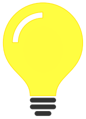
CLI11: Command line parsing made simple

Henry Schreiner

April 24, 2017

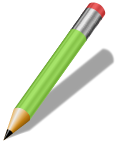
History



Origins in GooFit

- Analysis code in GooFit consists of two things:
  - PDFs, written in advanced CUDA/OpenMP
  - The model code
- GooFit tries make the model code simple
- But a lot of code was a command line or option parser
- Or (worse) lots of hard-coded values
- Lots of segfaults in examples from option errors

Alternatives



Requirements

- Clean and simple usage
- Plain types, no runtime lookup
- Easy to include
- Standard shell idioms
- Subcommands
- Configuration files
- Extendable and customizable by a toolkit

Alternatives

Boost Program Options

- Classic standard parser
- Big dependency for a library
- Hard to exit cleanly
- Peculiar syntax
- Interesting tidbit: CLI11 started as a wrapper to `Boost::PO`

A few others

- TCLAP: Header-only, but limited, poor support
- GFlags: Google's attempt, nice syntax, but too many macros, no subcommands

CLI11

CLI11

- Designed to mimic `plumbum.cli`, but native to `C++11`
- Expanded to include features from other libs, like Click
- Header only, single header file option
- Only depends on `C++11` (no regex required)
- Used stand alone or subclassed

Well tested

- Continuous Integration (CI) on Linux, Mac, and Windows
- GCC 4.7 and 6, Clang 3.5 and Mac, and Visual Studio 2015
- 100% test coverage on CodeCov, almost 200 tests
- Single header file version compiled from online build
- API documentation generated from online build
- Every function, method, and member documented

Command line parsing

```
./myprog 1 -vz -ffilename --long=2
```

Example 1

- The `1` is a "positional" option
- The `-v` is a short flag
- The `z` is a chained short flag
- The `-f` is a short option accepting an argument
- `filename` is the argument
- `--long` is a long option, followed by space or =

Command line parsing

```
git checkout -q -- myfile.txt
```

Example 2

- `checkout` is a subcommand
- `-n` is a short flag
- `--` is a positional separator
- Everything after that is a positional

Design

```
CLI::App app {"A discription"};

// Add options (next slide)

try {
    app.parse(argc, argv);
} catch (CLI::Error &e) {
    return app.exit(e);
}
```

Basics

- The parser is an instance of a `CLI::App`
- You set up your options (next slide)
- Parsing is (correctly) done with a `try` statement

Flags

```
app.add_flag("-n,--name", output, "Help string");
```

Flags

- All apps get a default help flag
- Names are given in a comma separated string
  - A "-" name is a short option
  - A "--" name is a long option
- SFINAE is used to select behavior, int-like or bool

Options

```
std::string output = "default";
app.add_option("filename", output, "Help string");
```

Options

- 6 behaviors: (int, float, string)-like $\times$ vector
- Works with `TStrings`, `boost::filesystem`, etc.
- Also a version accepting a transformation function
- Optional final `true` captures default value in help
- A name without “-” is positional

Options

Specialty

- `add_set`: Pick from a set
- `add_complex`: A complex number
- `add_config`: Add a option for config file

Pointer to options

- Adding options returns pointers
- The behavior of the option can be modified
- The option can be counted

Options

```
TString fname;
app.add_option("-f", fname, "Existing file")
    ->required()
    ->check(CLI::ExistingFile);
```

Normal usage example

- Configuring an option is simple
- Pointer often not needed

Option modifiers

- `->required()`
- `->expected(N)`
- `->requires(opt, ...)`
- `->excludes(opt, ...)`
- `->envname(name)`
- `->group(name)`
- `->ignore_case()`
- `->check(CLI::ExistingFile)`
- `->check(CLI::ExistingDirectory)`
- `->check(CLI::NonexistentPath)`
- `->check(CLI::Range(min,max))`

