# Ten Simple Rules for Taking Advantage of git and GitHub

Yasset Perez-RiverolYasset Perez-Riverol[1,*] Laurent Gatto[2], Rui Wang[1], Timo Sachsenberg [3], Julian Uszkoreit[4], Felipe da Veiga Leprevost[5], Christian Fufezan[6], Tobias Ternent[1], Stephen J. Eglen[7], Daniel S. Katz[8], Tom Pollard[9], Alexander Konovalov[10], Juan Antonio Vizcaíno[1,*]

**1 European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SD, UK.**
**2 Computational Proteomics Unit, Cambridge Systems Biology Centre, University of Cambridge, Tennis Court Road, Cambridge, CB2 1GA, UK.**
**3 Applied Bioinformatics and Department of Computer Science, University of Tübingen, D-72074 Tübingen, Germany.**
**4 Medizinisches Proteom-Center, Ruhr-Universität Bochum, Universitätsstr. 150, D-44801 Bochum, Germany.**
**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.**
**7 Centre for Mathematical Sciences, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK.**
**8 National Center for Supercomputing Applications, University of Illinois, 1205 W. Clark St., Urbana, Illinois 61801, USA.**
**9 MIT Laboratory for Computational Physiology, Institute for Medical Engineering and Science, Massachusetts Institute of Technology, E25-545B, 45 Carleton St, Cambridge, MA 02142, USA.**
**10 Centre for Interdisciplinary Research in Computational Algebra, University of St Andrews, St Andrews, Fife, KY16 9SX, UK.**

\* yperez@ebi.ac.uk and juan@ebi.ac.uk

## 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. To enable and ensure the replicability and traceability of scientific claims, it is essential that the scientific publication, the corresponding datasets and the data analysis are made publicly available [1,2]. All software used for the analysis should be either carefully documented (e.g., for commercial software) or, better, openly shared and directly accessible to others [3,4]. The rise of openly available software and source code, and concomitant collaborative development is facilitated by the existence of several code repository services such as SourceForge (http://sourceforge.net/), Bitbucket (https://bitbucket.org/), GitLab (https://about.gitlab.com/) and GitHub (https://github.com/), among others. These resources are also essential for collaborative software projects, since they enable the organization and sharing of programming tasks between different remote contributors. Here, we introduce the

main features of GitHub, a popular web-based platform that offers a free and    15
integrated environment for hosting the source code, documentation, and          16
project-related web content for open source projects. GitHub also offers paid plans for    17
private repositories (see Box 2) for individuals and businesses, as well as free plans    18
including private repository for research and educational use.                   19

GitHub relies, at its core, on the well-known and open source version control system    20
git, designed by Linus Torvalds for the development of the Linux kernel. Nowadays,    21
git is developed and maintained by the git community (https://github.com/git). One    22
reason for GitHub's success is that it offers more than a simple source code hosting    23
service [5,6]. It provides developers and researchers with a dynamic and collaborative    24
environment, often referred to as a social coding platform, with the ability to review,    25
comment and discuss code [7]. A diverse range of efforts, ranging from individual to    26
large bioinformatics projects, lab repositories, as well as global collaborations found    27
the perfect place in GitHub to share code, ideas and collaborate (see Table 1).    28

Some of our recommendations outlined below are applicable to other hosting    29
services. However our main aim here is to highlight specific GitHub features. We    30
provide a set of recommendations to take full advantage of GitHub's features to    31
manage small and large bioinformatics projects and increase their profile and visibility.    32
These rules are ordered to reflect a typical development process: learning git and    33
GitHub basics, collaboration, use of branches and pull requests, labelling and tagging    34
of code snapshots, tracking project bugs and enhancements using issues, and    35
dissemination of the final results.    36

