Programming for Today's Physicists & Engineers

V. Erkcan Özcan
Boğaziçi University
With inputs from Özgür Çobanoğlu

ISOTDAQ'12, February 01, 2012

Work Environment

- Today's (astro)particle, accelerator experiments and information industry: Large collaborations...
- Need more than ever:
  - Code sharing/reuse - object orientation, atomic code, portability, version control
    - Use languages/libraries/tools you might not know.
  - Code binding - framework integration
    - Usually with a "scripting" language: python, tcl, etc.
  - Documentation & good visualization
    - Doxygen, UML, wikis, bug-tracking, histogramming & graphing
  - Working remotely
    - Cloud computing/grid, batch systems, remote login/monitoring
  - Not reinventing the wheel
    - Finding the right libraries: thread-safe, well-tested, maintained
    - Open sourcing: Help others not reinvent the wheel

In these side boxes, there will be tiny tips for more advanced stuff.

Copy & Paste, But Know What You Do

- Inheriting code from others is good - sometimes almost compulsory.
- But don't do it before you understand the underlying logic.
- Ex: You are asked to write a piece of code in C++ that tells you how many days there is in a given month, ie.

  > howmanydays april
  april has 30 days.

- Luckily your colleague has a code that does a similar task: converting the month number into month name, ie.

  > whichmonth 6
  The 6th month is june.

Hash Maps

```
#include <iostream>
#include <tr1/unordered_map>

const char* suffix(unsigned int nm) {
  if (nm==1) return "st";
  if (nm==2) return "nd";
  return "th"; }

using namespace std;

int main(int argc, char*argv[]){

  if (argc!=2) return 1;
  int mnth = atoi(argv[1]);
  if (mnth<1 || mnth>12) return 1;

  tr1::unordered_map< int, const char* > months;
  months[ 1] = "january";
  months[ 2] = "february";
  months[ 3] = "march";
  months[ 4] = "april";
  months[ 5] = "may";
  months[ 6] = "june";
  months[ 7] = "july";
  months[ 8] = "august";
  months[ 9] = "september";
  months[10] = "october";
  months[11] = "november";
  months[12] = "december";

  cout << "The " << mnth << suffix(mnth)
       <<" month is " << months[mnth] << endl;
  return 0;
}
```

- Hash map: Convert some identifiers (keys) into some associated values.
  - Useful for fast search algorithms, for cacheing data, for implementing associative arrays.
  - tdaq software commonly use hash maps as part of pattern recognition by clusterization or as part of networking for resolving server/client communications.
- unordered_map is part of the STL in the upcoming C++0x standard.

A Simple Dictionary

```
#include <iostream>
#include <tr1/unordered_map>

using namespace std;

int main(int argc, char*argv[]){

  tr1::unordered_map< const char*, int > months;
  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << "february : ndays= " << months["february"] << endl;
  cout << "june     : ndays= " << months["june"] << endl;
  cout << "december : ndays= " << months["december"] << endl;
  return 0;
}
```

- Modification and testing
  - Checked from documentation that hashes for char* are also available.
  - Tested with a few examples: looks good...

```
> g++ test.cxx
> ./a.out
february  : ndays= 28
june      : ndays= 30
december : ndays= 31
```

- So now the final product?

Final Code

```
#include <iostream>
#include <tr1/unordered_map>

using namespace std;

int main(int argc, char*argv[]){

  if (argc!=2) return 1;

  tr1::unordered_map< const char*, int > months;
  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

- Assume enduser is well-behaved.
  - In real-life, never do that!

Final Code

```
#include <iostream>
#include <tr1/unordered_map>

using namespace std;

int main(int argc, char*argv[]){

  if (argc!=2) return 1;

  tr1::unordered_map< const char*, int > months;
  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

- Assume enduser is well-behaved.
  - In real-life, never do that!
- Final product…

Final Code

```
#include <iostream>
#include <tr1/unordered_map>

using namespace std;

int main(int argc, char*argv[]){

  if (argc!=2) return 1;

  tr1::unordered_map< const char*, int > months;
  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

- Assume enduser is well-behaved.
  - In real-life, never do that!
- Final product...
  - does NOT work!

