WORCS: A Workflow for Open Reproducible Code in the Social Sciences and Beyond

Version: 0.1.2, date: 20 May, 2020

Caspar J. Van Lissa, Andreas M. Brandmaier, Loek Brinkman, Aaron Peikert, Barbara M.I. Vreede, & Anna-Lena Lamprecht

1 Utrecht University faculty of Social and Behavioral Sciences, department of Methodology & Statistics
2 Open Science Community Utrecht
3 Center for Lifespan Psychology, Max Planck Institute for Human Development, Berlin, Germany
4 Max Planck UCL Centre for Computational Psychiatry and Ageing Research, Berlin, Germany and London, UK
5 UMC Utrecht
6 Humboldt-Universitaet zu Berlin
7 University Library Utrecht
8 Utrecht University, Department of Information and Computing Sciences

Author Note
This is a preprint, not a peer-reviewed publication. Comments, contributions, and coauthors are welcome. Please contact the lead author. Acknowledgements: Jeroen Ooms (feedback, suggesting gert, and using Remotes field).

Correspondence concerning this article should be addressed to Caspar J. Van Lissa, Padualaan 14, 3584CH Utrecht, The Netherlands. E-mail: c.j.vanlissa@uu.nl
Abstract

The social sciences are amidst a paradigm shift towards open science. In part, this transition has been fueled by cases of scientific fraud and increasing awareness of questionable research practices and failed replications. However, open science is not merely a cure (or punishment) for this crisis - it is also an opportunity. Technological advances enable researchers to more easily conduct reliable, cumulative, collaborative science. Capitalizing on these advances has the potential to accelerate scientific progress. This tutorial paper introduces a workflow for open reproducible code in science (WORCS). WORCS is a step-by-step procedure that researchers can follow to make an entire project Open and Reproducible. It is based on best practices, and can be used either in parallel to, or in absence of, top-down requirements by journals. WORCS lowers the threshold for adopting an open science workflow by providing this tutorial, along with an R-package with user-friendly support functions, and an RStudio project template.

Keywords: open science; reproducibility; r; dynamic document generation; version control

Word count: 5921
The (social) sciences are arguably past the tipping point of the paradigm shift towards open science. Several immediate causes stirred up support for this transition; including highly publicized cases of scientific fraud (as discussed by Levelt, Noort, & Drenth, 2012), increasing awareness of questionable research practices and their consequences for the credibility of findings (John, Loewenstein, & Prelec, 2012), and the replication crisis (Shrout & Rodgers, 2018). However, open science is not merely a cure for this crisis - it is also an opportunity (Adolph, Gilmore, Freeman, Sanderson, & Millman, 2012). Open science helps researchers to more easily conduct reliable, cumulative, and collaborative science (Nosek & Bar-Anan, 2012). Capitalizing on these advances has the potential to accelerate scientific progress (see also Coyne, 2016).

Many individual researchers are motivated to adopt best practices for open science and enjoy these benefits. And yet, the question of “how” to adopt open science practices can be daunting. Making the transition effectively requires researchers to become knowledgeable about different open science challenges and solutions, and to become proficient with new and unfamiliar tools. This paper is designed to ease that transition by providing a simple workflow that meets most requirements open science, based on best practices, and supported by user-friendly software in R: The Workflow for Open Reproducible Code in Science (WORCS). WORCS is a lightweight approach to open science and computational reproducibility. The workflow (Figure ??) constitutes a step-by-step procedure that researchers can follow to make a research project Open and Reproducible. WORCS is compatible with open science requirements already implemented by journals and institutions, and will help fulfill them. It can also be used to produce work in accordance with best practices in the absence of top-down support. The WORCS workflow is supported by an
R-package with user-friendly support functions, and an RStudio project template that is, effectively, a “one click solution”. This paper introduces the WORCS workflow, discusses how it meets best practices for open science, and illustrates the use of the `worcs` package.

**Meeting the requirements of Open Science**

Although Open Science is advocated by many, it does not have a unitary definition. Instead, several guidelines exist for practices that are considered “open”. The TOP-guidelines are one of the most influential concrete operationalisations of Open Science principles (Nosek et al., 2015). These guidelines describe eight standards for open science: 1) comprehensive citation of literature, data, materials, and methods; 2) sharing data, 3) sharing the code required to reproduce analyses, 4) sharing new research materials, and 5) sharing details of the design and analysis; 6) pre-registration of studies before data collection, and 7) pre-registration of the analysis plan prior to analysis; and 8) replication of published results. WORCS defines the goals of open science in terms of these guidelines, and is designed to facilitate meeting each of these guidelines, with one exception: We do not address replication of published results, because replication relates to the subject of a research project, not to its execution. We group the remaining seven guidelines into three categories: Citation (1), sharing (2-5), and preregistration (6-7).

