

## Lectures

# Introduction to the modern software tools and programs for the simulation of high energy physical processes

## Introduction

The purpose of this lecture series is to provide the information about the modern software tools for the simulation of physical processes that happen with particles at accelerators and colliders. Usually these are high energy particles, from MeV (mega electronvolt,  $10^6$  eV) to TeV (teraelectronvolt,  $10^{12}$  eV), but sometimes one has to simulate also the passage of particles down to 1 eV.

The files of these lectures can be downloaded here: <http://cern.ch/mkirsano/course.tar.gz>. This tarball contains in particular files for the exercises.

The lecture files can be seen here:

### 1. Operating systems (OS), shells, file systems

The main computer operating system (OS) in big HEP (High Energy Physics) scientific centers, such as CERN normally is UNIX, and since ~ 2000 year it is Linux. CERN supports the type of Linux called CentOS (CC) and, for backward compatibility, the older type Scientific Linux CERN (SLC) (<http://linux.cern.ch>). They come from the Red Hat Enterprise Linux versions recompiled from source. The computers of the general purpose CC computer cluster at CERN are called lxplus. They support the common file system AFS. The volumes of this system are visible from all computers lxplus, as well as from most of the desktop computers installed at CERN. CERN Linux cluster users usually have their home folders on AFS. It is also possible to make CERN AFS visible from distant computer clusters.

There are other possibilities to create a common file system for several computers, without installing AFS, for example NFS. This is the case for the UTFSM cluster.

For the central storage of common software CERN uses now the CernVM file system (cvmfs). For ordinary users it is read-only and optimized for the multiple user access to many files. The transition to cvmfs is rather recent, it was performed early in 2019. You can still find old documentation pages that refer to AFS SW storage, made unreadable after this transition.

For the large volume data storage at CERN the file system EOS is used.

External login to the CERN computers is allowed only through SSH (ssh , sftp, scp).

CERN supports also the system MacOS and has some corresponding computers.

A few experiments at CERN support WINDOWS as one of the OS.

Users, for small tasks or on laptops, can choose any convenient system. SSH access to the Linux computers at CERN from WINDOWS is possible through e.g. PuTTY. It is possible to download the corresponding programs here: <http://www.chiark.greenend.org.uk/~sgtatham/putty>

The computer architecture, operating system and compiler version together define the computer platform, for example x86\_64-centos7-gcc63-opt means that you have the Intel 64 bits processor with operating system CentOS and your default compiler is gcc version 6.3.

In the internet it is possible to find the UNIX/Linux tutorial for example here: <https://www.tutorialspoint.com/unix/index.htm> (or just search in Google “UNIX tutorial”).

A very important tool in the code development in UNIX is Make. The corresponding manual can be found here: <http://www.gnu.org/software/make/manual/>

A user communicates with the computer operating system through the shell. Several shells exist, the

most popular are c-shell, or csh (more often its extension tcsh) and b-shell, or sh (in Scientific Linux and other flavors of RedHat Linux, such as Fedora, sh points to bash). Experiments usually support both shells, but one of them is used by default. Some specialists recommend to use b-shell for scripts. Some considerations about this can be found here: <http://www.grymoire.com/Unix/index.html>

To know what is your default shell type echo \$SHELL

#### Exercise 1.1

Write a shell script for the shells sh and csh that copies all files from the subdirectory above the current to the current subdirectory. Encode the explicit loop to do this.

#### Exercise 1.2

Write Makefile that compiles the program myprog.cc (the executable file name myprog). Perform the compilation in two steps: creating the object file myprog.o (compiler option -c), then linking. Check that if you type make for the second time without changing the source file, the compiler is not run. Add the dependency on the file myname.inc, check that if you change it, the program is recompiled. Use the standard Make variables \$@ and \$<

A few words about the compilers. On the lxplus cluster at CERN one usually does not use the native compiler of the installed Linux system because it cannot be updated and quickly becomes obsolete. For the CentOS Linux the compiler version is 4.8. To use instead a newer version of compiler one has to source the script e.g. /cvmfs/sft.cern.ch/lcg/contrib/gcc/6.3/x86\_64-centos7-gcc63-opt/setup.csh (or .sh for the b-shell or bash). After this redefinition of the compiler the user should choose the tools that correspond to it. For this, one should use the platform that corresponds to the defined compiler. The platform that corresponds to the compiler above is x86\_64-centos7-gcc63-opt.

