(GIGA)^n SCIENCE

GigaScience, 10, 2021, 1–19

doi: 10.1093/gigascience/giaa140
Review

REVIEW

Streamlining data-intensive biology with workflow systems

Taylor Reiter [1], Phillip T. Brooks† [1], Luiz Irber† [1], Shannon E.K. Joslin† [2], Charles M. Reid† [1], Camille Scott† [1], C. Titus Brown [1] and N. Tessa Pierce-Ward* [1]

[1]Department of Population Health and Reproduction, University of California, Davis, 1 Shields Avenue, Davis, CA 95616, USA and [2]Department of Animal Science, University of California, Davis, 1 Shields Avenue, Davis, CA 95616, USA

*Correspondence address: N. Tessa Pierce-Ward, Department of Population Health and Reproduction, University of California, Davis, 1 Shields Avenue, Davis, CA 95616, USA. E-mail: ntpierce@ucdavis.edu

†Co-equal contributions.

Abstract

As the scale of biological data generation has increased, the bottleneck of research has shifted from data generation to analysis. Researchers commonly need to build computational workflows that include multiple analytic tools and require incremental development as experimental insights demand tool and parameter modifications. These workflows can produce hundreds to thousands of intermediate files and results that must be integrated for biological insight. Data-centric workflow systems that internally manage computational resources, software, and conditional execution of analysis steps are reshaping the landscape of biological data analysis and empowering researchers to conduct reproducible analyses at scale. Adoption of these tools can facilitate and expedite robust data analysis, but knowledge of these techniques is still lacking. Here, we provide a series of strategies for leveraging workflow systems with structured project, data, and resource management to streamline large-scale biological analysis. We present these practices in the context of high-throughput sequencing data analysis, but the principles are broadly applicable to biologists working beyond this field.

Keywords: workflows; automation; repeatability; data-intensive biology

Introduction

Biological research has become increasingly computational. In particular, genomics has experienced a deluge of high-throughput sequencing data that has already reshaped our understanding of the diversity and function of organisms and communities, building basic understanding from ecosystems to human health. The analysis workflows used to produce these insights often integrate hundreds of steps and involve a myriad of decisions ranging from small-scale tool and parameter choices to larger-scale design decisions around data processing and statistical analyses. Each step relies not just on analysis code written by the researcher but on third-party software, its dependencies, and the compute infrastructure and operating system on which the code is executed. Historically, this has led to the patchwork availability of underlying code for analyses, as well as a lack of interoperability of the resulting software and analysis pipelines across compute systems [1]. Combined with unmet training needs in biological data analysis, these conditions undermine the reuse of data and the reproducibility of biological research, vastly limiting the value of our generated data [2].

The biological research community is strongly committed to addressing these issues, recently formalizing the FAIR practices: the idea that all life sciences research (including data and analysis workflows) should be Findable, Accessible, Interoperable, and Reusable [3]. For computational analyses, these ideals are readily achievable with current technologies, but implementing them in

Received: 16 August 2020; **Revised:** 6 November 2020; **Accepted:** 13 November 2020

practice has proven difficult, particularly for biologists with little training in computing [3]. However, the recent maturation of data-centric workflow systems designed to automate and facilitate computational workflows is expanding our capacity to conduct end-to-end FAIR analyses [4]. These workflow systems are designed to handle some aspects of computational workflows internally: namely, the interactions with software and computing infrastructure, and the ordered execution of each step of an analysis. By reducing the manual input and monitoring required at each analysis juncture, these integrated systems ensure that analyses are repeatable and can be executed at much larger scales. In concert, the standardized information and syntax required for rule-based workflow specification makes code inherently modular and more easily transferable between projects [4, 5]. For these reasons, workflow systems are rapidly becoming the workhorses of modern bioinformatics.

Adopting workflow systems requires some level of up-front investment, first to understand the structure of the system and then to learn the workflow-specific syntax. These challenges can preclude adoption, particularly for researchers without significant computational experience [6]. In our experiences with both research and training, these initial learning costs are similar to those required for learning more traditional analysis strategies but then provide a myriad of additional benefits that both facilitate and accelerate research. Furthermore, online communities for sharing reusable workflow code have proliferated, meaning that the initial cost of encoding a workflow in a system is mitigated via use and reuse of common steps, leading to faster time-to-insight [4, 7].

Building upon the rich literature of "best" and "good enough" practices for computational biology [8–10], we present a series of strategies and practices for adopting workflow systems to streamline data-intensive biology research. This article is designed to help guide biologists towards project, data, and resource management strategies that facilitate and expedite reproducible data analysis in their research. We present these strategies in the context of our own experiences working with high-throughput sequencing data, but many are broadly applicable to biologists working beyond this field.

Workflows Facilitate Data-Intensive Biology

Data-intensive biology typically requires that researchers execute computational workflows using multiple analytic tools and apply them to many experimental samples in a systematic manner. These workflows commonly produce hundreds to thousands of intermediate files and require incremental changes as experimental insights demand tool and parameter modifications. Many intermediate steps are central to the biological analysis, but others, such as converting between file formats, are rote computational tasks required to passage data from one tool to the next. Some of these steps can fail silently, producing incomplete intermediate files that imperceptibly invalidate downstream results and biological inferences. Properly managing and executing all of these steps is vital but can be both time-consuming and error-prone, even when automated with scripting languages such as bash.

The emergence and maturation of workflow systems designed with bioinformatics challenges in mind has revolutionized computing in data-intensive biology [11]. Workflow systems contain powerful infrastructure for workflow management that can coordinate runtime behavior, self-monitor progress and resource usage, and compile reports documenting the results of a workflow (Fig. 1). These features ensure that the steps for data analysis are repeatable and at least minimally described from start to finish. When paired with proper software management, fully contained workflows are scalable, robust to software updates, and executable across platforms, meaning that they will likely still execute the same set of commands with little investment by the user after weeks, months, or years.

To properly direct an analysis, workflow systems need to encode information about the relationships between every workflow step. In practice, this means that each analysis step must specify the input (or types of inputs) needed for that step, and the output (or types of outputs) being produced. This structure provides several additional benefits. First, workflows become minimally self-documented because the directed graph produced by workflow systems can be exported and visualized, producing a graphical representation of the relationships between all steps in a pipeline (see "Visualize your Workflow" section). Next, workflows are more likely to be fully enclosed without undocumented steps that are executed by hand, meaning that analyses are more likely to be reproducible. Finally, each step becomes a self-contained unit that can be used and reused across multiple analysis workflows, so scientists can spend less time implementing standard steps and more time on their specific research questions. In sum, the internal scaffolding provided by workflow systems helps build analyses that are generally better documented, repeatable, transferable, and scalable.

Getting Started with Workflows

The workflow system that you choose will be largely dependent on your analysis needs. Here, we draw a distinction between two types of workflows: "research" workflows, which are under iterative development to answer novel scientific questions, and "production" workflows, which have reached maturity and are primarily used to run a standard analysis on new samples. In particular, research workflows require flexibility and assessment at every step: outliers and edge cases may reveal interesting biological differences, rather than sample processing or technical errors. Many workflow systems can be used for either type, but we note cases where their properties facilitate one of these types over the other.

Using workflows without learning workflow syntax

While the benefits of executing an analysis within a data-centric workflow system are immense, the learning curve associated with command-line systems can be daunting. It is possible to obtain the benefits of workflow systems without learning new syntax. Websites such as Galaxy, Cavatica, and EMBL-EBI MGnify offer online portals in which users build workflows around publicly available or user-uploaded data [12–14]. On the command line, many research groups have used workflow systems to wrap one or many analysis steps (specified in an underlying workflow language) in a more user-friendly command-line application that accepts user input and executes the analysis. These pipeline applications allow users to take advantage of workflow software without needing to write the workflow syntax or manage software installation for each analysis step. Some examples include the nf-core RNA sequencing (RNA-seq) pipeline [1, 15], the PiGx genomic analysis toolkit [16], the ATLAS metagenome assembly and binning pipeline [17, 18], the Sunbeam metagenome analysis pipeline [19, 20], and two from our own laboratory, the dammit eukaryotic transcriptome annotation pipeline [21] and the elvers *de novo* transcriptome pipeline

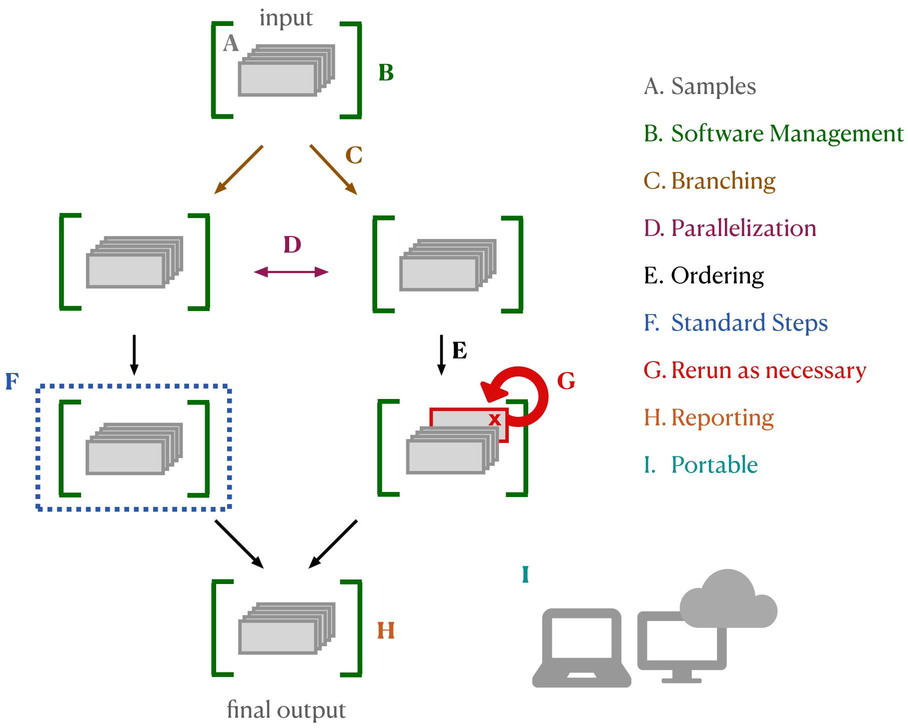


Figure 1: Workflow systems: bioinformatics workflow systems have built-in functionality that facilitates and simplifies running analysis pipelines. A. Samples: workflow systems enable the user to use the same code to run each step on each sample. Samples can be easily added if the analysis expands. B. Software management: integration with software management tools (e.g., conda, singularity, docker) can automate software installation for each step. C. Branching, D. parallelization, and E. ordering: workflow systems handle conditional execution, ensuring that tasks are executed in the correct order for each sample file, including executing independent steps in parallel if possible given the resources provided. F. Standard steps: many steps are now considered "standard" (e.g., quality control). Workflow languages keep all information for a step together and can be written to enable remixing and reuse of individual steps across pipelines. G. Rerun as necessary: workflow systems keep track of which steps executed properly and on which samples, and allow failed steps (or additional steps) to be rerun rather than re-executing the entire workflow. H. Reporting: workflow languages enable comprehensive reporting on workflow execution and resource utilization by each tool. I. Portability: analyses written in workflow languages (with integrated software management) can be run across computing systems without changes to code.

[22]. These pipeline applications typically execute a series of standard steps, but many provide varying degrees of customizability ranging from tool choice to parameter specification.

Choosing a workflow system

If your use case extends beyond these tools, there are several scriptable workflow systems that offer comparable benefits for carrying out your own data-intensive analyses. Each has it own strengths, meaning that each workflow software will meet an individual's computing goals differently (see Table 1). Our laboratory has adopted Snakemake [23, 24], in part due to its integration with Python, its flexibility for building and testing new analyses in different languages, and its intuitive integration with software management tools (described below). Snakemake and Nextflow [25] are commonly used for developing new research pipelines, where flexibility and iterative, branching development are key features. Common Workflow Language (CWL) and Workflow Description Language (WDL) are workflow specification formats that are more geared towards scalability, making them ideal for production-level pipelines with hundreds of thousands of samples [26]. WDL and CWL are commonly executed on platforms such as Terra [27] or Seven Bridges Platform [28]. Language-specific workflow systems, such as ROpenSci's Drake [29], can take full advantage of the language's internal data structures, and provide automation and reproducibility benefits for workflows executed primarily within the language ecosystem.

