

CMake Tutorial (P 2) | Understanding The Basics



→ The idea is to follow this tutorial  
and the Cpt instructions!

## Why Choose

<https://earthly.dev/blog/cmake-vs-make-diff>



- file that runs commands
- Something that runs in bash
- What cmake turns into
- Cmake creates the make files
- Runs the commands we tell it too

Basically we compile it and it runs our program... hummm based on the .cpp file we send as source

Entry point documentation <https://cmake.org/cmake/help/latest/>

How to install [Installing CMake in 2 minutes on Windows](#)



Install make gcc and g++ on windows How to install Make/GCC/G++ on Windows 10/11 - Scoop Package

## Install make, gcc, and g++ on Windows [How To Install MinGW](#)

### Manager

<https://gnuwin32.sourceforge.net/>

install gcc



<https://code.vilniusstudio.com/docs/cpp/config-minew/microsoft> - Ok this is the best link

Pacman is a package manager that can be used to install, update, and remove packages, including C/C++ packages:

But still need to install the MAKE one from the previous video

Once the project is building... let's move on

- molecules doesn't accept spaces
  - make files relies on a top level file called CmakeLists.txt
  - With the cmakefile, we can run cmake
    - Need to tell where the source is and build folder is
    - don't use the same folder .ID
    - Create a build directory
      - `cmake -B build`
    - But there is no standard
    - `cmake [options] <path-to-source> -B <path-to-build>`

ps -e |grep java |grep -v jps |awk '{print \$1}' |xargs kill -9  
ps -e |grep java |grep -v jps |awk '{print \$1}' |xargs rm -rf

Build with Convans  
Relies on file for Local/alg

↳ To run Cmake needs to specify the source and output

C Monarchs Ext  
will generate a level  
not found it!  
↳ Ext only the C

This first test was with an empty CmakeList.txt file. It doesn't build anything but generates a build source file with some configurations. That is nice but let's start to write some commands in the CmakeList.txt

```
Build file has been written to: C:\Projects\Concepts\Reviews\Onde\Onde Tutorial\OndeTutorial.sln
PS C:\Projects\Concepts\Reviews\Onde\Onde Tutorial> code . - B & < %USERPROFILE%\Desktop\Onde.html

● The C compiler identification is GNU 13.2.0
  The CXX compiler identification is GNU 13.2.0
  Check for working C compiler: C:/msys64/mingw/bin/gcc.exe - skipped
  Check for working CXX compiler: C:/msys64/mingw/bin/g++_exe - skipped
  Detecting C compiler features
  Detecting CXX compiler features
  -- Detecting C compiler ABI info
  -- Detecting CXX compiler ABI info - done
  Checking for working C compiler: C:/msys64/mingw/bin/gcc.exe - skipped
  Detecting C compiler features
  Detecting CXX compiler features
  Configuring done (4.4s)
  Generating done (0.83s)

Build file has been written to: C:\Projects\Concepts\Reviews\Onde\Onde Tutorial\Onde.sln
PS C:\Projects\Concepts\Reviews\Onde\Onde Tutorial>
```

- ✓ CHARGE TUTORIAL
- ✓ out.bakup
- ✓ CHARGE.tgz
- ✓ 3.0.5
- ✓ CHARGEStretch
- ✓ CHAS.DIF
- ✓ pigments.txt
- ✓ omktochd\_code
- ✓ CHARGEconfig\_inigrid
- ✓ CHARGE\_rectonInformation.omkce
- ✓ Model1.in.mokce
- ✓ Model2.in.mokce
- ✓ progressmarks
- ✓ TargetRectons.txt
- ✓ omkce\_recton.mokce
- ✓ CHARGErecton
- ✓ Makfile
- ✓ CHARGErecton.tgt
- ✓ man.csp

→ First you prepare the `main.c` file!  
there is no build yet but prepares the monopole with all instruction to build anywhere (unless no / library)

→ Here we see how the build process works! Based on the OS  
→ In this case generates our executable

→ Run the executable

## Generate a Build System

One of the main differences between CMake and Make is that **CMake creates output that can be used by build systems like Make**. This means that **CMake acts as a generator for other build systems and it's not responsible for the actual compilation**. In contrast, the **output of Make is a compiled binary that can be executed on the target computer**.

