (#866)
- 2025-02-25, add official rule on submitting one package at once only (`@maurolepore`, #876)
- 2025-03-11, mention the Air formatting tool wherever we mention the styler package (#875)
- 2025-02-25, require the default git branch to not be called master (#863)
- 2024-09-06, update math guidance for pkgdown based on pkgdown's update (#838)
- 2024-08-30, remove mention of Twitter since rOpenSci no longer maintains an active Twitter account (`@yabellini`, #827)
- 2024-07-17, document dashboard in editors' chapter (`@mpadge`, #829)
- 2024-06-27, document the author's submit response step in the author guide (`@jmaspons`, #832).

19.2 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)

19.3 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)

19.4 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`.

19.5 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)

19.6 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)

19.7 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)

19.8 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)

19.9 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).

19.10 0.1.5

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

19.11 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.

19.12 place-holder 0.0.1

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

Capítulo 20

Modelo de revisão

Você pode salvar isso como um arquivo RMarkdown ou excluir o YAML e salvá-lo como um arquivo Markdown.

20.1 Revisão do pacote

Marque as caixas conforme aplicável e elabore nos comentários abaixo. Sua avaliação não se limita a estes tópicos, conforme descrito no guia do(a) revisor(a)

- **Descreva resumidamente qualquer relacionamento de trabalho que você tem (teve) com as pessoas autoras do pacote.**

☐ Como revisor(a), confirmo que não há conflitos de interesse para que eu revise este trabalho (se você não tiver certeza se está em conflito, fale com seu(sua) editor(a) *antes* de iniciar sua revisão)

20.1.0.1 Documentação

O pacote inclui todas as seguintes formas de documentação:

☐ **Uma declaração de necessidade:** declarando claramente os problemas que o software foi projetado para resolver além do seu público-alvo no README
☐ **Instruções de instalação:** para a versão de desenvolvimento do pacote e quaisquer dependências que sejam fora do padrão no README
☐ **Vignette(s):** demonstrando as principais funcionalidades com exemplos que são executados localmente com sucesso
☐ **Documentação de funções:** para todas as funções exportadas
☐ **Exemplos:** (que são executados localmente com sucesso) para todas as funções exportadas

- ☐ **Diretrizes da comunidade:** incluindo diretrizes de contribuição no README ou CONTRIBUTING, e DESCRIPTION com `URL`, `BugReports` e `Maintainer` (o qual pode ser gerado automaticamente via `Authors@R`).

20.1.0.2 Funcionalidade

- ☐ **Instalação:** Processo de instalação documentado conclui com sucesso.
- ☐ **Funcionalidade:** Qualquer funcionalidade que foi assumida pelo software foi confirmada.
- ☐ **Desempenho:** Qualquer desempenho a mais que foi assumido pelo software foi confirmado.
- ☐ **Testes automatizados:**Os testes unitários cobrem funções essenciais do pacote e uma gama razoável de *inputs* e condições. Todos os testes passam na máquina local.
- ☐ **Diretrizes de empacotamento**: O pacote está em conformidade com as diretrizes de empacotamento da rOpenSci.

Horas estimadas gastas na revisão:

- ☐ Caso as pessoas autoras do pacote considerem apropriado, concordo em ser reconhecido(a) como revisor(a) do pacote (função "rev") no arquivo DESCRIPTION do pacote.

20.1.1 Comentários da revisão

Capítulo 21

Modelo para o(a) editor(a)

21.0.1 Checks do editor:

- ☐ **Documentação**: O pacote possui documentação suficiente e está disponível online (`README`, `pkgdown docs`), de modo que permita um estudo de suas funcionalidades e escopo sem a necessidade de instalar o pacote. Em particular,

  - ☐ O caso de uso do pacote é bem feito?
  - ☐ A página principal (*index*) da documentação é clara (agrupada por tópicos se necessário)?
  - ☐ As documentações longas (*vignettes*) são legíveis, e detalhadas o suficiente ao invés de serem muito superficiais?

- ☐ **Adequação**: O pacote atende aos critérios de adequação e sobreposição.

- ☐ **Instruções de instalação**: As instruções de instalação são claras o suficiente para um ser humano?

- ☐ **Testes**: Caso o pacote possua algum produto interativo / HTTP / gráfico etc. os seus testes estão utilizando ferramentas em estado-de-arte?

- ☐ **Instruções para contribuição**: A documentação para contribuição é clara o suficiente (e.g. tokens para testes e áreas de playground)?

- ☐ **Licença:** O pacote possui uma licença aceita no CRAN ou OSI.

•

21.1 [ ] Gerenciamento do projeto: O monitoramento de problemas (*issues*) e PRs (*pull requests*) está em bom estado, e.g. existem bugs muito críticos, está claro quando um pedido de *feature* está planejado para ser tratado?

