

Teaching research data management with DataLad: a multi-year, multi-domain effort

Michał Szczepanik^{*,†,1}, Adina S. Wagner^{*,2}, Stephan Heunis¹, Laura K. Waite¹,
Simon B. Eickhof^{2,3}, Michael Hanke^{1,3}

[†] m.szczepanik@fz-juelich.de ^{*} Both authors have contributed equally

¹ Psychoinformatics lab, Institute of Neuroscience and Medicine, Research Centre Jülich, Germany

² Cognitive and affective biopsychology lab, Institute of Neuroscience and Medicine, Research Centre Jülich, Germany

³ Institute of Systems Neuroscience, Medical Faculty, Heinrich Heine University Düsseldorf, Germany

Abstract

Research data management has become an indispensable skill in modern neuroscience. Researchers can benefit from following good practices as well as from having proficiency in using particular software solutions. But as these domain-agnostic skills are commonly not included in domain-specific graduate education, community efforts increasingly provide early career scientists with opportunities for organised training and materials for self-study. Investing effort in user documentation and interacting with the user base can, in turn, help developers improve quality of their software. In this work, we detail and evaluate our multi-modal teaching approach to research data management in the DataLad ecosystem, both in general and with concrete software use. Spanning an online and printed handbook, a modular course suitable for in-person and virtual teaching, and a flexible collection of research data management tips in a knowledge base, our free and open source collection of training material has made research data management and software training available to various different stakeholders over the past five years.

Introduction

Neuroscience, as well as science in general, is incremental and its outcomes go beyond journal articles – code, data, results, or tools of previously finished or unfinished projects are reused or extended in later activities (Mons, 2018), and the people who start and finish a project are not necessarily the same (Puce & Hämäläinen, 2017). Science is also increasingly collaborative and spatially distributed (Csomós et al., 2020). Benefiting from differential expertise of several collaborators requires mutual understanding of a project and its components. Over several decades already, data shared across institutions or even openly has been a cornerstone of neuroscientific progress (e.g., Ferguson et al., 2014; Niso et al., 2016), and this trend is increasing (Gorgolewski & Poldrack, 2016). A common challenge is thus to keep projects in such a state that they can be understood without the expertise of their original authors (Puce & Hämäläinen, 2017). This makes technical and organizational elements of scientific practice, such as research data management (RDM) a fundamental prerequisite.

While experts in their respective domains and methodologies, scientists may not have domain-agnostic technical skills. Task sets such as those found in RDM, however, require a broad set of technical skills and research curricula seldom teach computing ecosystem literacy (Grisham et al., 2016). In fact, even computer science curricula often miss critical topics about the computing ecosystem. At the Massachusetts Institute of Technology (MIT), USA, this lack famously resulted in the internationally popular, self-organized class, “The missing semester of your CS education”¹. In addition, the high usability of modern computers’ and applications’ front ends spares users the need to develop the same level of familiarity with their computers that previous generations of computer users had (Mehlenbacher, 2003). This makes general technical skill and RDM training a crucial element in preparing the next generation of neuroscientists.

In an ongoing multi-modal, multi-year effort, we combined various interconnected activities into a comprehensive RDM training centered around the software tool DataLad [datalad.org](https://data-lad.org); Halchenko et al., 2021. These activities spanned a community-led online RDM handbook with a printed paperback option and knowledge base, a matching online RDM course, and various workshops. In this reflective piece, we evaluate this teaching ecosystem, review its advantages and shortcomings, and share lessons-learned over its 5-year long history.

DataLad

DataLad is a Python-based, MIT-licensed software tool for the joint management of code, data, and their relationship. It builds up on git-annex, a versatile system for data logistics (Hess, 2010), and Git, the industry standard for distributed version control. To address the technical challenges of data management, data sharing, and digital provenance collection, it adapts principles of open-source software development and distribution to scientific workflows. Like Git and git-annex, DataLad’s primary interface is the command line. This makes familiarity with the computer terminal, common file system operations, and general knowledge about one’s operating system beneficial for software users.

Overarching goals for training materials

Our training material aims to provide even technical novices with the opportunity to use the software quickly, productively, and to easily integrate with other tools and services in real-world research. In part, it was motivated by concrete user needs, such as early career researchers in a research consortium. Beyond this, we aimed for the training material to be fully open source, accessible (both regarding language and technical requirements), flexible, multi-modal (everyone should find something that fits their learning needs), directly applicable to various research contexts, and maintainable.

A DataLad research data management handbook

As the mere existence of software is insufficient to ensure its uptake and use according to best practices, the cornerstone of our training efforts was a user-oriented documentation project for DataLad. Generally, documentation provides information about a software. It fulfills different roles depending on its target audience. Literature distinguishes several different types of documentation, Parnas (2011) categorizes documentation either as a *tutorial* or as a *reference work*. Both kinds are needed for different audiences: experienced users and contributors need reference documents to guide further development, such as the elements of an Application Programming Interface (API), whereas new users and contributors need a basic understanding of the software tool and its intended use

¹missing.csail.mit.edu

cases. Commonly distinguished are also *technical* and *user* documentation. The former contains information for developers and users by describing features, maintenance information, or design choices. The latter targets end users of a software product with accessible explanations or step-by-step instructions on how to install a tool and use its features. User documentation also matches the concept of *task-based* documentation, which is broken down into the activities that users will go through as they work, starting with basic tasks and continuing with more advanced tasks that become possible as users continue to work (Barker, 2003). Research software often lacks comprehensive documentation (Pawlik et al., 2012; Segal, 2007). Commonly reported reasons for this are a lack of funding, incentives, and interest by software developers (Pawlik et al., 2012). Yet although it is commonly regarded as separate from the actual piece of software, software documentation is heavily tied to the quality of a software tool and the software development process. A lack of documentation hinders knowledge transfer both among users and developers, impedes maintenance, and creates a steep learning curve for new users and new developers alike (Theunissen et al., 2022). Parnas (2011) describes a vicious cycle that sets in when the quality of software documentation is poor: “Reduced [documentation] quality leads to reduced [software] usage, [r]educed [software] usage leads to reductions in both resources and motivation, [r]educed resources and motivation degrade [software] quality further”.

