Manuscript Title

This manuscript (permalink) was automatically generated from lubianat/codigobonito@cfa3a53 on October 22, 2020.

Authors

- **John Doe**
  XXXX-XXXX-XXXX-XXXX · johndoe · johndoe
  Department of Something, University of Whatever · Funded by Grant XXXXXXXX

- **Jane Roe**
  XXXX-XXXX-XXXX-XXXX · janeroe
  Department of Something, University of Whatever; Department of Whatever, University of Something

Abstract

Reunião 24/09/2020

A Quick Guide to Organizing Computational Biology Projects

[1]

Highlights

Curriculum fails to address many of the day-to-day organizational challenges

Someone unfamiliar with your project should be able to look at your computer files and understand in detail what you did and why. Everything you do, you will probably have to do over again.

data directory for storing fixed data sets, a results directory for tracking computational experiments peformed on that data, a doc directory with one subdirectory per manuscript, and directories such as src for source code

In parallel with this chronological directory structure, I find it useful to maintain a chronologically organized lab notebook. In practice, I ask members of my research group to put their lab notebooks online, behind password protection if necessary.

driver script (I usually call this runall) that carries out the entire experiment automatically

Avoid editing intermediate files by hand. Doing so means that your script will only be semi-automatic Use relative pathnames to access other files within the same project

Even in a simple script, you should check for bogus parameters, invalid input, etc.

how much effort to put into software engineering ### Comments

It goes on to a lot of personal notes, which are okay, but far from statndard. GitHub logs work well.

Template

Ten Simple Rules for Effective Computational Research

[2]

Highlights

the present guide is aimed specifically at the vast majority of scientific researchers: those without formal training in computer science. conduct a software literature review to ascertain what software is available.

The program code itself can be made more understandable by using meaningful variable names and formatting the code consistently. you should maintain a record of the “big picture” functionality (i.e., interconnectivity of components and input/output formats) keep a record of your work

all scientists could be better educated in design, so any investment will be rewarded

Simple tests that the software behaviour matches expectations are essential for ensuring robust results. Get the computer to run tests for you automatically and alert you to problems, using a suitable testing framework.

we advocate an open approach of sharing source code, data, and results as freely as possible.

prioritise in a way that suits you and your projects and career aspirations

Comments

Gostei mais que o outro. Lista de links ao fim é interessante.

Ten Simple Rules for Taking Advantage of Git and GitHub

[3]

Highlights

It provides developers and researchers with a dynamic and collaborative environment, often referred to as a social coding platform, that supports peer review, commenting, and discussion [7] GitHub hooks can be used to automate numerous tasks to help improve the overall quality of your project

GitHub now integrates with archiving services such as Zenodo and Figshare, enabling DOIs to be assigned to code repositories.

Comments

Template

Title

[**???**]

Highlights

Comments

Ten Simple Rules for a Bioinformatics Journal Club

[4]

Highlights

Don’t be a “lonely bioinformatician”[1], create a journal club

We recommend you find a time that is typically free for the majority and suits both students and early career researchers, who may have time commitments during “normal” working hours. For us, that time was 8 A.M.

–> Claramente nao sao brasileiros.

a self-selection bias that we can live with.

ruled out formal presentations with PowerPoint slides.

for every paper, there will be someone who knows nothing about the topic and others who are near-experts

open membership approach and encourage attendance from different cohorts. Mixing research students, coursework students, and early career researchers

We have discussed overcoming procrastination, writing more effectively, online learning, editing Wikipedia, student groups, Software Carpentry, document preparation systems for sci- ence, and even the PLOS Ten Simple Rules collection itself

Although we didn’t originally envisage journal club as a platform for collaborations, we’ve found that meeting regularly to write amongst a group ofsupportive peers has led to a publica- tion, with a style that suits the informal atmosphere of the club.

Keep the barrier to entry low.

