CMake X Microsoft

Introduction to CMake and the Third-party C++ Dependency Manager

Jeff Ho

CENTRILLION
TECHNOLOGIES

Introduction to Modern CMake

Introduction

- CMake is code.

- CMake is a meta build system (build system generator).
  - Configuration Stage
    - Parse the CMakeLists.txt
    - Create a CMakeCache.txt file populated with cache variables.
  - Generation Stage
    - Generates project files or build system files (e.g. Makefile) according to the specified building system (e.g. Make, Xcode, MSBuild, Visual Studio, etc).
  - Building Stage
    - Targets are compiled (e.g. executables, libraries, etc.).
    - Actions associated with each targets are executed.

- Written in C++, cross-platform.

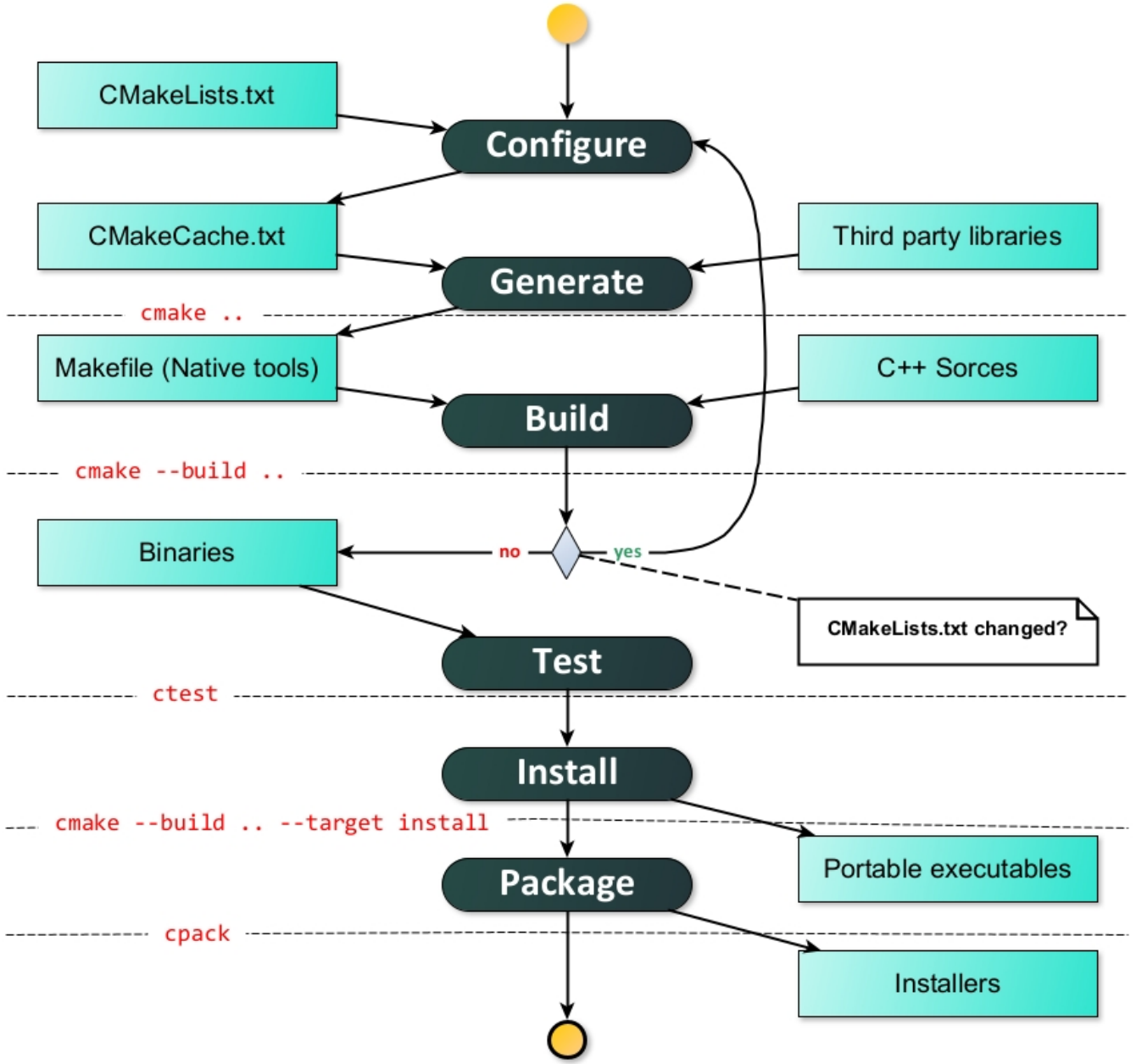
- CMake GUI & CMake command line.

- Used by many projects.

Organization and workflow

- Entry point: The top-level CMakeLists.txt
- Out-of-source build – the source directory is different from the binary directory.
  - Source directory: Directory where the project's CMakeLists.txt resides.
  - Binary directory: Directory where the project files are generated (& CMakeCache.txt resides).
- CMake commands
  - Generate project files or build system files (configuration stage & generation stage are combined in the command-line mode.) [Generator Support]
    cmake ..
  - Build (Compile) the project (building stage).
    cmake --build .. (or make)
  - Run unit-tests
    ctest .. (or make test)
  - Build project and install package
    cmake --build .. --target install (or make install)
  - Packaging project
    cpack ..
  - Command line options
    - -DCMAKE_BUILD_TYPE:STRING=Release or -DCMAKE_BUILD_TYPE="Release"

Organization and workflow



CMake Language

- Data type
  - String
  - List (of strings)
- Control Structures (if, foreach, while)
- Functions
  - Built-in (implemented inside CMake)
  - User-defined
- Macros
- Modules

From Configuration to Building Stage

Build the executables.

Building Procedure Management

- Flow control in CMake

```
set(var1 OFF)
set(var2 ON)

if(var1)
    message(STATUS "var1 is evaluated to false.")
elseif(var2)
    message(STATUS "var2 is evaluated to true.")
else()
    message(STATUS "The program will not enter this area.")
endif()
```

```
-- var2 is evaluated to true.
```

Building Procedure Management

- `add_subdirectory(subdir_name)`
  → Add a subdirectory to the build. Subdirectories must contain a CMakeLists.txt too.

- `include(cmake_filename.cmake or module_name)`
  → Load module or *.cmake scripts files to the build. Those scripts files may describe some CMake configurations.

