Realistic and Robust Reproducible Research for Biostatistics

Boris P. Hejblum[†*1], Kevin Kunzmann[†2], Ennio Lavagnini[3], Anna Hutchinson[2], David S. Robertson[2], Sacha C. Jones[4], and Annemarie H. Eckes-Shephard[5]

[1]Université de Bordeaux, Inserm Bordeaux Population Health U1219, Inria SISTM, Vaccine Research Institute, Bordeaux, F-33000, France.
[2]MRC Biostatistics Unit, University of Cambridge, Cambridge, CB2 0SR, UK.
[3]Department of Chemistry, University of Cambridge, Cambridge, CB2 1EW, UK.
[4]Office of Scholarly Communication, Cambridge University Library, University of Cambridge, West Road, Cambridge, CB3 9DR, UK.
[5]Department of Geography, University of Cambridge, Downing Place, Cambridge CB2 3EN, UK.

Abstract

The complexity of analysis pipelines in biomedical sciences poses a severe challenge for the transparency and reproducibility of results. Researchers are increasingly incorporating software development technologies and methods into their analyses, but this is a quickly evolving landscape and teams may lack the capabilities to set up their own complex IT infrastructure to aid reproducibility. Basing a reproducible research strategy on readily available solutions with zero or low set-up costs whilst maintaining technological flexibility to incorporate domain-specific software tools is therefore of key importance. We outline a practical approach for robust reproducibility of analysis results. In our examples, we rely exclusively on established open-source tools and free services. Special emphasis is put on the integration of these tools with best practices from software development and free online services for the biostatistics domain.

Keywords: Biostatistics; Data management; Reproducibility; Workflow automation

1 Introduction

Contemporary empirical research critically relies on computational workflows to make sense of the ever increasing amounts of data collected. Few research papers in the biomedical sciences stand a chance of being published without evidence in the form of data to support or refute their hypotheses. The complexity of computational analysis workflows and the tools used therein varies greatly among disciplines and can range from calculating simple univariate statistics in a point-and-click user interface to a full-fledged analysis pipeline involving high-performance-computing (HPC) resources, command scripts and large databases.

Reproducibility of analysis results is crucial to build trust in the presented work and to allow for the dissemination and re-use of workflow components or implementation ideas. Reproducibility has the power to increase the efficiency and impact of research, and should thus be a key focus of publicly funded research. However, the increasing complexity and dependency on domain-specific software makes reproducing results from published papers difficult, and often, descriptions of workflows are informal, incomplete, or completely absent.

The so-called science reproduciblity crisis [Ioannidis, 2005, 2014, Baker, 2016] has prompted the development of several resources on reproducible research in the broader scientific community, including the National Academies of Sciences, Engineering, and Medicine [2019] and The Turing Way Community et al. [2019]. Here we summarise the state-of-the-art practices of reproducible research practices oriented specifically towards the field of biostatistics.

We begin by delineating the terms "reproducibility" and "replicability" since their definitions can vary across scientific fields and have evolved over time [Barba, 2018]. The distinction between "reproducible"

*correspondence: boris.hejblum@u-bordeaux.fr
†these authors contributed equally

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

and "replicable" can be extended to a square diagram of all four possible combinations of the same (or new) data and the same (or different) code, as illustrated in Figure 1.

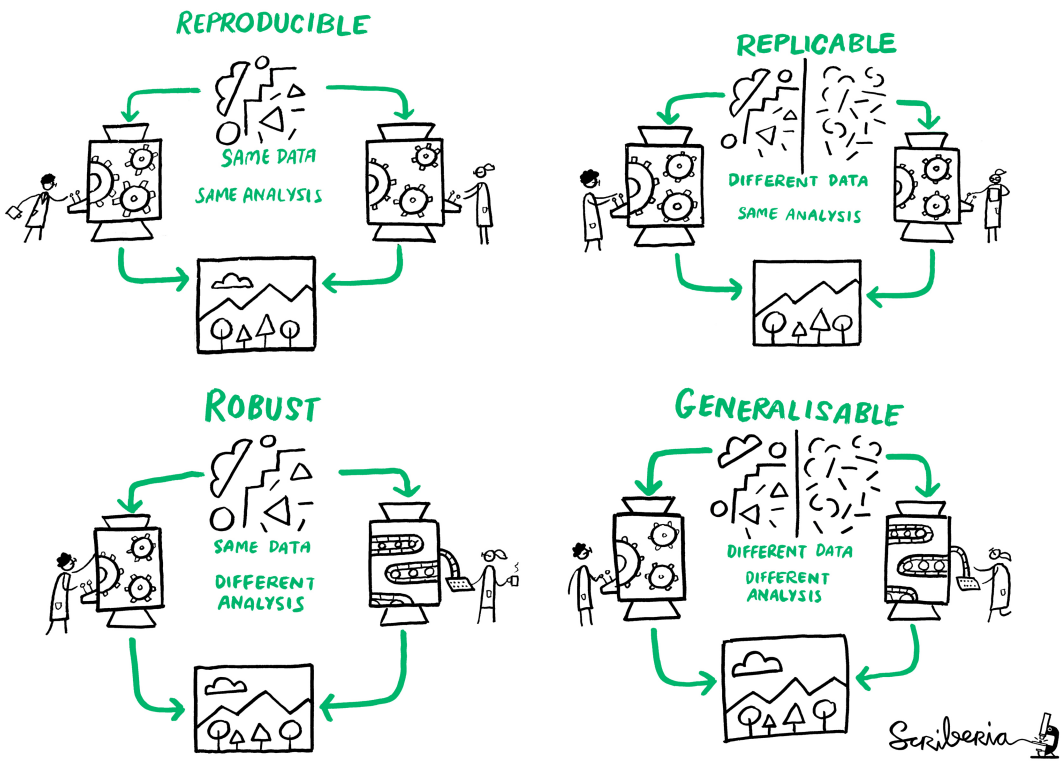

Figure 1: Reproducible, robust, replicable, and generalisable research defined in terms of same/different code and same/new data; image created by Scriberia for The Turing Way community, CC-BY [The Turing Way Community and Scriberia, 2020].

In concordance with The Turing Way Community et al. [2019], we define the combination of "same data" and "different code" (for the same analysis goal) as "robust"[1] if it leads to the same qualitative results, and "generalisable" if the conclusions are also stable to re-analysis using newly collected data. Note that a single analysis can never be proven empirically to be either "replicable" or "generalisable" and rather, these qualities need to be established over time by attempting to replicate a previous study or experiment.

The ultimate goal in science is to provide *generalisable* results. Evidence-based scientific methods in (life) sciences are eventually about convincing colleagues, decision makers, and the public of scientific facts by reasoning based on empirical observations and analyses. Data collection and data analysis have become an integral part of this, and the ability to reliably reproduce results from data, i.e. to make an argument verifiable, is therefore essential to the credibility of the scientific debate. To facilitate the generation of truly generalisable results, the first step is thus to make the primary analysis *reproducible*. With thorough (and also reproducible) sensitivity analyses, one may then claim to have a *robust* analysis, which in turn greatly facilitates the process of replication with new data since the entire analysis pipeline is already available, reaching towards generalisable results. In that sense, reproducible research methods and tools are a vehicle towards both making results more credible, and to facilitate replication and thus the output of truly generalisable results.

Particularly in the context of medical statistics and the life sciences it is important to distinguish between the terms "reproducible research" and "Open Science". The open science movement propagates the spread of knowledge through the use of "digital technologies and new forms of collaboration" [Directorate-General for Research and Innovation (European Commission) et al., 2018]. The OECD definition of Open Science is even more concrete and explicitly references research data management as

[1] This notion can be extended to include qualitatively similar (statistical) analysis methods but, in the strict sense, refers to the choice of hyperparameters and the like. Since individual analysis methods may sometimes be seen as a family of methods parameterised via hyperparameters (e.g., LASSO and ridge regression as special cases of elastic net, however, the distinction is not well-defined).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

well "to make the primary outputs of publicly funded research results - publications and the research data - publicly accessible in digital format with no or minimal restriction" [OECD, 2015]. For example, preprint services such as arXiv.org, bioRxiv.org or medRxiv.org serve this goal. Reproducible research methods can therefore be seen as an integral part of making the process of obtaining scientific results more transparent and help spread methodological knowledge by providing readily available examples. It should also be noted that the OECD definition of Open Science *does* account for the fact that some research data may be subject to restrictions on sharing but that researchers are required to keep access restrictions to a minimum. This is particularly important to the life sciences where personal health-care data will frequently be subject to privacy regulations such as GDPR [European Commission, 2018] or HIPAA [Public Law 104-191, 1996]. Even research based on data that are not fully open (not publicly available) should make the best effort to be reproducible. Efforts should be made to properly separate research data from both the definition of the required computing environment and the analysis code since the latter two can still be shared publicly and thus scrutinised by peer reviewers even without access to the raw data. Furthermore, upon obtaining access to the source data via a well-defined process, the results may still be reproduced.

Solutions to aid with reproducibility may be platform based – e.g. Gigantum [Gigantum Funding Team et al., 2018] –, specific to a particular programming language – e.g. Drake [Landau, 2018] – or cover different aspects of reproducibility – e.g. Conda for reproducible environments [Anaconda Inc., nd] or Jupyter notebooks for literate programming [Knuth, 1984, Jupyter Team, nd]. The number of different technologies and platforms is increasing so quickly that it can be a daunting task just to pick the set of tools and platforms to begin with. In this manuscript, we will put forward what we term a concept for "realistic and robust reproducible research" particularly geared towards analyses in biostatistics.

Definition: Reproducibility

Reproducible means that the *exact* same results (figures, tables, plots, or even entire documents) can be obtained from the *exact* same input.

Definition: Robust

Robust means that (software) tools used for a reproducible research pipeline should be mature enough to not themselves introduce more instability. Anticipated long-term availability and (community) support should be of primary concern.

Reproducible research is a continuum ranging from not retaining any information on the data processing steps to a fully automated process using integrated containers or virtual machines (cf. Section 5). User intervention should be minimised, in that intermediate steps should be automated as much as possible to reduce the potential for ambiguous or elusive process documentation, but any step – no matter how non-technical – that helps with understanding the data processing and facilitates reproduction of results should be welcomed and encouraged.

