52north

Publishing Research Software as Open Source on GitHub

Daniel Nüst, Ann Hitchcock, Simon Jirka

Supported by **GLUES** http://geoportal-glues.ufz.de

NACHHALTIGES LANDMANAGEMENT

52°North best practice

http://52north.org

http://github.com/52North

http://twitter.com/FiveTwoN

http://youtube.com/user/52NorthInitiative

Published with GitBook

Table of Contents

1. Introduction
2. Scope & Goals
3. Science & Software
    i. Reproducibility
    ii. Software Quality
    iii. Software Development
    iv. Software Documentation
    v. Guide
4. Open Source Basics
    i. Mindset
    ii. Arguments against open source... and how to disprove them
    iii. Success Stories
    iv. Legal Stuff
    v. People
    vi. Guide
5. GitHub
    i. Basics: Accounts & Repositories
    ii. Fork & Pull Workflow
    iii. Social Coding
    iv. GitHub for Education
6. Software Communities
    i. Community Building and Openness
    ii. Marketing and Public Relations
    iii. Types of Contributors and Tasks
    iv. Open Source in Your Domain
7. Scientific Publishing of Data and Software
8. Contribute
9. Glossary

Introduction

How can you publish research software as open source? And do so without too much overhead and actually gain impact by leveraging the open source approach?

These questions are answered in this best practice "Publishing Research Software as Open Source on GitHub". It is published by the GLUES project's SDI team.

LICENSE

About this best practice

The "source code" of this document is hosted on GitHub and the book was written and published using GitBook.

The text is designed to be read in the web view, but PDF and other formats, e.g. for e-readers, area available as well.

Version: 0.1

Contributors

Thanks to these people for providing contents, giving valuable feedback, reporting errors, ...

- Daniel Nüst
- Simon Jirka
- Ann Hitchcock

Want to become a contributor? Check our contribution guidelines.

Contact information

- Authors
  - Daniel Nüst, @nuest on GitHub, d.nuest@52north.org

    - Simon Jirka, s.jirka@52north.org
    - Ann Hitchcook, a.hitchcook@52north.org
- Twitter: @FiveTwoN https://twitter.com/fivetwon

Scope and Goals

Research today is driven by data. It comes from various sources and is of manifold aspects. Often this data is published and analyzed as "open" data out of research projects ranging from single thesis to international consortium. Such projects have a life span of 2 to 4 years. Often the software developed for analysis is deemed unfit for publication, although the idea of open source software is widely known. Too often **the reason for not publishing the software is lack of time or knowledge**. This best practice will provide arguments and workflows that encourage effective and cost-efficient publication, as well as ensure long-term maintenance and management of research software as open source software.

The **target audience** is project managers, work group leaders and scientists from all domains that use code for their research but have not yet published it. The document provides an overview of relevant topics and provides references to more detailed work. The authors aim to use plain language to explain technical topics, but also introduce the "lingo" of open source software development. Where useful we add **Open Guides** to approach a topic from an opinionated standpoint to answer the most relevant questions for practical applications.

The **goal of this publication** is to give research project leaders, principal investigators, and researchers a set of workflows that increase the amount of reusable open source software to optimize research and its impact. It also provides arguments for publication when negotiating with supervisors or funding agencies. Please note that it is based on an open source initiative/small enterprise's experience in funded research projects. Open Source software is used while doing applied research in the geoinformatics domain.

We cannot cover specifc programming languages or recommend specific libraries of application domains, such as hydrology, physics, or geomorphology. But we think such recommendations would be useful, so please feel free to contribute such information as chapters to this document.

We use icons in the text to show when content is important, up for discussion, an opinion, covers legal aspects, especially relevant for PhD students, based on external resources or quotes, related to scientific publications, covers new or "hot" topics, open for discussion or your contributions are needed, or we simply really like it.

Guide: Should I publish my code, when, and what part of it?

Yes, publish reusable abstract core functions from the start. Don't publish your code as part of a "dissemination" for "others to use" at the end of your work. ~~If you don't plan to continue working on something, don't use open-sourcing to silence your conscience.~~

By publishing your code you can effectively

- demonstrate scientific rigor,
- improve the quality of the code and subsequently your research,
- start collaborations with other researchers.

What part of the code should be published? Publish reusable partitions / core libraries / abstract functionality, instead of the "solution" to your own problem. This allows others to re-use your code and contribute to improve it, and you to take advantage of their improvements. Your requirements are still met by the published code, even though you might actually work in a derived specialized project.

If possible, do not start a new project, but identify an existing project, framework or community and contribute your new work to it. If you have a valid contribution and can maintain it for some time, any community will welcome you with open arms.

The guides in this book apply both to very small projects, such as a new algorithm for a specific problem in biochemistry, and to large ones, such as a new cloud-based communication framework for the next generation of the internet of thing's sensor nodes.

This is an opinionated view. What is your experience? Get in touch and contribute!</p>

Further resources

- https://wiki.52north.org/bin/view/Documentation/BestPracticeOpenSourceForUniversityResearchers

---

Science & Software

In this chapter we try to connect the worlds of scientific research with open source software development.

How can I develop software as part of research? What is the difference between software and scientific papers? What is good software? What is research? How is software developed and documented?

Reproducibility

Reproducibility lies at the core of science. You should worry about it, especially if you use or develop software within your research. ~~But (sadly) this guide is not about good science, so~~ We kindly ask you to contribute good resources to this section to answer questions such as

- *What is it, why does it matter?*
- *How does it relate to software?*

Further resources:

- Making Your Code Citable on GitHub
- http://researchcompendia.org/
- http://reproducibleresearch.net/
- Reproducibility, Reproducible research and Open research computing on Wikipedia
- CRAN Task View: Reproducible Research
- Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013) Ten Simple Rules for Reproducible Computational Research. PLoS Comput Biol 9(10): e1003285. doi:10.1371/journal.pcbi.1003285
- Great blog post "10 non-trivial things GitHub & friends can do for science"

Software quality

What is good software?

