Further Adventures in QA for the Perl 5 Core Distribution

Author

James E Keenan (`jkeenan@cpan.org`)

Location

Conference in the Cloud

Thursday, June 25 • 12:00pm - 12:20pm EDT

Synopsis

As compilers like gcc and clang advance, they probe deeper for weaknesses in source code, generating additional build-time warnings. These enable Perl 5 contributors to write more accurate and portable code. In this talk we discuss tools we have written in the past year to identify such warnings. We also identify salient developments in QA for the core distribution over the past year.

In our last episode ...

Over the previous three years I've given a number of TPC presentations describing some of the efforts we have been making to ensure the quality of the Perl 5 core distribution.

In 2017 I stressed the importance of smoke-testing Perl on platforms other than Linux and of close analysis of situations where changes to Perl 5 blead had an adverse impact on existing CPAN modules: the **Blead Breaks CPAN** (or BBC) problem.

In 2018 I discussed testing the so-called **CPAN River** against Perl 5 monthly development releases -- the CPAN River being a subset of all of CPAN scheduled for installation in dependency order.

In 2019 I introduced **multisection** -- a tool similar to bisection -- used to determine multiple points in a software project's commit history where its test output changed and hence where bugs may have been introduced. My CPAN distribution Devel-Git-MultiBisect implements this concept.

In this episode … build-time warnings

This year I'm going to talk about one, modest extension to the concept of multisection: **How to determine when a given build-time warning first appeared in the development version of Perl.**

When you build `perl` from source you first run a shell script called `Configure` which probes your machine for its characteristics and which takes options you specify on the command-line and writes a `Makefile` reflecting those characteristics and options. Among the options you can specify on `Configure`'s command-line is the C-compiler you will use to build the `perl` executable.

You then run `make` on particular targets specified in the `Makefile` to build the actual `perl` executable. You'll then go on to call `make test` and perhaps `make install` -- those are beyond our scope for today.

Today our task is to analyze the warnings which your C-compiler emits in the course of running `make`. Suppose that:

- Back on August 20, 2019, I wanted to build `perl` from development version 5.31.3 which was released on that date.
- Suppose further that I was working on FreeBSD version 11 and wanted to compile with `clang`, the default C-compiler for FreeBSD, and specfically with `clang` version 6.0 (which on `Configure`'s command-line is spelled `-Dcc=clang60`).
- And suppose further that I wanted to capture the `STDOUT` and `STDERR` emitted by `make` and redirect them to a gzipped text file for further analysis. I might then wrap up my invocations of `sh ./Configure` and `make` in a Perl script which I would call like this:

```
    perl run-make-for-build-warnings.pl \
        --cc=clang60 \
        --commit=v5.31.3
```

I'm sure any of you in the audience could write such a script in about 20 minutes, so I'm not going to go into its details now. They'll be available on my web site. For now, let's suppose that running this program concludes by pointing us to a file where the output of `make` has been recorded.

```
See output in /tmp/make-output/v5.31.3.freebsd.clang60.make.output.txt.gz
```

Locating build-time warnings in `make` output

Examining this file in an editor, we notice that build-time warnings look like this:

```
Encode.c:1356:5: warning: unused variable 'ix' [-Wunused-variable]
    dXSI32;
    ^
../../XSUB.h:185:20: note: expanded from macro 'dXSI32'
```

```
#define dXSI32 I32 ix = XSANY.any_i32
                   ^
Encode.c:1398:5: warning: unused variable 'ix' [-Wunused-variable]
    dXSI32;
    ^
../../XSUB.h:185:20: note: expanded from macro 'dXSI32'
#define dXSI32 I32 ix = XSANY.any_i32
                   ^
2 warnings generated.
```

If we did this sort of thing often enough, we'd notice a pattern as to how the build-time warnings are composed. We's notice **colon-delimited** lines like this:

```
Encode.c:1356:5: warning: unused variable 'ix' [-Wunused-variable]
```

- The first thing reported is the name of the **source code file** that was being processed when the warning was generated -- or, at least, the **relative path** to that source code file from whatever directory `make` is currently in. In this case:

  ```
  Encode.c
  ```

- Following a colon (`:`) delimiter, the **line number** where the warning was generated:

  ```
  Encode.c:1356
  ```

- Next, the **position** within the line of source code where the warning was generated:

  ```
  Encode.c:1356:5
  ```

- Next, the string `warning`:

  ```
  Encode.c:1356:5: warning
  ```

- Next, the **text** of the warning:

  ```
  Encode.c:1356:5: warning: unused variable 'ix'
  ```

- And finally, the **category** the warning falls into.

  ```
  Encode.c:1356:5: warning: unused variable 'ix' [-Wunused-variable]
  ```

  Here the warnings category is `Wunused-variable`.

  The warnings category is actually the most interesting thing in this line. That's because over time C-compilers evolve and introduce new categories of warnings. So if we were to build `perl` at the very same commit but with a more recent version of `clang` we might see additional warnings falling into newer categories.

