output:
pdf_document: default

html_document: default

Module 5: Open Research Software and Open Source

Table of Contents

[IDEA: ONCE TEXT IS COMPLETE; CREATE AUDIO RECORDING FOR ALL OF THIS FOR EACH MODULE, and release as a podcast]

Introduction

Welcome to **Module 5** of the Open Science MOOC: **Open Research Software and Open Source**.

This module has been developed in the open through collaboration by an international team of Open Source afficianados. Everything you see here has been developed in the open through interactive feedback and collaboration from the wider community. It comprises a series of videos, infographics, text-based reading (sorry), and practical tasks for you to sink you teeth into.

Don't forget you can join in the discussions over at our open **Slack channel**. Please do introduce yourself at #module5opensource, and tell us a bit about who you are, your background, and how you ended up here!

Why should you take this module?

[TO ADD: Introductory video]

Who is this module for?

This module is designed primarily for computational researchers at the graduate and undergraduate level, as well as budding data scientists and any other researcher who uses analytical code or software. In a modern day research environment, this covers pretty much anyone who uses a computer for ther work.

> "An article about computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result." - J. Buckheit and D. L. Donoho, 1995.

Software and technology underpin much of modern research, which is now almost inevitably computational in one way or another - search engines, social networking platforms, analytical software, and digital publishing. With this, there is an ever-increasing demand for more sophisticated Open Source Software, matched by an increasing willingness for researchers to openly collaborate on new tools.

The power of Open Source is in that it lowers the barriers to collaboration and adoption, therefore allowing ideas and technology to spread more rapidly. This Module will introduce the necessary tools required for transforming software into something that can be openly accessed and re-used by others.

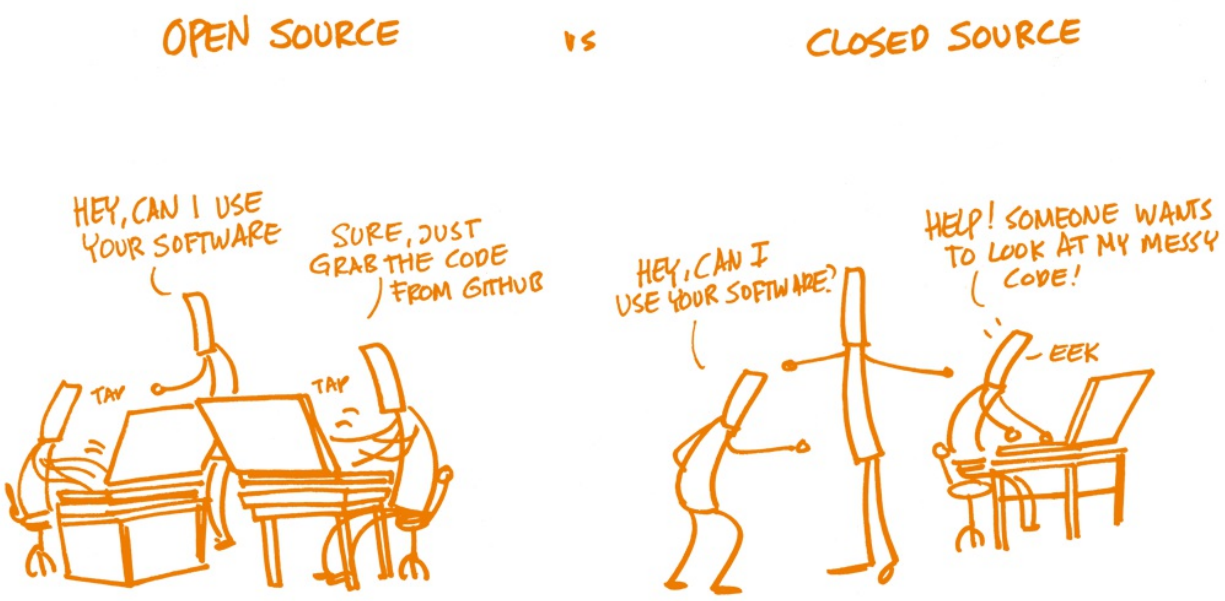


Image by Patrick Hochstenbach (CC0 1.0 Universal)

Specific learning objectives for this Module:

1. Learn the characteristics of open software; understand the **ethical, legal, economic, and research impact arguments for and against Open Source Software**, and further understand the quality requirements of open code.
2. Be able to turn code made for personal use into open code which is accessible by others.
3. Use software (tools) that utilizes open content and encourages wider collaboration.

