ROOT 6 on Windows

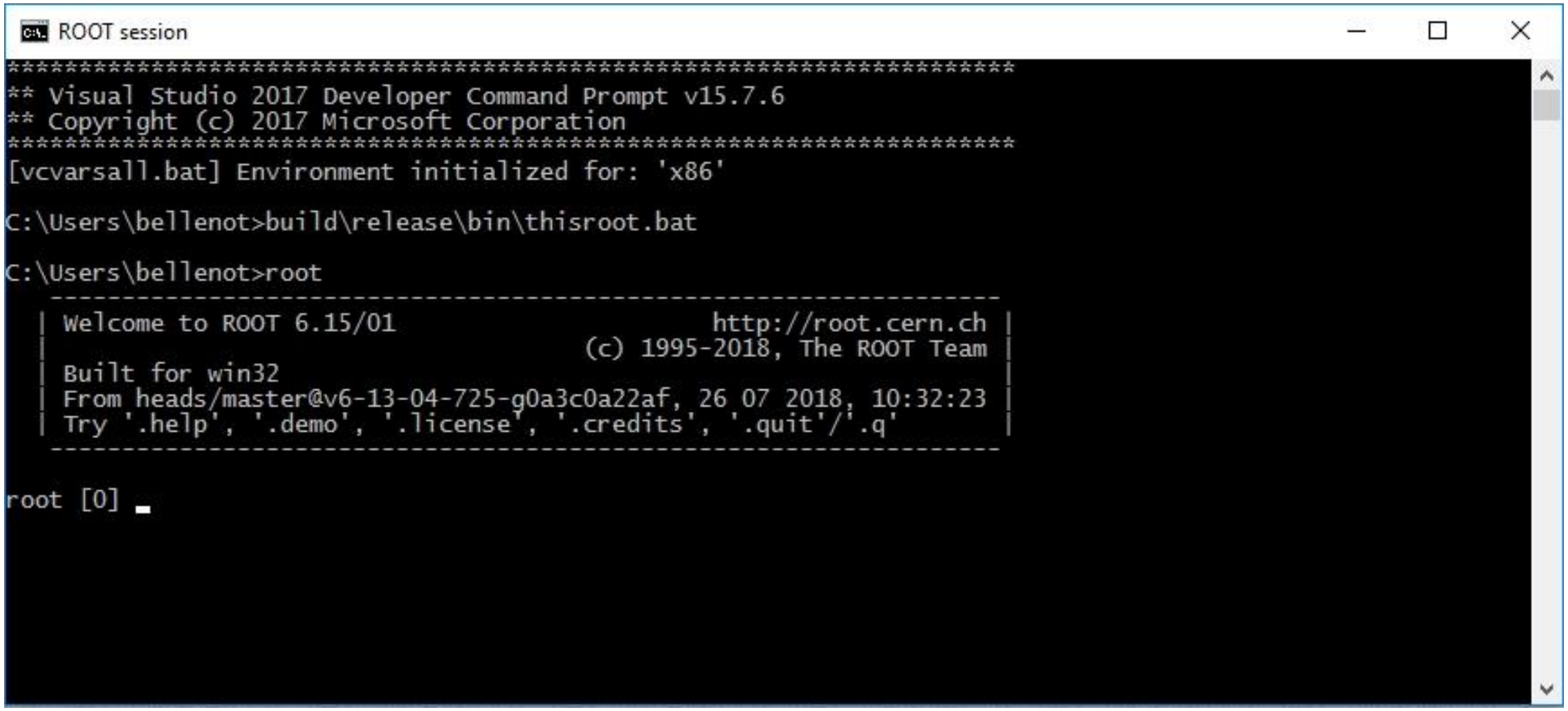
(Finally)

Bertrand Bellenot (CERN EP-SFT)

Introduction

After many years without any ROOT 6 release on Windows, we got our first ***preview*** release of ROOT 6 on Windows (6.14/00) and Visual Studio 2017.