Targets as Objects

- Constructor:
  - `add_executable()`
  - `add_library()`
    → Library (static, shared, header-only, imported), compiler configurations.
  - `add_test()`
  - `add_custom_target()`
- Member variables:
  - Target properties
- Member functions:
  - `target_include_directories()`
  - `target_link_libraries()`
  - `set_target_properties()`
  - `target_compile_definitions()`
    → Preprocessor definitions for compiling a target's sources.
  - `target_compile_features()` See CMAKE_CXX_KNOWN_FEATURES
    → List of features to be supported by the compiler when building the target.
  - `target_compile_options()`
    → List of options to be passed to the compiler when building the target.

Build Specification and Usage Requirements

```
add_executable(example main.cpp)

target_link_libraries(example
    PUBLIC
        spdlog::spdlog
    PRIVATE
        maths_library)
```

- PUBLIC, PEIVATE, and INTERFACE
  - PRIVATE populates the non-INTERFACE_XXXX property only.
  - INTERFACE populates the INTERFACE_XXXX property only.
  - PUBLIC populates both.
- Building and consuming the target
  - Non-INTERFACE_XXXX properties define the build specification of a target.
    – XXXX property is used only when building the target and won't affect its users in any way.
  - INTERFACE_XXXX properties define the usage requirements of a target (for downstream applications).
    – XXXX property is used when building users of the target.
- maths_library: LINK_LIBRARIES.
- spdlog::spdlog: LINK_LIBRARIES and INTERFACE_LINK_LIBRARIES.

Build Specification and Usage Requirements

Header-only libraries

```
add_library(example INTERFACE)

target_link_libraries(example INTERFACE spdlog::spdlog)
```

- INTERFACE libraries (header-only libraries) have no build specification.
- They only have usage requirements (nothing to build).

Using External Libraries

- There are mainly two ways to use external libraries:
  - For using the pre-built binaries → find_package()
  - For building from the source code → FetchContent module

Using External Libraries

```
find_package(GTest 1.10 CONFIG REQUIRED)

add_executable(example example.cpp)
target_link_libraries(
    example
    INTERFACE
        GTest::gtest
        GTest::gtest_main
)
```

- Find the specified package installed on the local system and loads exported targets.
- REQUIRED is added to specify some package is necessary for the configure stage of the project.
- Two distinct ways for searching the libraries:
  - Module mode (default)
    - search Find<PackageName>.cmake
  - Config mode
    - search <lowercasePackageName>-config.cmake or <PackageName>Config.cmake
    - Also search for <lowercasePackageName>-config-version.cmake or <PackageName>ConfigVersion.cmake if version details were specified.
- (Optional) CMAKE_PREFIX_PATH may be set to find the prebuilt binaries.

Using External Libraries

```
include(FetchContent)
FetchContent_Declare(
    googletest
    GIT_REPOSITORY https://github.com/google/googletest.git
    GIT_TAG release-1.10.0
)
FetchContent_MakeAvailable(googletest)
# FetchContent_GetProperties(googletest)
# if (NOT googletest_POPULATED)
#     FetchContent_Populate(googletest)
# endif ()
# add_subdirectory(${googletest_SOURCE_DIR} ${googletest_BINARY_DIR})
add_executable(example example.cpp)
target_link_libraries(
    example
    INTERFACE
        gtest
        gtest_main
)
```

- FetchContent will build the source as a subproject.

Variables

```
set(var_name "world")
set(list_var "I" "am a" "list")
set(Hello    Hello)
message(STATUS "${Hello} ${var_name} ${list_var}")
message(STATUS ${Hello} ${var_name} ${list_var})
```

```
-- Hello world I;am a;list
-- HelloworldIam alist
```

- Variables are set with the set( ) command.
- Expand with ${ }.
- Lists are separated by some specific delimiter (e.g. ; (semicolon))
- Quotes are used in setting a variable when the value has a whitespace or semicolons.
- Unset variable expands to empty string.
- CMake built-in variables (e.g. CMAKE_XXXX, PROJECT_XXXX, <PROJECT-NAME>_XXXX, etc.)
- More info: Here.

Variables

```
set(var1 OFF)
set(var2 "var1")

if(${var2})
    message(STATUS "dollar var2")
endif()
if("${var2}")
    message(STATUS "quoted dollar var2")
endif()
if(var2)
    message(STATUS "clean var2")
endif()
if("var2")
    message(STATUS "quote var2")
endif()
```

```
-- clean var2
```

- ${var2} will be evaluated to its value var1 before it is passed into if condition.
- `if(${var2})` → `if(var1)` → if(<variable>) case.
- For if(<string>), a quoted string always evaluates to false unless its value is one of the true constants. (1, ON, YES, TRUE, Y, or a non-zero number)

Generator Expression

```
set(run_time_exe_dir $<IF:$<PLATFORM_ID:Darwin>,@loader_path,$ORIGIN>)
```

- Generator expressions use the $<> syntax.
- Evaluated during the build system file generation (generation stage).
- Some commands are not supported in the specific platform.
- Generator expressions are often used to specify which mode we're used (building the library or using it from an installed target).

Function

```
function(setVar varName value)
    set(${varName} ${value})
    set(${varName} ${${varName}} PARENT_SCOPE)
    message(STATUS "setVar: ${varName} = ${${varName}}")
endfunction()

setVar(FooVar "Assign this value to FooVar")
if(DEFINED FooVar)
    message(STATUS "FooVar = ${FooVar}")
else()
    message(STATUS "FooVar is undefined")
endif()
```

```
-- setVar: FooVar = Assign this value to FooVar
-- FooVar = Assign this value to FooVar
```

- Like in C/C++, functions introduce a new scope.
- Variables are scoped to the function, unless set with the PARENT_SCOPE argument.
- When a new command replaces an existing command, the old one can be accessed with a _ prefix.

