



**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

***Rocky Enterprise Linux 9.2 Manual Pages on command 'Pod::Usage.3perl'***

***\$ man Pod::Usage.3perl***

Pod::Usage(3perl) Perl Programmers Reference Guide Pod::Usage(3perl)

NAME

Pod::Usage - extracts POD documentation and shows usage information

SYNOPSIS

```
use Pod::Usage;

my $message_text = "This text precedes the usage message.";

my $exit_status = 2;    ## The exit status to use

my $verbose_level = 0;  ## The verbose level to use

my $filehandle = \*STDERR; ## The filehandle to write to

pod2usage($message_text);

pod2usage($exit_status);

pod2usage( { -message => $message_text ,
            -exitval => $exit_status ,
            -verbose => $verbose_level,
            -output => $filehandle } );

pod2usage( -msg => $message_text ,
            -exitval => $exit_status ,
            -verbose => $verbose_level,
            -output => $filehandle );

pod2usage( -verbose => 2,
            -noperldoc => 1 );

pod2usage( -verbose => 2,
            -perlcmd => $path_to_perl,
```

```
-perldoc => $path_to_perldoc,  
-perldocopt => $perldoc_options );
```

## ARGUMENTS

pod2usage should be given either a single argument, or a list of arguments corresponding to an associative array (a "hash"). When a single argument is given, it should correspond to exactly one of the following:

- ? A string containing the text of a message to print before printing the usage message
- ? A numeric value corresponding to the desired exit status
- ? A reference to a hash

If more than one argument is given then the entire argument list is assumed to be a hash.

If a hash is supplied (either as a reference or as a list) it should contain one or more elements with the following keys:

"-message" string

"-msg" string

The text of a message to print immediately prior to printing the program's usage message.

"-exitval" value

The desired exit status to pass to the exit() function. This should be an integer, or else the string "NOEXIT" to indicate that control should simply be returned without terminating the invoking process.

"-verbose" value

The desired level of "verboseness" to use when printing the usage message. If the value is 0, then only the "SYNOPSIS" and/or "USAGE" sections of the pod documentation are printed. If the value is 1, then the "SYNOPSIS" and/or "USAGE" sections, along with any section entitled "OPTIONS", "ARGUMENTS", or "OPTIONS AND ARGUMENTS" is printed. If the corresponding value is 2 or more then the entire manpage is printed, using perldoc if available; otherwise Pod::Text is used for the formatting. For better readability, the all-capital headings are downcased, e.g. "SYNOPSIS" => "Synopsis". The special verbosity level 99 requires to also specify the -sections parameter; then these sections are extracted and printed.

"-sections" spec

There are two ways to specify the selection. Either a string (scalar) representing a selection regexp for sections to be printed when -verbose is set to 99, e.g.

"NAME|SYNOPSIS|DESCRIPTION|VERSION"

With the above regexp all content following (and including) any of the given "=head1" headings will be shown. It is possible to restrict the output to particular subsections only, e.g.:

"DESCRIPTION/Algorithm"

This will output only the "=head2 Algorithm" heading and content within the "=head1 DESCRIPTION" section. The regexp binding is stronger than the section separator, such that e.g.:

"DESCRIPTION|OPTIONS|ENVIRONMENT/Caveats"

will print any "=head2 Caveats" section (only) within any of the three "=head1" sections.

Alternatively, an array reference of section specifications can be used:

```
pod2usage(-verbose => 99, -sections => [
    qw(DESCRIPTION DESCRIPTION/Introduction) ] );
```

This will print only the content of "=head1 DESCRIPTION" and the "=head2 Introduction" sections, but no other "=head2", and no other "=head1" either.

#### "-output" handle

A reference to a filehandle, or the pathname of a file to which the usage message should be written. The default is "\\*STDERR" unless the exit value is less than 2 (in which case the default is "\\*STDOUT").

#### "-input" handle

A reference to a filehandle, or the pathname of a file from which the invoking script's pod documentation should be read. It defaults to the file indicated by \$0 (\$PROGRAM\_NAME for users of English.pm).

