

Train The Trainers

ROOT

Data Analysis Framework

<https://root.cern>



Welcome from the Hosts

- Wifi: hpeguest
- Full agenda ([link](#))
 - Coffee breaks at the restaurant area
 - Lunch: will give you coupons for starter, main, drink and coffee
- Access to terrace



- Objectives for today
- Recent Features with impact on training
- Relevant tools and techniques for training
- Overview of existing training material
- Creation of **the ideal course**
 - Preparation of TOCs for beginners and advanced course
 - Content merging
- Discussion about career, recognition and rewards for trainers

ROOT 6.18 unless
stated otherwise



Introductions

- Introduce yourself, please!
- What do you expect to get from this event?



ROOT Objectives for Today

- **Establish new and improve existing collaborations** between ROOT trainers and core development team
- **Increase the number of ROOT trainers**, also for additional channels to broadcast new features
- Get **your feedback** and **discover training material**
- Produce a draft for an **Ideal Beginners Course** and an **Advanced Course**

Create a document with the contributions of everybody



Features With Impact on Training



Features with Impact on Training

- **Aim:** review features that are particularly relevant for training
 - Not an exhaustive list!
- Some are **new**
 - It is important to keep trainings up to date with new shiny interfaces
 - Can simplify teaching in some cases
- Some are **old**
 - But they are the basis of how ROOT works



Cling: A Compiler-based Interpreter

- ROOT 6 has a C++ interpreter based on llvm technology
 - **All C++ is supported**: recent standards, templates, lambdas ...
- Excellent error diagnostic but +/- **correct C++ is needed**
- Seamless **mixing of compile-time and runtime machine code**
 - E.g. No need for dictionaries for interactivity (still needed for I/O)
- ACLIC still available: useful for auto-generation of dictionaries, performance gap wrt interpreted (now jitted) code much reduced since CINT times.



Interactivity w/o Dictionaries

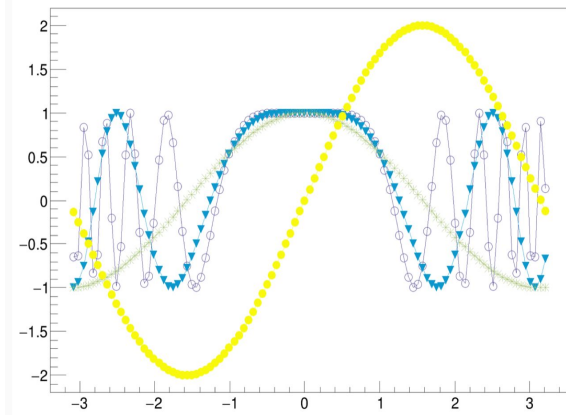
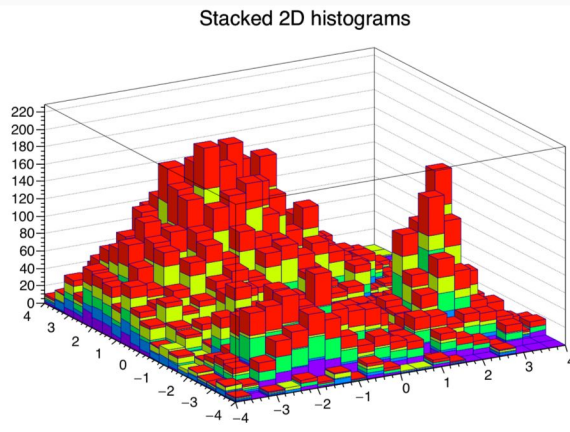
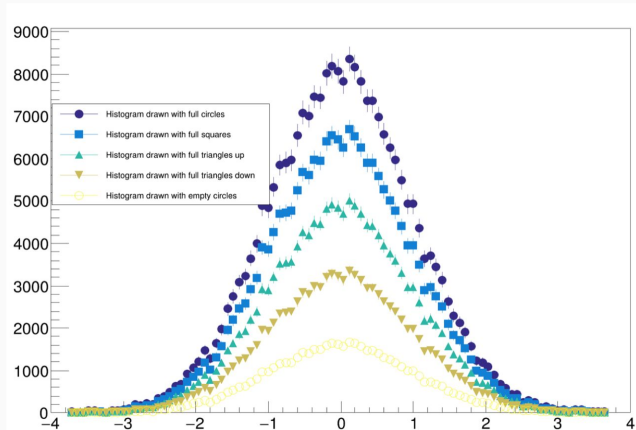
```
root [0] #include "a.h"  
root [1] A o("ThisName");o.printName()  
ThisName  
root [2] dummy()  
(int) 42
```

```
#include <iostream>  
class A{  
public:  
    A(const char* n):m_name(n){};  
    void printName(){std::cout << m_name  
                        << std::endl;}  
private:  
    const std::string m_name;};  
int dummy(){return 42;}
```



