Apply PERL to BioInformatics

Lecture Note for Computational Biology 1 (LSM 5191)

Jiren Wang
http://www.bii.a-star.edu.sg/~jiren
BioInformatics Institute
Singapore
Outline

- Web Interface to Bioinformatics Tools.
- Introduction to BioPerl.
Web Interface to Bioinformatics Tools

- **NCBI BLAST**

- **FASTA**
  - http://www.ebi.ac.uk/fasta33/index.html

- **ClustalW**

- **EMBOSS**

- **The Sequence Manipulation Suite**
High Throughput BLAST Based on Web Services (@BII)

- Introduction
- Standard BLAST
- PSI-BLAST
- Examples

Standard BLAST

Choose program to use and database to search:

Program: blastp
Database: swissprot

Upload the sequence file in FASTA format below:

(Note: Only the first 200 sequences will be processed if the uploaded file contains more than 200 sequences.)

Or

Enter or paste some sequences in FASTA format below:

(Note: Only the first 200 sequences will be processed if the number of above entered or pasted sequences is more than 200.)
High Throughput BLAST Based on Web Services (@BII)

| Introduction | Standard BLAST | PSI-BLAST | Examples |

### Standard BLAST

**Submission Status:**
Total submission number by all users currently in processing: 1

**Checking Connection to Compute Nodes:**
- Current time: (at 10:47:32 on 09-08-03).
- Connection to compute node "african" is OK (at 10:47:33 on 09-08-03).
- Connection to compute node "asian" is OK (at 10:47:34 on 09-08-03).
- Connection to compute node "indian" is OK (at 10:47:34 on 09-08-03).

**Current System CPU Load:**
- CPU load of compute node "african" is: 1.09%
- CPU load of compute node "asian" is: 1.16%
- CPU load of compute node "indian" is: 1.08%
High Throughput BLAST (3)

Processing Status:
Seq 1 processed on compute node "indian" at 10:47:38 on 09-08-03.
Seq 2 processed on compute node "asian" at 10:47:40 on 09-08-03.
Seq 3 processed on compute node "african" at 10:47:41 on 09-08-03.
Seq 4 processed on compute node "indian" at 10:47:44 on 09-08-03.
Seq 5 processed on compute node "indian" at 10:47:48 on 09-08-03.
Seq 6 processed on compute node "african" at 10:47:54 on 09-08-03.
Seq 7 processed on compute node "asian" at 10:47:55 on 09-08-03.
Seq 8 processed on compute node "indian" at 10:48:02 on 09-08-03.

Sequence and Result List:

(URL: http://mammoth.bii.a-star.edu.sg/webservices/htblast/output/09-08-03/mydir8199/index.html)

Please write down the above URL if you would like to view the sequence and result list again later and save the results you need on your machine.

(Seq1, Result1: gi|116583|sp|P18545|CNRG_HUMAN)
(Seq2, Result2: gi|3183002|sp|P97767|EYA1_MOUSE)
(Seq3, Result3: gi|122207|sp|P08100|OPSD_HUMAN)
(Seq4, Result4: gi|15963539|gb|A1111062.1)
(Seq5, Result5: gi|3183005|sp|P99502|EYA1_HUMAN)
(Seq6, Result6: gi|2492753|sp|Q92781|RDH1_HUMAN)
(Seq7, Result7: gi|9296946|sp|O88874|CYCK_MOUSE)
(Seq8, Result8: gi|132212|sp|P23942|RDS_HUMAN)

Note: Some above results are not ready to be viewed as they are still being processed. These results are Result8. It will take a few seconds to process. Please wait.

ALL RESULTS ARE READY TO BE VIEWED NOW (at 10:48:13 on 09-08-03).
High Throughput BLAST (4)

Processing Time:

Below is the list of sequence processing time by each compute node.

