Ten Simple Rules for taking advantage of GitHub in bioinformatics

Yasset Perez-Riverol (1)[1], Rui Wang (1), Timo Sachsenberg (2), Julian Uszkoreit (3), Laurent Gatto (4), Felipe da Veiga Leprevost (5), Christian Fufezan (6), Juan Antonio Vizcaíno (1)[2]

(1) European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SD, UK.

(2) Applied Bioinformatics and Department of Computer Science, University of Tübingen, D-72074 Tübingen, Germany.

(3) Medizinisches Proteom-Center, Ruhr-Universität Bochum, Universitätsstr. 150, D-44801 Bochum, Germany

(4) Computational Proteomics Unit, Cambridge Systems Biology Centre, University of Cambridge Tennis Court Road Cambridge, CB2 1GA, UK

(5) Department of Pathology, University of Michigan, Ann Arbor, Michigan 48109, USA

(6) Institute of Plant Biology and Biotechnology, University of Muenster, Schlossplatz 8, 48143 Muenster, Germany

Short title: Ten Simple Rules for taking advantage of GitHub in bioinformatics

Introduction

Bioinformatics is a broad discipline in which the common denominator is the need to produce and/or use software that can be applied to biological data in different contexts. For instance software is routinely needed for the analysis, visualisation, integration or storage of biological information. To enable and ensure the reproducibility of scientific claims, it is essential that, the scientific publication, the corresponding datasets and the data analysis are made publicly available to the scientific community (Goodman et al. 2014; Perez-Riverol et al. 2015). In addition, all the software used for the analysis should be either well described (e.g. in case of using commercial software) or, better, openly shared and directly accessible to others (Osborne et al. 2014; Vihinen 2015). At present the latter is becoming more common in the form of of *home made* scripts or more fully fledged bioinformatics open source projects (Leprevost et al. 2014). The rise of openly available software and source code and concomitant collaborative development is

[1]Corresponding author: Email: yperez@ebi.ac.uk.
[2]Corresponding author: Email: juan@ebi.ac.uk.

facilitated thanks to the existence of several code repository-hosting services such as SourceForge (http://sourceforge.net/), Bitbucket (https://bitbucket.org/) and GitHub (https://github.com/), among others. These resources are also essential for collaborative software projects, since they enable the organisation and sharing of programming tasks between different contributors to the same project. Here we aim to introduce the main features of GitHub, a popular web-based platform which offers a free and integrated environment for hosting the source code, documentation and web page for open source projects. One reason for GitHub's success is that it offers more than a simple source code hosting service. It provides developers with a dynamic and collaborative environment, often coined as social coding platform, with the ability to review, comment and discuss code (Dabbish et al. 2012). The cornerstone of GitHub is the well-known and open source version control system git, designed and developed by Linus Torvalds in 2005 to control and extend the Linux kernel development. It has become the most widely adopted version control system, used by major companies such as Google, Facebook, or Twitter, among others. GitHub is free of charge for public projects, therefore hosting millions of open source projects. In addition, it offers paid plans for private repositories. Some of our recommendations outlined below would be applicable to other hosting services as well. However our main aim here is to highlight specific GitHub features. We will then provide a set of recommendations for taking full advantage of its core features to facilitate and manage the work on a given bioinformatics project and increase its profile and visibility in the scientific community. Below, we present a set of rules that will help scientists and research software engineers to efficiently make use of GitHub. The rules have been ordered to reflect a typical development process: learning git and GitHub basics, use branches, label and tag code, track the project using issues, etc.