In biostatistics, reproducibility has been a long standing concern. Pharmaceutical companies currently have to file hundreds of pages documenting the various clinical evidence in favour of a new drug or treatment (often trials of various phases) to the FDA (Food and Drug Administration) or the EMA (European Medicines Agency). With the advent of modern tools for data management, version control, archiving and reporting, reproducibility and transparency of analyses have never been as accessible for biostatisticians. However, the very reasons that lead to the emergence of biostatistics as its own field warrants the need to adapt these tools to the specifics of biostatistics. For instance, whilst there is no reason not to release the source code used for a data analysis (regardless of the programming language used), ethical and privacy considerations generally prevent a full open data strategy [European Commission, 2018]. Fortunately, although open-source software and open data can be important ingredients in reproducible research, these are not mandatory. Indeed, being open is not the same thing as being reproducible – even though openness facilitates external reproducibility. In some specific contexts, for instance in transcriptomics data (which have a long tradition of openness through the Gene Expression Omnibus data archive), part of the data can be shared publicly. In such cases, the data should be provided in a non-proprietary and non-binary format to ensure cross-platform readability and posterity on a persistent archive. In most cases, however, data can only be shared privately or not shared at all. This is due to study participant consent which usually restrains further re-use of health data due to their particular personal and sensitive nature. Exceptions to this are if appropriate participant consent was sought and obtained before and during data collection for future research, data re-use and data sharing.

The remainder of this article is organised as follows: Section 2 discusses the importance of good data management, Section 3 presents version control tools for source code, Section 4 explains why and how to automate a workflow, Section 5 considers how one can manage code dependencies, Section 6 highlights the benefits of persistent archiving for long term citability, and Section 7 outlines our perspective for the future of realistic and reproducible research in biostatistics.

2 Data Management

Definition: Data Management
Any activity related to the description, documentation, storage, usage and sharing of research data.

Good management of data is essential for the reproducibility and overall integrity of research outputs. Research Data Management (RDM) is important throughout the timeline of a project, from the project planning stages and while applying for funding, until the end of the project and beyond when data sharing, archiving and preservation are critical practices.

Many major funders require researchers to include a data management plan in their grant applications, and to share their data as open data (i.e. public sharing) at the time of publication, within a specified time frame, or within a reasonable period from the project's end. For example, Research Councils in the UK instil the principle that "publicly funded research data are a public good, produced in the public interest, which should be made openly available with as few restrictions as possible in a timely and responsible manner" [UKRI, 2015]. Where the data contain personal or sensitive information, sharing restrictions can be applied and anonymisation protocols followed, but in general "data should be as open as possible, as closed as necessary" [European Commission. Directorate-General for Research & Innovation, 2016].

Good RDM is the responsibility of all members of a research project. This section highlights key elements of RDM that facilitate robust reproducible research that should be applied to every research project, no matter how big or small, or if it is funded or unfunded. Table 1 provides links to some helpful resources for the whole RDM life cycle. At the end of this section, we make particular reference to managing specifically biostatistical data, especially given that these often contain personal or sensitive information (e.g. patient data), thus eliciting caution when it comes to sharing data (and sometimes code).

The importance of writing a data management plan (DMP) for a robust and reproducible research project cannot be underestimated, regardless of whether or not supplying one is a funder requirement. DMPs are used to plan how existing and new datasets and code will be synthesised, created, stored, preserved and shared during and after the project, taking into account any ethical or legal considerations and detailing quality assurance measures. DMPs should aim to cover the entire project lifecycle and be considered as 'living documents' to be revisited and reworked as necessary. Key benefits of creating, maintaining and adhering to a DMP are: i) to pre-empt any data-related predicaments before they occur, ii) to factor into the budget any data-related costs prior to the project's inception, iii) to increase the efficiency of research, iv) to guarantee the reusability of the data, and v) to support reproducibility. There are a number of resources to assist researchers in writing a DMP, from the guidance documents of specific funders to online resources devoted solely to DMPs. An excellent example of the latter is DMPonline, a tool provided by the Digital Curation Centre [Digital Curation Centre, nd] to help researchers create and share DMPs whilst meeting institutional and funder requirements. Likewise, there are also resources to support the management of research software; for example, from the Software Sustainability Institute who provide advice on writing a Software Management Plan [Jackson, Michael (ed.), 2018].

The basics of RDM that should be applied to every research project include: i) storing data carefully and securely (according to the appropriate standards in the case of sensitive data), ii) backing up frequently and in at least two separate locations, and iii) using a file naming convention so that others within and outside a project can understand a file's content. Simple (informal) forms of version control are important for all data files, and version control systems should be applied to source code in particular (see Section 3). An additional and critical area of RDM is data documentation, or metadata that describes a dataset fully and comprehensively. The Biological and Biomedical community use a plethora of standardised metadata vocabulary, from taxonomic ontologies (e.g. vertebrate taxonomy ontology [Midford et al., 2013]), to vocabularies on general and specific methods (e.g. Minimum Information for Biological and Biomedical Investigations (MIBBI, Taylor et al. [2008]), BioAssay Ontology (BAO, Abeyruwan et al. [2014])) and more – see the Ontology Lookup Service [EMBL-EBI, nd]. The consequent use of vocabularies not only helps humans to better interpret data and its context, but also enables analysis

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

software to be more interoperable, with implications for the reproducibility and reusability of datasets and tools. In contrast, without clear documentation, dataset or analysis workflow reuse can be difficult and reproducibility unlikely, and worryingly, the risks of misinterpretation or misuse of data can be high.

Data documentation should be a continuous process, from the point of data collection to data sharing and/or archiving, and essentially involves providing all the necessary information to permit comprehension by others, data discovery, data reuse and research reproducibility. Some tools exist to facilitate this process. For example, LabKey, an open-source specimen and data management platform, uses standardised metadata vocabulary oriented towards immunological research. The platform has some internal analysis and collaborative features such as specimen-sharing and secure data sharing capabilities, suitable for many types of biomedical data [Nelson et al., 2011], and is extensible through the addition of contributed specific data type modules. Another tool, which focuses on data management and submission to repositories is COPO (Collaborative Open Plant Omics), a metadata data annotation and data brokering platform [Etuk et al., 2019], especially for sequence data management and submission to the EMBL-EBI.

While this documentation remains a time-consuming task, both platforms have error-checking and batch-processing features in place, thus saving some time and effort for the researcher. Another RDM feature which some see as labour-intensive at first (yet has long-term benefits for the data producers and users) is the creation of a Readme file (or series of Readme files for complex multi-file datasets) that is deposited together with the data. For datasets, clear advice on producing Readme files is provided by Cornell University [Research Data Management Service Group, Cornell University, nd].

Practicing the aforementioned elements of RDM are important regardless of whether or not the data can be shared with others (i.e., enabling the data creators, and not just others external to a project, to access and understand the data in future years). In addition to these elements, there are two critical procedures to follow in order to share data well and to preserve it to enable future access: choose a trusted data repository (e.g. one that has CoreTrustSeal certification [Core Trust Seal, nd], and adhere to the FAIR principles (see below). There are a large number of repositories available for data. These can be discipline-specific (domain) repositories, institutional repositories or general purpose repositories (see re3data.org [re3data.org, nd] for a registry of research data repositories [Pample et al., 2013]). It is important that the chosen repository meets certain standards; for example, one that provides persistent identifiers (e.g. DOIs) for datasets, collects comprehensive and machine-readable metadata (e.glicensing information), offers long-term data preservation (see Section 6), and provides a service that performs checks on the metadata and data. Trustworthy discipline-specific repositories, followed by institutional repositories, are preferable over general purpose repositories due to the additional curation and standards checks that these provide [OpenAIRE, nd].

The FAIR principles and research data repositories are inherently linked. Making data FAIR – Findable, Accessible, Interoperable, Reusable [Wilkinson et al., 2016] – involves making it available in a trusted data repository, with comprehensive metadata, in open (non-proprietary) formats, with a permanent identifier (DOI), and with a clear and machine readable license. As we shall see below, however, not all data can be shared in a repository or made available as open data: data that is restricted can still be made FAIR but there are a number of ethical considerations that need to be acknowledged when applying the FAIR principles (e.g. see Boeckhout et al. [2018] for a critical appraisal of the FAIR principles, with particular reference to genomic data). There are several good sources of information about the FAIR data movement [Collins et al., 2018, Sansone et al., 2019]; as well as related initiatives that aim to support researchers (e.g., a short course provided by FOSTER on 'Assessing the FAIRness of data' [FOSTER Open Science, nd]).

Definition: FAIR data

FAIR data are all data or metadata that complies with the principles of **F**indability, **A**ccessibility, **I**nteroperability and **R**eusability [Wilkinson et al., 2016]. These definitions were introduced to provide a guideline for the storage and management of scientific data.

As already recognised, some data is of a sensitive nature and cannot be shared as open data, yet certain steps can be pursued to facilitate data sharing. Although data sharing most often occurs towards the end of (or after) the project life cycle, one step needs to be implemented during the project's early stages; namely providing appropriately phrased participant information and consent forms to facilitate data sharing. For example, anonymised data can be shared openly as long as participant consent does not preclude this. An additional step to facilitate sharing is to use a repository that specialises in controlled (or managed) data access and/or supports the handling of clinical data; for a review, see Banzi et al. [2019].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

Ultimately, the prevailing problem in making biostatistical analyses more reproducible is the fact that data is often subject to specific privacy regulations such as GDPR or HIPAA. Even pseudonymised patient data should be properly secured and will, in general, not be publicly available. This makes true reproducibility harder to achieve since data can rarely be made publicly accessible on the same infrastructure as analysis. Usually, there are no reasons not to make analysis scripts available without access control, with the exception of cases where a script itself might reveal personal data, e.g. by processing certain special cases explicitly. A sufficiently hardened infrastructure to store and process data (e.g. such as LabKey) is necessary to ensure internal reproducibility and traceability of data and analyses inside institutions working with personal data, and a sophisticated access control system must be in place to facilitate data and specimen-sharing with both internal and external researchers. Access should only be granted on the basis of a data sharing agreement [Ohmann et al., 2017] ensuring that the recipient takes full responsibility under the applicable protective legislation for (in particular not trying to de-anonymise individuals and not re-distributing data). Additionally, data access should only be granted on a temporary basis to prevent an ever increasing circle of authorised users. All in all, the strict privacy controls for data storage necessitate a strict separation of analysis code and data since the code might still be shared publicly even if the data cannot. While publicly available code may not suffice to reproduce an analysis without access to the data, it still facilitates independent verification and dissemination of the process.

