The LXF Guide: Write a Perl module

Articles / LXF magazine
Posted by M-Saunders on Oct 08, 2008 - 12:16 PM

Looking to make your Perl code more re-usable and easier to maintain? **Juliet Kemp** explains how to create, build and install your own Perl modules, and it's not as difficult as you might think...

A Perl module is a self-contained piece of Perl code that can be used by other modules or Perl programs. It has a unique name - Perl has a hierarchical namespace for modules to reduce collisions, so there are modules like Math::Complex, Net::SMTP, and so on. You can check out ***CPAN, the Comprehensive Perl Archive Network*** **[1]** for a list of all available modules and to get an idea of the namespace structure. Modules all end with a `.pm` extension.

The major advantage of modules is that they make your code more reusable. It's not much more work to create a module than it is just to hack together a one-time script, and you'll save yourself significant work in the future if the problem you're solving is ever likely to crop up again. Modules are also more maintainable, which is another way of helping out your future self!

Creating a module

Setting up the module

Start off by creating a development directory for your perl module. cd into this directory, then use the h2xs tool. This creates a structure for a Perl module from a C library `.h` file; but it's become the standard tool even if you're not starting from a `.h` file. For now, let's just set up a test module to demonstrate the principles.

```
h2xs -X -n Acme::Test
```

This creates and populates an `Acme-Test` directory (the `-X` switch says that it's not related to any C code, and `-n` specifies the module name).

Take a look at the contents of this new directory. The `MANIFEST` file lists all the files in the module - mostly to ensure that modules are received intact. `README` gives you a README skeleton - filling this in is optional but good practice. `t/Acme-Test.t` is a test directory and a skeleton test routine. `Makefile.pl` is a perl program to handle building the module, and `lib/Acme/Test.pm` is the actual perl module, again with some skeleton code and documentation.

To build the module, there are 4 steps:

```
perl Makefile.pl
make
make test
make install
```

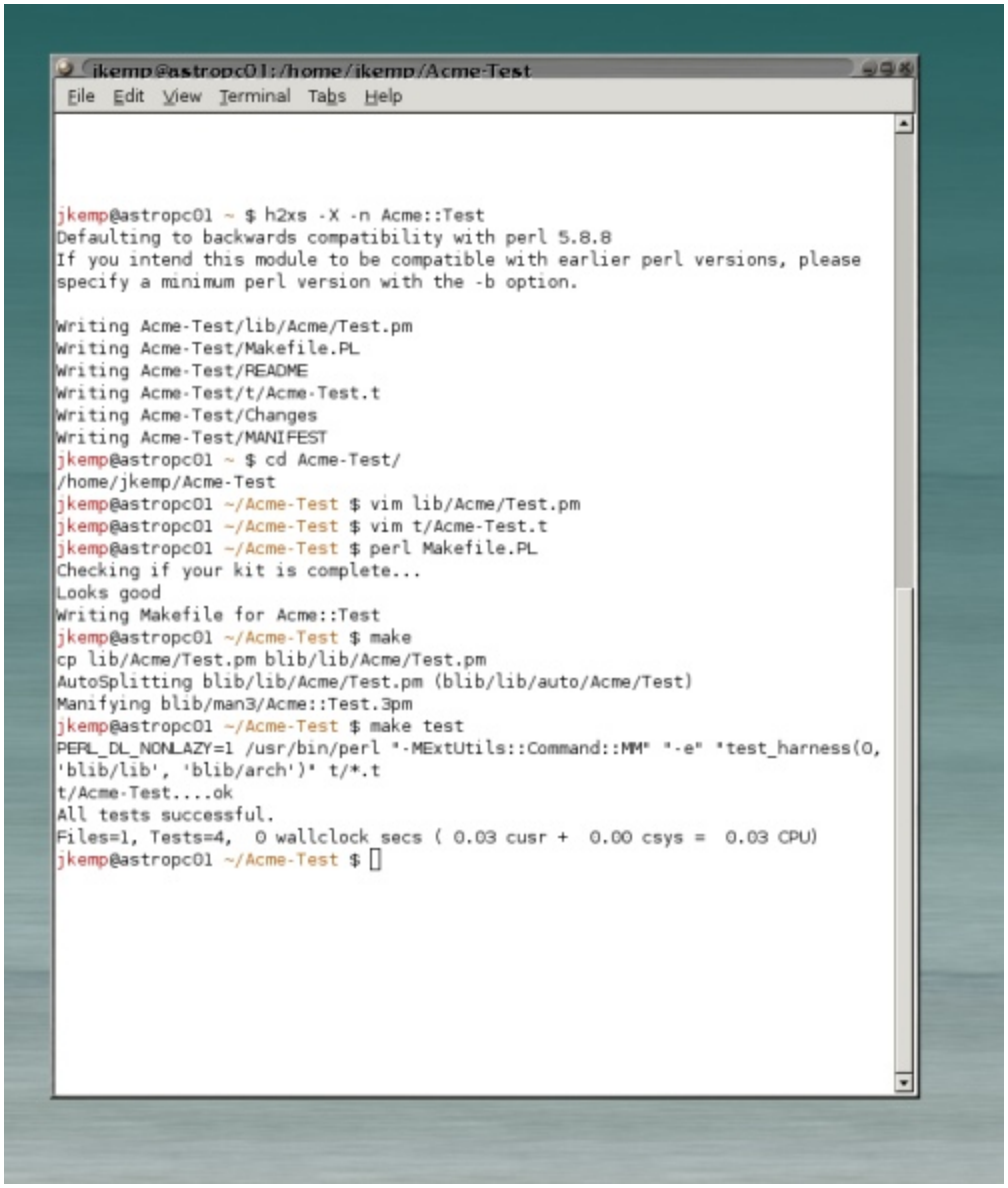
The make install line installs your module centrally on the system (you'll usually need root access for this), so don't do this until you're sure it works!