Rule 1. Structure your projects: users, organisations, repositories and teams

Open source projects on GitHub are visible to everyone, but write permissions, i.e. the possibility to directly modify the content, need to explicitly be granted. On the other hand, everyone with a GitHub account can fork any public project and start developing in ones own fork. This forking is the basis of social coding; it allows anyone to develop and test novel features into existing code and offers the possibility to merge novel features back the into the main project, thereby becoming a contributor. Structuring your projects allows to manage permissions and restrict access at different levels: users, teams and organisations. Users are the keystone of GitHub, as for any other social network. Every user has a profile listing their GitHub projects and activities, which can be populated optionally with personal information including name, e-mail address, image and webpage. To stay up to date with the activity of other users you can *follow* their accounts. Collaboration can be achieved by simply adding a trusted *Collaborator* and thereby granting write access. However, development in large projects is usually

done by teams of people, within a larger organisation. GitHub organisations are a great way to manage team-based access permissions for the individual projects of institutes, research labs, and large open source projects that need multiple owners and administrators (Figure 1). We recommend that you (as an individual researcher) make your profile visible to other users and display all the projects and organisations you are working in, including a list of the latest activities on the site (Figure 1). Finally, repositories (the shortened term is *repo*) are versioned directories or dedicated storage spaces for your software projects, which can be included inside an organisation or can belong to particular users. Users can usually keep code, text files, images and small data files inside a repo. And while many users store programs and code projects, there is nothing preventing you from keeping text documents such as analysis reports and manuscripts, or other file types in your projects. Note that until recently, GitHub was lacking support for storing large files (>100 MB), a issue that has been recently addressed by the GitHub large file storage. However, please be aware that large files will make the cloning and forking problematic and post-processing hooks (Rule 6) might suffer from unnecessary files.

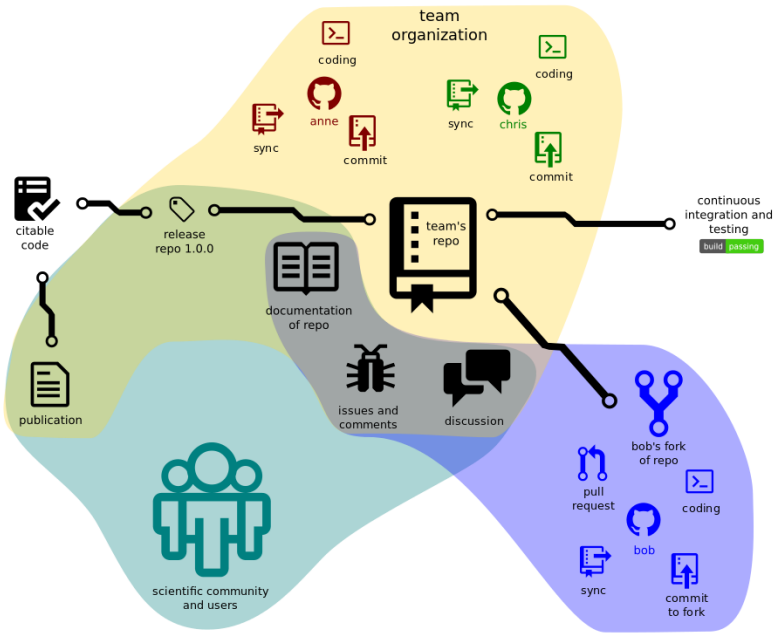

Figure 1: The structure of a GitHub-based project illustrating the projects structure and the interactions with the community.

Rule 2. Learn Git and embrace its power

The cornerstone of GitHub is the distributed version control system git. Every change, from fixing a typo to a complete redesign of the software is controlled by versions, so called revisions. While beginners may consider the learning curve of Git steep, many introductory and detailed tutorials are available. A revision can be considered as a *snapshot* (version) of a file system. Git is remarkably effective in archiving the complete history of a project (all revisions) by, amongst other things, storing only the differences between them. To create a new revision, the set of changes (e.g. new, deleted or modified files) introduced are committed to the repository. Following the rule: "commit often, as most as you can, perfection later", one can keep track of the development in small incremental changes. At any time it is possible to go back to a previous commit. In larger projects, multiple users contribute to the same repository.

