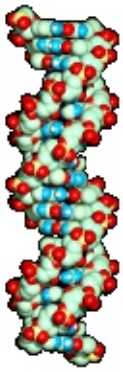
Linux & Perl, des outils pour l'étude et l'analyse d'informations biologiques

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

L´auteur:

Carlos Andrés Péres est spécialiste en Simulation Moléculaire et candidat au doctorat en Biotechnologie. Conseiller technique du Grupo de Investigación en Educación Virtual (GIEV) – Groupe de Recherche en Apprentissage Virtuel. Adresse : Universidad Santiago de Cali, Calle 5ª carrera 62 Campus Pampalinda, Cali – Colombia.

Traduit en Français par:
Jean–EtiennePoirrier
(homepage)



Résumé:

Cet article veut montrer quelques uns des avantages de la programmation Perl sous Unix pour l'extraction d'information dans les bases de données de séquences d'ADN, ARN et de protéines. Elles peuvent être utilisées dans les processus comparatifs ou l'analyse. Le projet Génome Humain et les techniques de clonage d'ADN ont accéléré le progrès scientifique dans ce domaine. Les informations générées quotidiennement dépassent souvent la capacité de traitement de cette information d'un point de vue évolutif.

La prolifération rapide de l'information biologique sur différents génomes (l'ensemble des gènes d'un organisme) conduit la bioinformatique à être une discipline fondamentale pour la manipulation et l'analyse de ces données.

Bioinformatique

La bioinformatique est née lorsque les scientifiques ont commencé à stocker les séquences biologiques dans un format numérique et les premiers programmes pour les comparer ont suivi. Pendant un bon moment, la bioinformatique a été limitée à l'analyse de séquences. Néanmoins, l'importance de l'établissement de la forme structurelle des molécules a transformé les ordinateurs en un outil important pour la recherche en biochimie théorique. Chaque jour, il y a plus d'informations et plus de collections de données sur la conformation tridimensionnelle des molécules. Les gènes ont changé : étudiés de manière individuelle au début, ils sont maintenant étudiés ensemble (ou une partie non négligeables d'entre eux). Il est maintenant plus facile de comprendre comment ils se comportent entre eux, avec les protéines et comment ils s'organisent dans les voies métaboliques. À chaque fois, nous devenons plus conscients de l'importance d'organiser les données.

Chacune des activités décrites a au−moins deux faces pour lesquelles ils sont intéressants. D'un côté, l'intérêt biologique est de connaître les relations entre les molécules de la vie ; et de l'autre côté, l'assemblage devient un problème de conception de logiciel intéressant à résoudre. La nécessité est de combiner et d'intégrer l'information biologique pour obtenir une vision globale et effective des processus biologiques sous−jacents. Nous avons également remarqué la nécessité de combiner différentes branches de l'informatique pour arriver à une solution effective. Par exemple, la gestion de bases de données, l'intégration des données, des algorithmes efficaces, du matériel puissant − grilles, multi−processeurs, etc.

Perl

Larry Wall a commencé le développement de Perl en 1986. Perl est un langage de programmation interprêté, idéal pour manipuler des textes, fichiers et processus. Perl permet le développement rapide de petits programmes. On peut dire que Perl est un mélange optimisé de langage de haut niveau (par exemple, C) et de langage de script (par exemple, Bash).

Les programmes Perl peuvent tourner sous différents systèmes d'exploitation/plates−formes. Cependant, Perl est né et s'est répandu sur les systèmes d'exploitation UNIX. Perl a complètement dépassé sa portée initiale, grâce à l'impulsion qu'il a reçu de son utilisation immédiate comme langage pour applications web. Avant que Perl ne soit utilisé, ***awk***, ***thirst*** et ***grep*** étaient les outils d'analyse de fichier et d'extraction de l'information.

Perl a réuni les possibilités de ces outils UNIX dans un seul programme étendant et modernisant chacun d'entre eux avec plus de fonctionnalités.

Perl est un langage de programmation libre et il est peut être exécuté sur tout système d'exploitation qui est généralement présent dans les laboratoires de recherche biologique. Sous UNIX et MacOSX, il est déjà précompilé, sur d'autres systèmes, il est nécessaire d'installer Perl. Il suffit de l'obtenir du site : http://www.cpan.org pour le système que nous utilisons.

