CHAPTER 10

POD

POD (Plain Old Documentation) is the method you want to use to document your Perl modules. While simple comments work well for pointing out tricky code, they are cumbersome when you want to give a detailed explanation of your module.

POD was designed so you can create formatted comments. POD uses tags to specify text formats such as fonts as well as text positions.

POD can be embedded within your code or placed in a separate file. When embedded within your code, there are methods of telling the Perl interpreter when to execute code and when to skip over lines.

You put POD documentation together in paragraph format. Each POD paragraph is separated by a blank line. There are three types of paragraphs in POD:

- **Command** – A command paragraph is used to treat a chunk of text is a certain fashion. For example, you can use a Command paragraph to generate a list.
- **Text (or ordinary)** – A text or ordinary paragraph is used to generate text that can either be formatted (bold, italic, hyperlink) or just plain text. A markup feature, similar to HTML or XML, is used to specify formatting.
- **Verbatim** – A verbatim paragraph is used to generate text that contains no formatting. Anything in a verbatim paragraph is considered `"plain text."`

POD commands

POD commands begin with the equal sign and are immediately followed by the command. Text can follow the command as well. The end of the line is the end of the command.

© William "Bo" Rothwell of One Course Source, Inc. 2020
W. "Bo" Rothwell, *Advanced Perl Programming*, https://doi.org/10.1007/978-1-4842-5863-7_10

The following is a summary of the POD commands that are available:

POD command	Meaning
`=back`	Used in conjunction with `=over` and `=item` to generate a list
`=begin` *`format`*	Allows you to specify text to be interpreted by a different parser such as HTML
`=cut`	Used to specify the end of POD text
`=end`	Ends a `=begin` block
`=for` *`format`*	Specifies a format change that will affect the next paragraph only
`=head1` *`text`*	Specifies that the following text is formatted by the predefined `"head1"` format
`=head2`	Specifies that the following text is formatted by the predefined `"head2"` format
`=item`	Used in conjunction with `=over` and `=back` to generate a list
`=over`	Used in conjunction with `=back` and `=item` to generate a list
`=pod`	Specifies the beginning of POD text

Using headings

The POD commands **`=head1`**, **`=head2`**, **`=head3`**, and **`=head4`** provide predefined formats. These are primarily designed to create an indented style format for documentation. The following code describes how they are used:

```
#head.pod
=head1  This is head1
Text under head1
=head2  This is head2
Text under head2
=head3  This is head3
Text under head3
=head4  This is head4
Text under head4
```

How these are displayed really depends on the POD interpreter that is used. Some POD interpreters, such as the **perldoc** command (this command will be covered in more detail later in this Chapter), may consider **=head2**, **=head3**, and **=head4** to be the same or very similar. For example, perldoc indents the head values differently as shown in the following output but doesn't display them in different fonts or font sizes:

```
[student@OCS student]$ perldoc head.pod

This is head1
    Text under head1

  This is head2
    Text under head2

   This is head3
    Text under head3

   This is head4
    Text under head4
```

Starting and ending POD text

When you embed POD within a Perl script, you need to tell the Perl interpreter when to stop "looking at" the code. The **=pod** command tells Perl to stop interpreting code. The **=cut** tells Perl to start interpreting code again.

The **=pod** command is rarely used because any POD command (except **=cut**) will "turn on" POD. However, if you want to start POD documentation with a verbatim paragraph, then **=pod** would be necessary.

When POD is "on," the Perl interpreter will ignore everything until POD is "turned off." POD interpreters, which will be discussed later, will ignore Perl code and only parse POD Commands.

Using other formats

Instead of using POD formats, you can specify other formats. Additional formats that POD can handle are "html," "text," "man," "latex," "tex," and "roff." If you already know how to format text in these other formats, this can be a useful option.