Function

```
function(AddExe targetName dependency)
    add_executable(${targetName} ${ARGN})
    target_link_libraries(${targetName} PRIVATE ${dependency})
endfunction()

AddExe(ExeFoo Foo foo.cpp)
```

- Available Variables:
  - ARGC: The total count of arguments passed to the function. (3)
  - ARGV: A list of all arguments passed to the function (including the required ones). (ExeFoo;Foo;foo.cpp)
  - ARGN: A list of non-required arguments passed to the function (foo.cpp)
  - ARGVx → ARGV0, ARGV1, ARGV2, …

Macro

```
macro(setFoo value)
    set(Foo ${value})
    message(STATUS "setFoo: ${Foo}")
endmacro()

setFoo("Assign this value to Foo")
if(DEFINED Foo)
    message(STATUS "Foo: ${Foo}")
else()
    message(STATUS "Foo is undefined")
endif()
```

```
-- setFoo: Assign this value to Foo
-- Foo: Assign this value to Foo
```

- Like in C/C++, macros → string substitutions
- Variables defined in macro's body will pollute the calling scope.
- Like functions, when a new command replaces an existing command, the old one can be accessed with a _ prefix.

Testing Stage

Make sure all functions can be executed as expected.

Google Test

- Add an executable target and register the target to Ctest through `gtest_discover_tests()` command.

```
include(CTest)
add_executable(testexe basic_math.cpp)
target_link_libraries(
    testexe
    PRIVATE
        dependencies
        gtest
        gtest_main
)

include(GoogleTest)
gtest_discover_tests(testexe)
```

- Call the test in the building directory after building the testing target.

Catch2

- Add an executable target and register the target to Ctest through `catch_discover_tests()` command.

```
include(CTest)
add_executable(testexe basic_math.cpp)
target_link_libraries(
    testexe
    PRIVATE
        dependencies
        Catch2WithMain
)

include(Catch)
catch_discover_tests(testexe)
```

- Call the test in the building directory after building the testing target.

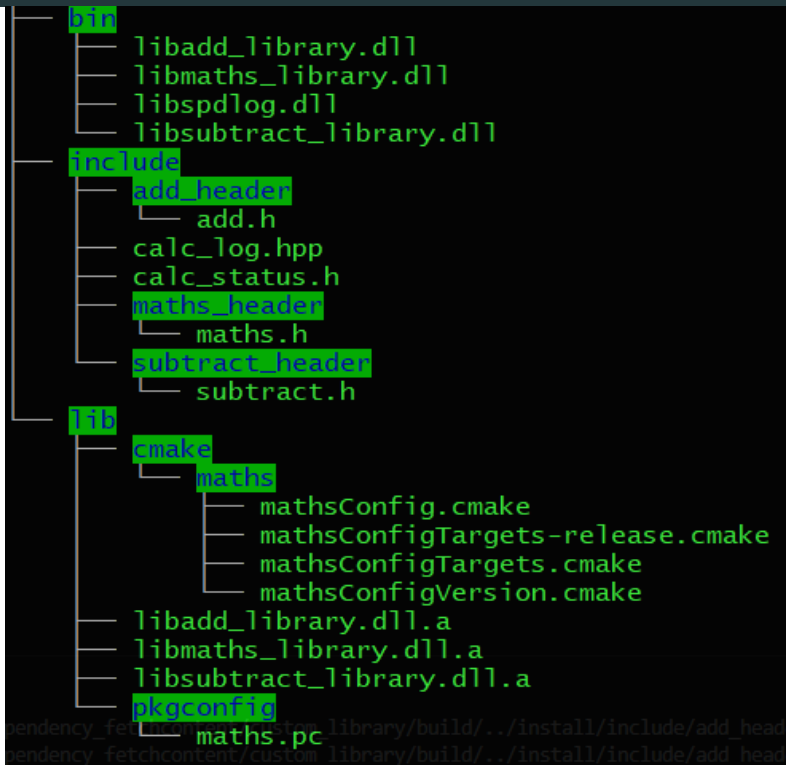
Installation Stage

Copy or generate files and configurations used for the downstream applications.

Export Our Library

- Static or dynamic library.
  - `add_library(static_library_name STATIC sources1.cpp)`
    `add_library(shared_library_name SHARED sources2.cpp)`
  - CMake built-in option
    - `BUILD_SHARED_LIBS=OFF/ON`

- Needed information:
  - Correct building and installing information.
  - Header files that will be used in the downstreams.
  - Easy integration for the use of the library.
    - <project_name>Config.cmake
      → For the use of the find_package() (package configuration, for easy integration).
    - <project_name>Targets.cmake
      → For the downstreams to import all targets listed in the install command.
    - <project_name>ConfigVersion.cmake
      → For the determination of the compatibility with the requested version.
    - <project_name>.pc (optional, for some specific scenarios)
      → For providing the necessary details for compiling and linking a program to a library.



Export Our Library

- Correct building and installing information.
  - Provide the flexibility to install into different platform layouts.

```
include(GNUInstallDirs)
```

  - Use generator expressions to differentiate which mode we're used (building the library or using it from an installed target).

```
target_include_directories(
    library_name
    PUBLIC
        $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
        $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>
)
```

  - Project installation destination – set the `CMAKE_INSTALL_PREFIX` variable.

- Header files that will be used in the downstreams.

```
install(
    FILES ${CMAKE_CURRENT_SOURCE_DIR}/header_name.hpp
    DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}
)
```

Export Our Library

- Easy integration for the use of the library
  - Specify the target, the export target name and the destinations that tell CMake where to install the targets (the following variables are provided by `GNUInstallDirs` module).

```
install(
    TARGETS module_or_library
    EXPORT library_projectTargets
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

  - Install the export targets to export all targets in "`library_projectTargets`" (contains usage requirements, e.g. INTERFACE_XXXXX) to a file.

```
install(
    EXPORT library_projectTargets
    NAMESPACE library_project::
    FILE library_projectTargets.cmake
    DESTINATION ${export_dest_dir}
)
```

Export Our Library

- Easy integration for the use of the library
  - Create a package version file to determine the compatibility with the requested version (e.g. semver concept).

```
include(CMakePackageConfigHelpers)
write_basic_package_version_file(
    library_projectConfigVersion.cmake
    VERSION ${PROJECT_VERSION}
    COMPATIBILITY SameMajorVersion
)
```

  - Create the package configuration template file (*Config.cmake.in) and specified all required dependencies of this library in that file.

```
@PACKAGE_INIT@

include(CMakeFindDependencyMacro)
find_dependency(dependencies REQUIRED)
include(${CMAKE_CURRENT_LIST_DIR}/library_projectTargets.cmake)

set_and_check(library_project_INCLUDE_DIR "@PACKAGE_CMAKE_INSTALL_INCLUDEDIR@")
check_required_components(library_project)
```

Export Our Library

- Easy integration for the use of the library
  - Create the package configuration file through the template file and the `configure_package_config_file()` command (ensuring the resulting package is relocatable).

