Lenguajes de programación para la bioinformática

Noé Fernández Pozo
Contratado predoctoral de la Universidad de Málaga, Plataforma Andaluza de Bioinformática
noefp@uma.es

Saber programar te puede facilitar la vida

Hoy en día, con el avance de la informática, las redes de comunicación y las nuevas técnicas de laboratorio, podemos disponer de gran cantidad de información en formato digital. En el caso de la biología, hay multitud de bases de datos que almacenan información procedente de experimentos biológicos y se están realizando experimentos con aparatos que nos pueden devolver una gran cantidad de información en ficheros informáticos, como por ejemplo los secuenciadores de nueva generación y los experimentos con micromatrices.

Las bases de datos de secuencias muestran un crecimiento exponencial, de modo que el volumen de información que se acumula no puede ser procesado manualmente. Así que, si queremos sacar conocimientos útiles de las grandes cantidades de datos, no nos queda más remedio que recurrir a la informática. Si conocemos algún lenguaje de programación, podremos acceder a muchos programas y módulos ya creados para procesar datos biológicos e incluso desarrollar nuestros propios *scripts* (pequeños programas realizados en lenguajes interpretados), de modo que podamos realizar un análisis a medida de nuestros datos.

En principio, un usuario básico puede pensar que le costará mucho que los lenguajes de programación sirvan para sus intereses. Para eliminar este prejuicio conviene empezar con cosas sencillas que según nuestras necesidades nos ahorren bastante trabajo. Por ejemplo, ese usuario debería aprender a utilizar interpretes de comandos (lo que se conoce como el *shell*), por ejemplo el *bash*, que vienen instalados necesariamente en los sistemas operativos Linux y Mac OS X. Vamos a ver un par de ejemplos muy sencillos, en los que nos podemos ahorrar mucho tiempo, tan solo escribiendo una línea de código. Si tecleas

```
cat *fasta > todas_las_seqs.fasta
```

podrás unir en un sólo fichero todos los ficheros de secuencias con extensión fasta que tengas en el directorio en el que te encuentres. El comando cat imprime en pantalla el fichero o ficheros que le indiquemos. Al poner *fasta le estamos indicando a cat que imprima todos los ficheros con cualquier nombre (*) que acabe en la cadena de texto fasta. Después de esto, al poner el '>' seguido de un nombre de fichero de salida, estamos indicando que queremos que el resultado, en lugar de salir por la pantalla, se escriba en el fichero de salida indicado. Con:

```
grep -c '^>' todas_las_seqs.fasta
```

podremos saber cuantas secuencias hay dentro del fichero todas_las_seqs.fasta porque el comando grep sirve para imprimir únicamente las líneas que contiene un texto. Para indicar este texto se puede utilizar lo que se conoce como expresiones regulares. Al indicar a grep que imprima las líneas que comiencen (^) por '>', extraerá las líneas que contienen el nombre de la secuencias, pero si utilizamos el parámetro '-c' (*count)*, en lugar de esto, obtenemos el número de líneas que comienzan por '>' en nuestro fichero fasta, o lo que es lo mismo, cuantas secuencias contiene.

Lenguajes interpretados frente a lenguajes compilados

Los lenguajes interpretados o de *scripting*, utilizan un programa que interprete el código para que se pueda ejecutar. El interprete verifica que no haya errores sintácticos antes de ejecutar el programa. De este modo se pueden realizar muchos cambios sobre la marcha e ir viendo cómo han afectado. Sin embargo, los programas realizados en lenguajes compilados hay que convertirlos en binarios ejecutables cada vez que se quiere comprobar un cambio y, si no funcionan correctamente, se vuelven a editar, compilar y ejecutar. En cambio, cuando el programa ya esta acabado y compilado sin errores, son mucho más rápidos que los *scripts* realizados en lenguajes interpretados. Por estos motivos, con los lenguajes interpretados, como R, Perl, Ruby o Python, se aprende a programar con más facilidad y se desarrollan algoritmos y prototipos en muy poco tiempo. Sin embargo, los lenguajes compilados, como C o C++, proporcionan ejecutables mucho más rápidos, a los que se les habrá dedicado mucho más esfuerzo y tiempo.