## 2. Cross – platform compilation tools

As one could see in the exercise to write a Makefile, the Make does not solve completely the problem of the build automatization. For example you have to specify the dependencies on included files by hands. If your build system should work on different platforms, your Makefile will contain several or even many conditionals (“if” commands). This quickly becomes a problem for complicated SW projects. To solve this problem several cross-platform tools were created. If used on Linux or MacOSX, almost all these tools are made as a higher-level system above Make. This means that at the first step the tool creates a Makefile that is run at the second step by the usual commands “make” and “make install”. Note that the automatically created Makefile is usually rather difficult to read even for simple SW systems. Below I list two tools that historically were used and can be now encountered in HEP.

**Autotools.** Historically one of the first tools of this kind, a revolutionary system at the beginning. It can be encountered in some event generators. However, from the beginning it was oriented to c and c++ codes. Although it can be used for Fortran codes, it does not work with the dependencies on files included in the Fortran code files. It can be extended to such files using the so-called macros, but this makes its usage more complicated from the beginning. Many programmers believe it too complicated and do not recommend to use it if you prefer to be more concentrated on the code itself.

**CMake.** It was created later and took into account some problems that appeared in the development of autotools. The dependencies for the Fortran code files work out of the box, so there are less problems for the legacy event generators. I use this tool for my projects and below we will have an introduction to it.

Several other tools exist on the market. Maybe somebody could have a look at them. I don’t know about HEP projects that use them. These tools are: SCons, Premake, Ninja, Meson, FASTbuild,

Sharpmake, Maven, Ant, Gradle).

To get started with CMake there is for example the WEB page

<https://cmake.org/cmake/help/latest/guide/tutorial/index.html>,

which is a part of the general CMake documentation page <https://cmake.org/documentation/>

Below I give some hints about CMake usage that make the work with CMake projects more convenient.

1. Use GLOB for the automatic composition of source files list:

```
file(GLOB sources_Core ${PROJECT_SOURCE_DIR}/src/Core/*.cc)
```

### 3. Version control systems

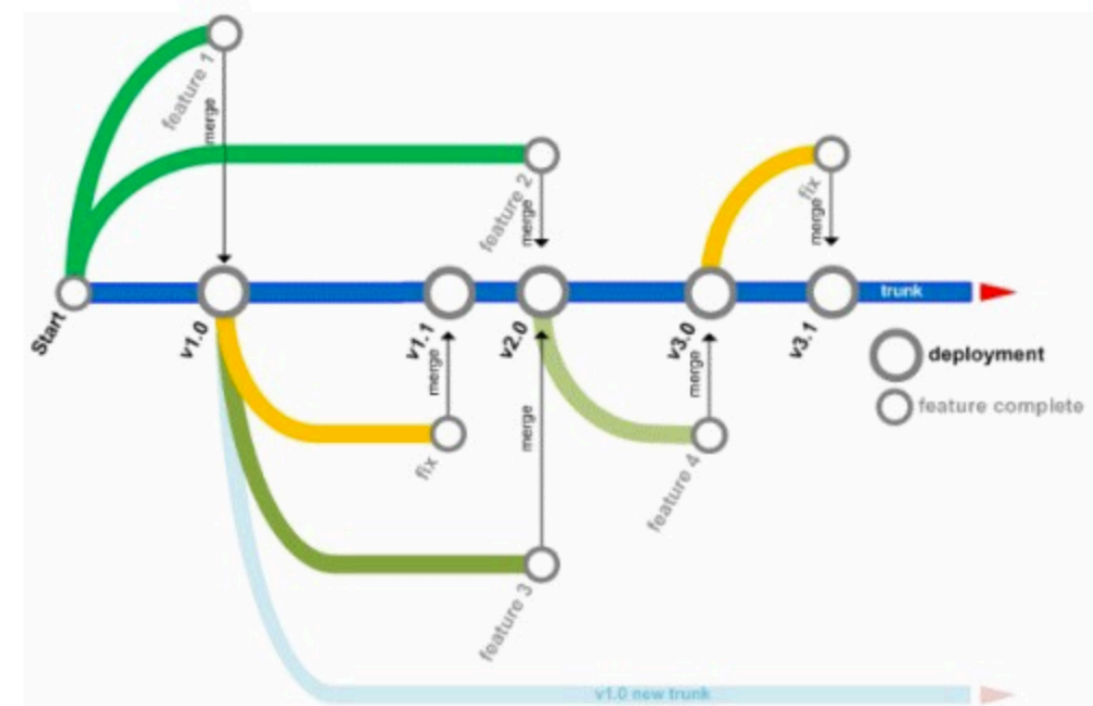
If a SW project is developed by several programmers, it is convenient to use version control systems. With the increase of the complexity of a project and number of developers the use of such system quickly becomes necessary.

