COLUMNS

Practical Perl Tools

Oh Say Can You CPAN?

DAVID N. BLANK-EDELMAN

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs. David is honored to have been the recipient of the 2009 SAGE Outstanding Achievement Award and to serve on the USENIX Board of Directors beginning in June of 2010.
dnb@ccs.neu.edu

Every once in a while I get asked to join a conference panel about scripting languages. It will be me, a Pythonista, a Rubyist, and a PHP developer (do they have a cute name?) all onstage together. In most cases, I think the organizers are hoping for the equivalent of a steel cage match in professional wrestling—the fewer participants standing at the end, the better. In these scenarios, I'm almost always a disappointment because I come to praise the other languages, not to bury them. I have a deep appreciation for the other languages, and I'm not afraid to state it even while I'm representing Perl. One of the key reasons I can say "I dig all of the other languages, but I choose to stick with Perl most of the time" is CPAN. This column will focus on CPAN, how to cope with both its triumphs and shortfalls, and some of the ways to interact with it that you may not have encountered before. There probably won't be any code in this issue's column but that's okay because you'll be learning ways to have other people write Perl code for you. We're going to focus on how to consume content from CPAN; discussion about how to contribute to it will have to wait for a future column.

What Is CPAN and How Do I Get Me Some?

I would be really surprised if there are Perl programmers who have never heard of CPAN, but I've been surprised before so pardon me as I go over the basics. CPAN is short for the Comprehensive Perl Archive Network. This is a massive repository of Perl code (largely modules meant for use in other people's code) that has been online since 1995 or so. How massive? As of this writing, cpan.org says:

The Comprehensive Perl Archive Network (CPAN) currently has 138,392 Perl modules in 30,406 distributions, written by 11,739 authors, mirrored on 254 servers.

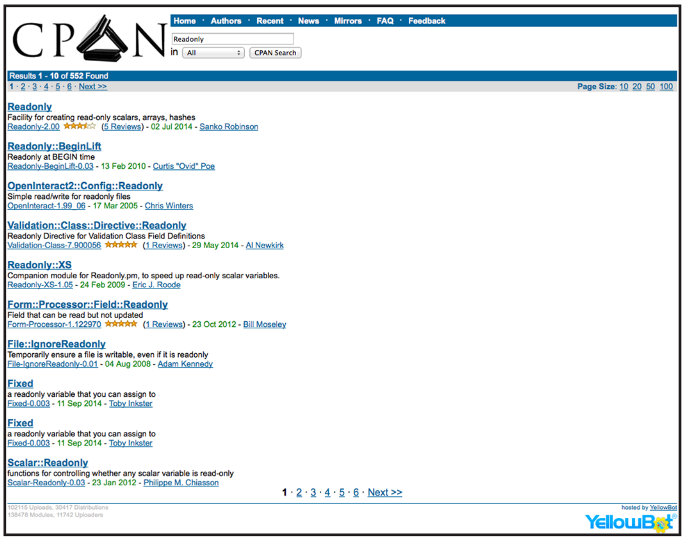
All of this code has been uploaded so other people may make use of it, so it can be a tremendous resource. Any time you have a problem or a task that sounds like someone else may have solved it, it always behooves you to search CPAN first. We'll talk about ways to do this in a moment.

The plus of having such a massive store of donated code to draw upon is that you often can find someone else has already written (almost) exactly what you need. The minus of having this massive store is some percentage of it is (to be charitable) duplicated effort, and (to be less charitable) some of it is crap. I'll offer tips about this problem later in this column as well.

Deep CPAN Diving

So how do you find what is available on CPAN? Many people start with the search.cpan.org engine. This Googley-looking search engine returns a page like the one in Figure 1.