WORCS meets the TOP-guidelines in a way that is amenable to the FAIR Guiding Principles for scientific data management and stewardship (???). These principles advocate that research data should be Findable, Accessible, Interoperable and Reusable. However, they are increasingly applied as a standard for other types of research output as well (???). WORCS helps users meet the FAIR principles by making all research objects - including (origina) data and code - accessible in a (public) repository. The `worcs` package additionally facilitates meeting the FAIR principles in several ways. First, each `worcs` project has a YAML file (https://yaml.org/) that lists the research objects in the project, which will allow web crawlers to index `worcs` projects. Second, a codebook is automatically generated for each data file stored using `open_data()` or `closed_data()`. We encourage
users to elaborate on the default codebook by adding variable description and categories, which might - in the future - enable indexing data by topic area. Incidentally, worcs is itself an example of FAIR research software (see ??? and https://fair-software.eu/). It is hosted on a public version controlled repository (“GitHub”), it has an open source licence (GPL v3.0), is registered in a community registry (CRAN), it enables the citation of the software (using the \texttt{citation("worcs")} command, or by citing this paper), and followed the CII Best Practices software quality checklist during development.

Sharing all research output also provides a proper basis for research evaluation according to the San Francisco Declaration on Research Assessment (DORA; https://sfdora.org/), which plays an increasing role in grant funding, hiring, and promotion procedures. Direct access to research objects allows stakeholders to evaluate research quality based on content rather than relying on spurious surrogate indicators like journal impact factors, conference rankings, and h-indexes. The detailed version control and commit tracking of “GitHub” and similar platforms furthermore make it possible to assess the relative contributions made by different researchers.

\textbf{Existing solutions}

There have been several previous efforts to promote grass-roots adoption of open science principles. Each of these efforts has a different scope, strengths, and limitations that set it apart from WORCS. For example, there are “signalling solutions”; guidelines to structure and incentivize disclosure about open science practices. Specifically, Aalbersberg et al. (2018) suggested publishing a “TOP-statement” as supplemental material, which discloses the authors’ adherence to open science principles. Relatedly, Aczel et al. (2019) developed a consensus-based Transparency Checklist that authors can complete online to generate a report. Such signalling solutions are very easy to adopt, and they address TOP-guidelines 1-7. Many journals now also offer authors the opportunity to earn “badges” for adhering to open science guidelines (Kidwell et al., 2016). These signalling solutions help structure authors’ disclosures about, and incentivize adherence to, open science practices.
A different class of solutions instead focuses on the practical issue of how researchers can meet the requirements of open science. One notable example is the workflow for reproducible analyses developed by Peikert and Brandmaier (n.d.). This excellent workflow strives to ensure strict computational reproducibility for even the most sophisticated analyses. It has some limitations, however: In focusing on reproducibility, however, it only addresses TOP-guidelines 2, 3, and 5. Moreover, in aiming for strict computational reproducibility it ends up being relatively complex to implement. WORCS builds upon the same general principles as Peikert and Brandmaier, which means the two workflows are compatible. What sets WORCS apart is that is an easy and sufficient solution for projects that can be conducted entirely within R, resulting in a much more lightweight approach to safeguard computational reproducibility under most circumstances. WORCS also addresses a unique issue not covered by other existing solutions, namely to provide a workflow most conducive to satisfying the TOP-guidelines and FAIR principles, while being compatible with existing requirements.

Prerequisites

Although the principles underlying this workflow are universal, WORCS has been designed for R (???). Several arguments support the choice to work in R. First, R and all of it extensions are free and open source software, which should be the tool of choice for open science. Second, as of this writing, R is the most advanced programming language in terms of the implementation of tools required for an open science workflow. Third, R is the second-most cited statistical software package (Muenchen, 2012), following SPSS, which has no support for any of the open science tools discussed in this paper. Fourth, R is well-supported; a vibrant online community exists for R, developing new R methods and packages, and providing support and tutorials for existing ones. Finally, R is highly interoperable: Packages are available to load nearly every imaginable filetype, and output can be written to most file types, including DOCX and PDF (in APA style), and HTML format. Moreover, wrappers are available for many tools developed in other programming
languages, and code written in other programming languages can be evaluated from R (including C++, Fortran, and Python, Allaire, Ushey, RStudio, & Tang, n.d.). There are excellent free resources for learning R, such as the book R for Data Science (???).

Working with R is simplified immensely by using the RStudio integrated development engine (IDE) for R (???). Most of the tools used in WORCS are embedded directly into the user interface of RStudio. RStudio automates and streamlines tedious or complicated aspects of working with R, Rmarkdown, and Git/GitHub. This makes RStudio a comprehensive solution for open science research projects. Another important feature of RStudio is project management. A project bundles writing, analyses, data, references, etcetera, into a self-contained folder, that can be uploaded entirely to “GitHub” and downloaded by future users. The worcs R-package installs a new RStudio project template. When a new project is initialized from this template, the bookkeeping required to set up an open science project is performed automatically. The worcs package has a vignette on how to set up your system with all of the required tools.