```
> g++ test.cxx
> ./a.out june
june has 0 days
```

Final Code 2

```
#include <iostream>
#include <tr1/unordered_map>
#include <ext/hashtable.h>

using namespace std;

struct stringEqual{
  bool operator()(const char* str1, const char* str2) const {
    return strcmp(str1,str2)==0; }
};

int main(int argc, char*argv[]){

  if (argc!=2) return 1;

  tr1::unordered_map< const char*, int,
    __gnu_cxx::hash<const char*>, stringEqual > months;

  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

- Write a comparison function between entries...
- Template also needs a hash function.
- good news: gcc extensions have one.

```
> g++ test.cxx
> ./a.out june
june has 30 days
```

- It works!

Final Code 3

```
#include <iostream>
#include <ext/hash_map>

using namespace std;

struct stringEqual{
  bool operator()(const char* str1, const char* str2) const {
    return strcmp(str1,str2)==0; }
};

int main(int argc, char*argv[]){

  if (argc!=2) return 1;

  __gnu_cxx::hash_map< const char*, int,
    __gnu_cxx::hash<const char*>, stringEqual > months;

  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

- Mixing tr1::ordered_map with __gnu_cxx::hash is a really bad choice.
  - Why? Find this out yourself, by finding out how many times stringEqual is being called.
- Proper code without mixing - all using gnu extensions.
  - Why? Find this out yourself, by finding out how many times stringEqual is being called.
- Finally we have code that works fast, reliably & correctly.
  - We are done...

```
> g++ test.cxx
> ./a.out december
december has 31 days
```

Final Code 4

```
#include <iostream>
#include <string.h>
#include <tr1/unordered_map>

using namespace std;

int main(int argc, char*argv[]){

  if (argc!=2) return 1;

  tr1::unordered_map< string, int > months;

  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

- How about a portability?
  - Not portable in time: tr1::xxx has a chance of becoming part of C++, while __gnu_cxx are likely to disappear.
  - Not portable in space: No chance of your code working with any other compiler.
- Need a simple, clean implementation.
  - Know and use STL consistently.
  - Steep learning curve, but STL containers & classes saves you a lot of effort.
  - They are also MUCH safer - resistance to buffer overflows, thread-safety, etc.
- Finally we have code that works fast, reliably & correctly &
  it is short and portable.
  - Or at least this is what you might believe.

Documentation

```cpp
#include <iostream>
#include <string.h>
// Might need to remove tr1 when C++X0 finalises
#include <tr1/unordered_map>

using namespace std;

int main(int argc, char*argv[]){

  // Don't do anything if number of arguments != 1
  if (argc!=2) return 1;

  // Might need tr1 removed, when C++X0 finalises
  // Using unordered_map - should be scalable
  // This would not work if string => char* array
  tr1::unordered_map< string, int > months;

  months["january"] = 31;
  months["february"] = 28; // have not considered leap years
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  // not implemented any catches for non-existing month names
  cout << argv[1] << " has " << months[argv[1]]
       << " days" << endl;
  return 0;
}
```

