Breve introducción a Linux

GNU / Linux

Es un sistema operativo de fuente libre y gratuito

En el entorno científico y en particular la física de partículas Linux es mayoritario

Scientific Linux CERN (SLC): http://linux.web.cern.ch/linux/scientific5/

Existen aplicaciones que permiten ejecutar Linux dentro de Windows
(Sin necesidad de crear particiones de disco)

VMWare: emulación completa de Linux
http://www.vwmare.com

Cygwin: entorno de aplicaciones Linux *exportadas* a windows
http://www.cygwin.com/

Portable Ubuntu: corre Ubuntu desde Windows y es fácilmente portable

Linux: Usuarios

Usuarios:

Linux es un sistema multi-usuario

Cada usuario posee unos permisos concretos (lectura, escritura, ejecución...)

El usuario **root** puede hacer (casi) todo

usuario: alumno
pwd: alumnos-2012

Grupos:

Cada usuario pertenece, al menos, a un grupo

Los permisos se pueden regular a nivel usuario

Linux: Usuarios

Terminología:

Ficheros = archivos = documentos
Directorios = carpetas

Cada usuario posee unos permisos concretos (lectura, escritura, ejecución...)

Todos los ficheros pertenecen a algún usuario, que controla su visibilidad para el resto de usuarios / grupos.

Directorios especiales:

¡pruébame!

. → Directorio actual

.. → Directorio superior

/ → Directorio raíz

~ → Directorio de usuario /home/alumno/

Subdirectorios:

El camino hasta un directorio se construye encandenando los directorios intermedios separados por "/":
/el/camino/a/mi/directorio

Linux: Comandos

Los comandos se ejecutan tecleando su nombre y pulsando la tecla **`enter`**

Los comandos aceptan opciones y argumentos:

```
>> comando --opción

>> comando -o

>> comando --opción=bla

>> comando --o=bla


>> man comando

>> comando --help
```

¡pruébame!

Linux: Directorios

Crear un directorio

```
>> mkdir nombrecarpeta
```

Borrar un directorio (vacío)

```
>> rmdir nombrecarpeta
```

Moverse a un directorio

```
>> cd nombrecarpeta
```

Averiguar cuál es mi directorio actual

```
>> pwd
```

Linux: Ficheros

Obtener la lista de ficheros en el directorio actual

```
>> ls [directorio|fichero]
```

Opciones:

```
>> ls -l: Con detalles
>> ls -a: Todos (incluido ocultos)
>> ls -t: Ordenar por fecha de modificación
>> ls -r: Ordenar inversa
>> ls -rhrt: combinando
```

Borrar un fichero(directorio)

```
>> rm (-r) fichero
```

Renombrar/Mover un fichero

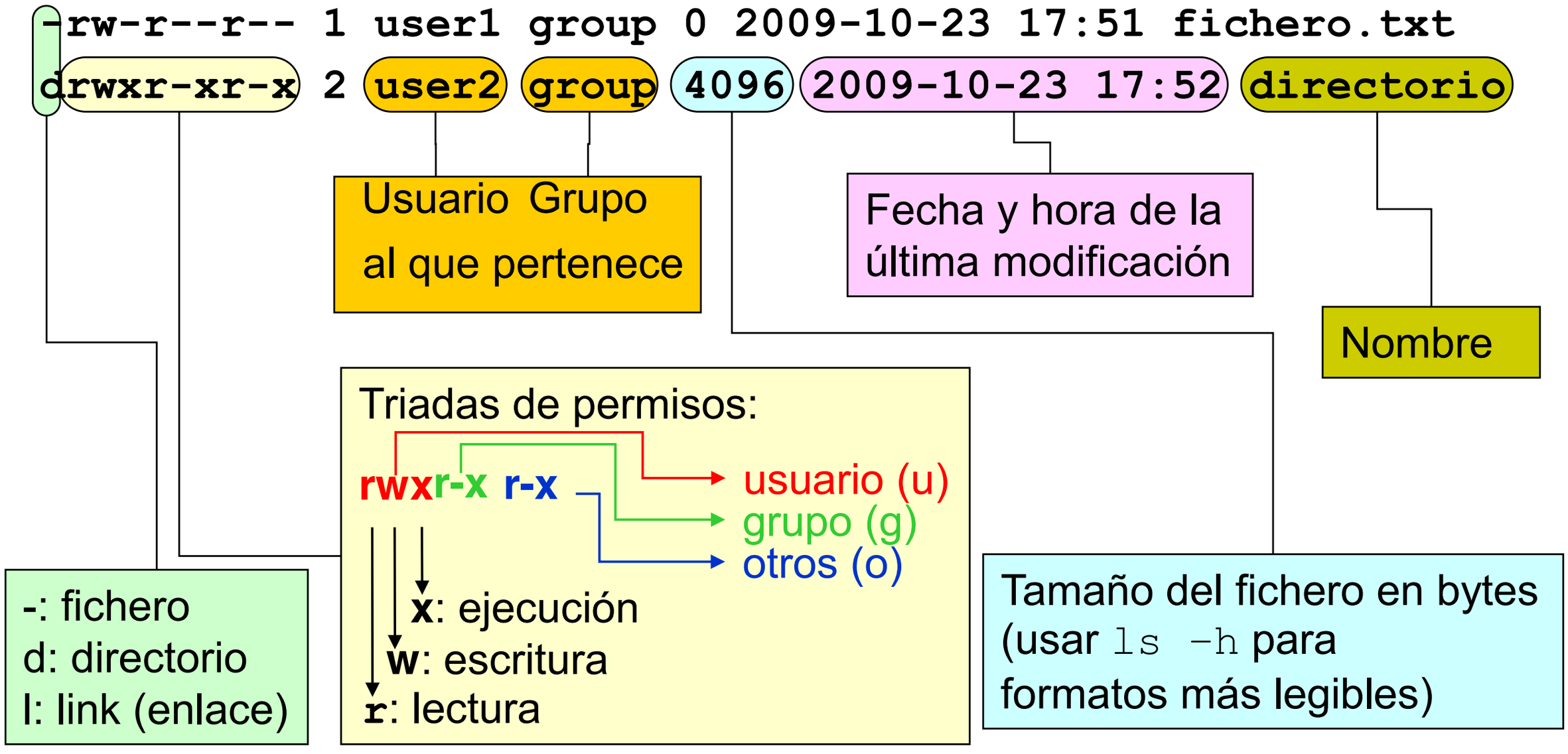
```
>> mv ficherooriginal ficherofinal
```

Copiar un (directorio) fichero:

```
>> cp (-r) fichero(directorio)
```

Linux: Permisos

Al hacer un listado largo de un fichero (ls -l) o directorio (ls -ld) obtenemos



>> chmod [u|g|o|a] [+|-] [r|w|x] fichero

C++ / ROOT

C++

Es un lenguaje de programación, diseñado a mediados de los años 80

Es una extensión (o un superconjunto) del lenguaje C

Incluye mecanismos que permiten la manipulación de objetos

Por tanto, C++ es un lenguaje híbrido

por su capacidad de hacer programación estructurada (al igual que C)

y programación orientada a los objetos (Object Oriented Programming)

Programación Orientada a Objetos: Clase y Objeto

Objeto:

Un objeto es algo de lo que hablamos y podemos manipular

Por ejemplo, **mi coche rojo**. Yo puedo pintarlo de amarillo, cambiarle las ruedas, etc

Clase:

Una clase describe los objetos de un mismo tipo. Esto incluye las propiedades y el comportamiento de los objetos

Todos los objetos son instancias de una clase

En el ejemplo, tenemos el objeto **mi coche rojo**, y la clase **coches**

Los coches tienen varios atributos, entre ellos el color, la potencia, el peso, etc. Mi coche es de color amarillo, tiene 100 CV y pesa 1,200 kg

Programación Orientada a Objetos: Herencia

La herencia es una relación característica de la OOP

Puede expresar tanto especialización como generalización

Evita definir múltiples veces características comunes a varias clases

Ejemplo, vehículos de transporte por carretera

Todos tienen ruedas, motor, asientos, ...

Clases que heredan: motos, coches, camiones, etc

Las características comunes sólo de definen una vez. Luego, en las clases más especializadas, se añaden atributos particulares. En los camiones, por ejemplo, capacidad de carga, tipo de mercancía, etc

C++: Tipos fundamentales

Nombre	¿Qué representa?	Entero, real, lógico
`char`	**Un** carácter	**Entero**
`short int`	**Un** entero **corto**	**Entero**
`int`	**Un** entero	**Entero**
`long int`	**Un** entero **con** mayor rango **de validez**	**Entero**
`float`	**Un** real	**Real**
`double`	**Un** real **de** doble precisión	**Real**
`long double`	**Un** real **de** doble precisión **y** mayor	**Real**
`bool`	**cierto (`true`) o falso (`false`)**	**Lógico**

C++: Operadores Aritméticos

Operador	Función	Uso
=	**asignación**	`int i = 7;`
*	**multiplicación**	`double r = 3.5 * i;`
/	**división**	`double t = r / 1.2;`
%	**modulo (resto)**	`i = 21 % 6; // i = 3`
+	**suma**	`double rt = r + t;`
-	**resta**	`r = rt - t;`

++,--	**incremento**	`int i = 0;`
		`int j = i++; //j = 0`
		`int k = --i; //k = 0`

+=,-=,
*=,/=
	opera y asigna	`r += 2.6; //r = r + 2.6`
		`r *= 2.6; //r = r * 2.6`

