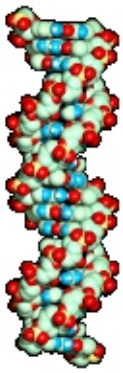
# LINUX Y PERL, integración de herramientas informáticas para estudio y análisis de información biológica

por Carlos Andrés Pérez
<caperez(at)usc.edu.co>

*Sobre el autor:*

Carlos Andrés Pérez es especialista en Simulación Molecular, Doctorando en Biotecnología. Asesor técnico del Grupo de Investigación en Educación Virtual (GIEV). Dirección: Universidad Santiago de Cali, Calle 5ª carrera 62 Campus Pampalinda, Cali – Colombia.



*Resumen*:

El presente artículo pretende exponer algunas de las ventajas de la programación en Perl bajo entornos Unix, para la extracción de información biológica de las bases de datos de secuencias de DNA, RNA y proteínas, con el fin de utilizarlas en procesos comparativos o de análisis. El proyecto Genoma Humano y las diferentes técnicas de clonación de DNA han acelerado el proceso de generación de información sobre la secuencia de muchos genes funcionales a tal punto que la información sobre secuencias genéticas que se genera a diario supera la capacidad de analizar dicha información desde el punto de vista evolutivo y los procesos de interacción de estos genes y su forma de regularse. La rápida proliferación de la información biológica sobre diferentes genomas (dotación de genes de un organismo) está constituyendo a la bioinformática como una disciplina fundamental para el manejo y análisis de estos datos.

## Bioinformática

La bioinformática nace cuando se empiezan a almacenar las secuencias biológicas en un formato digital y surgen los primeros programas para compararlas. Durante mucho tiempo la bioinformática se limito al análisis de secuencias . Sin embargo, la importancia de establecer la forma estructural de las moléculas hace que las computadoras se conviertan en una importante herramienta investigación en Bioquímica teórica. Cada vez hay más información y más colecciones de datos sobre la conformación 3D de las moléculas. La expresión de los genes ha pasado de estudiarse en forma individual a hacerlo sobre el conjunto completo, o una parte muy extensa, de los genes de un organismo. Se comprende la importancia de la interacción entre genes, entre

proteínas y su organización en las rutas metabólicas. Y cada vez más nos percatamos de la necesidad de utilizar todo este colosal conjunto de datos de forma integrada.

Cada una de las actividades descritas tienen al menos dos caras desde las que resultan interesantes. Por una parte es indudable el interés biológico de conocer las relaciones entre las moléculas de la vida, y por otra parte se hace interesante el conjunto de problemas computacionales que se plantean. Es indudable la necesidad de combinar e integrar la información biológica para obtener una visión global y efectiva de los procesos biológicos que subyacen en ella. De la misma forma nos hemos percatado de la necesidad de combinar las diferentes áreas de la informática para dar una solución efectiva. No solo se trata de gestión de bases de datos, también de integración de datos; no solo de algoritmos eficientes, sino de hardware potente –grids, multiprocesadores, etc; no solo de algoritmos exactos, sino también de aprendizaje y heurísticos.

# Perl

***Larry Wall*** comenzó el desarrolló de Perl en 1986. Perl es un lenguaje de programación interpretado, ideal para manipular textos, ficheros y procesos. Perl permite desarrollar rápidamente trabajos que normalmente se realizarían en C o en alguna Shell. Podría decirse que Perl es una mezcla optimizada de un lenguaje de alto nivel (por ejemplo C) y un lenguaje de script (por ejemplo bash).

Lo que se programe en Perl se puede ejecutar en varios sistemas operativos/plataformas. No obstante, donde nació y donde más se ha difundido es bajo el sistema operativo UNIX, es un intérprete de código, un lenguaje de programación, pensado inicialmente para recoger en un único lenguaje ampliado las características de varios. Perl sobrepasó con creces sus objetivos iniciales especialmente por el impulso que recibió su inmediato uso como programador web. Antes que Perl se utilizaban awk, sed y grep para analizar ficheros y extraer de ellos información que permitieran ejecutar acciones.

Perl reunió las posibilidades de estas ideas UNIX en un sólo programa ampliando y modernizando cada función y haciéndolamás potente, práctica y más rápida.