An experienced CPAN spelunker will scan the returned list of modules and look not just at the description to determine whether a module is appropriate for the task at hand, but also at the metadata. For example, has the module been updated recently? Does it have any reviews



Results 1 - 10 of 552 Found
1 · 2 · 3 · 4 · 5 · 6 · Next >> Page Size: 10 20 50 100

Readonly
Facility for creating read-only scalars, arrays, hashes
Readonly-2.00 (5 Reviews) - 02 Jul 2014 - Sanko Robinson

Readonly::BeginLift
Readonly at BEGIN time
Readonly-BeginLift-0.03 - 13 Feb 2010 - Curtis "Ovid" Poe

OpenInteract2::Config::Readonly
Simple read/write for readonly files
OpenInteract-1.99_06 - 17 Mar 2005 - Chris Winters

Validation::Class::Directive::Readonly
Readonly Directive for Validation Class Field Definitions
Validation-Class-7.900056 (1 Reviews) - 29 May 2014 - Al Newkirk

Readonly::XS
Companion module for Readonly.pm, to speed up read-only scalar variables.
Readonly-XS-1.05 - 24 Feb 2009 - Eric J. Roode

Form::Processor::Field::Readonly
Field that can be read but not updated
Form-Processor-1.122970 (1 Reviews) - 23 Oct 2012 - Bill Moseley

File::IgnoreReadonly
Temporarily ensure a file is writable, even if it is readonly
File-IgnoreReadonly-0.01 - 04 Aug 2008 - Adam Kennedy

Fixed
a readonly variable that you can assign to
Fixed-0.003 - 11 Sep 2014 - Toby Inkster

Fixed
a readonly variable that you can assign to
Fixed-0.003 - 11 Sep 2014 - Toby Inkster

Scalar::Readonly
functions for controlling whether any scalar variable is read-only
Scalar-Readonly-0.03 - 23 Jan 2012 - Philippe M. Chiasson

1 · 2 · 3 · 4 · 5 · 6 · Next >>

102115 Uploads, 30417 Distributions
138478 Modules, 11742 Uploaders
hosted by YellowBot

Figure 1: A screenshot of search.cpan.org showing some results

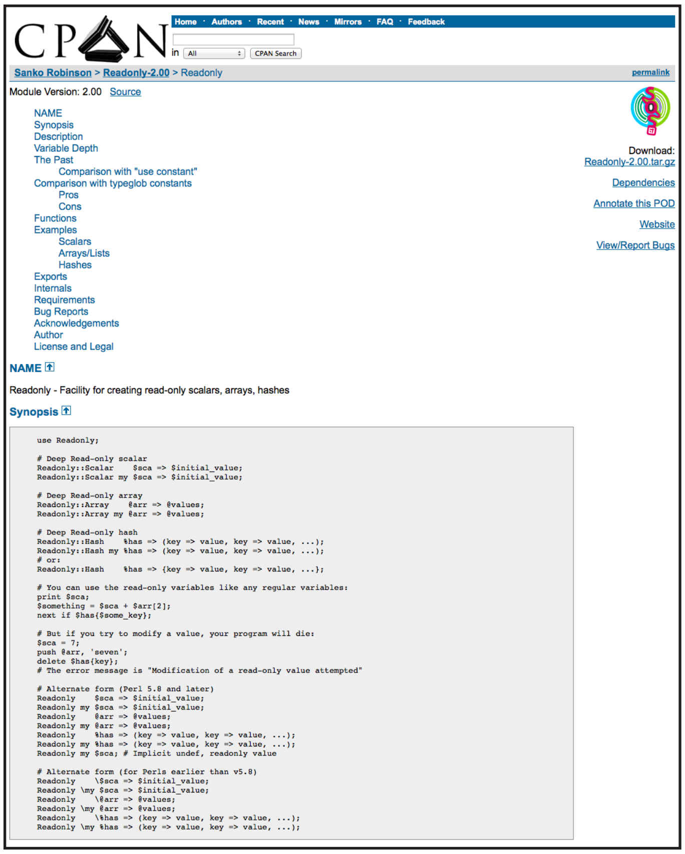
and are they positive? Is the module part of a distribution I recognize? Is the module author well known in the Perl community?, etc.

