http://search.cpan.org/~cjfields/BioPerl-1.6.901/BioPerl.pm

What is bioperl

Bioperl is a collection of perl modules that facilitate the development of perl scripts for bioinformatics applications. The intent of the BioPerl development effort is to make reusable tools that aid people in creating their own sites or job-specific applications.

Bioperl is NOT

As such, it does not include ready to use programs in the sense that many free/commercial softwares do such as BLAST, HMMER, FASTA, MAFFT, MEGA etc.

What Bioperl can do

Bioperl provides reusable perl modules that facilitate writing perl scripts for sequence manipulation, accessing of databases using a range of data formats and execution and parsing of the results of various molecular biology programs including Blast, clustalw, TCoffee, genscan, ESTscan and HMMER etc.
E.g. run other tools, parse results, convert formats, retrieve sequences etc.

As the modules (through objects) do most of the hard work for you, all you have to do is to combine a number of objects together sensibly to make useful scripts.

BioPerl is the product of a community effort to produce Perl code useful in biology. Examples include Sequence **objects** (created from modules), Alignment objects and database searching objects.

SeqIO, SearchIO, AlignIO, TreeIO, Tools, DB, Seq

These objects **also interact** - Alignment objects are made from the Sequence objects, Sequence objects have access to Annotation and SeqFeature objects and databases, Blast objects can be converted to Alignment objects, and so on. This means that the objects provide a coordinated and extensible framework to do computational biology.

What is exactly an object?
An object is a specific instance of a module or subroutine

Jamison D. Perl Programming for Biologists (Wiley,2003) (ISBN 0471430595)

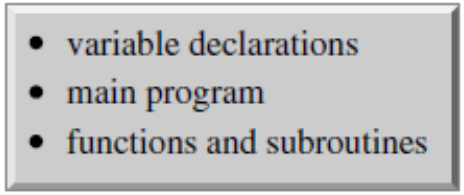

Figure 10.1 Procedural code layout

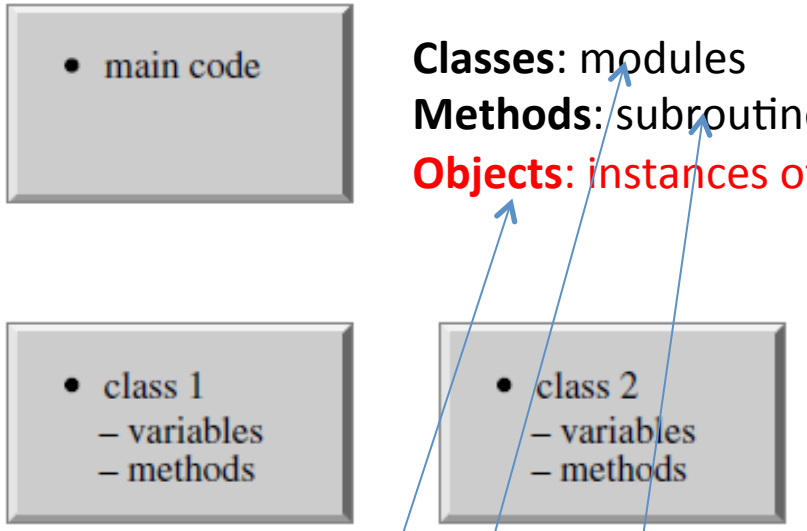

Figure 10.2 Object-oriented code layout

Classes: modules
Methods: subroutines
Objects: instances of classes

Bioperl modules are called in the main perl scripts in a fashion of Object-Oriented paradigm, which is in contrast to the procedural paradigm. Procedural code is typically used for short programs while OOP is often used for complex medium and long programs.