A version control system allows users to keep track of the changes in software development projects, and provides a possibility to collaborate on those projects. Using it, the developers can work together on code and separate their tasks through branches.

There can be several branches in a version control system, according to the number of collaborators. The branches maintain individuality as the code changes remain in a specified branch(s).

Developers can combine the code changes when they are ready. Further, they can view the history of changes and go back to previous versions.

# Branch and Merge



A version control system keeps codes in the repository. The common internal feature of all such systems is that only differences are stored, together with the information about the time and authors of the changes and comments written by the authors.

Below is the list of most known systems

**CVS** is one of the oldest systems. It has been used for a long time at CERN for various projects.

**SVN** was developed as a successor to CVS with the idea to improve the functionality. Most of projects moved to it after CVS, but some skipped it in favor of GIT. For me, the branch creation system seemed strange and inconvenient in SVN. This is not only my opinion.

**GIT**. This system is used through **GitHub** or **GitLab**, the WEB – based GIT repositories. It is advertised as a system with many features convenient for professional developers of complicated SW projects. Most of these features are difficult to understand for a physicist. On the other hand, there is an enormous drawback: you cannot work with subdirectories. Unfortunately, to save resources CERN stopped the support of central SVN repositories soon after the transition of most of projects to GIT.

As another system I only mention **Apache Subversion**, others are too exotic.

The documentation can be easily found in the internet. Below I list a few main actions that a user or developer performs when he works with version control systems (very simplified, for a physicist).

1. Checkout the code. This command copies the code from the repository to your computer in such a way that you can use or change it and send the changes to the same repository. In GIT the command is git clone
2. Commit the code. These actions put your changes in the repository. In GIT it is a two steps action: git commit, then git push.
3. Update your local copy of the code by the changes committed by other developers. In GIT the command is git pull.

4. Branch actions. Getting or creating a branch
5. Pull requests. Some repositories are organized in such a way that only a few users can commit directly the changes. A user without such privilege, instead of commit, makes a pull request that goes into the repository only after the approval of a more privileged developer.

## 4. ROOT

ROOT <https://root.cern.ch/> is the analysis and visualisation tool. It is written in c++ and has the c++ command interpreter.

On the CERN Linux computer cluster various ROOT versions are installed in /cvmfs/sft.cern.ch/lcg/releases.

Example: /cvmfs/sft.cern.ch/lcg/releases/LCG\_87/ROOT/6.08.02/x86\_64-centos7-gcc62-opt/bin/root  
In all installed versions of ROOT there is a folder /tutorials with a very extensive collection of usage examples as macros (macrocommands to be executed by ROOT, files with extension .C). Often it is simpler to choose one of the examples and modify it looking in the reference manual (classes and method description) than to encode your own macro from zero using the ROOT user's manual, which is rather large.

To install ROOT on a computer, e.g. laptop, the following steps are needed:

1. Download ROOT source tree from the WEB site specified above, the file name is root\_vX.XX.XX.tar.gz, where X.XX.XX is a ROOT version. Copy this file to e.g. /swdisk/ROOT. It is better to use a production version, not development.
2. cd to this directory and expand ROOT tarball: tar xvfz root\_vX.XX.XX.tar.gz  
The subdirectory root will be created. Rename it to X.XX.XX: mv root X.XX.XX  
It is better to rename because another version of ROOT could be needed later.
3. Change directory to ROOT. Set the environment variable ROOTSYS:  
setenv ROOTSYS /swdisk/ROOT/X.XX.XX (for shells csh, tcsh) or  
export ROOTSYS=/swdisk/ROOT/X.XX.XX (for shells sh, bash). This means that the environment-driven installation method is used (the manual is /README/INSTALL.
4. Configure ROOT: configure linux --enable-roofit . The first argument is a system type, can be used for Fedora, slc6, centos on Intel computers. The second argument will install the additional package RooFit. The full list of possible options can be seen by typing configure --help.
5. make
6. make install
7. Create environment to work with ROOT:  
setenv LD\_LIBRARY\_PATH \${ROOTSYS}/lib (csh,tcsh) or  
export LD\_LIBRARY\_PATH=\${ROOTSYS}/lib (sh,bash)  
setenv PATH \${PATH}:\${ROOTSYS}/bin (csh,tcsh) or  
export PATH=\${PATH}:\${ROOTSYS}/bin (sh.bash)  
For immediate use the command rehash could be needed
8. With this environment ROOT will run after typing from any place root. Setting ROOTSYS, LD\_LIBRARY\_PATH, PATH (items 3,7) it is better to put into start files .cshrc (.shrc).

### Exercise 2.1

Write a ROOT macro (file with extension .C) that build a graph of the function  $f(x)$ , where  $x$  takes values (10, 15, 20, 30, 40, 60, 70),  $f$  takes values (200,250,300,400,500,700,780), with errors equal to  $\sqrt{f}$ . Use ROOT /tutorials and reference manual.

## 5. Event Generators

The interactions of particles characteristic for an experiment are often simulated by special programs called Event Generators. Such programs allow to simulate many particular, sometimes exotic or hypothetical, often very rare processes.

Event Generators are subdivided into General Purpose and more specialized Matrix Element generators. The former can simulate an interaction to the level of detectable particles, the latter simulate only a primary interaction described by some matrix element. The final state particles of Matrix Element generators can be e.g. gluons and quarks – particles that cannot be observed directly. In order to simulate the response of detectors to such particles the special processes called hadronization should be simulated. The result of hadronization is a number of particles that can be observed directly. The main difficulty here is that the hadronization cannot be described in the framework of the perturbation theory. There are no exact recipe how to perform it, only half-phenomenological models realized in the first type generators (General Purpose). This means that Matrix element generators can be used only together with one of the General Purpose generators.

The examples of General Purpose generators are PYTHIA, HERWIG, PYTHIA8, HERWIG7, SHERPA. The former two are encoded in FORTRAN, the next two are their further development realized in c++ using the Object Oriented technologies. SHERPA is one more generator encoded in c++ that uses some algorithms of PYTHIA and some other ideas. The hadronization in PYTHIA is based on the model of strings breaking while in HERWIG it is based on the clusterization model. The FORTRAN versions are now obsolete and rarely used, although they still work and can be run.

The General Purpose generators have some internal matrix element generators. In general they are simpler to use, but have some limits, for example pythia8 cannot make simulations in NLO. The attempt towards NLO is made in SHERPA, but because of this it is more complicated to use.

As the event generation is only the first step in the full simulation of samples, it is convenient to have a standard data record structure and input-output format are needed. HepMC (<http://lcgapp.cern.ch/project/simu/HepMC>) and its ASCII input-output are most often used as such.

## 6. Matrix Element Event Generators

The Matrix Element generators are much more numerous because of the absence of complicated parts that perform hadronization. Some of them are multipurpose, others are more specialized. The most known multipurpose generator is Madgraph5\_aMC@NLO. The example of more specialized generator is POWHEG-BOX.

The most known generators repository is HepForge:

<http://www.hepforge.org/projectshttp://lcgapp.cern.ch/project/simu/generator/>

If you are looking for a generator with a name GENNAME as a rule it is sufficient to search in Google “GENNAME HepForge”.

### Exercise 3.1

Install HepMC (production version);

Download and compile PYTHIA8;

Modify example main78 from /examples: type the transverse momentum  $P_t$  of Z-boson in each event, introduce the lower boundary on this value in PYTHIA8 parameters for the simulation and check that it works (there are no events below this value) and that the cross section decreases if you increase the boundary;

Write the events to the output file and write a ROOT macro that draws the  $P_t$  histogram.



### Упражнение 3.2

Включить гистограммирование с помощью ROOT в executable упражнения 3.1. Необходимые header файлы, библиотеки и опции компилятора посмотреть в /ex3.2. Построить гистограмму Pt Z-бозона и нарисовать ее.