```
include(CMakePackageConfigHelpers)
configure_package_config_file(
    library_projectConfig.cmake.in
    library_projectConfig.cmake
    PATH_VARS CMAKE_INSTALL_INCLUDEDIR
    INSTALL_DESTINATION ${export_dest_dir}
)
```

  - Create the pkg-config file through the template file (*.pc.in) and the `configure_file()` command

```
configure_file("${PROJECT_NAME}.pc.in"
               "${CMAKE_BINARY_DIR}/${PROJECT_NAME}.pc" @ONLY)
```

Export Our Library

- Easy integration for the use of the library
  - The template file (*.pc.in) for creating the pkg-config file (*.pc).

```
prefix=@CMAKE_INSTALL_PREFIX@
exec_prefix=${prefix}
includedir=${prefix}/include
libdir=${exec_prefix}/@CMAKE_INSTALL_LIBDIR@

Name: lib@PROJECT_NAME@
Description: A CMake library template.
URL: http://gitlab.centrilliontech.com.tw:10088/centrillion/@PROJECT_NAME@
Version: @PROJECT_VERSION@
CFlags: -I${includedir} @PKG_CONFIG_DEFINES@
Libs: -L${libdir} -l@PROJECT_NAME@
Requires: @PKG_CONFIG_REQUIRES@
```

Export the Application Executables

- Static dependencies.
  - The executables of applications should be executed without any further dependencies' installation.
- Dynamic dependencies.
  - Application executables find dependencies through a built-in manner (depending on the OS).
  - Use the `fixup_bundle()` command to analyze the dependencies binaries or directly copy all binaries of the dependencies into specified directory (depending on the OS).
    - WINDOWS: Next to the application executables.
    - UNIX: lib folder in the portable package.
  - Set the RPATH of the application executables (UNIX only)
    - Run time executables directory symbol
      - MacOS: @loader_path
      - UNIX: $ORIGIN

Export the Application Executables

- Dependencies searching order.
  - UNIX
    1. The executable's rpath.
    2. The LD_LIBRARY_PATH environment variable.
    3. The executable's runpath.
    4. The /etc/ld.so.conf file.
    5. Default system libraries (/lib and /usr/lib)
  - Windows
    1. The executable's directory.
    2. The system directory (default: C:/Windows/System32)
    3. The 16-bit system directory.
    4. The Windows directory (default: C:/Windows).
    5. The current working directory.
    6. The directories that are added and listed in the PATH environment variable.

Troubleshooting Information

- Logging: message(STATUS “var=${var}”)
- Check CMakeCache.txt file
- Check the output generated files
- Add if() to judge specified conditions
- Check the compile_commands.json file after adding -DCMAKE_EXPORT_COMPILE_COMMANDS=TRUE option
- Add -DCMAKE_VERBOSE_MAKEFILE=TRUE option when configuring the project.

That’s it!

Remember these and you know the guts of Modern CMake.

Special Commands – Custom Targets

```
add_executable(exmaple_exe example.cpp)
add_custom_command(
    TARGET exmaple_exe
    POST_BUILD
    COMMAND
        ${CMAKE_COMMAND} -E touch ${CMAKE_CURRENT_BINARY_DIR}/generated_file1.txt
    COMMENT
        "Create the generated_file1.txt after all other rules within the target have
been executed (i.e. after building the example_exe target)."
)
```

- Doing things (through COMMAND) that is not related to just compile or link binary.
  → For more actions: Command-Line Tool

- ${CMAKE_COMMAND} represents the path to the CMake executable being used right now.

- add_custom_command does not create a new target.
  → Targets should be explicitly specified to make it visible (e.g. Use existing targets or create a new one.)

Special Commands – Custom Targets