preview Windows Visual Studio 2017 (dbg)	root_v6.14.04.win32.vc15.debug.exe	167M
preview Windows Visual Studio 2017 (dbg)	root_v6.14.04.win32.vc15.debug.zip	271M
preview Windows Visual Studio 2017	root_v6.14.04.win32.vc15.exe	73M
preview Windows Visual Studio 2017	root_v6.14.04.win32.vc15.zip	98M



Common Issues

Build infrastructure (CMake):

Windows was completely forgotten in the master, and that was requiring a lot of changes… For example, on Windows, one has to ***export*** the symbols from the objects to make them visible (accessible) from the shared library (DLL):

```
if(MSVC)
  set_target_properties(cling-demo PROPERTIES WINDOWS_EXPORT_ALL_SYMBOLS 1)
  set_property(TARGET cling-demo APPEND_STRING PROPERTY LINK_FLAGS
               "/EXPORT:?setValueNoAlloc@internal@runtime@cling@@YAXPEAX00D_K@Z
                /EXPORT:?setValueNoAlloc@internal@runtime@cling@@YAXPEAX00DM@Z
                /EXPORT:cling_runtime_internal_throwIfInvalidPointer")
endif()
```

BTW, even CMake had to be modified, to be able to export symbols from an executable (in order to be able to call functions from it):

```
set_target_properties(rootcling PROPERTIES WINDOWS_EXPORT_ALL_SYMBOLS 1)
```

Many thanks to the CMake developers who kindly accepted my requests…

Common Issues

C++ Standard:

In Visual Studio, until recently, “the __cplusplus macro has stubbornly remained at the value “199711L”, indicating (erroneously!) that the compiler conformed to the C++98 Standard” (quoting the Visual C++ Team Blog). So for example this simple line:

```
#if __cplusplus < 201402L
```

Had to be changed to:

```
#if ((__cplusplus < 201402L && !defined(R__WIN32)) ||
     (defined(_MSC_VER) && _MSC_VER < 1800))
```

The good news is that Visual Studio support most of the C++11, C++14, and C++17 standards

C++ non-Standard:

```
#include <process.h>
#define getpid() _getpid()
#define srandom(seed) srand(seed)
#define random() rand()
```

Common Issues

Variable Length Arrays:

```
double solution[TestFixture::fNumParams];
```

Here, ‘solution’ is a variable length array and is not standard in C++. Some compilers like GCC allow them as an extensions but Visual Studio will give the following error:

```
error C2131: expression did not evaluate to a constant
```

So one obvious solution would be:

```
double *solution = new double[TestFixture::fNumParams];
[...]
delete[] solution;
```

But then one has to delete the array to not create a memory leak. So the best solution is to use a STL collection, which handles the memory management for you:

```
std::vector<double> solution(TestFixture::fNumParams);
```

Common Issues

When formatting a pointer in a string:

```
sstr << "int& ref = *(int*)" << &res << ';';
```

on Windows, to prefix the hexadecimal value of a pointer with '0x', one need to write:

```
sstr << "int& ref = *(int*)" << std::hex << std::showbase << (size_t)&res << ';';
```

On Windows, (as the standard says) calling front() on an empty std::vector causes an undefined behaviour. One must check that the container contains something using empty() before calling front(). For example, this code:

```
fCoordErrorsPtr[i] = &fCoordErrors[i].front();
```

Would causes an undefined behaviour and should be:

```
fCoordErrorsPtr[i] = fCoordErrors[i].empty() ? 0 : &fCoordErrors[i].front();
```

Common Issues

And since `min()` and `max()` are defined as macros in Visual Studio, we had to enclose them in parentheses to prevent compilation errors:

```
(std::numeric_limits<unsigned>::max)()
```

We also had to deal with this kind of error from Clang itself:

```
cannot mangle this template type parameter type yet
```

