EDUCATION

Using prototyping to choose a bioinformatics workflow management system

Michael Jackson[1]*, **Kostas Kavoussanakis**[1], **Edward W. J. Wallace**[2]*

1 EPCC, The University of Edinburgh, Edinburgh, United Kingdom, **2** Institute for Cell Biology and SynthSys, School of Biological Sciences, The University of Edinburgh, Edinburgh, United Kingdom

* m.jackson@epcc.ed.ac.uk (MJ); Edward.Wallace@ed.ac.uk (EWJW)

Abstract

Workflow management systems represent, manage, and execute multistep computational analyses and offer many benefits to bioinformaticians. They provide a common language for describing analysis workflows, contributing to reproducibility and to building libraries of reusable components. They can support both incremental build and re-entrancy—the ability to selectively re-execute parts of a workflow in the presence of additional inputs or changes in configuration and to resume execution from where a workflow previously stopped. Many workflow management systems enhance portability by supporting the use of containers, high-performance computing (HPC) systems, and clouds. Most importantly, workflow management systems allow bioinformaticians to delegate how their workflows are run to the workflow management system and its developers. This frees the bioinformaticians to focus on what these workflows should do, on their data analyses, and on their science.

RiboViz is a package to extract biological insight from ribosome profiling data to help advance understanding of protein synthesis. At the heart of RiboViz is an analysis workflow, implemented in a Python script. To conform to best practices for scientific computing which recommend the use of build tools to automate workflows and to reuse code instead of rewriting it, the authors reimplemented this workflow within a workflow management system. To select a workflow management system, a rapid survey of available systems was undertaken, and candidates were shortlisted: Snakemake, cwltool, Toil, and Nextflow. Each candidate was evaluated by quickly prototyping a subset of the RiboViz workflow, and Nextflow was chosen. The selection process took 10 person-days, a small cost for the assurance that Nextflow satisfied the authors' requirements. The use of prototyping can offer a low-cost way of making a more informed selection of software to use within projects, rather than relying solely upon reviews and recommendations by others.

Author summary

Data analysis involves many steps, as data are wrangled, processed, and analysed using a succession of unrelated software packages. Running the right steps, in the right order, and putting the right outputs in the right places, is a major source of frustration. Workflow management systems require that each data analysis step be "wrapped" in a structured

Citation: Jackson M, Kavoussanakis K, Wallace EWJ (2021) Using prototyping to choose a bioinformatics workflow management system. PLoS Comput Biol 17(2): e1008622. https://doi.org/10.1371/journal.pcbi.1008622

Editor: Francis Ouellette, University of Toronto, CANADA

Published: February 25, 2021

Funding: MJ's initial work on RiboViz was supported by the Software Sustainability Institute through UKRI grants EP/H043160/1, Grant EP/N006410/1 and Grant EP/S021779/1 (https://www.ukri.org). The RiboViz project is funded by BBSRC grant BB/S018506/1 (https://bbsrc.ukri.org) and NSF-Bio grant 1936046 (https://www.nsf.gov/dir/index.jsp?org=BIO). EWJW is supported by a Sir Henry Dale Fellowship jointly funded by the Wellcome Trust and the Royal Society [208779/Z/17/Z] (https://wellcome.ac.uk/grant-funding/schemes/sir-henry-dale-fellowships). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

way, describing its inputs, parameters, and outputs. By writing these wrappers, the scientist can focus on the meaning of each step, and how they fit together, which is the interesting part. The system uses these wrappers to decide what steps to run and how to run these and takes charge of running the steps, including reporting on errors. This makes it much easier to repeatedly run the analysis and to run it transparently upon different computers. To select a workflow management system, we surveyed available tools and chose 4 in which we developed prototype implementations to evaluate their suitability for our project. We conclude that many similar multistep data analysis workflows can be rewritten in a workflow management system, and we advocate prototyping as a low-cost (both time and effort) way of making an informed selection of software for use within a research project.

Introduction

Bioinformatics data analysis takes many steps, and a crucial but frustrating part of bioinformatics work is to run the right processing steps, in the right order, on the right data, reliably [1]. Usually these steps will involve disparate pieces of software from different sources, all run from the command line. For example, high-throughput sequencing data analysis may involve demultiplexing, trimming, cleaning, alignment, deduplication, base quality score recalibration, and quantification. Phylogenetic analysis may involve selecting sequences, multiple sequence alignment, alignment trimming, and tree inference. Image analysis can also involve many steps applied to large numbers of images. Success in these multistep data analyses generally requires writing a script to automate the steps. However, traditional shell scripts and even Makefiles have limited error reporting, are hard to debug, can be hard to restart after they go wrong, and can be challenging to move from one computer architecture to another. For example, bash scripts do not support re-entrancy or incremental build unless these functionalities are explicitly implemented by their authors, which can be a nontrivial development activity.

