
HOW TO PACKAGE AND RELEASE YOUR CODE

Bron Reichardt Chu
Durham University



WHY?

Reproducibility

Robustness

Versatility

All publicly funded research should be open-source

OPTIONS



Minimal:

**release it on your
webpage**

GADGET-2 released this way

Unclear how to cite

Lack of transparency - no
change-log

Can be interpreted as less
trustworthy

OPTIONS



Minimal:

**release it on your
webpage**

GADGET-2 released this way

Unclear how to cite

Lack of transparency - no
change-log

Can be interpreted as less
trustworthy



Normal:

release on GitHub

Repo + license -> create a
release

Get a Zenodo DOI -> permanent
link, citeable

Repeat for every new version

Shows more software competence -
may be interpreted as more
trustworthy

OPTIONS



Minimal:

**release it on your
webpage**

GADGET-2 released this way

Unclear how to cite

Lack of transparency - no
change-log

Can be interpreted as less
trustworthy



Normal:

release on GitHub

Repo + license -> create a
release

Get a Zenodo DOI -> permanent
link, citeable

Repeat for every new version

Shows more software competence -
may be interpreted as more
trustworthy



Maximal:

**software paper (e.g.
JOSS)**

Requires documentation

Add to a package manager (conda
and/or PyPI)

Testing

Takes time, which is time not
spent on research

CHECKLIST: MINIMAL RELEASE



- ☐ A license
- ☐ A clear readme
- ☐ Documentation
 - o Examples
- ☐ A clear link on your website/in your paper

LICENSES



Grant specific permissions to the user and impose restrictions on how the code can be used, copied and distributed.

A license protects your intellectual property rights.

It also lets other people use your code, which would otherwise be protected by copyright laws.

Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.

{ Which of the following best describes your situation? }



I need to work in a community.

Use the **license preferred by the community** you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to **add a license**.



I want it simple and permissive.

The **MIT License** is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

Babel, **.NET**, and **Rails** use the MIT License.



I care about sharing improvements.

The **GNU GPLv3** also lets people do almost anything they want with your project, *except* distributing closed source versions.

Ansible, **Bash**, and **GIMP** use the GNU GPLv3.

LICENSES



>> For more information:

>> On the legalities of open-source code: <https://opensource.guide/legal/>

>> On how to choose a license: <https://choosealicense.com/>

>> On how to add a license to your git repo:
<https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository>

CHECKLIST: MINIMAL RELEASE



- ☒ A license
- ☐ A clear readme
- ☐ Documentation
 - o Examples
- ☐ A clear link on your website/in your paper

README



>> Brief description of your code

>> What does it do?

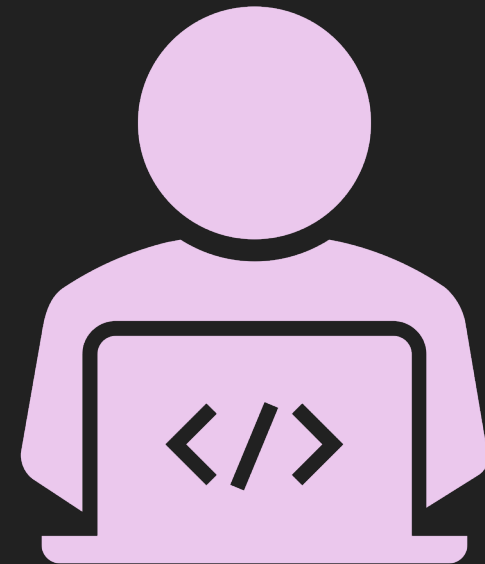
>> How does it do that?

>> Installation instructions

>> Brief example of usage

>> Python packaging user guide:

<https://packaging.python.org/en/latest/overview/>



CHECKLIST: MINIMAL RELEASE



- ☒ A license
- ☒ A clear readme
- ☐ Documentation
 - o Examples
- ☐ A clear link on your website/in your paper

