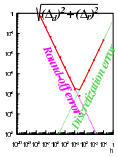
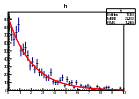
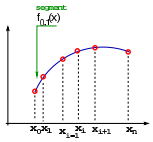
Computational Physics

numerical methods with C++ (and UNIX)

2018-19



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: fernando.barao@tecnico.ulisboa.pt

Computational Physics

Operating systems

UNIX (linux)

Fernando Barao, Phys Department IST (Lisbon)

UNIX shell

- ✔ the shell, the command line interface, is a program like any other one
- ✔ it takes commands from the user and transmit to the operating system the corresponding actions
- ✔ most shell commands are actually small programs, accepting options and arguments

  example: ls -l <dirname>

```
[80]vaioZ11[Aulas_Teoricas/linux]: ls -l
total 304
296 -rw-r--r-- 1 baraon baraon 301366 Sep  8 16:09 FIG.unix-shell.example.eps
  8 -rwxr-xr-x 1 baraon baraon   4357 Sep 22  2014 slide-FC.linux.g++_compiler.tex*
```

- ✔ unix shells

  bash: the default shell on most linux systems

  csh: C shell (similar to C programming language)

  ksh: korn shell

  tcsh: enhanced but compatible with C shell

- ✔ to find your current shell: *echo $SHELL*

```
[11]vaioZ11[FC_aulas/figs]: echo $SHELL
/bin/tcsh
```

- ✔ to find your available shells: *cat /etc/shells*

UNIX shell (cont.)

- ✔ **environment variables**
  - ▶ variables that are defined for the current shell and are inherited by any child shells or processes

    we can used them to pass information into processes
  - ▶ by convention, they are usually defined using capital letters

list all environment variables

```
> env

or
> printenv
```

```
[14]vaioZ11[Computational-Physics/Aulas_Teoricas]: env | head
XDG_CONFIG_DIRS=/etc/xdg/xdg-default:/etc/xdg
LC_TELEPHONE=pt_PT.UTF-8
LANG=en_US.UTF-8
DISPLAY=:0
XDG_VTNR=8
LOGNAME=baraon
PWD=/home/baraon/HOMEDIR/ist/Ensino/Computational-Physics/Aulas_Teoricas
MANDATORY_PATH=/usr/share/gconf/default.mandatory.path
GNOME_KEYRING_PID=1837
LC_NAME=pt_PT.UTF-8
```

UNIX shell (cont.)

set environment variables

```
# bash shell
> export VARNAME=value

# c-shel
> setenv VARNAME value
```

```
[35]vaioZ11[Ensino/Computational-Physics]: setenv FISCOMP "teste"
[36]vaioZ11[Ensino/Computational-Physics]: env | grep FISCOMP
FISCOMP=teste
```


UNIX shell (cont.)

- ✔ **some common environment variables**
  - SHELL
  - TERM terminal
  - USER current user logged in
  - PWD current working dir
  - PATH list of directories that will be check when looking for commands
  - HOME user home directory
  - LD_LIBRARY_PATH list of directories where shareable libs are located

UNIX shell initialization

- file sequence

bash shell non-login session

Korn shell	C Shell	Bourne Shell
/etc/environment	/etc/environment	/etc/environment
/etc/security/environ	/etc/security/environ	/etc/security/environ
/etc/profile	/etc/csh.cshrc	/etc/profile
/etc/csh.login		
$HOME/.profile	$HOME/.cshrc	$HOME/.profile
$HOME/.kshrc	$HOME/.login	

- */etc/environment* and */etc/security/environ* files are executed regardless of what shell is run
- *$HOME/.login* and */etc/csh.login (csh)* and *$HOME/.profile* and /etc/profile (ksh and bsh) get executed only at login
- you can force at any moment your current session to read the configuration file

read configuration file

```
source ~/.profile
```


Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

ROOT - start

✔ **root command help**

```
> root --help # get help

Usage: root [-l] [-b] [-n] [-q] [dir] [[file:]data.root] [file1.C ... fileN.C]
Options:
  -b : run in batch mode without graphics
  -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
  -x : exit on exception
 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
```

