

Replacing the Geant4 Build System with Modern Tools

Ben Morgan

THE UNIVERSITY OF
WARWICK

1: Why Replace the Current Buildsystem?

- Metaconfig based Configure script becoming increasingly problematic
 - *System checks (e.g. libraries) not comprehensive*
 - complete rewrite to fix...
 - *“Configure -ldlibs” type operations broken*
 - Actually a misuse of Metaconfig...
 - *Makefiles controlled by Environment Variables*
 - fragile build of toolkit and applications.
- More seriously
 - *Metaconfig should be considered OBSOLETE* – no upstream support.
 - *We have a requirement to produce binary packages* – better integration needed.

2: Considerations for Building Geant4

- ***Geant4 is ultimately a straightforward piece of software***
 - Dynamic and possibly static libraries (runtime/archive part).
 - Header and configuration files (files part)
- ***From the perspective of building the toolkit there are complexities***
 - Global vs Granular library builds (or even both!)
 - Libraries have optional components (e.g. GDML in libG4persistency)
 - Libraries require external packages (e.g. CLHEP)
 - *Inter-library and external library dependencies.*
 - Need to build on all supported platforms (Linux flavours, Windows, Mac OSX).
- ***From the user perspective, we need to provide tools for working with G4***
 - “Easy build” tool for simple applications (like current Makefile system)
 - Tool for querying a Geant4 install for headers/libraries for advanced users.

3: Evaluation of Replacement Systems

- Evaluation process identified requirements (*open for discussion!!*) for any replacement tool:
 - *Must work and be supported upstream on all supported Geant4 platforms.*
 - *Must allow Geant4 to be built with all current features.*
 - *Must have an easy to use interface for users and developers.*
 - *Must be easy to use and maintain by the build system developer.*
 - *Should have a minimal set of tools to install, ideally one.*
 - *Should integrate with binary packaging systems with minimal effort.*

4: Choice of Tools for Evaluation

- An evaluation of several well regarded build tools, used on small and very large projects, has been carried out.
 - **GNU Autotools**
 - **SCons**
 - **CMake**
- Somewhat arbitrary – tools with largest support and user communities chosen
 - Reject internally developed tool as too much work and reinvention of the wheel!
- *Tools were used to prototype a partial build of Geant4*
 - Evaluate system checks, global vs granular lib builds, general ease of use.

5: GNU Autotools

- The “classic” suite of Autoconf/Automake/Libtool
- ***Advantages:***
 - “Standard” of sorts on *NIX systems.
 - Many system checks already written (e.g. X11, Qt).
 - Familiar and easy to use interface “configure && make && make install”
- ***Disadvantages:***
 - 2-3 tools to understand and maintain (phrase “auto-hell” is quite common!)
 - No native Windows builds (?), requires Cygwin/MSYS layer.
 - Documentation a little opaque, very few “canonical” examples.
- ***Autotools usable for Geant4 but reject it due to complicated use on Windows, and possible issues with maintenance in the future.***

6: SCons

- Relatively new buildsystem based on the Python language.
 - *Evaluation for Geant4 is still at an early stage.*
- **Advantages:**
 - Complete system in one tool – build is literally “scons <options>”
 - Supports cross-platform builds.
 - Scripted in Python, so familiar syntax.
- **Disadvantages:**
 - Quite low level – many aspects of Geant4 build require (opaque..) Python coding.
 - Scalability issues(?) – evidence that larger projects have moved away from SCons.
 - Requires Python *and* SCons (and just because it's Python doesn't mean it's good!!)
- ***SCons of potential use for Geant4, reject it as frontline system due to level of coding needed to support Geant4 – Evaluation will continue!***

7: CMake

- Has existed since early this century, prominence in last few years - scripted in “CMake language”, underlying system in C++.
- ***Advantages:***
 - Cross-platform support “out the box” and designed to be so.
 - “Buildscript generator” - output GNU Makefiles, Eclipse/Visual Studio/XCode projects.
 - Many “out the box” system checks (X11, Qt etc), new checks via simple “recipe”
 - Very clean and friendly user interfaces.
- ***Disadvantages:***
 - Requires CMake, plus at least one tool to run buildscripts.
 - Limited documentation – offset by very active user/developer community.
 - Need to learn CMake scripting language (though very intuitive)
- ***Geant4 CMake prototype by far the easiest to develop and use– very intuitive scripting, clean interfaces – RECOMMENDED as frontline system.***

8: Geant4 CMake Build Prototype

- Current features:
 - Implements a check for CLHEP (though needs cross-platform work)
 - Builds of global/granular shared and static libraries.
 - Now in process of implementing full cross-platform and external package checks.
- Downloadable from
 - <http://www2.warwick.ac.uk/fac/sci/physics/staff/research/bmorgan/geant4/geant4-9.2.1.tar.bz2>
 - Just unpack, and the README and README.cmake provide instructions
- ***Moving to CMake brings many advantages for Geant4 developers and users.***