Since the first release (0.0.1, March 2015), DataLad had technical documentation with a design overview and reference documentation. Although any amount of documentation is better than no documentation at all, existing documentation can still be insufficient if it does not meet the needs of the target audience. Solely technical or reference documentation, for example, can be suboptimal for novices: it may be incomplete, narrowly focused on individual commands, or assume existing knowledge readers lack (Segal, 2007; Pawlik et al., 2015), and can thereby discourage potential users or inhibit the adoption of a tool. Even though technical documentation is useful for developers, a central target audience for documentation of the DataLad ecosystem are scientists. A considerable part of this target audience can thus be considered technical novices for whom technical documentation is not ideal. Research also suggests that scientists need documentation to go beyond reference manuals. In an analysis of user questions in support forums of scientific software packages, Swarts (2019) found that the focus in 80% of inquiries was on operations and tasks, such as a required sequence of operations to achieve a specific goal, instead of reference lists. In breaking down user questions by purpose, Swarts (2019) further found that users were most interested in a description of operations or tasks, followed by insights about the reasons behind the action. Separating documentation types into *feature-based* (closer related to the concept of reference documentation) or *task-based*, Swarts (2019) reports twice as many questions seeking explanations in software with *feature-based* compared to *task-based* documentation. This hints at a disconnect between knowing *how* something should be done and *why* it should be done this way. Overall, this highlights that users of scientific software show a clear need beyond the documentation of individual commands, but seek to understand general usage principles and master complex combinations of features to achieve specified goals. This type of empowerment is what the DataLad Handbook project aimed to achieve by complementing DataLad’s existing technical documentation.

Design considerations

We identified three types of stakeholders with different needs: researchers, planners and trainers. *Researchers* need accessible educational content to understand and use the tool; *planners*, such as principal investigators or funders, need high-level, non-technical information in order to make informed yet efficient decisions on whether the tool fulfills their needs; and *trainers* need reliable, open access teaching material. Based on this assessment, the following goals for the Handbook’s contents were set:

- **Applicability for a broad audience:** The Handbook should showcase domain-agnostic, real-world RDM applications.
- **Practical experience:** The Handbook should enable a code-along style usage, with examples presented in code that users can copy, paste, and run on their own computer. To allow a read-only style usage, too, the Handbook should also reveal what a given code execution’s output would look like. For an optimal code-along or read-only experience, the code output should match the current software behavior.
- **Suitable for technical novices:** The Handbook’s language should be accessible. Gradually, by explaining technical jargon and relevant tools or concepts in passing, it should provide readers with a broad set of relevant RDM skills rather than requiring prior knowledge.
- **Low barrier to entry:** The Handbook’s contents should be organized in short, topical units to provide the possibility to re-read or mix and match.

- **Integrative workflows:** The Handbook’s contents should build up on each other and link back to content already introduced to teach how different software features interact.
- **Empowering independent users:** Instead of showcasing successful code only, it should also explicitly demonstrate common errors to enable users to troubleshoot problems in their own use cases independently.

The following structure arose from this specification analysis (Wagner et al., 2020):

Introduction: The first part of the Handbook, covering high-level descriptions of the software and its features and detailed installation instructions for all operating systems.

Basics: The second part of the Handbook, written in the form of a continuous, code-along tutorial, set in a domain-agnostic fictional storyline about an RDM application, and covering all stable software features in chapters that build up on one another.

Advanced: The third part of the Handbook covering features beyond the basics in stand-alone chapters, added prior to the second release.

Use cases: The last part of the Handbook, containing short, standalone start-to-end descriptions of real-world use cases, with concise step-by-step instructions, and references to further reading in the Basics part.

Finally, the following technical goals further supported the design and content requirements:

- **Expandable details:** To keep visible text short while preserving the ability to explore advanced contents, topical custom content boxes and toggle-able sections (“click to expand”) should make information relevant to specific user types or optional details skippable and easily discoverable.
- **Integration tests:** To ensure functioning from a user’s point of view, the workflows included in the Handbook as code need to be tested as an automated integration test suite.
- **Multi-format:** The Handbook should be available in multiple formats, at least as a web-based rendering and as a portable download to be stored offline or printed.
- **Conjoint software development:** The Handbook should be developed alongside the software, and semantic versioning should ensure that users of a past software version can find the corresponding version of the Handbook.

The resulting implementation of the Handbook fulfilled these requirements as follows:

The technical backbone

The development environment of the Handbook was chosen with the intent to support declared goals, and to maximize configurability, autonomy, and reusability of the project. It builds up entirely on flexible and extendable open source infrastructure: on the highest level, it uses Sphinx as a documentation generator (sphinx-doc.org). Sphinx transforms documents written in reStructuredText, a lightweight markup language, to a variety of output formats, among them HTML, PDF, \LaTeX , or EPUB. Initially a by-product of the Python documentation, it has been adopted by the Open Source community at large; GitHub’s dependency graph reports that it is used by more than 300.000 projects in January 2024².

Sphinx supports an extension mechanism with which additional functionality can be integrated. Leveraging this mechanism, the Handbook project extended standard Sphinx features with custom admonitions and designs, for example toggle-able boxes for optional details. This is implemented as a Python package alongside the Handbook source code, making the Handbook project a reusable and installable Sphinx extension. Figure 1 provides an overview of the custom-developed design features. A major functional enhancement is provided with a separate Python package, `autorunrecord`, an additional custom-made Sphinx extension that allows sequential execution of code in a specified environment, and embedding a record of the code and its output as code snippets into the documentation³. Instructors can further use it to automatically create scripts from selected code blocks which can then be demonstrated in a remote-controlled terminal in live-coding tutorials.

