ROOT Trips & Tricks

Ole Hansen

Jefferson Lab

Hall A & C Analysis Workshop
June 26–27, 2017

Brief Introduction

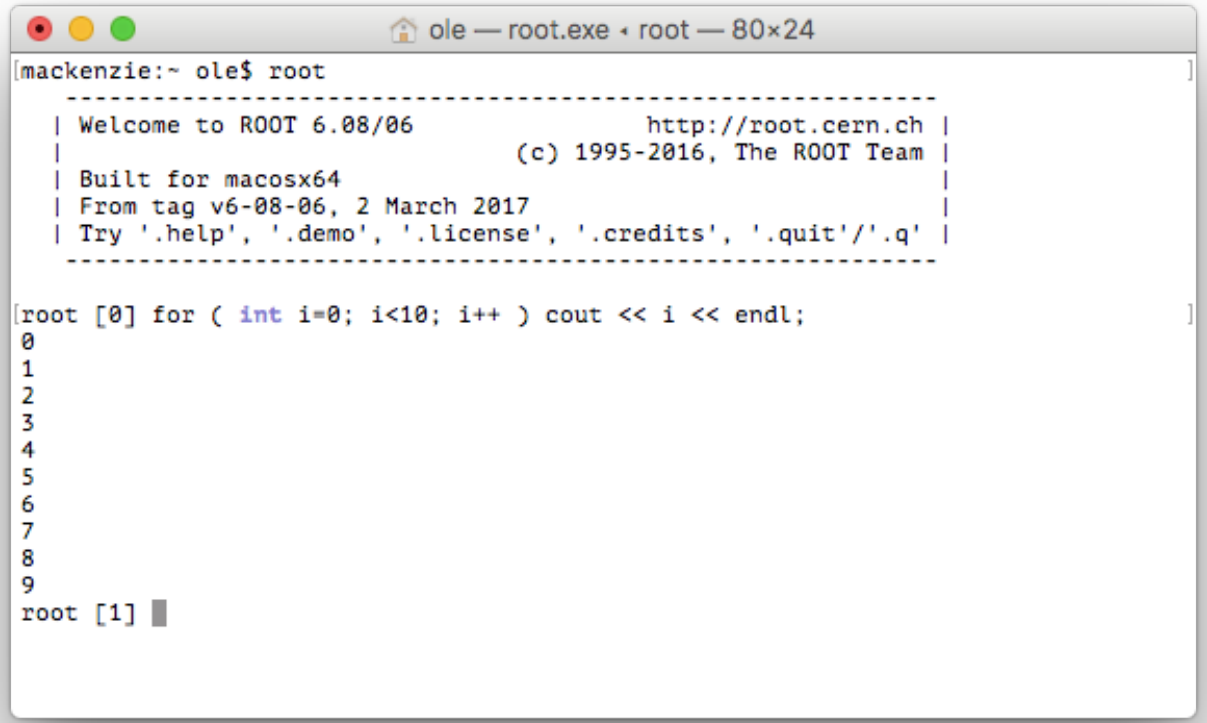
ROOT in a Nutshell

- Framework for large scale data handling
- Provides
  - efficient data storage, access and query (petabytes)
  - advances statistical analysis: histogramming, fitting, minimization, etc.
  - scientific visualization: 2D and 3D graphics
  - geometry toolkit
  - event display
  - parallel query engine (PROOF)
  - interactive C++11 interpreter
- Open Source. GNU LGPL. ~2M lines of code.
- Supported by ~10 staff at CERN and Fermilab
- Tens of thousands of users in high-energy and nuclear physics, and elsewhere (*e.g.* finance)

This talk covers the current version, ROOT 6

The Interpreter

- C++11 interpreter: `Cling` (based on LLVM/Clang)
- Provides interactive shell
- Allows rapid prototyping and testing
- Just-in-time and on-the-fly compilation
- Same language for scripting and compiled code
- Python bindings available



C++ Object I/O and Persistency

- ROOT lets you write C++ objects to file
  - Extraordinary: impossible in native C++!
  - Used for storage of hundreds of petabytes at LHC
  - Achieved with serialization of objects based on C++ reflection made possible by class dictionaries generated with Cling
- Can write single objects, collections (containers), entire object trees
  - One method for all ROOT objects: `TObject::Write`
- Evolution from flat n-tuples used in low and medium-energy physics
- Cornerstone for the storage of experimental data in HEP