Introducing the tools

WORCS relies on several free, open source software solutions. The first is dynamic document generation (DDG): Writing scientific reports in a format that interleaves written reports with blocks of code used to conduct the analyses. The text is automatically formatted; the worcs package can output manuscripts as an APA-style paper (thanks to the R-package papaja, Aust & Barth, 2014/2020), or as a host of other scientific formats (see ???), or as a plain “Markdown” format for “GitHub”. When the text is formatted, the results of the code blocks are automatically generated and insterted in the text, or rendered as Figures and Tables. Dynamic document generation supercedes the classical approach of using separate programs to write prose and conduct analyses, and then manually copy-pasting analysis results into the text. Although there is a slight learning curve to transitioning to DDG, the investment really pays off:

• Time saved from painstakingly copy-pasting output and manually formatting text soon
outweighs the investment of switching to a new program

- Human error in manually copying results is eliminated
- When revisions require major changes to the analyses, all results, Figures and Tables are automatically updated
- Flexibility in output formats means that a manuscript can be rendered to presentation format, or even to a website
- Reproducibility is guaranteed, and can be verified by reviewers and/or readers, because the code is run each time the document is compiled.

In sum, while writing academic papers in a programming environment might seem counter-intuitive at first, this approach is much more amenable to the needs of academics than most word processing software. It prevents mistakes, and saves time. We recommend centering a research project around one dynamically generated (RMarkdown) document, which includes all analysis code. Longer scripts can be stored in .R files, and called from the main document using the `source()` function.

The second solution is *version control*: Maintaining an indelible log of every change to all project files. Version control is a near-essential tool for scientific reproducibility, as anyone learns who has had the experience of accidentally deleting a crucial file, or of being unable to reproduce analyses because they ran some analyses interactively instead of using syntax (see also Blischak, Davenport, & Wilson, 2016). Many scientists use some form of *implicit* version control; for example, by renaming files after major changes (e.g., “manuscript_final_2.2-2019-10-12.doc”), tracking changes in word processing software, or using cloud hosting services that retain backups of previous versions. WORCS instead uses the *explicit* version control software “Git” (www.git-scm.com). Git tracks changes to files, and stores these changes when the user makes a “commit” (a snapshot of the version controlled files). Git retains a complete log of all commits, and users can compare changes between different commits, or go back to a previous version of the code (for example, after making a mistake, or to replicate a previous version of the results). A project version
controlled with Git is called a “Repository”, or Repo. Git only version controls files explicitly committed by the user. Moreover, it is possible to prevent files from being version controlled - which is useful for privacy sensitive data. The `.gitignore` file lists files that should not be version-controlled. These files thus exist only on the user’s private computer.

The functionality of “Git” is amplified by services such as “GitHub” (https://github.com). “GitHub” is best understood as a cloud storage service with social networking functionality. The cloud storage aspect of “GitHub” works as follows: You can “clone” (copy) a local “Git” repository to the “GitHub” website, as a backup or research archive. The social network aspect comes into play when a repository on “GitHub” is set to “public”: This allows other researchers to peruse the repository and see how the work was done, clone it to their own computer to replicate the original work or apply the methods to their own data, open “Issues” to ask questions or give feedback on the project, or even send a “Pull request” with suggested changes to the text or code for your consideration. “Git” and “GitHub” shine as tools for collaboration, because different people can simultaneously work on different parts of a project, and their changes can be compared and automatically merged on the website. Even on solo projects, working with “Git”/“GitHub” has many benefits: Staying organized, being able to start a new study with a clone of an old, similar repository, or splitting off an “experimental branch” to try something new, while retaining the ability to “revert” (return) to a previous state of the project, or to “merge” (incorporate) the experimental branch. This paper focuses on “GitHub”, which is the most widely used “Git” cloud storage services. However, as of version 0.1.2, worcs projects can interchangeably use the two main alternative services, “GitLab” (https://gitlab.com) and “Bitbucket” (https://bitbucket.org). The package has a vignette with additional instructions for using these platforms.

The third solution is dependency management: Keeping track of exactly what software was used to conduct the analyses. At first glance, it might seem sufficient to state that analyses were conducted in Program X. However, every program is susceptible to changes,
updates, and bugfixes. R-packages in particular are updated frequently, because they are Open Source, and there is an active community of developers contributing functionality and bugfixes. Potentially, any update could change the results of the code.

Many solutions exist to ensure computational reproducibility. These solutions typically work by enveloping your research project in a distinct “environment” that only has access to programs that are explicitly installed, and maintaining a record of these programs. These solutions differ in user-friendliness and effectiveness. If strict reproducibility is required, one might use the aforementioned Docker-based workflow by Peikert and Brandmaier (n.d.), which runs all analyses inside a “container”: A environment that behaves like a virtual computer that can be stored like a sort of time capsule, and identically reinstated on a user computer, or in the cloud. This is effective, and also preserves software outside the R environment (e.g., preprocessing tools for neuroimaging data). However, it can be difficult to set up for novice users - although a package is in development to facilitate this transition (???). The cloud-based research collaboration platform “Code Ocean” is also based on Docker. It is very user-friendly, but it is not clear whether the “Data Processing Agreement” is fully GDPR compliant - it seems to focus more on limiting liability than on compliance. This is a risk with any cloud-computing solution.

