Writing and running efficient Perl on Genepool

version control, analysis, packaging, and environment modules
Gene Design

CloneQC

BioStudio

bitbucket.org/notadoctor
search.cpan.org/~notdoctor/
Excellent Practices

Version Control

Dynamic Analysis – Profiling

Static Analysis – Criticism

Packaging and Testing

Environment Modules
Version Control

the coder’s laboratory notebook

Remember kids, the only difference between screwing around and science is writing it down.
richards:JGISynBio richards$ git status
# On branch master
# Changes not staged for commit:
#  (use "git add <file>..." to update what will be committed)
#  (use "git checkout -- <file>..." to discard changes in working directory)
#  modified:   bin/JGISB_Order_Stitch.pl
#  modified:   bin/JGISB_SubmitOligoJob.pl

# Untracked files:
#  (use "git add <file>..." to include in what will be committed)
#  JGISB_submit.sh
no changes added to commit (use "git add" and/or "git commit -a")
richards:JGISynBio richards$
richards:JGISynBio richards$
richards:JGISynBio richards$
richards:JGISynBio richards$ git diff bin/JGISB_Order_Stitch.pl
diff --git a/bin/JGISB_Order_Stitch.pl b/bin/JGISB_Order_Stitch.pl
index f39065a..55f8e8a 100755
--- a/bin/JGISB_Order_Stitch.pl
+++ b/bin/JGISB_Order_Stitch.pl
@@ -1,10 +1,7 @@
 #!/usr/bin/env perl

 use Bio::GeneDesign;
-use Spreadsheet::ParseExcel;
 use Spreadsheet::WriteExcel;
-use File::Basename;
-use Config::Auto;
 use Getopt::Long;
 use Pod::Usage;

@@ -22,11 +19,9 @@ $1 = 1;
**February 7**

### make sure sequences are uppercase

**Author:** Sarah Richardson  
**Hash:** ac6eb758502e57166e0ac4096cae30d17cf7da98

#### Summary:
- Add the pattern aligner to the engine
- Merge branch 'master' of https://bitbucket.org/notadocor/genedesign-dev
- Adjust suffix tree so that in peptide implementations, the originating nucleotide sequence is made uppercase

#### Details:
- When Bio::ConfigData isn't installed, this line causes an uninitialized warning.
- If running in simple mode, will NOT have enz/zne tags to be removed.

---

### Changes:

**lib/Bio/GeneDesign/Vector.pm**

- 2 changes, 2 insertions, 2 deletions
- Modified by Sarah Richardson

```perl
my %dsubs = map { join(q(), $_->get_tag_values("label")) => $_ } @ps;
if (exists $dsubs("CLH5")) {
  $self->{chew5} = $dsubs("CLH5")->{seq} = $dseq;
} else { 
  ...

if (exists $dsubs("CLH3")) {
  $self->{chew3} = $dsubs("CLH3")->{seq} = $dseq;
} else { 
  ...
```

---

**SHA:** ac6eb758502e57166e0ac4096cae30d17cf7da98

**Author:** Sarah Richardson

**Date:** February 7, 2013 4:08 PM

**Subject:** make sure sequences are uppercase

**Parent:** faecfc6860c468bbbad41c7dbd8dca05c4dd3709c9

**Tree:** 055a43c2c938c93c03e3c513f4cc97bd449f41cb74
Version Control Carrots

**branching**: risk free development while your production code ticks onwards unmolested. Take the fear out of refactoring!

**backups**: a git server and a healthy commit habit let you work from anywhere – and can prevent you from losing work with an unconsidered overwrite.

**blamestorming**: in groups (n >= 1) it is easy to find out who is contributing (or breaking) what, or to organize an issue system to assign tasks or track bugs tied to commits.

**ninja deploys**: cron jobs can automatically pull the latest version of your code from the repository and deploy it on genepool while you eat lunch or nap!

...and so many more
If you can think of a single situation in which not using version control is an entirely logical and defensible position email it to me at SMRichardson

I’m dying to hear it
git started now!