Comments

o cdb eh um grupo de estudo

Template

Ten simple rules to increase computational skills among biologists with Code Clubs

[5]

Highlights

participants need regular opportunities to engage in repeated deliberate practice that encourage them to practice their retrieval and application of the material

Initially, the Code Club was used to review code from trainee projects. Instead ofa presentation, the presenter would project their code onto a screen and the participants would go through the code, stating the logic behind each line.

someone may have an R script with a repeating chunk ofcode. The chal- lenge for the session would be to convert the code chunk into a function to be called through- out the script to make it “DRY” (i.e.,

Don’t Repeat Yourself [12]). The presenter leaves the session with several partial or working solutions to their problem, and the importance ofwrit- ing DRY code is reinforced.

We generally find that preparing a Code Club session takes similar effort to preparing a Journal Club presentation “think-pair-share” approach.

It is critical that the presenter and participants respect each other and that a designated indi- vidual (e.g., the lab director) enforces a code of conduct.

After the Code Club, try to incorporate that material into new code or refactor old code.

Set specific goals.

A typical schedule for Code Club is 10 minutes ofintroduction and instruction, 30 minutes ofpaired program- ming, 5 minutes to get groups to wrap up, and 10 minutes to report back to the group.

Using GitHub repositories for each Code Club can help make information, scripts, and data easily available to participants.

Perhaps a first Code Club could introduce using git and GitHub to engage in collaborative coding.

Central to the Code Club format is the use ofpaired programming.

Regardless ofhow partners are selected, consider asking the pairs to identify a navigator and a driver [16] ### Comments Table 1. Examples of successful Code Club topics. !! This is RICH.

How agile project management can work for your research

[6]

Highlights

In an agile project-management plan, an early, partial result, which can be improved on at a later stage, matters more than a perfect result reached only at the end of the project. Academics seem to be late in adopting the agile.

Planning and execution of a smaller number of experiments, followed by immediate data processing and interpretation.

Each layer is addressed in a dedicated, limited period of time (for example, 2–12 weeks), called a sprint.

short meeting (around 30 minutes) with the aim of defining the goal of your sprint:

- what was done the previous week to contribute to the goal?
- What will be done next week to contribute to the goal?
- are there any impediments?

At the end of the sprint, meet all of the stakeholders to discuss results and whether those are in line with expectations (review). Take some time to go into detail and do some analytical brainstorming together

Comments

Agile methods in biomedical software development: a multi-site experience report

[7]

Highlights

To date no group has published a study of agile methods in the context of biomedical informatics in a major bioinfor- matics journal.

The agile movement formally declared its existence in 2000 with the publication of the Agile Manifesto [7].

The agile view is that the best feedback comes from users interacting with working software. To facilitate this sort of feedback, agile methodologies promote early and frequent delivery of well-tested software.

During each of these cycles the software team works through all of the phases of software development – gath- ering requirements, building code that meets those requirements, testing to ensure that requirements are met, and possibly deploying the code into production for customer use.

XP

in XP, peer reviews are implemented as pair programming, i.e. continuous peer review

XP is organized around short iterations, typically one or two weeks long. Features in XP are described as user stories. Programmers estimate the effort to complete each story. Each iteration the customer gets an effort budget and selects a list of stories for implementation from the list of possible user stories.

When developing new features, the development team uses “the simplest design that could possibly work” [13]

When adding a new feature that requires extension of the existing design, programmers refactor the code first to improve the design, and then add the new feature.

Scrum

Scrum is an agile method that focuses on project management [28]. The key practices focus on management of feature backlogs. In addition to selecting features for each sprint, the development team organizes several sprints into releases. Project managers use “burn down” charts to track progress within a sprint or a release.

-Continuing

We asked each participant to focus on how the stated values of agile development (collaboration, work- ing software, embracing change, technical excellence and simplicity) contributed to their project

experiences. We also asked whether the project used any “non-agile” prac- tices, and whether there were any important issues not covered during the survey interview

Applications: - custom workflow engine - community tool to collect, analyze, report, and share genetic sequence data - freely available, open source cancer pathway database - tools to integrate and visualize integromic data

All of the projects used Java as the primary language for software development.

One of the major challenges reported by the projects was the need for a close working relationship between at least two different fields: software engineering and biology.

tacit knowledge becomes a challenge when communicating with software developers.

For scientists, software development is an ancillary task in the service of science, rather than a central goal. Software is only valuable if it can enable the otherwise impossible or save valuable time. As a result, some scientists and cli- nicians are wary of spending too much time on software issues, because this diverts energy away from core scien- tific projects and publications.