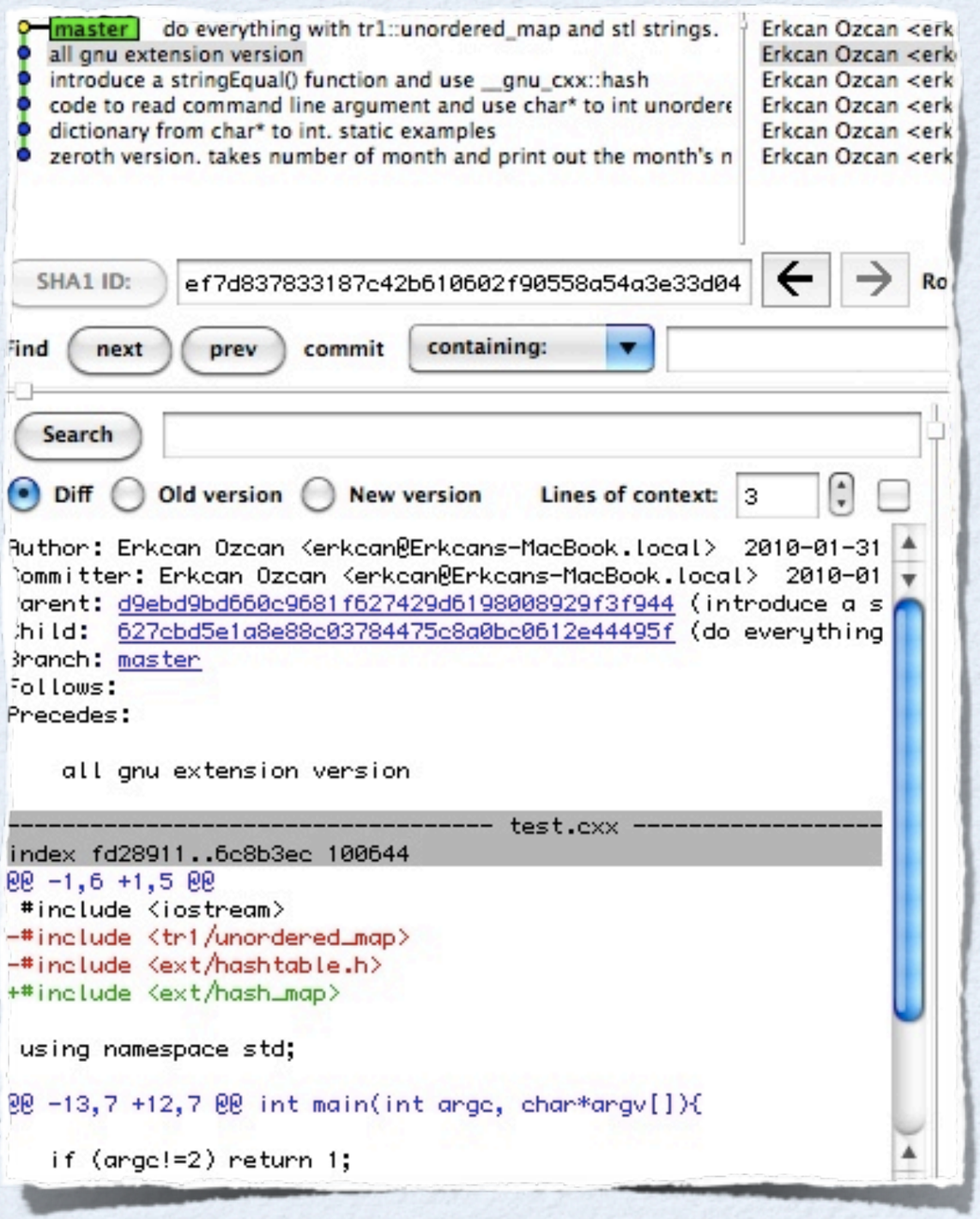
- Internal and external documentation!
  - It helps the poor stranger who inherits your code, i.e. be kind to maintainers.
  - 3 years not using your code: you will be a poor stranger!
  - Documentation generators like Doxygen are handy. www.doxygen.org
- For large projects, consider using UML (unified modelling language) from start, i.e. during design.
- PS: For your own benefit, it also helps to keep a history of how you compiled/ran your code.
  - More on this later...

Note: C++X0 finalized into C++11 & unordered_map is now part of the C++.

Keeping Track

- Version control systems are a must while collaborating.
  - But also excellent for personal use if you want to keep track of what you do.
- The "basic" ones are CVS and Subversion.
- Particularly for your private repositories, distributed management systems are a must.
  - Your instance is the full repository with history.
- My favorite is git : git-scm.com (but others like mercurial, bazaar, etc. around)
  - FOSS initially developed for Linux kernel code management.
    - Linus Torvalds: "The slogan of Subversion for a while was "CVS done right", or something like that, and if you start with that kind of slogan, there's nowhere you can go. There is no way to do CVS right."
  - git: (v) to go. [Turkish to English translation.]
    - That is what you will soon say to cvs/svn-users: "please git"!

GIT EXAMPLE



Start empty repository:

> git init

Add a new file and commit each version:

> git add test.cxx
> git commit test.cxx

Check differences to committed code:

> git diff test.cxx

tk-based GUI (among others):

> gitk &

clean/compress repository:

> git gc --aggressive

git makes a powerful collaboration tool when combined with a web-based file hosting service.

git with color: `git config --global color.ui true`

Keeping History

- Particularly when learning from someone or doing things for the first time, it is useful to keep a record of the commands, choices, etc. you have input.
  - Just dumping your shell history (history > mysteps.txt) is the simplest thing to do.
    - To make best use of it, try increasing its cache size. (export HISTSIZE=500).
  - However if the work you are doing requires logging into multiple machines, you might want to try the command `script`. It logs everything on the terminal.
  - Screen's scrollback buffer is also useful. Set its size in your ~/.screenrc file: defscrollback 500
  - If you jump between many directories, `pushd` / `popd` / `dirs` is useful. Consider `alias`'ing `pushd` as `cd` (needs some .bashrc magic though to keep all functionality of `cd` intact).

see more on pushd as cd in the back-up slides.

