The rworkflows suite: automated continuous integration for quality checking, documentation website creation, and containerised deployment of R packages

Brian M. Schilder
Department of Brain Sciences, Faculty of Medicine, Imperial College London, London, UK, W12 0BZ; UK Dementia Research Institute at Imperial College London, London, UK, W12 0BZ https://orcid.org/0000-0001-5949-2191

Alan E. Murphy
Department of Brain Sciences, Faculty of Medicine, Imperial College London, London, UK, W12 0BZ; UK Dementia Research Institute at Imperial College London, London, UK, W12 0BZ https://orcid.org/0000-0002-2487-8753

Nathan G. Skene (n.skene@imperial.ac.uk)
Department of Brain Sciences, Faculty of Medicine, Imperial College London, London, UK, W12 0BZ; UK Dementia Research Institute at Imperial College London, London, UK, W12 0BZ https://orcid.org/0000-0002-6807-3180

Article

Keywords: continuous integration, reproducibility, FAIR, containers, Docker, Singularity, workflows, GitHub, R packages, documentation, CRAN, Bioconductor

Posted Date: January 5th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2399015/v1

Abstract

Reproducibility is essential to the progress of research, yet achieving it remains elusive even in computational fields. Continuous Integration (CI) platforms offer a powerful way to launch automated workflows to check and document code, but often require considerable time, effort, and technical expertise to setup. We therefore developed the *rworkflows* suite to make robust CI workflows easy and freely accessible to all R package developers (https://github.com/neurogenomics/rworkflows). *rworkflows* consists of 1) a CRAN/Bioconductor-compatible R package template, 2) an R package to quickly implement a standardised workflow, and 3) a centrally maintained GitHub Action. Each time it is triggered by a push to a GitHub repository, it automatically creates virtual machines across multiple OS, installs all dependencies, runs code checks, builds/deploys a documentation website, and builds/deploys version-controlled containers with a built-in RStudio interface.

Additional analyses demonstrate that >50% of all R packages are only available via GitHub, highlighting the need for accessible solutions. Thus, *rworkflows* greatly reduces the barriers to implementing robust and reproducible best practices.

Introduction

Reproducibility is essential to the progress of research. Yet, >70% of researchers reported being unable to reproduce previously published results, according to a 2016 survey by *Nature* [1]. There are a variety of reasons contributing to this including pressure to publish, selective reporting, and methods not being reported in sufficient detail to replicate. Due to the computational nature of data analysis, there are unique opportunities to systematically maximise reproducibility and methodological transparency in this domain. Despite this, surveys of PubMed and GitHub have revealed that between 28-70% of bioinformatics software were never used beyond the original publication, and 68% have fewer than five citations [2,3]. Contributing factors may include a lack of coding standards, insufficient documentation, and discontinued maintenance post-publication. While general guidelines have been proposed for making bioinformatics software FAIR (Findable Accessible Interoperable Reusable) [4], exclusively placing the burden on individual developers to design and implement FAIR solutions is insufficient to stimulate substantial progress in this direction [5]. Instead, providing tools that can be easily applied to a wide variety of software applications with minimal effort and maximal reward for the individual developer are more likely to receive widespread adoption by the scientific community.

Within the sciences, especially bioinformatics and computational biology, R [6] has become one of the most commonly used programming languages [3,7]. Initiatives such as The Comprehensive R Archive Network (CRAN), Bioconductor (Bioc) [8,9], rOpenSci [10,11], and R-Forge have made great strides towards improving the accessibility and robustness of R packages through establishing centralised repositories that require certain coding/reproducibility standards. There are R functions to check whether a given R package meets best-practice coding standards include *rcmdcheck::rcmdcheck()* (for CRAN standards

when using the '*--as-cran*' flag) [12], *BiocCheck::BiocCheck()* (for Bioc standards) [13], and *pkgcheck::pkgcheck()* (for rOpenSci standards) [14]. However, initially learning how to set up R packages such that they are compatible with these standards, and manually rerunning checks to ensure they continue to meet these standards, incur non-trivial costs in terms of both time and effort. Even if all checks pass on one's local machine, this does not guarantee that the same software will run as expected on a different Operating System (OS) (e.g. due to version/availability conflicts across many software dependencies). As most journals, funders, and institutions do not currently require or systematically enforce any standards regarding passing quality tests, it is therefore usually left to each research group to decide how rigorously they test their software. Presently, many softwares are exclusively distributed through GitHub (e.g. via the function *remotes::install_github()*), due to the ease of doing so and the perceived challenges of submitting to dedicated R package repositories such as CRAN/Bioc/rOpenSci. Unlike these dedicated R package repositories, GitHub does not require R packages (or any other software) to meet any quality standards, or even install or run. In the absence of additional safeguards, this leaves even more opportunities for such softwares to fail or produce erroneous results.

A prevalent culture of openly sharing software source code and study-specific analysis scripts on freely available repositories has helped move the scientific computing community closer to the goals of FAIR. Over the last decade, GitHub has rapidly overtaken all other code repositories as by far the most widely used in the fields of bioinformatics and computational biology (>90% in 2017) [3]. At the same time, there have been considerable developments in the scope and depth of tools built directly into the GitHub architecture, including the relatively recent addition of GitHub Actions (GHA). GHA allows any user to run customised Continuous Integration (CI) workflows directly on GitHub servers for free and can be triggered simply by pushing updates to one's GitHub repository as usual. These workflows can call upon other bundled scripts hosted elsewhere on GitHub to perform sets of related steps, called "actions". These actions can be triggered to automatically launch by user-selected events, including pushes and pull requests. This ensures that every time a change is made to the underlying code, the software continues to work as expected across multiple OS with a fresh install of all dependencies. However, setting up these workflows currently takes considerable time, effort, and technical expertise.