- (Seq1, 5 seconds)
- (Seq2, 14 seconds)
- (Seq3, 11 seconds)
- (Seq4, 3 seconds)
- (Seq5, 13 seconds)
- (Seq6, 9 seconds)
- (Seq7, 7 seconds)
- (Seq8, 9 seconds)

Summation of above time: 71 seconds.
Total processing time by this system with 3 compute nodes: 33 seconds.
Percentage of time reducing: \((71 - 33) / 71 = 53\%\)
Stand-alone BLAST

Steps for downloading and installing stand-alone BLAST executables

```
$ ftp ftp.ncbi.nih.gov
Name (ftp.ncbi.nih.gov:username): anonymous
Password: anonymous
ftp>cd blast/executables
ftp>cd release/2.2.6
ftp>get blast-2.2.6-ia32-linux.tar.gz
ftp>quit
$gunzip blast-2.2.6-ia32-linux.tar.gz
$tar xvf blast-2.2.6-ia32-linux.tar
```
Program blastall

Some arguments

-p Program Name [String]
-d Database [String] default = nr
-i Query File [File In]
-e Expectation value (E) [Real] default = 10.0
-T Produce HTML output [T/F] default = F
Table of BLAST Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blastp</td>
<td>compares an amino acid query sequence against a protein sequence database.</td>
</tr>
<tr>
<td>blastn</td>
<td>compares a nucleotide query sequence against a nucleotide sequence database.</td>
</tr>
<tr>
<td>blastx</td>
<td>compares a nucleotide query sequence translated in all reading frames against a protein sequence database.</td>
</tr>
<tr>
<td>tblastn</td>
<td>compares a protein query sequence against a nucleotide sequence database dynamically translated in all reading frames.</td>
</tr>
<tr>
<td>tblastx</td>
<td>compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database.</td>
</tr>
</tbody>
</table>
Program formatdb

Some arguments

- **i** Input file(s) for formatting
- **p** Type of file
- **o** Parse options:
  - T - True: Parse SeqId and create indexes.
  - F - False: Do not parse SeqId. Do not create indexes.
- **n** Base name for BLAST files [String]

formatdb   -i testdb.txt –p F –o T –n testdb
Package and Module

**Package**
- A package is simply a set of Perl statements assigned to a user defined name space.
- Package declaration: package NAMESPACE

**Module**
- A module is just a reusable package that is defined in a library file whose name is the same as the name of the package (with a .pm on the end).
- Include Perl modules in your program: use Module.
Introduction to CGI

The CGI.pm module

- CGI – Common Gateway Interface
- CGI.pm is a module for programming interactive web pages.
- Its functions are geared toward formatting web pages and creating and processing forms in which user enter information.
How a CGI Application Is Executed?
Forms and CGI

HTML forms
- Provide input to your CGI scripts.

The `<form>` tag

```
<form action="/cgi-bin/test.cgi" method="post">
...
</form>
```
# Quick Reference to Form Tags