Por ejemplo, en el caso del algoritmo BLAST que se utiliza en los servidores del NCBI, es impensable que estuviera escrito en un lenguaje interpretado ya que, a través de dicho servidor, se realizan miles o millones de peticiones diarias. La diferencia de tiempo, por mínima que sea, se volverá muy significativa si en lugar de estar escrito en C, lo estuviera en Perl. Por eso, en muchos módulos de BioPerl y de otros lenguajes interpretados, existen subrutinas escritas en C que se encapsulan (sin el que usuario lo perciba) para ejecutar la parte más dura del algoritmo; el resultado se devuelve en el lenguaje interpretado sin que el usuario perciba el trasiego de lenguajes. De este se aceleran algunas partes básicas de los programas escritos en lenguajes interpretados.

Otro motivo por el cual los lenguajes compilados son más rápidos es porque no verifican el código del programa durante su ejecución, lo que pone obliga al inexperto a realizar el tedioso trabajo de arreglar los errores en el código. Sin embargo, en los lenguajes interpretados, la verificación del código es constante, lo que permite ir arreglando los errores que vayan surgiendo en nuestros *scripts* en el momento.

Para hacer pequeños programas o *scripts* que cambien el formato de nuestros datos, obtengan la información útil de un fichero o analicen nuestros datos de forma rápida y automática, es interesante para todo biólogo aprender, al menos a un nivel básico, un leguaje interpretado. Entre los más utilizados en biología están Perl, Python, Java y Ruby.

Perl (http://www.perl.org/)

Perl tiene muchos módulos que podemos utilizar a través de BioPerl (colección de módulos en Perl que realizan funciones útiles en bioinformática, como cargar un fichero fasta o analizar la salida de BLAST). Además, es el más rápido de los lenguajes interpretados y el más utilizado (dos motivos que se retroalimentan mutuamente), por lo que es del que más cosas hay hechas. Al ser un lenguaje de *scripting*, resulta fácil y rápido ir probando los cambios que vamos realizando. Por otro lado, permite resumir mucho el código, es decir, hacer mucho escribiendo muy poco, con lo que los scripts son tan crípticos que al cabo de unas semanas puede que nos resulte difícil comprenderlos incluso aunque lo haya escrito uno mismo. También es bastante desestructurado, con lo que permite escribir el código muy enmarañado. Para los más avanzados, hay que saber que no es verdaderamente un lenguaje orientado a objetos, aunque sea capaz de manejarlos.

Se podría decir que el principal problema de Perl es que al poder hacer mucho escribiendo muy poco código, finalmente hay que dedicar mucho tiempo a escribir lineas de comentarios explicativos que permitan para seguir entendiendo en el futuro cómo funciona el script, cosa que no sucede en otros lenguajes como Python o Ruby

Python (http://www.python.org/) y Ruby (http://www.ruby-lang.org/es/)

Son lenguajes con una sintaxis más limpia que Perl, lo que permite entender fácilmente el código aunque lo haya escrito otra persona. Por ahora son algo más lentos que Perl, pero son mucho más agradables para el programador y están orientados a objetos de verdad. Los objetos permiten una mejor organización del código y un considerable ahorro de tiempo a la hora de seguir evolucionando un programa durante bastante tiempo, además de permitirnos reutilizar partes del código sin apenas modificaciones Por estos motivos, la tendencia actual es utilizar menos Perl y pasarse a Python, principalmente, o Ruby. De hecho, en los últimos años parece que Python y BioPython se están haciendo cada vez más populares, al igual que BioRuby y Ruby, aunque este último es más conocido por su entorno de desarrollo web Ruby On Rails.

