NAME

perlpodstyle - Perl POD style guide

DESCRIPTION

These are general guidelines for how to write POD documentation for Perl scripts and modules, based on general guidelines for writing good Unix man pages. All of these guidelines are, of course, optional, but following them will make your documentation more consistent with other documentation on the system.

Here are some simple guidelines for markup; see *perlpod* for details.

bold (B<>)

> **NOTE: Use extremely rarely.** Do *not* use bold for emphasis; that's what italics are for. Restrict bold for notices like **NOTE:** and **WARNING:**. However, program arguments and options--but *not* their names!--are written in bold (using B<>) to distinguish the **-f** command-line option from the `-f` filetest operator.

italic (I<>)

> Use italic to emphasize text, like *this*. Function names are traditionally written in italics; if you write a function as function(), Pod::Man will take care of this for you. Names of programs, including the name of the program being documented, are conventionally written in italics (using I<>) wherever they occur in normal roman text.

code (C<>)

> Literal code should be in C<>. However metasyntactic placeholders should furthermore be nested in "italics" (actually, oblique) like C<I<>>. That way C<accept(I<NEWSOCKET>, < GENERICSOCKET>)> renders as `accept(`*`NEWSOCKET, GENERICSOCKET`*`)`.

files (F<>)

> Filenames, whether absolute or relative, are specified with the F<> markup. This will render as italics, but has other semantic connotations.

References to other man pages should be in the form "manpage(section)" or "`L<manpage(section)>`", and Pod::Man will automatically format those appropriately. Both will render as *manpage*(section). The second form, with L<>, is used to request that a POD formatter make a link to the man page if possible. As an exception, one normally omits the section when referring to module documentation because not all systems place it in section 3, although that is the default. You may use `L<Module::Name>` for module references instead, but this is optional because the translators are supposed to recognize module references in pod, just as they do variable references like $foo and such.

References to other programs or functions are normally in the form of man page references so that cross-referencing tools can provide the user with links and the like. It's possible to overdo this, though, so be careful not to clutter your documentation with too much markup. References to other programs that are not given as man page references should be enclosed in italics via I<>.

Major headers should be set out using a `=head1` directive, and are historically written in the rather startling ALL UPPER CASE format; this is not mandatory, but it's strongly recommended so that sections have consistent naming across different software packages. The translators are supposed to translate all caps into small caps. Minor headers may be included using `=head2`, and are typically in mixed case.

The standard sections of a manual page are:

NAME

> Mandatory section; should be a comma-separated list of programs or functions documented by this POD page, such as:

```
foo, bar - programs to do something
```

Manual page indexers are often extremely picky about the format of this section, so don't put anything in it except this line. Every program or function documented by this POD page should be listed, separated by a comma and a space. For a Perl module, just give the module name. A single dash, and only a single dash, should separate the list of programs or functions from the description. Do not use any markup such as C<> or I<> anywhere in this line. Functions should not be qualified with `()` or the like. The description should ideally fit on a single line, even if a man program replaces the dash with a few tabs.

SYNOPSIS

A short usage summary for programs and functions. This section is mandatory for section 3 pages. For Perl module documentation, it's usually convenient to have the contents of this section be a verbatim block showing some (brief) examples of typical ways the module is used.

DESCRIPTION

Extended description and discussion of the program or functions, or the body of the documentation for man pages that document something else. If particularly long, it's a good idea to break this up into subsections `=head2` directives like:

```
=head2 Normal Usage

=head2 Advanced Features

=head2 Writing Configuration Files
```

or whatever is appropriate for your documentation.

For a module, this is generally where the documentation of the interfaces provided by the module goes, usually in the form of a list with an `=item` for each interface. Depending on how many interfaces there are, you may want to put that documentation in separate METHODS, FUNCTIONS, CLASS METHODS, or INSTANCE METHODS sections instead and save the DESCRIPTION section for an overview.

OPTIONS

Detailed description of each of the command-line options taken by the program. This should be separate from the description for the use of parsers like *Pod::Usage*. This is normally presented as a list, with each option as a separate `=item`. The specific option string should be enclosed in B<>. Any values that the option takes should be enclosed in I<>. For example, the section for the option **--section**=*manext* would be introduced with:

```
=item B<--section>=I<manext>
```

Synonymous options (like both the short and long forms) are separated by a comma and a space on the same `=item` line, or optionally listed as their own item with a reference to the canonical name. For example, since **--section** can also be written as **-s**, the above would be:

```
=item B<-s> I<manext>, B<--section>=I<manext>
```

Writing the short option first is recommended because it's easier to read. The long option is long enough to draw the eye to it anyway and the short option can otherwise get lost in visual noise.

RETURN VALUE

What the program or function returns, if successful. This section can be omitted for programs whose precise exit codes aren't important, provided they return 0 on success and non-zero on failure as is standard. It should always be present for functions. For modules, it may be useful to summarize return values from the module interface here, or it may be more useful to

discuss return values separately in the documentation of each function or method the module provides.