Table 1: Four of the most widely used bioinformatics workflow systems (2020), with links to documentation, example workflows, and general tutorials.

Workflow system	Documentation	Example workflow	Tutorial
Snakemake	[30]	[31]	[32]
Nextflow	[33]	[34]	[35]
CWL	[36]	[37]	[38]
WDL	[39]	[40]	[41]

In many cases, there may be tutorials online that are tailored for use cases in a particular field. All of these systems can interact with tools or tasks written in other languages and can function across cloud computing systems and high-performance computing clusters. Some can also import full workflows from other specification languages.

The best workflow system to choose may be the one with a strong and accessible local or online community in your field, somewhat independent of your computational needs. The availability of field-specific data analysis code for reuse and modification can facilitate the adoption process, as can community support for new users. Fortunately, the standardized syntax required by workflow systems, combined with widespread adoption in the open science community, has resulted in a proliferation of open access workflow-system code for routine analysis steps [42, 43]. At the same time, consensus approaches for data analysis are emerging, further encouraging reuse of existing code [44–48].

The Getting Started Developing Workflows section contains strategies for modifying and developing workflows for your own analyses.

Wrangling Scientific Software

Analysis workflows commonly rely on multiple software packages to generate final results. These tools are heterogeneous in nature: they are written by researchers working in different coding languages, with varied approaches to software design and optimization, and often for specific analysis goals. Each program has a number of other programs it depends upon to function ("dependencies"), and as software changes over time to meet research needs, the results may change, even when run with identical parameters. As a result, it is critical to take an organized approach to installing, managing, and keeping track of software and software versions. On many compute systems, system-wide software management is overseen by system administrators, who ensure that commonly used and requested software is installed into a "module" system available to all users. Unfortunately, this system limits software version transparency and does not lend itself well to exploring new workflows and software because researchers do not have permission to install software themselves. To meet this need, most workflow managers integrate with software management systems that handle software installation, management, and packaging, alleviating problems that arise from complex dependencies and facilitating documentation of software versions. Software management systems range from lightweight systems that manage only the software and its dependencies to heavyweight systems that control for all aspects of the runtime and operating system, ensuring 100% reproducibility of results across computational platforms and time.

On the lightweight end, the conda package manager has emerged as a leading software management solution for research workflows (Fig. 2). Conda handles both cluster permission and version conflict issues with a user-based software environment system and features a straightforward "recipe" system that simplifies the process of making new software installable (including simple management of versions and updates). These features have led to widespread adoption within the bioinformatics community: packages for new software become quickly available and can be installed easily across platforms. However, conda does not completely isolate software installations and aims neither for bitwise reproducibility nor for long-term archiving of install packages, meaning that installations will not be completely reproducible over time. Heavyweight software management systems package not only the software of interest but also the runtime environment information, with the goal of ensuring perfect reproducibility in software installation over time. Tools such as singularity and docker [3, 11, 49, 50] wrap software environments in "containers" that capture and reproduce the runtime environment information. Container-based management is particularly useful for systems where some dependencies may not be installable by lightweight managers. However, software installation within these containers can be limited by similar reproducibility issues, including changes in dependency installations over time. "Functional package managers" such as GNU Guix and Nix strictly require all dependency and configuration details to be encoded within each software package, providing the most comprehensively reproducible installations. These have begun to be integrated into some bioinformatic tools [16] but have a steeper learning curve for independent use. In addition, standard installation of these managers requires system-wide installation permissions, requiring assistance from system administrators on most high-performance computing systems.

Getting Started with Software Management

Using software without learning software management systems

First, there are a number of ways to test software before needing to worry about installation. Some software packages are available as web-based tools and, through a series of data upload and parameter specifications, allow the user to interact with a tool that is running on a back-end server. Integrated development environments (IDE) such as PyCharm and RStudio can manage software installation for language-specific tools and can be helpful when writing analysis code. While these approaches do not integrate into reproducible workflows, they may be ideal for testing a tool to determine whether it is useful for your data before integration in your analysis.

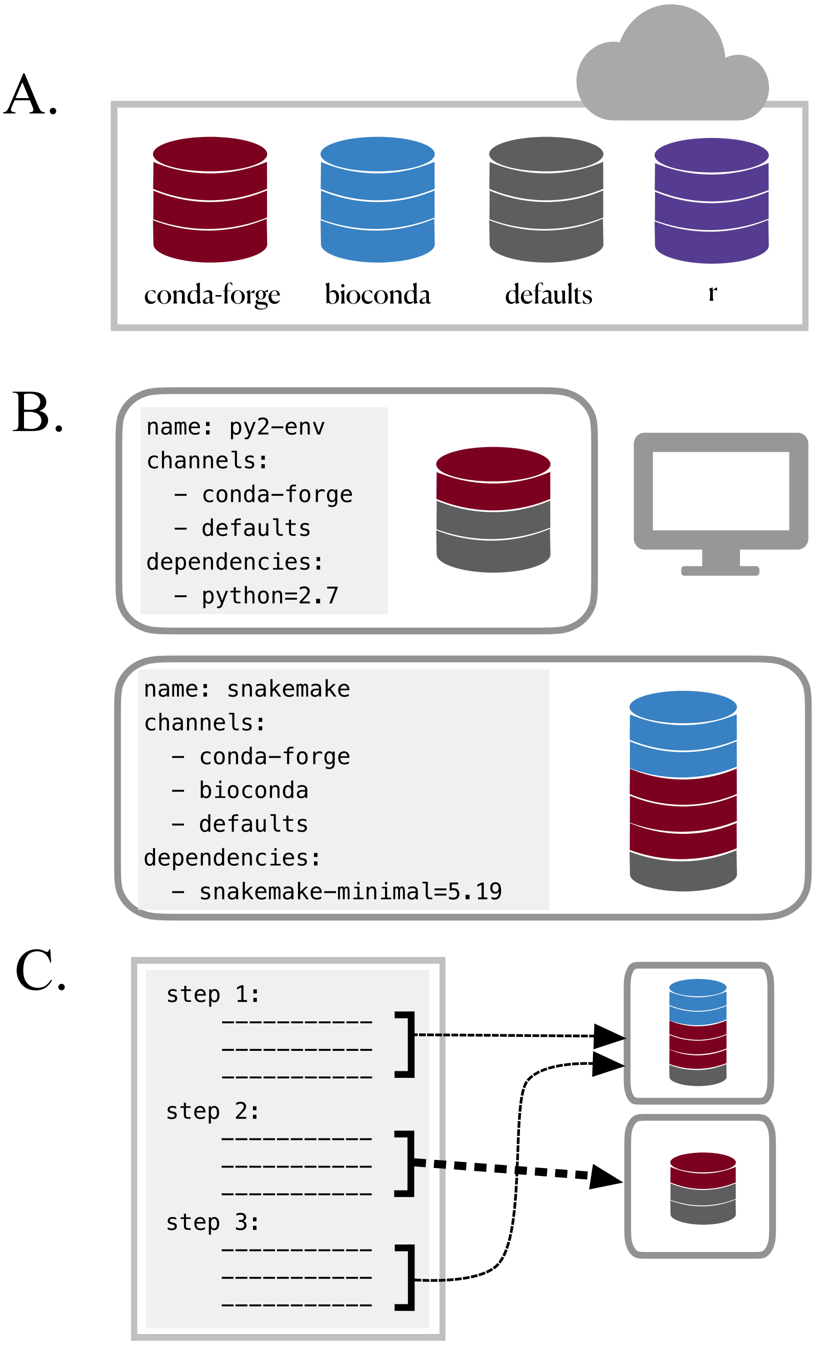
Choosing a software management system

It is important to balance the time needed to learn to properly use a software management system with the needs of both the project and the researchers. Software management systems with large learning curves are less likely to be widely adopted among researchers with a mix of biological and computational backgrounds. In our experience, software management with conda nicely balances reproducibility with flexibility and ease of use. These trade-offs are best for research workflows under active development, where flexible software installation solutions that enable new analysis explorations or regular tool updates are critical. For production workflows that require maximal reproducibility, it is worth the larger investment required to use heavyweight systems. This is particularly true for advanced users who can more easily navigate the steps required for utilizing these tools. Container-based software installation via docker and singularity is common for production-level workflows, and Guix and Nix-based solutions are gaining traction. Importantly, the needs and constraints of a project can evolve over time, as may the system of choice.

Integrating software management within workflows

Workflow systems provide seamless integration with a number of software management tools. Each workflow system requires different specification for initiation of software management but typically requires ∼1 additional line of code per step that requires the use of software. If the software management tool is installed locally, the workflow will automatically download and install the specified environment or container and use it for the specified step.

In our experience, the complete solution for using scientific software involves a combination of approaches. Interactive and exploratory analyses conducted in IDEs and jupyter notebooks (usually with local software installation with conda) are useful for developing an analysis strategy and creating an initial workflow. This is then followed by workflow-integrated software



```
name: py2-env
channels:
  - conda-forge
  - defaults
dependencies:
  - python=2.7
```

```
name: snakemake
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - snakemake-minimal=5.19
```

Figure 2: The conda package and environment manager simplifies software installation and management. A. Conda recipe repositories: each program distributed via conda has a "recipe" describing all software dependencies needed for installation using conda (each of which must also be installable via conda). Recipes are stored and managed in the cloud in separate "channels," some of which specialize in particular fields or languages (e.g., the "bioconda" channel specializes in bioinformatics software, while the "conda-forge" channel is a more general effort to provide and maintain standardized conda packages for a wide range of software) [11]. B. Use conda environments to avoid installation conflicts: conda does not require root privileges for software installation, thus enabling use by researchers working on shared cluster systems. However, even user-based software installation can encounter dependency conflicts. For example, you might need to use Python2 to install and run a program (e.g., older scripts written by members of your laboratory), while also using snakemake to execute your workflows (requires Python ≥3.5). By installing each program into an isolated "environment" that contains only the software required to run that program, you can ensure that all programs used throughout your analysis will run without issue. Using small, separate environments for your software, specifying the desired software version, and building many simple environments to accommodate different steps in your workflow is critical for reducing the amount of time it takes conda to resolve dependency conflicts between different software tools ("solve" an environment). Conda virtual environments can be created and installed either on the command line or via an environment YAML file, as shown. In this case, the environment file also specifies which conda channels to search and download programs from. When specified in a YAML file, conda environments are easily transferable between computers and operating systems. C. Most workflow management software enables specification of individual software environments for each step. In this example, steps 1 and 3 rely on the same environment, while step 2 uses a different environment. Broad community adoption has resulted in a proliferation of both conda-installable scientific software and tools that leverage conda installation specifications. For example, the Mamba package manager is an open source reimplementation of the conda manager that can install conda-style environments with increased efficiency [51]. The BioContainers Registry is a project that automatically builds and distributes docker and singularity containers for bioinformatics software packages using each package's conda installation recipe [52].

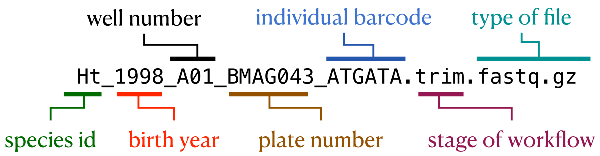


Figure 3: Consistent and informative file naming improves organization and interpretability. For ease of grouping and referring to input files, it is useful to keep unique sample identification in the filename, often with a metadata file explaining the meaning of each unique descriptor. For analysis scripts, it can help to implement a numbering scheme, where the name of the first file in the analysis begins with "00," the next with "01," and so on. For output files, it can help to add a short, unique identifier to output files processed with each analysis step. This particular file is a RAD-seq fastq file of a fish species that has been preprocessed with a fastq quality trimming tool.

management via conda, singularity, or nixOS for executing the resulting workflow on many samples. This process not linear: we often cycle between exploratory testing and automation as we iteratively extend our analyses.

Workflow-Based Project Management

Project management, the strategies and decisions used to keep a project organized, documented, functional, and shareable, is foundational to any research program. Clear organization and management is a learned skill that takes time to implement. Workflow systems simplify and improve computational project management, but even workflows that are fully specified in workflow systems require additional investment to stay organized, documented, and backed up.