Automatic Coloring

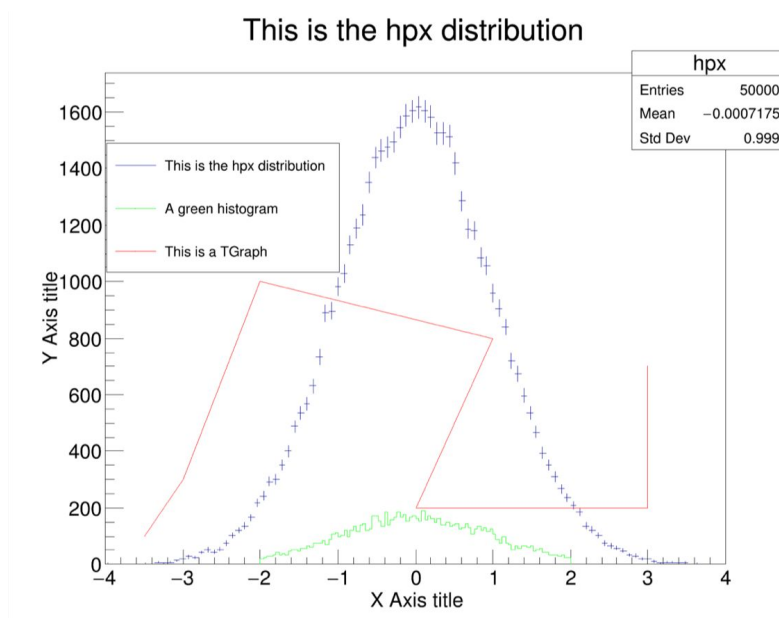
When several histograms or graphs are painted in the same canvas thanks to the option **"SAME"**, via a **THStack** or **TMultigraph** the options PFC (Palette Fill Color), PLC (Palette Line Color) and PMC (Palette Marker Color) allow to select automatically the histograms and graphs colors in the current palette.





Automatic Placement of Legends

- A new TLegend constructor not specifying the legend position.
- Only legend's width and height are defined.
- TPad::BuildLegend() triggers by default the automatic placement.





Implicit Parallelism

- ROOT supports parallelism
- Provides building blocks for expression of **explicit parallelism**
 - E.g. map reduce pattern implementation, (RW)mutex ...
- Provides **implicit parallelism**: behind the scenes, some operations are implicitly parallelised
 - An internal thread pool is created with [Intel TBB](#)

```
ROOT::EnableImplicitMT();
```

```
~> root -t
```





Implicit Parallelism

- **RDataFrame** internally runs the event-loop by parallelizing over clusters of entries
- **TTree::GetEntry** reads multiple branches in parallel
- **TTree::FlushBaskets** writes multiple baskets to disk in parallel
- **TTreeCacheUnzip** decompresses the baskets contained in a **TTreeCache** in parallel
- **THx::Fit** performs in parallel the evaluation of the objective function over the data
- **TMVA::DNN** trains the deep neural networks in parallel
- **TMVA::BDT** trains the classifier in parallel and multiclass BDTs are evaluated in parallel

See [online documentation](#) for more details.



