NAME

B::Lint - Perl lint

SYNOPSIS

perl -MO=Lint[,OPTIONS] foo.pl

DESCRIPTION

The B::Lint module is equivalent to an extended version of the **-w** option of **perl**. It is named after the program *lint* which carries out a similar process for C programs.

OPTIONS AND LINT CHECKS

Option words are separated by commas (not whitespace) and follow the usual conventions of compiler backend options. Following any options (indicated by a leading **-**) come lint check arguments. Each such argument (apart from the special **all** and **none** options) is a word representing one possible lint check (turning on that check) or is **no-foo** (turning off that check). Before processing the check arguments, a standard list of checks is turned on. Later options override earlier ones. Available options are:

magic-diamond

Produces a warning whenever the magic <> readline is used. Internally it uses perl's two-argument open which itself treats filenames with special characters specially. This could allow interestingly named files to have unexpected effects when reading.

```
% touch 'rm *|'
% perl -pe 1
```

The above creates a file named `rm *|`. When perl opens it with <> it actually executes the shell program `rm *`. This makes <> dangerous to use carelessly.

context

Produces a warning whenever an array is used in an implicit scalar context. For example, both of the lines

```
$foo = length(@bar);
$foo = @bar;
```

will elicit a warning. Using an explicit **scalar()** silences the warning. For example,

```
$foo = scalar(@bar);
```

implicit-read and **implicit-write**

These options produce a warning whenever an operation implicitly reads or (respectively) writes to one of Perl's special variables. For example, **implicit-read** will warn about these:

```
/foo/;
```

and **implicit-write** will warn about these:

```
s/foo/bar/;
```

Both **implicit-read** and **implicit-write** warn about this:

```
for (@a) { ... }
```

bare-subs

This option warns whenever a bareword is implicitly quoted, but is also the name of a subroutine in the current package. Typical mistakes that it will trap are:

```
use constant foo => 'bar';
@a = ( foo => 1 );
$b{foo} = 2;
```

Neither of these will do what a naive user would expect.

dollar-underscore

This option warns whenever $_ is used either explicitly anywhere or as the implicit argument of a **print** statement.

private-names

This option warns on each use of any variable, subroutine or method name that lives in a non-current package but begins with an underscore ("_"). Warnings aren't issued for the special case of the single character name "_" by itself (e.g. $_ and @_).

undefined-subs

This option warns whenever an undefined subroutine is invoked. This option will only catch explicitly invoked subroutines such as `foo()` and not indirect invocations such as `&$subref()` or `$obj->meth()`. Note that some programs or modules delay definition of subs until runtime by means of the AUTOLOAD mechanism.

regexp-variables

This option warns whenever one of the regexp variables $`, $& or $' is used. Any occurrence of any of these variables in your program can slow your whole program down. See *perlre* for details.

all

Turn all warnings on.

none

Turn all warnings off.

NON LINT-CHECK OPTIONS

-u Package

Normally, Lint only checks the main code of the program together with all subs defined in package main. The **-u** option lets you include other package names whose subs are then checked by Lint.

EXTENDING LINT

Lint can be extended by with plugins. Lint uses *Module::Pluggable* to find available plugins. Plugins are expected but not required to inform Lint of which checks they are adding.

The `B::Lint->register_plugin( MyPlugin => \@new_checks )` method adds the list of `@new_checks` to the list of valid checks. If your module wasn't loaded by *Module::Pluggable* then your class name is added to the list of plugins.

You must create a `match( \%checks )` method in your plugin class or one of its parents. It will be called on every op as a regular method call with a hash ref of checks as its parameter.

The class methods `B::Lint->file` and `B::Lint->line` contain the current filename and line number.

```
package Sample;
use B::Lint;
B::Lint->register_plugin( Sample => [ 'good_taste' ] );

sub match {
```

```
            my ( $op, $checks_href ) = shift @_;
            if ( $checks_href->{good_taste} ) {
                ...
            }
        }
```

TODO

while(<FH>) stomps $_

strict oo

unchecked system calls

more tests, validate against older perls

BUGS

This is only a very preliminary version.

AUTHOR

Malcolm Beattie, mbeattie@sable.ox.ac.uk.

ACKNOWLEDGEMENTS

Sebastien Aperghis-Tramoni - bug fixes