What is Open Source Software

Virtually all modern scientific research workflows rely on a range of software tools, either operating on different datasets, with different parameters, and applied iteratively in various ways (data science) or operating on different inputs and using models and methods to predict some output state (computational science). Open Source Software (OSS) is computer software in which the full source code is available under a specific license that enables other users to access, view, modify, and redistribute that code for any purpose. Because OSS requires such a license, it typically remains free of charge by default. This explicit licensing is also what differentiates OSS from free software. Re-using OSS for analysis, simulation and visualisation for research is also typically easier and more flexible compared to proprietary software. Often, whether we know it or not, we are already using OSS as part of our own research workflows.

OSS fits into the broader scheme of Open Science as it helps to make the full research environment, including the software that produced the research results, fully accessible and re-usable. As such, it forms a necessary component for the best practices (Jiménez et al., 2018) and repeatability and reproducibility of research (both personally and by others), along with other components, such as sharing data (Stodden, 2010).

In some cases, sharing of source code can even be conditional for the acceptance of associated research manuscripts (Shamir et al., 2013). It is also generally perceived to increase research impact (Vandwalle, 2012).

Some of common advantages for developers include:

- Increased developer loyalty and empowerment;
- Lower costs of services and marketing;
- Increased branding of services and products;
- Production of high quality software at lower expense;
- Flexibility and rapid innovation;
- Customisation and modular integration;
- Increased reliability and independence; and

- Based on open standards available to everyone.

As such, the main advantages for researchers (users) include **lower costs**, **increased transparency**, **increased security and stability**, **no vendor 'lock in' with increased user control**, and **overall higher quality**. Furthermore, sharing OSS allows researchers to receive credit for their efforts, for example through direct software citation (Smith et al., 2016).

Commonly used OSS include the Mozilla Firefox internet browser and the LibreOffice full office suite. LibreOffice is similar to the popular Microsoft Office, including a word processor, spreadsheet manager, and slide presentation software, but is completely free and Open Source.

Some regard the OSS movement to represent a counter-movement to neoliberalism and privatisation, through defiance of regulations and norms in the construction and re-use of information, and a potential transformation of modern-day capitalism through making software abundantly available with minimal effort. See The free/open source software movement: Resistance or change? by Panayiota Georgopoulou for more on this topic.

Principles of Open Source Software

The Open Source Initiative, one of the pioneers of OSS, offers the following definition:

Don't worry, you don't need to memorise all of this, but it's good to know the principles that OSS is coming from.

- **Free Redistribution**: The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
    - **Source Code**: The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
    - **Derived Works**: The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
    - **Integrity of The Author's Source Code**: The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
    - **No Discrimination Against Persons or Groups**: The license must not discriminate against any person or group of persons.
    - **No Discrimination Against Fields of Endeavour**: The license must not restrict anyone from making use of the program in a specific field of endeavour. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
    - **Distribution of License**: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
    - **License Must Not Be Specific to a Product**: The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
    - **License Must Not Restrict Other Software**: The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
    - **License Must Be Technology-Neutral**: No provision of the license may be predicated on any individual technology or style of interface.

Now, this all might be a little complex to remember. However, it can be summarised as *making software as re-usable as possible for future works, while also being freely available*.

An Open Source checklist

There are a number of existing platforms and tools that support OSS and collaboration. The Open Science Training Handbook provides a check-list to use for evaluating the 'openness' of existing research software, based on the Open Source Definition above:

- Is the software available to download and install?
- Can the software easily be installed on different platforms?
- Does the software have conditions on the use?

- Is the source code available for inspection?
- Is the full history of the source code available for inspection through a publicly available version history?
- Are the dependencies of the softw are (hardw are and softw are) described properly? Do these dependencies require only a reasonably minimal amount of effort to obtain and use?

Check, check, check, done! Simples.

The Open Source community and its governance

There are tw o main camps w ithin the free softw are community: The **free softw are movement**, and the **OSS movement**. Both have differing ideologies based on user liberties and the practical applications of softw are. Often, the term 'FLOSS' is used to reconcile these tw o political camps, and means 'Free/Libre and Open Source Softw are'; Libre being French and Spanish for 'free' in the context of freedom.