CHECKLIST: GITHUB RELEASE

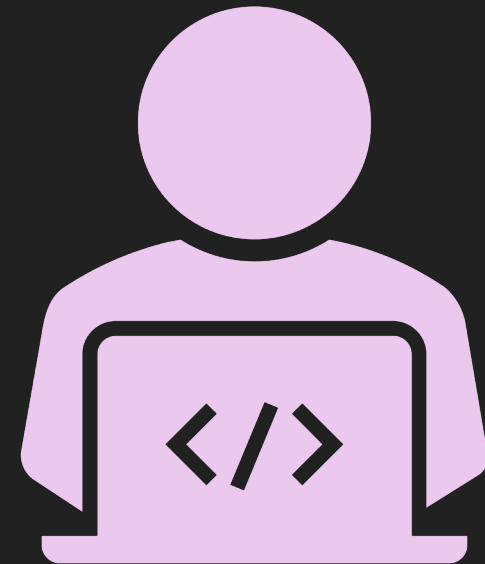


- ☒ A license
- ☐ A clear readme
- ☐ Documentation
 - o Examples
 - o API Documentation
 - o Sphinx
- ☐ Tag your release version
- ☐ A persistent identifier: Zenodo

README



```
>> Brief description of your code
  >> What does it do?
  >> How does it do that?
>> Installation instructions
>> Brief example of usage
>> Requirements (what do you need in your
environment?)
>> How to contribute
>> How to cite your code
```

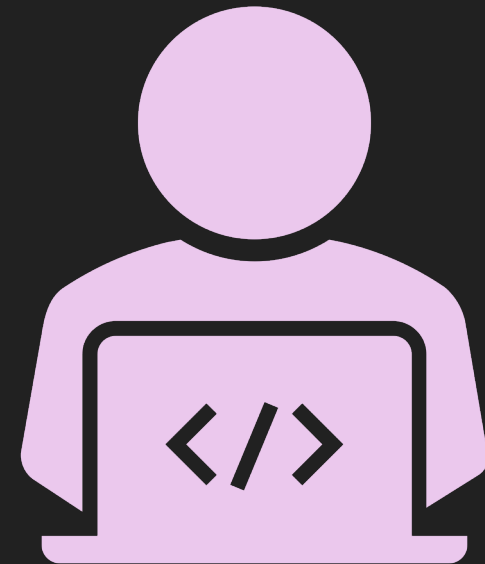


README

```
>> Brief description of your code
  >> What does it do?
  >> How does it do that?
>> Installation instructions
>> Brief example of usage
>> Requirements (what do you need in your
environment?)
>> How to contribute
>> How to cite your code
```



*This can be as simple as
git clone instructions*



CHECKLIST: GITHUB RELEASE



- ☒ A license
- ☒ A clear readme
- ☐ Documentation
 - o Examples
 - o API Documentation
 - o Sphinx
- ☐ Tag your release version
- ☐ A persistent identifier: Zenodo

SPHINX

MyProject/

|--src/

|-- __init__.py

|-- module1.py

|--tests/

|-- __init__.py

|--test_module1.py

|--docs/

|--examples/

|--tutorial.ipynb

|--.gitignore

|--LICENSE

|--README.md

|--CITATION.cff

|--pyproject.toml

|--requirements.txt



SPHINX

MyProject/

```
|--src/
  |-- __init__.py
  |-- module1.py
|--tests/
  |-- __init__.py
  |--test_module1.py
|--docs/
|--examples/
  |--tutorial.ipynb
|--.gitignore
|--LICENSE
|--README.md
|--CITATION.cff
|--pyproject.toml
|--requirements.txt
```




>> To install:

- `$ pip install -U sphinx`
- `$ conda install sphinx`
- `$ pip install myst-parser`
- `$ conda install -c conda-forge myst-parser`

SPHINX

```
MyProject/
|--src/
|   |-- __init__.py
|   |-- module1.py
|--tests/
|   |-- __init__.py
|   |--test_module1.py
|--docs/
|--examples/
|   |--tutorial.ipynb
|--.gitignore
|--LICENSE
|--README.md
|--CITATION.cff
|--pyproject.toml
|--requirements.txt
```



>> To setup, from within your docs folder:

- `$ sphinx-quickstart`

>> Separate source and build directories (y/n) [n]: <hit enter>

>> Project name: <your project name>

>> Author name(s): <your name>

>> Project release []: 0.1

>> Project language [en]: <hit enter>

SPHINX



```
MyProject/
|--src/
    |-- __init__.py
    |-- module1.py
|--tests/
    |-- __init__.py
    |--test_module1.py
|--docs/
    |--build
    |--make.bat
    |--Makefile
    |--source/
        |--conf.py
        |--index.rst
        |--_static
        |--_templates
        |--examples/
            |--tutorial.ipynb
        |--.gitignore
        |--LICENSE
        |--README.md
        |--CITATION.cff
        |--pyproject.toml
        |--requirements.txt
```