<table>
<thead>
<tr>
<th>Form Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;FORM ACTION=&quot;/cgi-bin/test.cgi&quot; METHOD=&quot;POST&quot;</code></td>
<td>Start the form</td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;text&quot; NAME=&quot;name&quot; VALUE=&quot;value&quot; SIZE=&quot;size&quot;&quot;&gt;</code></td>
<td>Text field</td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;password&quot; NAME=&quot;name&quot; VALUE=&quot;value&quot; SIZE=&quot;size&quot;&gt;</code></td>
<td>Password field</td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;hidden&quot; NAME=&quot;name&quot; VALUE=&quot;value&quot;&gt;</code></td>
<td>Hidden field</td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;checkbox&quot; NAME=&quot;name&quot; VALUE=&quot;value&quot;&gt;</code></td>
<td>Checkbox</td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;radio&quot; NAME=&quot;name&quot; VALUE=&quot;value&quot;&gt;</code></td>
<td>Radio button</td>
</tr>
<tr>
<td><code>&lt;SELECT NAME=&quot;name&quot; SIZE=1&gt;</code></td>
<td>Menu (drop-down)</td>
</tr>
<tr>
<td><code>&lt;OPTION SELECTED&gt;Item One&lt;/OPTION&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;OPTION&gt;Item Two&lt;/OPTION&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;/SELECT&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;TEXTAREA ROWS=yy COLS=xx NAME=&quot;name&quot;&gt;</code></td>
<td>Multiline text field</td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;/TEXTAREA&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;submit&quot; NAME=&quot;name&quot; VALUE=&quot;value&quot;&gt;</code></td>
<td>Submit button</td>
</tr>
<tr>
<td><code>&lt;INPUT TYPE=&quot;reset&quot; VALUE=&quot;value&quot;&gt;</code></td>
<td>Reset button</td>
</tr>
<tr>
<td><code>&lt;/FORM&gt;</code></td>
<td>End the form</td>
</tr>
</tbody>
</table>
Examples of Form Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu (drop-down)</td>
<td>What is your most &lt;strong&gt;un&lt;/strong&gt;favorite color:</td>
</tr>
<tr>
<td></td>
<td>Blue ▲</td>
</tr>
<tr>
<td>Select Box</td>
<td>Select one or more colors you do not like:</td>
</tr>
<tr>
<td></td>
<td>Green ▼</td>
</tr>
<tr>
<td></td>
<td>Red ▼</td>
</tr>
<tr>
<td></td>
<td>Blue ▼</td>
</tr>
<tr>
<td></td>
<td>Yellow ▼</td>
</tr>
<tr>
<td>Reset Button</td>
<td>Reset</td>
</tr>
<tr>
<td>Submit Button</td>
<td>Submit</td>
</tr>
<tr>
<td>Text Field</td>
<td>What is your email address?</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Password Field</td>
<td>What is your password?</td>
</tr>
<tr>
<td></td>
<td>!</td>
</tr>
<tr>
<td>Checkbox</td>
<td>Select one or more colors that you like:</td>
</tr>
<tr>
<td></td>
<td>Blue □</td>
</tr>
<tr>
<td></td>
<td>Red □</td>
</tr>
<tr>
<td></td>
<td>Pink □</td>
</tr>
<tr>
<td></td>
<td>Yellow □</td>
</tr>
<tr>
<td>Radio Button</td>
<td>What is your favorite color?</td>
</tr>
<tr>
<td></td>
<td>Blue ▼</td>
</tr>
<tr>
<td></td>
<td>Red ▼</td>
</tr>
<tr>
<td></td>
<td>Pink ▼</td>
</tr>
</tbody>
</table>
Form Action and Method

△ Action
- Specifies the URL of the CGI script that should receive the HTTP request made by the CGI script.
- HTTP – HyperText Transfer Protocol.

△ Method
- Specifies the HTTP request method used when calling the CGI script.
- GET is the standard request method for retrieving a document via HTTP on the Web.
- POST is used with HTML forms to submit information that alters data stored on the web server.
Example of HTML

<HTML>
<TITLE>BLAST Search</TITLE>
<H1><CENTER>BLAST (@BII)</CENTER></H1>
<FORM ACTION="/cgi-bin/blast/blast.pl"
     METHOD = POST
     NAME="MainBlastForm"
     ENCTYPE="multipart/form-data">
<B>Choose program to use and database to search:</B>

<a href="docs/blast_program.html">Program</a>
<select name = "PROGRAM">
 <option> blastn
 <option> blastp
 <option> blastx
 <option> tblastn
 <option> tblastx
</select>

<a href="docs/blast_databases.html">Database</a>
</FORM>
</HTML>
Example of HTML (cont’d)

<select name = "DATALIB">
    <option VALUE = "nr"> nr
    <option VALUE = "swissprot"> swissprot
    <option VALUE = "pat"> pat
    <option VALUE = "yeast"> yeast
    <option VALUE = "ecoli"> ecoli
    <option VALUE = "pdb"> pdb
    <option VALUE = "drosoph"> Drosophila genome
    <option VALUE = "month"> month
    <option VALUE = "est"> est
    <option VALUE = "est_human"> est_human
    <option VALUE = "est_mouse"> est_mouse
    <option VALUE = "est Others"> est_others
    <option VALUE = "htg"> htgs
    <option VALUE = "gss"> gss
    <option VALUE = "sts"> dbsts
    <option VALUE = "mito"> mito
    <option VALUE = "vector"> vector
</select>
Example of HTML (cont’d)

Enter sequence below in <a href="docs/fasta.html">FASTA</a> format

Set subsequence
From: <INPUT TYPE="text" NAME="FROM" SIZE=6>
To: <INPUT TYPE="text" NAME="TO" SIZE=6>