The core principle of re-use is w hat separates OSS from 'Free Softw are'. Free and Open Source Softw are (FOSS) is an inclusive term to describe softw are that can be classified as both free and Open Source. A good example of FOSS is the Ubuntu Linux operation system.

The big difference betw een free softw are and OSS is that the former must distribute updated versions under the same license as the original, w hereas new er versions of OSS can be distributed under different licenses. FOSS combines the best of both w orlds.

These definitions have now become w idely adopted, both by international governments, as w ell as some large organisations such as the Mozilla Foundation and the Wikimedia Foundation. Major organisations in the FLOSS space include the UK's Softw are Sustainability Institute, w ho produce valuable resources such as their recent Softw are Deposit Guidance for Researchers.

For individual projects

Within OSS projects, there are typically three main formal roles:

- **Maintainer**;
- **Contributor**; and
- **Committer**.

A **maintainer** is a user w ith 'commit' access to implement suggested changes to the project. They have responsibility for the direction and improvement of the project. A **contributor** is someone w ho directly adds value to the project through issue resolution, code w riting, or even external activities such as communications and event organisation. A **committer** is someone w ho can make 'commits' to the project (see Task 1).

Typically, roles are made public through either the `README` file, a Contributors file, or a separate team page for the project.

Existing platforms and tools for Open Source Software

Virtual environments and machines are becoming increasingly popular as high-pow ered research w orkflow enablers, and many of these are built upon OSS (e.g., operating systems, programming languages, and data processing framew orks). Popular services include Google Cloud and Amazon Web Services, w hich also assist w ith database storage and content delivery, as w ell as computational pow er. InsideDNA is a computing platform for reproducible research in bioinformatics, genomics and the life sciences.

As mentioned above, LibreOffice provides an Open Source alternative to Microsoft Office. The tw o are almost completely compatible, just w ith different default file formats. For citation managers, Zotero is the most popular Open Source alternative to proprietary platforms such as Mendeley or EndNote.

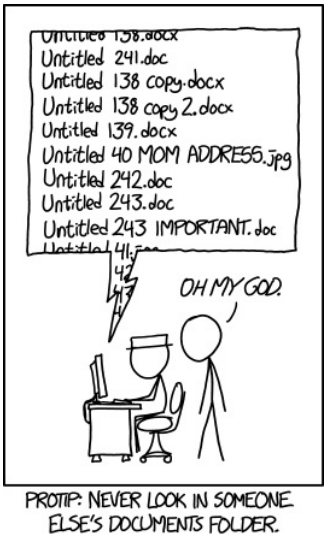
Zotero uses the BibTeX (pronounced 'bib-tech') format, based on LaTeX (pronounced 'lay-tech'), and has brow ser plugins to make citation management simple. By integrating this w ith other softw are such as LibreOffice, it is now possible to have a fully Open Source research w orkflow in many cases.

GitHub

> Did you know that this entire project w as build as an open and collaborative community effort in GitHub?

GitHub is a popular hosting site for both softw are and non-softw are content (often called 'notebooks'), w ith added capabilities for version control, project management and tracking, and storage services. GitHub is built on top of the OSS Git, w hich enables users to w ork remotely to maintain, share, and collaborate on research softw are and other non-softw are based projects.

Version control is essentially a process that takes snapshots of the files in a repository, and tracks modifications to them. It records w hen the changes w ere made, w hat they w ere, and w ho did them. If several people are w orking on one file at once, any overlapping changes are detected, and must be resolved prior to continuing. This provides a much more streamlined and automated process than manually saving and recording changes as projects develop. It also avoids the inevitable lists of confusing named file versions…



GitHub helps us to avoid, er, sub-optimal file naming conventions (source: XKCD)

One of the more popular and useful functions of GitHub is the issue tracker, which is used to organise OSS development. The above link takes you to the issue tracker for the development of this module! If you think there is something here that can improved, or you want to comment on, anyone can add or contribute to an issue there!

Other similar project hosting services include BitBucket, GitLab, and Launchpad. If the recent acquisition of GitHub by Microsoft is a bit off-putting to you, these are great alternatives.

