Argv Library

Version 2.7.0
February 2010

Gray Watson

Argv Library

The argv library has been designed to handle the argument processing needs of most Unix software and to provide a consistent usage framework for user applications.

The library is reasonably portable having been run successfully on at least the following operating systems: AIX, BSDI, DG/UX, FreeBSD, HPUX, Irix, Linux, MS-DOG, NeXT, OSF, Solaris, SunOS, Ultrix, Unixware, and even Unicos on a Cray Y-MP.

The package includes the library, configuration scripts, shell-script utility application, test program, and extensive documentation (text, texi, info, ps). The library and its documentation are available online at URL `http://256.com/sources/argv/`. See Section 2.2 [How To Get], page 3.

I can be reached via my web page `http://256.com/gray/` with any questions or feedback. Please include the version number of the library that you are using as well as your machine and operating system types.

Gray Watson.

1 Library Copying Conditions

2 How to Use the Library

2.1 The General Concepts Behind the Library

One thing that almost all Unix executables need to do is process the *command line arguments*. Whether this is to enable verbose mode or specify the files for a utility to work on, code has to be written to process these user specified options.

```
int main(int argc, char **argv)
{
        ...
}
```

As you must know, the command line arguments in most Unix systems are passed in as arguments to `main()` (seen above). The `argc` integer argument contains the number of arguments specified. The `argv` variable holds the arguments themselves. It can be thought of as a pointer to a list of character pointers – or an array of character pointers.

To get a particular argument from `argv`, you use `argv[x]` where `x` is an integer whose value is from 0 to `argc - 1`. In most Unix implementations, the zeroth argument is always the name the program was executed with. For instance, if you typed '`./ls -al`', `argc` would equal 2 and the value of `argv[0]` would be '`"./ls"`'. The value for `argv[1]` would be '`"-al"`'.

Currently, most programmers either write code on a per program basis to process arguments or they use the `getopt()` routine. Writing argument processing code for each program results in improper and inconsistent argument handling. Although better, `getopt()` does not provide the structure needed to ensure conformity in argument processing and still requires significant code to be written by the programmer.

The goal for this library was to achieve a standardized way of processing arguments – especially in terms of error and usage messages. Important consideration was also given to reducing the programming time necessary to enable the functionality.

2.2 How to get the library.

The newest versions of the argv library are available on the web at `http://256.com/sources/argv/`.

The versions in this repository also include such files as a postscript version of the manual and other large files which may not have been included in the distribution you received.

2.3 Installing the Library

To configure, compile, and install the library, follow these steps carefully.

1. Type *`sh ./configure`* to configure the library. You may want to first examine the '`config.help`' file for some information about configure. Configure should generate the '`Makefile`' and some configuration files automatically.

   NOTE: It seems that some versions of tr (especially from HP-UX) don't understand `tr '[a-z]' '[A-Z]'`. Since configure uses tr often, you may need to either get GNU's tr (in their textutils package) or generate the '`Makefile`' and '`conf.h`' files by hand.

2. You may want to examine the '`Makefile`' and '`conf.h`' files created by configure to make sure it did its job correctly.
3. Typing *`make`* should be enough to build '`libargv.a`' and the '`argv_shell`' utility. If it does not work, please send me some notes so future users can profit from your experiences.

   NOTE: The code is pretty dependent on a good ANSI-C compiler. If the configure script gives the '`WARNING`' that you do not have an ANSI-C compiler, you may still be able to add some sort of option to your compiler to make it ANSI. If there such is an option, please send it to the author so it can be added to the configure script.
4. Typing *`make tests`* should build the '`argv_t`' test program. This can be run and given arguments to test the various library features.
5. Typing *`make install`* should install the '`libargv.a`' library in '`/usr/local/lib`', the '`argv_shell`' utility in '`/usr/local/bin`', and the '`argv.info`' documentation file in '`/usr/local/info`'.

   You may have specified a '`--prefix=PATH`' option to configure in which can '`/usr/local`' will have been replaced with '`PATH`'.

See the Getting Started section to get up and running with the library. See Section 2.4 [Getting Started], page 4.

2.4 Getting Started with the Library

This section should give you a quick idea on how to get going.

1. Make sure you have the latest version of the library. It is available on the web at `http://256.com/sources/argv/`. See Section 2.2 [How To Get], page 3.
2. Follow the installation instructions on how to configure and make and install the library (i.e. type: *`make install`*). See Section 2.3 [Installation], page 3.
3. Examine the '`argv_t.c`' test program source to see an example of how to program with the library. After adding the appropriate `argv_t` structure array to your main source file, you need to compile and link your programs with the library.
4. The first time your program is run, the library makes a number of checks as to the validity of the argument structures being used. You may have to note and fix reported problems.
5. Run your program with the '`--usage`' argument and voila.

3 The Library's Operations

3.1 The argv_t Structure and It's Usage

The argv_t argument structure is as follows:

```
typedef struct {
        char    ar_short_arg;           /* short argument, 'd' if '-d' */
        char    *ar_long_arg;           /* long version of arg, '--delete' */
        short   ar_type;                /* type of variable */
        void    *ar_variable;           /* address of associated variable */
        char    *ar_var_label;          /* label for variable description */
        char    *ar_comment;            /* comment for usage message */
} argv_t;
```

The `ar_short_arg` element contains the character value of the short option ('d' for '`-d`') or special codes such as ARGV_LAST which identifies the last element in the array. See Section 3.2 [Special Short Args], page 5.

The `ar_long_arg` element (if not-NULL) holds the string which is the long version of `ar_short_arg`. For instance, with '`-d`', you might have "delete". This would mean that '`-d`' and '`--delete`' would be equivalent. '`--`' is the long-option prefix per POSIX specs.

You would define an array of these arguments at the top of the file with `main()` in it.

```
static char copy = ARGV_FALSE;

static argv_t args[] = {
  { 'c', "copy", ARGV_BOOL, &cp_files, NULL, "copy-files flag" },
  { 'g', "group", ARGV_CHAR_P, &group, "group", "name of group to set" },
  ...
  { ARGV_LAST }
};

...

int main(int argc, char ** argv)
{
  argv_process(args, argc, argv);
}
```

3.2 The Special ar_short_arg Values

There are 3 types of arguments:

optional Arguments that may or may not be supplied by the user.

mandatory

Arguments that must be supplied by the user. For instance grep must be given an expression on the command line.

If the argument is a mandatory argument which has no -%c prefix then the `ar_short_arg` element should be assigned ARGV_MAND.

maybe Arguments that might be specified by the caller but are not mandatory. For instance, you can grep a file or you can grep standard-input. The file should be a maybe argument.

If this is a maybe argument then use ARGV_MAYBE in the `ar_short_arg` field.

To mark the last entry in the structure list use ARGV_LAST.

3.3 The argv_t Structure and It's Usage

Ar_type holds the type of the argument whether an optional argument or mandatory. Below are the available values for this field.

`ARGV_BOOL`
: character type, sets the variable to ARGV_TRUE if used

`ARGV_BOOL_NEG`
: like ARGV_BOOL but sets the variable to ARGV_FALSE if used

`ARGV_BOOL_ARG`
: like ARGV_BOOL but takes a yes/no argument

`ARGV_CHAR`
: a single character

`ARGV_CHAR_P`
: a string of characters (character pointer)

`ARGV_FLOAT`
: a floating pointer number

`ARGV_SHORT`
: a short integer number

`ARGV_INT` an integer number

`ARGV_U_INT`
: an unsigned integer number

`ARGV_LONG`
: a long integer number

`ARGV_U_LONG`
: an unsigned long integer number

`ARGV_BIN` a binary base-2 number (0s and 1s)

`ARGV_OCT` an octal base-8 number (0 to 7)

`ARGV_HEX` a hexadecimal base-16 number (0 to 9 and A to F)

`ARGV_INCR`
: a integer type which is incremented each time it is specified

`ARGV_SIZE`
: a long integer size number which understands b for bytes, k for kilobytes, m for megabytes, and g for gigabytes

`ARGV_U_SIZE`
: an unsigned long integer version of ARGV_SIZE

`ARGV_BOOL_INT`
: like ARGV_BOOL except the variable is an integer and not a character

`ARGV_BOOL_INT_NEG`
: like ARGV_BOOL_NEG except the variable is an integer and not a character

`ARGV_BOOL_INT_ARG`
: like ARGV_BOOL_ARG except the variable is an integer and not a character

For printing out of the type of the argument on the command line, use the '`--argv-display`' option which will display the argument, its type and value. It will display the variables' default values if no arguments specified before it on the command line otherwise it will show the values the variables are set to after processing the arguments.

Basically the argument processing routines, examine the type of the variable, absorb another argument (if necessary), and then translate the string argument (if necessary) and write the data into the address stored in the ar_variable field.

ARGV_BOOL, ARGV_BOOL_NEG, ARGV_INCR, ARGV_BOOL_INT, and ARGV_BOOL_INT_NEG are special in the above list in that they do not require another argument. With '`ls -l`', for example, the '`-l`' flag lives on its own. With '`install -m 444 ...`', on the other hand, '`444`' is an octal number argument associated with '`-m`' and will be translated and assigned to the '`-m`' mode variable.

3.4 Using Arguments Which "Absorb" Arrays.

Needs to be written. Sorry.

4 Invoking Programs Which Use the Library

4.1 How to get usage messages from argv programs

If a program '`install`' has the library compiled in you should be able to do a '`install --usage-long`' to get the long-format usage message.

```
Usage: install
  [-c]              or --copy-files       = copy file(s), don't move %t
  [-g group]        or --group-id         = group id name (default bin) %s
  [-m octal-mode]   or --mode-value       = permissions mode value %o
  [-o owner]        or --owner-id         = owner id name (default bin) %s
  [-s]              or --strip            = strip destination binary %t
  [file(s)] directory/destination         = files to install or mkdir arg
```

In the above example, the program install's usage message is detailed. The '`[-c]`' line details the copy-files flag. You can either enable it with a '`-c`' or '`--copy-files`'. The description of the flag follows with lastly, a '`%t`' showing that it is a *true/false* flag.

The '`[-g]`' line shows the group-id flag. It is different from the '`-c`' flag since, if used, it takes a group string argument (notice the '`%s`' at the end of the line indicating it takes a string argument).

'`install --usage-short`' or just '`--usage`' will get you a condensed usage message:

```
Usage: install [-cs] [-g group] [-m octal-mode] [-o owner] [file(s)]
       directory/destination
```

4.2 How to Specify Arguments to Argv Programs

Specifying arguments to a program which uses the library is quite straight-forward and standardized. Once you have learned how to do it once, you can use any program with it.

There are five basic types of arguments as defined by the library:

true/false flags

Do not have an associated value and the program will get a True if one is specified else False.

The '`-c`' in '`install -c`'.

variable flags

Have an associate value which will be supplied to the program.

The '`-m`' in '`install -m 0644`' will get the value '`0644`'.

values Arguments without a '`-`' and are associated values for the variable flags.

mandatory

Arguments without a '`-`' but are *not* associated to variable flags. These can be supplied to the program if allowed. They are mandatory in that they must be supplied. If the program asks for 3 arguments, 3 must be supplied. *NOTE* that order is important with these.

The '`from`' and '`to`' arguments in '`install from to`'.

maybe These are the same as the mandatory arguments except they are optional arguments and can but do not have to be supplied.

The '`file`' argument in '`ls file`' since '`ls`' does not require a file to be listed to work.

The values for the variable flags are assigned in a straight First-In-First-Out queue. In '`install -m -g 0644 bin`', the value '`0644`' is assigned to the '`-m`' flag and the value '`bin`' is assigned to '`-g`'.

Additional values that cannot be matched to variable flags will become mandatory or maybe arguments if the program is configured to accept them.

```
install from -c -m -g 0644 -o wheel -s jim to
```

In the previous convoluted example, '`from`' and '`to`' are mandatory arguments, '`-c`' and '`-s`' are true/false flags, '`-m`' gets assigned '`0644`', '`-g`' gets '`wheel`', and '`-o`' gets '`jim`'. It would be much easier to write it as:

```
install -cs -m 0644 -g wheel -o jim to from
```

4.3 Long Versus Short Arguments

Needs to be written. Sorry.

4.4 Global Settings For All Argv Programs

An *environment variable* is a variable that is part of the user's working environment and is shared by all the programs. The '`GLOBAL_ARGV`' variable is used by the argv library to customize its behavior at runtime. It can be set by hand and should probably be entered into your shell's runtime configuration or *RC* file.

To set the variable, C shell (csh or tcsh) users need to invoke:

```
setenv GLOBAL_ARGV value
```

Bourne shell (sh, bash, ksh, or zsh) users should use:

```
GLOBAL_ARGV=value
export GLOBAL_ARGV
```

The value in the above example is a comma separated list of tokens each having a corresponding value. The tokens and their values are described below:

- close – close argument acceptance

  Enables the handling of arguments such as '`-m=444`' where '`-m`' is a flag and '`444`' is its value.

  Values: disable, enable.

  - disable – treat '`=`' like a normal argument
  - enable (default) – enable the '`-x=10`' format
- env – environment variable handling

  Enables the processing of the '`ARGV_*`' variables. If you have a set of options that you always use for '`ls`' for instance, you cat set the '`ARGV_LS`' environmental variable to hold these options. For instance: '`setenv ARGV_LS "-sCF"`'.

  Values: none, before, after.

  - none – No processed at all
  - before (default) – options from env variable are processed Before command line
  - after – env options processed After command line
- error – handling of usage errors

  Whenever you do not use a command correctly, this token determines how the library reports errors to you.

  Values: none, see, short, shortrem, long, all.
  - none – on errors print nothing but error message
  - see (default) – on errors print see –usage for more info.
  - short – on errors print the short-format usage messages
  - shortrem – on errors print short-format + how to get long
  - long – on errors print the long-format usage messages
  - all – on errors print the long-format usage messages + help, etc.
- multi – the handling of arguments specified more than once

  If you use am argument twice on the command line, this token determines if the library should say it is an error.

  Values: accept, reject.
  - accept (default) – it's NOT an error if specified more than once
  - reject – it's an error if specified more than once
- usage – usage messages for –usage

  Determines what messages the library prints when you use the '`--usage`' option.

  Values: short, shortrem, long, all.
  - short (default) – default is the short-format messages
  - shortrem – default is the short-format messages + how to get long
  - long – default is the long-format messages
  - all – default is the long-format messages + help, usage, version

Examples:

```
          # accept -x=10, no env variables, long messages on errors,
          # accept multiple uses, and print all messages on --usage.
  setenv GLOBAL_ARGV close=accept,env=none,error=long,multi=accept,usage=all

          # process env variable options before command line,
          # and reject multiple argument uses
  setenv GLOBAL_ARGV env=before,error=long,multi=reject
```

4.5 Arguments For a Specific Argv Program

Needs to be written. Sorry.

Concept Index

Table of Contents