Use the Right Tool

- Do not use a sledge hammer to crack a nut!
  - For quick and not-so-dirty solutions, use interpreted languages, like python, perl, tcl...
  - These languages are also commonly used as part of binding frameworks: fast C/C++ modules instantiated, executed and their results bridged.
  - Personal favorite Python: Very expressive language with largest standard library after Java.

- Our dictionary example is a treat with the built-in dictionary type, `dict`.
- Realise that using the right tool might mean convincing colleagues/boss who like the "old way".

Shorter, and does more stuff

```
from sys import argv

if len(argv)!=2:
    exit()

months={'january':31, 'february':28, 'march'    :31,
        'april'  :30, 'may'     :31, 'june'     :30,
        'july'   :31, 'august'  :31, 'september':30,
        'october':31, 'november':30, 'december' :31}

usrmonth = argv[1].lower()
if usrmonth in months:
    print usrmonth, "has", months[usrmonth], "days"
else:
    print "sorry no such month is known"
```

Swiss Army Knife

- Your swiss army knife in the *nix world is `awk`!
  - Named after Aho, Weinberger, Kernighan.
  - A full-fledged (Turing-complete) interpretted (compilers also exist) programming language hidden inside one single command, and present in ANY *nix environment.
  - Ex: browse all pictures from your camera - haphazardly distributed in a directory and resuffix all .mp4 files to .3gp.

```
find . | awk -F. '{if ($NF=="mp4") print "mv",$0,$0}' |
sed s/'\.mp4'/'\.3gp'/2 | awk '{system($0)}'
```

- In the *nix world small gears make big machines...
  - awk goes best with `sed`, `head`, `tail`, `sort`, `find`, `grep`.

If you prefer a leatherman tool instead of a swiss army knife, there is perl, python, ruby, etc.

Organize Your Code

- Have a meaningful directory structure.
- Do not create all your projects at the root of your home directory.
- When installing from source:
  - ./configure --help is your friend. Use it to learn how to direct your installation to non-default (/usr/local/) directories.
  - Choose directory names wisely - put version numbers.
  - Softlinks are your friends. Use them to define hide different versions of code. Ex:

```
lrwxr-xr-x   1 erkcan  admin    12 May  3  2010 dev -> root5.26.00b
dr-xr-xr-x  29 erkcan  admin   986 Oct 21  2009 root5.18.00
dr-xr-xr-x  30 erkcan  admin  1020 May  3  2010 root5.26.00b
```

- Exercise permission features properly. Minimum rights principle as usual in all *nix.

Back to Portability

- Use makefiles.
  - Makefiles that come with many modern packages might look complex at first.
  - Write your own once, and it will be all clear.
  - Parallel compilation (with many cores): make -j4

```
all: a.out

test.o: test.cxx
	g++ -c -o $@ $<
a.out: test.o
	g++ $<

.PHONY: clean all
clean:
	rm -f test.o ./a.out
```

- Learn about autoconf, automake, CMake, etc.
  - Even if you don't know how to write configurations, learn how to use them.
  - For Java + parallel compilation, try Ant, Maven.
    ant.apache.org maven.apache.org

Debugging, Profiling

- Injecting printf/cout statements for debugging your code becomes unmanageable when your code becomes too much integrated in a framework.
  - gdb, GNU Debugger, is the way to go.
  - Most crashes are due to accessing memory locations that are not to be accessed: dereferencing NULL pointers, overflowing arrays,... gdb can give you a stack trace at the minimum - your core files become meaningful.
    - Basic gbd commands: `run`, `bt`, `info <*>`, `help`
- However gbd is missing a major functionality: Large piece of code frequently means memory leaks.
  - Try the smart pointers, as they become more common (part of C++x0 standard, you can also try BOOST libraries, www.boost.org).
  - Use a profiling tool like Valgrind (available also on MacOSX)! valgrind.org

Working Remotely

- ssh is a way of life.
  - Don't write your password all the time, by using public key authentication.
    - Generate keys with 'ssh-keygen -t dsa'. Use a passphrase. Don't copy id_dsa, only copy id_dsa.pub. Use ssh-agent to save repeatedly entering passphrase. Append your public key to ~/.ssh/authorized_keys on machines that you want to log in to.
- sshfs is a nice way to mount ssh-accessible space.
  - But does not offer the goodies in using AFS.
  - When you want to share files with other users on AFS, remember that simple UNIX file permissions are not enough.

