A brief Introduction to **ROOT**

- An Object-Oriented Data Analysis Framework

Luca Perrozzi - HPK E23 - luca.perrozzi@phys.ethz.ch
Mario Masciovecchio - HPK E32 - mario.masciovecchio@cern.ch

Many thanks to Philipp Eller for his help with these slides
Outline

• What is ROOT?
• A basic introduction to Bash, C++ and Python
• Interactive ROOT
• Basics:
  – a first Example
  – objects, pointers & references, memory
  – Reading/writing data with ROOT
  – Ntuples, 1D/2D Histograms
  – Style, options, legend, canvas
  – Functions & Fitting
• Editors
• Exercises
What is ROOT?

• ROOT is an object oriented framework for data analysis
  – Read data from different sources
  – Write data (persistent object)
  – Select data with some criteria
  – Produce results as plots, fits, etc…

• Supports “interactive” (C/C++, Python) as well as “compiled” usage (C++)

• ROOT integrates several tools:
  – Random number generators
  – Fit methods (Minuit)
  – Neural Network framework (TMVA)

• Developed and supported by HEP community
What is ROOT for?

Luca Perrozzi

Introduction to ROOT
General Information

Official website (also for binaries and sources):
http://root.cern.ch/

BaBar ROOT tutorials:

Our website:
http://ihp-lx.ethz.ch/Stamet/
Basic UNIX commands

bash

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ ls</code></td>
<td>List files in a directory</td>
</tr>
<tr>
<td><code>$ ls -ltr</code></td>
<td>List files in a directory in a nice way</td>
</tr>
<tr>
<td><code>$ mkdir myDir</code></td>
<td>create a directory “myDir”</td>
</tr>
<tr>
<td><code>$ cd myDir/mySubDir</code></td>
<td>change current directory</td>
</tr>
<tr>
<td><code>$ mv a.txt b.txt</code></td>
<td>rename a file from a.txt to b.txt</td>
</tr>
<tr>
<td><code>$ rm b.txt</code></td>
<td>remove b.txt</td>
</tr>
<tr>
<td><code>$ rmdir myDir</code></td>
<td>remove empty directory “myDir”</td>
</tr>
<tr>
<td><code>$ rm -rf myDir</code></td>
<td>remove not-empty directory “myDir”</td>
</tr>
<tr>
<td><code>$ cat b.txt or less b.txt</code></td>
<td>print the content of b.txt</td>
</tr>
<tr>
<td><code>$ pwd</code></td>
<td>print your current directory path</td>
</tr>
</tbody>
</table>

Wildcard

vim, emacs

emacs, kate, gedit, etc...

‘$’ just denotes the shell prompt, don’t type it.

List files in a directory
List files in a directory in a nice way
create a directory “myDir”
change current directory
rename a file from a.txt to b.txt
remove b.txt
remove empty directory “myDir”
remove not-empty directory “myDir”
print the content of b.txt
print your current directory path

Editors (as manu as you want):
(console editors)
(with graphics)
basic Python knowledge

control structures:
if condition:
    something
elif condition:
    something else
else:
    other stuff
while condition:
    do this

catching exceptions: try except else finally

functions:
def a(*args,**kwargs):
    pass

classes:
class b(object):
    pass

some types:
bool, int, float, double, string...

comments:
# bla
or
""
bla
more bla
""

blocks: must be indented (4 white spaces is recommended)

some links:
official website:
http://www.python.org/
tutorials:
http://docs.python.org/2/tutorial/
http://learnpythonthehardway.org/book/
basic C++ knowledge

control structures:
if (condition) {}
else {}
while (condition) {}
switch(var) {
case 0: {}
case 1: {}
...
}

functions:
void f(args) {}

classes:
class myClass{
public:
    myClass();
    ~myClass();
private:
}

some types:
bool, int, float, double, void, char, string, ...

comments:
// bla
or
*/
bla
more bla
*/

blocks are in curly brackets {}
lines end on semicolon ;

some links:
http://www.cplusplus.com/doc/tutorial/
http://www.cprogramming.com/tutorial/lesson1.html
**interactive root**

**root**

start root (CINT):
$ root

quit:
root [0] .q

launch and open a root file:
$ root file.root

suppress the startup screen:
$ root –l

load code from external file:
root [0] .L fileName.C

execute a Script:
root [0] .x script.C

use the GUI browser:
root [0] TBrowser x

---

CINT is the C/C++ interpreter of ROOT.
Aclic is the C/C++ compiler invoked by ROOT when you ask ROOT to compile something.

