Practical Perl Tools

Perl on a Plane

DAVID N. BLANK-EDELMAN

David Blank-Edelman is the Technical Evangelist at Apcera (the comments/views here are David's alone and do not represent Apcera/Ericsson) . He has spent close to 30 years in the systems administration/DevOps/SRE field in large multiplatform environments including Brandeis University, Cambridge Technology Group, MIT Media Laboratory, and Northeastern University. He is the author of the O'Reilly Otter book *Automating System Administration with Perl* and is a frequent invited speaker/organizer for conferences in the field. David is honored to serve on the USENIX Board of Directors. He prefers to pronounce Evangelist with a hard 'g'. dnb@usenix.org

I travel a great deal these days for my work, so it isn't uncommon for me to find myself on an airplane hoping to get some work done with only dribbles of WiFi. In those cases, you often have to make do with whatever is already on your laptop. I thought it might be interesting to explore what sort of goodies you might have available under those conditions from a stock Perl installation. To make this column extra realistic, let me report that as I write this I am flying at 34,153 ft at a speed of 437 mph over Lake Ontario (honest truth). Right before I left for the airport, I used perlbrew to install a stock version of the stable version of Perl (5.24.1) on my laptop. Let's switch to it and start our exploration:

```
$ source ~/perl5/perlbrew/etc/bashrc
$ perlbrew --notest install perl-5.24.1
$ perlbrew use perl-5.24.1
```

Perlbrew is a lovely tool for installing a discrete installation of Perl on a machine without perturbing any version of Perl shipped with the system. It will pull down the source for the version you desire and compile it. In the second line, I had to add `--notest` because 5.24.1 appears to have an issue on the version of OS X I'm running, which has a few broken tests in Time::Hires to be fixed in future versions of Perl. After hitting that failure a few times, I didn't think it would materially change what happens in this column, so I chose to skip the tests normally run as part of installing Perl.

The first place to look for interesting material is in the documentation system. Say what you'd like about Perl, no one can accuse it of not shipping with enough documentation. If I type "perldoc perl" it lists the following (heavily excerpted) list:

```
Overview
    perl            Perl overview (this section)
    perlintro       Perl introduction for beginners
    perlrun         Perl execution and options
    perltoc         Perl documentation table of contents

Tutorials
    perlreftut      Perl references short introduction
    perldsc         Perl data structures intro
    perllol         Perl data structures: arrays of arrays

    perlrequick     Perl regular expressions quick start
    perlretut       Perl regular expressions tutorial

    perlootut       Perl OO tutorial for beginners

    perlperf        Perl performance and optimization techniques

    perlstyle       Perl style guide
```

```
        perlcheat          Perl cheat sheet
        perltrap           Perl traps for the unwary
        perldebtut         Perl debugging tutorial

        perlfaq            Perl frequently asked questions

    Reference Manual
        perlsyn            Perl syntax
        perldata           Perl data structures
        perlop             Perl operators and precedence
        perlsub            Perl subroutines
        perlfunc           Perl built-in functions
...
        perluniintro       Perl Unicode introduction
...
        perlunitut         Perl Unicode tutorial
        perlebcdic         Considerations for running Perl on
                           EBCDIC platforms

        perlsec            Perl security

        perlmod            Perl modules: how they work
...
    Internals and C Language Interface
        perlembed          Perl ways to embed perl in your C or C++
                           application
        perldebguts        Perl debugging guts and tips
        perlxstut          Perl XS tutorial
...

    Miscellaneous
        perlbook           Perl book information
        perlcommunity      Perl community information

        perldoc            Look up Perl documentation in Pod
                           format

        perlhist           Perl history records
        perldelta          Perl changes since previous version

        perlexperiment     A listing of experimental features in Perl
...

    Language-Specific
        perlcn             Perl for Simplified Chinese (in EUC-CN)
        perljp             Perl for Japanese (in EUC-JP)
        perlko             Perl for Korean (in EUC-KR)
        perltw             Perl for Traditional Chinese (in Big5)

    Platform-Specific
        perlaix            Perl notes for AIX
        perlamiga          Perl notes for AmigaOS
        perlandroid        Perl notes for Android
        perlbs2000         Perl notes for POSIX-BC BS2000
...
```

Be sure to run that command to see the full list for yourself. There are 178 documents in all. So even if you just decide to spend your time reading Perl docs on a plane, you've got plenty of material available to you.

The Weirdest Module Search You Ever Did See

There's a straightforward way to find the modules installed with Perl, but let's go looking for interesting modules the hard way. What if we searched for all of the modules mentioned in the Perlfaq documents and used that as the starting place for our exploration? There are more sophisticated ways to find all of the modules, but let's start with a crude hammer and look for all of the :: sequences in the FAQs. And as we do it, let's eliminate all of those mentioned with CPAN on the same line (since we theoretically don't have great access to it here in the air):

```
for i in 1 2 3 4 5 6 7 8; do
   perldoc perlfaq$i|grep '::'|grep -v CPAN
done
```

This yields 307 lines (not all of which actually include non-CPAN-dwelling module names), so I'm going to cherry-pick a few that look interesting and talk about them:

