Using Modules

Dave Messina

v5 2014

Why use modules?

Sometimes you may want to use the same subroutines 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

Using modules somebody else wrote

File::Basename

Subroutine: `basename`

Input: a UNIX path, like `/home/dave/dna.fa`
Output: just the file name (the last part of the path), like `dna.fa`

Subroutine: `dirname`

Input: a UNIX path, like `/home/dave/dna.fa`
Output: just the directory (everything before the basename), like `/home/dave/`

Using modules somebody else wrote

```
#!/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.
```

Another module somebody else wrote

This module comes with Perl. It imports a set of scalar variables that describe your environment, such as `$HOME`, `$PATH`, and `$USER`.

By adding `use Env;`, we can bring those variables into our script and access them just as if we had declared them in the script.

```
#!/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";
```

$HOME, $PATH, and $USER are not declared in this script!

Output:

```
My home is /home/pfbhome/dave
My path is /usr/local/bin:/bin:/usr/bin:/usr/local/sbin: ...
My username is dave
```

Which modules are installed?

```
$ perldoc perlmodlib
```

Which modules are installed with basic perl installation?

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

```
$ perldoc perllocal
```

Which modules are installed on your machine?

Setting up your Perl environment

Download this .bashrc file

```
$ cd ~
$ wget http://bit.ly/sample_bashrc_pfb2014
$ cat sample_bashrc >> .bashrc
$ source .bashrc
```

This should now be in your ~/.bashrc

```
# Perl setup
export PERL_LOCAL_LIB_ROOT="$HOME/perl5";
export PERL_MB_OPT="--install_base $HOME/perl5";
export PERL_MM_OPT="INSTALL_BASE=$HOME/perl5";
export PERL5LIB="$HOME/perl5/lib/perl5/x86_64-linux-gnu-thread-multi:$HOME/perl5/
lib/perl5:$PERL5LIB";
```

Installing modules manually

```
$ wget http://search.cpan.org/CPAN/authors/id/G/GL/GLASSCOCK/FASTAid-v0.0.4.tar.gz
$ tar zxvf FASTAid-v0.0.4.tar.gz
    x FASTAid-v0.0.4/
    x FASTAid-v0.0.4/Changes
    ...

    $ cd FASTAid-v0.0.4
    $ perl Makefile.PL
    Checking if your kit is complete...
    Looks good
    Writing Makefile for FASTAid

    $ make
    cp lib/FASTAid.pm blib/lib/FASTAid.pm
    Manifying blib/man3/FASTAid.3pm

    $ make test
    ERL_DL_NONLAZY=1 /usr/bin/perl "-MExtUtils::Command::MM" "-e" "test_harness(0,
    'blib/lib', 'blib/arch')" t/*.t
    t/FASTAid.t .. ok
    All tests successful.
    Files=1, Tests=11,  0 wallclock secs ( 0.02 usr  0.01 sys +  0.03 cusr  0.01 csys
    =  0.07 CPU)
    Result: PASS

    $ make install
    cp lib/FASTAid.pm blib/lib/FASTAid.pm
    Manifying blib/man3/FASTAid.3pm
    Installing /home/pfbhome/dave/perl5/lib/perl5/FASTAid.pm
    Installing /home/pfbhome/dave/perl5/man/man3/FASTAid.3pm
```

Installing Modules Using the CPAN Shell

```
% 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, you need to set one thing.

```
cpan> o conf prefs_dir /home/your_username/
cpan> o conf commit
```

cpan will also ask you a lot of configuration questions. Generally, you can just hit return to accept the defaults.

To search for a module:

```
cpan> i /Wrap/
Going to read '/Users/dave/.cpan/Metadata'
  Database was generated on Thu, 18 Oct 2012 12:07:03 GMT
...

Module  < Text::Wrap             (MUIR/modules/Text-Tabs+Wrap-2013.0523.tar.gz)
...
41 items found

cpan> install Text::Wrap
Running install for module Text::Wrap
...
```

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

Dave Messina

v4 2013

What is a module?

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

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

A module is an ***container*** which holds a collection of related code.

It allows you to use the code over and over again without copying and pasting.

Module

```
package MySequence;

# file: MySequence.pm

use strict;
use warnings;


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

use strict;
use warnings;
use MySequence;

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

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

This one line lets you use all the code in `MySequence`.

By default, to use subroutines from `MySequence`, you must explicitly *qualify* each `MySequence` function by using the notation `MySequence::function_name`

Module using Exporter

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

use strict;
use base 'Exporter';

our @EXPORT = qw(reverseq);
our @EXPORT_OK = qw(seqlen);

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 when `MySequence` exports reverseq

```
#!/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";
```

Now that `MySequence` exports `reverseq` automatically, you can use the `reverseq` subroutine without the `MySequence::` prefix.

`reverseq` is now is the same namespace as the main script, just as if it were defined in the same file.

`Exporter` — implements default import method for modules

```
use base 'Exporter';

our @EXPORT = qw(reverseq);
our @EXPORT_OK = qw(seqlen);
```

`use base 'Exporter'` tells Perl that this module is a type of "Exporter" module (more about this in a future lecture).

`our @EXPORT = qw(reverseq)` tells Perl to export the subroutine `reverseq` automatically.

`our @EXPORT_OK = qw(seqlen)` tells Perl that it is OK for the user to import the `seqlen` subroutine, but not to export it automatically.

Also, you can export variables along with subroutines:
`our @EXPORT = qw(reverseq seqlen $scalar @array %hash);`

If I make a module, where should I put it?

Once you've made your own module, you will want to put it somewhere Perl knows to look.

```
$ printenv PERL5LIB
```

`Getopt::Long` - Extended processing of command line options

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

These programs often take *named* command line arguments, so that the order in which you write arguments doesn't matter and so that it's clear which argument does what.

```
$ grep -i ‘AGCG’ > capture.txt

$ make_fake_fasta.pl --length 100
```

By convention, single-letter arguments are prefixed with one dash -, and full-word arguments are prefixed with two dashes (--).

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";
}
```

*