Large Collection of Additional Libraries

- **Graphics & Visualization**
  - 1D, 2D and 3D histograms
  - Very large collection of data plot types
  - Can save graphics in many formats (e.g. PDF, JPEG, PNG)
- **Geometry Toolkit and Event Display**
  - Detailed description of detector geometry and materials
  - Used extensively in simulations and event reconstruction
  - 3D visualization of geometry, detector hits and tracks
- **Mathematical Functions**
  - TMath: commonly used math functions, constants, statistics
  - ROOT::Math: very large collection of special functions, interface to Gnu Scientific Library (GSL)
- **Minimization**
  - Extensive library of fitting and minimization routines
  - Widely used in data analysis and significance testing

Using ROOT

Before We Get Started

Please update the "tutorial" subdirectory in your virtual machine

```
[me@centos7 ~]$ cd ~/tutorial
[me@centos7 tutorial]$ git pull
```

Exercise 1: ROOT Command Line Basics

Starting up, version check

```
[me@centos7 ~]$ cd ~/tutorial/ROOTtutorial
[me@centos7 ~]$ root
root [0] gROOT->GetVersion()
(const char *) "6.08/06"
```

Defining variables

```
root [1] i = 10
(int) 10
root [2] x = 3.14
(double) 3.14000
root [3] e = 2.718 ;
root [4] e
(double) 2.71800
```

- May omit trailing ";" → return value printed
- May omit type specification → treated like "auto"

Exercise 2: ROOT Command Line Arithmetic

ROOT as a (hyper-charged) pocket calculator

```
root [0] 5+7*(3-2)
(int) 12
root [1] sqrt(2)
(std::enable_if<true, double>::type) 1.41421
root [2] double angle = 45.;
root [3] sin(angle * TMath::DegToRad())
(double) 0.707107
root [4] TMath::Erf(1.)
(Double_t) 0.842701
root [15] cout << setprecision(16) << TMath::Pi() << endl;
3.141592653589793
```

Exercise 3: C++11 in ROOT

C++11 in ROOT: a few simple examples

```
// Initialize a std::vector: initializer list
analyzer [1] vector<double> dvars {3.45, 1.5, 9.91, 6.28, -2.718}
(std::vector<double> &) { 3.45000, 1.50000, 9.91000, 6.28000, -2.71800 }

// Much simpler looping over containers (vectors etc.)
analyzer [2] for( auto x : dvars ) cout << x << ", "; cout << endl;
3.45, 1.5, 9.91, 6.28, -2.718,

// This is old, but still works, now even in ROOT: std::sort
analyzer [3] std::sort(dvars.begin(), dvars.end());
analyzer [4] dvars
(std::vector<double> &) { -2.71800, 1.50000, 3.45000, 6.28000, 9.91000 }

// Define functions on the fly: lambda expressions
analyzer [5] std::sort(dvars.begin(), dvars.end(),
  [](double a, double b) {return b<a;} );

analyzer [6] for( auto x : dvars ) cout << x << ", "; cout << endl;
9.91, 6.28, 3.45, 1.5, -2.718,
```

Exercise 4: Macros/Scripts

add42.C

```
int add42(int value) {
  return value+42;
}
```

Running and Loading a Macro/Script

```
// Run plain script
root [1] .x add42.C(11)
(int) 53

// Load, then execute
// In this way, you can use multiple functions defined in the script
root [2] .L add42.C
root [3] add42(11)
(int) 53
```

Exercise 4 (cont.): Compiling Macros/Scripts

Compiling a macro/script

```
// Compile on the fly -- much faster for bigger scripts
root [0] .L add42.C+
root [1] add42(11)
(int) 53

// Reuse compiled script -- rebuilt only if changed
// Execute directly
root [0] .x add42.C+(11)
(int) 53

// Load, then execute
root [0] .L add42.C+
root [1] add42(11)
(int) 53

// Force recompilation
root [2] .L add42.C++
```

Exercise 5: Other Useful Commands

Control commands

```
// Quit ROOT ;)
root [0] .q

// Get help on commands
root [0] .?
root [1] .help

// Run shell command
root [2] .!pwd
root [3] .!cat add42.C

// Redirect output to a file
root [5] .> output.txt   // overwrites
root [6] cout << TMath::C() << endl;
root [7] .>              // stops redirection
root [8] cout << 123 << endl;
123
root [9] .!cat output.txt
2.99792e+08
root [10] .>> output.txt  // appends if file exists
root [11] (your pick)
```