Systematically document your workflows

Pervasive documentation provides indispensable context for biological insights derived from an analysis, facilitates transparency in research, and increases reusability of the analysis code. Good documentation covers all aspects of a project, including organization of files and results, clear and commented code, and accompanying explanatory documents for design decisions and metadata. Workflow systems facilitate building this documentation because each analysis step (with chosen parameters) and the links between those steps are completely specified within the workflow syntax. This feature streamlines code documentation, particularly if you include as much of the analysis as possible within the automated workflow framework. Outside of the analysis itself, applying consistent organizational design can capitalize on the structure and automation provided by workflows to simplify the generation of high-quality documentation for all aspects of your project. Below, we discuss project management strategies for building reproducible workflow-enabled biological analyses.

Use consistent, self-documenting names

Using consistent and descriptive identifiers for your files, scripts, variables, workflows, projects, and even manuscripts helps keep your projects organized and interpretable for you and collaborators. For workflow systems, this strategy can be implemented by tagging output files with a descriptive identifier for each analysis step, either in the filename or by placing output files within a descriptive output folder. For example, the file shown in Fig. 3 has been preprocessed with a quality control trimming step. For large workflows, placing results from each step of your analysis in isolated, descriptive folders can be essential for keeping your project workspace clean and organized.

Store workflow metadata with the workflow

Developing biological analysis workflows can involve hundreds of small decisions: What parameters work best for each step? Why did you use a certain reference file for annotation as compared with other available files? How did you finally manage to get around the program or installation error? All of these pieces of information contextualize your results and may be helpful when sharing your findings. Keeping information about these decisions in an intuitive and easily accessible place helps you find it when you need it. To capitalize on the utility of version control systems described below, it is most useful to store this information in plain text files. Each main directory of a project should include notes on the data or scripts contained within, so that a collaborator could look into the directory and understand what to find there (especially because that "collaborator" is likely to be you, a few months from now). Code itself can contain documentation—you can include comments with the reasoning behind algorithm choice or include a link to online documentation or a solution that helped you decide how to shape your differential expression analysis. Larger pieces of information can be kept in "README" or notes documents kept alongside your code and other documents. For example, a GitHub repository documenting the reanalysis of the Marine Microbial Eukaryote Transcriptome Sequencing Project uses a README alongside the code to document the workflow and DOIs for data products [53, 54]. While this particular strategy cannot be automated, it is critical for interpreting the final results of your workflow.

Document data and analysis exploration using computational notebooks

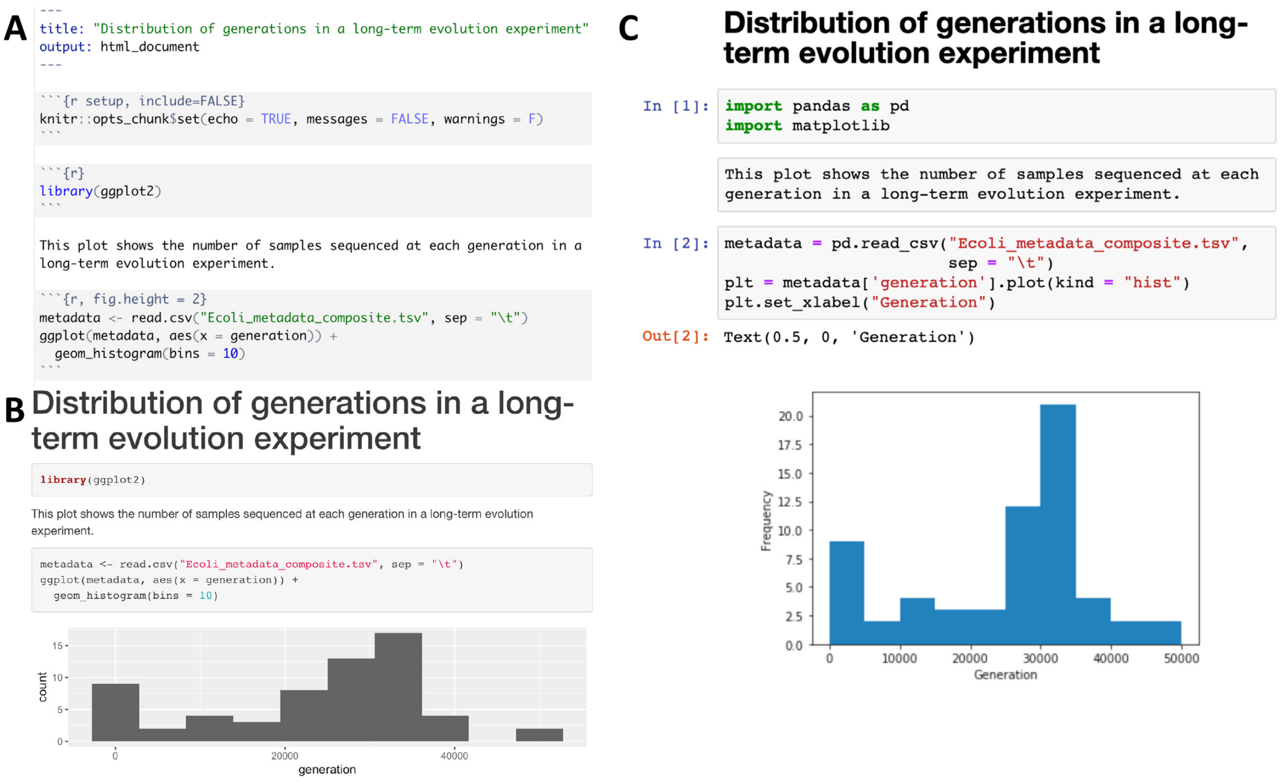
Computational notebooks allow users to combine narrative, code, and code output (e.g., visualizations) in a single location, enabling the user to conduct analysis and visually assess the results in a single file (see Fig. 4). These notebooks allow for fully documented iterative analysis development and are particularly useful for data exploration and developing visualizations prior to integration into a workflow or as a report generated by a workflow that can be shared with collaborators.

Visualize your workflow

Visual representations can help illustrate the connections in a workflow and improve the readability and reproducibility of your project. At the highest level, flowcharts that detail relationships between steps of a workflow can help provide big-picture clarification, especially when the pipeline is complicated. For individual steps, a graphical representation of the output can show the status of the project or provide insight on additional analyses that should be added. For example, Fig. 5 exhibits a modified Snakemake workflow visualization from an RNA-seq quantification pipeline [59].

Version control your project

As your project develops, version control allows you to keep track of changes over time. Several methods can be used to track changes even without version control software, including frequent hard drive backups or manually saving different versions of the same file—e.g., by appending the date to a script name or



~~~
---
title: "Distribution of generations in a long-term evolution experiment"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, messages = FALSE, warnings = F)
~~~

```{r}
library(ggplot2)
```

This plot shows the number of samples sequenced at each generation in a
long-term evolution experiment.

```{r, fig.height = 2}
metadata <- read.csv("Ecoli_metadata_composite.tsv", sep = "\t")
ggplot(metadata, aes(x = generation)) +
  geom_histogram(bins = 10)
```
```

Distribution of generations in a long-term evolution experiment

```
library(ggplot2)
```

This plot shows the number of samples sequenced at each generation in a long-term evolution experiment.

```
metadata <- read.csv("Ecoli_metadata_composite.tsv", sep = "\t")
ggplot(metadata, aes(x = generation)) +
  geom_histogram(bins = 10)
```

Distribution of generations in a long-term evolution experiment

```
In [1]: import pandas as pd
        import matplotlib

        This plot shows the number of samples sequenced at each
        generation in a long-term evolution experiment.

In [2]: metadata = pd.read_csv("Ecoli_metadata_composite.tsv",
                               sep = "\t")
        plt = metadata['generation'].plot(kind = "hist")
        plt.set_xlabel("Generation")

Out[2]: Text(0.5, 0, 'Generation')
```

Figure 4: Examples of computational notebooks. Computational notebooks allow the user to mix text, code, and results in 1 document. A. RMarkdown document viewed in the RStudio integrated development environment; B. rendered HTML file produced by knitting the RMarkdown document [55]. C. Jupyter Notebook, where code, text, and results are rendered inline as each code chunk is executed [56]. The second grey chunk is a raw Markdown chunk with text that will be rendered inline when executed. Both notebooks generate a histogram of a metadata feature, number of generations, from a long-term evolution experiment with *Escherichia coli* [57]. Computational notebooks facilitate sharing by packaging narrative, code, and visualizations together. Sharing can be enhanced further by packaging computational notebooks with tools like Binder [58]. Binder builds an executable environment (capable of running RStudio and jupyter notebooks) out of a GitHub repository using package management systems and docker to build reproducible and executable software environments as specified in the repository. Binders can be shared with collaborators (or students in a classroom setting), and analysis and visualization can be ephemerally reproduced or altered from the code provided in computational notebooks.

appending "version_1" or "version_FINAL" to a manuscript draft. However, version control systems such as Git or Mercurial can both simplify and standardize this process, particularly as workflow length and complexity increase. These systems can keep track of all changes over time, even across multiple systems, scripting languages, and project contributors (see Fig. 6). If a key piece of a workflow inexplicably stops working, consistent version control can allow you to rewind in time and identify differences from when the pipeline worked to when it stopped working. Furthermore, backing up your version-controlled analysis in an online repository such as GitHub, GitLab, or Bitbucket can provide critical insurance as you iteratively modify and develop your workflow.

When combined with online backups, version control systems also facilitate code and data availability and reproducibility for publication. For example, to preserve the version of code that produced published results, you can create a "release": a snapshot of the current code and files in a GitHub repository. You can then generate a DOI for that release using a permanent documentation service such as Zenodo [61] and make it available to reviewers and beyond.

Share your workflow and analysis code

Sharing your workflow code with collaborators, peer reviewers, and scientists seeking to use a similar method can foster discussion and review of your analysis. Sticking to a clear documentation strategy, using a version control system, and packaging your code in notebooks or as a workflow prepare them to be easily shared with others. To go a step further, you can package your code with tools like Binder, ReproZip, or Whole Tale, or make interactive visualizations with tools like Shiny apps or Plotly. These approaches let others run the code on cloud computers in environments identical to those in which the original computation was performed (Figs 4 and 7) [58, 62, 63]. These tools substantially reduce overhead associated with interacting with code and data and, in doing so, make it fast and easy to rerun portions of the analysis, check accuracy, or even tweak the analysis to produce new results. If you also share your code and workflows publicly, you will also help contribute to the growing resources for open workflow–enabled biological research.

Getting Started Developing Workflows

In our experience, the best way to have your workflow system work for you is to include as much of your analysis as possible within the automated workflow framework, use self-documenting names, include analysis visualizations, and keep rigorous documentation alongside your workflow that enables you to understand each decision and entirely reproduce any manual steps. Some of the tools discussed above will inevitably
```

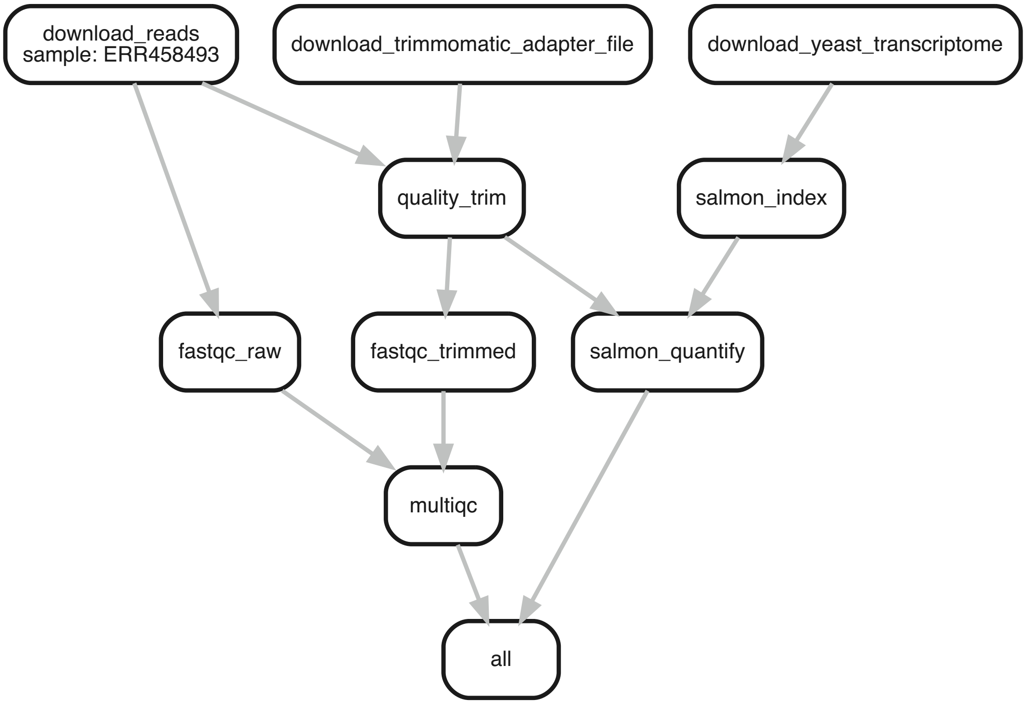


**Figure 5:** A directed acyclic graph (DAG) that illustrates connections between all steps of a sequencing data analysis workflow. Each box represents a step in the workflow, while lines connect sequential steps. The DAG shown in this figure illustrates a real bioinformatics workflow for RNA-seq quantification that was generated by modifying the default Snakemake workflow DAG. This example of an initial workflow used only to quality control and then quantify 1 FASTQ file against a transcriptome more than doubles the amount of files in a project. When the number of steps are expanded to carry out a full research analysis and the number of initial input files are increased, a workflow can generate hundreds to thousands of intermediate files. Fortunately, workflow system coordination alleviates the need for a user to directly manage file interdependencies. For a larger analysis DAG, see [60].

