

NAME

shdoc – sh function for processing online man page

SYNOPSIS

[FPATH=/path/to/shdoc]

shdoc *help=y* | *ps* | *pdf* | *html* [*doc=pod* | *man*] [*file=...*]

DESCRIPTION

Generally, large programs possess big documentation – with books, technical papers, MAN pages, etc. Examples include Solaris, Oracle, Perl, shells, etc. However, the shell scripts that we write every day usually aren't that big, and it's overhead to maintain separate documentation. In addition, shell programmers and administrators are notorious for not providing documentation. Why not simplify the task?

Therefore, it's desirable to maintain online documentation within the code, and when code does change, documentation updates exist in only one place. This man page describes shdoc, a shell function which processes documentation embedded within a shell script.

Perl programmers conveniently embed documentation in Perl scripts using the POD format, plain old documentation. The shdoc function uses the perldoc utility to process POD formatted documentation embedded in shell scripts. In addition, it the function can process embedded documentation already in MAN format. Besides viewing the man page online, it can also convert the documentation in Postscript or PDF format with groff and ps2pdf tools.

POD / MAN FORMAT – sh STYLE

The documentation is entered in scripts using the POD / MAN format, proceeding each line with “## ”, where the space can be omitted for empty lines. The beginning and the ending of the documentation is marked by POD_START and POD_STOP, or MAN_START and MAN_STOP depending on the type of documentation.

shdoc supports splitting documentation, so you can include comments and code in between sh style POD format. But usually the documentation is included at the end.

Using “##”, we differentiate sh style POD from normal shell comments, and, also, maintain 100% legal sh code since sh style POD are still comment lines.

At run-time, anything after the exit command is not interpreted by the shell, so it is possible to include “bare” POD inside sh scripts. However, this approach confuses the shell parser. For example, the two-line script:

```
exit 0
)
```

produces the following error when parsed with “ksh -n”, which checks the syntax but does not execute:

```
syntax error at line 2: ')' unexpected
```

That is why we insist on 100% legal sh syntax.

EXECUTING shdoc

Execute the shdoc function either from a shell script or from the command-line:

```
shdoc help=y | ps | pdf [doc=pod | man] [file=...]
```

The default is “help=y”, “doc=pod”. “help=y” displays a man page, “help=ps” generates a Postscript document, and “help=pdf” generates a PDF document.

EXECUTING FROM A SCRIPT

This is a typical example of calling shdoc from a script. Executing:

```
shdoctest.sh help=y
```

outputs the man page converted from POD to standard output.

The following lines from within the test script:

```
my_getopts "$@"
[[ $HELP = Y ]] && { shdoc file=$0; exit 0; }
```

sets `HELP="Y"`, calls the `shdoc` to process the POD, and exits the script. The `my_getopts` function which evaluates the command-line was a UnixReview.com Shell Corner subject. See

<http://www.unixreview.com/documents/s=1344/uni1042138723500/>

The argument `file=$0` instructs `shdoc` to search the present file for POD. Unfortunately, the argument is necessary since `$0` inside a function in `ksh93` is the function name, instead of the calling program name. Passing the program name works in most of shells.

Executing:

```
shdoctest.sh
```

without `"help=y"` command-line argument, skip printing the documentation, and continue normal program flow. Of course, production scripts may have have other command-line arguments.

You don't need to use `my_getopts`. Any command-line argument processing sensing that help should be printed can be used:

```
[[ $1 = "-h" ]] && { shdoc file=$0; exit 0; }
```

The above command executes `shdoc` if argument 1 equals `"-h"`.

So far, the examples have all displayed man pages. Sometimes you want to generate PS or PDF file for publishing. All you need is to tell the program what you want to do, pass it to `shdoc` function, and your wish be granted.

Here is a one-line example:

```
[[ $HELP = @(Y|PS|PDF) ]] && { shdoc file=$0 help=$HELP; exit 0; }
```

UTILITIES USED AND PASSING THE PATH

`shdoc` executes correctly only if the function can find the `perldoc` and `pod2man` utilities, and standard Unix commands. `groff` is needed for creating Postscript documents with `"help=ps"` option, and `ps2pdf` is needed for converting Postscript to PDF documents when specifying the `"help=pdf"` option.

`perldoc` and `pod2man` are part of Perl package; `groff` is a GNU utility while `ps2pdf` is part of the Ghostscript package.

The function does not set `PATH`. It is expected the proper `PATH` is set up in the calling program.

REFERENCES

Wall, Larry, Tom Christiansen, and Jon Orwant Programming Perl July 2000, Sebastopol, CA: O'Reilly & Associates, Inc.

Bolsky, Morris, David Korn The New Kornshell Command and Programming Language 1995, Upper Saddle River, NJ: Prentice Hall PTR.

SEE ALSO

`"my_getopts _help=y"` on <http://www.unixlabplus.com/>.

AUTHOR

Michael Wang <xw73@columbia.edu>, Ed Schaefer.

This is free software. You may copy or redistribute it under the same terms as Perl. However, if you modify it, you are required to send the modification to the author via email.

VERSION HISTORY

version 3.0, 2007-01-26.

- A rewrite: added MAN format; make it portable (renamed from `kshdoc` to `shdoc`); simplified the code.

version 2.1, 2003-01-27.

- Added “su bin -c ...” for root user.

version 2.0, 2003-05-08.

- Use perldoc instead of pod2man – older version of pod2man does not support standard input.
- Support file=command, file=function without full path, and lack of file= for shdoc itself.
- Support split documentation.
- “show=y” implies “help=n”.

version 1.0, 2002-08-22.