



**NAME**

process-getopt – a wrapper around getopt

**SYNOPSIS**

**process-getopt** [ options ]

and (more usefully) when sourced:

**process\_getopt\_version**

**add\_opt** NAME [ desc ] [ short\_opt [ arg\_name ]] [ long\_opt [ arg\_name ]]

**del\_opt** name [ name ... ]

**add\_std\_opts**

**call\_getopt** [ args ]

**process\_opts** [ args ]

**print\_all\_opts**

**print\_short\_flags**

**print\_long\_flags**

**print\_short\_args** [ man ]

**print\_long\_args** [ man ]

**print\_all\_args** [ man ]

**get\_opt\_letter** name

**get\_opt\_string** name

**get\_opt\_sarg** name

**get\_opt\_larg** name

**get\_opt\_desc** name

**get\_opt\_name** option

**clean\_process\_getopt**

**DESCRIPTION**

There is a common class of errors in **bash**(1) scripts which arise in the processing of options, even when **getopt**(1) is used; namely, a disagreement between:

- the constants used to define the option letters or long words (eg '-c' or '--cdrom') in the call to **getopt**(1) itself,
- the targets of the case statement used to process the returned values from **getopt**(1) and
- the option letters and words displayed in the help and man pages.

These inconsistencies are particularly likely to creep in after maintenance is done on scripts. In any case, this part of the code is tedious to write and is often skimped.

**process-getopt**(1) is intended to eliminate these errors and to automate the creation of the **getopt**(1) command line, the processing of the options and the printing of the help and man pages. It also takes a lot of the drudgery and repetition out of these areas of writing scripts, hopefully increasing the probability that **getopt**(1) is used and that documentation is written.

**process-getopt**(1) should be sourced by a **bash**(1) script and the various functions above are then called to process command-line options. Some checking is done by **process-getopt**(1) as it starts up.

**process-getopt**(1) allows the script to:

- define short and long options in one central place together with descriptions.
- provide perfect consistency between the getopt calling parameters, the case statement processing the user's options and the help/man pages
- be easy to use and maintain
- automate help-page and man-page printing – in fact, **process-getopt**(1) provides a default **usage**() function which suffices for most simple scripts.

## FUNCTIONS

Note that if you want to call one of the accessor functions (**get\_opt\_letter()**, **print\_short\_args()**, ... etc) when generating the help page, you can't call them within the **USAGE** parameter and so you won't be able to use the default **usage()** function. Instead, you'll need to create a custom **usage()** function.

### **process\_getopt\_version**

Prints the version of **process-getopt(1)**. This will always be a string of two numbers of the form **1.0** and calling scripts can use this string to ensure they have the right version or, at least a compatible version. Just like shared libraries, the final digit will be for bug fixes and the first digit will denote an API change. Versions will march 1.0 .. 1.9 1.10 1.11 etc.

### **add\_opt**

Defines a new option. It takes the following parameters:

**name** is the abstract name that is to be assigned to the option. It must be given and must be unique.

**desc** is the description to be assigned to the option. It should be a fairly short explanation, although it will be re-formatted if it needs more than one line on the help screen. It is optional. If not given (eg by using the "" null string as a position holder) then the option is *silent* and is not printed in the help or man pages.

**short\_opt** is the single letter option (without a leading '-' ) to be assigned to the option. It is optional, although at least one short or one long option must be given.

**arg\_name** For short and long options, if the option takes a value, then define **arg\_name**. This allows **process-getopt(1)** to look for that value as well as print the argument name in the help and man pages. If **arg\_name** is defined for the short option, then it becomes the default for the long option, and vice versa.

**long\_opt** is the long option word (without a leading '-' ) to be assigned to the option. It is optional, although at least one short or one long option must be given.

Note that the order of printing the options in the help and man pages follows the order of definition by **add\_opt**. The standard options are usually printed after the others by calling **print\_all\_opts**.

Before **add\_opt** is called, a callback function named **\_\${name}\_func** must have been defined. **add\_opt** will immediately call the callback function (with no arguments) to check for its presence. Later on, if that option is given on the command line, the callback function will be called from **process\_opts** with 2 arguments, namely the option string that triggered the function and the value associated with the option on the command line, if any.

### **del\_opt**

Deletes option(s) (by name). This can be used, for example, to back out one of the built-in options eg QUIET, VERBOSE, VERSION, END\_OPTIONS. It's probably meaningless to remove the HELP option!!

### **add\_std\_opts**

Adds the 'standard options', namely

**help** (-h, --help),

**version** (-V, --version),

**verbose** {-v, --verbose),

**quiet** {-q, --quiet),

**end-options** (—),

**print-man-page** (--print-man-page)

**print-man-page** is a *silent* option, not printed in the help or man pages. It is primarily a tool for the developer and when the script is invoked with this option, **process-getopt(1)** prints a skeleton **man(1)** page on stdout and exits. This provides only a starting point for a typical man page, but at

least all the options are formatted, saving a fair amount of fiddly work. It should probably be called once the script is more or less finished before using the skeleton to produce a final manual page.

**call\_getopt** [ "\$@" ]

Uses the option data created by **add\_opt** to assemble arguments to be passed to **getopt(1)** and then calls it to re-order the arguments. See **getopt(1)** for full details.

**process\_opts** [ "\$@" ]

Takes each option from the command line and calls the callback function for that option (previously stored by **add\_opt**) with up to two arguments – the option letter or word itself and any argument, if appropriate. It is up to the callback function to take the correct action on receiving the option eg by setting a boolean or storing the value.

**print\_all\_opts** [ "\$@" ]

Prints all the options in a neatly formatted list. Normally called from the caller's usage function.