For analyses conducted entirely in R, a slightly less sophisticated solution is offered by the R-package packrat, which installs all packages used in a project to a dedicated library that can be stored and copied with the project. This has the disadvantage that for each project, all packages must be installed anew, which can take a long time (sometimes hours) and a lot of hard drive space. Moreover, “GitHub” is not designed to store your entire project library, so packrat does not address the issue of how other users are supposed to get access to your library. In conclusion, packrat is somewhat unwieldy. The package checkpoint offers an extremely lightweight solution: Instead of storing all R-packages used, it simply records the calendar date, and when another user loads your project, it installs the version of all required packages that was available on the central R repository CRAN on that
date from an archival website. Thus, all that is needed to restore a project is the date on which it was conducted. The main limitation is that this solution is limited to CRAN packages, and cannot mix package versions from different dates.

A recently developed package strikes a great balance between user-friendliness and effectiveness: renv, developed by the team behind “RStudio”. This package maintains a text-based, human-readable log of the packages used, their version numbers, and where they were installed from (e.g., “CRAN”, “Bioconductor”, “GitHub”). This text-based log can be version controlled with “Git”. renv installs all packages in a cache that is shared between all projects that use renv. Thus, if one of your projects uses a package that is already in the cache, it does not need to be installed again. This overcomes the limitations of long installation times and large space requirements of packrat. When someone else loads your project, renv will install all of the required packages onto a cache on their computer. Whereas packrat copies the source code of all packages, renv obtains the packages from their original repositories. Thus, as long as these repositories are maintained, this solution is very lightweight and effective. It is not foolproof, but all solutions for computational reproducibility have a limited shelf life (Brown, 2017). Ultimately, the best solution might be to use a good-enough solution, like renv, and acknowledge that all code requires some maintenance if you want to reproduce it in the future.

Text-based files are better

A key consideration when developing a research project is what filetypes to use. A case can be made to use text-based files only (or as much as possible), instead of binary files. Text-based files can be read by machines and humans alike. Binary files, such as “Word” (.docx) or “SPSS” (.sav, .spo) files, must be decoded by (commercial) software first. “Git” is designed to track any changes to the content of text-based files. Each change reflects a line of text added or removed. This means that, when writing a version-controlled paper in plain-text, it is recommended to insert a line break after every sentence (instead of after every paragraph). That way, the change log will indicate which specific sentence was edited.
Binary files should not be version controlled with “Git”, as changes in the encrypted file are uninterpretable. Binary files are often also larger than text-based files, which means they take up more space on cloud hosting services, such as “GitHub”. For those reasons, uploading large binary files to these platforms is frowned upon. “GitHub” does allow visitors to peruse text-based files online, and conveniently renders certain filetypes: For example, “GitHub” displays .md (Markdown) files as web pages, and .csv files as spreadsheets. This is why worcs uses Markdown and .csv extensively.

How WORCS relates to the TOP-guidelines

Comprehensive citation

TOP encourages comprehensive citation of literature, data, materials, methods, and software. In principle, researchers can meet this requirement by simply citing every reference used. Unfortunately, citation of data and software is less commonplace than citation of literature and materials. Crediting these resources is important, because it incentivizes data sharing and the development of open-source software, supports the open science efforts of others, and helps researchers receive credit for all of their research output, in line with DORA.

To facilitate citing datasets, researchers sometimes publish data papers; documents that detail the procedure, sample, and codebook. Specialized journals, such as the Journal of Open Psychology Data, aid in the publication of these data papers. For smaller projects, researchers sometimes publish their data in an online repository - like “GitHub” - along with a text file with the preferred citation for the data (which can be a substantive paper), and the license that applies to the data, such as Creative Commons BY-SA or BY-NC-SA. When in doubt, one can always contact the data creators and ask what the preferred citation is.

When using R, software (R packages) can be conveniently cited by calling citation("packagename"). This returns an APA-style reference, and a BibTeX entry. In addition to the manual citation of important packages, the worcs project template for RStudio offers a comprehensive record of all dependencies. WORCS generates a template
manuscript in APA6 style, based on the papaja package (Aust & Barth, 2014/2020), which automatically cites all packages used.

One important impediment to comprehensive citation is the fact that print journals operate with space constraints. Print journals often discourage comprehensive citation, either actively, or passively by including the reference list in the manuscript word count. Researchers can overcome this impediment by preparing two versions of the manuscript: One version with comprehensive citations for online dissemination, and another version for print, with only the essential citations. The print version should reference the online version, so interested readers can find the comprehensive reference list. Step 20 of the WORCS procedure suggests uploading the online version to a preprint server. This is important because most preprint servers - at least those hosted by the Open Science Framework (OSF) - are indexed by Google Scholar. This means that authors will receive credit for cited work; even if they are cited only in the online version. Moreover, preprint servers ensure that the online version will have a persistent DOI, and will remain reliably accessible, just like the print version.

