Lecture 8: Scripting using ROOT 6

1 Introduction to Scripting with ROOT

If you have set up `ROOT` on your own machine, you are good to go: Make sure you have run

```
source <your_root_installation_folder>/bin/thisroot.sh
```

either through your `.bashrc` file or locally in the `shell`, and you can directly get started.

If you have **not** installed ROOT (https://root.cern.ch/releases/release-64002/) yet, you will have to use the installed binaries already existing within our `physik.uzh.ch` cluster.
First, you need to establish an `SSH` connection to one of the available and properly set-up computers on our cluster via:

```
ssh -X <username>@<hostname>.physik.uzh.ch
```

The available hosts (with 1 logical core and are 4 GB of RAM each) are:

```
ohm01
ohm02
ohm03
ohm04
ohm05
ohm06
```

They are configured such that you can access them remotely without tunneling through another frontend. You should have requested a `Linux` account on the Physics Institute's cluster, and in return received your `<username>`. You will be asked your password each time you log in (you should have received your initial password together with your username), unless you set up an `ssh-key` connection.

Once you are logged in, simply run the following command to set up the necessary environment variables:

```
source /app/cern/root_v6.36.10/bin/thisroot.sh
```

There will be no confirmation, so let's try to start `ROOT`.

```
~> root -l
root [0]
```

Now you are in the `ROOT` command line. From here, you can type in small sections of your code to test them interactively. For example:

```
root [0] for (int i = 0; i < 10; i++){
root (cont'ed, cancel with .@) [1]cout << "blah " << i << endl;
root (cont'ed, cancel with .@) [2]}
blah 0
...
blah 9
root [3]
```

To exit the ROOT command line, type:

```
root [4] .q
```

Extensive documentation on ROOT is available at: https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html. In this class we will essentially only need the following ROOT *prompt commands*:

```
.x // Executes the script that follows .x
.q // Quits the ROOT command line
```

You can also write a short ROOT script as follows:

```
1  // short.cxx
2  {
3    for (int i = 0; i < 10; i++){
4      cout << "script " << i << endl;
5    }
6  }
```

From the terminal, you can execute the script as follows:

```
> root -l short.cxx
root [0]
Processing short.cxx...
script 0
...
script 9
root [1]
```

Many physicists rely on ROOT to perform data analysis in such a way that compilation of code is rarely done. However, this approach has significant disadvantages:

- *Lower performance and stability*: ROOT scripts are interpreted, so they run slower and can be less stable than fully compiled programs.
- *Limited access to parallel processing features:* Certain optimizations and multi-threading options are not available in scripting mode.

In the next class, we will discuss how to compile code using ROOT packages, but today we will focus on scripting.

It is also possible to write ROOT code in the form of a proper function or a C++ program, but some modifications are required. For example, the following code block will not work:

```
1  // main.cxx
2  #include <iostream>
3  #include <vector>
4  using namestapce std;
5  void check(int i = 0){
6    cout << "check " << i << endl;
7  }
8  int main(int argc, char *argv[]){
9    vector<int> test = {2, 3, 5, 7};
10   for (int i = 0; i < test.size(); i++){ check(test[i]); }
11   return 0;
12 }
```

It will instead produce the following error message:

```
> root -l main.cxx
root [0]
Processing main.cxx...
input_line_9:2:2: error: no matching function for call to 'main'
 main() /* .x tries to invoke function `main` */
 ^~~~
/home/ttp/kallweit/phy224/2025/Lecture_8/not-working/main.cxx:6:5: note: candidate function
↪  not viable: requires 2 arguments, but 0 were provided
int main(int argc, char *argv[]){
    ^
root [1]
```

The problem lies in the definition of `main`: for this scripting-style, command-line use of ROOT, the `main` function must not take any command-line arguments.

Moreover, it is important to know that the ROOT command line already provides many commonly used headers and libraries. This means that much of the required functionality is already available, so the corresponding headers often do not need to be included manually.