>> To enable markdown, in conf.py:

- `extensions = ["myst_parser"]`

<https://myst-parser.readthedocs.io/en/v0.16.1/sphinx/intro.html>

SPHINX



```
MyProject/
|--src/
|   |-- __init__.py
|   |-- module1.py
|--tests/
|   |-- __init__.py
|   |--test_module1.py
|--docs/
|   |--build
|   |--make.bat
|   |--Makefile
|   |--source/
|       |--conf.py
|       |--index.rst
|       |--_static
|       |--_templates
|--examples/
|   |--tutorial.ipynb
|--.gitignore
|--LICENSE
|--README.md
|--CITATION.cff
|--pyproject.toml
|--requirements.txt
```

>> Build from within source/:

- `$ sphinx-build . _build`

>> Tip: Creates `_build` folder, which you should add to your `.gitignore` so that it is not added to your Git repo

SPHINX

MyProject/

|--src/

|-- __init__.py

|-- module1.py

|--tests/

|-- __init__.py

|--test_module1.py

|--docs/

|--build

|--make.bat

|--Makefile

|--source/

|--conf.py

|--index.rst

|--_static

|--_templates

|--_build/

|--index.html

|--examples/

|--tutorial.ipynb

|--.gitignore

|--LICENSE

|--README.md

|--CITATION.cff

|--pyproject.toml

|--requirements.txt



>> To see your website locally:

Linux:

- `$ xdg-open _build/index.html`

macOS:

- `$ open _build/index.html`

Windows:

- `$ start _build/index.html`

SPHINX

```
MyProject/
|--src/
|   |-- __init__.py
|   |-- module1.py
|--tests/
|   |-- __init__.py
|   |-- test_module1.py
|--docs/
|   |--build
|   |--make.bat
|   |--Makefile
|   |--source/
|       |--conf.py
|       |--index.rst
|       |--_static
|       |--_templates
```

```
|--examples/
|   |--tutorial.ipynb
|--.gitignore
```



>> Auto-generate documentation - in the conf.py file:

- `extensions = ["myst_parser", "autodoc2"]`
- `autodoc2_packages = ["multiply.py"]`

>> In the index.rst file:

- `.. toctree::`
- `:maxdepth: 2`
- `:caption: Contents:`
- `apidocs/index`

<https://sphinx-autodoc2.readthedocs.io/en/latest/quickstart.html>

SPHINX



```
MyProject/
|--src/
    |-- __init__.py
    |-- module1.py
|--tests/
    |-- __init__.py
    |--test_module1.py
|--docs/
    |--build
    |--make.bat
    |--Makefile
    |--source/
        |--conf.py
        |--index.rst
        |--_static
        |--_templates
        |--examples/
            |--tutorial.ipynb
        |--.gitignore
        |--LICENSE
        |--README.md
        |--CITATION.cff
        |--pyproject.toml
        |--requirements.txt
```

>> Build from within source/:

- `$ sphinx-build . _build`

>> Tip: Creates `_build` folder, which you should add to your `.gitignore` so that it is not added to your Git repo

SPHINX



```
>> Host your docs on a GitHub  
pages website:
```

```
https://coderefinery.github.io/  
documentation/gh\_workflow/
```

CHECKLIST: GITHUB RELEASE



- ☒ A license
- ☒ A clear readme
- ☒ Documentation
 - o Examples
 - o API Documentation
 - o Sphinx
- ☐ Tag your release version
- ☐ A persistent identifier: Zenodo

TAGGING A RELEASE ON GITHUB

The screenshot shows the GitHub interface for the 'BlackWidow' repository. The repository is public and has 0 watches, 0 forks, and 0 stars. A notification bar at the top indicates that 'dev_ext-corr' had recent pushes 38 minutes ago, with a 'Compare & pull request' button. The repository has 2 branches (main) and 0 tags. The commit history shows a series of commits by 'Bjarki-B' starting with 'Initial commit' and followed by several 'Empty directories added' and 'Added' commits. The file list includes 'BlackWidowPipeline', 'docs', 'tests', '.gitignore', '.readthedocs.yaml', 'CODEOFCONDUCT.md', 'CONTRIBUTING.md', 'NEWS.md', 'README.md', 'blackwidow_env.yml', and 'pyproject.toml'. On the right sidebar, the 'About' section is visible, followed by the 'Releases' section which is circled in blue. The 'Releases' section shows 'No releases published' and a link to 'Create a new release'. Below this is the 'Packages' section showing 'No packages published' and a link to 'Publish your first package'. At the bottom of the sidebar is the 'Contributors' section showing 5 contributors.

