

Beginners HOWTO

Brian Osborne, *Cognia Cor*

necessary to run all of *Bioperl* [<http://bioperl.org>]. You'll need to search other repositories to install all of these accessory modules. See the *INSTALL.WIN* [<http://bioperl.org/Core/Latest/INSTALL.WIN>] file for details on how to install them by hand.

6. Creating a sequence, and an object

Our first script will create a sequence. Well, not just a sequence, you will be creating a "sequence object", since *Bioperl* [<http://bioperl.org>] is written in an object-oriented way. Why be object-oriented? Why introduce these odd or intrusive notions into software that should be "biological" or "intuitive"? The reason is that thinking in terms of

Note that `>` in the `-file` argument. This character indicates that we're going to *write* to the file named "sequence.fasta", the same character we'd use if we were using Perl's `open ()` function to write to a file. The `"-format"` argument, `"fasta"`, tells the SeqIO object that it should create the file in fasta format.

Let's put our 2 e

function, and *Bioperl* [<http://bioperl.org>]'s way of retrieving sequences may look odd and overly complicated, at first. But don't use `open`! Using `open` immediately forces you to do the parsing of the sequence file and this can get complicated very quickly. Trust the `SeqIO` object, it's built to open and parse all the common sequence do seqs, it can read and write to files, and it's built to operate with all the other *Bioperl* [<http://bioperl.org>] modules that you will want to use.

Let's read the file we created previously, "sequence.fasta", using `SeqIO`. The syntax will look familiar:

```
#!/bin/perl -w

use Bio::SeqIO;

$seqio_obj = Bio::SeqIO->new(-file => "sequence.fasta",
                             -do seq => "fasta" );
```

One difference is immediately apparent: there is no `>` character. Just as with with the `open()` function this means we'll be reading from the "sequence.fasta" file. Let's add the key line, where we actually retrieve the Sequence object from the file using the `next_seq` method:

```
#!/bin/perl -w

use Bio::SeqIO;

$seqio_obj = Bio::SeqIO->new(-file => "sequence.fasta",
                             -do seq => "fasta" );

$seq_obj = $seqio_obj->next_seq;
```

Here we've used the `next_seq()` method of the [<http://bioperl.org>]

Tip

It may be useful to tell SeqIO the alphabet of the input, using the "-alphabet" argument. What this does is to tell SeqIO not to try to determine the alphabet ("dna", "rna", "protein"). This helps because *Bioperl*


```
$seq_obj->seq("MMTYDFFFFVVNNNNPPPPAAAW");
```

TITLE The nucleotide sequence of the rho gene of E. coli K-12
 JOURNAL Nucleic Acids Res. 11 (11), 3531-3545 (1983)
 MEDLINE 83220759
 PUBMED 6304634
 COMMENT Original source text: Escherichia coli (strain K-12) DNA.
 A clean copy of the sequence for [2] was kindly provided by
 J.L.Pinkham and T.Platt.
 FEATURES Location/Qualifiers
 source 1..1880
 /organism="Escherichia coli"
 /mol_type="genomic DNA"
 /strain="K-12"
 /db_xref="taxon:562"
 aVQ 212..> 1..1880

[illegible]

The corresponding set of values is shown below

```
$report_obj = $blast_obj->blastall($seq_obj);  
  
$result_obj = $report_obj->next_result;  
  
print $result_obj->num_hits;
```

By calling the `blastall` method you're actually running BLAST, creating the report file, and parsing the file's contents.

There's only one requirement here, the term or id that you use to retrieve the sequence object must be unique in the index, these indices are not built to retrieve

`id_parser` method, which accepts the name of a function as an ar

Read about any module, including any of the *Bioperl* [<http://bioperl.org>] modules:

```
>perldoc Bio::SeqIO
```

16.