9: CMake Work Cycle: Installing CMake

- In most cases, users and developers won't have to install CMake
 - Packaged on all mainline Linux distributions for instance.
- I do not consider having to install CMake a show stopper!
 - Installing CMake is trivial – compared to other tools Geant4 needs!
 - Binary installers for all platforms available from <http://www.cmake.org>
- We could, depending on licensing issues, even distribute it with Geant4.
 - I don't believe this to be necessary.
- One also needs a build tool – Make, Visual Studio, Xcode.
 - Expect people building from source will have these already!

10: CMake Work Cycle: Configuration Setup

- Geant4 CMake enforces an “out of source” build.
 - Keeps generated files like Makefiles out of source tree, where they might get committed to source control accidentally.



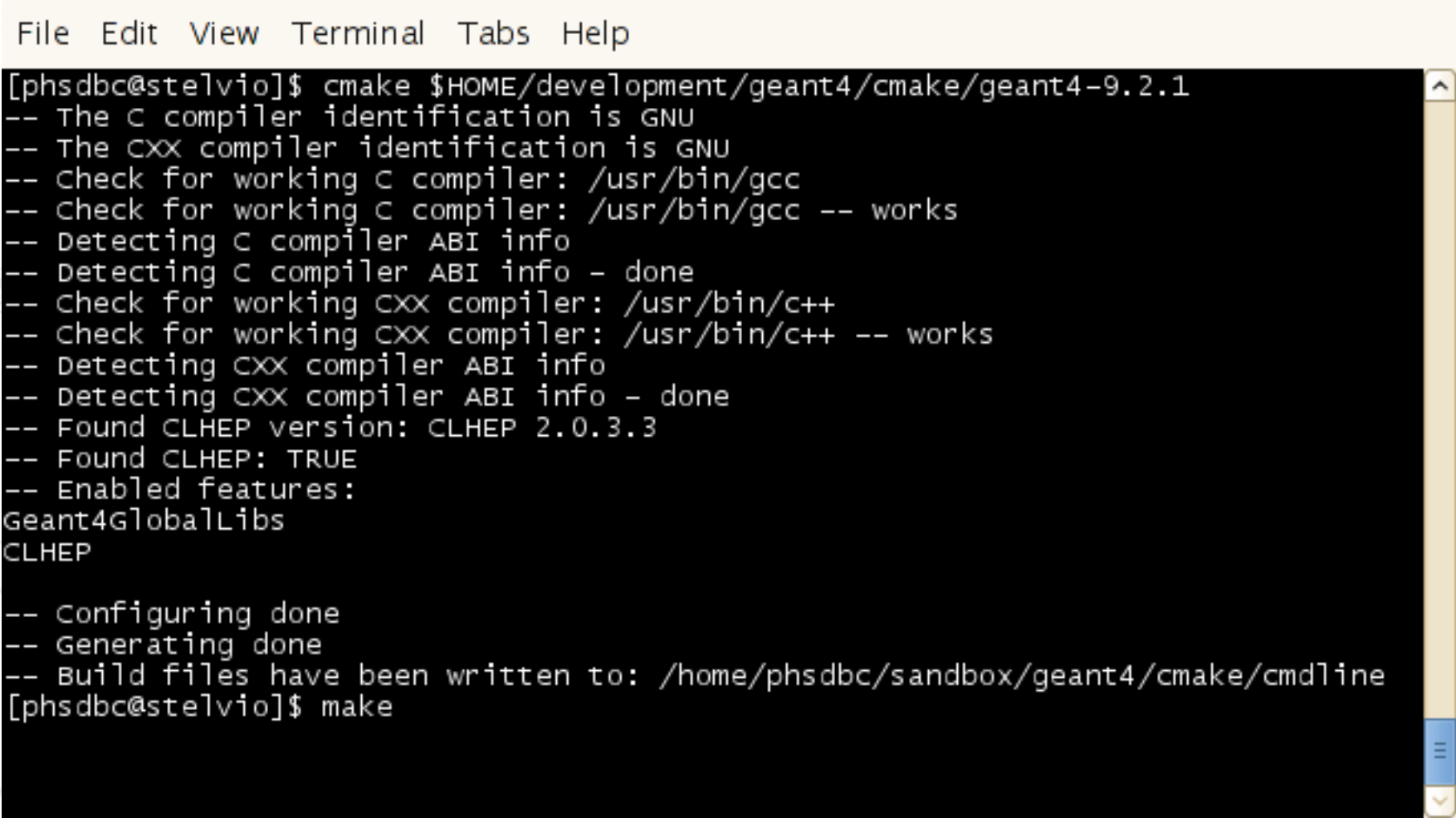
- ***This brings several major advantages for the user/developer:***
 - Create ***different build configurations*** but only edit ***one set of code***
 - e.g. you want to check your code runs with Qt3 AND Qt4.
 - Edit source tree only – each build tree knows about changes when rebuilding
 - CMake stores build configuration in a cache file – ***no environment variables!!***

