NAME

common::sense - save a tree AND a kitten, use common::sense!

SYNOPSIS

```
use common::sense;

# Supposed to be mostly the same, with much lower memory usage, as:

# use utf8;
# use strict qw(vars subs);
# use feature qw(say state switch);
# use feature qw(unicode_strings unicode_eval current_sub fc evalbytes);
# no feature qw(array_base);
# no warnings;
# use warnings qw(FATAL closed threads internal debugging pack
# portable prototype inplace io pipe unpack malloc
# glob digit printf layer reserved taint closure
# semicolon);
# no warnings qw(exec newline unopened);
```

DESCRIPTION

```
“Nothing is more fairly distributed than common sense: no one thinks
he needs more of it than he already has.”

– René Descartes
```

This module implements some sane defaults for Perl programs, as defined by two typical (or not so typical - use your common sense) specimens of Perl coders. In fact, after working out details on which warnings and strict modes to enable and make fatal, we found that we (and our code written so far, and others) fully agree on every option, even though we never used warnings before, so it seems this module indeed reflects a ‘‘common’’ sense among some long-time Perl coders.

The basic philosophy behind the choices made in common::sense can be summarised as: ‘‘enforcing strict policies to catch as many bugs as possible, while at the same time, not limiting the expressive power available to the programmer’’.

Two typical examples of how this philosophy is applied in practise is the handling of uninitialised and malloc warnings:

uninitialised

`undef` is a well-defined feature of perl, and enabling warnings for using it rarely catches any bugs, but considerably limits you in what you can do, so uninitialised warnings are disabled.

malloc

Freeing something twice on the C level is a serious bug, usually causing memory corruption. It often leads to side effects much later in the program and there are no advantages to not reporting this, so malloc warnings are fatal by default.

Unfortunately, there is no fine-grained warning control in perl, so often whole groups of useful warnings had to be excluded because of a single useless warning (for example, perl puts an arbitrary limit on the length of text you can match with some regexes before emitting a warning, making the whole `regexp` category useless).

What follows is a more thorough discussion of what this module does, and why it does it, and what the advantages (and disadvantages) of this approach are.

RATIONALE

use utf8

While it’s not common sense to write your programs in UTF-8, it’s quickly becoming the most common encoding, is the designated future default encoding for perl sources, and the most convenient

encoding available (you can do really nice quoting tricks...). Experience has shown that our programs were either all pure ascii or utf-8, both of which will stay the same.

There are few drawbacks to enabling UTF-8 source code by default (mainly some speed hits due to bugs in older versions of perl), so this module enables UTF-8 source code encoding by default.

use strict qw(subs vars)

Using `use strict` is definitely common sense, but `use strict 'refs'` definitely overshoots its usefulness. After almost two decades of Perl hacking, we decided that it does more harm than being useful. Specifically, constructs like these:

```
@{ $var->[0] }
```

Must be written like this (or similarly), when `use strict 'refs'` is in scope, and `$var` can legally be `undef`:

```
@{ $var->[0] || [] }
```

This is annoying, and doesn't shield against obvious mistakes such as using `""`, so one would even have to write (at least for the time being):

```
@{ defined $var->[0] ? $var->[0] : [] }
```

... which nobody with a bit of common sense would consider writing: clear code is clearly something else.

Curiously enough, sometimes perl is not so strict, as this works even with `use strict` in scope:

```
for (@{ $var->[0] }) { ...
```

If that isn't hypocrisy! And all that from a mere program!

use feature qw(say state given ...)

We found it annoying that we always have to enable extra features. If something breaks because it didn't anticipate future changes, so be it. 5.10 broke almost all our XS modules and nobody cared either (or at least I know of nobody who really complained about gratuitous changes - as opposed to bugs).

Few modules that are not actively maintained work with newer versions of Perl, regardless of use feature or not, so a new major perl release means changes to many modules - new keywords are just the tip of the iceberg.

If your code isn't alive, it's dead, Jim - be an active maintainer.

But nobody forces you to use those extra features in modules meant for older versions of perl - common::sense of course works there as well. There is also an important other mode where having additional features by default is useful: commandline hacks and internal use scripts: See ''much reduced typing'', below.

There is one notable exception: `unicode_eval` is not enabled by default. In our opinion, `use feature` had one main effect - newer perl versions don't value backwards compatibility and the ability to write modules for multiple perl versions much, after all, you can use feature.