The **arrow operator ->** is widely used to call a subroutine of a module to build an object.

```
#!/usr/bin/perl -w

use Bio::SeqIO;

$new=Bio::SeqIO->new(-file=>$ARGV[0],
-format=>"fasta");

while($seq=$new->next_seq){
  print $seq->id,"\t", length $seq-
>seq, "\n";
}
```

Find out where bioperl modules are installed to on glu:

```
locate bioperl | less

locate Bio | less
```

Manual: /usr/share/man/man3
Code: /usr/share/perl5/Bio/

Bio::SeqIO

```
yyin@glu:~/work/class$ ls /usr/share/perl5/Bio/
Align                       Coordinate                  Location                    PrimarySeq.pm               SeqUtils.pm
AlignIO                     Das                         LocationI.pm                PullParserI.pm              SimpleAlign.pm
AlignIO.pm                  DasI.pm                     Map                         RangeI.pm                   SimpleAnalysisI.pm
AnalysisI.pm                DB                          MapIO                       Range.pm                    Species.pm
AnalysisParserI.pm          DBLinkContainerI.pm         MapIO.pm                    Restriction                 Structure
AnalysisResultI.pm          DescribableI.pm             Matrix                      Root                        Symbol
AnnotatableI.pm             Draw                        MolEvol                     Search                      Taxonomy
Annotation                  Event                       Nexml                       SearchDist.pm               Taxonomy.pm
AnnotationCollectionI.pm    Factory                     NexmlIO.pm                  SearchIO                    Taxon.pm
AnnotationI.pm              FeatureHolderI.pm           Ontology                    SearchIO.pm                 Tools
ASN1                        FeatureIO                   OntologyIO                  Seq                         Tree
Assembly                    FeatureIO.pm                OntologyIO.pm               SeqAnalysisParserI.pm       TreeIO
Biblio                      HandlerBaseI.pm             ParameterBaseI.pm           SeqEvolution                TreeIO.pm
Biblio.pm                   IdCollectionI.pm            Perl.pm                     SeqFeature                  UpdateableSeqI.pm
Cluster                     IdentifiableI.pm            Phenotype                   SeqFeatureI.pm              Variation
ClusterIO                   Index                       PhyloNetwork                SeqIO                       WebAgent.pm
ClusterIO.pm                Installer                   PhyloNetwork.pm             SeqIO.pm
ClusterI.pm                 LiveSeq                     PopGen                      SeqI.pm
CodonUsage                  LocatableSeq.pm             PrimarySeqI.pm              Seq.pm
```

Bio::SeqIO

The SeqIO folder contains modules for all different sequence formats

The main module for sequence IO, interconnecting with other more basic modules e.g. to Bio::Seq

```
yyin@glu:~/work/class$ ls /usr/share/perl5/Bio/SeqIO
SeqIO/    SeqIO.pm
yyin@glu:~/work/class$ ls /usr/share/perl5/Bio/SeqIO
abi.pm          chadoxml.pm     excel.pm            game.pm       kegg.pm       MultiFile.pm  scf.pm      tigr.pm
ace.pm          chaos.pm        exp.pm              gbdriver.pm   largefasta.pm nexml.pm      seqxml.pm   tigrxml.pm
agave.pm        chaosxml.pm     fasta.pm            gbxml.pm      lasergene.pm  phd.pm        strider.pm  tinyseq
alf.pm          ctf.pm          fastq.pm            gcg.pm        locuslink.pm  pir.pm        swissdriver.pm tinyseq.pm
asciitree.pm    embldriver.pm   flybase_chadoxml.pm genbank.pm    mbsout.pm     pln.pm        swiss.pm    ztr.pm
bsml.pm         embl.pm         FTHelper.pm         Handler       metafasta.pm  qual.pm       table.pm
bsml_sax.pm     entrezgene.pm   game                interpro.pm   msout.pm      raw.pm        tab.pm
yyin@glu:~/work/class$
```

Different sequence formats can inter-converted, e.g. GenBank to fasta, or fastq to fasta and qual