When you click on the module name, you'll be greeted with the documentation for that module. More often than not, I will click on the breadcrumb link that brings me to the page for the whole distribution (Figure 2).

I do this for two reasons: First, I often want to poke around in a module's code (especially looking at the test code in it for examples of how to use the module). This can be done from the Browse link. Second, I might be curious about bugs filed against the module ("View/Report Bugs") or what other modules this module depends on ("Dependencies"—we'll talk more about that soon). Some of these links can be reached from the search results or the first page linked off the search results, but I'm so used to using the Browse link that going to the distribution page is habitual at this point.

Another way you can search for modules on CPAN is to use metacpan.org. MetaCPAN attempts to be an even spiffier search engine. Figure 3 shows the same search from before, this time run at MetaCPAN.org.

First, let's talk about what is spiffier on the service. When I first started typing "Readonly" into the search box, it attempted to auto-complete my query. Next, not only are Readonly and Readonly::XS next to each other, but at the bottom of the results you can see MetaCPAN has bunched together related modules in a distribution. When I click on the first module link on this page, I see the page that begins like Figure 4.



Sanko Robinson > Readonly-2.00 > Readonly permalink

Module Version: 2.00 Source

NAME
Synopsis
Description
Variable Depth
The Past
    Comparison with "use constant"
Comparison with typeglob constants
    Pros
    Cons
Functions
Examples
    Scalars
    Arrays/Lists
    Hashes
Exports
Internals
Requirements
Bug Reports
Acknowledgements
Author
License and Legal

Download:
Readonly-2.00.tar.gz
Dependencies
Annotate this POD
Website
View/Report Bugs

