BIOINFORMATICS APPLICATIONS NOTE Vol. 28 no. 11 2012, pages 1525–1526
doi:10.1093/bioinformatics/bts167

Genome analysis

Advance Access publication April 12, 2012

Bpipe: a tool for running and managing bioinformatics pipelines

Simon P. Sadedin[1,*], Bernard Pope[2] and Alicia Oshlack[1]

[1]Murdoch Childrens Research Institute, Royal Children's Hospital, Flemington Road, Parkville, Victoria 3052 and [2]Victorian Life Sciences Computation Initiative, The University of Melbourne, Carlton, Victoria 3010, Australia

Associate Editor: Alex Bateman

ABSTRACT

Summary: Bpipe is a simple, dedicated programming language for defining and executing bioinformatics pipelines. It specializes in enabling users to turn existing pipelines based on shell scripts or command line tools into highly flexible, adaptable and maintainable workflows with a minimum of effort. Bpipe ensures that pipelines execute in a controlled and repeatable fashion and keeps audit trails and logs to ensure that experimental results are reproducible. Requiring only Java as a dependency, Bpipe is fully self-contained and cross-platform, making it very easy to adopt and deploy into existing environments.

Availability and implementation: Bpipe is freely available from http://bpipe.org under a BSD License.

Contact: simon.sadedin@mcri.edu.au

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on December 18, 2011; revised on March 28, 2012; accepted on April 2, 2012

1 INTRODUCTION

Bioinformatics is a rapidly expanding field in which the arrival of new technologies and tools, and the evolution of experimental techniques is a constant occurrence. Therefore, data analysis pipelines cannot be static and researchers are faced with a continual need to adapt, understand, experiment with and integrate new computational tools into their analyses. Such analyses are usually composed of a chain of tools that perform separate stages of the process. For example, calling variants from exome sequencing data typically involves several tasks such as aligning the raw data to the genome; removing duplicate reads; recalibrating quality scores; calling variants; and filtering variants. For each step, several alternative computation tools are available, but ultimately a tool for each step needs to be chosen and integrated into a complete pipeline to produce results of biological significance.

Integration of such diverse computational tasks into a cohesive unit is approached in different ways. Where the expertise is available, scripting languages such as Perl or Python are frequently employed to create 'pipeline' scripts. At the other end of the spectrum, tools such as Galaxy (Goecks *et al.*, 2010) and Taverna (Oinn *et al.*, 2004) have emerged which offer many advantages in terms of ease of use and level of automation.

Despite the advantages of other tools, some bioinformaticians still prefer to run jobs as shell scripts. Although this makes running them easy, it has many limitations. For example, when scripts fail before completion, it is often difficult to determine where or why they failed and even harder to restart from the point of failure. There is no automatic log of commands or capture of output. Failed jobs may leave half-created files that can be confused with completed files. Modifying the pipeline often requires changes in multiple places, meaning that a missed change can cause commands to fail, or use incorrect data. Bpipe tries to solve these problems while retaining much of the simplicity and syntax of a shell script.

*To whom correspondence should be addressed.

2 BPIPE LANGUAGE

Bpipe is implemented in Groovy, a language that supports creation of Domain-Specific Languages for the Java Virtual Machine. While Bpipe retains the ability for advanced users to extend scripts using Java or Groovy, it does not require knowledge of either language to implement pipelines (see Supplementary Material).

The key drivers behind the design of the Bpipe language are simplicity and concision, allowing expression of the pipeline behavior with very little syntactical overhead. There are two steps to creating a Bpipe pipeline. The first is to define the computational stages of the pipeline that perform the specific tasks, such as aligning sequencing reads to the genome. The second step is to then combine the tasks together into a pipeline. This achieves reusability of pipeline stages and easy modification of the pipeline sequence.