Bio::AlignIO

```
yyin@glu:~/work/class$ ls /usr/share/perl5/Bio/AlignIO
arp.pm      clustalw.pm  fasta.pm  largemultifasta.pm  mase.pm  meme.pm
msf.pm      nexus.pm     phylip.pm proda.pm            psi.pm   stockholm.pm
bl2seq.pm   emboss.pm    Handler   maf.pm              mega.pm  metafasta.pm
nexml.pm    pfam.pm      po.pm     prodom.pm           selex.pm xmfa.pm
```

```
NAME
    Bio::SeqIO - Handler for SeqIO Formats

SYNOPSIS
    use Bio::SeqIO;
    $in  = Bio::SeqIO->new(-file => "inputfilename" ,
                           -format => 'Fasta');
    $out = Bio::SeqIO->new(-file => ">outputfilename" ,
                           -format => 'EMBL');

    while ( my $seq = $in->next_seq() ) {
            $out->write_seq($seq);
    }

    # Now, to actually get at the sequence object, use the standard Bio::Seq
    # methods (look at Bio::Seq if you don't know what they are)

    use Bio::SeqIO;

    $in  = Bio::SeqIO->new(-file => "inputfilename" ,
                           -format => 'genbank');

    while ( my $seq = $in->next_seq() ) {
        print "Sequence ",$seq->id, " first 10 bases ",
            $seq->subseq(1,10), "\n";
    }

    # The SeqIO system does have a filehandle binding. Most people find this
    # a little confusing, but it does mean you can write the world's
    # smallest reformatter

    use Bio::SeqIO;

    $in  = Bio::SeqIO->newFh(-file => "inputfilename" ,
                             -format => 'Fasta');
    $out = Bio::SeqIO->newFh(-format => 'EMBL');
```

Format conversion

Print seq

Check the documentation of SeqIO at bioperl website

http://www.bioperl.org/wiki/HOWTO:SeqIO

```
perldoc Bio::SeqIO
perldoc /usr/share/perl5/Bio/SeqIO.pm
less /usr/share/perl5/Bio/SeqIO.pm
```

Space to page down

u to page up

q to exit

```
    # World's shortest Fasta<->EMBL format converter:
    print $out $_ while <$in>;
```

DESCRIPTION

Bio::SeqIO is a handler module for the formats in the SeqIO set (eg, Bio::SeqIO::fasta). It is the officially sanctioned way of getting at the format objects, which most people should use.

The Bio::SeqIO system can be thought of like biological file handles. They are attached to filehandles with smart formatting rules (eg, genbank format, or EMBL format, or binary trace file format) and can either read or write sequence objects (Bio::Seq objects, or more correctly, Bio::SeqI implementing objects, of which Bio::Seq is one such object). If you want to know what to do with a Bio::Seq object, read Bio::Seq.

The idea is that you request a stream object for a particular format. All the stream objects have a notion of an internal file that is read from or written to. A particular SeqIO object instance is configured for either input or output. A specific example of a stream object is the Bio::SeqIO::fasta object.

Each stream object has functions

```
$stream->next_seq();
```

and

```
$stream->write_seq($seq);
```

As an added bonus, you can recover a filehandle that is tied to the SeqIO object, allowing you to use the standard <> and print operations to read and write sequence objects:

```
use Bio::SeqIO;

$stream = Bio::SeqIO->newFh(-format => 'Fasta',
                            -fh     => \*ARGV);
# read from standard input or the input filenames
while ( $seq = <$stream> ) {
    # do something with $seq
}
```

As an added bonus, you can recover a filehandle that is tied to the SeqIO object, allowing you to use the standard <> and print operations to read and write sequence objects:

```
use Bio::SeqIO;

$stream = Bio::SeqIO->newFh(-format => 'Fasta',
                            -fh     => \*ARGV);
# read from standard input or the input filenames

while ( $seq = <$stream> ) {
      # do something with $seq
}
```

and

```
print $stream $seq; # when stream is in output mode
```

This makes the simplest ever reformatter

```
#!/usr/bin/perl
use strict;
my $format1 = shift;
my $format2 = shift || die
    "Usage: reformat format1 format2 < input > output";

use Bio::SeqIO;

my $in  = Bio::SeqIO->newFh(-format => $format1, -fh => \*ARGV );
my $out = Bio::SeqIO->newFh(-format => $format2 );
# Note: you might want to quote -format to keep older
# perl's from complaining.

print $out $_ while <$in>;
```

CONSTRUCTORS

Bio::SeqIO->new()

```
    $seqIO = Bio::SeqIO->new(-file => 'filename',   -format=>$format);
    $seqIO = Bio::SeqIO->new(-fh   => \*FILEHANDLE, -format=>$format);
    $seqIO = Bio::SeqIO->new(-format => $format);
```

