Introduction to Programming: Perl for Biologists

Timothy M. Kunau
Center for Biomedical Research Informatics
Academic Health Center
University of Minnesota
kunau@umn.edu

Bioinformatics Summer Institute 2007
Introduction to Programming: Day three

Timothy M. Kunau
Center for Biomedical Research Informatics
Academic Health Center
University of Minnesota
kunau@umn.edu

Bioinformatics Summer Institute 2007
Day I

- Art and Programming
- Getting Started
- Biology and Computer Science
- Bioinformatics Data
- Perl basics:
  - Strings and Variables
  - Math and Logic
  - Looping, operators, and functions
Day II

- Data from outside the program
- Writing out data
- Data into arrays and hashes
  - Array operations
- Scope and Good practices
- RegEx
Day III

- Review Homework
- DBI
- BioPerl
- Singularity
- Resources
- Links
Day II: assignment.

1. Modify one of your existing programs to do something useful using a **Regular Expression**. (see the last lab)

2. Read about **Perl DBI**. (Safari on-line documentation is available.)

3. Read about **BioPerl**. (Safari and CPAN)

4. **Write a paragraph** describing what you hope to do with Perl in your BSI project and email it to me.
Perl DBI

- **Database Interface** module, provides connectivity to databases.

- creates a clean [API](#) for connecting to databases, performing queries, updates and deletions without the need to know the details concerning the underlying database.

- A script written with DBI can be used with a [MySQL](#) database, an [Oracle](#) database, or [Sybase](#), and many others with only minor changes.

- a layer of indirection?
Perl DBI

- Tim Bunce, (Mr. DBI)
- Author and maintainer of the DBI, DBD::Oracle, and Oracle::OCI modules, and author and co-maintainer of The Perl Module List.
- Founder and CTO of Data-Plan Services, a Perl, database consultancy with an international client base.
- Awarded by British Telecom for his role in the rapid development of their Call Management Information service, a system implemented in Perl.
- He wrote the book.
Programming the Perl DBI

- Hardcover: 364 pages
- Publisher: O'Reilly Media, Inc.; 1 edition (February 4, 2000)
- Language: English
- ISBN: 1565926994
- Product Dimensions: 9.2 x 7.0 x 0.7 inches
- Shipping Weight: 1.3 pounds
- Average Customer Review: 4.5/5 based on 47 reviews.
DBI extensions

DBIx -- Extensions to the DBI

DBIx::
::Abstract
::AnyDBD
::CGITables
::Copy
::FullTextSearch
::glueHTML
::HTMLView
::OracleSequence
::Password
::Recordset
::Table
::TableAdapter
::Tree
::XML_RDB
::DBSchema
::XMLMessage
::Browse
::XHTML_Table
::SchemaView
::SystemCatalog
::dbMan
::Sequence
::TextIndex
::FetchLoop

Rmp0? Wrapper for DBI that generates SQL
bdp0? Module to make cross db applications easier
adp0? Easy DB access from a CGI
adp0? Copying databases
bmp0? Index documents with MySQL as storage
bdp0? CGI interface to DBI databases
cdp0? Creating web userinterfaces to DBI dbs
adp0? 00 access to Oracle sequences via DBD-Oracle
Mdp0? Abstraction layer for database passwords
bmp0? DB-Abractionlayer / Access via Arrays/Hashs
bdp0? 00 access to DBI database tables
adp0a A object-relational table adapter class
adp0? Expand self-referential table into a tree
????? Creates XML from DBI datasources
bmp0? Database-independent schema objects
bdp0? Exchange of XML messages between DBI sources
bdp0g A class to browse related tables via CGI/Web
Rdp0? SQL query result set to XHTML table
adp0? Retrieving and drawing of DB schema (Tk)
Rdp0? Accessing system catalog in common databases
bmp0p Tk/cmdline database manipulating tool
Mdp0? Database independent ID generation
bdp0? Creates fulltext indexes of SQL text columns
Rdp0g Fetch with change detection and aggregates
DBD extensions

DBD::
::ASAny
::Altera
::CSV
::DB2
::Empress
::FreeTDS
::SearchServer
::Illustra
::Informix
::Informix4
::Ingres
::Multiplex
::ODBC
::Oracle
::QBase
::RAM
::SQLrelay
::Solid
::Sqlflex
::Sybase
::Unify
::XBase
::mSQL
::mysql
::pNET
::InterBase
::RDB
::DtfSQLmac
::ADO
::Excel
::Recall
::Sprite
::PrimeBase
::mysqlPP
::PgPP
::Oracle
::Ingperl
::DBD::DBI
::SMIRNIO
::DSOUFLIS
::JZUCKER
::MHH
::SWILLIAM
::SPANNIN
::SHARI
::PMH
::JOHNLI
::GTHYNI
::HTOUG
::TKISHEL
::DBIML
::DBIML
::BENLI
::JZUCKER
::DMOW
::TWENRICH
::INFOFLEX
::MEWP
::HMBRAND
::JANPAZ
::JWIED
::JWIED
::JWIED
::EDPRATOMO
::ASTILLER
::TWEBGNER
::TLOWREY
::KWIITKNR
::AGUL
::TURNERJW
::PRIMEBASE
::OYAMA
::OYAMA
::HTOUG
#!/usr/local/bin/perl -w
use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT',
                        'adminuser',
                        'password_goes_here',
                        {
                          RaiseError => 1,
                          AutoCommit => 0
                        }
                      ) || die "DB connection not made: $DBI::errstr";

my $sql = qq{ SELECT USERNAME FROM dba_users }; my $sth = $dbh->prepare( $sql ); $sth->execute();

my( $username ); $sth->bind_columns( undef, \$username );

print "list of usernames:

";
while( $sth->fetch() ) {
    print "$username
";
}

$sth->finish(); $dbh->disconnect();