The Bpipe syntax for defining stages is very similar to running a tool with the command line; the command is simply placed inside curly braces and executed using the keyword *exec*. For example, to define the stage of aligning sequencing reads to a genome using bwa (Li *et al.*, 2009), we would write:

```
1 align_reads = {
2       exec "bwa aln -t 8 $input > $output"
3 }
```

The two variables, $input and $output, are provided implicitly by Bpipe and ensure that Bpipe can verify and track both the inputs and outputs of the stage. Next, once a set of stages have been defined, a pipeline can be joined together using a concise syntax that utilizes mathematical operators to symbolize pipeline construction:

```
4 Bpipe.run {
5   align_reads + dedupe + call_variants
6       }
```

Bpipe pipeline construction includes other useful features such as running stages in parallel, indicated by placing the sections in square brackets:

```
7 Bpipe.run {
8   align_reads+[ dedupe, calculate_stats ] +
```

```
9               call_variants
10      }
```

and handling batch inputs using simple pattern matching functionality:

```
11 Bpipe.run {
12   "input%_*.txt" * [ align_reads + dedupe ]
13                  + call_variants
14      }
```

where each group will be processed in parallel by the align_reads and dedupe stages that follow.

3 RUNNING A BPIPE PIPELINE

Bpipe scripts are executed by a simple command line tool called 'bpipe'. The tool can launch a pipeline, display status, stop a pipeline or retry a partially executed pipeline from the point of failure. It automatically captures all output and executed commands to log files while forwarding the output to the console for the user to see or leave running in the background as desired. Requiring only Java as a dependency, it can run in a user's local directory without any installation, making it especially easy to deploy into a new environment. In addition, Bpipe can interact with cluster resource management systems and includes built-in support for the TORQUE Resource Manager system (http://www.adaptivecomputing.com/products/torque.php). We provide a generic interface to resource managers, making it easy to support other systems in the future.

4 APPLICATION DOMAIN

There are numerous existing tools for performing computational workflows offering differing degrees of automation and control. Some tools such as Galaxy and Taverna feature high levels of automation and support sophisticated graphical user interfaces. These tools are ideal for a broad range of practicing bioinformaticians as they are easy to use, built from well-established tools and protocols and allow easy access to a wide range of online data sources. Other tools, such as shell scripts, are mostly manual but offer complete control over how tasks execute. Bpipe and Ruffus (Goodstadt, 2010), a pipeline construction toolkit based on Python, sit somewhere in the middle of this spectrum where there is some level of automation but also still a significant level of fine-grained control.

While Bpipe has similar goals and features to Ruffus, it differs in several ways. Bpipe was created in response to a need to frequently run many variations of a pipeline with stages deleted, inserted, reordered or adjusted. Although Ruffus supports such activity, it was found to be challenging to implement because Ruffus does not explicitly model the joining of stages together as a language construct. Rather it combines the definition of pipeline stages and ordering by using Python annotations attached to each pipeline stage. In addition, Bpipe, through its support of shell variable syntax, enables the user to copy and paste a shell command they already use directly into their pipeline with little modification. Ruffus, however, usually needs commands to be converted to a Python-based syntax for variable substitution. Thus, although both tools can accomplish the same job, Bpipe is optimized for users who wish to execute shell commands as directly as possible and desire the ability to frequently modify or reorder them as their pipeline evolves.

To illustrate how Bpipe eases pipeline modifications, the pipeline above could be trivially modified to remove the 'dedupe' stage by simply deleting it from line 5 as follows:

```
4 Bpipe.run {
5   align_reads + call_variants
6      }
```

As the pipeline stages were not modified, both versions of the pipeline can be maintained simultaneously. By comparison, unless the user went to special efforts to enable it, a Ruffus script would require modifications to several places including the pipeline stages themselves and it would be harder to maintain both versions of the pipeline simultaneously.

In summary, Bpipe contains attractive features to build and maintain bioinformatics pipelines including the following:

- Simple definition of tasks—Bpipe runs shell commands almost as is.
- Transactional management of tasks—outputs of failed commands are cleaned up, log files saved and the pipeline cleanly aborted.
- Automatic connection of stages—removing or adding new stages never breaks the flow of data.
- Easy restarting of jobs—when a job fails, Bpipe cleanly restarts from the point of failure.
- Audit trail—Bpipe keeps a journal of which commands were executed and all their inputs and outputs.

ACKNOWLEDGEMENTS

We acknowledge Nadia Davidson for testing and providing feedback, and other members of the Oshlack group for helpful discussion.

Conflict of Interest: none declared.

REFERENCES

Goecks,J. *et al.* (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.*, **11**, R86.

Goodstadt,L. (2010) Ruffus: a lightweight Python library for computational pipelines. *Bioinformatics*, **26**, 2778–2779.

Li,H. *et al.* (2009) Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics*, **25**, 1754–1760.

Oinn,T. *et al.* (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, **20**, 3045–3054.