An Overview of the *SETOPT* Command Line Option Parser Generator

Gary Perlman

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

SETOPT is a set of macros for generating C functions to parse command line options for UNIX System (TM Bell Laboratories) commands. A simple language describes all the options allowed with a program. Each option is defined by its name, purpose, data type, size, range, and other information. This information helps generate program code to standardize the interface between users and the program, and between the programmer and the parser. Programmers can assume a perfect user interacting with their programs because on-line help and diagnostic messages tell users what is available and makes sure that legal option values are supplied. Extra abilities include generating summary documents and menu/form-filling interfaces.

The Problem

With no consistent syntax to UNIX System commands, people are bound to have problems learning and using commands. UNIX System commands consist of a program name, a set of options that control the behavior of the program, and arguments (usually file names). Traditionally, these options are single characters preceded by a dash, and if a value is to be supplied (logical true/false options take no values), it follows the option. For example, a simple text formatter, tf, might have an option to delete leading blank space on lines (-d), and one to control the width of lines (-w), and be called with options as follow.

```
tf -d -w 70
```

This syntax is a simplification because some options for UNIX System commands are multi-character names, and some are not preceded by a dash. Some values for options immediately follow the option name, while others are preceded by a space.

Hemenway and Armitage (1984) proposed a syntax standard based on an analysis of the standard set of over 400 UNIX System commands. Their proposal included a set of 13 rules for the syntax of commands and options and was meant to insure the consistency of commands developed in the future, and to guide the gradual migration of old commands toward a standard. Software to aid programmers in following the standard was recommended.

The published aid to parsing command line options, *getopt*, has some differences with the Hemenway and Armitage (1984) standard, but with some minor modifications, it could be consistent with the proposed syntax. To use *getopt*, programmers supply the names of the options and the ones taking values. For example, the string "abx:y:" defines two logical options, a and b, and two taking values, x and y. This *getopt* string is passed as an argument to the *getopt* function along with the unprocessed command line options, and *getopt* returns each option and its value (if appropriate) one by one. *getopt*'s sole purpose is to help parse the command line options.

The user interface provided by programs using *getopt* depends on programmer efforts. No standard on-line help nor diagnostic messages are provided; programmers must write their own. Programmers are also required to write code to check the data type and range of values supplied, and do any conversions and assignments of the basic string-type option values to internal variables of type integer, real, and so on. Many programmers do not do a good job at these tasks, leaving values unchecked or providing inconsistent error messages. Most do not do them all.

Presentation at the 1984
Winter USENIX Conference
Dallas, Texas.

Published in the conference
proceedings by USENIX Assn.

The Solution

SETOPT is a collection of text generators written in the macro text processing language **m4** (Kernighan, 1983). *SETOPT* helps programmers deal with command line options. A simple language describes the attributes of each option for a command, and this specification is expanded into C language functions to automate most of the tasks involved in dealing with options. The user interface provided by *SETOPT* is one that allows users to request help with options, gives diagnostic error messages when supplied values are incorrect, and promotes a consistent syntax that users have to learn once. The interface to the programmer allows the programmer to call the generated parser once, and when it returns, an error is reported, or the options are known to have been validated and set.

The language describing command line options is based on the interface language used by the S system for data analysis (Becker and Chambers, 1984). The S system uses a command language in which each option to a function has a name, a data type (like integer, or character), a size, a default value, whether it is required or not, and some other information. While the syntax for UNIX System command line options is different, and the requirements for the options differ, the S system interface language makes sure that options are correct, and that is the most basic goal of an option parser. *SETOPT* is more advanced than the S interface system in that it allows more user interaction and on-line help.

The *SETOPT* option specification language is based on two macros:

```
PGM(NAME, HELP, PROCESS)
OPT(FLAG, VAR, HELP, TYPE, SIZE, DEFAULT, TEST, ACTION)
```

written using the general purpose **m4** macro pre-processor (Kernighan, 1983). The PGM macro specifies the name and purpose of the program, and the data type of arguments processed by the program; these are used in on-line help and error messages. The OPT macro specifies the attributes of UNIX System command options for the program.

FLAG	A single character name for the option to be used by the program user. This is checked by *SETOPT* to make sure it conforms to the standard of being alphanumeric.
VAR	A program variable that will be set by *SETOPT* to be used by the programmer. This is used to name a component in a data structure that contains all the options.
HELP	A short phrase describing the option.
TYPE	The data type of the option. It can be one of: logical, string, character, integer, real, file, directory, date, time.
SIZE	A size of zero implies a scalar value while larger values imply an array of values.
DEFAULT	The default value of a scalar option if the user does not specify it.
TEST	An expression to test the validity of a supplied option value.
ACTION	Actions implied by setting an option. For example, setting one option often implies other are set.