It is easiest to mark the distinction between essential and non-essential references from the start, instead of going back to cut references prior to publication. Therefore, the worcs package distinguishes between the traditional at-symbol (@) used to cite a reference, and the “double at”-symbol (@), used to cite a non-essential reference. Users can render a manuscript to PDF either with, or without, comprehensive citations.

The procedure for citation in WORCS is as follows:

1. During writing, maintain a plain-text .bib file with the BibTeX references for all citations.

   - You can make this file by hand; e.g., Figure ?? shows how to obtain a BibTeX reference from Google Scholar; simply copy-paste each reference into the .bib file
   - You can export a .bib file from most reference manager programs; the free, open-source reference manager Zotero is excellent and user-friendly, and highly
interoperable with other commercial reference managers. Here is a tutorial for using Zotero with Rmarkdown.

2. To cite a reference, use the citekey - the first word in the BibTeX entry for that reference. Insert it in the Rmarkdown file like so: @yourcitekey2020. For a parenthesized reference, use [@citekeyone2020; @citekeytwo2020]. For more options, see the Rmarkdown cookbook.

3. To indicate a non-essential citation, mark it with a double at-symbol: @nonessential2020.

4. When Knitting the document, adapt the knit command in the YAML header.

   knit: worcs::cite_all renders all citations, and
   knit: worcs::cite_essential removes all non-essential citations.

5. Optional: To be extremely thorough, you could make a “branch” of the GitHub repository for the print version of the manuscript. Only in this branch, you use the function knit: worcs::cite_essential. The procedure is documented in this tutorial.
Data sharing

Data sharing is important for computational reproducibility and secondary analysis. Computational reproducibility means that a third party can exactly recreate the results from the original data, using the published analysis code. Secondary analysis means that a third party can conduct sensitivity analyses, explore alternative explanations, or even use existing data to answer a different research question. From an open science perspective, data sharing is always desirable. From a practical point of view, it is not always possible. Data sharing may be impeded by legal constraints, ethical restrictions, absence of informed consent for data sharing, and privacy concerns. For example, one concern that applies when working with human participant data, is pseudonimization. It is crucial to pseudonimize data as soon as they are collected, by removing or deleting any sensitive personal information and contact details. Pseudonimized data can often be shared if participants have provided informed consent to that effect. Nevertheless, it is important to note that the European GDPR prohibits storing “personal data” (information which can identify a natural person whether directly or indirectly) on a server outside the EU, unless it offers an “adequate level of protection”. Although different rules may apply to pseudonimized data, there are many repositories that are GDPR compliant, such as the European servers of the Open Science Framework. Different Universities, countries, and funding bodies also have their own repositories that are compliant with local legislation. Before sharing any human participant data, it is recommended to obtain approval from an internal (ethical) review board, guidance from Research Data Management Support, and informed consent from participants.

If data can be shared openly, it can simply be committed it to the “Git” repository, along with the analysis code. This way, others can download the entire repository and reproduce the analyses from start to finish. If data cannot be shared, researchers should aim to safeguard the potential for computational reproducibility and secondary analysis as much as possible. The \texttt{worcs} package uses two solutions to accomplish this goal. The first solution is to publish a checksum of the original data file. Think of a checksum as a 32-character
summary, or as a “one word” description, of the contents of a file. Any change to the file will result in a different checksum. Thus, one can use a checksum to verify the identity of a file, and ensure that its contents are unchanged. When data cannot be shared, the risk of fraud or abuse can be mitigated by publishing a checksum for the original data, in addition to the complete analysis code. Using the checksum to verify the identity of a private dataset, researchers can prove to an independent auditer that running the public analysis code on the private data results in the published results of their work.

The second solution implemented in worcs is to share a synthetic dataset with similar characteristics to the real data. Synthetic data mimic the level of measurement and (conditional) distributions of the real data (see ???). Sharing synthetic data allows any third party to 1) verify that the published code works, 2) debug the code, and 3) write valid code for alternative analyses. It is important to note that complex multivariate relationships present in the real data are often lost in synthetic data. Thus, findings from the real data might not be replicated in the synthetic data, and findings in the synthetic data should not be substantively interpreted. Still, sharing synthetic data facilitates data requests from third parties. A third party can write analysis code based on the synthetic data, and send it to the authors who evaluate it on the real data and send back the results.

**Processing data in WORCS.** Regardless of whether the data will be open or closed, it is important that the raw data will be left unchanged as much as possible. This eliminates human error. Any alterations to the data - including processing steps and even error corrections - should be documented in the code, instead of applied to the raw data. This way, the code will be a pipeline from the raw data to the published results. Every worcs project contains an empty R-script called prepare_data.R. As soon as the data are collected, researchers should use this file to document the steps necessary to load the data into R, pseudonimize it, and prepare it for analysis. For example, if the data was originally in SPSS format with IP addresses and GPS location, this file might just contain the code required to read the SPSS file, and to remove those columns of sensitive personal information.
As soon as the data are pseudonimized and processed into a “tidy” format (i.e., data that can be stored as a spreadsheet, see ???), the researcher should version control some indelible record of the raw data, and avoid making any further changes to the version controlled file.

