Automation part 2

Web link

From the project description

Generation of results should be automated - tables, calculations, figures in your report should be generated within a single command/script (as much as possible).

What this means

- Inputs and outputs are fixed by your program.
  - GUIs are superfluous
  - Do not query for user input through the command line (slow, possibly not reproducible)
  - *program outputs should be traceable to their inputs*
- One *shell script* should carry out a series of instructions that runs various programs to produce the output from your input. (Instructions for compilation of C code can be included separately.)
- The outputs should correspond to figures/tables/numbers for scenarios you write about in your report.

How can the program be run on another machine?

- *Declare all dependencies (packages, programs, and their version numbers).*
  - there are tools to facilitate this called virtual environments, Docker, etc.
  - virtual environments started with Python and conventionally did not handle *gcc* (but possible now)
  - MATLAB does not have virtual environments but may require special *toolboxes*
  - it is only necessary to list dependencies in a file for the project submission

- Do not use absolute path names – use relative path names.

- **Do not submit your compiled executable file.**
  - *specificity*: compiled files are specific to the type of machine you compiled on
  - *safety*: executable files could carry out harmful instructions

Project directory structure

- There are many project templates
- Depends on project needs – there is no “single” template to follow
- General principles – separate different elements into different directories
  - raw data
  - processed data
  - results
  - documentation
  - code
- README or README.md in the *root directory* to describe directory contents

Example – not necessary to use this template

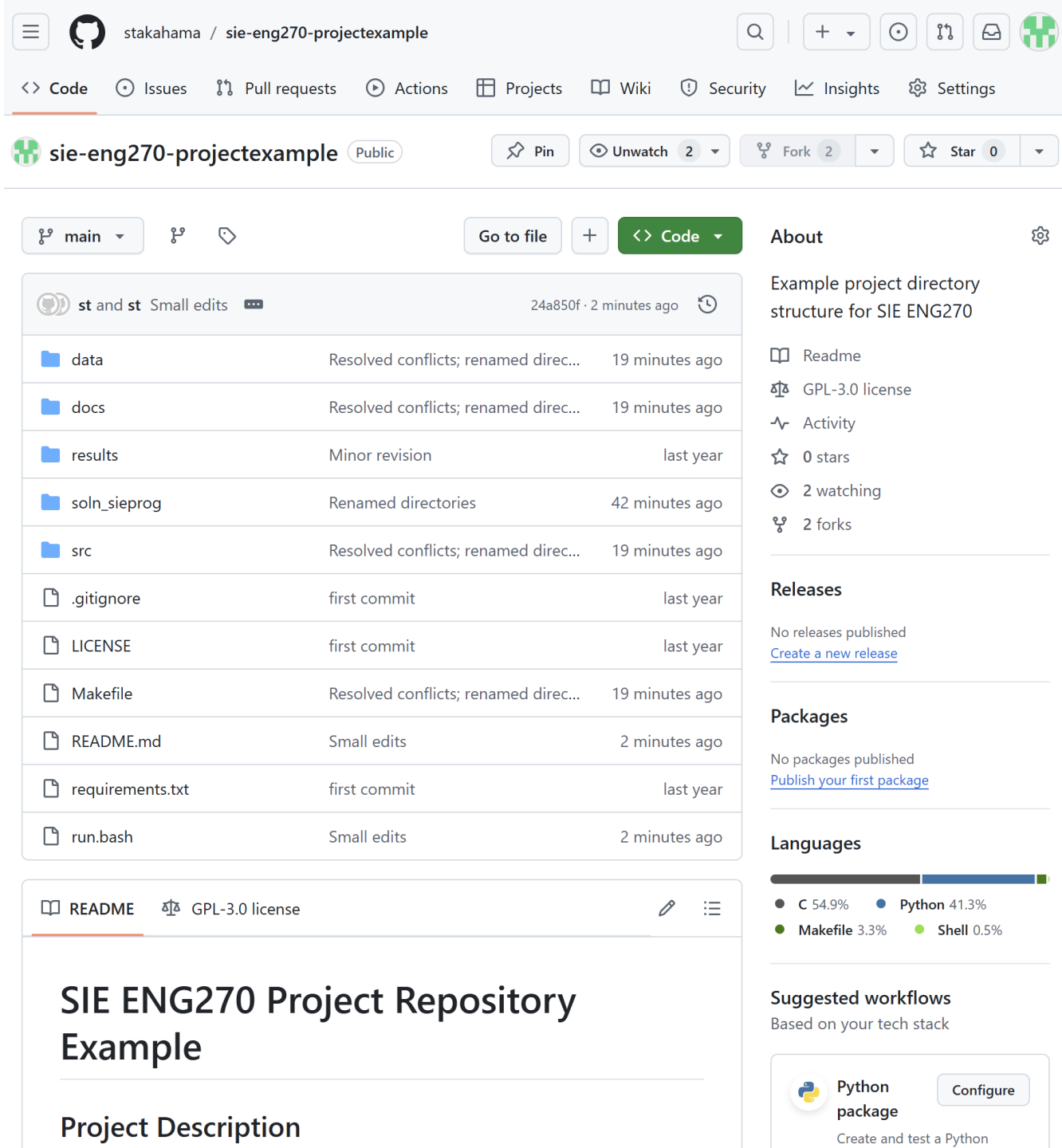
📦 Repository
- 📁 .github : contain the github settings
  - 📁 ISSUE_TEMPLATE : contains issues templates
    - 📜 *.yaml
  - 📁 workflows : contains CICD processes
    - 📜 code_quality.yml : Ruff + Black + mypy
    - 📜 tests.yml : pytest + CodeCov
- 📁 docs: contains the documentation.
- 📁 project_name: contains the project code.
  - 📜 *.py
  - 📁 notebooks
    - 📜 *.ipynb
- 📁 test: contains the project tests.
  - 📜 test_*.py
- 📜 _config.yml: config file for the github pages documentation.
- 📜 .gitignore: lists the files/folders to ignore for git.
- 📜 pre-commit-config.yaml: configuration file for pre-commit.
- 📜 CITATION.cff: citation information.
- 📜 CODE_OF_CONDUCT.md: code of conduct.
- 📜 CONTRIBUTING.md: contributing guidelines.
- 📜 Dockerfile
- 📜 LICENSE: license file.
- 📜 pyproject.toml: project configuration file.
- 📜 README.md: markdown file containing the project's readme.
- 📜 readthedocs.yml: Settings for readthedocs.

