So you want to be a computational biologist?

Nick Loman & Mick Watson

Two computational biologists give advice when starting out on computational projects.

The term 'computational biologist' can encompass several roles, including data analyst, data curator, database developer, statistician, mathematical modeler, bioinformatician, software developer, ontologist—and many more. What's clear is that computers are now essential components of modern biological research, and scientists are being asked to adopt new skills in computational biology and master new terminology (**Box 1**). Whether you're a student, a professor or somewhere in between, if you increasingly find that computational analysis is important to your research, follow the advice below and start along the road towards becoming a computational biologist!

Box 1 Glossary of useful computing terms

Command line interface. A means of interacting with a computer whereby the user issues commands in the form of successive lines of text. The term 'shell', or 'UNIX shell', refers to a command line interpreter for the UNIX/Linux operating system. Microsoft provides a command line interface to Windows, but this is not commonly used in bioinformatics.

Compute cluster. A collection of computers that work together, often to run many jobs at once through a job scheduling and resource management system.

Pipeline. In computer jargon, this is a series of steps, or software tools, run in a specified order, where the input to one tool may be the output of a previous tool. Can include automated logical decisions.

Source code (code). Refers to computer instructions written in a particular programming language.

Software. We don't really need to define this, do we? For completeness, let's just say this is a set of instructions that instructs a computer to carry out certain operations. Can be an executable file that is 'compiled' from source code or a collection of source code that is interpreted.

Script. Source code written in an interpreted language, often used in bioinformatics to perform particular tasks, for example, running other software in a specified order, such as in a pipeline.

Source control and version control. A system by which changes in source code are tracked and managed, and under which multiple versions of source code can be maintained.

UNIX/Linux. UNIX is a stable, multiuser, multitasking system for servers, desktops and laptops, with both a graphical and command-line interface. UNIX comes in many different versions. Linux refers to a number of different UNIX-like operating systems that are developed under an open-source model.

Understand your goals and choose appropriate methods

Key to good computational biology is the selection and use of appropriate software. Before you can usefully interpret the output of a piece of software, you must understand what the software is doing. You wouldn't go into the laboratory and perform a polymerase chain reaction without a basic understanding of the method. Why would you do the same with a computational analysis? Understanding the underlying methods and algorithms gives you the tools to interpret the results. That doesn't mean you need to read through each line of source code, but you should have a grasp of the concepts.

Software tools are often implementations of a particular algorithm that may be well-suited for particular types of data; for example, in *de novo* assembly, an Overlap-Layout-Consensus assembler is optimized for longer sequence reads, whereas de Bruijn graphs were designed with short reads in mind. Choosing software employing the most appropriate algorithm will save you a lot of time.

Set traps for your own scripts and other people's

How do you know your script, software or pipeline is working? Computers will happily output results for the most bizarre of input data, and the absence of an error message is not an indication of success. Create tests, small datasets for

Mick Watson is at The Roslin Institute, University of Edinburgh, Edinburgh, UK, and is Head of Bioinformatics at Edinburgh Genomics, an academic genomics facility developing bioinformatics training in next-generation sequence analysis (http://genomics.ed.ac.uk). Follow him on Twitter, @BioMickWatson, and on his blog at http://biomickwatson.wordpress.com/. Nick Loman works as an independent research fellow in the Institute for Microbiology and Infection at the University of Birmingham, Birmingham, UK, sponsored by a Medical Research Council Special Training Fellowship in Biomedical Informatics. Follow him on Twitter, @pathogenomenick, and on his blog at http://pathogenomics.bham.ac.uk/blog.
e-mail: n.j.loman@bham.ac.uk or mick.watson@roslin.ed.ac.uk

