

```
my @ftree = map { # add to parent's Child list unless ( $found );
    s/\s+$/;/ push CPANdeps }->{'Children'} }, $node;
my $depth = s/\s+$/;/ # number of indent blocks
$depth++; # start (one thought rather than 0)
$parent_stack[ $depth ] = $node;
while ( $found -> ( $parent_stack[ $depth ] = $node;
# create node structure $parent, $stack[ $depth ] = $node;
```



# CPANdeps

## Do you know your dependencies?

by David Cantrell  
david@cantrell.org.uk

One of Perl’s strongest selling points is the Comprehensive Perl Archive Network (CPAN). Having a vast library of freely-reusable code available in one place, having good quality tools for downloading and installing it, and looking at the documentation is enough for me to love Perl even if the language itself were as much of a pain as C or C++.

However, the quality of the code on the CPAN is decidedly variable. And while the mechanisms I have for specifying and fetching prerequisites (e.g. using `PREREQ_PM` in `ExtUtils::MakeMaker`) make it really easy to re-use the good bits of the CPAN, they also make it distressingly easy to re-use some real rubbish.

CPANdeps, <http://cpandeps.cantrell.org.uk/>, is a web-based tool to let you easily sift the good from the bad, and help module authors choose their dependencies.

### ■ Why CPANdeps exists -----

Like many of my Perlsh projects, CPANdeps started as a discussion on a mailing list. Someone was grumbling about how Perl authors seem to go mad with dependencies and so when I try to install a module, it fails:

```
$ perl -MCPAN -e 'install qw(Some::Module) '
[snip 1978 lines of output]
Tests succeeded but one dependency not OK
(Other::Module)
  FOO/Other-Module-2.078.tar.gz
  [dependencies] -- NARunning make install
  make test had returned bad status, won't
  install without force
```

It’s all too easy for the installation to fail deep down in the dependency tree because something didn’t work. “What do you mean we go mad with dependencies?” I thought. I rarely have more than two or three in my `Makefile.PL`. But then when I started looking a bit deeper, I found that those in turn had dependencies, which had dependencies; it’s really easy to have ten or more pre-requisites, all of which must work. Of course, they all *do* work on my development machine and my testing boxes, but I can’t test them in all possible environments. The end result is that sometimes someone who doesn’t do Perl for a living—one of our long-bearded sysadmin brethren, for example—will try to install some of my software, have problems with a dependency, and decide to use something else. That’s not a problem; I don’t care if he doesn’t use my software. But it does become a problem when he goes on to say “Installing Perl

software is so hard; I’ll avoid it in future. And I’ll recommend to management that our developers don’t use Perl either because it’s so hard to get working.”

I decided to write a little tool to help authors figure out what’s good to depend on and what’s not. It pulls together `CPAN.pm`, <http://search.cpan.org/> and the CPAN Testers database <http://testers.cpan.org> to figure out what depends on what, find all of their test results, and can then give an indication of how good (or bad) an idea it is to depend on a particular module.

If I were one of those painfully trendy Web 2.0 types, I’d call it a “mash-up”. But I’m not, so I won’t. I’d appreciate it if you don’t either. It’s just what it is.

### ■ How it works -----

First, I have to translate a module name into the distribution in which it lives. If I’m interested in `File::Spec`, for example, that lives in the `PathTools` distribution. `CPAN.pm` knows all about this, and exposes some of its guts to programmers. The documentation isn’t particularly clear, so I also wrote the beginnings of a `CPAN.pm` “cookbook”, which is distributed with recent versions of the module as `CPAN::API::HOWTO`. If I’ve already examined the distribution, I skip to the next one.

Next I take the distribution name and turn it into a URL for the distribution’s `META.yml` file. In the case of `File::Spec` that would be <http://search.cpan.org/src/KWILLIAMS/PathTools-3.25/META.yml>, which I fetched using `LWP`. If there’s no `META.yml`, we warn the user, otherwise I parsed it to get a list of modules, each of which gets the same treatment in turn, recursing through the tree of dependencies.

Finally, for each module, I look up the test results in a local copy of the CPAN Testers database, and count up the number of passes and fails. Earlier versions instead screen-scraped the CPAN Testers web site (e.g. <http://cpantesters.perl.org/show/PathTools.html>). This was both horribly slow as I had to download a web page for every single module in the tree (potentially very large pages too) and error-prone; it would, for example, return test results for a developer release of a module instead of for a stable release that `CPAN.pm` would try to install.

Once I gather all the data, I display the results for the user, with a tree of modules on the left, coloured bars showing green for test passes, red for fails, and yellow for unknowns in the middle, and then the actual number of pass/fail/unknown results on the right. At the bottom there’s a grand total which attempts to show the likelihood of the entire dependency tree working. I calculate this by multiplying together the probabilities of success for all the modules in the tree.