11: CMake Work Cycle: Running Configuration

- You run CMake in the build directory to generate buildscripts for chosen tool.
- Familiar configure step
 - Choose build options, e.g. dynamic granular libraries, GDML support
 - Check for needed system features.
- Real benefit of CMake – choice of clean, friendly user interfaces.
 - Command line
 - Ncurses
 - Metaconfig-like question and answer
 - GUI (Windows).

12: CMake Command Line

- Syntax:
 - “cmake <options> <path to source tree>”

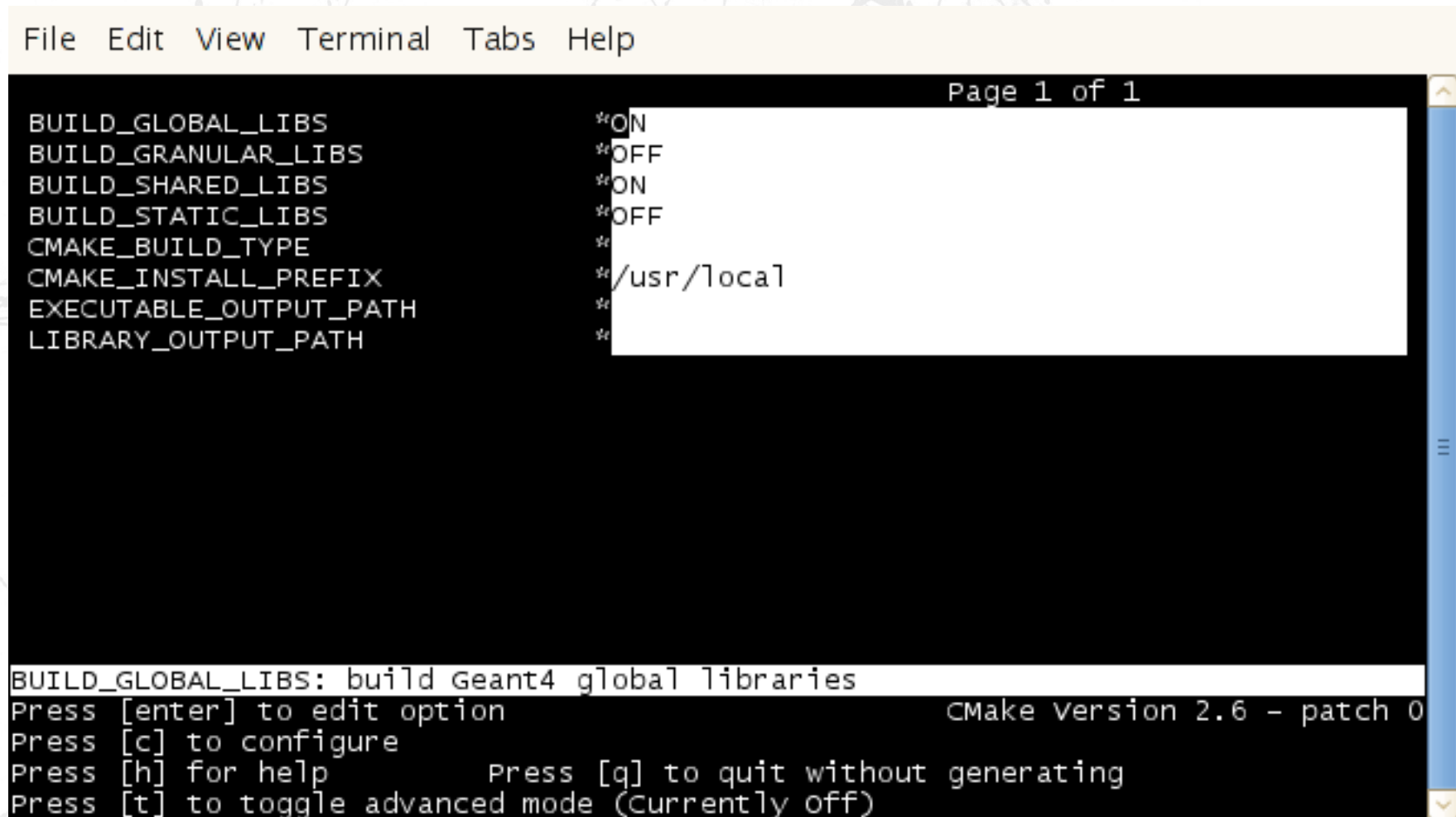
A terminal window with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a black background with white text. The text shows the execution of the 'cmake' command in a shell. It displays various status messages about compiler identification (GNU), checking for working compilers (gcc, g++), detecting ABI info, finding CLHEP version (2.0.3.3), and enabling features (Geant4GlobalLibs, CLHEP). It concludes with 'Configuring done', 'Generating done', and the path where build files were written. The prompt then changes to 'make'.

```
File Edit View Terminal Tabs Help
[phsdbc@stelvio]$ cmake $HOME/development/geant4/cmake/geant4-9.2.1
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Found CLHEP version: CLHEP 2.0.3.3
-- Found CLHEP: TRUE
-- Enabled features:
Geant4GlobalLibs
CLHEP

-- Configuring done
-- Generating done
-- Build files have been written to: /home/phsdbc/sandbox/geant4/cmake/cmdline
[phsdbc@stelvio]$ make
```