## Rule 1. Use GitHub to track your projects    37

The backbone of GitHub is the distributed version control system git. Every change,    38
from fixing a typo to a complete redesign of the software, is tracked and controlled by    39
individual and unique identifier. While git has a complex set of commands and can be    40
used for rather complex operations, learning and apply the basics only requires a    41
handful of new concepts and commands, and will provide a solid ground to efficiently    42
track your coding and research projects. Many introductory and detailed tutorials are    43
available (see Table 1 below for a few examples). In particular, we recommend *A    44
Quick Introduction to Version Control with Git and GitHub* by Blischak *et al.* [5].    45

In a nutshell, initialising a (local) repository (often abbreviated *repo*) marks a    46
directory as one to be tracked (Fig. 1). All or parts of its content can be added    47
explicitly to the list of files to track.    48

```
cd project ## move into directory to be tracked                                  49
git init   ## initialise local repository                                        50
## add individual files such as project description, reports, source code        51
git add README project.md code.R                                                 52
git commit -m "initial commit" ## saves the current local snapshot               53
```

From now on, every change to these tracked files, once committed, will be recorded    54
as a new revision, or *snapshot*, uniquely identifying the changes in all the modified    55
files. Git is remarkably effective in archiving the complete history of a project (all    56
revisions) by, among other things, storing only the differences among them.    57

In addition to local copies of the repository, it is easy to create remote repositories    58
on GitHub (called `origin`, with default branch `master` - see below) using the web    59
interface, and then synchronise local and remote repositories.    60

```
git push origin master ## push local changes to the remote repo                  61
git pull origin master ## pull remote changes into the local repo                62
```

Following Tony Rossini's advice in 2005 to "commit early, commit often, and commit in a repository from which we can easily roll-back your mistakes", one can keep track of the development in small incremental changes. At any time it is possible to go back to a previous version. In larger projects, multiple users contribute to the same remote repository and all their contributions are recorded, attributed and can be restored.

Users usually track source code, text files, images and small data files inside their repos, and generally do not track derived files such as build logs or compiled binaries. And while the majority of GitHub repositories are used for software development, there is nothing preventing users from keeping text documents such as analysis reports and manuscripts (see for example the repository for this manuscript at https://github.com/ypriverol/github-paper), or other file types in your projects.

Due to its distributed design, every local git repository with the latest commit is an entire exact historical copy of everything that was committed - file changes, commit message logs, etc. These copies act as independent backups as well, present on each user's storage device. Git can be considered to be fault-tolerant because of this, which is a win over centralized version control systems. If the remote GitHub server is unavailable, collaboration and work can continue between users, as opposed to centralized alternatives.

The web interface offered by GitHub provides friendly tools to perform many basic operations and a gentle introduction to a more rich albeit complex set of functionalities. There exists also various graphical user-interface driven clients to manage git and GitHub repositories (https://www.git-scm.com/downloads/guis). Many editors and development environments such as, for example, the popular Rstudio editor (https://www.rstudio.com/) for the R programming language [8], directly integrate with code versioning using git and GitHub. In addition, for remote git repositories GitHub provides its own features that will be described in subsequent rules (Fig. 1).

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

**Box 1** Using GitHub, or any similar versioning/tracking system is not a replacement for good projects management; it is an extension, an improvement for good project and file managing (see for example [9]). One practical consideration, when using GitHub, for example, is dealing with large binary files. Binary files such as images, videos, executable files or many raw data used in bioinformatics, are stored as a single large entity in git. As a result, every change, even if minimal, leads to a complete new copy of the file in the repository, producing large size increments and the inability to search (see https://help.github.com/articles/searching-code/) and compare file content across revisions. Git offers a Large File Storage (LFS) module (https://git-lfs.github.com/) that replaces such large files with pointers, while the large binary file can be stored remotely, which results in small and faster repositories. Git LFS is also supported by GitHub, albeit with a space quota or for a fee, to retain your usual GitHub workflow (https://help.github.com/categories/managing-large-files/) (Supplementary Note, Section 1).

