Perl for Bioinformatics
Perl and BioPerl I

Jason Stajich

July 13, 2011
Outline

Perl Intro

Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl

Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
Outline

Perl Intro

Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
Why Perl for data processing & bioinformatics

- Fast text processing
- Regular expressions
- Extensive module libraries for pre-written tools
- Large number of users in community of bioinformatics
- Scripts are often faster to write than full compiled programs

Cons  
Syntax sometimes confusing to new users; ’There’s more than one way to do it’ can also be confusing. (TMTOWTDI)

Cons  
Not a true object-oriented language so some abstraction is clunky and hacky

Scripting languages (Perl, Python, Ruby) generally easier to write simply than compiled ones (C, C++, Java) as they are often not strongly typed and less memory management control.
Perl packages

- CPAN - Comprehensive Perl Archive

http://www.cpan.org
Help!

- Perldoc online http://perldoc.perl.org/
- Or on your computer - type ’perldoc’
- For functions use -f ’perldoc -f sprintf’
- For modules just the name ’perldoc List::Util’
#! /usr/bin/perl -w
use strict;
print "hello world\n";

>> perl hello.pl
>> hello world
Variables & Syntax

$scalar — single value, can be string, number
@array — list of values
%hash — paired values: key and value

# comments look like this
my $variable; # a var declared with my for this scope
my ($n1,$n2) = (10,20); # declared and initialized
Strings

my $number = "12";
my $msg = "This could be a message";
my $composite = "There are $number apples";
print $composite, "\n";

>> There are 12 apples
my $str = 'literally';
my $str2 = "interpreted as $str";
my $executed = ‘/usr/bin/clustalw seqs.fa’;

# special characters
my $tab = "\t";
my $newline = "\n";
my $singlequote = "’";
my $dblquote = "\""; # OR ‘’ ;
my $n = 16;
my $g = 3**2;  # 3^2
my $d = 12.34;
my $ir = 1/3;
my $h = 1e-3;
print $ir, "\n";
print $h, "\n";
printf "%e\n", $h;  # print in scientific notation

>>  0.333333333333333
>>  0.001
>>  1.000000e−03
#!/usr/bin/perl
use strict;

print "a message\n";
my $v1 = 'CFTR';
my $v2 = '20.34';
printf "a LOD score for marker %s is %d\n", $v1, $v2;

?>> a LOD score for the marker CFTR is 20
if ( $A < $B ) { } # if this
elsif ( $B < $C ) { } # otherwise if this
else { } # do this as last resort

unless ( $bool_not ) { } # do something if $bool_not is false

while ( $bool ) { } # loop if $bool is true
until ( $bool_not ) { } # loop if $bool_not is false
for( INIT ; TEST ; INCREMENT ) { }
for( $i = 0 ; $i < 5 ; $i++ ) { print $i, "\n"; }

>>> 1
>>> 2
>>> 3
...

Perl for Bioinformatics
Perl Intro
Syntax and Variables
Slide 14/83
if ( $a == $b )  # a is numerically equivalent to b
if ( $a eq $b )  # a is lexically equivalent to b
if ( $a < $b )   # a is less than b
if ( $a != $b )  # a is not equal to b (numerically)
if ( $a ne $b )  # a is not lexically equal (string comparison)
if ( @list )    # true if @list is not empty
if ( $a )       # true if $a is not 0 and not undefined
my @fruit = ('apple', 'pear', 'peach');
my @tropical = qw(kiwi passion star); # qw for quote words
my @all = (@fruit, @tropical); # lists are flattened
my @mixed = (1, 'pineapple', 18.3); # can be mixed types
my @sorted = sort @mixed; # alpha numeric sort
my @ordered = sort { $a <=> $b } (10, 3, 17, 200, 9);
print $ordered[0], "\n"; # print the 1st item
print $ordered[-1], "\n"; # print the last item

>> 3
>> 200

Lists and strings start counting at '0' not '1'
Arrays

To add or remove entries from lists: can operate as linked-lists and stacks

- pop to remove from end
- push to add to end
- shift remove from front unshift add to front
- splice to arbitrarily remove from any position

my @tools = qw(rake shovel);
push @tools, 'hammer'; # now there will be 3 items
my $first = shift @tools; # will be 'rake'
splice (@tools, 1, 0, 'saw'); # insert 'saw' after 'hammer'
print join (",", @tools), "\n";