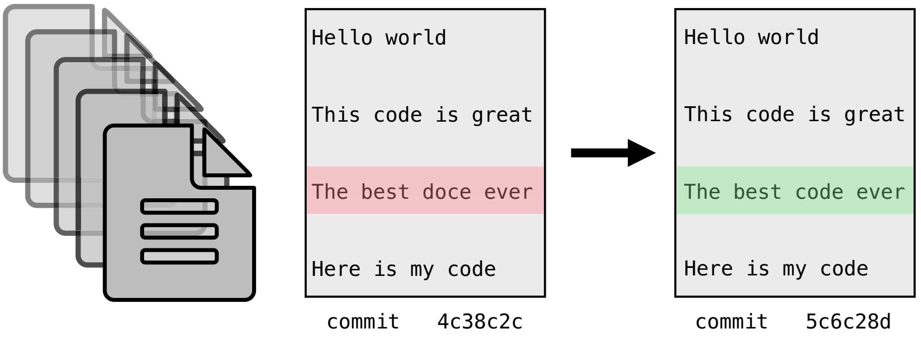
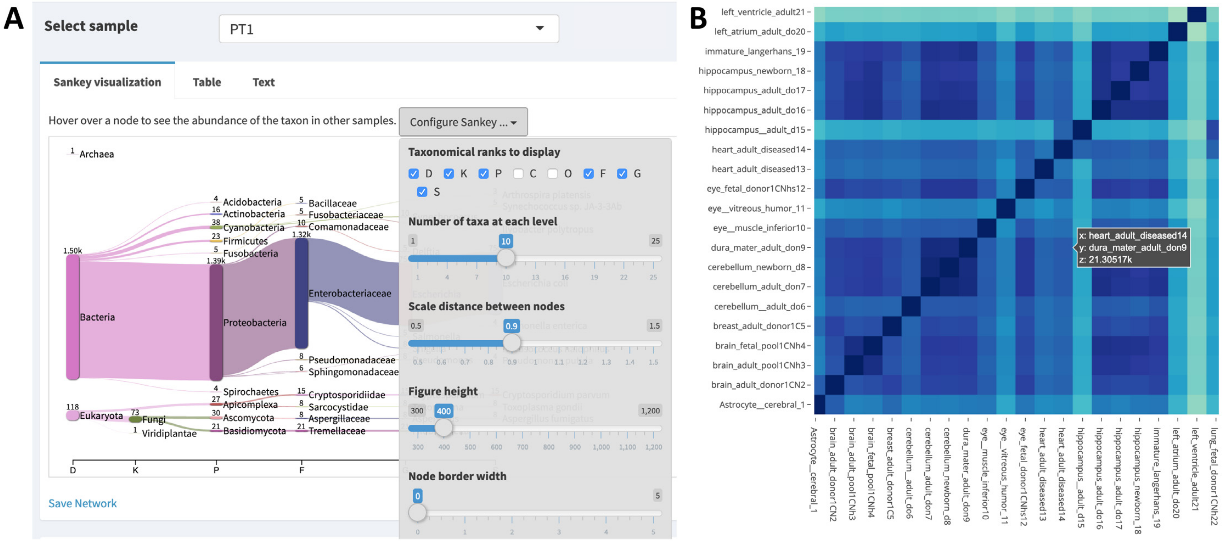


**Figure 6:** Version control systems (e.g., Git, Mercurial) work by storing incremental differences in files from 1 saved version (“commit”) to the next. To visualize the differences between each version, text editors such as Atom and online services such as GitHub, GitLab, and Bitbucket use red highlighting to denote deletions and green highlighting to denote additions. In this trivial example, a typographical error in version 1 (in pink) was corrected in version 2 (in green). These systems are extremely useful for code and manuscript development because it is possible to return to the snapshot of any saved version. This means that version control systems save you from accidental deletions, preserve code that you thought you no longer needed, and preserve a record of project changes over time.

change over time, but these principles apply broadly and will help you design clear, well-documented, and reproducible analyses. Ultimately, you will need to experiment with strategies that work for you—what is most important is to develop a clear set of strategies and implement them tenaciously. Below, we provide a few practical strategies to try as you begin developing your own workflows.

## Start with working code

When building a workflow for the first time, start from working examples provided as part of the tool documentation or otherwise available online. This functioning example code then provides a reliable workflow framework free of syntax errors that you can customize for your data without the overhead of generating correct workflow syntax from scratch. Be sure to run this analysis on provided test data, if available, to ensure that the tools and command-line syntax function at a basic level. Table 1 provides links to official repositories containing tutorials and example biological analysis workflows, and workflow tutorials and code-sharing websites such as GitHub, GitLab, and Bitbucket have many publicly available workflows for other analyses. If a workflow is available through Binder, you can test and experiment with workflow modification on



Select sample: PT1

Sankey visualization | Table | Text

Hover over a node to see the abundance of the taxon in other samples.

Configure Sankey ...

Taxonomical ranks to display

☑ D ☑ K ☑ P □ C □ O ☑ F ☑ G ☑ S

Number of taxa at each level: 10

Scale distance between nodes: 0.9

Figure height: 400

Node border width: 0

Save Network

x: heart_adult_diseased14
y: dura_mater_adult_don9
z: 21.30517k

**Figure 7:** Interactive visualizations facilitate sharing and repeatability. A. Interactive visualization dashboard in the Pavian Shiny app for metagenomic analysis [64, 65]. Shiny allows you to build interactive web pages using R code. Data are manipulated by R code in real time in a web page, producing analysis and visualizations of a dataset. Shiny apps can contain user-specifiable parameters, allowing a user to control visualizations or analyses. In (A), sample PT1 is selected, and taxonomic ranks class and order are excluded. Shiny apps allow collaborators who may or may not know R to modify R visualizations to fit their interests. B. Plotly heat map of transcriptional profiling in human brain samples [66]. Hovering over a cell in the heat map displays the sample names from the x and y axis, as well as the intensity value. Plotting tools such as plotly and vega-lite produce single interactive plots that can be shared with collaborators or integrated into websites [67, 68]. Interactive visualizations are also helpful in exploratory data analysis.

Binder's cloud system without needing to install a workflow manager or software management tool on your local compute system [58].

## Test with subsampled data

Once you have working workflow syntax, test the step on your own data or public data related to your species or condition of interest. First, create a subsampled dataset that you can use to test your entire analysis workflow. This set will save time, energy, and computational resources throughout workflow development. If working with FASTQ data, a straightforward way to generate a small test set is to subsample the first million lines of a file (first 250,000 reads) as follows: head -n 1000000 FASTQ_FILE.fq > test_fastq.fq.

While there are many more sophisticated ways to subsample reads, this technique should be sufficient for testing each step of most workflows prior to running your full dataset. In specific cases, such as eukaryotic genome assembly, you may need to be more intentional with how you subsample reads and how much sample data you use as a test set.

## Document your process

Document your changes, explorations, and errors as you develop. We recommend using the Markdown language so your documentation is in plain text (to facilitate version control) but can still include helpful visual headings, code formatting, and embedded images. Markdown editors with visual previewing, such as HackMD, can greatly facilitate notetaking, and Markdown documents are visually rendered properly within your online version control backups on services such as GitHub [69].

## Develop your workflow

From your working code, iteratively modify and add workflow steps to meet your data analysis needs. This strategy allows you to find and fix mistakes on small sections of the workflow. Periodically clean your output directory and rerun the entire workflow to ensure that all steps are fully interoperable (using small test data will improve the efficiency of this step). If possible, using mock or control datasets can help you verify that the analysis you are building actually returns correct biological results. Tutorials and tool documentation are useful companions during development; as with any language, remembering workflow-specific syntax takes time and practice.

## Assess your results

Evaluate your workflow results as you go. Consider what aspects (e.g., tool choice, program parameters) can be evaluated rigorously, and assess each step for expected behavior. Other aspects (e.g., filtering metadata, joining results across programs or analysis, software and workflow bugs) will be more difficult to evaluate. Wherever possible, set up positive and negative controls to ensure that your analysis is being performed as desired. Once you are certain that an analysis is executing as designed, tracking down unusual results may reveal interesting biological differences.

## Back up early and often

As you write new code, back up your changes in an online repository such as GitHub, GitLab, or Bitbucket. These services support both drag-and-drop and command-line interaction.

## Scale up your workflow

Bioinformatics tools vary in the resources that they require: some analysis steps are compute-intensive, other steps are memory intensive, and still others will have large intermediate storage needs. If using a high-performance computing system or the cloud, you will need to request resources for running your pipeline, often provided as a simultaneous execution limit or purchased by your research group on a cost-per-compute basis. Workflow systems provide built-in tools to monitor resource usage for each step. Running a complete workflow on a single sample with resource monitoring enabled generates an estimate of computational resources needed for each step. These estimates can be used to set appropriate resource limits for each step when executing the workflow on your remaining samples.

### Find a community and ask for help when you need it

Local and online users' groups are helpful communities when learning a workflow language. When you are first learning, help from more advanced users can save you hours of frustration. After you have progressed, providing that same help to new users can help you cement the syntax in your mind and tackle more advanced uses. Data-centric workflow systems have been enthusiastically adopted by the open science community, and as a consequence, there is a critical mass of tutorials and open access code, as well as code discussion on forums and via social media, particularly Twitter. Post in the relevant workflow forums when you have hit a stopping point that you are unable to work through. Be respectful of people's time and energy and be sure to include appropriate details important to your problem (see Strategies for troubleshooting section).

## Data and Resource Management for Workflow-Enabled Biology

Advancements in sequencing technologies have greatly increased the volume of data available for biological query [70]. Workflow systems, by virtue of automating many of the time-intensive project management steps traditionally required for data-intensive biology, can increase our capacity for data analysis. However, conducting biological analyses at this scale requires a coordinated approach to data and computational resource management. Below, we provide recommendations for data acquisition, management, and quality control that have become especially important as the volume of data has increased. Finally, we discuss securing and managing appropriate computational resources for the scale of your project.

### Managing large-scale datasets

Experimental design, finding or generating data, and quality control are quintessential parts of data-intensive biology. There is no substitute for taking the time to properly design your analysis, identify appropriate data, and conduct sanity checks on your files. While these tasks are not automatable, many tools and databases can aid in these processes.

### Look for appropriate publicly available data

With vast amounts of sequencing data already available in public repositories, it is often possible to begin investigating your research question by seeking out publicly available data. In some cases, these data will be sufficient to conduct your entire analysis. In other cases, particularly for biologists conducting novel experiments, these data can inform decisions about sequencing type, depth, and replication and can help uncover potential pitfalls before they cost valuable time and resources.