Perl es un lenguaje de programación gratuito y se puede ejecutar en cualquiera de los sistemas operativos que generalmente se encuentran en los laboratorios de investigaciones biológicas, aunque presenta una gran integralidad con Uníx y Linux. En UNIX y MacOSX viene instalado, en los demás hay que instalarlo. Basta con obtenerlo del sitio: http://www.cpan.org/ para el sistema que estemos usando.

Los programas en perl bajo Linux se pueden Llamar con el nombre del fichero que contiene las instrucciones a ejecutar. Es lo más usual. Las instrucciones están guardas en un fichero y se llama a perl con el nombre del fichero como argumento.

Otro método tan frecuente como el anterior es el de guardar las instrucciones Perl en un fichero pero sin llamar a Perl explícitamente. Para ello se tienen que hacer dos cosas: (a) poner un indicativo especial en la primera línea del programa:

```
#!/usr/bin/env perl

print "Hola\n";
```

y (b) guardar el fichero y asignarle el atributo UNIX de fichero ejecutable:

```
% chmod +x saludo.pl
```

Una vez hecho esto se puede llamar al fichero−programa cuantas veces se desee directamente por su nombre, llegando a olvidarse finalmente que es un programa Perl. De hecho la extensión muchas veces se quita a este tipo de programas quedando mezcladas con las utilidades del sistema.

# Manipulación de Ficheros en Perl:

Cuando tenemos una base de datos de secuencias moleculares en formato texto, una posible herramienta que podemos crear con Perl, es un buscador de secuencias. Así, en este ejemplo, vamos a ver como se puede buscar una secuencia de proteínas en una base de datos con formato SWISS−PROT (db_human.swissprot), a partir de su código de identificación.

```
#!/usr/bin/perl
# Buscar una secuencia de aminoacidos en una base de datos
# con formato SWISS-PROT, dado un codigo de identificacion
# Pide el codigo de identificacion que esta en el campo ID
# y lo pasa de la entrada estandar (STDIN) a una variable
print "Introduce el ID de la secuencia a buscar: ";
$id_query=<STDIN>;
chomp $id_query;
# Abrimos el fichero de la base de datos
# pero si no se puede, finaliza el programa
open (db, "human_kinases.swissprot") || die "problema abriendo el fichero human_kinases.swissprot
# Miramos linea por linea en la base de datos
while (<db>) {
chomp $_;
# Comprobamos si estamos en el campo ID
if ($_ =~ /^ID/) {
# Si estamos en el campo ID, recogemos el identificador
# fragmentando la linea por espacios
($a1,$id_db) = split (/\s+/,$_);
# Pero si no es el ID elegido, pasamos a la siguiente linea
next if ($id_db ne $id_query);
# Y si lo es, ponemos un marcador
$signal_good=1;
# Luego comprobamos si estamos en el campo de la secuencia
# y si, a la vez, el marcador estaba puesto a 1 (secuencia elegida)
# Si es asi, ponemos el marcador a 2, para asi recoger la secuencia
} elsif (($_ =~ /^SQ/) && ($signal_good==1)) {
$signal_good=2;
# Finalmente, si la marca es igual a 2, presentamos cada linea
# de la secuencia, hasta que la linea comience por //
# en cuyo caso nos salimos del bucle del while
} elsif ($signal_good == 2) {
last if ($_ =~ /^\/\//);
print "$_\n";
}
}
# Al salir del bucle, comprobamos si la marca existe
# y si no es asi, es que no hemos encontrado la secuencia elegida
# con lo que daremos un error
if (!$signal_good) {
print "ERROR: "."Secuencia no encontrada\n";
}
# Por ultimo, cerramos el fichero de la base de datos,
# que aun estaba abierto
close (db);
exit;
```