>>> rake, saw, shovel, hammer
my @lst = qw(In the locust wind comes a rattle and hum);
print join("","", @lst), "\n"; # combine list to a string
my @newlist = split(/\s+/, "GENE1 GENE2 GENE3 VALUE1");
print $newlist[2], "\n"; # split string, get 3rd item

>>> In, the, locust, wind, comes, a, rattle, and, hum
>>> GENE3
my %fruit = ('apple' => 'red'); # initialize with value
$fruit {'banana'} = 'yellow'; # add a value
$fruit {'apple'} = 'green'; # update a value
my @keys = keys %fruit; # get keys of hash
my @vals = values %fruit; # get the values
print join ("\n", @keys), "\n";
print join ("\n", @vals), "\n";

>> banana, apple
>> yellow, green
Manipulate strings

- `substr` - get a substring and also manipulate in place
- `length` - get length of a string
- `.` - concatenate two strings

```perl
my $left = 'ABC';
my $right = 'XYZ';
my $concat = $left . $right;
print length($concat), " is length of string $concat\n";
print "pos 3-4 is ", substr($concat, 2, 2), "\n";
my $lastchar = substr($concat, -1, 1);
substr($concat, 1, 2, ''); # replace 2nd and 3rd characters
print "concat is $concat\n";
```

```
>> 6 is the length of string ABCXYZ
>> pos 2-4 is CX
>> concat is AXYZ
```
Manipulate strings 2

How about walking through each base in a sequence? Could use split and turn it into an array and use a for loop OR Can use substr to request each character in the string, one at a time. Turns out this is faster.

```perl
my $sequence = 'ACGGTAGCATA';
for ( my $i = 0; $i < length $sequence; $i++) {
    my $base = substr($sequence, $i, 1); # get the i-th base
    print $base, "\n";
}

>>> A
>>> C
>>> G
>>> G
>>> T
...
Outline

Perl Intro

Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl

Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
my %info = ( 'gene123' => 1.02);
my $ref = \%info; # backslash to make reference
print $ref->{'gene123'}, "\n"; # arrow to dereference
my $ref2 = { 'gene1' => 2 }; # {} for anonymous hash reference
print $ref2->{'gene1'}, "\n";
print join(",", keys \%{$ref2}), "\n"; # \%{} to deref a hash
my $refarray = ['gene1','gene2']; # [] for anonymous array reference
print join(";", @{$refarray}), "\n"; # @{} to dereference array

>> 1.02
>> gene1
>> gene1;gene2
If you wanted to store more than 1 thing per key in a hash:

```perl
my %kitchen = ( 'ingredients' => [ qw(flour eggs milk) ] );
print join(",", @{ $kitchen{'ingredients'} }), "\n";
$kitchen{'chefs'} = [ qw( Anthony Emeril ) ];
push @{ $kitchen{'chefs'} }, 'Julia';
for my $info ( keys %kitchen ) {
    print "$info: ", join("\t", @{ $kitchen{$info} }), "\n";
}
```

>> flour, eggs, milk
>> chefs: Anthony Emeril Julia
>> ingredients: flour eggs milk
my @library = (
    { 'title' => 'To Kill a Mockingbird',
      'author_last' => 'Lee',
      'author_first' => 'Harper' },
    { 'title' => 'Moby Dick',
      'author_last' => 'Mellville',
      'author_first' => 'Herman' },
    { 'title' => 'Old Man and the sea',
      'author_last' => 'Hemmingway',
      'author_first' => 'Ernest' }
);

for my $book (@library)
    { printf "%s,%s %s\n", $book->{author_last}, $book->{author_first}, $book->{title};
}

>> Hemmingway, Ernest Old Man and the sea
>> Lee, Harper To Kill a Mockingbird
>> Mellville, Herman Moby Dick
Array or Arrays (AoA) - great for a matrix

```perl
my @matrix = ();
$matrix[0] => [0] = 1;
$matrix[0] => [1] = 2;
$matrix[0] => [2] = 3;
$matrix[1] = [ 4, 5, 6];
$matrix[2] = [ 7, 8, 9];
for my $row (@matrix) {
    print join(" ", @$row), 

};

