Creating great documentation for bioinformatics software

Mehran Karimzadeh¹ and Michael M. Hoffman¹,²,³

¹Princess Margaret Cancer Centre, Toronto, ON, Canada
²Department of Medical Biophysics, University of Toronto, Toronto, ON, Canada
³Department of Computer Science, University of Toronto, Toronto, ON, Canada

July 18, 2016

Abstract

Abstract:
Investing in documenting your bioinformatics software well can increase its impact and save your time. To maximize the effectiveness of your documentation, we suggest following a few guidelines we propose here. We recommend providing multiple avenues for users to use your research software, including a navigable HTML interface with a quick start, useful help messages with detailed explanation, and thorough examples for each feature of your software. By following these guidelines, you can assure that your hard work maximally benefits yourself and others.

Summary:
Great bioinformatics software documentation provides detailed instructions for installation, usage and all available options. It begins with a quick start guide with walk-through examples. Details of software capabilities are navigable through a hierarchical interface. Users can request further assistance through a searchable forum.

Author biographies:

Michael Hoffman is a Scientist at the Princess Margaret Cancer Centre and Assistant Professor in the Departments of Medical Biophysics and Computer Science, University of Toronto. He researches the application of machine learning techniques to epigenomic data.

Mehran Karimzadeh is a PhD student at the Department of Medical Biophysics, University of Toronto. His research focuses on the dynamics of transcription factor behavior in cancer.

Contact:
michael.hoffman@utoronto.ca

Key words:
software, documentation
1 Introduction

You have written a new software package far superior to any existing method. You submit a paper describing it to a prestigious journal, but it is rejected after Reviewer 3 complains they can’t get it to work. Eventually, a less exacting journal publishes the paper, but you never get as many citations as you expected. Meanwhile, there is not even a single day when you aren’t inundated by emails asking very simple questions about using your software. Your years of work on this method have not only failed to reap the dividends you expected, but have become an active irritation. And you could have avoided all of this by writing effective documentation in the first place.

Academic bioinformatics curricula rarely train students in documentation. Many bioinformatics software packages lack sufficient documentation. Developers often prefer spending their time elsewhere. In practice, this time is often borrowed, and by ducking work to document their software now, developers accumulate documentation debt. Later, they must pay off this debt, spending even more time answering user questions than they might have by creating good documentation in the first place. Of course, when confronted with inadequate documentation, some users will simply give up, reducing the impact of the developer’s work.

To avoid this, we suggest several guidelines for improving multiple aspects of your documentation (Table 1). These guidelines improve the usability of your software and reduce time spent supporting users. Well-documented software also receives contributions from other people more often.

<table>
<thead>
<tr>
<th>Format</th>
<th>Content</th>
<th>Audience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript</td>
<td>Conceptual and technical details of the method</td>
<td>New users</td>
</tr>
<tr>
<td>Readme</td>
<td>Basic instructions for installation and use of the software and where to find more information</td>
<td>New users</td>
</tr>
<tr>
<td>Quick start</td>
<td>Step-by-step instructions for installation and use of the software on a provided test dataset</td>
<td>New users</td>
</tr>
<tr>
<td>Reference manual</td>
<td>Complete details of every configurable setting, input, and output</td>
<td>All users</td>
</tr>
<tr>
<td>FAQ</td>
<td>Answers to commonly asked or anticipated questions</td>
<td>All users</td>
</tr>
<tr>
<td>Searchable forum or mailing list</td>
<td>News and discussion of details not otherwise provided in the documentation or not apparent to users</td>
<td>All users</td>
</tr>
<tr>
<td>Built-in help</td>
<td>Concise description of a software component and its parameters</td>
<td>Experienced users</td>
</tr>
<tr>
<td>News</td>
<td>Changes in behavior, bug fixes, new features, and caveats</td>
<td>Experienced users</td>
</tr>
<tr>
<td>Code comments</td>
<td>Extensive details of implementation</td>
<td>Power users</td>
</tr>
</tbody>
</table>

Table 1: A taxonomy of research software documentation.
Figure 1: Multi-level hierarchy in the MEME Suite documentation. The MEME Suite provides a variety of tools for motif enrichment analysis. (a) The “Manual” tab in the sidebar organizes individual tools into broad categories. Each tool then has a link to its own detailed reference manual. (b) The website’s main page describes application of different tools in a flow chart, providing the context of how they work together. (c) A section of the DREME tool’s reference manual, showing further hierarchy and comprehensive detail. A four-column table describes details of each option in the DREME program. Each row describes a single option, and these options are categorized into broader option groups.

2 Guidelines for great documentation

2.1 Hierarchical documentation

Your documentation should consist in hierarchically grouped and carefully sorted components. This allows users to efficiently find the detail they need without overwhelming them with a large span of top-level material. It limits the amount of information shown to the user at one time. Sort the most important materials at the top and less frequently used details at the bottom.

The MEME Suite contains multiple programs for sequence motif analysis. Its documentation begins with a flow chart that describes its modules and their relationship to each other (Figure 1b). Other top-level items provide information on installation, databases that the programs rely on, and
Figure 2: **Hierarchy, extensive examples, and visual diagrams in the Bedtools documentation.** (a) The Bedtools documentation uses a well-organized hierarchy to provide appropriate entry points for new or experienced users. (b) Bedtools breaks its documentation into an individual page for every sub-command, describing every parameter of every module in detail and documenting changes in different versions. (c) A visual diagram demonstrates the effects of various options. (d) Bedtools demonstrates examples for every option.

ways to get support. The MEME Suite also has a top-level menu that groups programs by function (Figure 1a). More commonly used modules appear first. This grouping and ordering makes it easier for users to find the module they need and to compare with related tools for their task.

For example, the “Manual” section of the sidebar, groups the programs into four categories—“Motif Discovery”, “Motif Enrichment”, “Motif Scanning”, and “Motif Comparison” (Figure 1a). The manual of each program within describes both the web and command-line interfaces. As an illustrative sub-example, we will examine further the manual for DREME, one of the MEME Suite’s motif discovery tools. Its command-line documentation consists in several components. “Usage” describes the minimal parameters for using the program. “Description” includes a technical but abstract explanation of DREME’s functionality. The manual comprehensively defines “Input” and “Output” formats and describes options in detail using a table (Figure 1c). This table groups the options in several categories such as “Input/Output”, “Alphabet”, “General”, and “Miscellaneous”. For each option, this table describes the parameters, description, and the default behavior in subsequent columns. The MEME Suite concludes each program’s manual with a citation to the peer-reviewed manuscript describing that program.

Bedtools\(^1\) provides another example of well-documented and widely used bioinformatics software. Bedtools has a table of contents that directs users to the information they need (Figure 2a). These contents consist in a hierarchy of information structured and stored for optimal retrieval (Fig-
2.2 Tools for documentation

Several software packages automatically generate up-to-date documentation from a markup language in the source code and elsewhere. These tools transform your code and markup into formats such as Unix manual (“man”) page, Hypertext Markup Language (HTML), and Portable Document Format (PDF). Ideally you will create all these formats, but we consider a HTML manual most essential.

Examples of documentation generators include Doxygen\textsuperscript{2} and Sphinx\textsuperscript{3}. Sphinx has particular popularity in bioinformatics due to its use of the intuitive markup language reStructuredText\textsuperscript{4} and extensive formatting options. Some tools generate documentation specifically for one programming language, such as Javadoc\textsuperscript{5} for Java, or Roxygen\textsuperscript{6} for R.

The main disadvantage of automatically generated documentation is that you have less control of how to organize the documentation effectively. Whether you used a documentation generator or not, however, there are several advantages to an HTML website compared to a PDF document. Search engines will more reliably index HTML web pages. In addition, users can more easily navigate the structure of a web page, jumping directly to the information they need.

2.3 Quick start

Design your manual with a “quick start” that tells users exactly how to get a result with a small number of explicit steps on a specified test dataset. If this dataset is not included in your package, ensure one can download the dataset quickly. The user should be able to follow your script exactly and get the same results you describe. Ensure that these steps execute quickly.