Capturing requirements

We all used the combination of short development iterations and customer feedback to resolve the details of specific features. Another feature all projects had in com- mon was the use of web-enabled software tools, such as XPlanner [12], to manage and track requirements.

Automated acceptance testing

For each feature devel- oped, customers wrote machine-executable acceptance tests whose successful execution confirmed programmers had correctly completed their work. The group used the FitNesse open-source testing tool [14]

The process of writing the tests refined and confirmed the users’ understanding of their requirements. This exercise helped demonstrate to customers the real complexity of their requirements and the value of their close involvement to getting the behavior right.

Iteration planning

All of the groups used an iterative approach to organize software development activities. Iterative approaches break up work into smaller pieces, each of which can be measured individually for progress and feedback. Iteration planning consisted of the selec- tion of features from the backlog for completion in the next iteration.

Beyond noting what work was still incomplete, five groups also measured velocity, a simple metric for the amount of work completed during an iteration

Release planning

One characteristic apparent in our analysis that may be common in biomedical informatics was that our customers generally placed less value on fixed release dates and more value on frequent releases of software.

Coding practices

For all of the groups automatedtesting, refactoring and continuous integration were core practices.

Pair programming

One of the most controversial practices suggested by Extreme Programming is pair programming [13].

Comments

Agile development is not business as usual – it represents a real change in how we develop software in this domain.

Historically this has guided practitioners toward throw-away, quick-and-dirty solutions, or in some medical applications, to avoid soft- ware solutions entirely. Agile strikes the right balance by pairing frequent feedback and short iterations with auto- mated testing and refactoring approaches that maintain quality while embracing change.

Project Management 101: The Complete Guide to Agile, Kanban, Scrum and Beyond

[8]

Highlights

Muller's project management system was a resounding success. NASA put the first humans on the moon and brought them back to earth safely in less than a decade of Kennedy's announcement. That was only possible by breaking down the enormous project into manageable, repeatable steps, ones that guaranteed success even when working with so many individuals and companies. It was a project management system—and teamwork—that won the space race.

Invented independently by Korol Adamiecki and Henry Gantt in the early 20th century, the Gantt chart lists a project schedule based on start and finish dates. You list how long a task takes, and if any other tasks have to be completed before that task can start—for instance, you can't serve your meal before you've cooked it

True traditional project management is perhaps an old school model, but its strengths have allowed it to keep hold. It requires upper management to clearly define what it is they want, giving the project focus and consistency early on. TPM's rigidity is also its greatest downfall. It's like an old, dry tree: it's rigid, and doesn't do well with change.

Comments

Be an Agile Academic

[9]

Highlights

Comments

How does this work as an academic? - Rather than saying “this year I want to write two chapters” or “This term, I want to do all the research for x and y topics”, I set discrete, deliverable goals like:

In this month, I’m going to write the conference paper that will eventually become part of x chapter In these two weeks, I’m going to draft the first section of this larger chapter.

Review

It takes some honesty but it’s a real shock to have to write down “yesterday, I did not do a single shred of work on [whatever short term goal] because I spent the entire day researching for a prospective syllabus that isn’t due for three months.”

Writing the blocks down let me see what specifically wasn’t working, and then let me take steps to remove the block.

Testing

Plan out what work I needed that section to do: This section needs to historicize how zoo exhibits incorporated natural element I built myself a failsafe against overwriting, and a “test” to run when doing the each round of editing, not just the first.

Template

Title

[**???**]

Highlights

Comments

Template

Title

[**???**]

Highlights

Commentse

[**???**]

Highlights

Comments

Template

Title

[**???**]

Highlights

Comments

Template

Title

[**???**]

Highlights

Comments

Challenge to scientists: does your ten-year-old code still run?

[10]

Highlights

The challenge dares scientists to find and re-execute old code, to reproduce computationally driven papers they had published ten or more years earlier.

The Ten Years Reproducibility Challenge aims “to find out which of the ten-year-old techniques for writing and publishing code are good enough to make it work a decade later”.