These option description parameters evolved with use of *SETOPT*. For example, new data types were added as their need became known, and others might be added. Also, some parameters have been added (e.g., implied actions) after it was realized that their inclusion would reduce programmer effort. Other parameters that might be supported in the future include a way to require certain options be set and allow pre-validation transformations (e.g., mapping to upper/lower case to allow for more general date formats).

The programmer describes all allowed options and runs a *SETOPT* program to generate a function to parse those options. Generating a parser has advantages over having a single parsing function to handle all possibilities. The generated parser is as compact as possible to deal with the data types and ranges of options particular to a given program. A parser that could check all kinds of data types and test range validity with a variety of expressions would be large and adversely affect portability to small systems. With a generated parser, if specific types of options, say dates or times, are not used, then code to test for them need not be generated. Generated code uses compiled validity test expressions, which are faster than any interpreted function. This is not a large saving because command line option parsing is done once, and for a few options. The real saving is the space that an interpretive expression evaluator would take.

When the programmer calls the parser, usually the first statement in a program, several activities take place before it returns.

* Each option and any value are taken from the command line.
* The option is checked to make sure it is recognized. If not, an error message is printed.
* If it takes a value, *SETOPT* checks to make sure there is a value supplied. An error message indicates when an option value is missing.
* The type of the value is checked. An message indicating the correct type is printed for invalid inputs.
* The option, in the string format typed by the program user, is converted to the correct type format and assigned to the program variable named in the option specification language. This variable is an entry in a structure defined in a header file created by a *SETOPT* generator. Access to this structure is done with macros to simplify the programmer's task and guard against possible changes in the structure standards.
* The validity (for example, range) test is applied to the converted value. If there is an error, the value is reset to the default value after an error message is printed.
* Any implied actions, like assignments, are made.

If an error is detected, a standard error message is printed naming the program and the problem. Inappropriate actions are not taken after an error; for example, implied actions take place if all previous activities were error free. When errors are detected, a negative value is returned by the parser. Usually, the program will exit after an uncorrected error. Note the detail required to deal properly with just one option, and the need for a programmer's attention to make sure that error messages conform to standards. Besides making the parsing task easier for programmers, *SETOPT* makes sure that option handling is high quality.

SETOPT provides several built-in options available in every program. Some of these are hand programmed into many programs.

help	Get on-line help about the attributes of all the options.
synopsis	Get a short synopsis of the command usage.
version	Get the version date of the program options.
set	Set options interactively. This combines with on-line help to provide a menu interface.
read	Read options from a file.
done	Signal the end of options. This is part of the syntax standard and allows program arguments (like files) to begin with a dash.
exit	Exit from the program.
shell	Temporarily leave the program to run another program.
chdir	Change the working directory of the program.

Because any effort put into *SETOPT* affects all the programs using it, the large programming task of adding many built-in options is economical.

Additional Features

SETOPT is designed to allow simplified versions for aesthetic, security, or program-size reasons. The generated parser has program code for all the operations described in this paper, but not all have to be compiled into the final program. Along with the parsing program code are special control statements (for the C compiler pre-processor) so that the programmer can independently control at compilation time whether on-line help is to be supplied, options can be set interactively or from files, or option values are checked.

SETOPT can help document programs by generating summaries of options based on the option specification language. Even programmers who take the time to write documentation find it is difficult to keep the documentation up to date with the program code, and often, important details are omitted. When the option parser and the documentation are generated from the same source, as in *SETOPT*, the documentation is guaranteed to be accurate. For the parser, *SETOPT* generates C program code, while for the documentation, *SETOPT* generates text in the **troff** text formatting language used on the UNIX System. Part of an example manual entry is shown at the end of the paper. A collection of target languages, including a user interface programming language (Vo, 1984) that provides a menu and form-filling interface to programs, are possible and have been prototyped.

Conclusion

SETOPT provides programmers with a tool for handling command line options for UNIX System commands. People using *SETOPT*-based programs benefit from a consistent user interface with standard built-in option and standard error messages. To the extent that the same conventions are used in a large set of commands, people will find programs easier to learn and use; there is activity to standardize the user interface of all UNIX System commands. From the programming perspective, programmers are spared the tedious programming of the routine chores of parsing and checking options and providing on-line help and error messages. They do not have to worry about meeting any syntax standard because *SETOPT* generates parsers that follow the Hemenway and Armitage (1984) standard. In addition, programmers are able to generate up-to-date user documentation with minimal effort.

An Example

The example below shows part of the option specification for a simple text formatter. From this compact description, about 400 lines of parsing code are produced, resulting in a substantial time savings for programmers. With no extra effort, all the information is available to users in a more humane format, both on-line and in standard UNIX System manual format. Removed from the example is supplementary text, added by the programmer, to make the manual entry more informative. Some of this is shown in the sample manual entry at the end of the paper.

```
PGM(tf, Simple Text Formatter, FILES)
OPT(b, breaklines, Break ALL Lines of Text)
OPT(c, center, Center Input Lines, LGL, 0, FALSE)
OPT(d, delspace, Delete Space Around Lines)
OPT(i, indent, Indent Output Lines, INT, 0, 0, value >= 0)
OPT(j, justify, Justify the Right Margin)
OPT(N, number, Number Output Lines, LGL, 0, FALSE)
OPT(p, paginate, Paginate Output)
OPT(s, spacing, Output Line Spacing, INT, 0, 1, value > 0)
OPT(t, tabs, Absolute/Relative Tab Stops, INT, 20)
OPT(w, width, Width of Output Lines, INT, 0, 72, value > 0)
```

Availability

SETOPT was developed at AT&T Bell Laboratories and they retain the rights to its distribution. Their plans for *SETOPT* are not known. Since leaving Bell Labs, I have been developing a new version of *SETOPT* based on the details in this publicly released paper. If AT&T does not force the issue, I will be happy to distribute the macros for the parser and manual entry generators. These macros depend on the version of **m4** distributed with UNIX System V, although a version may be possible without the dependence on its new features.

In the meantime, I recommend using the simple *getopt* command line option parser. It will help standardize the user interface of your programs and it will help structure your code so that migration toward the Hemenway & Armitage syntax standard or toward an advanced parser like *SETOPT* is simplified. Although part of System V, there are public domain versions of *getopt* that have been posted to the UNIX System network.

Change of Address

More information about *SETOPT* can be obtained from me at my new address:

Professor Gary Perlman
Wang Institute of Graduate Studies
Tyng Road
Tyngsboro, MA 01879
(617) 649-9731
wivax!perlman or sdcsla!perlman

References

Becker, R. A. & Chambers, J. M. (1984) S: A Language and System for Data Analysis. New York: Wadsworth.

Kernighan, B. W. (1983) **m4**: A Macro Preprocessor. Standard UNIX System documentation.

Hemenway, K. & Armitage, H. (1984) Proposed Syntax Standard for UNIX System Commands. Paper presented at the 1984 Winter USENIX conference, Washington, D.C.

Vo, K.-P. (1984) Integration, Interaction: The IFS Approach. Paper submitted to the AT&T Bell Laboratories Technical Journal.

NAME

tf Simple Text Formatter

SYNOPSIS

tf [-bcdjnp] [i indent] [s spacing] [t tabs] [w width] [-] [files]

SUMMARY

Option	Purpose	Default	Range
-b	Break ALL Lines of Text	FALSE	
-c	Center Input Lines	FALSE	
-d	Delete Space Around Lines	FALSE	
-i I	Indent Output Lines	0	value >= 0
-j	Justify the Right Margin	FALSE	
-n	Number Output Lines	FALSE	
-p	Paginate Output	FALSE	
-s I	Output Line Spacing	1	value > 0
-t I20	Set Absolute/Relative Tab Stops		
-w I	Width of Output Lines	72	value > 0

DESCRIPTION

tf is a simple text formatter. It formats plain text files by filling, spacing, indenting, numbering, centering, or setting tabs based on program options. This program has *all the options*, almost.

tf processes the named files, but if no files are specified, the standard input is read. If a single dash is supplied as an argument, the standard input is read there; this allows inserting the standard input into a list of files.

OPTIONS

The following options are allowed with *tf*. Options should be flagged with a dash ( ) followed by the option character. If the flag takes a value, then it should be the next string on the command line. In the options below, option characters followed by a letter take values. I flags are followed by an integer. Logical flags (TRUE/FALSE variables) take no values.

tf begins by reading options in a file in your login directory called *.setopt/tf* (if its exists). Each line of this file should begin with an option character followed by a value, if appropriate. Do *not* use the flag option indicator ( ). After reading the option file, options on the command line are processed.

Detailed Description of Options

<detail omitted>