The new() class method constructs a new Bio::SeqIO object. The returned object can be used to retrieve or print Seq objects. new() accepts the following parameters:

-file

A file path to be opened for reading or writing. The usual Perl conventions apply:

```
    'file'      # open file for reading
    '>file'     # open file for writing
    '>>file'    # open file for appending
    '+<file'    # open file read/write
    'command |' # open a pipe from the command
    '| command' # open a pipe to the command
```

-fh You may provide new() with a previously-opened filehandle. For example, to read from STDIN:

```
    $seqIO = Bio::SeqIO->new(-fh => \*STDIN);
```

Note that you must pass filehandles as references to globs.

If neither a filehandle nor a filename is specified, then the module will read from the @ARGV array or STDIN, using the familiar <> semantics.

A string filehandle is handy if you want to modify the output in the memory, before printing it out. The following program reads in EMBL formatted entries from a file and prints them out in fasta format with some HTML tags:

```
    use Bio::SeqIO;
    use IO::String;
    my $in  = Bio::SeqIO->new(-file => "emblfile",
                              -format => 'EMBL');
    while ( my $seq = $in->next_seq() ) {
        # the output handle is reset for every file
        my $stringio = IO::String->new($string);
        my $out = Bio::SeqIO->new(-fh => $stringio,
                                  -format => 'fasta');
```

To see the actual code of this subroutine

```
less /usr/share/perl5/Bio/SeqIO.pm
```

new The "new" function

```
    Title   : new
    Usage   : $stream = Bio::SeqIO->new(-file => $filename,
                                        -format => 'Format')
    Function: Returns a new sequence stream
    Returns : A Bio::SeqIO stream initialised with the appropriate format
    Args    : Named parameters:
              -file => $filename
              -fh => filehandle to attach to
              -format => format

              Additional arguments may be used to set factories and
              builders involved in the sequence object creation. None of
              these must be provided, they all have reasonable defaults.
              -seqfactory   the Bio::Factory::SequenceFactoryI object
              -locfactory   the Bio::Factory::LocationFactoryI object
              -objbuilder   the Bio::Factory::ObjectBuilderI object

   See Bio::SeqIO::Handler
```

```
next_seq
    Title   : next_seq
    Usage   : $seq = stream->next_seq
    Function: Reads the next sequence object from the stream and returns it.

              Certain driver modules may encounter entries in the stream
              that are either misformatted or that use syntax not yet
              understood by the driver. If such an incident is
              recoverable, e.g., by dismissing a feature of a feature
              table or some other non-mandatory part of an entry, the
              driver will issue a warning. In the case of a
              non-recoverable situation an exception will be thrown.  Do
              not assume that you can resume parsing the same stream
              after catching the exception. Note that you can always turn
              recoverable errors into exceptions by calling
              $stream->verbose(2).

    Returns : a Bio::Seq sequence object, or nothing if no more sequences
              are available

    Args    : none

   See Bio::Root::RootI, Bio::Factory::SeqStreamI, Bio::Seq
```

perldoc Bio::SeqIO

```
vi get-length.pl
perl get-length.pl
metagenemark_predictions.faa | less
```

```
#!/usr/bin/perl -w

use Bio::SeqIO;

$new=Bio::SeqIO->new(-file=>$ARGV[0],
-format=>"fasta");

while($seq=$new->next_seq){
  print $seq->id,"\t", length $seq->seq, "\n";
}
```

Step 1: Create a $new object from a fasta file to hold the reference to the fasta format sequences

Step 2: Call the next_seq method to extract one seq block per cycle and create the $seq object to hold the block

Step 3: Call the id method and the seq method; print the length of the sequences

vi get-seq-bioperl.pl

```
#!/usr/bin/perl -w

open(ID,$ARGV[1]);
while(<ID>){
  chomp $_;
  $id_hash{$_}=1;
}

use Bio::SeqIO;

$new=Bio::SeqIO->new(-file=>$ARGV[0], -format=>"fasta");

while($seq=$new->next_seq){
  if(defined $id_hash{$seq->id}){
    print ">",$seq->id,"\n",$seq->seq."\n";
  }
}
```

Has the same function as get-seq4.pl etc.