In an effort to promote FAIRness, as well as enhance software usability and longevity, we developed *rworkflows*: a robust, reusable, flexible and automated CI suite specifically for the development of R packages. *rworkflows* includes three main components: 1) ***templateR* template**: a CRAN/Bioc-compatible R package template that automatically generates essential documentation using package metadata, 2) ***rworkflows* R package**: a lightweight CRAN package to automatically setup short, customisable workflows that trigger the *rworkflows* action, and 3) ***rworkflows* action**: an open-source action available on the GHA Marketplace (see **Methods** for a more detailed description of each step in the *rworkflows* action). Importantly, the *rworkflows* action is designed to work with any R package out-of-the-box and can be set up by a one-time call to the R function *rworkflows::use_workflow()*. This means users do not need to manually edit any workflow scripts, obviating the need to invest time in learning GHA-specific syntax or configuration. In addition, the *rworkflows* action produces three main resources. First, a fully

containerized installation of the R package and all of its dependencies are automatically created and pushed to Docker Hub so that users can easily install local copies of the fully setup environment as either Docker or Singularity containers. Second, it creates a dedicated documentation website entirely from *README* files, in-code *roxygen* notes [15] and vignettes [16], and then deploys the website to the associated GitHub repository via GitHub Pages. Finally, a variety of status reports can be directly displayed in the *README*/landing page of the GitHub repository as badges (e.g. whether all GHA have been passed, code coverage reports, number of downloads, last commit date) [17], allowing maintainers and users to immediately assess the current state of the software package.

In an effort to assist the development community in adopting *rworkflows* and make it a *de facto* standard for R package maintainers, we have already begun to expand its user base by making Pull Requests to GitHub repositories of R packages. In particular, we have focused on R packages that have a large user base (e.g. *Seurat* [18,19]) or are core Bioc dependencies that thousands of other softwares rely upon (e.g. *GenomicRanges* [20], *rtracklayer* [21], *RSamtools* [22], *VariantAnnotation* [23]). We also present novel evidence that over 52% of all R packages currently in existence are exclusively distributed via GitHub. This further emphasises the need for robust, GitHub-based quality control/documentation standards that can be frictionlessly utilised by non-experts.

Results

rworkflows adoption

To date, *rworkflows* has been successfully implemented in over 11 R packages hosted on GitHub. This includes packages both internal and external to our own research group, as well as the *rworkflows* R package itself. To illustrate this, we created a graph illustrating all R packages that currently use *rworkflows*, or depend on packages that do (i.e. second-order dependents) (**Fig. 2**). As a proxy of *rworkflows*'s downstream impact on the R development community, metadata was systematically gathered from GitHub. Totals across all 40 dependents (excluding *rworkflows* itself) there were: 145 stars, 93 unique clones, 70 forks, 287 unique views, and 8,182 downloads (across all distribution repositories).

An interactive and continuously updated version of this graph on the dedicated *rworkflows* documentation website (see **Data availability** section). This online version also displays the metadata for each repository when users hover the cursor over the respective node.

GitHub as a package distributor

Many developers who distribute their R packages through dedicated repositories like CRAN, Bioc or rOpenSci still maintain a copy of their software on GitHub for the purposes of development, collaboration and transparency. However many packages go through a lengthy period of development (months to years) before being eventually accepted to one of the dedicated R package repositories. In fact, many

developers may never submit their packages to these dedicated repositories, and depending on where and if they publish their work, these packages can be introduced into the scientific community without ever being thoroughly tested. As more software becomes exclusively distributed on GitHub, there is an increased need for GitHub-native solutions which make CI seamless. Since there are currently few to no set standards imposed by journals or GitHub, it is incumbent upon the R developer community to provide tools which not only make best-practice coding, documentation and CI easy to implement, but immediately beneficial enough to incentivise researchers to widely adopt these practices.

To evaluate the magnitude of need for GitHub-based solutions in the R community, we gathered comprehensive data on which repositories R packages are hosted on (**Fig. 3**). An upset plot was generated to visualise how many R packages are distributed via one or multiple repositories. Of the 49,469 R packages we identified, 38.2% (18,911) are available via CRAN, 7.13% (3,526) are available via Bioc, 0.613% (303) are available via rOpenSci, 4.37% (2,162) are available via R-Forge, and 63.9% (31,592) are available on GitHub. Of particular note, 52% (25,713) of all R packages are exclusively distributed through GitHub. This is likely a very conservative underestimate, as the data on GitHub R packages comes from a static snapshot previously collected in February 2018, whereas all the CRAN/Bioc/rOpenSci/R-Forge data is fully up-to-date. Thus, over half of all R packages are currently not vetted by dedicated R package distributors and are instead left to the developers to determine their own standards and strategies for reproducibility.

Comparisons with *biocthis*

It should be noted that there has been at least one other effort to implement reproducible workflows for R package development via GHA, namely through the Bioc R package *biocthis* [24]. While *rworkflows* was heavily influenced by *biocthis*, there are several key differences. First, *rworkflows* operates primarily as an action which is merely called upon by a short workflow script that supplies certain parameters, whereas *biocthis::use_bioc_github_action()* generates a static workflow script that dictates each step of the workflow in the file itself. This distinction becomes important when updates need to be made (e.g. new system dependencies, changes to R function implementations, deprecation of certain actions). Actions such as *rworkflows* need only be updated on the centralised Github repository (see **Code availability** section), which then propagates to all users who call the *rworkflows* action, even if they implemented *rworkflows* in their package prior to the changes. In contrast, the static *biocthis* workflow scripts must be updated by every user individually. In some cases, it may take a while for the user to infer that the errors they're experiencing are due to changes in the VM provided by the GHA server (for example), rather than something the user is doing wrong, or eventually abandon using the workflow entirely. That said, if users wish to create a more customised workflow that diverges from the *rworkflows* action (and only use it as an initial basis for their script), a full workflow version can be created with *rworkflows::use_workflow(name="rworkflows_static")*, which offers functionality analogous to that of *biocthis::use_bioc_github_action()*.