Documenting your module

Ideally, you should document your module before you write it - because you should know what it's going to do! `perldoc`, which uses the `pod` format, is excellent for documentation ease-of-use: your skeleton module will include some skeleton documentation code for you to fill in. `perldoc perlpod` will give you more information about POD.

The one critical documentation section that you need to create yourself is FUNCTIONS or METHODS (if your code is function-based or object-oriented). Here, you should document every method/function intended for public use - at the least, state what parameters the method takes and what return values it gives back. The aim is for someone to be able to use the module purely by looking at the documentation, rather than needing to examine the code.

To read the documentation after install, type **perldoc Acme::Test**.



[2]

Creating and making the module (click for bigger)

Writing your module

When you come to actually write your code, you'll notice that there's a bunch of stuff already there. This includes `use strict`, and some statements about AutoLoader and Exporter that enable modules to be used from other pieces of code. Your documentation is right at the end of the file.

Rather than get into the murky world of object-oriented code, let's stick with a functional module. This means we don't need a new method, so we can just write a single method that works.

```
# Preloaded methods go here.

sub test($) {
  my $string = $_[0];
  return "String is $string\n";
```

```
}
```

Now you've got a module with one method, so it's time to test it.

Testing your module

Have a look at the `t/Acme-Test.t` file. We're going to set up 4 tests, so change the line

```
use Test::More test => 1;
```

to

```
use Test::More tests => 4
```

We'll first add a couple of tests to check that the module loads OK:

```
# This first line will already be in the file
BEGIN { use_ok( 'Acme::Test' ) };
require_ok( 'Acme::Test' );
```

These check that you can `use` and `require` the module OK. Now let's test the method output: first that there is any output at all, and then that it is what we expect.

```
my $result = Acme::Test::test("testing!");
ok( (defined($result)), "Result is defined and is $result");
ok( ($result eq "String is testing!\n"), "Result is as expected and is
$result");
```

To run the test, go to the top of your module directory and type `perl Makefile.PL; make; make test`. If everything goes OK, you'll just be told that everything passed. If a test failed, then the string after the comma in `ok ( test, output );` will be used to identify the failed test. In this case, I've put the `$result` value in this identification string to make it easier to find out what the problem is if something does go wrong.

Installing and using your module

Finally, you need to install your module somewhere where the Perl interpreter will look for it. To find out the contents of `;@INC`, type `perl -v`. Your modules want to be installed under the appropriate sub-directory in the `site_perl` directory - on my system this is `/usr/local/lib/site_perl`, so I would install the Math::Complex module in `/usr/local/lib/site_perl/Math/Complex.pm`.

`make install` will do this for you, but it may not install it where you want. To specify an install directory, type

```
perl Makefile.PL INSTALL_BASE=/install/dir
make install
```

If this is somewhere that isn't in `;@INC`, you'll need to set the `PERL5LIB` variable to include `/install/dir`.

To use your module in another piece of code, you'll need something a bit like this:

```
use Acme::Test;
```

```
my $newacme = new Acme::Test;
print $newacme->test("hello world");
```

CPAN

One final note: if you want to share your module with other people (probably not worth it in the case of this test module!) you can upload it to ***CPAN*** **[3]**, the Comprehensive Perl Archive Network. Check the FAQ for upload instructions. You can also of course download other people's modules - there is an absolutely enormous amount of stuff available on CPAN, and it's well worth checking out - although be warned that since absolutely anyone can upload, it's not guaranteed to be any good!

Links in this article
[1] http://www.cpan.org
[2] http://linuxformat.co.uk/blog/wp-content/perlmodules.png
[3] http://www.cpan.org