

**Tree Fun!**

# Taxonomy data

- Local indexed files or access NCBI (HTTP)
- Also access to BioSQL
- scripts/taxa: local\_taxonomydb\_query, taxid4species

# Querying Local Taxonomy DB

Download nodesfile and namesfile from  
NCBI /pub/taxonomy

```
use Bio::DB::Taxonomy;
my $db = Bio::DB::Taxonomy->new(
    -source => 'flatfile',
    -nodesfile=> $nodefile,
    -namesfile=> $namesfile);
$node = $db->get_Taxonomy_Node(
    -taxonid => '9606');
$node = $db->get_Taxonomy_Node(
    -name => 'Homo sapiens');
```

# Querying Remote Taxonomy DB

```
use Bio::DB::Taxonomy;
my $db = new Bio::DB::Taxonomy(
    -source => 'entrez');

$node = $db->get_Taxonomy_Node(
    -taxonid => '9606');
$node = $db->get_Taxonomy_Node(
    -gi => $gi); # lookup a taxonid for seq GI
```

**Careful!** Like RemoteBlast and DB::GenBank you can get  
your site cut off from NCBI!!

# Put it together

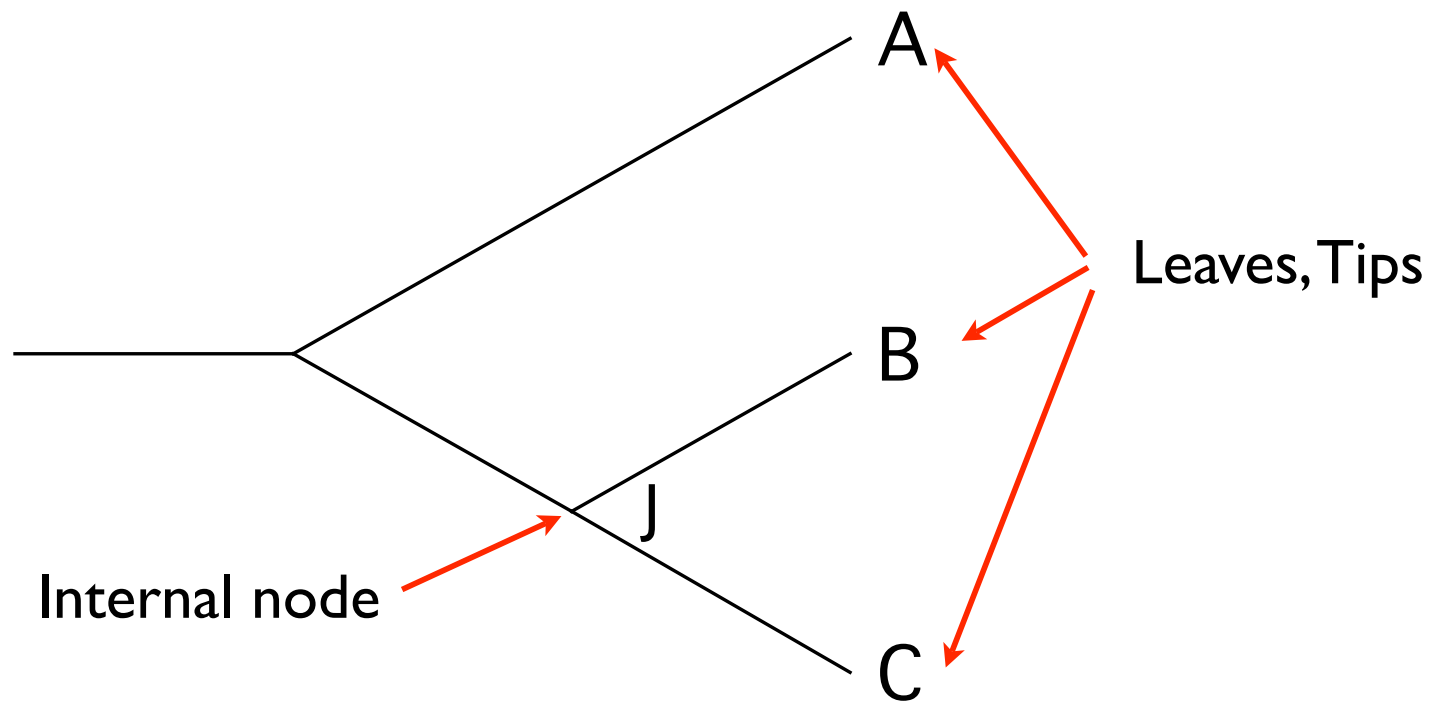
```
use Bio::DB::Taxonomy;
use Bio::SearchIO;
my $db = Bio::DB::Taxonomy->new(
    -source => 'flatfile',
    -nodesfile=> $nodefile,
    -namesfile=> $namesfile);
my $in = Bio::SearchIO->new(-format => 'fasta',
    -file => 'blastfile.FASTX');

while( my $r = $in->next_result ) {
    while( my $h = $r->next_hit ) {
        my ($gi) = ( $h->name =~ /gi\|(\d+)/ );
        my $kingdom = &gi_to_kingdom($gi);
        if( $kingdom ) {
            $classify{$r->query_name}->{$kingdom}++;
        }
    }
}
```

```
sub gi_to_kingdom {
  my $gi = shift;
  my $taxid = $GI2TaxId{$gi}; # build a local index from NCBI files
  my $node = $TaxDB->get_Taxonomy_Node($taxid);
  if( ! $node ) {
    warn("cannot find node for gi=$gi ($hname) (taxid=$taxid)\n");
    next;
  }
  my $kingdom;
  my $nm = $taxon->scientific_name;
  while( my $n = $taxon->ancestor ) {
    my $rank = lc $n->rank;
    my $name = $n->scientific_name;

    if( $rank eq 'kingdom' ||
        $rank eq 'superkingdom' || $name eq 'Viruses') {
      $kingdom = $name;
      last;
    }
    $taxon = $n;
  }
}
```