You tell us. For the sake of this guide, software is good simply if it **does the job**.

~~Yes, we actually added a full chapter on Software quality, just to make it look like we would have a short answer for you!~~

If you worry about software quality, we recommend the following:

- Publish your software as open source from the start - it will make you more aware of documentation and code "beauty" (simple tips for more beautiful and better coding by 52°North developers) and thus increase the quality.
- Use an established documentation framework
- Follow an established software development lifecycle.

A longer answer can be found in any of these **resources**:

- Software Quality on Wikipedia
- David Chappell: The Three Aspects of Software Quality: Functional, Structural, and Process (industry white paper)

Software Development

Process

Whether you work alone or in a team, it is worth understanding the software development process you apply. Knowledge about software development lifecycles can help organize your research better!

Please read about the software development processes (about 30 mins. of reading):

- Software Development Process
- Selecting Development Approach
- Agile Software Development

If you skipped the list, then simply use **Scrum** (and read this or get it here, and this).

Our recommendation: Everything but an agile and iterative approach will eventually fail, cost a lot of money, take a lot of time and therefore does not work in research. State which agile approach you apply, then you have a *common language*.

Collaborative Software Development

Software development is an endless line of *"collect, compare, decide"*. To successfully employ this process, you need a few features, all of which are provided by the GitHub platform:

- Source code versioning
- Documentation platform (wiki)
- Issue or task tracking
- Website
- (mailing list only if you still use emails...)

There are other source code versioning systems (e.g. Mercurial, Bazaar, SVN), but Git is the real deal. *Go with the flow* - it makes it easier for new developers to join your project.

There are many task management systems. Trello, for example, is great and integrates very well with GitHub. Does your institution host Jira, Trac, Redmine, or Basecamp? Go for it.

You can host websites anywhere. If you need a blog, hosting with GitHub is possible but might not be your first choice. (see marketing...)

There are also **alternatives to GitHub** (see http://alternativeto.net/software/github/ and http://toppersworld.com/top-10-free-github-alternatives-for-private-repositories/), for example BitBucket, or R-Forge. *Go with the flow* of your language/domain. It lowers the barrier for collaboration. You can get commercial support and private repositores, or deploy an Open Source copy using GitLab

What platform works best for you? Would you like to learn about dependency management, build management, release management, or bug trial?

Documentation

Documentation is extremely important. Here is our short list of recommendations:

- Document from the start and use "future you" as a reference: imagine the information needed to understand a piece of code.
- Write "speaking" code, use long variable and function names, and follow software design patterns - it saves 80% of documentation.
- Use a documentation framework that suits your programming language, (a) with a version support and (b) that allows external contributions.
  - Plain text files within the code
  - **R**: Markdown vignettes
  - **Python**: reStructuredText
  - Top ten reasons GitHub is a great tool for creative writers
  - Read the Docs
  - GitHub pages
  - Wiki hosting services on Wikipedia

Guide: What license should I use for Documentation

You should **not use a software/code license for text and images**. These are licenses suitable for documentation:

- Any of the Creative Commons licenses. Depending on your goals, chose one!
- GNU Free Documentation License (discussed on Wikipedia)
  - See also the lists of free and nonfree documentation licenses at GNU.

This is an opinionated view. What is your experience? Get in touch and contribute!

Guide: Do software "right" from the start

You can greatly improve the quality of code and *of your research* by following established practices of the language of your choice **from the start**. Spend a little bit of effort in (a) setting up a reproducible dependency and build system in the beginning and (b) documenting the software all the time. This is a lot easier than trying to add documentation or "overhaul" when the code is "finished". Code is never finished. Seemingly simple information, such as how to set up the environment, or versions of used libraries, is simply lost after code is unused for a few months.

- Read a minimum of software development principles so that you know where you can find help. No need to learn something you can look up!
- Don't reinvent the wheel, use established libraries.
- Use established build and dependency management solutions for your language of choice. Even if you don't plan to publish your code in public repositories, do package it for them because it will enforce good practices.
  - use Maven or Gradle for **Java**
  - write an extension package for **R** and load required packages from CRAN
  - write a PyPi module when using **Python**
  - use Grunt and Bower for **JavaScript**, and package your code in an NPM module
- Document constantly
  - use "future you" without any previous knowledge as the recipient to judge level of detail of documentation
  - use a common documentation solution - see Documentation

Do you know typical frameworks for other languages used in open source development?

Open Source Basics

In this chapter we tackle the very basics of open source software.

What is open source? What are licenses for open source software? How do "open source people" think? Is open source really working?

Read on in the sections about Open Source Mindset, Arguments for Open Source, Success Stories, and Legal Stuff.

Mindset

To understand open source development one must understand how open source developers think and work.This section mostly points to worthwhile reading material for a rainy weekend (unless otherwise noted).

Open source has a strong relation to science and research. Many institutions use and create, but also research open source software. Explore this world using Google Scholar or Microsoft Academic Search. See, e.g. Amazon for printed publications.

What is Open Source?

Definition

The OSI defines open source as software following a set of criteria that ensure free distribution, extendibility, and non-discrimination. The full criteria are online and the annotated version is a highly recommended read. The minimal excerpt of the definition by the Free Software Foundation (FSF) is as follows.

> *“Free software” means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, “free software” is a matter of liberty, not price.* (source)

Read the full definition and make up your own mind about the FSF "campaign". Is nonfree software really "unethical"? How do FSF and OSI relate to each other? You can find an optional read about the controversy defining what open source software is here and here.