Hosting for the project is provided by Read the Docs (readthedocs.org), a full-featured software documentation

²github.com/sphinx-doc/sphinx/network/dependents

³github.com/mih/autorunrecord

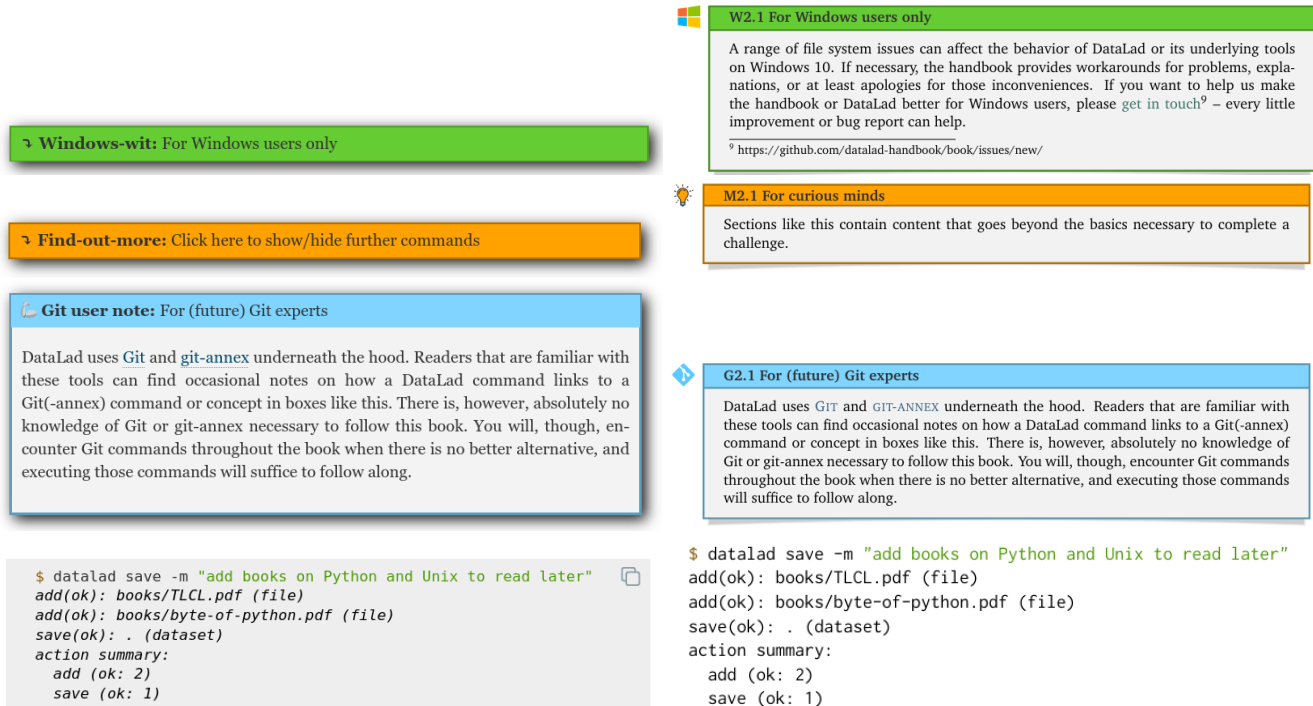


Figure 1: Custom admonitions and code blocks used in the Handbook. In each pair, the top image corresponds to the web version of the admonition, and the bottom image corresponds to its PDF rendering. From top to bottom: Windows-wits, toggle-able in the HTML version, contain information that is only relevant for the Windows operating system. Find-out-more admonitions, also toggle-able in the HTML version, contain miscellaneous extra information for curious readers. Git user notes are colored boxes with references to the underlying tools of DataLad, intended for advanced Git users as a comparison or technical explanation. Code blocks show one or more commands and the resulting output, provided using the `autorunrecord` Sphinx extension. In the web version, a copy-button (top right corner) allows to copy relevant commands automatically to the clipboard. Internal annotations allow generating custom scripts from any sequence of code-blocks for live coding demonstrations.

deployment platform that integrates with Sphinx. Illustrations in the Handbook are based on the undraw project by Katerina Limpitsouni (undraw.co).

The ability of the documentation to sequentially execute code and record its outcomes allows using the Handbook as an integration test for the DataLad software in addition to a user guide. If new software developments in the DataLad core packages break documented workflows, a continuous integration test suite will fail, alerting developers to the fact that their changes break user workflows.

To ensure reusability, such as the adaptation by Brooks et al. (2021), the project is released under a CC-BY-SA 4.0 license. Under its terms, all elements can be reused in original or derived form for all purposes under the condition that the original project is attributed and that derivative work is shared under an identical (“not more restrictive”) license⁴.

Content

As of January 2024, the web and PDF versions of the Handbook were organized into four parts – “Introduction”, “Basics”, “Advanced”, and “Use cases” – which comprised a total of 21 chapters. The “Introduction” part has two different target audiences: first, it provides *researchers* with detailed installation instructions, a basic general command line tutorial, and an overview of the Handbook. Beyond this, it gives a high-level overview of the software and its capabilities to *planners*.

The “Basics” part is organized into nine chapters. Following a narrative about a fictional college course on RDM, it teaches different aspects of DataLad functionality and general RDM to *researchers* in each topical chapter. Broadly, those topics can be summarized as follows: 1) Local version control, 2) Capturing and re-executing process provenance, 3) Data integrity, 4) Collaboration and distributed version control, 5) Configuration, 6) Reproducible

⁴creativecommons.org/licenses/by-sa/4.0

data analysis, 7) Computationally reproducible data analysis, 8) Data publication, and 9) Error management. The “Advanced” part includes independent chapters on advanced DataLad features and workflows, big data projects, DataLad use on computational clusters, DataLad’s internals, and selected DataLad extensions. The latter two parts are accompanied with code demonstrations, slides, executable notebooks, and/or video tutorials that *trainers* can reuse freely to teach tool use and improve scientific practice. The last part, “Use cases”, targets *planners* and *researchers* with short step-by-step instructions which show *planners* what is possible, and help *researchers* to connect their knowledge into larger workflows.