**Box 2** By default, GitHub repositories are freely visible to all. Many projects decide to share their work publicly and openly right from the early stage of projects, in order to attract visibility and to benefit from

contributions from the community early on. However, many groups prefer to work privately on projects until they are ready to share their work. Private repositories ensure your work is hidden but equally limit collaborations to just those users that are given access to the repository. These repositories can then be made public at a later stage, such as, for example, upon submission, acceptance, or publication of corresponding journal articles. In some cases, when the collaboration was exclusively meant to be private, some repositories might never be made publicly accessible.

**Box 3** Every repository should ideally have the following three files. The first, and arguably most important file in a repository is a LICENCE file (see also Rule 8), that clearly defines the permissions and restrictions attached to the code and other files in your repository. The second important file is a README file, which provides, for example, a short description of the projects, a quick tutorial start guide, information on how to contribute, a TODO list, and links to additional documentation. Such README files are typically written in markdown, a simple markup language that is automatically rendered on GitHub. Finally, adding a CITATION file to the repository will inform your users how to cite and credit your project.

## Rule 2. GitHub for single users, teams and organizations

Public projects on GitHub are visible to everyone, but write permissions, i.e. the possibility to directly modify the content of a repo, need to be granted explicitly. As a repository owner, you can grant these rights to other GitHub users. In addition to being owned by users, repositories can also be created and managed as part of teams and organization.

Project managers can structure projects to manage permissions at different levels: users, teams and organizations. Users are the central element of GitHub, as in 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 one can *follow* their accounts (see also Rule 10). 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 organization. GitHub organizations 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 (Fig. 1). We recommend that you (as an individual researcher) make your profile visible to other users and display all of the projects and organisations you are working in, including a list of the latest activities on the site (Fig. 1).

## Rule 3. Developing and collaborating on new features: branching and forking

Anyone with a GitHub account can *fork* any public repository and start developing the same code base in one's own fork (https://help.github.com/articles/fork-a-repo/) under their username (see for example

https://github.com/ypriverol/github-paper/network/members for this work) or  <sub>156</sub> organization (see Rule 2). A fork is a complete copy of a repository content that retains a link to the original repository. Forking a repository allows you to freely experiment with changes without affecting the original project. This forking is the basis of social coding. It allows anyone to develop and test novel features with existing code and offers the possibility of contributing (by opening *pull requests*) novel features, bug fixes, and improvements to documentation (https://twitter.com/rgfitzjohn/status/708309767240982528) back into the original (upstream) project, thereby improving the original repository and becoming a contributor. Forking a repository and providing pull requests constitute an easy method for collaboration inside loosely defined teams and over more formal organizational boundaries, while the original repository owner(s) retain control over which external contributions to include. Once a pull request is opened for review and discussion, it usually results in additional insights and increased code quality [7].

Concurrent development - that is, users making commits in parallel to the same repository - can be organised using different 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 e.g. those used for a software release, are kept in a master or release branch (see for example https://github.com/OpenMS/OpenMS/branches). In practice, developers often work concurrently 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 5), new features can be merged back into the development line or master branch. In addition, one can always pull the currently up-to-date master branch into a feature branch, to adapt the feature to the changes in the master branch.

When developing different features in parallel, there is a risk to apply incompatible changes in different branches/forks; these are said to become out of sync. Branches are just short-term departures from master. If you pull frequently, you will keep your copy of the repo up to date, and you will have the opportunity to merge your changed code with others' contributors without requiring to manually address conflicts to bring the branches in sync again.

## Rule 4. Naming branches and commits: tags and semantic versions

Tags offer the possibility of labeling versions during the development process. 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/. This semantic versioning scheme provides users with coherent version numbers that document the extent (bug fixes or new functionality) and backwards compatibility of new releases. Correct labelling allows developers and users to easily recover older versions, compare them, or simply use them to reproduce results described in publications (see Rule 8). This approach also help to define a coherent software publication strategy.

