SOFTWARE TOOL ARTICLE

Creating and sharing reproducible research code the workflowr way [version 1; peer review: 3 approved]

John D. Blischak[1], Peter Carbonetto[1,2], Matthew Stephens[1,3]

[1]Department of Human Genetics, University of Chicago, Chicago, IL, 60637, USA
[2]Research Computing Center, University of Chicago, Chicago, IL, 60637, USA
[3]Department of Statistics, University of Chicago, Chicago, IL, 60637, USA

v1 **First published:** 14 Oct 2019, **8**:1749 (https://doi.org/10.12688/f1000research.20843.1)
Latest published: 14 Oct 2019, **8**:1749 (https://doi.org/10.12688/f1000research.20843.1)

Abstract

Making scientific analyses reproducible, well documented, and easily shareable is crucial to maximizing their impact and ensuring that others can build on them. However, accomplishing these goals is not easy, requiring careful attention to organization, workflow, and familiarity with tools that are not a regular part of every scientist's toolbox. We have developed an R package, **workflowr**, to help all scientists, regardless of background, overcome these challenges. **Workflowr** aims to instill a particular "workflow" — a sequence of steps to be repeated and integrated into research practice — that helps make projects more reproducible and accessible.This workflow integrates four key elements: (1) version control (via **Git**); (2) literate programming (via R Markdown); (3) automatic checks and safeguards that improve code reproducibility; and (4) sharing code and results via a browsable website. These features exploit powerful existing tools, whose mastery would take considerable study. However, the **workflowr** interface is simple enough that novice users can quickly enjoy its many benefits. By simply following the **workflowr** "workflow", R users can create projects whose results, figures, and development history are easily accessible on a static website — thereby conveniently shareable with collaborators by sending them a URL — and accompanied by source code and reproducibility safeguards. The **workflowr** R package is open source and available on CRAN, with full documentation and source code available at https://github.com/jdblischak/workflowr.

Keywords

reproducibility, open science, workflow, R, interactive programming, literate programming, version control

This article is included in the RPackage gateway.

Open Peer Review

Reviewer Status ✓ ✓ ✓

	Invited Reviewers		
	1	**2**	**3**
version 1 published 14 Oct 2019	✓ report	✓ report	✓ report

1 **Peter F. Hickey**, Walter and Eliza Hall Institute of Medical Research, Parkville, Australia

2 **Przemysław Biecek**, Warsaw University of Technology, Warsaw, Poland

3 **Peter Baker**, University of Queensland, Herston, Australia

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding author: John D. Blischak (jdblischak@gmail.com)

Author roles: Blischak JD: Conceptualization, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Carbonetto P**: Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Stephens M**: Conceptualization, Funding Acquisition, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This work was supported by the Gordon and Betty Moore Foundation [4559].
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Blischak JD, Carbonetto P and Stephens M. **Creating and sharing reproducible research code the workflowr way [version 1; peer review: 3 approved]** F1000Research 2019, **8**:1749 (https://doi.org/10.12688/f1000research.20843.1)

First published: 14 Oct 2019, **8**:1749 (https://doi.org/10.12688/f1000research.20843.1)

Introduction

A central tenet of the scientific method is that results should be independently verifiable — and, ideally, extendable — by other researchers. As computational methods play an increasing role in many disciplines, key scientific results are often produced by computer code. Verifying and extending such results requires that the code be "reproducible"; that is, it can be accessed and run, with outputs that can be corroborated against published results[1–9]. Unfortunately, this ideal is not usually achieved in practice; most scientific articles do not come with code that can reproduce their results[10–13].

There are many barriers to sharing reproducible code and corresponding computational results[14]. One barrier is simply that keeping code and results sufficiently organized and documented is difficult — it is burdensome even for experienced programmers who are well-trained in relevant computational tools such as version control (discussed later), and even harder for the many domain scientists who write code with little formal training in computing and informatics[15]. Further, modern interactive computer environments (e.g., R, Python), while greatly enhancing code development[16], also make it easier to create results that are irreducible. For example, it is all too easy to run interactive code without recording or controlling the seed of a pseudo-random number generator, or generate results in a "contaminated" environment that contains objects whose values are critical but unrecorded. Both these issues can lead to results that are difficult or impossible to reproduce. Finally, even when analysts produce code that is reproducible in principle, sharing it in a way that makes it easy for others to retrieve and use (e.g., via GitHub or Bitbucket) involves technologies that many scientists are not familiar with[13,17].

In light of this, there is a pressing need for easy-to-use tools to help analysts maintain reproducible code, document progress, and disseminate code and results to collaborators and to the scientific community. We have developed an open source R[18] package, **workflowr**, to address this need. The **workflowr** package aims to instill a particular "workflow" — a sequence of steps to be repeated and integrated into research practice — that helps make projects more reproducible and accessible. To achieve this, **workflowr** integrates four key features that facilitate reproducible code development: (1) version control[19,20]; (2) literate programming[21]; (3) automatic checks and safeguards that improve code reproducibility; and (4) sharing code and results via a browsable website. These features exploit powerful existing tools, whose mastery would take considerable study. However, the **workflowr** interface is designed to be simple so that learning it does not become another barrier in itself and novice users can quickly enjoy its many benefits. By simply following the **workflowr** "workflow", R users can create projects whose results and figures are easily accessible on a static website — thereby conveniently shareable with collaborators by sending them a URL — and accompanied by source code and reproducibility safeguards. The Web-based interface, updated with version control, also makes it easy to navigate through different parts of the project and browse the project history, including previous versions of figures and results, and the code used to produce them. By using **workflowr**, all this can be achieved with minimal experience in version control systems and Web technologies.

The **workflowr** package builds on several software technologies and R packages, without which this work would have been impossible. **Workflowr** builds on the invaluable R Markdown literate programming system implemented in **knitr**[22,23] and **rmarkdown**[21,24], which in turn build on **pandoc**, the "Markdown" markup language, and various Web technologies such as Cascading Style Sheets and Bootstrap[25]. Several popular R packages extend **knitr** and **rmarkdown** for specific aims such as writing blogs (**blogdown**[26]), monographs (**bookdown**[27]), and software documentation (**pkgdown**[28]). Analogously, **workflowr** extends **rmarkdown** with additional features such as the reproducibility safeguards, and adds integration with the version control system **Git**[19,20]. **Git** was designed to support large-scale, distributed software development, but in **workflowr** it serves a different purpose: to record, and provide access to, the development history of a project. **Workflowr** also uses another feature of **Git**, "remotes", to enable collaborative project development across multiple locations, and to help users create browsable projects via integration with popular online services such as GitHub Pages and GitLab Pages. These features are implemented using the R package **git2r**[29], which provides an interface to the **libgit2** C library. Finally, beyond extending the R programming language, **workflowr** is also integrated with the popular **RStudio** interactive development environment[30].

In addition to the tools upon which **workflowr** directly builds, there are many other related tools that directly or indirectly advance open and reproducible data analysis. A comprehensive review of such tools is beyond the scope of this article, but we note that many of these tools are complementary to **workflowr** in that they tackle aspects of reproducibility that **workflowr** currently leaves to the user, such as management and deployment of computational environments and dependencies (e.g., **conda**, **Homebrew**, **Singularity**, **Docker**, **Kubernetes**, **packrat**[31], **checkpoint**[32], **switchr**[33], **RSuite**[34]); development and management of computational pipelines (e.g., **GNU Make**, **Snakemake**[35], **drake**[36]); management and archiving of data objects (e.g., **archivist**[37], Dryad[38], Zenodo); and distribution of open source software (e.g., CRAN, Bioconductor[39], Bioconda[40]). Most of these tools or services could be used in combination with **workflowr**. There are additional, ambitious efforts to develop cloud-based services that come with many computational reproducibility features (e.g., Code Ocean, Binder, Gigantum, The Whole Tale). Many of these platforms manage individual projects as **Git** repositories, so **workflowr** could, in principle, be installed and used on these platforms, possibly to enhance their existing features. Other R packages with utilities to facilitate reproducibility that could complement **workflowr** include **ProjectTemplate**[41], **rrtools**[42], and **usethis**[43], as well as many of the R packages listed in the "Reproducible Research" CRAN Task View.

