COMPUTING METHODS IN HIGH ENERGY PHYSICS

S. Lehti
Helsinki Institute of Physics

Spring 2019
Lecture 1

Practical 'hands-on' course for computing in high energy physics.
Credits: 5op (3ov), 13 lectures + 6 exercises + home exam. Exercises split in two to have 12 x 1h exercises.
Recommended prerequisites: Introduction to particle physics and programming skills.
Literature: Recommending an advanced C++ book for reference.

Outline:

- Short review to Unix basics,latex,makefile,git
- FORTRAN
- C++
- ROOT
- Combining programming languages
- Cross section and branching ratio calculations
- Event generators
- Detector simulation and reconstruction
- Fast simulations
- Grid computing

Unix Shells

The Shell is a program that runs automatically when you log in to a Unix/linux system. The Shell forms the interface between users and the rest of the system. It reads each command you type at your terminal and interprets what you have asked for. The Shell is very much like a programming language with features like

- variables
- control structures (if,while,..)
- subroutines
- parameter passing
- interruption handling

These features provide you with the capability to design your own tools. Shell scripting is ideal for any small utilities that perform relatively simple task, where efficiency is less important than easy configuration, maintenance and portability. You can use the shell to organize process control, so commands run in predetermined sequence, dependent on the successful completion of each stage.

Shell Name	History
sh (Bourne)	The original shell
csh,tcsh,zsh	The C shell.
bash	Bourne Again Shell, from the GNU project
rc	More C than **csh**, also from GNU

File Handling

Most Unix commands take input from the terminal keyboard and send output to the terminal monitor. To redirect the output use $>$ symbol:

```
$ ls > myfiles
```

Likewise you can redirect the input with the $<$ symbol:

```
$ wc -l < myfiles
```

The need for redirection becomes more apparent in shell programming.

Pipes

The standard output of one process (or program) can be the standard input of another process. When this is done a "pipeline" is formed. The pipe operator is the | symbol. Examples:

```
$ ls | sort
$ ls | sort | more
$ ls | sort | grep btag | more
```

Here the standard output from left becomes the standard input to the right of |.

Pipelines provide a flexible and powerful mechanism for doing jobs easily and quickly, without the need to construct special purpose tools. Existing tools can be combined. Another useful command:

```
$ find .  -name "*.cpp" | xargs
grep btag
```

This is searching every file in this directory and any subdirectory with ending cpp and grepping the string btag.

Shell Programming

Suppose you have a file called 'test' which contains unix commands. There are different ways to get the system to execute those commands. One is giving the filename as an argument to the shell command, or one can use the command source in which case the current shell is used

```
$ csh test (or sh test)
$ source test
```

One can also give executing rights to the file and then run it

```
$ chmod 755 test
$ test
```

Here the number coding in the chmod command comes from rwxrwxrwx with rwx being binary number: rwx=111=7 and r-x=101=5. You can check the rights of your files with the -l option of the ls command.

A shell script starts usually with a line which tells which program is used to execute the file. The line looks like this (for bourne shell)

#!/bin/sh

Short review to Unix basics

Comments start with a # and continue to the end of the line. The shell syntax depends on which shell is in use. At CERN people have been using widely csh (tcsh), but also bash as the linux default has gained ground. It's up to you which shell you prefer.

csh

Variables are used to hold temporary values and manage changeable information. There are two types of variables:

- Shell variables
- Environment variables

Shell variables are defined locally in the shell, whereas environment variables are defined for the shell and all the child processes that are started from it.

The set command is used to create new local variables and assign a value to them. Different ways of invoking the set command are as follows:

- set
- set $name$
- set $name = word$
- set $name = (wordlist)$
- set $name[index] = word$

The first three forms are used with scalar variables, whereas the last two are used with array variables. The setenv command is used to create new environment variables. Environment variables are passed to shell scripts and invoked commands, which can reference the variables without first defining them.

- setenv
- setenv $name\ value$

To obtain the value of a variable a reference ${name} is used. Example

mv a.out analysis_${CHANNEL}_$RUN.out

Here the variables CHANNEL and RUN contain some values. A reference ${#name} returns the number of elements in array, and ${?name} returns 1 if the variable is set, 0 otherwise.