To "turn on" a different format for a single paragraph, use the **=for** command:

```
=for text
```

To "turn on" a different format for a block, use the **=begin** command to start the block and **=end** to stop the block:

```
=begin text
This is plain text
=end text
```

Using POD to create multi-line comments

A common question people have when learning Perl is "does Perl support multi-line comments like some other languages, like C++, offer?" Typically, novice Perl programmers are told "no," but in reality multi-line comments are possible by using POD.

Recall that any POD commands or text will not be interpreted by the Perl interpreter. So, you could create a multi-line comment by doing the following:

```
=for text

    This is a comment
    It can span multiple lines

=cut
```

However, the problem with this technique is if you are actually using POD in this program to create documentation, then the multi-line comment will be included with the POD output.

To avoid this, use the following:

```
=for comment
```

Anything between this command and **=cut** will be ignored by both the Perl interpreter and any POD interpreter. You can even omit the extra blank lines after **=for** and before **=cut** in this specific case.

Creating a list

You can create an indented numbered or bulleted list with POD. To do this, you need to use three commands:

`=over`	This command tells POD to begin a list. You can specify a numeric value to indicate how many "`spaces`" to use to intend (e.g., `=over 4`).
`=item`	This command tells POD to add an item to the list. To specify bulleted lists, use this syntax: "`=item *`". To specify numbered lists, use this syntax: "`=item 1.`", "`=item 2.`", etc.
`=back`	This command tells POD to stop the list.

The following example demonstrates a simple two-item list:

```
=over 4

=item *
This is one item

=item *
This is another item

=back
```

The following demonstrates how the pod list would appear:

```
[student@OCS student]$ perldoc podlist.pod
    *   This is one item

    *   This is another item
```

POD text

POD text can be displayed as "`plain text`" or formatted using the following commands:

Text command	Meaning
I<text>	Italicize text
B<text>	Bold text
S<text>	Nonbreaking spaces text
C<code>	Display "code" in a typewriter font to distinguish from other text
L<name>	A link to either another document or a section in this document
E<lt>	A less than character: "<"
E<gt>	A greater than character: ">"
E<sol>	A slash character: "\"
E<verbar>	A pipe character: "|"

Notes:

- Look at the *perlpod* man page for additional text commands.
- Depending on your POD interpreter, some of the preceding commands may not result in any text change. For example, the perldoc command typically ignores these options as the output is normally in a terminal window:

```
[student@OCS student]$ perldoc podformat.pod
    This is very important - please report all data to Frank.
```

POD verbatim

If you don't want any formatting done on your text, you can use the verbatim method. With the verbatim method, no "special" characters are allowed (like italics, bold, etc.).

If you don't want to have formatted text, you can use verbatim text by indenting your text with at least one space:

```
This is an example
```

The primary purpose of POD verbatim is when you want to use a character that is normally considered a formatting character. For example, consider the following:

```
=for text
To display in bold, use B<text>
```

A POD interpreter might attempt to make "text" bold when it displays this line. If you don't want the code to be interpreted, but rather display just as it appears, just put a space in front of the line:

```
=for text

To display in bold, use B<text>
```

POD examples

In the following example, we have a file that only contains POD commands:

```
#pod1.pod
=head1 EXAMPLE

An example of POD

=head2 DESCRIPTION

This is an example of how bulleted lists work:
=over 4

=item *

Item one

=item *

Item two

=back

We are still displaying text under the head2 command here

Here is an example of numbered lists

=over 4

=item 1.

Item one

=item 2.
```

```
Item two

=back

One last thing: This is an example of verbatim text:

 A \ is just a slash, a < is just a less than sign...
```

The following demonstrates the output of `pod1.pl`:

```
[student@OCS student]$ perldoc pod1.pod
EXAMPLE
    An example of POD

   DESCRIPTION

   This is an example of how bulleted lists work:

     • Item one

     • Item two

   We are still displaying text under the head2 command here

   Here is an example of numbered lists

     1. Item one

     2. Item two

   One last thing: This is an example of verbatim text:

   A \ is just a slash, a < is just a less than sign...
```