Make → Responsible for compiling the solution to be executed in the operating system  
Cmake → Generates the **cmakefile** that can generate the executable over various builds  
↳ focus on a cross platform app

```
cmake_minimum_required(VERSION 3.16.2)
project(TRIN)
add_executable(trin main.cpp)
install(TARGETS trin DESTINATION "c:/Users/ConceptReview/Cmake/trin/build")
```

↳ focus the install path to use

```
-- Configuring done (0.2s)
-- Generating done (0.2s)
-- Build files have been written to: c:/Users/ConceptReview/Cmake/trin/build
[100%] Built target trin
PS C:\Users\ConceptReview\Cmake\trin> make install
[100%] Built target trin
Install the project...
-- Install configuration: -
-- To install an already built project
PS C:\Users\ConceptReview\Cmake\trin> cd ..
PS C:\Users\ConceptReview\Cmake> cd ..
PS C:\Users\ConceptReview> cd ..\trin> make install
[100%] Built target trin
PS C:\Users\ConceptReview>
```

↳ This can put the executable somewhere else in the project - just change the path!

↳ Configure the correct path

## LIBRARIES

### Static Lib

↳ Define .cpp code first

↳ We need a main because lib don't have an entry point functions

↳ We need a simple example with an overload and 2 static methods

### None Shared

```
cmake_minimum_required(VERSION 3.16.2)
project(addLib)
add_library(meanyMath cp/adder.cpp)
```

↳ the main defined here!  
add a file instead of an executable

```
make
```

↳ this is the lib now and the all of an executable  
↳ lib at the beginning is just a simple

↳ So, it's pretty cool → can be built for Linux or windows with similar commands

↳ Define the CMakeLists + the cpp implementation files

↳ Main command to build it for Linux (less clear)

↳ NoBuild command for windows → Remember to add the compiler to the path

```
cmake_minimum_required(VERSION 3.16.2)
project(addLib)
add_library(meanyMath cp/adder.cpp)
```

↳ meanyMath

↳ windows

↳ D:\Dev\meanyMath

✓ Theodore has the list of things to expect.

```
mje> ls  
CMakeLists.txt adder.cpp adder.h build  
mje> cd build/  
mje> ls  
CMakeCache.txt CMakeFiles Makefile cmake_install.cmake libnearlymath.a  
mje>
```

(Need to include footer!!)

```
M CMakeLists.txt X
M CMakeLists.txt
1  cmake_minimum_required(VERSION 3.11)
2
3  project(msvibd)
4
5  add_executable(msvibd ${CMAKE_SOURCE_DIR}/main.cpp)
6
7  target_link_directories(msvibd PRIVATE ${CMAKE_SOURCE_DIR}/build/linux)
8
9  target_link_libraries(msvibd msvibd)
```

→ CMAB file when referencing  
files from within another project

→ Create the mountable file with  
the logic using the file

Add the actual path under the lib is  
( -a -so +dll) → Verify the OS here

→ Live life like by name  
without any prefix or suffix

→ Remove dependencies Across Projects

↳ Build the static file relations with file outputs, guarantee to some hard folder

Set file  
forget project

*Set the project project to /out file location with public areas*

*Put the destination to the lib & header files to a common directory (in this case the c drive)*