Workflow management systems—systems to represent, manage, and execute analyses—address these problems [2–4]. They can provide a common language for describing analysis workflows, contributing to reproducibility and the building of libraries of reusable components. They can support both incremental build and re-entrancy, providing the ability to selectively re-execute parts of a workflow in the presence of additional inputs or changes in configuration and the ability to resume execution from where a workflow previously stopped. Many workflow management systems provide support to exploit software containers and package managers, high-performance computing systems (HPC), and clouds. Most importantly, the declarative aspect of most workflow management systems allows bioinformaticians to concentrate on specifying what they want the workflow management system to do—for example, what inputs to read, steps to execute, or outputs to produce. The bioinformatician then delegates how these steps are executed, and how their workflows are run, to the workflow management system.

In this article, we describe the process that we used for selecting a workflow management system for our ribosome profiling software, RiboViz [5]. While Leipzig [4] offers advice on choosing a workflow management system based on the qualities of classes of workflow management system, we used an approach to selection focused on both the popularity of the candidate systems within the bioinformatics community and on the specific merits of the candidate systems in the context of our project's specific requirements.

To select a workflow management system, a rapid survey of available workflow management systems was undertaken to shortlist candidates for a more in-depth, hands-on, evaluation. Four candidates were shortlisted: Snakemake (https://github.com/snakemake/snakemake) [6]; cwltool (https://github.com/common-workflow-language/cwltool), a reference implementation of the Common Workflow Language (CWL) [7], a language for describing workflows; Toil (https://github.com/DataBiosphere/toil) [8], a production implementation of CWL; and Nextflow (https://github.com/nextflow-io/nextflow) [9].

We quickly implemented a subset of the steps of our workflow in each workflow management system. Development of these prototype implementations allowed us to evaluate each candidate more thoroughly than relying solely upon reviews and recommendations from others. From our evaluation, via development of these prototypes, Nextflow was chosen.

Using prototyping in this way offers a low-cost (both time and effort) approach for a more informed selection of myriad software applications, frameworks, packages, and libraries for use within projects. This article is intended to serve as a demonstration of this approach, the key activities of which are summarised in Table 1.

The intent of this article is not to make a recommendation as to the use of a specific workflow management system for all bioinformatics projects. Nor is this article intended to claim that using prototyping is suitable for the selection of all software or for all projects. Rather, it is to demonstrate how we found prototyping to be a useful approach to selecting a workflow management system that met the specific requirements of our project, and to discuss our experiences with the workflow management systems that we considered.

The RiboViz "workflows" repository (https://github.com/riboviz/workflows) [10] contains notes made during the selection and the Snakemake, Nextflow, and CWL workflows that were prototyped.

RiboViz and the requirement for a workflow management system

RiboViz is a high-throughput sequencing analysis pipeline specialised for ribosome profiling data. RiboViz takes raw data from sequencing machines; estimates how much each part of RNA is translated into protein and how the amount of translation is controlled by the code of that RNA; and produces analysis data, tables, and graphs.

RiboViz is under active development by The Wallace Lab and EPCC at the University of Edinburgh, The Shah Lab at Rutgers University, and The Lareau Lab at University of California, Berkeley. The source code is available on GitHub (https://github.com/riboviz/riboviz), under an Apache 2.0 open-source licence.

Table 1. Prototyping to select software for a project.

Step	Effort (person-days)	Tasks
Survey available software to shortlist candidates	3	Discover what software is available which may suit your requirements, which is in common use and is well regarded within the community. Draw up a shortlist of candidates based on their popularity, whether they have an acceptable software licence, and whether they are well established, stable, and with evidence of a future.
Evaluate candidates via prototyping	2–3 (per candidate)	Quickly prototype a subset of functionality required by the project and assess each candidate's suitability against objective project-specific criteria (required and useful functionality, supported platforms, etc.) and general subjective criteria (ease of installation, ease of implementation, quality of supporting documentation, etc.).

https://doi.org/10.1371/journal.pcbi.1008622.t001

At the heart of RiboViz is an analysis workflow to process ribosome profiling data across several samples, whose information along with all parameters for processing is described in a single input YAML file. Sample-specific read data can be provided as separate (fastq) input files or within a multiplexed input file. This workflow invokes a series of steps per sample (for example, adapter trimming, contaminant ribosomal RNA (rRNA) and open reading frame (ORF) alignment, and trimming 5′ mismatches). In addition, there are some initial, sample-independent, steps (for example, creating rRNA and ORF indices). When all samples have been processed, results from the sample-specific analyses are aggregated and summarised. Some steps (for example, Unique Molecular Identifier/UMI extraction and deduplication) are conditional upon the nature of the samples and configuration parameters set by the user. The workflow does not include any loops. S1 and S2 Figs show the steps of the RiboViz workflow that are invoked when processing demultiplexed samples and when processing multiplexed samples, respectively.

Each discrete step in a sequencing analysis workflow corresponds to the invocation, via bash, of a command-line tool. Some of these tools are open-source packages in widespread use within the bioinformatics community and include HISAT2 [11], Cutadapt [12], Samtools [13], Bedtools [14], and UMI-tools [15]. Other tools, implemented in Python and R (for demultiplexing multiplexed files, trimming reads, and generating analysis data, tables, and graphs), have been developed by the RiboViz team.

The RiboViz analysis workflow was implemented in a Python script. Each time a command-line tool is invoked, a log file is created for each invocation, in which standard output and error is captured. A log file for the execution of the Python script itself is also created. Sample-specific data and log files are written to sample-specific directories. The Python script also logs all the commands executed via bash to a script which can be run standalone and which allows a specific analysis to be rerun outwith the Python script. The RiboViz Python script can be configured to run in a "dry run" mode, whereby it will validate its configuration, check that input files exist, and output this complete bash script without executing the steps.