Second, *rworkflows* is more flexible in several regards. Users can easily control which version of the *rworkflows* action to use with the *tag* argument to indicate a branch (e.g. “master” for the latest version) or release tag (e.g. “v1” for a stable release version tied to a specific commit). *biocthis* does not currently provide the ability to use different versions of the same workflow, unless users install a different release of Bioc (and all Bioc packages) each time they wanted to use a different version of the workflow. Furthermore, *rworkflows* offers greater customisability with additional flags (e.g. *as_cran*, *on*, *branches*, *run_vignettes*, *cache_version*) as well as optionals to support *act* [25], a separate software for running and troubleshooting actions locally before launching them to GitHub.

Finally, *rworkflows* obviates the need for a user-supplied Dockerfile as it creates one on the fly instead (see section **Container usage**). This level of abstraction serves to expand the usage of containers to those who do not know how to successfully set them up manually, or are unfamiliar with the Docker-specific syntax necessary to do so. None of this is to say that the *biocthis* package is obsolete, but rather that it offers other complementary features such as more fine-grained control over template creation than the all-in-one strategy adopted by *templateR*, as well as automated code styling.

Comparisons with Bioconductor servers

The *rworkflows* suite is not mutually exclusive to the package checking services provided by Bioc, which regularly run standardised checks on multiple OS. To the contrary, *rworkflows* fills an important gap for developers of Bioc packages who wish to comprehensively test their package before pushing to the upstream Bioc copy, as the upstream copy can take several days to rerun checks. Having an intermediate checking solution via GitHub provides feedback within minutes or hours, as opposed to days, thus greatly accelerating the development cycle. While Bioc does provide a dedicated Docker container with several prerequisite software installed (e.g. *BiocManager*, *BiocCheck*), these containers do not have any other Bioc packages installed. *rworkflows* in fact uses the Bioc Docker container as a base and then builds upon it to generate a package-specific containerised environment ready for distribution to users. This greatly speeds up the time it takes for any given user to successfully install and start using the developer's R package.

Discussion

Most researchers would agree that FAIR principles are noble goals and something the field as a whole should strive for over time. However, the costs associated with putting these principles into practice (e.g. time, learning curve, lack of computational resources) often deter researchers from ever effectively implementing them. Therefore, there is a dire need to reduce the burden put on individual researchers by automating reproducibility-promoting strategies, while at the same time increasing the amount of useful output generated by such strategies. This will greatly improve the overall cost/benefit ratio of highly reproducible science, which will ideally incentivise widespread adoption of best practices in reproducibility and open science. *rworkflows* aims to do exactly this, by enabling the implementation of a

robust GHA through a single R function that is usable by even novice programmers and requires minimal local computing power.

Peer-reviewed journals, as well as repositories like CRAN, Bioc, and rOpenSci, rely almost entirely on volunteer community members to review and approve software packages for official release [11,26]. Each additional cycle in the review-response process due to common and avoidable issues can incur substantial and unnecessary delay. This is only exacerbated by the limited time and considerable demands both parties are faced with [27,28]. *rworkflows* serves to significantly reduce the burden of back-and-forth troubleshooting by decreasing the prevalence of installation errors (through containerisation), coding bugs (through package checks), and miscommunications (through documentation). As the exponentially expanding scientific literature continues to outpace the proportion of qualified researchers willing to volunteer as reviewers [28], making this process more efficient will become increasingly critical for the sustainability of timely, high-quality peer-reviewed research [26,29]. Therefore, journals may wish to consider requiring tools such as *rworkflows* to be implemented as a prerequisite for progressing the review process.

Providing containerised environments with all necessary dependencies pre-installed and an interactive development platform (i.e. *RStudio*) virtually eliminates the troubleshooting installation. This also helps reduce the burden of maintaining software across hundreds to thousands of users, each with one or more slightly different computing environments. As an additional incentive to developers, continued maintenance of bioinformatics tools post-publication is associated with several metrics of impact [3]. *rworkflows* also allows users to control which versions of R and Bioc they wish to have installed within the container. By default, it uses the most up-to-date development versions of R/Bioc so that developers can stay ahead of the curve and identify issues in future versions before they have been released to the public. This is important, as it prevents situations where developers are suddenly faced with many bugs that are already affecting a large number of users and must be fixed urgently.

Beyond the initial publication of an R package, *rworkflows* offers a variety of benefits for different stakeholders. Automating clean and consistent documentation website generation without any additional effort encourages developers to keep their documentation up to date and accessible. Having thorough documentation is not only an invaluable resource for new users, but also trainees in the developers' own lab, or even when reteaching themselves after a long period of not being active on the project.

To conclude, the *rworkflows* suite offers essential tools for developers and users at any level of experience. This includes developers who 1) currently (or plan to) distribute their R packages through repositories like CRAN/Bioc/rOpenSci and want to run quality checks before resubmitting a new version for official release, 2) wish to exclusively distribute their code through GitHub while maintaining a high level of coding standards, 3) want to keep the documentation updated without constant manual upkeep of a website, 4) want to distribute their software in a fully reproducible Docker/Singularity container. Therefore, *rworkflows* fills a gap that an increasing number of R developers find themselves in by

reducing the burden of FAIR practices, and increasing its immediate benefits for developers and users alike.

