The Journal of Open Source Software

osfr: An R Interface to the Open Science Framework

Aaron R. Wolen[1], **Chris H.J. Hartgerink**[2], **Ryan Hafen**[3], **Brian G. Richards**[4], **Courtney K. Soderberg**[5], **and Timothy P. York**[6]

1 Transplant Research Institute, Department of Surgery, University of Tennessee Health Science Center **2** Liberate Science GmbH **3** Department of Statistics, Purdue University **4** Merkle Group Inc. **5** Center for Open Science **6** Data Science Lab, Department of Human and Molecular Genetics, Virginia Commonwealth University

DOI: 10.21105/joss.02071

Software

- Review
- Repository
- Archive

Editor: Kristen Thyng

Reviewers:

- @kthyng

Submitted: 21 January 2020
Published: 06 February 2020

Background

Reproducible research requires effective project management workflows that promote consistency and facilitate transparency. Hallmarks of effective workflows include strategies for tracking the provenance of results, recording intermediate changes, conveniently documenting procedures, and working with collaborators without duplicating effort (Sandve, Nekrutenko, Taylor, & Hovig, 2013). For technically skilled researchers, the combination of version control software (VCS) such as git and cloud-based project repositories (e.g., GitHub and GitLab) enable highly effective workflows (Ram, 2013) that facilitate automation and computational reproducibility. However, these tools have a steep learning curve, especially for researchers whose training is far removed from software development. Alternatively, the Open Science Framework (OSF) offers much of the same functionality through an intuitive point-and-click web-based interface, significantly lowering the barrier to adopting best practices for researchers of all skill levels, or research groups composed of individuals with different levels of computational expertise (Sullivan, DeHaven, & Mellor, 2019). Yet the increase in accessibility comes at the cost of limiting opportunities for automation. `osfr` fills this gap for R users by allowing them to programmatically interact with OSF through a suite of functions for managing their projects and files.

Functional Overview

On OSF, individual repositories are referred to as *projects* and serve as the top-level unit of content organization. New projects can be created with osfr using `osf_create_project()`, which allows you to specify the project's title, provide a description, and indicate whether it should be private (the default) or publicly accessible. Every OSF project includes a cloud-based storage bucket where files can be stored and organized into directories. You can use `osf_mkdir()` to add directories and `osf_upload()` to populate the project with files. osfr supports recursively uploading nested directories, making it possible to easily mirror the contents of a local project on OSF. For existing projects and project files, osfr provides functions for most common file operations such as copying (`osf_cp()`), moving (`osf_mv()`), deleting (`osf_rm()`), and downloading (`osf_download()`).

A key organizational feature of OSF is the ability to augment a project's structure with subprojects, which are referred to as *components* on OSF, and can be added with the `osf_create_component()` function. Like top-level projects, every component is assigned a unique URL upon creation and contains its own cloud-based storage bucket, activity log, wiki, and user permissions. This allows users to create arbitrarily nested projects that can easily scale to meet the needs of even large, multi-institutional collaborations (see the Cancer Biology

Reproducibility Project for a great example). Whatever the scale of your work, adopting a consistent structure across projects creates predictable expectations, facilitates understanding for you and your collaborators (Wilson et al., 2017), and makes it easier to stay organized as a project inevitably grows in complexity over time. Maintaining a consistent structure can be a challenge, especially if implemented in an *ad hoc* process, but osfr enables you to codify your preferred organizational structure of components and directories in a simple script that can be run at the outset of every new project.

Implementation and Design

osfr is built on the OSF public REST API, available at https://developer.osf.io, and uses rOpenSci's HTTP client, crul (Chamberlain, 2019), for API Communication. In order to provide an interface that feels natural to R users, items retrieved from the OSF are represented as `data.frame`-like objects called `osf_tbls`. The `osf_tbl` class is built on top of the tibble package (Müller & Wickham, 2019) and, like googledrive's dribble class (D'Agostino McGowan & Bryan, 2019), uses a list-column to encapsulate JSON responses from the API. These deeply nested structures are rarely of interest to the end user but are essential for the package's internal methods. The vast majority of osfr functions return `osf_tbls` as output and expect them as input, so that method chaining is possible using magrittr's pipe operator (Bache & Wickham, 2014).

Exported osfr functions all start with the prefix, `osf_`, following the `<prefix>_<verb>` naming convention used in packages like stringr (Wickham, 2019), which facilitates auto-completion in supported IDEs (like RStudio) and avoids namespace clashes with other packages that perform similar file-based operations. Where possible, we adopt the names of common Unix utilities that perform analogous tasks (e.g., `osf_cp()`, `osf_mkdir()`).

Summary

`osfr` provides an idiomatic R interface to OSF (Open Science Framework, https://www.osf.io), a free and open source web application that is part open-access repository and part collaborative project management tool.

References

Bache, S. M., & Wickham, H. (2014). *Magrittr: A forward-pipe operator for R*. Retrieved from https://CRAN.R-project.org/package=magrittr

Chamberlain, S. (2019). *Crul: HTTP client*. Retrieved from https://CRAN.R-project.org/package=crul

D'Agostino McGowan, L., & Bryan, J. (2019). *Googledrive: An interface to google drive*. Retrieved from https://cran.r-project.org/package=googledrive

Müller, K., & Wickham, H. (2019). *Tibble: Simple data frames*. Retrieved from https://CRAN.R-project.org/package=tibble

Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. *Source code for biology and medicine*, *8*(1), 7. doi:10.1186/1751-0473-8-7

Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). Ten simple rules for reproducible computational research. *PLOS Computational Biology*, *9*(10), 1–4. doi:10.1371/journal.pcbi.1003285

Sullivan, I., DeHaven, A., & Mellor, D. (2019). Open and reproducible research on open science framework. *Current protocols*, *18*(1), e32. doi:10.1002/cpet.32

Wickham, H. (2019). *Stringr: Simple, consistent wrappers for common string operations*. Retrieved from https://cran.r-project.org/package=stringr

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, *13*(6), 1–20. doi:10.1371/journal.pcbi.1005510