Les programmes en Perl sous Linux sont appelé par le nom du fichier qui contient les instructions pour l'exécuter. Les instructions sont placées dans un fichier et Perl est évoqué avec le nom de ce fichier comme argument.

Une autre méthode fréquente est de placer les instructions Perl dans un fichier mais sans invoquer Perl avec le fichier en argument. Pour cela, nous devons réaliser 2 choses : (a) mettre un commentaire spécial à la première ligne du programme :

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

print "Hi\n";
```

et (b) stocker le fichier et lui assigner les propriétés UNIX d'exécution :

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

Une fois cela fait, le fichier/programme peut être utilisé simplement en l'appelant par le nom de fichier

La gestion de fichiers en Perl :

Lorsque nous avons une base de données de séquences moléculaires au format texte, nous pouvons faire de Perl un outil de recherche de séquence. Dans cet exemple, nous voyons comment rechercher une séquence protéique dans une base de données au format SWISS−PROT (db_human_swissprot), en utilisant son code id.

```
#!/usr/bin/perl
# Recherche de séquence d'acide aminé dans une base de données
# au format SWISS-PROT avec un code id donnée
# Demander le code dans le champ ID
# et il l'assigne de l'entrée standard (STDIN) à une variable
print "Entrez l"ID à rechercher : ";
$id_query=<STDIN>;
chomp $id_query;
# Nous ouvrons le fichier de base de données
# mais si ce n'est pas possible, le programme se termine
open (db, "human_kinases_swissprot.txt") ||
 die "problème à l'ouverture du fichier human_kinases_swissprot.txt\n";
# Regarder chaque ligne dans la base de données
while (<db>) {
chomp $_;
# Vérifier si nous sommes dans le champ ID
if ($_ =~ /^ID/) {
# Si c'est possible, nous récoltons l'information
# en coupant la ligne aux espaces
($a1,$id_db) = split (/\s+/,$_);
# mais s'il n'y a pas de concordance d'ID, nous continuons ...
next if ($id_db ne $id_query);
# Lorsqu'ils concordent, nous plaçons un marqueur
$signal_good=1;
# Ensuite, nous vérifions le champ de séquence
# et si le marqueur est 1 (séquence choisie),
# Si c'est positif, nous changeons le marqueur à 2, pour collecter la
# séquence
} elsif (($_ =~ /^SQ/) && ($signal_good==1)) {
$signal_good=2;
# Finalement, si le marqueur est 2, nous présentons chaque ligne
# de la séquence, jusqu'à ce que la ligne commence par //
# si c'est le cas, nous cassons le while
} elsif ($signal_good == 2) {
last if ($_ =~ /^\/\//);
print "$_\n";
}
}
# Lorsque nous quittons l'instruction while, nous vérifions le marqueur
# s'il est négatif, cela signifie que nous ne trouvons pas la séquence
# choisie, ce qui nous donnera une erreur
if (!$signal_good) {
print "ERROR: "."Sequence not found\n";
}
# Finalement, nous fermons le fichier qui est toujours ouvert
close (db);
exit;
```

Recherche de motifs d'aminoacides

```
#!/usr/bin/perl
# Recherche de motifs d'aminoacides
# Demande à l'utilisateur les motifs à chercher
print "SVP introduisez le motif à chercher dans query.seq : ";
$patron = <STDIN>;
chomp $patron;
# Ouvre le fichier de base de données
# mais s'il ne peut pas, le programme se termine
open (query, "query_seq.txt")
 || die "problème à l'ouverture du fichier query_seq.txt\n";
# Recherche la séquence SWISS-PROT ligne par ligne
while (<query>) {
chomp $_;
# Lorsqu'arrive au champ SQ, placer le marqueur à 1
   if ($_ =~ /^SQ/) {
         $signal_seq = 1;
# Lorsqu'arrive la fin de la séquence, laisser la courbure
# Vérifier que cette expression est placée avant la vérification
# le marqueur = 1, parce que cette ligne n'appartient pas à la séquence
# d'acides aminés
   } elsif ($_ =~ /^\/\//) {
         last;
# Vérifie si le marqueur est égale à 1 ; s'il est positif,
# élimine les espaces blancs dans la ligne de la séquence
# et joint chaque ligne dans une nouvelle variable
# Pour la concaténation, nous pouvons aussi écrire :
# $secuencia_total.=$_;
   } elsif ($signal_seq == 1) {
         $_ =~ s/ //g;
         $secuencia_total=$secuencia_total.$_;
   }
  }
# Maintenant, vérifier la séquence collectée dans son entièreté
# pour le motif donné
  if ($secuencia_total =~ /$patron/) {
   print "La séquence query.seq contient le motif $patron\n";
  } else {
   print "La séquence query.seq ne contient pas le motif $patron\n";
  }
# Finalement, nous fermons le fichier
# et quittons le programme
close (query);
exit;
```

Si nous voulons connaître la position exacte où il a trouvé le motif, nous devons utiliser une variable spéciale `$&'. Cette variable garde le motif trouvé après avoir évalué une expression régulière (il faudra la mettre juste après la ligne ***if ($$secuencia_total>= ~/$$patron>/) {*** `. En plus, il est possible de combiner les variables ` $` ' et ` $ ´ ' qui stockent tout à gauche et à droite du motif trouvé. Modifions le programme précédent avec ces nouvelles variables pour donner la position exacte du motif. Note : vous pouvez aussi trouver intéressante la fonction ***length*** qui donne la longueur de la chaîne.