13: CMake NCurses

- Syntax:
 - “ccmake <path to source tree>”



The screenshot shows the CMake ncurses interface. At the top is a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. Below the menu bar, on the right, is a status bar that says 'Page 1 of 1'. The main area is a list of configuration options, each with a status indicator: BUILD_GLOBAL_LIBS is ON, BUILD_GRANULAR_LIBS is OFF, BUILD_SHARED_LIBS is ON, BUILD_STATIC_LIBS is OFF, CMAKE_BUILD_TYPE is *, CMAKE_INSTALL_PREFIX is */usr/local, EXECUTABLE_OUTPUT_PATH is *, and LIBRARY_OUTPUT_PATH is *. At the bottom, there is a prompt 'BUILD_GLOBAL_LIBS: build Geant4 global libraries' and instructions: 'Press [enter] to edit option', 'Press [c] to configure', 'Press [h] for help', 'Press [q] to quit without generating', and 'Press [t] to toggle advanced mode (Currently off)'. The CMake version 'CMake Version 2.6 - patch 0' is also displayed.

```
File Edit View Terminal Tabs Help

Page 1 of 1

BUILD_GLOBAL_LIBS      *ON
BUILD_GRANULAR_LIBS   *OFF
BUILD_SHARED_LIBS     *ON
BUILD_STATIC_LIBS     *OFF
CMAKE_BUILD_TYPE      *
CMAKE_INSTALL_PREFIX  */usr/local
EXECUTABLE_OUTPUT_PATH *
LIBRARY_OUTPUT_PATH   *

BUILD_GLOBAL_LIBS: build Geant4 global libraries
Press [enter] to edit option
Press [c] to configure
Press [h] for help
Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently off)

CMake Version 2.6 - patch 0
```


14: CMake Interactive

- Enabled through the “-i” option on the command line interface:

File Edit View Terminal Tabs Help

```
[phsdbc@stelvio]$ cmake -i $HOME/development/geant4/cmake/geant4-9.2.1  
would you like to see advanced options? [No]:  
Please wait while cmake processes CMakeLists.txt files....
```

```
Variable Name: BUILD_GLOBAL_LIBS  
Description: build Geant4 global libraries  
Current value: ON  
New Value (Enter to keep current value):
```

