Testing CPAN against the Perl 5 Core Distribution: Where Do We Stand?

Author

James E Keenan (jkeenan@cpan.org)

Location

```
The Perl Conference: North America
(TPC::NA::2018)
Salt Lake City UT
June 18 2018
```

Synopsis

As Perl 5 continues to evolve, how do we measure its impact on 30,000+ libraries on CPAN?

In our last episode ...

At last year's Perl Conference in Alexandria, Virginia, I gave a presentation entitled **How Do We Assess and Maintain the Health of the Perl 5 Codebase?**. I spoke about 5 different stages of testing the Perl 5 core distribution:

1 Individual Contributor

2 Committer

3 Smoke Testing

4 Testing CPAN Modules

5 Outer World

Today's talk is a follow-up, but I'm only going to focus on one stage: testing CPAN modules against the core distribution.

A note on terminlogy

The main development branch in the Perl 5 repository is called **blead** (pronounced *"bleed"*). If we suspect that a commit to blead has caused a problem with a CPAN distribution, we call that situation a possible "Blead Breaks CPAN" -- or "BBC" for short.

The "Blead Breaks CPAN" Problem

More specifically, we want to know whether a CPAN distribution which previously passed all its tests now fails to configure, build, test or install when run against Perl 5 blead?

If we think that might be the case, we open up a so called "BBC ticket" in the Perl 5 bug queue at http://rt.cpan.org. We then triage to determine the cause of the breakage. Do we have?

- A defect in the code recently committed to Perl 5 blead which causes a failure in valid Perl 5 code in the CPAN module which broke; or
- Code in the CPAN distribution which was substandard and whose defects have been exposed by a valid change in Perl 5 blead; or
- Some combination of the above.

If the defect is, in whole or in part, in Perl 5 blead, then that BBC ticket is listed as a blocker for the next production release of Perl.

We strive for backwards compatibility, but sometimes breakage is inevitable, particularly when a security problem in the core distribution must be addressed. In addition, sometimes the forward evolution of the Perl 5 language requires that we deprecate existing functionality in order to make way for better functionality in the future. We understand that breaking existing functionality may disrupt Perl used in production -- the so-called "DarkPAN", so we use CPAN as a proxy for all the Perl code in the world.

The question we have to ask ourselves is: *How **well** do we test the core distribution against CPAN?* The answer is: Not systematically enough.

CPANtesters

For nearly twenty years we **have** systematically tested new uploads to CPAN against multiple versions of Perl on many different operating systems. That's the work you see on CPANtesters. Kudos to veteran CPANtesters like Andreas J. Koenig, Slaven Rezic and Chris Williams -- and a special shout-out to Carlos Guevara, who saw a video of my talk at last year's TPC and was inspired to start smoke-testing Perl 5 blead on FreeBSD, OpenBSD, Solaris and other platforms. Not only do these contributors test new CPAN releases against past **production** releases of Perl; they test them against the monthly development releases of Perl as well. So if your CPAN module failed against, say, development release 5.27.1, you would have gotten a notice of that failure just as if it had failed against 5.26.0 production release.

So far so good -- but here's where we run into problems. CPAN modules can experience test failures for many reasons -- not just because of a change in blead. Not all CPAN module authors are diligent about correcting test failures. At any given moment, let's say that 1000 out of 36,000 CPAN distributions are not getting a `PASS` from CPANtesters. Someone -- some human researcher -- has to scour CPAN for failures and try to determine whether a `FAIL` is in blead, in CPAN or some mix thereof. We have a bisection program, `Porting/bisect.pl` which can help identify a "breaking" commit in blead; it's generally effective, but time-consuming.

The researcher then has to file a `perlbug` ticket, after which the Perl 5 Porters have to discuss the ticket. That discussion can get heated. Other distributions which break due to the same commit to blead are added to that same bug ticket.

Limitations of Current Approach

This approach has a number of limitations.

- It requires deeply committed volunteers to run a high volume of tests and to maintain the testing infrastructure.
- It's very dependent on CPANtesters.org's operational status. While in recent months, particularly since the most recent Perl Toolchain Summit in Oslo in April, that site's status has been good, there were many times over the past two years when failure reports would not be available for many days after filing.
- Historically it has been difficult to search CPAN for failures.
- In the Perl 5 bug queue there is no easy way to get a list of all distributions failing due to a given commit. You may have to scroll through many posts to a BBC ticket to locate them all. Then someone has to manually record when those distributions start to `PASS` again.
- More critical, from my perspective, is that we have no way to measure our progress over time. We cannot answer questions such as:
  - How does the number of currently "broken" distributions compare to last month in the **current** development cycle?
  - How does the number of currently "broken" distributions compare to the same time **last year** in the previous development cycle?