We still call the same module Bio::SeqIO in a new program

No need to load seq database into the memory

```
perl get-seq-bioperl.pl test-query.fa.cowrument.out.m9.hitid
metagenemark_predictions.faa | less
```

Step 1: load the id file into the memory as a hash

Step 2: Create a $new object from a fasta file to hold the reference to the fasta format sequences

Step 3: Call the next_seq method to extract one seq block per cycle and create the $seq object to hold the block

Step 4: Call the id method to check if the id is used in the pre-defined hash; if yes, print the sequence

```
#
# target name                                  accession  tlen query name            accession
#------------------------------------------------ ---------- ----- -------------------- ----------
```

# target name (1)	accession (2)	tlen (3)	query name (4)	accession (5)
NODE_457020_length_97146_cov_14.955994_orf_01700	-	782	GH5.hmm	-
NODE_457020_length_97146_cov_14.955994_orf_01700	-	782	GH5.hmm	-
NODE_2854003_length_94157_cov_5.769428_orf_67030	-	378	GH5.hmm	-
NODE_2314521_length_30819_cov_0.660826_orf_30190	-	715	GH5.hmm	-
NODE_2314521_length_30819_cov_0.660826_orf_30190	-	715	GH5.hmm	-
NODE_3609387_length_51250_cov_2.036859_orf_24440	-	423	GH5.hmm	-
NODE_2891766_length_19360_cov_5.591064_orf_12550	-	409	GH5.hmm	-
NODE_457020_length_97146_cov_14.955994_orf_01790	-	995	GH5.hmm	-
NODE_457020_length_97146_cov_14.955994_orf_01790	-	995	GH5.hmm	-
NODE_4002281_length_100204_cov_2.154804_orf_16350	-	624	GH5.hmm	-
NODE_421339_length_112723_cov_3.569067_orf_68070	-	413	GH5.hmm	-

qlen (6)	--- full sequence --- E-value (7)	score (8)	bias (9)	# (10)	of (11)	--- this domain --- c-Evalue (12)	i-Evalue (13)	score (14)	bias (15)	hmm coord from (16)	to (17)	ali coord from (18)	to (19)	env coord from	to	acc	description of target
275	2.9e-71	247.3	13.4	1	2	1.2e-45	8.1e-43	154.0	4.7	2	239	68	328	67	341	0.80	complement(17022..19367)
275	2.9e-71	247.3	13.4	2	2	2.3e-28	1.5e-25	97.4	0.3	7	228	409	651	403	666	0.74	complement(17022..19367)
275	2.2e-55	195.2	2.8	1	1	4.6e-58	3e-55	194.8	1.9	22	241	10	271	3	294	0.80	complement(3376..4509)
275	3.3e-55	194.6	8.6	1	2	4.7e-32	3.1e-29	109.5	1.3	4	243	41	301	38	311	0.80	complement(21709..23853)
275	3.3e-55	194.6	8.6	2	2	6.9e-26	4.5e-23	89.3	0.4	2	239	344	601	343	628	0.79	complement(21709..23853)
275	6.2e-55	193.8	3.0	1	1	1.3e-57	8.8e-55	193.3	2.1	24	244	95	357	83	379	0.80	complement(33514..34782)
275	1.4e-54	192.6	1.1	1	1	2.8e-57	1.8e-54	192.2	0.8	22	242	80	343	73	364	0.81	complement(11478..12704)
275	1.7e-54	192.3	5.4	1	2	6.3e-29	4.1e-26	99.2	0.6	2	237	41	311	40	322	0.74	34656..37640
275	1.7e-54	192.3	5.4	2	2	1.1e-27	7e-25	95.2	0.2	2	240	358	625	357	642	0.74	34656..37640

```
hmmsearch --domtblout GH5.hmm.cowrumen.dm GH5.hmm  metagenemark_predictions.faa >
GH5.hmm.cowrumen.out &

less GH5.hmm.cowrumen.dm | grep -v '^#' | awk '{print
$1,$3,$6,$7,$12,$13,$16,$17,$18,$19}' | awk '$6<1e-2&&($8-$7)/$3>.8' | sed 's/ /
\t/g' > GH5.hmm.cowrumen.dm.ps
```