Project and community management

Ensuring the longevity of software projects beyond the duration of individual researchers’ contracts requires community building (Koehler Leman et al., 2020). A user-driven alternative to documentation by software developers, “Documentation Crowdsourcing”, has been successfully employed by the NumPy project (Pawlik et al., 2014). The Handbook project extends this concept beyond reference documentation. To achieve this, it is set up to encourage and welcome improvements by external contributors. The project is openly hosted on GitHub. Mirroring processes in larger crowd-sourced documentation projects such as the “The Turing Way handbook for reproducible, ethical and collaborative research” (The Turing Way Community, 2022), credit is given for both code-based and non-code-based contributions. Contributors are recognized in the source repository, on the DataLad Website, and as co-authors in both the printed version of the Handbook and its Zenodo releases. As of January 2024, a total of 60 contributors provided input in the form of content, bug fixes, or infrastructure improvements.

Paperback version

A digest of the Handbook was published via the Kindle Direct Publishing (KDP) print-on-demand service to make the Handbook available in a printed paperback version. This fulfilled user demands for physical copies of the documentation, and was possible with minimal additional technical work, building up on the automatically generated LaTeX sources of the Handbook. The printed book’s contents were sub-selected for longevity, graphics or graphical in-text elements were optimized for black-and-white printing, and a dedicated hyperlink index was created.

RDM online course

While documentation is the primary way of disseminating information about software, workshops are another often practiced way of software education. As maintainers and contributors of DataLad, we receive invitations to teach such workshops for different audiences, most commonly involving early career researchers. Some such events arise from obligations related to consortium participation (such as the CRC 1451⁵, Collaborative Research Center, investigating mechanisms of motor control, where RDM training was organised with course credit for involved doctoral students); others stem from more informal collaborations. To be better prepared for organizing training events, we decided to create a curriculum for a short RDM course centered around DataLad⁶. Our design approach aligns with the “Ten simple rules for collaborative lesson development” (Devenyi et al., 2018), and the course content and format were inspired by the Software Carpentry courses (Wilson, 2016).

Design considerations

The course curriculum was created with the following goals in mind:

- **Smaller extent:** While the Handbook is a comprehensive set of documentation covering multiple aspects, the course materials should provide a shorter and more focused overview of the key features of DataLad software. The course should be self-contained, but may link to the Handbook when further detail or context is needed.
- **Workshop focused:** The course materials should be tuned to presentation during a hands-on workshop (either online or in-person). However, they should remain equally useful for self-study. The envisioned hands-on course duration was two half-days (four blocks of two hours).

⁵crc1451.uni-koeln.de

⁶psychoinformatics-de.github.io/rdm-course

- **Ground-up introduction:** The course should provide an introduction to using the DataLad software by introducing its basic commands. They should be illustrated with interdependent examples, which make it easy to present not just usage but also the purpose of a given command. While it is impossible to cover all the details, demonstrating the principle and pointing to documentation should enable participants to adapt the usage to their own real-world use cases.
- **Good RDM practice:** While the course focuses on DataLad software specifics, it should also contain information about good practices in research data management which are software-independent. Such information can be especially useful for early career researchers.
- **Modular and reusable:** It should be easy for tutors (also those not involved in course preparation) to use the course materials on different occasions. As these may involve different audiences or different time constraints, contents should be divided into related blocks, some optional, which can be mixed and matched.
- **Open resource:** The course materials should be openly available. This includes publishing them as a publicly accessible website, and hosting sources used for generating the website in a public repository, so that they can be collaboratively edited. This also means licensing the materials under Creative Commons Attribution License, and reusing other permissively licensed materials.

During the planning phase, we identified a set of data management tasks which should be covered, from dataset creation and local version control, through data publishing in external repositories and collaboration, to reusing datasets published by others and creating an analysis with modular datasets. The major theme for the software-agnostic part about good RDM practices, which came up in an informal poll among our colleagues, was file naming and organisation. Although it may sound trivial at first glance, this includes topics such as rationales for naming schemes, interoperability considerations related to file names (lengths, character sets), avoiding leakage of identifying information through file names, using sidecar files for metadata, clear semantics for separating inputs and outputs, and standard file organization structures, e.g. BIDS (Gorgolewski et al., 2016) or research compendium (Gentleman & Temple Lang, 2007). In addition to these, we also decided to discuss the distinction between text and binary files, and show examples of how the former can be used to store different kinds of data and metadata in an interoperable fashion (tabular files, serialization formats, lightweight markup).

The technical backbone

The course website with the full course material is created based on The Carpentries⁷ lesson template⁸. Website content is written in Markdown, and the website is built with the Ruby-based static site generator Jekyll (note, however, that The Carpentries recently redesigned their tooling to use R’s publishing ecosystem instead⁹). Course material is split into sections, each starting with an overview (questions, objectives, time estimate), and ending with a summary of key points. The content is presented using a combination of text paragraphs and template-defined boxes with code samples, expected output, call-outs, challenges, and more.

During courses, we use Jupyter Hub to provide a unified, pre-configured software environment for participants, accessible through a web browser. While Jupyter Hub is mainly associated with notebooks, we mostly use its terminal feature, effectively providing participants with browser-based access to a terminal running on a remote machine. To simplify deployment, we use The Littlest JupyterHub¹⁰ (TLJH) distribution to set up the hub for all users on a single machine. We have used Amazon Web Services to provision virtual machines, but other cloud computing providers or local infrastructure can be used to the same effect. Setup instructions, expanded from TLJH’s documentation, were included in the course website, in the “For instructors” section.

Content

Following the design considerations, we organized the course in the following modules:

- **Content tracking with DataLad:** learning the basics of version control, working locally to create a dataset, and practicing basic DataLad commands.

⁷carpentries.org

⁸github.com/carpentries/styles

⁹carpentries.github.io/workbench/

¹⁰tljh.jupyter.org

- **Structuring data:** listing good practices in data organization, distinguishing between text and binary data, and exploring lightweight text files and how they can be useful.
- **Remote collaboration:** exercising data publication and consumption, and demonstrating the dissociation between file content and its availability record.
- **Dataset management:** demonstrating dataset nesting (subdatasets), investigating structure and content of a published dataset, and creating a simple model of a nested dataset.
- (optional) **The basics of branching:** understanding Git’s concept of a branch, creating new branches in a local dataset and switching between them, and mastering the basics of a contribution workflow.
- (optional) **Removing datasets and files:** learning how to remove dataset content, and removing unwanted datasets.