# Búsqueda de patrones de aminoácidos

```
#!/usr/bin/perl
# Buscador de patrones aminoacidicos
# Pide al usuario el patron aminoacidico a buscar
print "Por favor, introduce un patron para buscar en query.seq: ";
$patron = <STDIN>;
chomp $patron;
# Abrimos el fichero de la base de datos
# pero si no se puede, finaliza el programa
open (query, "query.seq") || die "problema abriendo el fichero query.seq\n";
# Miramos linea por linea la ficha SWISS-PROT
while (<query>) {
chomp $_;
# Cuando lleguemos al campo SQ, ponemos el marcador a 1

   if ($_ =~ /^SQ/) {

         $signal_seq = 1;
# Cuando lleguemos al final de secuencia, nos salimos del bucle

# Obsevar que esta expresion la ponemos antes de comprobar si

# el marcador=1, ya que esta linea no corresponde a la secuencia

# aminoacidica en si

   } elsif ($_ =~ /^\/\//) {

         last;
# Comprobamos si el marcador esta puesto a 1, y si es asi,

# eliminamos los espacios en blanco de la linea de secuencia

# y vamos concatenando cada linea en una nueva variable

# Para concatenar, tambien podriamos haber puesto:

# $secuencia_total.=$_;

   } elsif ($signal_seq == 1) {

         $_ =~ s/ //g;

         $secuencia_total=$secuencia_total.$_;

   }

  }
# Ahora comprobamos si la secuencia, recogida ya enteramente,

# contiene el patron dado

  if ($secuencia_total =~ /$patron/) {

   print "La secuencia query.seq contiene el patron $patron\n";

  } else {

   print "La secuencia query.seq no contiene el patron $patron\n";
```

```
  }
# Por ultimo, cerramos el fichero de la secuencia

# y salimos del programa

close (query);

exit;
```

Si queremos saber la posición exacta en donde ha encontrado el patrón, tenemos que hacer uso de un variable especial ‘$&’. Esta variable guarda el patrón encontrado después de evaluar una expresión regular (habría que ponerlo justo después de la línea ‘***if ($secuencia_total =~ /$patron/) {***‘. Además se puede combinar con las variables ‘$`’ y ‘$´’ que guardan todo lo que queda a la izquierda del patrón encontrado y a la derecha, respectivamente. Modifica el programa anterior con estas nuevas variables, para dar la posición exacta del patrón. Nota: También te puede ser útil la función length, que da la longitud de una cadena.

```
# tan solo hay que modificar la if en la que se encuentra el patron
# Ahora comprobamos si la secuencia, recogida ya enteramente,

# contiene el patron dado

# y damos su posicion en la secuencia

  if ($secuencia_total =~ /$patron/) {

   $posicion=length($`)+1;

   print "La secuencia query.seq contiene el patron $patron en la posicion $posicion\n";
  } else {

   print "La secuencia query.seq no contiene el patron $patron\n";

  }
```

