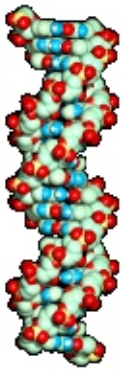
LINUX & PERL, computer tools for study and analysis of biological information

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

About the author:

Carlos Andrés Pérez is specialist in Mollecular Simulation, Candidate to Doctor in Biotechnolgy. Technical advisor of the Grupo de Investigación en Educación Virtual (GIEV) – Research Group in Virtual Learning. Address: Universidad Santiago de Cali, Calle 5ª carrera 62 Campus Pampalinda, Cali – Colombia.



Abstract:

This article wants to show some of the advantages of Perl programming on Unix, for extraction of the biological information of the DNA, RNA and proteine sequences Databases. They can be used in comparative processes or analysis. The Human Genome project and the DNA clonation techniques have accelerated the scientific progress in this area. Daily generated information in this field outgrows often the capability of processing this information from an evolutive viewpoint.

The fast proliferation of the biological information on different genomes (dowry of genes of an organism) is driving bioinformatics as one fundamental discipline for the handling and analysis of these data.

Bioinformatics

Bioinformatics was born when scientists began to store the biological sequences in a digital format and the first programs to compare them arose. For a long time bioinformatics was limited to the analysis of sequences. Nevertheless, the importance to establish the structural form of molecules caused that computers become an important tool for investigation in theoretical biochemistry. Each day there is more information and more collections of data on the 3D conformation of molecules. Genes changed from being studied in an individual way to be studied over the complete or an extense part of them. It's now easier to understand how they behave between them, the the proteines and how they organize in the metabolic pathways. Every time we are more conscious of how important is to organize the data.

Each one of the described activities has at least two faces from which they are interesting. On one hand the biological interest is to know the relations between life molecules, and on the other hand the assembly becomes an interesting software design problem to solve. The necessity is to combine and to integrate the biological information to obtain a global and effective vision of the biological processes that lies there. We have also noticed ourselves of the necessity to combine the different areas in computer science to come to an effective solution. One is management of data bases, also data integration; efficient algorithms, powerful hardware – grids, multiprocessors, etc.

Perl

Larry Wall began the development of Perl in 1986. Perl is an interpreted programming language, ideal to manipulate texts, files and processes. Perl allows to quickly develop small programs. It could be said that Perl is an optimized mixture of a high–level language (for example C) and a scripting language (for example bash).

Perl programs can run on several operating systems / platforms. However, where Perl was born and where it has spread is under the UNIX operating systems. Perl fully exceeded its initial scope thanks to the impulse that it received trought the immediate use as a web applications language. Before Perl was used***awk***,***thirst*** and***grep*** were the tools to analyze files and to extract information.

Perl reunited the possibilities of these UNIX tools in a single program extending and modernizing each one with more functionality.

Perl is a free programming language and it is possible to be run in any of the operating systems that are generally present in the biological research laboratories. Under UNIX and MacOSX it comes pre–installed, in others is necessary to install perl. It is enough to obtain it from the site:http://www.cpan.org for the system that we are using.

The programs in Perl under Linux are called with the name of the file that contains the instructions to execute. The instructions are keep in a file and Perl is invoked with the name of the file as argument.

Another frequent method is to keep the Perl instructions in a file but without invoking perl with the file as argument. For that we must make two things: (a) to put a special comment at the first line of the program:

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

print "Hi\n";
```

and (b) store the file and assign it the UNIX properties for execution:

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

Once made this, the file program can be used by just calling it with the file name.

Perl File Management:

When we have a database of molecular sequences in text format, we can make in Perl a sequence search tool. In this example we see how to search for a proteine sequence in a database with SWISS–PROT format (db_human_swissprot), using its id code.

```
#!/usr/bin/perl

# Look for aminoacid sequence in a database

# SWISS-PROT formated, with a given id code

# Ask for the code in the ID field

# and it assigns it from the input(STDIN)to a variable

print "Enter the ID to search: ";
$id_query=<STDIN>;
chomp $id_query;
# We open the database file

# but if it isn't possible the program ends

open (db, "human_kinases_swissprot.txt") ||
 die "problem opening the file human_kinases_swissprot.txt\n";
# Look line by line in the database

while (<db>) {
chomp $_;
# Check if we are in the ID field
if ($_ =~ /^ID/) {
# If it is possitive we gather the information

# breaking the line by spaces

($a1,$id_db) = split (/\s+/,$_);
# but if there is no coincidence of ID we continue to the following

next if ($id_db ne $id_query);
# When they coincide, we put a mark

$signal_good=1;
# Then we check the sequence field

# and if the mark is 1 (chosen sequence)
# If possitive, we change the mark to 2,to collect the sequence

} elsif (($_ =~ /^SQ/) && ($signal_good==1)) {
$signal_good=2;
# Finally, if the mark is 2, we present each line

