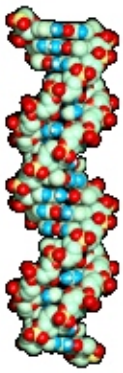
LINUX & PERL: Werkzeuge für das Studium und die Analyse von biologischen Informationen

von Carlos Andrés Pérez
<caperez/at/usc.edu.co>

Über den Autor:

Carlos Andrés Pérez ist Spezialist für Molekularsimulation und Doktorand der Biotechnologie. Technischer Berater der Grupo de Investigación en Educación Virtual (GIEV) [Forschungsgruppe für e−Learning]. Adresse: Universidad Santiago de Cali, Calle 5ª carrera 62 Campus Pampalinda, Cali – Colombia.

Übersetzt ins Deutsche von:
Viktor Horvath
<ViktorHorvath(at)gmx.net>



Zusammenfassung:

Dieser Artikel will einige der Vorteile zeigen, die Perl−Programmierung unter Unix für die Auswertung biologischer DNS−, RNS− und Proteinsequenz−Datenbanken bietet, in Vergleichsprozessen oder Analyse. Das Humangenomprojekt und die DNS−Klontechniken haben den Fortschritt der Wissenschaft auf diesem Gebiet beschleunigt. Die täglich generierten Daten übersteigen häufig die Kapazitäten, sie unter dem Gesichtspunkt der Evolution zu verarbeiten.

Die schnelle Vermehrung der biologischen Informationen zu verschiedenen Genomen (Summen der Gene eines Organismus) treibt die Bioinformatik als eine fundamentale Disziplin für die Handhabung und Analyse dieser Daten an.

Bioinformatik

Die Bioinformatik entstand, als Wissenschaftler Gensequenzen digital zu speichern und mittels Programmen zu vergleichen begannen. Lange war Bioinformatik auf die Sequenzanalyse beschränkt. Die Wichtigkeit, die Molekülstruktur festzustellen, sorgte jedoch dafür, daß Computer zu einem wichtigen Werkzeug für Untersuchungen in theoretischer Biochemie wurden. Täglich gibt es mehr Informationen und mehr Datensammlungen zur dreidimensionalen Molekülstruktur. Gene werden nicht mehr einzeln, sondern im Ganzen oder in einem größeren Teil untersucht. Es ist jetzt einfacher zu verstehen, wie sie sich untereinander

und in bezug auf die Proteine verhalten und wie sie sich in Stoffwechselwegen organisieren. Jedes Mal werden wir uns mehr darüber bewußt, wie wichtig es ist, die Daten zu organisieren.

Jede der beschriebenen Aktivitäten hat mindestens zwei Gesichtspunkte, unter denen sie interessant ist. Einerseits ist es das biologische Interesse, die Beziehungen zwischen organischen Molekülen zu kennen, und andererseits gerät der Zusammenbau zu einem interessanten Problem des Softwaredesign. Biologische Informationen müssen kombiniert und integriert werden, um eine globale und effektive Sicht der zugrundeliegenden biologischen Prozesse zu erhalten. Wir haben selber die Notwendigkeit bemerkt, für eine wirksame Lösung die verschiedenen Felder der Informatik zusammenzuführen: Datenbankverwaltung, auch Datenintegration, effiziente Algorithmen, leistungsfähige Hardware – Parallelrechner, Multiprozessorsysteme etc.

Perl

Larry Wall begann die Entwicklung von Perl 1986. Perl ist eine interpretierte Programmiersprache, ideal zur Handhabung von Text, Dateien und Prozessen. Perl erlaubt die schnelle Entwicklung kleiner Programme. Man könnte sagen, daß Perl eine optimierte Mischung einer Hochsprache (z.B. C) und einer Skriptsprache (z.B. bash) ist.