The following example works correctly:

```
1  // main.cxx
2  void check(int i = 0){
3    cout << "check " << i << endl;
4  }
5  int main(){
6    vector<int> test = {2, 3, 5, 7};
7    for (int i = 0; i < test.size(); i++){ check(test[i]); }
8    return 0;
9  }
```

The output is:

```
> root -l
root [0] .x main.cxx
check 2
check 3
check 5
check 7
(int) 0
root [1]
```

Note that ROOT does not execute the standard C++ `main()` function. Instead, it automatically executes the function whose name is derived from the filename (by removing the extension), which in the previous case happens to coincide with `main`. And this function may not have arguments. Other functions in the file can be called either from inside this main function or explicitly from the ROOT command line.

As you can see, a standard C++ program may or may not work directly in ROOT, depending on which language features it uses. Some modifications are often required to make a C++ source file executable as a ROOT script.

Because these modified C++ files can be executed without a separate compilation step, they are effectively used as scripts rather than as conventional compiled programs. This makes it particularly important to write careful C++ code, since issues such as memory leaks and undefined behaviour can still occur and may only become apparent during execution.

Before we dive deeper into ROOT 's features, we will begin practicing scripting with what we have already learned.

We will now start a 60-minute practical programming session.

2 Practical Session – Part I

Command line and scripting in C++ using ROOT

– Using the `ROOT` command line:

1. Use a `for` loop to count from `6` to `15`.
2. Declare all simple variables on the *stack*.
3. Declare all simple variables on the *heap*, then free them and delete the pointers.
4. Perform simple calculations and use `ROOT` as a calculator.

– Using the script execution feature:

1. Write a script enclosed in { } that counts from `2` to `11` using a `for` loop.
2. Write a main script that defines simple calculation functions (addition, multiplication, etc.).
3. Write a main script with functions for adding and multiplying 2×2 matrices.
4. Write a main script that defines a custom object containing 5 internal variables.
5. Write a main script with a function that computes the mean, median, sample standard deviation, sample skewness, maximum, and minimum for an `array` stored on the *stack*.
6. Repeat the previous task for a `vector` stored both on the *heap* and *stack*, and for an `array` allocated on the *heap*.

We will now continue with a 15-minute theory session.

3 Basic ROOT features

Now that we are somewhat familiar with the ROOT command line and scripting features, let's explore some of its most useful capabilities. One particularly useful feature of ROOT is its file browser. You can open the browser, called TBrowser, directly from the command line to preview any code, objects, or images stored in ROOT files.

```
root [0] new TBrowser
(TBrowser *) 0x2f8f9d0
root [1]
```

If connected remotely, your TBrowser will open via XQuartz, Xming, or natively through an X server, as shown in Fig. 1.