BlackWidow Public

dev_ext-corr had recent pushes 38 minutes ago [Compare & pull request](#)

main 2 Branches 0 Tags

Go to file Add file Code

Bjarki-B <feat> Added pytest to .yaml for env 011f9c0 · 1 hour ago 17 Commits

File	Commit Message	Time
BlackWidowPipeline	<Setup> Empty directories added	2 hours ago
docs	<Setup> Empty directories added	2 hours ago
tests	<Setup> Empty directories added	2 hours ago
.gitignore	Initial commit	yesterday
.readthedocs.yaml	<Setup> Added .readthedocs.yaml	yesterday
CODEOFCONDUCT.md	Created CODEOFCONDUCT	yesterday
CONTRIBUTING.md	<style> Fixed markdown for CONTRIBUTING.md	1 hour ago
NEWS.md	<Setup> Added a NEWS.md file	yesterday
README.md	<feat> Added instructions to create a conda environment ...	1 hour ago
blackwidow_env.yml	<feat> Added pytest to .yaml for env	1 hour ago
pyproject.toml	Added member information	yesterday

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Contributors 5

New release

Tag: **Select tag** ▾

Target: **main** ▾

Choose an existing tag, or create a new tag when you publish this release.

Release title

Title

Release notes

Generate release notes

Write

Preview

H B I | | @

Describe this release

Markdown is supported

Paste, drop, or click to add files

↓ Attach binaries by dropping them here or selecting them.

☐ Set as a pre-release
This release will be labeled as a pre-production ready

Publish release

Save draft

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be v1.0.0 or v2.3.4.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2.0-alpha or v5.9-beta.3.

Semantic versioning

If you're new to releasing software, we highly recommend to [learn more about semantic versioning](#).

A newly published release will automatically be labeled as the latest release for this repository.

If 'Set as the latest release' is unchecked, the latest release will be determined by highest semantic version and creation date. [Learn more about release settings](#).

TAGGING A RELEASE ON GITHUB

New release

Tag: Select tag

Target: main

Choose an existing tag, or create a new tag when you publish this release.

Release title

MAJOR.MINOR.PATCH

Release notes

>> MAJOR version when you make incompatible API changes
>> MINOR version when you add functionality in a backward compatible manner
>> PATCH version when you make backward compatible bug fixes

For more on semantic versioning: <https://semver.org/>

For more on how to create a release:

<https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository>

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be v1.0.0 or v2.3.4.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2.0-alpha or v5.9-beta.3.

Semantic versioning

If you're new to releasing software, we highly recommend to [learn more about semantic versioning](#).

A newly published release will automatically be labeled as the latest release for this repository.

If 'Set as the latest release' is unchecked, the latest release will be determined by higher semantic version and creation date. [Learn more about release settings](#).

TAGGING A RELEASE ON GITHUB

Publish release

Save draft

CHECKLIST: GITHUB RELEASE



- ☒ A license
- ☒ A clear readme
- ☒ Documentation
 - o Examples
 - o API Documentation
 - o Sphinx
- ☒ Tag your release version
- ☐ A persistent identifier: Zenodo

ZENODO

DOI: Digital Object Identifier

Log in with GitHub

<https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content>

zenodo

Search records...

Q

Communities

My dashboard

Log in

Sign up

There is a [newer version](#) of the record available.

Published September 21, 2014 | Version 0.8.0

Software Open

LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python

Newville, Matthew¹ ; Stensitzki, Till² ; Allen, Daniel B.³ ; Ingargiola, Antonino⁴ [Show affiliations](#)

Lmfit provides a high-level interface to non-linear optimization and curve fitting problems for Python. Lmfit builds on Levenberg-Marquardt algorithm of `scipy.optimize.leastsq()`, but also supports most of the optimization method from `scipy.optimize`. It has a number of useful enhancements, including:

- Using Parameter objects instead of plain floats as variables. A Parameter has a value that can be varied in the fit, fixed, have upper and/or lower bounds. It can even have a value that is constrained by an algebraic expression of other Parameter values.
- Ease of changing fitting algorithms. Once a fitting model is set up, one can change the fitting algorithm without changing the objective function.
- Improved estimation of confidence intervals. While `scipy.optimize.leastsq()` will automatically calculate uncertainties and correlations from the covariance matrix, Lmfit also has functions to explicitly explore parameter space to determine confidence levels even for the most difficult cases.
- Improved curve-fitting with the Model class. This which extends the capabilities of `scipy.optimize.curve_fit()`, allowing you to turn a function that models for your data into a python class that helps you parametrize and fit data with that model.
- Many pre-built models for common lineshapes are included and ready to use.