Short review to Unix basics

In addition to ordinary variables, a set of special variables is available.

Variable	Description
$0	Shorthand for $argv[0]
$1,$2,...,$9	Shorthand for $argv[n]
$*	Equivalent to $argv[*]
$$	Process number of the shell
$<	input from file

Conditional statements are created with
if(expression) then
else
endif

and loops with

foreach name (wordlist)
end

while (expression)
end

An example csh script

```
#!/bin/csh

if( ! ${?ENVIRONMENT} ) setenv ENVIRONMENT INTERACTIVE
if( $ENVIRONMENT != "BATCH" ) then
     if( ! ${?SCRATCH} ) then
          echo Setting workdir to HOME
          setenv WORKDIR $HOME
     else
          echo Setting workdir to SCRATCH
          setenv WORKDIR $SCRATCH
     endif
     setenv LS_SUBCWD $PWD
endif
#########################################
setenv INPUTFILE analysis.in
#setenv DATAPATH $LS_SUBCWD/data
setenv DATAPATH /mnt/data/data
#########################################

make

cd $WORKDIR
if( -f $INPUTFILE ) rm $INPUTFILE
cp -f $LS_SUBCWD/$INPUTFILE $WORKDIR
setenv LD_LIBRARY_PATH $LS_SUBCWD/../lib:${LD_LIBRARY_PATH}

echo
echo START EXECUTION OF JOB
echo

$LS_SUBCWD/../bin/${CHANNEL}_analysis.exe >& analysis.out
if( -f $WORKDIR/analysis.out ) mv $WORKDIR/analysis.out $LS_SUBCWD

echo
echo JOB FINISHED
echo


exit
```

Short review to Unix basics

bash

In bash the standard sh assignment to set variables is used
$name = value$
The array variables can be set in two ways, either by setting a single element
$name[index] = value$
or multiple elements at once
$name = (value1, .., valueN)$
Exporting variables for use in the environment
export $name$
export $name = value$
Arithmetic evaluation is performed when the following form is encountered: $((expression)). The basic if statement syntax is
if $condition$; then
else
fi
Most often the condition given to an if statement is one or more test commands, which can be invoked by calling the test as follows:
test $expression$
$[expression]$
The other form of flow control is the case-esac block. Bash supports several types of loops: for, while, until and select loops. All loops in bash can be exit by giving the built-in break command.

Perl

Perl has become the language of choice for many Unix-based programs, including server support for WWW pages. Perl is a simple yet useful programming language that provides the convenience of shell scripts and the power and flexibility of high-level programming languages. In perl the variable can be a string, integer or floating-point number. All scalar variables start with the dollar sign $. The following assignments are all legal in perl:
$variable = 1;
$variable = "my string";
$variable = 3.14;

Short review to Unix basics

To do arithmetic in Perl, the following operators are supported: + - * / ** %. Logical operators are divided into two classes, numeric and string. Numeric logical operators are <, >, ==, <=, >=, !=, ||, && and !. Logical operators that work with strings are lt, gt, eq, le, ge and ne. The most common assignment operator is =. Autoincrement ++ and decrement - - are also available. Strings can be combined with . and .= operators, for example

$a = "be" . "witched";

$a = "be"; $a .= "witched";

The conditional statement 'if' has the following structure: if (expr) {..}. Repeating statement can be made using while and until, looping can be done with the for statement.

Shell (system) commands can be run with the command system(shell command string)

More about csh(tcsh),bash and perl: man pages and www like
http://www.gnu.org/software/bash
http://www.tcsh.org
http://www.faqs.org/faqs/unix-faq/shell/csh-whynot
http://www.perl.com