Perl−Programme können unter verschiedenen Betriebssystemen oder Plattformen laufen. Es ist jedoch unter den UNIX−Systemen entstanden und hat sich dort verbreitet. Perl hat seinen anfänglichen Bereich endgültig verlassen, seit es als Sprache für Web−Applikationen eingesetzt wird. Vor Perl waren ***awk***, ***thirst*** und ***grep*** die Werkzeuge zur Dateianalyse und der Extraktion von Informationen.

Perl vereinigte die Fähigkeiten dieser UNIX−Werkzeuge in einem einzigen Programm, wobei jedes mit mehr Funktionalität erweitert und modernisiert wurde.

Perl ist eine freie Programmiersprache, und es läuft auf jedem Betriebssystem, das in biologischen Forschungslaboren gängig ist. Unter UNIX und Mac OS X ist es schon vorinstalliert, ansonsten muß man es installieren. Es genügt, die passende Version von dieser Seite zu laden: http://www.cpan.org

Unter Linux wird Perl mit der Datei als Argument aufgerufen, die die auszuführenden Befehle enthält. Eine andere, häufig verwendete Methode gestattet es, direkt diese Datei aufzurufen. Dafür müssen wir zweierlei machen: (a) einen besonderen Kommentar als erste Zeile in das Programm schreiben:

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

print "Hi\n";
```

und (b) die Datei unter Unix ausführbar machen:

```
% chmod +x greetings.pl
```

Dann können wir die Programmdatei einfach unter ihrem Namen aufrufen.

Perl−Dateiverwaltung

Wenn wir eine Datenbank von Molekularsequenzen in Textformat haben, können wir mit Perl eine Sequenz−Suchmaschine schreiben. Dieses Beispiel zeigt, wie man nach einer Proteinsequenz in einer Datenbank im Format SWISS−PROT (db_human_swissprot) durch Angabe ihrer ID sucht.

```
#!/usr/bin/perl

# Suche nach einer Aminosäure-Sequenz in einer Datenbank
# im Format SWISS-PROT, mit einem gegebenen ID-Code.
# Frage nach dem ID-Code und speichere ihn
# von der Eingabe (stdin) in eine Variable

print "Enter the ID to search: ";
$id_query=<STDIN>;
chomp $id_query;

# Wir öffnen die Datenbank-Datei.
# Wenn das unmöglich ist, endet das Programm

open (db, "human_kinases_swissprot.txt") ||
 die "problem opening the file human_kinases_swissprot.txt\n";

# Suche Zeile für Zeile in der Datenbank

while (<db>) {
chomp $_;

# Prüfe, ob wir in einem ID-Feld sind

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

# Wenn ja, sammeln wir die Information
# und trennen die Zeile bei den Leerzeichen

($a1,$id_db) = split (/\s+/,$_);

# Wenn das ID-Feld nicht das gesuchte ist,
# gehen wir zum nächsten

next if ($id_db ne $id_query);

# Wenn wir darauf stoßen, setzen wir eine Marke

$signal_good=1;

# Dann prüfen wir, ob es sich um das Sequenz-Feld handelt
# und ob die Marke 1 ist (gesuchte Sequenz)
# Wenn ja, ändern wir die Marke auf 2, um die Sequenz zu sammeln

} elsif (($_ =~ /^SQ/) && ($signal_good==1)) {
$signal_good=2;

# Wenn die Marke schließlich 2 ist, zeigen wir jede Zeile
# der Sequenz an, außer wenn die Zeile mit // beginnt
# In dem Fall verlassen wir die while-Schleife

} elsif ($signal_good == 2) {
last if ($_ =~ /^\/\//);
print "$_\n";
}
}

# Wenn wir die while-Schleife verlassen haben, prüfen wir die Marke.
# Ist der Test negativ, heißt das, daß wir die gesuchte Sequenz
```

```
# nicht gefunden haben und wir einen Fehler ausgeben.

if (!$signal_good) {
print "ERROR: "."Sequence not found\n";
}

# Als letztes schließen wir die noch offene Datei

close (db);
exit;
```

Suche nach Aminosäure–Mustern

```
#!/usr/bin/perl
# Suchmaschine für Aminosäure-Muster
# Frage den Nutzer nach den zu suchenden Mustern

print "Please, introduce the pattern to search in query.seq: ";
$patron = <STDIN>;
chomp $patron;

