frontiers in
GENETICS

OPINION ARTICLE
published: 02 July 2014
doi: 10.3389/fgene.2014.00199

On best practices in the development of bioinformatics software

Felipe da Veiga Leprevost[1,2]*, Valmir C. Barbosa[3], Eduardo L. Francisco[2], Yasset Perez-Riverol[4] and Paulo C. Carvalho[1]

[1] Laboratory for Proteomics and Protein Engineering, Carlos Chagas Institute - Fiocruz, Curitiba, Brazil
[2] Hexabio Bioinformatics, Curitiba, Brazil
[3] Systems Engineering and Computer Science Program, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[4] Proteomics Services, European Molecular Biology Laboratory, European Bioinformatics Institute, Cambridge, UK
*Correspondence: felipe@leprevost.com.br

Edited by:
Max A. Alekseyev, George Washington University, USA

Reviewed by:
Son Pham, Salk Institute, USA
Jeremy Goecks, George Washington University, USA
Geir Kjetil Sandve, University of Oslo, Norway

Keywords: bioinformatics, best practices, source control, test, repository

1. INTRODUCTION

Bioinformatics is one of the major areas of study in modern biology. Medium- and large-scale quantitative biology studies have created a demand for professionals with proficiency in multiple disciplines, including computer science and statistical inference besides biology. Bioinformatics has now become a cornerstone in biology, and yet the formal training of new professionals (Perez-Riverol et al., 2013; Via et al., 2013), the availability of good services for data deposition, and the development of new standards and software coding rules (Sandve et al., 2013; Seemann, 2013) are still major concerns. Good programming practices range from documentation and code readability through design patterns and testing (Via et al., 2013; Wilson et al., 2014). Here, we highlight some points for best practices and raise important issues to be discussed by the community.

2. SOURCE-CODE AVAILABILITY TO REVIEWERS

It is debated among researchers whether source codes should be made available to reviewers, as doing so could allow for a more complete review and evaluation of the manuscript's results. It could also ultimately enable reviewers to demand quality and clarity in the same way as from manuscripts originating from laboratory experiments, in which a bad PCR or a Western-Blot without controls may lead to wrong interpretations of the results (Ince et al., 2012). In the case of software, a clear indication that best practices were not followed can bespeak carelessness and therefore indirectly signal that something may be wrong. It is our opinion that reviewing the source code from submitted papers should be possible if desired, though publishers would obviously have to search for even more specialized reviewers for the task. The review process does not necessarily need to be done at the code level but can be accomplished by evaluating the structure of the project, availability of test units, and functional tests. By organizing and providing tests with different case scenarios the authors can easily demonstrate how the software works and how it behaves in different occasions. The possibility of executing the code (without having to go deeply into it) and of looking into how particular issues are handled in the code is important at all stages of the work (both pre- and post-publication). Further inspection by the scientific community will eventually lead to the same advantages we see in open-source projects like the Linux kernel (Torvalds, 2014b) or the protocols used in the Internet. Bugs can be spotted and improvements suggested by the community. This is especially important because, as science is an ever changing enterprise, always adapting and growing, the opportunity is given for the software to evolve along with the field.

3. SOFTWARE INDEXING AND AVAILABILITY

A topic that we should address as a community is the possibility of indexing software with a solution like the well-known DOI system. An example of such an initiative is the combined work of the Mozilla Science Lab (Mozilla Foundation, 2014), GitHub (GitHub, 2014), and Figshare (Figshare, 2014). This would enable researchers and practitioners to easily keep track of different software versions, thereby facilitating access and deployment (Summers, 2014). Currently, it is common for bioinformatics software to be hosted by university or even personal or laboratory websites. Although they are convenient and provide users with quick access to the material in question, such solutions are also the source of a major problem in bioinformatics, namely the discontinuation of software availability. An ideal solution to this problem would be a central hosting repository where each version could be archived and made available. This would also help when old versions became necessary for old, third-party workflows. Another important aspect is the ability to prevent the deletion of previous versions of a project, which would also help prevent other projects from ceasing to exist after a certain time or being abandoned.