## Rule 5: Let GitHub do some tasks for you: integrate

The first rule of software development is that the code needs to be ready to use as soon as possible [10], to remain so during development, and that it should be well-documented and tested. In 2005, Martin Fowler defined the basic principles for

continuous integration in software development [11]. These principles have become the main reference for best practices in continuous integration, providing the framework needed to deploy software, and in some way, also data. In addition to mere error-free execution, dedicated code testing is aimed at detecting possible bugs introduced by new features, or changes in the code or dependencies, as well as detecting wrong results, often known as *logic errors*, where the source code produces a different result than what was intended. Continuous integration provides a way to automatically and systematically run all of tests in the repository by checking data and software dependencies and can be automated through GitHub.

GitHub offers a set of hooks (automatically executed scripts) that are run after each push to a repository, making it easier to follow the basic principles of continuous integration. The GitHub web hooks allows third-party platforms to access and interact with a GitHub repository and thus to automate post-processing tasks. Continuous integration can be achieved by *Travis CI* (https://travis-ci.org), a hosted continued integration platform that is free for all open source projects. Travis CI builds and tests the source code using a plethora of options such as different platforms and interpreter versions (Supplementary Note, Section 2). In addition, it offers notifications which allow your team and contributors to know if the new changes work, and prevent the introduction of errors in the code (for instance when merging pull requests), making the repo always ready to use.

## Rule 6: Let GitHub do more tasks for you: automate

There is more than code compilation and testing that can be integrated into your software project. GitHub hooks can be used to automate numerous tasks to help improve the overall quality of your project. A important complement to successful test completion, is to demonstrate that they cover the existing code base sufficiently. For this, the integration of *Codecov* is recommended (https://codecov.io). This service will report how much of the code base and which lines of code are being executed as part of your code tests. The Bioconductor project, for example, highly recommends packages to implement unit testing to support developers in their package development and maintenance (http://bioconductor.org/developers/unitTesting-guidelines/), and systematically tests the coverage of all of its packages (https://codecov.io/github/Bioconductor-mirror/). One might also consider generating the documentation upon code/documentation modification (Supplementary Note, Section 3). This implies that your projects provide comprehensive documentation so others can understand, and contribute back to them. For Python or C/C++ code, automatic documentation generation can be done using sphinx (http://sphinx-doc.org/) and subsequently integrated into GitHub using "Read the Docs" (https://readthedocs.org/). All of these platforms will create reports and badges (also called shields) that can be included on your GitHub project page, helping to demonstrate that the content is high quality and well-maintained.

## Rule 7. Use GitHub to openly and collaboratively discuss, address and close issues

GitHub *issues* are a great way to keep track of bugs, tasks, feature requests and enhancements. Classical issue trackers are primarily intended to be used as bug trackers. In contrast, GitHub issue trackers (*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. *Issues*

main focus is on promoting collaboration, providing context by using cross-references. 250
Issues also have very clear formatting: (i) a title and description, (ii) color-coded 251
labels help to categorize and filter issues, (iii) milestones, (iv) one assignee responsible 252
for working on the issue, and (v) comments that allow anyone with a github account 253
to provide feedback. Another aspect of the issue functionality is its simplicity. For 254
instance, raising an issue does not require lengthy forms to be completed, or even 255
including every piece of information that might be valuable to reproduce a bug. It 256
only requires a title and provides space for optional text (comments). If the developer 257
needs more information, he or she can simply request it in a comment. Alternatively, 258
it is possible for repository administrators to create issue and pull request (see Rule 3) 259
templates 260
(https://help.github.com/articles/helping-people-contribute-to-your-project/) to 261
customise and standardise the information to be included when contributors open 262
issues. GitHub issues are thus dynamic and pose a low barrier-to-entry for users to 263
report bugs and request features. A well-organized and tagged issue tracker helps new 264
contributors and users to understand a project more deeply. As an example, the 265
following issue in the OpenMS repository 266
(https://github.com/OpenMS/OpenMS/issues/1095) allowed the interaction of eight 267
developers and more than hundred comments. Contributors can add figures, 268
comments and references to other *issues* and *pull requests* in the repository, as well as 269
direct references to code. 270