“That’s the problem of actually making backups but not checking that you can still read your backups ten years later”

Several participants chose an alternative that, Courtès suggests, “could very much represent the ‘gold standard’ of reproducible scientific articles”: a Linux package manager called Guix. It promises environments that are reproducible down to the last bit, and transparent in terms of the version of the code from which they are built

“Software is a living thing,” she says. “And if it’s living it will eventually decay, and you will have to repair it, and you’ll have to replace it.”

REPRODUCIBILITY CHECKLIST

- Code
- Document
- Record
- Test
- Guide (master script)
- Archive
- Track (version control)
- Package
- Automate (continuous integration)
- Simplify
- Verify

Python 2.7 puts “at our disposal an advanced programming language that is guaranteed not to evolve anymore”, Rougier writes.

If these data could talk

[11]

Highlights

Even when authors do their best to make their research and data accessible, this lack of formalism reduces the clarity and efficiency of reporting, which contributes to issues of reproducibility. Data provenance aids both reproducibility through systematic and formal records of the relationships among data sources, processes, datasets, publications and researchers.

Seemingly simple details, such as the version of the initial (raw) data or versions of the analytical software programs, are often difficult to identify, and their absence makes replication of analyses impossible, even if the code is available.

Provenance data is most frequently represented as a directed, acyclic graph. Interactions are recorded as a set of edges that relate data-items, transformations (computations), and persons or organizations associated with the data, all represented as vertices (see Fig. 1) This model has been standardized for interoperability by the World Wide Web Consortium (W3C) as the PROV data model (https://www.w3. org/TR/prov-dm/)

The amount of data produced (up to 40 TB s−1)at anLHC experiment is impossible to store due to technological limitations. Thus, details, such as what selection was being performed on the collision data and the detector conditions, directly impact what was being recorded during a LHC run.

–> Para quem trabalha com dados públicos, como voces pegam informaçoes sobre como os dados foram gerados? If data provenance becomes a well-established convention, eventually the provenance metadata associated with each dataset will provide the complete data record.

Comments

Ten Simple Rules for Reproducible Computational Research

[12]

Highlights

We want to emphasize that reproducibility is not only a moral responsibility with respect to the scientific field, but that a lack of reproducibility can also be a burden for you as an individual researcher.

We believe that the rewards of reproducibility will compensate for the risk of having spent valuable time developing an annotated catalog of analyses that turned out as blind alleys.

As a minimal requirement, you should at least be able to reproduce the results yourself.

For every involved step, you should ensure that every detail that may influence the execution of the step is recorded.

If manual operations cannot be avoided, you should as a minimum note down which data files were modified or moved, and for what purpose.

Archiving the exact ver- sions of programs actually used may thus save a lot of hassle at later stages

For analyses that involve random numbers, this means that the random seed should be recorded

–> Das análises que voces fazem, sabem se alguma usa numeros aleatorios? Muitas vezes eles estão escondidos em algoritmos.

Rule 7: Always Store Raw Data behind Plots

When plotting is performed using a command-based system like R, it is convenient to also store the code used to make the plot.

To allow efficient retrieval of details behind textual statements, we suggest that statements are connected to underlying results already from the time the statements are initially formulated (for instance in notes or emails). Such a connection can for instance be a simple file path to detailed results, or the ID of a result in an analysis framework.

Last, but not least, all input data, scripts, versions, parameters, and inter- mediate results should be made publicly and easily accessible

Comments

Experimenting with reproducibility: a case study of robustness in bioinformatics

[13]

Highlights

Here, we present a case study of our difficulties in reproducing a published bioinformatics method even though code and data were available

Hidden reproducibility issues are like an underwater iceberg. Scientific journal readers have the impression that they can almost see the full work in- volved in the method. In reality, articles do not take into account adjustment and configuration for significant replication in most cases

To conclude,wewereabletotestthe robustness of the methodwith our Python implementation.However, this took ap- proximately two months for a senior researcher and six months for a master student.