`unicode_eval` doesn't add a new feature, it breaks an existing function.

no warnings, but a lot of new errors

Ah, the dreaded warnings. Even worse, the horribly dreaded `-w` switch: Even though we don't care if other people use warnings (and certainly there are useful ones), a lot of warnings simply go against the spirit of Perl.

Most prominently, the warnings related to `undef`. There is nothing wrong with `undef`: it has well-defined semantics, it is useful, and spitting out warnings you never asked for is just evil.

The result was that every one of our modules did `no warnings` in the past, to avoid somebody accidentally using and forcing his bad standards on our code. Of course, this switched off all warnings,

even the useful ones. Not a good situation. Really, the `-w` switch should only enable warnings for the main program only.

Funnily enough, perllexwarn(1) explicitly mentions `-w` (and not in a favourable way, calling it outright "wrong"), but standard utilities, such as prove, or MakeMaker when running `make test`, still enable them blindly.

For version 2 of common::sense, we finally sat down a few hours and went through *every single warning message*, identifying - according to common sense - all the useful ones.

This resulted in the rather impressive list in the SYNOPSIS. When we weren't sure, we didn't include the warning, so the list might grow in the future (we might have made a mistake, too, so the list might shrink as well).

Note the presence of `FATAL` in the list: we do not think that the conditions caught by these warnings are worthy of a warning, we *insist* that they are worthy of *stopping* your program, *instantly*. They are *bugs*!

Therefore we consider `common::sense` to be much stricter than `use warnings`, which is good if you are into strict things (we are not, actually, but these things tend to be subjective).

After deciding on the list, we ran the module against all of our code that uses `common::sense` (that is almost all of our code), and found only one occurrence where one of them caused a problem: one of elmex's (unreleased) modules contained:

```
$fmt =~ s/([^\s\[]*)\[( [^\]]* )\]/\x0$1\x1$2\x0/xgo;
```

We quickly agreed that indeed the code should be changed, even though it happened to do the right thing when the warning was switched off.

much reduced typing

Especially with version 2.0 of common::sense, the amount of boilerplate code you need to add to get *this* policy is daunting. Nobody would write this out in throwaway scripts, commandline hacks or in quick internal-use scripts.

By using common::sense you get a defined set of policies (ours, but maybe yours, too, if you accept them), and they are easy to apply to your scripts: typing `use common::sense;` is even shorter than `use warnings; use strict; use feature ....`

And you can immediately use the features of your installed perl, which is more difficult in code you release, but not usually an issue for internal-use code (downgrades of your production perl should be rare, right?).

mucho reduced memory usage

Just using all those pragmas mentioned in the SYNOPSIS together wastes <blink>***776 kilobytes***</blink> of precious memory in my perl, for *every single perl process using our code*, which on our machines, is a lot. In comparison, this module only uses ***four*** kilobytes (I even had to write it out so it looks like more) of memory on the same platform.

The money/time/effort/electricity invested in these gigabytes (probably petabytes globally!) of wasted memory could easily save 42 trees, and a kitten!

Unfortunately, until everybody applies more common sense, there will still often be modules that pull in the monster pragmas. But one can hope...

THERE IS NO 'no common::sense'!!!! !!!! !!

This module doesn't offer an unimport. First of all, it wastes even more memory, second, and more importantly, who with even a bit of common sense would want no common sense?

STABILITY AND FUTURE VERSIONS

Future versions might change just about everything in this module. We might test our modules and upload new ones working with newer versions of this module, and leave you standing in the rain because we didn't tell you. In fact, we did so when switching from 1.0 to 2.0, which enabled gobs of warnings, and made

them FATAL on top.

Maybe we will load some nifty modules that try to emulate `say` or so with perls older than 5.10 (this module, of course, should work with older perl versions - supporting 5.8 for example is just common sense at this time. Maybe not in the future, but of course you can trust our common sense to be consistent with, uhm, our opinion).

WHAT OTHER PEOPLE HAD TO SAY ABOUT THIS MODULE

apeiron

```
"... wow"
"I hope common::sense is a joke."
```

crab

```
"i wonder how it would be if joerg schilling wrote perl modules."
```

Adam Kennedy

```
"Very interesting, efficient, and potentially something I'd use all the time."
[...]
"So no common::sense for me, alas."
```

H.Merijn Brand

```
"Just one more reason to drop JSON::XS from my distribution list"
```

Pista Palo

```
"Something in short supply these days..."
```

Steffen Schwigon

```
"This module is quite for sure *not* just a repetition of all the other
'use strict, use warnings'-approaches, and it's also not the opposite.
[...] And for its chosen middle-way it's also not the worst name ever.
And everything is documented."
```

