Elie Khairallah

Internship Report

24 August 2018

Centralisation of ROOT Help Systems

1. Introduction

ROOT is a modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.
ROOT is an enormous program and contains a lot of commands, which makes it hard to memorise them all as well as their options. That is why for each command there is usually two references for help:

- A manual page for the command which consists of a file with a ".1" extension, written in a special language which is named groff.
- A command line options help, which is displayed when writing command followed by the help flag on a command line (for ex: root -h). The code to generate this is usually embedded in the code of the called command.

For a while, things were good. A programmer would typically write a command or new option, and then update the corresponding man page and command line option code. But with time, this started to cause problems ..

Mainly the problems were caused by 2 major things:

- The programmers would forget to update one of the man page or command line option, so after a while those two become out of sync one with the other.
- Each programmer would write his documentation in his own way, making the documentations not adhering to a certain pattern which is confusing to the users.

So we needed to centralise the generation of help so that the man pages and command line options stay always in perfect sync and we also needed to create a certain pattern.

2. Examples of Different Ways that Command Line Options Would be Written:

a. In a header as raw string

```
constexpr static const char kCommandLineOptionsHelp[] = R"RAW(
Usage: %s [-l] [-b] [-n] [-q] [dir] [[file:]data.root] [file1.C ... fileN.C]
Options:
  -b : run in batch mode without graphics
  -x : exit on exception
  -e expression: request execution of the given C++ expression
  -n : do not execute logon and logoff macros as specified in .rootrc
  -q : exit after processing command line macro files
  -l : do not show splash screen
  -t : enable thread-safety and implicit multi-threading (IMT)
  --web: display graphics in a web browser
  --notebook : execute ROOT notebook
  dir : if dir is a valid directory cd to it before executing

  -?      : print usage
  -h      : print usage
  --help  : print usage
  -config : print ./configure options
  -memstat : run with memory usage monitoring

)RAW";
```

Fig.1

b. In the main of the function as text display

```
int main( int argc, char **argv )
{
   if ( argc < 3 || "-h" == std::string(argv[1]) || "--help" == std::string(argv[1]) ) {
      std::cout << "Usage: " << argv[0] << " [-f[fk][0-9]] [-k] [-T] [-O] [-a] \n"
      "            [-n maxopenedfiles] [-cachesize size] [-j ncpus] [-v [verbosity]] \n"
      "            targetfile source1 [source2 source3 ...]\n" << std::endl;
      std::cout << "This program will add histograms from a list of root files and write them" << std::endl;
      std::cout << "   to a target root file. The target file is newly created and must not" << std::endl;
      std::cout << "   exist, or if -f (\"force\") is given, must not be one of the source files." << std::endl;
      std::cout << "   Supply at least two source files for this to make sense... ;-)" << std::endl;
      std::cout << "If the option -a is used, hadd will append to the output." << std::endl;
      std::cout << "If the option -k is used, hadd will not exit on corrupt or non-existant input\n"
                   "   files but skip the offending files instead." << std::endl;
      std::cout << "If the option -T is used, Trees are not merged" <<std::endl;
      std::cout << "If the option -O is used, when merging TTree, the basket size is re-optimized" <<std::endl;
      std::cout << "If the option -v is used, explicitly set the verbosity level;\n"\
                   "   0 request no output, 99 is the default" <<std::endl;
      std::cout << "If the option -j is used, the execution will be parallelized in multiple processes\n" << std::endl;
      std::cout << "If the option -dbg is used, the execution will be parallelized in multiple processes in debug mode."
                   " This will not delete the partial files stored in the working directory\n"
                   "                 << std::endl;
      std::cout << "If the option -d is used, the partial multiprocess execution will be carried out in the specified "
                   "directory\n"
                   "                 << std::endl;
      std::cout << "If the option -n is used, hadd will open at most 'maxopenedfiles' at once, use 0\n"
                   "   to request to use the system maximum." << std::endl;
      std::cout << "If the option -cachesize is used, hadd will resize (or disable if 0) the\n"
                   "   prefetching cache use to speed up I/O operations." << std::endl;
      std::cout << "If the option -experimental-io-features is used (and an argument provided), then\n"
                   "   the corresponding experimental feature will be enabled for output trees." << std::endl;
      std::cout << "When -the -f option is specified, one can also specify the compression level of\n"
                   "   the target file.  By default the compression level is 4."
                   "                 << std::endl;
      std::cout << "If \"-fk\" is specified, the target file contain the baskets with the same\n"
                   "   compression as in the input files unless -O is specified.  The meta data will\n"
                   "   be compressed using the compression level specified in the first input or the\n"
                   "   compression setting specified follow fk (206 when using -fk206 for example)" <<std::endl;
      std::cout << "If \"-ff\" is specified, the compression level use is the one specified in the\n"
                   "   first input." <<std::endl;
      std::cout << "If \"-f0\" is specified, the target file will not be compressed." <<std::endl;
      std::cout << "If \"-f6\" is specified, the compression level 6 will be used.  \n"
                   "   See TFile::SetCompressionSettings for the support range of value." <<std::endl;
      std::cout << "If Target and source files have different compression settings a slower method\n"
                   "   is used.\n"<<std::endl;
      std::cout << "For options that takes a size as argument, a decimal number of bytes is expected.\n"
                   "If the number ends with a ``k'', ``m'', ``g'', etc., the number is multiplied\n"
                   "   by 1000 (1K), 1000000 (1MB), 1000000000 (1G), etc. \n"
                   "If this prefix is followed by i, the number is multipled by the traditional\n"
                   "   1024 (1KiB), 1048576 (1MiB), 1073741824 (1GiB), etc. \n"
                   "The prefix can be optionally followed by B whose casing is ignored,\n"
                   "   eg. 1k, 1K, 1Kb and 1KB are the same."<<std::endl;

      return 1;
   }
```