Classic interfaces: TTree::Draw

```
myDataset->Draw("pt", "eta > 2")
```

```
myDataset->Draw("Muon_pt", "Sum$(Muon_pt*(Muon_eta > 1)) > 30")
```

- ad-hoc language allows to quickly specify queries
- can only produce histograms/graphs
- one event loop per histogram
- parallelisation is not possible
- relies on ROOT memory management of the histogram

Can we address these limitations without losing expressivity?



ROOT Declarative Analysis: RDataFrame



Customisation point,
public interface!

Goals:

- Be **the fastest** way to manipulate HEP data
- Be the go-to ROOT analysis interface from laptop to cluster
- Consistent interfaces in **Python and C++**
- Top notch [documentation and examples](#)



An ergonomic, fast C++ dataframe

```
ROOT::RDataFrame df(dataset); ..... on this (ROOT, CSV, ...) dataset  
auto df2 = df.Filter("x > 0") ..... only accept events for which  $x > 0$   
    .Define("r2", "x*x + y*y"); ..... define  $r2 = x^2 + y^2$   
auto rHist = df2.Histo1D("r2"); ..... plot r2 for events that pass the cut  
df2.Snapshot("newtree", "out.root"); ..... write the skimmed data and r2  
                                         to a new ROOT file
```



An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT(); ..... Run a parallel analysis  
ROOT::RDataFrame df(dataset); ..... on this (ROOT, CSV, ...) dataset  
auto df2 = df.Filter("x > 0") ..... only accept events for which  $x > 0$   
    .Define("r2", "x*x + y*y"); ..... define  $r2 = x^2 + y^2$   
auto rHist = df2.Histo1D("r2"); ..... plot r2 for events that pass the cut  
df2.Snapshot("newtree", "out.root"); ..... write the skimmed data and r2  
                                     to a new ROOT file
```



An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT(); ..... Run a parallel analysis  
ROOT::RDataFrame df(dataset); ..... on this (ROOT, CSV, ...) dataset  
auto df2 = df.Filter("x > 0") ..... only accept events for which  $x > 0$   
    .Define("r2", "x*x + y*y"); ..... define  $r2 = x^2 + y^2$   
auto rHist = df2.Histo1D("r2"); ..... plot r2 for events that pass the cut  
df2.Snapshot("newtree", "out.root"); ..... write the skimmed data and r2  
                                         to a new ROOT file
```

Lazy execution guarantees that all operations are performed in **one event loop**

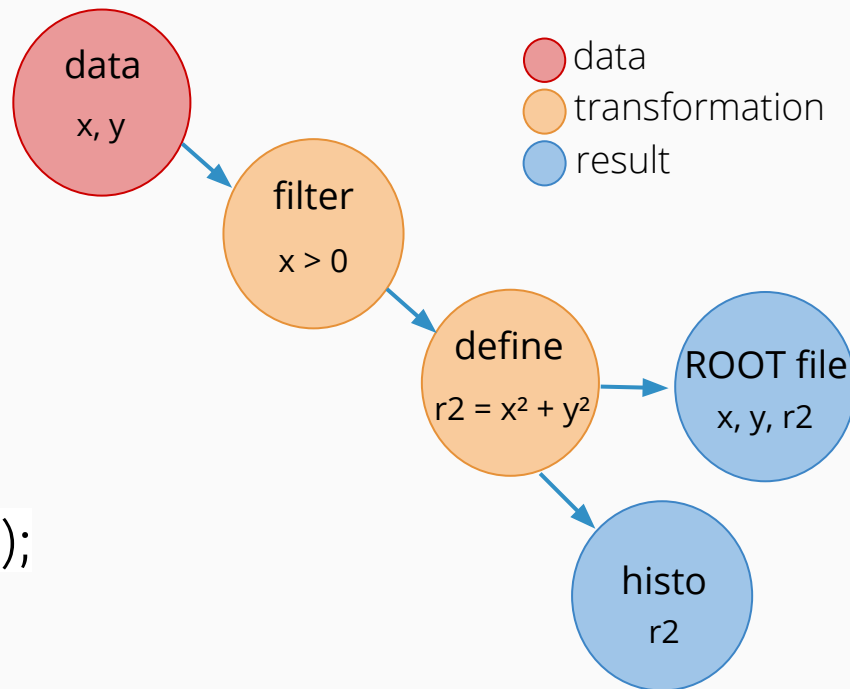


Analyses as computation graphs

```
ROOT::RDataFrame df(dataset);  
auto df2 = df.Filter("x > 0")  
    .Define("r2", "x*x + y*y");  
auto rHist = df2.Histo1D("r2");  
df2.Snapshot("newtree", "newfile.root");
```



Write datasets to disk, also in parallel.





The New PyROOT (1/2)

- A new (experimental) PyROOT is in the making
 - **Prototype available in 6.18, will come in 6.20**
 - CMake build with `-Dpyroot_experimental=ON`
- Goal: make PyROOT more modern, interoperable and pythonic
 - **Modern** C++ features (built on top of latest Cppyy)
 - **Interoperable** with Python ecosystem (NumPy, pandas)
 - **Pythonisations** for ROOT and user classes



The New PyROOT (2/2)

- Support for **Python 2 & 3**
- PyROOT **documentation** and examples
 - Integrated in the reference guide



Example: Modern C++ Support

- Possible to use C++ lambdas from Python

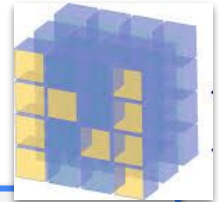
```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine(
"auto mylambda = [](int i) { std::cout << i << std::endl; };")

>>> ROOT.mylambda
<cppyy.gbl.function<void(int)>* object at 0x35f9570>
>>> ROOT.mylambda(2)
2
```



ROOT -> NumPy : TTree

- Read a **TTree** into a NumPy array
 - Branches of arithmetic types (ntuples)



```
myTree # Contains branches x and y of type float

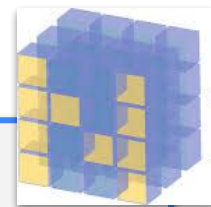
# Convert to numpy array and apply numpy methods
myArray = myTree.AsMatrix()
m = np.mean(myArray, axis = 0)

# Read only specific branches
onlyX = myTree.AsMatrix(columns = ['x'])
```




ROOT -> NumPy : RDataFrame

- Even more powerful way to read **TTrees** into NumPy
 - All **RDataFrame** operations available
 - Optional parallelism



```
from ROOT import RDataFrame
```

```
df = RDataFrame('myTree', 'file.root')
```

```
# Apply cuts, define new columns
```

```
df = df.Filter('x > 0').Define('z', 'x*y')
```

```
# Column dictionary, each column is a NumPy array
```

```
cols = df.AsNumpy()
```



ROOT -> pandas : RDataFrame

```
# Run input pipeline with C++ performance that can process TBs of data, reads from remote, ...
```

```
df = RDataFrame('tree', 'file.root')  
    .Filter('pT_j0>30', 'Trigger requirement')  
    .Filter('n_jet >= 2', 'Jet multiplicity cut')  
    .Define('r_j0', 'sqrt(eta_j0*eta_j0 + phi_j0*phi_j0)')
```

```
# Read out final selection with defined variables as NumPy arrays
```

```
col_dict = df.AsNumpy(['r_j0', 'eta_j0', 'phi_j0'])
```

```
# Wrap data with pandas
```

```
import pandas
```

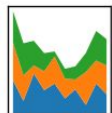
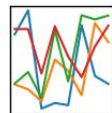
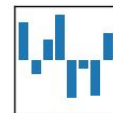
```
p = pandas.DataFrame(col_dict)
```

```
print(p)
```