Methods

templateR template

For users who are creating a new R package from scratch, we have provided a CRAN/Bioc-compatible template (*templateR)*. To get started, one simply forks the template by navigating to the GitHub repository (see **Code availability** section), clicking “Use this template”, and cloning a copy of the new R package to begin editing it (**Fig. 1a**). The user need only replace key metadata fields (e.g. Package, Title, Description, URL) in the *DESCRIPTION* file (a required file for all R packages). What makes this template unique is that all other components of the package (README, vignettes, unit test setup scripts) are programmatically autofilled based on the *DESCRIPTION* file. This strategy greatly minimises redundant and error-prone aspects of R package documentation.

Alternatively, users can start with any pre-existing R package and skip directly to the next step: using *rworkflows* R package. In either case, we have created a companion Wiki page to help guide users who are unfamiliar with the Bioc standards and offer a variety of tips and tricks to make this process easier, which we continue to maintain (see **Code availability** section).

rworkflows R package

The *rworkflows* R package is available on both CRAN and GitHub (see **Code availability**). Workflow scripts (written in *yaml* format) placed within a specific subdirectory within the GitHub repository (*.github/workflows/*.yml*), dictate which actions are triggered under which conditions. For those not familiar with creating GHA workflows, learning the GHA-specific expressions and idiosyncrasies can be a time-consuming and iterative process. Instead, we have abstracted this step away by autogenerating workflow scripts from a single R command in the dedicated R package: *rworkflows::use_workflow()*. This creates a fully functional workflow file in the correct subdirectory even with no arguments supplied, and only needs to be run once per R package (**Fig. 1b**). For greater flexibility, users can supply the function with their preferred arguments to generate (or regenerate) a customised workflow script to trigger the *rworkflows* action. By default, the workflow will trigger the *rworkflows* action (see ***rworkflows* action** section below) upon pushes or pull requests to the remote GitHub repository. For minor pushes (e.g. fixing a typo in the *README* text), one can avoid triggering the action by simply adding the string “[skip ci]” to the commit message. Triggers can be set to activate for specific GitHub branches only (e.g. “main”, “master”) or even *regex* expressions (e.g. “RELEASE_**”), which can be quite helpful for developing Bioc packages with regular release updates without having to modify the workflow script each time. Finally, the *rworkflows::use_workflow()* allows users to control exactly which specific release of the *rworkflows*

action they wish to trigger (via the *tag* argument). For a full description of all arguments of the *rworkflows::use_workflow()* function, please refer to **Table S1**.

In addition, the *rworkflows* R package contains other useful functions for developers, including *rworkflows::use_badges()*, which dynamically generates badges indicating various aspects of the software package's status to the documentation pages (e.g. the *README* file). It also provides the function *rworkflows::use_dockerfile()*, which writes a Docker recipe file (i.e. Dockerfile) to create a Docker image with the user's R package (and all of its dependencies) pre-installed). Note that this same function is called automatically in step 8 of the *rworkflows* action, but if a pre-existing Dockerfile in the current working directory is detected, this step is skipped and the pre-existing Dockerfile is used instead. Thus, if preferred, users can have more customised control over how their Docker container is configured. Finally, *rworkflows::use_readme(), rworkflows::use_vignette_docker()* and *rworkflows::use_vignette_getstarted()* can generate autofiled templates for each of these R package documentation components respectively.

rworkflows action

Once triggered by a workflow, the *rworkflows* action launches three virtual machines (VM) in parallel to test the R package across multiple OS, including Linux, Mac, and Windows. Within each VM, the following steps are performed (**Fig. 1d**):

1. **Install system**: Installs all OS-specific system dependencies that account for a variety of different functionalities that R users may .
2. **Install R**: Installs all R dependencies for the R package being tested. Three rounds dependency installation are attempted using slightly different methods to ensure robustness of this procedure without requiring the user to manually troubleshoot this step.
3. **CRAN checks**: Run CRAN checks via *rcmdcheck::rcmdcheck()*. When *run_rcmdcheck=TRUE*, all checks must pass in order for the GHA to succeed. This step uses CRAN standards by default, but can run *rcmdcheck* without CRAN standards by setting the argument *as_cran=FALSE*.
4. **Bioc checks**: Run Bioc checks via *BiocCheck::BiocCheck()*. When *run_bioc=TRUE*, all checks must pass in order for the action to succeed.
5. **Unit tests**: Runs unit tests implemented via the *testthat*[30] and/or *RUnit*[31] R packages and generates a downloadable report of the results.
6. **Code coverage**: Runs code coverage tests and uploads the results to Codecov.
7. **Build website**: (Re)builds the documentation website from *README* files, in-line *roxygen* notes, and vignettes using the *pkgdown*[16]. It then deploys the website via GitHub Pages in a new branch named "gh-pages" in the same repository. Deploying the website via a separate branch is advantageous as it avoids accidentally adding large HTML/CSS/JavaScript source files and libraries to the R package itself (which can slow down its installation and performance in some situations).

8. **Push container**: Pushes a container to Docker Hub with your R package, all of its dependencies, and an interactive Rstudio interface pre-installed. Included in *templateR* is an auto-filled vignette for how to create a local Docker or SIngularity container. This step requires a valid DockerHub authentication token, which can be stored as a GitHub Secrets variable. This ensures that only users with appropriate push permissions to a given Docker Hub account can update the container there.

Steps 6-8 are only run on the Linux VM to avoid redundancy and avoid conflicts due to simultaneous pushes to their respective repositories (i.e. Codecov, GitHub, Docker Hub).