History

There are plenty of online descriptions of the history of open source. We recommend the following pages (roughly 1 hour of reading):

- "History of free software" on Wikipedia
- "Open-source movement" on Wikipedia
- "History of the OSI"
- "The 9 most important events in Open Source history", including a comment thread showing that boiling down open source to "X events" is impossible.

Terms "free", "libre" and "open source"

> *Free software is a matter of liberty, not price. To understand the concept, you should think of free as in free speech, not as in free beer.* Richard Stallmann

"Free software", "libre software" and "open source software" have slightly different meanings. These distinctions are important for the history and understanding of "somehow publicly available software".

Wikipedia has a summary of the naming, fun quotes about the unfortunate acronym *FLOSS*, as well as a whole page discussing the ambiguity of "free".

When we say open source software, we mean "free and libre open source software". We might use the acronym *FOSS*, but trust you to understand the ambiguity of the English word "free".

Extended reading:

- Free as in Freedom (2.0): Richard Stallmann and the Free Software Revolution by Sam Williams (book's Wikipedia page, 208 pages plus appendix!)

Hacking

The term "hacking" or "hackers", i.e. people who hack, is part of mainstream culture today. In open source communities, the term "hacking" has diverse meanings, some of them very positive. The same skill set is required for "cracking" illegaly into secure systems and for solving a very hard informatics problem creatively. So don't worry if someone talks about a "good hack"!

For a full history of the term and related concepts such as "white/black hat", "cracking", or "hacker ethic", see Appendix A of Free as in Freedom (2.0): Richard Stallmann and the Free Software Revolution and, as always, Wikipedia on "Hacker" or on "Hackerspace" (a.k.a hacklab, hackspace or makerspace).

What is Code?

To a large extent, open source developers are no different from any other software developers. A **must read** for understanding the "software world" (not only open source) and the people living and working in it, is "What is Code?" by Paul Ford for Bloomberg Businessweek. Josh Tyrangiel's introduction states:

> *Software has been around since the 1940s. Which means that people have been faking their way through meetings about software, and the code that builds it, for generations.*

> *[...] a single story devoted to demystifying code and the culture of the people who make it.*

Open Souce (Hardware)

While this book is concerned with software, (a) "Open source" also exists without software, and (b) if your research involves creation of hardware as well, there's a whole other world out there!

Extended reading:

- "Open source" on Wikipedia (without software)
- History of Open Source Hardware at the Open Source Hardware Association

The extra mile

Get into the culture of software developers and open source enthusiasts and try any combination of the following.

- Regularly read XKCD, What if?, Dilbert and Geek&Poke web comics so you can make conversation at lunch.
- Follow technology news on Slashdot or Ars Technica.
- Watch This Week in Tech.
- If you're in Germany, heise online is also good to follow for technology news.

Arguments against Open Source... and How to disprove them

No Professional can help Me if I have Problems

In most cases, the opposite is true. Several factors may even ensure a better support for open source components than for closed source software. For many open source software projects, there is a large developer community available. Communication takes place via the Web, mailing lists, etc. Here you can get direct, first-hand support from the developers of the code. Since most communication is extremely transparent, you can evaluate in advance how these different support channels work when deciding which open source software package to use.

Many open source projects are backed by commercial companies. These companies offer professional support services for specific open source packages. Furthermore, you are not bound to one single vendor (see vendor lock-in) for these support services. Since the source code is open, anyone can develop fixes or enhancements. However, if you rely on proprietary software and the vendor discontinues the product, goes out of business, or if the support contracts offered by the vendor become unattractive, you may run into a problem.

Open Source Software means I cannot sell it

An open source license requires the source code to be published, however, there are many ways to sell it successfully. In many cases, customers need not only the software, but also a package of related materials and services. Thus, an attractive offer including specific support agreements, enhanced documentation, help for installing/integrating the software, training, etc. would make sense. Open Source Software does not mean “non-commerical”!

> ’Free software’ does not mean ‘noncommercial’. A free program must be available for commercial use, commercial development, and commercial distribution. Commercial development of free software is no longer unusual; such free commercial software is very important. You may have paid money to get copies of free software, or you may have obtained copies at no charge. But regardless of how you got your copies, you always have the freedom to copy and change the software, even to sell copies (source).

"Linux is only for free if your Time has no Value" (source)

While it might be true that Linux requires more time to customize it according to your needs, it is not true that this time is wasted. In fact, Linux, as do many other open source software packages, provides a lot more flexibility in setting up systems to fit your needs exactly. It might take more time initially, but you will have a system which allows you to work extremely efficiently in the end.

You cannot earn Money with Open Source

We focus on publishing research outputs as open source software. When we talk about research, the focus is not on money, but on making the research community aware of your work, showcasing your results, and providing a basis that can be re-used and advanced by future research. There are, however, many ways to make money with open source software. The following paragraphs introduce two possible open source business models.

The Open Source Geospatial Foundation provides a compact summary about commercial services for open source software. The information on this page is pretty similar to the revenue opportunities followed by vendors of proprietary software, except that there are no license fees for the software. Typical elements of such an open source business model may comprise consulting, custom developments (e.g. new or modified features) based on an open source software package, maintenance, support with specific service levels, training, etc. Open source software may help researchers successfully submit new research and development project proposals to further advance and improve their software.

Wikipedia also has a comprehensive article on different approaches for open source business models.

For example:

- Dual-licensing: Publish your software under an open source license, but for those users who can't comply with this license, sell commercial licenses.
- Software as a service: Sell your software as a service (e.g. by providing access to your software as a cloud service).
- Proprietary/Open Source combination: Combine your open source software with proprietary software that you might sell (e.g. proprietary extensions).
- Other sources of revenue may comprise the sale of branded merchandise, voluntary donations, partnerships with funding organizations, open-source bounties, advertising-supported software, or crowdfunding.

There are many other opportunities to make money from open source software. Share your experience.