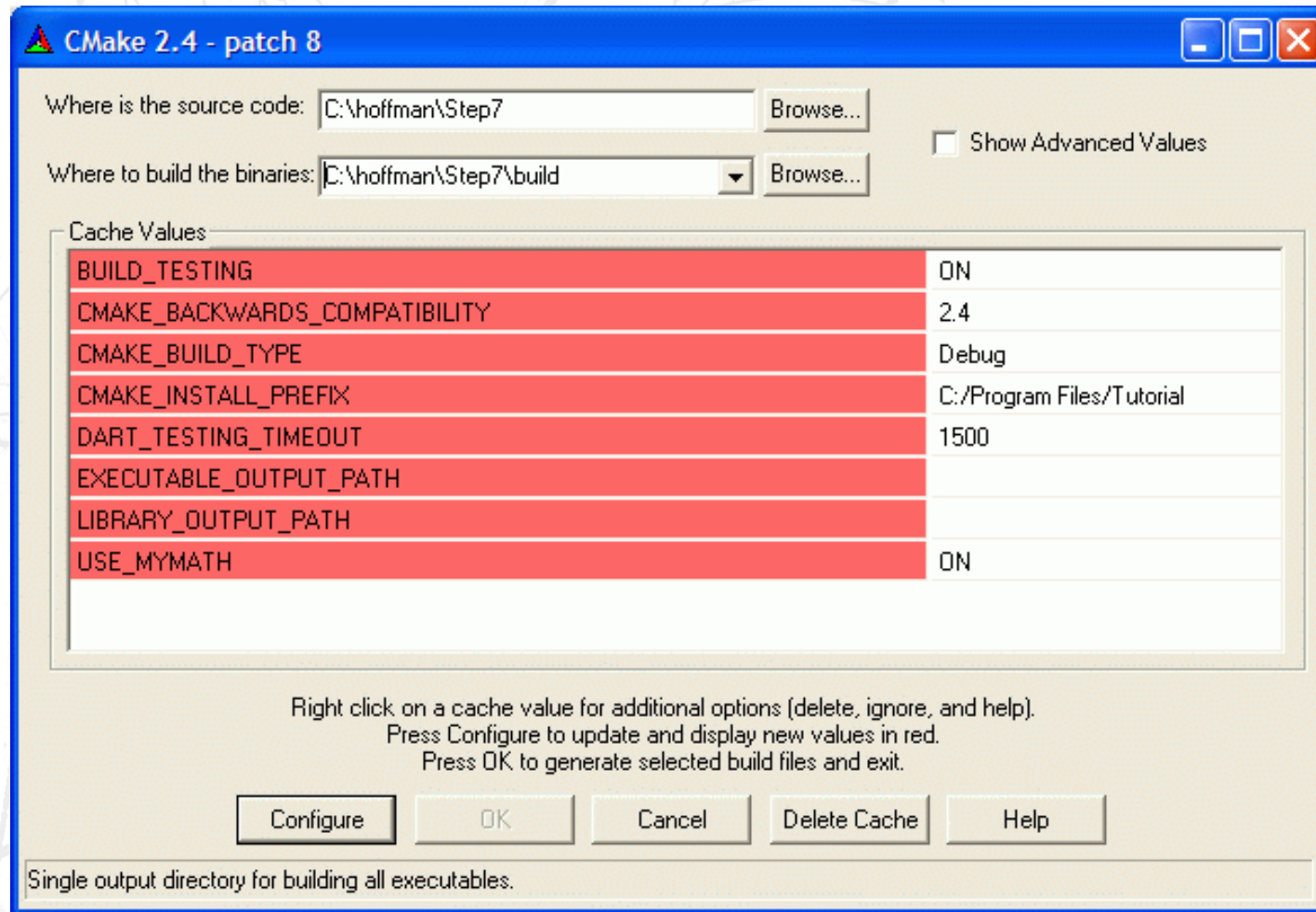
```
Variable Name: BUILD_GRANULAR_LIBS  
Description: build Geant4 granular libraries  
Current value: OFF  
New Value (Enter to keep current value):
```

```
Variable Name: BUILD_SHARED_LIBS  
Description: build Geant4 dynamic libraries  
Current value: ON  
New Value (Enter to keep current value):
```

```
Variable Name: BUILD_STATIC_LIBS  
Description: build Geant4 static libraries  
Current value: OFF
```