RDM activity	Description	Resource
Data Management Plan	Useful templates and funder-specific guidance	`https://dmponline.dcc.ac.uk`
Software Management Plan	Guidance and checklist	`https://doi.org/10.5281/zenodo.2159713`
Readme	Advice on producing Readme files	`https://data.research.cornell.edu/content/readme`
Metadata annotation	List of life science-specific metadata vocabularies	`https://www.ebi.ac.uk/ols/ontologies`
Data Deposition	Help to choose your discipline-specific repository. Also: watch out for the CoreTrustSeal	`https://www.openaire.eu/opendatapilot-repository-guide` `https://www.coretrustseal.org`
FAIR principles	Short course on assessing the FAIRness of your data	`https://www.fosteropenscience.eu/learning/assessing-the-fairness-of-data`

Table 1: RDM resources helpful throughout the research data life cycle.

3 Source Code Version Control

3.1 Overview

Contemporary data analysis often involves interrelated code, data sets and output files. As the analysis develops over time, small changes to the files – such as the misplacement of a bracket in the code – may result in large changes to the output or even break the pipeline entirely. To handle this complexity, it is generally advisable to employ some form of version control.

Definition: Version control

Version control is a system that records changes to a file or set of files over time so that users can recall specific versions later. [Chacon and Straub, 2014]

A very basic example of version control is re-saving a file with a new name (e.g. "version2") after amending the file, but this approach is error-prone and becomes difficult to manage when there are multiple files. Moreover, there is no systematic way to view changes across versions and instead, one has to manually inspect differing versions. These limitations are immediately compounded if the project involves collaboration between researchers, yielding it largely insufficient for contemporary data analyses.

The collection of files or folders comprising the project is more commonly known as a *repository* and it is this that is said to be under version control, whereby all changes to the repository over time are saved. This enables users to 'time-travel' between different snapshots of the files in the repository.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

Modern version control systems come in two flavours: i) "centralised version control systems" (CVCS) such as Subversion [The Apache Software Foundation, nd], and ii) the more recent "distributed version control systems" (DVCS) such as Mercurial [Mercurial community, nd] or Git [Git community, nd]. In centralised systems, a single central copy of the repository is stored on a server and provides access to users. In distributed systems, users clone a copy of the repository locally, including the full history of the project. We particularly focus on Git in the remainder of this section, due to its widespread popularity within the Biostatistics field.

3.2 Git

Git is extremely reliable and well-supported due to extensive use by the open source community and large software companies. The community support is exceptional and there are several popular *hosting services* for Git repositories (e.g. GitHub, GitLab and Bitbucket). These hosting services facilitate online collaboration, allowing easy access and interaction with the 'master copy' of a project, with additional features such as issue tracking and task management tools. However, it should be noted that none of these Git hosting services are directly suited for long-term storage and archiving of source code. Specifically, they do not guarantee long-term availability nor citeability. Instead, dedicated archiving services such as Zenodo provide tight integration with these Git hosting services (see Section 6).

Git provides a complete history of all the changes made to a repository over time, including informative messages describing the edits called *commit messages*, alleviating problems involving duplicate files which may arise when using informal version control strategies. Commit messages serve as easily identifiable 'checkpoints', ensuring that all changes to files in the repository are formally documented and ensuring that files or whole repositories can be safely reverted to if necessary [Ram, 2013].

Git accommodates 'offline editing' unlike earlier version control systems such as SVN which requires users to be online to look at the history and to make changes to the repository. Git also has specific advantages when managing a project with collaborators. Collaborators are able to fork or branch the repository, before merging their proposed changes with the original version, meaning that multiple collaborators can work on the project at the same time. This asynchronous workflow is extremely flexible, and enables regular and structured reconciliation of all versions of the files in the repository [Bryan, 2017]. Git provides a full history of authorship that anyone can access, allowing researchers and reviewers to track and assess individual author contributions to a project, and also offers cryptographically secure commits to ensure that they are from a trusted source. Table 2 gives some suggested online resouces to learn about Git and GitHub.

Resource	Level
`http://swcarpentry.github.io/git-novice/`	Beginner (free online course)
`https://try.github.io/`	Beginner (tutorials)
`www.udemy.com/course/the-ultimate-git-5day-challenge`	Beginner (free online course)
`https://www.atlassian.com/git/tutorials`	Multiple (tutorials)
`https://lab.github.com/githubtraining/paths`	Multiple (tutorials)

Table 2: Resources for learning Git and GitHub

3.3 Version control and reproducibility

Version control is not synonymous with reproducibility and having one does not guarantee the other. Rather, version control is a tool that can help make science more reproducible. A basic way it does so is facilitating code sharing through a well-established online community, for example code being made publicly available on GitHub. Transparent access to all the project code gives computational details that are required for reproducing results, such as parameter values and the source code of functions including those within wrapper functions.

Potential drawbacks of using Git for version control are that it is not designed to easily compare or merge changes made to binary files (e.g. Microsoft Word documents) and there are size limits on files that can be uploaded to Git hosting services. The latter may hinder reproducibility – for example, in machine learning applications where large amounts of data are used to train statistical models, it may be problematic to track all the changes that have been made to the training data [Herron, 2019]. An alternative version control system for machine learning applications is DVC [DVC community, nd] which

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

is optimised for storing and versioning large files. For a project to be fully reproducible, ideally all files – including large data sets (in the absence of sensitive data) – should be made available.

Version control not only helps to ensure that results can be reproduced, but it also helps researchers to build on and extend the work of others, and hence generalise methods to answer new research questions. As a first step, a researcher can obtain a complete copy of a repository by forking or branching it, so that they can safely experiment with new methods. A researcher wishing to extend the analysis (for example, by using different data) is able to access a detailed history of the workflow to see where and why key decisions were made, and whether these are applicable to their problem. Thus, version control helps facilitate reproducible, replicable *and* generalisable research.

4 Workflow Automation

Even small real-word analyses often require a multitude of outputs such as intermediate data sets, whether it is for the production of single figures and tables, or entire reports. The task of manually running all code and compilations in the correct sequence, whilst respecting their potential inter-dependencies, can quickly become challenging.

> "Unfortunately, it is very easy for a programmer to forget which files depend on which others, which files have been modified recently, and the exact sequence of operations needed to make or exercise a new version of the program. [...] recompiling everything in sight just to be safe is very wasteful."
>
> — S.I. Feldman [Feldman, 1979a]

These early words used by Feldman, the developer of the ubiquitous build automation tool *make* (Free Software Foundation, 2016) still hold true today. Replace *"programmer"* with *scientist*, *"program"* with *analysis* and *"recompiling"* with *rerunning*, and it becomes evident that this issue is the same as that faced today by computational scientists.

Definition: Workflow

A **workflow** is a description of the steps required to obtain a set of predefined outputs. A **pipeline** is a linear workflow with a single output.

Definition: Workflow engine

A **workflow engine** is any tool for parsing and executing a defined workflow.

A 'workflow' is a description of the steps required to obtain the desired outputs starting either from scratch or a defined set of inputs. 'Workflow engines' are software tools that can alleviate the issues named above by keeping track of the necessary intermediate steps for producing a user-requested output and executing these in the correct sequence. An analogy to the interplay between workflow and workflow engine would be a detailed written description of the individual steps (the workflow) and a person following the description (the workflow engine). Obviously, this is a tedious and error prone approach, and software-based workflow engines allow the automatic execution of formalised workflows. Workflow engines offer an attractive opportunity to organise the complexity of producing outputs in a well-documented, consistent way and enable interested parties to re-run an entire analysis or parts of it in an entirely automated fashion. An example workflow is laid out in Figure 2. A brief description of the most prominent workflow engines in increasing order of complexity is given in the following.

4.1 The ubiquitous: bash

Anyone who has worked with a Unix terminal will have come across the widely used default login shell bash (Free Software Foundation, 2019). Putting bash in context with workflow engines, a simple bash shell script can be seen as a formalised workflow description and the executing shell (usually bash) as the corresponding workflow engine. Since bash is a scripting language itself, this is a highly generic way of describing workflows allowing one to 'glue' together any program that can be run in a Unix shell. In fact, many of the more advanced workflow engines can be seen as wrappers around executing shell scripts (for example *make* and snakemake). The generality of shell scripting comes at the cost of having to manually implement routines for recurring tasks in standard analysis workflows, such as caching intermediate results and checking the existence of required outputs.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