Java

Java es otro lenguaje de programación ampliamente utilizado en bioinformática que también está orientado a objetos. Los programas en Java se traducen a un código intermedio que posteriormente es interpretado por la JVM (java virtual machine). Además, este lenguaje puede estar semicompilado, lo que lo hace más rápido que el resto de los lenguajes interpretados, sin llegar a ser tan difícil y tedioso como los lenguajes compilados. Entre las ventajas de Java están que es un lenguaje muy utilizado, por lo que hay una gran cantidad de código disponible, y tiene la posibilidad de crear entornos gráficos para nuestros programas de modo que puedan funcionar en cualquier plataforma (Windows, Linux ,Mac OS X). Pero esto es sobre el papel, ya que en la práctica es difícil que el mismo código valga para todas las plataformas.

Ruby On Rails (http://www.rubyonrails.org.es/) y Django (http://www.djangoproject.com/)

En la comunidad bioinformática también es común la utilización de entornos de desarrollo web como Ruby On Rails o Django, basados en Ruby y Python respectivamente. Estos entornos nos permiten crear páginas web con bases de datos de un modo muy rápido y fácil. Se basan en una estructura Modelo-Vista-Controlador (MVC), en la que el modelo indica las relaciones entre las clases (objetos) de la base de datos, la vista muestra para cada clase una página web (en la que ponemos la información de la base de datos que queremos mostrar) y el controlador es el intermediario que manda los datos a cada vista. De este modo, por ejemplo, si queremos hacer una web en la que mostrar la información de 10 000 secuencias, es muy sencillo crear una única vista (fichero con Ruby embebido en html) en la que ponemos lo que queremos mostrar de las secuencias y automáticamente el controlador sabe sustituir en esa plantilla para la vista de la web, los datos de las 10 000 secuencias.

En nuestro laboratorio hemos desarrollado la herramienta web SeqTrimNext (http://www.scbi.uma.es/seqtrimnext) para la limpieza y preprocesado de secuencias procedentes de secuenciadores de nueva generación. La Web de está aplicación esta hecha con Ruby On Rails y el algoritmo del programa en Ruby. Su ejecución se realiza de modo distribuido, usando a la vez varias CPU de diferentes ordenadores, en los supercomputadores de la UMA Pablo y Picasso. También hemos creado una base de datos basada en Ruby On Rails para los análisis transcriptómicos de pino (http://www.scbi.uma.es/pindb).

R (http://www.r-project.org/) y MATLAB (http://www.mathworks.com/products/matlab/)

R y MATLAB son lenguajes y entornos ampliamente utilizados en la bioinformática para los análisis estadísticos, y aunque podemos desarrollar scripts, comúnmente se utilizan de modo interactivo en una consola. Ambos tienen una gran cantidad de paquetes con funciones para biología y se utilizan con frecuencia para generar gráficos y trabajar con tablas de datos organizados en matrices como las generadas en los experimentos de micromatrices. Sin embargo si los datos tienen una estructura más compleja, son bastante lentos realizando bucles para recorrer la información. Una gran diferencia reside en que MATLAB es de pago y R es gratuito, aunque existen versiones gratuitas compatibles con MATLAB, como Octave (http://www.gnu.org/software/octave/). El paquete que contiene las herramientas bioinformáticas para R es Bioconductor (http://www.bioconductor.org), que ya contiene hasta herramientas para el análisis de experimentos de ultrasecuenciación y su anotación masiva.

Lecturas recomendadas para saber más:

- Falgueras J., et al. SeqTrim: a high-throughput pipeline for preprocessing any type of sequence reads BMC Bioinformatics 2010, 11:38
- Goto et al. BioRuby: bioinformatics software for the Ruby programming language. Bioinformatics 2010, 26(20):2617-9
- Cock et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. 2009, 25(11):1422-3
- Fourment M. and Gillings M.R. A comparison of common programming languages used in bioinformatics. BMC Bioinformatics 2008, 9:82
- Stajich JE. An introduction to BioPerl. Meth Mol Biol 2007, 406:535-48