ERRORS

Exceptions, error return codes, exit statuses, and errno settings. Typically used for function or module documentation; program documentation uses DIAGNOSTICS instead. The general rule of thumb is that errors printed to STDOUT or STDERR and intended for the end user are documented in DIAGNOSTICS while errors passed internal to the calling program and intended for other programmers are documented in ERRORS. When documenting a function that sets errno, a full list of the possible errno values should be given here.

DIAGNOSTICS

All possible messages the program can print out and what they mean. You may wish to follow the same documentation style as the Perl documentation; see perldiag(1) for more details (and look at the POD source as well).

If applicable, please include details on what the user should do to correct the error; documenting an error as indicating "the input buffer is too small" without telling the user how to increase the size of the input buffer (or at least telling them that it isn't possible) aren't very useful.

EXAMPLES

Give some example uses of the program or function. Don't skimp; users often find this the most useful part of the documentation. The examples are generally given as verbatim paragraphs.

Don't just present an example without explaining what it does. Adding a short paragraph saying what the example will do can increase the value of the example immensely.

ENVIRONMENT

Environment variables that the program cares about, normally presented as a list using `=over`, `=item`, and `=back`. For example:

```
=over 6

=item HOME

Used to determine the user's home directory.  F<.foorc> in this
directory is read for configuration details, if it exists.

=back
```

Since environment variables are normally in all uppercase, no additional special formatting is generally needed; they're glaring enough as it is.

FILES

All files used by the program or function, normally presented as a list, and what it uses them for. File names should be enclosed in F<>. It's particularly important to document files that will be potentially modified.

CAVEATS

Things to take special care with, sometimes called WARNINGS.

BUGS

Things that are broken or just don't work quite right.

RESTRICTIONS

Bugs you don't plan to fix. :-)

NOTES

Miscellaneous commentary.

AUTHOR

Who wrote it (use AUTHORS for multiple people). It's a good idea to include your current email address (or some email address to which bug reports should be sent) or some other contact information so that users have a way of contacting you. Remember that program documentation tends to roam the wild for far longer than you expect and pick a contact method that's likely to last.

HISTORY

Programs derived from other sources sometimes have this. Some people keep a modification log here, but that usually gets long and is normally better maintained in a separate file.

COPYRIGHT AND LICENSE

For copyright

```
Copyright YEAR(s) YOUR NAME(s)
```

(No, (C) is not needed. No, "all rights reserved" is not needed.)

For licensing the easiest way is to use the same licensing as Perl itself:

```
This library is free software; you may redistribute it and/or
modify
it under the same terms as Perl itself.
```

This makes it easy for people to use your module with Perl. Note that this licensing example is neither an endorsement or a requirement, you are of course free to choose any licensing.

SEE ALSO

Other man pages to check out, like man(1), man(7), makewhatis(8), or catman(8). Normally a simple list of man pages separated by commas, or a paragraph giving the name of a reference work. Man page references, if they use the standard `name(section)` form, don't have to be enclosed in L<> (although it's recommended), but other things in this section probably should be when appropriate.

If the package has a mailing list, include a URL or subscription instructions here.

If the package has a web site, include a URL here.

Documentation of object-oriented libraries or modules may want to use CONSTRUCTORS and METHODS sections, or CLASS METHODS and INSTANCE METHODS sections, for detailed documentation of the parts of the library and save the DESCRIPTION section for an overview. Large modules with a function interface may want to use FUNCTIONS for similar reasons. Some people use OVERVIEW to summarize the description if it's quite long.

Section ordering varies, although NAME must always be the first section (you'll break some man page systems otherwise), and NAME, SYNOPSIS, DESCRIPTION, and OPTIONS generally always occur first and in that order if present. In general, SEE ALSO, AUTHOR, and similar material should be left for last. Some systems also move WARNINGS and NOTES to last. The order given above should be reasonable for most purposes.

Some systems use CONFORMING TO to note conformance to relevant standards and MT-LEVEL to note safeness for use in threaded programs or signal handlers. These headings are primarily useful when documenting parts of a C library.

Finally, as a general note, try not to use an excessive amount of markup. As documented here and in *Pod::Man*, you can safely leave Perl variables, module names, function names, man page references, and the like unadorned by markup, and the POD translators will figure it all out for you. This makes it

much easier to later edit the documentation. Note that many existing translators will do the wrong thing with email addresses when wrapped in L<>, so don't do that.

You can check whether your documentation looks right by running

```
% pod2text -o something.pod | less
```

If you have *groff* installed, you can get an even better look this way:

```
% pod2man something.pod | groff -Tps -mandoc > something.ps
```

Now view the resulting Postscript file to see whether everything checks out.

SEE ALSO

For additional information that may be more accurate for your specific system, see either *man(5)* or *man(7)* depending on your system manual section numbering conventions.

This documentation is maintained as part of the podlators distribution. The current version is always available from its web site at <http://www.eyrie.org/~eagle/software/podlators/>.

AUTHOR

Russ Allbery <rra@stanford.edu>, with large portions of this documentation taken from the documentation of the original **pod2man** implementation by Larry Wall and Tom Christiansen.

COPYRIGHT AND LICENSE