Histograms

Exercise 6: Creating One-Dimensional Histograms

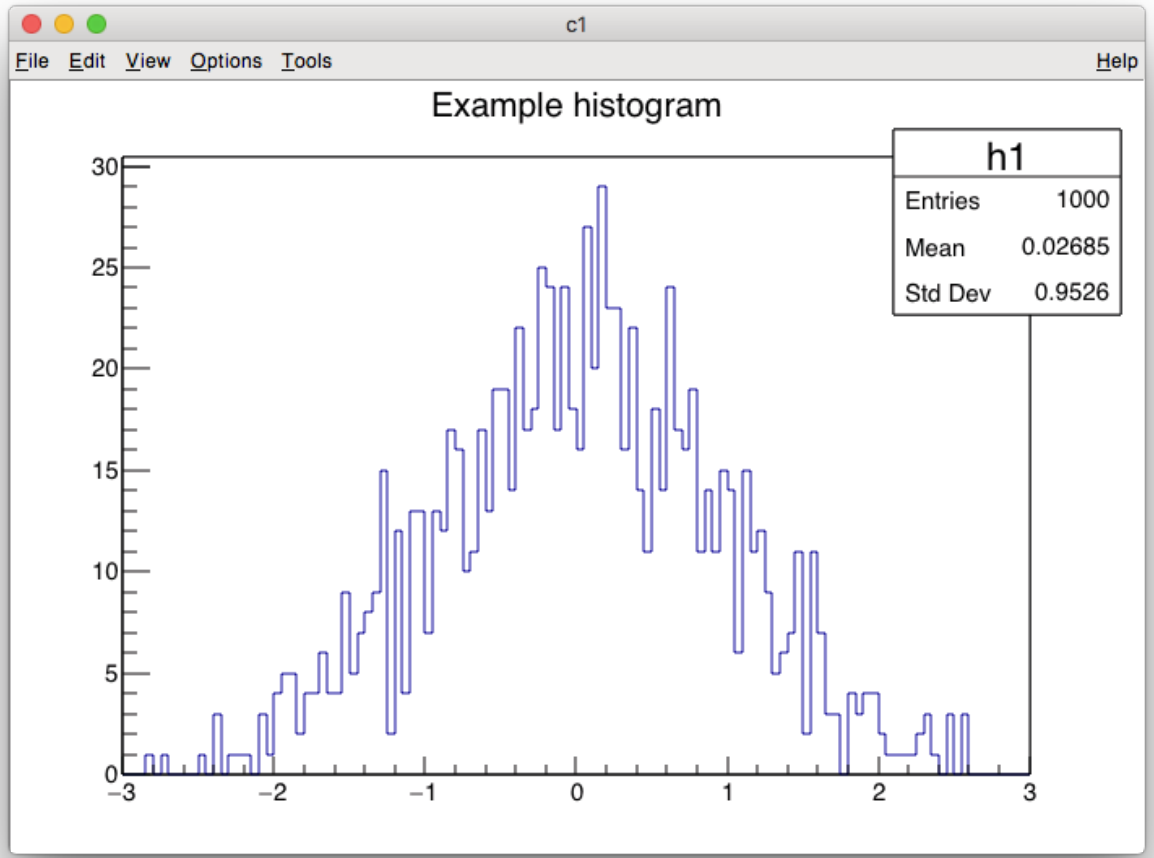
Creating & Filling Histograms

```
// Creating
//                            name  description       #bins min,max
root [0] TH1* h1 = new TH1I("h1","Example histogram",120,-3.,3.);

// Filling
   h1->Fill(x);

// Filling with 1000 random values
root [2] for ( int i=0; i<1000; ++i ) {
root (cont'ed, cancel with .@) [3]double x = gRandom->Gaus(0,1);
root (cont'ed, cancel with .@) [4]h1->Fill(x);
root (cont'ed, cancel with .@) [5]}

root [6] h1->GetEntries()
(Double_t) 1000.00
```