```
# Nous devons seulement changer le if où le motif a été trouvé
# Maintenant, vérifier la séquence collectée dans son entièreté,
# pour le motif donné
# et vérifier sa position dans la séquence
  if ($secuencia_total =~ /$patron/) {
   $posicion=length($`)+1;
```

```
 print "La séquence query_seq.txt contient le motif $patron à la position suivante $posicion\n"
} else {
print "La séquence query_seq.txt ne contient pas le motif $patron\n";
}
```

Calcul des fréquences d'acides aminés :

La fréquence des différents acides aminés dans les protéines est variable. Cela résulte de ses différentes fonctions ou environnements favoris. Donc, dans cet exemple, nous allons voir comment calculer la fréquence en acides aminés d'une séquence donnée d'acide aminé.

```
#!/usr/bin/perl
# Calcule la fréquence d'un acide aminé dans une séquence protéique
# Récupère le nom de fichier de la ligne de commande
# (formaté selon SWISS-PROT)
# Peut aussi être demandé avec print de <STDIN>
if (!$ARGV[0]) {print "La commande devrait être : program.pl fichier_swissprot\n";}
$fichero = $ARGV[0];
# Initialise la variable $errores
my $errores=0;
# Ouvre le fichier en lecture
open (FICHA, "$fichero") || die "problème lors de l'ouverture du fichier $fichero\n";
# D'abord, nous vérifions la séquence comme nous l'avons fait dans l'exemple 2
while (<FICHA>) {
chomp $_;
if ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
   last if ($_ =~ /^\/\//);
   $_ =~ s/\s//g;
   $secuencia.=$_;
}
}
close (FICHA);
# Maintenant, utilisons une courbure qui vérifie chaque position de l'acide
# aminé dans la séquence (à partir d'une fonction propre qui peut être
# utilisée par après par d'autres programmes)
comprueba_aa ($secuencia);
# Affiche le résultat à l'écran
# D'abord les 20 acides aminés et ensuite les chaines avec leurs fréquences
# Dans ce cas, 'sort' ne peut pas être utilisée dans le foreach,
# parce que la chaine contient des fréquences (nombres)
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";
}
# Ensuite, on donne les erreurs possibles
# et termine le programme
print "\nerreurs = $errores\n";
exit;
# Fonctions
# Cette fonction calcule la fréquence de chaque acide aminé
# d'une séquence protéique
sub comprueba_aa {
# Acquérir la séquence
my ($secuencia)=@_;
# et tourner, acide aminé par acide aminé, en utilisant un for allant
# de 0 jusqu'à la longueur de la séquence
```

```
for ($posicion=0 ; $posicion<length $secuencia ; $posicion++ ) {
# Acquérir les acides aminés
$aa = substr($secuencia, $posicion, 1);
# et vérifier lequel utilise if
# and checks which one is using if
# lorsqu'il est coché, il ajoute 1 à la fréquence correspondante
# dans une chaîne utilisant un pointeur pour chacun d'eux
# classés par ordre alphabétique
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 l'acide aminé n'est pas trouvé,
# il ajoute 1 aux erreurs
} else {
print "ERREUR: Acide aminé non trouvé : $aa\n";
$errores++;
}
}
# Finalement, retourne la chaîne de fréquences
return @aa;
}
```

Maintenant, nous allons effectuer les étapes suivantes qui suivent le flux d'information dans une cellule après la transcription. Une de ces étapes est la traduction, par laquelle une séquence d'ARN provenant du gène (qui y était sous forme d'ADN) se transforme en protéines ou séquences d'acides aminés. Pour cela, nous devons utiliser le code génétique qui est basé sur des triplets d'ARN/ADN correspondants à un acide aminé. La

séquence que nous allons extraire est une portion d'un gène d'*Escherichia coli* au format EMBL et nous allons bientôt vérifier la traduction avec l'acide aminé réel. Pour cet exemple, il sera nécessaire d'introduire les variables associatives de chaînes ou tables de hachage (*tables hash*). Dans le programme, nous devrions considérer que seule la région codificarte est nécessaire, région incluse dans le champ 'FT CDS field.