DBI: Oracle

12
#!/usr/local/bin/perl -w

use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT',
    'system',
    'password_goes_here',
    {
        RaiseError => 1,
        AutoCommit => 0
    }
) || die "DB connection not made: $DBI::errstr";

my $sql = qq{ SELECT USERNAME FROM dba_users };

my $sth = $dbh->prepare( $sql );

$sth->execute();

my( $username );
$sth->bind_columns( undef, \$username );

print "list of usernames:

";
while( $sth->fetch() ) {
    print "$username
";
}

$sth->finish();
$dbh->disconnect();
/usr/local/bin/perl -w

use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT',
                        'system',
                        'password_goes_here',
                        {RaiseError => 1,
                        AutoCommit => 0
                        } ) || die "DB connection not made: $DBI::errstr";

my $sql = qq{ SELECT USERNAME FROM dba_users };

my $sth = $dbh->prepare( $sql );

$sth->execute();

my( $username );
$sth->bind_columns( undef, \$username );

print "list of usernames:

";
while( $sth->fetch() ) {
    print "$username
";
}

$sth->finish();
$dbh->disconnect();
#!/usr/local/bin/perl -w
use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT',
                       'system',
                       'password_goes_here',
                       { RaiseError => 1,
                         AutoCommit => 0
                       } ) || die "DB connection not made: $DBI::errstr";

my $sql = qq{ SELECT USERNAME FROM dba_users };
my $sth = $dbh->prepare( $sql );

$sth->execute();
my( $username );
$sth->bind_columns( undef, \$username );

print "list of usernames:

";
while( $sth->fetch() ) {
    print "$username
";
}

$sth->finish();
$dbh->disconnect();
#!/usr/local/bin/perl -w
use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT', 'system', 'password_goes_here', {
    RaiseError => 1,
    AutoCommit => 0
});  

my $sql = qq{SELECT USERNAME FROM dba_users};

my $sth = $dbh->prepare( $sql );

$sth->execute();

my( $username );
$sth->bind_columns( undef, \$username );

print "list of usernames:\n\n";

while( $sth->fetch() ) {
    print "$username\n";
}

$sth->finish();
$dbh->disconnect();
#!/usr/local/bin/perl -w
use strict;
use DBI;
my $dbh = DBI->connect('dbi:Oracle:PGT', 'system', 'password_goes_here',
    { RaiseError => 1, AutoCommit => 0 });
    } || die "DB connection not made: $DBI::errstr";

my $sql = qq{ SELECT USERNAME FROM dba_users };
my $sth = $dbh->prepare($sql);
$sth->execute();
my( $username );
$sth->bind_columns( undef, $username );

print "list of usernames:

while( $sth->fetch() ) {
    print "$username
";
}

$sth->finish();
$dbh->disconnect();
#!/usr/local/bin/perl -w
use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT',
                        'system',
                        'password_goes_here',
                        {
                          RaiseError => 1,
                          AutoCommit => 0
                        }
) || die "DB connection not made: $DBI::errstr"

my $sql = qq{ SELECT USERNAME FROM dba_users }
my $sth = $dbh->prepare( $sql );

$sth->execute();
my( $username );
$sth->bind_columns( undef, \$username );

print "list of usernames:\n\n";
while( $sth->fetch() ) {
    print "$username\n";
}

$sth->finish();
$dbh->disconnect();
#!/usr/local/bin/perl -w

use strict;
use DBI;

my $dbh = DBI->connect( 'dbi:Oracle:PGT',
                        'system',
                        'password_goes_here',
                        { RaiseError => 1,
                          AutoCommit => 0
                        } ) || die "DB connection not made: $DBI::errstr";

my $sql = qq{ SELECT USERNAME FROM dba_users };

my $sth = $dbh->prepare( $sql );

$sth->execute();

my( $username );
$sth->bind_columns( undef, $username );

print "list of usernames:\n\n";
while( $sth->fetch() ) {
    print "$username\n";
}

$sth->finish();
$dbh->disconnect();
#!/usr/local/bin/perl -w

use strict;
use DBI;

my $server = 'host=paralog.ahc.umn.edu';
my $db = 'test';
my $username = 'tester';
my $password = 'test_password_goes_here';

my $dbh = DBI->connect('dbi:mysql:$db:$server',
                        $username,
                        $password,
                        {
                          RaiseError => 1,
                          AutoCommit => 0
                        })
          or die "DB connection not made: $DBI::errstr";

# something interesting could be done with $dbh

print "connected\n"
$dbh->disconnect;
exit;
#!/usr/local/bin/perl -w

use strict;
use DBI;

my $server =   'host=paralog.ahc.umn.edu';
my $db =       'test';
my $username = 'tester';
my $password = 'test_password_goes_here';

my $dbh = DBI->connect('dbi:mysql:$db:$server',
                        $username,
                        $password,
                        {
                        RaiseError => 1,
                        AutoCommit => 0
                        }
                    ) or die "DB connection not made: $DBI::errstr"

# something interesting could be done with $dbh

print "connected\n";
$dbh->disconnect;

exit;
Perl DBI: MySQL and Oracle

● MySQL

```perl
my $dbh = DBI->connect('dbi:mysql:$db:$server',
   $username, $password,
     { RaiseError => 1,
       AutoCommit => 0
      }
   ) || die "DB connection not made: $DBI::errstr";
```

● Oracle

```perl
my $dbh = DBI->connect( 'dbi:Oracle:PGT',
   'system',
   'password_goes_here',
     { RaiseError => 1,
       AutoCommit => 0
      }
   ) || die "DB connection not made: $DBI::errstr";
```
What can it do for you?

