Poster B-39

Automating the Maintenance of Bioinformatics Services with Aap

Authors:
Estienne Swart (*National Bioinformatics Network (South Africa)*)
Ruediger Brauning (*National Bioinformatics Network (South Africa)*)

Short Abstract: Bioinformatics Services are complex systems that require a significant amount of effort to maintain. Much of this burden can be eased by using a unified framework for concise, declarative specification and flexible, imperative scripting. For our purposes we employed a generic build system, Aap (http://www.a-a-p.org).

Long Abstract:

Automating the Maintenance of Bioinformatics Services with Aap
****

Bioinformatics services are often intricate systems, with both data and code components that are in a state of flux. Maintaining these servies requires a considerable amount of manual labour, which is time-consuming, resource intensive and error-prone. One of the major hurdles to be overcome in the integration of biological databases is the sheer amount of system maintenance involved.

We are maintaining local mirrors of some of the most important international bioinformatics services. In order for acceptance of these mirrors by local researchers it is essential that we offer these resources with comparable performance and functionality to the international sources. To meet these requirements it is essential that our services are kept up-to-date with their international counterparts, and fine-tuned to offer the most reliable and quickest possible services we can deliver.

To this end, we have been developing maintenance tasks using a generic build system - Aap (http://www.a-a-p.org). Build systems were initially developed for the purposes of software construction, but have also been used to great effect in the maintenance of software systems, most notably the Ports system of BSD derivatives. Build systems have also been suggested as a manner to compose and execute bioinformatics workflows and as an approach to formulating large-scale data processing tasks[1]. They have been employed in data-centric processing which employs relational databases for intermediate recording and reporting[2].

Build systems seem well suited to these kinds of tasks: they make clear the distinction between declarative specification (targets and dependencies), which is use to construct execution paths with observable intermediates; and imperative specification (actions and scripting), which is used to direct the construction of the declared components. Furthermore, maintenance of the build files in a code versioning repository facilitates close scrutiny of development and corrections, and alignment of a particular version of maintenance code with a particular version of the system being maintained.

Unfortunately, traditional build systems, such as Make, suffer from a number of problems. Among these problems: that their syntax is somewhat obscure and there are niggling syntactic issues; shell scripting can introduce portability issues when system commands are employed and may be difficult to debug; variable namespaces are messy and confusing; basic language constructs are ad hoc or missing; and processing of tasks based upon timestamps can be flummoxed.

Aap attempts to address all these issues. By employing Python and whitespace delimitation of blocks similar to Python, consistent syntax is provided. The core scripting language is a fully-fledged, modern programming language, and is portable as long as calling system specific commands is avoided. The completion of targets is no longer dependent upon timestamps, but rather md5 checksums, so time related problems will not arise. Aap also provides inbuilt commands to simplify many of the common processes for software construction, including those for fetching data, and basic retrieval and checkin of code from a versioning system. These commands can easily be extended or supplemented.

To date, we have written basic recipes (the Aap term for build files) for scheduled downloading of mirrored databases, and recipes for maintenance of our Ensembl mirror. We are in the process of writing further down-the-line processing recipes, such as those required for reformatting data and creating database indices. All our code is available in a publicly accessible subversion repository. We have a web interface to this code and an associated bug reporting and feature request system (http://www.nbn.ac.za/auto).

Aap seems to be well-suited to the task of maintaining bioinformatics services. It has proven relatively straightforward to code additional improvements to the core system, as we find them necessary. In future, we would like to incorporate other data types commonly encountered in Bioinformatics, such as relational databases, as target types.

In the final analysis, we believe that it is not essential to define the declarative and imperative components of a build file as two separate languages. The build system Scons, which processes Python code alone, is an example of a single language approach, but Rake and Rant appear to be far more concise and elegant solutions (http://www.martinfowler.com/articles/rake.html), and seem to be worthy candidates for the development of a general-purpose Ruby-based workflow system for bioinformatics.

References
==*=*=*=

[1] DS Parker, MM Gorlick, and CJ Lee. Evolving from bioinformatics in-the-small to bioinformatics in-the-large. OMICS, 7(1):37--48, 2003.
[2] S. Conery, M. Catchen, and Michael Lynch. Rule-based workflow management for bioinformatics. The VLDB Journal, 14(3):318--329, 2005.