Container usage

Containerisation is especially useful when distributing R packages to many users using a wide variety of OS platforms, including high-performance computing (HPC) clusters which may have software installation restrictions for non-root users. Once the *rworkflows* action has successfully completed at least once on the Linux VM, both developers can create Docker and/or Singularity images from the container hosted on Docker Hub. If *templateR* was used as a template, a vignette detailing a step-by-step reproducible example is autogenerated. A rendered version of this vignette can be accessed via the dedicated GitHub Pages site, and a link to this vignette is automatically rendered within the *templateR* template *README* file (see **Code availability** section) under the "Documentation → Docker/Singularity" subheader.

rworkflows adoption

Metadata was gathered from the GitHub application programming interface (API) for each repository using the R packages *echodeps* [32]. This was used to both identify which packages are currently using the *rworkflows* action (i.e. dependents), and to gather relevant metadata on each of the repositories. Of particular interest were the following metrics; stars (the number of users that bookmarked the GitHub repo with a star), unique clones (the number of unique instances that the GitHub repo was downloaded from Github), and unique views (the number unique instances the GitHub repo was viewed in a web browser). Here, "unique" means the number of distinct internet protocol (IP) addresses. Sums of each of these metrics across all were computed to represent the total downstream impact of *rworkflows*. All dependents were visualised as nodes in a directed graph, connecting to an additional node representing the *rworkflows* action (**Fig. 2**).

To identify the R packages with the highest potential for downstream impact on other packages, we collected data on the number of downloads for every packages in CRAN and Bioc using *echogithub*(Schilder et al. 2021). We then selected the packages with the greatest numbers of downloads and prioritised them for making Pull Requests on their respective GitHub repos to implement *rworkflows*.

An R markdown script to fully reproduce these analyses, as well as an interactive version of the graph with additional metadata, is available as a vignette on the official *rworkflows* GitHub Pages documentation website (See the **Code availability** section for link).

GitHub as a package distributor

To comprehensively assess which repositories R packages are distributed via, we collected metadata on all known R packages from base R, CRAN, Bioc, rOpenSci, R-Forge, and GitHub using the package *echogithub*[32]. The total and intersection between packages in each of these repositories were then computed and visualised using the R package *UpSetR*[33] (**Fig. 3**).

It should be noted that the data on GitHub-hosted R packages comes from a static snapshot previously collected in February 2018 via the *echogithub* dependency *githubinstall* [34], whereas all the CRAN/Bioc/rOpenSci/R-Forge data is fully up-to-date. This means that our estimates of the proportion of R packages that are distributed exclusively through GitHub are almost certainly an underestimate. An R markdown script to fully reproduce these analyses is available as a vignette on the *rworkflows* documentation website (See the **Code availability** section).

Declarations

Acknowledgements

The authors would like to thank the development/maintenance teams at GitHub, Bioconductor, and CRAN, as well as the respective contributors of the GitHub Actions that *rworkflows* depends on.

Funding

This work was supported by a UK Dementia Research Institute (UK DRI) Future Leaders Fellowship [MR/T04327X/1] and the UK DRI which receives its funding from UK DRI Ltd, funded by the UK Medical Research Council, Alzheimer’s Society and Alzheimer's Research UK.

Data availability

Up-to-date data on *rworkflows* adoption:

https://github.com/neurogenomics/rworkflows/network/dependents

Wiki page on creating a Bioconductor package:
https://github.com/neurogenomics/labwiki/wiki/Creating-a-Bioconductor-package

Bioconductor Guidelines for Mentors and Mentees:

https://bioconductor.github.io/bioc_mentorship_docs

Code availability

Each component of *rworkflows* is freely available on GitHub and/or Docker Hub:

templateR R package template:

https://github.com/neurogenomics/templateR

rworkflows R package:

https://github.com/neurogenomics/rworkflows

rworkflows GitHub Action:

https://github.com/marketplace/actions/rworkflows

rworkflows Docker container:

https://hub.docker.com/repository/docker/neurogenomicslab/rworkflows

rworkflows Docker/Singularity container vignette:

https://neurogenomics.github.io/rworkflows/articles/docker

rworkflows dependency graph vignette:

https://neurogenomics.github.io/rworkflows/articles/depgraph

R package repository distribution vignette:

https://neurogenomics.github.io/rworkflows/articles/repos

Supplementary Materials

Links

act:

https://github.com/nektos/act

Codecov:

https://codecov.io

Docker Hub:

https://hub.docker.com

GitHub Actions documentation:

https://github.com/features/actions

GitHub Pages:

https://pages.github.com

References

1. Baker, M. 1,500 scientists lift the lid on reproducibility. *Nature* **533**, 452–454 (2016).
2. Duck, G. *et al.* A Survey of Bioinformatics Database and Software Usage through Mining the Literature. *PLoS One* **11**, e0157989 (2016).
3. Russell, P. H., Johnson, R. L., Ananthan, S., Harnke, B. & Carlson, N. E. A large-scale analysis of bioinformatics code on GitHub. *PLoS One* **13**, e0205898 (2018).
4. Wilkinson, M. D. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3**, 160018 (2016).
5. Clarke, D. J. B. *et al.* FAIRshake: Toolkit to Evaluate the FAIRness of Research Digital Resources. *Cell Syst* **9**, 417–421 (2019).
6. Ihaka, R. & Gentleman, R. R: A Language for Data Analysis and Graphics. *J. Comput. Graph. Stat.* **5**, 299–314 (1996).
7. Giorgi, F. M., Ceraolo, C. & Mercatelli, D. The R Language: An Engine for Bioinformatics and Data Science. *Life* **12**, (2022).
8. Gentleman, R. C. *et al.* Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol.* **5**, R80 (2004).
9. Huber, W. *et al.* Orchestrating high-throughput genomic analysis with Bioconductor. *Nat. Methods* **12**, 115–121 (2015).
10. Boettiger, C., Chamberlain, S., Hart, E. & Ram, K. Building software, building community: Lessons from the rOpenSci project. *J. Open Res. Softw.* **3**, 8 (2015).
11. Ram, K. *et al.* A Community of Practice Around Peer Review for Long-Term Research Software Sustainability. *Comput. Sci. Eng.* **21**, 59–65 (2019).