*Need to rebuild everything*

*Still lots of warnings but it's not required for linux*

→ Bulk modules

→ Integration with githubs (libs based on my repos)

command → git submodule add <https://github.com/glfw/glfw.git> external glfw

→ This is a git feature to bring libs into the project

It clones the repos into one of the folders following the command

→ git submodule add

< Repo name> < folder to past

PS C:\Projetos\ConceptsReview\Make\Submodule> cd ..  
PS C:\Projetos\ConceptsReview\Make\Submodule> Cmake -S . -B ./out -G "Unix Makefiles"  
-- Configuring done (0.2s)  
-- Generating done (0.1s)  
-- Build files have been written to: C:/Projetos/ConceptsReview/Make/Submodule/out  
PS C:\Projetos\ConceptsReview\Make\Submodule> cmake --build ./out  
[ 25%] Building CXX object Adder/CMakeFiles/adder.dir/adder.cpp.o  
[ 50%] Built static library libadder.a  
[ 50%] Built target Adder  
[ 75%] Linking CXX executable OLAS.dir/main.cpp.o  
[100%] Linking CXX executable OLAS.exe  
[100%] Built target OLAS  
PS C:\Projetos\ConceptsReview\Make\Submodule> ./OLAS.exe  
Hey Buddy!  
2 + 3 = 5  
PS C:\Projetos\ConceptsReview\Make\Submodule> cd ..  
PS C:\Projetos\ConceptsReview\Make> git submodule add https://github.com/glfw/glfw.git external glfw  
Cloning into 'external glfw'...  
remote: Enumerating objects: 3289, done.  
remote: Total 3289 (delta 0), pack-reused 3288 (from 1)  
Receiving objects: 100% (3280/3280), 16.32 MiB | 646.00 KiB/s, done.  
Resolving deltas: 100% (2301/2301), done.  
warning: LF will be replaced by CRLF in external glfw/.gitignore.  
The file will have its original line endings in your working directory  
PS C:\Projetos\ConceptsReview\Make> [redacted]

→ The up level lib directory is copied to the solution after building. So, it actually builds the lib together with the source code!

→ Can also create bat files to move the process easier and not type the commands all the time



## CMake Tutorial EP 6 | Installing Your Software! (part 1/2 of install)

CMake Tutorial EP 6 | Installing Your Software! (part 1/2 of install)



Way to install the program.... not sure if I need right now but let's keep it

*→ Progress everything up*

CMake Tutorial EP 7 | Installing With CPack! (part 2/2 of install)



find\_library(...)

*→ Different way to import lib*

CMake Tutorial EP 8 | find\_library... (part 1/2 of find lib)



## Running CMake on Windows (like linux)

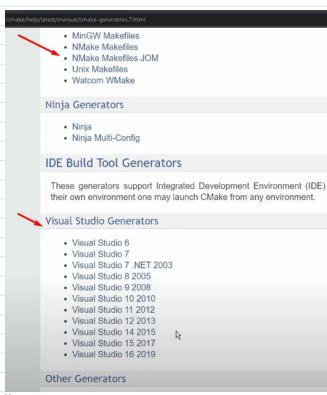
CMake Tutorial EP 10 | Running CMake on Windows (like linux) | Over-explained



- Add things to PATH variables
  - cmake
  - glib
- Compiler options
  - GCC
    - Can install WSL
    - MinGW
    - VS Generator
    - Controlled by -G

Cmake - VSCode MSBuild example

```
PS C:\Proyectos\ConceptsReview\Cmake\test> cmake -G "Visual Studio 16 2019"
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.19045.
-- The C compiler identification is MSVC 19.29.30146.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cxx.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.1s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Proyectos/ConceptsReview/Cmake/test/out/build
PS C:\Proyectos\ConceptsReview\Cmake\test>
```



*→ Can use any of the generators to create file build but it doesn't build it*

*→ Just create file project based on the selected version*

```
PS C:\Proyectos\ConceptsReview\Cmake\test> cd out
PS C:\Proyectos\ConceptsReview\Cmake\test\out> cd build
PS C:\Proyectos\ConceptsReview\Cmake\test\out\build> msbuild.exe ./test.sln
```

```
Time Dec/20/2019 08:00:00.81
PS C:\Proyectos\ConceptsReview\Cmake\test\out\build> cd ..\Debug
PS C:\Proyectos\ConceptsReview\Cmake\test\out\build\Debug> ./test.exe
● Hey dude
● PS C:\Proyectos\ConceptsReview\Cmake\test\out\build\Debug>
```

msbuild.exe ->
 • Required to add in the path folder
 o C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Current\Bin
 • nOw CAN RUN THE msBUILD.EXE .\TEST.SLN COMMAND IN THE OUTPUT FOLDER TO BUILD IT
 • And then execute the command to run the executable file