If you are calling pod2usage() from a module and want to display that module's POD, you can use this:

```
use Pod::Find qw(pod_where);
pod2usage( -input => pod_where({-inc => 1}, __PACKAGE__ ) );
```

#### "-pathlist" string

A list of directory paths. If the input file does not exist, then it will be searched for in the given directory list (in the order the directories appear in the list). It defaults to the list of directories implied by \$ENV{PATH}. The list may be specified either by a reference to an array, or by a string of directory paths which use the

same path separator as \$ENV{PATH} on your system (e.g., ":" for Unix, ";" for MSWin32 and DOS).

"-noperldoc"

By default, Pod::Usage will call perldoc when -verbose >= 2 is specified. This does not work well e.g. if the script was packed with PAR. This option suppresses the external call to perldoc and uses the simple text formatter (Pod::Text) to output the POD.

"-perlcmd"

By default, Pod::Usage will call perldoc when -verbose >= 2 is specified. In case of special or unusual Perl installations, this option may be used to supply the path to a perl executable which should run perldoc.

"-perldoc" path-to-perldoc

By default, Pod::Usage will call perldoc when -verbose >= 2 is specified. In case perldoc is not installed where the perl interpreter thinks it is (see Config), the -perldoc option may be used to supply the correct path to perldoc.

"-perldocopt" string

By default, Pod::Usage will call perldoc when -verbose >= 2 is specified. This option may be used to supply options to perldoc. The string may contain several, space-separated options.

#### Formatting base class

The default text formatter is Pod::Text. The base class for Pod::Usage can be defined by pre-setting \$Pod::Usage::Formatter before loading Pod::Usage, e.g.:

```
BEGIN { $Pod::Usage::Formatter = 'Pod::Text::Termcap'; }  
use Pod::Usage qw(pod2usage);
```

Pod::Usage uses Pod::Simple's \_handle\_element\_end() method to implement the section selection, and in case of verbosity < 2 it down-cases the all-caps headings to first capital letter and rest lowercase, and adds a colon/newline at the end of the headings, for better readability. Same for verbosity = 99.

#### Pass-through options

The following options are passed through to the underlying text formatter. See the manual pages of these modules for more information.

alt code indent loose margin quotes sentence stderr utf8 width

pod2usage will print a usage message for the invoking script (using its embedded pod documentation) and then exit the script with the desired exit status. The usage message printed may have any one of three levels of "verboseness": If the verbose level is 0, then only a synopsis is printed. If the verbose level is 1, then the synopsis is printed along with a description (if present) of the command line options and arguments. If the verbose level is 2, then the entire manual page is printed.

Unless they are explicitly specified, the default values for the exit status, verbose level, and output stream to use are determined as follows:

- ? If neither the exit status nor the verbose level is specified, then the default is to use an exit status of 2 with a verbose level of 0.
- ? If an exit status is specified but the verbose level is not, then the verbose level will default to 1 if the exit status is less than 2 and will default to 0 otherwise.
- ? If an exit status is not specified but verbose level is given, then the exit status will default to 2 if the verbose level is 0 and will default to 1 otherwise.
- ? If the exit status used is less than 2, then output is printed on "STDOUT". Otherwise output is printed on "STDERR".

Although the above may seem a bit confusing at first, it generally does "the right thing" in most situations. This determination of the default values to use is based upon the following typical Unix conventions:

- ? An exit status of 0 implies "success". For example, diff(1) exits with a status of 0 if the two files have the same contents.
- ? An exit status of 1 implies possibly abnormal, but non-defective, program termination. For example, grep(1) exits with a status of 1 if it did not find a matching line for the given regular expression.
- ? An exit status of 2 or more implies a fatal error. For example, ls(1) exits with a status of 2 if you specify an illegal (unknown) option on the command line.
- ? Usage messages issued as a result of bad command-line syntax should go to "STDERR". However, usage messages issued due to an explicit request to print usage (like specifying -help on the command line) should go to "STDOUT", just in case the user wants to pipe the output to a pager (such as more(1)).
- ? If program usage has been explicitly requested by the user, it is often desirable to exit with a status of 1 (as opposed to 0) after issuing the user-requested usage message. It is also desirable to give a more verbose description of program usage in

this case.