For example, Segway\textsuperscript{7} includes a quick start guide with four subsections covering installation and configuration, acquiring data, running Segway, and results. Segway includes the dataset for testing the software in its own repository.

2.4 Installation

Describe how to install your software and all of its dependencies, in detail. At a minimum, provide exact instructions for the most recent versions of Debian, Red Hat Enterprise Linux, macOS, and Windows—or the subset of those systems that you support. It is laborious to support multiple versions of an operating system, but that does not excuse avoiding these instructions for at least one version. Indicate a known working version of all of the dependencies, as well. Many scientists use computing clusters or network computers where they lack root privileges. When possible, your instructions should cover root and non-root installation.

Ensure you test installation on a new, unconfigured environment. A continuous integration service (see below) provides a great means for accomplishing this. If you use non-standard build tools or your software has complex dependencies, document the installation thoroughly and extensively. Sometimes it is easier for you to make installation easier for users. If your installation instructions seem very complex, consider ways to make it easier, perhaps by contributing your software to a package repository such as Debian Med\textsuperscript{8}, Homebrew\textsuperscript{9}, or the Comprehensive R Archive Network (CRAN)\textsuperscript{10}.

PLINK\textsuperscript{11} provides a good example of bioinformatics software supporting all major operating systems, with detailed instructions for each platform.
2.5 Readme and news

Provide a readme file at the top level of your source code with basic information about installation and use of your software, and details on where users can find more information. The readme should show up to users visiting your source code repository and will provide the first impression for many.

Also, provide a news section dedicated to the changes in each release of the software. Discuss bug fixes, caveats, new features, and changes in behavior of the software in detail. Users will often upgrade after a several new versions, and want a place to find the details of all that has changed since their last install. Include the news as another file in the top level of your source code and link to it from the readme.

2.6 File formats

If you must create a new file format (and please don’t, if you can avoid it), make sure to specify it in detail. Burying specification details in your code make operation with future software by others frustrating. A detailed specification, however, makes it easier to use your software in a larger pipeline, and reduce the chance you’ll have to debug interoperability problems later. The MEME Suite\textsuperscript{12} and PLINK\textsuperscript{13} both exemplify detailed description of input and output formats.

2.7 Communication with users

Users may need to contact you if they can’t find the answers they need in the documentation. Set up a mailing list to allow users to send questions and feedback. Archive the mailing list where search engines can find it. People who encounter an error, will report the message, allowing others to easily find the solution. Mailing lists facilitate an open development process, which may lead to users developing and submitting new features for your software. Some bioinformatics software packages, such as GATK\textsuperscript{14}, also host a forum which serves a similar purpose in making answers available to all. Forums, however, perform more poorly than mailing lists in getting others to contribute. They rely on those who answer the questions periodically checking the forum. Also, unlike mailing lists, forums are less effective as a means to communicate about new versions.

Issue trackers provide a great way to communicate about specific potential bugs or requests. GitHub\textsuperscript{15} and Bitbucket\textsuperscript{16} provide a free service for issue tracking, along with a repository for your code and documentation.

Adding a comment section to your documentation web page encourages users to contribute helpful feedback. So does Read the Docs\textsuperscript{17}, which makes it easy for users to submit a pull request correcting the documentation. If you receive repeated inquiries on one aspect of your software, this is evidence for insufficient documentation. Take this as a sign to revise the documentation.

MISO\textsuperscript{18}, ggplot2\textsuperscript{19}, and Bedtools\textsuperscript{1} provide detailed documentation in HTML format, have a public GitHub repository to track issues, and also have a mailing list for other communications with users.

2.8 Frequently asked questions (FAQ)

Prepare a frequently asked questions (FAQ) document to answer common questions you expect to, or have received. Many users find the FAQ format more compelling than a reference manual, and it is easier to link to an answer to a common question from a mailing list. PLINK has an FAQ which covers a variety of difficulties one may encounter before starting to use the software. It also includes questions that are related to unexpected outputs, and comparison with other packages.
2.9 Troubleshooting
Your software should provide meaningful warning and error messages when it receives unexpected input. Include a chapter in your documentation to thoroughly explain error and warning messages and how to resolve them. When the users search the internet for the text of these errors and warnings, they will find answers immediately.