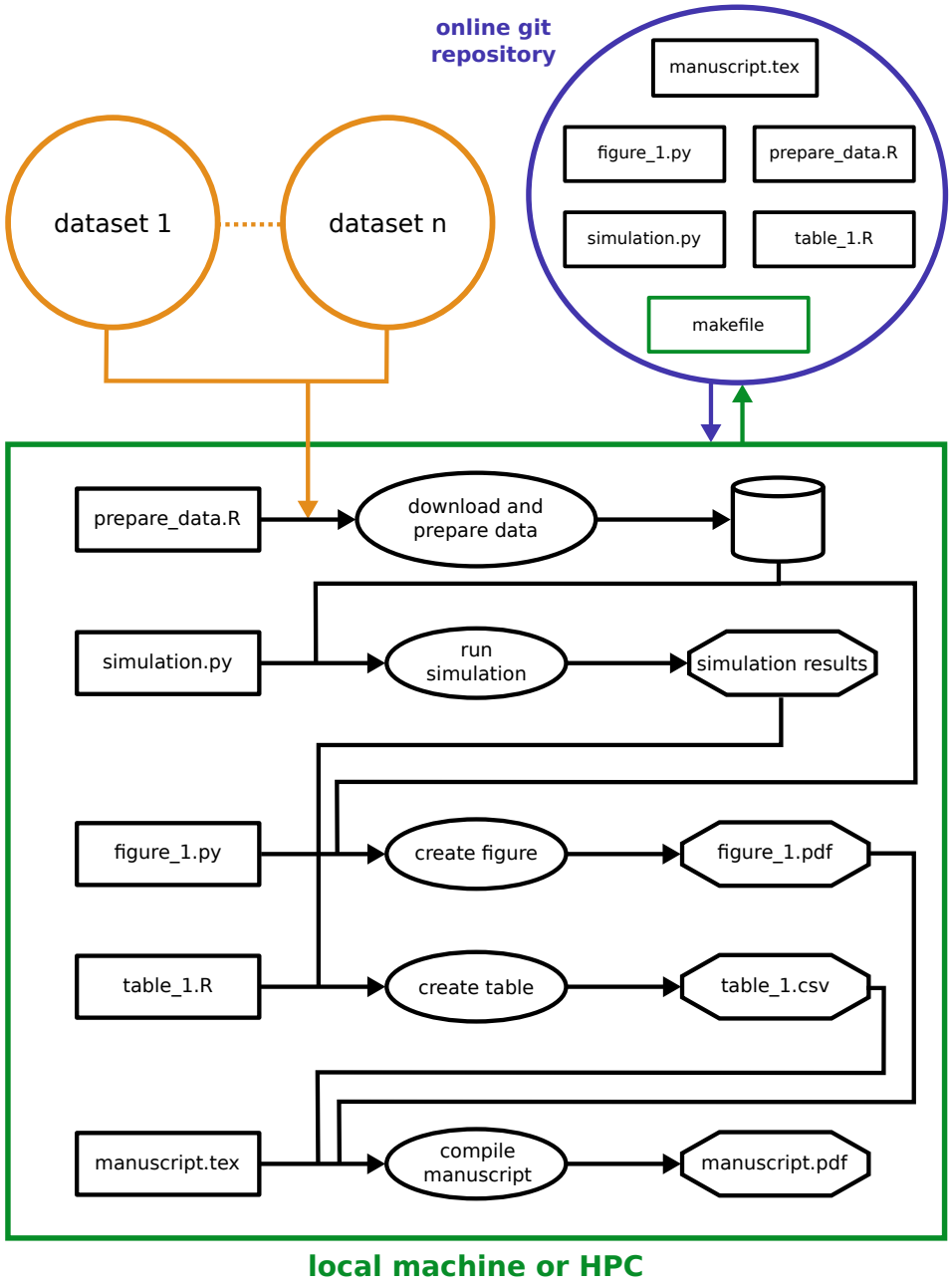

Figure 2: Overview of elements involved in a robust reproducible workflow. Orange circles represent data sources from a repository, where the data is openly accessible and citable, complies with format requirements and contains the minimum required metadata, and where the dataset license is legally adhered to during the process of data reuse. The blue circle represents all files necessary for the reproducibility of a research project, available at an online location such as a Git repository that is easily downloadable by the researcher. The repository includes scripts that retrieve and prepare the data, perform simulations and/or analysis, and create the figures used in the publication. Lastly, all results are integrated in the manuscript, which recompiles itself in the same run instant. This entire workflow is governed through a workflow engine script, a Makefile (e.g. a Snakefile in case of snakemake), which not only compiles and runs all executing scripts and outputs in the correct order, but also prepares the environments and dependencies before running the scripts. The black lines between the shapes in the green box represent the schematics of the Makefile. This analysis could be executed on a local machine or an HPC (green box). Once the Git repository is downloaded, the entire workflow should build the manuscript with one simple command: 'snakemake'.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

While a simple top-level shell script is the easiest way of defining a fairly simple workflow that does not require any of the more advanced features discussed below, shell scripting alone is not suitable as a workflow engine *lingua franca* for a wider range of projects.

4.2 Caching computationally intensive steps: make

Historically, *make* had its origins as a build system for compiling source code from statically compiled programming languages. Whilst its original aim was to help develop and maintain programmes, today it can also be used to automate the updating of any kind of inter-dependent files, which makes it relevant in the discussion of workflow engines here. *Make* also solves the issues of file or library inter-dependency and command sequencing. By tracking which files have been updated and those which have not, *make* can drastically speed up code compilation and development.

(Re-)compilation of a large program can be a computationally intensive task. Additionally, it may rely on external dependencies such as operating system libraries. When changing a subset of files, it is hard to manually keep track of which files need to be updated and which can remain unchanged. One solution is to run the entire compilation from the beginning, which wastes compilation and development time and resources. *Make* addresses these issues by introducing the notion of 'targets' (output files) which combine required inputs with a shell command to produce the desired output in the right sequence [Feldman, 1979b]. For example, the following Makefile contains two targets (a csv file and a pdf report) being produced by the same shell script (compiling an R markdown report):

```
output/report1.pdf output/table.csv: report1.Rmd data.csv
    Rscript -e "rmarkdown::render('reports/report1.Rmd')"
    mv reports/report1.pdf output/report1.pdf
    mv reports/table.csv output/table.csv

output/report2.pdf: output/table.csv
    Rscript -e "rmarkdown::render('reports/report2.Rmd')"
    mv reports/report2.pdf output/report2.pdf
```

Any of the following two commands will run the script to produce both targets

```
make output/report.pdf
make output/table.csv
```

Repeated invocation of these commands will not result in a re-compilation of the pdf report unless any of the input files changed or the output is removed.

Since *make* can run any shell script to produce a target, it is equally generic as a simple bash script but provides the additional convenience of caching. Since a Makefile follows a clear target/inputs/steps structure and dependencies are resolved automatically, it is often easier to infer the structure of a workflow from a Makefile than a simple shell script. Another advantage is that *make* can facilitate faster parallel compilation, adapted to today's mostly multicore machines. One problem of using *make* as workflow engine is the fact that outputs are not explicitly checked after executing a shell script for a target. In other words, for complex workflows it can be difficult to identify problems early since the absence of a file might only be noticed at a later point when it is used as input to a subsequent target.

4.3 Dedicated scientific workflow engines: snakemake and friends

Workflow engines geared specifically towards scientific use (primarily in bioinformatics) take a more formal approach to representing workflows. For example snakemake [Köster and Rahmann, 2012] internally represents a workflow as a directed acyclic graph (DAG) which allows inference on the optimal execution order given the available computing resources. By automatically running (conditionally) independent steps of a workflow in parallel, snakemake can easily speed up workflows with the appropriate structure (simulations, multi-model comparisons). As the name already suggests, snakemake was inspired by *make* but adds a few crucial new features. Snakemake files are written in pseudo python code, and since python is a widespread scripting language for data analysis, the syntax will be familiar to a wide range of users. Additionally, snakemake supports powerful wildcard schemes to easily generalise rules to classes of input (or output) files. Workflow parameters are easily handled through json or yml configuration files and snakemake works seamlessly with the popular cluster workload manager, slurm. This means that a workflow can be executed on a laptop and a slurm cluster without modifying the workflow itself.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

Alternatives to snakemake include nextflow and the common workflow language (CWL, the actual workflow engine is cwltool or any alternative implementation). Within limits, it is possible to automatically translate workflows between these different systems (via CWL). The primary reason for putting forward snakemake here is the low level of abstraction required by the user and the similarity with potentially familiar tools like make and the use of Python, well-known in the (bio)statistics community. Nextflow enables advanced concepts such as data-streaming via channels and is Java based. However, this also implies that workflows have to be written in Groovy which is much less known in the (bio)statistics community. CWL is also much more abstract than snakemake and requires a relatively large amount of boilerplate code to set up a workflow (making it easier to reuse components in different workflows later).

4.4 Cloud collaboration: CyVerse and Gigantum

CyVerse and Gigantum are two very different web-based solutions to the reproducibility of workflows. Their scope and type of analysis differ dramatically from each other but both have legitimate reasons to be named in this section. They both aim to reduce the complexity of software setup that a computational scientist is faced with. Both solve the issue of a portable compute context by the dockerisation of the work environment (see Section 5) and solve the issue of user friendliness through interface-based workflow building facilities. Both use cloud technology to some degree and both focus on collaborative working and data sharing, with CyVerse even providing free cloud computing resources to the users. However, the scope and type of analysis that can be made through these environments is very different, though overlaps occur.

CyVerse, together with DataStore is a big framework for integrating data and tools, running workflows and facilitating the publishing of data, workflow and results. CyVerse's particular strengths lies in what they call 'democratising access to supercomputing capabilities'. They enable small groups with restricted computing resources to run large analysis on the CyVerse HPC environment. Using a graphical interface called the DiscoveryEnvironment, the user can build and submit analysis workflows based on well-established software tools, for example a gene annotation pipeline. It is community driven, to the point that any experienced user can 'wrap' existing or newly developed apps into the CyVerse environment, write a documentation for it and share it with collaborators or the entire community. Thanks to docker, the new app will then work seamlessly on the CyVerse HPC cluster. While CyVerse can be entirely user interface based, the same outcome (an analysis workflow) can be achieved using the commandline. Workflows with crucial metadata such as the tool versions, file input and outputs can then be saved and rerun, similar to snakemake and nextflow. While RStudio or Jupyter can also be accessed through the discovery environment, the authors find that Gigantum is a better resource for collaboratively working on reproducible workflows, or in this case larger scripts of R or Python at its heart.

Gigantum is like a streamlined project framework, integrated with version control and the ability to share projects and updates with collaborators in real time. Similar to CyVerse, Gigantum 'Projects' can be integrated with datasets, in their case however, the Gigantum 'Dataset' usually hosts metadata only, instead of the actual file content. Gigantum's strengths are in interactive, collaborative and exploratory work. A local project's code and quick visualisations can be shared easily, via the Gigantum cloud and tracking features. It is a browser-based app, where workflows can be coded up on Jupyter or R notebooks. By using these notebooks, Gigantum requires the knowledge of either Python or R, and some understanding of version control concept is also a bonus, (even though Git is automated and works in the background). By focusing on R and Jupyter scripts, Gigantum seems to have a smaller scope than what is commonly understood for workflow engines such as snakemake, nextflow or CWL. However, the aspect of automatically documenting workflows (even if in this case only a single notebook), which enhances reproducibility is well catered for in Gigantum.

