If you are writing a module to expand an already existing set of modules, please coordinate with the author of the package. It helps if you follow the same naming scheme and module interaction scheme as the original author.

- Try to design the new module to be easy to extend and reuse.

  Try to `use warnings;` (or `use warnings qw(...);`). Remember that you can add `no warnings qw(...);` to individual blocks of code that need less warnings.

  Use blessed references. Use the two argument form of bless to bless into the class name given as the first parameter of the constructor, e.g.,:

```
sub new {
    my $class = shift;
    return bless {}, $class;
}
```

  or even this if you'd like it to be used as either a static or a virtual method.

```
sub new {
    my $self  = shift;
    my $class = ref($self) || $self;
    return bless {}, $class;
}
```

  Pass arrays as references so more parameters can be added later (it's also faster). Convert functions into methods where appropriate. Split large methods into smaller more flexible ones. Inherit methods from other modules if appropriate.

  Avoid class name tests like: `die "Invalid" unless ref $ref eq 'FOO'`. Generally you can delete the `eq 'FOO'` part with no harm at all. Let the objects look after themselves! Generally, avoid hard-wired class names as far as possible.

  Avoid `$r->Class::func()` where using `@ISA=qw(... Class ...)` and `$r->func()` would work (see *perlbot* for more details).

  Use autosplit so little used or newly added functions won't be a burden to programs that don't use them. Add test functions to the module after __END__ either using AutoSplit or by saying:

```
eval join('',<main::DATA>) || die $@ unless caller();
```

  Does your module pass the 'empty subclass' test? If you say `@SUBCLASS::ISA = qw(YOURCLASS);` your applications should be able to use SUBCLASS in exactly the same way as YOURCLASS. For example, does your application still work if you change: `$obj = new YOURCLASS;` into: `$obj = new SUBCLASS;` ?

  Avoid keeping any state information in your packages. It makes it difficult for multiple other packages to use yours. Keep state information in objects.

  Always use **-w**.

  Try to `use strict;` (or `use strict qw(...);`). Remember that you can add `no strict qw(...);` to individual blocks of code that need less strictness.

  Always use **-w**.

  Follow the guidelines in the perlstyle(1) manual.

  Always use **-w**.

- Some simple style guidelines

  The perlstyle manual supplied with Perl has many helpful points.

  Coding style is a matter of personal taste. Many people evolve their style over several years as they learn what helps them write and maintain good code. Here's one set of assorted suggestions that seem to be widely used by experienced developers: