FLEXOP: a Flexible Command Option Parsing Library

VERSION 1.0

Hui Liu

2018

Contents

1 Introduction

1.1 Overview

Command options (arguments) are important parts of Unix, Linux and Mac OS operating systems. In these operating systems, most command line utilies (programs, applications) have options, such as `ls`. Its manual can be read by running command `man ls`, which should be similar to the following,

```
       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       -b, --escape
              print C-style escapes for nongraphic characters

       -B, --ignore-backups
              do not list implied entries ending with ~

       -C     list entries by columns

       -d, --directory
              list directories themselves, not their contents

       -D, --dired
              generate output designed for Emacs' dired mode

       -f     do not sort, enable -aU, disable -ls --color

       -h, --human-readable
              with -l and -s, print sizes like 1K 234M 2G etc.

       -l     use a long listing format
```

The arguments start with ”-” are options, which control the behavior of a command line utility, such as `ls -a` will show hidden files, and `ls -l` will display output in long listing format, which shows file size, time stampes and attributes. Another advantage of options is that a utility will be friendly to script programming and automation.

1.2 Credit

The original code was from PHG (http://lsec.cc.ac.cn/phg/), a parallel framework for adaptive finite element methods.

1.3 License

The package uses GPL license. If you have any issue, please contact: hui.sc.liu@gmail.com

1.4 Citation

If you use FLEXOP library, please cite it like this,

```
@misc{flexop-library,
    author="Hui Liu",
    title="FLEXOP: a flexible command option parsing library",
    year="2018",
    note={\url{https://github.com/huiscliu/flexop/}}
}
```

1.5 Website

The official website for FLEXOP is https://github.com/huiscliu/flexop/.

2 Installation

FLEXOP uses `autoconf` and `make` to detect system parameters and user set parameters, to build and to install.

2.1 Configuration

The simplest way to configure is to run command:

```
./configure
```

2.2 Options

The script configure has many options, if user would like to check, run command:

```
./configure --help
```

Output will be like this,

```
'configure' configures this package to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local/flexop]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, 'make install' will install all the files in
'/usr/local/flexop/bin', '/usr/local/flexop/lib' etc.  You can specify
an installation prefix other than '/usr/local/flexop' using '--prefix',
for instance '--prefix=HOME'.

Optional Features:
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
  --disable-assert        turn off assertions
  --enable-big-int        use long int for INT
  --disable-big-int       use int for INT (default),
  --with-int=type         integer type(long|long long)
  --enable-long-double    use long double for FLOAT
  --disable-long-double   use double for FLOAT (default)

Some influential environment variables:
  CC          C compiler command
  CFLAGS      C compiler flags
  LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
              nonstandard directory <lib dir>
  LIBS        libraries to pass to the linker, e.g. -l<library>
  CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
              you have headers in a nonstandard directory <include dir>
  CPP         C preprocessor
```

The default integer and floating point number are `int` and `double`. However, user can change the type of integer, such as `long long int`, by using options `--disable-big-int --with -int="long long"`, and the type of floating point number, such as `long double`, by using `--enable-long-double`. The integer number has three choices, `int`, `long int` and `long long int`, and the floating point number has two choices, `double` and `long double`.

2.3 Compilation

After configuration, `Makefile` and related scripts will be set correctly. A simple **make** command can compile the package,

```
make
```

2.4 Installation

Run command:

```
make install
```

The package will be installed to a directory. The default is /usr/local/flexop/. A different directory can be set by `--prefix=DIR`.

3 Utilities

3.1 Print

`flexop_printf` outputs to stdout.

```
int flexop_printf(const char *fmt, ...);
```

`flexop_error` prints output error message and quits with error code.

```
void flexop_error(int code, const char *fmt, ...);
```

`flexop_warning` print warning info.

```
void flexop_warning(const char *fmt, ...);
```

`flexop_set_print_mark` sets print function mark, if `m` is non-zero (true) value, then `flexop_printf` acts as a normal print function. However, if `m` is zero (false), `flexop_printf` will not print anything. This function is important to parallel computing, since only one process prints info to stdout usually.

```
void flexop_set_print_mark(int m);
```

3.2 Memory

The following functions provide memory allocation, calloc, reallocation, freeing and copying.

```
void * flexop_alloc(size_t n);
void * flexop_calloc(size_t n);
void * flexop_realloc(void *ptr, size_t n);
void flexop_free(void *p);
```

3.3 Conversion

`flexop_atoi` converts string to integer, which checks if input is legal integer.

```
FLEXOP_INT flexop_atoi(const char *ptr);
```

`flexop_atou` converts string to unsigned integer, which checks if input is legal integer.

```
FLEXOP_UINT flexop_atou(const char *ptr);
```

`flexop_atof` converts string to floating point number, which checks if input is legal.

```
FLEXOP_FLOAT flexop_atof(const char *ptr);
```

4 Option Management

4.1 Data Types

FLEXOP_FLOAT is the floating point number type in FLEXOP, which could be double or long double, depending on the configuration. Its formal definition is as follows.

```
#if FLEXOP_USE_LONG_DOUBLE
typedef long double       FLEXOP_FLOAT;
#else
typedef double            FLEXOP_FLOAT;
#endif
```

FLEXOP_INT is the integer type, and as mentioned before, it could be int, long int, or long long int.

```
#if FLEXOP_USE_LONG_LONG
typedef long long int     FLEXOP_INT;
#elif FLEXOP_USE_LONG
typedef long int          FLEXOP_INT;
#else
typedef int               FLEXOP_INT;
#endif
```

FLEXOP_UINT is the unsigned integer type, and it could be unsigned int, unsigned long int, or unsigned long long int, depending on the configuration.

```
#if FLEXOP_USE_LONG_LONG
typedef unsigned long long int     FLEXOP_UINT;
#elif FLEXOP_USE_LONG
typedef unsigned long int          FLEXOP_UINT;
#else
typedef unsigned int               FLEXOP_UINT;
#endif
```

The FLEXOP supports many option types, such as integer, unsigned integer, floating point number and vector, which is represented by FLEXOP_VTYPE. Its formal definition is shown by the follows.

```
typedef enum {
    VT_TITLE,

    VT_BOOL,
    VT_KEYWORD,
    VT_HANDLER,
    VT_INT,
    VT_UINT,
    VT_FLOAT,
    VT_STRING,

    VT_VEC_INT,
    VT_VEC_UINT,
    VT_VEC_FLOAT,
    VT_VEC_STRING,

} FLEXOP_VTYPE;
```

Here are detailed explanations:

- VT_TITLE defines a section. For example, in many applications, options may be divided into many sections, such as model, numerical, gridding, visualization. In FLEXOP, when a title (or section) is registered, all following options registered after the title belong to this section, unless a new section (title) is registered.
- VT_BOOL defines a boolean option, which has true (1) or false (0) status.
- VT_KEYWORD defines a keyword, whose value is from a pre-defined set.
- VT_HANDLER defines a user handler, which handles option parsing.
- VT_INT defines an integery.
- VT_UINT defines an unsigned integery.
- VT_FLOAT defines a floating point number.
- VT_STRING defines a string.
- VT_VEC_INT defines a vector of integers.
- VT_VEC_UINT defines a vector of unsigned integers.
- VT_VEC_FLOAT defines a vector of floating point number.
- VT_VEC_STRING defines a vector of strings.

The VT_HANDLER type requires a user-provided function, which has the following type. It returns 0 if successful, otherwise returns non-zero value.

```
typedef int (*FLEXOP_HANDLER)(FLEXOP_KEY *o, const char *arg);
```

The vector has a uniform definition as follows.

```
typedef struct FLEXOP_VEC_
{
    void *d;                        /* data */
    char *key;                      /* key */

    FLEXOP_VTYPE type;              /* data type of the members */
    FLEXOP_INT size;                /* size of the vector */
    ...

} FLEXOP_VEC;
```

4.2 Usage

The following code sample shows basic calling sequences, which has one optional step and three mandatory steps.

```
{
    /* 1: preset values (optional) */
    flexop_preset("-i 23");

    /* 2: register (mandatory) */
    flexop_register_int("i", "int", &i);
    flexop_register_float("f", "float", &f);

    flexop_register_vec_int("vi", "vector of int", &vi);
    flexop_register_vec_float("vf", "vector of float", &vf);

    /* 3: parse (mandatory) */
    flexop_init(&argc, &argv);

    /* 4: clean memory (mandatory) */
    flexop_finalize();
}
```

4.3 Registration

4.4 Setting Values

4.5 Getting Values

4.6 Auxiliary Functions