15: CMake Gui

- On Windows, can use Cygwin, but also a GUI

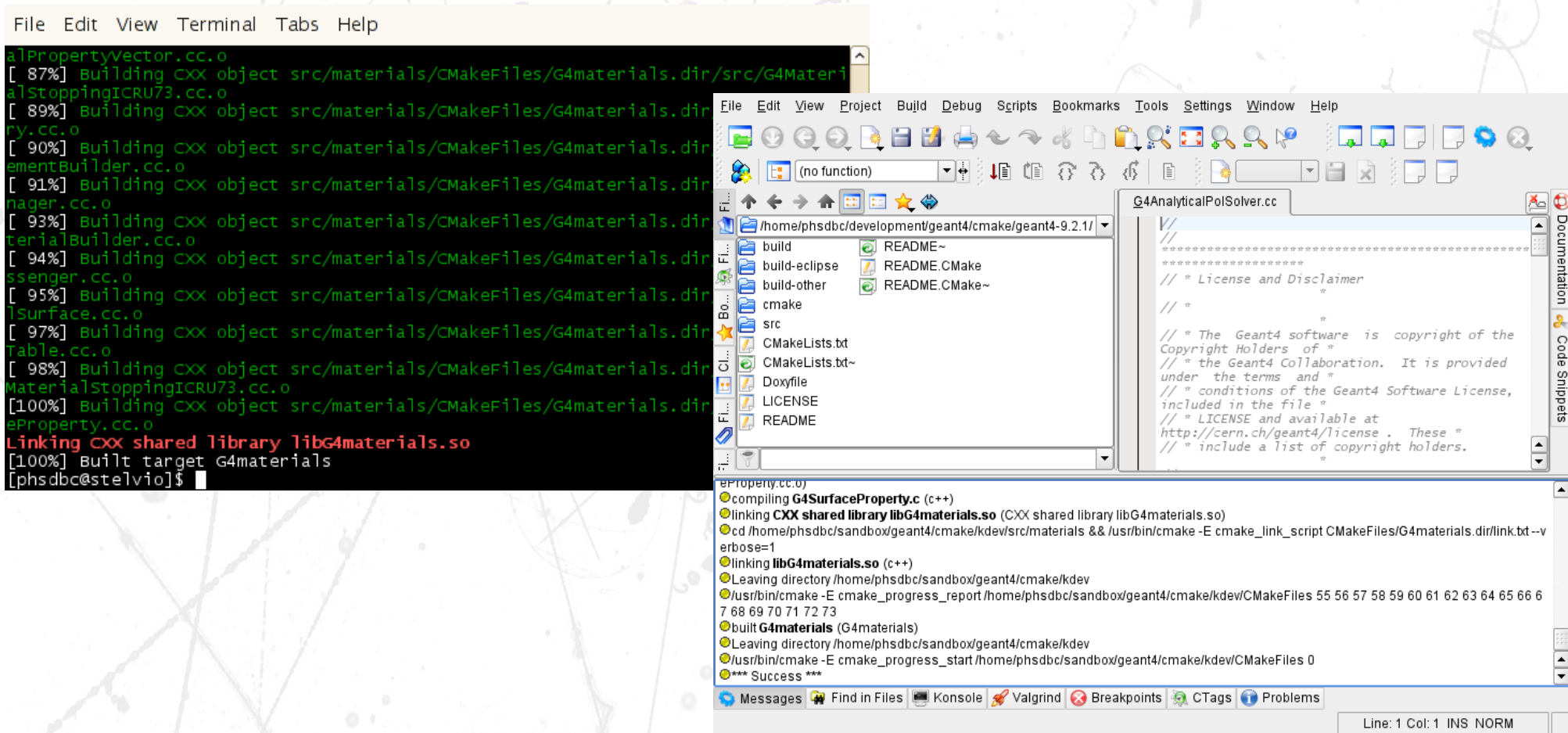


16: CMake User Interfaces Summary

- All these interfaces may just seem like eye-candy.
- Emphasize that they are *very intuitive*.
- Ncurses on Linux and GUI on Windows very useful for seeing how your build is configured.
 - Think these will be exceptionally useful for end users.
 - Also, storage of build configuration in cache files will help us to debug user problems.
- *Much cleaner than controlling through the environment!*

17: CMake Work Cycle: Build

- Running CMake generates – Makefiles or a Kdevelop3 project or a Visual Studio solution etc etc (you can choose!!).
- You then simply use the generated buildscripts with the appropriate tool:



The screenshot displays a terminal window on the left and an IDE window on the right. The terminal shows the execution of CMake commands, including building various CXX objects and linking a shared library. The IDE window shows the project structure and the source code of G4AnalyticalPolSolver.cc.

```
File Edit View Terminal Tabs Help
[ 87%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialPropertyVector.cc.o
[ 89%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialStoppingICRU73.cc.o
[ 90%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialElementBuilder.cc.o
[ 91%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialNuclearBuilder.cc.o
[ 93%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialTableBuilder.cc.o
[ 94%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialStoppingICRU73.cc.o
[ 95%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialSurface.cc.o
[ 97%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialTable.cc.o
[ 98%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialStoppingICRU73.cc.o
[100%] Building CXX object src/materials/CMakeFiles/G4materials.dir/src/G4MaterialPropertyVector.cc.o
Linking CXX shared library libG4materials.so
[100%] Built target G4materials
[phsdbc@stelvio]$
```

The IDE window shows the project structure and the source code of G4AnalyticalPolSolver.cc. The project structure includes files like build, build-eclipse, build-other, cmake, src, CMakeLists.txt, CMakeLists.txt~, Doxyfile, LICENSE, and README. The source code of G4AnalyticalPolSolver.cc is shown in the editor, including a license disclaimer.