```
add_custom_target(my_custom_target
    DEPENDS
        "${CMAKE_CURRENT_BINARY_DIR}/generated_file1.txt"
)
add_custom_command(
    OUTPUT
        "${CMAKE_CURRENT_BINARY_DIR}/generated_file1.txt"
    COMMAND
        ${CMAKE_COMMAND} -E touch ${CMAKE_CURRENT_BINARY_DIR}/generated_file1.txt
    COMMENT
        "Create the file each time the DEPENDS is modified or the first time
my_custom_target is built."
    DEPENDS
        ${CMAKE_CURRENT_SOURCE_DIR}/source.cpp
)
```

Special Commands – Custom Targets

- Custom target are a kind of target but doesn't produce an exe or lib only.
  → Still have other properties, including having or being dependencies.

- Build the custom target
  - cmake --build . --target my_custom_target
  - Add ALL argument to the `add_custom_target` and build with default command.

- Without specifying the `add_custom_target()`, the `add_custom_command()` will not be executed since there was not a defined target.

- The dependency between `add_custom_target` and `add_custom_command`
  → The `DEPENDS` argument for `add_custom_target` and the `OUTPUT` argument for `add_custom_command`.

More …

- Find module sample – CMake docs
  - find_package() – Find out all what is necessary to use some lib (find and load all settings).
    - find_program() – Find an executable file.
    - find_library() – Find the binaries of a library, shared or static.
    - find_file() – Find a file from the given full path.
    - find_path() – Find a directory that containing some specifically named file.
- Creating Packages – CPack
- Cross Compilation – toolchain.cmake

Third-party Dependency Manager for C++

Introduction

- Current Existing Package Managements
  - Git Submodule
  - **Fetchcontent**
  - Hunter
  - **Conan**
  - **vcpkg (Microsoft)**

- Prerequisites for an ideal package manger
  - Cross-platform support.
  - Automatically install dependencies specified by the project and the upstream dependencies.
  - Manage the dependency versioning problems.
  - Easy integration with our build system (including and linking libraries).
  - Steady development and maintenance in the long term.
  - Healthy community of developers & users (for solving problems).
  - Internal library maintenance, distribution and version management.
  - (For library providers) Follow the exporting rule of CMake as much as possible.
  - Good Compatibility among different libraries.
  - Supports a wide range of packages.

How to get started

- Cmake-tutorial project: http://gitlab.centrilliontech.com.tw:10088/centrillion/cmake-tutorial
  - For Fetchcontent (build the library and application)
    - 09a folder in the cmake_tutorial.

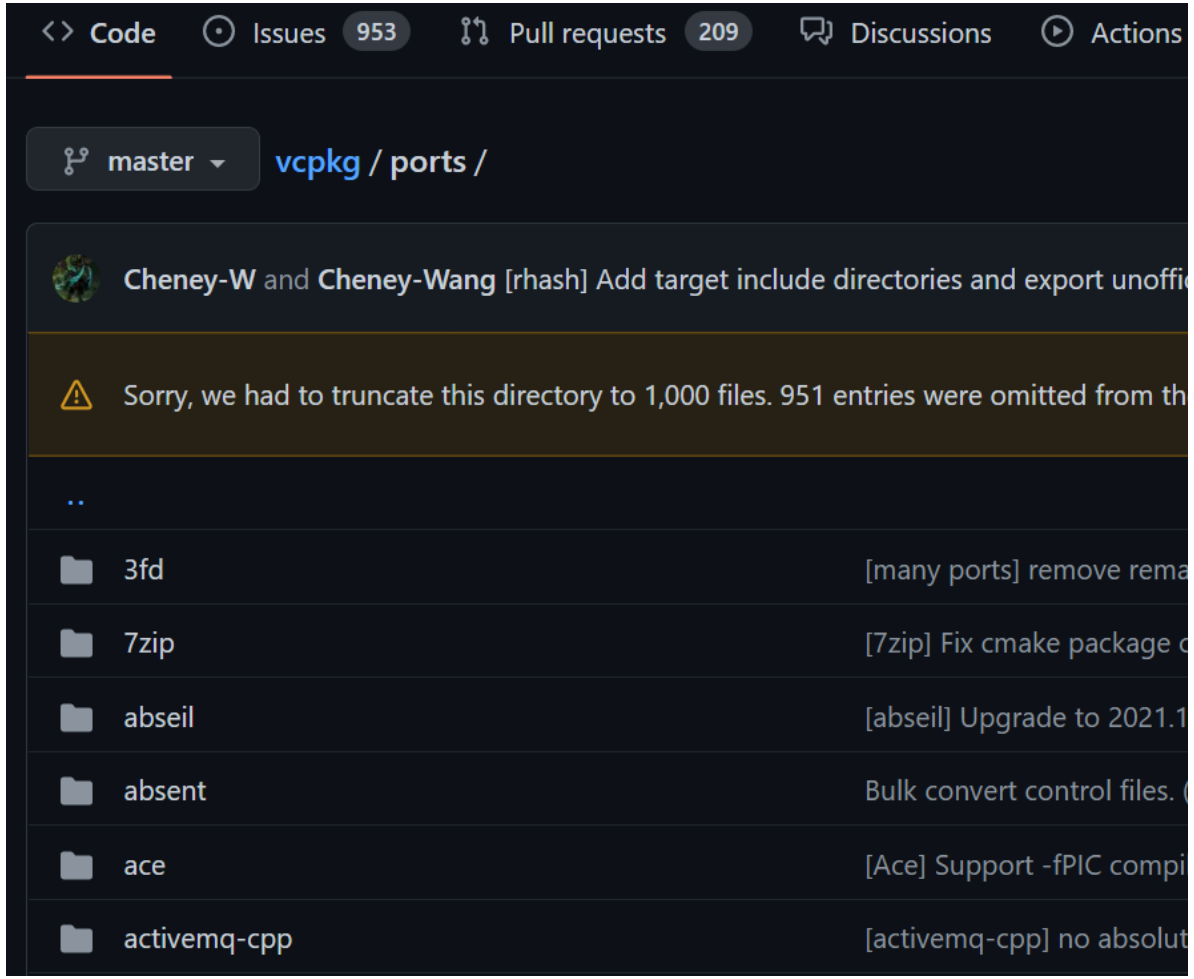
  - For Microsoft – vcpkg (build the library and application)
    - 08a folder (application, basic use for the manifest mode)
    - 09b folder (library)
    - 10 folder (library, patching mechanism for local fix)

  - For Conan (build the application case only)
    - 08b folder (application, basic use)

- For the library using the Conan framework, all the CMakeLists.txt files in the library should be substituted with the conanfile.txt written in Python and the CMake library provided by Conan.

Officially Supported Library List

- Port is a recipe for building a library.
- Triplet describes the build configuration (target architecture, OS, etc.).



Total Ports Available for Tested Triplets (2022)

triplet	ports available
x86-windows	1,731
x64-windows	1,776
x64-windows-static	1,667
x64-windows-static-md	1,691
x64-uwp	880
arm64-windows	1,315
arm-uwp	826
x64-osx	1,641
x64-linux	1,713

Officially Supported triplets

Manifest file: vcpkg.json

- Goal: Allows developers to specify libraries, library metadata, library versions, and more.

  - Use the imported library in CMakeLists.txt.

```
find_package(fmt CONFIG REQUIRED)

add_executable(ManifestDemo main.cpp)

target_link_libraries(ManifestDemo
    PRIVATE
        fmt::fmt
)
```

vcpkg.json

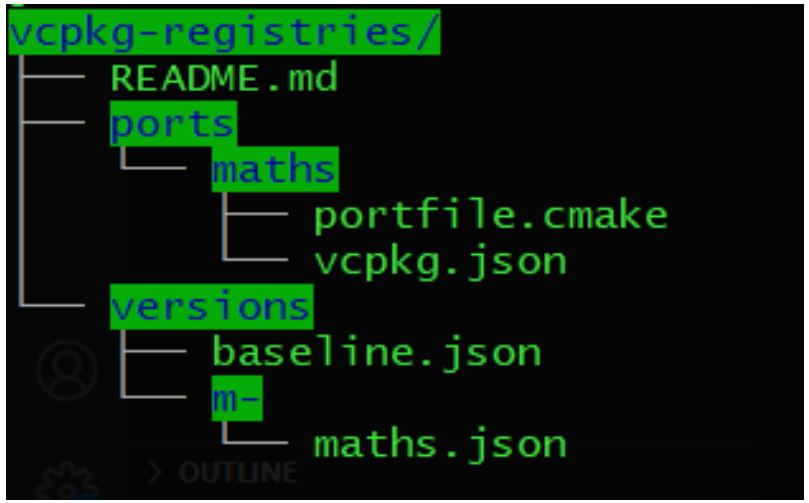
08a_basic_development_and_depoly_vcpkg > test_project > vcpkg.json > ...

You, 4 weeks ago | 1 author (You)