LBNL on bitbucket
https://commons.lbl.gov/display/itfaq/Version+Control+Systems

NERSC on git
http://www.nersc.gov/users/software/development-tools/git/

a free, easy to read, and all around wonderful introduction to source control and git reference

do right by your code.
git init my_code_dir/
Dynamic Analysis

check yourself before you wreck yourself

# This is O(scary), but seems quick enough in practice.

foreach my $a (@bigarray)
{
    foreach my $b (@biggerarray)
    {
        foreach my $c (@youdontwannaknow)
        {
            foreach my $d (@justgiveup)
            {
                push @newarray, function($a, $b, $c, $d);
            }
        }
    }
}
richards:makingoligos richards$ time -p GD_Design_Building_Blocks.pl -i Test_one_big.FASTA -o . -f genbank
Working on Monocot_A1 (6125 bp)...
Warning: integer value out of range 20 more than (reset to) 14

Made building blocks of Monocot_A1
Wrote ./Test_one_big_BB.genbank

Awkwardly brought to you by GD_Design_Building_Blocks_5.00

real 23.87
user 12.20
sys 8.79
richards:makingoligos richards$
All the Code That’s Fit to printf()

March 5, 2008, 1:43 pm  |  13 Comments

The New York Times Perl Profiler

By ADAM KAPLAN

I work in the NYTimes.com feeds team. We handle retrieving, parsing and transforming incoming feeds from whatever strange proprietary format our partners choose to give us into something that our CMS can digest. As you can imagine, we deal with a huge amount of text processing. To handle all of these transformations as efficiently as possible we rely heavily on the magic of Perl. Recently, as feeds become more and more important, we have begun to feel pains caused by past impromptu segments of inefficient code written to meet quick, episodic deadlines. A situation that we are especially prone to as a fast moving news organization.

I am a relatively new employee here at NYTimes.com and one of my responsibilities is to create tools to help ensure the integrity and scalability of our code. To this end, I would like to introduce you to The New York Times Perl Profiler, or Devel::NYTProf. The purpose of this tool is to allow developers to easily profile Perl code line-by-line with minimal computational overhead and highly visual output. With only one additional command, developers can generate robust color-coded HTML reports that include some useful statistics about their Perl program. Here is the typical usage:

perl -d:NYTProf myslowcode.pl
nytprofhtml

See? It’s easy! nytprofhtml is an implementation of the included reporting tool and is perfectly capable of running on its own. For more information on it and its capabilities, please see its documentation.
Performance Profile Index
For /usr/local/bin/BS_ChromosomeAnalyzer.pl
Run on Tue Feb 12 00:09:53 2013
Reported on Tue Feb 12 00:11:35 2013

Profile of /usr/local/bin/BS_ChromosomeAnalyzer.pl for 33.4s (of 51.1s), executing 37709980 statements and 2228909 subroutine calls in 230 source files and 76 string evals.

/Library/Perl/5.12/Bio/AnalysisParser.pm

Top 15 Subroutines

<table>
<thead>
<tr>
<th>Calls</th>
<th>P</th>
<th>F</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7.41s</td>
<td>7.76s</td>
<td>Bio::BioStudio::Mask::find_overlaps</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4.46s</td>
<td>4.81s</td>
<td>Bio::BioStudio::Mask::find_deserts</td>
</tr>
<tr>
<td>27282</td>
<td>9</td>
<td>2</td>
<td>4.22s</td>
<td>4.22s</td>
<td>DBI::st::execute (xsub)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3.42s</td>
<td>3.45s</td>
<td>Bio::BioStudio::Mask::add_to_mask</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.31s</td>
<td>1.31s</td>
<td>Bio::BioStudio::Mask::new</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>984ms</td>
<td>32.1s</td>
<td>Bio::BioStudio::Analyze::ProteinCodingGenes::_analyze</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>917ms</td>
<td>1.27s</td>
<td>Bio::GeneDesign::Codons::_codon_count</td>
</tr>
<tr>
<td>18240</td>
<td>1</td>
<td>1</td>
<td>445ms</td>
<td>5.32s</td>
<td>Bio::DB::SeqFeature::Store::DBI::mysql::fetch_SeqFeatures</td>
</tr>
<tr>
<td>3266</td>
<td>3</td>
<td>1</td>
<td>353ms</td>
<td>353ms</td>
<td>Bio::GeneDesign::Codons::CORE::match (opcode)</td>
</tr>
<tr>
<td>82201</td>
<td>7</td>
<td>1</td>
<td>321ms</td>
<td>448ms</td>
<td>Bio::DB::SeqFeature::Store::DBI::mysql::qualify</td>
</tr>
<tr>
<td>69945</td>
<td>5</td>
<td>2</td>
<td>305ms</td>
<td>917ms</td>
<td>Bio::SeqFeature::Generic::start</td>
</tr>
<tr>
<td>69945</td>
<td>5</td>
<td>2</td>
<td>294ms</td>
<td>831ms</td>
<td>Bio::SeqFeature::Generic::end</td>
</tr>
<tr>
<td>69945</td>
<td>1</td>
<td>1</td>
<td>291ms</td>
<td>439ms</td>
<td>Bio::Location::Simple::end</td>
</tr>
<tr>
<td>27281</td>
<td>1</td>
<td>1</td>
<td>283ms</td>
<td>398ms</td>
<td>DBI::::db::prepare_cached</td>
</tr>
<tr>
<td>73827</td>
<td>2</td>
<td>2</td>
<td>276ms</td>
<td>403ms</td>
<td>Bio::Location::Simple::start</td>
</tr>
</tbody>
</table>