Rule 3. Use Branches

The management of concurrent developments with commits to the same repository can be approached using several common approaches. The most common way is to use git *branches* to separate different lines of development. Active development is often performed on a development branch and stable versions as e.g. those used for a software release are kept in a master branch. In practice, developers work on one or several features or improvements. To keep commits of the different features logically separated, distinct branches are typically used. Later, when development is complete and none of the tests fail (see Rule 6) new features can be merged back into the development line or master branch. During such a development, the original branch might continuously be developed and other features might be merged into the master branch. Nevertheless, one can always pull the currently up-to date master branch into once fork, always allowing to react or adapt to the changes in the code. In projects involving more than one contributor, everyone wants to be sure that the contributions of others increase the quality and move the project forward. *Forking* a repository and providing *pull requests* constitute an easy way for collaboration inside and over organisations boundaries. A user that forks a repository creates a copy under their GitHub account. Modifications like a branch with new features or bug fixes can conveniently be provided to the forked (upstream) repository by opening a pull request. Once it is opened for review and discussion, it usually results in additional insights and in an increased code quality. Once a pull request gets accepted, typically it gets merged into the development branch.

Rule 4. Use Tags and semantic version numbering

Tags offer the possibility to label versions during development progress. Version numbering should follow semantic versioning in the form X.Y.Z, with X being

the major, Y the minor and Z the patch version of the release, including possible meta information, as described in http://semver.org/. Correct labelling allows developers and users to easily recover older versions, compared them, or simply used them to reproduce results described in publications (Rule 8). This approach will also help defining a coherent software publication strategy (Rule 6).

Rule 5. The code must always be ready to use: continuously integrate

The first rule of software development is that the code needs to be ready to use as soon as possible (Leprevost et al. 2014), remain so during development, and should be well-documented and tested. In 2005, Martin Fowler defined the basic principles for continuous integration in software development (Fowler 2006). These principles have become the main reference for continuous integration best practices, and provide the framework needed to deploy software, and in some way also data. Every repository, script, mathematical model, and function should contain a set of self-automated tests. A source code may run, but that does not mean it is doing the right thing. The simple use of those self-automated tests is to detect possible bugs introduced by new features, or changes in the code or dependencies, but also to detect wrong results, the so called *logic errors*, where the source code produces a different result compared to what you intended it to do. Then, continuous integration provides the way of automatically run all of these tests in the repository by checking data and software dependencies. Continuous integration can be done automatically on GitHub (See Rule 6).

Rule 6: Automate for better code quality

GitHub offers different type of hooks that are executed after each push to the repository, making it easier to follow the basic principles for continuous integration. The GitHub web hooks allows third-party platforms to access and interact with your GitHub repository and thus automate post-processing tasks. Such task demonstrate to the community that your project follows rigorous software engineering processes, often associated with high quality development, is currently working and has documentation that reflects the current code. We suggest that all three tasks become part of your project. Firstly, continuous integration can be achieved by *Travis* (https://travis-ci.org), a hosted continued integration platform that is free for all open source projects. Travis builds and tests the source code using a plethora of options such as different platforms and interpreter versions. Furthermore, it offers notification which allow your team and contributors to know if new changes work and prevent the introduction of errors in the code, making the repo always ready to use. Secondly, in addition to successful completion, you can also demonstrate that the tests cover your code base sufficiently, the integration of *Codecov* is recommended (https://codecov.io). Thirdly, you might consider to automatically update the documentation upon

code modification. This implies that your project provides comprehensive documentation so others can understand, and contribute back to your projects.

For Python or C/C++ code, auto documentation generation can be done using sphinx (http://sphinx-doc.org/) and auto integration into GitHub using read the docs (https://readthedocs.org/). All of these platforms will create reports and badges for your project that you can include on your GitHub page, thereby making your project easily identifiable as a high quality and maintained project.

Rule 7. Use and maintain your GitHub issue trackers

GitHub *issues* are a great way to keep track of bugs, tasks, and enhancements. Classical issue trackers are primarily intended to be used as bug trackers. In contrast, GitHub issues follow a different philosophy: each tracker has its own section in every repository, and can be used to trace bugs, new ideas, and enhancements, by using a powerful but optional tagging system for each issue. Its main focus is put in promoting collaboration, providing context by using cross-references, and an excellent text formatting for each issue: (i) a title and description, (ii) colour-coded labels help you to categorise and filter issues, (iii) milestone acts like a container for issues (e.g. weekly discussion related with datasets), (iv) one assignee responsible for working on the issue, and (v) comments that allow anyone with access to the repository to provide feedback. Another aspect is its simplicity. For instance, it does not require to fill lengthy forms including every piece of information that might be valuable to reproduce the bug. It only requires the user to give the title and provide some optional text. If the developer needs more information, they can simply request it in a comment. GitHub issues are then more dynamic and pose a lower barrier for users to report bugs and request features. A well-organised and tagged issue tracker will help upcoming contributors and users to understand the project more deeply.

