SOFTWARE TOOL ARTICLE

CGAT-core: a python framework for building scalable, reproducible computational biology workflows [version 1; peer review: 1 approved, 1 approved with reservations]

Adam P. Cribbs [1,2], Sebastian Luna-Valero[2], Charlotte George[2], Ian M. Sudbery[3], Antonio J. Berlanga-Taylor[4], Stephen N. Sansom[5], Tom Smith[6], Nicholas E. Ilott[5], Jethro Johnson[7], Jakub Scaber [2], Katherine Brown[8], David Sims[2], Andreas Heger[2]

[1]The Botnar Research Center, University of Oxford, Oxford, OX3 7LD, UK
[2]MRC WIMM Centre for Computational Biology, University of Oxford, Oxford, OX3 9DS, UK
[3]Department of Molecular Biology and Biotechnology, University of Sheffield, Sheffield, S10 2TN, UK
[4]MRC-PHE Centre for Environment and Health, Department of Epidemiology & Biostatistics, Imperial College London, Oxford, W2 1PG, UK
[5]Kennedy Institute of Rheumatology, University of Oxford, Oxford, OX3 7FY, UK
[6]Cambridge Centre for Proteomics, Department of Biochemistry, University of Cambridge, Cambridge, CB2 1QR, UK
[7]The Jackson Laboratory for Genomic Medicine, Farmington, CT, 06032, USA
[8]Division of Virology, Department of Pathology, University of Cambridge, Cambridge, CB2 1QP, UK

v1 **First published:** 04 Apr 2019, **8**:377 (https://doi.org/10.12688/f1000research.18674.1)
Latest published: 04 Apr 2019, **8**:377 (https://doi.org/10.12688/f1000research.18674.1)

Abstract

In the genomics era computational biologists regularly need to process, analyse and integrate large and complex biomedical datasets. Analysis inevitably involves multiple dependent steps, resulting in complex pipelines or workflows, often with several branches. Large data volumes mean that processing needs to be quick and efficient and scientific rigour requires that analysis be consistent and fully reproducible. We have developed CGAT-core, a python package for the rapid construction of complex computational workflows. CGAT-core seamlessly handles parallelisation across high performance computing clusters, integration of Conda environments, full parameterisation, database integration and logging. To illustrate our workflow framework, we present a pipeline for the analysis of RNAseq data using pseudo-alignment.

Keywords

workflow, pipeline, python, genomics

This article is included in the Python Collection collection.

Open Peer Review

Referee Status: ✔ ?

	Invited Referees 1	2
version 1 published 04 Apr 2019	✔ report	? report

1 **Devon P. Ryan**, Max Planck Institute of Immunobiology and Epigenetics (MPI-IE), Germany

2 **Alexander Peltzer**, University of Tübingen, Germany

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding authors: Adam P. Cribbs (adam.cribbs@imm.ox.ac.uk), David Sims (david.sims@imm.ox.ac.uk), Andreas Heger (andreas.heger@gmail.com)

Author roles: Cribbs AP: Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Luna-Valero S**: Software, Writing – Review & Editing; **George C**: Software, Writing – Review & Editing; **Sudbery IM**: Software, Writing – Review & Editing; **Berlanga-Taylor AJ**: Software, Writing – Review & Editing; **Sansom SN**: Software, Writing – Review & Editing; **Smith T**: Software, Writing – Review & Editing; **Ilott NE**: Software, Writing – Review & Editing; **Johnson J**: Software, Writing – Review & Editing; **Scaber J**: Software, Writing – Review & Editing; **Brown K**: Software, Writing – Review & Editing; **Sims D**: Project Administration, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Heger A**: Project Administration, Software, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This work was funded by the Medical Research Council (UK) Computational Genomics Analysis and Training programme (G1000902).