On a Mac OSX machine, sshfs is easily installed using MacFusion + OSXFuse (or Fuse4X).

Security

- Do not use the same password everywhere.
  - Particularly for one-time user passwords used on various websites, consider using webbrowser extensions that generate random passwords for you. Or generate random sequences yourself and save them with tools like Apple Keychain, KDE KWallet, Gnome Keyring, etc.
    - Firefox users: Don't forget to set a master password.
- Open a new terminal (actually a new session if possible) whenever you sit on a new public terminal.
  - It is as simple as running `script` with proper command line arguments to log everything you type on the terminal.

Protecting your Work Terminal

- `screen` is GNU's hidden gem.
  - Part of the GNU base system: Present by default on almost all *nix machines around.
  - Creates virtual terminals - that do not die when connection is lost, X crashes, etc.
    - Your processes can keep on working after you log-off. (Alternative is `nohup`, but has a lot fewer features and quite often it is blocked from users.)
- `screen` cannot be described, it is lived!
  - Try it. Tip: CTRL+A then ? to see shortcut keys.
  - Warning: It can be addictive...

If you want a colorful visualisation of your screens, try a program like byobu.

Protecting your Work Desktop

- VNC, Virtual Network Computing, is the equivalent of `screen`, but for full-fledged graphical desktops.
  - You can create virtual desktops that live without you being logged on.
  - You need a vnc client on your side, and a vnc server on the remote machine. (Mac OSX 10.5+ screen sharing is VNC compatible.)
  - NEVER use VNC directly - your desktop can/will be watched by men-in-the-middle.
  - ssh port forwarding is the right way to go! Ex:

```
ssh -L5902:<VNCserverIP>:5902 <user>@<remoteMachine>
vncserver :2 -geometry 1024x640 -localhost -nolisten tcp
```

  - Additional bonus: VNC communication is/can-be made much faster than X forwarding.

ssh port forwarding can allow you to go behind firewalls by connecting remote ports too!

Getting the Most out of your Machine

- Nowadays even the laptops are multicore.
  - However most physics-code authors don't know anything about threading, etc.
- Task spooler - vicerveza.homeunix.net/~viric/soft/ts/
  - Extremely light-weight batch system.
  - Pure C, no dependencies, compiles and works easily on GNU systems with gcc (Linux, Mac OSX, Cygwin, etc.).

```
export TS_MAXCONN=20
export TS_SLOTS=<#cores>
ts
ts <job>
```

If you don't know what Cygwin is and you are using Windows, you MUST see the backup slides.

If you don't set TS_MAXCONN, you might reach the OS's limit for maximum number of open files.

Batch Systems

- PBS or LSF are common in HEP institutions.
  - Good practice to learn about your resources as early as possible.
- GRID is the future. Get your certificate.
  - Beware! Getting a certificate can be time consuming.
  - You will also need to join a virtual organization.