As another illustration of *issues* and their generic and wide application, we 271
(https://github.com/ypriverol/github-paper/issues) and others 272
(https://github.com/ropensci/RNeXML/issues/121) used GitHub issues to discuss 273
and comment changes in manuscripts and address reviewers' comments. 274

# Rule 8. Make your code easily citable, and cite source code! 275 276

In research it is good practice to ensure permanent and unambiguous identifiers for 277
citable items like articles, datasets, or biological entities such as proteins, genes and 278
metabolites, among others (see also Box 3). Digital Object Identifiers (DOIs) have 279
been used for many years as unique and unambiguous identifiers for enabling the 280
citation of scientific publications. More recently, a trend has started to produce DOIs 281
for other types of scientific outputs such as datasets [12] and training materials (for 282
example [13]). A key motivation for this is to build a framework for giving scientists 283
broader credit for their work [14, 15], while simultaneously supporting clearer, more 284
persistent ways to cite and track it. Helping to drive this change are funding agencies 285
such as the NIH in the USA and Research Councils in the UK, who are increasingly 286
recognizng the importance of research products such as publicly available datasets and 287
software. 288

A common issue with software is that it normally evolves at a different speed than 289
text published in the scientific literature. In fact, it is common to find software having 290
novel features and functionalities that were not described in the original publication. 291
GitHub now integrates with archiving services such as Zenodo (https://zenodo.org/) 292
and Figshare (https://figshare.com/), enabling DOIs to be assigned to code 293
repositories. The procedure is relatively straightforward (see 294
https://guides.github.com/activities/citable-code/), requiring only the provision of 295
metadata and a series of administrative steps. By default, Zenodo takes an archive of 296
a repository each time a new release is created in GitHub, ensuring the cited code 297
remains up to date. Once the DOI has been assigned, it can be added to literature 298

information resources such as Europe PubMed Central [16].

As already mentioned in the introduction, reproducibility of scientific claims should be enabled by providing the software, the datasets and the process leading to interpretable results that was used in a particular study. As much as possible, npublications should highlight that the code is freely available in, for example, GitHub, together with any other relevant outputs that may have been deposited. In our experience, this openness substantially increases your chances of getting the paper accepted for publication. Journal editors and reviewers have the opportunity to reproduce your findings during the manuscript review process, increasing their confidence in your results. In addition, once the paper is published, your work can be reproduced by other members of the scientific community, which can increase citations and foster opportunities for further discussion and collaboration.

The availability of a public repository containing the source code does not make the software open source *per se*. You should use an OSI approved license (https://opensource.org/licenses/alphabetical) that defines how the software can be freely used, modified and shared. Common licenses such as those listed on http://choosealicense.com are preferred. Note that the LICENSE file in the repo should be a plain-text file with the contents of an OSI approved license, not just a reference to the license.

## Rule 9. Promote and discuss your projects: web page and more

The traditional way of promoting scientific software is by publishing them in the peer-reviewed scientific literature. Additional steps can boost the visibility of a organization. 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 frameworks such as Bootstrap (http://getbootstrap.com/) or platforms such as Disqus (https://disqus.com/), to support and moderate comments. In addition, several real-time communication platforms have been integrated with GitHub such as Gitter (http://gitter.im) and Slack (https://slack.com/). Real-time communication systems allows the user community, developers and project collaborators to exchange ideas and issues, and to report bugs or get support. For example, Gitter is a GitHub-based chat tool which enables developers and users to share aspects of their work. Gitter inherits the network of social groups operating around GitHub repositories, organizations, and issues. It relies on the identity within GitHub, creating IRC (Internet Relay Chat)-like chat rooms for public and private repositories. Within a Gitter chat, members can reference issues, comments, and pull requests. GitHub also supports wikis (which are version-controlled repositories themselves) for each repository, where users can create and edit pages for documentation, examples or general support.