Of the available software tools facilitating reproducible research, perhaps the closest in scope to **workflowr** are the R package **adapr**[44] and the Python-based toolkit **Sumatra**[45]. Like

workflowr, both **adapr** and **Sumatra** use version control to maintain a project development history. Unlike **workflowr**, both place considerable emphasis on managing and documenting dependencies (software and data), whereas **workflowr** only records this information. In contrast, **workflowr** places more emphasis on literate programming — the publishing of text and code in a readable form — and more closely integrates other features such as tracking project development history via **Git** with literate programming.

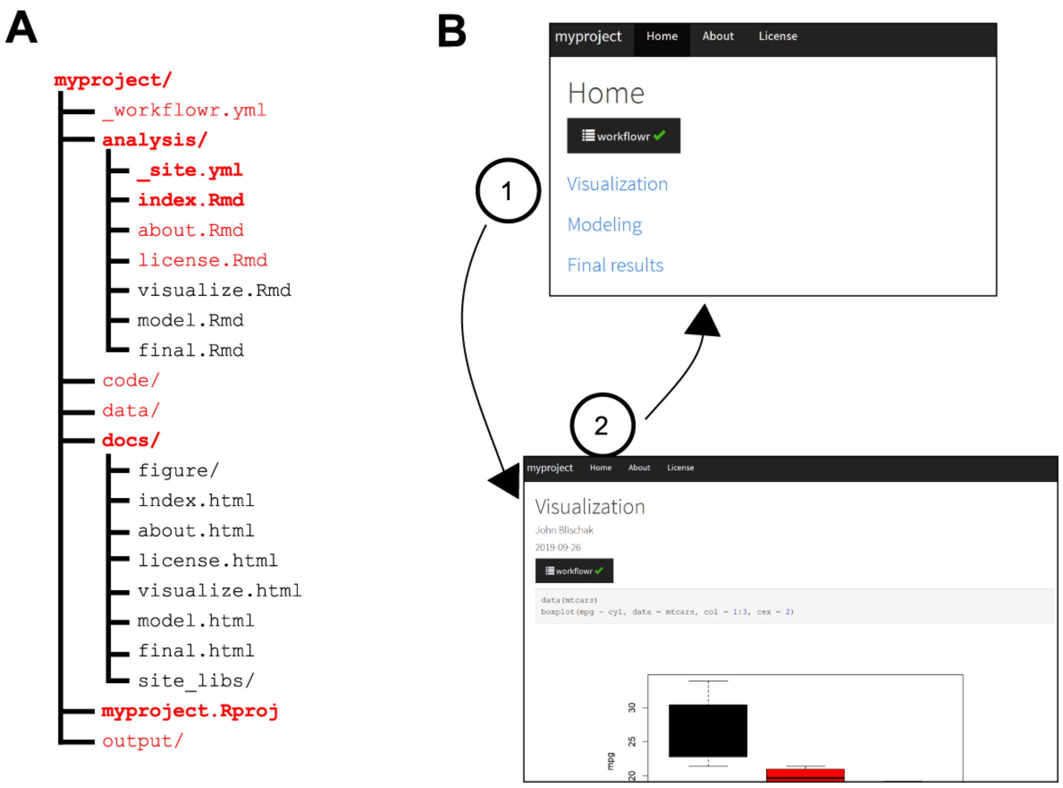
The **workflowr** R package is available from CRAN and GitHub, and is distributed under the flexible open source MIT license (see *Software availability*). The R package and its dependencies are straightforward to install while being highly customizable for more dedicated users. Extensive documentation, tutorials, and user support can be found at the GitHub site. In the remainder of this article, we describe the **workflowr** interface, explain its design, and give examples illustrating how **workflowr** is used in practice.

Operation

In this section, we give an overview of **workflowr**'s main features from a user's perspective. For step-by-step instructions on starting a **workflowr** project, see the "Getting started with **workflowr**" vignette.

For basic usage, only five functions are needed (summarized here, and described in more detail later):

- `wflow_start()` initializes a new project, including the template directory structure (Figure 1A);
- `wflow_build()` renders webpages from R Markdown (Rmd) analysis files, with reproducibility safeguards in place;
- `wflow_publish()` renders the webpages and updates the project development history—it commits the code, calls `wflow_build()`, then commits the webpages;



```
data(mtcars)
boxplot(mpg ~ cyl, data = mtcars, col = 1:3, cex = 2)
```

Figure 1. The workflowr package helps organize project files and results. A) The function `wflow_start()` populates a project directory with all the files and subdirectories (shown in red) needed to begin a **workflowr** project. This default directory structure encourages users to organize their files as the project progresses—as the project develops, additional Rmd files may be organized in the "analyses" folder. This is only a suggested structure; users can change the names of most files and directories. Required files are shown in boldface. **B**) All results are organized into a website (all HTML files generated by **workflowr** are automatically stored in `docs/`). The use of hyperlinks allows for efficient access to the results. The screenshots above illustrate how a **workflowr** website can be navigated. Clicking a hyperlink in the main page, `index.html`, (1) navigates the browser to a webpage containing some results, `visualize.html`; clicking on the "Home" hyperlink (2) in the navigation bar brings the browser back to the main page. For larger projects, the navigation bar can be used to quickly access different sections of a project.

- `wflow_status()` reports the status of the project files; and
- `wflow_git_push()` uploads the results from the user's local repository to a website hosting service.

The primary output of **workflowr** is a project website for browsing the results generated by the Rmd analysis files (Figure 1B). The use of websites to organize information is, of course, now widespread. Nonetheless, we believe they are under-utilized for organizing the results of scientific projects. In particular, hypertext provides an ideal way to connect different analyses that have been performed, and to provide easy access to relevant external data (e.g., related work or helpful background information); see Figure 1B and *Use cases* below.

Organizing the project: `wflow_start()`

The function `wflow_start()` facilitates project organization by populating a directory with suggested subdirectories, scripts, and configuration files for a data analysis project (Figure 1A). The subdirectories created by default are `analysis/`, where the Rmd analysis files are stored; `docs/`, which stores the website HTML files; `code/`, which is intended for longer-running scripts, compiled code (e.g., C++) and other source code supporting the data analyses; `data/`, for storing raw data files; and `output/`, for saving processed data files and other outputs generated by the scripts and analyses. This setup is flexible and configurable; only two of the directories, `analysis/` and `docs/`, are required, and both can be renamed later.

In addition to creating a default file structure for a data analysis project, `wflow_start()` also initializes the project development history: it creates a **Git** repository, and commits the files and directories to this repository. This is all done behind the scenes so no familiarity with **Git** is needed. We give more details about the **Git** repository in the *Implementation* section below.

In some cases, a user will have an existing project (with files that may or may not be tracked by **Git**), and would like to incorporate **workflowr** into the project — `wflow_start()` also easily accommodates this scenario, with additional arguments to control how the **workflowr** files are added to the existing project. See the package vignette, "Migrating an existing project to use **workflowr**," for more details; it can be accessed by running `vignette("wflow-03-migrating")` after loading the **workflowr** package in R.

Finally, `wflow_start()` changes R's working directory to the root of the project directory. Although this is a simple step, it is important for correctly resolving file paths. Forgetting to change the working directory is a very common source of errors in data analyses.

Generating results reproducibly: `wflow_build()`

In a **workflowr** project, analyses are performed using the R Markdown literate programming system[21]. The user develops their R code inside Rmd files in the `analysis/` directory, then calls `wflow_build()`, which runs the code and renders the results as HTML files in the `docs/` directory. The `wflow_build()` function extends the `render_site()` command from the **rmarkdown** package with several reproducibility safeguards:

1. It creates a clean R session for executing the code. This is critical for reproducibility—results should not depend on the current state of the user's R environment, and all objects necessary to run the code should be defined in the code or loaded by packages.

2. It automatically sets the working directory in a consistent manner (the exact setting is controlled by a configuration file; see *Implementation* below). This prevents one of the most common failures to reproduce in R—not setting the working directory before running the R script, resulting in incorrectly resolved relative file paths.