Exercise 7: Plotting Histograms

Drawing any histograms

```
root [7] h1->Draw();
```



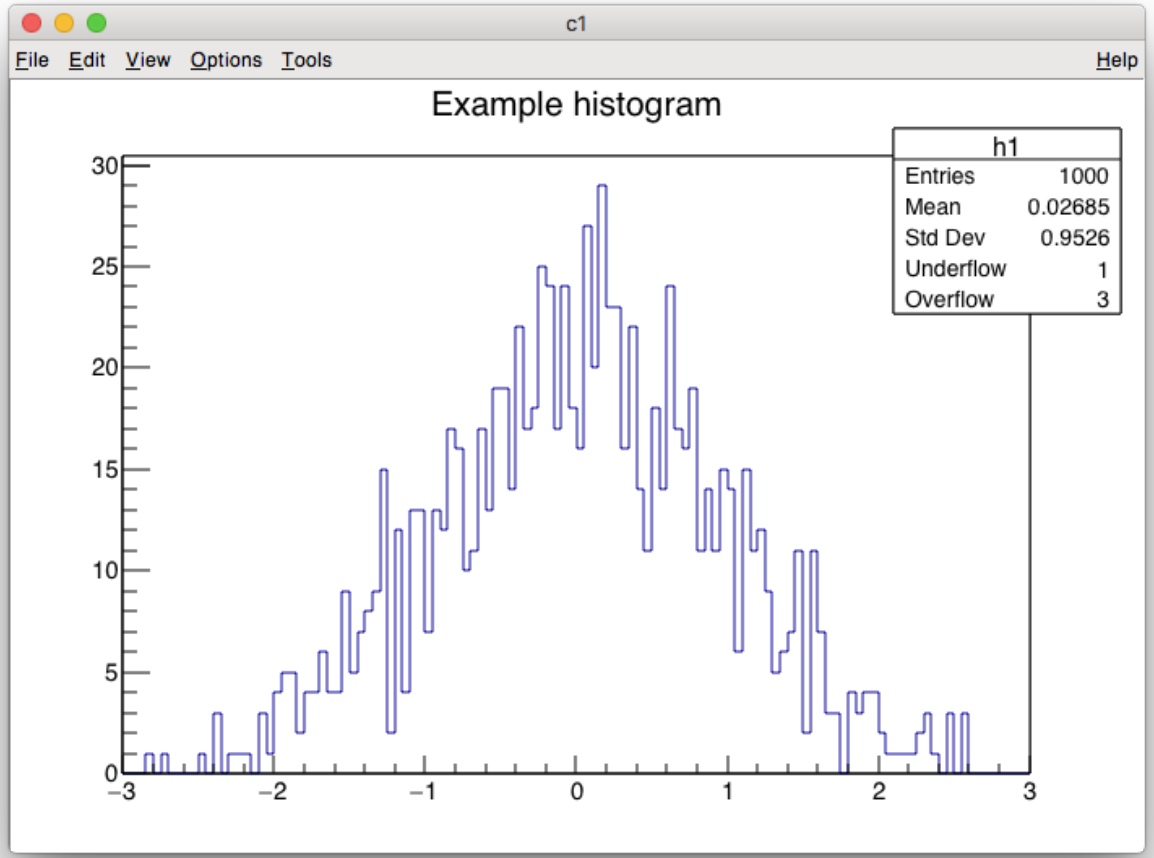
The box shows

- Name
- Number of entries
- Mean value
- Std. deviation (RMS)

Exercise 8: Plotting Histograms (cont.)

Always show over/underflow!

```
root [8] gStyle->SetOptStat(111111);
```



Note how the statistics box automatically changes

Exercise 9: Histogram Drawing Options

- Draw error bars on every bin

```
root [9] h1->Draw("E");
```

- Draw another histogram without replacing current plot. Can be used to plot one histgram on top of another

```
// Restart ROOT, then
root [0] .x make_histos.C
root [1] h2->SetLineColor(kRed);
root [2] h2->Draw("SAME");
```

- Display one axis with logarithmic scale

```
root [3] gPad->SetLogy();       // y-axis log scale on
root [4] gPad->SetLogy(false);  // off again
```

More About Histograms

- `TH1` is the base class for *all* histogram classes (even 2D and 3D . . .)
- `TH1`, `TH2` and `TH3` are generic. Must use specialized ones (e.g. `TH1D`) for constructing.
  - TH1C: 1 byte (char) per channel, maximum bin content = 127
  - TH1S: 2 bytes (short), max. content = 32767
  - TH1I: 4 bytes (int), max. content = 2147483647
  - TH1F: 4 bytes (float), max. precision 7 digits
  - TH1D: 8 bytes (double), max. precision 14 digits
- `TH1(2,3)F` and `TH1(2,3)D` are most common, unless you either need integer precision (`TH1(2,3)I`) or need to save memory (`TH1(2,3)(C,S)`).