# Wir öffnen die Datenbank-Datei.
# Wenn das unmöglich ist, endet das Programm

open (query, "query_seq.txt") || die "problem opening the file query_seq.txt\n";

# Suche Zeile für Zeile nach der SWISS-PROT-Sequenz

while (<query>) {
chomp $_;

# Wenn wir beim SQ-Feld ankommen, Marke auf 1 setzen

   if ($_ =~ /^SQ/) {
         $signal_seq = 1;

# Wenn wir am Ende der Sequenz ankommen, verlasse die Schleife.
# Dieser Ausdruck muß vor dem Test, ob die Marke 1 ist, stehen,
# denn diese Zeile gehört nicht zur Aminosäure-Sequenz.

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

# Teste, ob die Marke 1 ist. Wenn ja, lösche die Leerzeichen
# in der Sequenzzeile und vereinige jede Zeile in einer neuen Variable.
# Zum Zusammenfügen können wir auch folgendes machen:
# $secuencia_total.=$_;

   } elsif ($signal_seq == 1) {
         $_ =~ s/ //g;
         $secuencia_total=$secuencia_total.$_;
   }
  }

# Jetzt vergleiche die komplett zusammengefügte Sequenz
# mit dem gesuchten Muster

  if ($secuencia_total =~ /$patron/) {
   print "The sequence query.seq contains the pattern $patron\n";
```

```
  } else {
   print "The sequence query.seq doesn't contain the pattern $patron\n";
  }

# Als letztes schließen wir die noch offene Datei

close (query);
exit;
```

Falls wir die genaue Fundstelle des Musters erfahren wollen, müssen wir eine bestimmte Variable `$&` benutzen. Sie behält das gefundene Muster nach der Auswertung eines regulären Ausdrucks (man müßte sie direkt nach die Zeile `if ($$secuencia_total>= ~/$$patron>/) {` schreiben. Das können wir mit den Variablen `$ ` ` und `$ ´` verbinden, die alles links und rechts vom gefundenen Muster speichern. Hier die Modifikation des vorigen Programms mit diesen neuen Variablen, um die genaue Position des Musters anzugeben. Hinweis: Auch die Funktion `length` kann nützlich sein, die die Länge einer Zeichenkette liefert.

```
# Wir müssen nur das "if" da ändern, wo das Muster gefunden wurde.
# Jetzt vergleiche die komplett zusammengefügte Sequenz
# mit dem gesuchten Muster und prüfe dessen Position in der Sequenz

  if ($secuencia_total =~ /$patron/) {
   $posicion=length($`)+1;
   print "The sequence query_seq.txt contains the pattern $patron in the following position $posi
  } else {
  print "The sequence query_seq.txt doesn't contain the pattern $patron\n";
  }
```

Berechnung von Häufigkeiten der Aminosäuren

Die Häufigkeiten der einzelnen Aminosäuren in einem Protein ist unterschiedlich, je nach seiner Funktion oder bevorzugten Umgebung. Daher sehen wir in diesem Beispiel, wie man die Häufigkeiten der Aminosäuren in einer gegebenen Aminosäuren–Sequenz berechnet.

```
#!/usr/bin/perl
# Berechnet die Häufigkeiten der Aminosäuren in einer Proteinsequenz
# Erhält den Dateinamen von der Kommandozeile (Format SWISS-PROT)
# Kann auch mit "print" von <stdin> erfragt werden

if (!$ARGV[0]) {print "The execution line shall be: program.pl file_swissprot\n";}
$fichero = $ARGV[0];

# Initialisiere die Variable $errores

my $errores=0;

# Öffne die Datei zum Lesen

open (FICHA, "$fichero") || die "problem opening the file $fichero\n";

# Zuerst überprüfen wir die Sequenz wie im zweiten Beispiel

while (<FICHA>) {
chomp $_;
if ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
```