In this example, POD is embedded in a `Perl` script:

```
#!perl
#pod2.pl

=head1 EXAMPLE

Just an example of POD within a perl script...easy, isn't it?

=cut

sub hello {
```

```
    print "hello\n"
}

&hello;
```

Note that `pod2.pl` can be run as a normal Perl script. The POD documentation has no impact on the execution of the script:

```
[student@OCS student]$ perl pod2.pl
hello
```

You can also view the POD documentation of the `pod2.pl` script:

```
[student@OCS student]$ perldoc pod2.pl
EXAMPLE
    Just an example of POD within a perl script...easy, isn't it?
```

Note While you will sometimes see POD documentation in a separate file, it is much more common to see it in the script itself. Typically, it appears at the top of the script, but you will sometimes see it at the bottom as well. There aren't any rules as to where you place POD documentation.

Common POD problems

As you can probably tell, POD is very simple. It's not very fancy, but to create "`Plain Old Documentation`," quickly and easily, it works very well.

Because it is so simple, very few things can go wrong. Here is a list of the major potential problems:

1. Forgetting to use **`=pod`** to begin (or **`=cut`** to end) when you embed POD in Perl scripts.
2. Some POD interpreters require blank lines between commands and the text that follows.
3. Not understanding the "`text commands`." For example, some POD interpreters add additional text when you use the **`L<link>`** command.

POD utilities

There are several utilities that are useful when you work with POD.

Note Not all of these utilities may be available on your system.

---

podchecker

This utility is useful to check the syntax of your POD file. If there are no syntax errors, podchecker reports "syntax OK":

```
[student@OCS student]$ podchecker pod1.txt
pod1.pod pod syntax OK.
```

Based on the current documentation, **podchecker** will check the following (note: this list hasn't changed for a very long time, so expect this will be the same check for future Perl 5 releases):

- Unknown '=xxxx' commands, unknown 'X<...>' interior-sequences, and unterminated interior sequences.
- Check for proper balancing of `=begin` and `=end`. The contents of such a block are generally ignored, i.e. no syntax checks are performed.
- Check for proper nesting and balancing of `=over`, `=item` and `=back`.
- Check for same nested interior-sequences (e.g. `L<...L<...>...>`).
- Check for malformed or non-existing entities `E<...>`.
- Check for correct syntax of hyperlinks `L<...>`. See perlpod for details.
- Check for unresolved document-internal links. This check may also reveal misspelled links that seem to be internal links but should be links to something else.

If there are any syntax problems, POD indicates what the problems are:

```
[student@OCS student]$ podchecker pod3.txt
*** ERROR: empty =head1 at line 3 in file pod3.txt
pod3.txt has 1 pod syntax error.
```

perldoc

The **perldoc** command will display the output of a POD file:

```
[student@OCS student]$ perldoc pod2.pl
EXAMPLE
    Just an example of POD within a perl script...easy, isn't it?
```

In addition to modules, the Perl core documentation is also in POD format. To see the core Perl documentation, use the following command:

```
[student@OCS student]$ perldoc perl
NAME
    perl - The Perl 5 language interpreter

SYNOPSIS
    perl [ -sTtuUWX ] [ -hv ] [ -V[:*configvar*] ]
    [ -cw ] [ -d[t][:*debugger*] ] [ -D[*number/list*] ]
    [ -pna ] [ -F*pattern* ] [ -l[*octal*] ] [ -0[*octal/hexadecimal*] ]
    [ -I*dir* ] [ -m[-]*module* ] [ -M[-]*'module...'* ] [ -f ]
    [ -C [*number/list*] ] [ -S ] [ -x[*dir*] ] [ -i[*extension*] ]
    [ [-e|-E] *'command'* ] [ -- ] [ *programfile* ] [ *argument* ]...

    For more information on these options, you can run "perldoc perlrun".

GETTING HELP
    The perldoc program gives you access to all the documentation that
    comes with Perl. You can get more documentation, tutorials and
    community support online at <http://www.perl.org/>.
{remaining output omitted}
```

