Software Management

Computational Methods, Dec. 2018, Kenji Doya

In addition to writing a program for each computing task, knowldge and skills are needed for designing and managing the entire data analysis or simulation procedure, testing and revising the codes, and sharing the data and tools among collaborators.

Unlike commercial software development, research computing often start from simple exploratory codes created by a single researcher. However, even for a single-person project, it is beneficial to follow the standard practices in software development because.

- If your project is successful, it will be succeeded by other members of your lab or the research community world wide.
- You, after a few months, do not remember where you put which file, what was this file, or why you wrote this code.

Reference:

- Greg Wilson, et al. (2017). Good enough practices in scientific computing. PLOS Computational Biology, 13(6): e1005510 (https://doi.org/10.1371/journal.pcbi.1005510 (https://doi.org/10.1371/journal.pcbi.1005510))

Data Management

Most research start with obtaining *raw* data, continues on with a series of pre-processing, visualization and analyses, and complete with paper writing. Handling all different files without confusion and corruption takes some good thoughts and habits.

- Keep the raw data and *metadata* and take back up.
- Store data as you wish to see when receiving.
- Record all the steps of processing, better with a script.

For a small scale data, *DropBox* is an easy solution for sharing and backup.

At OIST, for storing large scale data, you can use the *bucket* drive. See: https://groups.oist.jp/it/research-storage (https://groups.oist.jp/it/research-storage)

In [ ]:
```

```

Software Management

In writing programs, keep in mind:

- Make them *modular* and aviod duplicate codes.
- Give explanation at the beginning of each file/function.
- Use file/function/variable names that you can comprehend a year later.
- Never write a numeric parameter in an equation; define it as a variable/argument.
- Give comments for significant parts/lines of codes.
- Turn comment/uncomment into `if-else` for different modes of operation.
- Verify your code with a simple input for which the correct output is known.
- Prepare documentation even before somebody asks you, as you yourself will need that after a few months.

In some projects, all you need is to download pre-existing tools and apply them to the data. Even in that case, it is better to record the procedure as a *script* for

- avioding/detecting manual errors
- reproducibility of the result
- re-analysis with new data, tool, or parameters

Scritping

On Unix-like systems, the common way is a *shell script*, which is a file containing a series of commands you would type into a terminal.

For a more elaborate processing, a Python script is often preferred.

```
In [ ]:
```

Graphic User Interface

When running your own programs, working interactively within Jupyter Notebook or running scripts for systematic experiments is fine. When you let your collaborators run your program, especially those without computing background, it is often nice to create a graphic user interface (GUI).

There are several Python packages for building GUIs, such as `Tkinter`, `PyQt` and `kivy`.

Tkinter

`Tkinter` is a Python interface for a GUI library `Tcl/Tk`: https://www.tcl.tk (https://www.tcl.tk)
See the following for details:
https://wiki.python.org/moin/TkInter (https://wiki.python.org/moin/TkInter)
http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html (http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html)

You create a window and place menus, listboxes, buttons, canvas, etc. called *widgets*.

You define functions that are called when a button is pressed, etc.

Finally you run an *event loop* to get user's actions and call corresponding functions.

To use `matplotlib` with `Tkinter`, you can check the samples like
http://matplotlib.org/examples/user_interfaces/embedding_in_tk.html (http://matplotlib.org/examples/user_interfaces/embedding_in_tk.html)

```
In [ ]:
```

Version Control System

Software development is repetitions of coding, testing, and improving. A version control system (VCS) allows

- parallel development of parts and re-integration
- trace back to previous versions when a problem is detected

Git and GitHub

The most popular VCS today is *Git*, created by Linus Torvalds for developing Linux. There is a cloud service *GitHub*, which is free for open software.

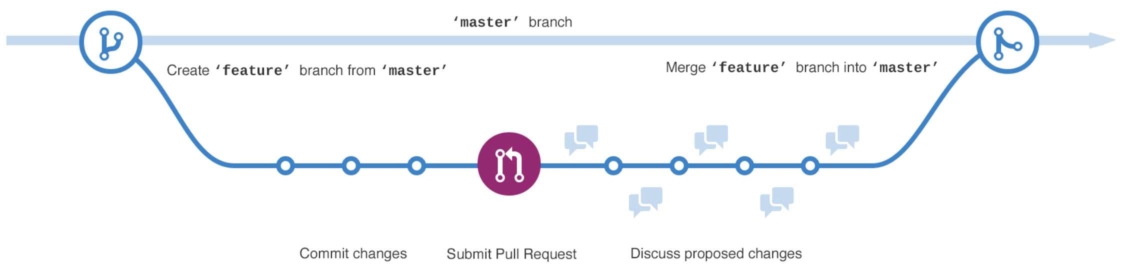
OIST subscribes to its commercial option, which can be used for proprietary software development: https://github.com/oist (https://github.com/oist)

For account setup, wee https://groups.oist.jp/it/github-oist (https://groups.oist.jp/it/github-oist)

This is a preferred way of sharing programs, or in some cases text data and manuscripts, among collaborators. It is also helpful for a single-person project, for succession by a future member of your lab, for open access after publication, or for yourself after some time.