Fig.2

c. Using the argparse module for python code (used inside cmdLineUtils)

```
import cmdLineUtils
import sys

#Help strings
COMMAND_HELP = """Display ROOT files contents in the terminal."""

ONE_HELP = "Print content in one column"
LONG_PRINT_HELP = "use a long listing format."
TREE_PRINT_HELP = "print tree recursively and use a long listing format."

EPILOG = """Examples:
- rootls example.root
  Display contents of the ROOT file 'example.root'.

- rootls example.root:dir
  Display contents of the directory 'dir' from the ROOT file 'example.root'.

- rootls example.root:*
  Display contents of the ROOT file 'example.root' and his subdirectories.

- rootls file1.root file2.root
  Display contents of ROOT files 'file1.root' and 'file2.root'.

- rootls *.root
  Display contents of ROOT files whose name ends with '.root'.

- rootls -1 example.root
  Display contents of the ROOT file 'example.root' in one column.

- rootls -l example.root
  Display contents of the ROOT file 'example.root' and use a long listing format.

- rootls -t example.root
  Display contents of the ROOT file 'example.root', use a long listing format and print trees recursively.
"""

def execute():
    #Collect arguments with the module argparse
    parser = cmdLineUtils.getParserFile(COMMAND_HELP, EPILOG)
    parser.add_argument("-1", "--oneColumn", help=ONE_HELP, action="store_true")
    parser.add_argument("-l", "--longListing", help=LONG_PRINT_HELP, action="store_true")
    parser.add_argument("-t", "--treeListing", help=TREE_PRINT_HELP, action="store_true")

    #Put arguments in shape
    sourceList, optDict = cmdLineUtils.getSourceListOptDict(parser)

    #Process rootLs
    return cmdLineUtils.rootLs(sourceList, oneColumn=optDict["oneColumn"], \
                               longListing=optDict["longListing"], treeListing=optDict["treeListing"])

sys.exit(execute())
```

