

Boost.ProgramOptions

Dmitri Nesteruk
<http://activemesa.com>
dn@activemesa.com



Overview

- `main(int argc, char *argv[])`
- `git commit -m "some changes"`
 - `argc = 4`
 - `argv[0] = c:\git\git.exe`
 - `argv[1] = commit`
 - `argv[2] = -m`
 - `argv[3] = some changes` *quotes aren't included*
- Arguments can be required or optional
 - With or without default values
- Arguments can have longer and shorter form
 - `git commit -a` or `--all`
- Arguments must be self-documenting
- Arguments can be external (e.g. in config file or environment vars)
- `Boost.ProgramOptions` covers these needs!

```
C:\Users\Dmitri>git
usage: git [--version] [--exec-path=DIR]
        [-p|--paginate|--no-pager]
        [--git-dir=<path>] [--work-tree=<path>]
        [-c name=value] [--help]
        <command> [<args>]

The most commonly used git commands are:
add      Add file contents to the repository
bisect   Find by binary search the commit that introduced a bug
branch   List, create, or delete branches
checkout Checkout a branch or file from a branch
clone    Clone a repository into a new directory
commit   Record changes to the repository
diff     Show changes between commits or commit and working tree
fetch    Download objects and refs from another repository
grep     Print lines matching a pattern
init     Create an empty git repository or clone
log      Show commit logs
merge    Join two or more development branches
mv       Move or rename a file, a directory, or a symlink
pull     Fetch from and merge with the upstream (remote)
push     Update remote refs along with associated files
rebase   Forward-port local commits to the latest upstream version
reset    Reset current HEAD to the specified commit
rm       Remove files from the working tree and from the repository
show     Show various types of objects
status   Show the working tree status
tag      Create, list, delete or verify a tag name pointing to a commit
```

See 'git help <command>' for more information

Fundamentals

- **options_description**

- Root container
- All allowed options declared here
- Writing to a stream
`cout << desc << endl;`
prints documentation

- **od.add_options()**
("help", "print help message")

Fluent interface for defining options. Can customize

- Default values
- Required/optional
- Prefix (default - -) allows delimiting
`foo.exe --src`
`a.txt b.txt c.txt`

- **variables_map**

- Stores parsed variables
- `vm.count("foo")` counts # of "foo" arguments (args may repeat)
- `vm["foo"]` gets the value of the argument

- **Namespace-global functions proxy from argc/argv to variables_map**

- `store()` used to get a parser to parse args and store in map
- `notify()` must be called after parsing – needed in advanced scenarios

Customization

- **Default value**
 - `po::value<string>()->default_value("file.txt")`
- **Binding to variable**
 - `int n;`
`po::value<int>(&n)`
- **Multiple aliases**
 - `("filename,F", po::value<string>(), ...)`
 - Lets you pass argument with `--filename` or `-F`
- **Multiple option arguments**
 - `("files", po::value<vector<string>>())`
 - `foo.exe --files a.txt b.txt`
- **Positional arguments**
 - Command-line tokens which have *no* option name, i.e.
 - Treat
`foo.exe myfile.txt`
as
`foo.exe --filename myfile.txt`
- `po::positional_options_description`
- `p.add("filename", -1);`
- **Explicitly create a `command_line_parser` and initialize it with positional options**
 - `.positional(p).run()`

Summary

- `#include <boost/program_options.hpp>`
`namespace po = boost::program_options;`
- Describe all options in `options_description`
- Positional options go into `positional_options_description`
- Create a `variables_map` to store the options
- Create a parser, call `store()` and then `notify()` to assign
- Check if option is provided with `vm.count("foo")`
- Get the option with `vm["foo"]` or bind to variable