See all 4273 subroutines

You can view a tree map of subroutine exclusive time, grouped by package. NYTPProf also generates call-graph files in Graphviz format: inter-package calls, all inter-subroutine calls (probably too complex to render easily).

You can hover over some table cells and headings to view extra information. Some table column headings can be clicked on to sort the table by that column.
sub find_overlaps

# spent 7.76s (7.41+358ms) within Bio::BioStudio::Mask::find_overlaps which was called:
# once (7.41s+358ms) by Bio::BioStudio::Analyze::ProteinCodingGenes::_analyze at line 206 of Bio/B
{
    my ($self) = @_;
    my @ranges;

    my $len = $self->{length};
    my $init = $self->{mask}->{0}->{0};
    my $start = $init > 1 ? $init : undef;
    my $last = $init > 1 ? $init : 1;
    my $flag = $init > 1 ? $init : 0;

    my %list = ();
    for (my $x = 0; $x < $len; $x++)
    {
        my @here = @{ $self->{mask}->{ $x }};
        my $stat = shift @here;

        # moving from non overlap to overlap
        if ($stat > 1 && $last <= 1)
        {
            $start = $x+1;
            $flag = 1;
            %list{$_} = [$x+1, $x+1] foreach @here;
        }
        # moving from overlap to non overlap
        elsif ($last > 1 && $stat <= 1)
        {
            my $seqid = $self->{seqid};
            my $lapfeat = Bio::SeqFeature::Generic->new(
                # spent 90.7ms making 471 calls to Bio::SeqFeature::Generic::new, avg 193μs/call

                -start   => $start,
                -end     => $x,
                -display_name  => "$start..$x",
                -primary_tag  => "overlap",
                -seq_id    => $seqid
            );
            foreach my $featid (keys %list)
            {
                my ($lstart, $lstop) = @{$list{$featid}};
                my $subfeat = Bio::SeqFeature::Generic->new(
                    # spent 149ms making 942 calls to Bio::SeqFeature::Generic::new, avg 158μs/call

                    -start   => $lstart,
                    -end     => $lstop,
                    -primary_tag  => "overlap",
                    -seq_id    => $seqid
                );
            }
        }
    }

    my $seqid = $self->{seqid};
    my $lapfeat = Bio::SeqFeature::Generic->new(
        # spent 90.7ms making 471 calls to Bio::SeqFeature::Generic::new, avg 193μs/call

        -start   => $start,
        -end     => $x,
        -display_name  => "$start..$x",
        -primary_tag  => "overlap",
        -seq_id    => $seqid
    );
    foreach my $featid (keys %list)
    {
        my ($lstart, $lstop) = @{$list{$featid}};
        my $subfeat = Bio::SeqFeature::Generic->new(
            # spent 149ms making 942 calls to Bio::SeqFeature::Generic::new, avg 158μs/call

            -start   => $lstart,
            -end     => $lstop,
            -primary_tag  => "overlap",
            -seq_id    => $seqid
        );
    }
}
sub find_overlaps
{
    my ($self) = @_;
    my @ranges;

    my $len = $self->{length};
    my $mask = $self->{mask};
    my $seqid = $self->{seqid};
    my $init = $mask->[0]->[1];
    my $start = $init > 1 ? $init : undef;
    my $last = $init > 1 ? $init : 1;
    my $flag = $init > 1 ? $init : 0;

    my @list = ();
    for (my $x = 0; $x < $len; $x++)
    {
        my $stat = $mask->[0]->[1];