I cannot patent my Code

Software patenting is a highly controversial topic. Depending on the country in which you live/work, it might not even be possible to patent your software.

An important factor in today's science community is to maximize the impact of your research. This is why many funding agencies start requiring publications to be open access and software to be open source. The idea of patenting code is completely contrary to these requirements and the principles of open and reproducible science.

Recommeded reading on software patents should start with Wikipedia article.

Success Stories

Open source software is not a niche product. In fact, this very book is a product created with open source software, such as Git for collaboration and version management, Firefox for editing and most likely Apache or nginx on a Linux web server for content delivery.

There are many success stories in the open source world. Here are a few examples:

- Apache Software Foundation: The Apache Software Foundation is responsible for the Apache software projects. One of the most-well known Apache projects is the Apache HTTP Server, the world's most widely used web server software. The relevance of the Apache projects becomes apparent if you look at the companies supporting the Apache Software Foundation (e.g. Google, Microsoft, IBM, Yahoo).
- LibreOffice: This is an open source office suite published and maintained by The Document Foundation. It is based on OpenOffice.org, from which it was forked in 2010, which was originally an open-sourced version of StarOffice. LibreOffice enjoys quite a large user community and is a common alternative to proprietary products.
- Linux: Linux is one of the most prominent examples of open source projects. From its first release in 1991, it has become a widely used operating system. Popular platforms such as Android rely on the Linux kernel.
- Web browsers: Two of the most popular projects rely on open source licenses. Mozilla Firefox is available under the Mozilla Public License. The Chrome browser published by Google is distributed as freeware, however, significant parts of its source code are available as a project called Chromium under the BSD license.

In the geospatial community, two projects for offering map services contribute to/publish open source software and are the basis for a software as a service business model as well.

- CartoDB offers services to create and host maps on the Web as a freemium service. However, interested users may access the platform implementation as an open source package to install their own instance or to modify it.
- MapBox offers similar services to CartoDB's. Significant modules of this platform have been published as open source software. Furthermore, MapBox has contributed to many other open source projects.

Different organizations needing a certain functionality can team-up to fund the necessary developments. An example hereof is the 52°North Sensor Observation Service (SOS) implementation. This Web service allows users to publish measurement data/sensor data on the Web through a standardized interface. Several partners and (research) projects contributed either funding for 52° North developers or code to develop this component. Due

to recent legal obligations, various European countries need a tool to report their air quality measurements in a standardized format to the European Environment Agency (EEA). To get the necessary 52°North SOS enhancements, several countries (i.e. Belgium, The Netherlands, Sweden and the United Kingdom) have teamed up to fund the development (see also Streamlining European air quality data reporting and exchange via SOS 4.3). This is an excellent example of how different stakeholders can join forces to advance open source software.

Open Hub offers a variety of metrics to assess open source projects: their popularity, activities, etc.

Overviews of successful open source projects are available:

- http://www.itworld.com/article/2827587/enterprise-software/10-most-successful-open-source-projects-of-2012.html
- http://www.techdrivein.com/2010/08/11-biggest-open-source-success-stories.html
- http://royal.pingdom.com/2009/05/29/the-8-most-successful-open-source-products-ever/

Legal Stuff

"IANAL" - I am not a lawyer. We have some experience with open sources licensing, but we cannot give you the professional advice that your require. There are many different legal systems in the world. If you are in doubt about legal matters regarding your open source software or software that you use, then get a lawyer or get in touch with colleagues and your legal department.

There is, however, a plethora of guides and opinions available online and in books. In this section, we introduce some basic information and try to summarize the matter for the most typical scenarios.

Basics

Copyright

©

Copyright is a legal concept that gives the creator of a work certain rights to (exclusively) use that work, grant permissions to other persons, etc. It is intended to protect the creator(s) from economical and ideological abuse of their works. You, as a creator, have the copyright by default and usually you can determine under which conditions your work may be used by others. However, the exact scope depends on the specific applicable law (e.g. the copyright laws of specific countries).

Please note that there are also limitations regarding the copyright, which grant some exceptions for specific cases. This is referred to as “fair use”. We recommend reading this Wikipedia article. Other limitations concern the copyright's expiration after certain periods (depending on the specific laws).

Copyright requires that the work you created has a certain threshold of originality. This threshold is generally rather low. An interesting discussion about this topic can be found here.

History: In the old Roman Empire and the medieval ages, there was no copyright. At that time, it was extremely laborious to copy a text, so this was not really an issue. However, this changed with the invention of the letterpress and led to the “Printer’s Privilige” Act of 1709, also known as the Statute of Anne (British law). Other examples include the Droit d’auteur (France, 1793: authors wanted acknowledgment not money) or the German

Urheberrechtsgesetz (UrhG) from 1965.

Copyleft

If your work is free, you might also want that any modification or extensions of this work be free. This concept is called “copyleft”. Copyleft ensures that your software is free and that all modifications and extensions of your code have to be free too. GNU explains this as follows:

"Proprietary software developers use copyright to take away the users' freedom; we use copyright to guarantee their freedom. That's why we reverse the name, changing 'copyright' into 'copyleft.'"

We distinguish between different types of copyleft. Generally, there are three main categories:

- Strong copyleft requires all derived work to inherit the original copyleft license
- Weak copyleft refers to licenses, which do not require that all derived work inherit the copyleft license. For example, you may link your software to a weak copyleft licensed library without using the same license. However, if you change the original software, the copyleft license needs to be kept. This is useful, for example, if you want to publish libraries that should be re-usable by a broad range of other projects.
- Non-copyleft does not have requirements on the license of derived software.

As you will see later on, the different types of copyleft are an important factor when choosing an open source license for your software.

Recommended reading:

- http://www.gnu.org/copyleft/
- http://en.wikipedia.org/wiki/Copyleft
- http://www.visionmobile.com/blog/2011/03/theopen-source-trials-hanging-in-the-legal-balance-of-copyright-and-copyleft/

Global Trade Agreements

International conventions regulate how to handle copyright between different countries. An important element is the Berne convention.