# Anatomy of a tree



$(A, (B, C)J);$

# Tools for Evolutionary and Population analyses

- Population Genetics Modules
- Taxonomy
- Molecular Evolution
- Phylogenetic Tree Building and Manipulation

# Molecular Evolution Tools

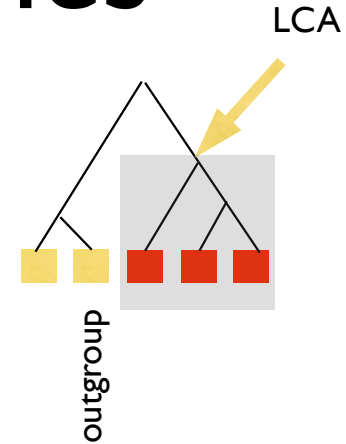
- Bio::Align::DNAStatistics - native calculation of distances, Ks, Ka
- Bio::Align::Utilities - translate aa to nt alignments
- Bio::Tools::Phylo::PAML - parse Codeml output
- Bio::Tools::Run::Phylo::PAML::Codeml - run Codeml (or YN00)

# Phylogenetic Trees

- Support reading of Tree data
  - newick, nexus, nhx formats
- Manipulation of Trees
- Trees are connections of Nodes which have ancestor and children pointers

# Simple Tree Routines

- Lowest Common Ancestor
- Tests of monophyly, paraphyly
- Reroot tree
- Distances between two nodes
- Find a node by name



# Constructing Trees

- `Bio::Tree::DistanceFactory` has UPGMA, Neighbor-Joining implemented
- Build a matrix with `Align::DNAStatistics` or `Align::ProteinStatistics` OR read in one from Phylip with `Bio::Matrix::IO`
- Create NJ tree

# Interfacing with Phylip

- Bioperl 1.5.1 supports Phylip 3.6
- Can run tools with bioperl-run package
- `Bio::Tools::Run::Phylo::Phylip`
  - ProtDist (also parser `Bio::Matrix::IO`)
  - Neighbor, ProtPars, SeqBoot, Consense
  - DrawGram, DrawTree

# Reading/Writing a Tree

```
#!/usr/bin/perl -w
use Bio::TreeIO;
use strict;
my $in = Bio::TreeIO->new(-format => 'nexus',
                          -file   => 'trees.nex');
my $out = Bio::TreeIO->new(-format => 'newick',
                          -file   => '>trees.nh');
while( my $tree = $in->next_tree ) {
    $out->write_tree($tree);
}
```