The design of the RiboViz Python script follows the majority of van Vliet's 7 quick tips for analysis scripts [16], tips which are generally applicable beyond neuroimaging. The development of RiboViz has also been strongly guided by Wilson and colleagues' best practices for scientific computing [17]. We use version control, unit test libraries, and, to help manage collaboration, an issue tracker. We turn bugs into test cases and endeavour to write programs for people. Their recommendation to "Write code in the highest-level language possible" motivated our migration to Python from a previous implementation of the analysis as a bash script.

However, as our Python script evolved, we were aware that we were adding more features related to managing the invocation of the analysis steps, rather than the nature of these steps themselves—we were implementing a custom workflow management system for RiboViz. This was problematic for several reasons. Our code was becoming more difficult to maintain as it evolved to accommodate additional requirements which were not envisaged when its implementation began in 2016. Our code did not support re-entrancy or incremental build, both of which we viewed as essential for implementing workflows to process large datasets. Nor did our code support parallel execution of the workflow which would be necessary to support the future execution of RiboViz on large-scale datasets. Implementing these would have incurred significant development effort, effort which would be better spent implementing the steps within the workflow, that is, the science itself.

It was time for us to adopt 2 more of the Wilson and colleagues best practices, to "Use a build tool to automate workflows" and to "Re-use code instead of rewriting it," that is, to use an off-the-shelf workflow management system, the adoption of which, we estimated, would incur significantly less effort than implementing re-entrancy, incremental build, and support for parallel processing ourselves.

A survey of available workflow management systems to shortlist candidates

We first conducted a rapid survey of available workflow management systems to shortlist candidates for prototyping. The criteria we used to select candidates for shortlisting are summarised in Table 2.

We conducted web searches to find out what systems, and existing surveys of systems, were available using combinations of the terms "workflow management system" and "bioinformatics," "survey," and "list." In keeping with our pragmatic, low-cost, approach, a systematic literature review was not undertaken as our goal was not to produce a comprehensive survey of every workflow management system available, but to, in a rapid way, identify which systems are in common use, and are well regarded, within the bioinformatics community.

Our searches found resources including introductory articles and blog posts [2–3], objective surveys of the types of workflow management systems available [4], lists and catalogues of workflow management systems [18–19], case studies [20] and discussion threads [21], and ad hoc polls [22] capturing the subjective opinions of researchers.

To avoid being rendered indecisive through over-choice, and, as we did not have time to review every popular option, even for shortlisting, we decided to choose only 2 systems: Snakemake and Nextflow. These systems had been most frequently, and positively, mentioned within the foregoing resources and had been specifically recommended by our team's bioinformaticians and their colleagues. Furthermore, these systems are free and open source, and also support the command-line, text editor–based development environment that we prefer. CWL had also been frequently and positively mentioned so we chose both its reference implementation, cwltool, and one of its production implementations, Toil. This necessarily meant that other, popular, workflow management systems, including Cromwell [23], Galaxy [24], Luigi (https://github.com/spotify/luigi), Pegasus [25], and Taverna [26], were not considered.

It was also important that we adopt a workflow management system that was well established, stable, and with a future [27]. We did not want to migrate to a system only for development around that system to stop. To assess the stability of and development activity around each tool, we reviewed statistics from their open-source repositories and the number of web search results for them (Table 3).

"Last updated" and "Number of contributors" were documented on 28 February 2020. "Search results" are the number of search results for the search term "<workflow management system> bioinformatics" on Google on 28 February 2020, although we note that changing the search term order yielded different numbers of results.

Table 2. Shortlisting criteria.

Criteria	Description
Popularity	The system seems to be in common use and is well regarded within the bioinformatics community. The system is likely to be practically usable.
Free and open-source licence	The system is free and has an open-source licence, as RiboViz itself is free and open-source software, and we seek to use open-source software where possible.
Well established, stable, and with a future	The system has been around for at least a year, has regular releases, and there is evidence that it is actively maintained, developed, and supported. Development of the system is unlikely to stop after we migrate to it.
Text-based workflow development environment	The system uses a command-line, text editor–based workflow development environment. As the RiboViz workflow is under ongoing development, this is the development environment that we prefer.

https://doi.org/10.1371/journal.pcbi.1008622.t002

Table 3. Statistics on the shortlisted workflow management systems.

Workflow management system	Software licence	Project start date	Last updated	Number of contributors	Search results
Snakemake	MIT	2013	28 February 2020	122	23,000
Nextflow	Apache 2.0	2013	28 February 2020	230	23,800
cwltool	Apache 2.0	2014	28 February 2020	72	2,440
Toil	Apache 2.0	2011	28 February 2020	81	20,700

https://doi.org/10.1371/journal.pcbi.1008622.t003

The shortlisted systems have free open-source licenses, have been in existence for several years, have many contributors, and are regularly updated. Overall, this survey gave us confidence that the candidate systems were being widely and actively used, developed, and supported and will continue to be so for the foreseeable future.