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. Gists can be generated in two different 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. Use GitHub to be social: follow and watch

A key responsibility of scientific researchers is to actively follow the developments in their field. Analogously, scientific programmers need to follow publicly available (e.g., open source) projects and code that might be interesting for their research. Therefore, you should try to learn as much as possible from your peers, and keep up-to-date with all the developments of relevant projects. GitHub enables this functionality by *following* other GitHub users (see also Rule 2) 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 involved in scientific research and have not used git and GitHub before, we recommend you to explore its potential as soon as possible. As with many other tools, a certain learning curve lays ahead. However, basic yet very useful features can be relatively easily learned and applied to many different use-cases [6]. We anticipate the reward will be worth your effort. To conclude, we would like to recommend some examples of bioinformatics repositories in GitHub (Table 1) and some useful training materials including workshops, online courses and manuscripts (Table 2).

## Disclaimer

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

## References

1. Goodman A, Pepe A, Blocker AW, Borgman CL, Cranmer K, Crosas M, et al. Ten simple rules for the care and feeding of scientific data. PLoS Comput Biol. 2014 Apr;10(4):e1003542.

2. Perez-Riverol Y, Alpi E, Wang R, Hermjakob H, Vizcaíno JA. Making proteomics data accessible and reusable: current state of proteomics databases and repositories. Proteomics. 2015 Mar;15(5-6):930–49.

3. Osborne JM, Bernabeu MO, Bruna M, Calderhead B, Cooper J, Dalchau N, et al. Ten simple rules for effective computational research. PLoS Comput Biol. 2014 Mar;10(3):e1003506.

4. Vihinen M. No more hidden solutions in bioinformatics. Nature. 2015 May;521(7552):261.

5. Blischak J, Davenport E, Wilson G. A Quick Introduction to Version Control with Git and GitHub. PLoS computational biology. 2016;12(1):e1004668.

6. Ram K. Git can facilitate greater reproducibility and increased transparency in science. Source code for biology and medicine. 2013;8(1):1.

7. Dabbish L, Stuart C, Tsay J, Herbsleb J. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative

| Name of the Repository | Type | URL |
|---|---|---|
| Galaxy Project [17] | Community Project, Bioinformatics Repository | https://github.com/galaxyproject/galaxy/ |
| Retinal wave data repository [18] | Binary Data organized | https://github.com/sje30/waverepo/tree/master/ |
| OpenMS [19] | Bioinformatics Repository, Issue discussion, branches | https://github.com/OpenMS/OpenMS/issues/10 |
| Pride Inspector Toolsuite [20] | Project Organization, Multiple project repositories | https://github.com/PRIDE-Toolsuite |
| BioPython [21] | Community Project, Multiple contributors | https://github.com/biopython/biopython/graphs |
| Adam | Community Project, Multiple forks | https://github.com/bigdatagenomics/adam/ |
| Computational Proteomics Unit | Lab Repository | https://github.com/ComputationalProteomicsUn |
| MSnbase [22] | Individual project repository | https://github.com/lgatto/MSnbase |
| SAMtools [23] | Bioinformatics Repository, Project Organization | https://github.com/samtools |
| rOpenSci | Community Project, Issue discussion | https://github.com/ropensci |
| The Global Alliance For Genomics and Health | Community Project | https://github.com/ga4gh |
| GitHub Paper | Manuscript Repository, Issue discussion, Community Project | https://github.com/ypriverol/github-paper |

**Table 1.** Bioinformatics repository examples with good practices using GitHub. The table contains the name of the repository, the type of the example (issue tracking, branch structure, unit tests) and the URL of the example.