# Calculo de las frecuencias de aminoácidos:

La frecuencia de los diferentes aminoácidos en las proteínas es variable, como consecuencia de sus distintas funciones o entornos preferidos. Así, en este ejemplo, veremos como calcular la frecuencia aminoacídica de una secuencia de aminoácidos dada.

```
#!/usr/bin/perl
# Calcula la frecuencia de aminoacidos en una secuencia proteica
# Recoge el nombre de fichero de la linea de comando
# (Ficha en formato SWISS-PROT)
# Tambien se puede pedir con un print, y recoger con <STDIN>
if (!$ARGV[0]) {print "La linea de ejecucion del programa debe ser: programa.pl ficha_swissprot\n
$fichero = $ARGV[0];
# Inicializamos la variable $errores
my $errores=0;
# Abrimos el fichero para lectura
open (FICHA, "$fichero") || die "problema abriendo el fichero $fichero\n";
```

```
# Primero recogemos la secuencia, similar a como lo hicimos en el ejemplo 2
while (<FICHA>) {
chomp $_;
if ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
   last if ($_ =~ /^\/\//);
   $_ =~ s/\s//g;
   $secuencia.=$_;
}
}
close (FICHA);
# Ahora usamos un bucle que coja todas las posiciones aminoacidicas
# de la secuencia (desde una funcion propia, que nos puede servir
# luego en otros programas)
comprueba_aa ($secuencia);
# Imprime los resultados por pantalla
# Primero los 20 aminoacidos y luego el array con sus frecuencias
# En este caso no se debe usar &lsquo;sort&rsquo; en el bucle foreach, ya que
# lo que contiene el array son las frecuencias (numeros)
print"A\tC\tD\tE\tF\tG\tH\tI\tK\tL\tM\tN\tP\tQ\tR\tS\tT\tV\tW\tY\n";
foreach $each_aa (@aa) {
print "$each_aa\t";
}
# Por ultimo da el numero de posibles errores
# y finaliza el programa
print "\nerrores = $errores\n";
exit;
#Subrutinas
# Esta subrutina calcula la frecuencia de cada aminoacido
# de una secuencia proteica
sub comprueba_aa {
# Recoge la secuencia
my ($secuencia)=@_;
# Y recorre aminoacido por aminoacido, con un bucle for que va
# desde 0 hasta la longitud de la secuencia
for ($posicion=0 ; $posicion<length $secuencia ; $posicion++ ) {
# Coge el aminoacido
$aa = substr($secuencia, $posicion, 1);
# Y ahora comprueba cual es, por una serie de if
# Cuando descubre cual es, suma uno al elemento correspondiente
# de un array, segun el subindice que le corresponde a cada
# aminoacido por su orden alfabetico
if ( $aa eq 'A' ) {
$aa[0]++;
} elsif ( $aa eq 'C' ) {
$aa[1]++;
} elsif ( $aa eq 'D' ) {
$aa[2]++;
} elsif ( $aa eq 'E' ) {
$aa[3]++;
} elsif ( $aa eq 'F' ) {
$aa[4]++;
} elsif ( $aa eq 'G' ) {
$aa[5]++;
} elsif ( $aa eq 'H' ) {
$aa[6]++;
} elsif ( $aa eq 'I' ) {
$aa[7]++;
} elsif ( $aa eq 'K' ) {
$aa[8]++;
} elsif ( $aa eq 'L' ) {
$aa[9]++;
} elsif ( $aa eq 'M' ) {
```

```
$aa[10]++;
} elsif ( $aa eq 'N' ) {
$aa[11]++;
} elsif ( $aa eq 'P' ) {
$aa[12]++;
} elsif ( $aa eq 'Q' ) {
$aa[13]++;
} elsif ( $aa eq 'R' ) {
$aa[14]++;
} elsif ( $aa eq 'S' ) {
$aa[15]++;
} elsif ( $aa eq 'T' ) {
$aa[16]++;
} elsif ( $aa eq 'V' ) {
$aa[17]++;
} elsif ( $aa eq 'W' ) {
$aa[18]++;
} elsif ( $aa eq 'Y' ) {
$aa[19]++;
# Si no encontramos el aminoacido, la letra no es correcta
# y sumamos un error
} else {
print "ERROR: No reconozco este aminacido: $aa\n";
$errores++;
}
}
# Finalmente retornamos el array de frecuencias
return @aa;
}
```