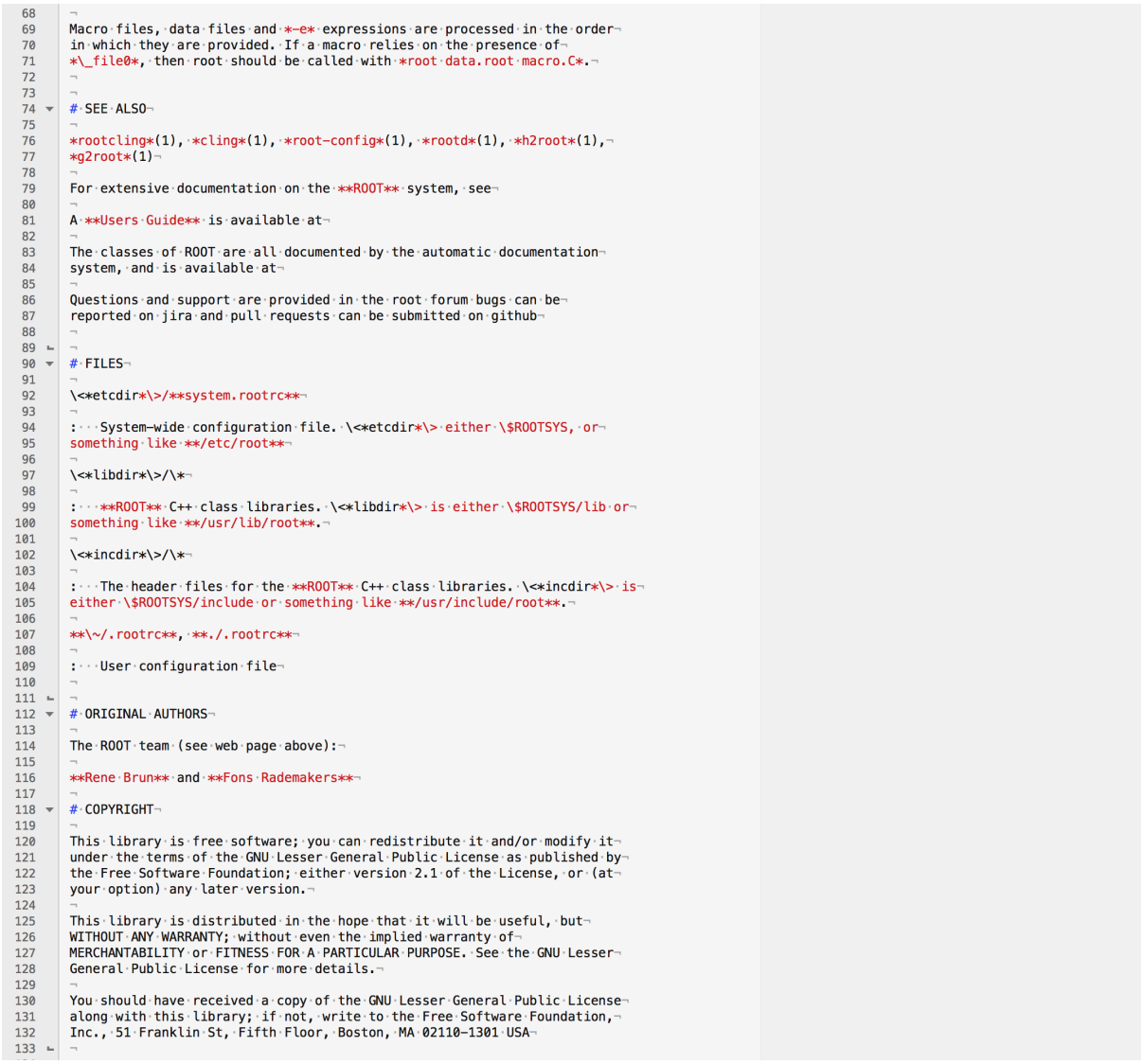
Fig.3

3. First Approach to Solving the Problem

Our first approach was to create a markdown description file for each command. Markdown is a simple markup language that would be relatively easy to use by programmers. Having a markdown for each command, we could use pandoc (a program that can convert any markup language to another) to generate the manual pages.
I drafted a proof of concept for the root command.

```
% ROOT(1)
% This  manual  page  was  written by Christian Holm Christensen <cholm@nbi.dk>, for the Debian GNU/Linux system (but may be used by others).
% Version 6

# NAME

root - Interpreter of C++ for the ROOT framework

# SYNOPSIS

\[*datafile**\...\] \[*macrofile**\...\] macrfile.C
macrofile.C(arguments*\...) macrofile.C(\"string arguments\")
macrofile.C+ macrofile.C++ macrofile.C+g macrofile.C+O

# DESCRIPTION

**R**OOTs **O**bject-**O**riented **T**echnologies.

**root** is an interactive interpreter of C++ code. It uses the **ROOT**
framework. For more information on **ROOT**, please refer to

An extensive *Users Guide* is available from that site (see below).

# OPTIONS

**-?**   Show summary of options.

**-h**   Show summary of options.

**-b**   Run in batch mode without graphics.

**-x**   Exit on exceptions.

**-e**   Execute the command passed between single quotes

**-n**   Do not execute logon and logoff macros as specified in **.rootrc**

**-t**   Enable thread-safety and implicit multi-threading (IMT)

**-q**   Exit after processing command line macro files

**-l**   Do not show splash screen

**-config**   print **./configure** options

**-memstat**   run with memory usage monitoring

**\--help**   Show summary of options.

**\--notebook**   Execute ROOT notebook.

**\--web**   Display graphics in a web browser.

**dir**   if dir is a valid directory cd to it before executing

*Data files* are opened and accessible in root as *\_file0*, *\_file1*
\...

*Macro files* are either intrepreted (filename provided alone) or
compiled (with + added to the filename). Afterwards the function with
the same name as the filename (without extension) in the macro file is
executed (eg. *void macro()* in a file *macro.C*).

Compilation is done on-demand if the macro is newer than a previously
generated shared library. Two plus signs (macro.C++) force a
recompilation. Adding *O* after a plus results in an optimised build,
adding *g* adds debug symbols (eg. *macro.C+Og*).
```