Additionally, the course website contains a short glossary, setup instructions for users (if using their own computers), slides, and instructor notes about the technical setup.

Knowledge base and online office hours

The educational resources were designed to be broadly applicable and domain-agnostic, but could not necessarily cover arbitrary use cases. Practical application of DataLad in RDM scenarios involves developing solutions to complex problems. No individual solution will always be one-size-fits-all, and no documentation can ever be comprehensive for everyone without becoming overwhelming for some. Likewise, while useful for discovering and learning usage patterns, most resources were of limited utility for trouble-shooting software issues. To this end, we offer a weekly online office hour to provide flexible assistance, and we invested resources into creating a knowledge base¹¹. Office hours are a one-hour open video call during which (prospective) users can join flexibly and without prior notice, and ask questions or discuss use cases, often live-demoing relevant information via screen-sharing. The knowledge base, on the other hand, is a collection of documents, each document focusing on a particular topic (application, problem, solution) and considered standalone with respect to other documents. The nature of these documents resembles technical reports, the creation of which has a long-standing tradition in science and engineering (Brearley, 1973; Pinelli et al., 1982). When explorations in an office hour uncover technical limitations requiring workarounds or interesting use cases that are too peculiar to be prominently documented in the Handbook or RDM course, they typically inspire an entry in the knowledge base.

Design considerations

A knowledge base provides resources to anyone seeking particular solutions. It can also be used to accumulate the outcomes of investigations of technical issues as they occur when supporting users, thereby yielding persistent resources that streamline future support efforts, and increasing the efficacy of resources invested in support (turning incoming feedback into knowledge). An analysis of the content written on-demand, or its access frequency can also be used to inform prioritization of development efforts to improve technical implementations and/or documentation elsewhere.

Suitable topics for a knowledge base item (KBI) are:

- an answer to a frequently asked question (be that from office hours, issue trackers, or community forums),
- tips and strategies for a particular use case,
- a description of a technical limitation and possible workaround.

Each KBI needs to have: a descriptive title; metadata, such as keywords, to aid discovery; and a persistent URL to share it.

¹¹knowledge-base.psychoinformatics.de

The technical backbone

The technical framework for the knowledge base is a simplified version of that used for the Handbook. In summary, KBIs are plain-text documents with reStructuredText markup. All KBI files are kept in a Git repository. A rendered knowledge base in HTML format is created with the Sphinx tool. A knowledge base Git repository is managed with the aid of a Git hosting solution, such as GitHub/GitLab. Respective continuous integration and website publishing tools are used to publish the knowledge base. Coordination for the office hour is done using a public matrix¹² chatroom, in which questions can also be asked asynchronously.

Content

As of January 2024, the knowledge base contains 29 KBIs of varying length, describing various use cases. For example, the first KBI that we created describes a situation in which DataLad (or Git) users working on shared infrastructure can trigger Git’s safety mechanism, added to Git versions released after March 2022, which causes certain operations to end with an error message displayed to the user. The knowledge base format allowed us to explain in detail not only the configuration options that need to be set in order to perform the operation, but also the broader rationale for the safety mechanism being present in Git in the first place (quoting, e.g., the informative commit messages which accompanied the changes made in Git).

Impact and scope

Online handbook

Work on the Handbook began in June 2019, and the first release followed in January 2020. It has been under continuous development for more than four years, averaging two releases per year, and complements the DataLad ecosystem with a comprehensive user guide. Its PDF version spans more than 600 pages. Releases of the DataLad core package are coordinated with matching releases of the Handbook project, and past release versions remain accessible online.

Confirming observations from the literature (van Loggem & van der Veer, 2014), the conjunct development of user documentation has positive effects on software quality. As the writing process involved manual software testing, initial developments were accompanied by a higher discovery rate of software errors. This user-focused approach uncovers deficiencies of the technical documentation and API elements with suboptimal user experience. The workflow-based nature of demonstrations highlights API inconsistencies, and the integration test that the Handbook constitutes catches incompatibilities between the software and common usage practice. These documentation features facilitate software development, and had a major impact on the conjoint 0.12.0 release of DataLad (Jan 2020), the first with a matching Handbook release. The popularity data in Figure 2 confirms a marked increase in downloads from this date onward. In addition, differences in web traffic confirm that user documentation is in higher demand than the technical documentation. An analysis of visits to the web version of the Handbook from December 2022 to July 2023 revealed that handbook.datalad.org averaged 22.000 total page views per 30 days. In the same time span, the technical documentation of DataLad at docs.datalad.org averaged 6600 total page views per 30 days, less than a third of the traffic of the user documentation (Figure 2). In summary, the development of the DataLad Handbook had a measurable positive impact on the number of users, the popularity of the package, and the software quality.

Workshops

We conducted a post-workshop survey among participants of the first two instances of the workshop conducted for the CRC 1451 early career researchers. Most participants were PhD students (who received course credit for workshop participation), and the workshops were conducted online. Out of 17 respondents, 13 indicated that they would be “likely” or “very likely” to recommend the workshop to their colleagues, and as many gave the course an overall rating of 4 or higher (on a scale of 1-“insufficient” to 5-“excellent”). When asked about the learning pace (material *vs.* time), 6 respondents rated it as “neither too fast nor too slow”, 6 as “slightly too fast”, 4 as “slightly too slow”, and 1 as “too fast”. When asked whether the knowledge would be applicable to their work, 8 respondents selected “definitely” or “mostly”, while another 8 respondents selected “somehow”. Given that the CRC 1451 project combines clinical, preclinical, and computational neuroscience, and in light of the overall ratings,