An example Perl script

```
#!/usr/local/bin/perl
##############################################
$ibg = 2020;

$eventsPerFile = 500;
$allEvents = 100000;
$firstEvent = 1;

$BatchQueue = "1nw";
$Jobfile = "offlineAnalysis.job";
$runNumber = 1;

##############################################

$pwd = $ENV{'PWD'};
$orca = rindex("$pwd","ORCA");
$slash = index("$pwd","/",$orca);
$ORCA_Version = substr("$pwd",$orca,$slash-$orca);
$famos = rindex("$pwd","FAMOS");
$slash = index("$pwd","/",$famos);
$FAMOS_Version = substr("$pwd",$famos,$slash-$famos);

$ORCA_VERSION = $ORCA_Version;
if($famos > $orca) {
$ORCA_VERSION = $FAMOS_Version;
}

print "Submitting $ORCA_VERSION job(s)\n";

##############################################

for ($i = 1; $i <= $allEvents/$eventsPerFile; $i++) {
$LAST = $firstEvent + $eventsPerFile - 1;
system("bsub -q $BatchQueue $Jobfile $ORCA_VERSION $runNumber $eventsPerFile
$firstEvent $LAST $ibg");
$firstEvent = $LAST + 1;
}

if($firstEvent < $allEvents){
$LAST = $allEvents - 1;
system("bsub -q $BatchQueue $Jobfile $ORCA_VERSION $runNumber $eventsPerFile
$firstEvent $LAST $ibg"); }
```

Python

Python is a dynamic programming language used in a wide variety of application domains.

It is object-oriented, interpreted, and interactive programming language.

It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems. New built-in modules are easily written in C or C++ (or other languages).

A recommended coding style is to use 4-space indentation, and no tabs

CMS software (CMSSW) configuration files use Python.

More in http://www.python.org/

An example Python script

```
#!/usr/bin/env python
import sys
import os
import re

root_re = re.compile("(?P<rootfile>([^/]*))\.root")

def main():
    if len(sys.argv) == 1:
        print "\n"
        print "### Usage: lsmulticrabroot.py <multicrabdir>\n"
        print "\n"
        sys.exit()
    path = sys.argv[1]
    pwd = os.getcwd()
    if path.find(pwd) == -1:
        path = os.path.join(pwd,path)
    dirs = execute("ls %s"%path)
    for dir in dirs:
        dir = os.path.join(path,dir)
        if os.path.isdir(dir):
            subdirs = execute("ls %s"%dir)
            if subdirs.count("res") == 1:
                dir = os.path.join(dir,"res")
                files = execute("ls %s"%dir)
                for file in files:
                    match = root_re.search(file)
                    if match:
                        print "file:"+os.path.join(dir,file)

def execute(cmd):
    f = os.popen(cmd)
    ret=[]
    for line in f:
        ret.append(line.replace("\n",""))
    f.close()
    return ret

if __name__ == "__main__":
    main()
```

Version Control Systems

In HEP experiments the software is typically written by the collaboration members. Some part of the software can be commercial programs, but most of the code must be written from scratch.

CVS is a tool to manage the source code.

- to prevent overwriting others' changes
- keeps record of the versions
- allows users and developers an easy access to releases and prereleases

Documentation for CVS can be found in WWW, e.g.:
http://ximbiot.com/cvs

Subversion (SVN) is another version control system, and it's used widely. For example the LHC Higgs XS WG is using Subversion to manage a document latex source code.
http://subversion.apache.org

Git is an open source, distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git does not use a centralized server.

Projects using Git: Git, Linux Kernel, Perl, Ruby on Rails, Android, WINE, Fedora, X.org, VLC, Prototype

Useful links for documentation
http://git-scm.com/
http://jonas.nitro.dk/git/quick-reference.html

First you need to introduce yourself to git with your name and public email address before doing any operation.

```
$ git config --global user.name
 ''Your Name Comes Here''
$ git config --global user.email
 you@yourdomain.example.com
```

Version Control Systems

Some useful commands

```
$ git clone <repo>
$ git commit -a -m "Comment text"
$ git add file
$ git rm file
$ git remote add public $HOME/
 public/html/HiggsAnalysis.git
$ git push public
 refs/heads/master:refs/heads/master
$ git branch -a
```

How to make a public repository (lxplus)

```
$ cd $HOME/public/html
$ mkdir MyGitRepo.git
$ cd MyGitRepo.git
$ git --bare init
$ mv hooks/post-update.sample
    hooks/post-update
$ git update-server-info
$(git config http.sslVerify false)
```

Taking code from others:

```
$ git remote add matti <repo>
$ git fetch matti
$ git merge matti/master
```