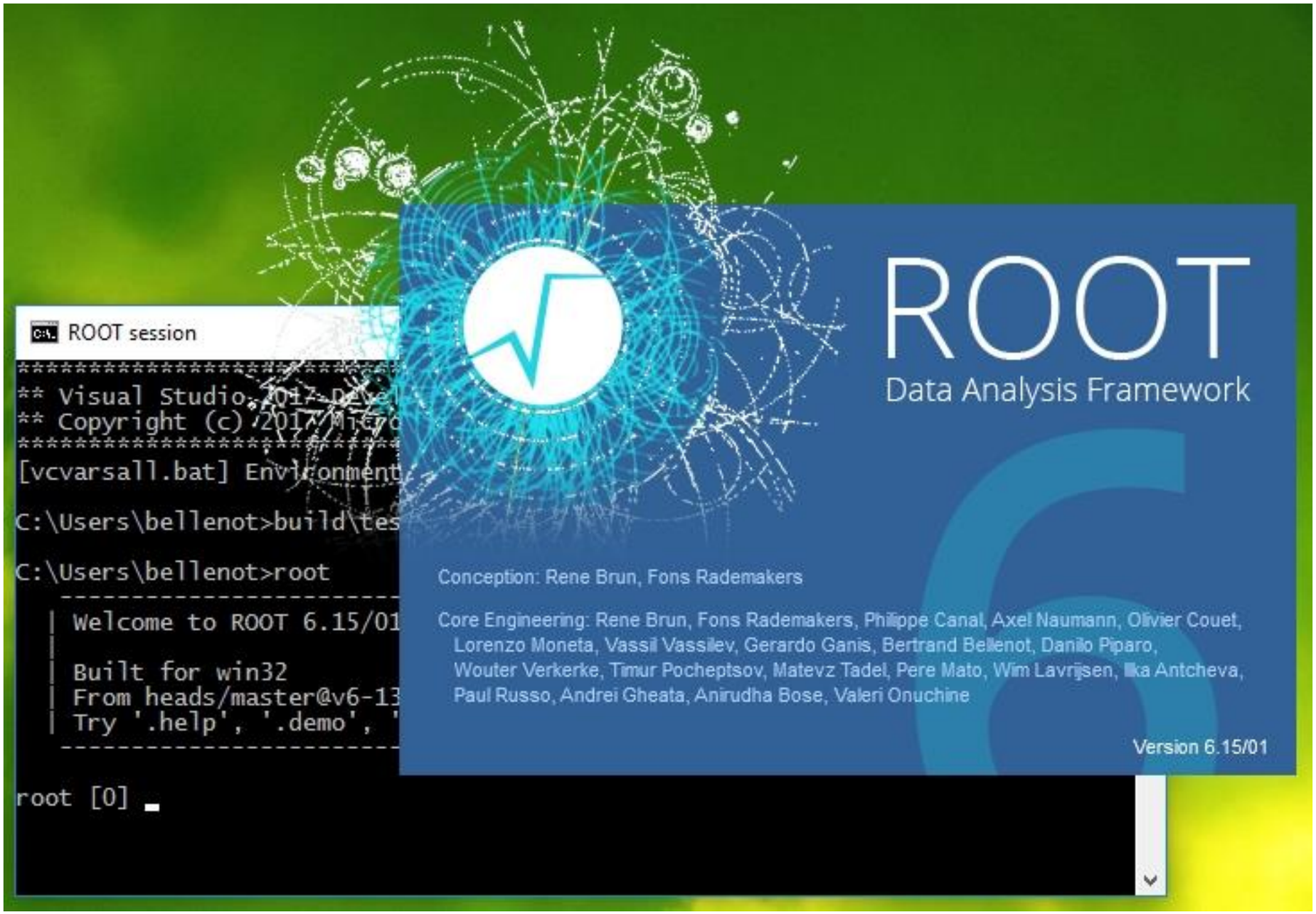
There are still an open issue in Visual Studio (wrong error C2668 - ambiguous call to overloaded function). And after the latest update of Visual Studio, we now have:

```
fatal error C1001: An internal error has occurred in the compiler
```