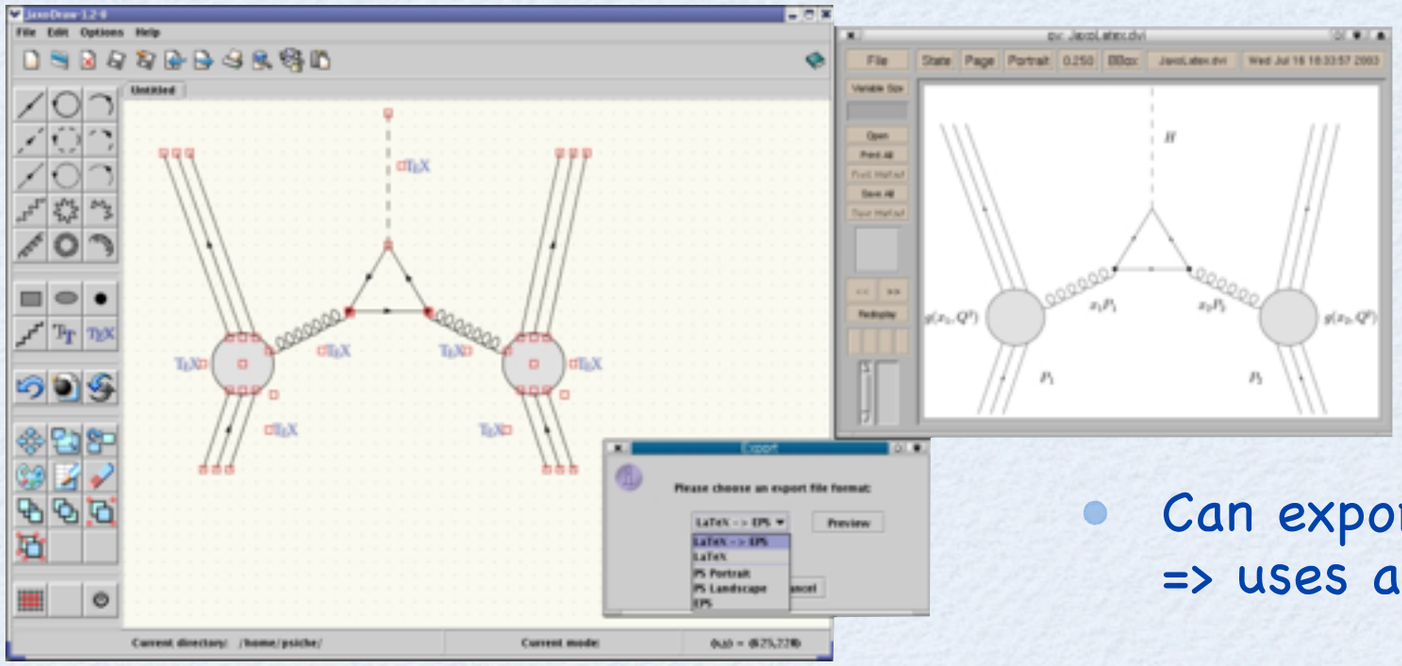
```
# ts wrapper script to make it behave like PBS's qsub command
# Last modified 16/11/09 veo
# Currently understood command-line options: -N (name of job) -o (stdout location)
# Known issues:
# 1-always takes the last string as the name of the process to be run
# 2-only bash scripts are properly handled, other shell scripts will need trivial modifications

echo $* |\
awk '{nid=0; soid=0;
      gtl="file "$NF; gtl | getline filetype;
      split(filetype,fta);
      for (i=1;i<NF;++i) { if ($i=="-N") nid=i+1; if ($i=="-o") soid=i+1; }
      print "ts",(nid?"-L "$nid:""),(soid?"sh -c '\''":"")(fta[2]=="Bourne-Again"?"bash ":"")$NF,(so
id?">"$soid"'\''":"");}' | awk '{system($0)}'
```

Not Reinventing the Wheel

- GNU Scientific Library (GSL) - www.gnu.org/software/gsl/
  - thread-safe numerical C library for many applied math topics
  - pros: no dependencies, extensive test suite, 1000+ functions
    - complex numbers, special functions, differential equations, FFT, histograms, n-tuples, random distributions, linear algebra, root-finding, minimization, least-squares fitting, physical constants,...
  - cons: many of these are done better/faster by specialized packages.
- Ex: FFTW, Fastest Fourier Transform in the West - www.fftw.org
  - C library district Fourier transform, competitive even with commercial codes. Threading support.
- Ex: GMP, GNU Multi-PRecision library - gmplib.org
  - C library used in GCC, GNU Classpath, in Mathematica, Maple, SAGE...
- Ex: Complex numbers are already in C99 standard. #include<complex.h>

Knowing your Real Needs



- JaxoDraw - jaxodraw.sourceforge.net
- Can export to postscript but also to latex
  => uses axodraw latex package