pod2usage does not force the above conventions upon you, but it will use them by default if you don't expressly tell it to do otherwise. The ability of pod2usage() to accept a single number or a string makes it convenient to use as an innocent looking error message handling function:

```
use strict;
use Pod::Usage;
use Getopt::Long;

## Parse options
my %opt;

GetOptions(\%opt, "help|?", "man", "flag1") || pod2usage(2);
pod2usage(1) if ($opt{help});
pod2usage(-exitval => 0, -verbose => 2) if ($opt{man});

## Check for too many filenames
pod2usage("$0: Too many files given.\n") if (@ARGV > 1);
```

Some user's however may feel that the above "economy of expression" is not particularly readable nor consistent and may instead choose to do something more like the following:

```
use strict;
use Pod::Usage qw(pod2usage);
use Getopt::Long qw(GetOptions);

## Parse options
my %opt;

GetOptions(\%opt, "help|?", "man", "flag1") ||
    pod2usage(-verbose => 0);
pod2usage(-verbose => 1) if ($opt{help});
pod2usage(-verbose => 2) if ($opt{man});

## Check for too many filenames
pod2usage(-verbose => 2, -message => "$0: Too many files given.\n")
    if (@ARGV > 1);
```

As with all things in Perl, there's more than one way to do it, and pod2usage() adheres to this philosophy. If you are interested in seeing a number of different ways to invoke pod2usage (although by no means exhaustive), please refer to "EXAMPLES".

The Pod::Usage distribution comes with a script pod2usage which offers a command line interface to the functionality of Pod::Usage. See pod2usage.

## EXAMPLES

Each of the following invocations of "pod2usage()" will print just the "SYNOPSIS" section to "STDERR" and will exit with a status of 2:

```
pod2usage();
pod2usage(2);
pod2usage(-verbose => 0);
pod2usage(-exitval => 2);
pod2usage({-exitval => 2, -output => \*STDERR});
pod2usage({-verbose => 0, -output => \*STDERR});
pod2usage(-exitval => 2, -verbose => 0);
pod2usage(-exitval => 2, -verbose => 0, -output => \*STDERR);
```

Each of the following invocations of "pod2usage()" will print a message of "Syntax error." (followed by a newline) to "STDERR", immediately followed by just the "SYNOPSIS" section (also printed to "STDERR") and will exit with a status of 2:

```
pod2usage("Syntax error.");
pod2usage(-message => "Syntax error.", -verbose => 0);
pod2usage(-msg => "Syntax error.", -exitval => 2);
pod2usage({-msg => "Syntax error.", -exitval => 2, -output => \*STDERR});
pod2usage({-msg => "Syntax error.", -verbose => 0, -output => \*STDERR});
pod2usage(-msg => "Syntax error.", -exitval => 2, -verbose => 0);
pod2usage(-message => "Syntax error.",
    -exitval => 2,
    -verbose => 0,
    -output => \*STDERR);
```

Each of the following invocations of "pod2usage()" will print the "SYNOPSIS" section and any "OPTIONS" and/or "ARGUMENTS" sections to "STDOUT" and will exit with a status of 1:

```
pod2usage(1);
pod2usage(-verbose => 1);
pod2usage(-exitval => 1);
pod2usage({-exitval => 1, -output => \*STDOUT});
pod2usage({-verbose => 1, -output => \*STDOUT});
```

```
pod2usage(-exitval => 1, -verbose => 1);
```

```
pod2usage(-exitval => 1, -verbose => 1, -output => \*STDOUT});
```

Each of the following invocations of "pod2usage()" will print the entire manual page to "STDOUT" and will exit with a status of 1:

```
pod2usage(-verbose => 2);
```

```
pod2usage({-verbose => 2, -output => \*STDOUT});
```

```
pod2usage(-exitval => 1, -verbose => 2);
```

```
pod2usage({-exitval => 1, -verbose => 2, -output => \*STDOUT});
```