        # moving from non overlap to overlap
        if ($stat > 1 && $last <= 1)
        {
            $start = $x+1;
            $flag = 1;
            @list{$_} = [$x+1, $x+1] foreach keys %{$mask->[0]->{0}};
        }

        # moving from overlap to non overlap
        elsif ($last > 1 && $stat <= 1)
        {
            my $lapfeat = Bio::SeqFeature::Generic->new(
                -start => $start,
                -end => $x,
                -display_name => "$start..$x",
                -primary_tag => "overlap",
                -seq_id => $seqid
            );

            foreach my $featid (keys %list)
            {
                my ($start, $stop) = @{$list{$_}};
                my $subfeat = Bio::SeqFeature::Generic->new(
                    -start => $start,
                    -end => $stop,
                    -display_name => "$start..$stop",
                    -primary_tag => "overlap",
                    -seq_id => $seqid
                );

            }
        }

    }

    my $lapfeat = Bio::SeqFeature::Generic->new(
        -start => $start,
        -end => $x,
        -display_name => "$start..$x",
        -primary_tag => "overlap",
        -seq_id => $seqid
    );

    foreach my $featid (keys %list)
    {
**Performance Profile Index**

For /usr/local/bin/BS_ChromosomeAnalyzer.pl  
Reported on Tue Feb 12 01:28:08 2013

Profile of /usr/local/bin/BS_ChromosomeAnalyzer.pl for 31.1s (of 49.8s), executing 34426805 statements and 2228909 subroutine calls in 230 source files and 76 string evals.

```
/Library/Perl/5.12/Bio/AnalysisParser1.pm
```

## Top 15 Subroutines

<table>
<thead>
<tr>
<th>Calls</th>
<th>P</th>
<th>F</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>27282</td>
<td>9</td>
<td>2</td>
<td>4.34s</td>
<td>4.34s</td>
<td>DBI::st::execute (xsub)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4.33s</td>
<td>4.71s</td>
<td>Bio::BioStudio::Mask::find_overlaps</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4.20s</td>
<td>4.58s</td>
<td>Bio::BioStudio::Mask::find_deserts</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.85s</td>
<td>2.89s</td>
<td>Bio::BioStudio::Mask::add_to_mask</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.08s</td>
<td>2.08s</td>
<td>Bio::BioStudio::Mask::new</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.38s</td>
<td>29.9s</td>
<td>Bio::BioStudio::Analyze::ProteinCodingGenes::analyze</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>935ms</td>
<td>1.29s</td>
<td>Bio::GeneDesign::Codons::codon_count</td>
</tr>
<tr>
<td>18240</td>
<td>1</td>
<td>1</td>
<td>465ms</td>
<td>5.50s</td>
<td>Bio::DB::SeqFeature::Store::DBI::mysql::fetch_SeqFeatures</td>
</tr>
<tr>
<td>3266</td>
<td>3</td>
<td>1</td>
<td>359ms</td>
<td>359ms</td>
<td>Bio::GeneDesign::Codons::CORE::match (opcode)</td>
</tr>
<tr>
<td>82201</td>
<td>7</td>
<td>1</td>
<td>327ms</td>
<td>458ms</td>
<td>Bio::DB::SeqFeature::Store::DBI::mysql::qualify</td>
</tr>
<tr>
<td>69945</td>
<td>5</td>
<td>2</td>
<td>318ms</td>
<td>944ms</td>
<td>Bio::SeqFeature::Generic::start</td>
</tr>
<tr>
<td>69945</td>
<td>5</td>
<td>2</td>
<td>315ms</td>
<td>864ms</td>
<td>Bio::SeqFeature::Generic::end</td>
</tr>
<tr>
<td>69945</td>
<td>1</td>
<td>1</td>
<td>301ms</td>
<td>452ms</td>
<td>Bio::Location::Simple::end</td>
</tr>
<tr>
<td>27281</td>
<td>1</td>
<td>1</td>
<td>298ms</td>
<td>415ms</td>
<td>Bio::Location::Simple::start</td>
</tr>
<tr>
<td>73827</td>
<td>2</td>
<td>2</td>
<td>283ms</td>
<td>414ms</td>
<td>Bio::Location::Simple::prepare_cached</td>
</tr>
</tbody>
</table>

See all 4273 subroutines

You can view a [treemap of subroutine exclusive time](#), grouped by package. NYTPprof also generates call-graph files in [Graphviz format](#): inter-package calls, all inter-subroutine calls (probably too complex to render easily).