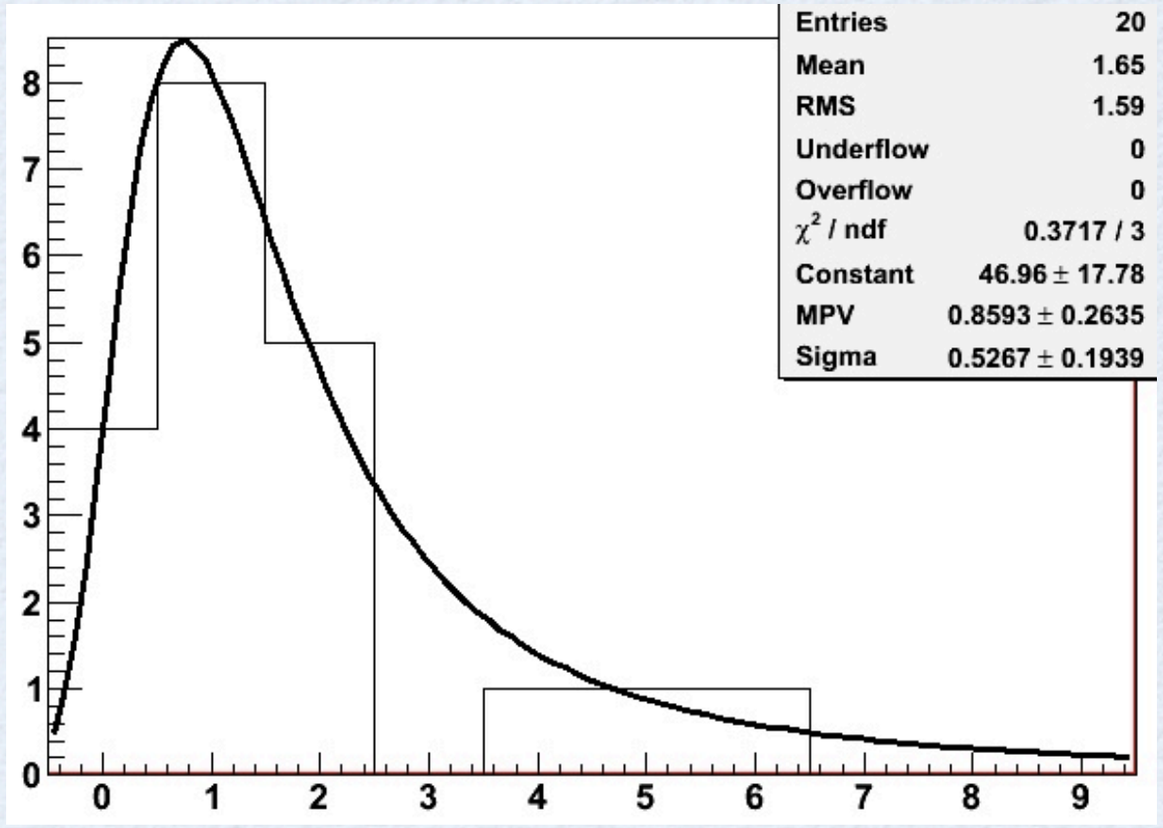
- UNU.RAN, Universal Non-Uniform RANdom number generators - statistik.wu-wien.ac.at/unuran
  - Pseudo-random number generation is the core of a good Monte Carlo generator.
  - Mersenne twister MT19937 has period of $2^{19937}-1$. It is fast. It passes many of the statistical tests, ex. DieHard tests. www.stat.fsu.edu/pub/diehard
  - Excellent for physics MC. Default generator in many modern libraries/languages, like python.
  - But if you want to use it for encrypting your data, it is useless!!!

Other FOSS Packages

- GNU R - www.r-project.org
  - "lingua franca among statisticians" - including people in finance, genetics
  - Interpreted programming language + software environment for statistical data analysis and graphical representation
- Java Analysis Studio - jas.freehep.org
  - Part of Freehep - JAVA based HEP & related software
- GNU Octave - www.gnu.org/software/octave/
  - Open-source Matlab alternative
- SAGE - www.sagemath.org
  - Open-source alternative to Maple, Mathematica, Matlab
    - An excellent list for more good stuff:
      - Andy Buckley's website www.insectnation.org/howto/academic-software

ROOT

- Among other packages, one is (unfortunately?) almost compulsory: ROOT - root.cern.ch
  - Covers everything needed for statistical data analysis: Graphing, fitting, histogramming, ...
  - Has bindings/wrappers for many other libraries: GSL, UNU.RAN, various MC programs, TMVA, Roofit, etc.



- Comes with a C++ interpreter for quick and DIRTY jobs.
  - Try its python interface: pyroot root.cern.ch/drupal/content/how-use-use-python-pyroot-interpreter
- Lecture by Dr. Çobanoğlu next Monday will have the details.

Using New Software

- Whenever you are supposed to start using a new software tool/package, be PATIENT. Investment in the beginning, ALWAYS pays off.
  - Try to install the package from scratch
  - Search the web for tutorial lectures and follow them
  - Try to run basic examples
  - Don't jump into implementing something complex - first outline a very basic "project" for testing the package and your understanding of it.
  - After these make sense, then start complex coding.
- For example, if you are supposed to learn Geant4, a simple project would be shooting muons at a thin rectangular prism and seeing if the lost energy fits a Landau distribution. If you are supposed to learn ROOT, you could fill a histogram with randomly generated numbers and do a fit to see if the generation agrees with extracted results.