C++: Operadores Lógicos

Operador	Función	Uso
<	**menor que**	`i < 5`
<=	**menor o igual**	`r <= 5`
>	**mayor que**	`i > 5`
>=	**mayor o igual**	`i >= 5`
==	**igualdad**	`i == 5`
!=	**desigualdad**	`i != 5`
!	**NOT lógico**	`!true; //false`
&&	**Y lógico**	`i < 5 && j > 4`
||	**O lógico**	`i < 5 || j > 4`

C++: if

```
if (condición 1) {
    printf(“Se cumple la condición 1\n”);
} else if (condición 2) {
    printf(“Se cumple la condición 2\n”);
} else {
    printf(“No se cumple ninguna condición\n”);
}
```

C++: for

```
for (int i=0; i<5; i++) {
    printf("%d\n", i);
}
```

El lazo **for** declara (int i) e inicializa un contador (i=0), pone un final (i < 10) y nos indica el tamaño de cada paso (i++). En el ejemplo, tendremos:

0

1

2

3

4

ROOT

http://root.cern.ch

ROOT es un entorno Orientado a Objetos programado en C++ para el desarrollo de aplicaciones que manejen grandes volúmenes de datos

Proporciona, entre otras cosas:

- Un intérprete de C++ (CINT)
- Algoritmos de análisis estadístico avanzados
- Herramientas de visualización científica con gráficos 2D y 3D
- Un avanzado interfaz de usuario gráfico

En nuestro caso particular (PCs de las prácticas), cargamos el entorno ROOT con el comando linux:

```
source /opt/root/bin/thisroot.sh
```

ROOT: ¿Compilar o Interpretar?

ROOT permite ambas opciones

Compilar

- Pros: ejecución más rápida, soporta todo C++
- Cons: compilar lleva tiempo y es más estricto con el lenguaje

Interpretar

- Pros: ideal para pruebas rápidas
- Cons: ejecución más lenta, sobre todo en programas grandes

ROOT: Ejecutando código

root -l mycode.C

root -l

root[0] .x mycode.C

root -l

root[0] .L mycode.C

root[1] mycode()

3 formas equivalentes

ROOT interpretado

---

root -l mycode.C+

root -l

root[0] .x mycode.C+

root -l

root[0] .L mycode.C+

3 formas equivalentes

ROOT compilado

ROOT: Tutorial

Contenido

Introducción práctica a ROOT

- Iniciando ROOT
- Funciones
- Macros
- Histogramas
- Ficheros
- Trees
- TBrowser
- Crear clases de ROOT
- Debugging

Nomeclatura:

Azul: lo que tecleas

Rojo: lo que obtienes

Histogramas y macros de ejemplo en:
http://www.hep.uniovi.es/jfernan/FAEyA/PL.tgz tar xzf PL.tgz

primeros pasos en ROOT

Iniciando ROOT

$ root (-l)

La consola de ROOT

root[ ] 2 +3

root[ ] log(5)

root[ ] int i = 42

root[ ] printf("%d\n", i)

Historia

Con las flechas de dirección

Buscar con CTRL-R

Ayuda en linea

root[ ] new TF1 (<TAB>

TF1 TF1()

TF1 TF1(const char* name, const char* formula, Double_t xmin=0, Double_t x max=1)

primeros pasos en ROOT (2)

Comandos multi-línea

```
$ root (-l)
```

Comandos multilínea

```
root[ ] for (i=0; i<3; i++)
            printf("%d\n", i)
```

o

```
root[ ] for (i=0; i<3; i++) {
end with '}', '@': abort >
     printf("%d\n", i)
end with '}', '@': abort > }
```

Cuando algo sale mal...

```
root[ ] printf("%d\n", i)
end with '}', '@': abort >  @
```

Don't panic!
Don't press CTRL-C!
Just type @

Macros

Combina líneas de código en macros

Macro sin nombre (sin parámetros, macro1.C)

```
{
    for (Int_t i=0; i<3; i++)
      printf("%d\n",i);
}
```

Ejecutando macros:

root[ ] .x macro1.C

$ root -l macro1.C

$ root -l -b macro1.C (sin entorno gráfico)

$ root -l -q macro1.C (salir después de ejecutar)

Data types in ROOT
Int_t (4 Bytes)
Long64_t (8 Bytes)
…
to achieve platform-independency

Macros (2)

Macro con nombre (puede tener parámetros, macro2.C)

```
void macro2(Int_t max = 10)
{
   for (Int_t i=0; i<max; i++)
     printf("%d\n",i);

}
```

No olvides cambiar el nombre de la función después de re-nombrar la macro.

Ejecutando macros:

```
root[ ] .x macro2.C(12)
```

Cargando macros

```
root[ ] .L macro2.C
root[ ] macro2(12)
```

Figuras para trabajos
Es muy útil tener el código para crear las gráficas en una macro. Nunca crees las gráficas "finales" utilizando el modo consola (lo tendras que hacer una y otra vez)

Macros o modo en linea

Utiliza la consola para probar lineas sueltas mientras escribes tu código
Pon el código que vayas a re-utilizar en macros.

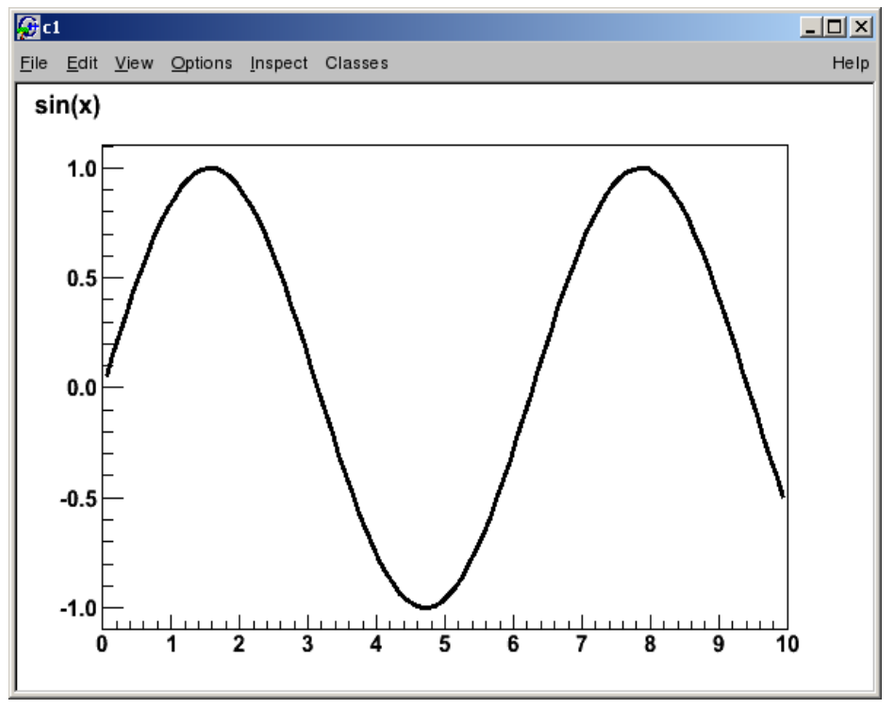
Funciones

La clase TF1 permite pintar funciones:

root[ ] f = new TF1("func", "sin(x)", 0, 10)

- **“func”** es un nombre único
- **“sin(x)”** es la fórmula
- **0, 10** es el rango x de la función

root[ ] f->Draw()



El estilo de la función puede cambiarse en la línea de comando con el menú.:

root[ ] f->SetLineColor(kRed)

La clase TF2(3) es para grá cas de 2 y 3 dimensiones

http://root.cern.ch/root/html/TF1.html

Histogramas

Contiene datos bineados - probablemente la clase más utilizada en ROOT por los físicos.

Crear un TH1F (= uni-dimensional, precisión float)

root[ ] h = new TH1F("hist", "my hist; Bins;Entries",10, 0, 10)

- **“hist”** es un nombre único
- **“my hist; Bins; Entries”** son el título y las etiquetas del eje x, y.
- **10** es el número de bines
- **0, 10** es el rango x de la función

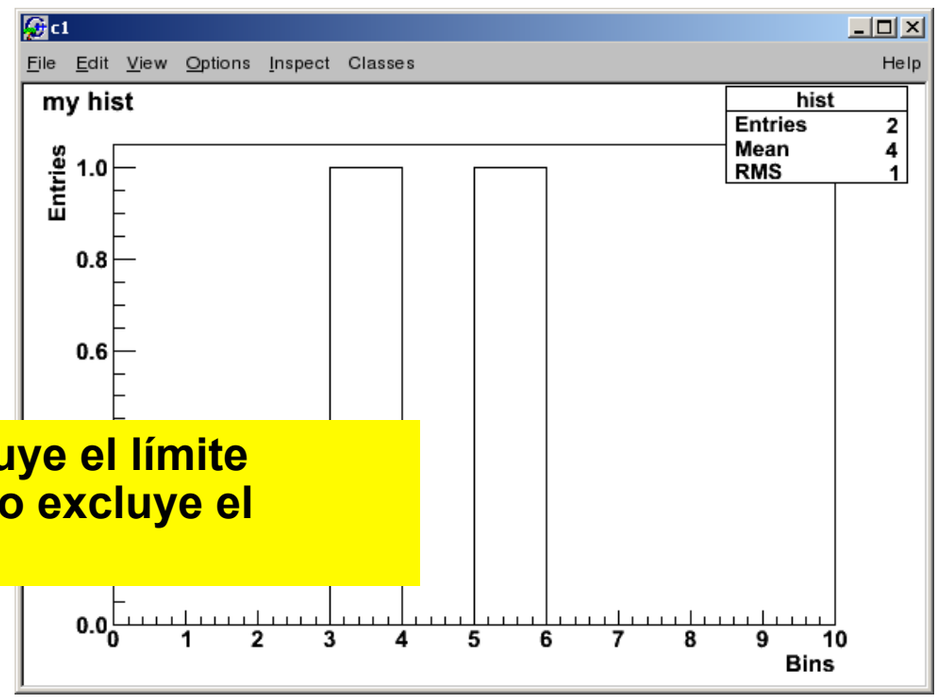
Llenar el histograma

root[ ] h->Fill(3.5)

root[ ] h->Fill(5.5)

