**A Quick Guide to Organizing Computational Biology Projects**

It is good to have your basics down with regards to organizing your project before you pick up to many bad habits.

Keep your core principles simple.

People who aren’t familiar with your work should be able to easily pick up where you left off. And you should also be able to come back to your older projects and be able to quickly realize what you were meaning to do. You will probably have to come back to your own projects to tweak certain aspects many times and it is important to have good documentation of what you’ve done and why.

Organize your project in coherent structures of hierarchical directories. Use a few common directory names to organize your projects in a standardized way that you familiarize yourself with. Make use of both logical and chronologically organized levels in your hierarchy of directories. It can be particularly good to organize your data and results directories in a chronological order. When generating many files of a particular kind, that usually warrants the creation of a new folder for those files.

Use a document in your root folder to keep track of your overall progress within the project. Here it is good to write details of what you’ve done as well as observations, conclusions and ideas. One can also record conversations and e-mails in this document to be able to look back and go through reasoning done in collaboration with other researchers. This document can also be linked online for others to get up to speed on the project. One should also think about security and perhaps put the document behind a password-wall of some kind.

Record every operation that you perform and make those operations as reproducible as possible. Have a README file and/or a “driver script” that carries out the entire experiment automatically.

It is good to comment generously. It is generally bad editing files manually in intermediate steps because then the project isn’t fully automated anymore.

Store all file and directory names in one script to make it easier to keep track of your output.

Use relative pathnames so that the project is more easily reproduced for other people.

Make the script restartable so that you can rerun specific parts of it without having to rerun all parts.

It may be useful to have an additional script besides your “runall”-script to be able to summarize incomplete experiments.

Make use of standard libraries for all viable applications and have your experiment abort when errors occur.

Have your program generate temporary names for output files and rename the files when they are complete. This makes it easier to restart your program and not confuse partial results for complete.

Use a hierarchy of scripts if needed, with one “master” or “driver” script and downstream have single-use, project specific and multi-project scripts.

Version control software can be invaluable but requires the user to use it regularly. Files that need version control are first and foremost hand-edited files and version control has little application for automatically generated files.

**Ten Simple Rules for Reproducible Computational Research**

To be able to accommodate the need for reproducible research there is a call for well-organized and comprehensively disclosed bioinformatical procedures in published research.

Reproducible research also aids the individual researcher.

While there is an ever-present pressure to meet deadlines, there are a lot of time-saving and productive effects of organizing a project in a fully-automated and more easily reproduced fashion.

It is important to keep track of how every result was produced.

Manually manipulating data sets should be avoided and when done should be carefully noted.

If external programs are used it is important to archive the versions used within the project to be able to reproduce the results. And at a minimum note names and versions of the programs used.

One should record intermediate results to more easily be able to identify bugs and/or faulty interpretations that may not be as easily noticeable in the final results.

Keep track of seeds used in random processes within the project.

Always store the raw data.

Generate hierarchical analysis output allows one to be able to inspect all the layers of increasingly detailed output.

It is helpful to have some form of description of the connection between interpreted results in text and the underlying computed result.

It is important to make all the input data and tools used to get the final results accessible to others.

**So you want to be a computational biologist?**

There is a need for computational skills in biology research. It is important to understand the underlying methods used in the computational analysis.

There needs to be a balance between perfectly written code and getting things done in a reasonable time-frame.

Version control software is useful and is helpful when publishing your research.

Automating your analysis and pipelining the steps can be useful but shouldn’t be done too early and needs to be weighed against speed and efficacy.

Training courses are good for learning but not the end-all be-all, you need to constantly teach yourself as well

Remember your biological knowledge when interpreting the analytical results and try setting “traps” for your pipeline to figure out if it really is trust-worthy.

The UNIX/Linus command line is more efficient then working in native windows.

A more popular programming language has larger “third-party” support in terms of already existing libraries and toolsets.

Interpreting results will still be difficult and demanding, the computational part helps us get there but it doesn’t necessarily tell us what conclusions we should draw from the results.

It is useful to discuss difficult computational problems online and let other people’s mistakes and corrections be of use to you, chances are great that someone else has run into a very similar problem to yours.

**Ten Simple Rules for the Open Development of Scientific Software**

Open source software has impacted scientific research in a big way.

You should publish the source code for your research so as to make it more transparent and more likely to be published. It facilitates others building upon your work.

Do some research before starting a project and ascertain what others have done before you so that you don’t have to reinvent the wheel yourself every time.

It is important to train yourself in the craft of software development. Get familiar with other people’s code.

It is important to think about the application of your software to other scientific situations.

Being transparent in your work can establish your name in that research field if you are able to provide useful software for other researchers.

Keeping your software simple is important for others to be able to reproduce your results and for others to be able to collaborate with you.

Trying to achieve perfection takes away from efficiency, it is a good idea to follow the mantra of “release early, release often”.

It is good to establish communication with others using your software and to acknowledge contributors. This helps guide you in how to further develop your software without

It can be beneficial to use an established platform to advertise your software project, like GitHub or Google code.

A software project that grows will eventually need some level of funding and outside help.

Scientific software development should aim to aid you in your research endeavors.

**Best Practices for Scientific Computing**

The importance of software in scientific research is ever growing and scientist have to develop their own software to tackle challenges in their work. Scientists are usually self-taught and lack the formal training in development practices of writing code, including having an easy-to-maintain code, version control etc.

Code often gets reused and errors in code can have compounding effects on the research community.

Code needs to be written in a way that facilitates us as humans using it.

It should be automated and pipelined in an efficient manner.

Changes to a software project should be incremental and with a version control system in place.

Code should be built in a modular fashion and parts re-used rather than reinvented in every step.

There needs to be a caution to mistakes and bugs which will inevitably appear.

Optimization comes after checking that the code is doing what it should.

Documentation should focus on design-features and the idea behind the code.

It is important to take advantage of a community of researchers writing code.