>> 1 2 3
>> 4 5 6
>> 7 8 9
```
my %geneset;
$geneset{‘YFG111’} = { ‘name’ => ‘YFG11’,
   ‘aliases’ => qw( IFU1 GEO887 ),
   ‘chrom’  => ‘chrom11’,
   ‘start’  => ‘1002131’,
   ‘end’    => ‘1003075’,
   ‘strand’ => ‘+’ }
my $ref = $geneset{‘YFG11’};
$ref->{length} = $ref->{end} - $ref->{start} + 1;
Hash of Hashes (HoH)

```perl
# DATA FILE 1
# GENE    SCORE
# YFG123  0.1
# DATA FILE 2
# GENE    LENGTH
# YFG123  200
my %data;
while(<$fh1>) {
    next if /^\#/;
    my ($gene, $score) = split;
    $data{$gene}->{$score} = $score;
}

while(<$fh2>) {
    next if /^\#/;
    my ($gene, $length) = split;
    $data{$gene}->{$length} = $length;
}
for my $gene ( keys %data ) {
    print join( "\t", $gene, $data{$gene}->{$score},
                $data{$gene}->{$length} ), "\n";
}
```
Outline

Perl Intro

Basics
Syntax and Variables
More complex: References

Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
Subroutines are reusable set of code that can be called as a unit

```perl
sub add_up {
    my @arguments = @_;    
    print "argument #1 is $arguments[0]\n";
    return $arguments[0] + $arguments[1];
}
print "sum of 10 and 23 is ", add_up(10,23), "\n";
```

> argument1 is 10
> sum of 10 and 23 is 33
Subroutines take list as arguments. If you want to pass in an array and NOT have to flattened, you have to pass in as a reference.

```perl
my ($A, @B) = (20, 40, 50);
my @C = (65, 21);
doThis($A, @B, @C);  # sometimes you'll see &doThis(...)

sub doThis {
  my ($in_A, $in_B, @in_C) = @_;  
  print "A= $in_A\n";
  print "B= " . join(",", @in_B) . "\n";
  print "C= " . join(",", @in_C) . "\n";
}

$A = 20
$B = 40,50
$C = 65,21
```
What is scope (why do we use my again?)

Scope defines the context where a variable is valid for. Re-declaring a variable will cause a warning. Undeclared variables will cause compile time error.

```perl
use strict;
use warnings;
my $score = 5;
my $score = 10; # Last declaration wins
if ( $score == 10 ) {
    my $score = 20;
    print "In if score=$score\n";
}
print "At end score=$score\n";
```

"my" variable $score masks earlier declaration in same scope at test_scope.pl line 4.

>> In if score=20
>> At end score=10
Outline

Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules

Perl for Bioinformatics
Filehandles for reading

```perl
open(IN, "results.dat") || die $!; # open for reading
open(IN2,"<results2.dat") || die $!; # equivalent
open(my $fh => "<results3.dat") || die $!; # equivalent
while(<IN>) {
    my $line = $_;
}
```
Filehandles for writing

```perl
open(OUT, ">output.out") || die $!; # open for writing
open($fh => ">output2.out") || die $!;

print OUT join "\t", qw(onion 2.03),"\n";
print $fh join "\t", qw(garlic 0.78),"\n";
```
Implicit variable 

while(<FH>) {
  # $_[is updated with each line
  my ($col1,$col2) = split; # split works on $_[}

Tab (or comman, or space, or any other) delimited columns are easy to parse in Perl.
Here is a simple script to parse a BLAST tabular output (outfmt 7 for BLAST+ or -mformat 8 or 9 from blastall)

```perl
open($fh => $filename) || die $!
while(<$fh1>) {
    next if /^\#/;
    my @row = split;
    my ($query, $subject, $percent_id, $aln_len, $mismatches, $gaps,
        $qstart, $qend, $hitstart, $hitend, $evalue, $bitscore) = @row;
}
```
When things go wrong...

How to read the error messages and debug your code?

- Note the error line
- Try using 'print' to print out the variable's value
- You can see if your program will just compile with 'perl -w'
- always 'use strict'
- You can use the perl debugger (perl -d)
- Did you miss a ';;'
open(my $fh => "zcat seqs.fa.gz |") || die $!
open(my $fh2 => "zgrep '^>' seqs.fa.gz |") || die $!;  # get FASTA
while(<$fh>) {
    my $id = $_;
    print $id, "\n";
}
Also can write to a dynamic filehandle