However, we also know that GitHub can have quite a high learning curve. Which is why the first practical task for this MOOC will teach you how to set up your first GitHub project repository!

GO TO TASK 1: Building your first GitHub repository

Open Source Software used in research

Especially in scientific research, Open Source Software usage and development has become practically the norm. There's a number of reasons for this beyond those that apply to the general acceptance of OSS by, for example, consumers, industry, or government. Among these reasons are:

- Increasingly, algorithms implemented in analysis software form an integral part of the methods described in scholarly publications. As such, it is completely at odds with rigorous peer review if these algorithm implementations are closed to outsiders.
- Scientific collaboration more often than not spans multiple institutions and distributed research networks where secrecy and command hierarchy is not maintained in a way that is 'necessary' for closed source development.
- Many computational analyses are run in virtualized environments (such as institutional, national, or international 'cloud' infrastructures) and hosted on multi-user servers. Closed-source, commercial software often disallows such usage.
- OSS development often relies on volunteers. In a time of budgetary constraints for scientific research, this is a clear advantage.

For these and other reasons, Open Source tools are very commonly used in scientific research. This includes usage in fields where many researchers are amateur developers themselves and rely on tools such as R for statistical analysis and scripting, which, in the last decade, has almost completely displaced commercial software for statistical analysis such as SPSS or JMP in a lot of fields. In fields such as bioinformatics, that involve a lot of file handling of the outputs of DNA sequencing platforms, general purpose scripting languages such as Python and commonly used libraries built on top of it (such as biopython) have become a vital part of the toolkit of many researchers.

Python

Tools such as R and Python are essentially software for writing software. Although programming is an increasingly common activity among researchers, of course not *every* scientist does this. One step away from programming is the chaining together of the inputs and outputs of various analysis tools in longer workflows. As an example from genomics, a very common workflow is to start out with high-throughput sequencing reads and then i) do basic quality control checks; ii) map the reads against a reference genome; iii) identify the points where the new data are at variance with the reference. These steps are routinely executed as a workflow where a different Open Source executable is run in a Linux command-line environment for each of the three steps. Although this is arguably not quite open source software development, it does involve the usage and production of open source artifacts (such as Linux shell scripts) for which the principles that we discuss in this module are applicable.

R

Lastly, OSS is also used in scientific research for reasons that more closely mirror those that drive the adoption of OSS in wider society, namely that it is cheap. For example, individuals or organizations might decide to switch from Microsoft Office to LibreOffice for manuscript writing or spreadsheet processing because the latter is free (both as in **'free beer'** and 'free speech'). Likewise, the choice to switch from ArcGIS to QGIS for the analysis of geographic information might be prompted simply by cost considerations.

Getting Started with OSS - FAQ

I'm using X[e.g. Matlab,STATA,Excel] and I want to transition to something more open. What are the next steps?

Even if you are using proprietary software, you can usually still share your source code/documents etc. *The best first step is sharing whatever you can.*

Great! I can put them in my new github repo.

If that's enough for you for now great! If not for most pieces of proprietary software there are Open Source equivalents. Have a go with one and see what you think.

Closed	Open
Matlab	Python, Julia
STATA/SPSS	R
MS Office	LibreOffice
Mathematica	JupyterLab
Test out your new Pull Request -PR- Skills …	… by adding your own example here

Cool! But if I make the switch will I be stuck: taking ages to learn a new tool/ without support /with buggy software.

Good question! The answer is it depends. The best thing to do is find someone who's made the switch before and learn from their experience. Or just do a Google search! Some OSS is much better than their closed counterparts, some aren't, so it's worth choosing carefully.

Making good software for re-use

The most likely person who might want to re-use your software in the future is…you! So while sharing is always better than not sharing, you can make your own life, and that of others, much easier through appropriate documentation. Documentation can include several things, such as including helpful comments and annotations in the code that help to explain why a particular action was performed, rather than what it is intended to achieve.

One of the most critical aspects of this is including an informative `README` file, that accompanies almost every OSS project, and some times even more than one. It can be a good practice to include one such file in every directory, that includes a list of files, a table of contents, and what the purpose of the directory is. The `README` file is typically just plain text or markdown (again, such as all of the ones for the MOOC!), and can include critical information for how to install and run software, previous dependencies and requirements, as well as tutorials or examples.