Welcome and quit:
$ root .q

Launch and open a root file:
$ root file.root

Suppress the startup screen:
$ root –l

Load code from external file:
root [0] .L fileName.C

Execute a Script:
root [0] .x script.C

Use the GUI browser:
root [0] TBrowser x

---

**Introduction to ROOT**

Luca Perrozzi
TBROWSER, TREEVIEWER

- You can open a new TBrowser in an interactive ROOT session.
- Can be useful to interactively browse the content of root files, available histograms, etc.
- For the tree, you can use the tree viewer. Right-click on the tree and StartViewer.

- Draw complex expressions
- Impose your cuts
- Draw multidimensional histograms putting different variables in different axes
## A Basic ROOT Example

Reminder: indentation is optional in C++, but mandatory in Python!

### C++

```cpp
#include <TH1.h>
#include <TFile.h>
#include <TRandom.h>

int main() {
    TH1F *histo = new TH1F("hgaus", "A Gauss Function", 100, -5.0, 5.0);
    TRandom rnd;
    for (int i = 0; i < 1000; ++i) {
        double x = rnd.Gaus(1.5, 1.0);
        histo->Fill(x);
    }
    TFile outfile ("gaus.root", "RECREATE");
    histo->Write();
    outfile.Close();
    return 0;
}
```

### Python

```python
from ROOT import TH1F,TFile,TRandom

histo = TH1F("hgaus", "A Gauss Function", 100, -5.0, 5.0)
rnd = TRandom()
for i in range(1000):
    x = rnd.Gaus(1.5, 1.0)
    histo.Fill(x)

outfile = TFile("gaus.root", "RECREATE")
histo.Write()
outfile.Close()
```
A Basic ROOT Example

C++
#include <TH1.h>
#include <TFile.h>
#include <TRandom.h>

int main() {
    TH1F *histo = new TH1F("hgaus", "A Gauss Function", 100, -5.0, 5.0);
    TRandom rnd;
    for (int i = 0; i < 1000; ++i) {
        double x = rnd.Gaus(1.5, 1.0);
        histo->Fill(x);
    }
    TFile outfile ("gaus.root", "RECREATE");
    histo->Write();
    outfile.Close();
    return 0;
}

Python
from ROOT import TH1F, TFile, TRandom

histo = TH1F("hgaus", "A Gauss Function", 100, -5.0, 5.0)
rnd = TRandom()
for i in range(1000):
    x = rnd.Gaus(1.5, 1.0)
    histo.Fill(x)
outfile = TFile("gaus.root", "RECREATE")
histo.Write()
outfile.Close()
### Pointers & References

#### C++

```cpp
int a = 5;
int c = a; // calls copy constructor
int * b = &a; // gives b the address of a
int & r = a; // r is a reference to a
const & d = a; // c is a const ref to a

// Example
cout << a;
5
cout << &a;
0x7fdc049bd3d0
cout << b;
0x7fdc049bd3d0
cout << *b;
5
cout << c;
5
cout << &c;
0x7fdc03c31be0
```

- the operator * returns the value pointed to
- the operator & returns the pointer of a variable

#### Python

```python
# python is taking care of it for you!

# In case you need a hard copy:

# from copy import copy, deepcopy
b = copy(a)
# or for nested instances:
b = deepcopy(a)
```

Luca Perrozzi

Introduction to ROOT

12
Objects

### C++

// create Objects:
int a = 7;

// deleted automatically when out of scope

// create new Objects:
Object * myobj = new Object;
...
// do something with myobj
...
delete myobj;

// access members with `.` of an instance
// or with `->` for pointers

### Python

# create Objects:
a = 7

# or for example:
histo = ROOT.TH1F(*args)

# access members always with `.`

# join as many `.` as you wish
Root class index

- Fundamental resource to define and use ROOT objects

  https://root.cern.ch/root/html/ClassIndex.html

Luca Perrozzi

Introduction to ROOT
Memory management

- **ROOT objects** (Histograms, Canvas, etc) are managed in memory (and disk) by root using “names”
- In the same directory you cannot have two object with the same name (ROOT will complain about memory leaks)
  - ROOT does not like the following:

```cpp
TH1F * histos[10];
for(int i = 0; i < 10 ; i++) histos[i]= new TH1F("hist","hist",1,2,3);
```

same name! BE CAREFUL
Reading data

• ROOT can read data from different sources such as files, network, databases

• In ROOT framework the data is usually stored in TNuple (or TTree)
  – Trees/Ntuples are like “tables”
  – Each raw represent usually one “event”
  – Each column is a given quantity (energy, mass, angle, etc… )

• TFile
  – Trees and Histograms can be read from “ROOT files” in which they are stored
  – TFiles contain directories
  – can be saved by the user
# A few file commands:

