NAME

perlnewmod - preparing a new module for distribution

DESCRIPTION

This document gives you some suggestions about how to go about writing Perl modules, preparing them for distribution, and making them available via CPAN.

One of the things that makes Perl really powerful is the fact that Perl hackers tend to want to share the solutions to problems they've faced, so you and I don't have to battle with the same problem again.

The main way they do this is by abstracting the solution into a Perl module. If you don't know what one of these is, the rest of this document isn't going to be much use to you. You're also missing out on an awful lot of useful code; consider having a look at *perlmod*, *perlmodlib* and *perlmodinstall* before coming back here.

When you've found that there isn't a module available for what you're trying to do, and you've had to write the code yourself, consider packaging up the solution into a module and uploading it to CPAN so that others can benefit.

Warning

We're going to primarily concentrate on Perl-only modules here, rather than XS modules. XS modules serve a rather different purpose, and you should consider different things before distributing them - the popularity of the library you are gluing, the portability to other operating systems, and so on. However, the notes on preparing the Perl side of the module and packaging and distributing it will apply equally well to an XS module as a pure-Perl one.

What should I make into a module?

You should make a module out of any code that you think is going to be useful to others. Anything that's likely to fill a hole in the communal library and which someone else can slot directly into their program. Any part of your code which you can isolate and extract and plug into something else is a likely candidate.

Let's take an example. Suppose you're reading in data from a local format into a hash-of-hashes in Perl, turning that into a tree, walking the tree and then piping each node to an Acme Transmogrifier Server.

Now, quite a few people have the Acme Transmogrifier, and you've had to write something to talk the protocol from scratch - you'd almost certainly want to make that into a module. The level at which you pitch it is up to you: you might want protocol-level modules analogous to *Net::SMTP* which then talk to higher level modules analogous to *Mail::Send*. The choice is yours, but you do want to get a module out for that server protocol.

Nobody else on the planet is going to talk your local data format, so we can ignore that. But what about the thing in the middle? Building tree structures from Perl variables and then traversing them is a nice, general problem, and if nobody's already written a module that does that, you might want to modularise that code too.

So hopefully you've now got a few ideas about what's good to modularise. Let's now see how it's done.

Step-by-step: Preparing the ground

Before we even start scraping out the code, there are a few things we'll want to do in advance.

Look around

Dig into a bunch of modules to see how they're written. I'd suggest starting with *Text::Tabs*, since it's in the standard library and is nice and simple, and then looking at something a little more complex like *File::Copy*. For object oriented code, `WWW::Mechanize` or the `Email::*`

modules provide some good examples.

These should give you an overall feel for how modules are laid out and written.

Check it's new

There are a lot of modules on CPAN, and it's easy to miss one that's similar to what you're planning on contributing. Have a good plough through the *http://search.cpan.org* and make sure you're not the one reinventing the wheel!

Discuss the need

You might love it. You might feel that everyone else needs it. But there might not actually be any real demand for it out there. If you're unsure about the demand your module will have, consider sending out feelers on the `comp.lang.perl.modules` newsgroup, or as a last resort, ask the modules list at `modules@perl.org`. Remember that this is a closed list with a very long turn-around time - be prepared to wait a good while for a response from them.

Choose a name

Perl modules included on CPAN have a naming hierarchy you should try to fit in with. See *perlmodlib* for more details on how this works, and browse around CPAN and the modules list to get a feel of it. At the very least, remember this: modules should be title capitalised, (This::Thing) fit in with a category, and explain their purpose succinctly.

Check again

While you're doing that, make really sure you haven't missed a module similar to the one you're about to write.

When you've got your name sorted out and you're sure that your module is wanted and not currently available, it's time to start coding.

Step-by-step: Making the module

Start with *module-starter* or *h2xs*

The *module-starter* utility is distributed as part of the *Module::Starter* CPAN package. It creates a directory with stubs of all the necessary files to start a new module, according to recent "best practice" for module development, and is invoked from the command line, thus:

```
module-starter --module=Foo::Bar \
   --author="Your Name" --email=yourname@cpan.org
```

If you do not wish to install the *Module::Starter* package from CPAN, *h2xs* is an older tool, originally intended for the development of XS modules, which comes packaged with the Perl distribution.

A typical invocation of *h2xs* for a pure Perl module is:

```
h2xs -AX --skip-exporter --use-new-tests -n Foo::Bar
```

The `-A` omits the Autoloader code, `-X` omits XS elements, `--skip-exporter` omits the Exporter code, `--use-new-tests` sets up a modern testing environment, and `-n` specifies the name of the module.