The worcs package offers two functions for version controlling a record of the data:

One for open, and one for closed data:

- **open_data(data)**: This function stores the data object (a `data.frame` or `matrix`) in a spreadsheet file (called `data.csv` by default). This file should now be (manually) added to the “Git” repository, committed, and pushed to GitHub. Once the GitHub repository is made public, these data will be open to the public. Assume that, once you do this, the data cannot be un-shared.

- **closed_data(data)**: This function also stores the data in a local file (called `data.csv` by default), but that file is added to the `.gitignore` file so it cannot be accidentally added to the “Git” repository. The function also adds a checksum for the original data in the `.worcs` project file, and creates a synthetic dataset (called `synthetic_data.csv` by default). The user should commit all changed files and push them to GitHub. Once the GitHub repository is made public, people will have access to a checksum for the original data that exist on your local machine, a codebook describing those data, and a synthetic copy of the data.

The final line of the `prepare_data.R` file should thus be either `open_data()` or `closed_data()`. The analysis code, on the other hand, should begin with `load_data()`. This function loads the real data if it is available on the user’s computer, and otherwise, loads the synthetic data. This will have the effect that third party users who copy the GitHub repository will automatically load the synthetic dataset, whereas the study authors, who have the real data stored locally, will automatically load the real data. This function makes it possible for reviewers, coauthors, and auditors, to write analysis scripts without requiring access to the original data. They can simply start the script with `load_data()`, and write their code based on the synthetic data. Then, they can submit their code to you -
by email or as a “Pull request” on GitHub. When you run their code on your system, `load_data()` will detect the original data, and use that to run their script. You can then return the results to the third party.

As of version 0.1.1, the `worcs` package only stores data as `.csv` files. As explained above, this is intentional, because `.csv` files are human- and machine readable, and because data typically need to be in a spreadsheet format for analysis. Many types of data can be represented as a spreadsheet - text corpora can have one row per document, and EEG- or EKG waveforms can have one row per measurement occasion. The `prepare_data.R` file should document any steps required to convert these data into tabular format. If it is data must be stored in a different file type anyway, readers are encouraged to follow the development of the `repro` package (Peikert, Brandmaier, & van Lissa, 2020), which will enable such enhancements.

Some users may intend to make their data openly available through a restricted access platform, such as an institutional repository, but not publically through GitHub. In this case, it is recommended to use `closed_data()`, and to manually upload the original `data.csv` file to the restricted access platform. If users wish to share their data through the Open Science Framework, it is sufficient to connect the OSF page to the GitHub repository as an Add-on.

**Sharing code, research materials, design and analysis**

When writing a manuscript in Rmarkdown, analysis code is embedded in the prose of the paper. Thus, the TOP-guideline of sharing analysis code can be met simply by committing the source code of the manuscript to GitHub, and making this GitHub repository Public upon publication. If authors additionally use open data and a reproducible environment (default in WORCS), then a third party can simply replicate all analyses by copying the entire GitHub repository, and Knitting the manuscript on their local computer.

Aside from analysis code, the TOP-guidelines also encourage sharing new research materials, and details of the study design and analysis. These goals can be accomplished simply by placing any relevant documents in the RStudio project folder, committing them to
the Git repository, and pushing to GitHub. As with any document version controlled in this way, it is advisable to use plain text only.

**Preregistration**

Lindsay and colleagues (2016) define preregistration as “creating a permanent record of your study plans before you look at the data. The plan is stored in a date-stamped, uneditable file in a secure online archive.” Two such archives are well-known in the social sciences: AsPredicted.org, and OSF.io. However, GitHub also conforms to these standards. Thus, it is possible to preregister a study simply by pushing a text document with the study plans to GitHub. By tagging the commit as a preregistration (see the workflow vignette), it is distinct from all other commits, and easily findable by people and programs.

The advantages, disadvantages, and pitfalls for preregistering different types of studies have been extensively debated, and that has been summarized elsewhere (e.g., Lindsay et al., 2016). We take the pragmatic position that, in confirmatory (hypothesis-testing) research, it is beneficial to plan projects before executing them, to preregister these plans, and adhere to them. All deviations from this procedure should be disclosed; e.g., prior exposure to the data, whether direct (e.g., by computing summary statistics) or indirect (e.g., by reading papers using the same data); deviations from the analysis plan to handle unforseen contingencies, such as violations of model assumptions; or additional exploratory analyses.

WORCS uses the following approach to preregistration:

2. Commit these documents to the local Git repository, and push them to GitHub
3. On GitHub, tag the release as “Preregistration”. A tagged release helps others retrieve this commit.
4. Optional: Upload the files as attachments to another pre-registration repository, e.g., AsPredicted.org or OSF.io.
   - The `.Rmd` file can be Knitted to a PDF form before uploading to a repository
If the GitHub repository is Public, the preregistration is immediately visible to the world, and reviewers (both formal reviewers designated by a journal, and informal reviewers recruited by other means) can submit comments through pull requests. If the repository is Private, Reviewers can be invited as “Collaborators”. Private repositories can be made public at a later date, along with their entire time-stamped history. It is important to note that contributing to the repository in any way will void reviewers’ anonymity, as their contributions will be linked to a user name.