CMS code repository at CMS-github

https://github.com/cms-sw/cmssw

```
$ git cms-addpkg DQM/Physics
$ git cms-merge-topic <user>:<branch>
$ git cms-merge-topic
slehti:from-CMSSW_6_2_0
$ git push my-cmssw <myownbranch>
$ git push my-cmssw from-CMSSW_6_2_0
$ git branch MyNewUpdateBranch
$ git fetch my-cmssw
$ git cherry-pick <commit-id>
$ git rebase --onto from-CMSSW_7_4_6
MyNewUpdateBranch
```

Sidestep to LaTeX

LaTeX is the tool to write your papers, and at this stage of your studies you should already know how to use it. If not, then learn it now! LaTeX documentation can be found in literature and in the web, google gives you several options for getting the documentation. You might try e.g.

www.ctan.org/tex-archive/info/lshort/english/lshort.pdf
wwwinfo.cern.ch/asdoc/psdir/texacern.ps.gz

Here we concentrate on two tools which you might need in the future: BibTeX and feynMF. The first is an easy way of managing your references, and the second one is a tool to draw Feynman graphs.

BibTeX

When using BibTeX, LaTex is managing your bibliography for you. It puts your references in the correct order, and it does not show references you have not used. This way you can have the same custom bib file for every paper you write, you just add more references when needed. You need two files, a bib file, and a style file. Here is an example style file based on JHEP style
http://cmsdoc.cern.ch/~slehti/lesHouches.bst

The bib file you have to write yourself, or you can use INSPIRE automated bibliography generator http://inspirehep.net to make the entry for yourself. Here is an example reference from a bib file:

```
@Article{L1_TDR,
      title = "The Level-1 Trigger",
      journal = "CERN/LHCC 2000-038",
      volume = "CMS TDR 6.1",
      year = "2000"
}
```

The citation (\cite{L1_TDR}) in the text works as usual. In your tex file you need to have \bibliographystyle{lesHouches}. BibTeX is run like LaTeX: latex-bibtex-latex-latex-dvips.

Sidestep to LaTeX

feynMF

feynMF is a package for easy drawing of professional quality Feynman diagrams with METAFONT. The documentation for feynMF can be found in

http://www.ctan.org/pkg/feynmf

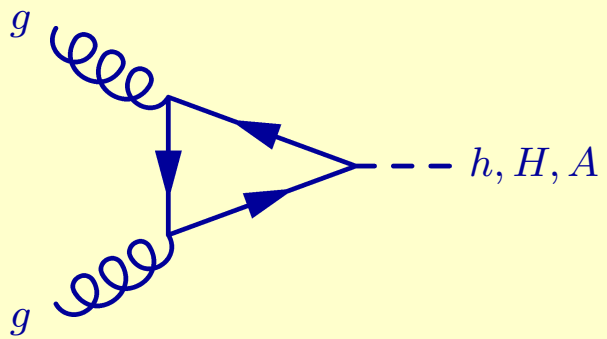
Instructing LaTeX to use feynMF is done including the feynmf package: (files feynmf.sty and feynmf.mf needed)
\usepackage{feynmf}

Processing your document with LaTeX will generate one or more METAFONT files, which you will have to process with METAFONT. The METAFONT file name is given in the document in fmffile environment:
\begin{fmffile}{$< METAFONT - file >$}
...
\end{fmffile}

Example

The following code produces the figure below:

```
\begin{fmffile}{triangle_loop}
\begin{figure}{h}
\centering
\parbox{50mm}{
\begin{fmfchar*}(100,60)
    \fmfleft{i1,i2}
    \fmfright{o1}
    \fmf{curly,tension=0.5}{i1,v1}
    \fmf{curly,tension=0.5}{i2,v2}
    \fmf{fermion,tension=0.1}{v2,v1}
    \fmf{fermion,tension=0.3}{v1,v3}
    \fmf{fermion,tension=0.3}{v3,v2}
    \fmf{dashes}{v3,o1}
      \put(0,60){\small $g$}
      \put(0,-5){\small $g$}
      \put(100,28){\small $h$, $H$, $A$}
\end{fmfchar*}}
\end{figure}
\end{fmffile}
```



Run METAFONT (file feynmf.mf needed)
mf "\mode=localfont;\input triangle_loop.mf;"
The whole chain is latex-mf-latex-dvips.