Evaluation of candidates via prototyping

Once a shortlist had been drawn up, we evaluated each candidate system. Our evaluation focused on quickly prototyping a subset of the RiboViz workflow into each system. There were 3 reasons for this. Firstly, using each system, and their documentation, would provide more insight into whether they met our requirements, their ease of use, their capabilities, and the quality of their supporting documentation than could be ascertained by solely reading their documentation. Secondly, focusing on implementing our workflow would give us more insight into these qualities than solely working through tutorial examples specifically designed by the developers of the systems to demonstrate their software. And, thirdly, whatever system we adopted, we would have the corresponding prototype to build upon.

A total of 2 to 3 person-days were allotted to each system. If nothing productive could be implemented within that period, then the system would be left and the next considered.

Our evaluation criteria are shown in Table 4.

It will be noted that the first 3 criteria are subjective, and necessarily so [28]. Ease of use, readability of documentation, and ease of implementation are very much dependent upon the skills, knowledge, and experience of those who will use a system and its supporting resources. For RiboViz, users are expected to be familiar with bash command-line tools and developers familiar with development of bash, Python, and R scripts under Linux. We sought a system that would enable us, and our user community, to execute, implement, maintain, and extend our workflow in a way that is easier than at present.

These, then, were our project-specific selection criteria. Table 5 summarises how each tool met our objective evaluation criteria (rows 4 to 6 on Table 4).

Snakemake

Snakemake was easy to download and install, via the conda (https://docs.conda.io/en/latest/) package manager, and had a comprehensive tutorial. Snakemake adopts the same model of operation as the GNU Make automated build tool (https://www.gnu.org/software/make/)—users specify the output files they want to build, Snakemake looks for rules to create these output files and runs the commands (in Snakemake, bash commands or Python scripts) specified in these rules to create the output files. Rules can specify dependencies—files used by the commands to create the output files. If these files do not exist, then Snakemake looks for rules to create these, and so on.

Snakemake is implemented in Python. Python code can also be embedded within a Snakefile, for example, to create file paths or validate configuration parameters.

Table 4. Evaluation criteria.

Criteria	Description
Ease of initial use	Ease of download, install, and initial use of each system.
Quality of supporting documentation	Readability and utility of documentation and tutorials.
Ease of implementation of initial steps of the RiboViz workflow	Ease of implementation of the initial 5 steps of the RiboViz workflow, which includes both sample-independent steps—build rRNA and ORF indices using HISAT2—and sample-specific steps—cut out sequencing library adapters using Cutadapt, remove rRNA or other contaminating reads by alignment to rRNA index files using HISAT2, and align to ORFs by alignment to ORF index files using HISAT2.
Required functionality	Functionality required by the RiboViz workflow. This includes iteration over multiple samples; error recovery strategies (so other samples can be processed even if processing of 1 sample fails); aggregation of sample-specific results; and conditional invocation of steps depending on configuration parameters set by a user.
Useful functionality	Functionality that is not required to execute the RiboViz workflow but that is useful to us for both porting and further developing the workflow. This includes producing step-specific log files to aid debugging; parsing YAML configuration files, so that our current configuration files could continue to be used and to ease migration of users onto the selected workflow management system; a "dry run" option to validate configuration and check that input files exist and display the commands that would be run, without actually running these, which can be a useful way of checking time-consuming or complex workflows before running them; and outputting a bash script that can be rerun standalone which can be used when debugging to help understand why a workflow failed and to rerun specific steps in the workflow.
Execution within containers, HPC systems, or cloud	This is desirable to support the future execution on RiboViz on large-scale datasets.

HPC, high-performance computing; ORF, open reading frame; rRNA, ribosomal RNA.

https://doi.org/10.1371/journal.pcbi.1008622.t004

Implementing the steps from the RiboViz workflow was very straightforward, greatly helped by the authors' prior experience with GNU Make from other projects. This, coupled with there being no requirement to implement any additional code beyond the Snakemake workflow itself, meant that a functional version of the complete RiboViz workflow of 14 steps (everything bar steps specifically to handle multiplexed files) was implemented in less than a person-day. This ease of implementation accounts for the discrepancy in the Number of RiboViz processing steps implemented, as compared to cwltool, Toil, and Nextflow, shown in Table 5.

Snakemake provided all the required and useful functionality listed in our evaluation criteria. Snakemake provides a "keep going" configuration parameter which can be used to continue processing other samples if processing of 1 sample fails. Like Make, Snakemake supports incremental build and re-entrancy. Conditional behaviour can be executed via the use of Python conditions. Step-specific log files can be implemented, but Snakemake does not automatically capture these—the bash commands executed by each step explicitly need to redirect standard output and standard error streams into log files. Like Make, Snakemake supports a "dry run" option that can check that input files exist and that displays the commands that would be run, without running these. As for Make, the ability to specify exactly the files to build can be useful for debugging.

While Snakemake does not output a bash script that can be run standalone, it can output a summary file with the commands submitted to bash for execution. This file could be parsed, and the commands extracted and constructed into a bash script.

Snakemake has support for running its jobs within containers, HPC systems, and clouds.

Table 5. Summary of workflow management systems and objective evaluation criteria.