These are typical steps in contributing to a project in GitHub.

- Join as a member of a repository.
- Copy the existing files and see how they work.
- Make a new *branch* and add or modify the codes.
- After tesing locally, *commit* the new version.
- Open a *pull request* for other members to test your revision.
- Your pull request is merged into the *master* branch.


'master' branch
Create 'feature' branch from 'master'
Merge 'feature' branch into 'master'
Commit changes
Submit Pull Request
Discuss proposed changes

See "Hello World" in GitHub Guide for details (https://guides.github.com (https://guides.github.com)).

```
In [ ]:
```

Software/Data Licenses

Today, increasingly more journals and agencies request that you make the data and programs publicly accessible for

- reproducibility of research results
- enable meta-analysis
- facilitate reuse of data and programs

You should set an appropriate condition in making your data or program public, to facillitate their use and to keep your (and your organization's) intellectural property. Points of consideration in making your data/programs public include:

- copyright
- acknowledgement
- revision
- re-distribution
- commercial use

It is also important to know the licenses of the software you use for your development, as that can limit the way you can use/distribute your programs.

Creative Commons

Creative Commons (https://creativecommons.org (https://creativecommons.org)) is an emerging standard using combination of three aspects:

- Attribution (BY): request aknowldgement, e.g., citing a paper
- NonCommercial (NC): no commercial use
- ShareAlike (SA) or NoDerivs (ND): allow modification and re-distribution or not

See https://creativecommons.org/licenses/?lang=en (https://creativecommons.org/licenses/?lang=en) for typical combinations.

GPL, BSD, MIT, Apache, etc.

In open software community, several types of licensing have been commonly used:

- Gnu General Public Licence (GPL): redistribution requires access to source codes in the same license. Called *copy left*.
- BSD and MIT license: do not require source code access or succession of the same license.
- Apache License: does not even require the license terms.
- Public Domain (CC0): no copyright insisted. Free to use/modify/distribute.

See https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses (https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses) for further details.

```
In [ ]:
```

Exercise

1. GitHub

Let's try team software development using OIST GitHub: https://github.com/oist (https://github.com/oist)

If you cannot connect there, you are not registered as an OIST member on GitHub. Follow the guide here to be registered:
https://groups.oist.jp/it/github-oist (https://groups.oist.jp/it/github-oist)

- Create your GitHub account if you haven't
- Send your GitHub ID to it-help@oist.jp to request registration

DynaView

Goto https://github.com/oist/ComputationalMethods (https://github.com/oist/ComputationalMethods) and download the entire repository. The folder `DynaView2018` includs the Python codes for this exercise. There are three (or more) files:

- DynaView.py: the main python script
- first.py, second.py: python modules for dynamical systems

Run the script in a console, by

```
$ python DynaView.py
```

This should open a window with a list of system names, three buttons, and a plot area. You can play with the buttons:

- Run: run the system currently selected and plot the trajectory.
- Reset: reset the state to the default initial state.
- Select: after selecting a system in the list, press this button to confirm a selection.

After figuring out how the program works, try the following.

1) Add a dynamical system of your interest to DynaView.

- make a copy of `first.py` or `second.py` and modify the `dynamics()` function. Also change the name, default initial state, and parameters. Don't forget to include your name in the header so that we can see who made it.
- add lines in `DynaView.py` so that your module is imported and registerd in `system_list` (please position in alphabetic order).
- test on your computer that your system appears in the listbox and runs as expected.
- make a new *branch* to register your changes.
- *commit*, i.e., upload your new and modified files.
- send a *pull request* so that other people can test your changes.

Follow the tutorial here:
https://guides.github.com/activities/hello-world/ (https://guides.github.com/activities/hello-world/)

In [ ]:
```

```

Optional) In addition to adding a new module, you are welcome to improve the main program `DynaView.py` itself. For example,

- add text boxes to see/change states and parameters. (rows 1 and 2 are inteded for such use).
- add `quit` button for clean finish.
- enable phase space trajectory plot (with a checkbox/menu to select the plot mode, or in a second canvas).
- make the window look cooler by adjusting the widget locations.
- fix any bugs or improve error handling.
- add documentation.
- ...

In [ ]:
```

```