• Read and write LOBs in chunks
• Create and manipulate collections, iterators, user defined types
• Cursors, variable-length arrays, nested tables, etc.
• Have multiple users share the same database connection
  • very handy for web servers
• Have multiple processes share the same transaction
  • very handy for high volume data loading
• Non-blocking mode for OCI function calls
  • very handy for GUIs etc.
Perl DBI References

- http://dbi.perl.org/
- http://www.oreilly.com/catalog/perldbi/
- http://www.cpan.org/
#!/usr/bin/perl
#
use strict;
use DBI;

my $server = 'host=paralog.cbri.umn.edu';
my $db = 'bsi_students';
my $username = 'bsi_students';
my $password = 'bi0m3s';

my $dbh = DBI->connect("dbi:mysql:$db:$server", $username, $password)
or die "Database connection not made: $DBI::errstr";

print "connected\n";

$dbh->disconnect;

exit;
#!/usr/bin/perl
#
use strict;
use DBI;

my $server =   'host=paralog.cbri.umn.edu';
my $db =       'bsi_students';
my $username = 'bsi_students';
my $password = 'bi0m3s';

my $dbh = DBI->connect("dbi:mysql:$db:$server", $username, $password)
or die "Database connection not made: $DBI::errstr";

print "connected\n";
$dbh->disconnect;
exit;

Something interesting.
**LAB: DBI**

```sql
mysql> desc crow.legumefams_est;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| accession | varchar(12) |      | PRI |         |       |
| taxon_id  | int(11)     |      | MUL | 0       |       |
| seqlen    | int(11)     |      | MUL | 0       |       |
| seqtext   | text        |      |     |         |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```
LAB: DBI, something interesting

```
print "connected\n";

my $sql = qq{ select * from crow.legumefams_est limit 5 };
my $sth = $dbh->prepare( $sql );
$sth->execute();

my $accession; my $taxon_id; my $seqlen; my $seqtext;
$sth->bind_columns( $accession, $taxon_id, $seqlen, $seqtext );

while( $sth->fetch() ) {
    print "$accession:$taxon_id:$seqlen:$seqtext\n";
}
$sth->finish();

$dbh->disconnect;

