

Pre-class assignment #6

PHY-905-005
Computational Astrophysics and Astrostatistics
Spring 2023

This assignment is due the evening of Monday January 30, 2023. Turn in all materials via the GitHub Classroom.

The overall purpose of this pre-class assignment is to help you think through the development of open source scientific software – in particular, the techniques and tools that are considered to be “best practices” by researchers who develop software that is likely to be used for a long time, and possibly by other researchers. While you may not use all of these tools and practices right now, it’s important for you to know about them, and to have some familiarity with their use for when it becomes relevant!

Note that there are quite a few readings in this pre-class assignment. Most are quite short, but for the longer ones I want you to read quickly and try to take away the key points. And, as you’re reading, make notes on those key points and also about how you might use the things you’re reading about in your own work!

Readings about software development and workflows:

1. Look at the E-CAM software library’s [Scientific Software Best Practices](#) – in particular, the “General Programming Guidelines” section. **Read this carefully!**
2. Revisit the Git tutorial (which you looked at prior to the first day of class): <http://rogerdudler.github.io/git-guide/>
3. [The Definitive Guide to Forks and Branches in Git](#) (blog post)
4. [GitHub documentation on Pull Requests](#) – make sure to read the “Getting Started,” “Create & Delete Branches,” “About Pull Requests,” “Creating a pull request,” “Creating a PR from a fork,” “About merge conflicts,” and “Resolve merge conflicts” sub-pages. Browse the rest if you have time.
5. [GitHub documentation on issues](#) – make sure to read the “About issues,” “Create an Issue,” “Link PR to issue,” and “Assign issues & PRs” sub-pages. Browse the rest if you have time.

Reading about software testing:

1. [Types of software testing](#).
2. [Wikipedia article on continuous integration](#) (as a software development process)
3. [GitHub Actions documentation](#)
4. [PyTest documentation](#)
5. [unittest](#) Python unit testing framework (reference – look at this in comparison to PyTest)

Reading about software style, linting, and static analysis:

1. [Python PEP 8 Style Guide](#) (reference - only skim this to get the main point!)
2. [Pycodestyle documentation](#)

3. [Wikipedia article on software linting](#)
4. [Wikipedia article on static program analysis](#)
5. [Pylint documentation](#)

Your assignment:

1. For each of the sets of readings, summarize your key takeaways and remaining questions in `ANSWERS.md`. What points that were made resonate with you? What tools seem like they might be particularly useful?
2. Make sure that you can do the following things with git and GitHub, by working through the Git tutorial and GitHub documents linked to above. In `ANSWERS.md`, make notes about what went well and what you struggled with.
 - (a) Create an empty repository on GitHub.
 - (b) Clone it to your local computer.
 - (c) Add some code to it (from a previous pre- or in-class assignment), commit the new code, and push the changes to GitHub.
 - (d) Make a local branch (on your local computer), make changes to the code in that branch, and push those changes to GitHub.
 - (e) Merge the changes from your branch into your master branch and delete the old branch. Push those changes to GitHub.
3. Consider the code that you used for the in-class assignment on debugging. What kind of testing might you implement to make sure that the bugs you found don't occur in that code again? How might you go about using either PyTest or unittest? How might you create an integration test for this code?
4. Find a relatively complicated piece of Python code that you've written in the past and run both `Pycodestyle` and `Pylint` on it. If you are using the Anaconda python distribution these should already be installed; if not, you will have to install them at the command line with “`pip install pycodestyle`” and “`pip install pylint`”, respectively. Note that you cannot always run these on Jupyter Notebooks – you should try running it on standalone Python code. Make notes in `ANSWERS.md` about the similarities and differences between the two tools.
5. For the piece of code described above, what do you have to do in order to get rid of all of the errors produced by `Pylint`? Does it seem to produce better code? (And what does “better” mean to you?)
6. Finally, **pick one** of the following astrophysics community codes, look at its GitHub project page, and try to identify which of the best practices described in the readings are (or aren't!) implemented. Make sure to look at the Issues, Pull Requests, and Actions tabs for the one you've chosen and see what's going on there. Also look at the documentation and see what seems particularly useful to a new user! Make some notes in `ANSWERS.md` about your findings.
 - (a) [YT GitHub](#) ([Project web page](#); [documentation](#)). YT is a widely-used data analysis and visualization tool.
 - (b) [Enzo GitHub](#) ([Project web page](#); [documentation](#)). Enzo is a multiphysics self-gravitating fluid dynamics code that is primarily used for cosmological structure formation and star formation.
 - (c) [Tardis GitHub](#). Tardis is a radiative transfer spectral synthesis code used to model supernova ejecta. ([Project web page](#); [documentation](#))
 - (d) [Parthenon GitHub](#) ([documentation](#)). Parthenon is a GPU-optimized adaptive mesh framework based off of the Athena++ code, which serves as the basis for several very fast and scalable plasma astrophysics and numerical relativity codes.

Handing it in: Include your modified code and your answers to the questions (in the file `ANSWERS.md`) in your assignment.