The difficulty in accessing and understand- ing code may lead to applying code blindly without checking the validity of the results. Often, scientists prefer to believe that re- sults are correct because checking the validity of the resultsmay require significant time.

following recommendations: - Improve life scientists software development skills. r Use online repositories and tools to help other scientists in their exploration of the method [25, 26, 30]. - Enhance the cooperation between academia and industry [39, 40, 46]. - Develop an open-source continuous testing ecosystem with community standards, well-identified datasets to validate tools across versions and datasets, and go beyond the pub- lication of a PDF file.

Comments

Title

[**???**]

Highlights

Comments

Good enough practices in scientific computing

[14]

Highlights

Ensure that raw data are backed up in more than one location. Record all the steps used to process data. Use a unique identifier for every record. Submit data to a reputable DOI-issuing repository

Consider changing file permissions to read-only or using spreadsheet protection features consult with your local Information Technology (IT) group or library. Alternatively, cloud computing resources, like Amazon Simple Storage Service (Amazon S3) CSV for tabular data “tidy” data write scripts for every stage of data processing Sites such as Figshare, Dryad, and Zenodo allow others to find your work, use it, and cite it

Decompose programs into functions Give functions and variables meaningful names Make dependencies and requirements explicit Do not comment and uncomment sections of code to control a program’s behavior Submit code to a reputable DOI-issuing repository.

Functions should take no more than 5 or 6 input parameters and should not reference outside information. The key motivation here is to fit the program into the most limited memory of all: ours adding a file called something like requirements.txt to the root directory of the project

Create an overview of your project. Make the license explicit.

project's title, a brief description, up-to-date contact information, and an example or 2 of how to run various cleaning or analysis tasks Have a LICENSE file in the project's home directory that clearly states what license(s) apply to the project's software, data, and manuscripts.

doc directory data directory results directory src directory

For a small project with 1 main output, a single controller script should be placed in the main src directory and distinguished clearly by a name such as "runall"

Use a version control system: GitHub

Write manuscripts using online tools with rich formatting, change tracking, and reference management Write the manuscript in a plain text format that permits version control.

Instead of an email-based workflow, we recommend mirroring good practices for managing software and data to make writing scalable, collaborative, and reproducible Good intentions aside, it always comes down to, "just give me a Word document with tracked changes", or similar

The semantic web. Ontologies and other formal definitions of data are useful, but in our experience, even simplified things like Dublin Core are rarely encountered in the wild. –> Tiago cries in OWL

Comments

Software engineering for scientific big data analysis

[15]

Highlights

many of the computational tools released can only be described, politely, as "research-grade." These tools can place an exorbitant burden on the end users who find themselves in the position of attempting to execute and chain them into complex analysis pipelines.

Spend adequate time designing the scope and expectations of your project. Clearly define the scope of your work, and anticipate corner cases that you think you might need to account for

When assessing the quality of an existing tool, you should follow the same guidelines that are laid out here for designing your own software: i) Can you understand what the code does, so that you can verify its logic and assumptions, can fix bugs, and improve the code if required? (ii) Does the software contain tests? (iii) Does it have a high test coverage? (iv) Does it use continuous integration? (v) What other dependencies does it rely on? (vi) Are those dependencies deemed robust, reliable software? (vii) Are the licensing and implementation compatible with integration in your workflow?

Names must be self-explanatory to those who will be interacting with them Metadata is as important as the primary dataset.

Installation of the computational tool is likewise important; installation procedures for a tool and its dependencies can become a challenge for users of different backgrounds. Simply put, if a tool cannot be installed, it cannot be used.

Comments

Title

[???]

Highlights

Comments

References

1. **A Quick Guide to Organizing Computational Biology Projects**
   William Stafford Noble
   PLoS Computational Biology (2009-07-31) https://doi.org/fbbpkn
   DOI: 10.1371/journal.pcbi.1000424 · PMID: 19649301 · PMCID: PMC2709440

2. **Ten Simple Rules for Effective Computational Research**
   James M. Osborne, Miguel O. Bernabeu, Maria Bruna, Ben Calderhead, Jonathan Cooper, Neil Dalchau, Sara-Jane Dunn, Alexander G. Fletcher, Robin Freeman, Derek Groen, … Charlotte Deane
   PLoS Computational Biology (2014-03-27) https://doi.org/gdb9wq
   DOI: 10.1371/journal.pcbi.1003506 · PMID: 24675742 · PMCID: PMC3967918