The Lmfit package is Free software, using an MIT license

Files

Files (846.7 kB)

37K VIEWS

2K DOWNLOADS

Show more details

	All versions	This version
Views	37,086	23,286
Downloads	2,180	886
Data volume	2.1 GB	811.1 MB

More info on how stats are collected...

Versions

Version 1.3.4	Jul 19, 2025
10.5281/zenodo.16175987	
Version 1.3.3	Mar 12, 2025
10.5281/zenodo.15014437	
Version 1.3.2	Jul 19, 2024
10.5281/zenodo.12785036	

CHECKLIST: JOSS RELEASE



- ☒ A license
- ☒ A clear readme
- ☒ Documentation
 - o API Documentation
 - o Examples
 - o Sphinx
- ☐ Ability to pip install your code
- ☐ Software paper

CHECKLIST: JOSS RELEASE



>> How to make
your package
pip
installable:
<https://packaging.python.org/en/latest/tutorials/packaging-projects/>

- ☒ A license
- ☒ A clear readme
- ☒ Documentation
 - o API Documentation
 - o Examples
 - o Sphinx
- ☐ Ability to pip install your code
- ☐ Software paper

JOSS REQUIREMENTS

Bron Reichardt Chu | ASTRODAT

Functionality

- **Installation:** Does installation proceed as outlined in the documentation?
- **Functionality:** Have the functional claims of the software been confirmed?
- **Performance:** If there are any performance claims of the software, have they been confirmed? (If there are no claims, please check off this item.)

Documentation

- **A statement of need:** Do the authors clearly state what problems the software is designed to solve and who the target audience is?
- **Installation instructions:** Is there a clearly-stated list of dependencies? Ideally these should be handled with an automated package management solution.
- **Example usage:** Do the authors include examples of how to use the software (ideally to solve real-world analysis problems).
- **Functionality documentation:** Is the core functionality of the software documented to a satisfactory level (e.g., API method documentation)?
- **Automated tests:** Are there automated tests or manual steps described so that the functionality of the software can be verified?
- **Community guidelines:** Are there clear guidelines for third parties wishing to 1) Contribute to the software 2) Report issues or problems with the software 3) Seek support

Software paper

- **Summary:** Has a clear description of the high-level functionality and purpose of the software for a diverse, non-specialist audience been provided?
- **A statement of need:** Does the paper have a section titled 'Statement of need' that clearly states what problems the software is designed to solve, who the target audience is, and its relation to other work?
- **State of the field:** Do the authors describe how this software compares to other commonly-used packages?
- **Quality of writing:** Is the paper well written (i.e., it does not require editing for structure, language, or writing quality)?
- **References:** Is the list of references complete, and is everything cited appropriately that should be cited (e.g., papers, datasets, software)? Do references in the text use the proper [citation syntax](#)?



12/9/25

34

SUMMARY

- Easily accessible
- Clear installation and usage instructions
- Clear how to cite it



MORE RESOURCES

>> Making good README files:

<https://www.makeareadme.com/>

>> Convert from markdown to rst:

<https://pandoc.org/>

>> How to document your research software:

<https://coderefinery.github.io/documentation/wishlist/>

>> Python packaging user guide:

<https://packaging.python.org/en/latest/overview/>

>> Sphinx documentation:

<https://www.sphinx-doc.org/en/master/usage/quickstart.html>

>> Connecting Sphinx documentation to a GitHub Pages website:

https://coderefinery.github.io/documentation/gh_workflow/

>> Connect Zenodo DOI to GitHub release:

<https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content>

>> Make your package conda installable:

<https://docs.conda.io/projects/conda-build/en/stable/user-guide/tutorials/building-conda-packages.html>

>> GitHub's instructions for building and testing Python packages:

<https://docs.github.com/en/actions/tutorials/build-and-test-code/python>

>> Run your tests automatically every time you push changes to the repo:

<https://coderefinery.github.io/testing/continuous-integration/>

>> Submitting a paper to JOSS:

<https://joss.readthedocs.io/en/latest/submitting.html>