Scientific Computing Good Practices

A good-enough approach to scientific computing practices!

Vid (@svaksha)

2016-06-02,Thu

Agenda

Software Carpentry

- About Software Carpentry

I. Data Management

1. Save data in the rawest form available.
2. Store the data you would re-use.
3. Create analysis-friendly data.
4. Create datasets for specific analyses and figures.

SOFTWARE

- Rules for modular code.

III. Collaboration

IV. Project Organization

V. Version Control

When is Version Control Not Necessary?

- When is Version Control Not Necessary

Software Carpentry

I. Data Management

II. Software

III. Collaboration

IV. Project Organization

Software Carpentry

About Software Carpentry

I. Data Management

1. Save data in the rawest form available.

2. Store the data you would re-use.

3. Create analysis-friendly data.

4. Create datasets for specific analyses and figures.

SOFTWARE

Rules for modular code.

III. Collaboration

IV. Project Organization

V. Version Control

When is Version Control Not Necessary?

When is Version Control Not Necessary

Software Carpentry

About Software Carpentry

- 1998, Software Carpentry (Data Carpentry), members of NumFOCUS, 501(c)3 US non-profit Org.
- Teach computing skills to researchers.
- Field : science, engineering, medicine, and related disciplines.
- Volunteer instructors (Run hundreds of events for thousands of scientists in 2.5 years).
- Lessons are freely reusable under the Creative Commons (CC) - Attribution license.
- Credits: 2016, Wilson G. et al, https://github.com/swcarpentry/good-enough-practices-in-scientific-computing

I. Data Management

Project data may need to exist in various forms, ranging from raw to highly processed. We recommend some guiding principles to manage it.

Never lose data.

Data should be findable, accessible, interoperable and reusable. People, including yourself, can use it without pestering you with questions.

Data should be comprehensible ("human readable") collaborators can easily understand what the data contains.

Data should be machine readable. i.e., programs should be able to load data correctly without extra programming effort.

I. Data Management

1. Save data in the rawest form available.

- Save the data file produced by an instrument or raw results from a survey.
- Keep the raw data! helps recover from analytical mishaps and go back in time to experiment
- Don't overwrite raw data files with cleaned-up versions.

I. Data Management

2. Store the data you would re-use.

- Data files are the starting point for downstream analyses.
- Maximize machine readability : Microsoft Excel to CSV
- Maximize human readability - use self-explaining variable names. ex. name1, name2 (Versus) first-name, family-name
- Store metadata in the filename to aid pattern matching. Ex. '2016-05-eu-uk-london.csv', easy to read. A script can parse by year (2016-*.csv), month or by location.
- ETL = Extract, Transform, Load! Say NO to the Transform phase. Dont massage your data : No data tweaking / adding external information.

I. Data Management

3. Create analysis-friendly data.

- Columns with more than one variable's worth of information should be split.
- Ex. "kg" in "5.5 kg" is parsed as character data, not numeric. Two columns : mass "5.5" and units "kg".
- Multiple columns with single variable's worth of information when taken together should be combined.
- Ex. one row per field site and then columns for measurements made at each of several time points.
- Reformat values as needed to match your environment's built-in parsing rules.
- Ex. use date-time format that will be recognized automatically. ISO format, etc..

I. Data Management

4. Create datasets for specific analyses and figures.

- Filter rows, select relevant variables, discard redundant fields and merge with external information.
- Data reduction, amalgamation of multiple datasets == store relevant data for analysis.
- Above != data massaging.
- Dont manipulate the raw data to fit your research analysis or statistical hypotheses.

II. Software

- Tens of thousands of lines of code == software engineering.
- Scientists and researchers == few scripts for automation, merging data, or statistical analysis and visualization.
- Code that is readable, reusable, and testable are not disjunct things - a.k.a. modular code !
- Reusable code == Adopting a few key practices.
- Modular code == Key to productivity and reproducibility.
- Create programs out of short, single-purpose functions with clearly-defined inputs and outputs. Libs!

II. Software

Rules for modular code.

- 1. Every analysis step should be represented textually (complete with parameter values).
- 2. Brief explanatory comment at the start of every program. Ex. open-close quotation marks (""").
- 3. Construct functions into programs that are not too long. Avoid global variables (constants are OK) and not have more than half a dozen parameters.
- 4. Eliminate duplication : re-use functions and use data structures (arrays, lists).
- 5. Give functions and variables meaningful names documenting its purpose. No alpha chars (i,j).
- 6. Make dependencies and requirements explicit : Ex. Python has 'requirements.txt'
- 7. Use conditional loops (if/else statements) and NOT (un)comments to control a program's behavior.
- 8. Provide a simple example or test data set for users to test-run your code.

III. Collaboration

- Keep it Simple. Make it easy for people to collaborate and to give you credit.
- Lower entry barriers with 'low-hanging fruits' tasks, and setting up a local workspace to start work.
- Clarity: Document the 'how2contribute' process to increase collaborator potential.
- Document your project in a README file, installation, describe features that work and the code structure.
- Create a Shared To-Do List in the open repository.
- Make the License Explicit: CC licenses for data and text and Libre copyleft licenses for code.
- Keep a CITATION file with info on how to cite your project, including datasets, code, and other DOI artifacts.

IV. Project Organization

- Each project should be in its own directory, properly named with subdirectories:
- 'doc' contains the project text documents, installation, electronic lab notebook recordings, etc..
- 'data' contains raw data and metadata, organized into subdirectories. Cleaned or modified data files are treated like results.
- 'src' has the source code for scripts and programs, say R or Python.
- 'bin' contains executable scripts and programs brought in or compiled from code in the src directory.
- 'results' for all files that are generated as part of the project. intermediate results, Ex. cleaned data sets or simulated data, and final results, figures and tables.

V. Version Control

- Tracking changes that collaborators make to data, software, and manuscripts is a critical part of research.
- Git, Mercurial, Bazaar have similar functionality and are distributed VCS.
- DVCS aids reproducibility and allows us to reference historical data and code commits.
- Improves fixability: can retrieve a specific version at a given system timestamps or revert changes.
- DVCS supports sharing and collaboration, tracks changes that would overwrite others work.

V. Version Control

When is Version Control Not Necessary

- File size of data sets can be a problem: Git cannot handle files larger than 2GB.
- Raw data should not change, hence does not require version tracking
- Synthesized or modified datasets can be re-generated, hence dont need a DVCS. But the data-cleaning scripts must definitely be under version control.
- Data format: Binary files (HDF5) can be in a DVCS but 'diffs' you won't be viewable. Reordering the rows or columns of tabular (CSV) data will create a big change in the DVCS.
- Privacy: Medical /patient data) should not be publicly stored.
- The results directory and other generated files such as figures should not be placed under version control. Analytical reproducibility requires results for data analysis without needing to regenerate it all.

Conclusion

- Interlinked core ideas - simple and effective techniques for Data Management.
- Software good practices.
- Learned how to collaborate and Organize projects.
- Purpose of DVCS and what not to version control.
- Thank You !!
- Credits: Adapted from the paper by Dr. Greg Wilson, Ex-Prof UoT and Founder 'Software Carpentry Foundation'.
- Questions 'n Answers !