> **Did you know…** The term `README` is some times playfully ascribed to the famous scene in Lewis Carroll's Alice's Adventures In Wonderland in which

> Alice confronts magic munchies labeled with "Eat Me"" and "Drink Me". Potent.

The purpose here is to provide sufficient information to maximise the re-use and reproducibility of the computational environment, such that someone with no experience with the project can easily access and re-use the software (Sandve et al., 2013). By lowering the barriers to entry, you increase the chances of others being able to re-use your work, which is one of the ultimate goals of OSS (Ince et al., 2012).

An extension of this that can help to make things even easier for future re-use is 'container' technology. Containers are like an ecosystem frozen in time, where the code, the data, any other dependencies, are all perfectly preserved, packaged and saved in the present functioning versions. This means that anyone in the future any one can come in and run the analyses again. As such, they are generally good for re-use, but this can come at the sacrifice of modification or understanding by others, as often a lot of details can be hidden within the source code and its dependencies. Common examples of container implementation in research include Rocker (a Docker container for the R language), Binder, and Code Ocean.

Sustainable software is good software.

10 simple rules for reproducible computational research

The 10 simple rules for making computational research more reproducible, based on Sandve et al., (2013), are:

1. For every result, keep track of how it was produced.
2. Avoid manual data manipulation steps.
3. Archive the exact versions of all external programs used.
4. Version control all custom scripts.
5. Record all intermediate results, when possible in standardised formats.
6. For analyses that include randomness, note underlying random seeds.
7. Always store raw data behind plots.
8. Generate hierarchical analysis output, allowing layers of increasing detail to be inspected.
9. Connect textual statements to underlying results.
10. Provide public access to scripts, runs, and results.

3. ARCHIVE THE EXACT VERSIONS OF ALL EXTERNAL PROGRAMS USED.

4. VERSION CONTROL ALL CUSTOM SCRIPTS.

5. RECORD ALL INTERMEDIATE RESULTS, WHEN POSSIBLE IN STANDARDISED FORMATS.

6. FOR ANALYSES THAT INCLUDE RANDOMNESS, NOTE UNDERLYING RANDOM SEEDS.

7. ALWAYS STORE RAW DATA BEHIND PLOTS.

8. GENERATE HIERARCHICAL ANALYSIS OUTPUT, ALLOWING LAYERS OF INCREASING DETAIL TO BE INSPECTED.

9. CONNECT TEXTUAL STATEMENTS TO UNDERLYING RESULTS.

10. PROVIDE PUBLIC ACCESS TO SCRIPTS, RUNS, AND RESULTS.

Infographic adapted from Sandve et al., (2013). Feel free to download this and print it out to keep handy during your research!

If you follow these steps, along with the processes in **Task 1** and **Task 2**, you should be fine!

Open Source licensing

An Open Source license is a type of license designed specifically for software and code that make it explicit what the legal conditions for sharing and re-use are. As mentioned above, the addition of a suitable license is what differentiates publicly shared software from OSS. For example, the widely used MATLAB is proprietary software, and Octave is an openly licensed alternative programming language.

There are currently more than 1,400 unique Open Source licenses, a complexity born from the difficulty in understanding the differences between the legal implications across different license.

Some of the more common licenses include:

- Berkeley Software Distribution ("BSD"),
- Apache,
- MIT-style (Massachusetts Institute of Technology), or
- GNU General Public License ("GPL").

You don't need to know all the legal itty gritty behind all of these, but it is good to at least know what options are availlable to you.

There are two ways in which contributions to a project become licensed:

1. *Explicitly*, whereby the individual contribution has a clearly indicated license independent of the main project; or
2. *Implicitly*, whereby the contribution falls under the original licensing code of the main project.

Thankfully, the process of selecting an Open Source license is relatively trivial, thanks to user-friendly tools such as Choose A License. Each of these licenses allows other users to use, copy, distribute, and build upon your work, often while ensuring that the creators are appropriately recognised for their work. Here, the key is selecting an appropriate license for your work, depending on what you want, or do not want, others to do with it.