3. It sets a seed for the pseudorandom number generator before executing the code. This ensures that analyses that use random numbers always return the same result.

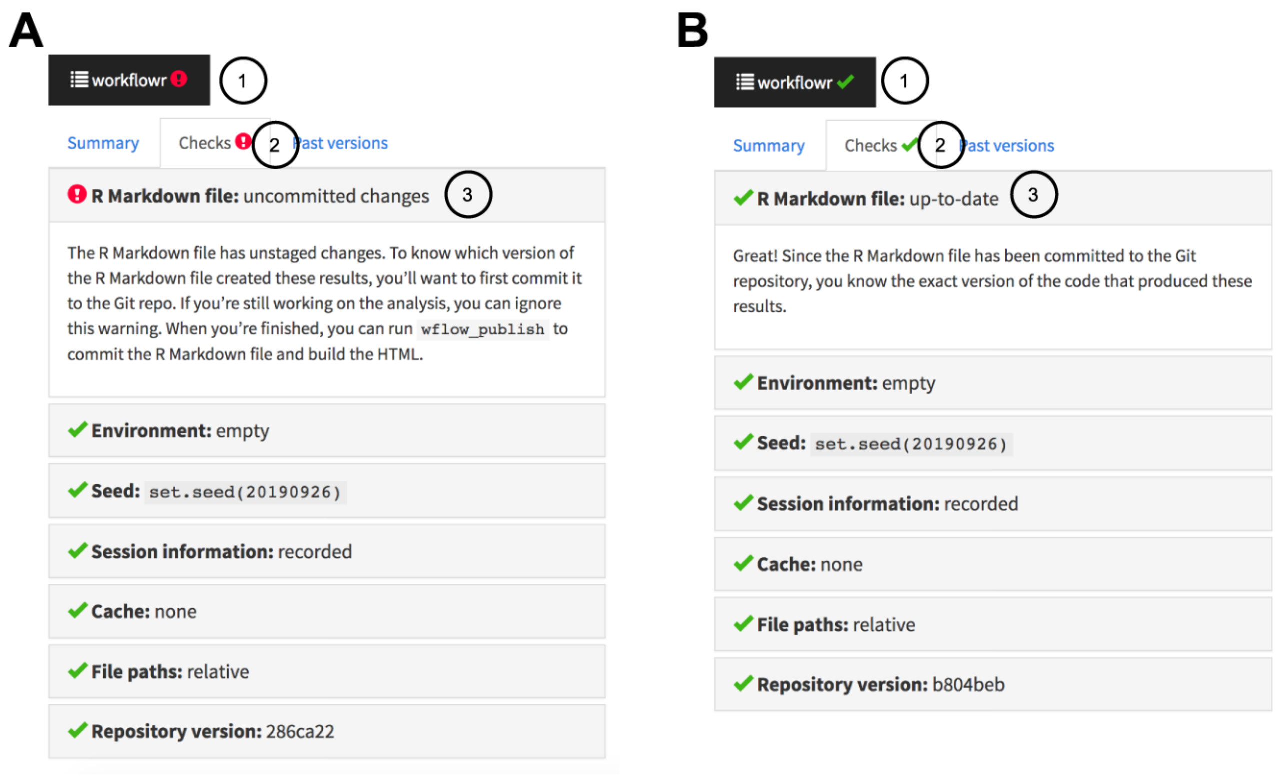
4. It records information about the computing environment, including the operating system, the version of R used, and the packages that were used to produce the results.

Finally, `wflow_build()` summarizes the results of these reproducibility safeguards in a report at the top of the webpage, along with additional "reproducibility checks", which alert the user to potential reproducibility issues, such as changes that were not committed to the project development history, and the use of (non-reproducible) absolute file paths (Figure 2).

Keeping track of the project's development: `wflow_publish()`

As a project progresses, many versions of the results will be generated as results are scrutinized, analyses are revised, errors are corrected, and new data are considered. Keeping track of a project's evolution is important for documenting progress and retracing the development of the analyses. This is sometimes done without version control tools by copying code and results whenever an important change is made. This typically results in a large collection of files with names such as `results-v2-final_final.pdf` or `anova_analyses_before_adding_new_samples.R`. This approach is tedious and error-prone, and makes it difficult to communicate changes to collaborators.

The version control system, **Git**, provides a more systematic and reliable way to keep track of a project's development history. However, **Git** was designed to manage source code for large-scale software projects, and using it for scientific analyses brings some specific challenges. The relative complexity of **Git** provides a high barrier to entry, discouraging many researchers from adopting it for their projects. And **Git** is not ideally suited to data analysis projects where one wants to coordinate the tracking of source code, data, and the results


A

workflowr ❗ 1

Summary | Checks ❗ 2 | Past versions

❗ **R Markdown file:** uncommitted changes 3

The R Markdown file has unstaged changes. To know which version of the R Markdown file created these results, you'll want to first commit it to the Git repo. If you're still working on the analysis, you can ignore this warning. When you're finished, you can run `wflow_publish` to commit the R Markdown file and build the HTML.

✓ **Environment:** empty

✓ **Seed:** `set.seed(20190926)`

✓ **Session information:** recorded

✓ **Cache:** none

✓ **File paths:** relative

✓ **Repository version:** 286ca22

B

workflowr ✓ 1

Summary | Checks ✓ 2 | Past versions

✓ **R Markdown file:** up-to-date 3

Great! Since the R Markdown file has been committed to the Git repository, you know the exact version of the code that produced these results.

✓ **Environment:** empty

✓ **Seed:** `set.seed(20190926)`

✓ **Session information:** recorded

✓ **Cache:** none

✓ **File paths:** relative

✓ **Repository version:** b804beb

Figure 2. **The workflowr reproducibility report summarizes the reproducibility checks inside the results webpage.** (**A**) A button is added to the top of each webpage. Clicking on the button (1) reveals the full reproducibility report with multiple tabs. If any of the reproducibility checks have failed, a red warning symbol (!) is shown. Clicking on the "Checks" tab (2) summarizes the reproducibility checks, with icons next to each check indicating a pass or failure. Clicking on an individual item (3) reveals a more detailed description of the reproducibility check, with an explanation of why it passed or failed. In (**A**), the Rmd file contains changes that have not yet been committed, so one of the reproducibility checks has failed (uncommitted changes are acceptable during active development, but not acceptable when results are published). In this case, the recommendation is given to run `wflow_publish()` to fix the issue. (**B**) If all the **workflowr** reproducibility checks pass, the **workflowr** button shows a green check mark (✔), and clicking an individual item in the reproducibility report (3) gives more detail on the reproducibility check.

generated by the code and data. Using **Git** commands to identify the version of the code that was used to generate a result can be non-trivial.

The `wflow_publish()` function is designed to address these challenges: it takes the steps necessary to coordinate tracking of code and results, and reduces these steps to calling a single, easy-to-use function. The command performs three steps, detailed in Figure 3. These steps are designed to ensure that each new collection of results added to the project development history has been produced by a unique and identifiable version of an Rmd analysis file.