```
{       You, 4 weeks ago • Initial commit …
  "name": "demotest",
  "version": "0.0.1",
  "dependencies":[
    {
      "name": "fmt",
      "version>=": "7.1.3#1"
    },
    {
      "name": "gtest",
      "version>=": "1.10.0"
    }
  ],
  "builtin-baseline": "3426db05b996481ca31e95fff3734cf23e0f51bc"
}
```

Manage Custom Libraries: Registries

- portfile.cmake – A file that describes how to download, compile, and install the library.
- vcpkg.json – A file that describes a project's dependencies and their version info.
- <port>.json – A file that lists all the versions available for a package and contain a Git tree-ish object that vcpkg can check out to obtain that version's portfiles (in the ports folder).
- baseline.json – This file contains a version declaration (a minimum version constraint) for each library in the vcpkg library repository (vcpkg-registries).
  - For any given revision of the registry (vcpkg-registries), the versions declared in the baseline file must match the current versions of the ports in the registry at that revision.
- vcpkg-configuration.json – A file that describes the download source of the vcpkg package repository (vcpkg-registries).
  - This file is needed when an unofficial dependency is used (to tell vcpkg where it can find and build this unofficial library).



Manage Custom Libraries: Registries

maths.json

09b_deploy_remotely_third_dependency_vcpkg > vcpkg-registries > versions > m- > maths.json > ...

You, 2 weeks ago | 2 authors (You and others)

```
{
  "versions":[
    {
      "version": "0.1.0",
      "port-version": 1,
      "git-tree": "6ffb7897b666d8b045ad5e52ff0c813b29180a42"
    },
    {
      "version": "0.1.0",
      "git-tree": "b61b11dfc5368bbb6545160d88fb422cbfe3108e"
    }
  ]
}       You, last month • Initial commit.
```

<port>.json

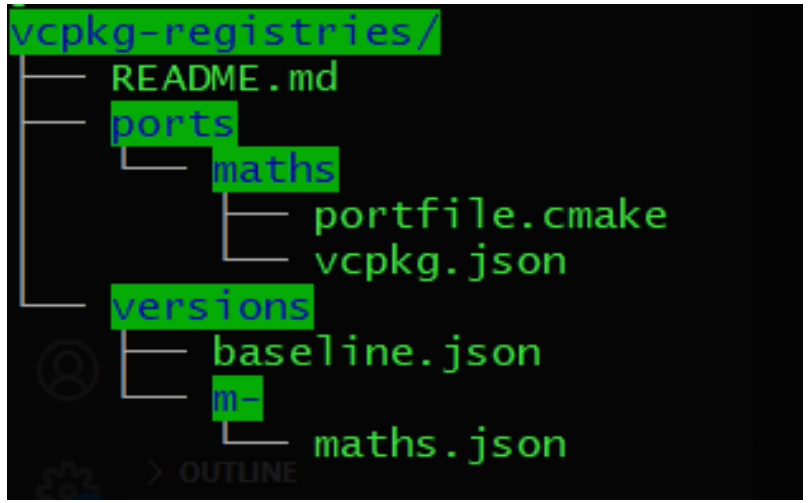
baseline.json

09b_deploy_remotely_third_dependency_vcpkg > vcpkg-registries > versions > baseline.json > ...

You, 2 weeks ago | 1 author (You)

```
{       You, last month • Initial commit. ...
  "default":{
    "maths": {"baseline": "0.1.0", "port-version": 0}
  }
}
```

Manage Custom Libraries: Registries



vcpkg-configuration.json

09b_deploy_remotely_third_dependency_vcpkg > test_project > {..} vcpkg-configuration

You, 2 weeks ago | 1 author (You)

```
{
    "registries": [
        {
            "kind": "git",
            "repository": "http://gitlab.cent
            "baseline": "a89a20b3c5968bccbb3
            "packages": [ "maths" ]
        }
    ]
}
```

You, 4 weeks ago • Initial commit …

vcpkg.json

09b_deploy_remotely_third_dependency_vcpkg > test_project > {..} vcpkg.json > ...

You, 4 weeks ago | 1 author (You)

```
{
  "name": "mathsdemotest",
  "version": "0.0.1",
  "dependencies":[
    {
      "name": "nlohmann-json",
      "version>=": "3.10.5"
    },
    "maths"
  ],
  "builtin-baseline": "97b723c3467f53fc49ea9c8c118658ee526d7817",
  "overrides":[
    {
      "name": "maths",
      "version": "0.1.0"
    },
    {
      "name": "spdlog",
      "version": "1.9.2"
    }
  ]
}
```

You, 4 weeks ago • Initial commit …

Build from Local Sources: Patching Mechanism

- Specify the patch in the portfile.cmake.
- Configure the downstream application with the DVCPKG_OVERLAY_PORTS CMake option.

```
-DVCPKG_OVERLAY_PORTS="../../vcpkg-registries/ports/maths"
```

portfile.cmake

10_locally_fix_remotely_third_dependency_vcpkg > vcpkg-registries > ports > maths > portfile.cmake

You, 2 weeks ago | 2 authors (You and others)