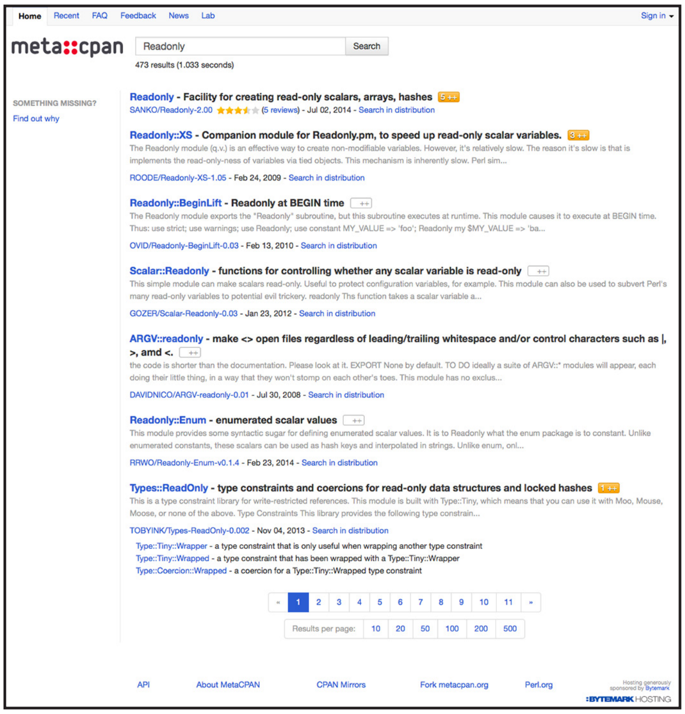
NAME

Readonly - Facility for creating read-only scalars, arrays, hashes

Synopsis

```
    use Readonly;

    # Deep Read-only scalar
    Readonly::Scalar    $sca => $initial_value;
    Readonly::Scalar my $sca => $initial_value;

    # Deep Read-only array
    Readonly::Array    @arr => @values;
    Readonly::Array my @arr => @values;

    # Deep Read-only hash
    Readonly::Hash    %has => (key => value, key => value, ...);
    Readonly::Hash my %has => (key => value, key => value, ...);
    # or:
    Readonly::Hash    %has => {key => value, key => value, ...};

    # You can use the read-only variables like any regular variables:
    print $sca;
    $something = $sca + $arr[2];
    next if $has{$some_key};

    # But if you try to modify a value, your program will die:
    $sca = 7;
    push @arr, 'seven';
    delete $has{key};
    # The error message is "Modification of a read-only value attempted"

    # Alternate form (Perl 5.8 and later)
    Readonly    $sca => $initial_value;
    Readonly my $sca => $initial_value;
    Readonly    @arr => @values;
    Readonly my @arr => @values;
    Readonly    %has => (key => value, key => value, ...);
    Readonly my %has => (key => value, key => value, ...);
    Readonly my $sca; # Implicit undef, readonly value

    # Alternate form (for Perls earlier than v5.8)
    Readonly    \$sca => $initial_value;
    Readonly \my $sca => $initial_value;
    Readonly    \@arr => @values;
    Readonly \my @arr => @values;
    Readonly    \%has => (key => value, key => value, ...);
    Readonly \my %has => (key => value, key => value, ...);
```

Figure 2: Clicking on the breadcrumb link brings up the page for the whole distribution.



473 results (1.033 seconds)

SOMETHING MISSING?
Find out why

Readonly - Facility for creating read-only scalars, arrays, hashes 5 ++
SANKO/Readonly-2.00 (5 reviews) - Jul 02, 2014 - Search in distribution

Readonly::XS - Companion module for Readonly.pm, to speed up read-only scalar variables. 3 ++
The Readonly module (q.v.) is an effective way to create non-modifiable variables. However, it's relatively slow. The reason it's slow is that is implements the read-only-ness of variables via tied objects. This mechanism is inherently slow. Perl sim...
ROODE/Readonly-XS-1.05 - Feb 24, 2009 - Search in distribution

Readonly::BeginLift - Readonly at BEGIN time ++
The Readonly module exports the "Readonly" subroutine, but this subroutine executes at runtime. This module causes it to execute at BEGIN time. Thus: use strict; use warnings; use Readonly; use constant MY_VALUE => 'foo'; Readonly my $MY_VALUE => 'ba...
OVID/Readonly-BeginLift-0.03 - Feb 13, 2010 - Search in distribution

Scalar::Readonly - functions for controlling whether any scalar variable is read-only ++
This simple module can make scalars read-only. Useful to protect configuration variables, for example. This module can also be used to subvert Perl's many read-only variables to potential evil trickery. readonly Ths function takes a scalar variable a...
GOZER/Scalar-Readonly-0.03 - Jan 23, 2012 - Search in distribution

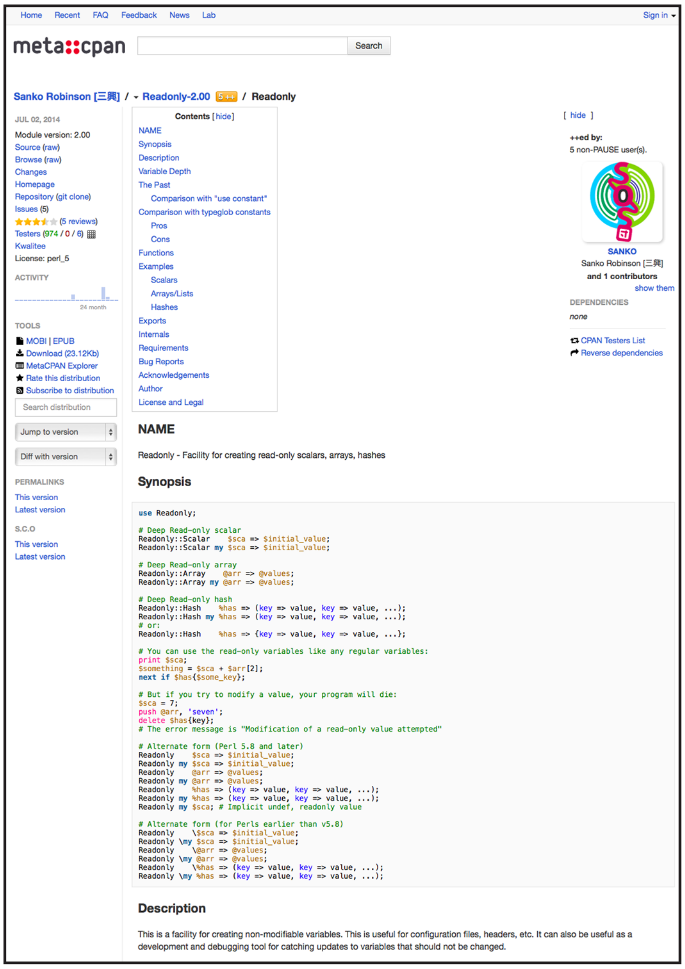
ARGV::readonly - make <> open files regardless of leading/trailing whitespace and/or control characters such as |, >, amd <. ++
the code is shorter than the documentation. Please look at it. EXPORT None by default. TO DO ideally a suite of ARGV::* modules will appear, each doing their little thing, in a way that they won't stomp on each other's toes. This module has no exclus...
DAVIDNICO/ARGV-readonly-0.01 - Jul 30, 2008 - Search in distribution