Both cloud services enhance collaborative work and make it easier for scientists to focus on the research without the focus on the software setups. In both cases reproducibility is enhanced by integrating with standardised datasets.

4.5 Workflow engines: no-size-fits-all

Formalising workflows is absolutely crucial for reproducible research since written descriptions of sequences of commands and steps are tedious to write, almost impossible to maintain up-to-date during the development process, and hard to test properly. A reproducible research project should therefore aim to use some form of workflow engine and formalise the workflow to produce defined outputs such as reports, figures, tables, or data sets. In doing so, the workflows are automatically documented in the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

respective workflow language. With minimal additional comments in the respective workflow definition files, these can serve as detailed, tested, and up-to-date human-readable description of a workflow.

Picking a workflow engine suitable for a particular project can be daunting. Key factors to consider are accessibility for the target audience, required features (complexity of the workflow at hand), and software requirements. Firstly, it is a good idea to use the simplest tool available to get the job done. In cases where a single, well-commented bash script suffices to run all steps of an analysis and produce the required outputs it might very well be the most portable and easy-to-understand way of defining a workflow since bash is ubiquitous. Make is a straight-forward extension of this approach whenever caching might be useful (mostly during development to avoid recompiling unchanged results). Snakemake (or similar scientific workflow engines) add tools for automatic parallel execution (potentially on a cluster) and more rigorous output checking but come at the cost of having to install the workflow engine on the executing system (requiring Python or Java). Since these dependencies are usually available on an HPC system and users can easily install these standard components via package managers on their local machines, in practice, this is a minor problem though. In any case, the required steps to set up the workflow engine and run a workflow should be prominently documented to reduce the hurdle of reproducing results for novice users. While fully integrated systems like Gigantum or CyVerse can be highly effective and take away the need to maintain computing infrastructure for individual research groups, the fact that they are cloud based requires careful analysis of data protection laws. Furthermore, such fully integrated systems require interested users to sign up for a potentially paid account. Long term availability of workflows also depends on the survival of the service provider which might be an additional risk to reproducibility. Solutions Gigantum or CyVerse which are to some degree cloud-based enable real-time collaboration, as data and workflow sharing is made easy, and documentation is ensured by workflow version tracking in the case of Gigantum.

5 Dependency Management

Section 3 discussed the benefits of source code version control. Section 4 addressed the problem of bundling together complex workflows spread across several interdependent script files using workflow engines to produce more user-friendly workflows that are reproducible by eliminating the need for manual interventions to produce a desired output. Both aspects are crucial in achieving reproducibility but fail to address the issue of managing software dependencies. To make sure that the workflow can be executed as intended, all software dependencies must be installed with exactly the same versions. The factor that complicates reproducible research the most is thus not so much the sharing of the code itself but rather the challenge to reproduce the exact same execution environment in a portable way.

Within individual scripting languages like R or Python, language-specific package managers can be used to download and install specific versions of packages (e.g. packrat and pip). Meta-package managers like Anaconda even bring together packages from multiple different languages. This approach, however, fails to isolate operating system level dependencies and relies on the continued availability of legacy package versions for download to replicate the package ecosystem at the time of the initial analysis.

A more robust way of ensuring that an instance of the exact same computing environment is preserved for future re-runs of analyses is isolation at the operating system level via virtualisation. Arguably the easiest way of achieving this is via virtual machines. Virtualisation is a tried and tested technology frequently employed in server environments to make software deployment independent from the underlying hardware it is running on. With respect to reproducible research, virtual machines have several drawbacks. Firstly, since the virtualisation is implemented on the hardware level they are relatively slow. Secondly, virtual machines are unnecessarily large since they contain an entire operating system. A more light-weight alternative to virtual machines are containers. Containers avoid virtualisation of the entire hardware and instead use the host operating system to communicate with the underlying hardware directly. This means that they tend to run faster and are smaller than virtual machines.

Using ‘recipes’, container images can be specified and built to include a snapshot of all software dependencies readily installed in a convenient image file. Since all required software is installed into the container image, the computing environment is independent of the future availability of external sources and completely self-contained. At execution time of a workflow, commands can be run within a container instance using the exact same image file as during the original analysis. Since the container image is just a single file, it can easily be stored and shared.

The most prominent containerisation software is Docker [Merkel, 2014]. Docker, however, is not ideal for isolating the software dependencies of analysis pipelines since it requires root access during execution. This is fine for workflows running on personal computers and laptops but poses a severe limitation on

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

shared resources like cloud or HPC systems where users typically do not have root access for privacy and security reasons. Singularity containers [Syslabs, nd] overcome this limitation and allows containers to run in user-space. This means that singularity container images can simply be uploaded to an HPC cluster and run without any elevated permission [2]. The flexibility to define a computing environment that can be exactly instantiated on both laptops as well as HPC systems means that users only need to learn about one technology instead of hitting a barrier for more computationally intensive tasks. It also means that it is easier to test code locally before issuing long-running tasks to the HPC system since the computing environment (container image) is exactly the same. In fact, once singularity is installed on the respective HPC, no other software needs to be installed since users can do so on their local laptops and simply upload their container images to the HPC. In this sense containers are a key element in making a research project robust since they reduce any software dependencies to the availability of the container software on the executing system and the pre-build container image (a single file) itself. At execution time, no software needs to be downloaded or installed and the analysis scripts can be run in exactly the same computational environment as during the initial analysis. Since they are also generic with regard to the type of software that can be installed (anything that can be installed on a Linux system) they also add to the 'realism' of the approach since there is no restriction to a specific set of software packages.

A sign for the success of containerised analyses workflows and singularity in particular is the fact that all major workflow engines (e.g. snakemake and nextflow) natively support executing individual tasks or entire workflows in singularity containers.

6 Archiving and Citability

6.1 Archiving

Defining a clear time window is crucial for a reproducible research project. The steps and suggestions outlined in the sections above are intended to facilitate the technical aspects of reproducibility. However, it is challenging to ensure that the technical pipeline for reproducing results will work in the long term. Even a completely containerised environment depends on support for the container software and a compatible host operating system. Additional steps should therefore be undertaken to ensure long-term availability of both code and data. When it comes to the storage of sensitive patient data, popular generalist repositories such as Zenodo may not be suitable [Banzi et al., 2019] – for example, such repositories may not have guidelines or perform checks related to de-identification of data, nor allow controlled data access. Code, and if available, containers, on the other hand are much easier to archive for a prolonged period since they should not contain any sensitive data in the first place. It is important to realise though that popular Git repository hosting services such as GitHub, GitLab or Bitbucket are not necessarily suited for long-term storage of analysis-related code (or entire container images). Despite being extremely popular and almost constituting a gold standard for code sharing, as noted by Jackson [2018], they do not create unique or persistent digital identifiers. In addition, they are not committed to providing long-term storage, and their business model and terms of use might change in the future. This has started to change recently, with initiatives like the GitHub archive program [GitHub Inc., nd], where a large selection of open source projects will be backed up in multiple forms of storage to ensure their preservation. However, these initiatives remain limited to a small portion of current projects. It is thus advisable to back up important revisions/releases of publication-related repositories to a dedicated long-term storage solution.

As already mentioned in Section 2, the choice of repository is critical in ensuring that both data and code can be accessed in the future. In the last decade, a large number of repositories have emerged, including general purpose repositories (such as Zenodo, Figshare, Dryad, Mendeley Data and Dataverse) and discipline-specific repositories (such as the Worldwide Protein Data Bank, UniprotKB and Genbank) which accept only a restricted number of data types and formats. Further subject-specific data repositories can be found at Nature Scientific Data policies for example [Nature Scientific Data, nd]. Repositories also exist specifically for software, with initiatives such as the Software Heritage project (`https://www.softwareheritage.org/`) where free/open source software can be uploaded and preserved for long-term storage [Di Cosmo et al., 2019]. Alternatively, there are language-specific repositories such as CRAN and Bioconductor for R, PyPI for Python, CPAN for Perl, and Maven for Java-based projects. Table 3 gives some advantages and disadvantages of different archiving locations.

This heterogeneous landscape offers several opportunities but also introduces challenges in effectively

[2] Note that root access is still required to build a container but this can be done on a standard user-administered laptop

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

Type	Archiving location	Pros	Cons
Code/software	Git repositories (GitHub, GitLab, Bitbucket)	continuous, version-tracked improvements	no DOI (but see below), no commitment to long-term storage
	General purpose repositories (Zenodo, Dryad, etc.)	DOI, stable version, some have integration with Git repositories	Specific code is less findable, and may not be up-to-date
Data	Discipline-specific repos (ENA, ArrayExpress, etc.)	DOI, standardised, often APIs in place to push/pull data, minimal metadata must be provided	Only specific data types or formats may be supported
	General purpose repositories (Zenodo, Dryad, etc.)	DOI, wide range of data types/formats supported	Specific data content is less findable

Table 3: Pros and cons for code and data archiving in different archiving locations. DOI = digital object identifier.

finding the most appropriate repositories for either storing or accessing software and data. This is where Re3data represents a valuable resource, having registered more than 2000 research data repositories so far, which can be searched (e.g. by name, discipline, content type or country) to identify the most appropriate data repositories [Kindling, 2017]. Another useful resource is FAIRsharing (`https://fairsharing.org/`), which describes and interlinks data standards, databases and data policies [Sansone et al., 2019]. The information collected ensures that the FAIR data principles are followed, and allows users to easily search for repositories that fit with funder or journal data policies. Section 2 gives further factors that should be considered when choosing a research repository, including whether long-term data preservation is offered. Despite these resources, the large number of available repositories still risks limiting the ability of researchers to easily access or deposit data from the right location. In our opinion, more focus should be placed on developing standards for field-specific databases rather then starting redundant repositories. An important initiative to tackle this issue is the ELIXIR-Europe project, which aims to create a single infrastructure to provide easy access to life science resources from across Europe [ELIXIR, nd]. The project comprises of different platforms, including the data platform and the interoperability platform. The data platform aims to provide users with robust, long-term sustainable data resources within a coordinated data ecosystem. This includes Literature-Data integration, to streamline the process of moving from a scientific article to its underlying datasets. Meanwhile, the interoperability platform aims to establish standardised file formats, metadata, vocabularies and identifiers for the life-science community. The platform will also lead to connections between data resources, and help create the infrastructure needed to integrate data from disparate resources.

