Introduction à *BioPerl*

Jean-Michel Richer
jean-michel.richer@univ-angers.fr
http://www.info.univ-angers.fr/pub/richer

Université d'Angers
LERIA - Laboratoire d'Etudes et de Recherche
en Informatique d'Angers

2008

Plan

Qu'est ce que BioPerl ?

BioPerl

BioPerl est un ensemble de modules (packages) Perl dédiés à la bioinformatique qui permettent notamment de :

- lire, écrire, traduire, manipuler des séquences (fasta, msf, pileup, phylip, ...)
- accèder à des bases de données (GenBank, GenPept, SwissProt)
- rechercher des séquences (Blast distant au NCBI)
- recherche des gènes
- manipuler des alignements
- lire des structures 3D des fichiers PDB

Obtenir et Installer BioPerl

BioPerl est disponible à l'adresse suivante :

http ://www.bioperl.org/

Un certain nombre de *packages* peuvent être téléchargés :

- **Core** : les modules de base de BioPerl
- **Run** : exécution de programmes externes (Clustalw, Phylip, ...)
- **BioPerl db** : base de données BioSQL
- **Ext** : extensions pour le langage C
- **Microarray** : manipulation de microarray

Programmation Orientée Objet

BioPerl et les objets

Le code de *BioPerl* repose sur l'utilisation des objets.

Programmation Orientée Objet

La POO repose sur un paradigme qui consiste à :

- encapsuler les données dans une structure appelée classe
- associer des méthodes (sous-programmes) de traitement des données à la classe

- Programmation classique :

```
sous-programme(donnée, paramètres)
```

- Programmation objet :

```
donnée.sous-programme(paramètres)
```

POO en Perl

La mise en place de la POO en Perl n'est pas naturelle comparativement à d'autres langages comme C++ ou Java.

Exemple

On désire définir une classe **Employee** afin de représenter un employé par son nom et son salaire. On veut également pouvoir augmenter le salaire de l'employé de 2%.

On écrit un module Perl pour la classe Employee que l'on nomme `Employee.pm`

Employee.pm

```perl
package Employee;

# constructeur
sub new {
   my ($class_name,$a_name,$a_salary) = @_;
   my $this = {
      name   => $a_name,
      salary => $a_salary
   };
   bless $this, $class_name;
   return $this;
}
```

- les données sont stockées dans un mapping
- `bless` attribue le type Employee au nouvel objet

Employee.pm

```perl
# augmentation de salaire
sub increase_salary {
   my ($self, $percent) = @_;
   $p = (100.0 + $percent)/100.0;
   $self->{salary} = $self->{salary} * $p;
}

# récupération des données sous forme d'une chaîne
sub get_info {
   my ($self) = @_;
   return $self->{name} . " " . $self->{salary};
}

1;
```

Utilisation du package Employee.pm

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

# utilisation du package Employee
use Employee;

# création d'un employé
$dupond = Employee->new("Jean Dupond", 2000.0);

print $dupond->get_info(), "\n";

# augmentation de 2 %
$dupond->increase_salary(2.0);
print $dupond->get_info(), "\n";
```

Remarque : on utilise $\rightarrow$ pour faire référence à une méthode de la classe.

Utilisation du package Employee.pm

Le résultat de l'exécution du programme est :

```
Jean Dupond 2000
Jean Dupond 2040
```

Les Classes de BioPerl

Définition d'une séquence

La classe la plus utilisée est **Bio**::**Seq** qui permet de représenter une séquence de nucléotides ou d'acides aminés.

```
use Bio::Seq;

$sequence = Bio::Seq->new(
 -seq => "ACTGTGTGTCCC",
 -id  => "Chlorella sorokiniana",
 -accession_number => "CAA41635"
);

print $sequence->display_id(), "\n";
print $sequence->seq(), "\n";
print $sequence->length(), "\n";
print $sequence->alphabet(), "\n";
$sous_sequence = $sequence->subseq(2,6);
print "$sous_sequence\n";
```

Définition d'une séquence

Le résultat de l'affichage est :

```
Chlorella sorokiniana
ACTGTGTGTCCC
12
dna
CTGTG
```

Traduction

On peut traduire les séquences de nucléotides en acides aminés grâce à la méthode **translate** :

```
$protein = $sequence->translate(
 -unknown => 'X',
 -frame   => 0
);

print $protein->display_id(), "\n";
print $protein->seq(), "\n";
```

Le résultat est :