4. DOCUMENTING THE SOURCE CODE

Software documentation can be categorized into two groups, one targeted at

software developers, the other at the end users. The former is usually found in the source code, or is linked to it, and is used to explain the particularities of the code itself, which is important especially for software updating and customization. The latter typically uses nontechnical language and is aimed at aiding the user in the process of software installation and execution. Without proper code documentation the process of resolving a bug or including new developers in the team becomes a very complicated task. Users likewise need to have access to the documentation explaining its usage, which must include all directives for installation under different operating systems (when such is the case) and for the handling of parameters and input data prior to a run. It is also important to note that we need proper documentation for biologists, as they will be the ones installing and using the programs. With easy-to-follow guidelines and instructions for non-programmers, it is possible to improve software usability.

5. SOURCE-CODE MANAGEMENT

During a software's life cycle, a varying number of developers can be involved with its production and different versions of it can be created. One of the main goals of having source-code management is to have all these aspects automatically taken care of through the building of a historical registry of development. Solutions such as Git allow the simultaneous collaboration with several projects while greatly simplifying each maintainer's tasks of tracking and resolving bugs, handling feature requests, and launching upgrades (Torvalds, 2014a). This also helps to promote the collaborative aspect of software development since anyone can join an ongoing project and provide patches.

6. TEST LIBRARIES, SAMPLE DATA, AND DATASET REPOSITORIES

A test library is a series of scripts designed to test a given piece of software. It is meant to aid in quickly determining whether the software's main modules are working as expected. Ideally, all functions of the code should be thus tested, but sometimes this is not possible because of the size or complexity of the project. What is fundamental to test, though, is whether the main logic and operations are working correctly whatever the running environment happens to be. Normally a test library is shipped together with the software and the tests are executed before installation to certify that the main features are working on the machine at hand. Another important aspect of any scientific software is that sample data be provided along with it, in a manner similar to that in which supplementary files are provided together with a manuscript. Through "real-world" examples, users can verify what to expect of the various analyses. Such examples also allow for comparisons with other datasets (Perez-Riverol et al., 2014).

7. THE ADVANTAGES OF THE OPEN-SOURCE DEVELOPMENT

There are several advantages to making a software project open source (Perez-Riverol et al., 2014). In computer science, projects are usually classified into two major categories: open source and proprietary. Being open source means making the code freely available, a simple gesture that can have powerful implications for user projects, especially those that are science-related. One of the greatest advantages of an open-source program is that it is possible to see and understand all functionalities and every calculation it does, thus ensuring full transparency. The same cannot be said of proprietary software, in which case users are required, essentially, to have faith in the product's developer/seller and become unable to criticize or properly know how results are obtained. In general, open source means a greater tendency toward reliability, as anyone can peruse the source code and eventually spot some bug. As such, an open-source project is continually reviewed by the community. When someone spots an error and then corrects it, a patch can be generated and sent to the code maintainer. One of the key aspects of having an open-source project is to provide clarity about how results are generated and can be reproduced (Prli and Procter, 2012).

8. FINAL CONSIDERATIONS

During the development phase of a software project, adopting best practices in programming involves investing time and effort to better structure ideas as both the code and the documentation are written. Although such investment may at times seem cumbersome, in the long run it benefits both developers and users, and is therefore valuable. In a related vein, another crucial issue is trustworthiness: from the perspective of the scientists using it, a software tool abiding by good practices can provide more confidence as their own projects are developed, which in turn is a key aspect of any work based on data analysis. All of this point in the direction of the software having more quality, since ultimately, quality depends on programming practices. The more quality a software has, the longer it will live and the more people will use it (Altschul et al., 2013). In this regard, a noteworthy initiative is the GMOD Galaxy, an open and integrated workflow system which allows the sharing of customized analyses (Giardine, 2005). Other examples of softwares following the best practices listed above are Tophat (Trapnell et al., 2009), Bowtie (Langmead et al., 2009), and the BioPerl project (Stajich, 2002).

ACKNOWLEDGMENTS

Felipe da Veiga Leprevost, Valmir C. Barbosa, and Paulo C. Carvalho are supported by Capes and CNPq; Valmir C. Barbosa is supported by the FAPERJ BBP grant; Yasset Perez-Riverol is supported by the BBSRC PROCESS grant [reference BB/K01997X/1].