Pintar el histograma

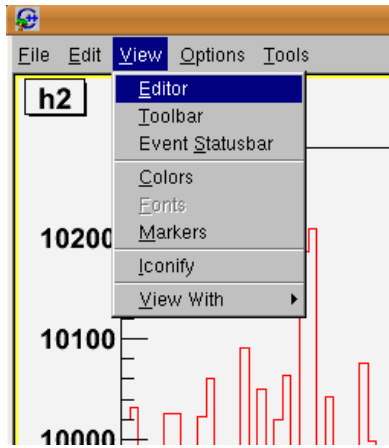
root[ ] h->Draw()



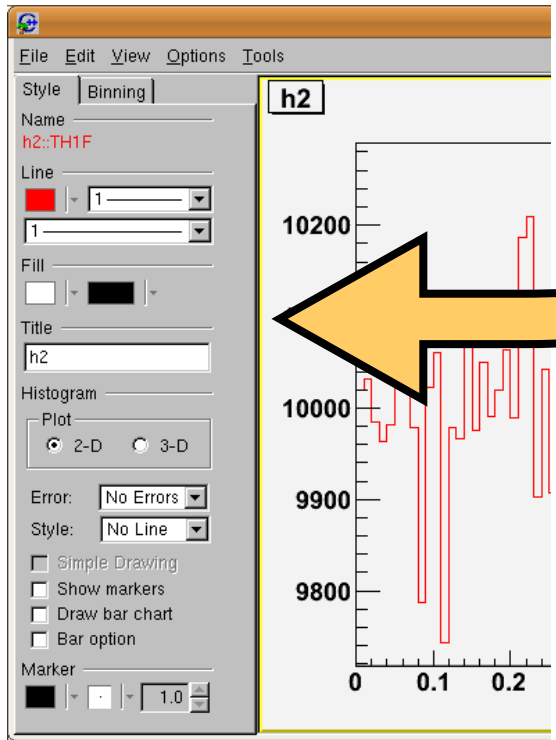
Un bin incluye el límite inferior pero excluye el superior.

Histogramas - Interactividad

- Ir al menu View → Editor



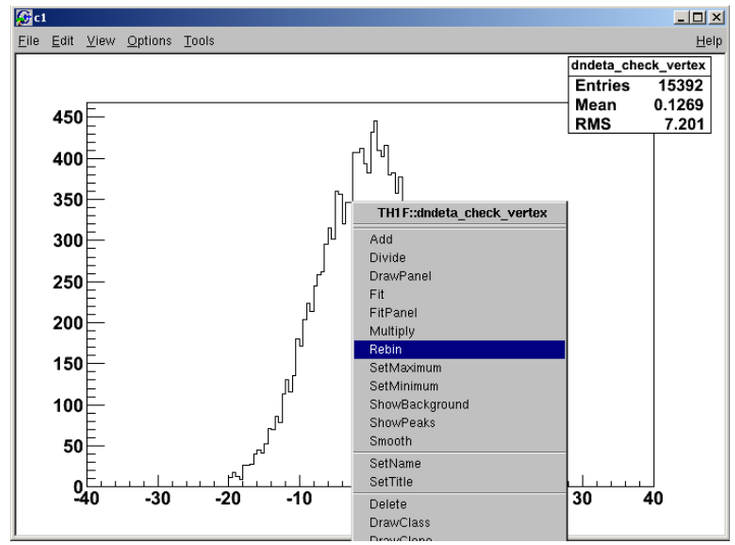
- Pinchar en el histograma
  - Podemos modificar un montón de parámetros
- Pinchar en el eje
  - Renombrarlo
- …



Histogramas (2)

Rebinear

root[ ] h->Rebin(2)



Cambiar los rangos

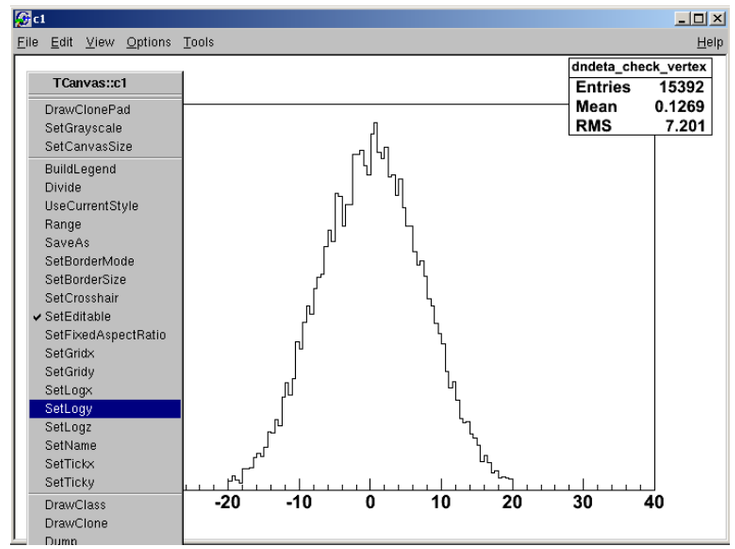
- Con el ratón,
- Con el menú contextual
- Con la línea de comando

root[ ] h->GetXaxis()->SetRangeUser(2,5)

En escala logarítmica

- botón derecho en el área blanca en el lado del canvas y selecionar “SetLogx”
- linea de comandos:

root[ ] gPad->SetLogy()

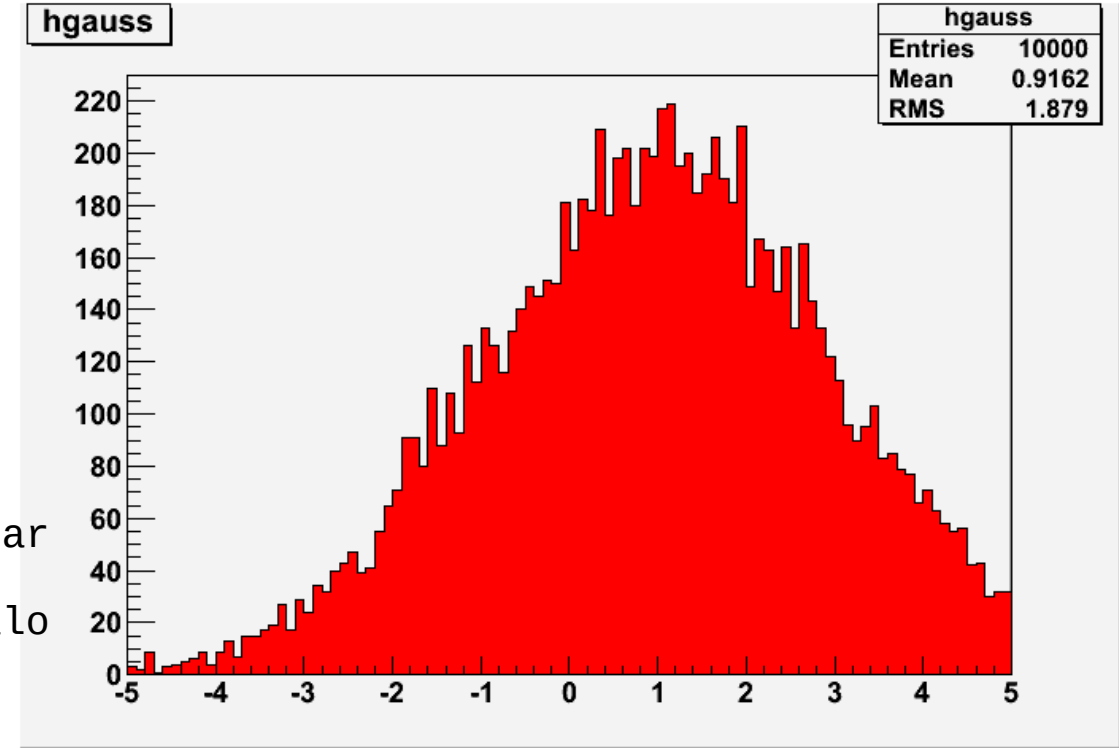


Nota: histograma de ejemplo está en hist.root

http://root.cern.ch/root/html/TH1.html

Histogramas - Ejercicio

- Hacer una macro que llene un histograma con una distribución aleatoria de forma gaussiana
  - Media=1
  - Varianza=2
  - Intervalo (-5,5)