12. Hornik, K. Are There Too Many R Packages? *AJS* **41**, 59–66 (2012).
13. Bioconductor, Shepherd, L. & Ramos, M. *BiocCheck: Bioconductor-specific package checks*. (2022). doi:doi:10.18129/B9.bioc.BiocCheck.
14. Wujciak-Jens, M. P. M. S. *pkgcheck: Check whether a package is ready for submission to rOpenSci's peer-review system*. (rOpenSci, 2022).
15. Wickham, H., Danenberg, P., Csárdi, G. & Eugster, M. roxygen2: In-Line Documentation for R, 2020. *R package version*.
16. Wickham, H. & Hesselberth, J. Pkgdown: Make static html documentation for a package. *R package version*.
17. Yu, G. *badger: Badge for R Package*. (2022).
18. Hao, Y. *et al.* Integrated analysis of multimodal single-cell data. *Cell* **184**, 3573–3587.e29 (2021).
19. Satija, R., Farrell, J. A., Gennert, D., Schier, A. F. & Regev, A. Spatial reconstruction of single-cell gene expression data. *Nat. Biotechnol.* **33**, 495–502 (2015).
20. Lawrence, M. *et al.* Software for computing and annotating genomic ranges. *PLoS Comput. Biol.* **9**, e1003118 (2013).
21. Lawrence, M., Gentleman, R. & Carey, V. rtracklayer: an R package for interfacing with genome browsers. *Bioinformatics* **25**, 1841–1842 (2009).
22. M Morgan, H Pagès, V Obenchain, N Hayden N. *Rsamtools: Binary alignment (BAM), FASTA, variant call (BCF), and tabix file import*. (2022).
23. Obenchain, V. *et al.* VariantAnnotation: a Bioconductor package for exploration and annotation of genetic variants. *Bioinformatics* **30**, 2076–2078 (2014).
24. Collado-Torres, L. *biocthis: automate package and project setup for Bioconductor packages*. (2022). doi:10.18129/B9.bioc.
25. Lee, C. *act: run your GitHub actions locally*. (2022).
26. Vesper, I. Peer reviewers unmasked: largest global survey reveals trends. *Nature Publishing Group UK* http://dx.doi.org/10.1038/d41586-018-06602-y (2018) doi:10.1038/d41586-018-06602-y.
27. Woolston, C. How burnout and imposter syndrome blight scientific careers. *Nature Publishing Group UK* http://dx.doi.org/10.1038/d41586-021-03042-z (2021) doi:10.1038/d41586-021-03042-z.
28. Milojević, S., Radicchi, F. & Walsh, J. P. Changing demographics of scientific careers: The rise of the temporary workforce. *Proc. Natl. Acad. Sci. U. S. A.* **115**, 12616–12623 (2018).
29. Petrescu, M. & Krishen, A. S. The evolving crisis of the peer-review process. *Journal of Marketing Analytics* **10**, 185–186 (2022).
30. Wickham, H. Testthat: Get started with testing. *R J.* **3**, 5 (2011).
31. Matthias Burger, Klaus Juenemann, Thomas Koenig, Roman Zenka. *RUnit: R Unit Test Framework*. (2018).
32. Schilder, B. M., Humphrey, J. & Raj, T. echolocatoR: an automated end-to-end statistical and functional genomic fine-mapping pipeline. *Bioinformatics* (2021)

doi:10.1093/bioinformatics/btab658.

33. Conway, J. R., Lex, A. & Gehlenborg, N. UpSetR: an R package for the visualization of intersecting sets and their properties. *Bioinformatics* **33**, 2938–2940 (2017).
34. Makiyama, K. *githubinstall: A Helpful Way to Install R Packages Hosted on GitHub*. (2018).