Software Copyright and Copyright Infringement

All software, proprietary and free and open source, relies on copyright. Although copyright in and of itself has a long history, its application to the specifics of software is still in the making. In the EU and USA, there is mainly Case Law on this topic. The concept of copyright lays the foundation for the conditions of open source licenses.

Recommended Reading:

- Software Copyright

Copyright Infringement

If you are using copyrighted works without permission from the copyright holder, or if you do not follow their terms and conditions, this is called copyright infringement. This could have serious legal consequences. In order to avoid copyright infringements, you should consider several aspects:

- When you use open source code, look for two types of legal information 1) the license(s) of the code and 2) the copyright notices of contributing authors (see also Managing Copyright Information).
- Make sure you understand the compatibility of the license(s) of this code with the license you are using for your own code (and with the licenses of other components you are using).
- When you publish your code under an open source license, you are required not only to provide information about the license you are using yourself, but also about the licenses of all libraries/software that you are using in your source code. There are efficient tools and mechanisms that help you trace contributions and licenses of any third party code in many distributed version control systems.

License Types

In addition to the copyleft licenses "strong copyleft", "weak copyleft" and "non-copyleft" mentioned above, there is a specific category known as "public domain". This covers works for which the intellectual property rights have expired or have been forfeited. Other types of licenses, such as Creative Commons, are not really intended for software. They do not contain any provisions for the distribution of source code. It is NOT recommended to use Creative Commons licenses for software.

Popular Licenses

Open source projects tend to favor certain licenses over others. The following overview shows some of the most commonly used open source software licenses:

- MIT License (MIT) and BSD
  - Non-copyleft licenses
  - Very permissive
  - http://en.wikipedia.org/wiki/MIT_License and http://en.wikipedia.org/wiki/BSD_licenses
- Apache Software License (ASL)
  - Non-copyleft
  - Permissive
  - http://www.apache.org/
- GNU General Public License (GPL)
  - Strong copyleft
  - http://www.gnu.org/licenses/
- GNU Lesser General Public License (LGPL)
  - Weak copyleft
  - http://en.wikipedia.org/wiki/LGPL
  - http://www.gnu.org/licenses/
- Affero General Public License (AGPL)
  - Strong copyleft
  - Closes a loophole that mainly applies to software deployed on Web servers (see http://www.gnu.org/licenses/why-affero-gpl.en.html)
  - http://www.gnu.org/licenses/

See explanations of the different licenses for software code.

Other types of licenses, would be better suited for works other than software code.

- Creative work or art: Licenses such as Creative Commons
- Open data: Licenses such as Open Data Commons (ODC)
- Open hardware: Many projects use licenses inspired by open source software licenses but with important differences (see also Open Source Hardware).

Contributor License Agreements

If you want to contribute to an open source project, you are often required to sign a so-called contributor license agreement (CLA). The signed CLA enables the legal body behind an open source project

- to work with your code contribution (e.g. to reproduce, to modify, to create derivative

works and to combine the contribution with other software code)
- to grant the Open Source License to potential users
- to change the terms of the Open Source License for a particular software project if necessary (e.g. changing from a strong copyleft license to a weak copyleft license)
- to handle legal disputes.

The CLA does NOT require you to transfer the full intellectual property rights (IRP) associated with your copyright (in some countries this is legally impossible and, from a company viewpoint, it is often undesirable). Instead, you transfer only those rights necessary to allow the long-term license management of the project to which you are contributing. The following example of CLAs give you an impression of how they work:

- Apache: http://www.apache.org/licenses/#clas
- 52°North: http://52north.org/about/licensing/contributors-licenseagreement-faq

Recommended readings:

- http://www.oss-watch.ac.uk/resources/cla
- http://en.wikipedia.org/wiki/Contributor_License_Agreement
- http://development.contributoragreements.org/
- https://wiki.eclipse.org/CLA

Notices and License Headers

Copyright notices and license headers are not required to secure copyright. Based on most national laws, the creator of a work automatically holds the copyright the moment the work is created (e.g. if you have written a line of code). However, this does NOT mean that you should not provide copyright notices and license headers as they have certain legal implications.

- If someone infringes your copyright and there are no copyright notices/license headers, he might use this as an argument to reduce your claims.
- The license header makes it easy to immediately determine the author and license of a certain piece of source code.
- You improve your visibility as the author of a certain piece of source code.

You can find more information on this topic here.

Trademarking

Many open source projects (e.g. Eclipse, Apache) rely on trademarking to maintain control

ofa project's identity. You can control better who is using your logo or name and in what context if you have registered corresponding trademarks. As you can imagine, this can become a serious issue.

Recommended Reading:

- http://communityovercode.com/2011/01/trademarks-in-open-source/
- http://www.ifosslr.org/ifosslr/article/view/11

Further Resources

The UK-based organization OSS watch is a great resource addressing many aspects of this chapter: **http://oss-watch.ac.uk/resources/ipr**

People

In this section we briefly list some people you should have heard about from the world of open source and free software.

Richard Stallman

- FSF Founder
- Free software activist
- In 1983 he wrote the "GNU is not Linux" a.k.a. the GNU Manifesto
- Coined the term "Free as in speech vs. free as in beer"
- More: https://de.wikipedia.org/wiki/Richard_Stallman

Eric S. Raymond

- Founder of OSI
- Wrote these notable essays - worth reading!
  - "The Cathedral and the Bazaar"
  - "Homesteading the Noosphere"
- More: http://en.wikipedia.org/wiki/Eric_S._Raymond

Linus Torvalds

- "Inventor" of Linux and git
- More: https://en.wikipedia.org/wiki/Linus_Torvalds

Lawrence Lessig

- Professor at Harvard Law School and political activist
- Cofounder of Creative Commons
- More: https://en.wikipedia.org/wiki/Lawrence_Lessig

Guide: What license should I use for my software project?