**Preregistering study plans.** Several excellent preregistration templates are available in the R-package `prereg` (Aust, 2019), including templates from organizations like AsPredicted.org and OSF.io, and from researchers (e.g., van ’t Veer & Giner-Sorolla, 2016). When opening a new RStudio project with the WORCS project template, it is possible to select one of these preregistration templates, which will generate a file called `preregistration.Rmd`. This file should be used to document study plans, ideally prior to data collection.

Because the practice of preregistration is strongly tied to the collection of primary data, it has been a matter of some debate whether secondary data analyses can be preregistered (but see Weston, Ritchie, Rohrer, & Przybylski, 2019 for an excellent discussion of the topic). Preregistration is the only way to ensure that a researcher is not “HARKing”: Hypothesizing after the results are known (Kerr, 1998). When analyzing existing data, it is difficult to *prove* that a researcher did not have direct (or indirect, through collaborators or by reading studies using the same data) exposure to the data, before composing the preregistration. However, the question of proof is only relevant from a perspective of preventing scientific misconduct. Good faith preregistration efforts always improve the quality of deductive (theory-testing) research, because they avoid HARKing, ensure reliable significance tests, avoid overfitting noise in the data, and limit the number of forking paths researchers wander during data analysis (Gelman & Loken, 2014). In all cases, researchers should minimize exposure to the data, and disclose any exposure.
**Preregistering analyses.** The ideal preregistered analysis consists of a complete analysis script that can be evaluated once the data are obtained. This ideal is often unattainable, because the data present researchers with unanticipated challenges; e.g., assumptions are violated, or analyses work differently than expected. Some of these challenges can be avoided by simulating the data one expects to obtain, and writing the analysis syntax based on the simulated data. This topic is beyond the scope of the present paper, but many user-friendly methods for simulating data are available in R, including the package `simstudy`. When using the WORCS-workflow, it is recommended to preregister, if possible, a `.R` script with the planned analyses, along with a verbal, conceptual description. If any changes must be made to the R-script after obtaining the data, one can refer to the conceptual description to explain why these changes are necessary.

**The workflow**

**Preparing your system**

Before you can use the WORCS workflow, you have to install the required software. You have to execute this setup procedure only once for each computer you intend to use `worcs` on. The procedure should take about 30 minutes, and it is documented in the setup vignette. After preparing your system, you can use the WORCS workflow for all of your projects. This step-by-step procedure is detailed in the workflow vignette. In this paper, we provide only a brief outline of the WORCS procedure (as of Version 0.1.2) below. We highly recommend users to refer to the workflow vignette for the most up-to-date procedure (subject to improvements).

**Phase 1: Study design**

1. Create a new (Public or Private) repository on “GitHub”
2. Create a new RStudio project using the WORCS template
3. Optional: Preregister your analysis
4. Optional: Upload preregistration to another repository
5. Optional: Add study Materials to the repository
Phase 2: Data analysis

6. Load the raw data

7. Save the data using `open_data()` or `closed_data()`. Never commit data to “Git” that you do not intend to share

8. Write the manuscript in `manuscript.Rmd`, using code chunks to perform the analyses.

9. Commit every small change

10. Cite essential references with `@`, and non-essential references with `@@`

Phase 3: Submission/publication

11. Store the R environment by calling `renv::snapshot()`

12. Optional: Add a WORCS-badge to your README file and complete the optional elements of the WORCS checklist

13. Make the Private “GitHub” repository Public

14. Create a project page on the Open Science Framework

15. Connect your “OSF” project page to the “GitHub” repository.[https://help.osf.io/hc/en-us/articles/360019929813-Connect-GitHub-to-a-Project]

16. Add an open science statement to the Abstract or Author notes, which links to the “GitHub” repository or “OSF” page

17. Knit the paper to PDF

18. Optional: Publish a preprint

19. Submit the paper, and tag the release of the submitted paper as in Step 3.

Notes for cautious researchers

Some researchers might want to share their work only once the paper is accepted for publication. In this case, we recommend creating a “Private” repository in Step 1, and completing Steps 13-18 upon acceptance.
In this tutorial paper, we have presented a workflow for open reproducible code in science. The workflow aims to lower the threshold for grass-roots adoption of open science principles. It is accompanied by an R-package with convenience functions, and a WORCS project template for RStudio. This relatively light-weight workflow meets most of the requirements for open science, as detailed in the TOP-guidelines. However, several limitations and issues for future development remain.

One potential challenge is the learning curve associated with the tools outlined in this paper. Learning to work with R, Rmarkdown, and Git/GitHub requires an initial time investment. The amount of time required is reduced by the availability of tutorials, such as this one, and other tutorials cited throughout this document. Moreover, the time investment tends to pay off. Working with R opens the door to many cutting edge analysis techniques. Working with Rmarkdown saves time and prevents mistakes by avoiding tedious copying of results into a text document. Working with Git/GitHub keeps projects organized, prevents accidental loss of work, enables integrating changes by collaborators in a non-destructive manner, and ensures that entire research projects are archived and can be accessed or copied by third parties. Thus, the time investment is eminently worthwhile.