And before trying to build ROOT on Windows, we had to fix Cling… But I'm not going to talk about it here…

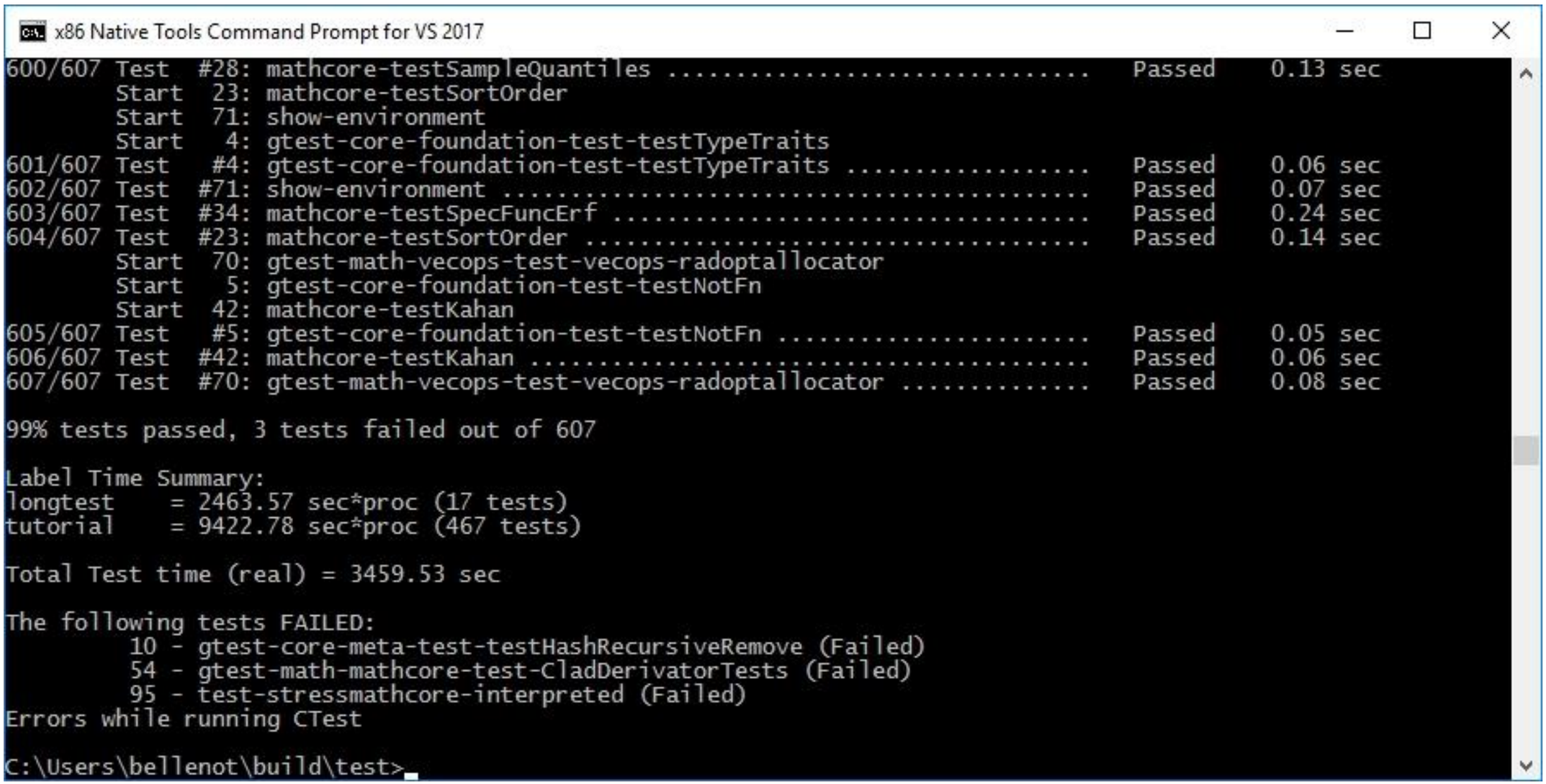
Common Issues

Even the splash screen had to be (almost completely) re-written on Windows!