We could write a Perl script to tell us which warnings categories were populated during a given build and how many instances of each category there were.

```
$ perl report-build-warnings v5.31.3.freebsd.clang60.make.output.txt.gz
File:  v5.31.3.freebsd.clang60.make.output.txt.gz
```

```
      Wunused-variable                           2
```

I'll have more to say about that program, `report-build-warnings`, in a minute. We could write a second Perl script to dump data from those warnings to our terminal.

```
    $ perl parse-build-warnings v5.31.3.freebsd.clang60.make.output.txt.gz
    File:  v5.31.3.freebsd.clang60.make.output.txt.gz

    [
      {
        char   => 5,
        group  => "Wunused-variable",
        line   => 1356,
        source => "Encode.c",
        text   => "unused variable 'ix'",
      },
      {
        char   => 5,
        group  => "Wunused-variable",
        line   => 1398,
        source => "Encode.c",
        text   => "unused variable 'ix'",
      },
    ]
```

At this point you're probably wondering, *"Is there a CPAN module for programs like this?"*

And, of course, there is. It's called Perl5::Build::Warnings.

Discovering new build-time warnings

In the course of working on the Perl 5 core distribution, the Perl 5 Porters are quite rigorous about making sure that the test suite runs without failures during `make test`. But we're not quite as rigorous about making sure that no new build-time warnings have crept in while running `make`. Some build-time warnings will only appear for the first time when we switch to a newer, pickier version of a C-compiler such as `clang`, `gcc` or `g++`. Nonetheless, when we do notice a new build-time warning, we file a bug ticket about that. Here is an instance of that.

Suppose that about two weeks after we analyzed development version 5.31.3 for build-time warnings, we repeated that process but at a later commit, `9d9546e0fa` (Sep 02 2019).

```
    perl run-make-for-build-warnings.pl \
        --cc=clang60 \
        --commit=9d9546e0fa

    See output in
/tmp/make-output/9d9546e0fa.freebsd.clang60.make.output.txt.gz

    $ report-build-warnings
```

```
/tmp/make-output/9d9546e0fa.freebsd.clang60.make.output.txt.gz
    File:  /tmp/make-output/9d9546e0fa.freebsd.clang60.make.output.txt.gz

      Wdeprecated-declarations                   6
      Wunused-variable                           2
```

What's this? We've picked up 6 instances of a new warnings category, `Wdeprecated-declarations`. Let's get more data:

```
    $ parse-build-warnings
/tmp/make-output/9d9546e0fa.freebsd.clang60.make.output.txt.gz
    File:  /tmp/make-output/9d9546e0fa.freebsd.clang60.make.output.txt.gz

    [
      {
        char   => 38,
        group  => "Wdeprecated-declarations",
        line   => 805,
        source => "./intrpvar.h",
        text   => "'Perl_sv_nosharing' is deprecated",
      },
      {
        char   => 37,
        group  => "Wdeprecated-declarations",
        line   => 806,
        source => "./intrpvar.h",
        text   => "'Perl_sv_nosharing' is deprecated",
      },
      {
        char   => 39,
        group  => "Wdeprecated-declarations",
        line   => 813,
        source => "./intrpvar.h",
        text   => "'Perl_sv_nounlocking' is deprecated",
      },
      {
        char   => 38,
        group  => "Wdeprecated-declarations",
        line   => 805,
        source => "./intrpvar.h",
        text   => "'Perl_sv_nosharing' is deprecated",
      },
      {
        char   => 37,
        group  => "Wdeprecated-declarations",
        line   => 806,
        source => "./intrpvar.h",
        text   => "'Perl_sv_nosharing' is deprecated",
      },
      {
        char   => 39,
        group  => "Wdeprecated-declarations",
        line   => 813,
        source => "./intrpvar.h",
        text   => "'Perl_sv_nounlocking' is deprecated",
      },
      {
        char   => 5,
        group  => "Wunused-variable",
        line   => 1356,
        source => "Encode.c",
        text   => "unused variable 'ix'",
      },
      {
        char   => 5,
        group  => "Wunused-variable",
        line   => 1398,
        source => "Encode.c",
        text   => "unused variable 'ix'",
      },
    ]
```

So, as of commit 9d9546e0fa, we've picked up 6 new build-time warnings, all of them of category `Wdeprecated-declarations`, and all of them in source code file `intrpvar.h`. (This was in fact reported in github issue 17144.)

In which commit did the new build-time warnings first appear?

When a new build-time warning appears, we need to rule out the possibility that this indicates a serious defect in our C-level source code. We want to determine the commit into our `git` repository where the warning first appeared so that we can ask the Author and/or Committer of that commit to investigate further. As you might suspect, we apply the principle of **bisection** to this problem.

Suppose that we wrote a Perl script to detect those points in a large series of commits where the set of warnings emitted during `make` changed from the previous commit. We could then examine those `transitions` to identify the errant commit. Such a Perl script might be called like this:

```
    perl warnings-transitions.pl \
      --git_checkout_dir=/tmp/perl2 \
      --workdir=/tmp/testing/clang" \
      --first=1f6c9461cb3775550f70cd0c579d874dc80c5038 \
      --last=9d9546e0faa646b5770bb5111bf61259779b0fd7 \
      --compiler=clang60 \
      --configure_command='sh ./Configure -des -Dusedevel -Dcc=clang60 -
Duseithreads Doptimize="-O2 -pipe -fstack-protector -fno-strict-aliasing"
1>/dev/null 2>&1' \
      --pattern_sought="intrpvar.h:_:_: warning: 'Perl_sv_nosharing' is
deprecated [Wdeprecated-declarations]"
```

Oooh, that's too much code on one slide. What's really important are these command-line switches:

```
      --first=1f6c9461cb3775550f70cd0c579d874dc80c5038
      --last=9d9546e0faa646b5770bb5111bf61259779b0fd7
      --compiler=clang60
      --configure_command='sh ./Configure -des -Dusedevel -Dcc=clang60 -
Duseithreads Doptimize="-O2 -pipe -fstack-protector -fno-strict-aliasing"
1>/dev/null 2>&1'
```

… where:

- `first` is the full 48-character SHA of the last known "good" commit -- in this case, the SHA for development release 5.31.3 last August;
- `last` is the 48-character SHA of a commit where we have detected new build-time warnings; and
- `compiler` is the name and version of the C-compiler, spelled in a way appropriate to your operating system (*e.g.*, `which clang60` on FreeBSD, but `which clang-6.0` on my Ubuntu Linux).
- `configure_command` is the complete invocation of `sh ./Configure` we're using to build perl at each of `first` and `last`.

We run this program and its output ends like this:

```
    Examining /tmp/testing/clang/57f51a6.make.warnings.rpt.txt
    Likely commit with first instance of warning is
57f51a64ab975e4a9036295a6bc803071e86de43
    See results in:
    /tmp/testing/clang/transitions.clang60.pl
```

If we look in file `transitions.clang60.pl`, we see a hash entry like this:

```
    transitions => [
      {
        newer => {
                   file => "/tmp/testing/clang/57f51a6.make.warnings.rpt.txt",
                   idx => 76,
                   md5_hex => "270e0a9e2db6fac756d25f9ec22b009d",
                 },
        older => {
                   file => "/tmp/testing/clang/b45969d.make.warnings.rpt.txt",
                   idx => 75,
                   md5_hex => "8bad7b60ca32d372a43728026e47b28d",
                 },
      },
    ],
```

We see that in the array of commits between `first` and `last`, there was a change in the set of warnings generated during `make` between index 75 and index 76. If we `diff` the files associated with those commits, we get:

```
    $ diff -w b45969d.make.warnings.rpt.txt \
              57f51a6.make.warnings.rpt.txt
    0a1,6
    > ./intrpvar.h:_:_: warning: 'Perl_sv_nosharing' is deprecated
[Wdeprecated-declarations]
```

```
    > ./intrpvar.h:_:_: warning: 'Perl_sv_nosharing' is deprecated
[Wdeprecated-declarations]
    > ./intrpvar.h:_:_: warning: 'Perl_sv_nounlocking' is deprecated
[Wdeprecated-declarations]
    > ./intrpvar.h:_:_: warning: 'Perl_sv_nosharing' is deprecated
[Wdeprecated-declarations]
    > ./intrpvar.h:_:_: warning: 'Perl_sv_nosharing' is deprecated
[Wdeprecated-declarations]
    > ./intrpvar.h:_:_: warning: 'Perl_sv_nounlocking' is deprecated
[Wdeprecated-declarations]
```

We therefore conclude that it was at commit 57f51a6 that the new build-time warnings were introduced. We comment in the bug ticket requesting that the committer investigate.

And by now you know that there must be a CPAN module to do this. It is Devel::Git::MultiBisect::BuildTransitions, which is my most recent addition to the Devel-Git-MultiBisect CPAN distribution, about which I spoke at last year's Perl conference in Pittsburgh. The program I used above to identify the commit where build-time warnings were introduced, warnings-transitions.pl, is included in the `xt/` directory of that distribution on CPAN.

Other topics in Perl core distribution QA 2019-2020

At this point I'm going to pause to mention some other aspects of QA for the Perl 5 core distribution that have emerged in the past year. We can discuss these in the time remaining or you can buttonhole me in the virtual hallway track: `irc.perl.org #p5p`.

- "Ordinary" bisection of the core distribution via `Porting/bisect.pl`

  A tool we have long used to identify points where bugs were introduced (or perhaps cleared up) within the core distribution. More EXAMPLES have been added to the documentation: `perldoc Porting/bisect-runner.pl`.

- Fuzzing

  Worthy of a talk in its own right. Over the past few years fuzzing has generated more bug tickets than any other technique. Way outside my range of expertise, but I can point you to the best practitioners.

- Centralized QA tools

  Use of Travis, Appveyor and Github Actions in configurations managed by Perl 5 Porters.

- All volunteer

The biggest challenge to our QA efforts is that it's all-volunteer. When a particular volunteer goes away, too often so does a particular form of QA for Perl 5.

- Perl 7 and beyond

  What will change when we increment Perl's major version from `5` to `7`? And what will remain the same?