## 6. GEANT4

This program, the so-called toolkit, i.e. a number of codes, from which a user can choose the needed parts and create his own application, serves to simulate the passage of particles through matter, possibly in presence of electromagnetic fields. For this, the toolkit has a possibility to simulate various interactions of particles with matter and particle decays, except those that have very small probability. The WEB site address is <http://geant4.web.cern.ch/geant4/>

To start to work with Geant4 one should choose or download some set of precompiled Geant4 libraries or download the source and build the libraries. The libraries to download can be chosen on the Geant4 WEB site. On the CERN lxplus cluster it is possible to use the precompiled libraries installed on cvmfs. Below is the recipe of building the libraries. At least 2.5 GB of disk space will be needed

1. Download the source code geant4.XX.XX.tar.gz and put it to some general folder
2. Untar the archive tar xvfz geant4.XX.XX.tar.gz, this results in directory geant4.XX.XX
3. Create the folder /geant4\_build and cd to it
4. Type the following command in one line:  
“cmake -DCMAKE\_INSTALL\_PREFIX=../geant4\_install -DGEANT4\_INSTALL\_DATA=ON  
../geant4.XX.XX
5. make (or make -j4 if your computer is multicore). With relevant -j4 the compilation on modern Intel computer takes 10 minutes
6. make install

To start learning Geant4, as usual, it is most convenient to start from the available examples. They can be found in the subdirectory /examples in the Geant4 source tree. There are many examples there in different folders that correspond to the learning level.

Посмотрим на пример N02 в /novice. В папке src находятся несколько файлов с кодами.

В файле \*DetectorConstruction.cc определяется геометрия установки.

В файле \*PhysicsList.cc определяются нужные частицы и процессы. Определение процессов с частицами, не имеющими сильного взаимодействия (как примеры N02, N03) имеет обычно довольно простую структуру. Если же нужно работать с адронами не очень низкой энергии, то нужно использовать стандартные листы из библиотек GEANT4.

В файле \*SD.cc определяются действия, которые должны осуществляться, когда частица попадает в части установки, определенные как чувствительные детекторы.

### Упражнение 4.1

Если не установлен, то установить вспомогательный пакет CLHEP

<http://proj-clhep.web.cern.ch/proj-clhep/>

В CERN CLHEP установлен на AFS в external (полный путь см. выше).

Скачать дистрибутив GEANT4 с Web site и растарить его;

В папке /ex4.1 можно найти файл configure4.9.1 (подставить правильную версию в название). Его нужно отредактировать, подставив правильные пути к CLHEP и GEANT4, затем сделать

source ./configure4.9.1;

Войти в папку /source и запустить make. Компиляция занимает почти час;

Скопировать в свою рабочую папку пример N02, войти в эту папку; взять в ex4.1 файл configure, отредактировать его (правильные пути к CLHEP и GEANT4), source ./configure, make;

После этого программу в batch можно запустить так: ./bin/Linux-g++/exampleN02 run1.mac

А в интерактивной моде, с графикой, так: ./bin/Linux-g++/exampleN02. Выйти из программы в интерактивной моде можно командой exit

#### Упражнение 4.2

Скачать программу vtmlview, например здесь: <http://www.km.kongsberg.com/sim>

Binary желательно установить в /usr/local/bin. Программа старая, при запуске она будет требовать старую версию libstdc++. Нужно установить compat версию gcc (yum list | grep gcc; yum install ...). Если она недостаточно старая, то можно сделать symlink, например:

/usr/lib/libstdc++-libc6.1-1.so.2 -> libstdc++-3-libc6.2-2-2.10.0.so

В Windows файлы VRML можно смотреть программой Deep Exploration.

Заменить в файле vis.mac DOWNFILE на VRML1FILE в команде /vis/open;

Запустить GEANT4 binary без параметра (при этом используется файл vis.mac);

Образуется файл g4\_00.wrl (при повторном запуске g4\_01.wrl и т.д.). Его можно смотреть программой vtmlview;

В интерактивной моде можно посмотреть какие команды имеются в данной программе (они имеют древесную структуру). Для этого нужно использовать команду help. Help имеет соответствующую древесную структуру. Кроме стандартных команд GEANT4 есть команды, существующие только в данном example, они находятся под директорией N02;

Найти в help команду, включающую магнитное поле, и убедиться, что траектория частицы искривляется;

#### Упражнение 4.3

Откомпилировать пример N03;

Вывести в файл полное энерговыделение частицы (электрона) в калориметре и построить гистограмму с помощью ROOT. RMS этой гистограммы можно считать разрешением калориметрического детектора по энергии (более точно фитировать гистограмму распределением Гаусса).

Убедиться, что

1. Разрешение ухудшается при увеличении количества пассивного вещества в калориметре (sampling fluctuations).
2. Разрешение улучшается при увеличении энергии частицы примерно по закону  $1/\sqrt{E}$  (выше энергии примерно 1 ГэВ).