Figures

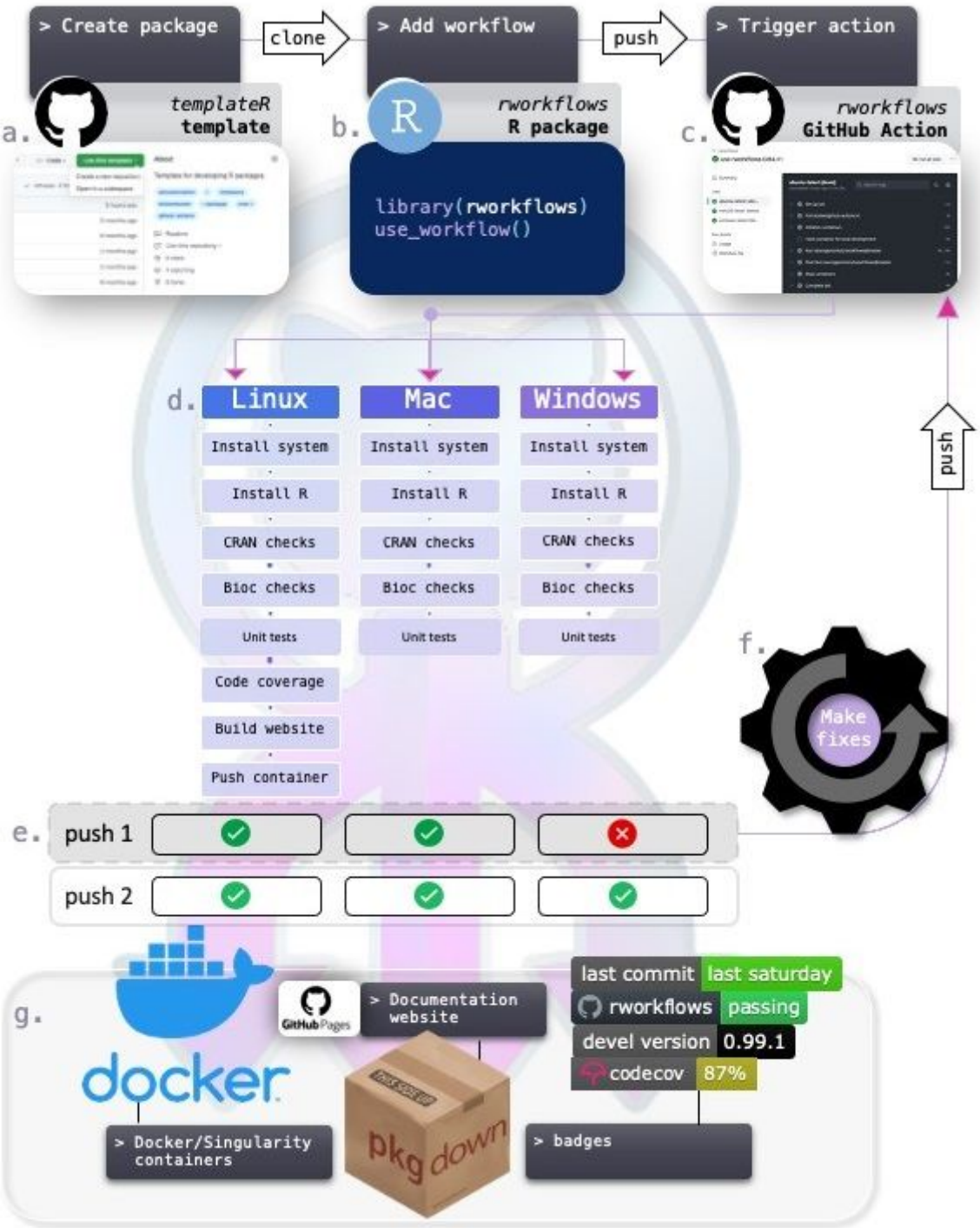

Figure 1

The rworkflows suite

Example usage of *rworkflows*. **a. Create package**: create a new R package by forking and cloning the *templateR* template, or use an existing R package. **b. Add workflow**: Install the rworkflows R package and use the *use_workflows()* command to generate a workflow *yaml* file in the correct folder structure. Arguments to customise the workflow are detailed in **Table S1**. **c. Trigger action**: trigger the rworkflows GitHub Action by pushing to GitHub. **d.** Run the R package through the workflow on three different OS platforms in parallel, **e.** Inspect the results of the workflow run. If one or more workflows fail, an email is automatically sent to the user. **f.** If issues are found, make fixes to the software and push again to retrigger the *rworkflows* action. **g.** When all workflows have passed, the documentation website is built using *pkgdown* [16] and deployed via GitHub Pages. The containerised R package is then deployed to Docker Hub. Badges embedded into markdown or HTML files (e.g. *README* documentation) will also be automatically updated to reflect the R package's current status.