Criteria	Snakemake	cwltool	Toil	Nextflow
Installation method used	conda	Python pip	Python pip	conda
Number of RiboViz processing steps implemented	14	3	3	5
Time (person-days) to implement steps	1	1	1[1]	2
Iteration over multiple samples	Yes	See note[2]	See note[2]	Yes
Error recovery strategies	Yes ("keep going" parameter)	See note[2]	See note[2]	Yes (step-specific error strategies)
Aggregation of sample-specific results	Yes	See note[2]	See note[2]	Yes
Conditional execution of steps	Yes (step-specific Python conditions)	No	No[3]	Yes (step-specific "when" clauses)
Step-specific log files	Yes (must be captured explicitly by each step)	See note[2]	See note[2]	Yes (automatically captured)
YAML configuration files	Yes	See note[2]	See note[2]	Yes
"dry run" option	Yes	See note[2]	See note[2]	No (but configuration validation can be implemented)
Output a rerunnable bash script to see the commands that were actually executed	No (outputs a summary file with bash commands)	See note[2]	See note[2]	No (outputs step-specific bash scripts)
Execution within containers, HPC systems, and cloud	Yes	Containers only	Yes	Yes

[1]The time taken relates to writing CWL workflows, not Toil-specific workflows.

[2]These criteria were not explored as the decision had been made to not consider CWL further considering its lack of support for conditional execution of steps.

[3]The lack of support for conditional execution is a restriction of CWL, not Toil.

HPC, high-performance computing.

https://doi.org/10.1371/journal.pcbi.1008622.t005

Common workflow language, cwltool, and toil

Both cwltool and Toil were easy to install, via the Python pip package manager (https://pip.pypa.io/). It was easy to run a CWL "hello world!" example via both cwltool and Toil. A comprehensive, step-by-step tutorial to the language is available at https://www.commonwl.org/user_guide [29].

CWL tool wrappers, which describe the inputs and outputs of command-line tools, and job configuration files, which describe workflows, are written as YAML or JSON documents. JavaScript can be embedded for any additional computation that is required, for example, to create file paths or validate configuration parameters.

Implementing 3 steps of the RiboViz workflow took a person-day. It was not necessary to implement any additional code beyond the CWL tool wrappers and job description files themselves. The "edit-run-debug" development cycle felt slow and painful, due to the richness of CWL and the occasionally cryptic error messages that arose during execution.

Conditional behaviour is not yet supported within CWL—a "Collecting use cases for workflow level conditionals" issue [30] was added in February 2020 to their 1.2 milestone, but, at the time of writing (August 2020), this has no due date. The lack of conditional invocation means that CWL is not currently suitable for RiboViz, or for other projects that require input-dependent control of workflow structure. (A colleague had evaluated CWL about a year and a half ago and, while they felt that simple workflows showed promise, the lack of conditionals meant that they could not adopt CWL for their project. Similarly, we felt that CWL would not be suitable for RiboViz at this time.) This limitation could have been identified at the shortlisting stage, but we had to achieve a balance between how many criteria to consider during shortlisting and how many during our prototyping. We (incorrectly as it turned out) assumed that support for conditional execution would be a fundamental feature of any workflow management system or languages, such as CWL, executed by them.

Nextflow

Nextflow was easy to download and install, via the conda package manager, and had a simple tutorial.

A Nextflow workflow has a structure analogous to a Makefile or Snakefile—it consists of a set of processes which define inputs, outputs, and commands describing how to create the outputs from the inputs. However, Nextflow adopts a dataflow programming model whereby the processes are connected via their outputs and inputs to other processes, and processes run as soon as they receive an input. Unlike Snakemake, a user does not specify the files they want to create, rather, they declare their input files and related configuration, and Nextflow continues to invoke processes until no process has any outstanding inputs.

Nextflow is implemented in Java. Code written in Groovy, which runs under Java, can be embedded within a Nextflow workflow. This code can be used to, for example, create file paths or validate configuration parameters.

Implementing steps from the RiboViz workflow was straightforward. It was not necessary to implement any additional code beyond the Nextflow workflow itself. A functional version of the subset of the RiboViz workflow was implemented in 2 person-days. Nextflow's documentation was comprehensive, but additional concrete examples of how to implement common use cases, and an expanded tutorial how to implement a complex multistep workflow, would have been helpful. Although we overlooked it during prototyping, the Nextflow web site does have a repository of common implementation patterns [31]. Similarly, we later found a Nextflow 2017 workshop tutorial [32], but our understanding of Nextflow had, at that point, passed beyond its content. Despite this, writing a Nextflow workflow was easier than writing a CWL workflow.

Nextflow provided all the required and most of the useful functionality listed in our evaluation criteria. Each step can have an "error strategy" which indicates what to do if an error is encountered. This can be used to ensure that other samples are processed if processing of 1 sample fails, or to adjust process resource parameters if a reported error arises from a lack of memory or a time limit that is too low. Nextflow, like Snakemake, supports both incremental build and re-entrancy, via a "resume" option. Conditional execution of steps is supported via a "when" declaration. Unlike Snakemake, it is not possible to specify the exact files to build, which can make debugging more challenging. However, every invocation of a step takes place in its own isolated subdirectory which includes a bash script with the command that was invoked, symbolic links to input files, output files, and files with the contents of the standard output and error streams. The step-specific bash scripts can be run within their step-specific directories which is useful for debugging the implementation of individual steps. These directories have auto-generated names, but Nextflow allows the contents of these directories to be written into known locations with more readable names.