Table 1 Essential tools for the biological software developer

Task	Tools
Collaborative software development	Share data and code through online collaborative working environments such as Github, Sourceforge and Bitbucket. Use Google to find tutorials on these systems, e.g., http://try.github.io/
Build powerful pipelines	There are modern software libraries, such as Ruffus, and more traditional tools, such as Make, to build pipelines from existing software tools. Your choice will depend on personal preference and on your favorite programming language.
Make your pipelines available	You may be comfortable on the command line, but your collaborators may not be. Therefore you can deliver your pipelines through graphical environments such as Galaxy (http://www.galaxyproject.org/) or Taverna (http://www.taverna.org.uk/).
Integrated development environment (IDE)	Whether you want to adopt a full IDE, such as Eclipse, or an advanced text editor, such as Emacs, you will need something to use to develop your code. Again, this will likely depend on your choice of language and personal preference. However, at some point, you'll have to use a command line–based editor, such as vim or nano, so it's advisable to learn at least the basics.

which the answer is known, and check that the software or pipeline can reproduce that answer. Try and do that for every 'type' of answer you expect to find. Double-check the results of everything, to see if those results make sense. Laboratory scientists wouldn't dream of running experiments without the necessary positive and negative controls, and these tests are the computational biology equivalent.

You're a scientist, not a programmer

The perfect is the enemy of the good. Remember you are a scientist and the quality of your research is what is important, not how pretty your source code looks. Perfectly written, extensively documented, elegant code that gets the answer wrong is not as useful as a basic script that gets it right. Having said that, once you're sure your core algorithm works, spend time making it elegant and documenting how to use it. Use your biological knowledge as much as possible—that's what makes you a computational biologist.

Use version control software

Versioning will help you track changes to your code, maintain multiple versions and to work collaboratively with others. Using a standard tool, such as Git or Subversion, you will also be able to publish your code easily. Be nice to your future self. A few well-placed README files explaining the choices you made and why you made them will be a boon in months or years when you return to a project. Document your code and scripts so that you understand what they do. When you come to publish your work, try publishing the scripts and methods you used to generate your results so that others can reproduce them. Also consider keeping a digital laboratory notebook to document your analyses as you perform them. Repositories, such as Github, are ideal for this and also help you maintain copies of the repository to serve as off-site backups (**Table 1**).

Pipelineitis is a nasty disease

A pipeline is a series of steps, or software tools, run in sequence according to a predefined plan. Pipelines are great for running exactly the same set of steps in a repetitive fashion, and for sharing protocols with others, but they force you into a rigid way of thinking and can decrease creativity.

> "Laboratory scientists wouldn't dream of running experiments without the necessary positive and negative controls... tests are the computational biology equivalent."

Warning: don't pipeline too early. Get a method working before you turn it into a pipeline. And even then, does it need to be a pipeline? Have you saved time? Is your pipeline really of use to others? If those steps are only ever going to be run by you, then a simple script will suffice and any attempts at pipelining will simply waste time. Similarly, if those steps will only ever be run once, just run them once, document the fact you did so and move on.

An Obama frame of mind

Yes you can! As a computational biologist, you will need to be creative, from tweaking existing methods to developing entirely new ones. Be adventurous, be prepared to fail, but keep going. It's amazing what you can achieve by using Google, by asking other people in the field and by teaching yourself how to solve particular problems.

Attending training courses (**Table 2**) can be useful, but these are only really the start of your learning, not the end. Continue by teaching yourself afterwards.

Be suspicious and trust nobody

The following experiment is often performed during statistics training. First, a large matrix of random numbers is created and each column is designated as 'case' or 'control'. A statistical test is then applied to each row to test for significant differences between the case data and the control data. You should not be surprised to learn that hundreds of rows come back with *P* values indicating statistical significance. Biological datasets, such as those generated by genomics experiments are just like this, large and full of noise. Your data analysis will produce both false positives and false negatives; and there may be systematic bias in the data, introduced either in the experiment or during the analysis.