Extracting domain regions is easy if using perl and bioperl

1	Hit-ID	6	i-E-values
2	Hit-length	7	HMM-start
3	HMM-length	8	HMM-end
4	Full length E-values	9	Hit-start
5	c-E-values	10	Hit-end

```
yyin@glu:~/work/class$ less GH5.hmm.cowrumen.dm.psl head
NODE_457020_length_97146_cov_14.955994_orf_01700      782    275    2.9e-71 1.2e-45 8.1e-43 2    239    68    328
NODE_457020_length_97146_cov_14.955994_orf_01700      782    275    2.9e-71 2.3e-28 1.5e-25 7    228    409   651
NODE_2314521_length_30819_cov_0.660826_orf_30190      715    275    3.3e-55 4.7e-32 3.1e-29 4    243    41    301
NODE_2314521_length_30819_cov_0.660826_orf_30190      715    275    3.3e-55 6.9e-26 4.5e-23 2    239    344   601
NODE_457020_length_97146_cov_14.955994_orf_01790      995    275    1.7e-54 6.3e-29 4.1e-26 2    237    41    311
NODE_457020_length_97146_cov_14.955994_orf_01790      995    275    1.7e-54 1.1e-27 7e-25   2    240    358   625
NODE_4002281_length_100204_cov_2.154804_orf_16350     624    275    2.2e-52 5.3e-55 3.4e-52 1    239    180   522
NODE_4159482_length_690369_cov_12.576118_orf_32020    762    275    1.6e-51 4.2e-54 2.7e-51 1    239    371   711
NODE_2008511_length_308405_cov_4.203573_orf_06380     745    275    3.7e-50 9.5e-53 6.2e-50 2    240    47    433
NODE_459624_length_630037_cov_23.694723_orf_16430     676    275    5.6e-50 1.2e-52 8e-50   2    241    283   625
```

```
#!/usr/bin/perl -w

open(ID,$ARGV[1]);
while(<ID>){
  chomp $_;
  $id_hash{$_}=1;
}

use Bio::SeqIO;

$new=Bio::SeqIO->new(-file=>$ARGV[0], -format=>"fasta");

while($seq=$new->next_seq){
  if(defined $id_hash{$seq->id}){
    print ">",$seq->id,"\n",$seq->seq."\n";
  }
}
```

@col=split(/\t/,$_);
$id_hash{$col[0]}=$col[-2].”-”.$col[-1];

How to we extract the fasta sequences for the domain matches?

We need to modify this program to make it work for this task:

1) Cut the tabular file and create a hash with ID as the key and domain positions as the values
2) Since there are two lines with the same ID, so the 2nd line will overwrite the 1st line
3) Hash of array could solve this program

Hash of array is one of the complex data structures built on top of basic ones (scalar, array and hash), designed for one-to-many data tables

Key	**Value**
Asia	(“China”, “Japan”, “India”)
Europe	(“UK”, “France”, “Germany”)
North America	(“US”, “Canada”, “Mexico”)

$country{$col[0]}=$col[1];

Asia	China
Asia	Japan
Asia	India
Europe	UK
Europe	France
Europe	Germany
North America	US
North America	Canada
North America	Mexico

```
While(<>){
    @col=split(/\t/,$_);
    push(@{$country{$col[0]}},$col[1]);
}
```

value

key

This suggests it’s an array

Curley for the hash name part

push(@array, $element);
In a loop, push $element to an array to create it

vi get-subseq-bioperl.pl

```
#!/usr/bin/perl

use Bio::SeqIO;

open(IN,$ARGV[0]);

while(<IN>){
  chomp $_;
  @col=split(/\t/,$_);
  push(@{$id_hash{$col[0]}},$_);
}

$new=Bio::SeqIO->new(-file=>$ARGV[1], -format=>"fasta");

while($seq=$new->next_seq){
  if(defined $id_hash{$seq->id}){
   @id_array=@{$id_hash{$seq->id}};

   foreach(@id_array){
     @id_col=split(/\t/,$_);
     print ">",$id_col[0],"|",$id_col[-2],"-",$id_col[-1],"\n",
     $seq->subseq($id_col[-2],$id_col[-1])."\n";
   }
  }
}
```

```
perl get-subseq-bioperl.pl GH5.hmm.cowrumen.dm.ps
metagenemark_predictions.faa | less
```