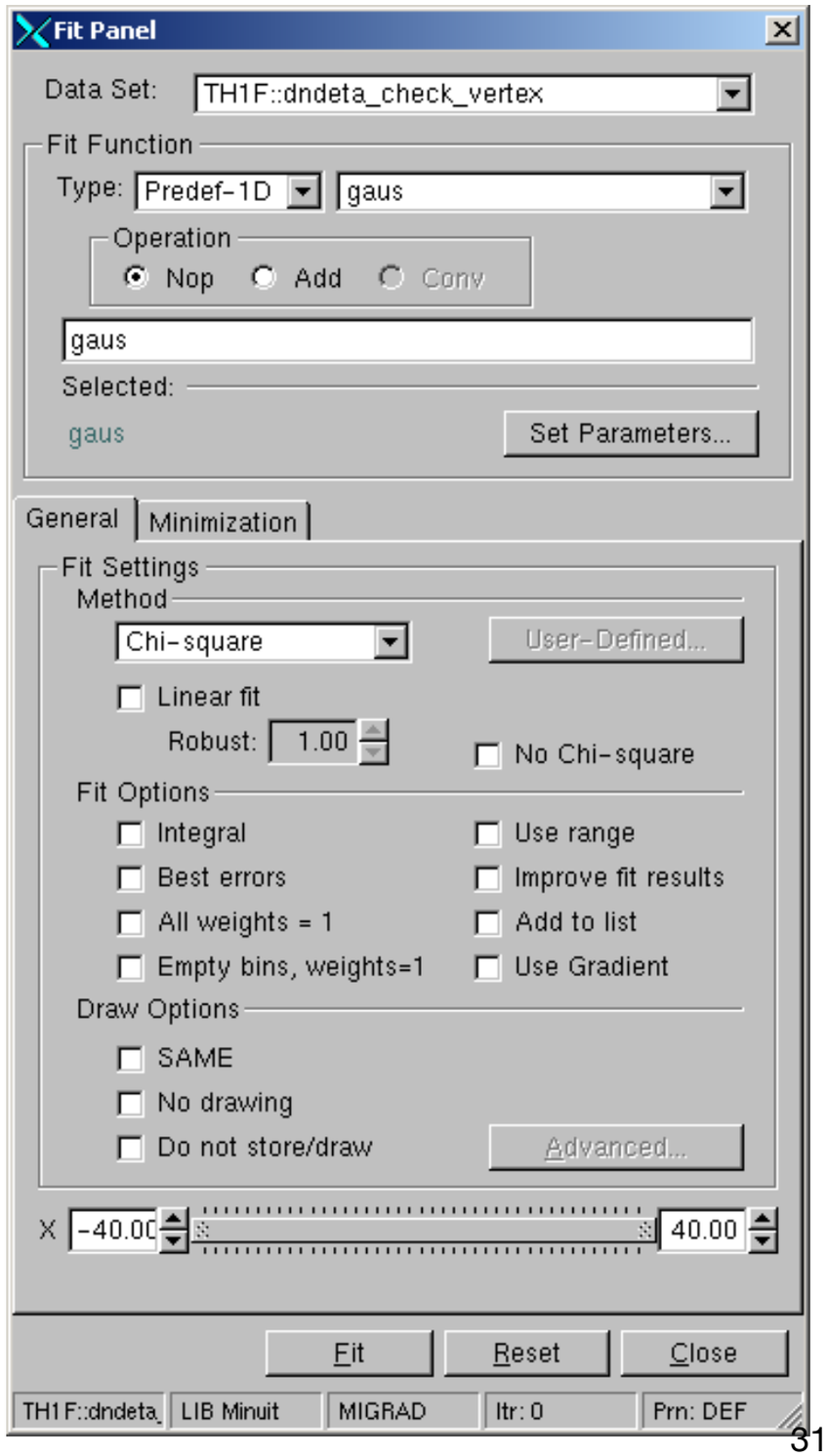



```
Ayuda:
- hay que crear una funcion
gaussiana de variable x
- gaussiana en TF1 es "gaus"
y hay que meter parametros
con SetParameter(...)
- usamos la funcion para generar
valores con a esa distribucion
- llenar un TH1F en el intervalo
con el metodo FillRandom
```

Ajuste de Histogramas

Interactivo

- Botón derecho y elegir “fit panel”
- Seleccionar la función y click en fit
- Ajustar parámetros
(mostrados en línea de comandos)
(en el canvas)



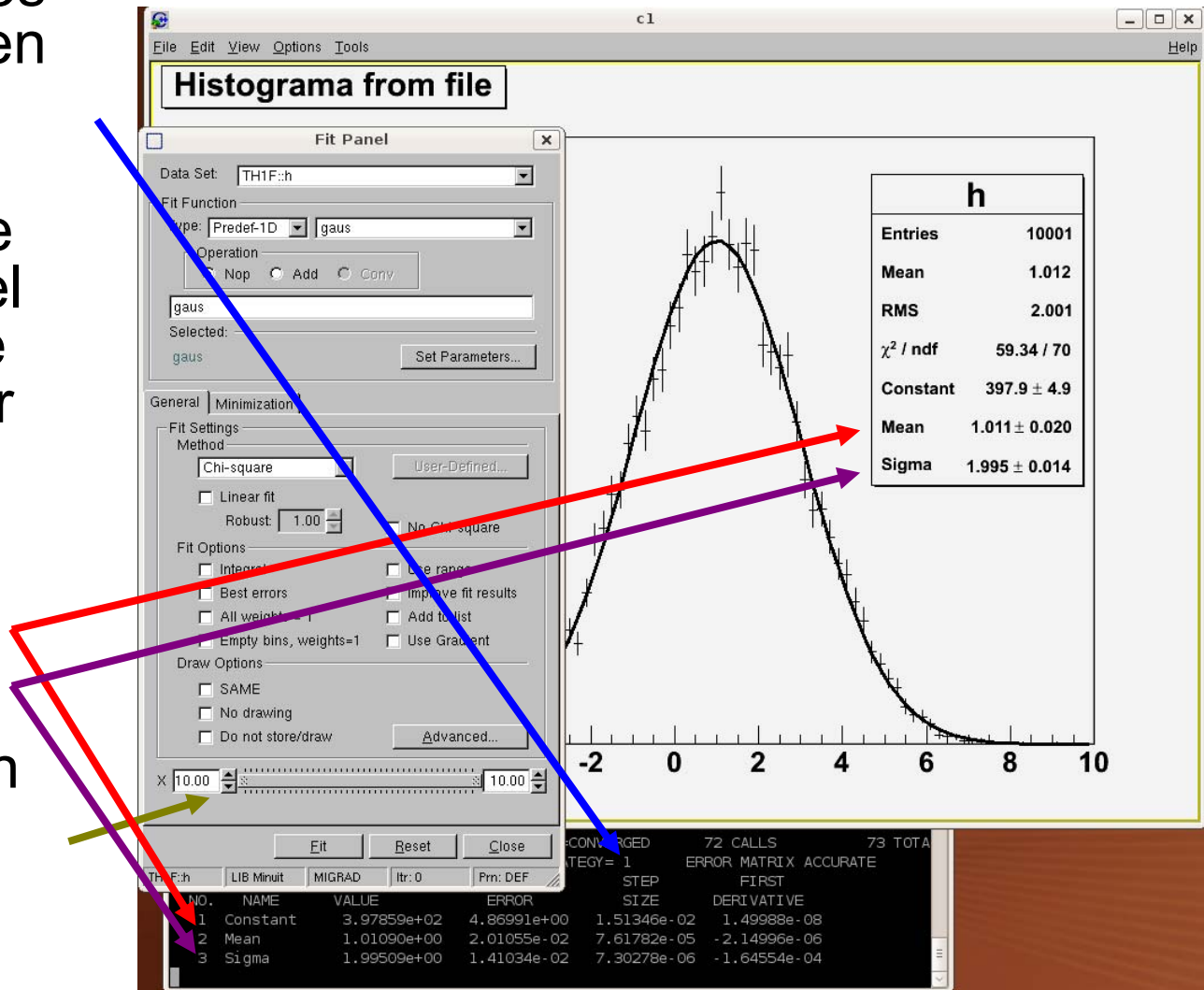
Línea de comando

root[ ] h->Fit("gaus")

- Otras funciones pre-definidas polN, expo, landau

Ajustes

- Los parámetros del ajuste salen en el texto
- Con el botón derecho sobre el resumén del histograma se pueden añadir también ahí (`SetOptFit`)
  - Media: 1.0
  - Sigma: 2.0
- Podemos ajustar solo en un intervalo