You realize **having no license does not make your project usable by others**. If not, please read the entry about "No license" in the FSF license list and this study about licenses on GitHub.

There are so many choices out there and the open source license world is complex. We strongly recommend to go with one of the most used and established licenses. These are divided into three broad categories and briefly explained by GitHub folks at choosealicense.com. They don't suggest any license in parcticular and even explain the "no license" option. Choosing a license is now a two-minute-task.

Based on *our experience*, we recommend the Apache License. It is closer to the real world, as compared to the "short" licenses (MIT, BSD). It takes into consideration that the real world is a bad place with software patents, but also allows any commercial application to integrate and extend a software project. Your own ideas, third-party libraries, i.e. software that you use or extend, have an important impact on choosing a license.

The minimum number of steps required for choosing a license for software resulting from a research project are as follows.

1. Create a comprehensive list of your "third-party licenses" - software that you strictly rely on or directly extend and their licenses.
2. Go to choosealicense.com and pick a license that fits your goals.
3. Check all third-party licenses for compatibility with your chosen license.
   This requires web searches and potential involvement of a lawyer specializing in open source if the dependencies' licenses are not typical ones. Sadly, we are NOT lawyers and can NOT make clear recommendations here. Check the following websites, but judge for yourself if the recommendations apply to you. In 90% of the cases, the answer is straightforward, but due diligence is required.
   - "Comparison of free and open-source software licenses" on Wikipedia
   - OSSWATCH licence differentiator
   -
   -
   - 52°North's list of third-party components
   -
4. Make sure ALL software contributors are ok with the chosen license.
5. Check with guidelines or agreements accompanying the funding for your development, e.g. research grants.
6. Revise your decision if any of 3. to 5. result in issues. If choosealicense.com is not detailed enough, answer seven questions in the OSSWATCH licence differentiator.

7. Done.

For more extensive lists of open source licenses, please see the list of OSI-approved licenses and the license list by the FSF. The latter also has very short summaries and recommendations for laypeople yet focuses on the FSF's perspective.

GitHub is *the* platform for open source software development. Millions of projects use it (cf. GitHut). Its success comes from its great collaboration workflow, which allows developers, documenters and users to build something together. GitHub enables the idea of social coding, a software that is developed effectively by a distributed virtual community.

GitHub provides extensive high quality online documentation: https://help.github.com/. Thus, we give you a brief overview and describe our opinion of why GitHub simply "works". We also link to the relevant contents of the excellent GitHub Bootcamp.

Basics: Git, Accounts & Repositories

GitHub is a free platform for code and task management. At the core, it is based on an open source version control system (VCS) called Git.

There are many tutorials to help you get started (Getting Started Git Basics).

If you do not like to work with the command line, or - god forbid - you are not working on Linux, install Git (available for all operating systems) and a Git user interface.

We **strongly recommend** to start with Git at the command line. Very experienced and intense Git users do not use an UI! It will help understand how Git works if you type commands instead of clicking fancy buttons.

Sign up - https://github.com/join - to get an account. Create your first *repository* - something like a project base folder - with a single click and you are ready to go. Or, *fork* a repository, i.e. make a copy of an existing repository, and make the copy your own. This way your work won't affect the original project.

Bootcamp articles:

- Set up Git
- Fork a Repo
- Good Resources for Learning Git and GitHub

Fork & Pull Workflow

Even when you work with only two people, we highly recommend the "Fork & Pull" development model. From the GitHub help on pull requests:

> *The fork & pull model lets anyone fork an existing repository and push changes to their personal fork without requiring access be granted to the source repository. The changes must then be pulled into the source repository by the project maintainer. This model reduces the amount of friction for new contributors and is popular with open source projects because it allows people to work independently without upfront coordination.*

The advantages are also true for projects with close collaboration, i.e. where there is a lot of upfront coordination, maybe even within the same office space:

- Everybody can do what they want within their fork.
- Changes to the main repo can be *discussed and reviewed* within the pull requests.
- Access control to the main repository are with the project maintainer.
- Pull requests work between any of the forks.

Bootcamp articles:

- Fork a Repo

Further reading:

- GitHub Glossary
- Using pull requests

Social Coding

GitHub is so successful because, it makes collaboration with others easy. Successful collaboration between people makes (software) projects themselves a success.

> *Code is about the people writing it. We focus on lowering the barriers of collaboration by building powerful features into our products that make it easier to contribute. The tools we create help individuals and companies, public and private, to write better code, faster.*

The GitHub old mission statement

Chris Wanstrath, one of the GitHub founders, gave a great talk (60 min.) at the Esri User Conference 2014 about GitHub and open source in general. Recommended watch.

After viewing the video, "watch" a couple of repositories and "follow" some people to stay updated. Explore users profiles to see what people are working on and connect with those who use similar software or do work related to your own.

Bootcamp articles:

- Be social

Further reading on social coding:

- Social coding -- the next wave in development (TechRepublic)
- Eclipse wiki and FAQ

If I work on my own, do I still need GitHub?

Yes! See it as a backup solution, but Git improves your workflow, it encourages you to wander off with new ideas (*branches*), switch between them, and even merge them. And GitHub is simply a great Git user interface. Friends and colleagues can follow your work on GitHub and become collaborators.

With Git (or any VCS), you work in plain text files and keep code and documentation closely together. This ensures that anybody can understand what is going on in one week, one month, or one year.

GitHub for Education

GitHub offers educational institutes the opportunity to run classes on GitHub or put a PhD in a private repository before publishing the accompanying paper!