Create a hash of array as explained in the above example

If the ID is defined in the %id_hash, create a temporary array to hold the lines of existing IDs (it's ok if there is only one line in the array)

subseq method is called to help retrieve the domain segments using the position info

Build the fasta description line in a format: >ID|start-end

```
seqret -sequence /media/DATAPART4/z1003529/sequence.gp -outseq g -sformat genbank -
osformat fasta
```

vi format-bioperl.pl

```
#!/usr/bin/perl -w

use Bio::SeqIO;

$in=Bio::SeqIO->new(-file=>$ARGV[0],-format=>$ARGV[1]);
$out=Bio::SeqIO->new(-file=>">$ARGV[2]",-format=>$ARGV[3]);

while ( my $seq = $in->next_seq() ) {
  $out->write_seq($seq);
}
```

```
perl format-bioperl.pl /media/DATAPART4/z1003529/sequence.gp genbank
sequence.gp.fa fasta

perl format-bioperl.pl /media/DATAPART4/z1003529/sequence.gp genbank
sequence.gp.fa fasta 2>h.err
```

```
yyin@glu:~/work/class$ ls /usr/share/perl5/Bio/SeqIO
SeqIO/    SeqIO.pm
yyin@glu:~/work/class$ ls /usr/share/perl5/Bio/SeqIO
abi.pm          chadoxml.pm     excel.pm            game.pm       kegg.pm       MultiFile.pm  scf.pm          tigr.pm
ace.pm          chaos.pm        exp.pm              gbdriver.pm   largefasta.pm nexml.pm      seqxml.pm       tigrxml.pm
agave.pm        chaosxml.pm     fasta.pm            gbxml.pm      lasergene.pm  phd.pm        strider.pm      tinyseq
alf.pm          ctf.pm          fastq.pm            gcg.pm        locuslink.pm  pir.pm        swissdriver.pm  tinyseq.pm
asciitree.pm    embldriver.pm   flybase_chadoxml.pm genbank.pm    mbsout.pm     pln.pm        swiss.pm        ztr.pm
bsml.pm         embl.pm         FTHelper.pm         Handler       metafasta.pm  qual.pm       table.pm
bsml_sax.pm     entrezgene.pm   game                interpro.pm   msout.pm      raw.pm        tab.pm
yyin@glu:~/work/class$
```

Perl one-liner

You don't write codes into a file and then issue "`perl file.pl`" on the command line;
You write the codes directly on the command line, like you are typing regular Linux commands

```
#!/usr/bin/perl

open (IN,$ARGV[0]);

@a=<IN>;

foreach(@a){
   @col=split(/\t/,$_);
   print $col[1],"\tmutation\n";
}
```

=

```
#!/usr/bin/perl

while (<>){
   @col=split(/\t/,$_);
   print $col[1],"\tmutation\n";
}
```

=

```
perl -e 'while(<>){@col=split(/\t/,$_);print $col[1],"\tmutation\n";}' cosmicRaw.txt.head10.6col
```

=

```
cat cosmicRaw.txt.head10.6col | cut -f2 | awk '{print $1,"mutation"}' | sed 's/ /\t/'
```

Perl one-liner

You don't write codes into a file and then issue "`perl file.pl`" on the command line;
You write the codes directly on the command line, like you are typing regular Linux commands

```
perl get-length.pl
metagenemark_predictions.faa | less
```

```
#!/usr/bin/perl -w

use Bio::SeqIO;

$new=Bio::SeqIO->new(-file=>$ARGV[0],
-format=>"fasta");

while($seq=$new->next_seq){
  print $seq->id,"\t", length $seq->seq, "\n";
}
```

```
perl -e 'use Bio::SeqIO; $new=Bio::SeqIO->new(-file=>$ARGV[0],-
format=>"fasta"); while($seq=$new->next_seq){print $seq->id,"\t", length $seq-
>seq, "\n";}' metagenemark_predictions.faa | less
```