NAME

strictures – Turn on strict and make most warnings fatal

SYNOPSIS

```
use strictures 2;
```

is equivalent to

```
use strict;
use warnings FATAL => 'all';
use warnings NONFATAL => qw(
  exec
  recursion
  internal
  malloc
  newline
  experimental
  deprecated
  portable
);
no warnings 'once';
```

except when called from a file which matches:

```
(caller)[1] =~ /^(?:t|xt|lib|blib)[\\\/]/
```

and when either `.git`, `.svn`, `.hg`, or `.bzr` is present in the current directory (with the intention of only forcing extra tests on the author side) — or when `.git`, `.svn`, `.hg`, or `.bzr` is present two directories up along with `dist.ini` (which would indicate we are in a `dzil test` operation, via Dist::Zilla) — or when the `PERL_STRICTURES_EXTRA` environment variable is set, in which case it also does the equivalent of

```
no indirect 'fatal';
no multidimensional;
no bareword::filehandles;
```

Note that `PERL_STRICTURES_EXTRA` may at some point add even more tests, with only a minor version increase, but any changes to the effect of `use strictures` in normal mode will involve a major version bump.

If any of the extra testing modules are not present, strictures will complain loudly, once, via `warn()`, and then shut up. But you really should consider installing them, they're all great anti-footgun tools.

DESCRIPTION

I've been writing the equivalent of this module at the top of my code for about a year now. I figured it was time to make it shorter.

Things like the importer in `use Moose` don't help me because they turn warnings on but don't make them fatal — which from my point of view is useless because I want an exception to tell me my code isn't warnings-clean.

Any time I see a warning from my code, that indicates a mistake.

Any time my code encounters a mistake, I want a crash — not spew to STDERR and then unknown (and probably undesired) subsequent behaviour.

I also want to ensure that obvious coding mistakes, like indirect object syntax (and not so obvious mistakes that cause things to accidentally compile as such) get caught, but not at the cost of an XS dependency and not at the cost of blowing things up on another machine.

Therefore, strictures turns on additional checking, but only when it thinks it's running in a test file in a VCS checkout — although if this causes undesired behaviour this can be overridden by setting the `PERL_STRICTURES_EXTRA` environment variable.

If additional useful author side checks come to mind, I'll add them to the `PERL_STRICTURES_EXTRA` code path only — this will result in a minor version increase (e.g. 1.000000 to 1.001000 (1.1.0) or similar). Any fixes only to the mechanism of this code will result in a sub-version increase (e.g. 1.000000 to 1.000001 (1.0.1)).

CATEGORY SELECTIONS

strictures does not enable fatal warnings for all categories.

exec
: Includes a warning that can cause your program to continue running unintentionally after an internal fork. Not safe to fatalize.

recursion
: Infinite recursion will end up overflowing the stack eventually anyway.

internal
: Triggers deep within perl, in places that are not safe to trap.

malloc
: Triggers deep within perl, in places that are not safe to trap.

newline
: Includes a warning for using stat on a valid but suspect filename, ending in a newline.

experimental
: Experimental features are used intentionally.

deprecated
: Deprecations will inherently be added to in the future in unexpected ways, so making them fatal won't be reliable.

portable
: Doesn't indicate an actual problem with the program, only that it may not behave properly if run on a different machine.

once
: Can't be fatalized. Also triggers very inconsistently, so we just disable it.

VERSIONS

Depending on the version of strictures requested, different warnings will be enabled. If no specific version is requested, the current version's behavior will be used. Versions can be requested using perl's standard mechanism:

```
use strictures 2;
```

Or, by passing in a `version` option:

```
use strictures version => 2;
```

VERSION 2

Equivalent to:

```
use strict;
use warnings FATAL => 'all';
use warnings NONFATAL => qw(
  exec
  recursion
  internal
  malloc
  newline
  experimental
  deprecated
  portable
);
```

```
no warnings 'once';

# and if in dev mode:
no indirect 'fatal';
no multidimensional;
no bareword::filehandles;
```

Additionally, any warnings created by modules using warnings::register or `warnings::register_categories()` will not be fatalized.

VERSION 1

Equivalent to:

```
use strict;
use warnings FATAL => 'all';
# and if in dev mode:
no indirect 'fatal';
no multidimensional;
no bareword::filehandles;
```

METHODS

import

This method does the setup work described above in "DESCRIPTION". Optionally accepts a `version` option to request a specific version's behavior.

VERSION

This method traps the `strictures->VERSION(1)` call produced by a use line with a version number on it and does the version check.

EXTRA TESTING RATIONALE

Every so often, somebody complains that they're deploying via `git pull` and that they don't want strictures to enable itself in this case — and that setting `PERL_STRICTURES_EXTRA` to 0 isn't acceptable (additional ways to disable extra testing would be welcome but the discussion never seems to get that far).

In order to allow us to skip a couple of stages and get straight to a productive conversation, here's my current rationale for turning the extra testing on via a heuristic:

The extra testing is all stuff that only ever blows up at compile time; this is intentional. So the oft-raised concern that it's different code being tested is only sort of the case — none of the modules involved affect the final optree to my knowledge, so the author gets some additional compile time crashes which he/she then fixes, and the rest of the testing is completely valid for all environments.

The point of the extra testing — especially `no indirect` — is to catch mistakes that newbie users won't even realise are mistakes without help. For example,

```
foo { ... };
```

where foo is an & prototyped sub that you forgot to import — this is pernicious to track down since all *seems* fine until it gets called and you get a crash. Worse still, you can fail to have imported it due to a circular require, at which point you have a load order dependent bug which I've seen before now *only* show up in production due to tiny differences between the production and the development environment. I wrote <http://shadow.cat/blog/matt-s-trout/indirect-but-still-fatal/> to explain this particular problem before strictures itself existed.

As such, in my experience so far strictures' extra testing has *avoided* production versus development differences, not caused them.

Additionally, strictures' policy is very much "try and provide as much protection as possible for newbies — who won't think about whether there's an option to turn on or not" — so having only the environment variable is not sufficient to achieve that (I get to explain that you need to add `use strict` at least once a week on freenode #perl — newbies sometimes completely skip steps because they don't understand that

that step is important).

I make no claims that the heuristic is perfect — it's already been evolved significantly over time, especially for 1.004 where we changed things to ensure it only fires on files in your checkout (rather than strictures-using modules you happened to have installed, which was just silly). However, I hope the above clarifies why a heuristic approach is not only necessary but desirable from a point of view of providing new users with as much safety as possible, and will allow any future discussion on the subject to focus on "how do we minimise annoyance to people deploying from checkouts intentionally".

SEE ALSO

- indirect
- multidimensional
- bareword::filehandles

COMMUNITY AND SUPPORT

IRC channel

irc.perl.org #toolchain

(or bug 'mst' in query on there or freenode)

Git repository

Gitweb is on http://git.shadowcat.co.uk/ and the clone URL is:

```
git clone git://git.shadowcat.co.uk/p5sagit/strictures.git
```

The web interface to the repository is at:

```
http://git.shadowcat.co.uk/gitweb/gitweb.cgi?p=p5sagit/strictures.git
```

AUTHOR

mst - Matt S. Trout (cpan:MSTROUT) <mst@shadowcat.co.uk>

CONTRIBUTORS

Karen Etheridge (cpan:ETHER) <ether@cpan.org>

Mithaldu - Christian Walde (cpan:MITHALDU) <walde.christian@gmail.com>

haarg - Graham Knop (cpan:HAARG) <haarg@haarg.org>

COPYRIGHT

LICENSE