```perl
open(my $fh => "| gzip -c > output.gz") || die $!
print $fh "Data1\n"; # etc
print $fh "Data2\n"; # etc
```

This will create a file called output.gz of compressed version of the output data.
Run a multiple alignment without writing a file

```perl
open(my $fh => "| muscle -in - -out $outfile.aln") || die $!
print $fh ">seq1\n", "ACTAA\n";
print $fh ">seq2\n", "ACATCA\n";
```

*Can you read and write to the same file handle?*

Only if you use IPC::Run3 module which gives you access to STDIN, STDOUT, and STDERR for an external program.
Special variable @ARGV are the command line options

my ($arg1, $arg2) = @ARGV;
# OR (TMTOWTDI)
# my $arg1 = shift @ARGV;
# my $arg2 = shift @ARGV;
print "start wearing $arg1!\n";

>> perl cmdline.pl purple
>> start wearing purple!
Modules make it easier to process command line arguments with options

```perl
use Getopt::Long; # I really like this module,
# also see Getopt::Std
my ($name, $rank, $serialnum, $active);
GetOptions(
    'n|name:s' => $name,
    'r|rank:s' => $rank,
    's|serial:s' => $serialnum,
    'a|active!' => $active);
my @other_args = @ARGV; # all of these options are consumed
```

```bash
>> perl cmdline2.pl -n "James Davis" -r Private -s 867100 --active
>> perl cmdline2.pl -n "Leonard Lawrence" -r Private -s 811220 --noactive
```
Outline

Perl Intro
- Basics
- Syntax and Variables
- More complex: References
- Routines
- Reading and Writing
- Regular Expressions

BioPerl
- Intro
- Useful modules
- Data formats
- Databases in BioPerl
- Sequence objects details
- Trees
- Multiple Alignments
- BLAST and Sequence Database searching
- Other general Perl modules
Regular expressions

- Very powerful feature of Perl built-in
- expressions go in / / (though this can be overridden)
  - match any of the symbols in there: [A-Z] all capital letters
- shortcuts: \d - all digits, \w - all alphanumeric characters + more, \s - white space,
- \D - not digits, \W - not a word character, \S - not whitespace

```
my $var = 'caterpillar';
if ( $var =~ /cat/ ) { # would be true }
if ( $var ~! /cat/ ) { # would be false (the string contains 'cat')
```

See the PerlRE page: http://perldoc.perl.org/perlre.html
my $var = 'GENE1:230-250';
if ( $var =~ /((\w+):(\d+)-(\d+))/ ) {
    my ($gene, $start, $end) = ($1, $2, $3);
    print "gene=$gene start=$start end=$end\n";
}
Outline
Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
Outline

Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
BioPerl

- Collection of Perl modules for life sciences data and analysis
- Modules are interfaces to data types: Sequences, Alignments, Features, Locations, Databases
- Example: Parser of sequence files, Alignment (BLAST) or Multiple alignment formats
- http://www.bioperl.org/ - Website has lots of information
- http://www.bioperl.org/wiki/HOWTOs - How To guides
Outline

Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
SeqIO: Read in a Fasta sequence file, count number of sequences

```perl
#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;
my $seqfile = "sequences.fa";
my $in = Bio::SeqIO->new(-format=>'fasta',
                          -file=>$seqfile);
my $count = 0;
while( my $seq = $in->next_seq ) {
    $count++;
}
print "There are $count sequences\n";
```
SeqIO: Read in a Fasta sequence file, count number of bases