	r_j0	eta_j0	phi_j0
0	0.26	0.1	-0.5
1	1.0	-1.0	0.0
2	4.45	2.1	0.2

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$





NumPy -> ROOT : RDataFrame

- Close the circle: go back to ROOT from NumPy

```
import ROOT
import numpy

data = {
    "x": numpy.array([1, 2, 3]),
    "y": numpy.array([4, 5, 6])
}

df = ROOT.RDF.MakeNumpyDataFrame(data)
df = df.Define("z", "x + y")

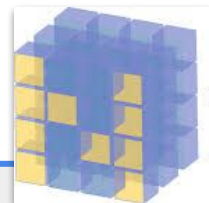
print(df.Mean("z").GetValue()) # Returns 7.0
```

```
import ROOT

df = ROOT.RDataFrame(10).Define("x",
                                "(int)rdfentry_")
data = df.AsNumpy()

# Do something with the data (in NumPy)

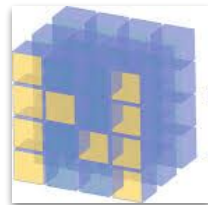
df2 = ROOT.RDF.MakeNumpyDataFrame(data)
df2.Snapshot("tree", "file.root")
```





NumPy -> ROOT : RVec

- Convert NumPy arrays to [RVecs](#)
 - RVec: C++ view on top of contiguous memory
 - Can own, too
 - Pass them into C++ functions
 - Conversion could be done implicitly in the future



```
arr = np.array([1,2,3])  
vec = ROOT.AsRVec(arr) # zero-copy operation  
my_cpp_fun(vec)
```



User Pythonizations

- Allow ROOT users to define pythonizations for their own classes
 - Lazily executed

ROOT 6.20

```
@pythonization('MyCppClass')
```

```
def my_pythonizer_function(klass):
```

```
    # Inject new behaviour in the class
```

```
    klass.some_attr = ...
```

Python proxy of the class





TFormula based now on Cling. New functionality:

- better parameter definition `TF1("f1", "gaus(x, [Constant], [Mean], [Sigma])");`
- function composition by concatenating expressions
 - `TF1 fs("sigma", "[0]*x+[1]");`
 - `TF1 f1("f1", "gaus(x, [C], [Mean], sigma(x, [A], [B]))");`
- normalized sum for component fitting
 - `TF1 model("model", "NSUM(expo, gaus)");`
- convolutions
 - `TF1 voigt("voigt", "CONV(breitwiegner, gaus)", xmin, xmax);`
- can define vectorized functions for faster fitting and evaluation
- support for auto-differentiation (automatic generation of gradient function)



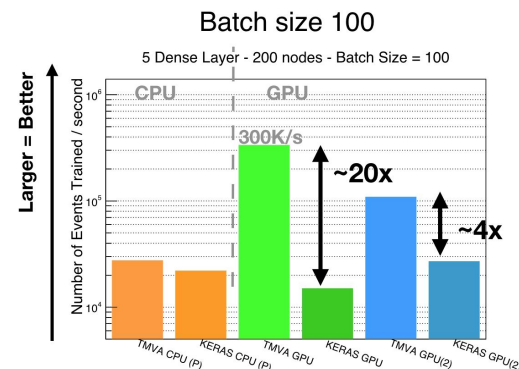
Some old classes are being replaced with modern interfaces and richer in functionality

Examples:

- **TVirtualFitter, TFitter** -> **ROOT::Fit::Fitter, ROOT::Fit::FitResult**
- **TMinuit** -> **ROOT::Math::Minimizer**
 - Minimizer is common interface to all minimization algorithms in ROOT (Minuit, Minuit2, Fumili, minimizers from GSL)
- **TVector** -> **ROOT::Math::XYZVector**
- **TLorentzVector** -> **ROOT::Math::XYZTVector, PtEtaPhiEVector**



- On-going modernisation of TMVA
- New deep learning tools introduced with support for
 - dense layers (fully connected neural networks)
 - convolutional layers
 - recurrent layers
 - support for several DL optimizers (ADAM, SGD, etc.)
 - efficient implementation running on both CPU (MT) and GPU (Cuda)
- Direct interfaces to Python ML tools
 - Keras and scikit-learn





Relevant Tools and Techniques



- Training tools are a key aspect of any training
- Main objective: students should **focus on the content** of the training, not on setting up the tools!
- No perfect solution (yet)
 - A combination of two or more can be used to increase reach (“VMs on Windows”)
 - Always good to have a “plan B”



Install ROOT From Sources

- ROOT is open source and sources are packaged for every [release](#)
- Configure and build by hand. Maximal flexibility, can easily (re-)use later.
- 15+ minutes needed (necessary if want to contribute)
- Not adequate for absolute beginners



Install ROOT From Binaries

- ROOT is open source and binaries are packaged for every [release](#)
- Works out of the box
- Might land on messy environments, e.g. customisations, layered installations during the years...
- Maybe some participants have an unsupported platform. These need special treatment / be provided with laptop.