**print\_short\_flags**

Prints a concatenated list of all the short option letters which do not take a parameter. These can be considered flags or booleans. They are listed in the order they were added.

**print\_long\_flags**

Prints a concatenated list of all the long option strings which do not take a parameter. These can be considered flags or booleans. They are listed in the order they were added.

**print\_short\_args** [ *man* ]

Prints a concatenated list of all the short options which take a parameter together with the parameter name. They are listed in the order they were added. If any argument is given then the list is formatted for a man page.

**print\_long\_args** [ *man* ]

Prints a concatenated list of all the long options which take a parameter together with the parameter name. They are listed in the order they were added. If any argument is given then the list is formatted for a man page.

**print\_all\_args** [ *man* ]

Prints a concatenated list of all the options which take a parameter together with the parameter name. They are listed in the order they were added. If any argument is given then the list is formatted for a man page.

**get\_opt\_letter** *name*

Prints the option letter (if any) assigned to the option *name*. Available only after **add\_opt**.

**get\_opt\_string** *name*

Prints the long option string (if any) assigned to the option *name*. Available only after **add\_opt**.

**get\_opt\_sarg** *name*

Prints the argument label assigned to the short option *name*. Available only after **add\_opt**.

**get\_opt\_larg** *name*

Prints the argument label assigned to the long option *name*. Available only after **add\_opt**.

**get\_opt\_desc** *name*

Prints the description assigned to the option *name*. Available only after **add\_opt**.

**get\_opt\_name** *opt*

Prints the name of the option that uses the letter or long string *opt* (without the leading – or --). Available only after

**clean\_process\_getopt**

Cleans up **process\_getopt(1)**'s address space ready for another bout of option processing. See **command-processor** for a sample of use.

**OPTIONS**

- h, --help**  
print this help and exit
- V, --version**  
print version and exit
- v, --verbose**  
do it verbosely
- explicitly ends the options

**EXIT STATUS**

All the functions return 0 on success and non-zero on error except for **process\_opts** which returns the number of items which should be 'shift'ed off the argument list in order to remove the options and their arguments.

**ENVIRONMENT**

The following environment parameters are recognised by **process-getopt(1)**:

**PROG** Mandatory. This should be set to the program name, typically **\$(basename \$0)**

**VERSION**

Mandatory. This should be set to the version of the program. It will be printed in response to the **-V, --version** option.

**VERBOSE**

Optional. This should be set to the null string before calling **process\_opts** and it will be set to non-null if the **-v, --verbose** option is given.

**ARGUMENTS**

Optional. This should be set to the list of arguments that the script can take (ie the parameters after the options). It should also be used in the script's **usage()** function, if any.

**SHORT\_DESC**

Optional. This should be set to a one line description that will be inserted into the man page. It should also be used in the script's 'usage' function, if any.

**USAGE**

Optional. This should be a long string describing the command. It will be inserted into the help and man pages. It should also be used in the script's 'usage' function, if any. The text will be processed through **fmt(1)** so for best results each paragraph should be coded on a single line. See the **EXAMPLES** section.

**ARGP\_HELP\_FMT**

Optional. This is the same environment variable recognised by GNU's **argp(3)** C function – see <http://www.gnu.org/software/libtool/manual/libc/Argp-User-Customization.html> The following comma-separated clauses are supported here:

**short-opt-col=n** This prints the first short option in column n. The default is 2.

**long-opt-col=n** This prints the first long option in column n. The default is 6.

**opt-doc-col=n** This prints the documentation for options starting in column n. The default is 29.

**rmargin=n** This will word wrap help output at or before column n. The default is 79.

The default is:

**short-opt-col=2, long-opt-col=6, opt-doc-col=29, rmargin=79**

**STOP\_ON\_FIRST\_NON\_OPT**

If this is set, then **getopt(1)** will stop processing options as soon as the first non-option argument is reached without the user adding a **'--'**. This is useful in scripts that take another command as its arguments eg if we wrote a wrapper around **sudo(1)** we would otherwise have to write:

```
my_sudo -- ls -l /root
```

If **my\_sudo(1)** has

```
export STOP_ON_FIRST_NON_OPT=yes
```

then we can write:

```
my_sudo ls -l /root
```

## FILES

## EXAMPLES

See **example-script(1)** in this package for a documented sample of use. See **boilerplate(1)** in this package for a minimalist boilerplate. See **testecho(1)** in this package for a simple test program. See **command-processor(1)** in this package for a slightly more complex program that accepts commands which can themselves take options (like **openssh(1)** or **cvs(1)**). **tiny(1)** in this package for a very simple example

## NOTES

If the calling program defines a **usage()** function then it is called in response to the `--help` option. If not, a default usage function is provided which relies on the environment parameters and the options added by **add\_opt()** to automatically format a help page.

**process-getopt(1)** uses **tput(1)** to determine the width of the terminal for the help page.

**process-getopt(1)** is designed to be portable to versions of **getopt(1)** that do not support long options, although this has not been extensively tested.

## BUGS

The `-V`, `--verbose` and `--` options are not very useful when calling **process-getopt(1)** as a script - but then, that's a bit lame anyway.

You probably don't want to source **process-getopt(1)** from the command line as it will log you off unless you have the prerequisite environment set up.

## SEE ALSO

**getopt(1)**

<http://sourceforge.net/projects/process-getopt>

<http://process-getopt.sourceforge.net>

<http://bhepple.freeshell.org/oddmuse/wiki.cgi/process-getopt>

## AUTHOR

Written by Bob Hepple <[bhepple@freeshell.org](mailto:bhepple@freeshell.org)>

## COPYRIGHT

Copyright (c) 2008-2009 Robert Hepple

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA