Practical Perl Tools
Top of the Charts

DAVID N. BLANK-EDELMAN

David has over 30 years of experience in the systems administration/DevOps/SRE field in large multiplatform environments and is the author of the O'Reilly Otter book (new book on SRE forthcoming!). He is one of the co-founders of the now global set of SREcon conferences. David is honored to serve on the USENIX Board of Directors where he helps to organize and engineer conferences like LISA and SREcon. dnb@usenix.org

I sometimes wonder if the people who make statements about Perl's health in the world (some nostalgic, some a little more mean-spirited) have a sense of just how vibrant the Perl world is. I wonder whether seeing some of the interesting things being developed even as we speak or the range of projects available would change their thinking.

This leads to a good question: how do you find out about the interesting things happening in Perl on a week-to-week basis? In this column I'd like to focus on one of the answers to that question: the weekly reports that are published about modules.

We'll look at three of them and for fun pick and consider interesting modules from each. All three of these listings are published each week on a blog created by Spanish Perl hacker Miguel Prz ("NICEPERL"), which can be found at http://niceperl.blogspot.com. Before we dive in, I should mention that these listings came to my attention thanks to the lovely newsletter started by Gabor Szabo called *Perl Weekly*. You can sign up and find past issues at http://perlweekly.com.

CPAN Great Modules Released Last Week

The first list we are going to look at is indeed titled "CPAN great modules released last week." This is an ordered list of modules newly published to the Comprehensive Perl Archive Network which has garnered 12 or more "favorite" ratings—that is, 12 or more people "++"d these modules on the MetaCPAN (metacpan.org) listing site.

On the off chance you are brand new to the Perl world, let me quickly mention that CPAN is an archive where people in the Perl community share their modules and other Perl work for everyone to use. It is one of the best things Perl has going for it. And to be totally candid, it is not always the best; some of the code there isn't the greatest. To give you a sense of its scale, here are the stats as of today from the cpan.org home page:

```
The Comprehensive Perl Archive Network (CPAN) currently has 194,457 Perl modules
in 35,953 distributions, written by 13,329 authors, mirrored on 256 servers.
The archive has been online since October 1995 and is constantly growing.
```

So what's in the list of great modules for the week of December 17, 2017, the one in which I am writing?

This week we find a few old friends of the column like Mojolicious and perlbrew. Instead of retreading, let's instead look at DBIx::Simple. In the past, we've talked a bit about the framework that was a true innovator in the space at the time it was introduced: DBI. The idea of having a single portable abstraction layer for code that communicated with databases independent of the database backend being used was a great step forward at the time. This idea was subsequently expanded in many different directions (and the basic concept was repurposed for other non-database contexts as well). My magic 8-ball predicts an entire column on DBI-related expansions coming in our near future… But in the meantime, let's look at DBIx::Simple.

#	CPAN module	Version	Votes	Abstract
1	App::perlbrew	0.81	149	App::perlbrew - Manage Perl installations in your $HOME
2	Catalyst::Action::REST	1.21	16	Automated REST method dispatching
3	Data::Alias	1.21	12	Comprehensive set of aliasing operations
4	DBIx::Simple	1.37	27	Very complete easy-to-use OO interface to DBI
5	Digest::SHA	6.00	19	Perl extension for SHA-1/224/256/384/512
6	experimental	0.019	29	Experimental features made easy
7	libwww::perl	6.30	135	The World Wide Web library for Perl
8	Math::Prime::Util	0.70	12	Utilities related to prime numbers, including fast sieves and factoring
9	Mojolicious	7.58	352	Real-time web framework
10	SQL::Translator	0.11023	32	SQL DDL transformations and more
11	Test::Class::Moose	0.91	14	Serious testing for serious Perl
12	Text::Xslate	v3.5.3	58	Scalable template engine for Perl5

Table 1: This is what the table at https://niceperl.blogspot.com/2017/12/clxii-cpan-great-modules-released-last.html looked like on December 17, 2017.

Standard DBI has you writing code that looks like this (examples excerpted from the DBI doc with my comments inserted):

```
use DBI;

# connect to a database
$dbh = DBI->connect($data_source, $username, $auth, \%attr);

# execute a random SQL statement
$rv  = $dbh->do($statement);

# various ways of retrieving the results
$ary_ref  = $dbh->selectall_arrayref($statement);
$ary_ref  = $dbh->selectcol_arrayref($statement);
@row_ary  = $dbh->selectrow_array($statement);
$ary_ref  = $dbh->selectrow_arrayref($statement);
$hash_ref = $dbh->selectrow_hashref($statement);

# more efficient ways of running/rerunning queries
$sth = $dbh->prepare($statement);
$rv  = $sth->execute;

# other ways of retrieving results
@row_ary  = $sth->fetchrow_array;
$ary_ref  = $sth->fetchrow_arrayref;
$hash_ref = $sth->fetchrow_hashref;

# close connection to the database
$rc  = $dbh->disconnect;
```

These are some of the more commonly used statements when working with DBI, certainly when first getting started. It's worth reading the entire doc (several times) to get a good handle on the proper idioms and performant ways to work with DBI. And, hoo boy, is there plenty of doc to read—124 pages if you were to print it all out as of the time of this writing.

DBIx::Simple aims to, well, you probably guessed it, make some of the coding with DBI more simple. With DBIx::Simple, you write code that looks almost identical to plain DBI:

```
use DBIx::Simple;

my $db = DBIx::Simple->connect(
  DBI:mysql:database=test,     # DBI source specification
  test, test,                  # Username and password
  { RaiseError => 1 }          # Additional options
);
```

but then you can write code of this form (as stated in the doc):

```
$db->query($query, @variables)->what_you_want;
```

Some examples of this from the doc would be:

```
my ($name, $email) = $db->query(
  SELECT name, email FROM people WHERE email = ? LIMIT 1,
  $mail
)->list;
```

Here we're querying the people table for a list of two fields—name and email—given the email address (`$mail`). We ask for the information back as a list.

If we didn't want to chain methods like that, we could write:

```
$result = $db->query(...)
```

and then work from `$result` object returned using methods like:

```
@columns = $result->columns
```

or

```
@row  = $result->list     # return as a list
@rows = $result->flat     # return as a flattened list
$row  = $result->array    # return as an array ref
@rows = $result->arrays   # return as an array of arrays
$row  = $result->hash     # return as a hash
@rows = $result->hashes   # return as an array of hashes
```

Here's an example of a query that returns a set of rows which we then iterate over to print separate fields:

```
for my $row (
  $db->query(SELECT name, email FROM people)->hashes) {
    print "Name: $row->{name}, Email: $row->{email}\n";
}
```

I like how legible something like `$row->{fieldname}` is in the code.

DBIx::Simple also hooks into a few other modules (some of which we'll likely talk about soon). For example, if you have Text::Table installed, then code like:

```
print $result->text("box")
```

makes pretty output like:

```
+----+--------+----------+
| ID | Animal | Type     |
+----+--------+----------+
|  1 | camel  | mammal   |
|  2 | llama  | mammal   |
|  3 | owl    | bird     |
+----+--------+----------+
```

MetaCPAN Weekly Report

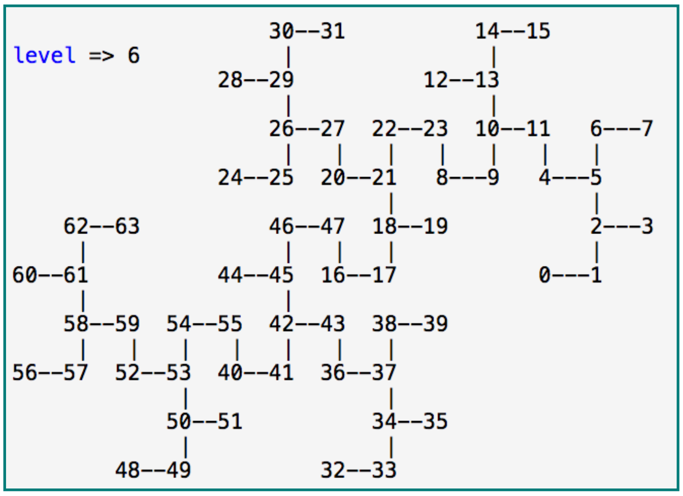
The second report to mention is also from data pulled from the MetaCPAN site. This report lists both the newly favorited modules ("Clicked for the first time") and those whose popularity is on the rise ("Increasing its reputation"). If enough people vote for a particular module, this report will call out that module as "special," but that did not happen this week.

You can find an example of the table at https://niceperl.blogspot.com/2017/12/ccxciv-metacpan-weekly-report.html.

In the "first click" section, I found a couple of different modules to be interesting, not because they helped me discover a module I might use, but because they offered solutions for problems in spaces that I knew very little about. It can be fun to have something like this shoot you off on tangents (not to mention build your procrastination muscles). For example, I had never heard of a Confusion Matrix until I met AI::ConfusionMatrix. In case you are curious, Wikipedia defines them as follows:

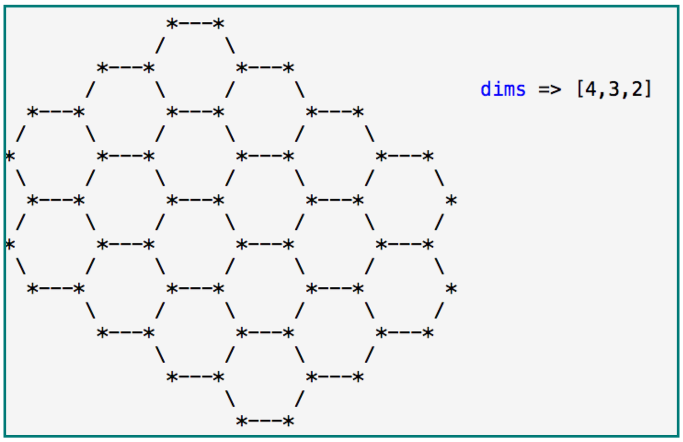
> In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

Similarly, I realized how woefully inadequate my understanding of graph theory was when I encountered Graph::Maker::Other. This appears to be a collection of modules for making graphs like Beineke, bi-star, quartet tree, twindragon area tree, and others I hadn't heard of. Some are quite pretty—for example, that last one:



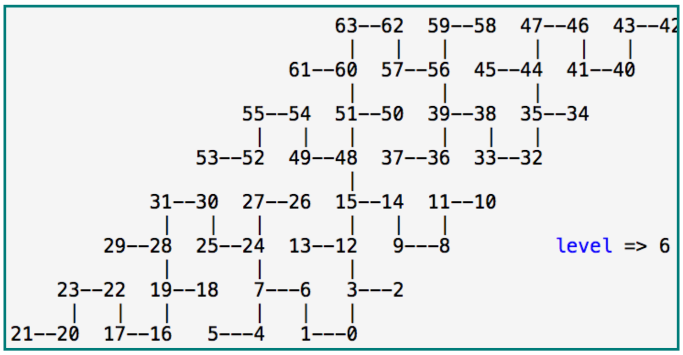
```
                    30--31          14--15
level => 6           |               |
                28--29          12--13
                     |               |
                    26--27  22--23  10--11   6---7
                     |   |   |   |   |   |   |
                24--25  20--21   8---9   4---5
                             |               |
    62--63          46--47  18--19           2---3
     |               |   |   |               |
60--61          44--45  16--17           0---1
     |               |
    58--59  54--55  42--43  38--39
     |   |   |   |   |   |   |
56--57  52--53  40--41  36--37
             |               |
            50--51          34--35
             |               |
        48--49          32--33
```

or hexgrid:



```
              *---*
             /     \
        *---*       *---*
       /     \     /     \               dims => [4,3,2]
  *---*       *---*       *---*
 /     \     /     \     /     \
*       *---*       *---*       *---*
 \     /     \     /     \     /     \
  *---*       *---*       *---*       *
 /     \     /     \     /     \     /
*       *---*       *---*       *---*
 \     /     \     /     \     /     \
  *---*       *---*       *---*       *
       \     /     \     /     \     /
        *---*       *---*       *---*
             \     /     \     /
              *---*       *---*
                   \     /
                    *---*
```

or twin alternate area tree:



```
                            63--62  59--58  47--46  43--42
                             |   |   |       |   |   |
                        61--60  57--56  45--44  41--40
                             |       |       |
                    55--54  51--50  39--38  35--34
                     |   |   |       |   |   |
                53--52  49--48  37--36  33--32
                             |
            31--30  27--26  15--14  11--10
             |   |   |       |   |   |
        29--28  25--24  13--12   9---8         level => 6
             |       |       |
    23--22  19--18   7---6   3---2
     |   |   |       |   |   |
21--20  17--16   5---4   1---0
```