Fig.4: Markdown file for root command(1)



```
¬
Macro·files,·data·files·and·*-e*·expressions·are·processed·in·the·order¬
in·which·they·are·provided.·If·a·macro·relies·on·the·presence·of¬
*\_file0*,·then·root·should·be·called·with·*root·data.root·macro.C*.¬
¬
¬
#·SEE·ALSO¬
¬
*rootcling*(1),·*cling*(1),·*root-config*(1),·*rootd*(1),·*h2root*(1),¬
*g2root*(1)¬
¬
For·extensive·documentation·on·the·**ROOT**·system,·see¬
¬
A·**Users·Guide**·is·available·at¬
¬
The·classes·of·ROOT·are·all·documented·by·the·automatic·documentation¬
system,·and·is·available·at¬
¬
Questions·and·support·are·provided·in·the·root·forum·bugs·can·be¬
reported·on·jira·and·pull·requests·can·be·submitted·on·github¬
¬
¬
#·FILES¬
¬
\<*etcdir*\>/**system.rootrc**¬
¬
:···System-wide·configuration·file.·\<*etcdir*\>·either·\$ROOTSYS,·or¬
something·like·**/etc/root**¬
¬
\<*libdir*\>/\*¬
¬
:···**ROOT**·C++·class·libraries.·\<*libdir*\>·is·either·\$ROOTSYS/lib·or¬
something·like·**/usr/lib/root**.¬
¬
\<*incdir*\>/\*¬
¬
:···The·header·files·for·the·**ROOT**·C++·class·libraries.·\<*incdir*\>·is¬
either·\$ROOTSYS/include·or·something·like·**/usr/include/root**.¬
¬
**\~/.rootrc**,·**./.rootrc**¬
¬
:···User·configuration·file¬
¬
¬
#·ORIGINAL·AUTHORS¬
¬
The·ROOT·team·(see·web·page·above):¬
¬
**Rene·Brun**·and·**Fons·Rademakers**¬
¬
#·COPYRIGHT¬
¬
This·library·is·free·software;·you·can·redistribute·it·and/or·modify·it¬
under·the·terms·of·the·GNU·Lesser·General·Public·License·as·published·by¬
the·Free·Software·Foundation;·either·version·2.1·of·the·License,·or·(at¬
your·option)·any·later·version.¬
¬
This·library·is·distributed·in·the·hope·that·it·will·be·useful,·but¬
WITHOUT·ANY·WARRANTY;·without·even·the·implied·warranty·of¬
MERCHANTABILITY·or·FITNESS·FOR·A·PARTICULAR·PURPOSE.·See·the·GNU·Lesser¬
General·Public·License·for·more·details.¬
¬
You·should·have·received·a·copy·of·the·GNU·Lesser·General·Public·License¬
along·with·this·library;·if·not,·write·to·the·Free·Software·Foundation,¬
Inc.,·51·Franklin·St,·Fifth·Floor,·Boston,·MA·02110-1301·USA¬
¬
```

Fig.5: Markdown file for root command(2)

ROOT(1) ROOT(1)

NAME
root a Interpreter of C++ for the ROOT framework

SYNOPSIS
[datafile ...] [macrofile ...] macrfile.C macrofile.C(arguments ...) macrofile.C("string arguments") macrofile.C+ macrofile.C++ macrofile.C+g macrofile.C+O

DESCRIPTION
ROOTs **O**bject-**O**riented **T**echnologies.

root is an interactive interpreter of C++ code. It uses the **ROOT** framework. For more information on **ROOT**, please refer to

An extensive Users Guide is available from that site (see below).

OPTIONS
-? Show summary of options.

-h Show summary of options.

-b Run in batch mode without graphics.

-x Exit on exceptions.

-e Execute the command passed between single quotes

-n Do not execute logon and logoff macros as specified in **.rootrc**

-t Enable thread-safety and implicit multi-threading (IMT)

-q Exit after processing command line macro files

-l Do not show splash screen

-config print **./configure** options

-memstat run with memory usage monitoring

--help Show summary of options.

--notebook Execute ROOT notebook.

--web Display graphics in a web browser.

dir if dir is a valid directory cd to it before executing

Data files are opened and accessible in root as _file0, _file1 ...

Macro files are either intrepreted (filename provided alone) or compiled (with + added to the filename). Afterwards the function with the same name as the filename (without extension) in the macro file is executed (eg. void macro() in a file macro.C).

Compilation is done on-demand if the macro is newer than a previously generated shared library. Two plus signs (macro.C++) force a recompilation. Adding O after a plus results in an optimised build, adding g adds debug symbols (eg. macro.C+Og).

Macro files, data files and -e expressions are processed in the order in which they are provided. If a macro relies on the presence of _file0, then root should be called with root data.root macro.C.

SEE ALSO
rootcling(1), cling(1), root-config(1), rootd(1), h2root(1), g2root(1)

For extensive documentation on the **ROOT** system, see

A **Users Guide** is available at

The classes of ROOT are all documented by the automatic documentation system, and is available at

Questions and support are provided in the root forum bugs can be reported on jira and pull requests can be submitted on github

FILES
<etcdir**>/system.rootrc**
System-wide configuration file. <etcdir> either $ROOTSYS, or something like **/etc/root**

<libdir**>/***
ROOT C++ class libraries. <libdir> is either $ROOTSYS/lib or something like **/usr/lib/root.**

<incdir**>/***
The header files for the **ROOT** C++ class libraries. <incdir> is either $ROOTSYS/include or something like **/usr/include/root.**

~/.rootrc, ./.rootrc
User configuration file

ORIGINAL AUTHORS
The ROOT team (see web page above):

Rene Brun and **Fons Rademakers**

COPYRIGHT

AUTHORS
This manual page was written by Christian Holm Christensen <cholm@nbi.dk>, for the Debian GNU/Linux system (but may be used by others)..