Current Status

ROOT 6 now works on Windows, and most of the tests pass:



Current Status

The status of roottest is not that good, mostly due to the use of .so extensions in the code, relying on posix tools (e.g. grep), shell scripts, and so on…

x86 Native Tools Command Prompt for VS 2017

```
        Start 778: roottest-root-treeproxy-make
779/779 Test #778: roottest-root-treeproxy-make ................................................
.........***Not Run (Disabled)   0.00 sec

63% tests passed, 242 tests failed out of 653

Label Time Summary:
cling          = 248.56 sec*proc (93 tests)
longtest       = 138.06 sec*proc (24 tests)
matrix         =   1.39 sec*proc (1 test)
regression     = 243.41 sec*proc (92 tests)
roottest       = 249.95 sec*proc (94 tests)

Total Test time (real) = 1424.05 sec

The following tests did not run:
          1 - roottest-cling-array-runarray1 (Disabled)
         47 - roottest-cling-operator-runEqualTest (Disabled)
         50 - roottest-cling-operator-ConversionOp (Disabled)
         80 - roottest-cling-reflex-make (Disabled)
        130 - roottest-cling-template-separateDict-make (Disabled)
        131 - roottest-cling-template-separateDictNamespace-make (Disabled)
        144 - roottest-cling-typedef_global-scopeTest (Disabled)
        167 - roottest-root-collection-runDeleteWarning (Disabled)
        173 - roottest-root-core-execStatusBitsCheck (Disabled)
        252 - roottest-root-html-runMakeIndex (Disabled)
        253 - roottest-root-io-abstractclass-make (Disabled)
        254 - roottest-root-io-alloc-make (Disabled)
        255 - roottest-root-io-arrayobject-make (Disabled)
        256 - roottest-root-io-bigevent-make (Disabled)
```

Missing bits and pieces

Here is a list of what is missing and the possible reasons (when understood)

- PyROOT: It compiles, but doesn’t work, due to some symbols resolution problems
- TMVA: The dictionary generation is failing
- Davix: Not supported on Windows (major issue!)
- FFTW3: Not tested yet
- Mathmore: Not tested yet (GSL is not easy to get on Windows)
- Pythia 8: Pythia 8 itself has to be ported on Windows, but in principle it should work (I have a running version)

Portable Code

- Please use the methods provided by TSystem! Most of the time, you will find what you need
- Avoid calling low-level functions like dlopen(), dlsym(), and use #ifndef if you have no other choice:

```
#ifndef _MSC_VER
#include <unistd.h>
#endif
```

- Think cross-platform from the beginning! It's much more complicated to port existing code than doing it right from the beginning (please believe me…)

The good news is that now we have a couple of Windows nodes in Jenkins for the PRs, the incremental, and the nightly builds, so we can detect those potential issues early enough

Windows 64

The Windows 64 port is another story. The major issue is coming from the conversion of pointer to Long_t (or Ulong_t) happening a bit everywhere in ROOT. For example:

```
TCanvas *TROOT::MakeDefCanvas() const
{
   return (TCanvas*)gROOT->ProcessLine("TCanvas::MakeDefCanvas();");
}
```

With:

```
Long_t TROOT::ProcessLine(const char *line, Int_t *error)
```

pointer is 64bit, long is 32bit, cast breaks pointers!

And there are other potential problems with some of the components (e.g. GDK) that might have some issues being compiled on Win64.

Future work

- Add missing features/packages
- Make all tests (of supported features) passing
- Make roottest successful (disable unsupported tests)
- Port to 64 bit (first evaluation)

Thank you for your attention!