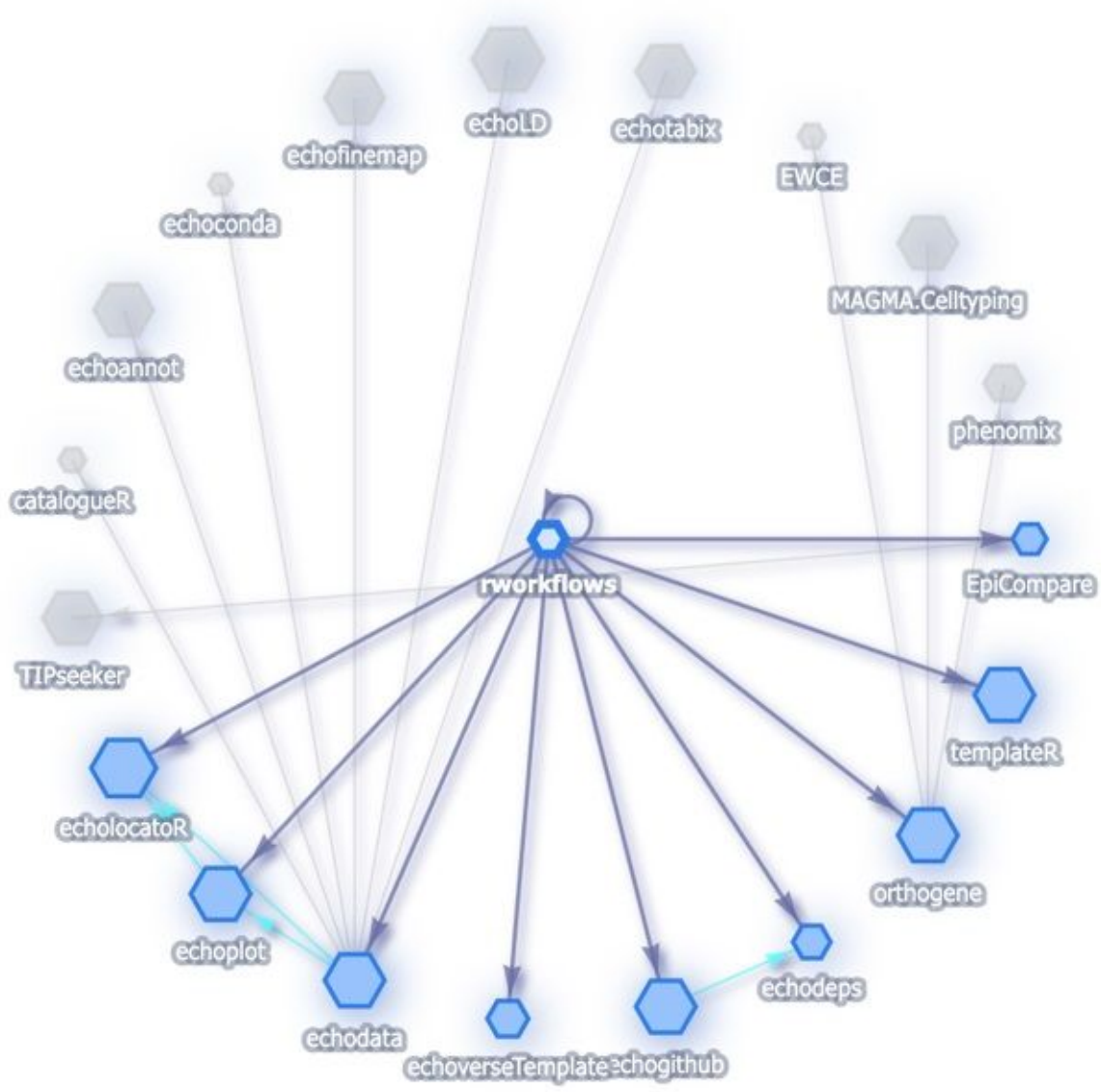

Figure 2

Reverse dependency graph.

A reverse dependency graph showing all R package GitHub repositories that currently utilise the *rworkflows* action at the time of this article's publication (blue nodes), or are dependent on a package that does (grey nodes). Each graph node size is scaled to the number of times that package has been downloaded. An interactive, continuously updated version of this graph is also available online (see **Data availability** section).

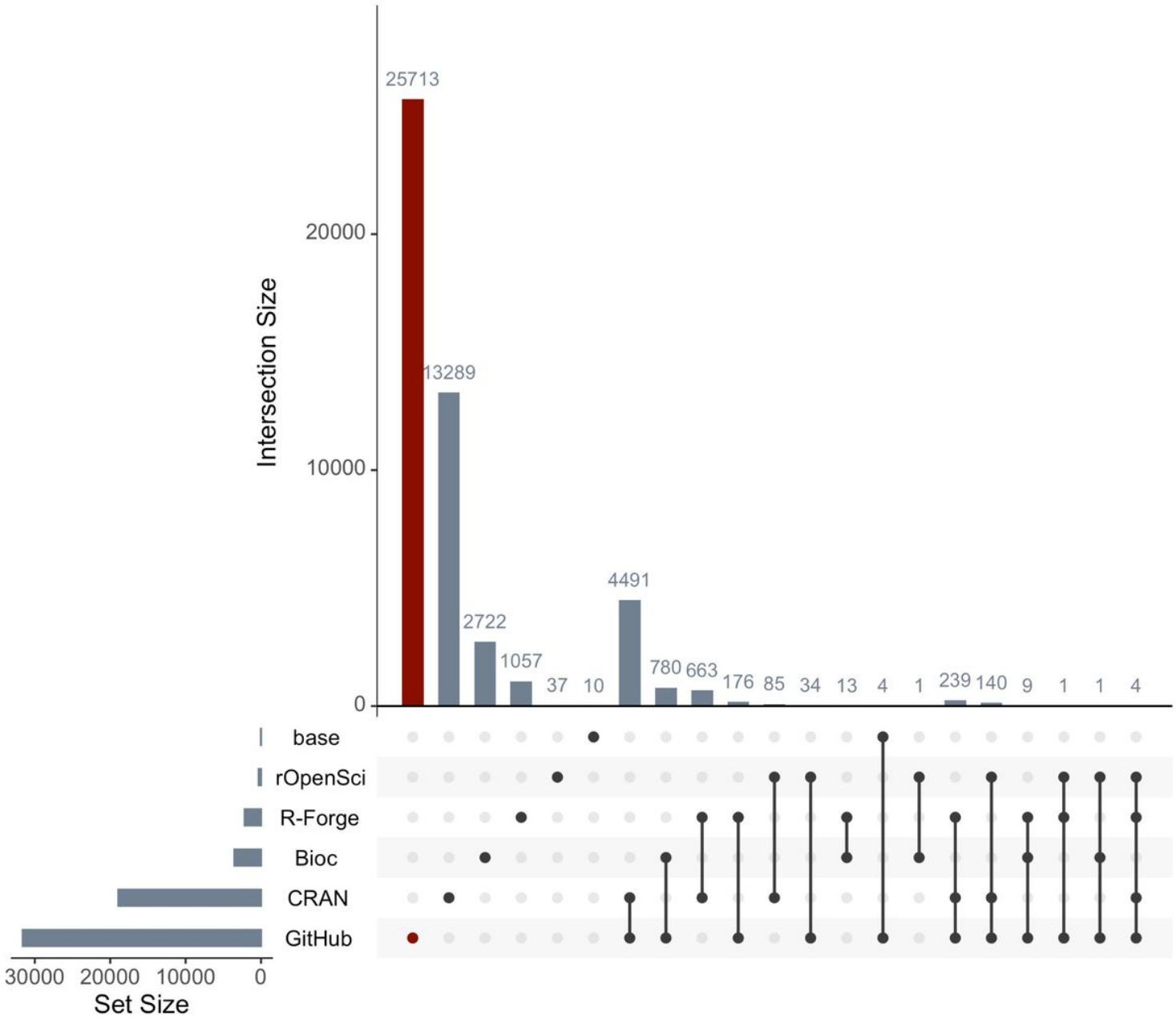

Figure 3

Repositories through which R packages are distributed.

Upset plot of how R packages are distributed through base R, dedicated R packages repositories (CRAN, Bioc, rOpenSci, R-Forge), or code repositories (GitHub). Rows indicate the total number of R packages available through a given distributor. Columns with single dots indicate the number of R packages that are exclusively available through one repository. Columns with multiple dots indicate the number of R packages available via two or more repositories. The number of R packages exclusively distributed through GitHub is highlighted in red.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- glossary.xlsx
- table1.xlsx