Makefiles

The make utility automatically determines which pieces of a large program need to be recompiled and issues command to recompile them. However, make is not limited to programs, it can be used to describe any task where some files must be updated automatically from others whenever the others change.

To run make, you must have a makefile. Makefile contains the rules which make executes. The default names for the makefiles are GNUmakefile, makefile and Makefile (in this order). If some other name is preferred, it can used with command make -f myMakefile

A documentation about make can be found in www.gnu.org/software/make/manual/make.html

A makefile is an ASCII text file containing any of the four types of lines:

- target lines
- shell command lines
- macro lines
- make directive lines (such as include)

Rules

Target lines tell make what can be built. Target lines consist of a list of targets, followed by a colon (:), followed by a list of dependencies. Although the target list can contain multiple targets, typically only one target is listed. Example:

```
clean:
        rm -f *.o
```

The clean command is executed by typing

```
$ make clean
```

Notice that the shell command line after the semicolon has a tab in front of the command. If this is replaced by blanks, it won't work! This applies to every line which the target is supposed to execute.

Makefiles

Dependencies are used to ensure that components are built before the overall executable file. Target must never be newer than any of its dependent targets. If any of the dependent targets are newer than the current target, the dependent targets must be made, and then the current target must be made. Example:

```
foo.o: foo.cc
        g++ -c foo.cc
```

If foo.o is newer than foo.cc, nothing is done, but if foo.cc has been changed so that the timestamp of file foo.cc is newer than the timestamp of foo.o, then foo.o target is remade.

One of the powerful features of the make utility is its capability to specify generic targets. Suppose you have several cc files in your program. Instead of writing every one object file a separate rule, one can use a suffix rule:

```
.cc.o:
        g++ -c $<
main: a.o b.o c.o d.o
        g++ a.o b.o c.o d.o
```

Here $< is a special built-in macro, which substitutes the current source file in the body of a rule. When main is executed, and the timestamp of the object files is newer than that of exe file main, the dependent targets a.o b.o c.o d.o are made using the suffix rule: if the corresponding cc file is newer, new object file is made. When all object files needing updating are made, then the exe file main is made.

Macros

In the above example the object files were typed in two places. One could use a macro instead:

```
OBJECTS = a.o b.o c.o d.o
main: $(OBJECTS)
        g++ $(OBJECTS)
```

Functions

The function call syntax is $(ftion\ args)$ There are several functions available for various purposes: text and file name manipulation, conditional functions etc.
Example: let's assume that the program containing the source files a.cc, b.cc, c.cc and d.cc are located in a directory where there are no other cc files. So every file ending .cc must be compiled into the executable. One can use functions wildcard, basename and addsuffix.

```
files = $(wildcard *.cc)
filebasenames = $(basename $(files))
OBJECTS = $(addsuffix .o,$(filebasenames))
```

Here the function wildcard gives a space separated list of any files in the local directory matching the pattern *.cc. If you now modify your program to include a fifth file e.cc, your Makefile will work without modifications.

An example Makefile

Notice the usage of macros $(CXX) and $(MAKE). What does the @ do? If new line is needed, one can make the break with \.

```
files = $(wildcard ../src/*.cc ../src/*.cpp *.cc *.cpp)
OBJS = $(addsuffix .o,$(basename $(files)))

OPT = -O -Wall -fPIC -D_REENTRANT

INC = -I$(ROOTSYS)/include -I. -I../src \
-I../src/HiggsAnalysis/MssmA2tau2l/interface

LIBS = -L$(ROOTSYS)/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad \
-lTree -lRint -lPostscript -lMatrix -lPhysics -lpthread -lm -ldl \
-rdynamic

.cc.o:
        $(CXX) $(OPT) $(INC) -c $^  -o $@
.cpp.o:
        $(CXX) $(OPT) $(INC) -c $^  -o $@

all:
        @$(MAKE) –no-print-directory All
All:
        @$(MAKE) compile; $(MAKE) analysis.exe

compile: $(OBJS)

analysis.exe: $(OBJS)
        $(CXX) $(LIBS) -O $(OBJS) -o analysis.exe

clean:
        rm -f $(OBJS)
```