Rule 8. Make your code easily citable, and cite source code!

In research, it is a good practice to ensure permanent and unambiguous identifiers for citable items like articles, datasets, or biological entities such as proteins, genes and metabolites, among others. Digital Object Identifiers (DOIs) have been used for many years as unique and unambiguous identifiers for enabling the citation of scientific publications. More recently, a trend has started to produce DOIs for other types of scientific outputs such as datasets (Vizcaíno et al. 2014) or training materials (for example (Ahmadia et al. 2015)). The main motivation behind this is to give scientists a better credit for their work ("Credit Where Credit Is Overdue." 2009), enabling at the same time a better way to cite and track it. A common issue with software is that it normally evolves at a different speed than what is published in the scientific literature. In fact, it is common to find software having novel features and functionalities that were not described in

the original publication. GitHub now enables the use of DOIs to cite the code deposited, using the data archiving tool Zenodo (https://zenodo.org/). The procedure is simple (see https://guides.GitHub.com/activities/citable-code/) and, by default, Zenodo takes an archive of a repository each time a new release is created in GitHub. However, before Zenodo can issue a DOI, metadata need to be provided about the archived repository. Once the DOI has been assigned, apart from using it in your CV, you can add it to literature information resources such as Europe PubMed Central (Europe PMC Consortium 2015).

As already mentioned in the introduction, reproducibility of scientific claims should be enabled by providing openly the software, the datasets and the process leading to interpretable results that are used in a particular study. You should always highlight as much as possible in your publications that the code is freely available in, for example, GitHub, together with any other relevant piece of information that may have been deposited. In our experience, this openness substantially increases your chances of getting the paper accepted for publication. On one hand, journal editors and reviewers have the opportunity to reproduce your findings during the manuscript review process, increasing the confidence of your results. On the other hand, once the paper is published, your work can be reproduced by any member of the scientific community, which can increase citations and foster opportunities for further discussion and collaboration. Also have in mind that a public repository with available source code does not make the software open source *per se*, as it needs to have an appropriate license (http://opensource.org/licenses/).

Rule 9. Promote your projects in the scientific community - create a web page and more

Additional steps can be done to boost the visibility of your organisation. For example, GitHub *Pages* are simple landing webpages that GitHub hosts for free without the need for a server or database. GitHub users can create and host blog websites, help pages, manuals, tutorials and websites related to specific projects. *Pages* comes with a powerful static site generator called Jekyll (https://jekyllrb.com) that can be integrated with other platforms such as Bootstrap (http://getbootstrap.com/) or Disqus (https://disqus.com/), to support and moderate comments.

In addition, GitHub also provides mechanisms for real-time communication called Gitter (http://gitter.im). Gitter is a GitHub-based chat tool (in limited beta at the time of writing) which enables developers and users to share aspects of their work. Gitter inherits the shape of the social groups operating around GitHub repositories, organisations, and issues. It relies on the identity within GitHub, creating IRC (Internet Relay Chat)-like chat rooms for public and private repositories. From within a Gitter chat, members can reference issues, comments, or pull-requests. A different service is Gist (https://gist.github.com), which represents a unique way to share *code snippets*, single files, parts of files,

or full applications. Gist can work in two ways: public *gists*, that can be browsed and searched, and secret gists that are not provided through *Discover* (https://gist.github.com/discover). One of the main features of Gist is the possibility to embed code snippets in other applications, enabling users to embed gists in any text field that supports JavaScript.