Readonly::Enum - enumerated scalar values ++
This module provides some syntactic sugar for defining enumerated scalar values. It is to Readonly what the enum package is to constant. Unlike enumerated constants, these scalars can be used as hash keys and interpolated in strings. Unlike enum, onl...
RRWO/Readonly-Enum-v0.1.4 - Feb 23, 2014 - Search in distribution

Types::ReadOnly - type constraints and coercions for read-only data structures and locked hashes 1 ++
This is a type constraint library for write-restricted references. This module is built with Type::Tiny, which means that you can use it with Moo, Mouse, Moose, or none of the above. Type Constraints This library provides the following type constrain...
TOBYINK/Types-ReadOnly-0.002 - Nov 04, 2013 - Search in distribution
  Type::Tiny::Wrapper - a type constraint that is only useful when wrapping another type constraint
  Type::Tiny::Wrapped - a type constraint that has been wrapped with a Type::Tiny::Wrapper
  Type::Coercion::Wrapped - a coercion for a Type::Tiny::Wrapped type constraint

Results per page: 10 20 50 100 200 500

Figure 3: Using MetaCPAN as your search engine



NAME

Readonly - Facility for creating read-only scalars, arrays, hashes

Synopsis

```
use Readonly;

# Deep Read-only scalar
Readonly::Scalar    $sca => $initial_value;
Readonly::Scalar my $sca => $initial_value;

# Deep Read-only array
Readonly::Array    @arr => @values;
Readonly::Array my @arr => @values;

# Deep Read-only hash
Readonly::Hash    %has => (key => value, key => value, ...);
Readonly::Hash my %has => (key => value, key => value, ...);
# or:
Readonly::Hash    %has => {key => value, key => value, ...};

# You can use the read-only variables like any regular variables:
print $sca;
$something = $sca + $arr[2];
next if $has{$some_key};

# But if you try to modify a value, your program will die:
$sca = 7;
push @arr, 'seven';
delete $has{key};
# The error message is "Modification of a read-only value attempted"

# Alternate form (Perl 5.8 and later)
Readonly    $sca => $initial_value;
Readonly my $sca => $initial_value;
Readonly    @arr => @values;
Readonly my @arr => @values;
Readonly    %has => (key => value, key => value, ...);
Readonly my %has => (key => value, key => value, ...);
Readonly my $sca; # Implicit undef, readonly value

# Alternate form (for Perls earlier than v5.8)
Readonly    \$sca => $initial_value;
Readonly \my $sca => $initial_value;
Readonly    \@arr => @values;
Readonly \my @arr => @values;
Readonly    \%has => (key => value, key => value, ...);
Readonly \my %has => (key => value, key => value, ...);
```

Description

This is a facility for creating non-modifiable variables. This is useful for configuration files, headers, etc. It can also be useful as a development and debugging tool for catching updates to variables that should not be changed.