**Table 2:** References for experimental design and considerations for common sequencing chemistries

| Sequencing type | Resources |
|---|---|
| RNA-seq | [44, 83, 84] |
| Metagenomic sequencing | [45, 85, 86] |
| Amplicon sequencing | [87–89] |
| Microbial isolate sequencing | [90] |
| Eukaryotic genome sequencing | [91–94] |
| Whole-genome resequencing | [95] |
| RAD-seq | [96–100] |
| single-cell RNA-seq | [101, 102] |

Most journals now require data for all manuscripts to be made accessible, either at publication or after a short moratorium. Furthermore, the FAIR data movement has improved the data-sharing ecosystem for data-intensive biology [71–77]. You can find relevant sequencing data either by starting from the "data accessibility" sections of papers relevant to your research or by directly searching for your organism, environment, or treatment of choice in public data portals and repositories. The International Nucleotide Sequence Database Collaboration (INSDC), which includes the SRA, European Nucleotide Archive (ENA), and DNA DataBank of Japan (DDBJ), is the largest repository for raw sequencing data but no longer accepts sequencing data from large consortium projects [78]. These data are instead hosted in consortium-specific databases, which may require some domain-specific knowledge for identifying relevant datasets and have unique download and authentication protocols. For example, raw data from the Tara Oceans expedition is hosted by the Tara Ocean Foundation [79]. Additional curated databases focus on processed data instead, such as gene expression in the Gene Expression Omnibus (GEO) [80]. Organism-specific databases such as Wormbase (*Caenorhabditis elegans*) specialize in curating and integrating sequencing and other data associated with a model organism [81]. Finally, rather than focusing on certain data types or organisms, some repositories are designed to hold any data and metadata associated with a specific project or manuscript (e.g., Open Science Framework, Dryad, Zenodo [82]).

### Consider analysis when generating your own data

If you are generating your own data, proper experimental design and planning are essential. For cost-intensive sequencing data, there are a range of decisions about experimental design and sequencing (including sequencing type, sequencing depth per sample, and biological replication) that affect your ability to properly address your research question. Conducting discussions with experienced bioinformaticians and statisticians, prior to beginning your experiments if possible, is the best way to ensure that you will have sufficient statistical power to detect effects. These considerations will be different for different types of sequence analysis. To aid in early project planning, we have curated a series of domain-specific references that may be useful as you go about designing your experiment (see Table 2). Given the resources invested in collecting samples for sequencing, it is important to build in a buffer to preserve your experimental design in the face of unexpected laboratory or technical issues. Once generated, it is always a good idea to have multiple independent backups of raw sequencing data, as they typically cannot be easily regenerated if lost to computer failure or other unforeseeable events.

As your experiment progresses, keep track of as much information as possible: dates and times of sample collection, storage, and extraction; sample names; aberrations that occurred during collection; kit lot used for extraction; and any other sample and sequencing measurements you might be able to obtain (e.g., temperature, location, metabolite concentration, name of collector, well number, plate number, machine on which your data were sequenced). These metadata allow you to keep track of your samples, to control for batch effects that may arise from unintended batching during sampling or experimental procedures, and make the data you collect reusable for future applications and analysis by yourself and others. Wherever possible, follow the standard guidelines for formatting metadata for scientific computing to limit downstream processing and simplify analyses requiring these metadata (see [10]). We have focused here on sequencing data; for data management over long-term ecological studies, we recommend [103].

## Getting Started with Sequencing Data

### Protect valuable data

Aside from the code itself, raw data are the most important files associated with a workflow because they cannot be regenerated if accidentally altered or deleted. Keeping a read-only copy of raw data alongside a workflow, as well multiple backups, protects your data from accidents and computer failure. This also removes the imperative of storing intermediate files because these can be easily regenerated by the workflow.

When sharing or storing files and results, data version control can keep track of differences in files such as changes from tool parameters or versions. The version control tools discussed in the Workflow-Based Project Management section are primarily designed to handle small files, but GitHub provides support for Git Large File Storage, and repositories such as the Open Science Framework, Figshare, Zenodo, and Dryad can be used for storing larger files and datasets [61, 82, 104–106].

In addition to providing version control for projects and datasets, these tools also facilitate sharing and attribution by enabling generation of DOIs for datasets, figures, presentations, code, and preprints. Because free tools often limit the size of files that can be stored, a number of cloud backup and storage services are also available for purchase or via university contract, including Google Drive, Box, Dropbox, Amazon Web Services, and Backblaze. Full computer backups can be conducted to these storage locations with tools like rclone [107].

### Ensure data integrity during transfers

If you are working with publicly available data, you may be able to work on a compute system where the data are already available, circumventing time and effort required for downloading and moving the data. Databases such as the SRA are now available on commercial cloud computing systems, and open source projects such as Galaxy enable working with SRA sequence files directly from a web browser [12, 108]. Ongoing projects such as the NIH Common Fund Data Ecosystem aim to develop a data portal to make NIH Common Fund data, including biomedical sequencing data, more FAIR.

In most cases, you will still need to transfer some data—either downloading raw data or transferring important intermediate and results files for backup and sharing (or both). Transferring compressed files (e.g., gzip, bzip2, BAM/CRAM) can improve transfer speed and save space, and checksums can be used to to ensure file integrity after transfer (see Fig. 8).

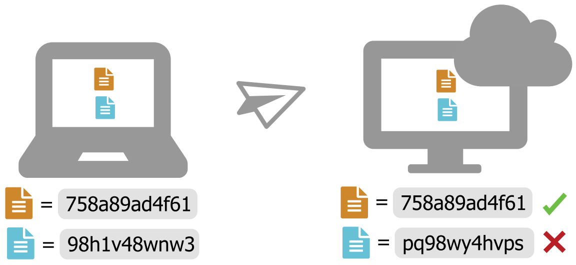


**Figure 8:** Use Checksums to ensure file integrity. Checksum programs (e.g., md5, sha256) encode file size and content in a single value known as a "checksum." For any given file, this value will be identical across platforms when calculated using the same checksum program. When transferring files, calculate the value of the checksum prior to transfer, and then again after transfer. If the value is not identical, there was an error introduced during transfer (e.g., file truncation). Checksums are often provided alongside publicly available files so that you can verify proper download. Tools like rsync and rclone that automate file transfers use checksums internally to verify that files were transferred properly, and some GUI file transfer tools (e.g., Cyberduck [109]) can assess checksums when they are provided [107]. If you generated your own data and receieved sequencing files from a sequencing center, be certain that you also receive a checksum for each of your files to ensure that they download properly.

### Perform quality control at every step

The quality of your input data has a major effect on the quality of the output results, no matter whether your workflow analyzes 6 samples or 600. Assessing data at every analysis step can reveal problems and errors early, before they waste valuable time and resources. Using quality control tools that provide metrics and visualizations can help you assess your datasets, particularly as the size of your input data scales up. However, data from different species or sequencing types can produce anomalous quality control results. You are ultimately the single most effective quality control tool that you have, so it is important to critically assess each metric to determine those that are relevant for your particular data.

#### *Look at your files*

Quality control can be as simple as looking at the first few and last few lines of input and output data files, or checking the size of those files (see Table 3). To develop an intuition for what proper inputs and outputs look like for a given tool, it is often helpful to first run the test example or data that is packaged with the software. Comparing these input and output file formats to your own data can help identify and address inconsistencies.

#### *Visualize your data*

Visualization is another powerful way to pick out unusual or unexpected patterns. Although large abnormalities may be clear from looking at files, others may be small and difficult to find. Visualizing raw sequencing data with FastQC (Fig. 9A) and processed sequencing data with tools like the Integrative Genome Viewer and plotting tabular results files using Python or R can make aberrant or inconsistent results easier to track down [110, 111].

#### *Pay attention to warnings and log files*

Many tools generate log files or messages while running. These files contain information about the quantity, quality, and results

**Table 3:** Some commands to quickly explore the contents of a file

| Command | Function | Example |
|---|---|---|
| ls -lh | List files with information in a human-readable format | ls -lh ∗fastq.gz |
| head | Print the first 6 lines of a file to standard out | head samples.csv |
| tail | Print the last 6 lines of a file to standard out | tail samples.csv |
| less | Show the contents of a file in a scrollable screen | less samples.csv |
| zless | Show the contents of a gzipped file in a scrollable screen | zless sample1.fastq.gz |
| wc -l | Count the number of lines in a file | wc -l ecoli.fasta |
| cat | Print a file to standard out | cat samples.csv |
| grep | Find matching text and print the line to standard out | grep ">" ecoli.fasta |
| cut | Cut columns from a table | cut -d"," -f1 samples.csv |

These commands can be used on Unix and Linux operating systems to detect common formatting problems or other abnormalities.

from the run, or error messages about why a run failed. Inspecting these files can be helpful to make sure tools ran properly and consistently or to debug failed runs. Parsing and visualizing log files with a tool like MultiQC can improve the interpretability of program-specific log files (Fig. 9 [113]).

#### *Look for common biases in sequencing data*

Biases in sequencing data originate from experimental design, methodology, sequencing chemistry, or workflows and are helpful to target specifically with quality control measures. The exact biases in a specific dataset or workflow will vary greatly between experiments, so it is important to understand the sequencing method that you have chosen and incorporate appropriate filtration steps into your workflow. For example, PCR duplicates can cause problems in libraries that underwent an amplification step, and often need to be removed prior to downstream analysis [114–118].

#### *Check for contamination*

Contamination can arise during sample collection, nucleotide extraction, library preparation, or through sequencing spike-ins like PhiX, and could change data interpretation if not removed [119–121]. Libraries sequenced with high concentrations of free adapters or with low-concentration samples may have increased barcode hopping, leading to contamination between samples [122].

#### *Consider the costs and benefits of stringent quality control for your data*

High-quality data are essential for good downstream analysis. However, stringent quality control can sometimes do more harm than good. For example, depending on sequencing depth, stringent quality trimming of RNA-seq data may reduce isoform discovery [123]. To determine what issues are most likely to plague your specific dataset, it can be helpful to find recent publications using a similar experimental design or to speak with experts at a sequencing core.

Because sequencing data and applications are so diverse, there is no one-size-fits-all solution for quality control. It is important to think critically about the patterns that you expect to see given your data and your biological problem, and consult with technical experts whenever possible.

## Securing and Managing Appropriate Computational Resources

Sequence analysis requires access to computing systems with adequate storage and analysis power for your data. For some smaller-scale datasets, local desktop or even laptop systems can be sufficient, especially if using tools that implement data reduction strategies such as minhashing [124]. However, larger projects require additional computing power or may be restricted to certain operating systems (e.g., linux). For these projects, solutions range from research-focused high-performance computing systems to research-integrated commercial analysis platforms. Both research-only and commercial clusters provide avenues for research and educational proposals to enable access to their computing resources (see Table 4). In preparing for data analysis, be sure to allocate sufficient computational resources and funding for storage and analysis, including large intermediate files and resources required for personnel training. Note that workflow systems can greatly facilitate faithful execution of your analysis across the range of computational resources available to you, including distribution across cloud computing systems.

## Getting Started with Resource Management

As the scale of data increases, the resources required for analysis can balloon. Bioinformatics workflows can be long-running, require high-memory systems, or involve intensive file manipulation. Some of the strategies below may help you manage computational resources for your project.

### Apply for research units if eligible

There are a number of cloud computing services that offer grants providing computing resources to data-intensive researchers (Table 4). In some cases, the resources provided may be sufficient to cover your entire analysis.

### Develop on a local computer when possible

Because workflows transfer easily across systems, it can be useful to develop individual analysis steps on a local laptop. If the analysis tool will run on your local system, test the step with subsampled data, such as those created in the Getting Started Developing Workflows section. Once working, the new work-

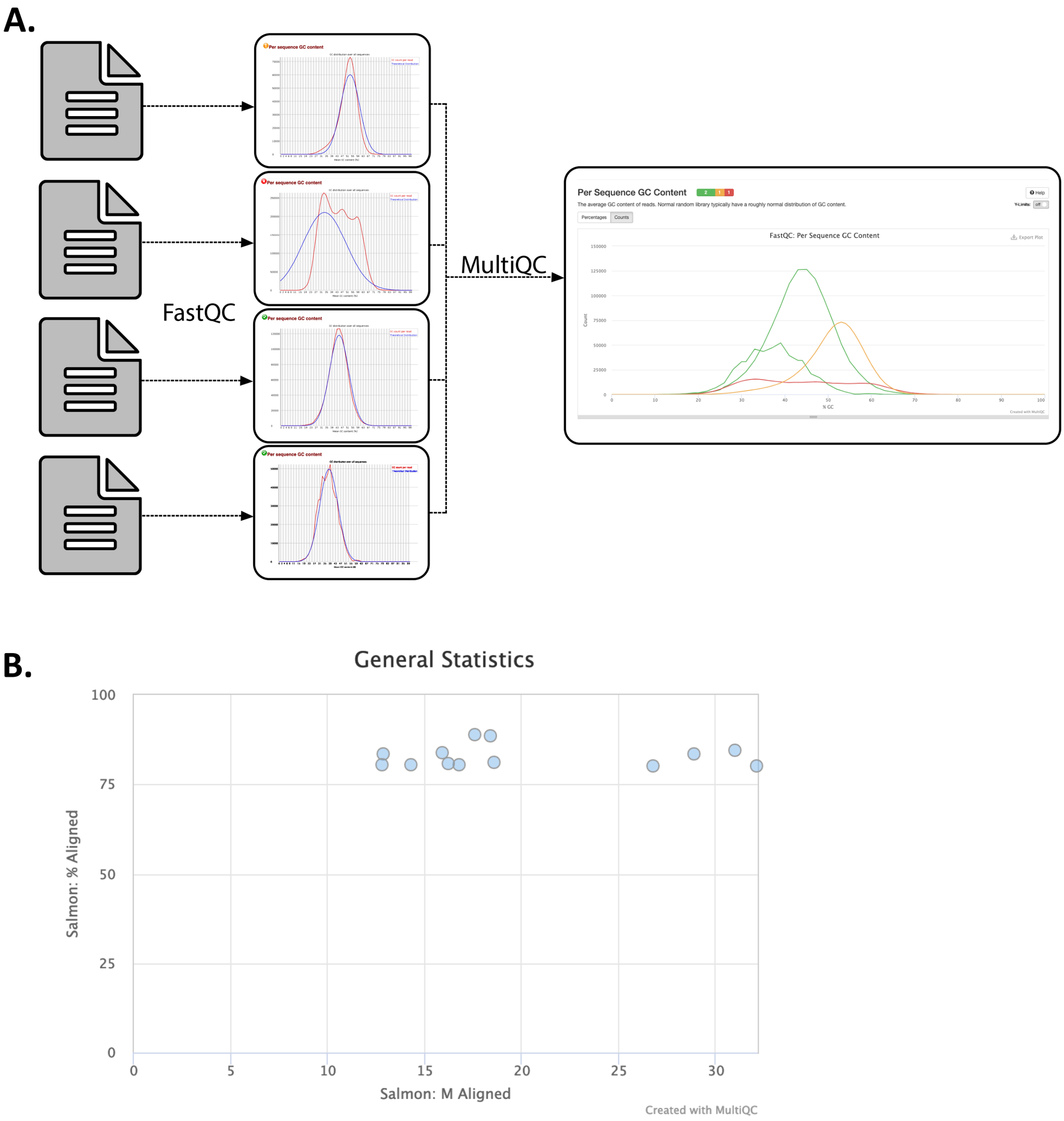


**Figure 9:** Visualizations produced by MultiQC. MultiQC finds and automatically parses log files from other tools and generates a combined report and parsed data tables that include all samples. MultiQC currently supports 88 tools. A. MultiQC summary of FastQC Per Sequence GC Content for 1,905 metagenome samples. FastQC provides quality control measurements and visualizations for raw sequencing data from a single sample and is a near-universal first step in sequencing data analysis because of the insights that it provides [110, 111]. FastQC measures and summarizes 10 quality metrics and provides recommendations for whether an individual sample is within an acceptable quality range. Not all metrics readily apply to all sequencing data types. For example, while multiple GC peaks might be concerning in whole-genome sequencing of a bacterial isolate, we would expect a non-normal distribution for some metagenome samples that contain organisms with diverse GC content. Samples like this can be seen in red in this figure. B. MultiQC summary of Salmon quant reads mapped per sample for RNA-seq samples [112]. In this figure, we see that MultiQC summarizes the number of reads mapped and percent of reads mapped, 2 values that are reported in the Salmon log files.

flow component can be run at scale on a larger computing system. Workflow system tool resource usage reporting can help determine the increased resources needed to execute the workflow on larger systems. For researchers without access to free or granted computing resources, this strategy can save significant cost.

## Gain quick insights using sketching algorithms

Understanding the basic structure of data, the relationship between samples, and the approximate composition of each sample can helpful at the beginning of data analysis and can often drive analysis decisions in different directions than those originally intended. Although most bioinformatics workflows gen-

**Table 4:** Computing resources for bioinformatics projects

| Provider | Access Model | Restrictions |
|---|---|---|
| Amazon Web Services | Paid | |
| Bionimbus Protected Data Cloud | Research allocation | Users with eRA commons account |
| Cyverse Atmosphere | Free with limits | Storage and compute hours |
| EGI federated cloud | Access by contact | European partner countries |
| Galaxy | Free with storage limits | Data storage limits |
| Google Cloud Platform | Paid | |
| Google Colab | Free | Computational notebooks, no resource guarantees |
| Microsoft Azure | Paid | |
| NSF XSEDE | Research allocation | USA researchers or collaborators |
| Open Science Data Cloud | Research allocation | |
| Wasabi | Paid | Data storage solution only |

Bioinformatics projects often require additional computing resources. If a local or university-run high-performance computing cluster is not available, computing resources are available via a number of grant-based or commercial providers.

erate these types of insights, there are a few tools that do so rapidly, allowing the user to generate quick hypotheses that can be further tested by more extensive, fine-grained analyses. Sketching algorithms work with compressed approximate representations of sequencing data and thereby reduce runtimes and computational resource use. These approximate representations retain enough information about the original sequence to recapitulate the main findings from many exact but computationally intensive workflows. Most sketching algorithms estimate sequence similarity in some way, allowing insights to be gained from these comparisons. For example, sketching algorithms can be used to estimate all-by-all sample similarity, which can be visualized as a principal component analysis or a multidimensional scaling plot, or can be used to build a phylogenetic tree with accurate topology. Sketching algorithms also dramatically reduce the runtime for comparisons against databases (e.g., all of GenBank), allowing users to quickly compare their data against large public databases.

Rowe [125] reviewed programs and genomic use cases for sketching algorithms and provided a series of tutorial workbooks (e.g., Sample QC notebook: [126]).

### Use the right tools for your question

RNA-seq analysis approaches like differential expression or transcript clustering rely on transcript or gene counts. Many tools can be used to generate these counts by quantifying the number of reads that overlap with each transcript or gene. For example, tools such as STAR and HISAT2 produce alignments that can be post-processed to generate per-transcript read counts [127, 128]. However, these tools generate information-rich output, specifying per-base alignments for each read. If you are only interested in read quantification, quasi-mapping tools provide the desired results while reducing the time and resources needed to generate and store read count information [129, 130].

### Seek help when you need it

In some cases, you may find that your accessible computing system is ill equipped to handle the type or scope of your analysis. Depending on the system, staff members may be able to help direct you to properly scale your workflow to available resources, or guide you in tailoring computational unit allocations or purchases to match your needs.

## Strategies for Troubleshooting

Workflows, and research software in general, invariably require troubleshooting and iteration. When first starting with a workflow system, it can be difficult to interpret code and usage errors from unfamiliar tools or languages [2]. Furthermore, the iterative development process of research software means that functionality may change, new features may be added, or documentation may be out of date [131]. The challenges of learning and interacting with research software necessitate time and patience [6].

One of the largest barriers to surmounting these challenges is learning how, when, and where to ask for help. Below we outline a strategy for troubleshooting that can help build your own knowledge while respecting both your own time and that of research software developers and the larger bioinformatics community. In the “Where to seek help” section, we also recommend locations for asking general questions around data-intensive analysis, including discussion of tool choice, parameter selection, and other analysis strategies. Beyond these tips, workshops and materials from training organizations such as the Carpentries, R-Ladies, and RStudio can arm you with the tools you need to start troubleshooting and jump-start software and data literacy in your community [132]. Getting involved with these workshops and communities provides not only educational benefits but also networking and career-building opportunities.

### How to help yourself: try to pinpoint your issue or error

Software errors can be the result of syntax errors, dependency issues, operating system conflicts, bugs in the software, problems with the input data, and many other issues. Running the software on the provided test data can help narrow the scope of error sources: if the test data successfully run, the command is likely free of syntax errors, the source code is functioning, and the tool is likely interacting appropriately with dependencies and the operating system. If the test data run but the tool still produces an error when run with your data and parameters, the error message can be helpful in discovering the cause of the error. In many cases, the error that you have encountered has been encountered many times before, and searching for the error online can turn up a working solution. If there is a software issue tracker for the software (e.g., on the GitHub, GitLab, or Bitbucket repository), or a Gitter, Slack, or Google Groups page, performing a targeted search with the error message may provide additional context or a solution for the error. If targeted searches do not return results, Googling the error message with the program name is a good next step. Searching with several variants and iteratively adding information such as the type of input data, the name of the coding language or computational platform, or other relevant information can improve the likelihood that there will be a match. There are a vast array of online resources for bioinformatics help, ranging from question sites such as Stack Overflow and BioStars to personal or academic blogs and even tutorials and lessons written by experts in the field [133]. This increases the discoverability of error messages and their solutions.

Sometimes, programs fail without outputting an error message. In cases like these, the software's help (usually accessible on the command line via tool-name –help) and official documentation may provide clues or additional example use cases that may be helpful in resolving an error. Syntax errors are extremely common, and typos as small as a single, misplaced character or amount of whitespace can affect the code. If a command matches the documentation and appears syntactically correct, the software version (often accessible at the command line via tool-name–version) may be causing the error.

Best practices for software development follow "semantic versioning" principles, which aim to keep the arguments and functionality the same for all minor releases of the program (e.g., version change from 1.1 to 1.2) and only change functions with major releases (e.g., 1.x to 2.0).

### How to seek help: include the right details with your question

When searching for the error message and reading the documentation do not resolve an error, it is usually appropriate to seek help either from the software developers or from a bioinformatics community. When asking for help, it is essential to provide the right details so that other users and developers can understand the exact conditions that produced the error. At minimum, include the name and version of the program, the method used to install it, whether the test data ran, the exact code that produced the error, the error message, and the full output text from the run (if any is produced). The type and version of the operating system you are using is also helpful to include. Sometimes, this is enough information for others to spot the error. However, if it appears that there may be a bug in the underlying code, specifying or providing the minimum amount of data required to reproduce the error (e.g., reproducible example [134, 135]) enables others to reproduce and potentially solve the error at hand. Putting the effort into gathering this information both increases your own understanding of the problem and makes it easier and faster for others to help solve your issue. Furthermore, it signals respect for the time that these developers and community members dedicate to helping troubleshoot and solve user issues.

### Where to seek help: online and local communities of practice

Online communities and forums are a rich source of archived bioinformatics errors with many helpful community members. For errors with specific programs, often the best place to post is the developers' preferred location for answering questions and solving errors related to their program. For open source programs on GitHub, GitLab, or Bitbucket, this is often the "Issues" tab within the software repository, but it could alternatively be a Google Groups list, Gitter page, or other specified forum. Usually, the documentation indicates the best place to ask questions. If question is more general, such as asking about program choice or workflows, forums relevant to your field such as Stack Overflow, BioStars, or SEQanswers are good choices because posts here are often seen by a large community of researchers. Before posting, search through related topics to double-check that the question has not already been answered. As more research software development and troubleshooting is happening openly in online repositories, it is becoming more important than ever to follow a code of conduct that promotes an open and harassment-free discussion environment [136]. Look for codes of conduct in the online forums you participate in, and make sure that you do your part to help ensure a welcoming community for participants of all backgrounds and computational competencies.

While there is lots of help available online, there is no substitute for local communities. Local communities may come in the form of a tech meetup, a users' group, a hacky hour, or an informal meetup of researchers using similar tools. While this may seem like just a local version of Stack Overflow, the local, member-only nature can help create a safe and collaborative online space for troubleshooting problems often encountered by your local bioinformatics community. The benefit to beginners is clear: learning the best way to post questions and the important parts of errors, while getting questions answered so they can move forward in their research. Intermediate users may actually find these communities most useful because they can also accelerate their own troubleshooting skills by helping others solve issues that they have already struggled through. While it can be helpful to have some experts available to help answer questions or to know when to escalate to Stack Overflow or other communities, a collaborative community of practice with members at all experience levels can help all its members move their science forward faster.

If such a community does not yet exist in your area, building this sort of community (discussed in detail in [137]) can be as simple as hosting a seminar series or starting meetup sessions for data analysis coworking. In our experience, it can also be useful to set up a local online forum (e.g., discourse) for group troubleshooting.