```perl
#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;
my $seqfile = "sequences.fa";
my $in = Bio::SeqIO->new(-format=>'fasta',
    -file=> $seqfile);
my $count = 0;
while( my $seq = $in->next_seq ) {
    $count += $seq->length;
}
print "There are $count bases\n";
```
use Bio::SeqIO;
my $seqfile = "sequences.gbk";
my $in = Bio::SeqIO->new(
    -format => 'genbank',
    -file => $seqfile);
my $out = Bio::SeqIO->new(
    -format => 'fasta',
    -file => ">outputfile.fa")
while( my $seq = $in->next_seq ) {
    $out->write_seq($seq);
}
Outline

Perl Intro
- Basics
- Syntax and Variables
- More complex: References
- Routines
- Reading and Writing
- Regular Expressions

BioPerl
- Intro
- Useful modules

Data formats
- Databases in BioPerl
- Sequence objects details
- Trees
- Multiple Alignments
- BLAST and Sequence Database searching
- Other general Perl modules
### GFF flavors

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Description</th>
<th>ID/Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curator gene</td>
<td>chr1</td>
<td>200-300</td>
<td>ID=GENE001;Name=YFG</td>
</tr>
<tr>
<td>Curator mRNA</td>
<td>chr1</td>
<td>200-300</td>
<td>ID=mRNA001;Parent=GENE001;Name=YFG.T0</td>
</tr>
<tr>
<td>Curator CDS</td>
<td>chr1</td>
<td>200-300</td>
<td>ID=CDS001;Parent=mRNA001</td>
</tr>
<tr>
<td>RMasker repeat</td>
<td>chr1</td>
<td>400-480</td>
<td>ID=Repeat1;Name=LINE1</td>
</tr>
<tr>
<td>RMasker repeat</td>
<td>chr1</td>
<td>600-750</td>
<td>ID=Repeat2;Name=hAT</td>
</tr>
</tbody>
</table>

### GFF2

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Description</th>
<th>ID/Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMasker repeat</td>
<td>chr1</td>
<td>600-750</td>
<td>ID=Repeat2;Name=hAT</td>
</tr>
</tbody>
</table>

### GTF

<table>
<thead>
<tr>
<th>Gene</th>
<th>Transcript</th>
<th>Region</th>
<th>Description</th>
<th>ID/Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chr_5 CC3_FINAL</td>
<td>start_codon</td>
<td>871198-871200</td>
<td>0</td>
<td>gene_id &quot;CC1G_00004&quot;; transcript_id &quot;CC1G_00004T0&quot;;</td>
</tr>
</tbody>
</table>
Sequence file formats: Fasta

>gi|45552454|ref|NM_206028.1| Drosophila melanogaster Adh transcription factor 1 (Adf1), transcript variant 1

TAATTGGCAGAGACGCGACTGAGCTGGGACGTACCGTTACCGTTGGCAGAGACGCGACTGAGAAATAAAA
TTAAAACGTCGACGTTCCTTCTCGTAGAAGAAACCAATCAAAATAAAAACAAACAGAGCGTTCGC

Perl for Bioinformatics

BioPerl

Data formats
Sequence file formats: GenBank

LOCUS      NM_206028  1678 bp  mRNA  linear  INV 01-FEB-2011
DEFINITION Drosophila melanogaster Adh transcription factor 1 (Adf1),
            transcript variant C, mRNA.
ACCESSION NM_206028
VERSION   NM_206028.1 GI:45552454
KEYWORDS   .
SOURCE     Drosophila melanogaster (fruit fly)
ORGANISM   Drosophila melanogaster
            Eukaryota; Metazoa; Arthropoda; Hexapoda; Insecta; Pterygota;
            Neoptera; Endopterygota; Diptera; Brachycera; Muscomorpha;
            Ephydroidea; Drosophilidae; Drosophila; Sophophora.
REFERENCE  1 (bases 1 to 1678)
AUTHORS    Hoskins,R.A., Carlson,J.W., Kennedy,C., Acevedo,D., Evans-Holm,M.,
            Frise,E., Wan,K.H., Park,S., Mendez-Lago,M., Rossi,F.,
            Villasante,A., Dimitri,P., Karpen,G.H. and Celniker,S.E.
TITLE      Sequence finishing and mapping of Drosophila melanogaster
            heterochromatin
JOURNAL    Science 316 (5831), 1625-1628 (2007)
PUBMED     17569867
FEATURES
source
  1..1678
  /organism="Drosophila melanogaster"
  /mol_type="mRNA"
  /db_xref="taxon:7227"
  /chromosome="2R"