Figure 4: Clicking on the first module link after searching at MetaCPAN

I'd like to call your attention to a few of the things in the left and right sidebars. On the left sidebar, I have the "Browse" link I crave, an easy way to look at the Changelog, pointers to the Git repo, the Bug tracker page for the module, reviews, test results (more on this later), a link to an automated system for determining module quality ("Kwalitee"), an indication how active the development is of the module, and even a way to download the module doc in several ebook formats plus a bunch more stuff. On the right sidebar, we see the dependencies of the module and, perhaps even cooler, a way to see the reverse dependencies (i.e., which modules depend on this one). Super cool.

These are just some of the immediately visible features of MetaCPAN. One thing you probably can't see is a key underlying building block. The search being run on metacpan.org is actually the output of calls to api.metacpan.org (documented here: https://github.com/CPAN-API/cpan-api). If you'd like to create your own client, the API is open, and example code to use it is freely available. As you can probably guess, metacpan.org is my usual "go-to" search method for searches. I even use this template from within one of my OS X helper applications (Launchbar) to make looking up module documentation quick and easy: https://metacpan.org/search?q=*

Separating the Wheat from the Chaff

Now that you know how to find more modules than you can shake a stick at, how do you figure out which are the good ones? I've mentioned a couple of ideas already in passing, but let's take a closer look at this question.

First, I think it is worthwhile to favor modules that appear to be actively maintained. This increases the likelihood that there is an author out there improving the module and also available to fix issues should you find any. The last release date is a good hint about this, the activity indictor provided by metacpan.org is an even better indication.

Second, closely related to the first idea is the number of bugs opened against the module. I don't believe zero active bugs is necessarily a good thing. I'd much rather see a few open bugs (shows community involvement) alongside a number of closed bugs (shows author involvement and responsiveness). A queue full of unresolved bugs is also a great red flag that may indicate an orphaned module. Use this as one of your parameters for judgment but not the only one.

Third, consider whether the module appears to actually work. One way to determine this is to look at the Testers link off of the metacpan module page for a module. To return to the panels I mentioned at the beginning of the column, another thing I believe Perl can be proud about is the strong cultural inclination towards testing in the community. One way this manifests is that every version of every module that gets submitted to CPAN gets "smoke tested" on close to a thousand different combinations of Perl versions and operating systems. If a module includes tests (and indeed, every module is encouraged to have as complete a test suite as possible), these tests are run in each of these environments and the results reported back to the central test result repository for you to peruse. This gives you a good indication (again, if the test suite is decent) of how portable and how fragile the module code is likely to be. I've mentioned a couple of times that a good test suite makes this metric useful, and I'd recommend using the Browse link to see what sort of tests are included with a module. Similarly, if you browse and find the module has a README that contains boilerplate that the author hasn't bothered to change along the lines of:

The README is used to introduce the module and provide instructions on how to install the module, any machine dependencies it may have (for example, C compilers and installed libraries) and any other information that should be provided before the module is installed....

(which is how the boilerplate provided by Module::Starter begins), that's generally a bad sign. There are other signs of slapdashery you can find in module land, but this is one of my favorite indicators.

Fourth, pay attention to both the dependency and reverse-dependency hints provided by metacpan.org. Looking at the dependency list for a module can give you some sense of things such as how hard it will be to get a module to install (do you already have the dependencies installed?), does the author tend to use already existing code (which can be good or bad) or prefer to rewrite everything from scratch, and in general what modules that author trusts.

I find the list of reverse dependencies to sometimes be an even more useful metric for module trustworthiness. Using the same basic underlying principle of Google PageRank, if lots of other modules depend on a module you are considering installing, that's almost always a really good thing. If there's a problem with the module, a whole bunch of other module authors have an incentive to see that problem resolved. Similarly, those authors are also invested in the continued stability and incremental improvement of the module you are considering. In case you are curious, according to the CPAN Top 100 site (http://ali.as/top100/), the module with the most dependencies is App::Munchies (a Catalyst demonstration Web app), and the module that the most other modules depend on is Test::Harness.