# Open a file:
```cpp
TFile f("myfile.root");
```

# Inspect (list) contents with:
```cpp
f->ls();
```

# Change into a directory of the file with:
```cpp
f->cd("mydirectory");
```

# List the histograms in that directory:
```cpp
gDirectory->ls();
```

# Plot a histogram in that directory:
```cpp
histo->Draw();
```

# A few file commands:

# Open a file:
```python
f = ROOT.TFile("myfile.root");
```

# Inspect (list) contents with:
```python
f.ls()
```

# Change into a directory of the file with:
```python
f.cd("mydirectory")
```

# List the histograms in that directory:
```python
gDirectory.ls()
```

# Plot a histogram in that directory:
```python
histo.Draw()
```
reading data from ASCII File

C++

TNtuple calls("calls",
    "calls","cost:time:type")
calls.ReadFile("calls.txt")
calls->Scan()

Python

calls = ROOT.TNtuple("calls",
    "calls","cost:time:type")
calls.ReadFile("calls.txt")
calls.Scan()

calls.txt:

<table>
<thead>
<tr>
<th>#cost</th>
<th>time</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.46</td>
<td>127</td>
<td>2</td>
</tr>
<tr>
<td>2.25</td>
<td>124</td>
<td>11</td>
</tr>
<tr>
<td>0.82</td>
<td>71</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Saving/reading ROOT files

• We can save TNuple in a file

```c
root[2] TFile f("rootfile.root","CREATE")
root[3] f.cd()
root[4] calls.Write()
```

• And read it back form a new ROOT console

```c
root[0] TFile f("rootfile.root")
root[1] TNtuple * calls = f.Get("calls")
```

• When you read back, the pointer to the NTuple is owned by root, you should not delete it
• The “Get” method identify the objects with their name
• You can list the names and the types of the objects in a ROOT file
## TTree

### C++

```cpp
// list all variables in the tree
mytree->Print();

// Draw one variable:
mytree->Draw("track momentum");

// Draw two variables as scatter plot:
mytree->Draw("px:py");

// scatter plot with cut a cut on pz:
mytree->Draw("px:py","pz>5");

// Print out values with cut
mytree->Scan("px:py","pz>5");
```

### Python

```python
// list all variables in the tree
mytree.Print()

// Draw one variable:
mytree.Draw("track momentum")

// Draw two variables as scatter plot:
mytree.Draw("px:py")

// scatter plot with cut a cut on pz:
mytree.Draw("px:py","pz>5")

// Print out values with cut
mytree.Scan("px:py","pz>5")
```
## TH1F

### C++

```cpp
// Make a first histogram:
TH1F *histo = new TH1F("h_name", "h_title", nbins, xlow, xhi);

// Now draw the (currently empty) histo:
histo->Draw();

// Fill with a few entries:
histo->Fill(1.);
histo->Fill(3,10);

// Update the drawing:
histo->Draw();
```

### Python

```python
// Make a first histogram:
histo = ROOT.TH1F("h_name", "h_title", nbins, xlow, xhi)

// Now draw the (currently empty) histo:
histo.Draw()

// Fill with a few entries:
histo.Fill(1.)
histo.Fill(3,10)

// Update the drawing:
histo.Draw()
```

---

**Dimensions**
- TH1F, TH1D, TH2F, TH3D

**Float, Double**
// Change the line colour:
  h1->SetLineColor(kRed);
// Histogram Title
  h1->SetTitle("My title");
// X axis title:
  h1->SetXTitle("The x axis");
// Change x-axis range:
  SetAxisRange(4., 15);  // zoom
// Line colours:
  SetMarkerColor(kBlue);  // etc
// Point size:
  SetMarkerSize(1.);
// Point style:
  SetMarkerStyle(20);
// Fill colour:
  SetFillColor(kGreen);
// Draw a filled histo:
  SetFillStyle(3004);  // diagonal lines
// Histo with error bars:
  h1->Draw("e");  // error = sqrt[nentries]
// Usually need to redraw histo after any changes:
  h1->Draw();
// Second histo on same plot:
  h2->Draw("same");
Style

- **TCanvas**
- **TPad**

![Graph of Higgs boson mass (GeV) vs. Local p-value](image)

- **TLegend**
  - Combined obs.
  - Exp. for SM H
  - $H \rightarrow bb$
  - $H \rightarrow \tau \tau$
  - $H \rightarrow \gamma \gamma$
  - $H \rightarrow WW$
  - $H \rightarrow ZZ$

- **TAxis**

$\sqrt{s} = 7$ TeV, $L = 5.1$ fb$^{-1}$
$\sqrt{s} = 8$ TeV, $L = 5.3$ fb$^{-1}$
more plotting options

// Axis labeling
histo.GetXaxis().SetTitle("#sqrt{s}\)"

