

Using Modules

Programming for Biology 2011

Why use modules?

- Sometimes you may want to use the same functions over and over again in different programs
- Bad way: Copy and paste
- Good way: Make a module
- There are also many many modules that other people have written that you can use!
- To use modules they must be properly installed and called with the “use” command

File::Basename

basename

- Input = long UNIX path name
 - i.e. '/bush_home/bush1/lstein/dna.fa'
- Output = file name
 - i.e. 'dna.fa'

• *dirname*

- Input = long UNIX path name
 - i.e. '/bush_home/bush1/lstein/dna.fa'
- Output = directory
 - '/bush_home/bush1/lstein/'

File::Basename

```
#!/usr/bin/perl  
# file: basename.pl
```



```
use strict;  
use File::Basename;
```

```
my $path = '/bush_home/bush1/lstein/dna.fa';  
my $base = basename($path);  
my $dir  = dirname($path);
```

```
print "The base is $base and the directory is $dir.\n";
```

Output:

```
The base is dna.fa and the directory is /bush_home/bush1/  
lstein.
```

Common error: Undefined subroutine &main::basename called at basename.pl
line 8.

Env

- This standard module imports a set of scalar variables that describe your environment

- \$HOME
- \$PATH
- \$USER

Env

```
#!/usr/bin/perl
# file env.pl

use strict;
use Env;

print "My home is $HOME\n";
print "My path is $PATH\n";
print "My username is $USER\n";
```

Output:

```
My home is /bush_home/bush1/lstein
My path is /bush_home/bush1/lstein/pfb2011
My username is lstein
```

Installed modules

- perldoc perlmodlib
- modules installed with basic perl installation
- <http://perldoc.perl.org/perlmodlib.html>
- perldoc perllocal
- Tells you modules that are installed on your machine

Installing modules manually

```
% tar zxvf bioperl-1.6.1.tar.gz
bioperl-1.6.1/
bioperl-1.6.1/Bio/
...

% perl Makefile.PL
Generated sub tests. go make show_tests to see available subtests
...
Writing Makefile for Bio

% make
cp Bio/Tools/Genscan.pm blib/lib/Bio/Tools/Genscan.pm
...
Manifying blib/man3/Bio::Location::CoordinatePolicyI.3
Manifying blib/man3/Bio::SeqFeature::Similarity.3

% make test
PERL_DL_NONLAZY=1 /net/bin/perl -Iblib/arch -Iblib/lib
-I/net/lib/perl5/5.6.1/i686-linux -I/net/lib/perl5/5.6.1 -e 'use
Test::Harness qw(&runtests $verbose); $verbose=0; runtests @ARGV;' t/*.t
t/AChange.....ok
...
All tests successful, 95 subtests skipped.
Files=60, Tests=1011, 35 wallclock secs (25.47 cusr + 1.60 csys = 27.07 CPU)

% make install
Installing /net/lib/perl5/site_perl/5.6.1/bioback.pod
Installing /net/lib/perl5/site_perl/5.6.1/biostart.pod
...
```


Installing Modules Using the CPAN Shell

Perl has a CPAN module installer built into it. You run it like this:

```
% cpan
```

```
cpan shell -- CPAN exploration and modules installation (v1.59_54)  
ReadLine support enabled
```

```
cpan>
```

From this shell, there are commands for searching for modules, downloading them, and installing them.

[The first time you run the CPAN shell, it will ask you a lot of configuration questions. Generally, you can just hit return to accept the defaults. The only trick comes when it asks you to select CPAN mirrors to download from. Choose any ones that are in your general area on the Internet and it will work fine.]

To search for a module:

```
cpan> i /Wrap/
```

```
Going to read /bush_home/bush1/lstein/.cpan/sources/authors/01mailrc.txt.gz
```

```
CPAN: Compress::Zlib loaded ok
```

```
Going to read /bush_home/bush1/lstein/.cpan/sources/modules/02packages.details.txt.gz
```

```
Database was generated on Tue, 16 Oct 2001 22:32:59 GMT
```

```
...
```

```
Module          Text::Wrap      (M/MU/MUIR/modules/Text-Tabs+Wrap-2001.0929.tar.gz)
```

```
...
```

```
41 items found
```

```
cpan> install Text::Wrap
```

```
Running install for module Text::Wrap
```

```
quit
```

Where are module installed?

Module files end with the extension `.pm`. If the module name is a simple one, like **Env**, then Perl will look for a file named **Env.pm**. If the module name is separated by `::` sections, Perl will treat the `::` characters like directories. So it will look for the module **File::Basename** in the file **File/Basename.pm**

Perl searches for module files in a set of directories specified by the Perl library path. This is set when Perl is first installed. You can find out what directories Perl will search for modules in by issuing **perl -V** from the command line:

```
% perl -V
Summary of my perl5 (revision 5.0 version 6 subversion 1) configuration:
 Platform:
   osname=linux, osvers=2.4.2-2smp, archname=i686-linux
 ...
 Compiled at Oct 11 2001 11:08:37
 @INC:
   /usr/lib/perl5/5.6.1/i686-linux
   /usr/lib/perl5/5.6.1
   ...
```

You can modify this path to search in other locations by placing the **use lib** command somewhere at the top of your script:

```
#!/usr/bin/perl

use lib '/home/lstein/lib';
use MyModule;
...
```

This tells Perl to look in **/home/lstein/lib** for the module **MyModule** before it looks in the usual places. Now you can install module files in this directory and Perl will find them.