Get ROOT from CVMFS I

- Central installation on CVMFS (see [our releases](#))
- Good for beginners and advanced courses (full installations)
- Needs cvmfs or portal with mount point
- Not all possible Linux flavours supported



ROOT environment on cc7:

```
. /cvmfs/sft.cern.ch/lcg/app/releases/ROOT/6.16.00/x86_64-centos7-gcc48-opt/bin/thisroot.sh
```



Get ROOT from CVMFS II

- Gentoo prefix on CVMFS
`/cvmfs/sft.cern.ch/lcg/contrib/gentoo/startprefix`
- One prefix to rule them all (linux)
- Needs cvmfs or portal with mount point
- Living inside gentoo bubble



```
[shageboe@pcphsft98 ~]$ /cvmfs/sft.cern.ch/lcg/contrib/gentoo/startprefix
Entering Gentoo Prefix /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/x86_64
[shageboe@pcphsft98 ~]$ root-
root-6.12      root-6.14      root-6.16      root-config      root-config-6.12  root-config-6.14  root-config-6.16
[shageboe@pcphsft98 ~]$ root-6.16
```

```
-----
| Welcome to ROOT 6.16/00                               https://root.cern |
|                                                         (c) 1995-2018, The ROOT Team |
| Built for linuxx8664gcc on Jan 23 2019, 09:06:13      |
| From tags/v6-16-00@v6-16-00                          |
| Try '.help', '.demo', '.license', '.credits', '.quit'/.q' |
|-----
```

```
root [0] █
```



Packages For Linux Distributions

- Distros people package ROOT on a voluntary basis, e.g. Gentoo etc.
- Immediate and respectful of the environment of the machine
- Latency in updating the version
- Depend on volunteers



Go Full Container

- Prepare a container or a full VM and distribute to participants
- Maximal portability
- Need to distribute to all participants
- Probably needs to be curated by trainer(s)
- Runs in a “bubble” that doesn’t interact much with outside world



Install ROOT with Conda

- With Conda
- Ideal for beginners
- Needs good network and some space on disk

Given a working conda installation (one-line instructions [here](#)), install ROOT and its dependencies:

```
▶ conda create --name my-root-env --channel conda-forge python=3 root
```

Activate the environment with:

```
▶ conda activate my-root-env
```

Deactivate with:

```
▶ conda deactivate
```

root and **root-*** commands work out of the box, as well as PyROOT.
To compile your C++ source code, use **\$(root-config --cxx)** as the compiler.





Use The CERN Notebook Service

- In a notebook via the [CERN SWAN service](#)
- Possibly the easiest: everyone knows how to use a web browser!
- Perfect for beginners: log in and start programming
- ROOT GUI still not fully supported
- Needs CERN account
- Alternative: Binder + Conda
 - easy to prepare, no accounts needed but unsafe to rely on best effort service level





Summary of Options / Tools

- From sources
- From binaries
- Central installation on CVMFS
- Using the packaging of your Linux distro
- Containers & VMs
- Via Conda
- In a notebook:
 - Via the CERN SWAN service
 - Binder + Conda

Which way do
you prefer?