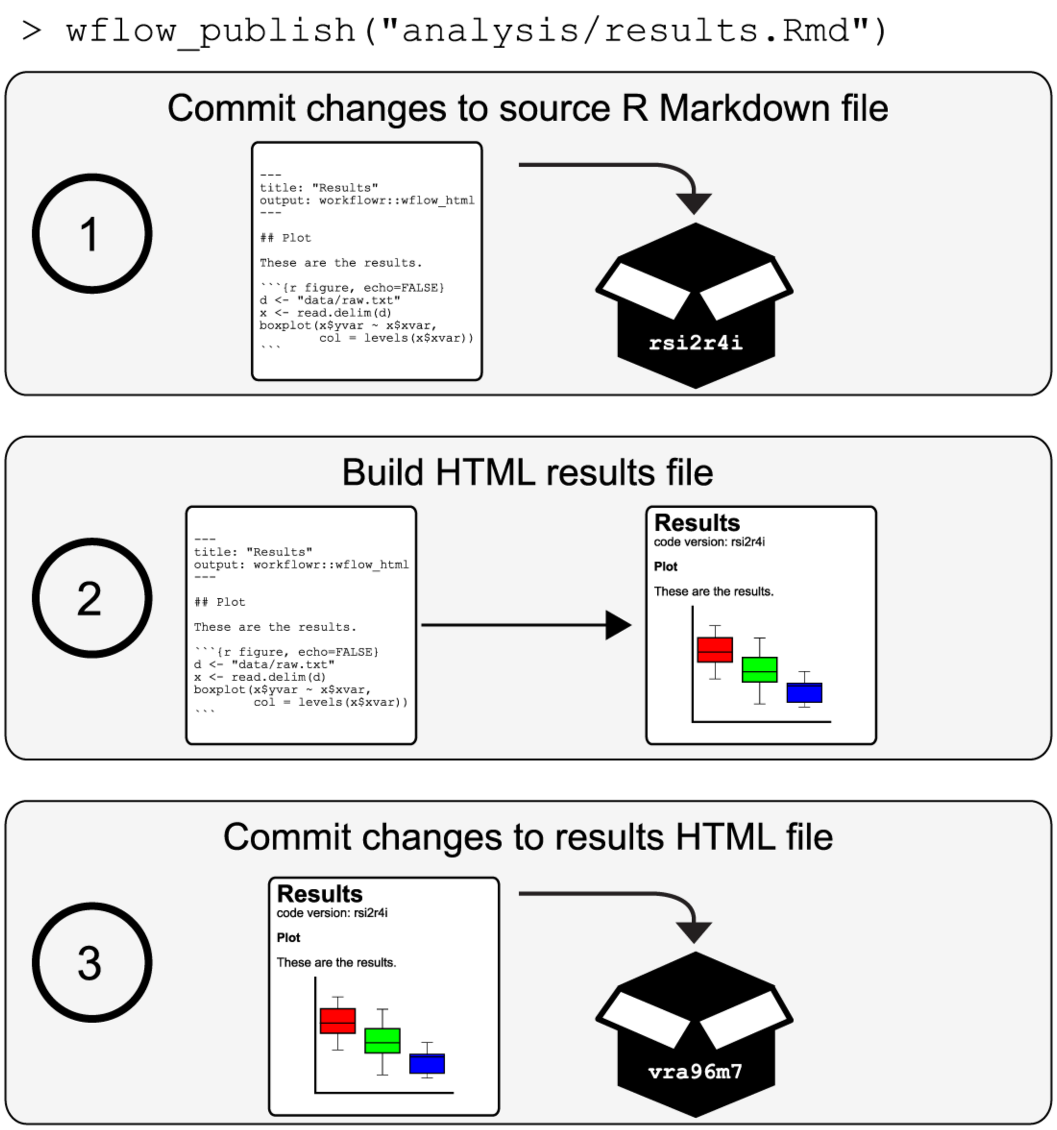
Even experienced **Git** users will benefit from using `wflow_publish()`. Besides the convenience of a single function, `wflow_publish()` ensures that:

1. Every commit to an (Rmd) analysis file is associated with a commit to the results file generated by that analysis file.
2. An analysis file is only published and committed if it runs successfully; on failure, `wflow_publish()` aborts, and neither code nor results are committed to the **Git** repository (R code that does not work can still be committed to a **workflowr** project via other methods, e.g., directly using **Git**, but it will not be associated with a committed results file).

Publishing an analysis is not necessarily final — after calling `wflow_publish()`, the analysis can be repeatedly updated and re-published using `wflow_publish()`. Each time `wflow_publish()` succeeds in committing a new version of the code and results, a link to previously published versions of the analysis are embedded in the webpage so that readers can easily access previous versions and compare with the latest results.

Checking in on the project's development: `wflow_status()`

As a **workflowr** project grows, it is important to be able to get an overview of the project's status and identify files that may need attention. This functionality is provided by the `wflow_status()` command, which gives the status of each Rmd



```
> wflow_publish("analysis/results.Rmd")
```

~~~
---
title: "Results"
output: workflowr::wflow_html
---

## Plot

These are the results.

```{r figure, echo=FALSE}
d <- "data/raw.txt"
x <- read.delim(d)
boxplot(x$yvar ~ x$xvar,
        col = levels(x$xvar))
```
~~~

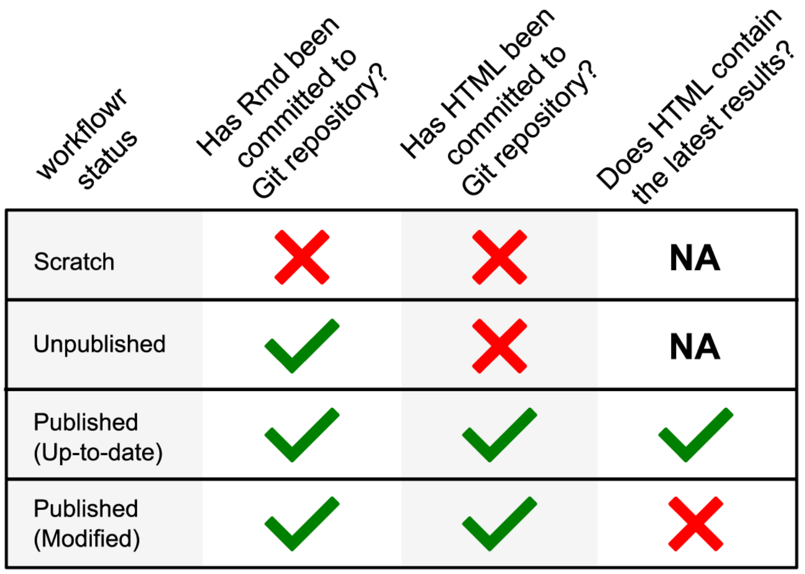
~~~
---
title: "Results"
output: workflowr::wflow_html
---

## Plot

These are the results.

```{r figure, echo=FALSE}
d <- "data/raw.txt"
x <- read.delim(d)
boxplot(x$yvar ~ x$xvar,
        col = levels(x$xvar))
```
~~~

Figure 3. The function `wflow_publish()` simplifies and coordinates tracking of the source code and results files in a Git repository. The function performs a three-step procedure to store the code and results in a project development history, and ensure that the results HTML file is always created from a unique and identifiable versioned Rmd analysis file. (1) The first step commits the changes to the Rmd analysis file. (2) The second step builds the results HTML file from the Rmd file. These two steps ensure that the results were generated from the committed version of the Rmd file. Furthermore, the unique version of the Git repository is inserted directly into the HTML file so that the source code used to generate the results is easily identified and accessed. If the code generates an error, the entire process is aborted and the previous commit made in the first step is undone. (3) The results HTML file, as well as any related figure files, are committed to the Git repository. Thus, the versioning of Rmd analysis files and corresponding HTML results files are coordinated whenever `wflow_publish()` is used.

file in the project — either "scratch", "unpublished", or "published", whose definitions are given in Figure 4. The "published" Rmd files, which are those that have been run through `wflow_publish()`, are further recorded as either "up-to-date" or "modified" depending on whether the Rmd file has been modified since `wflow_publish()` was run. The `wflow_status()` function highlights all Rmd files in the "scratch", "unpublished" or "modified" states, and suggests suitable next steps.

Sharing code and results: `wflow_git_push()`

The version-controlled website created by **workflowr** is self-contained, so it can be hosted by most Web servers with little effort. Once the website is available online, the code and results can be shared with collaborators and colleagues by providing them with the website's URL. Similarly, the **workflowr** repository can also serve as a companion resource for a manuscript by referencing the website URL in the paper.



workflowr status	Has Rmd been committed to Git repository?	Has HTML been committed to Git repository?	Does HTML contain the latest results?
Scratch	✗	✗	NA
Unpublished	✓	✗	NA
Published (Up-to-date)	✓	✓	✓
Published (Modified)	✓	✓	✗

Figure 4. The workflowr package is an R Markdown-aware version control system. The function `wflow_status()` assigns a state to each Rmd file in the **workflowr** project based on its status in the Git repository's working tree, and based on the Git status of the associated HTML results file.