REFERENCES

Altschul, S., Demchak, B., Durbin, R., Gentleman, R., Krzywinski, M., Li, H., et al. (2013). The anatomy of successful computational biology software. *Nat. Biotechnol.* 31, 894–897. doi: 10.1038/nbt0614-592b

Figshare (2014). *Figshare - Credit for All Your Research.* Available online at: http://figshare.com/

Giardine, B. (2005). Galaxy: a platform for interactive large-scale genome analysis. *Genome Res.* 15, 1451–1455. doi: 10.1101/gr.4086505

GitHub (2014). *Github*. Available online at: http://github.com/

Ince, D. C., Hatton, L., and Graham-Cumming, J. (2012). The case for open computer programs. *Nature* 482, 485–488. doi: 10.1038/nature10836

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.* 10:R25. doi: 10.1186/gb-2009-10-3-r25

Mozilla Foundation. (2014). *Mozilla Science Lab*. Available online at: http://mozillascience.org/

Perez-Riverol, Y., Hermjakob, H., Kohlbacher, O., Martens, L., Creasy, D., Cox, J., et al. (2013). Computational proteomics pitfalls and challenges: havanabioinfo 2012 workshop report. *J.*

Proteomics 87, 134–138. doi: 10.1016/j.jprot.2013.01.019

Perez-Riverol, Y., Wang, R., Hermjakob, H., Mller, M., Vesada, V., and Vizcano, J. A. (2014). Open source libraries and frameworks for mass spectrometry based proteomics: a developer's perspective. *Biochim. Biophys. Acta.* 1844(1 Pt A), 63–76. doi: 10.1016/j.bbapap.2013.02.032

Prli, A., and Procter, J. B. (2012). Ten simple rules for the open development of scientific software. *PLoS Comput. Biol.* 8:e1002802. doi: 10.1371/journal.pcbi.1002802

Sandve, G. K., Nekrutenko, A., Taylor, J., and Hovig, E. (2013). Ten simple rules for reproducible computational research. *PLoS Comput. Biol.* 9:e1003285. doi: 10.1371/journal.pcbi.1003285

Seemann, T. (2013). Ten recommendations for creating usable bioinformatics command line software. *Giga Sci.* 2:15. doi: 10.1186/2047-217X-2-15

Stajich, J. E. (2002). The bioperl toolkit: perl modules for the life sciences. *Genome Res.* 12, 1611–1618. doi: 10.1101/gr.361602

Summers, N. (2014). *Mozilla Science Lab, Github and Figshare Team up to Fix the Citation of Code in Academia.* Available online at: thenextweb.com/dd/2014/03/17/mozilla-science-lab-github-figshare-team-fix-citation-code-academia/

Torvalds, L. (2014a). *Git*. Available online at: http://git-scm.com

Torvalds, L. (2014b). *The Linux Kernel Project.* Available online at: http://kernel.org

Trapnell, C., Pachter, L., and Salzberg, S. L. (2009). Tophat: discovering splice junctions with rna-seq. *Bioinformatics* 25, 1105–1111. doi: 10.1093/bioinformatics/btp120

Via, A., Blicher, T., Bongcam-Rudloff, E., Brazas, M. D., Brooksbank, C., Budd, A., et al. (2013). Best practices in bioinformatics training for life scientists. *Brief Bioinform.* 14, 528–537. doi: 10.1093/bib/bbt043

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., et al. (2014). Best practices for scientific computing. *PLoS Biol.* 12:e1001745. doi: 10.1371/journal.pbio.1001745

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 24 April 2014; accepted: 13 June 2014; published online: 02 July 2014.

Citation: Leprevost FV, Barbosa VC, Francisco EL, Perez-Riverol Y and Carvalho PC (2014) On best practices in the development of bioinformatics software. Front. Genet. ***5****:199. doi: 10.3389/fgene.2014.00199*

This article was submitted to Bioinformatics and Computational Biology, a section of the journal Frontiers in Genetics.