Exercise 10: 2D Histograms

Creating and Filling a 2D histogram

```
// Create and fill a 2D histogram

root [0] .!cat make_2D.C
#include "TH2F.h"
#include "TRandom.h"

TH2* h2D;

void make_2D()
{
  h2D = new TH2F("h2D","2D Gaussian peak",160,-4.,4.,160,-4.,4.);
  for( int i=0; i<100000; ++i ) {
    double x = gRandom->Gaus(0,1);
    double y = gRandom->Gaus(1,2);
    h2D->Fill(x,y);
  }
}

root [1] .x make_2D.C
```

Exercise 11: Drawing 2D Histograms

Drawing 2D histograms

```
// Scatter plot
root [2] h2D->Draw();

// Color Z
root [3] h2D->Draw("COLZ");

// Contour plot
root [3] h2D->Draw("CONTZ");

// "Lego" plot (3D)
root [4] h2D->Draw("LEGO");
```

Exercise 12: Projections and Profile Histograms

Projections and Profile Histograms

```
//  Projection of 2D histogram onto one of its axes
// This sums all y-bins with the same x value
root [5] TH1* hX = h2D->ProjectionX();
root [6] hX->Draw();

//  Profile histogram: Plot the mean and standard deviation
// of all y-bins with the same x value
root [7] .! cat make_prof.C
#include "TProfile.h"
#include "TRandom.h"
TProfile* hp;
void make_prof() {
  hp = new TProfile("hp","Profile of shifted Gaussian",160,-4.,4.);
  for( int i=0; i<100000; ++i ) {
    double x = gRandom->Gaus(0,1);
    double y = gRandom->Gaus(1,2);
    hp->Fill(x,y);
  }
}
root [8] hp->Draw();
```

Exercise 13: Fitting Histograms

Fitting Histograms

```
// Basic
root [9] h1->Fit("gaus");
root [10] gStyle->SetOptFit(1111);

// More complex (identical to above, but flexible now)
root [11] TF1* f1 = new TF1("f1","[0]*TMath::Gaus(x,[1],[2])");
root [12] f1->SetParameters(1,0,1);
root [13] h1->Fit(f1);

// Retrieving fit results
root [14] TF1* ff = h1->GetFunction("f1");
root [15] ff->GetParameter(0)
root [16] ff->GetParError(0)

// More detailed information about the fit
root [17] r = h1->Fit(f1,"S");

// Can now retrieve full fit information from r
root [18] TMatrixDSym C = r->GetCorrelationMatrix();  // operator -> !
root [19] C.Print();
```

Exercise 14: More Fitting Options

More Fitting Options

```
// Restrict fit range
root [20] h1->Fit("gaus","","",-1.5,1.5);

// Fit a second function to a different part of the histogram
// with the "+" option
root [21] h1->Fit("expo","+","",2,4);

// Verbose - more detail than you ever want to know
root [22] h1->Fit("gaus","V");

// Likelihood fit
root [23] h1->Fit("gaus","L");

// Pass histogram plotting options as 3rd parameter
root [24] c1->Clear();
root [25] h1->Fit("gaus","L","E");
```

Files and Trees

Exercise 15: Opening an browsing a File

ROOT files and the browser

```
// Quit ROOT and start the analyzer so we have all class info
[me@centos7 ~]$ analyzer

// Open a ROOT file
analyzer [0] TFile* file = new TFile("/data/ROOTfiles/gmp_23062_good.root",
   "READ")

// Listing the file contents. "KEY": on disk, "OBJ": in memory
analyzer [1] .ls
TFile** /data/ROOTfiles/gmp_23062_good.root
 TFile* /data/ROOTfiles/gmp_23062_good.root
  KEY: THaRun Run_Data;2 gmp run 23062
  KEY: TTree T;1 Hall A Analyzer Output DST

// Much more user-friendly: the ROOT browser
analyzer [2] new TBrowser;
```

Exercise 16: Plotting Tree Variables

A very powerful method to analyze our trees is the `TTree::Draw` method

ROOT files and the browser

```
// Basic histogram
analyzer [3] T->Draw("R.vdc.u1.wire");

// Applying a selection
analyzer [3] T->Draw("R.vdc.u1.wire","R.vdc.u1.wire>100");

// Avoidng automatic binning: defining histograms on the fly
analyzer [4] T->Draw("R.vdc.u1.wire>>hw(368,0,368)","R.vdc.u1.wire>100");

// 2D histograms: Inspecting correlations
analyzer [5] T->Draw("R.vdc.u1.wire:R.vdc.u2.wire");

// Using histogram drawing options (see earlier)
analyzer [6] T->Draw("R.vdc.u1.wire:R.vdc.u2.wire","","CONTZ");

// CAUTION: u1.wire and u2.wire are NOT parallel arrays!
analyzer [7] T->Draw("Ndata.R.vdc.u1.wire:Ndata.R.vdc.u2.wire","","LEGO");
analyzer [8] gPad->SetLogz()
// The 2D plot shows only elements with indices i=j, which may or
// may not be meaningful
```

Thanks!