# Using Strict

A talk on everyone's (not so) favourite pragma.

Paul Fenwick
<pjf@perltraining.com.au>

13th February, 2001

# Who uses strict?

Let's see a show of hands.

# What does strict do?

- Requires all variables be properly scoped.
- Forbids symbolic references.
- Dis-allows (most) barewords.

Many people consider `use strict` to be invaluable for all large and most small projects.

# Who doesn't use strict?

Anyone? Anyone?

# Reasons for not using strict.

- Makes Perl code more wordy and longer to type.

- Prevents you from features you might want to use.

- Lack of barewords makes poetry much harder to write in production code.

# Strict is not needed if you don't make mistakes.

> When I was "growing up" as a PERL boy, I never used strict either, and I realize now that NONE of the errors that I have made in my scripting could have cought or found by strict.
> — [sic] *"bk"*

# The biggest problem of not using strict.

Other Perl programmers make sarcastic comments and try to convince you that you *should* be using strict.

> All real Perl experts recommend the use of 'use strict'. Anyone who doesn't isn't as much of an expert as they might think.
>
> — *Dave Cross*, author of Data Munging with Perl

# Intolerance leads to frustration. . .

> No, but its true– strict really does suck. I hate it. Its gay. Dont tell me ehat to do. And nobody wants your to be back, either.
> — [sic] *"bk"*

# The Root of the Problem

No good way of expressing your true feelings about strict . . . until now!

# Enter Acme::USIG

Richard Clamp's "Use Strict Is Gay" module allows you to express your true feelings about strict.

```
#!/usr/bin/perl
use Acme::USIG;

use strict is gay;

$foo = 1;
print $Foo; # Works great!
```

# Acme::USIG for all.

```
#!/usr/bin/perl -w
use Acme::USIG;

use strict is cool;

$foo = 1;
print $Foo; # Dies with some pesky error
```

# But why stop there?

The greatest feature of not using strict is the ability to use variables without needing to declare them.

Alas, we cannot do the same for subroutines.

```
print "Here is my diary entry today.\n\n";
print dairy();

# ...

sub diary {
        return "Today I fed the chickens.\n";
}
__END__

$ perl diary.pl
Undefined subroutine &main::dairy called at diary.
```

# But using Symbol::Approx::Sub. . .

Dave Cross' `Symbol::Approx::Sub` module allows you to be even more lazy with subroutines than you are with variables.

```
use Symbol::Approx::Sub;

print "Here is my diary entry today.\n\n";
print dairy();

# ...

sub diary {
        return "Today I fed the chickens.\n";
}

$ perl diary.pl
Here is my diary entry today.

Today I fed the chickens.
```

# Uses for Symbol::Approx::Sub

- Large projects with many developers with various levels of spelling aptitude.
- Legacy code when either you and/or the original author couldn't spell.
- Perl poetry in Olde English
- Drunken coding

# Putting it all together. . .

```
#!/usr/bin/perl
use Acme::USIG;
use Symbol::Approx::Sub;

use strict is gay;

foreach (<*>) {

Guinness and chips,
Kilkenny and pie,
beer(@the_pub),
is cool;
}

&PeArl_count;

sub BEER { open FILE,$_;
$perl += grep(/perl/, <FILE>); }
sub perl_count {
print "Perl rulez $perl times!!!!!!!!!!\n"; }
```

# References

**Use strict is gay**, message-board thread, http://www.cookwood.com/cgi-bin/lcastro/perlbbs.pl?read=4700

**Acme::USIG**, CPAN module, http://search.cpan.org/doc/RCLAMP/Acme-USIG-1.01/lib/Acme/USIG.pm

**Symbol::Approx::Sub**, CPAN module, http://search.cpan.org/doc/DAVECROSS/Symbol-Approx-Sub-00/Sub.pm