exit;
```
#!/usr/bin/perl
#
use strict;
use DBI;

my $server = 'host=paralog.cbri.umn.edu';
my $db =       'bsi_students';
my $username = 'bsi_students';
my $password = 'bi0m3s';

my $dbh = DBI->connect("dbi:mysql:$db:$server", $username, $password)
or die "Database connection not made: $DBI::errstr"

print "connected\n";

my $sql = qq{ select * from crown.legumefams_est limit 5 };
my $sth = $dbh->prepare( $sql );
$sth->execute();

my $accession; my $taxon_id; my $seqlen; my $seqtext;
$sth->bind_columns( \$accession, \$taxon_id, \$seqlen, \$seqtext );

while( $sth->fetch() ) {
    print "$accession:$taxon_id:$seqlen:$seqtext\n";
}
$sth->finish();

$dbh->disconnect;
exit;
LAB: break it.

What happens?:

1. You change the password variable?
2. Are the error messages helpful?
3. Can you anticipate possible errors?
4. How would you protect your user from errors?
LAB: add to it.

Can you?:

1. Process the database sequence strings?

2. Add a subroutine that provides a reverse transcription service for the database sequence strings?

3. Test for a poly-T tail and not print that entry?

4. Create a FASTA file of database row entries?
BioPerl Project

• A collaborative international project dedicated to the creation of Open Source modules for bioinformatics research.

• Common items in bioinformatics are represented as objects in BioPerl.

• Reading and writing sequence files is a matter of calling methods on Bio::SeqIO objects.

• Object methods can translate sequence files.

• Objects are available to parse output from common processing programs.

• Objects manage alignments and annotation.

• http://www.bioperl.org/
BioPerl

• Ewan Birney: initial work.
• Senior Scientist at EMBL, working at EBI.
• Ph.D. BioChemist
• The 2003 Francis Crick Award from the Royal Society
• The 2005 Benjamin Franklin Award for contributions in Open Source Bioinformatics, awarded by Bioinformatics.org in association with BioIT
What BioPerl can do.

- Retrieve sequences from public databases.
- Translate sequences from one format to another.
- Identify restriction and cleaving sites.
- Display 3D structures with PDB files.
- Construct phylogenetic trees and genetic maps.
- Interface with several common algorithms, such as BLAST | CLUSTALW | HMMer, performing the processing remotely or locally.
- Visualize results.
Remember yesterday?
#!/usr/bin/perl -w

use Bio::DB::GenBank;
use Bio::Perl;

my $gbid = "MUSIGHBA1";
my $acc  = "AF303112";

$gb = new Bio::DB::GenBank();

# returns a Seq object, by GenBankID
$seq1 = $gb->get_Seq_by_id($gbid);

write_sequence("./$gbid.fa", 'fasta', $seq1);

# this returns a Seq object by Accession Number:
$seq2 = $gb->get_Seq_by_acc($acc);

write_sequence("./$acc.fa", 'fasta', $seq2);

exit;

% ls -al *.fa
-rw-r--r--  1 kunau  cbri   1708 Jun  7 15:51 AF303112.fa
-rw-r--r--  1 kunau  cbri    494 Jun  7 15:51 MUSIGHBA1.fa
LAB: BioPerl, FASTA from GenBank

#!/usr/bin/perl -w
#
# Retrieve GenBank entries over the Web
#
use Bio::DB::GenBank;
use Bio::SeqIO;
my $gb = new Bio::DB::GenBank;

# the output stream can be a file
# or STDOUT, see Bio::SeqIO
my $seqout = new Bio::SeqIO(-fh => \*STDOUT, -format => 'fasta');

# Retrieve a nucleotide coding sequence when I have a protein gi number?
my $seq = $gb->get_Seq_by_id('8923604');
$seqout->write_seq($seq);
exit;

% pico getGenBank.pl
37
#!/usr/bin/perl -w
#
# Retrieve GenBank entries over the Web
#
use Bio::DB::GenBank;
use Bio::SeqIO;
my $gb = new Bio::DB::GenBank;

# the output stream can be a file
# or STDOUT, see Bio::SeqIO
my $seqout = new Bio::SeqIO(-fh => ">STDOUT, -format => 'fasta');

# Retrieve a nucleotide coding sequence when I have a protein gi number?
my $seq = $gb->get_Seq_by_id('8923604');
$seqout->write_seq($seq);

# by accession
$seq = $gb->get_Seq_by_acc('AF303112');
$seqout->write_seq($seq);

exit;
#!/usr/bin/perl -w
#
# Retrieve GenBank entries over the Web
#
use Bio::DB::GenBank;
use Bio::SeqIO;
my $gb = new Bio::DB::GenBank;

# the output stream can be a file
# or STDOUT, see Bio::SeqIO
my $seqout = new Bio::SeqIO(-fh => "/STDOUT, -format => 'fasta');

# Retrieve a nucleotide coding sequence when I have a protein gi number?
my $seq = $gb->get_Seq_by_id('8923604');
$seqout->write_seq($seq);

# by accession
$seq = $gb->get_Seq_by_acc('AF303112');
$seqout->write_seq($seq);

# or using the batch method
my $seqio = $gb->get_Stream_by_id([ qw( J00522 NP_000866 2981014 )]);

while( defined ($seq = $seqio->next_seq ) ) {
    $seqout->write_seq($seq);
}
exit;
LAB: BioPerl, FASTA from GenBank - results

kunau@l1 [~/bsi2007] % ./getGenBank.pl

>NP_060389 syntaxin 17 [Homo sapiens].
MSEDEEKVLRLRLEPAIQFIFIKIVIPTNLERLRKHQ1IEKYQRUCRIWDKLHEEHINAGR
TVQVQLRNSIREIKECLCLVRDDLVLVLRMRIDFVKEEAATAEFLQLHLESVEELKQF
NDEETLQLQPPLTRSMVGGAFHTTEAEASSQSLTQIYALPEIPDQNAAESRETLEADLI
ELSQVLTDPSLVLNSEQKIDSIAHDHNSSANVEEGTNKLGAAYKLAAPVAGALIG
GMVGGPIGLLACFKVAGIAALGGVLGFTGGKLIQRKKQKMMEKLTSSCPDLPSQTDKK
CS

>AF303112 Physarum polycephalum fragmin60 (frg60) mRNA, complete cds.
ATGGAGCCCAAGCAAGGATACCTTCTTGTAAAAATGTAGAAGCTCGCAAGCTAGCATCT
AAGGATGCTGAGGGAGGGATCATACGCGATTGTTGGACTCCTGAATTCTAAGGGT
GACCAAGGCCGACTTGTGCAGATACCCATGACCTGATCAAATTTCCCAGTGATCGTCAAAAGGTATTATGCAGAA
AATCTCTAACCACCAGGAAAGTAGAGGGAGTCTACGAATTTAACCGGATATTGACGGA
AAGGCTGCGATTTCTCCAGGTACGCCATGTGAGGTTACTTATGATGACTTTTCTG
GGCAATTTTAATATCAATCCGAGAGATCTGCTGAGCTCCAGGCTGATCGATAATTTGGTTC
CCCTACACCGACTGATCTCTTCTCCTTCTGATAAAATCCTAGCCCTTACAGAGAGGCTACAGAGATG
ACTGATCTGTTATGTACAAAAATCAATTTTTCAGTGAAGCTGAGGCTTACCTGCAACATCGCAAT
CTTGAGACCCGATCTCTAGAAAARGAGGTGCTGGCAGAGGCTGAGATCGTCTGCTGAC
AAATCCCGCAGGACAACACCGGCGCTCTGGAATCTGGGGATATCGTGAAGGAAAATCTGAT
GTGATCGATGAAAGTGTAGATAGTGGAAGTGCTGATTTTTGGAACAAACTGGGTGGCAAA
...
LAB: break it.

What happens when?:

1. Make up an accession number?
2. Was the error message helpful?
3. How would you protect your user from an error like this? (Create a subroutine with a regex to check if the accession number adheres to the standard format.)
LAB: add to it.

Can you?:

1. Add ‘use strict;’? Did it run without errors?

2. Find other gi numbers or access ids of interest?

3. Create a loop and iterate across an array of access ids, processing each in some interesting way?

4. Add a subroutine that provides a reverse transcription service, but only for the DNA entries?
Overview of Object packages

- The toolkit is divided into several packages, most people will only want to deal with the Core package.