The documentation in this module distribution also references a website called House of Graphs (https://hog.grinvin.org), which is a "Database of interesting graphs." And there goes that afternoon…

To pull things back to the world of Perl, I'd like to highlight the useful module File::HomeDir::Tiny, which describes itself as follows:

> This module is useful for the 90% of the time that you only need 10% of File::HomeDir's functionality. It depends on no other modules and consists of just fourteen lines of code.

so:

```
use File::HomeDir::Tiny;
$home = home;
```

Nothing complex, but highly useful. And if you want to have a quick party trick up your sleeve (presuming you go to the right parties) for when someone asks you about how nuts the Perl punctuation can be, check out the part of the docs that begins with:

> If your code is only going to run on Unix, you do not need to bother with any module. Just use the alien spaceship operator:

```
($home) = <~> ;
```

From the "Increasing its reputation" section of this report, there's a whole bunch of interesting modules to look at. There's Damien Conway's PPR (Pattern-based Perl Recognizer), which scares the pants off of me. It is basically a distribution of Perl regular expressions designed to match certain Perl constructs in a Perl program. Unlike PPI (which we looked at in a recent column), which actually parses Perl, this just allows you to easily say, "Is there a comment in this Perl code?" or give me back the code without the comment, the same way you might strip anything else out of text using a regular expression. I was too scared to look at the actual source code for this module.

Other interesting stuff:

- App::tt, a time tracking module/app. This is the sort of thing that people who have to keep track of their time spent on individual activities or projects would love. From the command line, you can say "started" and then later "finished." It can also do spiffy things like look at a directory with a Git repo in it and use the `reflog` there to automatically determine working times.
- App::RoboBot, an "event-driven, multi-protocol, multi-network, user-programmable, plugin-based, S-Expression chatbot. Any text-based chat service could be supported, with plugins currently for IRC, Slack, and Mattermost included." More info can be found at: https://robobot.automatomatromaton.com. Yup, there goes another afternoon.
- Promises, an implementation of Promises in Perl. If you ever get the chance to venture out into the wider programming world, especially in the world of JavaScript and the jQuery library, you may encounter a programming construct called "Promises." Promises are an attempt to deal with the complexity of writing reasonably sized asynchronous programs based on callbacks. At a certain point, those programs devolve into what people call "callback hell" because they invariably start to have callbacks calling other callbacks. If the forest of callbacks gets too thick or self-referential, it becomes very hard to debug or even to understand how the program will behave. As the doc says, "Promises give us back a top-to-bottom coding style, making async code easier to manage and understand. It looks like synchronous code, but execution is asynchronous." If you find yourself writing even medium-sized programs that are event/callback based, it would probably behoove you to check out the world of Promises to see how others are coping with the complexity you are sure to encounter. The external references in the docs are also well worth taking some time to go read.

StackOverflow Perl Report

Okay, last report. This one is less useful for finding cool modules or strange afternoon-wasting tangents and more helpful for keeping your head in the Perl game and refreshing your Perl knowledge. This section lists the top 10 rated Perl questions on StackOverflow. It also lists the number of answers provided for each. I find it useful to just scan the list each week to see if there are any questions that pique my curiosity (or that make me feel like "hmm, I know that" or "hmm, I really should know that"). Table 2 shows the current list of questions for 12/9/17.

1. Perl DBI (MySQL) puts single quote instead of actual parameter in prepared statement - [7/1]
2. How to search and replace multiple lines with multiple lines - [5/5]
3. In perl, when assigning a subroutines return value to a variable, is the data duplicated in memory? - [5/3]

4. Is there a way to configure the default mirror for App::cpanminus (cpanm)? - [5/1]
5. NBSP malformed while using Mojo::DOM - [5/0]
6. How can I assign a weight for frequency score to a graph&'s edge using Graph::Easy - [4/2]
7. How to run multiple perl Dancer2 apps in same nginx server - [3/2]
8. How to accept self-signed certificates with LWP::UserAgent - [3/2]
9. Why are the referenced arrays values not changed? - [3/2]
10. Time::Piece (localtime/gmtime) calculation vs bash date - [3/1]

Table 2: The list at https://niceperl.blogspot.com/2017/12/cccxviii-stackoverflow-perl-report.html

And with that, we come to the end of these reports that are great for finding interesting things in the Perl world. Take care, and I'll see you next time.