Ahora vamos a realizar el siguiente paso que sigue el flujo de información en una célula, tras la transcripción. Se trata de la traducción, por el cual una secuencia de ARN procedente de un gen, que era de ADN, pasa a proteinas o secuencia de aminoácidos. Para ello debemos hacer uso del código genético, que se basa en que a un triplete de ARN/ADN le corresponde un aminoácido. La secuencia la vamos a extraer de una ficha de un gen de *Escherichia coli*, en formato EMBL y luego comprobaremos la traducción con la existente en la ficha. Para este ejemplo, será necesario introducir las variables de arrays asociativos o tablas hash. En el programa hay que tener en cuenta que solo hay que recoger la región codificarte, incluida en el campo ‘FT CDS.

```
#!/usr/bin/perl
# Traduce una secuencia de ADN de una ficha EMBL
# en su correspondiente de aminoacidos
# Recoge el nombre de fichero de la linea de comando
# (Ficha en formato SWISS-PROT)
# Tambien se puede pedir con un print, y recoger con <STDIN>
if (!$ARGV[0]) {print "La linea de ejecucion del programa debe ser: programa.pl ficha_embl\n";}
$fichero = $ARGV[0];
# Abrimos el fichero para lectura
open (FICHA, "$fichero") || die "problema abriendo el fichero $fichero\n";
# Primero recogemos la secuencia, similar a como
# lo hicimos en el ejemplo 2
while (<FICHA>) {
chomp $_;
if ($_ =~ /^FT CDS/) {
$_ =~ tr/../ /;
($a1,$a2,$a3,$a4) = split (" ",$_);
}
elsif ($_ =~ /^SQ/) {
```

```
$signal_good = 1;
} elsif ($signal_good == 1) {
last if ($_ =~ /^\/\//);
# Eliminamos los espacios y los numeros
$_ =~ tr/0-9/ /;
$_ =~ s/\s//g;
$secuencia.=$_;
}
}
close (FICHA);
# Ahora definimos un array asociativo con la correspondencia
# de todos los aminoacidos con su/sus tripletes de nucleotidos
# correspondientes (tambien se puede poner en una funcion propia,
# por si este mismo codigo genetico
# lo usamos en distintos programas
my(%codigo_genetico) = (
'TCA' => 'S',# Serina
'TCC' => 'S',# Serina
'TCG' => 'S',# Serina
'TCT' => 'S',# Serina
'TTC' => 'F',# Fenilalanina
'TTT' => 'F',# Fenilalanina
'TTA' => 'L',# Leucina
'TTG' => 'L',# Leucina
'TAC' => 'Y',# Tirosina
'TAT' => 'Y',# Tirosina
'TAA' => '*',# Stop
'TAG' => '*',# Stop
'TGC' => 'C',# Cisteina
'TGT' => 'C',# Cisteina
'TGA' => '*',# Stop
'TGG' => 'W',# Triptofano
'CTA' => 'L',# Leucina
'CTC' => 'L',# Leucina
'CTG' => 'L',# Leucina
'CTT' => 'L',# Leucina
'CCA' => 'P',# Prolina
'CCC' => 'P',# Prolina
'CCG' => 'P',# Prolina
'CCT' => 'P',# Prolina
'CAC' => 'H',# Histidina
'CAT' => 'H',# Histidina
'CAA' => 'Q',# Glutamina
'CAG' => 'Q',# Glutamina
'CGA' => 'R',# Arginina
'CGC' => 'R',# Arginina
'CGG' => 'R',# Arginina
'CGT' => 'R',# Arginina
'ATA' => 'I',# Isoleucina
'ATC' => 'I',# Isoleucina
'ATT' => 'I',# Isoleucina
'ATG' => 'M',# Methionina
'ACA' => 'T',# Treonina
'ACC' => 'T',# Treonina
'ACG' => 'T',# Treonina
'ACT' => 'T',# Treonina
'AAC' => 'N',# Asparagina
'AAT' => 'N',# Asparagina
'AAA' => 'K',# Lisina
'AAG' => 'K',# Lisina
'AGC' => 'S',# Serina
'AGT' => 'S',# Serina
'AGA' => 'R',# Arginina
'AGG' => 'R',# Arginina
```

```
'GTA' => 'V',# Valina
'GTC' => 'V',# Valina
'GTG' => 'V',# Valina
'GTT' => 'V',# Valina
'GCA' => 'A',# Alanina
'GCC' => 'A',# Alanina
'GCG' => 'A',# Alanina
'GCT' => 'A',# Alanina
'GAC' => 'D',# Acido Aspartico
'GAT' => 'D',# Acido Aspartico
'GAA' => 'E',# Acido Glutamico
'GAG' => 'E',# Acido Glutamico
'GGA' => 'G',# Glicina
'GGC' => 'G',# Glicina
'GGG' => 'G',# Glicina
'GGT' => 'G',# Glicina
);
# Traduce cada codon in su aminoacido correspondiente
# y lo va sumando a la secuencia proteica
print $a3;
for($i=$a3 - 1; $i < $a4 - 3 ; $i += 3) {
$codon = substr($secuencia,$i,3);
# Pasamos el codon de minusculas (formato EMBL) a mayusculas
$codon =~ tr/a-z/A-Z/;
$protein.= codon2aa($codon);
}
print "La secuencia de proteinas de este gen:\n$secuencia\nes la siguiente:\n$protein\n\n";
exit;
```

# Referencias Bibliográficas:


http://bioperl.org/

http://changjiang.whlib.ac.cn/pylorus/download/book/Beginning%20Perl%20for%20Bioinformatics/contents.html

http://www.unix.org.ua/orelly/perl/prog3/


***Archivos de ejemplo:***

– human_kinases_swissprot.txt
– query_seq.txt
– ecoli_embl.txt