Fig.6: Pandoc output

To extract the command line options, I used a lex program, knowing that my markdown follow a certain pattern where the options are under a #OPTIONS section and are in **bold**

```
%{
#include <iostream>
#include <string>
%}
%x OPTIONS
%%
^"# OPTIONS\n\n" BEGIN OPTIONS;
<OPTIONS>^"**" printf("  ");
<OPTIONS>"**" ;
<OPTIONS>"\n\n" printf("\n");
<OPTIONS>^. BEGIN 0;
<OPTIONS>"\\" ;
. ;
"\n" ;
%%
int yywrap()
{
return 1;
}
int main()
{yylex();}
```

Fig.7: Lex Program

And when compiled and fed the markdown file, it outputs:

```
macbookepsft:week1 epsftss$ ./a.out < ./root.md
  -?   Show summary of options.
  -h   Show summary of options.
  -b   Run in batch mode without graphics.
  -x   Exit on exceptions.
  -e   Execute the command passed between single quotes
  -n   Do not execute logon and logoff macros as specified in .rootrc
  -t   Enable thread-safety and implicit multi-threading (IMT)
  -q   Exit after processing command line macro files
  -l   Do not show splash screen
  -config   print ./configure options
  -memstat   run with memory usage monitoring
  --help   Show summary of options.
  --notebook   Execute ROOT notebook.
  --web   Display graphics in a web browser.
  dir   if dir is a valid directory cd to it before executing
```

Fig.8: Lex Output

However, lex turned out to be not cross-platform compatible so we had to change the method. Moreover, other features were deemed useful and necessary which made us change the whole approach dropping as well the idea of using markdown and pandoc.

4. The New Approach

The idea of the new approach was to make use of the functionalities and simplicity of the python argparse module. This library allows us to create an ArgumentParser object in which we add all the options with their help, types, etc. Moreover, this object has a description field in which we can add additional information to be displayed in the manual pages as well as an epilogue field. For the time being we are only making use of the description field. Consequently there would be a python file for each command containing a function returning its ArgumentParser object.

```
import argparse

def get_argparse():
    parser = argparse.ArgumentParser(add_help=False, prog='root',
    description = """ROOTs Object-Oriented Technologies.\n
root is an interactive interpreter of C++ code. It uses the ROOT  framework.  For  more information on ROOT, please refer to\n
An extensive Users Guide is available from that site (see below).
""")
    parser.add_argument('-b', help='Run in batch mode without graphics')
    parser.add_argument('-x', help='Exit on exceptions')
    parser.add_argument('-e', help='Execute the command passed between single quotes')
    parser.add_argument('-n', help='Do not execute logon and logoff macros as specified in .rootrc')
    parser.add_argument('-t', help='Enable thread-safety and implicit multi-threading (IMT)')
    parser.add_argument('-q', help='Exit after processing command line macro files')
    parser.add_argument('-l', help='Do not show splash screen')
    parser.add_argument('-config', help='print ./configure options')
    parser.add_argument('-memstat', help='run with memory usage monitoring')
    parser.add_argument('-h','-?', '--help', help='Show summary of options')
    parser.add_argument('--notebook', help='Execute ROOT notebook')
    parser.add_argument('--web', help='Display graphics in a web browser')
    parser.add_argument('dir', help='if dir is a valid directory cd to it before executing')
    return parser
```

Fig.9: root-argparse.py

In the case of the commands written in c++ like root, we decided that a header file containing the command line options should be automatically generated from the ArgumentParser and imported inside the main code of the command to be printed. As for the commands written in python, the argument parser can do the trick without any additional files

So how do we create the header and man pages?
We decided that a function that iterates through the commands stored inside the ArgumentParser should generate the man page and another should generate the header.

```
def getLongest():
    longestSize = 0
    for arg in listArgs:
        if (len(arg.option_strings)==0):
            size = len(arg.dest)
        else:
            size = len(", ".join(arg.option_strings))
        longestSize = max(longestSize, size)
    return longestSize

def write_header(parser, fileName):
    longestSize = getLongest()
    file= open(fileName, "w+")
    splitPath = sys.argv[2].split("/")
    file.write("#ifndef ROOT_{}\n".format(splitPath[len(splitPath)-1].partition(".")[0]))
    file.write("#define ROOT_{}\n".format(splitPath[len(splitPath)-1].partition(".")[0]))
    file.write("constexpr static const char kCommandLineOptionsHelp[] = R\"RAW(\n")
    file.write(parser.format_usage() + "\n")
    file.write("OPTIONS:\n")
    for arg in listArgs:
        options = ""
        help = arg.help
        if (len(arg.option_strings)==0):
            listOptions = [arg.dest]
        else:
            listOptions = arg.option_strings
        options = ", ".join(listOptions)
        spaces = " " * (12 + longestSize - len(options))
        if help != None:
            help = help.replace("\n", "\n  {}".format(" "*(len(options)) + spaces))
            file.write("  {}{}{}\n".format(options, spaces, help))
        else:
            file.write("  {}\n".format(options))
    file.write(")RAW\";\n")
    file.write("#endif\n")
    file.close()
```