Since a **workflowr** project is also a **Git** repository, the most convenient way to make the website available online is to use a **Git** hosting service. The **workflowr** package includes functions `wflow_use_github()` and `wflow_use_gitlab()` to simplify the setup process on two of the most widely used services, GitHub and GitLab. Once a user has created a **Git** repository on one of these online platforms, the project can be easily uploaded using `wflow_git_push()` (there is also a companion function `wflow_git_pull()`, which is used when multiple people are collaborating on a **workflowr** project, or when a project is being updated from multiple computers).

The results files in a **workflowr** website include links to past versions of analysis and figures, making it easy for collaborators to benefit from the versioning of analyses without knowing anything about **Git**. For example, if a collaborator wants to download a previous version of a figure generated several months ago, this can be done by navigating the links on the **workflowr** website.

Installation
The **workflowr** package is available on CRAN. It works with R versions 2.3.5 or later, and can be installed on any major platform that is supported by R (Linux, macOS, Windows). It is regularly tested on all major operating systems via several continuous integration services (AppVeyor, CircleCI, Travis CI). It is also regularly tested by CRAN using machines running Debian GNU/Linux, Fedora, macOS, Solaris, and Windows.

Because **workflowr** uses the **rmarkdown** package to build the HTML pages, it requires the document conversion software **pandoc** to be installed. The easiest way for R users to install **pandoc** is to install **RStudio**.

Installing **Git** is not required because the R package dependency **git2r** includes **libgit2**, a minimal **Git** implementation (nonetheless, installing **Git** may be useful for occasional management of the **Git** repository outside regular **workflowr** usage).

Customization
Workflowr projects are highly customizable. For example, the look of the webpages can be customized, via options provided by the **rmarkdown** package, by editing the `analysis/_site.yml` configuration file. Additional settings specific to **workflowr**, such as setting the seed for the pseudorandom number generator, or setting the working directory for the Rmd files, can be controlled in the `_workflowr.yml` file.

Implementation
Here we give an overview of the **workflowr** package implementation. All **workflowr** commands can be invoked from R (or **RStudio**) so long as the working directory in R is set to the directory containing a **workflowr** project, or any subdirectory of a **workflowr** project (this is similar to how **Git** commands are invoked). To determine the root directory of a **workflowr** project from a subdirectory, whenever a command is called from the R console, **workflowr** uses the **rprojroot**[46] R package to search for the **RStudio** project file stored at the root of the project (the **RStudio** project file is a required file, so if this file is deleted, the **workflowr** commands will not work).

Organizing the project: `wflow_start()`
The function `wflow_start()` populates the project directory using predefined template files (see Figure 1). It uses the

glue[47] R package to insert relevant variables, e.g., the name of the project, directly into the newly created files. When `wflow_start()` is called with `git = TRUE` (which is the default), a **Git** repository is created in the project directory, and all newly created or modified files are committed to the repository. If the user has never previously created a **Git** repository on their computer, they may need to first call `wflow_git_config()` to configure **Git**.

Generating results reproducibly: `wflow_build()`

The `wflow_build()` function generates a responsive website from a collection of Rmd files. Both `wflow_build()` and `wflow_publish()` support file patterns, also known as "wildcard expansion"; for example, `wflow_build("analysis/*.Rmd")` will generate webpages for all the Rmd files in the `analysis/` directory.

The `wflow_build()` function extends the `render_site()` function from the **rmarkdown** package. The `render_site()` function in turn builds on the Bootstrap framework to create a responsive website with a navigation bar. This rendering step includes downloading and linking to the required CSS and JavaScript files. Many website settings, such as the labels and URLs included in the navigation bar, can be adjusted in the `analysis/_site.yml` configuration file (these options can also be set individually inside the Rmd files, which will override the default options set in `analysis/_site.yml`). Like other R packages that extend **rmarkdown** (e.g., **bookdown**), **workflowr** provides a custom site generator in the function `wflow_site()`, which alters the website generation process. For example, one change to this process is that the generated website files (the HTML, CSS, JavaScript and figures) are moved instead of copied from `analysis/` to `docs/`. This reduces unnecessary duplication of files. Most of **workflowr**'s key features, including the reproducibility report, are implemented in `wflow_html()`, which we describe next.

In the **rmarkdown** package, the rendering of individual webpages from Rmd files is controlled by a separate function, `html_document()`. The **workflowr** package provides an analogous function, `wflow_html()`. This function also extends `html_document()`, so all features implemented in **rmarkdown** (e.g., code chunk folding, generating a table of contents from the section headings) are inherited by `wflow_html()`.

Most of the **workflowr** content is added as a preprocessing step prior to executing the R code in the Rmd file. To achieve this, `wflow_html()` copies the original Rmd file to a temporary directory, incorporates the additional content, then executes the code. The content embedded into the Rmd file includes a code chunk that calls `set.seed()`, a code chunk toward the end of the file that calls `sessionInfo()`, and inline HTML tags for elements such as the reproducibility report (Figure 2) and links to previous versions of figures. There is also a brief postprocessing step to incorporate additional HTML, CSS, and JavaScript elements needed to display the **workflowr** elements added in the preprocessing step. This postprocessing is done when **pandoc** converts the generated markdown to the final webpage.

The process for embedding links to past versions of files — that is, files added to previous commits in a **Git** repository — requires some additional explanation. Links to past versions are included only if the user has set up a remote repository hosted by either GitHub or GitLab. Clicking on a link to a past version of an Rmd file (or figure file) in a Web browser will load a webpage displaying the R Markdown source code (or figure file) as it is saved in the given commit. For past versions of the webpages, we use an independent service raw.githack.com, which displays the HTML file in the browser like any other webpage (this is because GitHub and GitLab only show the raw HTML code). These links will point to valid webpages only after the remote repository (on GitHub or GitLab) is updated, e.g., using `wflow_git_push()`. In the current implementation, when an Rmd file (and its corresponding HTML file) is renamed, the webpage does not include links to past versions prior to renaming. So renaming files will limit the ability to browse the project development history.

The `wflow_html()` function allows for considerable customization of the **workflowr** reproducibility report, and other features. The settings in the `analysis/_site.yml` configuration file are passed to function `html_document()` in the **rmarkdown** package, whereas the settings in `_workflowr.yml` are read by `wflow_html()`; see `help(wflow_html)` for a full details on all **workflowr** settings that can be customized in this file. For example, the default function used to record the session information at the bottom of each webpage, `sessionInfo()`, can be overridden by adding the YAML field `sessioninfo` (e.g., the function from the **devtools**[48] package could be used instead by setting `sessioninfo: devtools::session_info()`).

To execute the code, `wflow_build()` first creates a new R session to execute the code. This is implemented using the R package **callr**[49].

By default, the **rmarkdown** package renders an Rmd file in the directory where the Rmd file is stored; that is, the R working directory is automatically changed to the directory containing the target Rmd file. By default, `wflow_html()` overrides the behaviour, and instead executes the R code with respect to the root project directory. This default is intended to improve reproducibility by resolving file paths from a consistent reference point. This execution directory can be controlled by the `knit_root_dir` option, which is set in the `_workflowr.yml` configuration file. By default, new projects execute the R Markdown code chunks in the root directory. If this setting is not configured, **workflowr** reverts to the **rmarkdown** default. It is also possible to have a different `knit_root_dir` setting for different files, but this is generally not recommended as it will make the code more difficult to follow.

Keeping track of the project's development: `wflow_publish()`

One of the steps in `wflow_publish()`, as we have mentioned, is a call to `wflow_build()`. It also runs **Git** commands to commit the source code and rendered HTML files

(Figure 3). These **Git** commands are executed behind the scenes. We have also implemented many checks and extensive error handling to make sure that the **Git** repository and R environment are in an acceptable state for committing the results. When an issue arises, `wflow_publish()` attempts to detect the issue as early as possible, then it reverts the **Git** repository to the initial state and, when possible, suggests how to fix the issue. For example, `wflow_publish()` will stop if any of the files contain conflicts from a previous merge using **Git**.