21.1.0.1 Comentários do(a) editor(a)

Capítulo 22

Modelo de solicitação de revisão

Os editores podem usar o modelo de e-mail abaixo para recrutar revisores.

Caro [REVISOR(A)]

Olá, aqui é [EDITOR(A)]. [BRINCADEIRA AMIGÁVEL]. Estou escrevendo para perguntar se você gostaria de revisar um pacote para a rOpenSci. Como você provavelmente sabe, a rOpenSci realiza revisão por pares de pacotes de R contribuídos para nossa coleção, de maneira semelhante aos periódicos.

O pacote, [PACOTE] de [AUTOR(ES)], faz [FUNÇÃO]. Você pode encontrá-lo no GitHub aqui: [LINK PARA REPOSITÓRIO]. Também conduzimos nosso processo de revisão aberta via GitHub, aqui: [ISSUE DE ONBOARDING]

Se você aceitar, observe que pedimos aos revisores que concluam as avaliações em três semanas. (Descobrimos que a revisão de um pacote leva um tempo semelhante ao de um trabalho acadêmico.)

Nosso guia para revisores detalha o que procuramos em uma revisão de pacote e inclui links para exemplos de revisão. Nossos padrões estão detalhados em nosso [guia de pacotes] e fornecemos um modelo de revisão para você usar. Certifique-se de que não haja um conflito de interesses que lhe impeça de revisar este pacote. Se você tiver dúvidas ou comentários, sinta-se à vontade para me perguntar.

A comunidade da rOpenSci é o nosso melhor ativo. Nosso objetivo é que as revisões sejam abertas, não adversas e focadas na melhoria da qualidade do software. Seja respeitoso(a) e gentil! Consulte nosso guia para revisores e o nosso código de conduta para mais informações.

[SE MENTORIA FOR REQUISITADA: Você indicou em seu formulário que prefere ter uma orientação para sua primeira revisão. Você é livre para me usar como recurso durante esse processo, incluindo fazer perguntas por e-mail e Slack (você receberá

um convite para o Slack da rOpenSci) e compartilhar os rascunhos de sua revisão para feedback antes de postá-los. Também estarei feliz de fazer uma breve videochamada para lhe explicar o processo. Por favor, me avise em sua resposta se você deseja agendar uma videochamada dessas.]

Você consegue revisar? Caso não consiga, você tem alguma sugestão de revisor(a)? Se eu não receber uma resposta sua dentro de uma semana, vou presumir que você não pode revisar neste momento.

Agradeço pelo seu tempo.

Atenciosamente

[EDITOR(A)]

Capítulo 23

Modelo de comentário de aprovação do(a) revisor(a)

23.1 Resposta do(a) revisor(a)

23.1.0.1 Aprovação final (pós-revisão)

- ☐ **O(a) autor(a) respondeu à minha revisão e realizou as mudanças requisitadas. Eu recomendo a aprovação deste pacote.**

Estimativa de horas dedicadas à revisão:

Capítulo 24

Modelo de notícias

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

### Novas funcionalidades

  * Nova função adicionada `do_things()` para fazer coisas (#5)

### Melhorias pequenas

  * Documentação foi aprimorada para a função `things()` (#4)

### Correções de bugs

  * Correção de um bug de parseamento em `parse()` (#3)

### Deprecado e extinto

  * `hello_world()` está deprecada agora e será removida em
     uma futura versão, utilize `hello_mars()`

### Correções em documentação

  * Esclareceu o papel de `hello_mars()` versus `goodbye_mars()`.


### (um especial: qualquer cabeçalho que agrupa um número grande de mudanças sobre uma única cois

    * blablabla.
```

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

### Novas funcionalidades

  * publicado no CRAN
```

Capítulo 25

Orientação para o lançamento de livros

Os(as) editores(as) que estão se preparando para um lançamento podem executar o script `prelease.R` na pasta `inst` deste repositório para abrir automaticamente um problema no GitHub com os pontos de verificação para todos os problemas atuais atribuídos ao marco da próxima versão, juntamente com a seguinte lista de verificação. Antes de executar o script, verifique manualmente a atribuição de problemas ao marco. Isso deve ser executado um mês antes do lançamento planejado.

25.1 Versão de lançamento do livro

25.1.1 Manutenção do repositório entre lançamentos

- ☐ Consulte a página de problemas para o guia dev e também para o repositório de revisões de software, procure por mudanças que ainda devem ser feitas no guia dev. Atribua os problemas encontrados no guia dev ao marco correspondente às versões, seja esta a próxima versão, ou, às versões seguintes, e.g versão 0.3.0. Encoraje novos PRs e revise eles.

25.1.2 1 mês antes do lançamento

- ☐ Lembre os editores de abrirem problemas/PRs para itens que desejam ver na próxima versão.
- ☐ Execute a função `devguide_prerelease()` do pacote `devguider`.