Table 2 Useful resources for learning

Type of information	Relevant URLs
MOOCs (massive open online courses)	These are very popular at the moment and offer free training over the internet. Coursera (https://www.coursera.org/), Udacity (https://www.udacity.com/), edX (https://www.edx.org/) and the Kahn Academy (https://www.khanacademy.org/) have a range of courses relevant to bioinformatics, genomics, computing, statistics and modeling.
Learning to code	Codecademy (http://www.codecademy.com/) and Code School (https://www.codeschool.com/) are not specific to biology but do offer simple ways to learn how to code. For a more biological perspective, "Python for biologists" (http://pythonforbiologists.com/) is always popular. For examples of best practices visit http://software-carpentry.org/.
Bioinformatics problem solving	Learn bioinformatics through problem solving and pit your wits against others at http://www.rosalind.info.
Web forums	These are essential when you start out—ask questions and receive answers from experts at http://www.seqanswers.com/ and http://www.biostars.org/.
International organizations	GOBLET is the global organization for bioinformatics learning education and training (http://www.mygoblet.org/), and ELIXIR is a European organization set up to provide an infrastructure, including training, for life sciences information (http://www.elixir-europe.org/).
Blogs and lists	A variety of blogs and lists exist online that detail computational biology courses, such as http://stephenturner.us/p/edu and http://ged.msu.edu/angus/bioinformatics-courses.html.

There is a temptation, even among biologists trained in statistical techniques, to throw caution to the wind when particular software or pipelines produce an interesting result. Instead, treat results with great suspicion, and carry out further tests to determine whether the results can be explained by experimental error or bias. If multiple approaches agree, then your confidence in those answers increases. But for many findings, validation and further work in the laboratory may be necessary. Knowledge of biology is vital in the interpretation of computational results. Setting traps, or tests, as mentioned above, is only part of this. Those tests are meant to ensure that your software or pipeline is working as you expect it to work; it doesn't necessarily mean that the answers produced are correct.

The right tool for the job

Become comfortable working from the UNIX/Linux command line. The command line is incredibly powerful, allowing you greater control over what software is doing and allowing you to run and control multiple jobs at once. Most bioinformatics software is designed to be run from the command line. Learn about compute clusters and how to run hundreds of jobs in parallel. You'll need to be able to code, but the choice of language is not as critical as you may be led to believe by computer scientists. Each language has strengths and weaknesses, and you may have to use more than one to get the job done.

Bear in mind that choosing a more popular language will let you benefit from a larger library of existing toolsets, for example the Bio* projects from the Open Bioinformatics Foundation (http://www.open-bio.org/wiki/Main_Page). Microsoft Excel is a spreadsheet program, and unless used very carefully, is not suitable for biological data (http://www.biomedcentral.com/1471-2105/5/80/). Store your experimental data, in structured text files or in an SQL database. Employing basic database practice, such as normalization (i.e., ensuring a single place for each piece of data associated with your project), means there are fewer chances to make a mistake later. Make sure everything is backed up, regularly.

> "Knowledge of biology is vital in the interpretation of computational results."

Be a detective

As a computational biologist, a lot of your time will be spent analyzing and interpreting data. The data are telling you something. They contain a story and it's your job to find out what that story is. Unless you're very lucky, it probably won't be obvious. Finding out will not be easy. You will have to think about how the experiment was performed; how the analysis was performed; and what the results are telling you. You will need to confidently disregard, or control for, what you think are errors and systematic biases in the data.

To do the above you may need to talk to other scientists involved in the work, or integrate and analyze additional data. You may need to propose follow-up experiments, designed to test any hypotheses you generate. Remember, the real story may not be in your data at all! If the biological system you're interested in hinges on phosphorylation of a protein, then you probably won't see this effect in your RNA-seq data. You are basically a detective. Work the data. Figure it out.

Someone has already done this. Find them!

No matter how gnarly a problem or how cutting-edge a method, there is a pretty good chance someone out there has tried to tackle it already. Two excellent resources for discussing problems with software are BioStars (http://www.biostars.org/) and SEQanswers (http://seqanswers.com/). Twitter is another place where you will be able to find advice and links to resources and papers. Hook up with other computational biologists in your department or institute. There is likely to be a local computational biology meeting or interest group in your area, so find it and join up; if there isn't, why not start your own like Nick did!

In conclusion, there is a huge amount of support available online and through local user groups, if you want to practice computational biology. The most important starting point is to be brave enough to try and to learn from these resources. Install Linux on your PC, and start working through some learning materials online. You will be astonished what you will be able to achieve very quickly, and ultimately you will have a very rewarding experience!

ACKNOWLEDGEMENTS

The authors would like to thank members of the Twitter community who brought the learning resources listed in this article to our attention.