NAME

ExtUtils::MakeMaker::Tutorial - Writing a module with MakeMaker

SYNOPSIS

```
use ExtUtils::MakeMaker;

WriteMakefile(
    NAME            => 'Your::Module',
    VERSION_FROM    => 'lib/Your/Module.pm'
);
```

DESCRIPTION

This is a short tutorial on writing a simple module with MakeMaker. Its really not that hard.

The Mantra

MakeMaker modules are installed using this simple mantra

```
perl Makefile.PL
make
make test
make install
```

There are lots more commands and options, but the above will do it.

The Layout

The basic files in a module look something like this.

```
Makefile.PL
MANIFEST
lib/Your/Module.pm
```

That's all that's strictly necessary. There's additional files you might want:

```
lib/Your/Other/Module.pm
t/some_test.t
t/some_other_test.t
Changes
README
INSTALL
MANIFEST.SKIP
bin/some_program
```

Makefile.PL

When you run Makefile.PL, it makes a Makefile. That's the whole point of MakeMaker. The Makefile.PL is a simple program which loads ExtUtils::MakeMaker and runs the WriteMakefile() function to generate a Makefile.

Here's an example of what you need for a simple module:

```
use ExtUtils::MakeMaker;

WriteMakefile(
    NAME            => 'Your::Module',
    VERSION_FROM    => 'lib/Your/Module.pm'
);
```

NAME is the top-level namespace of your module. VERSION_FROM is the file which contains the $VERSION variable for the entire distribution. Typically this is the same as your top-level module.

MANIFEST

A simple listing of all the files in your distribution.

```
Makefile.PL
MANIFEST
lib/Your/Module.pm
```

File paths in a MANIFEST always use Unix conventions (ie. /) even if you're not on Unix.

You can write this by hand or generate it with 'make manifest'.

See *ExtUtils::Manifest* for more details.

lib/

This is the directory where your .pm and .pod files you wish to have installed go. They are layed out according to namespace. So Foo::Bar is *lib/Foo/Bar.pm*.

t/

Tests for your modules go here. Each test filename ends with a .t. So *t/foo.t*/ 'make test' will run these tests. The directory is flat, you cannot, for example, have t/foo/bar.t run by 'make test'.

Tests are run from the top level of your distribution. So inside a test you would refer to ./lib to enter the lib directory, for example.

Changes

A log of changes you've made to this module. The layout is free-form. Here's an example:

```
1.01 Fri Apr 11 00:21:25 PDT 2003
    - thing() does some stuff now
    - fixed the wiggy bug in withit()

1.00 Mon Apr  7 00:57:15 PDT 2003
    - "Rain of Frogs" now supported
```

README

A short description of your module, what it does, why someone would use it and its limitations. CPAN automatically pulls your README file out of the archive and makes it available to CPAN users, it is the first thing they will read to decide if your module is right for them.

INSTALL

Instructions on how to install your module along with any dependencies. Suggested information to include here:

```
any extra modules required for use
the minimum version of Perl required
if only works on certain operating systems
```

MANIFEST.SKIP

A file full of regular expressions to exclude when using 'make manifest' to generate the MANIFEST. These regular expressions are checked against each file path found in the distribution (so you're matching against "t/foo.t" not "foo.t").

Here's a sample:

```
~$          # ignore emacs and vim backup files
```

```
.bak$       # ignore manual backups
\#          # ignore CVS old revision files and emacs temp files
```

Since # can be used for comments, # must be escaped.

MakeMaker comes with a default MANIFEST.SKIP to avoid things like version control directories and backup files. Specifying your own will override this default.

bin/

SEE ALSO

perlmodstyle gives stylistic help writing a module.

perlnewmod gives more information about how to write a module.

There are modules to help you through the process of writing a module: *ExtUtils::ModuleMaker*, *Module::Install*, *PAR*