3 Technical choices and software documentation

3.1 Programming environment
Using programming environments and languages that require difficult installation and configuration reduces the usability of your program, and they also require more complex documentation. For example, to run MATLAB programs without an expensive license, user must install a specific version of the MATLAB Compiler Runtime (MCR). Documenting all the things that can go wrong in installing an old version of MCR provides quite a challenge. This explains partially why few widely used bioinformatics tools rely on MATLAB.

3.2 Default parameters
Many users rely on your default parameters, so choose them carefully. Configuration options left to potentially inexpert users provide no substitute for sensible defaults. Document the rationale for selecting any default parameter. This will help users understand when they should change it.

3.3 Citation
Provide a citation to your own manuscript with a link to an open-access version. This makes it easier for users to find a description of your methodology and cite your work.

3.4 Writing code
At some point, the documentation will not answer every question. At this point, someone must examine the source code. Make it easy for that someone else to figure things out without help. That someone invariably, will end up being yourself sometime.

Put a premium on making your code easily intelligible to others. Use descriptive variable and function names following the standard format for your environment. PEP 8 supplies a format for Python and Google style guides provide them for other programming languages. Many text editors can check code style automatically.

3.5 Continuous integration of quick start and tests
Your quick start effectively provides a simple script on a small test dataset. Not only does this familiarize users with features of your software, but it also ensures that the software is installed properly and functions as expected.

You or other contributors can also use this script as a quick test to ensure that changes do not break any part of the software, or your instructions. You should therefore include the major options of your software in this script.

Consistent version control with Git or Mercurial helps you and collaborators track the development of the project and contribute easily. Using tools for coverage or mutation test of your code
and continuous integration services such as drone.io\textsuperscript{22} help you identify potential problems with your program faster.

4 Discussion

Effective documentation of bioinformatics software and adopting standard code style has specific importance in academia. Much academic software is developed by trainees who soon move on to other employment. Without good documentation, it becomes very difficult to continue developing or using the software. This results in premature abandonment of the software and a waste of the investment in the project. For this reason documentation can be even more important in academia than in industry, but much academic software remains under-documented.

Peer review of a bioinformatics software paper rarely assesses the software documentation directly. If the reviewers cannot figure out to run the software, however, this may result in rejection of the manuscript. The developer should ensure that described uses of their software remain reproducible. Long after the paper is accepted, published software remains part of the developer’s résumé and can affect their reputation.

When you lack the time to apply every guideline we propose, you should at least have the following minimum documentation:

1. GitHub or Bitbucket page with code and issue tracker.
2. Readme that covers installation, quick start, input formats, and output formats.
3. Reference manual with detailed description of every user-configurable parameter.

The Software Sustainability Institute’s online sustainability evaluation\textsuperscript{23} assesses how sustainable and reusable your software is. Many parts of this evaluation focus on adequate documentation. After following our other guidelines, we additionally recommend this evaluation for further detailed suggestions on creating great documentation.

Acknowledgments

We thank Aaron R. Quinlan and Timothy L. Bailey for giving us permission to use parts of their software documentation in figures, and their feedback on this manuscript. We also thank those who responded to Reddit\textsuperscript{24} and Twitter posts\textsuperscript{25,26} about best practices for and examples of great bioinformatics software documentation. This work was supported by the Canadian Cancer Society (703827 to M.M.H.), the Ontario Institute of Cancer Research (OICR), the Natural Sciences and Engineering Research Council of Canada (RGPIN-2015-03948 to M.M.H.), the Ontario Ministry of Research, Innovation, and Science (ER-15-11-223 to M.M.H.), the University of Toronto McLaughlin Centre (MC-2015-16 to M.M.H.), and the Princess Margaret Cancer Foundation.

References