Another important challenge is managing collaborations when only the lead author uses Rmarkdown, and the coauthors use Word. In this case, it is possible to Knit the manuscript to Word (.docx), by changing the line `output: papaja::apa6_pdf` in the manuscript’s YAML header to `output: papaja::apa6_docx`. There are some limitations to the conversion, discussed here. When soliciting feedback from co-authors, ask them to use Tracked Changes and comment bubbles in Word. Then, manually incorporate their changes into the `manuscript.Rmd` file. In most cases, this is the most user-friendly approach, as most lead authors would review changes by co-authors anyway. A second approach is to ask collaborators to work in plain text. In this case, send collaborators the `manuscript.Rmd` file, and ask them to open (and save) it in Word or Notepad as a plain text file. When they send
it back, make sure any changes to your local file are committed, and then simply overwrite your local version with their file. In RStudio, select the file in the Git tab, and click the Diff button to examine what changes the collaborator has made relative to your last committed version. Finally, Peikert and Brandmaier (n.d.) mention the redoc package (Ross, 2018/2020) as a potentially useful way to collaborate with Word users. This package enables bidirectional conversion between Word and Rmarkdown, with support for tracked changes. However, redoc is currently in suspended development, and for many users, a simple solution (i.e., asking collaborators to work in plain text) might be sufficient.

If all collaborators are committed to using WORCS, they can Fork the repository from the lead author on GitHub, clone it to their local device, make their own changes, and send a pull request to incorporate their changes. Working this way is extremely conducive to scientific collaboration (Ram, 2013). Recall that, when using “Git” for collaborative writing, it is recommended to insert a line break after every sentence so that the change log will indicate which specific sentence was edited, and to prevent “merge conflicts” when two authors edit the same line. The resulting document will be rendered to PDF without spurious line breaks.

Moreover, being familiar with GitHub opens doors to new forms of collaboration: In the open source software community, continuous peer review and voluntary collaborative acts by strangers who are interested in a project are commonplace (see Adolph et al., 2012). This kind of collaboration is budding in scientific software development as well; for example, the lead author of this paper became a co-author on several R-packages after submitting pull requests with bug fixes or additional functionality (Hallquist, Wiley, & Lissa, 2018; Rosenberg, Beymer, Anderson, Van Lissa, & Schmidt, 2018), and two co-authors of this paper became involved by contributing pull requests to worcs. It is also possible to invite such collaboration by opening Issues for tasks that still need to be accomplished, and tag known Collaborators to address them, or invite external collaborators to contribute their expertise.
Future developments

The WORCS procedure and accompanying R-package provide a user-friendly and lightweight workflow for open, reproducible research in R, that meets all TOP-guidelines. It is beyond the scope of WORCS to support the incorporation of tools outside of the R-environment, or to containerize a project so that it can be identically reinstated on a different system or virtual machine. One important area of future development is to enable these extensions of the workflow. To this end, we are integrating WORCS into a modular framework of reproducibility tools for R (Peikert et al., 2020). This will make it possible for researchers to enhance a WORCS project with strict reproducibility tools like “Docker” and “Make”, or vice versa, to initiate a WORCS project in a strictly reproducible environment. This combines the strengths of WORCS with those of Peikert and Brandmaier (n.d.).

Conclusion

WORCS offers a workflow for open reproducible code in science. The step-by-step procedure outlined in this tutorial helps researchers make an entire research project Open and Reproducible. The accompanying R-package provides user-friendly support functions for several steps in the workflow, and an RStudio project template to get the project set up correctly.

WORCS encourages and simplifies the adoption of Open Science principles in daily scientific work. It helps researchers make all research output created throughout the scientific process - not just manuscripts, but data, code, and methods - open and publicly assessible. This enables other researchers to reproduce results, facilitates cumulative science by allowing others to make direct use of these research objects.
References

Aalbersberg, I. J., Appleyard, T., Brookhart, S., Carpenter, T., Clarke, M., Curry, S., ...  
Statement. https://doi.org/10.31219/osf.io/sm78t

Aczel, B., Szaszi, B., Sarafoglou, A., Kekecs, Z., Kucharský, Š., Benjamin, D., ...  

https://doi.org/10.1080/1047840X.2012.705133


https://doi.org/10.1371/journal.pcbi.1004668


“Garden of forking paths”–explains why many statistically significant comparisons don’t hold up. *American Scientist, 102*(6), 460–466. Retrieved from https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=00030996&v=2.1&it=r&id=GALE%7CA389260653&sid=googleScholar&linkaccess=abs


https://www.psychologicalscience.org/observer/research-preregistration-101


for Increasing the Transparency of Analysis of Preexisting Data Sets. *Advances in Methods and Practices in Psychological Science.*

https://doi.org/10.1177/2515245919848684