How to cite this article: Cribbs AP, Luna-Valero S, George C *et al.* **CGAT-core: a python framework for building scalable, reproducible computational biology workflows [version 1; peer review: 1 approved, 1 approved with reservations]** F1000Research 2019, **8**:377 (https://doi.org/10.12688/f1000research.18674.1)

First published: 04 Apr 2019, **8**:377 (https://doi.org/10.12688/f1000research.18674.1)

Introduction

Genomic technologies have given researchers the ability to produce large amounts of data at relatively low cost. Bioinformatic analyses typically involve passing data through a series of manipulations and transformations, called a pipeline or workflow. The need for tools to manage workflows is well established, with a wide range of options available from graphical user interfaces such as Galaxy[1] and Taverna[2], aimed at non-programmers, to Snakemake, Nextflow, Toil, CGAT-Ruffus and others[3–10] developed with computational biologists in mind. These tools differ in their portability, scalability, parameter handling, extensibility, and ease of use. In a recent survey[11], the tool rated highest for ease of pipeline development was CGAT-Ruffus[12], a Python package that wraps pipeline steps in discrete Python functions, called 'tasks'. It uses Python decorators to track the dependencies between tasks, ensuring that dependent tasks are completed in the correct order and independent tasks can be run in parallel. If a pipeline is interrupted before completion, or new input files are added, only data sets that are missing or out-of-date are re-run. CGAT-Ruffus implements a wide range of decorators that allow complex operations on input files including: conversion of a single input file to a single output file; splitting of a file into multiple files (and vice versa) and conditional merging of multiple input files into a smaller number of outputs. More advanced options include combining combinations or permutations of input files and conditional execution based on input parameters. Use of decorators means that CGAT-Ruffus pipelines are native Python scripts, rather than the domain specific languages (DSLs) used in other workflow tools. A key advantage of this, in addition to python being an already widely understood language in computational biology, is that individual steps can use arbitrary python code, both in how they are linked together and in the actual processing task.

Here, we introduce Computational Genomics Analysis Toolkit (CGAT)-core[13], an open-source python library that extends the functionality of CGAT-Ruffus by adding cluster interaction, parameterisation, logging, database interaction and Conda environment switching.

Methods

CGAT-core[13] extends the functionality of CGAT-Ruffus by providing a common interface to control distributed resource management systems using Distributed Resource Management Application API (DRMAA),. Currently, we support interaction with Sun Grid Engine, Slurm and PBS-pro/Torque. The execution engine enables tasks to be run locally or on a high-performance computing cluster and supports cluster distribution of both command line scripts (*cgatcore.run*) and python functions (*cgatcore.cluster*). System resources (number of cores to use, amount of RAM to allocate) can be set on a per-pipeline, per-task, or per task-instance basis, even allowing allocation to be based in variables, for example input file size.

Operation

The parameter management component encourages the separation of workflow/tool configuration from implementation to build re-usable workflows. Algorithm parameters are collected in a single human-readable yaml configuration file. Thus, parameters can be set specifically for each dataset, without the need to modify the code. For example, sequencing data can be aligned to a different reference genome, by simply changing the path to the genome index in the yaml file. Both pipeline-wide and job-local parameters are automagically substituted into command line statements at execution-time.

To assist with reproducibility, record keeping and error handling CGAT-core provides multi-level logging during workflow execution, recording full details of runtime parameters, environment configuration and tracking job submissions. Additionally, CGAT-core provides a simple, lightweight interface for interacting with relational databases such as SQLite (*cgatcore.database*), facilitating loading of analysis results at any step of the workflow, including combining output from parallel steps in single wide- or long-format tables.

CGAT-core can load a different Conda environment for each step of the analysis, enabling the use of tools with conflicting software requirements. Furthermore, providing Conda environment files alongside pipeline scripts ensures that analyses can be fully reproduced.

CGAT-core workflows are Python scripts, and as such are stand-alone command line utilities that do not require the installation of a dedicated service. In order to reproducibly execute our workflows, we provide utility functions for argument parsing, logging and record keeping within scripts (*cgatcore.experiment*). Workflows are started, inspected and configured through the command line interface. Therefore, workflows become just another tool and can be re-used within other workflows. Furthermore, workflows can leverage the full power of Python, making them completely extensible and flexible.