Note: focus() - moving cursor back to a field.
Screenshot of GUI for BLAST
Filehandle

- **A filehandle** is just a name you give to a file, device, socket, or pipe to help you remember which one you are talking about.
- **STDIN** in Perl is a filehandle for standard input.
  ```perl
  $a = <STDIN>;
  @a = <STDIN>;
  chomp($a);
  ```
- **chomp function** removes all trailing newlines from string(s) in paragraph mode ($/ is empty).
  ```perl
  chomp($a);
  ```
Process Filehandle

Using processes as filehandles

- Create a process-filehandle that either captures the output from the process or provides input to the process.
- Open a process-filehandle for reading by putting the vertical bar on the right of the command.

```perl
#!/usr/bin/perl
open(LSPROC, "ls|"PREC);  
@lsoutput = <LSPROC>; 
close(LSPROC); 
print @lsoutput; 
```
#!/usr/bin/perl -w
use strict;
use CGI;
my ($form, $program, $datalib, $sequence, $from, $to);
my ($length);
my (@lines);
my ($state, $line, $title, $seq);
My ($from, $to);
my ($seqfilename, $options);
print "Content-type: text/html\n\n";
$form = new CGI;
$program = $form->param('PROGRAM');
$datalib = $form->param('DATALIB');
$sequence = $form->param('SEQUENCE');
$from = $form->param('FROM');
$to = $form->param('TO');
if ($sequence ne "") {
    $state = 0;
    $seq = "";
    my (@lines) = split /
/, $sequence;
    foreach $line (@lines) {
        if ($line =~ /^>/) {
            $state++;
            if ($state == 1) {
                $title = $line; $title =~ s/\[\s\]+//; $title =~ s/\[\s\]+//;
            }
        } else {
            if ($state == 1) {
                $seq .= $line;
            }
        }
    }
    $seq =~ s/\[\s\]+//g;
    $seq =~ tr/a-z/A-Z/;
    $length = length($seq);
if (($to > $from) && ($from < $length) && ($to > 1)) {
    if ($from < 0) {
        $from = 0;
    }
    if ($to > $length) {
        $to = $length;
    }
    $seq = substr($seq, $from - 1, $to - $from + 1);
}

$seqfilename = "/tmp/seq";
open(OUT, "$seqfilename");
print OUT "$title
$seq"
close(OUT);
$options = "-T -p $program -d /ncbi/blast/db/$datalib -i $seqfilename";
open(IN, "/ncbi/blast/executables/blastall $options |" );
while(<IN>) {
    print ";
}
close(IN);
} else {
    print "Please enter the query sequence in FASTA format.<br>";
}
Introduction to BioPerl

BioPerl

- A collection of perl modules that facilitate the development of perl scripts for bioinformatics applications.
- A project made up of biologists, bioinformaticians, and computer scientists.
- An open source toolkit for life science research.
The Goal of BioPerl

To help a user focus on his/her specific problem at hand, such as the logic needed to filter hits in a BLAST report by certain criteria, rather than on the actual mechanics of parsing that BLAST report.
Some Observations

The Bioperl project grew out of the following observations:

- Although file formats of different analysis programs are different, the information they represent is the same.
- The number of data structures needed to represent information flow is limited, and common to most applications such as sequences, annotation, features, and alignments.
- A set of operations is commonly performed on these data structures.
Object-oriented Methodology

Object-oriented programming is the practice of grouping related tasks together into logical and broadly applicable components.

Object-oriented programming methodology allows accurate modeling of the problem domain, leading to more robust, valid, and reusable code.