Module::CoreList—Why, yes, there is a madness in my method. Wait, strike that, reverse that. Module::CoreList is a great place to start because it is a module that can help us find and describe the modules that have shipped with Perl (core) over the years. We could either use the command line utility that comes with it (`corelist`) or write little snippets of code like:

```
use Module::CoreList;

print join("\n",Module::CoreList->find_modules('^Text::',$]));
```

This will display all of the Text::* modules that ship in core with the current version of Perl. `find_modules()` searches for a regular expression and also takes a second argument describing which Perl versions it should consider. The magic variable `$]` returns the current version of Perl. We print this using a join just to place each element in the returned array on its own line. And, yes, this would be a fine and dandy way to find all of the modules shipped with the current copy of Perl. Something like this:

```
print join("\n",Module::CoreList->find_modules('',$]));
```

But if I told you that, it might cut short our little wandering walk together, so let's keep this between the two of us. As a small aside, it probably would have made my cherry-picking of modules to discuss here more efficient if I had run them through `Module::CoreList::is_core` first.

ExtUtils::Installed—Okay, really I'm not cooking the books here. This is the next module that comes up in the FAQ. ExtUtils::Installed gives you a way to figure out the names of all of the modules installed and the files and directories for each.

This is distinct from the previous module that talks about what modules are shipped with the core vs. the ones that are currently installed (core + whatever else you installed). It does this by inspecting the special "dot file droppings" that get installed with a module (.packlist). When I first tried out this module, it briefly puzzled me. I wrote:

```
use ExtUtils::Installed;

my $inst = ExtUtils::Installed->new();
print join("\n",$inst->modules());
```

And it printed:

```
Perl
```

That's right, just "Perl." It turns out that `ExtUtils::Installed` attempts to be smart. It knows which modules are considered "core" and lumps those all into "Perl." When I ran the same script using an older version of Perl that had more modules that I had expressly installed, it did indeed report the list of installed modules in addition to just "Perl." `ExtUtils::Installed` can do other tricks like show you the files and directories installed by a module—for example:

```
print join("\n",$inst->directories("Perl"))
```

will indeed show you all of the directories of all of the modules shipped with that version of Perl live from your file system.

TimePiece—If you've ever found it annoying to use `localtime()` or `gmtime()` in Perl because it either returns an array of fields you have to guess how to index to find the field you want or (in a scalar context) just a string:

```
$ perl -de 0
DB<1> x localtime()
0  24
1  47
2  20
3  20
4  2
5  117
6  1
7  78
8  1
DB<2> x scalar localtime()
0  'Mon Mar 20 20:47:27 2017'
```

Time::Piece can help. It lets you write code that looks like this instead (to quote the docs):

```
use Time::Piece;

my $t = localtime;
print "Time is $t\n";
print "Year is ", $t->year, "\n";
```

`localtime()` now returns an object that has methods you can call to retrieve the part of the time structure you want (for example, "$t->hour" will return the current hour). It also gives you some convenience methods like "->isdst" to determine if it is currently daylight savings time. Check out the documentation for the full list.

TieFile—In a previous column many moons ago I went gaga for the cool and cruel things you can do with the Perl `tie()` function. This function lets you essentially run arbitrary code as part of the process of retrieving and setting variable contents. For example, instead of getting a value from memory when asking for `$weather{'Boston'}`, Perl could query some weather service on the Web and return the information instead. `Tie::File` isn't that futuristic, but it can do something pretty cool. If you use it like this:

```
use Tie::File;
tie @array, 'Tie::File', filename
```

you can access lines of the file (getting and setting) by just reading or changing array values. The doc gives these examples:

```
  $array[13] = 'blah';     # line 13 of the file is now 'blah'
  print $array[42];        # display line 42 of the file
```

If you truncate the array by changing its size, so too does the file change. Your other standard array operations (`push`, `pop`, etc.) behave exactly as you would expect. Oh, and here's a fun tidbit from the doc:

```
The file is not loaded into memory, so this will work even for
gigantic files.
```

FileCopy—Yup, does what you would expect.

FilePath—Probably not what you would expect. Use this to create or delete directory trees.

FileTemp—Use this, and probably only this, for dealing with temporary files.

TextBalanced—If you ever read Jeffrey Friedl's *Mastering Regular Expressions* you know that trying to extract things from delimited text (for example, some text that has parentheses around it, like this one) can be less than straightforward. This comes up in all sorts of situations, like when parsing HTML or XML, program source code, and so on.

TermANSIColor—I'm almost tempted not to mention this one because it has such a potential to be overused (thus allowing you to write code that outputs "angry fruit salad"), but I'm going to assume that we're all adults here and that with great power...

Yup, time to write code like (from the doc):

```
use Term::ANSIColor;
print color 'bold blue';
print "This text is bold blue.\n";
print color 'reset';
print "This text is normal.\n";
print colored("Yellow on magenta.", 'yellow on_magenta'), "\n";
```

Do me a favor and don't tell anyone where you got this superpower. On a serious note, I would commend you to consider that a larger part of the population than you probably think has some sort of color blindness (bring yourself up to speed about color blindness via a quick online search). Please consider this when writing code where the color of the output is significant and important.

And with that fun set of modules, I'm going to stop. Since I find myself on a plane too often it is entirely likely that this will be the first part in a several part series. Do let me know what you think of the idea. Take care, and I'll see you next time.