¹²matrix.org

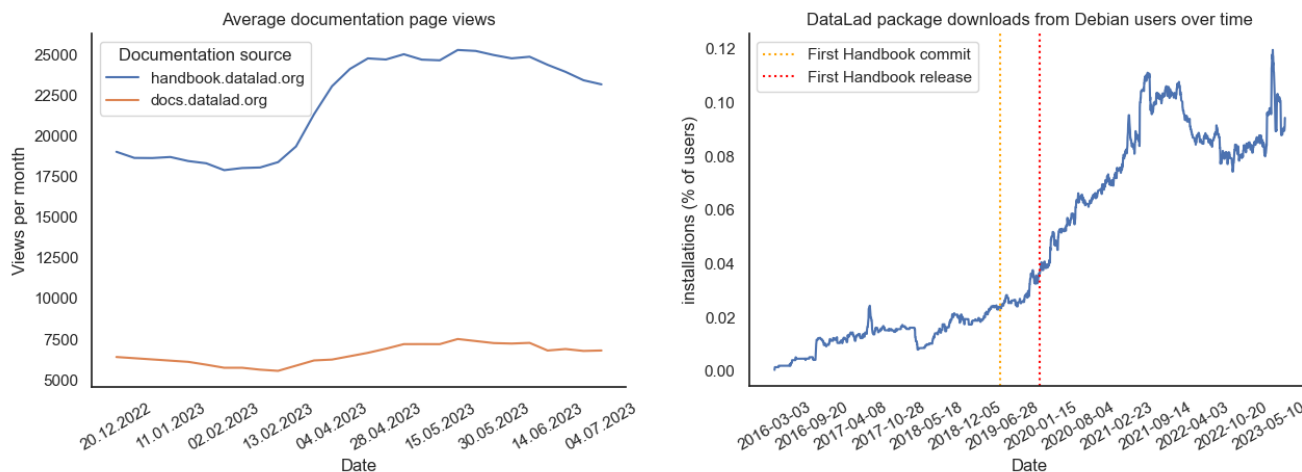


Figure 2: Software and documentation popularity measures. Left: Documentation page views for the technical documentation (`docs.data-lad.org`) and the DataLad Handbook (`handbook.data-lad.org`), displayed as average views per 30 days, between December 2022 and August 2023. The Handbook consistently sees higher traffic. Right: Popularity of the Debian packages `data-lad` and `python3-data-lad` over time, expressed in percent of Debian users that installed the package from the total amount of users submitting download statistics. Dashed lines indicate the first commit and the first release of the Handbook, the latter marks a notable increase in software downloads. Source: Debian Popularity Contest, August 2023.

we think that responses to the last two questions are indicative of the diversity of PhD students’ backgrounds, and a varying degree to which formal RDM is an already established practice in their respective research fields. One recurring suggestion for improvement was to include more examples of real-world applications. This highlights that although a course dedicated to software basics is a good start, transferring the knowledge to specific applications is the real challenge, which can be greatly improved with existing written documentation.

Knowledge base

In the span of ten months we accumulated 29 knowledge base items of various length. The knowledge base has been useful in answering recurring questions, or communicating recommended workflows. Beyond this, it was also a valuable method to keep the official Handbook or course material lean, as it provided a home to more temporary or niche use cases.

Lessons learned

In the previous sections we described the design considerations and their practical applications for the documentation and education aspect of the DataLad project. In our opinion, creating and maintaining this growing collection of materials was a worthwhile investment, which helped users to apply the software and developers to improve it. The open source, flexible approach to creating educational content was particularly valuable for its maintainability, adaptability, and applicability to various research contexts. In this closing section we want to share our comments – lessons learned in the process – on various related aspects.

Customization vs complexity

Our technical choices for the Handbook had to be weighed against non-technical features. Compared to other handbook projects such as the Turing Way project (The Turing Way Community, 2022), sources based on `reStructuredText` and `Sphinx`, as well as the many custom admonitions, constitute a higher barrier to entry for contributors. The Turing Way community, for example, explicitly chose Markdown-based Jupyter Book tooling to ease contributing for technical novices. Indeed, Handbook maintainers regularly have to assist new contributors with technical details, and complex technical contributions almost always come from the core contributor team. Nevertheless, in our case – especially with the requirements for multiple formats, integration tests, and reuse in print editions – the customization opportunities of `Sphinx` made up for slightly higher complexity than alternative documentation frameworks.

Yes, there can be too much documentation

Although a large amount of documentation appears universally positive, there are concrete downsides that can increase with the amount of documentation. If more content leads to duplication, maintenance costs increase steeply, and so does the threat of showcasing outdated information. Thus, wherever possible, information is only detailed in a single location, and other places refer or link this source rather than duplicating its content.

Additionally, a large amount of documentation can appear intimidating. In our experience, information that educational resources exists is met positively, but the notion of a “600 page handbook” can diminish this enthusiasm. We find anecdotal evidence that a (surprisingly) large amount of available documentation can be perceived as a warning sign regarding software complexity, and is interpreted as a requirement to process all available documentation before a meaningful proficiency can be reached. Designing all resources as best as possible in a modular, pick-what-you-need style proved to be important to allow for selective consumption and to lower the perceived cost of entry.

Keeping online workshops interactive

Keeping participants engaged during an online workshop is a particular challenge, as it is much harder to “read the room”. We have had positive experiences with using interactive poll and Q&A platforms, such as, e.g. DirectPoll¹³ or Slido¹⁴. Additionally, we believe that having co-presenters who can monitor text chat or take part of the questions is invaluable.

Avoiding installfest

Software should be easy to install, and we believe this is the case for DataLad. However, the preferred method of installation will differ between users. DataLad can currently be installed through several methods: conda, pip, apt and several other package managers (GNU/Linux), homebrew (MacOS). Selecting one of these methods will depend on how it integrates (or clashes) with the methods used for managing the entire software environment(s), and, if chosen hastily, may lead to future issues. To this end, we provide an overview of installation methods in the Handbook, and a note on debugging issues related to using multiple Python versions in the knowledge base. For this reason, performing an installation as part of the workshop may turn out to be time consuming, and we tend to avoid it. If installing a given software on participants’ computers is a goal (because it is required for the workshop or for future work), one approach that we found useful (at least with certain audiences) is to provide a link to detailed installation instructions beforehand and ask participants to e-mail the instructor with the output of a diagnostic command (or describe encountered problems). This encourages engagement from the participants, and may also provide instructors (maintainers) with an insight into how well the installation process works in practice.