Fig.10: Function generating the header file

The getLongest function returns the size of the longest series of commands corresponding for the same help, for example len("- - help, -h, -? ") is the longest in case of root (check root-argparse in fig above). This function is needed for formatting reasons, we want the distance between the options and help to be constant like in the image below. The write_header function creates the needed header file.

```
macbookepsft:bin epsftss$ root -h

usage: root [-b B] [-x X] [-e E] [-n N] [-t T] [-q Q] [-l L] [-config CONFIG]
            [-memstat MEMSTAT] [-h HELP] [--notebook NOTEBOOK] [--web WEB]
            dir

OPTIONS:
  -b                        Run in batch mode without graphics
  -x                        Exit on exceptions
  -e                        Execute the command passed between single quotes
  -n                        Do not execute logon and logoff macros as specified in .rootrc
  -t                        Enable thread-safety and implicit multi-threading (IMT)
  -q                        Exit after processing command line macro files
  -l                        Do not show splash screen
  -config                   print ./configure options
  -memstat                  run with memory usage monitoring
  -h, -?, --help            Show summary of options
  --notebook                Execute ROOT notebook
  --web                     Display graphics in a web browser
  dir                       if dir is a valid directory cd to it before executing
```

Fig.11: output of root -h

```
#ifndef ROOT_rootCommandLineOptionsHelp
#define ROOT_rootCommandLineOptionsHelp
constexpr static const char kCommandLineOptionsHelp[] = R"RAW(
usage: root [-b B] [-x X] [-e E] [-n N] [-t T] [-q Q] [-l L] [-config CONFIG]
            [-memstat MEMSTAT] [-h HELP] [--notebook NOTEBOOK] [--web WEB]
            dir

OPTIONS:
  -b                        Run in batch mode without graphics
  -x                        Exit on exceptions
  -e                        Execute the command passed between single quotes
  -n                        Do not execute logon and logoff macros as specified in .rootrc
  -t                        Enable thread-safety and implicit multi-threading (IMT)
  -q                        Exit after processing command line macro files
  -l                        Do not show splash screen
  -config                   print ./configure options
  -memstat                  run with memory usage monitoring
  -h, -?, --help            Show summary of options
  --notebook                Execute ROOT notebook
  --web                     Display graphics in a web browser
  dir                       if dir is a valid directory cd to it before executing
)RAW";
#endif
```

Fig.12: generated header rootCommandLineOptionsHelp.h

And then we would just call it in the main function of root to complete the cycle:

..... root code .....

```
47  #include "rootCommandLineOptionsHelp.h"
```

..... root code .....

```
383     if (!strcmp(argv[i], "-?") || !strncmp(argv[i], "-h", 2) ||
384         !strncmp(argv[i], "--help", 6)) {
385        fprintf(stderr, kCommandLineOptionsHelp);
386        Terminate(0);
```

..... root code .....

As for the generation of the man pages, it is done by another function operating similarly but outputting groff annotation for the man pages.

```
def write_man(parser, fileName):
    file= open(fileName, "w+")
    file.write(".TH {} 1 \n".format(parser.prog))
    file.write(".SH SYNOPSIS\n")
    file.write(parser.format_usage() + "\n")
    file.write(".SH DESCRIPTION\n")
    file.write(parser.description + "\n")
    file.write(".SH OPTIONS\n")
    for arg in listArgs:
        options = ""
        help = arg.help
        if (len(arg.option_strings)==0):
            listOptions = [arg.dest]
        else:
            listOptions = arg.option_strings
        options = "\ ".join(listOptions)
        if help != None:
            file.write(".IP {}\n".format(options))
            file.write(help.replace("\n","\n.IP\n")+"\n")
        else:
            file.write(".IP {}\n\n".format(options))
    file.close()
```