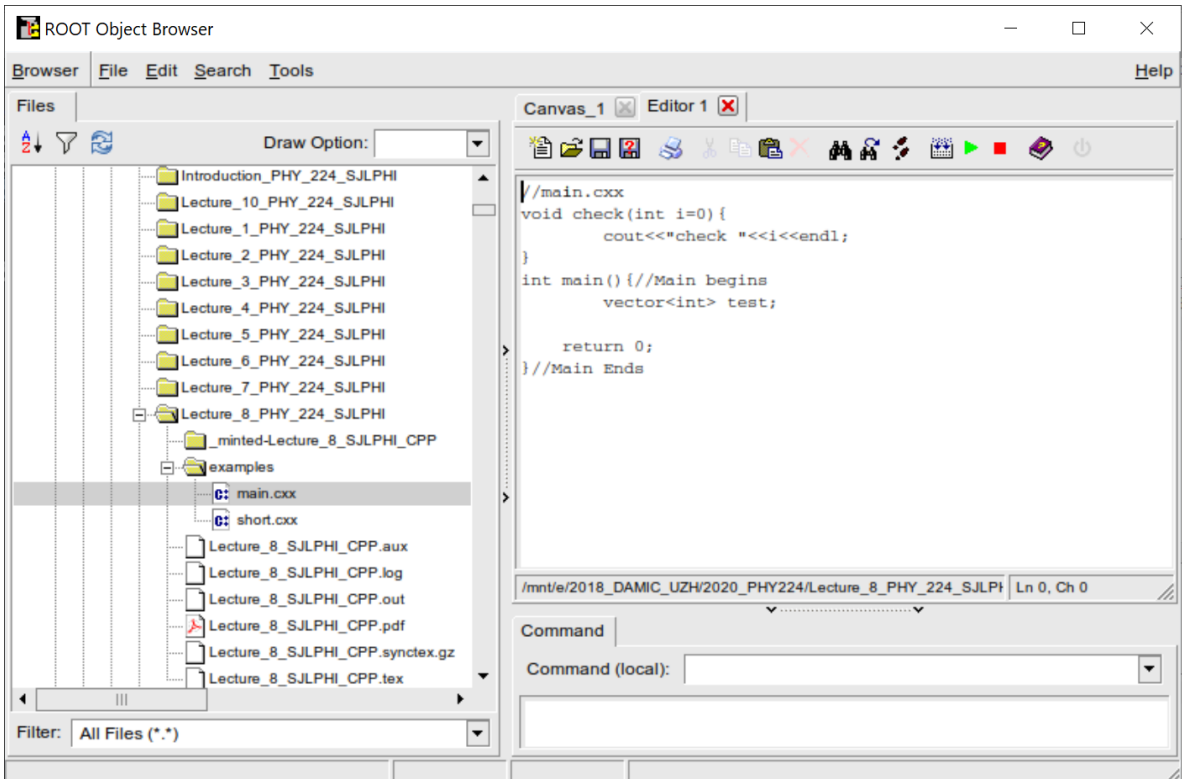

Figure 1: Your TBrowser may look different than this.

If you encounter a display error at this point, make sure that XQuartz or Xming is running, since these programs do not start automatically by default. Also, be aware that there may be delays when using X11 forwarding over SSH, which can affect how quickly the browser appears.

The TBrowser is also useful for exploring data stored in ROOT files (with the .root extension). For the purposes of this course, we will focus only on the basic features of ROOT.

Let's consider the following example:

```
1  // rootexample.cxx
2  int rootexample(){
3    TRandom3 * randomer = new TRandom3();
4    TFile * file = new TFile("myrootfile.root", "RECREATE");
5    TH1I * integerhist = new TH1I("integerhist", "integerhist", 10, 0, 100);
6    for (int i = 0; i < 1000; i++){
7      integerhist->Fill(randomer->Gaus(50, 10));
8    }
9    file->cd();
10   integerhist->Write();
```

```
11    file->Close();
12    delete file;
13    return 0;
14  }
```

In the example above, `TRandom3` is used to generate `1000` Gaussian-distributed random numbers with a mean of `50` and a standard deviation of `10`. These numbers are filled into a one-dimensional histogram (`TH1I`). The histogram is named `integerhist` and has `10` bins covering the range from `0` to `100`. Once the histogram has been filled, it is written to a `ROOT` file. We can run this script and subsequently quit `ROOT` using the following command:

```
> root -l rootexample.cxx -q
Processing rootexample.cxx...
(int) 0
```

The file `myrootfile.root` has now been created. You can quickly inspect this file and the histogram it contains using `TBrowser`, as shown below in Fig. 2.