Checking in on the project's development: `wflow_status()`

The `wflow_status()` function checks the status of each Rmd file in the project by comparing the state of the file in the **Git** repository's working tree against the **Git** status of the corresponding HTML file. In **Git** terminology, a "scratch" Rmd file in a **workflowr** project is an uncommitted file in a **Git** repository; "unpublished" means that the Rmd file is committed to the **Git** repository but the corresponding HTML is not; a "published" Rmd file and its HTML file are both committed to the **Git** repository; and a "modified" Rmd file has changes — these changes can be unstaged, staged, or committed — that were made since the last time the corresponding HTML file was committed (Figure 4).

Using **git2r**, it is mostly straightforward to determine the status of each file. The only complicated step is determining whether published Rmd files have been modified. If all changes to an Rmd file have been committed to the **Git** history, an Rmd file is considered "modified" if it has modifying commits that are more recent than commits modifying the corresponding HTML file.

Sharing the code and results: `wflow_git_push()`

To use `wflow_git_push()`, the remote **Git** repository must first be configured. The user can configure the remotes manually using the `git remote` subcommand or using `wflow_git_remote()`. Alternatively, the **workflowr** package provides two functions, `wflow_use_github()` and `wflow_use_gitlab()`, that simplify the creation and configuration of remote repositories hosted on GitHub and GitLab. These two convenience functions also add a navigation bar link with the URL of the remote source code repository. The `wflow_use_gitlab()` function takes the additional step of activating the GitLab Pages by creating a file `.gitlab-ci.yml` with the proper configuration (GitHub Pages must be set up manually; there is currently no way to automate this via the GitHub API).

Use cases

Workflowr was officially released on CRAN in April 2018. As of September 2019, it has been downloaded from CRAN over 7,000 times, and it has been adopted by many researchers. The most common use cases are 1) documenting research development and including the project website in the accompanying academic paper, and 2) developing reproducible course materials to share with students. Here we highlight some successful examples.

Repositories for research projects

Human dermal fibroblast clonality project

https://davismcc.github.io/fibroblast-clonality

A **workflowr** project accompanying a scientific paper on computational methods for decoding the clonal substructures of somatic tissues from DNA sequencing data[50]. The webpages describe how to reproduce the data processing and analysis, along with the outputs and plots.

Characterizing and inferring quantitative cell cycle phase in single-cell RNA-seq data analysis

https://github.com/jdblischak/fucci-seq

A **workflowr** project supporting a paper on measuring cell cycle phase and gene expression levels in human induced pluripotent stem cells[51]. The repository contains the processed data and the code implementing the analyses. The full results can be browsed on the website.

Flexible statistical methods for estimating and testing effects in genomic studies with multiple conditions

https://github.com/stephenslab/gtexresults

A **workflowr** project containing the code and data used to produce the results from the GTEx data set that were presented in Urbut *et al.*[52].

Investigations on "truncated adaptive shrinkage"

https://github.com/LSun/truncash

A **workflowr** project created by a Ph.D. student created to keep track of his investigations into controlling false discoveries in the presence of correlation and heteroskedastic noise. This repository illustrates the use of **workflowr** as a scientific notebook — the webpages contain written notes, mathematical equations, source code, and the outputs generated from running the code.

Repositories for courses

Stanford STATS 110

https://xiangzhu.github.io/stanford-stats110

A **workflowr** website for a statistics course taught at Stanford. The website includes working R examples, homework, the course syllabus, and other course materials.

Single-cell RNA-seq workshop

https://github.com/crazyhottommy/scRNA-seq-workshop-Fall-2019

A **workflowr** website for a workshop on analysis of single-cell RNA-seq data offered by the Harvard Faculty of Arts and Sciences Informatics group as part of a two-week long bioinformatics course. The R examples demonstrate how to use several bioinformatics packages such as **Seurat** and **msigdbr** to prepare and analyze single-cell RNA-seq data sets.

Introduction to GIS in R

`https://github.com/annakrystalli/intro-r-gis`

A **workflowr** website for a workshop given at the 2018 Evolutionary Biology Conference. The website includes working R demonstrations, setup instructions, and exercises.

Summary

Our main aim in developing **workflowr** is to lower barriers to open and reproducible code. **Workflowr** provides a core set of commands that can be easily integrated into research practice, and combined with other tools, to make projects more accessible and reproducible. The R package is straightforward to install, easy to learn, and highly customizable.

Since the first official release of **workflowr** (version 1.0.1, released in April 2018), the core functionality has remained intact, and we expect it to remain that way. The core features of **workflowr** have been carefully tested and revised, in large part thanks to feedback and issue reports from the user community. Our next aim is to implement several enhancements, including:

- Create a centralized **workflowr** project website to make it easier for researchers to share and discover **workflowr** projects.
- Provide additional functions to simplify website hosting on other popular platforms such as Netlify and Heroku.
- As **workflowr** projects grow, it becomes increasingly important to document not only the evolution of the code and results over time, but also how the results interrelate with one another. Therefore, we aim to implement syntax that allows file dependencies to be recorded in the Rmd files, and incorporate checking of dependencies as part of the **workflowr** reproducibility safeguards.

As **workflowr** has been used in a variety of settings, we have also uncovered some limitations. Here we report on some of the more common issues that have arisen.

One limitation is that **Git** — hence **workflowr** — is not well suited to tracking very large files. Therefore, large data files must be left out of the project development history, which reduces reproducibility. One possible workaround is to use **Git LFS** (Large File Storage) or related tools that allow large data files to be tracked and stored remotely inside a **Git** repository. This, however, requires considerable expertise to install and configure **Git LFS**, so it is not a satisfactory solution for some **workflowr** users. Also note that sensitive or secure data can be added to a **workflowr** project so long as the storage and access practices meet the data security requirements (**workflowr** has options to simplify creation and management of projects with security requirements).

Since **workflowr** builds on **Git**, users who already have experience with **Git** can use **Git** directly to manage their **workflowr** projects. This provides additional flexibility, but is not without risk; for example, **Git** commands such as `git reset` can be used to alter the project development history, and has the potential to break **workflowr**.

Finally, **workflowr** records information about the computing environment used to generate the results, but it does not provide any facilities for replicating the environment. This is an area with many recent software advances — there are many widely used tools for managing and deploying computational environments, from container technologies such as **Docker** to package managers such as Anaconda and **packrat**. We view these tools as being complementary to **workflowr**, and one future direction would be to develop easy-to-use functions that configure such tools for use in a **workflowr** project.

Data availability

All data underlying the results are available as part of the article and no additional source data are required.

Software availability

- Software available from: `https://cran.r-project.org/package=workflowr`
- Source code available from: `https://github.com/jdblischak/workflowr`
- Archived source code at time of publication: `https://doi.org/10.5281/zenodo.3241801`[53]
- License: MIT

Acknowledgments

We thank the **workflowr** contributors for helping improve the package. We are also grateful for the many **workflowr** users for testing the package and providing feedback—thanks especially to Lei Sun, Xiang Zhu, Wei Wang, and many other members of the Stephens lab, past and present. We also acknowledge the authors and contributors of the many great open source packages that the **workflowr** package builds on. R packages particularly critical to **workflowr**'s implementation are **git2r**, **knitr**, and **rmarkdown**.