BKB

```
"[Deleted - thanks to Steffen Schwigon for pointing out this review was
in error.]"
```

Somni

```
"the arrogance of the guy"
"I swear he tacked somenoe else's name onto the module
just so he could use the royal 'we' in the documentation"
```

Anonymous Monk

```
"You just gotta love this thing, its got META.json!!!"
```

dngor

```
"Heh. '"<elmex at ta-sa.org>"' The quotes are semantic
distancing from that e-mail address."
```

Jerad Pierce

```
"Awful name (not a proper pragma), and the SYNOPSIS doesn't tell you
anything either. Nor is it clear what features have to do with "common
sense" or discipline."
```

acme

```
"THERE IS NO 'no common::sense'!!!! !!!! !!"
```

apeiron (meta-comment about us commenting^Wquoting his comment)

```
"How about quoting this: get a clue, you fucktarded amoeba."
```

quanth

```
"common sense is beautiful, json::xs is fast, Anyevent, EV are fast and
furious. I love mlehmannware ;)"
```

apeiron

```
"... it's mlehmann's view of what common sense is. His view of common
sense is certainly uncommon, insofar as anyone with a clue disagrees
with him."
```

apeiron (another meta-comment)

```
"apeiron wonders if his little informant is here to steal more quotes"
```

ew73

```
"... I never got past the SYNOPSIS before calling it shit."
[...]
How come no one ever quotes me. :("
```

chip (not willing to explain his cryptic questions about links in Changes files)

```
"I'm willing to ask the question I've asked. I'm not willing to go
through the whole dance you apparently have choreographed. Either
answer the completely obvious question, or tell me to fuck off again."
```

FREQUENTLY ASKED QUESTIONS

Or frequently-come-up confusions.

Is this module meant to be serious?

Yes, we would have put it under the `Acme::` namespace otherwise.

But the manpage is written in a funny/stupid/... way?

This was meant to make it clear that our common sense is a subjective thing and other people can use their own notions, taking the steam out of anybody who might be offended (as some people are always offended no matter what you do).

This was a failure.

But we hope the manpage still is somewhat entertaining even though it explains boring rationale.

Why do you impose your conventions on my code?

For some reason people keep thinking that `common::sense` imposes process-wide limits, even though the SYNOPSIS makes it clear that it works like other similar modules - i.e. only within the scope that `use`s them.

So, no, we don’t - nobody is forced to use this module, and using a module that relies on common::sense does not impose anything on you.

Why do you think only your notion of

common::sense is valid?" 4 Well, we don’t, and have clearly written this in the documentation to every single release. We were just faster than anybody else w.r.t. to grabbing the namespace.

But everybody knows that you have to use strict and use warnings, why do you disable them?

Well, we don’t do this either - we selectively disagree with the usefulness of some warnings over others. This module is aimed at experienced Perl programmers, not people migrating from other languages who might be surprised about stuff such as `undef`. On the other hand, this does not exclude the usefulness of this module for total newbies, due to its strictness in enforcing policy, while at the same time not limiting the expressive power of perl.

This module is considerably *more* strict than the canonical `use strict; use warnings`, as it makes all its warnings fatal in nature, so you can not get away with as many things as with the canonical approach.

This was not implemented in version 1.0 because of the daunting number of warning categories and the difficulty in getting exactly the set of warnings you wish (i.e. look at the SYNOPSIS in how complicated it is to get a specific set of warnings - it is not reasonable to put this into every module, the maintenance effort would be enormous).

But many modules `use strict` or `use warnings`, so the memory savings do not apply?
I suddenly feel sad...

But yes, that's true. Fortunately `common::sense` still uses only a miniscule amount of RAM.

But it adds another dependency to your modules!
It's a fact, yeah. But it's trivial to install, most popular modules have many more dependencies. And we consider dependencies a good thing - it leads to better APIs, more thought about interworking of modules and so on.

Why do you use JSON and not YAML for your META.yml?
This is not true - YAML supports a large subset of JSON, and this subset is what META.yml is written in, so it would be correct to say ''the META.yml is written in a common subset of YAML and JSON''.

The META.yml follows the YAML, JSON and META.yml specifications, and is correctly parsed by CPAN, so if you have trouble with it, the problem is likely on your side.

But! But!
Yeah, we know.

AUTHOR

`Marc Lehmann <schmorp@schmorp.de>`
`http://home.schmorp.de/`

`Robin Redeker, "<elmex at ta-sa.org>".`