# of the sequence, until the line begins with //
# is such case we broke the while
} elsif ($signal_good == 2) {
last if ($_ =~ /^\/\//);
print "$_\n";
}
}
# When we left the while instruction we check the mark

# if negative that means that we don't find the chosen sequence

# that will give us an error

if (!$signal_good) {
print "ERROR: "."Sequence not found\n";
}
# Finally, we close the file
# that still si open
```

```
close (db);
exit;
```

Search for aminoacid patterns

```
#!/usr/bin/perl
# Searcher for aminoacid patterns
# Ask the user the patterns for search
print "Please, introduce the pattern to search in query.seq: ";
$patron = <STDIN>;
chomp $patron;
# Open the database file
# but if it can't it ends the program
open (query, "query_seq.txt") || die "problem opening the file query_seq.txt\n";
# Look line by line the SWISS-PROT sequence
while (<query>) {
chomp $_;
# When arrives to the SQ field,put the mark in 1

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

         $signal_seq = 1;
# When arrive to the end of sequence, leave the curl

# Check that this expression is put before to check

# the mark=1,because this line doesn't belong to the aminoacid sequence

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

         last;
# Check the mark if it is equal to 1, if possitive

# eliminate the blank spaces in the sequence line

# and join every line in a new variable

# To concatenate, we also can do:

# $secuencia_total.=$_;

   } elsif ($signal_seq == 1) {

         $_ =~ s/ //g;

         $secuencia_total=$secuencia_total.$_;

   }

  }
# Now check the sequence, collected in its entirety,

# for the given pattern

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

   print "The sequence query.seq contains the pattern $patron\n";

  } else {
```

```
   print "The sequence query.seq doesn't contains the pattern $patron\n";

  }
# Finally we close the file

# and leave the program

close (query);

exit;
```

If we want to know the exact position where it has found the pattern, we must make use of a special variable `$&'. This variable keeps the pattern found after evaluating a regular expression (would be necessary to put it just after the line ` ***if ($$secuencia_total>= ~/$$patron>/) {*** ` . In addition is possible to combine with the variables ` $ ` ' and ` $ ´ ' that store everything in the left and right of the found pattern. It modifies the previous program with these new variables, to give the exact position of the pattern. Note: Also you can find useful the ***length*** function, that gives the length of a chain.

```
# Only we need to change the if where the pattern was found
# Now check the sequence, collected in its entirety,

# for the given pattern

# and check its position in the sequence

  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 contains the pattern $patron\n";

  }
```

Calculus of aminoacid frequences:

The frequency of the different aminoacid in proteins is variable, as a result of its different functions or favourite surroundings. Thus, in this example, we will see how to calculate the aminoacide frequency of a given sequence of aminoacid.

```
#!/usr/bin/perl
# Calculates the frequency of aminoacid in a proteinic sequence
# Gets the file name from the command line
# (SWISS-PROT formatted)
# Also can be asked with print from the <STDIN>
if (!$ARGV[0]) {print "The execution line shall be: program.pl file_swissprot\n";}
$fichero = $ARGV[0];
# Initialize the variable $errores
my $errores=0;
# Open the file for reading
```

```
open (FICHA, "$fichero") || die "problem opening the file $fichero\n";
# First we check the sequence as did in the example 2
while (<FICHA>) {
chomp $_;
if ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
   last if ($_ =~ /^\/\//);
   $_ =~ s/\s//g;
   $secuencia.=$_;
}
}
close (FICHA);
# Now use a curl that checks every position of the aminoacid
# in the sequence (from a funcion of its own,that can be used after in other
# programs)
comprueba_aa ($secuencia);
# Print the results to the screen
# First the  20 aminoacids and then the array with their frequencies
# In this case 'sort' can't be used in foreach,
# because the array contains the frequencies (numbers)
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";
}
# Ten it gives the possible errors
# and ends the program
print "\nerrores = $errores\n";
exit;
# Functions
# This one calculates each aminoacid frequency
# from a proteinic sequence
sub comprueba_aa {
# Gets the sequence
my ($secuencia)=@_;
# and runs aminoacid by aminoacid, using a for running
# from 0 until the sequence length
for ($posicion=0 ; $posicion<length $secuencia ; $posicion++ ) {
# Gets the aminoacid
$aa = substr($secuencia, $posicion, 1);
# and checks which one is using if
# when it is checked it aggregates 1 to the correspondant frequency
# in an array using a pointer for each one
# ordered in alphabetic way
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]++;
# If the aminoacid is not found
# it aggregates 1 to the errors
} else {
print "ERROR: Aminoacid not found: $aa\n";
$errores++;
}
}
# Finally returns to the frequency array
return @aa;
}
```

Now we are going to make the following step that follows the flow of information in a cell, after the transcription. One is the translation, by which a sequence of ARN coming from a gene, that was of DNA, passes to proteins or aminoacid sequences. For that we must use the genetic code, that is based on which triplets of ARN/ADN correspond to an aminoacid. The sequence that we are going to extract of a card of a gene of*Escherichia coli*, in format EMBL and soon we will verify the translation with the existing one in the card. For this example, it will be necessary to introduce the associative variables of arrays or tables hash. In the program we should consider than only is needed the codificarte area, included in the 'FT CDS field.