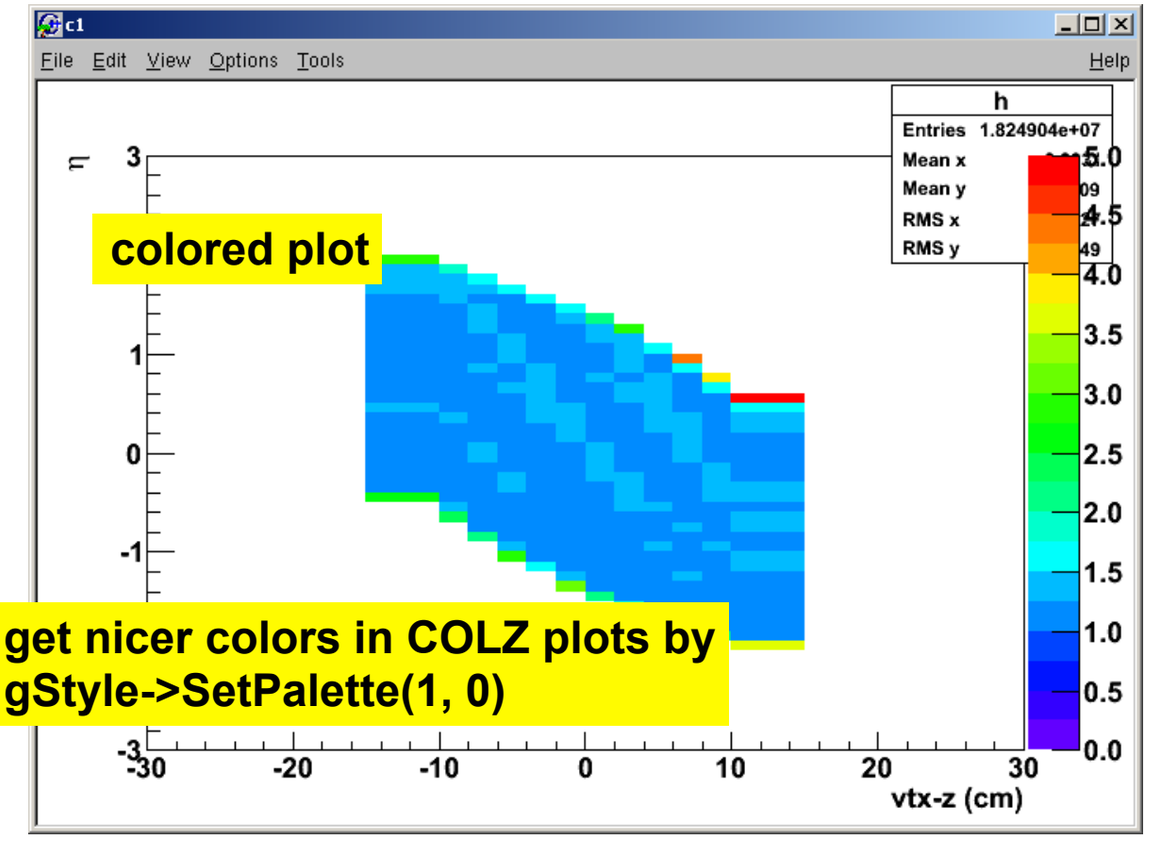


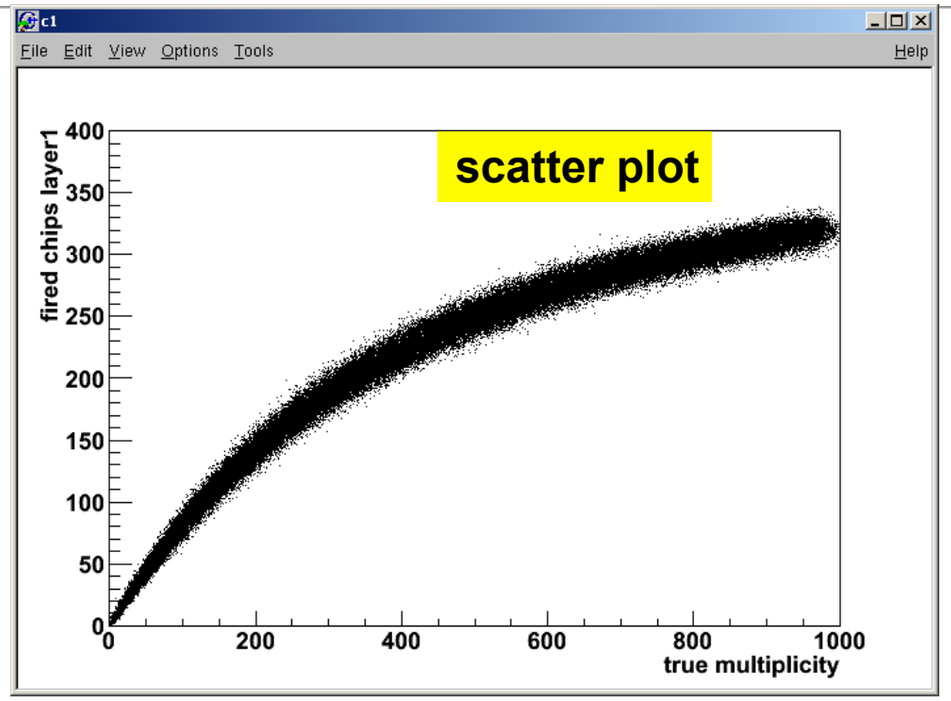
Histogramas 2D

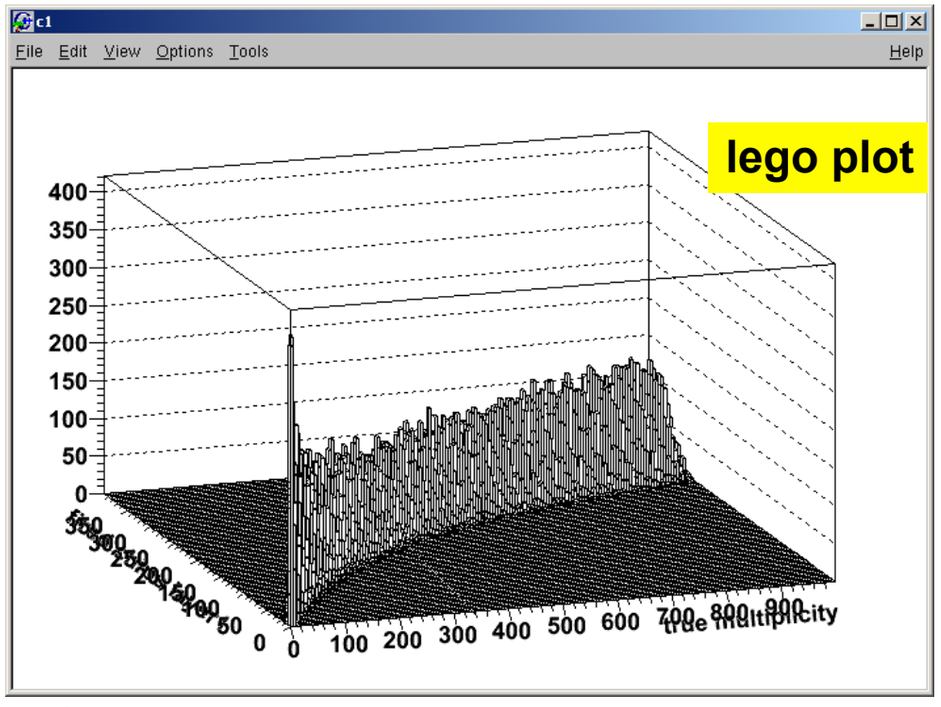
root [ ] h->Draw()

root [ ] h->Draw("LEGO")

root [ ] h2->Draw("COLZ")







Nota: histogramas de ejemplo está en hist2.root

Ficheros

La clase TFile permite guardar cualquier objeto ROOT en el disco.

Crear un histograma como antes:

root[ ] h = new TH1F("hist", "my hist; Bins;Entries",10, 0, 10)

"hist" will be the name in the file

Abrir un fichero para escribir.

root[ ] file = TFile::Open("file.root", "RECREATE")

Escribir un objeto en el fichero

root[ ] h->Write()

Cerrar el fichero

root[ ] h->Close()

`http://root.cern.ch/root/html/TFile.html`

Ficheros (2)

Abrir el fichero para leer:

root[ ] file = TFile::Open("file.root")

Leer un objeto del fichero

root[ ] hist->Draw()

Leer el objeto en **una** macro:

```
TH1F* h = 0;
file>GetObjet("hist",h);
```

¿Qué mas hay en fichero?

root[ ] .ls

Abrir un fichero al iniciar root

$ root hist.root

Acceder al fichero usando _file0

Object ownership
After reading an object from a file don't close it! Otherwise your object is not in memory anymore

Ficheros (3)

Entrar en root y abrir todos los ficheros del directorio para leer:

```
$ root *.root
root [0]
```

```
Attaching file chain.root as _file0...
Attaching file hist.root as _file1...
Attaching file hist2.root as _file2...
Attaching file tree.root as _file3...
Attaching file tree2.root as _file4...
```

Cambiar un fichero en particular

```
root[ ] _file1->cd()
```

¿Qué hay en el fichero?

```
root[ ] .ls
```

Dueño del objeto

Después de leer un objeto de un fichero, ¡no lo cierres! Si lo haces el objeto no estará en memoria nunca más.

Estructuras de datos en ROOT: TTrees

- Los `TTree's` han sido diseñados para soportar colecciones muy largas de objetos.
- Permiten acceso directo y aleatorio a cualquier entrada del árbol (aunque el acceso secuencial va mejor)
- Los `TTree's` tienen ramas (branches) y hojas (leaves). Se puede leer un subconjunto de las ramas.
- Hay métodos de alto nivel como `TTree::Draw` que iteran sobre las entradas con expresiones de selección.
- Se pueden ver los `TTree's` usando `TBrowser`
- Se pueden analizar los `TTree's` usando `TTreeViewer`