  - **Core package** provides the main parsers, this is the basic package and it's required by all the other packages (bioperl-live CVS directory)

  - **Run** package provides wrappers for executing some 60 common bioinformatics applications (bioperl-run in CVS)

  - **Ext** package is for C-language extensions including some alignment algorithms and an interface to the Staden IO library (bioperl-ext in CVS).

  - **GUI** package includes some basic widgets in Perl-Tk (bioperl-gui in CVS).
Overview of Objects

• **BioPerl db** is a subproject to store sequence and annotation data in a BioSQL relational database (bioperl-db in CVS).

• **Pedigree** package is for manipulating genotype, marker, and individual data for linkage studies (bioperl-pedigree in CVS).

• **Microarray** package has preliminary objects for manipulating some microarray data formats (bioperl-microarray in CVS).

★★ **Network** package parses and analyzes protein-protein interaction data (bioperl-network in CVS).

★★ **Pipeline** package is a project for creating analysis pipelines out of bioperl-run modules (bioperl-pipeline in CVS).
BioPerl: Modules of Interest

• **Bio::AlignIO**
  Handler for AlignIO Formats
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::Analysis**
  An interface to any (local or remote) analysis tool
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::AnalysisParser**
  Generic analysis output parser interface
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::AnalysisResult**
  Interface for analysis result objects
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::Annotatable**
  the base interface an annotatable object must implement
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::AnnotationCollection**
  Interface for annotation collections
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::Annotation**
  Annotation interface
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::Biblio**
  A Bibliographic Query Service module
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::BioEntry**
  DESCRIPTION of Object
  bioperl-db-1.5.2_100 - 06 Dec 2006 - Sendu Bala

• **Bio::Das**
  Interface to Distributed Annotation System
  Bio-Das-1.03 - 15 Sep 2006 - Lincoln D. Stein

• **Bio::DasLite**
  DAS-style access to a feature database
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

• **Bio::DasLite**
  a compatibility wrapper around Bio::Das::Lite
  Bio-Das-Lite-1.49 - 16 Mar 2007 - Roger Pettett

• **Bio::FASTASequence**
  Perl extension for Bioinformatics. Parsing sequence informations.
  Bio-FASTASequence-0.03 - 15 Oct 2004 - Renee Baecker

• **Bio::FastaStream**
  Perl extension for Bioinformatics. Parsing sequence informations.
  Bio-FastaStream-0.03 - 20 Jun 2006 - Renee Baecker
BioPerl: Modules of Interest

- **Bio::Graphics**
  Generate GD images of Bio::Seq objects
  bioperl-1.5.2_102 - 14 Feb 2007 - Sendu Bala

- **Bio::Grep**
  Perl extension for searching in Fasta files
  Bio-Grep-v0.6.0 - 27 Mar 2007 - Markus Riester

- **Bio::OntologyIO**
  Parser factory for Ontology formats
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Perl**
  Functional access to BioPerl for people who don't know objects
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Phylo**
  Phylogenetic analysis using perl.
  Bio-Phylo-0.16 - 30 Mar 2007 - Rutger Vos

- **Bio::Seq**
  Sequence object, with features
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::SeqAnalysisParserI**
  Sequence analysis output parser interface
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Affymetrix::CEL**
  Bio-Affymetrix-0.5 - 21 Jul 2005 - David J Craigon

- **Bio::Affymetrix::CHP**
  Bio-Affymetrix-0.5 - 21 Jul 2005 - David J Craigon

- **Bio::Annotation::OntologyTerm**
  An ontology term adapted to AnnotationI
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Biblio::PubmedArticle**
  Representation of a PUBMED article
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Restriction::Analysis**
  cutting sequences with restriction enzymes
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Restriction::Enzyme**
  A single restriction endonuclease (cuts DNA at specific locations)
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

- **Bio::Restriction::EnzymeCollection**
  Set of restriction endonucleases
  bioperl-1.4 - 23 Dec 2003 - Ewan Birney

**Bio::Affymetrix::CDF**
Bio-Affymetrix-0.5 - 21 Jul 2005 - David J Craigon
BioPerl success: Ensembl
Explore the *Homo sapiens* genome

Karyotype

Click on a chromosome for a closer view

About the Human genome

Assembly

This release is based on the NCBI 36 assembly of the [human genome](http://www.ncbi.nlm.nih.gov/) [November 2005]. The data consists of a reference assembly of the complete genome plus the Celera WGS and a number of alternative assemblies of individual haplotypic chromosomes or regions.

[Full list of assemblies](http://www.ensembl.org/info/data/assembly/index.html)

The International Human Genome Sequencing Consortium have published their scientific analysis of the finished human genome.

- [Nature 431, 931 - 945 (21 October 2004)](http://www.nature.com/nature/journal/v431/n7007/full/431931a0.html)
- [WT Sanger Institute Press Release](http://www.sanger.ac.uk/about/news/2005/10/human_genome_release)

Annotation
Ensembl at the UofMN

Each horizontal ‘track’ represents information requested from a remote host and displayed on demand.

The system that allows this level of collaboration is called DAS.
BioPerl: Be aware

• It is a work in progress.

• Until fairly recently, there was no document that provided a tutorial introduction to the project.

• Bioperl is big (over 1200 modules), written by volunteers, and evolving. Daunting, even to the initiated.

• Many of the newest modules are written by people with little or no previous programming experience.

• Much of the code it quite good. The hairy fringes can get pretty hairy.
#!/usr/bin/perl -w
#
# Simple driver for Bio::Seq revcom and translate.
#
use Bio::Seq;
use Bio::SeqIO;

# new sequence from raw memory...
# it is *very* important to get the type right so it
# is translated correctly.
$seq = Bio::Seq->new ( -id => "myseq",
   -seq => "CGCCGAAGAAGCATCGTTAAAGTCTCTCTTGCCGTCATGTCTAAGTCAGAGTCTCCT",
   -type => 'DNA');

$seqout = Bio::SeqIO->new('-format' => 'fasta', -fh => \*STDOUT);