Implementation

CGAT-core is implemented in Python 3 and installable via Conda and PyPI with minimal dependencies. We have successfully deployed and tested the code on OSX, Red Hat and Ubuntu. We have made CGAT-core and associated repositories open-source under the MIT licence, allowing full and free use for both commercial and non-commercial purposes. Our software is fully documented (https://pypi.org), version controlled and has extensive testing using continuous integration (https://travis-ci.org/cgat-developers.) We welcome community participation in code development and issue reporting through GitHub.

Use case

To illustrate a simple use case of CGAT-core, we have built an example RNAseq analysis pipeline, which performs read counting using Kallisto[14] and differential expression using DESeq2[15]. This workflow and Conda environment are contained within our CGAT-showcase repository (https://github.com/cgat-developers/cgat-showcase). The workflow highlights how simple pipelines can be constructed using CGAT-core, demonstrating how the pipeline can be configured using a yaml file, how third-party tools can be executed efficiently across a cluster or on a local machine, and how data can be easily loaded into a

database. Furthermore, we and others have been extensively using CGAT-core to build pipelines for computational genomics (https://github.com/cgat-developers/cgat-flow).

Discussion

CGAT-core[13] extends the popular Python workflow engine CGAT-Ruffus by adding desirable features from a variety other workflow systems to form an extremely simple, flexible and scalable package. CGAT-core provides seamless high-performance computing cluster interaction and adds Conda environment integration for the first time. In addition, our framework focuses on simplifying the pipeline development and testing process by providing convenience functions for parameterisation, database interaction, logging and pipeline interaction.

The ease of pipeline development enables CGAT-core to bridge the gap between exploratory data analysis and building production workflows. A guiding principle is that it should be as easy (or easier) to complete a series of tasks using a simple pipeline compared to using an interactive prompt, especially once cluster submission is considered. CGAT-core enables the production of analysis pipelines that can easily be run in multiple environments to facilitate sharing of code as part of the publication process. Thus, CGAT-core encourages a best-practice reproducible research approach by making it the path of least resistance. For example, exploratory analysis in Jupyter Notebooks can be converted to a Python script or used directly in the pipeline. Similarly, exploratory data analysis in R, or any other language, can easily be converted to a script that can be run by the pipeline. This lightweight wrapping of quickly prototyped analysis forms a lab book, enabling rapid reproduction of analyses and reuse of code for different data sets.

Data availability

All data underlying the results are available as part of the article and no additional source data are required.

Software availability

Source code available from: https://github.com/cgat-developers/cgat-core.

Archived source code at time of publication: https://doi.org/10.5281/zenodo.2598115[13].

Licence: MIT License.

Grant information

This work was funded by the Medical Research Council (UK) Computational Genomics Analysis and Training programme (G1000902).