6.2 Citability

A closely related issue to archiving code and data is ensuring their citability. As formalised by the FORCE11 declaration of data citation [FORCE 11 Data Citation Synthesis Group, 2014] and principles of software citation [Smith et al., 2016], both data and software should be considered legitimate and citeable products of research. This requires unique and specific identification, as well as metadata, that persists even beyond the planned lifespan of research project. Recognition of data and software is also one of the elements of DORA, the San Francisco Declaration of Research Assessment, which aims to promote change in research assessment via commitment from funders, publishers, institutions and individuals [DORA Program, nd]. As already mentioned, archiving data in appropriate repositories and following the FAIR principles [Wilkinson et al., 2016] are critical for citability and visibility. The FAIR principles should also be followed for code and software, but their application may require some revisions and elaboration to account for the specificities of software, see Lamprecht et al. [2019] for further details. Initiatives such as these are enabling data and software to be recognised in their own right, instead of being completely overshadowed by journal articles and associated metrics.

Digital Object Identifiers (DOIs) are one key way to provide unique and persistent identification, as well as specificity in terms of the version of data or software that was used. For code in particular, it is important to note that some commonly-used general-purposed repositories can provide DOIs. For example, Github accounts can be linked to Zenodo or Figshare, so that any new release on the GitHub repository will automatically be archived with a corresponding DOI. The automatic creation of release-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

specific DOIs facilitates the discoverability of the repositories, as they can then be cited in a robust way, even if the underlying content is moved to a different infrastructure in the future. Metadata is also critical for enabling visibility, as previously mentioned for datasets in Section 2. Metadata standards have also being developed for code: formats such as the CodeMeta Project [CodeMeta contributors, nd] and the Citation File Format [Druskat et al., 2019] assist accessibility, and allow for proper credit and attribution. A recent development that facilitates citation of complex projects and workflows is the tools platform of the ELIXIR-EUROPE project, which has a specific focus on software. As an example, bio.tools (`https://bio.tools/`) aims to provide a comprehensive registry of both software and databases [Ison et al., 2015]. This includes complex, multi-functional analysis workflows. Each resource is given a human-readable, unique identifier, which provide a persistent reference to essential information. Another recent development is the Open Science Framework (OSF – `https://osf.io/`). This expands on the integrations mentioned above by offering a project management layer and many add-ons [Foster and Deardorff, 2017]. The OSF can connect to many disparate services, allowing a researcher to share one link to a project space that contains code, repository data, citations and documentation. The OSF also allows for multiple contributors (with different permission levels) and each user, component, project, and file is given a unique URL. Public projects are given DOIs with archival resource keys, and there is an in-built version control system.

7 Outlook

7.1 The last mile: pre-prints and seamless publication services

In practice, it is often surprisingly difficult to format a paper for submission to a journal since many medical journals still require manuscripts to be submitted as .docx files which makes a seamless reproducible research process almost impossible and still requires extensive manual editing.

A promising solution to this problem is the tight integration of online LaTeX editors such as Overleaf together with GitHub, and finally with pre-print services. Overleaf [2012] takes away the complexity of setting up a LaTeX environment and provides a rich text view to facilitate editing LaTeX source files for collaborators not familiar with this complex but highly professional typesetting tool. The integration with GitHub allows the analysis code to be stored in the same repository as the final manuscript file with the only downside of having to store binary figure output files in the Git repository. Several pre-print services already allow submission directly from Overleaf (e.g. *Biostatstics*) and some even organise their peer review process on Overleaf. Since pre-printing services generally rely on the submitting authors to do the final layout and formatting, the growing ecosystem of pre-print services could provide an interesting solution for bridging that last gap between reproducible analysis and publication. Authors would initially write and publish to a pre-print service in a completely reproducible manner before considering submitting the manuscript to a traditional journal. Besides smoothing the publication process, this approach has the added advantage that all research is publicly available as a pre-print (cf. the FAIR principles). With the rise of bioRxiv and, more recently, medRxiv, the only reason not to publish biostatistics related research output as pre-print first is the risk of traditional journals rejecting acceptance of a manuscript due to copyright issues or restrictive pre-print policies. However, this risk is quickly disappearing as most journals readily accept submissions of work that are already available as a pre-print, as long as the pre-print was released before the submission. Sherpa Romeo (`https://v2.sherpa.ac.uk/romeo/`) provides useful summaries of pre-print policies on a journal-by-journal basis [Jisc, nd].

7.2 CI/CD

Definition: CI/CD

A **Continuous Integration** (CI) and **Continuous Deployment** (CD) allow users to automatically execute, test and generate outputs from several code modules. It helps to ensure the reproducibility, portability and overall robustness of research results, and facilitates dissemination.

The main advantage of CI is to prevent any "regression" of research software, i.e. an update (e.g. following a revision of a research manuscript) that would (inadvertently) break its previous capabilities. A good way to prevent such events is to use unit tests in software development to check all important capabilities and results from each single module (=unit) of the software. The added value of Continuous Integration (CI) systems (e.g. Jenkins, Travis, GitLab CI, GitHub Actions, etc.) is to automate these checks by running such unit tests automatically and checking for correct integration among modules

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 June 2020 doi:10.20944/preprints202006.0002.v1

on various platforms every time the code is updated (e.g. combined with version control tools ensuring prospective changes can be safely merged) or at regular time intervals (every day for instance). Those tools help ensure research software is portable, and that their results are reproducible and robust.

One immediate drawback of such tools is the additional overhead needed to maintain and fix the automated checks on the CI platform as software and infrastructure updates. But the benefit is the insurance that the software is portable to tested systems and setups. Linked to CI, CD can be very useful for reproducible research: it ensures that results directly are always up to date, generated from up to date software. Put together CI and CD can be useful tools to ensure that is research software is portable, and that its results remains reproducible when dealing with dependencies or software updates.

8 Summary

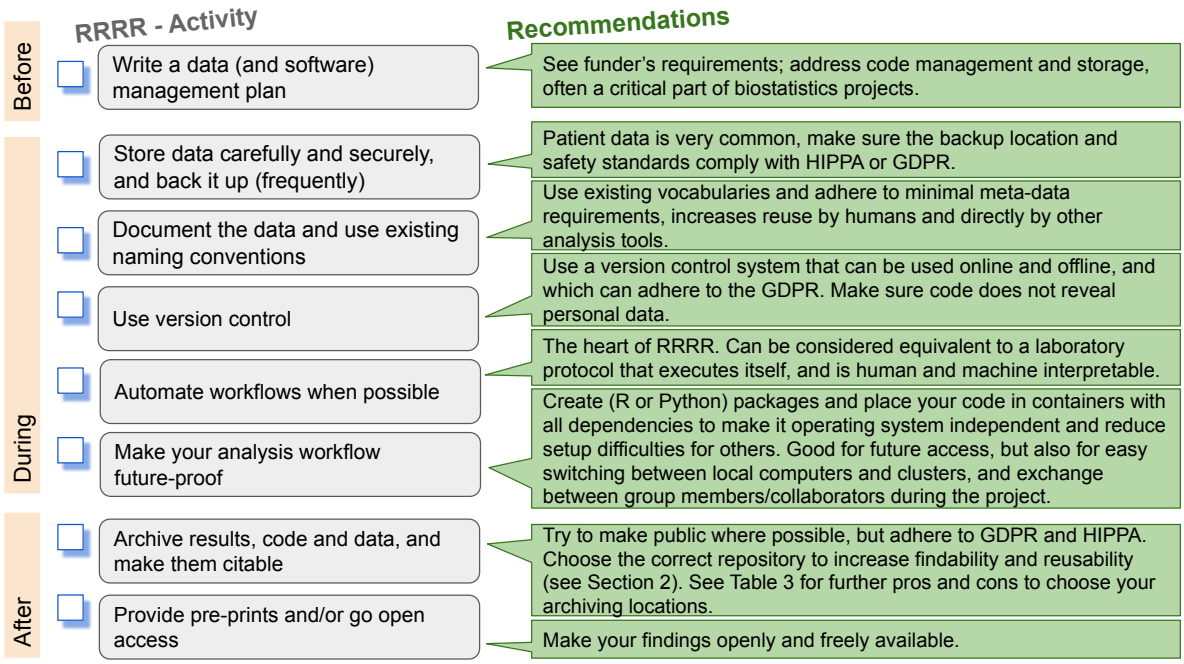

Figure 3: Summary of the different steps realistically involved when performing robust reproducible research that abide by the FAIR principles, with an emphasis on Biostatistics.

Biostatisticians face a multitude of hurdles when wanting to perform realistic, robust and reproducible research. Challenges occur at each stage of the research cycle and involve all aspects of research from data protection and management to analysis workflow creation and source code version control. Furthermore, there is the required maintenance and management of software dependencies used in the analysis and, ultimately, the archiving and citing of workflows and data, and seamless publication of research papers. While there are now a plethora of technologies and platforms available to help solving the above issues, the diversity of solutions creates a new challenge: the choice of the set of tools and platforms to perform research with. While following the recommendations in Figure 3 may seem time consuming at first, we argue that research on the whole will be sped up and increase in credibility. Nevertheless, time for robust reproducible research should be realistically accounted for in project proposal budgets (some might argue to go so far as Raphael et al. [2020]).

In this publication we have given a brief overview of the issues Biostatisticians commonly face in their day-to-day research activities, which impede them from performing realistic, robust and reproducible research. We also attempted to outline some practical approaches and guidance to finding solutions for performing realistic and robust reproducible research and give an outlook over existing services that allow for the seamless publication of research products such as data, workflows and publications. Finally, we stress that reproducibility is a continuum, an asymptotic ideal to yearn for. Reproducibility is not all or nothing, and every step along the way is a step in the right direction.

Acknowledgments

The authors thank Lauren Cadwallader, Mark J. Williamson and Jessica Minnier for feedback on the manuscript.

Conflict of Interest

None declared.