Included in the output of the preceding command is a list of other documents that you can view, such as the following:

```
[student@OCS student]$ perldoc perlcheat
NAME
    perlcheat - Perl 5 Cheat Sheet
```

```
DESCRIPTION
    This 'cheat sheet' is a handy reference, meant for beginning Perl
    programmers. Not everything is mentioned, but 195 features may already
    be overwhelming.
{remaining output omitted}
```

If you read through the main perldoc documentation, you will see a bunch of FAQs. The **-q** option to perldoc allows you to search the FAQs using a keyword:

```
[student@OCS student]$ perldoc -q sort
Found in /usr/bin/perl/lib/pods/perlfaq4.pod
  How do I sort an array by (anything)?
    Supply a comparison function to sort() (described in "sort" in
    perlfunc):

        @list = sort { $a <=> $b } @list;

    The default sort function is cmp, string comparison, which would sort
    "(1, 2, 10)" into "(1, 10, 2)". "<=>", used above, is the numerical
    comparison operator.
{remaining output omitted}
```

Note that the output of the **perldoc -q sort** command also showed you where the POD file exists (`/usr/bin/perl/lib/pods/perlfaq4.pod` in the previous example). This location can vary based on your platform (operating system) and type of Perl that you installed (standard, Strawberry, ActiveState, etc.). The advantage of knowing this location is you can go look at a large collection of POD files as a way of learning POD better.

If you want to see a list of all of Perl's functions, view the **perlfunc** document. This is also an excellent way to see a list of what functions are available on the version of Perl that you are currently using:

```
[student@OCS student]$ perldoc perlfunc
NAME
    perlfunc - Perl builtin functions
```

```
DESCRIPTION
    The functions in this section can serve as terms in an expression. They
    fall into two major categories: list operators and named unary
    operators. These differ in their precedence relationship with a
    following comma. (See the precedence table in perlop.) List operators
    take more than one argument, while unary operators can never take
    more than one argument. Thus, a comma terminates the argument of a
    unary operator, but merely separates the arguments of a list operator. A
    unary operator generally provides scalar context to its argument, while a
    list operator may provide either scalar or list contexts for its arguments.
    If it does both, scalar arguments come first and list argument follow,
    and there can only ever be one such list argument. For instance,
    "splice" has three scalar arguments followed by a list, whereas
    "gethostbyname" has four scalar arguments.
{skipping output…}
    Input and output functions
        "binmode", "close", "closedir", "dbmclose", "dbmopen", "die", "eof",
        "fileno", "flock", "format", "getc", "print", "printf", "read",
        "readdir", "readline", "rewinddir", "say", "seek", "seekdir",
        "select", "syscall", "sysread", "sysseek", "syswrite", "tell",
        "telldir", "truncate", "warn", "write"
{remaining output omitted}
```

Being able to view the list of functions can be helpful in many ways. For example, suppose you forget the name of a function that you rarely use. You could read through the list of functions, or, if your platform has a filter command like the **grep** command, you could send the output to the filter command to view just the lines you want to see.

To see a specific function's documentation, use the **-f** option:

```
[student@OCS student]$ perldoc -f sort
    sort SUBNAME LIST
    sort BLOCK LIST
    sort LIST
            In list context, this sorts the LIST and returns the sorted list
            value. In scalar context, the behaviour of "sort" is undefined.
```

```
            If SUBNAME or BLOCK is omitted, "sort"s in standard string
            comparison order. If SUBNAME is specified, it gives the name of
            a subroutine that returns an integer less than, equal to, or
            greater than 0, depending on how the elements of the list are to
            be ordered. (The "<=>" and "cmp" operators are extremely useful
            in such routines.) SUBNAME may be a scalar variable name
            (unsubscripted), in which case the value provides the name of
            (or a reference to) the actual subroutine to use. In place of a
            SUBNAME, you can provide a BLOCK as an anonymous, in-line
            sort subroutine.
{remaining output omitted}
```