A "dry run," analogous to that supported by Snakemake and Make, has been suggested in a Nextflow issue [33], but has not progressed due to challenges in implementing such a feature within a dataflow model. The Nextflow authors instead recommend using small datasets to validate scripts. It should be noted that it may be challenging to identify a small dataset that would allow adequate replication of the workflow's behaviour in the presence of a full dataset.

Nextflow has support for running its jobs within containers, HPC systems, and cloud.

Selecting a workflow management system

We decided to adopt Nextflow for the following reasons. The execution of each step within isolated subdirectories, within which individual steps can be re-executed, is useful for debugging. While writing Nextflow workflows does require knowledge of Groovy, the authors, familiar with Python and R, did not find learning Groovy challenging. The fact that Nextflow was based on Java incurs no additional installation overhead for either users or developers compared to Snakemake—each can be installed using the conda package manager using a single command. Based on our impressions of their documentation, Nextflow's built-in support for, and documentation around, containers, HPC systems, and cloud, seemed more thorough than that of Snakemake (although we appreciate that this may change as both tools evolve). It was our subjective impression that Nextflow felt far richer than Snakemake both in terms of features and expressivity, and it was felt that these outweighed its lack of a dry run feature. Finally, we had to pick one, and we just liked Nextflow better.

Completing the RiboViz workflow implementation in Nextflow

It took approximately 5 person-days to complete an implementation of the RiboViz workflow (including support for multiplexed files) within Nextflow. Our existing regression test framework for our Python script was used to validate the implementation of our Nextflow script.

The Nextflow implementation has been tested by the RiboViz development team on their own development platforms and also on EDDIE, The University of Edinburgh's HPC cluster (https://www.ed.ac.uk/information-services/research-support/research-computing/ecdf/high-performance-computing).

Release 2.0 of RiboViz [34] includes the Nextflow implementation of the RiboViz workflow. The Python implementation of the RiboViz workflow will be deprecated in a future release.

Nextflow has the nf-core collection of bioinformatics pipelines, a resource of open-source, reviewed, and validated Nextflow scripts implementing common data analyses [35]. The associated nf-core developer community (136 members as of 8 July 2020; https://nf-co.re/community) has some overlap with the Nextflow developers, but is primarily composed of bioinformaticians. Again, these provide strong evidence for a well-established system with a future, and we will consider contributing RiboViz to nf-core in the future.

However, no choice of software should, or can, be permanently binding. Our positive experiences with Snakemake, and the small effort that would be required to complete the implementation of RiboViz into Snakemake, give us confidence that if we need to migrate from Nextflow to Snakemake in future, then this would be a relatively straightforward migration to undertake.

Conclusions

In this article, we described the process that we devised for selecting a workflow management system for our ribosome profiling software, RiboViz. The use of prototyping for evaluating our options gave us a working prototype, from which we developed an implementation in our chosen workflow management system. Our approach took approximately 10 person-days for background reading, our survey and shortlisting and prototyping. In our view, this is a small cost for the assurance that our selected workflow management system meets our requirements, is well established, widely, and actively used, developed and supported, and will continue to be so for the foreseeable future. The use of Nextflow provides us with an implementation of RiboViz that is more maintainable, more portable, and which will allow us to exploit the power of distributed computing resources in the analysis of large-scale datasets. We would agree that workflow management systems are a technology that "bioinformaticians need to be using right now" [3] and that they can implement right now using well-engineered open-source tools.

Reiter and colleagues [36] also recommend prototyping, in this case for migrating to a chosen workflow management system. In particular they recommend that "When building a workflow for the first time, creating an initial workflow based on a subset of your sample data can help verify that the workflow, tools, and command line syntax function at a basic level. This functioning example code then provides a reliable workflow framework free of syntax errors which you can customize for your data without the overhead of generating correct workflow syntax from scratch." This was our experience. Our prototype of the RiboViz workflow in Nextflow, created as part of our comparative evaluation of our shortlisted systems, provided a sound basis for completing our implementation once we had decided upon Nextflow as our chosen workflow management system.

Reflecting upon our process, we are satisfied with our selection and evaluation criteria. However, if repeating this exercise, we would have applied some of our evaluation criteria (Table 4) during our surveying and shortlisting. Considering those criteria that could have been identified within user documentation prior to prototyping—required functionality; useful functionality; execution within containers, HPC systems, or cloud—we could have eliminated CWL prior to prototyping due to its lack of support for the conditional execution of steps.

It should also be noted that there are many other criteria that could be considered when selecting a workflow management system. These include the ability to specify the compute and memory resources required for each step in the workflow (for example, some steps may be computationally intensive, others memory intensive); their integration with job schedulers (if running the workflow within clusters or HPC environments); and support for the execution of specific steps, or collections of specific steps, within isolated software environments (for example, conda packages or Docker containers (https://www.docker.com/))).