Two basic aspects are explained in detail on the GitHub Education landing page:

Educators: You can teach more effectively by having a platform to collect assignments, distribute starter code and discuss issues with students "within" their code. You can take advantage of the free Organization accounts for your classes or get in touch with GitHub to request discounts.

Students: You will stay better organized and won't loose your work. You will also build a portfolio that displays your skills once you start looking for a job. There is more: the Student Developer Pack gives you free access to developer tools and infrastructure, such as cloud hosting or domains, for a limited time or as long as you are eligible.

Software Communities

In this chapter we tackle the human side of open source projects.

How can you build a community that lasts? How can you organize a project that serves different needs, schedules and people? How can you show what your project can do? How can you get help to take your project to the next level?

This section is *work in progress*.

We have sections in the making about community building, openness, as well as marketing and public relations.

Marketing and Public Relations

As Zach Holman puts it, "Open Source doesn't just market itself". Communication and interaction with the public and other developers is important to help the community grow and make your hard work known to the world!

The following quick reads provide an idea of what is important if you want your software/project to be used:

- Open Source Doesn't Just Market Itself (Zach Holman)
- Marketing open source is made for geeks (Sandro Groganz)
- Open-Source Projects Need More Than Good Code—They Need Marketing (Matt Asay)

We particularly like the emphasis Zach Holman puts on the value of great documentation as marketing!

Social Media is the way to go

The fastest way to disseminate information is to make use of social media. There are lots of different platforms, but usually there isn't a social media expert on hand to juggle them. Two important but "managable" means are blogging and twittering.

The blog

"You have something to say, and blogs provide a place to say it and be heard." (source)

A blog is more informal than a press release. It is "... a discussion or informational site published on the World Wide Web and consisting of discrete entries ("posts") typically displayed in reverse chronological order (the most recent post appears first)." (source)

If you are not aware of the concept of blogging, these helpful links provide background information:

- http://weblogs.about.com/od/startingablog/p/WhatIsABlog.htm
- http://codex.wordpress.org/Introduction_to_Blogging
- http://weblogs.about.com/od/startingablog/tp/Top-Ten-Reasons-to-Blog.htm

How to start blogging?

- http://www.wikihow.com/Start-a-Blog
- http://startbloggingonline.com/

- http://howtomakemyblog.com/10-elements-of-style-of-blog-post-writing/

Which blogging platform is best for me?

- http://thenextweb.com/apps/2013/08/16/best-blogging-services/
- Wordpress (https://en.wordpress.com/) is the number 1 platform.
- Blogger (full-fledged blog) and Tumblr (an interesting fusion between a full-fledged blog and a Twitter feed) are among the top 3 platforms.

Guide for blogging

- Post regularly, but don't post if you have nothing worth posting about.
- If you want to involve others in the project development, get feedback about certain developments, build the community, don't just blog about project results, share your ideas, plans, etc.
- Comment on other peoples' blogs (they normally visit back).

Microblogging with Twitter

"Twitter is an online social networking service that enables users to send and read short 140-character messages called "tweets" (source). There's a lot more to twitter than microblogging, but it's a start.

- http://webtrends.about.com/od/socialnetworking/a/what-is-twitter.htm
- http://www.wikihow.com/Use-Twitter
- http://mashable.com/guidebook/twitter/

Guide for Twitter basics:

1. First create useful content: e.g. blog, newsticker, presentation, etc.
2. Then twitter:
    - 140 characters in a tweet: It should contain a specific point from a longer piece of content, or give an idea of what to learn from content.
    - Use shortened URLS to save characters. There are a number of free link shortener tools available, e.g. http://ow.ly/url/shorten-url or https://bitly.com/
    - Use keywords (#hashtags) to find topics and have your topics be found.
    - Use @usernames to reach users with similar interests to that @username's followers.
    - Use pictures. Particularly when at an event. http://www.socialquant.net/images-for-twitter/

3. Keeping tabs on tweets
    - Keep it under control with Tweetdeck (https://tweetdeck.twitter.com/). Tweetdeck is a nice tool, which gives you an overview of tweets from those you follow as well as yours, notifications, messages and activities.
    - Twitter analytics (http://analytics.twitter.com) provides basic statistical information about your tweets.

Information dissemination via paper – outdated?

Although most people look for information online these days, there is still a need for printed material. Do you have a new project, software, solution, application? How about a cheat sheet/flyer?

First of all, be clear about who you want to address and what you want to achieve. Is your target group Users? Developers? ...?

Tips for flyers:

- Produce the flyer when the topic is hot, don't wait for an event.
- Keep the format simple and easy to update, e.g DIN A4 single or double paged.
- Make the flyer available to download online.
- Include QR codes (link to home page /product page, repository, i.e.GitHub or other repository).

Further topics

- managing expectations (openness)

Guide: How should I name my project?

If you worry about a *name*, you must have figured out all the other issues!

The first thing to consider is: Do I really need a name for a new open source project, or would it not be better to contribute my new code as an extension to an existing and established project?

If it is to be a new project, take a look at related projects and communities within the language or framework you base your code on. Are abbreviations common? Do people use hip or cool names? Check for existing names.

- https://www.google.de/search?q=good+software+project+names (trademarks, …)
- http://www.webdeveloper.com/forum/showthread.php?113873-Great-software-project-names&s=14592899cd3b32c3df12f39bf4616c5e&p=605183#post605183

Tips for naming:

- Find a good name
  - Avoid abbreviations
  - Be visionary and choose a name that fits the overall goal of what your code does, not a specific solution
  - Prefer existing "real world" words - they are a lot easier to remember
- Search for other projects with the same name, also consider trademarks (potentially from completely different applications)
- If a project exists, start from the beginning

This is an opinionated view. What is your experience? *Get in touch and contribute!*

Further resources

- ...

Logos

t.b.d.

Types of Contributors and Tasks

There are many tasks in open source projects aside from actually writing code. In this section, we point to some articles worth reading presenting the different types of contributions you can make yourself or that you can encourage others to give your project.