# Fetching subset of nodes

```
#!/usr/bin/perl -w
use strict;
use Bio::TreeIO;
my $in = Bio::TreeIO->new(-format => 'newick',
                        -fh    => \*DATA);

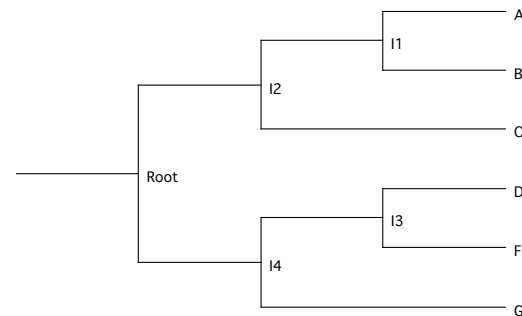
if( my $tree = $in->next_tree ) {
    my @nodes = $tree->get_nodes;
    my @tips = grep { $_->is_Leaf } @nodes;
    print "there are ",scalar @tips, " tips\n";
    my @internal = grep { ! $_->is_Leaf } @nodes;
    print "there are ",scalar @internal, " internal nodes\n";
    my ($A_node) = $tree->find_node(-id => 'A');
    print "branch length of Ancestor of ",$A_node->id,
          " is ", $A_node->ancestor->branch_length, "\n";
}

__DATA__
(((A:10,B:11):2,C:5),((D:7,F:6):17,G:8));
```

# Walking up the tree (tips to root)

```
if( my $tree = $in->next_tree ) {  
  my @tips = grep { $_->is_Leaf } $tree->get_nodes;  
  for my $node ( @tips ) {  
    my @path;  
    while( defined $node ) {  
      push @path, $node->id;  
      $node = $node->ancestor;  
    }  
    print join(",", @path), "\n";  
  }  
}  
  
__DATA__  
(((A:10,B:11)I1:2,C:5)I2,((D:7,F:6)I3:17,G:8)I4)Root;
```

```
C,I2,Root  
A,I1,I2,Root  
B,I1,I2,Root  
G,I4,Root  
D,I3,I4,Root  
F,I3,I4,Root
```



# From Alignments to Trees

- `Bio::AlignIO` to parse the alignment
- `Bio::Align::ProteinStatistics` to compute pairwise distances
- `Bio::Tree::DistanceFactory` to build a tree based on a matrix of distances using NJ or UPGMA
- More sophisticated tree building should be done with tools like `phylml`, `PAUP`, `MOLPHY`, `PHYLIP`, `MrBayes`, or `PUZZLE`

# Testing phylogenetic hypotheses

- No sophisticated ML methods are currently built in Bioperl for testing for phylogenetic correlations, etc
- Can export trees and use tools like Mesquite
- Work to be finished in Dec 2006 to fully integrate more phylogenetic tools into BioPerl

# Trees in BioPerl

- Trees are Nodes and Edges
- Nodes have pointers to parents (only 1) and children (0..N)
- Trees can be un-rooted or rooted

# Get a tree

```
use Bio::TreeIO;
my $in = Bio::TreeIO->new(-format => 'newick',
                          -file   => 'one.tre');
while( my $tree = $in->next_tree ) {
    my @nodes = $tree->get_nodes;
    print "There are ", scalar @nodes, " nodes in tree\n";
}
```

# Tree Methods

- `get_nodes` - get all the nodes
- `get_root_node` - get the root node
- `get_leaf_nodes` - get all the leaves
- `score` - if associated, a score value for the tree (likelihood, etc)
- `total_branch_length` - sum of all branch lengths

# Node Methods

- `id()` - human readable ID, this is the Taxon name
- `branch_length()` - if available the length of branch from the parent node to this node
- `ancestor()` - Node which is above this in the tree (or null if root)
- `each_Descendent` - all immediate descendants. Not recursive. This will be empty for leaves.
- `get_all_Descendents` - recursively fetch all descendants

# A Tree Problem

- Print out the name and branch length of every internal node in the tree.
- The name and branch length of its child.

```
use Bio::TreeIO;
my $in = Bio::TreeIO->new(-format => 'newick ',
                        -file => 'trees.tre ');
while ( my $tree = $in->next_tree ) {
  for my $node ( grep { ! $_->is_Leaf } $tree->get_nodes ) {
    next if ! $node->ancestor ; # ignore the root node
    print " Node : ", $node->id , " length : ",
          $node->branch_length , " ";
    for my $child ( $node -> get_Descendents ) {
      print " child : ", $child->id , " ",
            $child->branch_length, " ";
    }
    print "\n";
  }
}
```

# Notes on Trees

- Internal IDs CAN be bootstrap values, but the data formats do not dictate this. One must KNOW what information is encoded in internal node labels, the bootstrap() data will not be filled in automatically.