Rule 10. Check periodically existing open source projects

One of the main tasks of scientists is to follow the developments in their field. Analogously, scientific programmer need to revise publicly available projects and code that can be interesting for their research. You should try to learn as much as possible from your peers and colleagues and keep up-to-date with all the developments of a relevant. GitHub enables this functionality by *following* other GitHub users or *watching* the activity of projects, which is a common feature in many social media platforms. Take advantage of it as much as possible!

Conclusions

If you are interested and have not used GitHub so far, we recommend you to get started as soon as possible. As in any other topics, a learning curve is required for beginners. However, we anticipate the reward will be worth your effort.

Acknowledgements

Y.P.R is supported by the BBSRC *PROCESS* grant [reference BB/K01997X/1] and by the BBSRC *Quantitative Proteomics* grant [reference BB/I00095X/1]. R.W. is also funded by grant BB/I00095X/1. J.A.V. is supported by the Wellcome Trust [grant number WT101477MA]. J.U. and T.S. are funded by the BMBF grant de.NBI - German Network for Bioinformatics Infrastructure (FKZ031 A 534A). L.G. is supported by the BBSRC Strategic Longer and Larger grant (Award BB/L002817/1). F.V.L. is supported by NIH grant number R01-GM-094231.

Disclaimer

The authors have no affiliation with GitHub, nor any commercial entity mentioned in this article. The views described here reflect our owns without any input from any third party organisation.

References

Ahmadia, Aron, Matthew Aiello-Lammens, Joshua Ainsley, James Allen, Areej Alsheikh-Hussain, Piotr Banaszkiewicz, Diego Barneche, et al. 2015. "Software Carpentry: Programming with R." doi:10.5281/zenodo.27353. http://dx.doi.org/10.5281/zenodo.27353.

"Credit Where Credit Is Overdue." 2009. *Nat Biotechnol* 27 (7): 579. doi:10.1038/nbt0709-579.

Dabbish, Laura, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository." In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, 1277–86. CSCW '12. New York, NY, USA: ACM. doi:10.1145/2145204.2145396. http://doi.acm.org/10.1145/2145204.2145396.

Europe PMC Consortium. 2015. "Europe PMC: a Full-Text Literature Database for the Life Sciences and Platform for Innovation." *Nucleic Acids Res* 43 (Database issue): D1042–8. doi:10.1093/nar/gku1061.

Fowler, Martin. 2006. "Continuous Integration." http://www.martinfowler.com/articles/continuousIntegration.html.

Goodman, A, A Pepe, A W Blocker, C L Borgman, K Cranmer, M Crosas, R Di Stefano, et al. 2014. "Ten Simple Rules for the Care and Feeding of Scientific Data." *PLoS Comput Biol* 10 (4): e1003542. doi:10.1371/journal.pcbi.1003542.

Leprevost, Fda V, V C Barbosa, E L Francisco, Y Perez-Riverol, and P C Carvalho. 2014. "On Best Practices in the Development of Bioinformatics Software." *Front Genet* 5: 199. doi:10.3389/fgene.2014.00199.

Osborne, J M, M O Bernabeu, M Bruna, B Calderhead, J Cooper, N Dalchau, S J Dunn, et al. 2014. "Ten Simple Rules for Effective Computational Research." *PLoS Comput Biol* 10 (3): e1003506. doi:10.1371/journal.pcbi.1003506.

Perez-Riverol, Y, E Alpi, R Wang, H Hermjakob, and J A Vizcaíno. 2015. "Making Proteomics Data Accessible and Reusable: current State of Proteomics Databases and Repositories." *Proteomics* 15 (5-6): 930–49. doi:10.1002/pmic.201400302.

Vihinen, M. 2015. "No More Hidden Solutions in Bioinformatics." *Nature* 521 (7552): 261. doi:10.1038/521261a.

Vizcaíno, J A, E W Deutsch, R Wang, A Csordas, F Reisinger, D Ríos, J A Dianes, et al. 2014. "ProteomeXchange Provides Globally Coordinated Proteomics Data Submission and Dissemination." *Nat Biotechnol* 32 (3): 223–6. doi:10.1038/nbt.2839.