Overview of Existing Material



Target Audiences

- In your community/environment:
 - How is the target audience of a ROOT training?
 - How much ROOT, C++ and Python do they know when entering the course?
- Do you think this classification is accurate?
 - Non-HENP
 - Novice: knows a bit of C++, almost no ROOT (bachelor / master)
 - Advanced: knows C++/Python, some ROOT (PhD and above)



Existing sources of documentation (1/2)

- [Class documentation](#) in Doxygen, also known as *Reference Guide*
- [Example C++ macros and Python scripts](#), also known as *Tutorials*
- [Training git repository](#) within the root-project organisation
 - Hands-on exercises and slides!
- [Primer and Topical Manuals](#)
- ROOT Website, in particular the [third party courses page](#)



Existing sources of documentation (2/2)

- Man pages and --help switch for command line utilities (root, rootcp, ...)
- [User's Guide](#), e.g. ROOT manual

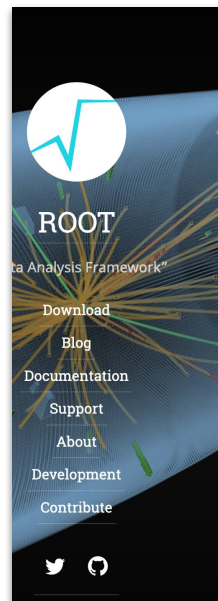
In addition, the [ROOT Forum](#), our knowledge base and support channel

Goal: rationalise all this, also based on the feedback of today.



ROOT Website: <https://root.cern>

- Currently based on Drupal technology
- Working on a [Jekyll based prototype](#)
 - Version controlled, simplify content updates, enable contributions
 - More modern looking
- Goals: quick access, “what’s ROOT” for non-HEP



ROOT



[Getting Started](#)



[Reference Guide](#)



[Forum](#)

ROOT is ... A modular scientific software toolkit. It provides all the functionality for working with big data processing, statistical analysis, visualisation and storage. It is modular but integrated with other languages such as Python and R.



- Most up to date piece of documentation, automatically generated daily
- Generated with **Doxygen, customized for ROOT**
- Available for all branches, e.g.
 - <https://root.cern/doc/v616/index.html>
 - <https://root.cern/doc/v618/index.html>
 - <https://root.cern/doc/master/index.html>
- Contains more than mere class documentation
 - see e.g. [RDataFrame manual](#)



- **Small introduction to ROOT available** (~60 pages)
- Usage of ROOT prompt, creating histograms and graphs, elements of ttree reading/writing, fitting
 - Objective: analyse comfortably measurements of a **2nd/3rd year lab during Physics bachelor**
- Available [here](#) as a PDF and in notebook format (prototype)



Code Examples a.k.a. *Tutorials*

- Rationale: **learn by example**. Provide minimal and clear examples for the most relevant ROOT features.
- Part of the [reference guide](#), one page per tutorial:
 - code, text, notebooks, images and output displayed
- Language: Python and C++.
 - New tutorials in both languages, existing ones being translated
- Plans for the future:
 - Present examples also in a “gallery”, to get to a plot's code
 - New organisation ([PR #3768](#)) with a decade of additional experience



Code Examples a.k.a. *Tutorials*

Files

file **df001_introduction.C**

View Notebook Open in SWAN This tutorial

chain like approach.

file **df001_introduction.py**

View Notebook Open in SWAN This tutorial

chain like approach.