The entities are represented in code through the use of Perl modules and, where appropriate, form elements in an inheritance hierarchy.
The Main Focus of Bioperl Modules

- Perform sequence manipulation.
- Provide access to various databases (both local and web-based).
- Parsing of the results of various molecular biology programs.
Modules for Typical Tasks

- Accessing sequence data from local and remote database.
- Transforming formats of database / file records.
- Manipulating individual sequences.
- Searching for "similar" sequences.
- Creating and manipulating sequence alignments.
- Searching for genes and other structures on genomic DNA.
- Developing machine readable sequence annotations.
PERL’s Objects

- **An object** is simply a referenced thingy that happens know which class it belongs to.
- **A class** is simply a package that happens to provide methods to deal with objects.
- **A method** is simply a subroutine that expects an object reference as its first argument.
Method Invocation

The object-oriented syntax form looks like this:

CLASS_OR_INSTANCE->METHOD(LIST)
## Major BioPerl Module Groups

<table>
<thead>
<tr>
<th>Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bio::Seq</td>
<td>Sequences and their properties</td>
</tr>
<tr>
<td>Bio::SeqIO</td>
<td>Sequence data input/output</td>
</tr>
<tr>
<td>Bio::Index</td>
<td>Flat-file sequence database indexing and retrieval</td>
</tr>
<tr>
<td>Bio::DB</td>
<td>Remote database access for sequences and references via HTTP</td>
</tr>
<tr>
<td>Bio::DB::GFF</td>
<td>SQL GFF database for DAS and GBrowse backends</td>
</tr>
<tr>
<td>Bio::SeqFeature</td>
<td>Annotations or features that have a sequence location</td>
</tr>
<tr>
<td>Bio::Annotation</td>
<td>Generic annotations such as Comments and References</td>
</tr>
<tr>
<td>Bio::AlignIO, Bio::SimpleAlign</td>
<td>Multiple sequence alignments and their Input/Output</td>
</tr>
<tr>
<td>Bio::LiveSeq, Bio::Variation</td>
<td>Sequence variations and mutations</td>
</tr>
<tr>
<td>Bio::Search, Bio::SearchIO</td>
<td>Sequence database searches and their Input/Output</td>
</tr>
<tr>
<td>Bio::Tools</td>
<td>Miscellaneous analysis tools</td>
</tr>
<tr>
<td>Bio::Tools::Run</td>
<td>Wrapper for executing local and remote analyses</td>
</tr>
<tr>
<td>Bio::Tree, Bio::TreelO</td>
<td>Phylogenetic trees and their Input/Output</td>
</tr>
<tr>
<td>Bio::Structure</td>
<td>Protein structure data</td>
</tr>
<tr>
<td>Bio::Map, Bio::MapLO</td>
<td>Biological maps and their Input/Output</td>
</tr>
<tr>
<td>Bio::Biblio, Bio::DB::Biblio</td>
<td>Bibliographic References and Database retrieval</td>
</tr>
<tr>
<td>Bio::Graphics</td>
<td>Graphical displays of sequences</td>
</tr>
</tbody>
</table>
Sequence File seqa.fasta

> seq1
MATHHTLWMGLALLGVLGDLQAAPEAQVSVQPNFQQDKFLGRWFSAGLAS
NSSWLREKKAALSMCKSVVAPATDGGLNLTSTFLRKNQCETRMLLQPAG
SLGSYSYRSHPHWGSTYSVSVVETDYDQYALLYSQGSKGPGEDFRMATLYS
RTQTPRAELKEKFTAFCKAQQGFTEDTIVFPQTDKCMTEQ
> seq2
APEAQVSVQPNFQPDKFLGRWFSAGLASNSSWLQEKKAALSMCKSVVAPA
ADGGFNLTSTFLRKNQCETRMLLQPGLSLGSYSYRSHPHWGSTYSVSVVE
TDYDHYALLYSQGSKGPGEDFRMATLYSRTQTPRAELKEKFTAFCKAQQGF
TEDSIVFPQTDKCMTEQ
Bio::SeqIO

Bio::SeqIO is a handler module for the formats in the SeqIO set (eg, Bio::SeqIO::fasta).

```perl
#!/usr/bin/perl -w
use Bio::SeqIO;
$str = Bio::SeqIO->new( -file=>'seqa.fasta', '-format' => 'Fasta');
$input = $str->next_seq();
$input2 = $str->next_seq();
print "Sequence 1
";
print $input->id, "\n";
print $input->seq, "\n\n";
print "Sequence 2
";
print $input2->id, "\n";
print $input2->seq, "\n\n";
```
Example: Sequence Data Format Transfer

```perl
#!/usr/bin/perl -w
use Bio::SeqIO;
use strict;
my $in  = Bio::SeqIO->new('-file'  => "seqa.fasta",
                          '-format' => 'Fasta');
my $out = Bio::SeqIO->new('-file'  => ">seqa.embl",
                          '-format' => 'EMBL');

while ( my $seq = $in->next_seq() ) {
  $out->write_seq($seq);
}
```
Example: Sequence Data Format Transfer (cont’d)