Software citation

Citations provide one of the most important interactions in scholarly research, forming the basis of our referencing and metrics systems. Typically, this is performed thanks to the assistance of a permanent unique identifier such as a Digital Object Identifiers (DOI). A DOI is a persistent identifier, implemented in the Handle System, that meets a common standard, depending on the purpose, such as for identifying academic information. Such identification is critical for tracking the genealogy and provenance of research, for reproducibility, as well as for giving appropriate credit to those who have created the software. Importantly, software should be considered a legitimate output from scholarly research, and citation is becoming an increasingly common way to indicate that.

In 2016, Smith et al., 2016 wrote a research paper about the principles of software citation as part of the FORCE11 Software Citation Working Group. In the same way that you would want to cite software that you have used as part of good research practices, it is important to make your research easily citable too. When citing any software used for your own research, you should include at minimum:

- The author name(s),
- Software title,
- Version number, and
- The unique identifier/locator (DOI or URL).

The six principles of software citation by Smith et al., (2016) are provided here:

- **Importance**: Software should be considered a legitimate and citable product of research. Software citations should be accorded the same importance in the scholarly record as citations of other research products, such as publications and data; they should be included in the metadata of the citing work, for example in the reference list of a journal article, and should not be omitted or separated. Software should be cited on the same basis as any other research product such as a paper or a book, that is, authors should cite the appropriate set of software products just as they cite the appropriate set of papers.

- **Credit and attribution**: Software citations should facilitate giving scholarly credit and normative, legal attribution to all contributors to the software, recognizing that a single style or mechanism of attribution may not be applicable to all software.

- **Unique identification**: A software citation should include a method for identification that is machine actionable, globally unique, interoperable, and recognized

by at least a community of the corresponding domain experts, and preferably by general public researchers.

- **Persistence**: Unique identifiers and metadata describing the software and its disposition should persist - even beyond the lifespan of the software they describe.
- **Accessibility**: Software citations should facilitate access to the software itself and to its associated metadata, documentation, data, and other materials necessary for both humans and machines to make informed use of the referenced software.
- **Specificity**: Software citations should facilitate identification of, and access to, the specific version of software that was used. Software identification should be as specific as necessary, such as using version numbers, revision numbers, or variants such as platforms.

Note: For instructions on 'how to make your software citable' see the section **Using GitHub and Zenodo** below and **Task 2: Linking GitHub and Zenodo**.

Using GitHub and Zenodo

GitHub is a popular tool for project management, content storage, and version control. Note that GitHub itself is not OSS. However, Git, the tool which it is based on, is. Git is designed to help manage the source code files, and the updates to them, for a software-related project. However, it can also be extended to other non-software projects; for example, this MOOC!

However, getting research onto GitHub is just the first step. It is equally important to make it persistent and re-usable, which is why having a Digital Object Identifier (DOI) associated with it can be useful. The simplest way to do this is through a service called Zenodo, which is a free and open source multi-disciplinary repository created by OpenAIRE and CERN, and can be used to assign a DOI to individual GitHub repositories. There is a GitHub Guide that explains the details, which involve linking GitHub repositories directly through to Zenodo so that when developers create formal releases for their software, Zenodo creates and archives a that version of the software.

There's nothing special about using Zenodo for creating DOIs, other than its **free of cost**; other general repositories can also be used, such as DataCite DOI Fabrica, or your own institutional repositories such as Caltech's.

A lot of researchers might typically be afraid of sharing code which is incomplete, buggy, or imperfect. However, in the OSS community, such a practice of sharing 'raw' code is fairly commonplace. Sharing code openly enables others to re-use and improve it, as well as to engage in a deeper way with any research associated with it. This is one of the fundamental aspects of peer-collaboration, perhaps best exemplified by the traditional process of research manuscript peer review.

Task 2 will guide you through the process of linking a GitHub repository to Zenodo for archiving.

> **Did you know...** All content produced for this MOOC is available as part of a community in Zenodo?

GO TO TASK 2: Linking GitHub and Zenodo

Collaborating and contributing through Open Source

Often, OSS is developed in a public, decentralised, collaborative manner between multiple contributors. The purpose of this is to enhance the diversity and scope of a project and its design, in order to become more beneficial and sustainable. Such an approach was famously likened to a 'bazaar' model by Eric Raymond, an early OSS proponent. One of the major guiding principles of this is that of **peer production**, which relies on self-organised communities to regulate the development of content, co-ordinated towards a shared goal or outcome.