References

1. Afgan E, Baker D, van den Beek M, *et al.*: **The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update.** *Nucleic Acids Res.* 2016; **44**(W1): W3–W10.
PubMed Abstract | Publisher Full Text | Free Full Text
2. Wolstencroft K, Haines R, Fellows D, *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; **41**(Web Server issue): W557–61.
PubMed Abstract | Publisher Full Text | Free Full Text
3. Okonechnikov K, Golosova O, Fursov M, *et al.*: **Unipro UGENE: a unified bioinformatics toolkit.** *Bioinformatics.* 2012; **28**(8): 1166–7.
PubMed Abstract | Publisher Full Text
4. Golosova O, Henderson R, Vaskin Y, *et al.*: **Unipro UGENE NGS pipelines and components for variant calling, RNA-seq and ChIP-seq data analyses.** *PeerJ.* 2014; **2**: e644.
PubMed Abstract | Publisher Full Text | Free Full Text
5. Nocq J, Celton M, Gendron P, *et al.*: **Harnessing virtual machines to simplify next-generation DNA sequencing analysis.** *Bioinformatics.* 2013; **29**(17): 2075–83.
PubMed Abstract | Publisher Full Text
6. Gafni E, Luquette LJ, Lancaster AK, *et al.*: **COSMOS: Python library for massively parallel workflows.** *Bioinformatics.* 2014; **30**(20): 2956–8.
PubMed Abstract | Publisher Full Text | Free Full Text
7. Vivian J, Rao AA, Nothaft FA, *et al.*: **Toil enables reproducible, open source, big biomedical data analyses.** *Nat Biotechnol.* 2017; **35**(4): 314–316.
PubMed Abstract | Publisher Full Text | Free Full Text
8. Fisch KM, Meißner T, Gioia L, *et al.*: **Omics Pipe: a community-based framework for reproducible multi-omics data analysis.** *Bioinformatics.* 2015; **31**(11): 1724–8.
PubMed Abstract | Publisher Full Text | Free Full Text
9. Köster J, Rahmann S: **Snakemake--a scalable bioinformatics workflow engine.** *Bioinformatics.* 2012; **28**(19): 2520–2.
PubMed Abstract | Publisher Full Text
10. Di Tommaso P, Chatzou M, Floden EW, *et al.*: **Nextflow enables reproducible computational workflows.** *Nat Biotechnol.* 2017; **35**(4): 316–319.
PubMed Abstract | Publisher Full Text
11. Leipzig J: **A review of bioinformatic pipeline frameworks.** *Brief Bioinform.* 2017; **18**(3): 530–536.
PubMed Abstract | Publisher Full Text | Free Full Text
12. Goodstadt L: **Ruffus: a lightweight Python library for computational pipelines.** *Bioinformatics.* 2010; **26**(21): 2778–9.
PubMed Abstract | Publisher Full Text
13. Heger A, Cribbs A, Luna-Valero S, *et al.*: **cgat-developers/cgat-core: First public release of code (Version v0.5.10).** *Zenodo.* 2019.
http://www.doi.org/10.5281/zenodo.2598115
14. Bray NL, Pimentel H, Melsted P, *et al.*: **Near-optimal probabilistic RNA-seq quantification.** *Nat Biotechnol.* 2016; **34**(5): 525–7.
PubMed Abstract | Publisher Full Text
15. Love MI, Huber W, Anders S: **Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2.** *Genome Biol.* 2014; **15**(12): 550.
PubMed Abstract | Publisher Full Text | Free Full Text

Open Peer Review

Current Referee Status: ✔ ?

Version 1

Referee Report 24 April 2019

https://doi.org/10.5256/f1000research.20448.r47003

? **Alexander Peltzer**
Quantitative Biology Center (QBIC), University of Tübingen, Tübingen, Germany

Cribbs *et al* describe CGAT-Core, a python framework for building scalable, reproducible computational biology workflows in their proposed software tool article.

The rationale behind the requirements for developing the software tool is explained properly, although there are many competitor tools and alternatives already "on the market" that can do similar or more things in general. The authors briefly summarize the large variance in portability, scalability, parameter handling and extensibility of the various tools in a sufficient form. One statement I cannot confirm in the last part of the introduction "... in addition to python being an already widely understood language in computational biology, is that individual steps can use arbitrary python code, both in how they are linked together and in the actual processing tasks". This is by all means not a drawback of a DSL, as some of these (e.g. nextflow, but to my knowledge also other competitors) allow for the direct integration of Python code in their respective tasks as well: https://www.nextflow.io/docs/latest/process.html#native-execution.

The statement in the methods section: "Thus, parameters can be set specifically for each datasets, without the need to modify the code" is a statement true for all workflow languages I know about (at least Snakemake, Nextflow and Toil separate configuration from actual pipeline code and can use e.g. parameter input files similar to the one that CGAT-Core uses, though with slightly different notation of course). As such, this statement should be probably changed or removed as it incorrectly makes readers assume that this is a unique feature of CGAT-Core. One could for example state that this is a best-practice pattern across various workflow languages and CGAT-Core also enables users to separate pipeline code from e.g. infrastructure or parameter descriptions.