This extensive list nicely shows how many different ways there are - so get started now and participate yourself and motivate others.

- 7 Ways Non-programmers Can Contribute to Open Source Projects
  - use the product
  - bug test
  - write documentation
  - translation
  - evangelise
  - donate
  - be professional
- 10 ways to contribute to an open source project without writing code
  - provide reports
  - create feature requests
  - test the code
  - write documentation
  - translate the UI and documentation
  - answer questions on forums and mailing lists
  - help design UI/logo/website
  - promote the project
  - provide hardware
  - thank the community
- 14 Ways to Contribute to Open Source without Being a Programming Genius or a Rock Star
  - start listening
    - join mailing list
    - follow blog
    - join IRC channel
  - work with tickets
    - diagnose a bug
    - close fixed bugs
  - work with code
    - beta test

        - fix a bug
        - write a test
        - silence a compiler warning
        - add a comment
    - work with documentation
        - create an example
    - work with community
        - answer a question
        - write a blog post
        - improve a website
- How to Contribute to Open Source in 10 steps

But of course, things are not that simple. The following paper is an extended reading on what barriers newcomers might face: A systematic literature review on the barriers faced by newcomers to open source software projects by I. Steinmacher, M.A. Graciotto Silva, M.A. Gerosa, D.F. Redmiles, A, Information and Software Technology (2014), doi: http://dx.doi.org/10.1016/j.infsof.2014.11.001

Open Source in Your Domain

We need your contributions to fill this section of the book. Here we want to collect links for specific domains, such as biology, biodiversity, physics, geology, computer graphics, ... to provide a starting point for researchers, especially PhD students, to get to know exisiting projects as well as common practices integrating research with open source software.

Scientific Publishing of Data and Software

There is a growing list of papers and platforms that support the publishing of scientific data and software. This guide is mostly concert with *creating* software, thus this section is a list of (partially commented) links to platforms, journals, repositories, etc.

Please note, data is *not* like software. While GitHub can be used for publishing and collecting data as well, you should see if there are more suitable platforms. The field of publishing software and data is an area of research itself: Why not collaborate and be one of the first in your lab, institution or domain to explore this avenue?

The idea of "weaving" code (one of the two aspects of literal programming, "tangling" being the second) and data into a single document, for example Sweave for R or Pweave for Python, is worth exploring for anyone using code within research, even if just for creating plots.

Contributions to the following list are very welcome!

- List of software journals by the Software Sustainability Institute
- EGU 2015 short course "Open Science goes Geo - Part II: Scientific Software", covers journals for publishing software and platforms for publishing (with DOIs etc.)
  - Recordings
  - Slides
- sciforge
- DataCite
- Zenodo
- Figshare
- Dryad
- PANGEA
- RunMyCode
- ResearchCompendia
- Victoria Stodden (Prof. at Graduate School of Library and Information Science, at the University of Illinois at Urbana-Champaign): Wiki: Best Practices for Researchers Publishing Computational Results
- Eclipse Science WG
- ...

This is the contribute.md of 52°North best practice document Publishing Research Softwas as Open Source on GitHub. Great to have you here! Here are a few ways you can help!

Contribute.md

Spreading the work

You like the best practice and it helped you? Great! Please talk about it:

- On Twitter, using the hashtag `#pubopen @fivetwon`.
- In your blog, linking to https://github.com/52North/pubopen and/or http://52north.gitbooks.io/pubopen
- Write to us: gitbook@52north.org
- In a publication, use the following citation: *Nüst, Daniel, Simon Jirka, and Ann Hitchcock. Publishing Research Software as Open Source on GitHub. June 2015. url: https://www.gitbook. com/book/52north/pubopen*
- Bibtex entry:

```
@Book{2015:52north:pubopen,
Title                    = {Publishing Research Software as Open Source on GitHub},
Author                   = {N\"ust, Daniel and Jirka, Simon and Hitchcock, Ann},
HowPublished             = {online},
Month                    = {jun},
Year                     = {2015},
Url                      = {https://www.gitbook.com/book/52north/pubopen}
}
```

Correcting and improving content

You found a spelling error? A sentence could be misunderstood and you would like to clarify it?

Great! This best practice is an open source document hosted on GitHub.

- Before you contribute your changes, please read the used license (see README.md) - by contributing to this book in any way you signal your understanding of the license and its implications.
- Fork the repository on GitHub.
- Make your changes to the respective files. Split them into thematically sound git commits with informative commit messages.

- Create a pull request.

Adding new sections

This section includes advice on how to add new sections to the best practice.

- Before you contribute your changes, please read the used license (see README.md) - by contributing to this book in any way you signal your understanding of the license and its implications.
- Open a new issue on GitHub and explain what information is missing in the best practice and why.
- After discussing with the maintainers of the work, fork the repository on GitHub.
- Make your changes.
- Create a pull request.

Translations

Currently, no translations of the book are planned. Do you think this would be useful? Open an issue on GitHub to discuss your ideas with the maintainers.

Documentation

- Using GitHub: http://help.github.com/
- Using GitBook: http://help.gitbook.com/
- This documentation is not sufficient? Contribute some that you were missing when you started!

If you have further questions, contact: gitbook@52north.org

Glossary

FSF

The Free Software Foundation (FSF) is a nonprofit with a worldwide mission to promote computer user freedom and to defend the rights of all free software users, http://www.fsf.org/

3.6. Guide 3.1. Mindset 3.5. People

OSI

The Open Source Initiative (OSI) is a global non-profit focused on promoting and protecting open source software, development and communities, http://opensource.org/

3.1. Mindset 3.5. People

OSI-approved

Open source licenses are licenses that comply with the Open Source Definition and completed OSI license review process, see http://opensource.org/licenses.