```
vcpkg_from_gitlab(
    GITLAB_URL http://gitlab.centrilliontech.com.tw:10088
    OUT_SOURCE_PATH SOURCE_PATH
    REPO centrillion/moduletemplate
    REF v0.1.0-vcpkg  # Specify the version tag or the commi
    SHA512  26f7a0f4aa897d0e8224c6600e58cf3a1ba9e9b507c999af
    # Compute the SHA512 of the tar.gz file of the REF versi
    HEAD_REF vcpkg    # Always build from the latest commit
    PATCHES
        disable-the-example-executable.patch        You, 2 we
)
```

disable-the-example-executable.patch

10_locally_fix_remotely_third_dependency_vcpkg > vcpkg-registries > ports > maths > d

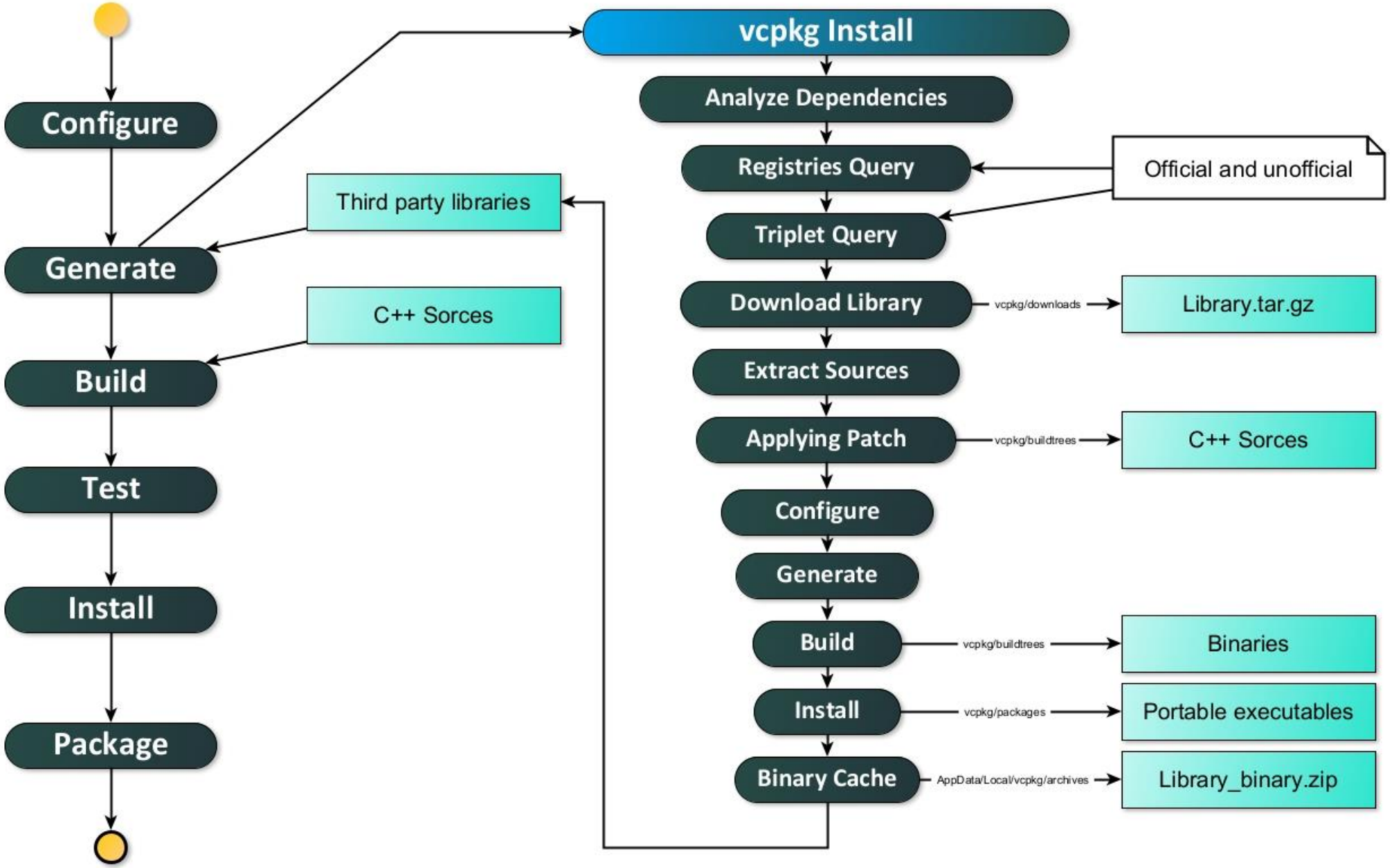
You, 2 weeks ago | 1 author (You)

```
diff --git a/CMakeLists.txt b/CMakeLists.txt
index 3d1f9b4..7e22e00 100644
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -101,7 +101,7 @@ if(PKG_INSTALL)
 endif()