Sometimes you really need to know where on your system a module is installed. **Perldoc** to the rescue again -- use the `-l` command-line option:

```
% perldoc -l File::Basename
/System/Library/Perl/5.8.8/File/Basename.pm
```

Making modules

Programming for Biology 2011

What is a module?



```
52 sub _get_description {
53     my $service = shift;
54     my $protocol = shift;
55
56     my $description;
57     if ($protocol) {
58         $description = " ($protocol";
59         if ($service) {
60             $description .= ", $service)";
61         }
62     } else {
63         $description .= ")";
64     }
65 }
66 return ($description);
67 }
```

```
4  ### The Central Dogma ###
5  #####
6
7  ### Genome
8  $DNA_seq = "ATGACCCTACTAGATCATCTATGATAGCTCAT";
9
10 ### Transcription
11 $RNA_seq = $DNA_seq;
12 $RNA_seq =~ s/T/U/gi;
13 print "$RNA_seq\n";
14
15 ### Translation
16 $position = 0;
17 while (substr $RNA_seq,$position,3) {
18     $codon = substr $RNA_seq,$position,3;
19     print translate_codon($codon);
20     $position = $position + 3;
21 }
22 sub translate_codon {
23     if ($_[0] =~ /GC./i) {return Ala;}
24     if ($_[0] =~ /UGC|UGU/i) {return Cys;}
```

continue

Module

```
package MySequence;  
#file: MySequence.pm  
  
use strict;  
our $EcoRI = 'ggatcc';
```

A package (or namespace) is an abstract **container** or **environment** created to hold a logical grouping of unique symbols (i.e., subroutines).

```
sub reverseq {  
    my $sequence = shift;  
    $sequence = reverse $sequence;  
    $sequence =~ tr/gatcGATC/ctagCTAG/;  
    return $sequence;  
}  
  
sub seqlen {  
    my $sequence = shift;  
    $sequence =~ s/[^gatcnGATCN]//g;  
    return length $sequence;  
}  
  
1;
```

A Perl module must end with a true value.

Script

```
#!/usr/bin/perl
#file: sequence.pl

use strict;
use warnings;
use MySequence;

my $sequence = 'gattccggatttccaaagggttcccaatttggg';
my $complement = MySequence::reverseq($sequence);

print "original = $sequence\n";
print "complement = $complement\n";
```

Must explicitly *qualify* each MySequence function by using the notation

MySequence::function_name

Module using Exporter

```
package MySequence;
#file: MySequence.pm

use strict;
use base 'Exporter';

our @EXPORT = qw(reverseseq seqlen);
our @EXPORT_OK = qw($EcoRI);
our $EcoRI = 'ggatcc';

sub reverseseq {
    my $sequence = shift;
    $sequence = reverse $sequence;
    $sequence =~ tr/gatcGATC/ctagCTAG/;
    return $sequence;
}

sub seqlen {
    my $sequence = shift;
    $sequence =~ s/[^gatcnGATCN]//g;
    return length $sequence;
}

1;
```

*

Script using Exporter

```
#!/usr/bin/perl
#file: sequence.pl

use strict;

use warnings;
use MySequence;

my $sequence = 'gattccggatttccaaagggttcccaatttggg';
my $complement = reverseseq($sequence);

print "original = $sequence\n";
print "complement = $complement\n";
```


Exporter - Implements default import method
for modules

```
use base 'Exporter';  
  
our @EXPORT = qw(reverseq seqlen);  
our @EXPORT_OK = qw($EcoRI);
```

use base 'Exporter' Tells Perl that this module is
a type of "Exporter" module

our @EXPORT = qw(reverseq seqlen) line tells
Perl to export the functions **reverseq** and **seqlen**
automatically.

Also, can export **qw(afunc \$scalar @array %hash);**

our @EXPORT_OK = qw(\$EcoRI) tells Perl that it
is OK for the user to import the \$EcoRI
variable, but not to export it automatically.

Getopt::Long - Extended processing of command line options

Command line operated programs traditionally take their arguments from the command line, for example filenames.

Besides arguments, these programs often take command line *options* as well. Options are not necessary for the program to work, hence the name 'option', but are used to modify its default behaviour.

Example:

```
courses:~ srynearson$ grep -i 'AGCG' > capture.txt
```

```
courses:~ srynearson$ perl GVF_Parser.pl -data file.txt
```

Script using Getopt::Long

```
#!/usr/bin/perl -w
```

```
use strict;  
use lib '/Users/srbio/GVF_DB_Variant/lib';  
use Utils;  
use GVF_DB_Connect;  
use IO::File;  
use GVF::DB::Variant;  
use Getopt::Long;
```

```
my $usage = "\n
```

DESCRIPTION:

Parsing script which takes gvf file and stores metadata and gvf line in data structures.

Options allow you to added to specific/all table in GVF_Variant database or none if just working with data structures.

```
USAGE:                ./Gvf_Parser.pl -option <GVF_file>
```

OPTIONS(required):

Each option corresponds to a table in the database.

-- all Option will add all areas of GVF file to database.

-- data Will print out the data structures to view.

```
\n";
```

```
my ($all, $data);  
my $input = $ARGV[1] || die $usage;
```

```
GetOptions(  
    'all' => \$all,  
    'data' => \$data,
```

```
) || die $usage;
```

*