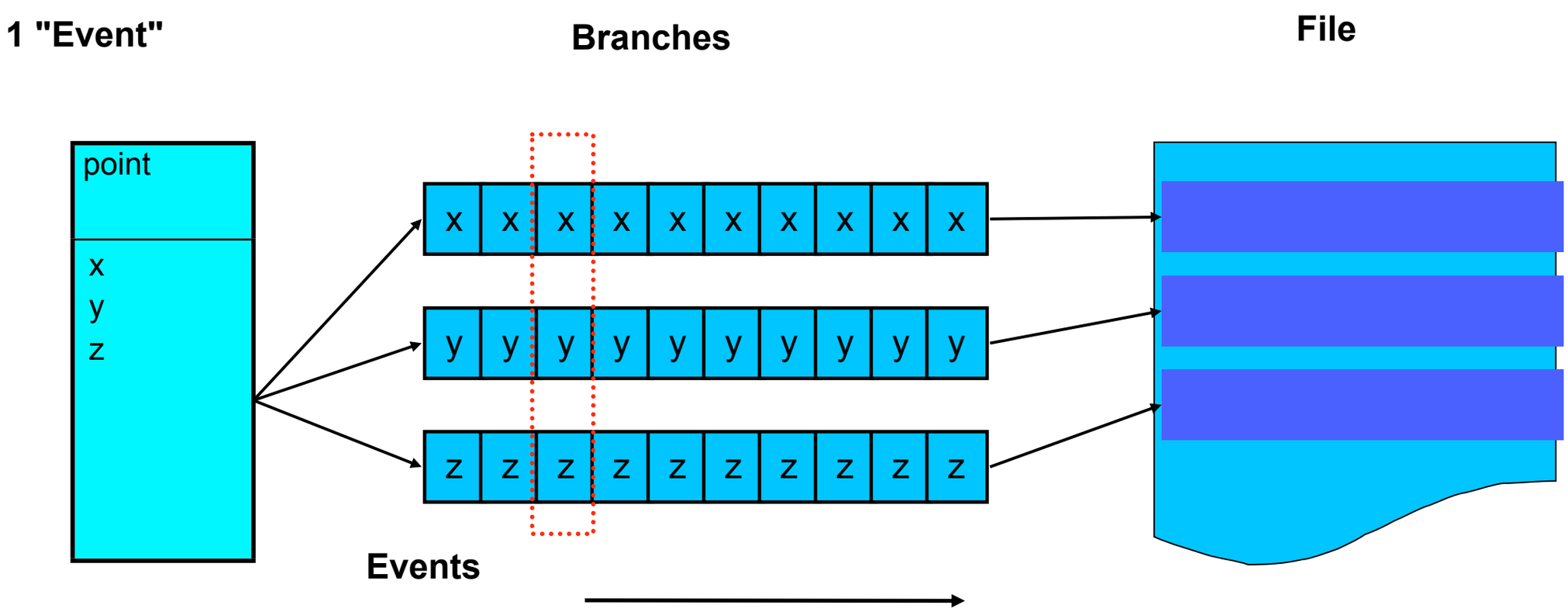
Trees

La clase TTree es el principal contenedor para guardar datos:

- Puede guardar cualquier clase o tipo básico

- Cuando lees un árbol, algunas ramas pueden desactivarse (para agilizar el análisis)

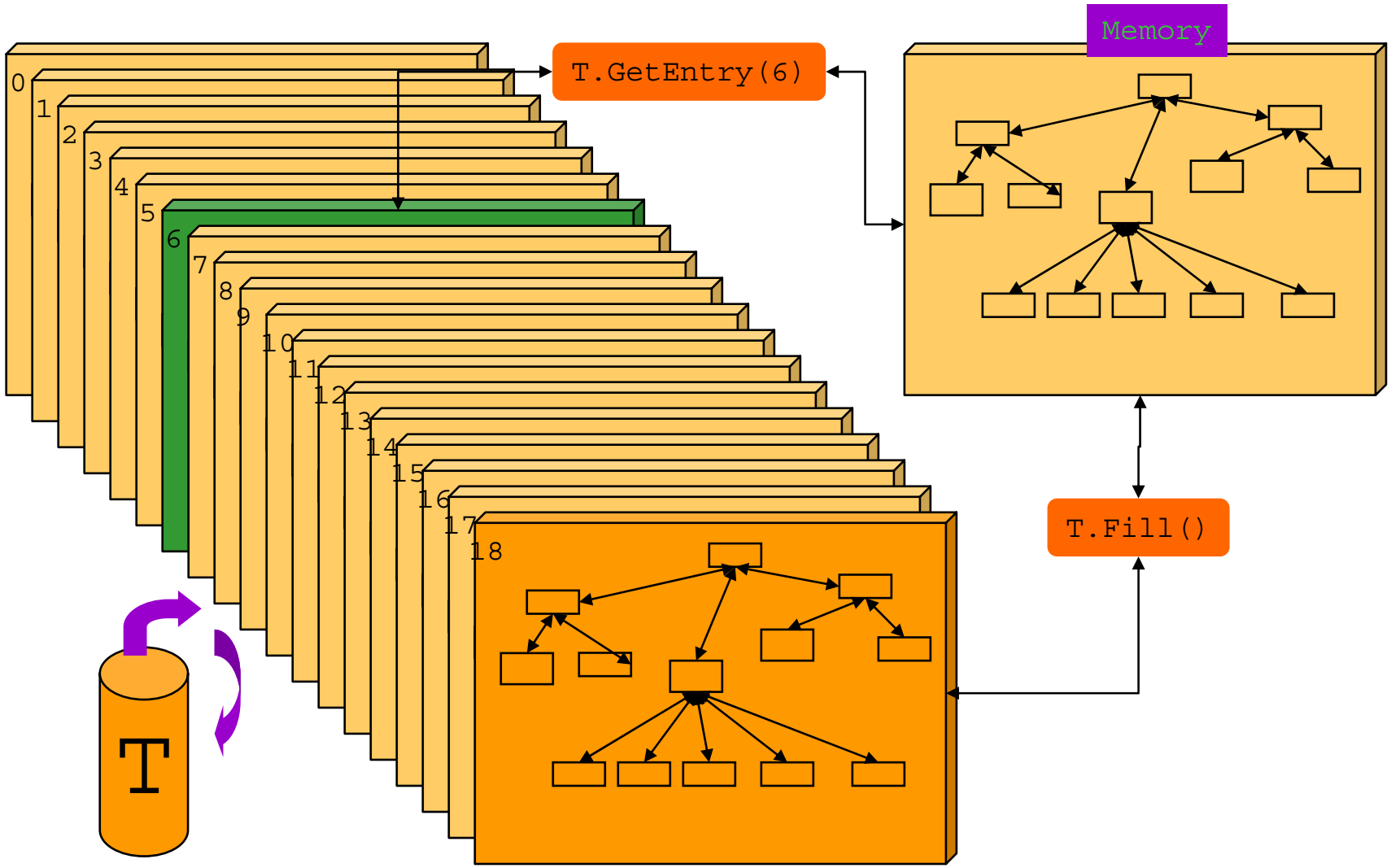
http://root.cern.ch/root/html/TTree.html



Estructura de un TTree

- Cada fichero de datos es una colección de N sucesos
- Cada suceso tiene estructura de árbol con varias ramas
  - Cada rama puede contener un valor (ej. Número de sucesos)…
  - … o una colección de valores (momento transverso de los jets)
- La estructura de cada suceso puede ser muy compleja (RECO, PAT) o plana (Flat Trees)

Contenido de un TTree



Trees (2)

Acceder a un árbol que contiene clases

- Los miembros son accesibles sin la librería apropiada.

Ejemplo: tree.root (con magnitudes cinemáticas de ALICE)

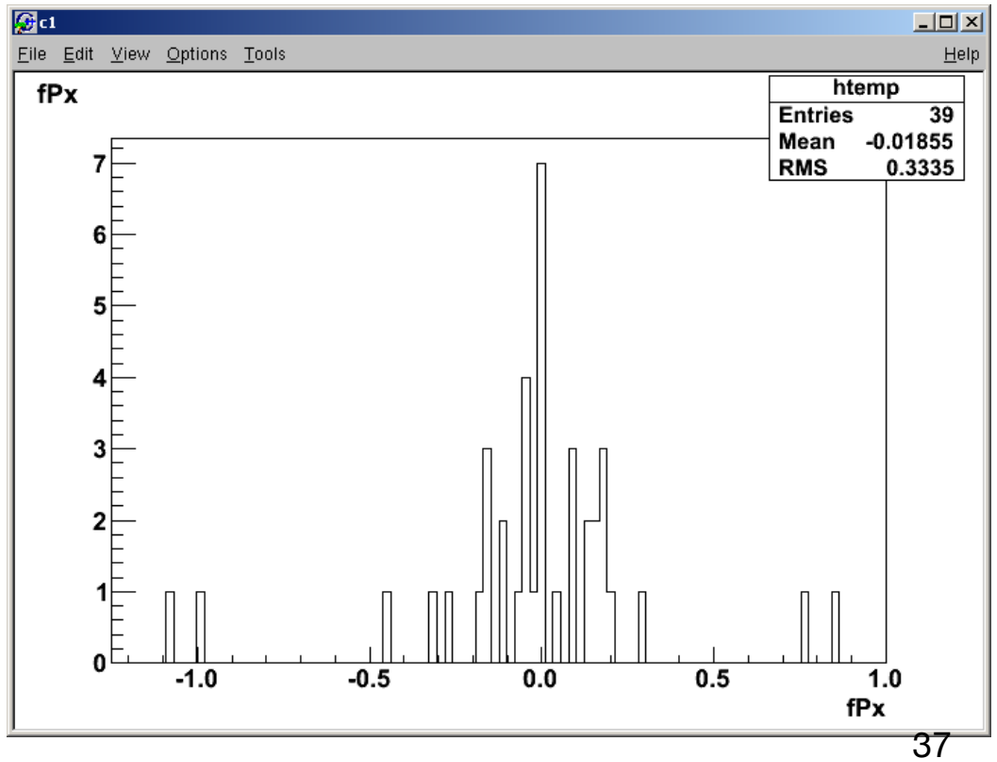
$ root tree.root

root[ ] tree->Draw(“fPx”)

root[ ] tree->Draw(“fPx”, “fPx < 0”)

root[ ] tree->Draw(“fPx”, “abs(fPdgCode) == 211”)

El Tree contiene TParticles.