  /genotype="y[1]; cn[1] bw[1] sp[1]; Rh6[1]"
gene
  1..1678
  /gene="Adf1"
  /locus_tag="Dmel_CG15845"
  /gene_synonym="Adf 1; adf-1; Adf-1; adf1; CG15845; Dmel\CG15845; 1(2)01349; 1(2)04065; nal"
  /note="Adh transcription factor 1"
  /map="42C3-42C3"
  /db_xref="FLYBASE:FBgn0000054"
  /db_xref="GeneID:47082"
Bio::DB::Fasta - Fast random access to Fasta seq databases

CDS

180..968
/gene="Adf1"
/locus_tag="Dmel_CG15845"
/gene_synonym="Adf 1; adf-1; Adf-1; adf1; CG15845; Dmel\CG15845; 1(2)01349; 1(2)04065; nal"
/note="CG15845 gene product from transcript CG15845-RC; CG15845-PC; Adf1-PC; nalyot; naylot"
/codon_start=1
/product="Adh transcription factor 1, isoform C"
/protein_id="NP_995750.1"
/translation="MHTLTAAIEMDKLDANLEQQFDLNLIEAVKLNPVIYDRSHYNKYHFVRKAQTWKQIAETLGVPEKCTKRWKSLRDKFAREMKLQCESRWRYFKQMQLVDSIRQYRESLGLGCANGSQSANVADPSQQQAQQQTVVDIFAQPFNGSATTTSAQALTHPHEITVTDQQLATAVGDQKYFYEPPBRERSEEEEHDNMLNTIKIFQNNVSQAVSAEDQSFGMVVTDLNLTGVRQKAEEKVIYLTDMQLLAQHNKY"

ORIGIN

1 taattgccag agacgcgact gagctgggac gtaccgttac cgttgccaga gacgcgactg
61 agaaataaaa ttaaagcgtc gagcttcctt cctcgtagaa gaaaccaatc aaaataaaaa
121 caaacagagc gtgcgttcgc gccaaatact taacaacaat tagcaaacgt aagaagcaaa
181 tgcataccct cacggcggcc attgagatgg acaagctgga tgccaatctt gagcagcagt
241 ttgatctcaa tctcatcgag gctgtcaagc tgaacccagt gatatacgac aggtcgcact
301 acaattacaa gcactttgtg cgcaaggccc agacctggaa acaaatcgcc gaaacgctcg
361 gtgtgcctga acaaaaatgt acgaagcgct ggaagagtct gcgcgacaag ttcgcccgcg
Sequence objects

- Bio::SeqIO to read and write, creates and uses Bio::Seq objects
- Methods for getting info from a sequence object
  - seq() - sequence as a string
  - length() - how long is the sequence
  - id() - what is the ID for the sequence
  - description() - what is the ID for the sequence
GenBank and other rich formats have features. Features are elements that are located on a Sequence

- `get_SeqFeatures()` will return the sequence features
- `Bio::SeqFeature::Generic` objects
- `start`, `stop`, `strand`, `length` are all feature object methods for position info
- `primary_tag`, `source_tag` - the tags for the (source, type)
Outline

Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
use Bio::DB::Fasta

# $dir can be a directory of *.fa|fasta) files or
# a single dir or file to index
my $dir = shift @ARGV;
my $dbh = Bio::DB::Fasta->new($dir);
my $seq = $dbh->get_Seq_by_acc("SEQ123");
# extract a substring from db
my $seqstr = $dbh->seq("chr1", 10012, 13454);
use Bio::DB::GenBank;
use Bio::SeqIO;
my $db = Bio::DB::GenBank->new;
my $seq = $db->get_Seq_by_acc("NM_206028.1");
my $out = Bio::SeqIO->new(format => 'fasta');
$out->write_seq($seq);
use Bio::DB::GenBank;
use Bio::SeqIO;
my $db = Bio::DB::GenBank->new;
my $seq = $db->get_Seq_by_acc("NM_206028.1");
my $out = Bio::SeqIO->new( format => 'genbank' );
$out->write_seq($seq);
use Bio::DB::GenBank;
use Bio::DB::Query::GenBank;
use Bio::SeqIO;
my $db = Bio::DB::GenBank->new;
my $query = Bio::DB::Query::GenBank->new(
    -db => 'nucleotide',
    -query => 'Zea mays[Organism] and mRNA',
    -mindate => 2010,
    -maxdate => 2010);
print "there are ", $query->count," records\n";
my $stream = $db->get_Stream_by_query($query);
my $out = Bio::SeqIO->new(-format => 'genbank');
while (my $seq = $stream->next_seq) {
    $out->write_seq($seq);
    last;
}
use Bio::DB::SeqFeature::Store;
my $db = Bio::DB::SeqFeature::Store->new(-dir => 'demo',
                                          -adaptor=>'berkeleydb');
my @genes = $db->features(-type => 'gene');
print "querying genes\n";
for my $g (@genes) {