```
#!/usr/bin/perl
# Translates an ADN sequence from an EMBL fiche
# to the aminoacid correspondant
# Gets the file name from the command line
# (SWISS-PROT formatted)
# Also can be asked with print from the <STDIN>
if (!$ARGV[0]) {print "The program line shall be: program.pl ficha_embl\n";}
$fichero = $ARGV[0];
# Open the file for reading
open (FICHA, "$fichero") || die "problem opening the file $fichero\n";
# First we check the sequence as did in the example 2
while (<FICHA>) {
chomp $_;
if ($_ =~ /^FT CDS/) {
$_ =~ tr/../ /;
($a1,$a2,$a3,$a4) = split (" ",$_);
}
elsif ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
last if ($_ =~ /^\/\//);
```

```
# Eliminate numbers and spaces
$_ =~ tr/0-9/ /;
$_ =~ s/\s//g;
$secuencia.=$_;
}
}
close (FICHA);
# Now we define an associate array with the correpondence
# of every aminoacids with their nucleotide
# correspondants (also in an own function,
# for if the same genetic code is used in other program
my(%codigo_genetico) = (
'TCA' => 'S',# Serine
'TCC' => 'S',# Serine
'TCG' => 'S',# Serine
'TCT' => 'S',# Serine
'TTC' => 'F',# Fenilalanine
'TTT' => 'F',# Fenilalanine
'TTA' => 'L',# Leucine
'TTG' => 'L',# Leucine
'TAC' => 'Y',# Tirosine
'TAT' => 'Y',# Tirosine
'TAA' => '*',# Stop
'TAG' => '*',# Stop
'TGC' => 'C',# Cysteine
'TGT' => 'C',# Cysteine
'TGA' => '*',# Stop
'TGG' => 'W',# Tryptofane
'CTA' => 'L',# Leucine
'CTC' => 'L',# Leucine
'CTG' => 'L',# Leucine
'CTT' => 'L',# Leucine
'CCA' => 'P',# Proline
'CCC' => 'P',# Proline
'CCG' => 'P',# Proline
'CCT' => 'P',# Proline
'CAC' => 'H',# Hystidine
'CAT' => 'H',# Hystidine
'CAA' => 'Q',# Glutamine
'CAG' => 'Q',# Glutamine
'CGA' => 'R',# Arginine
'CGC' => 'R',# Arginine
'CGG' => 'R',# Arginine
'CGT' => 'R',# Arginine
'ATA' => 'I',# IsoLeucine
'ATC' => 'I',# IsoLeucine
'ATT' => 'I',# IsoLeucine
'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',# Serine
'AGT' => 'S',# Serine
'AGA' => 'R',# Arginine
'AGG' => 'R',# Arginine
'GTA' => 'V',# Valine
'GTC' => 'V',# Valine
'GTG' => 'V',# Valine
'GTT' => 'V',# Valine
```

```
'GCA' => 'A',# Alanine
'GCC' => 'A',# Alanine
'GCG' => 'A',# Alanine
'GCT' => 'A',# Alanine
'GAC' => 'D',# Aspartic Acid
'GAT' => 'D',# Aspartic Acid
'GAA' => 'E',# Glutamic Acid
'GAG' => 'E',# Glutamic Acid
'GGA' => 'G',# Glicine
'GGC' => 'G',# Glicine
'GGG' => 'G',# Glicine
'GGT' => 'G',# Glicine
);
# Translate every codon in its correspondant aminoacid
# and aggregates to the proteinic sequence
print $a3;
for($i=$a3 - 1; $i < $a4 - 3 ; $i += 3) {
$codon = substr($secuencia,$i,3);
# Pass the codon from subcase (EMBL format) to uppercase
$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;
```

Bibliographic References

- 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/
- ***Example files :***
  - human_kinases_swissprot.txt
  - query_seq.txt
  - ecoli_embl.txt

Webpages maintained by the LinuxFocus Editor team © Carlos Andrés Pérez "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Translation information: es −−> −− : Carlos Andrés Pérez <caperez/at/usc.edu.co> en −−> es: Carlos Andrés Pérez <caperez/at/usc.edu.co>

2005−04−13, generated by lfparser_pdf version 2.51