Funding

EL acknowledges the Engineering and Physical Science Research Council for financial support, AH is funded by the Engineering and Physical Science Research Council and GlaxoSmithKline, DSR is funded by the Biometrika Trust and Medical Research Council (grant code MC_UU_00002/6).

Author Contribution

KK conceived the work, planned and defined the scope of the paper. BPH and KK wrote the introduction. BPH, SCJ and KK wrote the data management section. AH, DSR and KK wrote the section on version control. KK and AHE-S wrote the section on workflow automation. EL produced Figure 2. AHE-S and BPH produced Figure 3. KK wrote the section on dependency management. DSR, EL, SCJ and KK wrote the section on archiving and citability. BHP and KK wrote the outlook section. AHE-S wrote the summary section. All authors discussed ideas, provided feedback, edited the manuscript draft, and approved the manuscript for submission.

References

John P A Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):e124, 2005. doi: 10.1371/journal.pmed.0020124. https://doi.org/10.1371/journal.pmed.0020124.

John P A Ioannidis. How to Make More Published Research True. *PLoS Medicine*, 11(10):e1001747, 2014. doi: 10.1371/journal.pmed.1001747. https://doi.org/10.1371/journal.pmed.1001747.

Monya Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454, 2016. doi: 10.1038/533452a. https://doi.org/10.1038/533452a.

National Academies of Sciences, Engineering, and Medicine. *Reproducibility and Replicability in Science*. The National Academies Press, Washington, DC, 2019. ISBN 978-0-309-48613-2. doi: 10.17226/25303. URL `https://www.nap.edu/catalog/25303/reproducibility-and-replicability-in-science`.

The Turing Way Community, Becky Arnold, Louise Bowler, Sarah Gibson, Herterich, Rosie Higman, Anna Krystalli, Alexander Morley, Martin O'Reilly, and Kirstie Whitaker. *The Turing Way: A Handbook for Reproducible Data Science*. Zenodo. http://doi.org/10.5281/zenodo.3233986, 2019. doi: 10.5281/zenodo.3233986.

Lorena A. Barba. Terminologies for Reproducible Research. *arXiv:1802.03311*, 2018.

The Turing Way Community and Scriberia. Illustrations from the turing way book dashes, 2020. URL `https://doi.org/10.5281/zenodo.3332808`. https://doi.org/10.5281/zenodo.3695300.

Directorate-General for Research and Innovation (European Commission), Maarja Adoojan, Victoria Tsoukala, and Jean-François Dechamp. Access to and Preservation of Scientific Information in Europe, 2018. https://doi.org/10.2777/642887.

OECD. *Making Open Science a Reality*. Number 25 in OECD Science, Technology and Industry Policy Papers. OECD Publishing, Paris, 2015. doi: 10.1787/5jrs2f963zs1-en. https://doi.org/10.1787/5jrs2f963zs1-en.

European Commission. General Data Protection Regulation (GDPR) – Final Text. https://gdpr-info.eu/, 2018. URL https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf.

Public Law 104-191. Health Insurance Portability and Accountability Act (HIPAA) of 1996, 1996.

Gigantum Funding Team, Randal Burns, Dean Kleissas, Jacob Vogelstein, Joshua Vogelstein, and Tyler Whitehouse. Gigantum – a simple way to create and share reproducible data science and research. *eLife Labs, https://elifesciences.org/labs/bdbeac92/gigantum-a-simple-way-to-create-and-share-reproducible-data-science-and-research*, 2018.

William Michael Landau. The drake R package: a pipeline toolkit for reproducibility and high-performance computing. *Journal of Open Source Software*, 3(21), 2018. doi: 10.21105/joss.00550. https://doi.org/10.21105/joss.00550.

Anaconda Inc. Conda. https://conda.io/en/latest/, nd. URL https://conda.io. Accessed 2020-05-14.

Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

Jupyter Team. Jupyter notebooks. https://jupyter.org, nd. URL https://jupyter.org.

UKRI. Guidance on best practice in the management of research data. https://www.ukri.org/files/legacy/documents/rcukcommonprinciplesondatapolicy-pdf/, 2015. Accessed 2020-04-20.

European Commission. Directorate-General for Research & Innovation. Programme Guidelines on FAIR Data Management in Horizon 2020. https://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf, 2016. Version 3.0. Accessed 2020-04-20.

Digital Curation Centre. DMP Online. https://dmponline.dcc.ac.uk, nd. Accessed 2020-04-20.

Jackson, Michael (ed.). Checklist for a Software Management Plan. Zenodo. doi:10.5281/zenodo.2159713. Web site: https://www.software.ac.uk/software-management-plan, 2018. Version 1.0.

Peter E. Midford, Thomas Alex Dececchi, James P. Balhoff, Wasila M. Dahdul, Nizar Ibrahim, Hilmar Lapp, John G. Lundberg, Paula M. Mabee, Paul C. Sereno, Monte Westerfield, Todd J. Vision, and David C. Blackburn. The vertebrate taxonomy ontology: a framework for reasoning across model organism and species phenotypes. *Journal of Biomedical Semantics*, 4(1):1–6, December 2013. ISSN 2041-1480. doi: 10.1186/2041-1480-4-34. URL http://jbiomedsem.biomedcentral.com/articles/10.1186/2041-1480-4-34. https://doi.org/10.1186/2041-1480-4-34.

Chris F. Taylor, Dawn Field, Susanna-Assunta Sansone, Jan Aerts, Rolf Apweiler, Michael Ashburner, Catherine A. Ball, Pierre-Alain Binz, Molly Bogue, Tim Booth, Alvis Brazma, Ryan R. Brinkman, Adam Michael Clark, Eric W. Deutsch, Oliver Fiehn, Jennifer Fostel, Peter Ghazal, Frank Gibson, Tanya Gray, Graeme Grimes, John M. Hancock, Nigel W. Hardy, Henning Hermjakob, Randall K. Julian, Matthew Kane, Carsten Kettner, Christopher Kinsinger, Eugene Kolker, Martin Kuiper, Nicolas Le Novère, Jim Leebens-Mack, Suzanna E. Lewis, Phillip Lord, Ann-Marie Mallon, Nishanth Marthandan, Hiroshi Masuya, Ruth McNally, Alexander Mehrle, Norman Morrison, Sandra Orchard, John Quackenbush, James M. Reecy, Donald G. Robertson, Philippe Rocca-Serra, Henry Rodriguez, Heiko Rosenfelder, Javier Santoyo-Lopez, Richard H. Scheuermann, Daniel Schober, Barry Smith, Jason Snape, Christian J. Stoeckert, Keith Tipton, Peter Sterk, Andreas Untergasser, Jo Vandesompele, and Stefan Wiemann. Promoting coherent minimum reporting guidelines for biological and biomedical investigations: the MIBBI project. *Nature Biotechnology*, 26(8), August 2008. ISSN 1546-1696. doi: 10.1038/nbt.1411. URL http://www.nature.com/articles/nbt.1411. https://doi.org/10.1038/nbt.1411.

Saminda Abeyruwan, Uma D Vempati, Hande Küçük-McGinty, Ubbo Visser, Amar Koleti, Ahsan Mir, Kunie Sakurai, Caty Chung, Joshua A Bittker, Paul A Clemons, Steve Brudz, Anosha Siripala, Arturo J Morales, Martin Romacker, David Twomey, Svetlana Bureeva, Vance Lemmon, and Stephan C Schürer. Evolving BioAssay Ontology (BAO): modularization, integration and applications. *Journal of Biomedical Semantics*, 5(Suppl 1):S5, June 2014. ISSN 2041-1480. doi: 10.1186/2041-1480-5-S1-S5. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4108877/. https://doi.org/10.1186/2041-1480-5-S1-S5.

EMBL-EBI. Ontology Lookup Service. https://www.ebi.ac.uk/ols/ontologies, nd. Accessed 2020-05-14.

Elizabeth K. Nelson, Britt Piehler, Josh Eckels, Adam Rauch, Matthew Bellew, Peter Hussey, Sarah Ramsay, Cory Nathe, Karl Lum, Kevin Krouse, David Stearns, Brian Connolly, Tom Skillman, and Mark Igra. LabKey Server: An open source platform for scientific data integration, analysis and collaboration. *BMC Bioinformatics*, 12(1):71, March 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-71. https://doi.org/10.1186/1471-2105-12-71.

Anthony Etuk, Felix Shaw, Alejandra Gonzalez-Beltran, David Johnson, Marie-Angélique Laporte, Philippe Rocca-Serra, Elizabeth Arnaud, Medha Devare, Paul J. Kersey, Susanna-Assunta Sansone, and Robert P. Davey. COPO: a metadata platform for brokering FAIR data in the life sciences. *bioRxiv*, September 2019. doi: 10.1101/782771. URL `https://www.biorxiv.org/content/10.1101/782771v1`. https://doi.org/10.1101/782771.

Research Data Management Service Group, Cornell University. Guide to writing "readme" style metadata. https://data.research.cornell.edu/content/readme, nd. Accessed 2020-04-20.

Core Trust Seal. About Core Trust Seal. https://www.coretrustseal.org/about, nd. Accessed 2020-04-20.

re3data.org. re3data.org – Registry of Research Data Repositories. https://www.re3data.org/, nd. Accessed 2020-04-20, doi:10.17616/R3D.

H. Pample, P. Vierkant, F. Scholze, R. Bertelmann, M. Kindling, J. Klump, H-J. Goebelbecker, J. Gundlach, P. Schirmbacher, and U. Dierolf. Making research data repositories visible: The re3data.org registry. *PLoS ONE*, 8(11):e78080, 2013. doi: 10.1371/journal.pone.0078080. https://doi.org/10.1371/journal.pone.0078080.

OpenAIRE. Guide for researchers: how to select a data repository. https://www.openaire.eu/opendatapilot-repository-guide, nd. Accessed 2020-04-20.

Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan Van Der Lei, Erik Van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 2016. ISSN 2052-4463. doi: 10.1038/sdata.2016.18. https://doi.org/10.1038/sdata.2016.18.

Martin Boeckhout, Gerhard A. Zielhuis, and Annelien L. Bredenoord. The fair guiding principles for data stewardship: fair enough? *European Journal of Human Genetics*, 26:931–936, 2018. doi: 10.1038/s41431-018-0160-0. https://doi.org/10.1038/s41431-018-0160-0.