```
#!/usr/bin/perl
# Traduit une séquence d'ADN à partir d'une fiche EMBL
# vers l'acide aminé correspondant
# Il prend le nom du fichier dans la ligne de commande
# (fichier au format SWISS-PROT)
# Il peut également être demandé avec print à partir de <STDIN>
if (!$ARGV[0])
   {print "La ligne de commande devrait être : program.pl fiche_embl\n";}
$fichero = $ARGV[0];
# Ouvre le fichier en lecture
open (FICHA, "$fichero") ||
   die "problème à l'ouverture du fichier $fichero\n";
# D'abord, nous vérifions la séquence comme nous l'avons fait dans
# l'exemple 2
while (<FICHA>) {
chomp $_;
if ($_ =~ /^FT CDS/) {
$_ =~ tr/../ /;
($a1,$a2,$a3,$a4) = split (" ",$_);
}
elsif ($_ =~ /^SQ/) {
$signal_good = 1;
} elsif ($signal_good == 1) {
last if ($_ =~ /^\/\//);
# Elimine les nombres et espaces
$_ =~ tr/0-9/ /;
$_ =~ s/\s//g;
$secuencia.=$_;
}
}
close (FICHA);
# Maintenant, nous définissons un tableau associatif avec la
# correspondance entre chaque acide aminé et leurs nucléotides
# correspondant (c'est aussi dans une fonction propre, au cas où le même
# code génétique serait utilisé dans d'autres programmes
my(%codigo_genetico) = (
'TCA' => 'S',# Sérine
'TCC' => 'S',# Sérine
'TCG' => 'S',# Sérine
'TCT' => 'S',# Sérine
'TTC' => 'F',# Phénilalanine
'TTT' => 'F',# Phénilalanine
'TTA' => 'L',# Leucine
'TTG' => 'L',# Leucine
'TAC' => 'Y',# Tirosine
'TAT' => 'Y',# Tirosine
'TAA' => '*',# Stop
'TAG' => '*',# Stop
'TGC' => 'C',# Cystéine
'TGT' => 'C',# Cystéine
'TGA' => '*',# Stop
'TGG' => 'W',# Tryptophane
'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',# Methionine
'ACA' => 'T',# Tréonine
'ACC' => 'T',# Tréonine
'ACG' => 'T',# Tréonine
'ACT' => 'T',# Tréonine
'AAC' => 'N',# Asparagine
'AAT' => 'N',# Asparagine
'AAA' => 'K',# Lisine
'AAG' => 'K',# Lisine
'AGC' => 'S',# Sérine
'AGT' => 'S',# Sérine
'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',# Acide Aspartique
'GAT' => 'D',# Acide Aspartique
'GAA' => 'E',# Acide Glutamique
'GAG' => 'E',# Acide Glutamique
'GGA' => 'G',# Glycine
'GGC' => 'G',# Glycine
'GGG' => 'G',# Glycine
'GGT' => 'G',# Glycine
);
# Traduire chaque codon en son acide aminé correspondant
# et l'aggrège dans la séquence protéique
print $a3;
for($i=$a3 - 1; $i < $a4 - 3 ; $i += 3) {
$codon = substr($secuencia,$i,3);
# Transforme le codon de minuscule (format EMBL) en majuscule
$codon =~ tr/a-z/A-Z/;
$protein.= codon2aa($codon);
}
print "Cette séquence protéique du gène : \n$secuencia\n est la suivante :\n$protein\n\n";
exit;
```

Références bibliographiques

- 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/
- ***Fichiers d'exemples :***
  - human_kinases_swissprot.txt
  - query_seq.txt
  - ecoli_embl.txt