# make a reverse complement sequence
$rev = $seq->revcom();

# the actual sequence is here
$actual_bases = $rev->seq();
print "Reversed sequence as a string is [$actual_bases]\n";

# write it as fasta formatted output
$seqout->write_seq($rev);
exit;
LAB: Bio::Seq, translate

#!/usr/bin/perl -w
#
# Simple driver for Bio::Seq revcom and translate.
#
use Bio::Seq;
use Bio::SeqIO;

# new sequence from raw memory...
# it is *very* important to get the type right so it
# is translated correctly.

$seq = Bio::Seq->new ( -id => "myseq",
    -seq => "CGCCGAAGAAGCATCGTTAAAGTCTCTCTTGCCGTCATGTCTAAGTCAGAGTCTCCT",
    -type => 'DNA');

$seqout = Bio::SeqIO->new('format' => 'fasta', -fh => \*STDOUT);

# make a translation
$trans = $seq->translate();
print "Translated sequence!\n";

$seqout->write_seq($trans);

exit;
LAB: Bio::Seq, revcom and translate

#!/usr/bin/perl -w
#
# Simple driver for Bio::Seq revcom and translate.
#
use Bio::Seq;
use Bio::SeqIO;

# new sequence from raw memory...
# it is *very* important to get the type right so it
# is translated correctly.
$seq = Bio::Seq->new (-id => "myseq",
                    -seq => "CGCCGAAGAAGCATCGTTAAAGTCTCTCTTGCCGTCATGTCTAAGTCAGAGTCTCCT",
                    -type => 'DNA');

$seqout = Bio::SeqIO->new('-format' => 'fasta', -fh => \*STDOUT);

# make a reverse complement sequence
$rev = $seq->revcom();

# the actual sequence is here
$actual_bases = $rev->seq();
print "Reversed sequence as a string is [$actual_bases]\n";

# write it as fasta formatted output
$seqout->write_seq($rev);

# make a translation
$trans = $seq->translate();
print "Translated sequence!\n";
$seqout->write_seq($trans);
exit;
LAB: break it.

1. Add ‘use strict’?

2. Was the error message helpful?

3. How would you protect users from an error like this?
# Simple driver for Bio::Seq revcom and translate.

use strict;
use Bio::Seq;
use Bio::SeqIO;

# new sequence from raw memory...
# it is *very* important to get the type right so it
# is translated correctly.
my $seq = Bio::Seq->new ( -id => "myseq",
    -seq => "CGCCGAAGAAGCATCGTTAAAGTCTCTCTTCACCCTGCCGTCATGTCTAAGTCAGAGTCTCCT",
    -type => 'DNA');

my $seqout = Bio::SeqIO->new('-format' => 'fasta', -fh => \*STDOUT);

# make a reverse complement sequence
my $rev = $seq->revcom();

# the actual sequence is here
my $actual_bases = $rev->seq();
print "Reversed sequence as a string is $actual_bases\n";

# we could also write it as fasta formatted output
$seqout->write_seq($rev);

# make a translation
my $trans = $seq->translate();
print "Translated sequence!\n";

$seqout->write_seq($trans);
exit;
LAB: add to it.

Can you?:

1. Find another way to collect sequence strings?

2. Can you change the filehandle for seqout?

3. Add a subroutine that provides sequence strings from an array of GenBank accession numbers?
A brief break
The Future and Bioinformatics

Perl plays a leading role.

http://www.ks.uiuc.edu/Research/pro_DNA/dna_w_analog/dna-wat.big.gif
The Coming Singularity in Bioinformatics

(with apologies to Ray Kurzweil)
The Singularity is Near: When Humans Transcend Biology

- Hardcover: 672 pages
- Publisher: Viking Adult (September 22, 2005)
- Language: English
- ISBN: 0670033847
- Product Dimensions: 9.8 x 6.4 x 2.0 inches
- Shipping Weight: 2.1 pounds
- Average Customer Review: ****/***** based on 62 reviews.
Wherein he states:

- Humankind is at the threshold of an epoch.
- The rate of exponential growth is itself growing exponentially.
- ‘Singularity’ is described as the point at which a function takes an infinite value.
  - in space-time, when matter becomes infinitely dense.
  - as in a black hole.
- The normal Laws of Physics no-longer apply.

Giant Black Hole Rips Star Apart
Credit: Illustration: NASA/CXC/M.Weiss;
X-ray: NASA/CXC/MPE/S.Komossa et al.;
Optical: ESO/MPE/S.Komossa
His point is two-fold

• There are no constraints on human capacity, especially as we begin to build machines that learn from each other.

• This is a good thing.

An artist depiction of two black holes merging. (http://en.wikipedia.org/wiki/Black_hole)
Genomics and the Web

• Genomics and the Web grew up together.

• Research is inherently distributed.

• Publishing on the web is inexpensive.

• Web browsers have become an essential tool for experimental biologists.

• But, sharing data relies ‘screen-scraping’ techniques.
Bioinformatics Applications are difficult to automate, because:

- Platforms are heterogeneous.
- Web interfaces are not machine-friendly.
- Non-standard formats for data input and output.
- Little use of standards to define interface and message exchange.
- Protocols are not firewall friendly.
Genomic Data

- Often unstructured or semi-structured.

- Data appears as text strings:
  - DNA or Protein sequences: FASTA flat-files, et alia
  - Annotation: usually free-text

- Feudal states (Stein)
Mere access to information is insufficient.

- We need **new tools** to explore and combine data in meaningful ways.

- The **Semantic Web** is an idea to provide data in a **machine-readable** format, over the web, so computers can aggregate information and make inferences about **relationships** between certain types of data.
Overview of Web Service Technology

- **WSDL**: Web Service Description Language is the XML language that describes a web service. Defining the interface and the exchange of messages between the provider and requester.

- **SOAP**: Simple Object Access Protocol is an XML-based protocol for stateless message exchange. It is not bound to any transport protocol, but is generally built on top of HTTP, which makes it firewall friendly.

- **UDDI**: Universal Description, Discovery and Integration is a standard protocol designed to publish details about an organization and the web services. It provides a description and definition of web services in a central repository, which functions as a Yellow Pages for web services.
MOBY

• An early effort to facilitate integration of web-based bioinformatics resources.

• Yes, it is an acronym: Model Organism Bring Your own - Database Interface Conference.

• Two projects:
  
  • MOBY-S: web services oriented.

  • S-MOBY: semantic web oriented.
MOBY-Services

• **SOAP**-based web services approach.

• Services are registered centrally.

• Assumes service provider does not **want** to restructure data to make services interoperable.