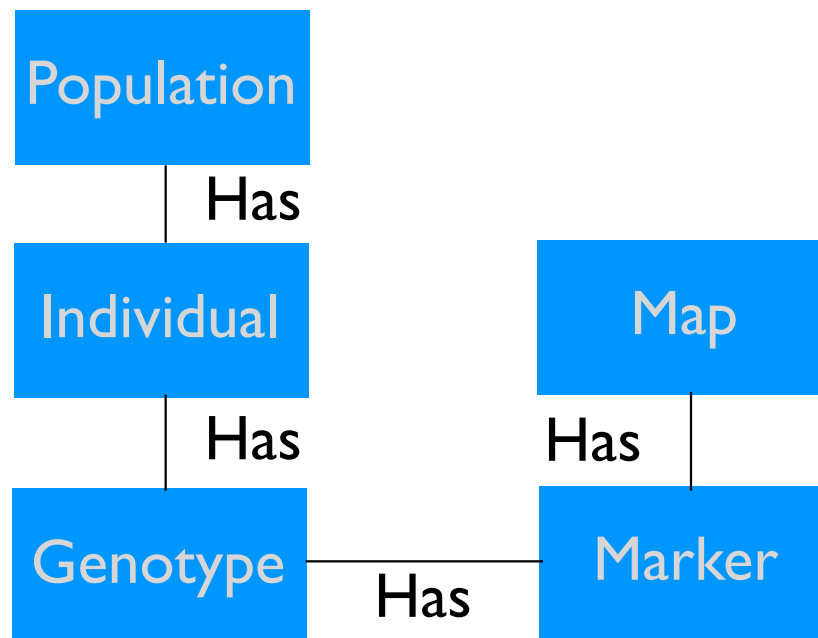
# Bioperl & Population Genetics

- Basic data
  - Marker - polymorphic region of genome
  - Individual - individual sampled
  - Genotype - observed allele(s) for a marker in an individual
  - Population - collection of individuals
  - See [http://bioperl.org/wiki/PopGen\\_modules](http://bioperl.org/wiki/PopGen_modules)

# The Players

- Bio::PopGen::Population - container for Individuals, calculate summary stats
- Bio::PopGen::Individual - container for Genotypes
- Bio::PopGen::Marker - summary info about a Marker (primers, genome loc, allele freq)
- Bio::PopGen::Genotype - pairing of Individual & Marker; allele container

# Population Genetics Data Objects



# Some Data Formats

- PrettyBase format (Seattle SNP data)

```
MARKER      SAMPLEID  ALLELE1  ALLELE2
ProcR2973EA    01      C        T
ProcR2973EA    02      N        N
ProcR2973EA    03      C        C
ProcR2973EA    04      C        T
ProcR2973EA    05      C        C
```

- Comma Delimited (CSV)

```
SAMPLE,MARKERNAME1,MARKERNAME2,...
SAMPLE,ProcR2973EA,Marker2
sample-01,C T,G T
```

- Phase format

```
3
5
P 300 1313 1500 2023 5635
MSSSM
#1
12 1 0 1 3
11 0 1 0 3
```

# Reading in Data

```
use Bio::PopGen::IO;
my $in = Bio::PopGen::IO->new(-format => 'csv'
                             -file   => 'dat.csv');

my @pop;
while( my $ind = $in->next_individual ) {
    push @pop, $ind;
}
# OR
my $pop = $in->next_population;
```

# Ready Built Stuff

- Bio::PopGen::PopStats - population level statistics (only  $F_{ST}$  currently)
- Bio::PopGen::Statistics - suite of Population Genetics statistical tests and summary stats.
- Bio::PopGen::Simulation::Coalescent - primitive Coalescent simulation
- Basic tree topology and branch length assignment.

# Using the Modules

```
use Bio::PopGen::Statistics;
my $stats = Bio::PopGen::Statistics->new();
my $pi = $stats->pi($population);
# or use an array reference of Individuals
my $pi = $stats->pi(\@individuals);
# Tajima's D
my $TajimaD = $stats->tajima_D($population);
# Fu and Li's D
my $FuLiD = $stats->fu_and_li_D($ingroup_pop,
                               $outgroup_ind);

# Fu and Li's D*
my $FLDstar = $stats->fu_and_li_D_star($population);

# pairwise composite LD
my %LDstats = $stats->composite_LD($population);
my $LDarray = $LDstats{'marker1'}->{'marker2'};
my ($ldval,$schisq) = @$LDarray;
```