References

1. Buckheit JB, Donoho DL: **WaveLab and reproducible research.** *Wavelets and Statistics.* 1995; **103**: 55–81.
Publisher Full Text

2. Easterbrook SM: **Open code for open science?** *Nat Geosci.* 2014; **7**(11): 779–781.
Publisher Full Text

3. Gentleman R, Lang TD: **Statistical analyses and reproducible research.** *J Comput Graph Stat.* 2007; **16**(1): 1–23.
Publisher Full Text

4. Ince CD, Hatton L, Graham-Cumming J: **The case for open computer programs.** *Nature.* 2012; **482**(7386): 485–488.
PubMed Abstract | Publisher Full Text

5. Lowndes JSS, Best BD, Scarborough C, *et al.*: **Our path to better science in less time using open data science tools.** *Nat Ecol Evol.* 2017; **1**(6): 160.
PubMed Abstract | Publisher Full Text

6. Morin A, Urban J, Adams PD, *et al.*: **Research priorities. Shining light into black boxes.** *Science.* 2012; **336**(6078): 159–160.
PubMed Abstract | Publisher Full Text | Free Full Text

7. Peng RD: **Reproducible research in computational science.** *Science.* 2011; **334**(6060): 1226–1227.
PubMed Abstract | Publisher Full Text | Free Full Text

8. Sandve GK, Nekrutenko A, Taylor J, *et al.*: **Ten simple rules for reproducible computational research.** *PLoS Comput Biol.* 2013; **9**(10): e1003285.
PubMed Abstract | Publisher Full Text | Free Full Text

9. Stodden V, McNutt M, Bailey DH, *et al.*: **Enhancing reproducibility for computational methods.** *Science.* 2016; **354**(6317): 1240–1241.
PubMed Abstract | Publisher Full Text

10. Ioannidis JP, Allison DB, Ball CA, *et al.*: **Repeatability of published microarray gene expression analyses.** *Nat Genet.* 2009; **41**(2): 149–155.
PubMed Abstract | Publisher Full Text

11. Ioannidis JP, Greenland S, Hlatky MA, *et al.*: **Increasing value and reducing waste in research design, conduct, and analysis.** *Lancet.* 2015; **383**(9912): 166–175.
PubMed Abstract | Publisher Full Text | Free Full Text

12. Merali Z: **Computational science: ...error.** *Nature.* 2010; **467**(7317): 775–777.
PubMed Abstract | Publisher Full Text

13. Stodden V, Seiler J, Ma Z: **An empirical analysis of journal policy effectiveness for computational reproducibility.** *Proc Natl Acad Sci U S A.* 2018; **115**(11): 2584–2589.
PubMed Abstract | Publisher Full Text | Free Full Text

14. Kitzes J, Turek D, Deniz F: **The practice of reproducible research: case studies and lessons from the data-intensive sciences.** Univ of California Press, 2017.
Reference Source

15. Wilson G, Aruliah DA, Brown CT, *et al.*: **Best practices for scientific computing.** *PLoS Biol.* 2014; **12**(1): e1001745.
PubMed Abstract | Publisher Full Text | Free Full Text

16. Findler RB, Clements J, Flanagan C, *et al.*: **DrScheme: a programming environment for Scheme.** *J Funct Program.* 2002; **12**(2): 159–182.
Publisher Full Text

17. Marwick B: **Computational reproducibility in archaeological research: basic principles and a case study of their implementation.** *J Archaeol Method Theory.* 2017; **24**(2): 424–450.
Publisher Full Text

18. R Core Team: **R: a language and environment for statistical computing.** R Foundation for Statistical Computing, Vienna, Austria, 2019.
Reference Source

19. Chacon S, Straub B: **Pro Git.** Springer, New York, NY, 2nd edition, 2014.
Publisher Full Text

20. Loeliger J, McCullough M: **Version control with Git.** O'Reilly Media, Sebastopol, CA. 2nd edition, 2012.
Reference Source

21. Xie Y, Allaire J, Grolemund G: **R Markdown: the definitive guide.** Chapman and Hall/CRC, New York, NY. 2018.
Reference Source

22. Xie Y: **knitr: a comprehensive tool for reproducible research in R.** In V. Stodden, F. Leisch, and R. D. Peng, editors, *Implementing Reproducible Computational Research.* Chapman and Hall/CRC, 2014.
Publisher Full Text

23. Xie Y: **knitr: a general-purpose package for dynamic report generation in R.** R package version 1.23. 2019.
Reference Source

24. Allaire J, Xie Y, McPherson J, *et al.*: **rmarkdown: dynamic documents for R.** R package version 1.13. 2019.
Reference Source

25. Spurlock J: **Bootstrap.** O'Reilly Media, Sebastopol, CA, 2013.
Reference Source

26. Xie Y, Hill AP, Thomas A: **blogdown: creating websites with R Markdown.** Chapman and Hall/CRC, Boca Raton, Florida, 2017.
Reference Source

27. Xie Y: **bookdown: authoring books and technical documents with R Markdown.** Chapman and Hall/CRC, Boca Raton, Florida, 2016.
Reference Source

28. Wickham H, Hesselberth J: **pkgdown: make static HTML documentation for a package.** R package version 1.4.1. 2019.
Reference Source

29. Widgren S, *et al.*: **git2r: provides access to Git repositories.** R package version 0.26.1. 2019.
Reference Source

30. R Studio Team: **RStudio: integrated development environment for R.** RStudio, Inc., Boston, MA. 2018.
Reference Source

31. Ushey K, McPherson J, Cheng J, *et al.*: **packrat: a dependency management system for projects and their R package dependencies.** R package version 0.5.0. 2018.
Reference Source

32. Ooi H: **checkpoint: install packages from snapshots on the checkpoint server for reproducibility.** R package version 0.4.7. 2019.
Reference Source

33. Becker G, Barr C, Gentleman R, *et al.*: **Enhancing reproducibility and collaboration via management of R package cohorts.** *J Stat Softw.* 2017; **82**(1): 1–17.
Publisher Full Text

34. Sokolowski W, Jakuczun W, Yakimechko Y, *et al.*: **RSuite: supports developing, building and deploying R solution.** R package version 0.37-253. 2019.
Reference Source

35. Köster J, Rahmann S: **Snakemake--a scalable bioinformatics workflow engine.** *Bioinformatics.* 2012; **28**(19): 2520–2522.
PubMed Abstract | Publisher Full Text

36. Landau WM: **The drake R package: a pipeline toolkit for reproducibility and high-performance computing.** *J Open Source Softw.* 2018; **3**(21): 550.
Publisher Full Text

37. Biecek P, Kosinski M: **archivist: an R package for managing, recording and restoring data analysis results.** *J Stat Softw.* 2017; **82**(11): 1–28.
Publisher Full Text

38. Vision T: **The dryad digital repository: published evolutionary data as part of the greater data ecosystem.** 2010.
Reference Source

39. Gentleman CR, Carey VJ, Bates DM, *et al.*: **Bioconductor: open software development for computational biology and bioinformatics.** *Genome Biol.* 2004; **5**(10): R80.
PubMed Abstract | Publisher Full Text | Free Full Text

40. Grüning B, Dale R, Sjödin A, *et al.*: **Bioconda: sustainable and comprehensive software distribution for the life sciences.** *Nat Methods.* 2018; **15**(7): 475–476.
PubMed Abstract | Publisher Full Text

41. White JM: **ProjectTemplate: automates the creation of new statistical analysis projects.** R package version 0.9.0. 2019.
Reference Source

42. Marwick B: **rrtools: creates a reproducible research compendium.** R package version 0.1.0. 2019.
Reference Source

43. Wickham H, Bryan J: **usethis: automate package and project setup.** R package version 1.5.1. 2019.
Reference Source

44. Gelfond J, Goros M, Hernandez B, *et al.*: **A system for an accountable data analysis process in R.** *R J.* 2018; **10**(1): 6–21.
PubMed Abstract | Publisher Full Text | Free Full Text

45. Davidson AP, Mattioni M, Samarkanov D, *et al.*: **Sumatra: a toolkit for reproducible resesearch.** In V. Stodden, F. Leisch, and R. D. Peng, editors, *Implementing Reproducible Computational Research.* Chapman and Hall/CRC. 2014.
Publisher Full Text

46. Müller K: **rprojroot: finding files in project subdirectories.** R package version 1.2. 2017.
Reference Source

47. Hester J: **glue: interpreted string literals.** R package version 1.3.1. 2019.
Reference Source

48. Wickham H, Hester J, Chang W: **devtools: tools to make developing R packages easier.** R package version 2.1.0. 2019.
Reference Source

49. Csárdi G, Chang W: **callr: call R from R**. R package version 3.3.2. 2019.
Reference Source

50. McCarthy DJ, Rostom R, Huang Y, *et al.*: **Cardelino: integrating whole exomes and single-cell transcriptomes to reveal phenotypic impact of somatic variants.** *bioRxiv.* 2018.
Publisher Full Text

51. Hsiao CJ, Tung P, Blischak JD, *et al.*: **Characterizing and inferring quantitative cell cycle phase in single-cell RNA-seq data analysis.** *bioRxiv.* 2019.
Publisher Full Text

52. Urbut SM, Wang G, Carbonetto P, *et al.*: **Flexible statistical methods for estimating and testing effects in genomic studies with multiple conditions.** *Nat Genet.* 2019; **51**(1): 187–195.
PubMed Abstract | Publisher Full Text | Free Full Text

53. Blischak J, Carbonetto P, Li J, *et al.*: **jdblischak/workflowr: workflowr 1.4.0.** 2019.
http://www.doi.org/10.5281/zenodo.3241801

Open Peer Review

Current Peer Review Status: ✓ ✓ ✓

Version 1

Reviewer Report 08 November 2019

https://doi.org/10.5256/f1000research.22923.r55117

✓ **Peter Baker**
School of Public Health, University of Queensland, Herston, Qld, Australia

The authors introduce an R package that provides an easy way to set up a workflow for data analysis using R and publish results to a web page.