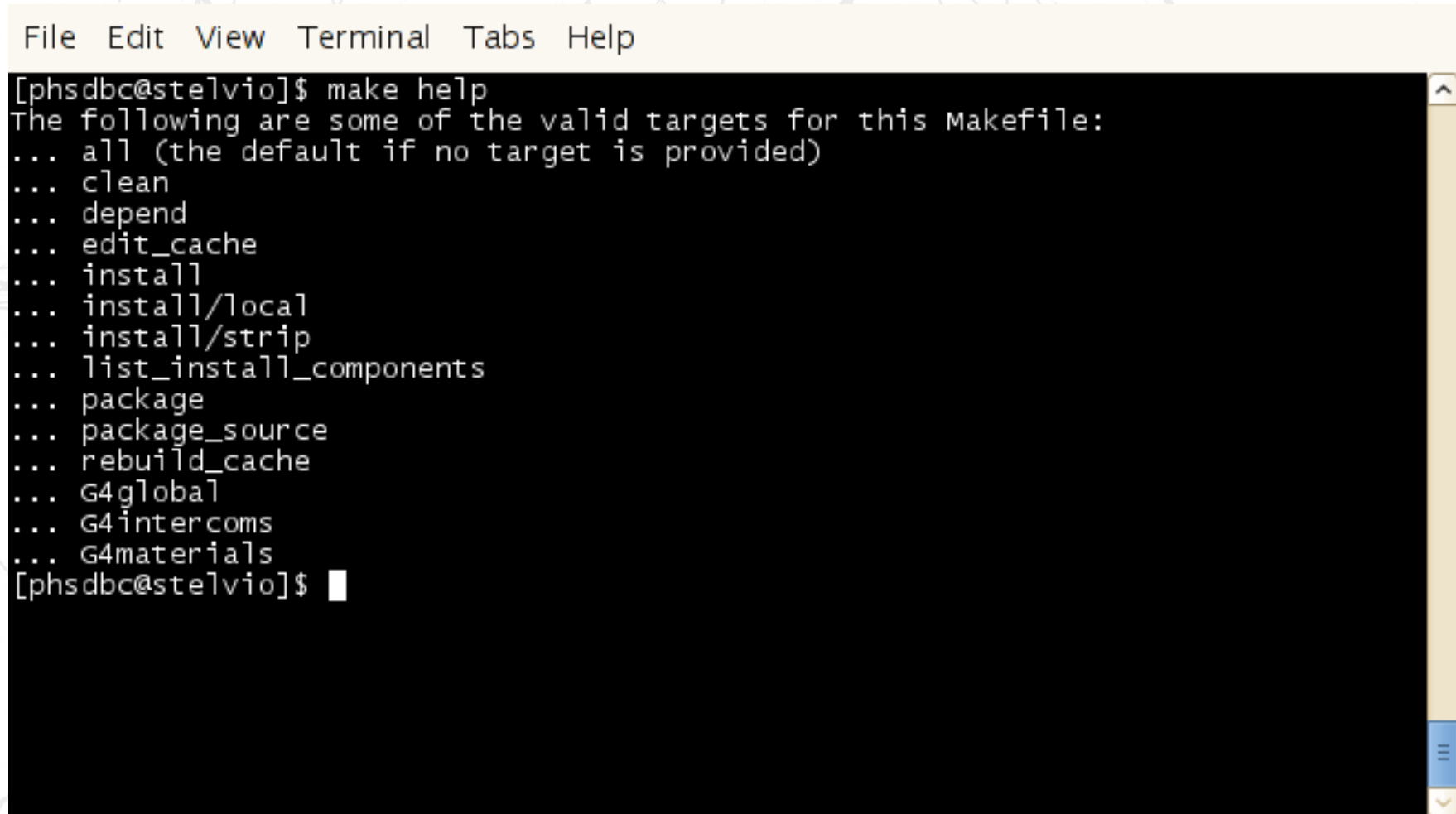
```
G4AnalyticalPolSolver.cc
//
// *****
// * License and Disclaimer
// *
// * The Geant4 software is copyright of the
// * Copyright Holders of *
// * the Geant4 Collaboration. It is provided
// * under the terms and *
// * conditions of the Geant4 Software License,
// * included in the file *
// * LICENSE and available at
// * http://cern.ch/geant4/license . These *
// * include a list of copyright holders.
// *
```

The IDE window also shows a list of build messages, including the compilation of G4SurfaceProperty.c (c++), linking of the CXX shared library libG4materials.so, and the final success message.

```
Messages Find in Files Konsole Valgrind Breakpoints CTags Problems
*** Success ***
```

18: Build Features: Targets

- Nice feature of CMake:
 - Each library is a target in the build tool – just build the ones you want.

A terminal window with a yellow title bar containing the menu items 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The terminal has a black background with white text. It shows the command '[phsdbc@stelvio]\$ make help' and its output, which lists various valid targets for the Makefile. The list includes 'all', 'clean', 'depend', 'edit_cache', 'install', 'install/local', 'install/strip', 'list_install_components', 'package', 'package_source', 'rebuild_cache', 'G4global', 'G4intercoms', and 'G4materials'. The prompt '[phsdbc@stelvio]\$' is shown at the bottom with a cursor.

```
File Edit View Terminal Tabs Help
[phsdbc@stelvio]$ make help
The following are some of the valid targets for this Makefile:
... all (the default if no target is provided)
... clean
... depend
... edit_cache
... install
... install/local
... install/strip
... list_install_components
... package
... package_source
... rebuild_cache
... G4global
... G4intercoms
... G4materials
[phsdbc@stelvio]$
```

