Using Modules

v2 2012

Why use modules?

Sometimes you may want to use the same functions over and over again in different programs

Bad way: Copy and paste a subroutine

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. '/home/dave/dna.fa'

Output = file name
i.e. 'dna.fa'

dirname

Input = long UNIX path name
i.e. '/home/dave/dna.fa'

Output = directory
'/home/dave/'

Let's try it.

File::Basename

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

use strict;
use File::Basename;

my $path = '/home/dave/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 /home/dave.
```

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 /home/dave
My path is /home/dave/pfb2012
My username is dave
```

Which modules are installed?

`dave$ perldoc perlmodlib`

Which modules are installed with basic perl installation?

http://perldoc.perl.org/perlmodlib.html

`dave$ perldoc perllocal`

Which modules 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/AAChange.........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:

```
% sudo 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 modules 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

What is a module?

```perl
sub _get_description {
    my $service  = shift;
    my $protocol = shift;

    my $description;
    if ($protocol) {
        $description = " ($protocol";
        if ($service) {
            $description .= ", $service)";
        }
        else {
            $description .= ")";
        }
    }
    return ($description);
}
```

```perl
#### The Central Dogma ####
############################

### Genome
$DNA_seq = "ATGACCCTACTAGATCATCTATGATAGCTCAT";

### Transcription
$RNA_seq = $DNA_seq;
$RNA_seq =~ s/T/U/gi;
print "$RNA_seq\n";

### Translation
$position = 0;
while (substr $RNA_seq,$position,3) {
      $codon = substr $RNA_seq,$position,3;
      print translate_codon($codon);
      $position = $position + 3;
}
sub translate_codon {
      if ($_[0] =~ /GC./i) {return Ala;}
      if ($_[0] =~ /UGC|UGU/i) {return Cys;}
```

continue

Module

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

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

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

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;
```

*

Script using Exporter

```
#!/usr/bin/perl

# file: sequence.pl

use strict;

use warnings;
use MySequence;


my $sequence ='gattccggatttccaaagggttcccaatttggg';

my $complement = reverseq($sequence);


print "original = $sequence\n";

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

*

Exporter - Implements default import method for modules

```
use base 'Exporter';

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

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

our @EXPORT = qw(reverseseq seqlen) line tells Perl to export the functions **reverseseq** 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 behavior.

Example:

```
coursemain:~ dmessina$ grep -i 'AGCG' > capture.txt

coursemain:~ dmessina$ make_fake_fasta.pl --length 100
```

Script using Getopt::long

```
#!/usr/bin/env perl

use strict;
use warnings;

use Getopt::Long;
my $length = 30;
my $number = 10;
my $help;
GetOptions('l|length:i' => \$length,
           'n|number:i' => \$number,
           'h|help'     => \$help);

my $usage = "make_fake_fasta.pl - generate random DNA seqs

Options:
-n <number>   the number of sequences to make (default: 10)
-l <length>   the length of each sequence     (default: 30)
";
die $usage if $help;

my @nucs = qw(A C T G);


for (my $i = 1; $i <= $number; $i++) {
     my $seq;

     for (my $j = 1; $j <= $length; $j++) {
     my $index = int(rand (4));
     my $nuc = $nucs[$index];
     $seq .= $nuc;
     }
     print ">fake$i\n";
     print $seq, "\n";
}
```

*