    print $g->name, " ", $g->location->to_FTstring, "\n";
}
print "querying repeats\n";
my $iterator = $db->get_seq_stream(-type => 'repeat');
while( my $feature = $iterator->next_seq ) {
    print $feature->name, " ", $feature->length, "\n";
}
#input

chr1  Curator gene  200  300  .  +  .  ID=GENE001;Name=YFG
chr1  Curator mRNA  200  300  .  +  .  ID=mRNA001;Parent=GENE001;Name=YFG.T0
chr1  Curator CDS  200  300  .  +  .  ID=CDS001;Parent=mRNA001
chr1  RMasker repeat  400  480  .  +  .  ID=Repeat1;Name=LINE1
chr1  RMasker repeat  600  750  .  +  .  ID=Repeat2;Name=hAT

#output

>> querying genes
>> YFG chr1:200..300
>> querying repeats
>> LINE1 81
>> hAT 151
Outline

Perl Intro
- Basics
- Syntax and Variables
- More complex: References
- Routines
- Reading and Writing
- Regular Expressions

BioPerl
- Intro
- Useful modules
- Data formats
- Databases in BioPerl

**Sequence objects details**
- Trees
- Multiple Alignments
- BLAST and Sequence Database searching
- Other general Perl modules
my $seqfile = "sequences.gbk";
my $in = Bio::SeqIO->new(-format => 'genbank',
                          -file => $seqfile);
my $out = Bio::SeqIO->new(-format => 'fasta',
                          -file => '>pepseqs.fa');
while (my $seq = $in->next_seq) {
    for my $feat ( $seq->get_SeqFeatures ) {
        if ( $feat->primary_tag eq 'CDS') {
            # get protein_id name
            warn("all tags are ", join("","",$feat->get_all_tags),"\n");
            if ( $feat->has_tag('protein_id') ) {
                my ($protein_id) = $feat->get_tag_values('protein_id');
                my ($pseq) = $feat->get_tag_values('translation');
                my $pepseq = Bio::Seq->new(-id => $protein_id,
                                               -description => $seq->accession_number,
                                               -seq => $pseq);
                $out->write_seq($pepseq);
            }
        }
    }
}
Sequence input file

<table>
<thead>
<tr>
<th>LOCUS</th>
<th>BT069887</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>Zea mays full-length cDNA clone ZM_BFb0307L22 mRNA, complete cds.</td>
</tr>
<tr>
<td>ACCESSION</td>
<td>BT069887</td>
</tr>
<tr>
<td>VERSION</td>
<td>BT069887.1</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>FLI_CDNA</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Zea mays</td>
</tr>
<tr>
<td>ORGANISM</td>
<td>Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta; Spermatophyta; Magnoliophyta; Liliopsida; Poales; Poaceae; PACCAD clade; Panicoideae; Andropogoneae; Zea.</td>
</tr>
</tbody>
</table>

FEATURES

source
1..973
/mol_type="mRNA"
/db_xref="taxon:4577"
/strain="B73"
/cline="ZM_BFb0307L22"
/organism="Zea mays"

CDS
98..613
/db_xref="GI:224035417"
/codon_start=1
/protein_id="ACN36784.1"
/translation="MGIISKLVPILTEGNFVERCLEILRNLSMEEAVARITRDRCL
ASVAEYLDGTSPERQHAVILLACSCSAEDCLLYMKEGVIPALVDSVNGTEAKG
CSTKLLHLRDMRRSDQFTNCSQVEAATGMVVEDAPKNSVHKQPASKSRFFQRKLN
IFSKPRSLILF"
/product="unknown"

ORIGIN
1 aatactggca tctgaagata cccaaggccc cgaattgtct ctgaagatca tctgcgagct
61 ttcatccgac ggcgatatga gatcttcggtt agtttcaatg ggaataatct cgaagctttg
121 tcccattttta accgaaggaa acttcgctga gcgcgttttg gagatcctgc ggaaccttag
>ACN36784.1 BT069887
MGIISKLVPILTEGNFVERCLEILRNLSMEEAVARITRTDRCLASVAEYLDTGSPTERQ
HAVVILLAVCSCSAEDCLLMKEGVIPALVDLGSNGTEEAHKGCSTKLLHLRRMRRSDQF
TNSCSQEVAATGMVVEDAPKNSVHKQPASKSRFFQRKNITSKPRSLTLF
>ACN36708.1 BT069811
MGNMMDNLLVRSLSKSKGRLVDDIAAPSPVKAPDDDETDAEGHEESPMETVRSKCIQL
LLLGAIDSIQKRYWSRLKATQQIAIMDILLSLEFASSYNPSNFTRMRHIPLERPLNL
LLRQELEVGTTIYLHSTVEQDKIDSIETNGLNVEGDQEKIKYLAEGKLVSFCGRYI