Fifth, and my last tip for picking good modules, is to find an opinionated author/expert you trust and follow their advice. Two examples of this are Damian Conway's *Perl Best Practices* (full disclosure, published by the same publisher as my book) and the Task::Kensho module. This module is basically a list of recommended modules, or as their doc puts it:

Task::Kensho is a list of recommended modules for Enlightened Perl development. CPAN is wonderful, but there are too many wheels and you have to pick and choose amongst the various competing technologies.

The list looks very solid to me, so I think you can't go wrong at least consulting it as part of your decision process.

Gimme, Gimme

Now that you've found the module of your dreams, how do you go about using it? There's a decision tree here that many a sysadmin has argued about in the past, namely do you install modules using the language native method or do you strictly only use pre-built packages in the context of the package management system your operating system uses (even if you have to build the package yourself). Let's look at both roads.

Back in the early days, people used a module called CPAN.pm to install their Perl modules. Later on, a spiffier version was created called CPANPLUS, and that's a fairly common way to install modules. It installs a command-line program called "cpanp" that you can run and use like this:

```
$ cpanp
CPANPLUS::Shell::Default -- CPAN exploration and module
installation (v0.9121)
*** Please report bugs to <bug-cpanplus@rt.cpan.org>.
*** Using CPANPLUS::Backend v0.9121.  ReadLine support
enabled.

*** Type 'p' now to show start up log

CPAN Terminal>i Readonly
```

This will search for and install the Readonly module (and all of the dependencies it has). By default it is fairly interactive, asking you each step of the way whether you want to install each dependency. This isn't my current method for module installation, but before I move on to what I prefer, let me mention one thing CPANPLUS does that is valuable. Instead of using the "i" command for install, typing "o" will output a list of the outdated modules on your system. Sort of like this:

```
1    1.5701    1.61    App::Cpan          BDFOY
2    0.58      0.68    Archive::Extract   BINGOS
3    1.82      1.90    Archive::Tar       BINGOS
4    5.72      5.73    AutoLoader         SMUELLER
5    1.17      1.18    B::Debug           RURBAN
6    1.14      1.17    B::Lint            RJBS
7    1.52      1.59    CAM::PDF           CDOLAN
8    3.59      3.63    CGI                MARKSTOS
...
```

The second column is the version you have installed, the third is the latest version found on CPAN. This can be very handy if you like to keep current.

My use of CPAN.pm and CPANPLUS has almost entirely been supplanted by a package called CPANMINUS. I typically use it in conjunction with the perlbrew system (http://perlbrew.pl), which allows you to have multiple versions of Perl installed on your system without conflict (including conflict with the one that ships with your operating system). If you are using perlbrew, "perlbrew install-cpanm" will install it for you. If you are not using perlbrew, there are a number of ways to install it, including this scary, scary way:

```
$ curl -L http://cpanmin.us | perl - App::cpanminus
```

See http://cpanmin.us for more details.

Once you have CPANMINUS installed, you will have a "cpanm" command. "cpanm" can take a few flags to modify its behavior, but more often than not, you'll just be typing:

```
$ cpanm {module name}
```

as in

```
$ cpanm Readonly
```

CPANMINUS will install the module and any dependencies it has lickety-split with basically no interaction and no fuss. It is really written for the "I want the thing. Thing is now installed." experience and does it very well.

So that's how you would install things independent of any package management system your operating system uses. Some find this to be fine; others feel it is reckless and contrary to the reason one has a package management system. If you want to stick to a package system, I do know that in addition to package manager-specific tools like dh-make-perl, the awesome FPM tool (https://github.com/jordansissel/fpm) by Jordan Sissel can also help create packages for you.

So, with that, we've learned how to find good Perl modules and install them easily. Let's leave it there. Take care and I'll see you next time.