Output result:

ID  seq1   standard; AA; UNK; 190 BP.
XX
AC  unknown;
XX
DE
XX
FH Key Location/Qualifiers
FH
XX
SQ  Sequence 190 BP; 17 A; 4 C; 14 G; 17 T; 138 other;
mathhtlwmq lallvlgdld qapeaqvsqv qpnfqqdkfl grwfsaglas nsswlrekka  60
alsmcksvva patdgllnlt stflrkncqce trtmllqpag sfigsyrsrp hwgstysvsv  120
vetdydqyal lysqgskkpg edfrmatlys rtqtpraelk ekftafckaq gftedtivfl  180
pqtdkcmteq  190
//
ID  seq2   standard; AA; UNK; 168 BP.
XX
AC  unknown;
XX
DE
XX
FH Key Location/Qualifiers
FH
XX
SQ  Sequence 168 BP; 14 A; 4 C; 11 G; 13 T; 126 other;
apeaqvsvqp nfqdpdkflgr wfsaglasns slqekkaal smcksvvapa adgfnltst  60
flrknqctr tmllgpdsgl gqstysphw qstysvsvve tdydhyally sqgskgpged  120
frmatlysrt qtpraelkek ftafckaqgf tedivflpq tdkcmteq  168
//
#!/usr/bin/perl -w
use Bio::Tools::Run::StandAloneBlast;

# use the following array to input parameter
@params = ('d' => 'swissprot',
           'o' => 'blast.out',
           '_READMETHOD' => 'Blast',
           'T' => 'F',
           'p' => 'blastp');

$factory = Bio::Tools::Run::StandAloneBlast->new(@params);

# Blast a sequence against a database:
$str = Bio::SeqIO->new( -file=>'seqa.fasta', '-format' => 'Fasta');
$input = $str->next_seq();
print "Generate the searching result now: \n";
$blast_report = $factory->blastall($input);
Example: Parsing a BLAST Report

```perl
#!/usr/bin/perl -w
use Bio::SearchIO;
# Let's parse a BLAST report
my $search = new Bio::SearchIO( -format => 'blast', -file   => 'blast.out');
my @HitsToSave = ();
my $cutoff_Evalue = 0.001;
my $cutoff_Len    = 120;
# iterate over each query sequence
while (my $result = $search->next_result) {
    # iterate over each hit on the query sequence
    while (my $hit = $result->next_hit) {
        # iterate over each HSP in a hit
        while (my $hsp = $hit->next_hsp) {
            if ($hsp->evalue < $cutoff_Evalue &&
                $hsp->length('total') >= $cutoff_Len) {
                push @HitsToSave, $hit;
                last; # Only want to process each Hit once
            }
        }
    }
}
print "Hits:\n";
foreach my $hit (@HitsToSave) {
    print $hit->name, "\n";
}
```
Input Sequence for ClustalW

>seq1
MATHHTLWMGLALLGVLGDLOQAPEAQVSVQPNFQQDKFLGRWFSAGLAS
NSSWLREKKAAALSCKSVVAPATDGGGLNLTSTFLRKNQCETRTMLLQPAG
SLGSYSYRSPHWGSTYSVSVVETDYDQYALLYSGSKGPGEDFRMATLYS
RTQTPRAELKEKFCAFCKAQGFTEDTIVFLPQTDKCMTEQ

>seq2
APEAQVSVQPNFQQDKFLGRWFSAGLASNSSWQEKKAALSMCKSVVAPA
ADGGFNLSTFLKNQCETRTMLLPDGLSGSYRSPHWGSTYSVSVVE
TDYDHYALLYSQGSKGPGEDFRMATLYSRTQTPRAELKEKFCAFCKAQGF
TEDSIVFLPQTDKCMTEQ