• Doesn’t try to solve the hard problems.

• Cheap and easy.
#!/usr/local/bin/perl
use SOAP::Lite;
print STDERR "Welcome to the SOAP demonstration\n";
my $res;
$servername = "inquiry.ccgb.umn.edu";
my $server = SOAP::Lite->
    ->uri("http://$servername/Backbeat")
    ->proxy("http://$servername/cgi-bin/bipod/BIFX.pl");
$res = $server->get_ticket(SOAP::Data->name("USER")="kunau", SOAP::Data->name("PASSWORD")="password_here");
my $ticket;
if ($res->result()) {
    print STDERR "Got ticket $ticket\n";
}
my $id = "nt:ABY13260";
$ticket = $res->result();
print STDERR "Submitted BLAST for $id\n";
my $jid = 0;
if ($res->result()) {
    $jd = $res->result();
} else {
    $id = "nt:ABY13260";
}
print "Submitted BLAST for $id\n";
print STDERR "Got job id $jid\n";
# Client side block
my $result = "";
while ($result ne "FINISHED") {
    print "Checking status for job $jid\n";
    $res = $server->job_status(SOAP::Data->name("JOB_ID")=$jid);
    handle_fault($res);
    if ($res->result()) {
        $result = $res->result();
        print "Got status $result\n";
        if ($result ne "FINISHED") {
            sleep 3;
        }
    }
}
print STDERR "fetched file for $id\n";
$res = $server->get_file(SOAP::Data->name("TICKET")=$ticket, SOAP::Data->name("FILENAME")="blastall.txt");
handle_fault($res);
if ($res->result()) {
    print $res->result();
}

# Subroutines
sub {
    my $res = shift;
    if (my $fault = $res->fault()) {
        my %fault = %$fault;
        while (my ($key, $val) = each (%fault)) {
            print "$key
$val\n";
        }
    }
}

Login
Get a ticket
Configure a service
Submit request
Check status (rinse, repeat)
Print result
Two principle problems:

• **distribution**: data is created and controlled by autonomous groups all over the world.

• **biology is hard and messy**: large collections of data, many numbers of data types and tools; few of which talk to each other.
The MOBY-Services problem

Chickens go in; Pies come out!
What sort of pies?
Tower of Babel, Pieter Brueghel the Elder, 1563.
Semantic-MOBY

• **OWL-DL (Description Logic)** based RDF graphs describe resources, ensure semantic consistency.

• Ontology management is distributed.

• Requires service discovery aggregation ‘servers’, **not very P2P**.

• OWL DL graphs used for machine-readable query, discovery, and engagement.
Semantic-MOBY

- Dependent on strictly negotiated and curated ontology domains.

- The right thing to do.

- Harder to do.

- Technology is not the hard part.
OWL-S

• Web Ontology Language for Services

• Approved by W3C in Feb. 2004.

• “intended to be used when information contained in documents needs to be processed by applications”

• http://www.w3.org/TR/owl-ref/
<table>
<thead>
<tr>
<th>Service Interfaces</th>
<th>MOBY-S</th>
<th>S-MOBY</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL, single operations, inputs/outputs</td>
<td>HTTP/FTP</td>
<td></td>
</tr>
<tr>
<td>Semantic Descriptions</td>
<td>API, enforced by registry</td>
<td>OWL-DL</td>
</tr>
<tr>
<td>Domain Ontology</td>
<td>community curated, GO style DAG</td>
<td>OWL-DL, distributed across the web</td>
</tr>
<tr>
<td>Registry / Matchmaker</td>
<td>centralized registry</td>
<td>one or more centralized matchmakers</td>
</tr>
<tr>
<td>Messaging</td>
<td>XML wrapper on legacy formats</td>
<td>OWL-DL</td>
</tr>
</tbody>
</table>
Taverna

- Provides a level of abstraction above web service interface technologies.
- The user adds the URL to the WSDL document describing a service.
- The user then creates graphical links between input documents and services.
- RPC
- Open Source project.

Figure 3: Scufl Workbench Screenshot of Taverna implementation of RPC based workflow retrieving EMBL information from DDBJ.
InforSense

- interactive environment for designing and executing scalable, bioinformatics solutions ranging from sequence analysis to microarray informatics and remote database annotation.
- Oracle database supported.
- MSI’s tool.
- Commercial product.
Lessons from Web Services projects

• Web Services are in their infancy.

• There is an increasing need to use metadata and negotiated ontologies to facilitate these services.

• Use of choreography engines is in the early stage.

• Taverna: offers an advanced GUI but support for document-oriented services is still limited. It’s free.

• InforSense: offers an advanced GUI, high levels of standards support. Only available under commercial license.

• Most simple work-flows are possible in hard-coded implementations.

• Very active area of study.
How does Singularity apply to Bioinformatics?

- Web interfaces, currently human-friendly, will become machine-friendly.

- Data formats and interfaces will begin to standardize.

- Heterogeneous platforms, applications, and systems will begin to interoperate.

- Machines will begin to communicate with each other in profound and powerful ways.
The Singularity is Near.
Resources

- **Perl**: http://www.perl.com/
- **CPAN**: http://www.cpan.org/
- **BioPerl**: http://www.bioperl.org/
  - http://www.bioperl.org/Core/Latest/bptutorial.html
  - http://bioperl.org/Core/Latest/modules.html
- **Command-line**: `man`, `perldoc`, and `perldoc perlvar`
- **Web Service Standards**: http://www.w3.org
Beginning Perl for Bioinformatics

- Hardcover: 400 pages
- Publisher: O'Reilly Media, Inc.; 1 edition (October 15, 2001)
- Language: English
- ISBN: 0596000804
- Product Dimensions: 9.2 x 7.1 x 0.9 inches
- Shipping Weight: 1.3 pounds.
- Average Customer Review: 4.5/5 based on 25 reviews.
Mastering Perl for Bioinformatics

- Hardcover: 377 pages
- Publisher: O'Reilly Media, Inc.; 1 edition (June, 2003)
- Language: English
- ISBN: 0596003072
- Product Dimensions: 9.4 x 6.8 x 0.9 inches
- Shipping Weight: 1.4 pounds.
- Average Customer Review: 4.5/5 based on 8 reviews.
<table>
<thead>
<tr>
<th>Title</th>
<th>Content Type</th>
<th>Publisher</th>
<th>Insert Date</th>
<th>Pub. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learning Perl, 4th Edition</strong></td>
<td>Book</td>
<td>O'Reilly</td>
<td>2005/08/10</td>
<td>2005/07/01</td>
</tr>
<tr>