We had a well-understood workflow and already had a working implementation in Python. Our goal was to port this to a workflow management system. Other researchers may be in a situation of both having to develop a new workflow, the steps within which and requirements of which are far less well defined or understood, and to select, and familiarise themselves with, a workflow management system. Our adopted process—survey and shortlist, then evaluate via prototyping—would still be applicable but with a recognition that, as the workflow evolves and its requirements become more understood, it may be that the workflow management system chosen initially is no longer adequate. This would necessitate a return to the surveying and shortlisting to identify other candidates that would meet the current requirements, to evaluate these and to select one to migrate to.

However, this is a scenario that applies even to well-understood workflows like RiboViz. Nextflow, like all the workflow management systems described in this article, is under active development and is constantly evolving, as is the RiboViz workflow. What Nextflow offers today may be deprecated, or nonexistent, 6 months or a year from now. Similarly, the nature of the RiboViz workflow may evolve so that what it requires from Nextflow is no longer being provided, and we need to migrate to another workflow management system. If that situation arises, then we would need to revisit and rerun the process we have described.

The use of prototyping may not be suitable for the selection for all software or for all projects. For example, it would not be suitable for selecting software for large-scale IT projects or critical infrastructure. However, the use of prototyping does offer a low-cost way of making a more informed selection of software to use within projects, rather than relying solely upon reviews and recommendations by others.

Supporting information

S1 Fig. RiboViz workflow steps invoked when processing demultiplexed sample files.
(TIF)

S2 Fig. RiboViz workflow steps invoked when processing multiplexed sample files.
(TIF)

Acknowledgments

We thank our colleagues for discussions and comments on the manuscript: Hywel Dunn-Davies, Wellcome Centre for Cell Biology, The University of Edinburgh; Alison Meynert, MRC Human Genetics Unit, The University of Edinburgh; Premal Shah, Department of Genetics, Rutgers University; and Kit Transue, Seattle, USA. We thank the Software Sustainability Institute for their support in starting this project.

References

1. Shade A, Teal TK. Computing Workflows for Biologists: A Roadmap. PLoS Biol. 2015; 13(11): e1002303. https://doi.org/10.1371/journal.pbio.1002303 PMID: 26600012
2. Perkel JM. Workflow systems turn raw data into scientific knowledge. Nature. 2019; 573:149–50. https://doi.org/10.1038/d41586-019-02619-z PMID: 31477884
3. Watson M. The three technologies bioinformaticians need to be using right now. Opiniomics [Internet]. 2019 [cited 2020 Aug 4]. Available from: http://www.opiniomics.org/the-three-technologies-bioinformaticians-need-to-be-using-right-now.
4. Leipzig J. A review of bioinformatic pipeline frameworks. Brief Bioinform. 2017 May 1; 18(3):530–6. https://doi.org/10.1093/bib/bbw020 PMID: 27013646
5. Carja O, Xing T, Wallace EWJ. Plotkin JB, Shah P. riboviz: analysis and visualization of ribosome profiling datasets. BMC Bioinform. 2017 Oct; 25(461):18. https://doi.org/10.1186/s12859-017-1873-8 PMID: 29070028
6. Köster J, Rahmann S. Snakemake–A scalable bioinformatics workflow engine. Bioinformatics. 2012 Oct; 28(19):2520–2. https://doi.org/10.1093/bioinformatics/bts480 PMID: 22908215
7. Amstutz P, Crusoe MR, Tijanic N, Chapman B, Chilton J, Heuer M, et al. Common Workflow Language, v1.0. Specification. 2016 [cited 2020 Aug 4]. Database: figshare [Internet]. https://doi.org/10.6084/m9.figshare.3115156.v2
8. Vivian J, Rao AA, Nothaft FA, Ketchum C, Armstrong J, Novak A, et al. Toil enables reproducible, open source, big biomedical data analyses. Nat Biotechnol. 2017; 35(4):314–6. https://doi.org/10.1038/nbt.3772 PMID: 28398314
9. Di Tommaso P, Chatzou M, Floden EW, Barja PP, Palumbo E, Notredame C. Nextflow enables reproducible computational workflows. Nat Biotechnol. 2017; 35(4):316–9. https://doi.org/10.1038/nbt.3820 PMID: 28398311
10. Jackson M, Wallace E, Kavoussanakis K, Anderson F. Options for RiboViz workflow management. [cited 2020 Oct 28] Database: figshare [Internet]. 2020. https://doi.org/10.6084/m9.figshare.13147979
11. Kim D, Langmead B, Salzberg SL. HISAT: a fast spliced aligner with low memory requirements. Nat Methods. 2015; 12:357–60. https://doi.org/10.1038/nmeth.3317 PMID: 25751142
12. Martin M. Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet J. 2011(1):10–2. https://doi.org/10.14806/ej.17.1.200
13. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009; 25(16):2078–9. https://doi.org/10.1093/bioinformatics/btp352 PMID: 19505943
14. Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics. 2010; 26(6):841–2. https://doi.org/10.1093/bioinformatics/btq033 PMID: 20110278
15. Smith T, Heger A, Sudbery I. UMI-tools: Modelling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy. Genome Res. 2017; 27:491–9. https://doi.org/10.1101/gr.209601.116 PMID: 28100584
16. van Vliet M. Seven quick tips for analysis scripts in neuroimaging. PLoS Comput Biol. 2020; 16(3). https://doi.org/10.1371/journal.pcbi.1007358 PMID: 32214316
17. Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. Best Practices for Scientific Computing. PLoS Biol. 2014; 12(1):e1001745. https://doi.org/10.1371/journal.pbio.1001745 PMID: 24415924
18. Di Tommaso P et al. Awesome Pipeline. GitHub [Internet]. 2014[updated 2020 Jun 20; cited 2020 Aug 4]. Available from: https://github.com/pditommaso/awesome-pipeline.