The workflowr package seems to work well and appears to be widely used.

What is relatively novel about this package is that while most workflow tools I have seen in R concentrate on setting up a directory structure, syntax files and producing output and reports from analyses, this package adds both the ability to publish the results to web pages and also to set up a Github or Gitlab project repositories with minimal effort.

The article is clear and well written as are the associated vignettes. The underlying functions in the R are also well written but perhaps could employ more error checking and reporting (see below).

While I am unlikely to use this approach myself I think it is an excellent approach for new users since it

1. sets up a project skeleton with instructions,

2. encourages users to document their project and workflow right from the start, and

3. also the package provides quite a few helpful vignettes and guides for various scenarios that should be useful for those starting in the area.

However, I do have minor reservations with the approach outlined, including

- for new users, they must not only learn R but also R Markdown which adds an extra level of complexity;

- while it seems necessary in workflowr, users do not really need to use R Markdown files to produce well documented code (see https://rmarkdown.rstudio.com/articles_report_from_r_script.html) but R Markdown seems better suited to reports and articles;

- the workflow is very R Markdown-centric: experienced users may wish to employ R or other software directly or a build system like Make or drake. While this is relatively straight forward outside of the package, e.g. by adding these to the git repository outside of the package and using employing R directly to update intermediate results the article or documentation do not give any details;

- some functions do not appear to be particularly error-proof for new users, e.g. wflow_git_config will overwrite existing settings without checking or even providing a warning although this may conceivably change in future; and

- new users will undoubtedly run into git merge issues and the version I reviewed (1.4.0) did not seem to cater for such eventualities although this appears to have been addressed according to the change log in the latest version (1.5.0).

Minor comments are:

- I am not sure why only some of the software on page 3 is cited when presumably a reader may benefit from the author's recommendations appropriate to the data analysis workflow area rather than tracking down a general reference for themselves;

- while it is good that the potential pitfalls of using git directly or using git reset are addressed on page 11, it may also prove useful for readers to address limitations and potential pitfalls in more depth, perhaps not in the article but with reference to other material or vignettes. For example, users at all levels might benefit from knowing where to get help on merge conflicts, whether users with large data sets should consider databases rather than git or whether typical git workflows like branching would work with this package,

- it would be nice to see how this package compares to other alternatives like drake but admittedly the scope this article is more an introduction to the workflowr package.

In summary, this article is clear and well written. The package is an original contribution to the range of software addressing reproducibility and workflow in data analysis projects.

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Biostatistics, R, workflow of data analysis

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 04 November 2019

https://doi.org/10.5256/f1000research.22923.r55119

✓ **Przemysław Biecek**
Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland

I like the workflowr package and I like the paper. Nicely written.
Reproducibility is a big thing and workflowr lowers the entrance barrier for non technical users. This is a huge benefit.
The package has has already some visibility (judging based on GitHub stars) and is being adopted to various applications (based on examples presented in the paper).
I recommend to accept the paper.

Here are some comments that authors may consider:

- **A.** The point that I am missing the most is the comparison against the drake package. These two packages seems to be similar, maybe complementary. Can they be used together? It would be good to show pros and cons/similarities and differences.

- **B.** I like the workflowr package it is a useful tool. What I am missing is the methodology / description of a process / good practices of how the reproducible analysis should look like. This would be very useful for people that look for precise guidelines on how to integrate the workflowr with every day practice. Wet labs researchers are used to "protocols" that directly guide step by step what to do during the analysis. Maybe it would be possible to give such protocols for reproducible research with the workflowr package.

- Just to give an example, in the Model Development Process (https://arxiv.org/abs/1907.04461) article there is an overview of phases and tasks shared across model development. In which phases the workflowr or similar tools shall be used?

- **C.** Authors have mentioned blogdown and bookdown packages. I think that even a closer match to the reproducibility problem is the package modelDown (see https://joss.theoj.org/papers/10.21105/joss.01444 or GitHub http://github.com/ModelOriented/modelDown).[1]
The modelDown package takes predictive models and creates a HTML website with information

about session info, binary models, training/test data and model explanations. The website is created without any additional effort. ModelDown automates the most boring part of the modeling i.e. model documentation.

- **D.** When mentioning tools for archivisation of binary objects, it may be also useful to add the pins package recently developed by RStudio (https://cran.r-project.org/web/packages/pins/index.html). It is more limited than other mentioned packages (do not keep information about meta data) but quickly gains popularity.

- **E.** After the "Unfortunately, this ideal is not usually achieved in practice; most scientific articles do not come with code that can reproduce their results" maybe authors could share their thoughts why it is the case. It will be useful to list specific reasons why reproducibility fails. Is it primarily because we do not have proper software, or they software is too complex, or one needs to pay for the proper software, or researchers are not aware of the problem?

References
1. Romaszko K, Tatarynowicz M, Urbański M, Biecek P: modelDown: automated website generator with interpretable documentation for predictive machine learning models. *Journal of Open Source Software*. 2019; **4** (38). Publisher Full Text

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Partly

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Human-Oriented machine learning, Human-Centered Artificial Intelligence, Software engineering

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 31 October 2019

https://doi.org/10.5256/f1000research.22923.r55176

✓ **Peter F. Hickey**
Epigenetics and Development Division, Walter and Eliza Hall Institute of Medical Research, Parkville, Vic, Australia

The workflowr package, available from CRAN for 1.5 years, has already demonstrated itself to be a valuable contribution to improving the reproducibility of scientific analyses. This paper does a thorough job setting out the rationale, design, and implementation of the workflowr package. It is clear that the authors have spent considerable time thinking about some of the key challenges of this endeavour, learning about best practises, getting feedback from users, and implementing these using 4 key technologies/features:

1. Version control.
2. Literate programming.
3. Automatic checks and safeguards to improve code reproducibility.
4. Sharing code and results via a website.

The paper is clearly written and I am happy to approve the article in its current form.

Some minor comments, queries, and corrections are given below:

- Figure caption 1: '"analyses" folder' is '"analysis" folder' in the figure.
- p5: Re workflowr executing code in a clean session. My impression was that rendering Rmarkdown documents, at least when done by clicking the 'knit' button in RStudio, was already run in a separate process. But I may be mistaken and perhaps this is different from running `rmarkdown::render_site()`?
- Regarding published sites. Not all scientific analyses can be made public, particularly in the early stages. Some discussion of options available for private/protected hosting would be valuable.
- I'm a little wary of relying on raw.githack.com for hosting past versions of the webpages. For example, what if the service becomes unavailable? Does or could workflowr support other hosting services?
- p11: Is the idea of a 'centralized workflowr project website' like that of the homepage https://bookdown.org/ or for an organisation/user to share their personal workflowr projects?

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: Matthew Stephens and I are both members of the Genotype-Tissue Expression (GTEx) consortium. The consortiumcontains hundreds of scientists. As members of the consortium, we are co-authors on papers where the 'GTEx Consortium' has been listed as an author. That is, the GTEx consortium is listed as a co-author on papers I have authored and, separately, papers that Matthew has authored. We have not directly worked together via the consortium and, to the best of my knowledge, we are not both listed individually as authors on the same paper.

Reviewer Expertise: Statistics, bioinformatics, R programming

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.