Software environment nuances

Exploring a command-line tool (particularly one for managing files) can hardly be separated from using basic command-line utilities (e.g. for changing the working directory or listing files). Although their usage can be weaved into the narrative of a workshop, this introduces additional complexity for command-line interface (CLI) novices. Moreover, while core utilities are similar across systems, there are differences, often subtle, in how they should be used to produce the same effect (e.g., compare `tree` vs `tree -F`, or `which` vs `Get-Command`, in Bash and PowerShell, respectively). The impact of these differences can be mitigated by providing toggle-able OS-specific instructions in the published materials, however, they still present a major challenge during live workshops when participants use different operating systems with their default sets of tools. For this reason, we prefer to use a common JupyterHub deployment for hands-on sessions.

Cloud computing

Both virtual and in-person workshops benefit from prepared virtual computing environments in particular. Costs per workshop amount to a few Euros, and typically never exceed 15 Euro even for multi-day workshops. The setup of the respective Amazon EC2 Cloud instance takes a few hours at most.

¹³directpoll.com

¹⁴slido.com

Data production and data consumption

The RDM course was created from the data producer perspective, and walks users through the process of building a dataset from scratch, covering data consumption only at a later stage. Aside from the fact that data analysis in computational neuroscience projects may just as often start with obtaining existing datasets, this narrative creates a situation where multiple steps are needed to reach a situation where benefits of version controlling the data can be seen. It could be an interesting change of perspective to start with obtaining a copy of an already created dataset, something which currently gets introduced in a later part of the workshop, and inspect its properties (such as content, history, and file availability information) in order to highlight the value added by RDM software. We tried this approach during shorter software demos, where having the target state communicated upfront was particularly useful for streamlining the presentation.

Technical writing takes time

Preparing a description of a discovered solution in the shape of a knowledge base item is time consuming, and a task on its own. However, it generates a resource which becomes useful with time, as the solution is captured with its context. Working on such solutions is a valuable way to learn about the program – also for developers who need not be familiar with all parts of the code base, or all potential applications.

Author contributions

ASW and MS conceptualized and wrote the first draft of the manuscript. All authors reviewed and edited the manuscript. ASW, LKW, and MH conceptualized the RDM Handbook. ASW, LKW, MH, MS, SH maintained the RDM Handbook and curated the majority of its content. ASW and MH prepared the print version of the RDM Handbook. MS conceptualized the RDM course, created the technical backbone, and curated its content. ASW, MH, and SH contributed to the technical backbone and content of the RDM course. ASW, MH, MS, and SH presented multiple online and in-person workshops based on the content of both the Handbook and RDM course. MH and MS conceptualized the knowledge base. ASW, LKW, MH, MS, and SH contributed to the technical backbone and content of the knowledge base. MH and SBE acquired funding.

Statements and declarations

Competing interests

The authors have no competing interests to declare that are relevant to the content of this article.

Funding

The RDM course was developed with funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant SFB 1451 (431549029, INF project). The DataLad project received support through the following grants: US-German collaboration in computational neuroscience (CRCNS) project “DataGit: converging catalogues, warehouses, and deployment logistics into a federated ‘data distribution’”, co-funded by the US National Science Foundation (NSF 1429999) and the German Federal Ministry of Education and Research (BMBF 01GQ1411); CRCNS US-German Data Sharing “DataLad - a decentralized system for integrated discovery, management, and publication of digital objects of science”, (NSF 1912266, BMBF 01GQ1905); Helmholtz Research Center Jülich, FDM challenge 2022; German federal state of Saxony-Anhalt and the European Regional Development Fund (ERDF), Project: Center for Behavioral Brain Sciences, Imaging Platform; ReproNim project (NIH 1P41EB019936-01A1); Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant SFB 1451 (431549029, INF project); European Union’s Horizon 2020 research and innovation programme under grant agreements Human Brain Project SGA3 (H2020-EU.3.1.5.3, grant no. 945539), VirtualBrainCloud (H2020-EU.3.1.5.3, grant no. 826421); EBRAINS 2.0 (HORIZON.1.3, grant no. 101147319).

Information sharing statement

The sources of all projects described in this manuscript are available from GitHub and licensed under CC-BY, CC-BY-SA, or MIT licenses:

- github.com/datalad-handbook/book,
- github.com/datalad-handbook/book-datalad-intro,
- github.com/psychoinformatics-de/rdm-course,
- github.com/psychoinformatics-de/knowledge-base

The authors wish to thank Tosca Heunis for her feedback on earlier versions of this manuscript. The DataLad software and its documentation are the joint work of more than 100 individuals. We are deeply grateful for these contributions to free and open source software (FOSS) and documentation. Likewise we are grateful to the many more people that produce and maintain the FOSS ecosystem that DataLad is built on. We are particularly indebted to Joey Hess, the author of the git-annex software, without which DataLad would not be what it is today.