You can hover over some table cells and headings to view extra information. Some table column headings can be clicked on to sort the table by that column.
Subroutine Exclusive Time Treemap
For /usr/local/bin/BS_ChromosomeAnalyzer.pl

Boxes represent time spent in a subroutine. Coloring represents packages. Click to drill-down into package hierarchy, reload page to reset.

Report produced by the NYTProf 4.25 Perl profiler, developed by Tim Bunce and Adam Kaplan.
profile away

NYTProfiler on CPAN – all the configuration options you could want
http://search.cpan.org/~timb/Devel-NYTProf-4.25/

an excellent lecture by the maintainer of the NYTProfiler

are you O(scary) or O(awesome)?
perl -d:NYTProf your_code.pl
nytprofhtml
Static Analysis

HATERS GONNA HATE

everyone’s a critic
1. Best Practices

<table>
<thead>
<tr>
<th>Three Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robustness</td>
</tr>
<tr>
<td>Efficiency</td>
</tr>
<tr>
<td>Maintainability</td>
</tr>
<tr>
<td>GD Build</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

```
richards:biostudio-dev richards$ perlcritic lib/Bio/BioStudio/Mask.pm
lib/Bio/BioStudio/Mask.pm source OK
richards:biostudio-dev richards$
```
<table>
<thead>
<tr>
<th>GD Build</th>
<th>GD Run</th>
<th>CQC Build</th>
<th>CQC Run</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
richards:~$ perlcritic --harsh lib/Bio/BioStudio/Mask.pm
Ambiguously named variable "last" at line 113, column 3. See page 48 of PBP. (Severity: 3)
Ambiguously named variable "last" at line 153, column 3. See page 48 of PBP. (Severity: 3)
Subroutine name is a homonym for builtin function length at line 336, column 1. See page 177 of PBP.
```