Trees (3)

Conectar una clase con un árbol:

```
root[ ] TParticle* particle = 0
root[ ] tree->SetBranchAddress("Particles",&particle)
```

Leer una entrada

```
root[ ] tree->GetEntry(0)
root[ ] particle->Print()
root[ ] tree->GetEntry(1)
root[ ] particle->Print()
```

The content of the TParticle instance is replaced with the current entry of the tree

Estos comandos puede usarse en un lazo

```
root [5] particle->Print()
TParticle: pi0                  p: -0.036864 -0.0
```

TBrowser

El TBrowser puede usarse:

- Abrir ficheros
- Navegar por ellos
- Mirar a los TTrees

Iniciar un TBrowser

root[ ] TBrowser t

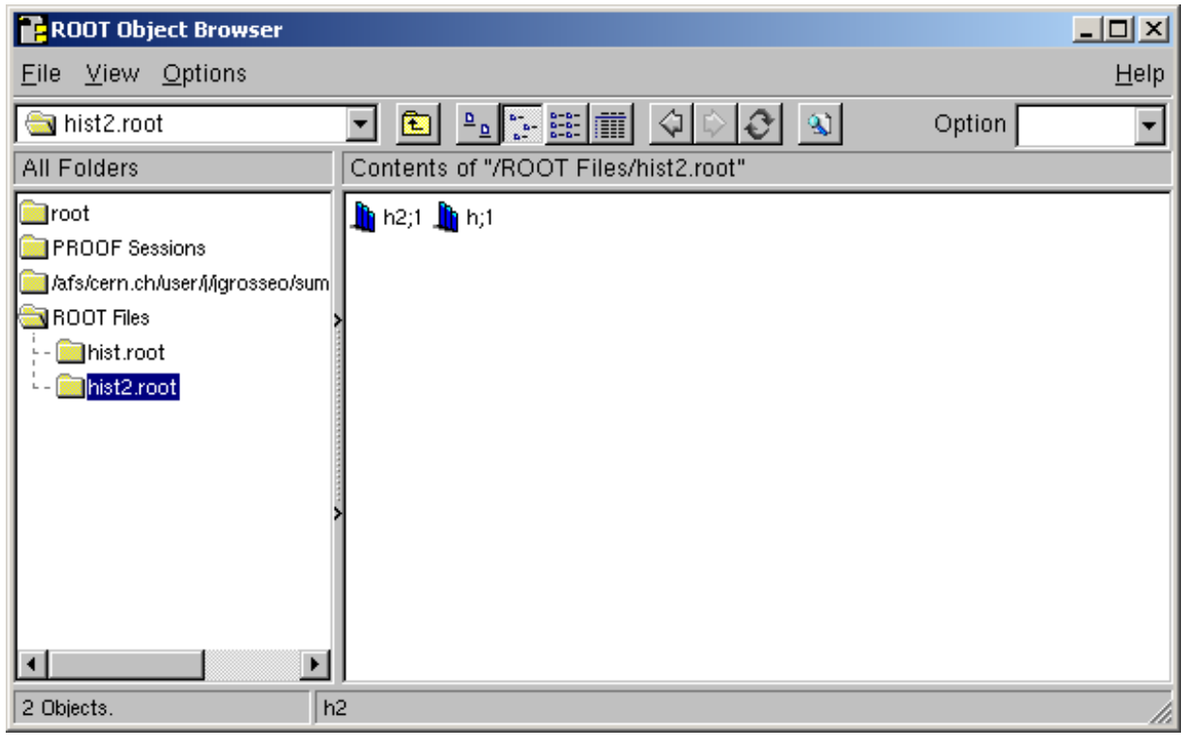
Abrir el fichero

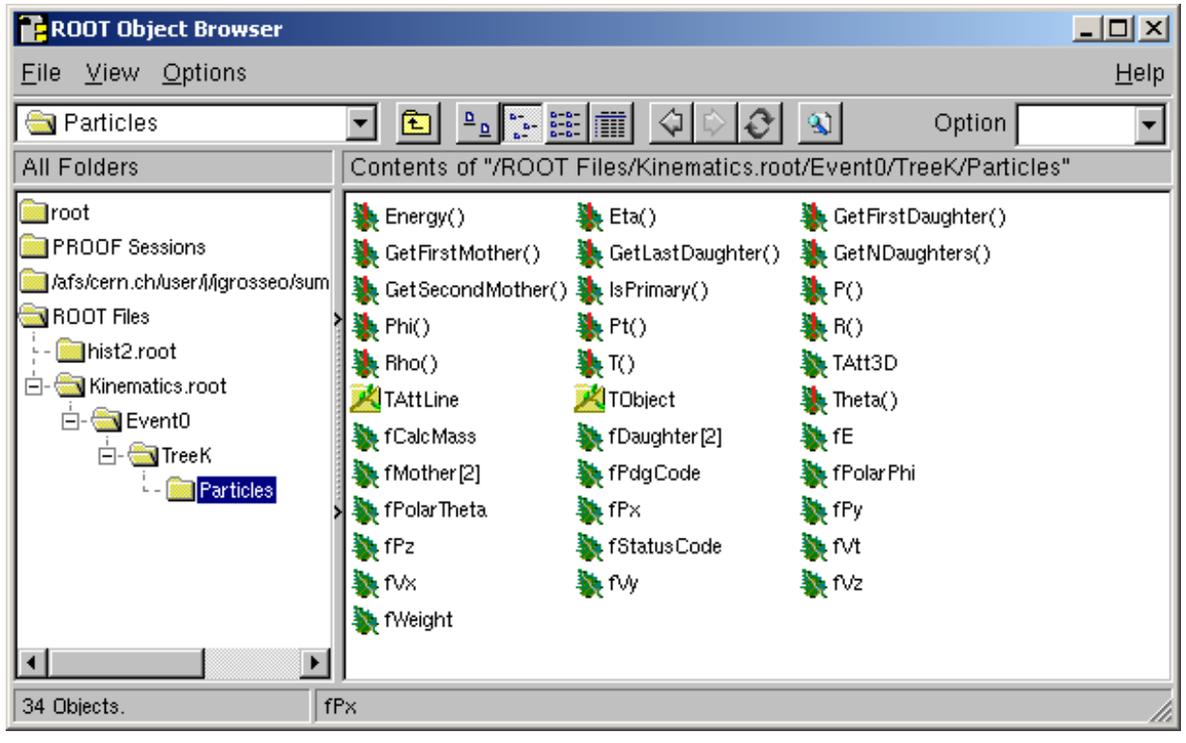
Navegar por el fichero

Pintar un histograma

Cambiar el estilo

Acceder a un Tree





Diferencia entre punteros y variables (I)

- Una **variable** es una asociación entre un nombre y un objeto
  - **int A**; //objeto permanente, se borra al final del bloque (programa, función, loop, if-else) automáticamente
- Un **puntero** es un objeto que se refiere (apunta) a otro objeto. Tienen la dirección física de memoria a la que apunta
  - **int *pA**; //puntero es un objeto temporal, hay que borrarlo cuando no lo necesitemos
- Una **referencia** es un alias de un objeto
  - **pA = &A**;

Diferencia entre punteros y variables (II)

- Si necesitamos **el objeto al cual se refiere el puntero**, usamos el operador *
  - *pA; //nos dá el valor del objeto al que apunta, A en este caso

- Si necesitamos **la dirección física de memoria de un objeto**, por ejemplo para inicializar un puntero, usamos el operador &
  - pA = &A; // &objeto nos da la direccion de objeto

- Si en vez de int es **un objeto de tipo clase** (con métodos),
  - usamos . cuando es un objeto real (variable)
  - usamos -> cuando es un puntero

- Los **punteros se pueden borrar** (comando **delete**), equivale a limpiar memoria, pero no estamos borrando el objeto al que apuntan. El puntero es temporal, por eso si se crea sin el comando **new**, hay que referenciarlo a un objeto real (variable)

- En ROOT CINT (linea de comandos) se puede usar indistintamente . ó -> pues CINT (el intérprete) no es extricto (otra ventaja más), pero C++ (código en macros) requiere **object.member()** ó **objectpointer->member()**

Ejemplo 1

```
root [0] TDatime now;

root [1] TDatime *p;

root [2] p = &now;

root [3] p->Print(); // es el mismo objeto
 Date/Time = Thu Mar 06 17:13:42 2013

root [4] (*p).Print(); // igual que now.Print()
 Date/Time = Thu Mar 06 17:13:42 2013

root [5] delete p;
```

Ejemplo 2

```
void swap0(Double_t x, Double_t y) {
    Double_t temp = x;
    x = y;
    y = temp;
  }
  void swap1(Double_t &x, Double_t &y) {
    Double_t temp = x;
    x = y;
    y = temp;
  }
  void swap2(Double_t *x, Double_t *y) {
    Double_t temp = *x;
    *x = *y;
    *y = temp;
  }
```

Explanation:
[swap0]
x is a Double_t initialized with the value of the first actual argument
y is a Double_t initialized with the value of the second actual argument
the values of x and y are swapped
[swap1]
x is an alias for the first actual argument
y is an alias for the second actual argument
the values of x and y are swapped
[swap2]
x is a pointer to Double_t initialized with the value of the first actual argument
y is a pointer to Double_t initialized with the value of the second actual argument
the values of the Double_t objects x and y are pointing to are swapped

```
root [0] .L swap.C
root [1] Double_t a = 1;
root [2] Double_t b = 2;
root [3] swap0(a,b);
root [4] cout << "a=" << a << " b="<< b << endl;
a=1 b=2
root [5] swap1(a,b);
root [6] cout << "a=" << a << " b="<< b << endl;
a=2 b=1
root [7] swap2(&a,&b);
root [8] cout << "a=" << a << " b="<< b << endl;
a=2 b=1
```

[3][4]
the function swap0 has no effect on its actual parameters
[5][6]
the function swap1 changes the value of a and b
[7][8]
the function swap2 is called with the addresses of a and b, it does not change its arguments (the addresses)but it changes the values of a and b

Comments:
C++ passes arguments by value.
To change the objects passed to a function they must be passed **by reference or by pointer**. [5][6][7][8]
Passing large objects **by values** [3][4] introduces performance problems because large objects have to be created as local variables. Passing arguments as reference or pointer does not introduce this performance penalty.