LKEASVLQPSSTGAEASADIHRVLDLRAPVIVKVLKMCIMDAQIFRRHLEKFEYPLITKLI
CCDQMDVRGALGDLSKQLTLPMP
Outline

Perl Intro
- Basics
- Syntax and Variables
- More complex: References
- Routines
- Reading and Writing
- Regular Expressions

BioPerl
- Intro
- Useful modules
- Data formats
- Databases in BioPerl
- Sequence objects details

Trees
- Multiple Alignments
- BLAST and Sequence Database searching
- Other general Perl modules
Tree objects

- Bio::TreeIO Tree parser/writer can read/write trees in New Hampshire/Newick, nexml, phyloxml, and Nexus formats
- Build up Bio::Tree::Tree objects with Bio::Tree::Node objects in memory (on dev branch, code to represent in SQLite db to reduce memory)
use Bio::TreeIO;
my $in = Bio::TreeIO->new(-format => 'nexus',
                          -file => shift @ARGV);
my $out = Bio::TreeIO->new(-format => 'newick');
while (my $tree = $in->next_tree) {
  $out->write_tree($tree);
}
use Bio::TreeIO;
my $in = Bio::TreeIO->new(-format => 'newick, -file => shift @ARGV);

while( my $tree = $in->next_tree ) {
  my @nodes = $tree->get_nodes;
  my (@tips) = grep { $_->is_Leaf() } @nodes;
  my @tips; # TMTOWTDI
  for my $n ( @nodes ) {
    if( $n->is_Leaf() ) {
      push @tips2, $n;
    }
  }
  my ($cat) = grep { $_[0]->id eq 'cat' } @tips;
  my ($dog) = grep { $_[0]->id eq 'dog' } @tips;
  my $lca = $tree->get_lca($cat,$dog); # get least common ancestor
}
Outline

Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees

Multiple Alignments
BLAST and Sequence Database searching
Other general Perl modules
use Bio::AlignIO;
my $in = Bio::AlignIO->new(
  format => 'clustalw',
  file => shift @ARGV);

my $out = Bio::AlignIO->new(
  format => 'phylip',
  file => shift @ARGV);

while (my $aln = $in->next_aln) {
  $out->write_aln($aln);
}
Multiple Alignment objects

- Formats supported: Clustalw, FastA, PHYLIP, pfam, stockholm, selex, psi-blast, xmfa, mega
- Alignment objects can be queried for conserved residues, consensus sequence, gapped positions
Outline
Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments

BLAST and Sequence Database searching
Other general Perl modules
Seq Database Search objects

- Parsing results from BLAST, FastA, HMMER, BLAT, etc
- Bio::SearchIO is the main framework for this
- Bio::Search::Result - result objects
- Bio::Search::Hit - Hit/Subject objects
- Bio::Search::HSP - Alignments (High scoring Segment Pairs)

See the HowTo
http://www.bioperl.org/wiki/HOWTO:SearchIO
my $in = Bio::SearchIO->new(-format => 'blast',
                            -file => shift @ARGV);

while( my $r = $in->next_result ){
    print $r->query_name, "\n";
    while( my $h = $r->next_hit ) {
        print "\t", $h->name, " ", $h->significance\n";
        while( my $hsp = $h->next_hsp ) {
            print "\t\t", $hsp->query->start, "..",$hsp->query->end, "\n";
            print "\t\t", $hsp->hit->start, "..",$hsp->hit->end, "\n";
            print "\t\t", $hsp->evalue, " ",$hsp->frac_identical, " ",
                   $hsp->frac_conserved, "\n";
            print "\t\t", $hsp->query_string, "\n";
        }
    }
}
}
Outline

Perl Intro
Basics
Syntax and Variables
More complex: References
Routines
Reading and Writing
Regular Expressions

BioPerl
Intro
Useful modules
Data formats
Databases in BioPerl
Sequence objects details
Trees
Multiple Alignments
BLAST and Sequence Database searching

Other general Perl modules
- List::Util for several list utilities
- Getopt::Long for command line argument processing
- Statistics::Descriptive can calculate median, mean for data distribution