References

- Barker, T. T. (2003). *Writing software documentation*. NY: Longman, New York.
- Brearley, N. (1973). The role of technical reports in scientific and technical communication. *IEEE Transactions on Professional Communication*, *PC-16*(3), 117–119. <https://doi.org/10.1109/tpc.1973.6592685>
- Brooks, P. P., McDevitt, E. A., Mennen, A. C., Testerman, M., Kim, N. Y., Visconti di Oleggio Castello, M., & Nastase, S. A. (2021). *Princeton handbook for reproducible neuroimaging* (Version v0.2.0). Zenodo. <https://doi.org/10.5281/zenodo.4317623>
- Csomós, G., Vida, Z. V., & Lengyel, B. (2020). Exploring the changing geographical pattern of international scientific collaborations through the prism of cities. *PloS one*, *15*(11), e0242468. <https://doi.org/10.1371/journal.pone.0242468>
- Devenyi, G. A., Emonet, R., Harris, R. M., Hertweck, K. L., Irving, D., Milligan, I., & Wilson, G. (2018). Ten simple rules for collaborative lesson development (S. Markel, Ed.). *PLOS Computational Biology*, *14*(3), e1005963. <https://doi.org/10.1371/journal.pcbi.1005963>
- Ferguson, A. R., Nielson, J. L., Cragin, M. H., Bandrowski, A. E., & Martone, M. E. (2014). Big data from small data: Data-sharing in the ‘long tail’ of neuroscience. *Nature neuroscience*, *17*(11), 1442–1447. <https://doi.org/10.1038/nn.3838>
- Gentleman, R., & Temple Lang, D. (2007). Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics*, *16*(1), 1–23. <https://doi.org/10.1198/106186007x178663>
- Gorgolewski, K. J., Auer, T., Calhoun, V. D., Craddock, C. R., Das, S., Duff, E. P., Flandin, G., Ghosh, S. S., Glatard, T., Halchenko, Y. O., et al. (2016). The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific data*, *3*(1), 1–9. <https://doi.org/10.1038/sdata.2016.44>
- Gorgolewski, K. J., & Poldrack, R. A. (2016). A practical guide for improving transparency and reproducibility in neuroimaging research. *PLoS biology*, *14*(7), e1002506. <https://doi.org/10.1371/journal.pbio.1002506>
- Grisham, W., Lom, B., Lanyon, L., & L. Ramos, R. (2016). Proposed training to meet challenges of large-scale data in neuroscience. *Frontiers in Neuroinformatics*, *10*, 28. <https://doi.org/10.3389/fninf.2016.00028>
- Halchenko, Y. O., Meyer, K., Poldrack, B., Solanky, D. S., Wagner, A. S., Gors, J., MacFarlane, D., Pustina, D., Sochat, V., Ghosh, S. S., Mönch, C., Markiewicz, C. J., Waite, L., Shlyakhter, I., de la Vega, A., Hayashi, S., Häusler, C. O., Poline, J.-B., Kadelka, T., ... Hanke, M. (2021). Datalad: Distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software*, *6*(63), 3262. <https://doi.org/10.21105/joss.03262>
- Hess, J. (2010). *git-annex*. <https://git-annex.branchable.com/>
- Koehler Leman, J., Weitzner, B. D., Renfrew, P. D., Lewis, S. M., Moretti, R., Watkins, A. M., Mulligan, V. K., Lyskov, S., Adolf-Bryfogle, J., Labonte, J. W., et al. (2020). Better together: Elements of successful scientific software development in a distributed collaborative community. *PLoS computational biology*, *16*(5), e1007507. <https://doi.org/10.1371/journal.pcbi.1007507>
- Mehlenbacher, B. (2003). Documentation: Not yet implemented, but coming soon. *The HCI handbook: Fundamentals, evolving technologies, and emerging applications*, 527–543.
- Mons, B. (2018). *Data stewardship for open science: Implementing fair principles*. CRC Press.
- Niso, G., Rogers, C., Moreau, J. T., Chen, L.-Y., Madjar, C., Das, S., Bock, E., Tadel, F., Evans, A. C., Jolicoeur, P., et al. (2016). Omega: The open meg archive. *Neuroimage*, *124*, 1182–1187. <https://doi.org/10.1016/j.neuroimage.2015.04.028>

- Parnas, D. L. (2011). Precise documentation: The key to better software. In S. Nanz (Ed.), *The future of software engineering* (pp. 125–148). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-15187-3_8
- Pawlik, A., Segal, J., & Petre, M. (2012). Documentation practices in scientific software development. *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)*, 113–119. <https://doi.org/10.1109/CHASE.2012.6223004>
- Pawlik, A., Segal, J., Sharp, H., & Petre, M. (2014). Crowdsourcing scientific software documentation: A case study of the numpy documentation project. *Computing in Science & Engineering*, 17(1), 28–36. <https://doi.org/10.1109/mcse.2014.93>
- Pinelli, T. E., Glassman, M., & Cordle, V. M. (1982, August). *Survey of reader preferences concerning the format of nasa technical reports* (tech. rep. No. NASA-TM-84502). National Aeronautics and Space Administration.
- Puce, A., & Hämäläinen, M. S. (2017). A review of issues related to data acquisition and analysis in eeg/meg studies. *Brain sciences*, 7(6), 58. <https://doi.org/10.3390/brainsci7060058>
- Segal, J. (2007). Some problems of professional end user developers. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, 111–118. <https://doi.org/10.1109/VLHCC.2007.17>
- Swarts, J. (2019). Open-source software in the sciences: The challenge of user support. *Journal of business and technical communication*, 33(1), 60–90. <https://doi.org/10.1177/1050651918780202>
- The Turing Way Community. (2022). *The Turing Way: A handbook for reproducible, ethical and collaborative research* (Version 1.0.2). Zenodo. <https://doi.org/10.5281/zenodo.7625728>
- Theunissen, T., Heesch, U., & Avgeriou, P. (2022). A mapping study on documentation in continuous software development. *Information and Software Technology*, 142, 106733. <https://doi.org/10.1016/j.infsof.2021.106733>
- van Loggem, B., & van der Veer, G. C. (2014). A documentation-centred approach to software design, development and deployment. In A. Ebert, G. C. van der Veer, G. Domik, N. D. Gershon, & I. Scheler (Eds.), *Building bridges: Hci, visualization, and non-formal modeling* (pp. 188–200). Springer Berlin Heidelberg.
- Wagner, A. S., Waite, L. K., Waite, A. Q., Reuter, N., Poldrack, B., Poline, J.-B., Kadelka, T., Markiewicz, C. J., Vavra, P., Paas, L. K., Herholz, P., Mochalski, L. N., Kraljevic, N., Heckner, Y. O., Marisa K. and Halchenko, & Hanke, M. (2020, June). The DataLad Handbook: A user-focused and workflow- based addition to standard software documentation. <https://doi.org/10.5281/zenodo.7906718>
- Wilson, G. (2016). Software carpentry: Lessons learned. *F1000Research*, 3, 62. <https://doi.org/10.12688/f1000research.3-62.v2>