En macros (ficheros externos):
Si vamos a pasar argumentos o tenemos muchos objetos, es **mejor hacerlo como punteros o referencias**, no como variables reales (by value puede no hacer lo que queremos!! Implica crear variables locales en la funcion/método)

ROOT trabaja mejor con punteros (objetos temporales) pues permiten gestionar la memoria ya que podemos borrarlos (limpiar memoria)

¿Cómo nos afecta esto en ROOT? (I)

- ROOT tiene muchas clases para representar objetos
  - Ejemplo `TF1` (todas las funciones en ROOT empiezan con “T”)
  - http://root.cern.ch/root/html/TF1.html
  - Hay un constructor (entre otros) que toma cuatro argumentos

```
TF1 TF1(const char* name, const char* formula,
        Double_t xmin = 0,Double_t xmax = 1)
```

- **Como los llamo/uso?**
  - **Objeto/variable real**

```
root [] TF1 f("f","sin(x)/x",0,10);
root [] f.Draw();
root [] TF1 g;
root [] g= TF1 (“g","x",0,10);
root [] g.Draw();
```

  - **Puntero**

```
root [] TF1 * f = new TF1("f","sin(x)/x",0,10);
root [] f->Draw();
root [] TF1 * g;
root [] g= new TF1 (“g","x",0,10);
root [] g->Draw();
```

¿Cómo nos afecta esto en ROOT? (II)

- **A la hora de crear un objeto de forma rápida…**

```
root [] TCanvas * c1 = new TCanvas //canvas temporal de nombre c1
root [] c1->SetLineColor(kRed)
root [] delete c1
root [] TCanvas micanvas  // crea un canvas con nombre “micanvas”
root [] micanvas.SetLineColor(kRed)
```

- …se crea con el constructor/parámetros por defecto

```
root [] TBrowser *t1 = new TBrowser  //borrarlo antes de salir
root [] delete t1
root [] TBrowser * t2
root [] t2 = new TBrowser
(class TBrowser*)0x426e6c0
root [] TBrowser p  //así no habría que borrarlo
root [] p.Delete()  // borrarlo = cerrarlo en este caso...
root [] TBrowser *m = &p
```

- **Truco**: Utiliza el tabulador para completar el comando o averiguar la lista de sus métodos y argumentos del mismo. Ejemplo:

```
root [] p.<TAB>
```

Herencia en C++ (ROOT)

MyClass.h:
```
class MyClass {
public:
   MyClass();
   MyClass(int x);
   MyClass(char* text);
   int GetX()
   virtual void Draw();
protected:
   SetX(int x);
private:
   int x;
};
```

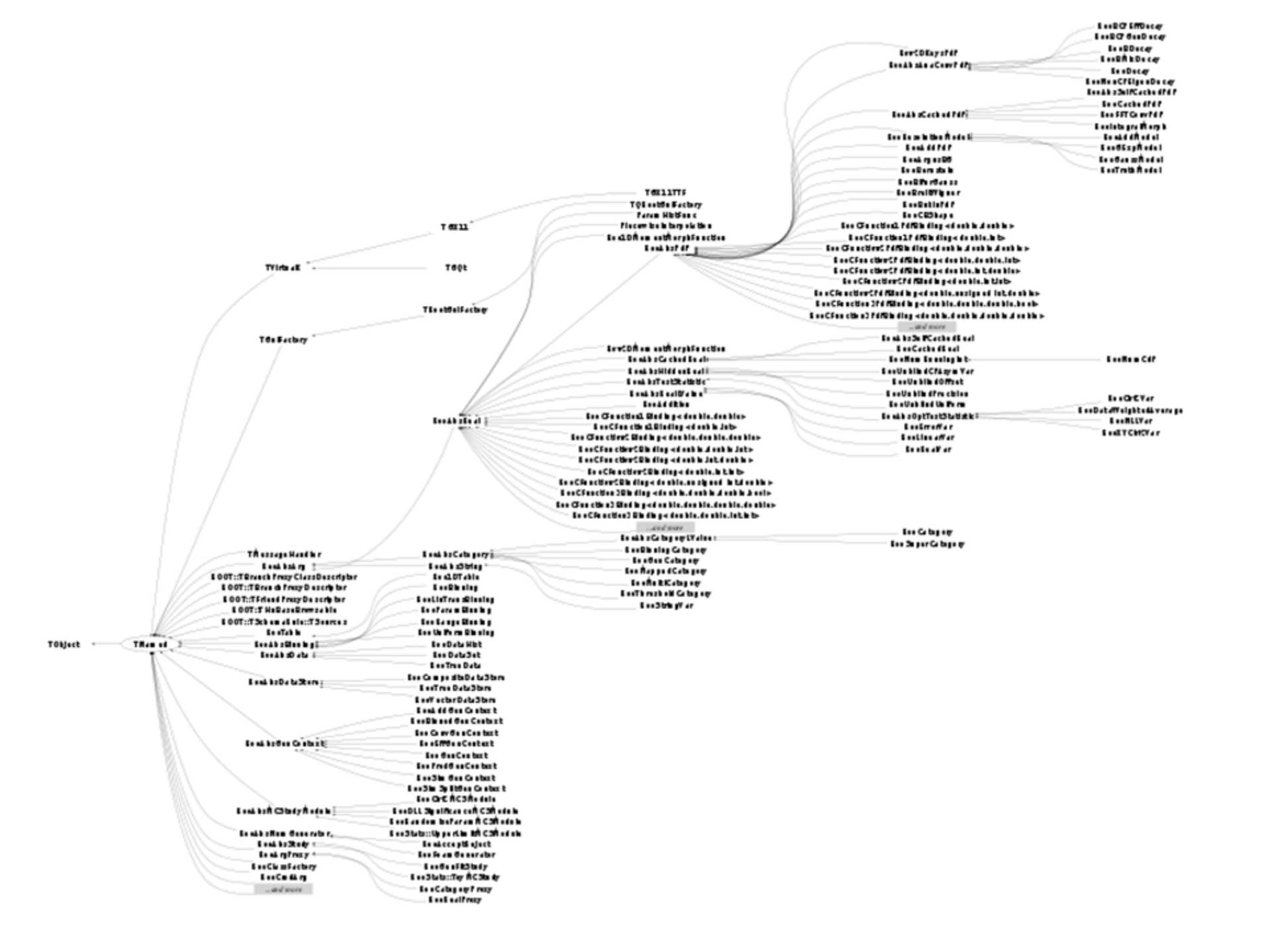
Inheritance for YourClass.h:
```
#include "MyClass.h"
class YourClass : public MyClass {
public:
   YourClass();
   YourClass(int x, int y);
   int GetY();
   virtual void Draw();
private:
   int y;
};
```

There are two major advantages of inheritance:

- **Common implementation**: It is easy to extend a class, to add some functionality without touching or copying the code of the base class. You need not to replicate the code of the base class. A lot of ROOT classes inherit from **TNamed**. Objects of all of these classes can have names and titles.
  http://root.cern.ch/root/html/TNamed.html which inherits from TObject http://root.cern.ch/root/html/TObject.html

- **Common behavior**: Inheritance creates a "is-a" or better "is-usable-as" relationship. Objects of the derived class can be used everywhere where objects of the base class can be used. One can write functions with base class objects as arguments.
  Caution: this way of using inheritance works only if used via reference or pointer

```
Shape *shape = new Shape[n];
shape[0] = new Triangle(...);
shape[1] = new Rectangle(...);
...
for(int i=0; i<n; i++) shape[i]->Draw();
```



Os recomiendo encarecidamente hacer este tutorial:

http://hadron.physics.fsu.edu/~skpark/document/ROOT/root_beginers/

Crear clases

Cualquier clase de c++ puede usarse en root:

Las clases derivadas de TObjects pueden usarse directamente con muchas otras clases de root (TList, TObjArray)

```
#include <TObject.h>
#include <TString.h>
class TAlumnos : public TObject {
  private:
    TString fFirstName;
    Int_t fAge;
  public:
    const TString GetFirstName() const { return
fFirstName; }     Int_t GetAge() const { return fAge; }
    TAlumnos(const char* firstname, Int_t age)
          : fFirstName(firstname), fAge (age) { }
    virtual ~TAlumnos () {}
    ClassDef(TAlumnos, 1)
};
```

TString to store strings

version number of class layout

when you add or change a member, increase the version number! 0 = not streamable

This macro adds some ROOT magic by including a dictionary created by CINT

Crear clases (2)

Incluir la clase en ROOT

```
root[ ] .L TAlumnos.C+
```

Úsala:

```
root[ ] s = new TAlumnos(“Pepe”,23)
root[ ] s->GetFirstName()
```

El objeto puede escribirse en un fichero

Puedes mostrar el contenido de cualquier clase de ROOT

```
root[ ] s->Dump()
```

Recursos

- Página principal de ROOT
  http://root.cern.ch

- Guía de referencia de clases
  http://root.cern.ch/root/html

- Tutorial C++
  http://www.cplusplus.com/doc/tutorial/
  http://www-root.fnal.gov/root/CPlusPlus/index.html

- Tutorial
  http://root.cern.ch/drupal/content/tutorials-and-courses

- Guía del usuario
  http://root.cern.ch/drupal/content/users-guide