```
Chlorella sorokiniana
TVCP
```

Autres types de séquences

Il existe d'autres types de séquences :

- **PrimarySeq** : version simplifiée de Seq
- **LocatableSeq**
- **RelSegment**
- **LiveSeq**
- **LargeSeq** : séquences longues ($>$100 Mo)
- **RichSeq**

Lecture et écriture des séquences

Le package **Bio**::**SeqIO** offre la possibilité de lire (ou écrire) une séquence depuis (ou vers) un fichier. Plusieurs formats sont pris en compte :

Format	Suffixes
clustalw	aln
emboss	water, needle
fasta	fasta fast fa seq nt aa
mega	meg ega
gcg	msf pileup gcg
nexus	nexus nex
phylip	phylip phlp phy ph
...	...

Changement de format

```
use Bio::Seq;
use Bio::SeqIO;

$input = Bio::SeqIO->new(
 -file   => "glutamate.fasta",
 -format => "fasta"
);

$output = Bio::SeqIO->new(
 -file   => ">glutamate.gcg",
 -format => "gcg"
);

while ($seq = $input->next_seq()) {
  $output->write_seq($seq);
}
```

Accès aux bases de données

BioPerl fournit des packages pour accèder aux bases de données suivantes :

- **GenBank** : Bio::DB::GenBank
- **GenPept** : Bio::DB::GenPept
- **SwissProt** : Bio::DB::SwissProt
- **RefSeq** : Bio::DB::RefSeq
- **EMBL** : Bio::DB::EMBL

Exemple d'accès à GenBank

```perl
use Bio::SeqIO;
use Bio::DB::GenBank;


$genbank = new Bio::DB::GenBank;
$sequence = $genbank->get_Seq_by_acc("CAA41635");

print $sequence->display_id(), "\n";
print $sequence->seq(), "\n";
print $sequence->desc(), "\n";
```

On peut également utiliser :

- `get_Seq_by_id`
- `get_Seq_by_gi`
- `get_Stream_by_query`

Autre exemple : requête GenBank

```
use Bio::DB::GenBank;
use Bio::DB::Query::GenBank;

$genbank=new Bio::DB::GenBank;

$query=Bio::DB::Query::GenBank->new(
  -query =>'glutamate dehydrogenase',
  -db    => 'protein');

$seqio=$genbank->get_Stream_by_query($query);

while($seq=$seqio->next_seq) {
  print $seq->display_id(),"\n";
  print $seq->seq(),"\n";
}
```

On peut redéfinir la variable d'environnement suivante si on utilise un proxy :

```
$ENV{"HTTP_PROXY"}="mon-proxy :xxxx";
```

Programmes externes

Au travers du package **Bio**::**Tools**::**Run**, BioPerl permet d'exécuter des programmes tels que Clustalw, Phylip (ProtDist, Neighbor, ...)

- **RemoteBlast** (Blast)
- **Alignment : :Clustalw** (Clustalw)
- **Alignment : :TCoffee** (TCoffee)
- **Phylo : :Phylip : :ProtDist** (ProtDist)
- **Phylo : :Phylip : :Neighbor** (Neighbor)
- ...

Exemple alignement avec Clustalw

```
use Bio::AlignIO;
use Bio::Tools::Run::Alignment::Clustalw;

@params = (
 'ktuple' => 3,
 'matrix' => 'BLOSUM',
 'outfile' => 'align.msf',
 'output' => 'gcg');

$clustalw = Bio::Tools::Run::Alignment::Clustalw->new(@params);

$alignment = $clustalw->align("glutamate.fasta");
```

Remarque : `$alignment` est une instance de **Bio**::**SimpleAlign**

Manipuler un alignement

```perl
use Bio::SimpleAlign;

# consensus with 50 % threshold
$consensus = $alignment->consensus_string(50);
print "$consensus\n";

if ($alignment->is_flush()) {
   print "length = ", $alignment->length(), "\n";
}

foreach $seq ($alignment->each_seq) {
   print $seq->display_id(), " ", $seq->seq(), "\n";
}
```

Remarque : la fonction `is_flush` vérifie que les séquences ont toutes la même longueur.

Lecture d'un alignement

On peut lire (ou écrire) un alignement depuis (ou vers) un fichier grâce au package **Bio**::**AlignIO** :

```
use Bio::AlignIO;

$file = Bio::AlignIO->new(
 -file   => 'align.msf',
 -format => 'msf'
);

$alignment = $file->next_aln;

$consensus = $alignment->consensus_string(50);
print "$consensus\n";
```

Conclusion

BioPerl se révèle être un outil très intéressant :

- pour le non informaticien, il permet rapidement d'automatiser une suite de traitements,
- BioPerl est sous licence GPL (GNU Public Licence) et peut être étendu ou modifié si le besoin s'en fait sentir,

Néanmoins, il se révèle difficile à maîtriser pour deux raisons :

- l'utilisation des objets, peu pratique en Perl
- une documentation très succinte de BioPerl