✔ **start root**

```
> root -l
```

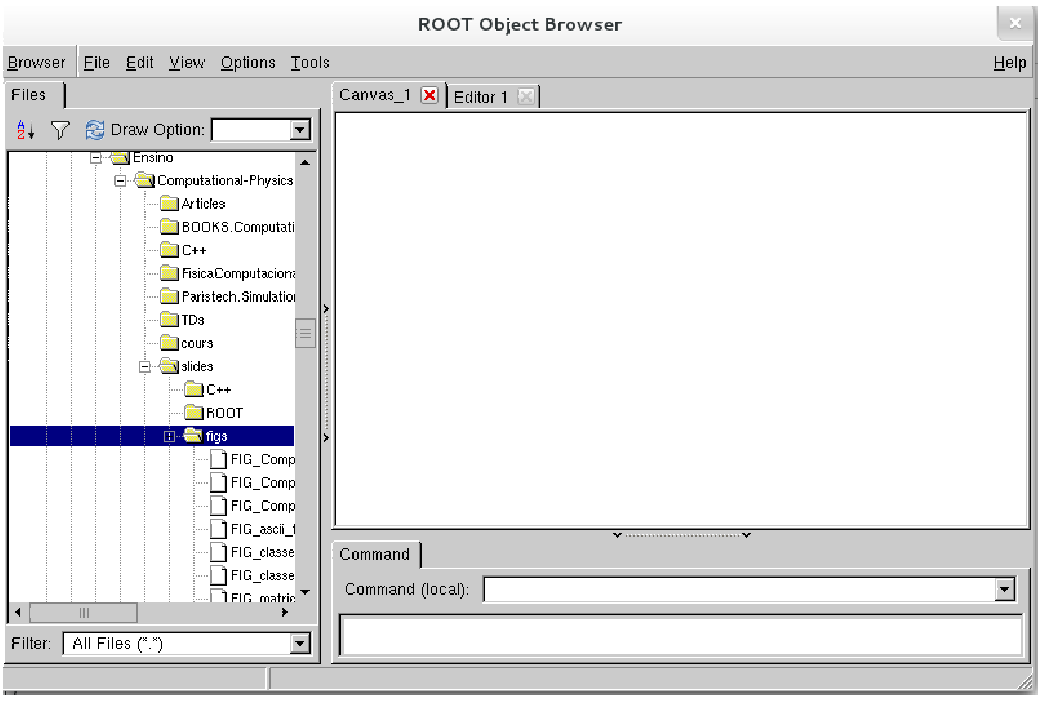
✔ **quit root**

```
> .q
```


ROOT - browser

Your directories and ROOT files (root objects) can be browsed by instantiating the *TBrowser* class

```
[0] TBrowser *b = new TBrowser()
```



ROOT - init

Reset all ROOT parameters before running any C++ macro and define the graphics options

```
[0] gROOT->Reset();
[1] gROOT->SetStyle("Plain");
[2] gStyle->SetOptStat(1111); // =0 to reset
[3] gStyle->SetOptTitle(0); // supress title box
[4] gStyle->SetOptFit(1111); // print fit results
[5] gStyle->SetPalette(1); // better than default
```


ROOT - running macros

A C++ function is known as a macro in ROOT. Let's make a macro that we name *hadd* for adding two histograms.

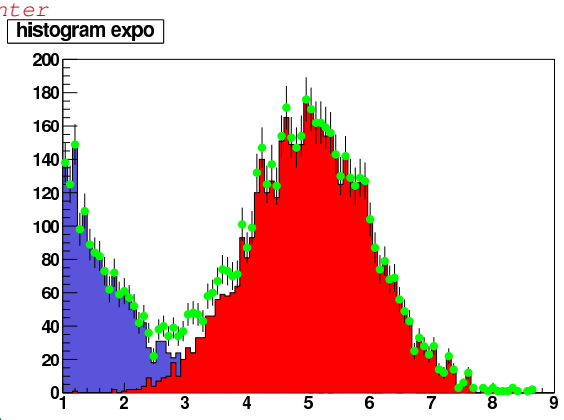
hadd.C

```
void hadd() {
  gROOT->SetStyle("Plain");
  gStyle->SetOptStat(0);
  TCanvas *c = new TCanvas();

  TH1F *hg = new TH1F("hg","histogram gauss", 100,1.,9.);
  for (int i=0; i<5000; i++) {hg->Fill(gRandom->Gaus(5,1.));}
  TH1F *he = new TH1F("he","histogram expo", 100,1.,9.);
  for (int i=0; i<5000; i++) {he->Fill(gRandom->Exp(1.));}

  TH1F *hsum = new TH1F(*hg); //dereference hg pointer
  hsum->Add(he,1.);

  he->GetYaxis()->SetRangeUser(0.,200.);
  he->SetFillColor(9);
  he->DrawCopy();
  hg->SetFillColor(kRed);
  hg->DrawCopy("same");
  hsum->SetMarkerStyle(20);
  hsum->SetMarkerColor(3);
  hsum->SetMarkerSize(1.2);
  hsum->Draw("Esame"); //draw with errors
}
```



ROOT - running macros (cont.)

The macro can be run at the unix prompt:

```
> root -l hadd.C
```

It can also be run with the CINT interpreter:

```
> root -l
root [0] .x hadd.C
```

It can also be loadded in CINT interpreter:

```
> root -l
root [0] .L hadd.C
root [1] hadd()
```

Running macro *hadd.C* within a macro

```
{
 gROOT->LoadMacro("hadd.C");
 hadd(); // calling function
}
```


ROOT - running macros (cont.)

Macros can be compiled with ACLIC (automatic compiler of libraries for CINT)
The compiled code runs much faster and language error checks easier!
The compilation process produces a shared library (.so) that can be used in ROOT
The shared library must be loaded before using user functions or classes

```
> root -l
root [0] .L hadd.C+
root [1] gSystem->Load("hadd_C.so")
root [2] hadd()
```

Notice that the include files of all functions being used in the code have to be added to the macro

```
#include "TROOT.h"
#include "TCanvas.h"
#include "TH1F.h"
#include "TRandom.h" //gRandom
#include "TStyle.h" //gStyle
void hadd() {
 ...
}
```

ROOT - running macros (cont.)

Remarks:

- the directory path name cannot be used with *.L*. In case of need replace it by:

```
root [0] gSystem->cd("directory_path")
root [0] gSystem->CompileMacro("hadd.C")
```

- If include files belonging to specific directories need to be include add their dir path through:

```
root [0] .include "-I$HOME/dir_path"

or within a macro:

gROOT->ProcessLine(".include my/include/dir");

or still:

gSystem->AddIncludePath(" -I/my/include/dir");
```