3. **Ten Simple Rules for Taking Advantage of Git and GitHub**
   Yasset Perez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, … Juan Antonio Vizcaíno
   PLOS Computational Biology (2016-07-14) https://doi.org/gbrb39
   DOI: 10.1371/journal.pcbi.1004947 · PMID: 27415786 · PMCID: PMC4945047

4. **Ten Simple Rules for a Bioinformatics Journal Club**
   Andrew Lonsdale, Jocelyn Sietsma Penington, Timothy Rice, Michael Walker, Harriet Dashnow
   PLOS Computational Biology (2016-01-28) https://doi.org/ghcwtj
   DOI: 10.1371/journal.pcbi.1004526 · PMID: 26820645 · PMCID: PMC4731213

5. **Ten simple rules to increase computational skills among biologists with Code Clubs**
   Ada K. Hagan, Nicholas A. Lesniak, Marcy J. Balunas, Lucas Bishop, William L. Close, Matthew D. Doherty, Amanda G. Elmore, Kaitlin J. Flynn, Geoffrey D. Hannigan, Charlie C. Koumpouras, … Patrick D. Schloss
   PLOS Computational Biology (2020-08-27) https://doi.org/gg92xw
   DOI: 10.1371/journal.pcbi.1008119 · PMID: 32853198 · PMCID: PMC7451508

6. **How agile project management can work for your research**
   Laura Pirro
   Nature (2019-04-10) https://www.nature.com/articles/d41586-019-01184-9
   DOI: 10.1038/d41586-019-01184-9

7. **:{unav)**
   David W Kane, Moses M Hohman, Ethan G Cerami, Michael W McCormick, Karl F Kuhlmman, Jeff A Byrd
   BMC Bioinformatics (2006) https://doi.org/bg2rwp
   DOI: 10.1186/1471-2105-7-273 · PMID: 16734914 · PMCID: PMC1539031

8. **Project Management 101: The Complete Guide to Agile, Kanban, Scrum and Beyond**
   Zapier
   Zapier (2016-05-12) https://zapier.com/learn/project-management/project-management-systems/

9. **Be an Agile Academic**
   katy peplin
   https://www.katypeplin.com/blog/2017/10/25/be-an-agile-academic

10. **Challenge to scientists: does your ten-year-old code still run?**
Jeffrey M. Perkel
Nature (2020-08-24) https://www.nature.com/articles/d41586-020-02462-7
DOI: 10.1038/d41586-020-02462-7

11. **If these data could talk**
Thomas Pasquier, Matthew K. Lau, Ana Trisovic, Emery R. Boose, Ben Couturier, Mercè Crosas, Aaron M. Ellison, Valerie Gibson, Chris R. Jones, Margo Seltzer
Scientific Data (2017-09-05) https://doi.org/gbvqjp
DOI: 10.1038/sdata.2017.114 · PMID: 28872630 · PMCID: PMC5584398

12. **Ten Simple Rules for Reproducible Computational Research**
Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, Eivind Hovig
PLoS Computational Biology (2013-10-24) https://doi.org/pjb
DOI: 10.1371/journal.pcbi.1003285 · PMID: 24204232 · PMCID: PMC3812051

13. **Experimenting with reproducibility: a case study of robustness in bioinformatics**
Yang-Min Kim, Jean-Baptiste Poline, Guillaume Dumas
GigaScience (2018-06-28) https://doi.org/gfj5zq
DOI: 10.1093/gigascience/giy077 · PMID: 29961842 · PMCID: PMC6054242

14. **Good enough practices in scientific computing**
Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, Tracy K. Teal
PLOS Computational Biology (2017-06-22) https://doi.org/gbkbwp
DOI: 10.1371/journal.pcbi.1005510 · PMID: 28640806 · PMCID: PMC5480810

15. **Software engineering for scientific big data analysis**
Björn A Grüning, Samuel Lampa, Marc Vaudel, Daniel Blankenberg
GigaScience (2019-05) https://doi.org/gf4f4m
DOI: 10.1093/gigascience/giz054 · PMID: 31121028 · PMCID: PMC6532757