| Name of the material | Type | URL |
| --- | --- | --- |
| `git help` and `git help -a` | Documentation | Local, installed with `git` |
| Karl Broman's git/github guide | Short tutorial | http://kbroman.org/github_tutorial/ |
| Version Control with GitVersion Control with Git | Tutorial | http://swcarpentry.github.io/git-novice/ |
| Introduction to Git | Tutorial | http://git-scm.com/book/ch1-3.html |
| Github Training | Courses and Training | https://training.github.com/ |
| Github Guides | Tutorial | https://guides.github.com/ |
| Good Resources for Learning Git and GitHub | Tutorials Directory | https://help.github.com/articles/good-resources-for-learning-git-and-github/ |
| Software Carpentry: Version Control with Git | Tutorial | http://swcarpentry.github.io/git-novice/ |

**Table 2.** Online courses, tutorials and workshops about GitHub and Git for scientists.

Work. CSCW '12. New York, NY, USA: ACM; 2012. p. 1277–1286. Available from: `http://doi.acm.org/10.1145/2145204.2145396`.

8. R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria; 2016. Available from: `https://www.R-project.org/`.

9. Noble WS. A quick guide to organizing computational biology projects. PLoS Comput Biol. 2009 Jul;5(7):e1000424.

10. Leprevost FV, Barbosa VC, Francisco EL, Perez-Riverol Y, Carvalho PC. On best practices in the development of bioinformatics software. Front Genet. 2014;5:199.

11. Fowler M. Continuous Integration; 2006. Accessed: 2015-12-09. `http://www.martinfowler.com/articles/continuousIntegration.html`.

12. Vizcaíno JA, Deutsch EW, Wang R, Csordas A, Reisinger F, Ríos D, et al. ProteomeXchange provides globally coordinated proteomics data submission and dissemination. Nat Biotechnol. 2014 Mar;32(3):223–6.

13. Ahmadia A, Aiello-Lammens M, Ainsley J, Allen J, Alsheikh-Hussain A, Banaszkiewicz P, et al.. Software Carpentry: Programming with R; 2015. Available from: `http://dx.doi.org/10.5281/zenodo.27353`.

14. Credit where credit is overdue. Nat Biotechnol. 2009 Jul;27(7):579.

15. FORCE11 Software Citation Working Group;. Accessed: 2016-03-31. `https://www.force11.org/group/software-citation-working-group`.

16. Europe PMC Consortium. Europe PMC: a full-text literature database for the life sciences and platform for innovation. Nucleic Acids Res. 2015 Jan;43(Database issue):D1042–8.

17. Goecks J, Nekrutenko A, Taylor J, Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. 2010;11(8):R86.

18. Eglen SJ, Weeks M, Jessop M, Simonotto J, Jackson T, Sernagor E. A data repository and analysis framework for spontaneous neural activity recordings in developing retina. Gigascience. 2014;3(1):3.

19. Sturm M, Bertsch A, Gröpl C, Hildebrandt A, Hussong R, Lange E, et al. OpenMS–an open-source software framework for mass spectrometry. BMC bioinformatics. 2008;9(1):163.

20. Perez-Riverol Y, Xu QW, Wang R, Uszkoreit J, Griss J, Sanchez A, et al. PRIDE Inspector Toolsuite: Moving Toward a Universal Visualization Tool for Proteomics Data Standard Formats and Quality Assessment of ProteomeXchange Datasets. Molecular & Cellular Proteomics. 2016;15(1):305–317. Available from: `http://www.mcponline.org/content/15/1/305.abstract`.

21. Cock PJ, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. 2009 Jun;25(11):1422–3.

22. Gatto L, Lilley KS. MSnbase-an R/Bioconductor package for isobaric tagged
    mass spectrometry data visualization, processing and quantitation.
    Bioinformatics. 2012;28(2):288–289. Available from:
    `http://bioinformatics.oxfordjournals.org/content/28/2/288.abstract`.

23. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The
    Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009
    Aug;25(16):2078–9.