```
   last if ($_ =~ /^\/\//);
   $_ =~ s/\s//g;
   $secuencia.=$_;
}
}
close (FICHA);

# Jetzt verwenden wir eine Schleife, die jede Position einer Aminosäure
# in der Sequenz testet (in einer eigenen Funktion, die später in anderen
# Programmen benutzt werden kann)

comprueba_aa ($secuencia);

# Gib die Ergebnisse auf dem Schirm aus
# Zuerst die 20 Aminosäuren und dann das Array mit ihren Häufigkeiten
# In diesem Fall kann "sort" nicht in "foreach" benutzt werden,
# denn das Array enthält die Häufigkeiten als Zahlen.

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";
}

# Dann gibt das Programm mögliche Fehler zurück und terminiert

print "\nerrores = $errores\n";
exit;


# Funktionen:
# Diese berechnet die Häufigkeit
# einer jeden Aminosäure einer Proteinsequenz

sub comprueba_aa {

# Sequenz nehmen

my ($secuencia)=@_;

# und jede Aminosäure ansehen, a läuft von 0 bis zur Sequenzlänge

for ($posicion=0 ; $posicion<length $secuencia ; $posicion++ ) {

# Aminosäure nehmen

$aa = substr($secuencia, $posicion, 1);

# und in einer Bedingung prüfen, welche es ist
# Bei Übereinstimmung wird die entsprechende Häufigkeit um 1 erhöht
# in einem Array, das für jede Aminosäure eine Variable bereithält,
# alphabetisch geordnet.

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]++;

# Wenn die Aminosäure nicht gefunden wurde,
# wird die Fehlerzahl inkrementiert

} else {
print "ERROR: Aminoacid not found: $aa\n";
$errores++;
}
}

# Schließlich das Array mit den Häufigkeiten zurückgeben

return @aa;
}
```

Jetzt werden wir den Schritt betrachten, der nach dem Informationsfluß und der Transkription in einer Zelle geschieht. Das ist die Translation, durch die eine RNS−Sequenz, die von einem aus DNS bestehenden Gen kommt, zu einem Baustein eines Proteins oder einer Aminosäure−Sequenz wird. Dafür müssen wir den genetischen Code kennen, der ein RNS/DNS−Tripel einer Aminosäure zuordnet. Die Sequenz extrahieren wir aus der EMBL−formatierten Datei eines Gens von *Escherichia coli*, und bald werden wir die Übersetzung mit der in der Datei existierenden überprüfen. Für dieses Beispiel müssen wir ein „Associate Array“ (Hashtabelle) einführen. Im Programm wird nur der Kodierbereich gebraucht, der sich im Feld „FT CDS“ befindet.

```
#!/usr/bin/perl
# Übersetzt eine DNS-Sequenz von einer EMBL-Datei
# zur korrespondierenden Aminosäure
# Erhält den Dateinamen von der Kommandozeile (Format SWISS-PROT)
# Kann auch mit "print" von <stdin> erfragt werden

if (!$ARGV[0]) {print "The program line shall be: program.pl ficha_embl\n";}
$fichero = $ARGV[0];
```

```
# Öffne die Datei zum Lesen

open (FICHA, "$fichero") || die "problem opening the file $fichero\n";

# Zuerst überprüfen wir die Sequenz wie im zweiten Beispiel

while (<FICHA>) {
chomp $_;
if ($_ =~ /^FT CDS/) {
$_ =~ tr/../ /;
($a1,$a2,$a3,$a4) = split (" ",$_);
}
elsif ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
last if ($_ =~ /^\/\//);

# Zahlen und Leerzeichen löschen

$_ =~ tr/0-9/ /;
$_ =~ s/\s//g;
$secuencia.=$_;
}
}
close (FICHA);