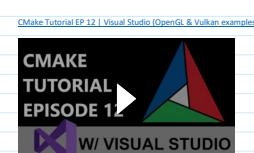
## VSCODE | VCPKG Libraries | Project Setup Basics



## Visual Studio (OpenGL & Vulkan examples)

To run files with *VS*

Create new project



- Type "CMake Project"
  - Install the vcpkg to install libs [https://learn.microsoft.com/en-us/vcpkg/get\\_started/get-started-msbuild?view=msbuild](https://learn.microsoft.com/en-us/vcpkg/get_started/get-started-msbuild?view=msbuild)
  - vcpkg is a free and open-source C/C++ package manager maintained by Microsoft and the C++ community.
- Install the lib with VCPKG
  - VCPKG install vulkan
    - af

```

CMakeLists.txt
1 # CMakeLists.txt : CMake project For DemoBase, include
2 # project specific logic here.
3 #
4 #cmake_minimum_required (VERSION 3.0)
5 #
6 # project ("DemoBase")
7 #
8 #Include sub_projects
9 #add_subdirectory("DemoBase")
10

90 %  No issues found  Ln: 10 Ch: 1 OVR TAE
PowerShell do Desenvolvedor
+ PowerShell do Desenvolvedor -  ⓘ
PS C:\Projetos\ConceptsReview\Cmake\DemoBase> vcpkg install vulkan
Computing installation plan...
The following packages are already installed:
vulkan-windows@0.2.1-17
vulkan-x64-windows is already installed
Total install time: 579 ms
vulkan is compatible with built-in CMake targets:
# FindVulkan.cmake (https://github.com/vcpkg/FindVulkan.cmake)
find_package(Vulkan REQUIRED)
target_link_libraries(${PROJECT_NAME} PRIVATE Vulkan::Vulkan)

PS C:\Projetos\ConceptsReview\Cmake\DemoBase> vcpkg integrate install
Applied user-wide integration for this vcpkg
CMake projects should use: "-DCMAKE_TOOLCHAIN_FILE=C:/Projetos/ConceptsReview/Cmake/lib/externa/vcpkg/scripts/buildsystems/vcpkg.cmake"

All MSBuild C++ projects can now #include any installed libraries. Linking will be handled automatically. Installing new libraries will make them instantly available.
PS C:\Projetos\ConceptsReview\Cmake\DemoBase> ^
PS C:\Projetos\ConceptsReview\Cmake\DemoBase> ^
PS C:\Projetos\ConceptsReview\Cmake\DemoBase>

```

CMake Settings

CMake Settings allows you to configure CMake project generation and build. Use this editor to edit settings defined in the underlying CMakeSettings.json file. Add a new configuration to build and debug on a remote Linux machine or the Windows Subsystem for Linux (WSL). To edit additional settings not shown here, go to CMakeSettings.json

Configurations

Toolset: Debug

Toolchain file: The path to the toolchain file supplied to CMake. Toolchain files specify locations of compilers and toolchain utilities, and other target platform and compiler related information. By default, Visual Studio will try and use the vcpkg toolchain file if this setting is unspecified. [More information](#)

Build root: The directory in which CMake generates project files. Corresponds to CMAKE\_BINARY\_DIR and specifies where the CMake cache will be created. The specified folder will be created if it does not exist. Supported macros are available to use in the file path. [More information](#)

Command arguments: Additional command line options passed to CMake when invoked to generate the cache.

## CTest