 # The following will not be used for building a library. Just an
-if(BUILD_EXAMPLE)
+if(OFF)
     add_executable(development_and_deploy_main main.cpp)
     target_link_libraries(development_and_deploy_main
         PRIVATE
```

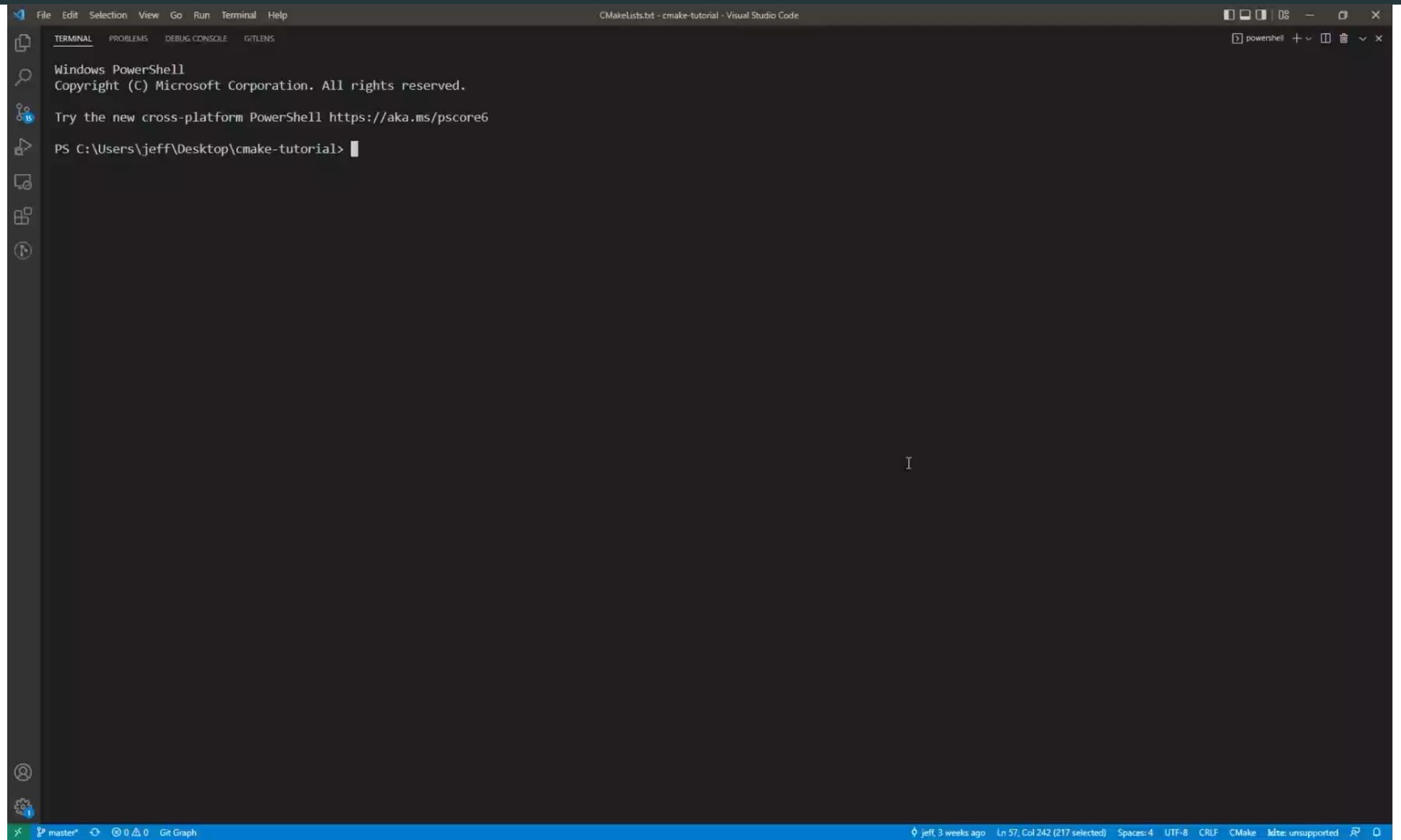
```
vcpkg-registries/
├── README.md
├── ports
│   └── maths
│       ├── disable-the-example-executable.patch
│       ├── portfile.cmake
│       └── vcpkg.json
└── versions
    ├── baseline.json
    └── m-
        └── maths.json
```

Overall Workflow for the vcpkg Dependency Manager



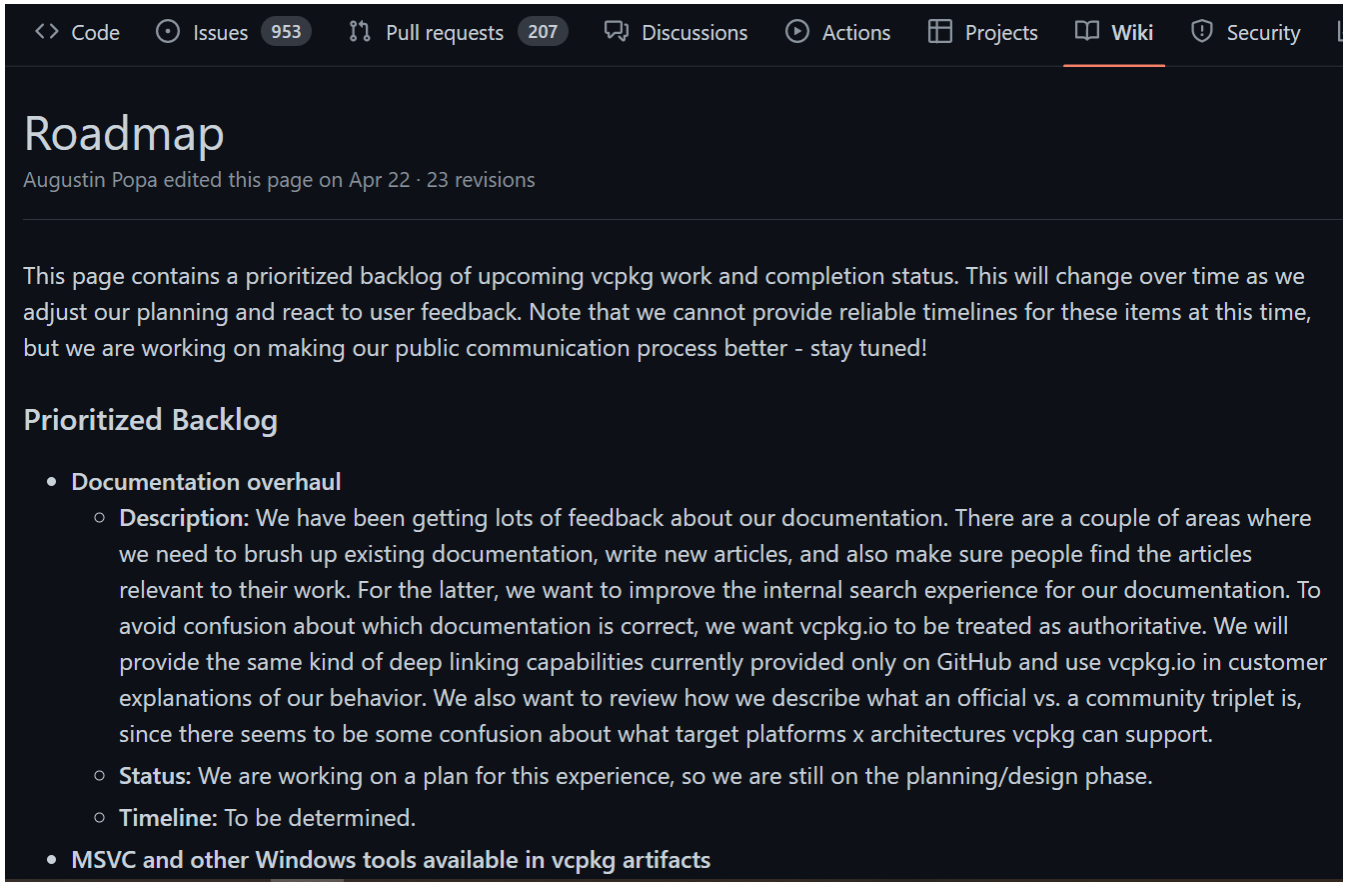
Demo

Manage Custom Libraries



Learn More …

- Binary Caching: vcpkg/binarycaching.md at dev/roschuma/binarycaching-spec · ras0219-msft/vcpkg (github.com)
- vcpkg product roadmap: Roadmap · microsoft/vcpkg Wiki (github.com)



- Get started with vcpkg: vcpkg/README.md at master · microsoft/vcpkg (github.com)

Learn More …

- Module Template: centrillion / ModuleTemplate · GitLab (centrilliontech.com.tw)
- Project Template: centrillion / ProjectTemplate · GitLab (centrilliontech.com.tw)