>seq3
MAALRMLWMLVLLGLLGFQTPAQGHDTVQPNFQQDKFLGRWYSAGLAS
NSSWFREKKAVLYMCKTVVAPSTEGLNLTTSTFLRKNQCETKIMVLQPAG
APGHYTYSSPHSGSIHSVSVEANYDEYALLFSRTKPGQDFRMATLYS
RTQTLKDELKEKFTTFSKAQGLTEEDIVFLPQPDKCIQE
#!/usr/bin/perl -w
use strict;
use Bio::Tools::Run::Alignment::Clustalw;
my $inputfilename = 'seq.txt';
my $outfile = "clustalw_result.txt";
# Build a clustalw alignment factory
my @params = ('ktuple' => 2, 'matrix' => 'BLOSUM', 'outfile'=> $outfile);
my $factory = Bio::Tools::Run::Alignment::Clustalw->new(@params);
my $clustalw_location = $factory->program("/home/webuser/clustalw1.82/clustalw");
print ("\nThe location of clustalw is $clustalw_location.\n");
# Pass the factory a list of sequences to be aligned.
my $aln = $factory->align($inputfilename); # $aln is a SimpleAlign object.
print " length = ", $aln->length, "\n";
print " no_residues = ", $aln->no_residues, "\n";
print " no_sequences = ", $aln->no_sequences, "\n";
print " percentage_identity = ", $aln->percentage_identity, "\n";
Output on Screen

The location of clustalw is /home/webuser/clustalw1.82/clustalw.

CLUSTAL W (1.82) Multiple Sequence Alignments

Sequence format is Pearson
Sequence 1: seq1          190 aa
Sequence 2: seq2          165 aa
Sequence 3: seq3          189 aa
Start of Pairwise alignments
Aligning...
Sequences (1:2) Aligned. Score: 95
Sequences (1:3) Aligned. Score: 72
Sequences (2:3) Aligned. Score: 70
Guide tree file created: ./seq.dnd
Start of Multiple Alignment
There are 2 groups
Aligning...
Group 1: Sequences: 2    Score:2467
Group 2: Sequences: 3    Score:2701
Alignment Score 2488
GCG-Alignment file created [clustalw_result.txt]
  length = 190
  no_residues = 26
  no_sequences = 3
  percentage_identity = 78.3783783783784
Output: clustalw_result.txt

PileUp

MSF: 190  Type: P  Check: 4001 ..

Name: seq1 oo Len: 190 Check: 5441 Weight: 23.3
Name: seq2 oo Len: 190 Check: 3046 Weight: 30.0
Name: seq3 oo Len: 190 Check: 5514 Weight: 46.6

//

seq1  MATHHTLWMG LALLGVLGDL QAAPEAQVSV QPNFQQDKFL GRWFSAGLAS
seq2  .......... .......... ..APEAQVSV QPNFQPDKFL GRWFSAGLAS
seq3  MAALRMLWMG LVLLGGLGFP QTPAQHDTV QPNFQQDKFL GRWYSAGLAS

seq1  NSSWLREQKAA LSMCKSVVAA PATDGGLNLT STFLRKNQCE TRTMLQPAKG
seq2  NSSWQ.EKKA LSMCKSVVAA PAADGFGNLT STFLKN.QCE TRTMLQPGD
seq3  NSSWREKKA VLYMCKTVVAA PSTEGGLNLTV STFLRKNQCE TKIMVLPAG

seq1  SLGSYSYRSVP HWGSTYSVSV VETDYDQYAL LYSQGSKGP EDFRMATLVS
seq2  SLGSYSYRSVP HWGSTYSVSV VETDYDHYAL LYSQGSKGP EDFRMATLVS
seq3  APGHYTRSSP HSGSIHSVSV VEANYDEYAL LFSRGTKGPQ QDFRMYATLVS

seq1  RTQTPRAELK EKFTAFCAKQ GFTEDTIVFL PQTDKCMTEQ
seq2  RTQTPRAELK EKF.AFCKAQ GFTEDSIVFL PQTDKCMTEQ
seq3  RTQTLKDELK EKFITFSKAP GLTEEDIVFL PQPDKCIQEQ.