  Lacking a temporal sense of the scope of any such breakage limits us in several ways:

  - It limits Perl 5 committers' ability to anticipate CPAN "breakage".
  - It limits Perl 5 Porters' ability to hold committers accountable for CPAN "breakage"

Developing Alternative Approaches

If we want to do better, we first have to specify what "better" means. We have multiple problems and no one solution can address **all** of them. Different people will have different criteria for "better". However, any given solution should address **some** of them.

My criteria

Here are the criteria I'm currently taking in my work on this problem.

- My solution should not depend on CPANtesters.org as the source of test reports. Other people may choose to depend on CPANtesters, but we should not be one-hundred-percent dependent on that site.
- A solution should not require years of expertise to set up or run.
- A solution should be runnable on multiple operating systems.
- A solution should provide an overall snapshot of the impact Perl 5 blead is having on CPAN.
- A solution should provide that snapshot within 24 hours of a request for one.
- A solution should be oriented toward the needs of the Perl 5 Porters.

My Solution: test-against-dev

In November of last year I began developing an application which, for short, I call **test-against-dev**. Perl releases a developmental tarball once a month. A `cron` job listens for that release and kicks off the process.

In principle, we could do this for any given commit to blead. Doing it against a monthly release, however, makes the process easier to schedule and provides the basis for easily understood time-series data.

test-against-dev downloads the monthly release from CPAN via FTP. It builds and installs that `perl`, installs `cpanm` against that `perl`, then uses `cpanm` to try to install a selected subset of CPAN modules -- 1000 of them, in the version which ran during the 5.27 development cycle -- against that `perl`. (Which 1000 CPAN modules? We'll come to that in a moment.)

Each time you run `cpanm`, a log file called `build.log` is generated. The process parses that log file and writes a compact JSON file for each module handled. Once all the JSON is written, the process summarizes the results in a pipe-separated values (or `PSV`) file and aggregates those results with the results from previous months in the annual development cycle in another PSV file suitable for opening in a spreadsheet. I then post links to that data on the Perl 5 Porters mailing list.

CPAN libraries used by process

I've written several CPAN modules to implement this process:

- `Test::Against::Dev` provides the wrapper for all the functionality just described.
- `Perl::Download::FTP` downloads the tarball via FTP.
- `CPAN::cpanminus::reporter::RetainReports` parses the `cpanm build.log` file and writes the JSON files for each module handled. This module is based on Breno G. de Oliveira's CPAN library `App::cpanminus::reporter`, which you use if you want to use Miyagawa's `cpanm` utility to install CPAN modules and generate CPANtesters reports as you do so. `App::cpanminus::reporter`, like similar CPAN libraries such as `CPAN::Reporter`, does not retain test reports on disk once they've been sent off to the CPANtesters database. That makes sense if you're doing heavy-volume CPANtesting -- otherwise your disk would fill up very quickly. But for `test-against-dev` we need a compact summary on disk so that we can create summary data for each monthly release. Hence, `CPAN::cpanminus::reporter::RetainReports`.

The CPAN river

A moment ago I said that the `test-against-dev` process tries to install as "selected subset" of all CPAN modules against a particular release. This begs the questions: "Why a subset?" and "Which subset?".

Why not test all of CPAN?

Testing all of CPAN would be overkill. Many modules are operating-system-specific. Many are outdated or abandoned. And many modules behave badly during automated testing processes like `test-against-dev`. They require interactive configuration. Or their tests rely too much on network connections. Or their tests time out. Or their tests fill up your disk. Any many modules have external dependencies which you might not be able to install on the platform you're using for testing.

So, which subset of CPAN should we use?

Chaque un à son goût! There is no one perfect set. I've chosen to go with a set derived from the `CPAN River`. The CPAN River is a metaphor developed by Neil Bowers and participants in the Perl Toolchain Summit. Imagine the Perl 5 core distribution as the source of a mighty river. CPAN distributions increase the volume of water in the river. The river eventually flows into the sea which is all Perl code everywhere. Pollution upstream, however, causes problems downstream.

We can list CPAN distributions in dependency order by imposing a directed acyclic graph (DAG) over them. During the 5.27 development cycle, I used the

"CPAN River Top 1000" -- the "farthest upstream" distributions -- as a proxy for "all of CPAN". At each monthly release I attempted to install those 1000 modules against the release, a process that, to over-simplify, looked like this:

```
        cat cpan-river-1000.txt | xargs ./bin/cpanm
```

I used Test::Against::Dev to parse the `cpanm build.log` and create a `.psv` file summarizing the results. I then notified the Perl 5 Porters mailing list.