One larger lack I see is that CGAT-Core unlike e.g. Snakemake, CWL, Nextflow and others does not support container technologies such as Docker or Singularity (to only name the two biggest solutions per market share out there). There have been papers critically investigating the effects of non-containerized conda environments "which are ideal for packaging" but not for long and mid-term reproducibility of analysis questions due to changing environments on a client side (see Grüning et al[1] for details). These issues are only properly addressed by using containerization approaches, which CGAT-Core at the moment seems not to address. I think the authors should mention that the containerization of pipeline dependencies is a crucial part of reproducible data analysis today, and maybe whether they intend to add this in an upcoming release of CGAT-Core.

To not only mention critical parts: I do think the authors did a really good job in documentation, proper GitHub organization set up with README and all required information as well as setting up a community,

which I'd like to congratulate them for.

I was also able to run the mentioned example pipeline with Kallisto on my local infrastructure, although I had to adapt certain parts in order to be able to do so.

I'm missing two points in the discussion: benefits for users to use CGAT-Core in general over other competitor tools, and also a more critical discussion on where the framework could be extended to support other environments (e.g. no mention of any cloud service/provider?) in general.

Overall the paper reads well and I think it only requires minor changes, especially highlighting differences to other available workflow tools and critically assessing pros/cons with respect to these.

Typos:

- Discussion: "... by adding desirable features from a variety other" (missing "of")

References

1. Grüning B, Chilton J, Köster J, Dale R, Soranzo N, van den Beek M, Goecks J, Backofen R, Nekrutenko A, Taylor J: Practical Computational Reproducibility in the Life Sciences.*Cell Syst*. 2018; **6** (6): 631-635 PubMed Abstract | Publisher Full Text

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Evolutionary Biology, Bioinformatics, Workflow/Pipeline Development, Systems Integration, Human Genetics, Population Genetics.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Referee Report 16 April 2019

https://doi.org/10.5256/f1000research.20448.r46758

Devon P. Ryan
Max Planck Institute of Immunobiology and Epigenetics (MPI-IE), Freiburg, Germany

Writing and using analysis pipelines has become a bioinformatician's (or more generally a data analyst's) bread and butter. There are a number of frameworks to perform such analyses, of which CGAT-Ruffus is preferred by many. CGAT-core brings some welcome functionality to Ruffus. In my mind, the addition of parameterisation and conda environment switching were the two biggest stumbling blocks to using Ruffus previously and CGAT-core seems to nicely address both of these.

After going through the documentation (and a fair bit of trial and error due to not being particularly used to using Ruffus syntax) I was able to create and run a very simple workflow from scratch that included paired-end read trimming (cutadapt) and alignment (STAR) and used the parameterisation added by CGAT-core.

While not critical to the manuscript, it'd be nice if the authors could address the following in the documentation (apologies to the authors if I missed some of these in the documentation):

- The conda install instructions should be `conda install -c conda-forge -c bioconda cgatcore`
- Can examples of processing paired-end data be included in the showcase or elsewhere in the documentation? A particular step that I found difficult to implement the first time was performing read trimming such that the resulting files had the same name but were placed in a different directory. It turns out that this can be done with `formatter()`, but an example like the following in the documentation would probably be useful to new users like me:

```
# pairs is a list of (read1, read2) tuples
@transform(pairs, formatter(".+/(?P.*)_R[12].fastq.gz$"), ("trimmed/{SAMPLE[0]}_R1.fastq.gz", "trimmed/{SAMPLE[0]}_R2.fastq.gz"))
```

- Is there any way to use standard commands to run jobs on a cluster rather than drmaa? Novice users are likely to have an easier time modifying `qsub` and `srun` commands than finding the drmaa shared libraries.
- At least in my testing it wasn't possible to use something like the following in a command:

```
cmd = """module load cutadapt
      cutadapt ..."""
```

One can combine the two commands with `&&` (or use a conda env), but I didn't notice this in the documentation. That's only a mild annoyance, but it should be mentioned to new users (especially those familiar with snakeMake, where commands are written to a shell script that's then submitted to the cluster).

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Bioinformatics, epigenetics, immunobiology and neuroscience

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.