# Getting Data from Alignments

- use `Bio::AlignIO` to read in Multiple Sequence Alignment data
- `Bio::PopGen::Utilities aln_to_population` will build Population from MSA
  - Will make a “Marker” for every polymorphic site (or if asked every site)
  - Eventually will have ability to only get silent/non-silent coding sites

# Population Tests for selection

- Tajima's  $D$ ,  $F_u$  and Li's  $D$ ,  $F_u$  and Li's  $F$
- McDonald-Kreitman

# Automating PAML

- PAML - phylogenetic analysis with maximum likelihood
- Estimate synonymous and non-synonymous substitution rates
- Along branches of a tree or in a pairwise fashion

# Preparing Data

- Multiple sequence alignments of protein coding sequence
- No stop codons!
- Must be aligned on codon boundaries
- Easiest way is to align at protein level, then project back into CDS alignment

# Doing Protein Alignments

- Bio::Tools::Run::Alignment::Clustalw or Bio::Tools::Run::Alignment::MUSCLE or just prepare the sequence files and run the alignment programs via scripts
- Bio::AlignIO to parse the alignment data
- Bio::Align::Utilities to project back into CDS space

# Build tree or assume a tree

- If doing analysis of genomes which have a known species tree - use that tree
- Branch lengths are not part of PAML. Multiple topologies can be provided to test alternative hypotheses (by comparing maximum likelihood values)

# Running PAML

```
#!/usr/bin/perl -w
use strict;
use Bio::Tools::Run::Phylo::PAML::Codeml;
use Bio::AlignIO;
my $factory = Bio::Tools::Run::Phylo::PAML::Codeml->new(
    -params => { 'runmode' => -2,
                'seqtype' => 1});
my $alnio = Bio::AlignIO->new(-format => 'clustalw',
                             -file    => 'cds.aln');
my $aln = $alnio->next_aln; # get the alignment from file
$factory->alignment($aln); # set the alignment
my ($returncode,$parser) = $factory->run();
my $result = $parser->next_result;
my $MLmatrix = $result->get_MLMatrix;

print "Ka = ", $MLmatrix->[0]->[1]->{'dN'}, "\n";
print "Ks = ", $MLmatrix->[0]->[1]->{'dS'}, "\n";
print "Ka/Ks = ", $MLmatrix->[0]->[1]->{'omega'}, "\n";
```

# Parsing PAML

```
#!/usr/bin/perl -w
use strict;
use Bio::Tools::Phylo::PAML;
my $parser = Bio::Tools::Phylo::PAML->new
  (-file => 'results/mlc', -dir => 'results');
if( my $result = $parser->next_result ) {
  my @otus = $result->get_seqs;
  # get Nei & Gojobori dN/dS matrix
  my $NGMatrix = $result->get_NGmatrix;
  printf "%s and %s dS=%.4f dN=%.4f Omega=%.4f\n",
    $otus[0]->display_id, $otus[1]->display_id,
    $NGMatrix->[0]->[1]->{dS}, $NGMatrix->[0]->[1]->{dN},
    $NGMatrix->[0]->[1]->{omega};
}
```

# Getting the Trees out

```
my @trees = $result->get_trees;
for my $tree ( @trees ) {
    print "likelihood is ", $tree->score, "\n";
    # do something else with the trees,
    # for non runmode -2 results
    # inspect the tree, branch specific rates
    # the "t" (time) parameter is available via
    # ("omega", "dN", etc.) are available via
    # ($omega) = $node->get_tag_values('omega');
    for my $node ( $tree->get_nodes ) {
        print $node->id, " t=", $node->branch_length,
            " omega ", $node->get_tag_values('omega');
    }
}
```

# Multiple Sequence Alignments

- Bio::AlignIO to read alignment files
- Produces Bio::SimpleAlign objects
- Interface and objects designed for round-tripping and some functional work
- Could really use an overhaul or a parallel MSA representation

# Using AlignIO

```
use Bio::AlignIO;
my $in = Bio::AlignIO->new(-format => 'clustalw',
                           -file    => 'filename.aln');
my $out = Bio::AlignIO->new(-format => 'phylip',
                            -file    => '>slice.phy');

while( my $aln = $in->next_aln ) {
    print $aln->no_sequences," sequence in alignment\n";
    for my $sequence( $aln->each_seq ) {
        print $sequence->display_id, "\n";
    }
    my $slice = $aln->slice(10,30); # slice of alignment
    $out->write_aln($slice);
}
```