## Conclusion

Bioinformatics-focused workflow systems have reshaped data-intensive biology, reducing execution hurdles and empowering biologists to conduct reproducible analyses at the massive scale of data now available. Shared, interoperable research code is enabling biologists to spend less time rewriting common analysis steps and more time on interesting biological questions. We believe that these workflow systems will become increasingly important as dataset size and complexity continue to grow. This article provides a directed set of project, data, and resource management strategies for adopting workflow systems to facilitate and expedite reproducible biological research. While the included data management strategies are tailored to our own experiences in high-throughput sequencing analysis, we hope that these principles enable biologists both within and beyond our field to reap the benefits of workflow-enabled data-intensive biology.

## Data Availability

Not applicable.

## Abbreviations

DAG: directed acyclic graph; DOI: digital object identifier; FAIR: Findable, Accessible, Interoperable, and Reusable; GC: guanine-cytosine; GUI: graphical user interface; IDE: integrated development environment; NIH: National Institutes of Health; NSF: National Science Foundation; RAD-seq: restriction site–associated DNA sequencing; RNA-seq: RNA sequencing; SRA: Sequence Read Archive.

## Competing Interests

The authors declare that they have no competing interests.

## Funding

T.E.R., L.C.I., C.M.R., C.S.W., and C.T.B. were funded by Moore Foundation grant GBMF4551. S.E.J. was funded by State and Federal Water Contractors grant A19-1844. N.T.P. was funded by NSF grant 1711984.

## Authors’ Contributions

T.E.R.: Conceptualization; methodology; writing—original draft; writing—review and editing; visualization.
P.T.B.†: Methodology; writing—review and editing.
L.C.I.†: Methodology; writing—review and editing.
S.E.J.†: Methodology; visualization; writing—review and editing.
C.M.R.†: Methodology; writing—review and editing.
C.S.W.†: Methodology; writing—review and editing.
C.T.B.: Methodology; writing—review and editing; supervision; funding acquisition.
N.T.P.: Conceptualization; methodology; writing—original draft; writing—review and editing; visualization; supervision; funding acquisition.

## Acknowledgements

The authors thank all the members and affiliates of the Lab for Data-Intensive Biology at University of California Davis for providing valuable feedback on earlier versions of this manuscript and growing these practices alongside us. We also thank the Carpentries Community for fundamentally shaping many of the ideas and practices that we cover in this manuscript. This manuscript was written with manubot [138] is available in a GitHub repository [139].

## References

1. Ewels PA, Peltzer A, Fillinger S, et al. The nf-core framework for community-curated bioinformatics pipelines. Nat Biotechnol 2020;**38**(3):276–8.
2. Barone L, Williams J, Micklos D. Unmet needs for analyzing biological big data: A survey of 704 NSF principal investigators. PLoS Comput Biol 2017;**13**(10): e1005755.
3. Grüning B, Chilton J, Köster J, et al. Practical computational reproducibility in the life sciences. Cell Syst 2018;**6**(6):631–5.
4. Atkinson M, Gesing S, Montagnat J, et al. Scientific workflows: Past, present and future. Future Gener Comput Syst 2017;**75**:216–27.
5. Conery JS, Catchen JM, Lynch M. Rule-based workflow management for bioinformatics. VLDB J 2005;**14**:318–29.
6. Cereceda O, Quinn DEA. A graduate student perspective on overcoming barriers to interacting with open-source software. Facets (Ott.) 2020;**5**, doi:10.1139/facets-2019-0020.
7. Möller S, Prescott SW, Wirzenius L, et al. Robust cross-platform workflows: how technical and scientific communities collaborate to develop, test and share best practices for data analysis. *Data Sci Eng* 2017;**2**:232–44.
8. Wilson G, Aruliah DA, Titus Brown C, et al. Best practices for scientific computing. PLoS Biol 2014;**12**(1):e1001745.
9. Shade A, Teal TK. Computing workflows for biologists: a roadmap. PLoS Biol 2015;**13**(11):e1002303.
10. Wilson G, Bryan J, Cranston K. Good enough practices in scientific computing. PLoS Comput Biol 2017;**13**(6):e1005510.
11. Grüning B, Dale R, Sjödin A, et al. Bioconda: sustainable and comprehensive software distribution for the life sciences. Nat Methods 2018;**15**(7):475–6.
12. Afgan E, Baker D, Batut B, et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Res 2018;**46**(W1): W537–44.
13. Volchenboum SL, Cox SM, Heath A, et al. Data commons to support pediatric cancer research. Am Soc Clin Oncol Educ Book 2017;**37**:746–52.
14. Mitchell AL, Almeida A, Beracochea M, et al. MGnify: the microbiome analysis resource in 2020. Nucleic Acids Res 2019;**48**(D1):D570–8.
15. nf-core/rnaseq. GitHub. https://github.com/nf-core/rnaseq, accessed: 10/01/2020, doi:10.5281/zenodo.1400710.
16. Wurmus R, Uyar B, Osberg B, et al. PiGx: reproducible genomics analysis pipelines with GNU Guix. Gigascience 2018;**7**(12),doi:10.1093/gigascience/giy123.
17. metagenome-atlas/atlas. GitHub. https://github.com/metagenome-atlas/atlas, accessed: 10/01/2020.
18. Kieser S, Brown J, Zdobnov EM, et al. ATLAS: a Snakemake workflow for assembly, annotation, and genomic binning of metagenome sequence data. BMC Bioinformatics 2020;**21**, doi:10.1101/737528.
19. sunbeam-labs/sunbeam. GitHub. https://github.com/sunbeam-labs/sunbeam, accessed: 10/01/2020.
20. Clarke EL, Taylor LJ, Zhao C, et al. Sunbeam: an extensible pipeline for analyzing metagenomic sequencing experiments. Microbiome 2019;**7**(1):46.
21. dib-lab/dammit. GitHub. https://github.com/dib-lab/dammit, accessed: 10/01/2020, doi: 10.5281/zenodo.3569831.
22. dib-lab/elvers. GitHub. https://github.com/dib-lab/elvers, doi: 10.5281/zenodo.3345045, accessed: 10/01/2020.
23. Koster J, Rahmann S. Snakemake–a scalable bioinformatics workflow engine. Bioinformatics 2012;**28**(19):2520–2.
24. Mölder F, Jablonski KP, Letcher B, et al. Sustainable data analysis with Snakemake. Zenodo 2020. https://doi.org/ghdx9x.
25. Tommaso PDi, Chatzou M, Floden EW, et al. Nextflow enables reproducible computational workflows. Nat Biotechnol 2017;**35**(4):316–9.
26. Amstutz P, Crusoe MR, Tijanić N, et al. Common Workflow Language, v1.0. figshare 2016. https://doi.org/gf6ppg.
27. Terra.Bio. https://terra.bio, accessed: 10/01/2020.

28. Seven Bridges. The Seven Bridges Platform. https://www.sevenbridges.com/platform/, accessed: 10/01/2020.
29. Landau WM. The drake R package: a pipeline toolkit for reproducibility and high-performance computing. J Open Source Softw 2018;**3**(21):550.
30. https://snakemake.readthedocs.io/, accessed: 08/01/2020.
31. https://github.com/snakemake-workflows/chipseq, accessed: 08/01/2020.
32. https://snakemake.readthedocs.io/en/stable/tutorial/tutorial.html, accessed: 08/01/2020.
33. https://www.nextflow.io/, accessed: 08/01/2020.
34. https://github.com/nf-core/sarek, accessed: 08/01/2020.
35. https://www.nextflow.io/docs/latest/getstarted.html, accessed: 08/01/2020.
36. https://www.commonwl.org/, accessed: 08/01/2020.
37. https://github.com/EBI-Metagenomics/pipeline-v5, accessed: 08/01/2020.
38. https://www.commonwl.org/user_guide/02-1st-example/index.html, accessed: 08/01/2020.
39. https://openwdl.org/, accessed: 08/01/2020
40. https://github.com/gatk-workflows/gatk4-data-processing, accessed: 08/01/2020.
41. https://support.terra.bio/hc/en-us/articles/360037127992–1-howto-Write-your-first-WDL-script-running-GATK-HaplotypeCaller, accessed: 08/01/2020.
42. Strozzi F, Janssen R, Wurmus R, et al. Scalable workflows and reproducible data analysis for genomics. Methods Mol Biol 2019;**1910**:723–45.
43. Cokelaer T, Desvillechabrol D, Legendre R, et al. "Sequana": a Set of Snakemake NGS pipelines. *J Open Source Softw* 2017;**2**(16):352.
44. Conesa A, Madrigal P, Tarazona S, et al. A survey of best practices for RNA-seq data analysis. Genome Biol 2016;**17**:13.
45. Quince C, Walker AW, Simpson JT, et al. Shotgun metagenomics, from sampling to analysis. Nat Biotechnol 2017;**35**(9):833–44.
46. Luecken MD, Theis FJ. Current best practices in single-cell RNA-seq analysis: a tutorial. Mol Syst Biol 2019;**15**(6):e8746.
47. da Fonseca RR, Albrechtsen A, Themudo GE, et al. Next-generation biology: Sequencing and data analysis approaches for non-model organisms. Mar Genomics 2016;**30**:3–13.
48. Knight R, Vrbanac A, Taylor BC, et al. Best practices for analysing microbiomes. Nat Rev Microbiol 2018;**16**(7):410–22.
49. Kurtzer GM, Sochat V, Bauer MW. Singularity: Scientific containers for mobility of compute. PLoS One 2017;**12**(5):e0177459.
50. Merkel D, . Docker: lightweight Linux containers for consistent development and deployment. Linux J 2014, **239**, 2.
51. mamba-org/mamba. GitHub. https://github.com/mamba-org/mamba, accessed: 10/01/2020.
52. Bai J, Bandla C, Guo J, et al. BioContainers Registry: searching for bioinformatics tools, packages and containers. bioRxiv 2020, doi:10.1101/2020.07.21.187609.
53. dib-lab/dib-MMETSP. GitHub. https://github.com/dib-lab/dib-MMETSP, accessed: 12/20/2019.
54. Johnson LK, Alexander H, Titus Brown C. Re-assembly, quality evaluation, and annotation of 678 microbial eukaryotic reference transcriptomes. Gigascience 2019;**8**(4), doi:10.1093/gigascience/giy158.
55. R Markdown. https://rmarkdown.rstudio.com/, accessed: 10/01/2020.
56. IOS Press Ebooks - Jupyter Notebooks – a publishing format for reproducible computational workflows. http://ebooks.iospress.nl/publication/42900.
57. Tenaillon O, Barrick JE, Ribeck N, et al. Tempo and mode of genome evolution in a 50,000-generation experiment. Nature 2016;**536**(7615):165–70.
58. Jupyter P, Bussonnier M, Forde J, et al. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In: Proc. of the 17th Python in Science Conference (SciPy 2018). 2018, doi:10.25080/majora-4af1f417-011.
59. ngs-docs/2020-ggg-201b-rnaseq. GitHub. https://github.com/ngs-docs/2020-ggg-201b-rnaseq, accessed: 10/01/2020.
60. Titus Brown C, Moritz D, O'Brien MP, et al. Exploring neighborhoods in large metagenome assembly graphs using spacegraphcats reveals hidden sequence diversity. Genome Biol 2020;**21**(1):164.
61. Zenodo - Research. Shared. https://zenodo.org/, accessed: 10/01/2020.
62. Brinckman A, Chard K, Gaffney N, et al. Computing environments for reproducibility: Capturing the "Whole Tale." Future Gener Comput Syst 2019;**94**:854–67.
63. Chirigati F, Rampin R, Shasha D, et al. ReproZip: Computational reproducibility with ease. In: SIGMOD/PODS'16: International Conference on Management of Data, San Francisco, CA. New York, NY: ACM; 2016, doi:10.1145/2882903.2899401.
64. Pavian. https://fbreitwieser.shinyapps.io/pavian/, accessed: 10/01/2020.
65. Breitwieser FP, Salzberg SL. Pavian: interactive analysis of metagenomics data for microbiome studies and pathogen identification. Bioinformatics 2019;**36**(4):1303–4.
66. Visualizing Biological Data. https://plotly.com/python/v3/ipython-notebooks/bioinformatics/, accessed: 10/01/2020.
67. Plotly: The front end for ML and data science models. https://plotly.com/, accessed 10/01/2020.
68. Satyanarayan A, Moritz D, Wongsuphasawat K, et al. Vega-Lite: A grammar of interactive graphics. *IEEE Trans Visual Comput Graphics* 2017;**23**(1):341–50.
69. HackMD. HackMD - Collaborative Markdown Knowledge Base. https://hackmd.io, accessed: 10/01/2020.
70. SRA Documentation. https://www.ncbi.nlm.nih.gov/sra/docs/sragrowth/, accessed: 06/01/2020.
71. Wilkinson MD, Dumontier M, Aalbersberg IJJ, et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* 2016;**3**:160018.
72. Karsch-Mizrachi I, Takagi T, Cochrane G, on behalf of the International Nucleotide Sequence Database Collaboration. The International Nucleotide Sequence Database Collaboration. Nucleic Acids Res 2018;**46**(D1):D48–51.
73. Becker P, Bosschaerts M, Chaerle P, et al. Public microbial resource centers: key hubs for findable, accessible, interoperable, and reusable (FAIR) microorganisms and genetic materials. Appl Environ Microbiol 2019;**85**(21):e01444–19.
74. Alaux M, Rogers J, Letellier T, et al. Linking the International Wheat Genome Sequencing Consortium bread wheat reference genome sequence to wheat genetic and phenomic data. Genome Biol 2018;**19**(1):111.
75. Reiser L, Harper L, Freeling M, et al. FAIR: A call to make published data more findable, accessible, interoperable, and reusable. Mol Plant 2018;**11**(9):1105–8.