If you want to see a module documentation, use the following syntax:

```
[student@OCS student]$ perldoc File::Copy
NAME
    File::Copy - Copy files or filehandles

SYNOPSIS
            use File::Copy;

            copy("sourcefile","destinationfile") or die "Copy failed: $!";
            copy("Copy.pm",\*STDOUT);
            move("/dev1/sourcefile","/dev2/destinationfile");

            use File::Copy "cp";

            $n = FileHandle->new("/a/file","r");
            cp($n,"x");

DESCRIPTION
    The File::Copy module provides two basic functions, "copy" and "move",
    which are useful for getting the contents of a file from one place to
    another.
{remaining output omitted}
```

You can even have **perldoc** tell you where the module is installed by using the **-l** option. This can be useful when you want to view the module code directly:

```
[student@OCS student]$ perldoc -l File::Copy
C:\Perl64\lib\File\Copy.pm
```

Or, to see the raw code of a module, use the **-m** option:

```
[student@OCS student]$ perldoc -m File::Copy
# File/Copy.pm. Written in 1994 by Aaron Sherman <ajs@ajs.com>. This
# source code has been placed in the public domain by the author.
# Please be kind and preserve the documentation.
#
# Additions copyright 1996 by Charles Bailey.  Permission is granted
# to distribute the revised code under the same terms as Perl itself.

package File::Copy;

use 5.006;
use strict;
use warnings; no warnings 'newline';
use File::Spec;
use Config;
# During perl build, we need File::Copy but Scalar::Util might not be built
# yet.  And then we need these games to avoid loading overload, as that
# will confuse miniperl during the bootstrap of perl.
my $Scalar_Util_loaded = eval q{ require Scalar::Util; require overload; 1 };
# We want HiRes stat and utime if available
BEGIN { eval q{ use Time::HiRes qw( stat utime ) } };
our(@ISA, @EXPORT, @EXPORT_OK, $VERSION, $Too_Big, $Syscopy_is_copy);
sub copy;
sub syscopy;
{remaining output omitted}
```

Additional utilities

Depending on factors such as your platform and version of Perl, you might also have access to the following POD utilities:

pod2fm – Translates POD files to FrameMaker formats

pod2html – Translates POD files to HTML format

pod2latex – Translates POD files to LaTeX format

pod2man – Translates POD files to man page format

pod2text – Translates POD files to text format

POD style

When creating POD documentation, there are several things that you may want to consider. For some Perl programmers, one of the most important things to consider is where to place your POD documentation.

To begin with, in most cases your POD documentation should be embedded within your Perl module file and not in a separate file. Placing POD in a separate file poses problems when the module is copied to another location (and the POD file is not).

When you place the POD documentation in your Perl module, you basically have three choices:

1. **At the top of your module file** – Several Perl programmers take this approach; however, this does tend to frustrate other programmers who view the module file directly as they don't want to have to scroll through a bunch of POD documentation to read the actual code.
2. **Spaced throughout the module file** – Some Perl programmers use this technique to provide documentation while also commenting sections of the Perl code. However, it does tend to make reading the code directly difficult.
3. **At the end of your module file** – Because other programmers rarely read POD directly from your file, this may be the best option. Place all of the Perl code first and POD at the end of the code.

For other POD style considerations, view the `perlpodstyle` documentation:

```
[student@OCS student]$ perldoc perlpodstyle
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.
{remaining output omitted}
```

Additional resources

In each chapter, resources are provided to provide the learner with a source for more information. These resources may include downloadable source code or links to other books or articles that will provide you more information about the topic at hand.

Resources for this chapter can be found here:

https://github.com/apress/advanced-perl-programming

Lab exercises

Use POD to document the module that you created in the Chapter 8 lab.