Fig.13: Function generating the man pages

```
.TH root 1
.SH SYNOPSIS
usage: root [-b B] [-x X] [-e E] [-n N] [-t T] [-q Q] [-l L] [-config CONFIG]
            [-memstat MEMSTAT] [-h HELP] [--notebook NOTEBOOK] [--web WEB]
            dir

.SH DESCRIPTION
ROOTs Object-Oriented Technologies.

root is an interactive interpreter of C++ code. It uses the ROOT  framework.  For  more information on ROOT, please refer to

An extensive Users Guide is available from that site (see below).

.SH OPTIONS
.IP -b
Run in batch mode without graphics
.IP -x
Exit on exceptions
.IP -e
Execute the command passed between single quotes
.IP -n
Do not execute logon and logoff macros as specified in .rootrc
.IP -t
Enable thread-safety and implicit multi-threading (IMT)
.IP -q
Exit after processing command line macro files
.IP -l
Do not show splash screen
.IP -config
print ./configure options
.IP -memstat
run with memory usage monitoring
.IP -h\ -?\ --help
Show summary of options
.IP --notebook
Execute ROOT notebook
.IP --web
Display graphics in a web browser
.IP dir
if dir is a valid directory cd to it before executing
```

Fig.14: generated man page root.1

```
root(1)                                                                                          root(1)

SYNOPSIS
       usage: root [-b B] [-x X] [-e E] [-n N] [-t T] [-q Q] [-l L] [-config CONFIG]
                   [-memstat MEMSTAT] [-h HELP] [--notebook NOTEBOOK] [--web WEB]
                   dir

DESCRIPTION
       ROOTs Object-Oriented Technologies.

       root is an interactive interpreter of C++ code. It uses the ROOT  framework.  For  more information on ROOT, please refer to

       An extensive Users Guide is available from that site (see below).

OPTIONS
       -b     Run in batch mode without graphics

       -x     Exit on exceptions

       -e     Execute the command passed between single quotes

       -n     Do not execute logon and logoff macros as specified in .rootrc

       -t     Enable thread-safety and implicit multi-threading (IMT)

       -q     Exit after processing command line macro files

       -l     Do not show splash screen

       -config
              print ./configure options

       -memstat
              run with memory usage monitoring

       -h -? --help
              Show summary of options

       --notebook
              Execute ROOT notebook

       --web  Display graphics in a web browser

       dir    if dir is a valid directory cd to it before executing

                                                                                                 root(1)
```

Fig.15: Display of root.1

So the design is as follow: an independent python "argparse2help.py" file contains those two functions. This python script take as input the path of the first python file containing the ArgumentParser object (for example root-argparse.py) and the output file name and path. If the output ends with a .1 extension, it calls the write_man function, otherwise, if it ends with a .h extension, the write_header function is called.

```python
if __name__ == "__main__":
    path = os.path.expanduser(sys.argv[1])
    splitPath = path.split("/")
    sys.path.insert(0, "/".join(splitPath[:len(splitPath)-1]))
    i = importlib.import_module(splitPath[len(splitPath)-1].partition(".")[0])
    parser = i.get_argparse()
    listArgs = parser._actions
    if (sys.argv[2].partition(".")[2] == "h"):
        write_header(parser, sys.argv[2])
    elif (sys.argv[2].partition(".")[2] == "1"):
        write_man(parser, sys.argv[2])
```

Fig.16: The main of argparse2help.py

In the case of the python commands (for example rootls), they already make use of a library called cmdLineUtils which in turn uses the argparse module. Since python files already have the command line options feature via the argparse module, we only needed to generate the man pages. To do so we had to split the command's code to two parts:

1. The first python file would contain a function returning the ArgumentParser object.
2. The second is the rest of the command's code that will import the part we moved.

```
#!/usr/bin/env·@python@¬
¬
#·ROOT·command·line·tools:·rootls¬
#·Author:·Julien·Ripoche¬
#·Mail:·julien.ripoche@u-psud.fr¬
#·Date:·20/08/15¬
¬
"""Command·line·to·dump·ROOT·files·contents·to·terminal"""¬
¬
import·cmdLineUtils¬
import·sys¬
¬
#·Help·strings¬
COMMAND_HELP·=·"""Display·ROOT·files·contents·in·the·terminal."""¬
¬
ONE_HELP·=·"Print·content·in·one·column"¬
LONG_PRINT_HELP·=·"use·a·long·listing·format."¬
TREE_PRINT_HELP·=·"print·tree·recursively·and·use·a·long·listing·format."¬
¬
EPILOG·=·"""Examples:¬
-·rootls·example.root¬
··Display·contents·of·the·ROOT·file·'example.root'.¬
¬
-·rootls·example.root:dir¬
··Display·contents·of·the·directory·'dir'·from·the·ROOT·file·'example.root'.¬
¬
-·rootls·example.root:*¬
··Display·contents·of·the·ROOT·file·'example.root'·and·his·subdirectories.¬
¬
-·rootls·file1.root·file2.root¬
··Display·contents·of·ROOT·files·'file1.root'·and·'file2.root'.¬
¬
-·rootls·*.root¬
··Display·contents·of·ROOT·files·whose·name·ends·with·'.root'.¬
¬
-·rootls·-1·example.root¬
··Display·contents·of·the·ROOT·file·'example.root'·in·one·column.¬
¬
-·rootls·-l·example.root¬
··Display·contents·of·the·ROOT·file·'example.root'·and·use·a·long·listing·format.¬
¬
-·rootls·-t·example.root¬
··Display·contents·of·the·ROOT·file·'example.root',·use·a·long·listing·format·and·print·trees·recursively.¬
"""¬
¬
def·execute():¬
····#·Collect·arguments·with·the·module·argparse¬
····parser·=·cmdLineUtils.getParserFile(COMMAND_HELP,·EPILOG)¬
····parser.add_argument("-1",·"--oneColumn",·help=ONE_HELP,·action="store_true")¬
····parser.add_argument("-l",·"--longListing",·help=LONG_PRINT_HELP,·action="store_true")¬
····parser.add_argument("-t",·"--treeListing",·help=TREE_PRINT_HELP,·action="store_true")¬
¬
····#·Put·arguments·in·shape¬
····sourceList,·optDict·=·cmdLineUtils.getSourceListOptDict(parser)¬
¬
····#·Process·rootLs¬
····return·cmdLineUtils.rootLs(sourceList,·oneColumn=optDict["oneColumn"],·\¬
·····························longListing=optDict["longListing"],·treeListing=optDict["treeListing"])¬
¬
sys.exit(execute())¬
```