19: Build Features: Dependencies

- Libraries can, and should, depend on others.
 - Even at granular library level, all dependencies are handled for you.
- However, at the granular library level, the dependency tree is horrific
 - The functionality is there to handle this, but questions on scalability.
- ***We can also build in proper handling of dependencies on external libraries***

20: Impact to Developers and Users

- Work cycle very similar to Metaconfig/Make, so familiar
 - Perform Configuration, run build tool
- What developers gain:
 - Clean, user friendly configuration.
 - Better work environment: only one source tree, many different build configurations.
 - More choice of build tool: Make, KDevelop3, Eclipse, Visual Studio.
 - Much cleaner dependency handling, almost automatically.
- What users gain:
 - Clean, intuitive interface.
 - In many cases, sensible choice of defaults will give two-click install.

21: Developer Downsides

- ***Geant4 Prototype enforces separate build and source trees:***
 - You will have to get used to this.
- ***No environment variables!***
 - Build and source tree method is cleaner though!
- ***Each granular module MUST provide a sources.cmake file***
 - Lists all sources, headers and dependencies of the module.
 - More work for you, though you SHOULD understand how your code integrates into the toolkit.
 - You do gain – very robust inter-module dependencies.

22: User Issues

- A new system for users to learn, and we have to provide
- Tool for “easy build” applications
 - CMake can help: it will generate Geant4Config.cmake and UseGeant4.cmake files.
 - These enable us to provide a very simple CMake template for users.
 - Very much like current makefile system, and more robust and cross-platform.
- Tools for advanced users
 - If they want to build with CMake, Geant4Config.cmake and UseGeant4.cmake.
 - **Work in Progress:** geant4-config shell script to query install.

23: Disadvantages of CMake

- ***CMake “targets” are groups of sources that end up in, e.g. a library.***
 - Each source is compiled once for every target it appears in.
 - Means that a global/granular + shared/static build results in ***four compilations***.
 - An annoyance more than anything (and not CMake specific).
- ***Global/Granular library build is a slightly awkward***
 - Each granular module defines a “sources.cmake” file, combining these gives the global module source list.
 - Also tricky in Automake and Scons.
- ***Granular library dependencies are spagettified***
 - Strictly a Geant4 design issue – CMake ***will*** handle interlibrary dependencies, but I am not yet sure of the scalability given the complexity.
- It should be noted that many of these issues are encountered in other build tools!

24: Binary Packaging

- CMake is part of a suite of tools, which can be integrated:
 - CPack for source/binary packaging
 - CTest for unit testing.
- CPack can create:
 - UNIX: rpm, deb
 - OSX: DragNDrop, Bundles, PackageMaker
 - Windows: NSIS, Cygwin source/binary
- So far, prototype only creates source packages.
 - “geant4.9.2.1.tar.bz2” etc.
- *Work in progress to integrate others – potential for trivial binary packaging!*

25: Coding Issues

- There are some coding issues which we might want to address longer term to give a cleaner build system and library structure.
- Main case here is `#ifdef` statements passed through compiler definitions.
 - Many “personal” symbols dotted through the code – consistent naming helps!
 - Can we just use `GEANT4_ALLOC_EXPORT/IMPORT` for all symbol visibility control?
- Many libraries are variant depending on configuration choices.
 - e.g. `libG4persistency` MAY contain `G4GDMLParser`
 - Currently only flagged by environment variables (BAD!!!).
 - Current idea is to use `Geant4Config.cmake` and `geant4-config` scripts to flag these.

26: Workplan

- If the collaboration decides to adopt CMake, what is the plan for integrating it?
- This is a **personal** evaluation based on the time I have available for Geant4 work.
 - 3-5 months to implement all Geant4 build features with CMake
 - Including full cross-platform testing.
 - Including creation of all user tools.
 - Probably a pre-alpha to alpha release at this point.
 - 3 months for testing/refinement/comments.
 - At that point a beta release?
 - 3 months for bug fixes, addition of CPack packaging.
- *I think that by the next collaboration meeting we'll have a working system*
 - *But I emphasize that manpower is limited, and this is a BIG job.*

27: Summary

- Geant4 Metaconfig/Make buildsystem has to be replaced in near term
 - Metaconfig has increasing issues, and is OBSOLETE.
- Evaluation of modern build tools has identified CMake as the current best option
 - Scons evaluation will continue at low level.
- Prototype Geant4 CMake build available
 - <http://www2.warwick.ac.uk/fac/sci/physics/staff/research/bmorgan/geant4/geant4-9.2.1.tar.bz2>
- Brings many nice features for developers/users – interface, dependencies, packaging
- *Discussion needed in Collaboration on adoption and workplan.*