Use *strict* and *warnings*

A module's code has to be warning and strict-clean, since you can't guarantee the conditions that it'll be used under. Besides, you wouldn't want to distribute code that wasn't warning or strict-clean anyway, right?

Use *Carp*

The *Carp* module allows you to present your error messages from the caller's perspective; this gives you a way to signal a problem with the caller and not your module. For instance, if you say this:

```
    warn "No hostname given";
```

the user will see something like this:

```
    No hostname given at
/usr/local/lib/perl5/site_perl/5.6.0/Net/Acme.pm
    line 123.
```

which looks like your module is doing something wrong. Instead, you want to put the blame on the user, and say this:

```
    No hostname given at bad_code, line 10.
```

You do this by using *Carp* and replacing your `warn`s with `carp`s. If you need to `die`, say `croak` instead. However, keep `warn` and `die` in place for your sanity checks - where it really is your module at fault.

Use *Exporter* - wisely!

Exporter gives you a standard way of exporting symbols and subroutines from your module into the caller's namespace. For instance, saying `use Net::Acme qw(&frob)` would import the `frob` subroutine.

The package variable `@EXPORT` will determine which symbols will get exported when the caller simply says `use Net::Acme` - you will hardly ever want to put anything in there. `@EXPORT_OK`, on the other hand, specifies which symbols you're willing to export. If you do want to export a bunch of symbols, use the `%EXPORT_TAGS` and define a standard export set - look at *Exporter* for more details.

Use *plain old documentation*

The work isn't over until the paperwork is done, and you're going to need to put in some time writing some documentation for your module. `module-starter` or `h2xs` will provide a stub for you to fill in; if you're not sure about the format, look at *perlpod* for an introduction. Provide a good synopsis of how your module is used in code, a description, and then notes on the syntax and function of the individual subroutines or methods. Use Perl comments for developer notes and POD for end-user notes.

Write tests

You're encouraged to create self-tests for your module to ensure it's working as intended on the myriad platforms Perl supports; if you upload your module to CPAN, a host of testers will build your module and send you the results of the tests. Again, `module-starter` and `h2xs` provide a test framework which you can extend - you should do something more than just checking your module will compile. *Test::Simple* and *Test::More* are good places to start when writing a test suite.

Write the README

If you're uploading to CPAN, the automated gremlins will extract the README file and place that in your CPAN directory. It'll also appear in the main *by-module* and *by-category* directories if you make it onto the modules list. It's a good idea to put here what the module actually does in detail, and the user-visible changes since the last release.

Step-by-step: Distributing your module

Get a CPAN user ID

Every developer publishing modules on CPAN needs a CPAN ID. Visit `http://pause.perl.org/`, select "Request PAUSE Account", and wait for your request to be approved by the PAUSE administrators.

`perl Makefile.PL; make test; make dist`

Once again, `module-starter` or `h2xs` has done all the work for you. They produce the

standard `Makefile.PL` you see when you download and install modules, and this produces a Makefile with a `dist` target.

Once you've ensured that your module passes its own tests - always a good thing to make sure - you can `make dist`, and the Makefile will hopefully produce you a nice tarball of your module, ready for upload.

Upload the tarball

The email you got when you received your CPAN ID will tell you how to log in to PAUSE, the Perl Authors Upload SErver. From the menus there, you can upload your module to CPAN.

Announce to the modules list

Once uploaded, it'll sit unnoticed in your author directory. If you want it connected to the rest of the CPAN, you'll need to go to "Register Namespace" on PAUSE. Once registered, your module will appear in the by-module and by-category listings on CPAN.

Announce to clpa

If you have a burning desire to tell the world about your release, post an announcement to the moderated `comp.lang.perl.announce` newsgroup.

Fix bugs!

Once you start accumulating users, they'll send you bug reports. If you're lucky, they'll even send you patches. Welcome to the joys of maintaining a software project...

AUTHOR

Simon Cozens, `simon@cpan.org`

Updated by Kirrily "Skud" Robert, `skud@cpan.org`

SEE ALSO

perlmod, *perlmodlib*, *perlmodinstall*, *h2xs*, *strict*, *Carp*, *Exporter*, *perlpod*, *Test::Simple*, *Test::More* *ExtUtils::MakeMaker*, *Module::Build*, *Module::Starter* http://www.cpan.org/ , Ken Williams' tutorial on building your own module at http://mathforum.org/~ken/perl_modules.html