```
richards:~$ perlcritic --cruel lib/Bio/BioStudio/Mask.pm
Missing "SYNOPSIS" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "SUBRoutines/Methods" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "DIAGNOSTICS" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "Configuration and Environment" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "DEPENDENCIES" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "INCOMPATIBILITIES" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "BUGS AND LIMITATIONS" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Missing "LICENSE AND COPYRIGHT" section in POD at line 5, column 1. See pages 133,138 of PBP. (Severity: 2)
Postfix control "unless" used at line 54, column 5. See pages 96,97 of PBP. (Severity: 2)
Postfix control "unless" used at line 57, column 4. See pages 96,97 of PBP. (Severity: 2)
Postfix control "unless" used at line 60, column 4. See pages 96,97 of PBP. (Severity: 2)
Link L<Bio::DB::SeqFeature> on line 76 does not specify text at line 72, column 1. Without text-
Postfix control "unless" used at line 87, column 7. See pages 96,97 of PBP. (Severity: 2)
-1 is not one of the allowed literal values (0, 1, 2). Use the Readonly or Const::Fast module on
Ambiguously named variable "last" at line 113, column 3. See page 48 of PBP. (Severity: 3)
C-style "for" loop used at line 115, column 7. See page 100 of PBP. (Severity: 2)
Ambiguously named variable "last" at line 153, column 3. See page 48 of PBP. (Severity: 3)
C-style "for" loop used at line 157, column 7. See page 100 of PBP. (Severity: 2)
Postfix control "Foreach" used at line 166, column 32. See page 96 of PBP. (Severity: 2)
Postfix control "Foreach" used at line 241, column 18. See page 96 of PBP. (Severity: 2)
Postfix control "Foreach" used at line 271, column 18. See page 96 of PBP. (Severity: 2)
Postfix control "Foreach" used at line 288, column 18. See page 96 of PBP. (Severity: 2)
Subroutine name is a homonym for builtin function length at line 336, column 1. See page 177 of PBP.
```
richards:biostudio-dev richards$ perlcritic --cruel --verbose '%p
' lib/Bio/BioStudio.pm

Documentation::RequirePodSections
Documentation::RequirePodSections
Documentation::RequirePodSections
Documentation::RequirePodSections
Documentation::RequirePodSections
Documentation::RequirePodSections
Documentation::RequirePodSections
Documentation::RequirePodSections

Controls::ProhibitPostfixControls

ValuesAndOperators::ProhibitMagicNumbers

NamingConventions::ProhibitAnonymousNames

ControlStructures::ProhibitPostfixControls

Subroutines::ProhibitBuiltInHomonyms
    # perlcriticrc
    [-Subroutines::ProhibitUnusedPrivateSubroutines]
    [-Subroutines::ProhibitExcessComplexity]
    [-Subroutines::ProhibitManyArgs]
    [-ControlStructures::ProhibitPostfixControls]

controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
controls::ProhibitPostfixControls
get critical

Upload files to be criticized
http://perlcritic.com/

Perl Best Practices on Safari Books Online (LBNL network only)

Perl::Critic on CPAN, all the documentation you could want
http://search.cpan.org/~thaljef/Perl-Critic-1.118/lib/Perl/Critic.pm

how severe are your stylistic transgressions?
perlcritic my_code.pl
Packaging Perl

sigils looks better with a bow on
richards:Repositories richards$ module-starter --module=JGI::SynBio --author="Scott Whetzel" --email=SMRichardson@lbl.gov --builder='Module::Build'

Added to MANIFEST: Build.PL
Added to MANIFEST: Changes
Added to MANIFEST: ignore.txt
Added to MANIFEST: lib/JGI/SynBio.pm
Added to MANIFEST: MANIFEST
Added to MANIFEST: README
Added to MANIFEST: t/00-load.t
Added to MANIFEST: t/boilerplate.t
Added to MANIFEST: t/manifest.t
Added to MANIFEST: t/pod-coverage.t
Added to MANIFEST: t/pod.t
Created starter directories and files
richards:Repositories richards$
use 5.006;
use strict;
use warnings;
use Module::Build;
use File::Basename;
use Carp;

print "\n";
my $llt = 0;
my ($cpath, $spath, $tpath) = ( '', q{}, q{}, q{});
my ($dcpath, $dspath, $dtpath) = ('/etc/GeneDesign/', '/usr/local/bin', '/tmp');
my ($g, $se, $bl, $sv) = ( 0, 0, 0, 0);
my ($dg, $de, $dbl, $dv) = ('Y', 'Y', 'Y', 'Y');

my $check = eval

{ require Bio::GeneDesign::ConfigData; }

if ($check)

{ $GDB = Module::Build->new
  (
    module_name => 'Bio::GeneDesign',
    license => 'bsd',
    dist_author => q{Sarah Richardson <SMRichardson@lbl.gov>},
    dist_version => '5.00',
    dist_abstract => 'Functions for the design of synthetic genes',
    add_to_cleanup => [ 'Bio::GeneDesign-*' ],
    create_makefile_pl => 'traditional',
    build_requires =>
    {
      'Test::More' => 0.8,
      'Test::Deep' => 0.1,
    },
    requires =>
    {
      'perl' => 5.006,
      'List::Util' => 1.22,
      'Math::Combinatorics' => 0.09,