## Recommended Use

Most scripts should print some type of usage message to "STDERR" when a command line syntax error is detected. They should also provide an option (usually "-H" or "-help") to print a (possibly more verbose) usage message to "STDOUT". Some scripts may even wish to go so far as to provide a means of printing their complete documentation to "STDOUT" (perhaps by allowing a "-man" option). The following complete example uses Pod::Usage in combination with Getopt::Long to do all of these things:

```
use strict;

use Getopt::Long qw(GetOptions);
use Pod::Usage qw(pod2usage);

my $man = 0;
my $help = 0;

## Parse options and print usage if there is a syntax error,
## or if usage was explicitly requested.
GetOptions('help?' => \$help, man => \$man) or pod2usage(2);
pod2usage(1) if $help;
pod2usage(-verbose => 2) if $man;

## If no arguments were given, then allow STDIN to be used only
## if it's not connected to a terminal (otherwise print usage)
pod2usage("$0: No files given.") if ((@ARGV == 0) && (-t STDIN));

__END__

=head1 NAME

sample - Using Getopt::Long and Pod::Usage

=head1 SYNOPSIS

sample [options] [file ...]
```

Options:

-help        brief help message

-man        full documentation

=head1 OPTIONS

=over 4

=item B<-help>

Print a brief help message and exits.

=item B<-man>

Prints the manual page and exits.

=back

=head1 DESCRIPTION

B<This program> will read the given input file(s) and do something

useful with the contents thereof.

=cut

## CAVEATS

By default, `pod2usage()` will use `$0` as the path to the pod input file. Unfortunately, not all systems on which Perl runs will set `$0` properly (although if `$0` is not found, `pod2usage()` will search `$ENV{PATH}` or else the list specified by the `"-pathlist"` option).

If this is the case for your system, you may need to explicitly specify the path to the pod docs for the invoking script using something similar to the following:

```
pod2usage(-exitval => 2, -input => "/path/to/your/pod/docs");
```

In the pathological case that a script is called via a relative path and the script itself changes the current working directory (see `"chdir"` in `perlfunc`) before calling `pod2usage`, `Pod::Usage` will fail even on robust platforms. Don't do that. Or use `FindBin` to locate the script:

```
use FindBin;

pod2usage(-input => $FindBin::Bin . "/" . $FindBin::Script);
```

## SUPPORT

This module is managed in a GitHub repository, <<https://github.com/Dual-Life/Pod-Usage>>

Feel free to fork and contribute, or to clone and send patches!

Please use <<https://github.com/Dual-Life/Pod-Usage/issues/new>> to file a bug report. The previous ticketing system, <<https://rt.cpan.org/Dist/Display.html?Queue=Pod-Usage>>, is deprecated for this package.

More general questions or discussion about POD should be sent to the "pod-people@perl.org" mail list. Send an empty email to "pod-people-subscribe@perl.org" to subscribe.

#### AUTHOR

Marek Rouchal <marekr@cpan.org>

Nicolas R <nicolas@atomic.org>

Brad Appleton <bradapp@enteract.com>

Based on code for Pod::Text::pod2text() written by Tom Christiansen <tchrist@mox.perl.com>

#### LICENSE

Pod::Usage (the distribution) is licensed under the same terms as Perl.

#### ACKNOWLEDGMENTS

Nicolas R (ATOOMIC) for setting up the Github repo and modernizing this package.

rjbs for refactoring Pod::Usage to not use Pod::Parser any more.

Steven McDougall <swmcd@world.std.com> for his help and patience with re-writing this manpage.

#### SEE ALSO

Pod::Usage is now a standalone distribution, depending on Pod::Text which in turn depends on Pod::Simple.

Pod::Perldoc, Getopt::Long, Pod::Find, FindBin, Pod::Text, Pod::Text::Termcap, Pod::Simple

perl v5.34.0

2023-11-23

Pod::Usage(3perl)