OSS projects rely heavily on volunteer collaboration, which often entails a constant flux of newcomers in order to become productive and sustainable (Steinmacher et al., 2014). Creating the right social atmosphere for a project, and a welcoming engagement environment, are often critical to successful collaboraitons in OSS.

Where to go from here

Hopefully now you have come to see the importance of software as a cornerstone of modern science, and the importance that OSS plays in this.

The **learning outcomes** from this should be:

1. You will now be able to define the characteristics of OSS, and some of the ethical, legal, economic and research impact arguments for and against it.
2. Based on community standards, you will now be able to describe the quality requirements of sharing and re-using open code.
3. You will now be able to use a range of research tools that utilise OSS.
4. You will now be able to transform code designed for their personal use into code that is accessible and re-usable by others.

5. Software developers will be able to make their software citable, and software users will know how to cite the software they use.

BONUS TASK

If you have completed Task 1 and Task 2, we also have a **BONUS TASK** for you, if you want to take your skills a step further. Task 3 will take you a step deeper into integrating Git into a typical research workflow by showing you how to integrate it with R Studio. It is recommended that you have completed the first 2 tasks before proceeding with this one.

However, your Open Source journey does not stop here! This was just the beginning, and there are some incredible resources out there if you would like to do or learn more:

- If you feel particularly inspired by this, you can endorse the Science Code Manifesto, which is based on the five principles of code, copyright, citation, credit, and curation.
- To launch and develop your own project, the Open Source Guides program offers a range of practical guides and skills to help launch and advance your OSS projects.
- For a detailed look at OSS-based research workflows, the Open Science, Open Data, Open Source hand-guide by Pedro L. Fernandes and Rutger A. Vos is one of the top resources online.
- More formalised journal venues also exist for software-based articles, including The Journal of Open Research Software and The Journal of Open Source Software. A list of such venues is also available.
- The PLOS Open Source Toolkit provides a global forum for Open Source hardware and software research and applications.
- The NumFOCUS is a nonprofit organization that supports and promotes world-class, innovative, open source scientific software. Some of the projects they sponsor include:
    - IPython and Jupyter Notebook initiatives.
    - rOpenSci, which promotes the open source R statistical environment for transparent and reproducible research.
    - To gain more hands on experience with OSS, the Software Carpentry community holds regular workshops to improve lab-based computing skills (Wilson et al., 2017).

Further reading

These references here are just the beginning. They include some of the most useful general overviews of the Open Source landscape in research. However, if you want to be find something more specific to your own research field, then that path is there for you to explore!

- The Future of Research in Free/Open Source Software Development (Scacchi, 2010).
- The Scientific Method in Practice: Reproducibility in the Computational Sciences (Stodden, 2010).
- The case for open computer programs (Ince et al., 2012).
- Current issues and research trends on open-source software communities (Martinez-Torres and Diaz-Fernandez, 2013).
- Ten simple rules for reproducible computational research (Sandve et al., 2013).
- A systematic literature review on the barriers faced by newcomers to open source software projects (Steinmacher et al., 2014).
- Knowledge sharing in open source software communities: motivations and management (Iskoujina and Roberts, 2015).
- Software citation principles (Smith et al., 2016).
- An introduction to Rocker: Docker containers for R (Boettiger and Eddelbuettel, 2017).
- Good enough practices in scientific computing (Wilson et al., 2017).
- Four simple recommendations to encourage best practices in research software (Jiménez et al., 2017).

Development Team

- Alex Morley, Open Sourceror, University of Oxford, UK.
- Kevin Moerman, Open Sourceror, MIT, USA.

- Tania Allard, Open Sourceress, Data Enchantress, University of Leeds, UK.
- Simon Worthington, Book Liberationist, TIB, Germany.
- Paola Masuzzo, Open Source Batman, Italy.
- Ivo Grigorov, Open Source Robin, Denmark.
- Rutger Vos, Open Sourceror, Naturalis Biodiversity Center, the Netherlands.
- Jon Tennant, Dinosaur Whisperer.

Know a way this content can be improved?

Time to take your new GitHub skills for a test-run! All content development primarily happens here. If you have a suggested improvement to the content, layout, or anything else, you can make it and then it will automatically become part of the MOOC content after verification from a moderator!