How well did this process run?

On the positive side, by late March other Perl 5 core committers were starting to pay attention to this data and use it while evaluating our overall perl-5.28.0-readiness.

But on the negative side, this was a one-person project and it was only partially automated. In addition, I came to feel that 1000 CPAN distributions was too shallow a subset to give an accurate picture of the impact of blead on CPAN.

The 5.29 Development Cycle

I hope to improve this during the 5.29 development cycle which is just beginning. I want to test more CPAN distributions -- 3000 instead of 1000. I want to run the application on a platform other than Linux, because we know that changes to the Perl 5 core can break CPAN modules on one platform while leaving those modules intact on Linux, the most frequently tested platform. I want to run the application in a more automated manner and with input from others -- particularly from experienced system administrators. And, most importantly, I want to have this project run in a way which strengthens open-source software communities.

To that end we in New York Perlmongers have been developing a partership with the New York City BSD User Group (NYCBUG). They've generously provided us with access to a server in their rack at a top-flight data center. NYCBUG members Mark Saad and George Rosamond have shared their sysadmin expertise with us.

In addition, after I gave an earlier version of this presentation to Philadelphia Perlmongers in March, three members of Philadelphia.pm helped out by taking sets of 10 CPAN distributions which were not getting a `PASS` from `cpanm` and filing patches and pull requests with those distributions' maintainers. Kudos to John Karr, Walt Mankowski, Thomas McKernan.

Finally, when I say "we" I'm now mainly referring to myself and New York Perlmonger and veteran system administrator Andrew Villano. We are currently in the process of debugging our program so that it can run smoothly on FreeBSD-11.1.

Other Approaches

I'm not the only person working on this problem, so I should mention some others. Ryan Voots (simcop2387) has been working on this, as have Todd Rinaldo, Nicolas R and others in Houston Perlmongers. Unfortunately, as of press time I don't know the status of their efforts.

Summary

Additional thanks go to:

- David Golden

  For his MongoDB-based program to calculate the graph of the CPAN river.

- Neil Bowers

  For raising consciousness about the CPAN River and for lending me two beautiful images.

If you'd like to help out with this project, please contact me at jkeenan at pobox dot com.

Thank you very much.

References

TPC::NA::2017 Presentation: How Do We Assess and Maintain the Health of the Perl 5 Codebase?

- PDF: http://thenceforward.net/perl/tpc/TPC-NA-2017/p5-codebase-health.pdf
- Slides: http://thenceforward.net/perl/tpc/TPC-NA-2017/slides/
- Video: https://www.youtube.com/watch?v=yLFHyxALAbE&list=PLA9_Hq3zhoFxdSVDA4v9Af3iutQxLI14m&index=65&t=4s

Perl 5 Smoke Testing

- Search smoke test reports: http://perl5.test-smoke.org/search
- Smoke test summaries: http://perl.develop-help.com/?b=blead
- First smoke test report on FreeBSD-10.3: http://perl5.test-smoke.org/report/48878
- First smoke test report on FreeBSD-11.0: http://perl5.test-smoke.org/report/50778

CPAN Distributions Cited

- App-cpanminus-reporter https://metacpan.org/pod/App::cpanminus::reporter/

- Perl-Download-FTP https://metacpan.org/pod/Perl::Download::FTP/
- CPAN-cpanminus-reporter-RetainReports https://metacpan.org/pod/CPAN::cpanminus::reporter::RetainReports/
- Test-Against-Dev https://metacpan.org/pod/Test::Against::Dev/
- Test-Smoke https://metacpan.org/pod/Test::Smoke/

CPAN River

- Original Neil Bower *River of CPAN* Post: http://neilb.org/2015/04/20/river-of-cpan.html
- Selected Neil Bower CPAN River Post: http://neilb.org/tag/cpan-river/
- David Golden's Computation of CPAN River: https://github.com/dagolden/zzz-index-cpan-meta

Output from Test::Against::Dev

- 5.27 Test-Against-Dev PSV (Original Format): http://thenceforward.net/perl/misc/cpan-river-1000-perl-5.27-master.psv.gz
- 5.27 Test-Against-Dev PSV (Enhanced Format): http://thenceforward.net/perl/misc/xformat-cpan-river-1000-perl-5.27-master.psv.gz