19. Amstutz P, Crusoe MR, Tijani N, Chapman B, Chilton J, Heuer M, et al. Computational Data Analysis Workflow Systems. GitHub [Internet]. 2016 [updated 2020 Jun 30; cited 2020 Aug 4]. Available from: https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems.

20. Baichoo S, Souilmi Y, Panji S, Botha G, Meintjes A, Hazelhurst S, et al. Developing reproducible bioinformatics analysis workflows for heterogeneous computing environments to support African genomics. BMC Bioinformatics. 2018; 19(457). https://doi.org/10.1186/s12859-018-2446-1 PMID: 30486782

21. zeekawla99ii. Given the experience of others writing bioinformatic pipelines, what are the pros/cons of Toil vs Snakemake vs Nextflow? Reddit [Internet]. 2018 [cited 2020 Aug 4]. Available from: https://www.reddit.com/r/bioinformatics/comments/a4fq4i/given_the_experience_of_others_writing.

22. Vilella A. Which Bioinformatics Workflow Manager / Tool / Platform / Language / Specification / Standard do you use or prefer? Twitter [Internet]. 2018 [cited 2020 Aug 4]. Available from: https://twitter.com/AlbertVilella/status/1069635987427532800.

23. Voss K, Van der Auwera G, Gentry J. Full-stack genomics pipelining with GATK4 + WDL + Cromwell [version 1; not peer reviewed]. F1000Res. 2017, 6(ISCB Comm J):1381 (slides) https://doi.org/10.7490/f1000research.1114634.1

24. Afgan E, Baker D, Batut B, van den Beek M, Bouvier D, Cech M, et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Res 2018 Jul 2. 46(W1): W537–W544. https://doi.org/10.1093/nar/gky379 PMID: 29790989

25. Deelman E, Vahi K, Juve G, Rynge M, Callaghan S, Maechling PJ, et al. Pegasus: a Workflow Management System for Science Automation. Future Gener Comput Syst. 2015 May; 46:17–35. https://doi.org/10.1016/j.future.2014.10.008

26. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Res. 2013 Jul 1; 41(W1) W557–W561. https://doi.org/10.1093/nar/gkt328 PMID: 23640334

27. The Software Sustainability Institute. Choosing the right open-source software for your project. The Software Sustainability Institute [Internet] c2011 [cited 2020 Aug 4]. Available from: https://www.software.ac.uk/choosing-right-open-source-software-your-project.

28. Nielsen N. What Is Usability? In: Wilson C, editor. User Experience Re-Mastered: Your Guide to Getting the Right Design. Morgan Kaufmann; 2010. pp3–22. https://doi.org/10.1016/B978-0-12-375114-0.00004–9

29. Hodges T, Crusoe MR. Common Workflow Language User Guide. [cited 2020 Jul 1] Database: zenodo [Internet]. 2017. https://doi.org/10.5281/zenodo.840663

30. Common Workflow Language. Collecting use cases for workflow level conditionals #725. GitHub [Internet]. 2018 [updated 2020 Feb 11; cited 2020 Aug 4]. Available from: https://github.com/common-workflow-language/common-workflow-language/issues/725.

31. Nextflow. Nextflow patterns. GitHub [Internet]. [cited 2020 Oct 19]. Available from: http://nextflow-io.github.io/patterns/index.html.

32. Nextflow. Nextflow tutorial. GitHub [Internet]. 2018 [cited 2020 Aug 4]. Available from: https://github.com/nextflow-io/nf-hack17-tutorial.

33. Nextflow. suggestion: run -dry #31. GitHub [Internet]. 2015 [updated 2020 Jan 14; cited 2020 Aug 4]. Available from: https://github.com/nextflow-io/nextflow/issues/31.

34. Wallace E, Anderson F, Kavoussanakis K, Jackson M, Shah P, Lareau L. riboviz: software for analysis and visualization of ribosome profiling datasets. 2020 Jul 8 [cited 2020 Aug 4]. Database: figshare [Internet]. https://doi.org/10.6084/m9.figshare.12624200

35. Ewels PA, Peltzer A, Fillinger S, Patel H, Alneberg J, Wilm A, et al. The nf-core framework for community-curated bioinformatics pipelines. Nat Biotechnol. 2020; 38:276–8. https://doi.org/10.1038/s41587-020-0439-x PMID: 32055031

36. Reiter T, Brooks PT, Irber L, Joslin SEK, Reid CM, Scott C, et al. Streamlining Data-Intensive Biology With Workflow Systems. bioRxiv [Internet]. 2020 [cited 2020 Jul 22]. https://doi.org/10.1101/2020.06.30.178673