

# Research Software Engineering Meetups: Sharing, Publication, and Collaboration

2023/10/23

# Overview

- Schedule/upcoming
- Sharing/collaboration overview
- Version control systems (git)
- Typical steps (interactive/group work)

# Schedule (provisional)

Date	Location	Rough Topic	Status
16th October	34-E-3180	Introduction & tooling	Done
23rd October	34-E-3180	Sharing, publication, and collaboration	In development
30th October	34-E-3180	Software Design: command-lines, UIs, and notebooks	Topic Suggested
6th November	34-E-3180	(topic TBD: based on feedback)	
13th November	34-E-3180	(topic TBD: based on feedback)	
20th November	34-E-3180	(topic TBD: based on feedback)	

**source:** <https://github.com/adamkewley/rse-meetups/>

# Sharing, Publication, and Collaboration

- What does this traditionally entail in academic research?
- What's changed over the last ~10 years?

# Traditional Sharing, Publication, and Collaboration

<b>Presenting at conferences</b>	<b>Team meetings</b>	<b>Collaborators</b>
<b>Publish a paper</b>	<b>Publish a book chapter that explains your research area</b>	<b>(... insert other ideas here...)</b>

# What's changed/changing over the last 10 years or so?

<p>Presenting work at conferences, <b>in addition to other media, such as YouTube</b></p>	<p>Team meetings, <b>and platform-based collaboration (Slack, Trello, GitHub, etc.)</b></p>	<p><b>Anonymous/spontaneous Collaborators on Open Projects with Open Data</b></p>
<p><b>Pre-print a paper so you can get it out there faster and get feedback on it before it goes to a journal</b></p>	<p><b>Publish a book chapter, or a free e-book/website, that explains your research area</b></p>	<p><b>Open Science</b></p>

# So, we'd like a workflow that...

- Enables rapid iteration, with some safety guarantees
- Can publish regularly, with the option for feedback
- Is compatible with modern media (websites, ebooks, lab notebooks, etc.)
- Makes teamwork and collaboration more transparent
- Is compatible with Open Science (important, come publication time)

# Git

- It's a version-control system that enables “fearless programming” by letting you always rewind (**rapid iteration**)
- Typically, used as a central part of day-to-day work, in conjunction with public online platforms, like GitHub (**publish regularly, get feedback**)
- Those platforms usually provide conventions/automation for (e.g.) hosting a website from your repository's content, or running your repository on a remote Jupyter instance, etc. (**websites, ebooks, lab notebooks, etc.**)
- And the platforms usually have built-in support for code reviews, or seeing changes visually (**makes teamwork/collaboration easier**)
- DOled archives, like Zenodo, have built-in support for publishing repositories. The Open Science guidelines for your projects probably require that you publish source code. Publishing a repository to a public platform is usually enough (**Open Science**)



# Interactive/teamwork: lets “publish” something

## 1) Publish *something*

- Get your source code into `git`
- Upload a snapshot of your source code to GitHub

## 2) Understand what's going on

- Get a feel for git by playing around with committing, snapshotting, etc.
- Play around with the GitHub UI, get an idea of what it's doing for you

## 3) Iterate

- Make changes to your code and then use git/GitHub/VSCoDe UI(s)
- Make a basic README.md file
- Keep iterating on the above – don't try and do it perfect the first time

### Resources (google)

python-mini-projects (e.g.)

Atlassian Git Cheatsheet  
visual studio code git  
GitHub Introduction  
cookiecutter-data-science

opensim-creator ;)  
nwo open science (DMP)  
zenodo.4629635 (TUD)  
zenodo.2842910

# Interactive/teamwork: lets “publish” something (alternate: if you already know git)

## 1) Ensure your code is in a “standard” layout

- Has README.md, LICENSE, and .gitignore files
- Has a top-level package (adam/\_\_init\_\_.py) or module (adam.py)
- Has a requirements.txt + setup.py files (older) or pyproject.toml file (newer)

## 2) Ensure your code runs in a fresh environment

- Use venv/conda to create a new python environment
- Enable it, then use your above local files to `pip install` your code into the environment
- Open a `python` terminal in the and ensure that you can run your code (e.g. `import adam`)

## 3) Push to PyPi

- Make a “test PyPi” account and publish your project to “test PyPi”
- As above, create a new python environment, but this time use `pip` to install your code from “test PyPi”, rather than from local files
- If you’re happy with it, repeat these steps with the real PyPi
- **Advanced:** get someone else in the room to install your package and run it

### Google

Python .gitignore example  
Python package vs module

\_\_init\_\_.py

\_\_main\_\_.py

What is pyproject.toml

pyproject.toml vs setup.py

python venv

Manage Anaconda Environments

(advanced) Windows WSL

“A practical guide to setuptools  
and pyproject.toml”

“Make python source distribution”

Python wheels vs eggs

# Repo Checklist

- ✓ **The code is:**

- ✓ In Git
- ✓ Online
- ✓ If you plan on publishing a library, then it's a "Standard" format (see other python projects, or ask)
- ✓ If you plan on publishing standalone scripts, or Jupyter notebooks, then it's in an "obvious" format
  - ✓ (e.g. a folder called 'Figure1' containing a Jupyter notebook and some files named `input\_data.csv` etc. - simple is fine too)


- ✓ **The README.md file:**

- ✓ Contains a single sentence summary of the code
- ✓ Contains a "How to Cite" section, if it is associated with a relevant publication (additionally, CITATION.cff/codemeta.json)
- ✓ Explains how other people can set up their environment to run your code, **including** how to acquire any necessary data/calibration files etc. if they are not provided within the repository.
- ✓ Contains an example of how to run the code **or** the repository contains example scripts (in an obvious location, like `examples/` or similar)
- ✓ Mentions any grant funders

- ✓ **Bonus:**

- ✓ Example code or certain commands/library functions can reproduce the exact result that is published
- ✓ (if it's a library) the code can be `pip install`ed, to aid ease-of-use by other researchers
- ✓ The code itself is easy to read and contains explanatory comments where appropriate

# Next Time (up for debate)

 **Note:** This list is just to give you an idea (it's very very provisional).

Topic	Description
Tools	IDEs, text editors, REPL, command line, basic git usage, LLMs (ChatGPT/GH Copilot)
Sharing, Publication, and Collaboration	Shared drives, GitHub, GitLab, writing a README, managing dependencies, releasing, publishing to package managers like <code>pip</code> , etc.
Software Design	Library vs. command-line vs. UI. User-/developer-experience. What is an application? The basics of how applications work.
Software Implementation	Iterative development methods. The REPL. Incremental application design. Assertions. Tests. How to implement things done more quickly - and with fewer surprises.
Libraries	Command-line parsers, configuration parsers, plotters, renderers
Advanced/Specialized topics (later)	Languages with type systems. Native application development (C/C++/Fortran /Rust). Multithreading. GP-GPU. Julia, hardware engineering, etc.