      'Text::Wrap' => 2009.0305,
      'Bio::Root::Root' => 0
    },
  );
}
#!/usr/bin/perl -T
use Test::More tests => 2;
use Bio::GeneDesign;
use Bio::Seq;
use strict;
use warnings;
my $GD = Bio::GeneDesign->new;
$GD->set_organism(-organism_name => "yeast",
                  -table_path => "codon_tables/Standard.ct",
                  -rscu_path => "codon_tables/yeast.rscu");
$GD->set_restriction_enzymes(-list_path => "enzymes/test");
my $orf = "ATGGACAGATCTTGGAGCGAAGCTGAACCGCAGACCCTGAGTGATGCTACCTGAGCCAGAAGGCTAGCGAGA";
my $seqobj = Bio::Seq->new(-seq => $orf, -id => "orf");

#TESTING define_sites
my $RES = {1};

my $TRES = $GD->enzyme_set;
is_deeply($TRES, $RES, "define_sites()" );

#TESTING define_site_status
my @enzes = values @TRES;
my $SITE_STATUS = {1};
my $TSITE_STATUS = $GD->restriction_status($seqobj);
is_deeply($TSITE_STATUS, $RES, "define_site_status()" );
where should configuration files be installed? [ /etc/GeneDesign/ ]
where should scripts be installed? [ /usr/local/bin ]
where should GeneDesign write tmp files? [ /tmp ]
enable GD::Graphics support? [ Y ]
enable EMBoss palindrome for hairpin detection? [ Y ]
enable BLAST+ for similarity detection? [ Y ]
enable wmatch for similarity detection? [ Y ]
configuration files will be in /etc/GeneDesign/
scripts will be in /usr/local/bin
will use BLAST+ executables found in /usr/local/bin/
will use wmatch executables found in /usr/local/wmatch/

Created MYMETA.yml and MYMETA.json
Creating new 'Build' script for 'Bio-GeneDesign' version '5.00'

richards:genedesign-dev richards$ sudo perl Build.PL
building Bio-GeneDesign
t/00-load.t ................... ok
t/01-basic.t .................. ok
t/02-codons.t ................ ok
t/03-reverse_translate.t .. ok
t/04-codon_juggle.t ......... ok
t/05-enzymes.t .............. ok
t/06-subtract_sequence.t .. ok
t/08-graph.t ................. ok
t/09-polindrome.t .......... Warning: integer value out of range 20 more than (reset to) 17
t/09-polindrome.t .......... ok
t/10-wmatch.t ............... ok
t/11-blast.t ................. ok
t/12-suffix.t ............... ok
t/13-random.t ............... ok
t/boilerplate.t ............. ok
t/manifest.t ................. skipped: Author tests not required for installation
t/pod-coverage.t ........... ok
t/pod.t ...................... ok
All tests successful.

Test Summary Report
---------------------
t/boilerplate.t (Wstat: 0 Tests: 22 Failed: 0)
  TODO passed: 1-22
Files=17, Tests=131, 38 wallclock secs ( 0.08 user 0.05 system 31.40 cpu 5.08 csys = 36.61 CPU)
Result: PASS

richards:genedesign-dev richards$ sudo ./Build install
building Bio-GeneDesign
installing /usr/local/share/man/man1/GD_Design_Building_Blocks.pl.1
installing /usr/local/share/man/man1/GD_Design_Building_Blocks_JGI.pl.1
installing /usr/local/share/man/man1/GD_Design_Building_Blocks_JGI.pl.1
installing /usr/local/share/man/man1/Run_Bio-GeneDesign_pl.1
installing /usr/local/share/man/man1/Run_Bio-GeneDesign_JGI.pl.1
installing /usr/local/share/man/man1/Build_Bio-GeneDesign_pl.1
installing /usr/local/share/man/man1/Build_Bio-GeneDesign_JGI.pl.1
Packaging Carrots

**version control:** manage releases and branches of an entire package, making it easy to refactor or upgrade multiple pieces of interacting code.

**ninja deploys:** a cron job can automatically pull the latest version of your code from a repository, build it, test it, and deploy it on genepool while you eat lunch or nap! And all you had to do was tweak the defaults for a build script.

**portability:** packaging frameworks encourage you to think about your code running on systems other than your own, reporting to people who aren’t you. This *will* make your code better.

**respectability:** packages look professional.
wrap it up

The module starter documentation
http://search.cpan.org/~xsawyerx/Module-Starter-1.60/lib/Module/Starter.pm

The Module::Build documentation
http://search.cpan.org/~leont/Module-Build-0.4003/lib/Module/Build.pm

ExtUtils::MakeMaker if you prefer
http://search.cpan.org/~mschwern/ExtUtils-MakeMaker-6.64/lib/ExtUtils/MakeMaker.pm

give genepool a present
module-starter --module=JGI::MeMeME --author="Yours Truly"\
   --email=donotreply@gmail.com --builder= "Module::Build"
Environment Modules

there is!

there has to be a better way

there is!
installing and running genedesign before environment modules

```
  git pull origin master
  perl Build.PL --install_base /jgi/tools/misc_bio/bioperl/bioperl-live/lib/perl5
  /jgi/tools/groups/synbio/bin/GD_Design_Oligos.pl --help
``` after environment modules

```
  git pull origin master
  perl Build.PL --like_last_time
  ./Build && ./Build test && ./Build install

  module load perl EMBOSS vmatch blast+ genedesign

  GD_Design_Oligos.pl --help
```