file **df002_dataModel.C**

View Notebook Open in SWAN This tutorial

```
import ROOT

def fill_tree(treeName, fileName):
    tdf = ROOT.ROOT.RDataFrame(50)
    tdf.Define("b1", "(double) tdfentry_")\
        .Define("b2", "(int) tdfentry_ * tdfentry_").Snapshot(treeName, fileName)

# We prepare an input tree to run on
fileName = 'df004_cutFlowReport_py.root'
treeName = 'myTree'
fill_tree(treeName, fileName)

# We read the tree from the file and create a RDataFrame, a class that
# allows us to interact with the data contained in the tree.
RDF = ROOT.ROOT.RDataFrame
d = RDF(treeName, fileName)

# ## Define cuts and create the report
# An optional string parameter name can be passed to
# Named filters work as usual, but also keep track of
filtered1 = d.Filter('b1 > 25', 'Cut1')
filtered2 = d.Filter('0 == b2 % 2', 'Cut2')

augmented1 = filtered2.Define('b3', 'b1 / b2')
filtered3 = augmented1.Filter('b3 < .5', 'Cut3')

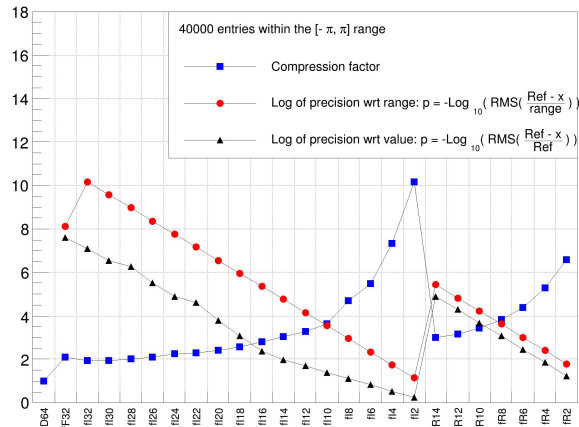
# Statistics are retrieved through a call to the Report()
# when Report is called on the main RDataFrame object
# all named filters declared up to that point. When
# state (i.e. a chain/graph node), it retrieves statistics
# the section of the chain between the main RDataFrame
# Stats are printed in the same order as named filters
# graph, and refer to the latest event-loop that has
# RDataFrame.
print('Cut3 stats:')
filtered3.Report()
print('All stats:')
allCutsReport = d.Report()
allCutsReport.Print()
```

```

Cut3 stats:
All stats:
Cut1      : pass=24      all=50      -- eff=48.00 %
Cut2      : pass=25      all=50      -- eff=50.00 %
cumulative eff=50.00 %

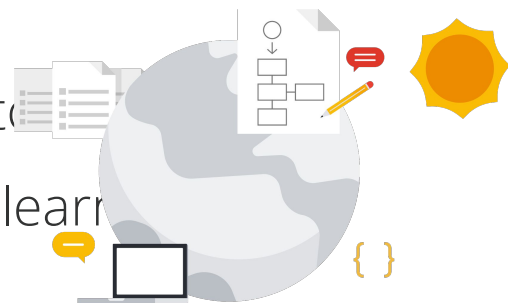
```

Double32_t compression and precision





- The [User's Guide](#) is the ROOT manual
 - Quite some written text, division in chapters, etc
- Is this the way in which scientists born in the 90ies learn
- It needs an update:
 - Shall we update it? Is it used? Should we keep it?
 - ROOT applied to the [Google Season of Docs 2019](#) with [this proposal](#) with the objective of bringing a professional technical writer and our Toolkit for a few months to work its documentation.





Future of the Doc Parts

- We would like to hear your opinion on the doc we just presented
 - What should we remove?
 - What should become commonly maintained by trainers?



Time for *Your* Material!



Creation of a Beginners' and Advanced Training



Objective: produce all together the table of contents of two ROOT trainings: basic and advanced.

1. Two teams are formed, both teams produce basic and advanced TOCs, written in the form of (structured) bullet lists on google docs ([team 1 doc](#), [team 2 doc](#)) - **1h**
2. One or more representatives per team will present the TOCs, also motivating the choices made - **30 m**
3. The proposals are merged: a moderated discussion takes place, a secretary writes down the outcome - **1h**



Helping the Trainers



- The activity of teaching requires significant effort
 - Preparing slides & exercises, tools for the students, ...
 - ... plus the actual training!
- Therefore, **recognition** as a trainer is important. How?
 - Official ROOT Trainer certificate (duration?)
 - Badge in the ROOT forum
 - Gallery of trainers on the ROOT website
 - What else?



Sharing and Evolving Material



Sharing and Evolving Material

- Goal: share ROOT training material, evolve it collaboratively by ROOT team members and trainers
- How? With material where?
 - Version control everything and accept pull requests? ROOT's (?) Github organization? Google slides with comments?
 - Permission to access/edit it?
 - Does it scale to ~50 contributors?
- Which material?
 - A new users guide with our tocs?
 - The ROOT website?



Wrap-Up