# Hier definieren wir ein "Associate Array", das
# Aminosäuren und Nukleotide zuordnet (ebenfalls in
# einer eigenen Funktion für den Fall, daß derselbe Code
# in einem anderen Programm benutzt wird)

my(%codigo_genetico) = (
'TCA' => 'S',# Serin
'TCC' => 'S',# Serin
'TCG' => 'S',# Serin
'TCT' => 'S',# Serin
'TTC' => 'F',# Phenylalanin
'TTT' => 'F',# Phenylalanin
'TTA' => 'L',# Leucin
'TTG' => 'L',# Leucin
'TAC' => 'Y',# Tyrosin
'TAT' => 'Y',# Tyrosin
'TAA' => '*',# stop
'TAG' => '*',# stop
'TGC' => 'C',# Cystein
'TGT' => 'C',# Cystein
'TGA' => '*',# stop
'TGG' => 'W',# Tryptophan
'CTA' => 'L',# Leucin
'CTC' => 'L',# Leucin
'CTG' => 'L',# Leucin
'CTT' => 'L',# Leucin
'CCA' => 'P',# Prolin
'CCC' => 'P',# Prolin
'CCG' => 'P',# Prolin
'CCT' => 'P',# Prolin
'CAC' => 'H',# Histidin
'CAT' => 'H',# Histidin
'CAA' => 'Q',# Glutamin
'CAG' => 'Q',# Glutamin
'CGA' => 'R',# Arginin
'CGC' => 'R',# Arginin
'CGG' => 'R',# Arginin
```

```
'CGT' => 'R',# Arginin
'ATA' => 'I',# Isoleucin
'ATC' => 'I',# Isoleucin
'ATT' => 'I',# Isoleucin
'ATG' => 'M',# Methionin
'ACA' => 'T',# Threonin
'ACC' => 'T',# Threonin
'ACG' => 'T',# Threonin
'ACT' => 'T',# Threonin
'AAC' => 'N',# Asparagin
'AAT' => 'N',# Asparagin
'AAA' => 'K',# Lysin
'AAG' => 'K',# Lysin
'AGC' => 'S',# Serin
'AGT' => 'S',# Serin
'AGA' => 'R',# Arginin
'AGG' => 'R',# Arginin
'GTA' => 'V',# Valin
'GTC' => 'V',# Valin
'GTG' => 'V',# Valin
'GTT' => 'V',# Valin
'GCA' => 'A',# Alanin
'GCC' => 'A',# Alanin
'GCG' => 'A',# Alanin
'GCT' => 'A',# Alanin
'GAC' => 'D',# Asparaginsäure
'GAT' => 'D',# Asparaginsäure
'GAA' => 'E',# Glutaminsäure
'GAG' => 'E',# Glutaminsäure
'GGA' => 'G',# Glycin
'GGC' => 'G',# Glycin
'GGG' => 'G',# Glycin
'GGT' => 'G',# Glycin
);

# Übersetze jedes Codon in die entsprechende Aminosäure
# und füge sie der Proteinsequenz hinzu

print $a3;
for($i=$a3 - 1; $i < $a4 - 3 ; $i += 3) {
$codon = substr($secuencia,$i,3);

# Schreibe das Codon groß (statt klein wie im EMBL-Format)

$codon =~ tr/a-z/A-Z/;
$protein.= codon2aa($codon);
}
print "This proteinic sequence of the gen:\n$secuencia\nis the following:\n$protein\n\n";
exit;
```

Referenzen

- http://bioperl.org/
- http://changjiang.whlib.ac.cn/pylorus/download/book/Beginning%20Perl%20for%20Bioinformatics/contents.h
- http://www.unix.org.ua/orelly/perl/prog3/

- ***Beispieldateien:***
  - human_kinases_swissprot.txt
  - query_seq.txt
  - ecoli_embl.txt

Der LinuxFocus Redaktion schreiben © Carlos Andrés Pérez "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Autoren und Übersetzer: es −−> −− : Carlos Andrés Pérez <caperez/at/usc.edu.co> en −−> es: Carlos Andrés Pérez <caperez/at/usc.edu.co> en −−> de: Viktor Horvath <ViktorHorvath(at)gmx.net>

2005−05−23, generated by lfparser_pdf version 2.51