Closing Advice

- Before doing any TDAQ programming, please make sure you know the following concepts by heart:
  - Compiler, interpreter, representation of objects in a computer's memory, pointers, passing by reference, etc., ie. what is under the hood.
  - If you feel you are not as comfortable with these concepts as you like, have a look at the excellent video lectures on the web.
    - Personal recommendation: Stanford CS107 lectures by J. Cain. It also contains some more interesting stuff like functional programming.
- Please think, then implement.
  - For a really smart solution for a tough programming problem, you can even think for days before implementing. (Take the problem on the backburner, do other things but brainstorm in the breaks.)
- Consider reading "basic stuff" before bugging people you don't know (like on mailing lists.)
  - Browse through readme files, use wikipedia, google, etc.
  - When you bug them, provide code snippets, software versions, etc.

Conclusion

- This "lecture" is full of starting points, it needs you to follow up...
  - It is full of stuff that will make your life easy. After you start using them, you might get surprised how you lived without them before.
  - But there is no "free lunch". They need a minimum amount of investment from you.
    - So pick some of the leads from this talk and start playing with them.
    - If you start testing them today, you can get direct help from us!
  - Examples to try: download and compile ts and try to push the CPU utilisation of your n-core machine to 100%; install git and start a repository; run screen on a remote terminal, kill the connection, reconnect and continue from where you left; do the exercise mentioned on slide "Final Code 3"; create a few fake .mp4 files and run the example command on slide "Swiss Army Knife" up to one pipe (|) at a time to understand what it does,...

BACKUPS

An Example .Bashrc

```
# Turn off clobbering
set -C

# Change default prompt
export PS1='\[\e]0;\w\a\]\[\e[32m\]\u@\h \[\e[35m\]\d \t \[\e[33m\]\w\[\e[0m\]\n\$ '

# Don't put duplicate lines in the history
export HISTCONTROL="ignoredups"

# Default to human readable figures
alias df='df -h'
alias du='du -h'

# Colorful commands - cmake.pl from the web
alias grep='grep --color'
alias ls='ls -G'
alias make='~/work/scripts/cmake.pl'

# stop processes from command line
alias stop='/bin/tcsh -c "stop \$argv"'
# ssh through tor (very slow)
alias tor-ssh='ssh -C -o ProxyCommand="nc -X 4 -x localhost:9050 %h %p"'

alias cd='push_cd'

# Mac OSX specific
alias ldd='otool -L'
alias lyx='/Applications/LyX.app/Contents/MacOS/lyx'
alias wget='curl -O'
```

- Red part follows the black part.

```
# function to run upon exit of shell
function _exit()
{ echo -e "Bye bye..."
  sleep 0.2
}
trap _exit 0

# defining pushcd to use pushd instead of cd
function push_cd
{
        if [ $# -ge 1 ]
        then
                if [ "$1" = "-" ]; then
                        if ( dirs | awk '{exit ($1==$3)}' ) ; then
                            pushd - > /dev/null
                        else
                            popd > /dev/null 2>&1
                        fi
                else
                        pushd "$@" >/dev/null
                fi
        else
                pushd $HOME >/dev/null
        fi
}
```

Surviving Windows = Cygwin

- Unlike most of my colleagues, I am not against using Windows as the OS of your development machine. However, IMHO, Windows is ok only if you have installed Cygwin.
  - GNU Screen is not available in the package list, but you can find it on the web.
  - A list of packages that I would immediately install in cygwin: autoconf, automake, bash, binutils, emacs-X11, gcc (with g++, g77, and possibly java), gcc4 (similar to gcc), gcc-mingw, git, make, openssh, subversion, tetex, xterm, xz + all their dependencies
  - ROOT in cygwin: While on its official website, ROOT is "not recommended" for use with cygwin gcc, I have used it for many years and encountered no problems. If you do not want to set up Visual Studio, I would recommend compiling ROOT in cygwin.

Notes & Licenses

- PS: I am aware of the small "problem" in the suffix() function shown on slide number 4. :-)
- The ts wrapper script on slide 24 is hereby licensed under GPLv3. Everything else in this presentation (including the images) is hereby released under Creative Commons Attribution-ShareAlike 3.0, except for the Bogaziçi University logo and the screenshot shown on slide 26, which has been taken from the jaxodraw website - it has been reduced in resolution and I believe its use like this falls under fair use conditions.
- These lectures have been prepared for the ISOTDAQ schools in Ankara, Rome and Cracow.