python ENAC-IT4R template

Example repository for class project

https://github.com/stakahama/sie-eng270-projectexample

- Recommended structure for this project.
- Get practice in using multiple folders (even if ENG270 project is small).
- Not necessary to follow this structure exactly if there are compelling reasons to deviate from it.



Absolute vs. relative paths

- *No*: “C:/Users/username/Desktop/myproject/data/” or “/Users/username/Desktop/myproject/data/”
- *Yes*: “./data” or “../data/” relative to code location or project root
- There are ways to define project root directory (e.g., in Python) but they can be overly complicated for small projects.
- You can either use ‘..’ syntax or include the relative path in a configuration file in the code directory (a text file with some repeatedly-used settings, often defined as a key-value pair) in the same directory.

Organizing your code into libraries

Canonical approach for packaging code into thematically cohesive units where parts can be reused in other projects:

- modules in Python
- libraries in C
- toolboxes in MATLAB

These modules must then be *installed* by the user to run your programs. These libraries can then be imported for any other program you write on your computer.

For smaller projects and the purposes of the project submission, *the approaches above* ***are not recommended*** as they require separate installation of the modules/ libraries/ toolboxes to reproduce results.

Simpler, recommended approach – create local libraries*:*

- Python: create empty __init__.py file in your code directory and you can import other .py files as modules locally (without installation)
- C: use header files and make files
- MATLAB: put all your functions in a subfolder and use addpath

The local libraries above are available when running code from your project directory, but not elsewhere.

Appropriate when modularizing code that is specific to the project. *(****Recommended for your project.****)*

Modules/libraries

- Breaking your code up into smaller collections of related functions and variables increases clarity when code base becomes larger.

- Modules can potentially be used in other programs.

Shell scripting

- Create and delete files/directories.
- Move files/directories around.
- What you do with the Finder (macOS) or Explorer (Windows), you can do with a series of shell commands in bash or Python. Include these shell commands in a script and automate the process.
- Your code should process the input and place the results in a separate directory without you having to manually move it (or having to ask the user to manually move it).

Text processing

- Generate file names and directories for reading/writing
- Parse information from file names
- Automate labeling
- Clean data

Common text processing operations

- Concatenate
- Format
- Split
- Strip (whitespace)
- Replace
- Substring

- *Regular expressions* expand capability for pattern matching and substitution/replacement