76. The Integrative HMP (iHMP) Research Network Consortium. The integrative human microbiome project. Nature 2019;**569**(7758):641–8.
77. El-Gebali S, Mistry J, Bateman A, et al. The Pfam protein families database in 2019. Nucleic Acids Res 2019;**47**(D1):D427–32.
78. Cochrane G, Karsch-Mizrachi I, Takagi T, International Nucleotide Sequence Database Collaboration. The International Nucleotide Sequence Database Collaboration. Nucleic Acids Res 2016;**44**(D1):D48–50.
79. Pesant S, Not F, Picheral M, et al. Open science resources for the discovery and analysis of Tara Oceans data. *Sci Data* 2015;**2**:150023.
80. Edgar R. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. Nucleic Acids Res 2002;**30**(1):207–10.
81. Harris TW, Arnaboldi V, Cain S, et al. WormBase: a modern Model Organism Information Resource. Nucleic Acids Res 2019;**48**(D1):D762–7.
82. Foster ED, Deardorff A. Open Science Framework (OSF). J Med Libr Assoc 2017;**105**, doi:10.5195/jmla.2017.88.
83. Schurch NJ, Schofield P, Gierliński M, et al. Erratum: How many biological replicates are needed in an RNA-seq experiment and which differential expression tool should you use? RNA 2016;**22**(10):1641.
84. Ching T, Huang S, Garmire LX. Power analysis and sample size estimation for RNA-Seq differential expression. RNA 2014;**20**(11):1684–96.
85. Knight R, Jansson J, Field D, et al. Unlocking the potential of metagenomics through replicated experimental design. Nat Biotechnol 2012;**30**(6):513–20.
86. Eisenhofer R, Minich JJ, Marotz C, et al. Contamination in low microbial biomass microbiome studies: issues and recommendations. Trends Microbiol 2019;**27**(2):105–17.
87. McLaren MR, Willis AD, Callahan BJ. Consistent and correctable bias in metagenomic sequencing experiments. eLife 2019;**8**:e46923.
88. Murray DC, Coghlan ML, Bunce M. From benchtop to desktop: important considerations when designing amplicon sequencing workflows. PLoS One 2015;**10**(4):e0124671.
89. Sinha R, Abu-Ali G, Vogtmann E, et al. Assessment of variation in microbial community amplicon sequencing by the Microbiome Quality Control (MBQC) project consortium. Nat Biotechnol 2017;**35**(11):1077–86.
90. Liao Yu-C, Lin S-H, Lin H-H. Completing bacterial genome assemblies: strategy and performance comparisons. Sci Rep 2015;**5**:8747.
91. Lewin HA, Robinson GE, Kress WJ, et al. Earth biogenome project: sequencing life for the future of life. Proc Natl Acad Sci USA 2018;**115**(17):4325–33.
92. Amarasinghe SL, Su S, Dong X, et al. Opportunities and challenges in long-read sequencing data analysis. Genome Biol 2020;**21**(1):30.
93. Zadesenets KS, Ershov NI, Rubtsov NB. Whole-genome sequencing of eukaryotes: From sequencing of DNA fragments to a genome assembly. Russ J Genet 2017;**53**:631–9.
94. Angel VDD, Hjerde E, Sterck L, et al. Ten steps to get started in Genome Assembly and Annotation. F1000Res 2018;**7**:ELIXIR–148.
95. Fuentes-Pardo AP, Ruzzante DE. Whole-genome sequencing approaches for conservation biology: Advantages, limitations and practical recommendations. Mol Ecol 2017;**26**(20):5369–406.
96. Shafer ABA, Peart CR, Tusso S, et al. Bioinformatic processing of RAD-seq data dramatically impacts downstream population genetic inference. Methods Ecol Evol 2017;**8**(8):907–17.
97. Díaz-Arce N, Rodríguez-Ezpeleta N. Selecting RAD-Seq data analysis parameters for population genetics: the more the better? *Front Genet* 2019;**10**:533.
98. Andrews KR, Good JM, Miller MR, et al. Harnessing the power of RADseq for ecological and evolutionary genomics. Nat Rev Genet 2016;**17**(2):81–92.
99. Catchen JM, Hohenlohe PA, Bernatchez L, et al. Unbroken: RADseq remains a powerful tool for understanding the genetics of adaptation in natural populations. Mol Ecol Resour 2017;**17**(3):362–5.
100. Lowry DB, Hoban S, Kelley JL, et al. Responsible RAD: Striving for best practices in population genomic studies of adaptation. Mol Ecol Resour 2017;**17**(3):366–9.
101. Bacher R, Kendziorski C. Design and computational analysis of single-cell RNA-sequencing experiments. Genome Biol 2016;**17**:63.
102. Haque A, Engel J, Teichmann SA, et al. A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications. Genome Med 2017;**9**(1):75.
103. Yenni GM, Christensen EM, Bledsoe EK, et al. Developing a modern data workflow for evolving data. bioRxiv 2018, doi:10.1101/344804.
104. Git Large File Storage. https://git-lfs.github.com/, accessed: 10/01/2020.
105. figshare - credit for all your research. https://figshare.com/, accessed: 10/01/2020.
106. Dryad Home - Publish and Preserve your Data. https://datadryad.org/stash, accessed: 10/01/2020.
107. Rclone syncs your files to cloud storage, https://rclone.org/, accessed: 06/01/2020.
108. SRA in the Cloud. https://www.ncbi.nlm.nih.gov/sra/docs/sra-cloud/, accessed: 10/01/2020.
109. Cyberduck | Libre server and cloud storage browser for Mac and Windows with support for FTP, SFTP, WebDAV, Amazon S3, OpenStack Swift, Backblaze B2, Microsoft Azure & OneDrive, Google Drive and Dropbox. https ://cyberduck.io/.
110. Babraham Bioinformatics - FastQC A Quality Control tool for High Throughput Sequence Data. https://www.bioinformatics.babraham.ac.uk/projects/fastqc/.
111. Thorvaldsdottir H, Robinson JT, Mesirov JP. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. Brief Bioinform 2013;**14**(2):178–92.
112. Patro R, Duggal G, Love MI, et al. Salmon provides fast and bias-aware quantification of transcript expression. Nat Methods 2017;**14**(4):417–9.
113. Ewels P, Magnusson M, Lundin S, et al. MultiQC: summarize analysis results for multiple tools and samples in a single report. Bioinformatics 2016;**32**(19):3047–8.
114. Meyer CA, Liu XS. Identifying and mitigating bias in next-generation sequencing methods for chromatin biology. Nat Rev Genet 2014;**15**(11):709–21.
115. Parekh S, Ziegenhain C, Vieth B, et al. The impact of amplification on differential expression analyses by RNA-seq. Sci Rep 2016;**6**:25533.
116. Schweyen H, Rozenberg A, Leese F. Detection and removal of PCR duplicates in population genomic ddRAD Studies by addition of a degenerate base region (DBR) in sequencing adapters. Biol Bull 2014;**227**(2):146–60. https://doi.org/f6q76c. DOI: 10.1086/bblv227n2p146 · PMID: 25411373.

117. Fu Yu, Wu P-H, Beane T, et al. Elimination of PCR duplicates in RNA-seq and small RNA-seq using unique molecular identifiers. BMC Genomics 2018;**19**(1):531.
118. Smith EN, Jepsen K, Khosroheidari M, et al. Biased estimates of clonal evolution and subclonal heterogeneity can arise from PCR duplicates in deep sequencing experiments. Genome Biol 2014;**15**(8):420.
119. Boothby TC, Tenlen JR, Smith FW, et al. Evidence for extensive horizontal gene transfer from the draft genome of a tardigrade. Proc Natl Acad Sci U S A 2015;**112**(52):15976–81.
120. Koutsovoulos G, Kumar S, Laetsch DR, et al. No evidence for extensive horizontal gene transfer in the genome of the tardigrade *Hypsibius dujardini*. Proc Natl Acad Sci U S A 2016;**113**(18):5053–8.
121. Mukherjee S, Huntemann M, Ivanova N, et al. Large-scale contamination of microbial isolate genomes by Illumina PhiX control. *Stand Genomic Sci* 2015;**10**:18.
122. Valk T, Vezzi F, Ormestad M, et al. Index hopping on the Illumina HiseqX platform and its consequences for ancient DNA studies. Mol Ecol Resour 2019;**20**(5):1171–81.
123. MacManes MD. On the optimal trimming of high-throughput mRNA sequence data. *Front Genet* 2014;**5**:13.
124. Rowe WPM, Carrieri AP, Alcon-Giner C, et al. Streaming histogram sketching for rapid microbiome analytics. Microbiome 2019;**7**(1):40.
125. Rowe WPM. When the levee breaks: a practical guide to sketching algorithms for processing the flood of genomic data. Genome Biol 2019;**20**(1):199.
126. will-rowe/genome-sketching. GitHub. https://github.com/will-rowe/genome-sketching, doi: 10.5281/zenodo.2637740.
127. Dobin A, Davis CA, Schlesinger F, et al. STAR: ultrafast universal RNA-seq aligner. Bioinformatics 2013;**29**(1):15–21.
128. Kim D, Paggi JM, Park C, et al. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. Nat Biotechnol 2019;**37**(8):907–15.
129. Bray NL, Pimentel H, Melsted P, et al. Near-optimal probabilistic RNA-seq quantification. Nat Biotechnol 2016;**34**(5):525–7.
130. Srivastava A, Sarkar H, Gupta N, et al. RapMap: a rapid, sensitive and accurate tool for mapping RNA-seq reads to transcriptomes. Bioinformatics 2016;**32**(12):i192–200.
131. Geiger RS, Varoquaux N, Mazel-Cabasse C, et al. The types, roles, and practices of documentation in data analytics open source software libraries. Comput Support Coop Work 2018;**27**:767–802.
132. Teal TK, Cranston KA, Lapp H, et al. Data carpentry: Workshops to increase data literacy for researchers. Int J Digit Curation 2015;**10**, doi:10.2218/ijdc.v10i1.351.
133. Parnell LD, Lindenbaum P, Shameer K, et al. BioStar: An online question & answer resource for the bioinformatics community. PLoS Comput Biol 2011;**7**(10):e1002216.
134. Stack Overflow. How to create a minimal, reproducible example. https://stackoverflow.com/help/minimal-reproducible-example.
135. RStudio Community. FAQ: How to do a minimal reproducible example (reprex) for beginners. 2020. https://community.rstudio.com/t/faq-how-to-do-a-minimal-reproducible-example-reprex-for-beginners/23061.
136. Tourani P, Adams B, Serebrenik A. Code of conduct in open source projects. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), Klagenfurt. 2017, doi:10.1109/saner.2017.7884606.
137. Stevens SLR, Kuzak M, Martinez C, et al. Building a local community of practice in scientific programming for life scientists. PLoS Biol 2018;**16**(11):e2005561.
138. Himmelstein(2019) PLOS Computational Biology e1007128, 10.1371/journal.pcbi.1007128, 1553-7358
139. Streamlining Data-Intensive Biology With Workflow Systems: Manubot repository. https://github.com/dib-lab/2020-workflows-paper/tree/09910b4e77280c979f332d6b11a8547fbb305d84, doi: 10.5281/zenodo.4276249.