Sandra Collins, François Genova, Natalie Harrower, Simon Hodson, Sarah Jones, Leif Laasonen, Daniel Mietchen, Ruta Petrauskaite, and Peter Wittenburg. *Turning FAIR into reality: Final report and action plan on FAIR data*. European Commission, Directorate General for Research Innovation, Brussels, 2018. doi: 10.2777/1524. https://doi.org/10.2777/1524.

Susanna-Assunta Sansone, Peter McQuilton, Philippe Rocca-Serra, Alejandra Gonzalez-Beltran, Massimiliano Izzo, Allyson L Lister, and Milo Thurston. Fairsharing as a community approach to standards, repositories and policies. *Nature Biotechnology*, 37(4):358, 2019. https://doi.org/10.1038/s41587-019-0080-8.

FOSTER Open Science. Assessing the FAIRness of data. https://www.fosteropenscience.eu/learning/assessing-the-fairness-of-data, nd. Accessed 2020-04-20.

Rita Banzi, Steve Canham, Wolfgang Kuchinke, Karmela Krleza-Jeric, Jacques Demotes-Mainard, and Christian Ohmann. Evaluation of repositories for sharing individual-participant data from clinical studies. *Trials*, 20(1):169, 2019.

Christian Ohmann, Rita Banzi, Steve Canham, Serena Battaglia, Mihaela Matei, Christopher Ariyo, Lauren Becnel, Barbara Bierer, Sarion Bowers, Luca Clivio, Monica Dias, Christiane Druml, Hélène Faure, Martin Fenner, Jose Galvez, Davina Ghersi, Christian Gluud, Trish Groves, Paul Houston, Ghassan Karam, Dipak Kalra, Rachel L Knowles, Karmela Krleža-Jerić, Christine Kubiak, Wolfgang Kuchinke, Rebecca Kush, Ari Lukkarinen, Pedro Silverio Marques, Andrew Newbigging, Jennifer O'Callaghan, Philippe Ravaud, Irene Schlünder, Daniel Shanahan, Helmut Sitter, Dylan Spalding, Catrin Tudur-Smith, Peter van Reusel, Evert-Ben van Veen, Gerben Rienk Visser, Julia Wilson, and Jacques Demotes-Mainard. Sharing and reuse of individual participant data from clinical trials: principles and recommendations. *BMJ Open*, 7(12), 2017. doi: 10.1136/bmjopen-2017-018647. URL `https://bmjopen.bmj.com/content/7/12/e018647`. https://doi.org/10.1136/bmjopen-2017-018647.

Scott Chacon and Ben Straub. *Pro Git*. Apress, Berkely, CA, USA, 2nd edition, 2014. ISBN 1484200772, 9781484200773.

The Apache Software Foundation. Apache Subversion. https://subversion.apache.org/, nd. Accessed 2020-05-14.

Mercurial community. Mercurial SCM. https://www.mercurial-scm.org/, nd. Accessed 2020-05-14.

Git community. Git. https://git-scm.com/, nd. Accessed 2020-05-14.

Karthik Ram. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, 8(1):7, 2013. ISSN 1751-0473. doi: 10.1186/1751-0473-8-7. URL `https://doi.org/10.1186/1751-0473-8-7`. https://doi.org/10.1186/1751-0473-8-7.

Jennifer Bryan. Excuse me, do you have a moment to talk about version control? *PeerJ Preprints*, 5: 3159v2, 2017. doi: 10.7287/peerj.preprints.3159v2. https://doi.org/10.7287/peerj.preprints.3159v2.

David Herron. Why Git and Git-LFS is not enough to solve the machine learning reproducibility crisis. https://web.archive.org/web/20190503003656/https://towardsdatascience.com/why-git-and-git-lfs-is-not-enough-to-solve-the-machine-learning-reproducibility-crisis-f733b49e96e8?gi=ec1787bd9ed6, 2019. Accessed: 2019-08-22.

DVC community. DVC. https://dvc.org/, nd. Accessed 2020-05-14.

Stuart I Feldman. Make – A Program for Maintaining Computer Programs. In *UNIX Programmer's Manual*, volume 2A. Bell Telephone Laboratories, 7$^{\text{th}}$ edition, 1979a.

Free Software Foundation. GNU Make Manual. https://www.gnu.org/software/make/manual/, May 2016. URL `https://www.gnu.org/software/make/manual/`.

Free Software Foundation. GNU Bash Manual. https://www.gnu.org/software/bash/manual/, May 2019. URL `https://www.gnu.org/software/bash/manual/`.

S. I. Feldman. Make – a program for maintaining computer programs. *Bell Laboratories*, 9:255–265, 1979b.

Johannes Köster and Sven Rahmann. Snakemake - a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012. doi: 10.1093/bioinformatics/bts480. https://doi.org/10.1093/bioinformatics/bts480.

Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239), March 2014. ISSN 1075-3583.

Syslabs. Singularity. https://sylabs.io/singularity/, nd. Accessed 2020-05-14.

Michael Jackson. Software Deposit: Where to deposit software (Version 1.0). Zenodo. doi:10.5281/zenodo.1327329, 2018.

GitHub Inc. GitHub Archive Program. https://archiveprogram.github.com/, nd. Accessed 2020-05-14.

Nature Scientific Data. Recommended Data Repositories. https://www.nature.com/sdata/policies/repositories, nd. Accessed 2020-05-14.

R. Di Cosmo, R. Gruenpeter, and S. Zacchiroli. 204.4 identifiers for digital objects: The case of software source code preservation. Jun 2019. doi: 10.17605/OSF.IO/KDE56. https://doi.org/10.17605/OSF.IO/KDE56.

Maxi Kindling. The landscape of research data repositories in 2015: A re3data analysis. *D-Lib Magazine*, 23(3):4, 2017. https://doi.org/10.1045/march2017-kindling.

ELIXIR. ELIXIR. https://elixir-europe.org/, nd. Accessed 2020-05-14.

FORCE 11 Data Citation Synthesis Group. Joint declaration of data citation principles. https://doi.org/10.25490/a97f-egyk, 2014. URL `https://doi.org/10.25490/a97f-egyk`.

Arfon M Smith, Daniel S Katz, and Kyle E Niemeyer. Software citation principles. *PeerJ Computer Science*, 2:e86, 2016. https://doi.org/10.7717/peerj-cs.86.

DORA Program. DORA - San Francisco Declaration on Research Assessment. https://sfdora.org/, nd. Accessed 2020-05-14.

Anna-Lena Lamprecht, Leyla Garcia, Mateusz Kuzak, Carlos Martinez, Ricardo Arcila, Eva Martin Del Pico, Victoria Dominguez Del Angel, Stephanie van de Sandt, Jon Ison, Paula Andrea Martinez, Peter McQuilton, Alfonso Valencia, Jennifer Harrow, Fotis Psomopoulos, Josep Ll. Gelpi, Neil Chue Hong, Carole Goble, and Salvador Capella-Gutierrez. Towards FAIR principles for research software. *Data Science*, Pre-press:1–23, 2019. doi: 10.3233/DS-190026. https://doi.org/10.3233/DS-190026.

CodeMeta contributors. The CodeMeta Project. https://codemeta.github.io/, nd. Accessed 2020-05-14.

Stephan Druskat, Jurriaan Spaaks, Neil Chue Hong, Robert Haines, and James Baker. Citation File Format (CFF). https://citation-file-format.github.io/about/, 2019. doi:10.5281/zenodo.3515946. Accessed 2020-05-14.

Jon Ison, Kristoffer Rapacki, Hervé Ménager, Matúš Kalaš, Emil Rydza, Piotr Chmura, Christian Anthon, Niall Beard, Karel Berka, Dan Bolser, Tim Booth, Anthony Bretaudeau, Jan Brezovsky, Rita Casadio, Gianni Cesareni, Frederik Coppens, Michael Cornell, Gianmauro Cuccuru, Kristian Davidsen, Gianluca Della Vedova, Tunca Dogan, Olivia Doppelt-Azeroual, Laura Emery, Elisabeth Gasteiger, Thomas Gatter, Tatyana Goldberg, Marie Grosjean, Björn Grüning, Manuela Helmer-Citterich, Hans Ienasescu, Vassilios Ioannidis, Martin Closter Jespersen, Rafael Jimenez, Nick Juty, Peter Juvan, Maximilian Koch, Camille Laibe, Jing-Woei Li, Luana Licata, Fabien Mareuil, Ivan Mičetić, Rune Møllegaard Friborg, Sebastien Moretti, Chris Morris, Steffen Möller, Aleksandra Nenadic, Hedi Peterson, Giuseppe Profiti, Peter Rice, Paolo Romano, Paola Roncaglia, Rabie Saidi, Andrea Schafferhans, Veit Schwämmle, Callum Smith, Maria Maddalena Sperotto, Heinz Stockinger, Radka Svobodová Vařeková, Silvio C.E. Tosatto, Victor de la Torre, Paolo Uva, Allegra Via, Guy Yachdav, Federico Zambelli, Gert Vriend, Burkhard Rost, Helen Parkinson, Peter Løngreen, and Søren Brunak. Tools and data services registry: a community effort to document bioinformatics resources. *Nucleic Acids Research*, 44:D38–D47, 2015. https://doi.org/10.1093/nar/gkv1116.

Erin D. Foster and Ariel Deardorff. Open science framework (OSF). *Journal of the Medical Library Association*, 105(2), April 2017. doi: 10.5195/jmla.2017.88. https://doi.org/10.5195/jmla.2017.88.

Overleaf. Online latex editor. Available at www.overleaf.com, 2012. (accessed 22 March 2020).

Jisc. Sherpa Romeo. https://v2.sherpa.ac.uk/romeo/, nd. Accessed 2020-05-14.

Marc P. Raphael, Paul E. Sheehan, and Gary J. Vora. A controlled trial for reproducibility. *Nature*, 579(7798):190–192, March 2020. doi: 10.1038/d41586-020-00672-7. URL `https://www.nature.com/articles/d41586-020-00672-7`. https://doi.org/10.1038/d41586-020-00672-7.