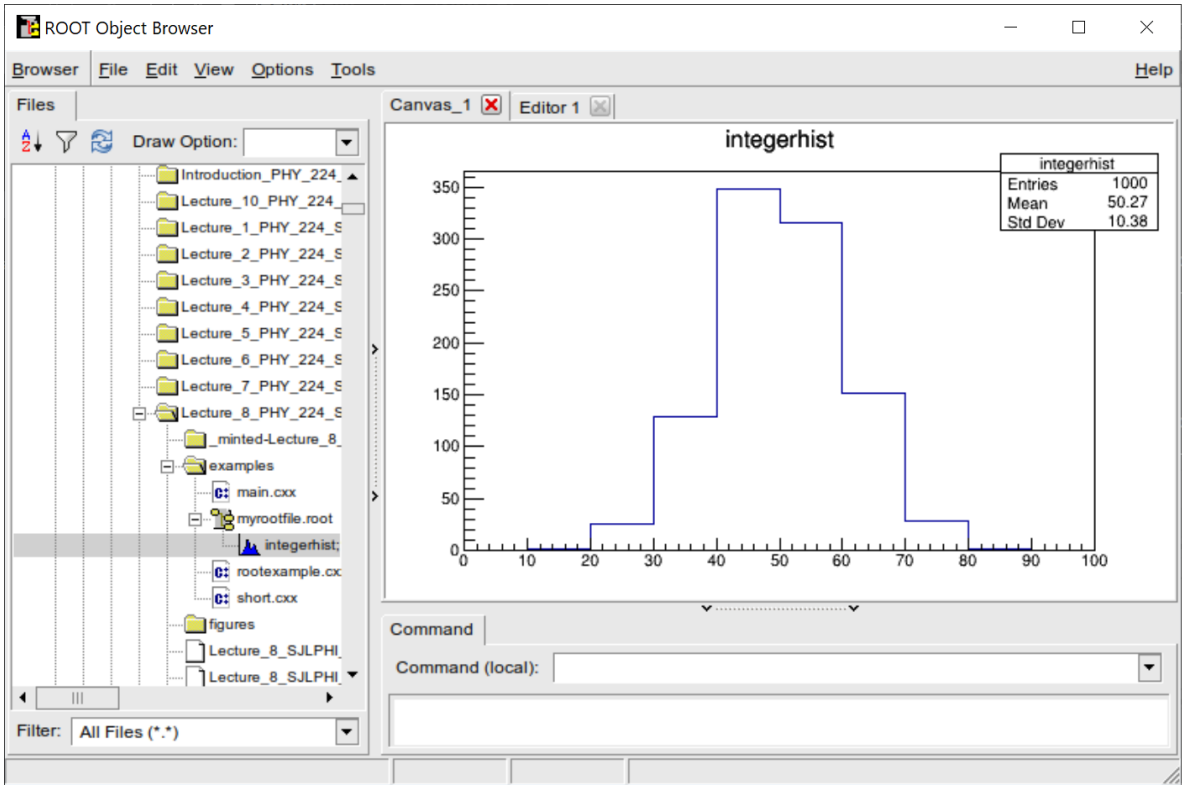

Figure 2: Newly generated `TH1I` in `myrootfile.root`.

Of course, many more operations can be performed on histograms, and `ROOT` also provides support for `2D` and `3D` histograms. We will explore some of these features during the practical session.

Keep in mind that there is extensive documentation available online for `ROOT`, and being able to search for information and apply it in your scripts or code is an essential skill. The following exercise builds on the concepts demonstrated so far, and relevant details can be found in the `ROOT` User's Guide.

For this exercise, you are encouraged to explore the guide and other resources to identify the appropriate commands and integrate them into your code.

We will now start a 60-minute practical programming session.

4 Practical Session – Part II

ROOT Scripting: ROOT features

1. Create a script that saves an empty `.root` file.
2. Create a script that reads the previously saved empty `.root` file.
3. Create a script that reads the above file, adds new information, and writes a `TH1I` histogram filled with `1000` random numbers generated by `TRandom3::Gaus()`.
4. Repeat the previous task but fill the histogram with `10000` random numbers generated by `TRandom3::Uniform()`.
5. Create a script that generates and fills a `2D TH2I` histogram with `10000` numbers drawn from `TRandom3::Gaus()`. Use `SetBinContent` to set the histogram values and draw it in `colz` mode before writing it to the `ROOT` file.
6. Use `TBrowser` to perform a fit on the above histograms.
7. Perform a fit on the above `TH1I` histograms directly from a script using the built-in `->Fit("gaus")` function.
8. Define a custom `TF1` function and perform a fit on the above `TH1I` histograms.

We will now conclude with a 15-minute question session.

5 Conclusion

In this session, we explored the command-line interface and scripting capabilities of `ROOT`, experimenting with some of its basic functionalities. In the next session, we will take a step further by working with compiled `C++` code while still making use of the powerful features provided by `ROOT`.