// Legends
leg = ROOT.TLegend(0.1,0.5,0.8,0.9)
leg.AddEntry(histo1,"description of histo1")
leg.AddEntry(histo2,"description of histo1")
leg.Draw("FLP")

Printing
You can print TPad in many different formats using TPad::Print function

ROOT support latex syntax
F = show the “Fill” color/style
L = show the “Line2 color/style
P = show the “Point” color/marker style
E = show error bars
2D histogram

- 2D histograms can be drawn with many different styles
- It is possible to rotate with the mouse 3D graphics (e.g. lego plot)
- `SetLogz` can be used to set log scale for the histogram bin
Functions

// ROOT has many predefined functions, e.g.
\( \sin(x) \), \( \exp(x) \), \( \ldots \), \( \text{cd}() \), \( \text{ls}() \), \( \ldots \)

// Many of the ROOT classes have associated functions, e.g.
Draw(), Print(), SetXTitle(), \ldots

// Easy to define new ROOT functions, e.g.
// 1-D function – type is TF1:
TF1 *f1 = new TF1("f1", "x*\sin(x)",0,10);

// 2-D function – type is TF2:
TF2 *f2 = new TF2("f2", "y*\sin(x)",0,10,0,20);

// Plot these functions with
f1->Draw();
f2->Draw("surf4"); // 5 surface options for 2D
Fitting

// Fitting in ROOT based on Minuit (ROOT class: TMinuit)

// ROOT has 4 predefined fit functions, e.g.
‘gaus’: \( f(x) = p0 \exp\{-\frac{1}{2}\left[\frac{x-p1}{p2}\right]^2\} \) // 3 params

// Fitting a histogram with pre-defined functions, e.g.
h1->Fit("gaus"); //landau, exp0, pol0->pol9
h1->Fit("landau", "R",", 3.,15);
// “R” says ‘fit only in range xmin -> xmax’

// User-defined: 1-D function (TF1) with parameters:
TF1 *myFit= new TF1("myfit","[0]*\sin(x) +[1]*\exp(-[2]*x)",0,2);

// Set param names (optional) and start values (must do):
myFit->SetParName(0,"paramA");
myFit->SetParameter(0,0.75); // start value for param [0]

// Fit a histo:
myHist->Fit("myfit");
## Editors

<table>
<thead>
<tr>
<th><strong>Vim</strong></th>
<th><strong>Emacs</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>command line based editor</td>
<td>command line and GUI based editor</td>
</tr>
<tr>
<td>start and open a file</td>
<td>start Emacs:</td>
</tr>
<tr>
<td>$ vim file</td>
<td>$ emacs &amp;</td>
</tr>
<tr>
<td>edit file: go to INSERT mode:</td>
<td>open file:</td>
</tr>
<tr>
<td>press ‘i’</td>
<td>Ctrl-x Ctrl-f</td>
</tr>
<tr>
<td>go back to command mode:</td>
<td>save file</td>
</tr>
<tr>
<td>press ‘Esc’</td>
<td>Ctrl-x s</td>
</tr>
<tr>
<td>save file:</td>
<td>quit Emacs:</td>
</tr>
<tr>
<td>‘:w’</td>
<td>Ctrl-x c</td>
</tr>
<tr>
<td>quit:</td>
<td></td>
</tr>
<tr>
<td>‘:q’</td>
<td></td>
</tr>
</tbody>
</table>
Exercise

• Let’s first work through the tutorial on our website:
  http://ihp-lx.ethz.ch/Stamet/root/ROOT_tutorial_EN.htm
  You will gain first experience using c/python, ROOT, bash and vim/emacs

• Then you will have time to work on todays exercise sheets
Example of standalone app

If you write your own main, say `myapp.cpp`

```cpp
#include <iostream>
#include "TH1F.h"

int main(int argc, char **argv){
    TH1F * h = new TH1F("h","my first TH1F",20, 0, 120);
    h->Draw();
}
```

you can compile it with `g++` just importing the ROOT libraries

```bash
$ g++ -o myapp myapp.cpp `root-config --glibs --cflags`
```

And then run it!

```bash
$ ./myapp
```
Last remark

• Do you have questions? Doubts?
• Can’t the teacher assistants answer?
• Are you feeling desperate?
Last remark

- Do you have questions? Doubts?
- Can’t the teacher assistants answer?
- Are you feeling desperate?
Other references

- http://pi.physik.uni-bonn.de/~bechtle/teaching/physics716_ss14/PBe_RootIntroduction08.pdf
- USEFUL PYTHON EXAMPLES
  http://indico.hep.manchester.ac.uk/getFile.py/access?resId=0&materialId=slides&confId=267