<td>By brian d foy, Tom Phoenix,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Randal L. Schwartz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slots: <strong>1.0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table of Contents</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... Learning Perl, better known as "the Llama book", starts the programmer on the way to mastery. Written by three prominent members of the Perl community who each have several years of experience teaching Perl around the world, this latest edition has been updated to account for all the recent changes to the language up to Perl 5.8. Perl is the language for people who want to get work done. It started as a tool for UNIX system administrators who needed something powerful for small tasks. Since then, Perl has blossomed... [Read More >]

<table>
<thead>
<tr>
<th>Title</th>
<th>Content Type</th>
<th>Publisher</th>
<th>Insert Date</th>
<th>Pub. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perl Cookbook, 2nd Edition</strong></td>
<td>Book</td>
<td>O'Reilly</td>
<td>2003/08/29</td>
<td>2003/08/01</td>
</tr>
<tr>
<td>By Tom Christiansen, Nathan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torkington</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slots: <strong>1.0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table of Contents</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... Find a Perl programmer, and you’ll find a copy of Perl Cookbook nearby. Perl Cookbook is a comprehensive collection of problems, solutions, and practical examples for anyone programming in Perl. The book contains hundreds of rigorously reviewed Perl “recipes” and thousands of examples ranging from brief one-liners to complete applications. The second edition of Perl Cookbook has been fully updated for Perl 5.8, with extensive changes for Unicode support, I/O layers, mod_perl... [Read More >]

<table>
<thead>
<tr>
<th>Title</th>
<th>Content Type</th>
<th>Publisher</th>
<th>Insert Date</th>
<th>Pub. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming Perl, 3rd Edition</strong></td>
<td>Book</td>
<td>O'Reilly</td>
<td>2002/09/26</td>
<td>2000/07/01</td>
</tr>
<tr>
<td>By Tom Christiansen, Jon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orwant, Larry Wall</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slots: <strong>1.0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table of Contents</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... Perl is a powerful programming language that has grown in popularity since it first appeared in 1988. The first edition of this book, Programming Perl, hit the shelves in 1990, and was quickly adopted as the undisputed bible of the language. Since then, Perl has grown with the times, and so has this book. Programming Perl is not just a book about Perl. It is also a unique introduction to the language and its culture, as one might expect only from its authors. Larry Wall is the inventor of Perl, and provides a unique perspective on the evolution of Perl and its future direction...
## Bioinformatics

Found 9 book(s) matching the category "Bioinformatics".

<table>
<thead>
<tr>
<th>Title</th>
<th>Content Type</th>
<th>Publisher</th>
<th>Insert Date</th>
<th>Pub. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. BLAST</td>
<td>Book</td>
<td>O'Reilly</td>
<td>2003/08/12</td>
<td>2003/07/01</td>
</tr>
<tr>
<td>By Joseph Bedell, Ian Korf, Mark Yandell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slots: 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table of Contents</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>... BLAST (Basic Local Alignment Search Tool) is a set of similarity search programs that explore all of the available sequence databases for protein or DNA. BLAST is the only book completely devoted to this popular and important technology and offers biologists, computational biology students, and bioinformatics professionals a clear understanding of this program. This book shows you how to get specific answers with BLAST and how to use the software to interpret results. If you have an interest in sequence analysis this is a book you should own... <a href="#">Read More &gt;</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 2. Mastering Perl for Bioinformatics       | Book         | O'Reilly  | 2003/10/09  | 2003/09/01|
| By James Tisdall                          |              |           |             |           |
| Slots: 1.0                                 |              |           |             |           |
| Table of Contents                          |              |           |             |           |
| ... Mastering Perl for Bioinformatics covers the core Perl language and many of its module extensions, presenting them in the context of biological data and problems of pressing interest to the biological community. This book, along with Beginning Perl for Bioinformatics, forms a basic course in Perl programming. This second volume finishes the basic Perl tutorial material (references, complex data structures, object-oriented programming, use of modules—all presented in a biological context) and presents some advanced topics... [Read More >](#) |

| 3. Sequence Analysis in a Nutshell         | Book         | O'Reilly  | 2003/04/19  | 2003/01/01|
| By Darryl León, Scott Markel              |              |           |             |           |
| Slots: 1.0                                 |              |           |             |           |
| Table of Contents                          |              |           |             |           |
| ... Sequence Analysis in a Nutshell: A Guide to Common Tools and Databases pulls together all of the vital information about the most commonly used databases, analytical tools, and tables used in sequence analysis. The book contains details and examples of the common database formats (GenBank, EMBL, SWISS-PROT) and the GenBank/EMBL/DDJB Feature Table Definitions. It also provides the command line syntax for popular analysis applications such as Readseq [Read More >](#) |
Collaboration and Communication

- Attend conferences. Consider:
  - ICMB/BOSC
  - OSCON
- Deliver papers at conferences.
- Join journal clubs or start a journal club!
- Participate, you are part of the community.
If you remember nothing else

- Biology is hard and messy.
- The key problems are social. Together we are smarter than any one of us.
- Technology is easy by comparison.
Questions?
Thanks to:

- Allison Langham
- Dan Bolintineanu
- John Crow, Ph.D.
- Brian Hanson
- BSI 2007
- You
Thank You.

Timothy M. Kunau, kunau@umn.edu