Fig.17: rootls.py old code

```
#!/usr/bin/env·@python@¬
¬
#·ROOT·command·line·tools:·rootls¬
#·Author:·Julien·Ripoche¬
#·Mail:·julien.ripoche@u-psud.fr¬
#·Date:·20/08/15¬
¬
"""Command·line·to·dump·ROOT·files·contents·to·terminal"""¬
¬
import·cmdLineUtils¬
import·sys¬
import·importlib¬
¬
¬
def·execute():¬
¬
Δ···i·=·importlib.import_module("rootls-argparse")¬
Δ···parser·=·i.get_argparse()¬
Δ···#·Put·arguments·in·shape¬
Δ···sourceList,·optDict·=·cmdLineUtils.getSourceListOptDict(parser)¬
¬
Δ···#·Process·rootLs¬
Δ···return·cmdLineUtils.rootLs(sourceList,·oneColumn=optDict["oneColumn"],·\¬
Δ···Δ···Δ···Δ···Δ···Δ···Δ···longListing=optDict["longListing"],·treeListing=optDict["treeListing"])¬
¬
sys.exit(execute())¬
```

Fig.18: New rootls.py

```python
import argparse
import cmdLineUtils

#Help strings
description = "Display ROOT files contents in the terminal."

ONE_HELP = "Print content in one column"
LONG_PRINT_HELP = "Use a long listing format."
TREE_PRINT_HELP = "Print tree recursively and use a long listing format."

EPILOG = """Examples:
- rootls example.root
  Display contents of the ROOT file 'example.root'.

- rootls example.root:dir
  Display contents of the directory 'dir' from the ROOT file 'example.root'.

- rootls example.root:*
  Display contents of the ROOT file 'example.root' and his subdirectories.

- rootls file1.root file2.root
  Display contents of ROOT files 'file1.root' and 'file2.root'.

- rootls *.root
  Display contents of ROOT files whose name ends with '.root'.

- rootls -1 example.root
  Display contents of the ROOT file 'example.root' in one column.

- rootls -l example.root
  Display contents of the ROOT file 'example.root' and use a long listing format.

- rootls -t example.root
  Display contents of the ROOT file 'example.root', use a long listing format and print trees recursively.
"""

def get_argparse():
    parser = cmdLineUtils.getParserFile(description, EPILOG)
    parser.prog = 'rootls'

    parser.add_argument("-1", "--oneColumn", help=ONE_HELP, action="store_true")
    parser.add_argument("-l", "--longListing", help=LONG_PRINT_HELP, action="store_true")
    parser.add_argument("-t", "--treeListing", help=TREE_PRINT_HELP, action="store_true")
    return parser
```

Fig.19: rootls-argparse.py

Example to generate the man page of root command

The generation for all commands (c++ and python) is done by calling the main function as follow:

python /pathToFile/argparse2help.py /pathToFile/root-argparse.py /pathToOutput/root.1

That is simple!!

I have applied those coding formats to the following commands:

_ root
_ hadd
_ rootcling
_ hist2workspace
_ rootls
_ rootbrowse
_ rooteventselector
_ rootmv
_ rootprint
_ rootrm
_ rootslimtree
_ rootmkdir
_ rootcp
_ rootdrawtree

Now all that is left is building them while building ROOT!

5. Building the System

Now we had to integrate the process in the building of root so we modified the CMake files to add the needed dependencies and commands to build the headers and manual pages in the correct places.

I would like to thank my supervisors Olivier Couet and Axel Naumann, as well as Guilherme Amadio for their help during this project.