- ☐ Peça aos(às) editores(as) por qualquer feedback que você precise antes do lançamento.
- ☐ Para cada contribuição/alteração verifique se as NOTÍCIAS no arquivo `Appendix.Rmd` foram atualizadas.
- ☐ Planeje uma data para o lançamento e se comunique com o/a gerente da comunidade da rOpenSci, que lhe dará uma data para publicar uma postagem no blog (ou nota técnica).

25.1.3 2 semanas antes do lançamento

- ☐ Escreva um rascunho para uma postagem de blog (ou nota técnica) sobre o lançamento com antecedência suficiente para que os(as) editores(as) e, em seguida, o(a) gerente da comunidade, possam revisá-lo (2 semanas). Exemplo, instruções gerais para a postagem no blog, instruções específicas para as postagens de lançamento.
- ☐ Crie um PR a partir da branch `dev` para a branch `master` e, em seguida, comunique aos editores através do GitHub e do Slack. Mencione o rascunho da postagem do blog em um comentário dentro deste PR.

25.1.4 Lançamento

- ☐ Verifique as URLs usando a função `devguide_urls()` do pacote {devguider}
- ☐ Verifique a ortografia usando a função `devguide_spelling()` do pacote {devguider}. Atualize também a WORDLIST conforme necessário.
- ☐ Realize um squash sobre os seus commits para o PR de `dev` para `master`.
- ☐ Atualize a página de *release* do GitHub, e confira a página de *release* do Zenodo.
  - [ ] Reconstrua (para atualizar os metadados do livro no Zenodo) ou aguarde o processo diário de construção do livro.
- ☐ Crie novamente a branch `dev`.
- ☐ Conclua o PR com a sua postagem de blog (ou nota técnica). Destaque os aspectos mais importantes a serem destacados em tweets (e publicações) como parte da discussão do PR.

Capítulo 26

Como definir um redirecionamento

26.1 Site que não seja de páginas do Github Pages (por exemplo, Netlify)

Substitua o conteúdo do site atual por dois arquivos chamados `index.html` e `404.html`. Ambos os arquivos devem conter o seguinte conteúdo:

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

26.2 Páginas do GitHub

Você pode configurar o redirecionamento no repositório gh-pages do seu usuário principal:

- crie um novo repositório (se você ainda não tiver um): `https://github.com/<username>/<username>.github.io`
- Nesse repositório, crie um diretório `<pkgname>` contendo 2 arquivos: um `index.html` e `404.html` que redirecionam para o novo local (consulte a subseção anterior).
- Teste o endereço `https://<username>.github.io/<pkgname>/index.html` que vai redirecionar.

Capítulo 27

Comandos do bot

27.1 Para todos

Vale ressaltar que nós limpamos os tópicos de problemas ao remover todo conteúdo estranho, portanto, o registro de que você solicitou ajuda de bots será rapidamente apagado ou ocultado.

27.1.1 Veja a lista de comandos disponíveis para você

Se você precisar de um lembrete rápido!

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

27.1.2 Veja o código de conduta

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

27.2 Para autores

27.2.1 Verificar o pacote com o pkgcheck

Quando seu pacote tiver mudado substancialmente.

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

27.2.2 Enviar resposta aos revisores

Para registrar sua resposta aos revisores.

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

onde `<response_url>` é o link para o comentário de resposta no tópico do problema.

27.2.3 Finalizar a transferência do repositório

Depois que você aceitar o convite para a organização do GitHub do rOpenSci e transferir seu repositório do GitHub para ela, execute este comando para obter novamente o acesso de administrador ao seu repositório.

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

27.2.4 Obter um novo convite após a aprovação

Se você perdeu o prazo de uma semana para aceitar o convite para a organização do rOpenSci no GitHub, execute isso para receber um novo convite.

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

27.3 Para o editor-chefe

27.3.1 Atribua um (a) editor (a)

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

27.3.2 Colocar o envio em espera

Veja política editorial.

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

27.3.3 Indique que o envio está fora do escopo

Não se esqueça de publicar primeiro um comentário explicando a decisão e agradecendo ao(s) autor(es) pelo envio.

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

27.4 Para o editor designado

27.4.1 Colocar o envio em espera

Veja política editorial.

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

27.4.2 Verificar o pacote com o pkgcheck

Geralmente, apenas em consultas pré-submissão ou quando os autores indicam que o pacote foi substancialmente alterado.

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

27.4.3 Verificar padrões estatísticos

Geralmente, apenas em consultas pré-submissão ou quando os autores indicam que o pacote foi substancialmente alterado.

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

27.4.4 Verifique se o README tem o selo de revisão de software

No final do processo de envio.

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

27.4.5 Indique que você está procurando revisores

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

27.4.6 Atribuir um (a) revisor (a)

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

ou

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

27.4.7 Remover um (a) revisor (a)

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

27.4.8 Ajustar a data de vencimento da revisão

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

27.4.9 Registre que uma revisão foi enviada

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

27.4.10 Aprovar o pacote

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