Subcommands

```
auto subcom = app.add_subcommand("pull", "Help str");
```

Subcommands

- Subcommands are just `CLI::App`'s
- Same features
- Can chain infinitely

Suggestion

- Use `auto& subcom = *app.add_subcom(...` to get reference

Command/Subcommand modifiers

- `.ignore_case()`
- `.fallthrough()`
- `.require_subcommand()`
- `.require_subcommand(N)`
- `.set_callback(function)`
- `.allow_extras()`
- `.get_subcommands()`

Subcommands

```
if(subcom->parsed()) ...
for(auto subcom : app.get_subcommands()) ...
subcom->set_callback([&](){...});
```

Three ways to use subcomands

- Check to see if they were parsed
- Run over list from `.get_subcommands()`
- Use callbacks to program inline
  - Correct parse ordering by CLI11

Best method depends on application

Configuration

```ini
; Example of INI file, [default] is assumed
value = 1.23
subcom.flag = true
```

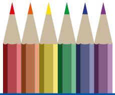
Ini files

- Support for configuration files
- Can read or produce INI
- Mixes with command line
- Subcommands, flags, etc. are all supported.

Environment variables

- Environment variable input can be added

Library integration



Library integration

- Supports customization for the main `App`
- Several hooks provided

Example integration: `GooFit::Application`

- Adds custom options for info and GPU control
- Adds MPI support setup/teardown
- `TApplication` style constructor/`run`
- Color support through Rang

PiPiPi0 Example

```
[root@8566ea534714 pipipi0DPFit]# ./pipipi0DPFit -h
pipipi0 Dalitz fit example
Usage: ./pipipi0DPFit [OPTIONS] [SUBCOMMAND]

GooFit:
  --goofit-details              Output system and threading details

Options:
  -h,--help                     Print this help message and exit
  --config STRING=config.ini    An ini file with command line options in it

Subcommands:
  toy                           Toy MC Performance evaluation
  truth                         Truth Monte Carlo fit
  sigma                         Run sigma fit
  efficiency                    Run efficiency fit
  canonical                     Run the canonical fit
  background_dalitz             Run the background Dalitz fit
  background_sigma              Run background sigma fit
  background_histograms         Write background histograms
  run_gen_mc                    Run generated Monte Carlo fit
  make_time_plots               Make time plots
```

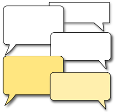
```
[root@8566ea534714 pipipi0DPFit]# ./pipipi0DPFit canonical -h
Run the canonical fit
Usage: ./pipipi0DPFit canonical [OPTIONS] data

Positionals:
  data STRING                 Data to use

Options:
  -h,--help                   Print this help message and exit
  -d,--data STRING            Data to use
  --luckyFrac FLOAT=0.5
  --mesonRad FLOAT=1.5
  --normBins INT=240
  --blindSeed INT=4
  --mdslices INT=1
  --offset FLOAT=0            Offest in GeV
  --upper_window INT=2
  --lower_window INT=-2
  --upper_delta_window FLOAT=2
  --lower_delta_window FLOAT=-2
  --upperTime FLOAT=3
  --lowerTime FLOAT=-2
  --maxSigma FLOAT=0.8
  --polyEff UINT=0
  --m23Slices INT=6
  --bkgRandSeed INT=-1
  --drop-rho_1450
  --drop-rho_1700
  --drop-f0_600
  --histSigma
  --makePlots
  --mkg2Model STRING in {histogram,parameter,sideband}=sideband
```

Status

Nearing 1.0 release

- Current version 0.9 was released yesterday
- API stable for last several versions
- Final tasks:
  - Evaluate user feedback
  - Evaluate compatibility with ROOT 6 or 7
- GooFit 2.0 release will be proceeded by CLI11 1.0

Try it out!



Three easy ways to try it

- Download CLI11.hpp from the latest release
- Get the git repository
- Use CLIUtils CMake `AddCLI.cmake` to automatically download
    - Look at `FindROOT.cmake` and other helpers

Get involved

- Open an Issue or a Pull Request
- Chat on Gitter