

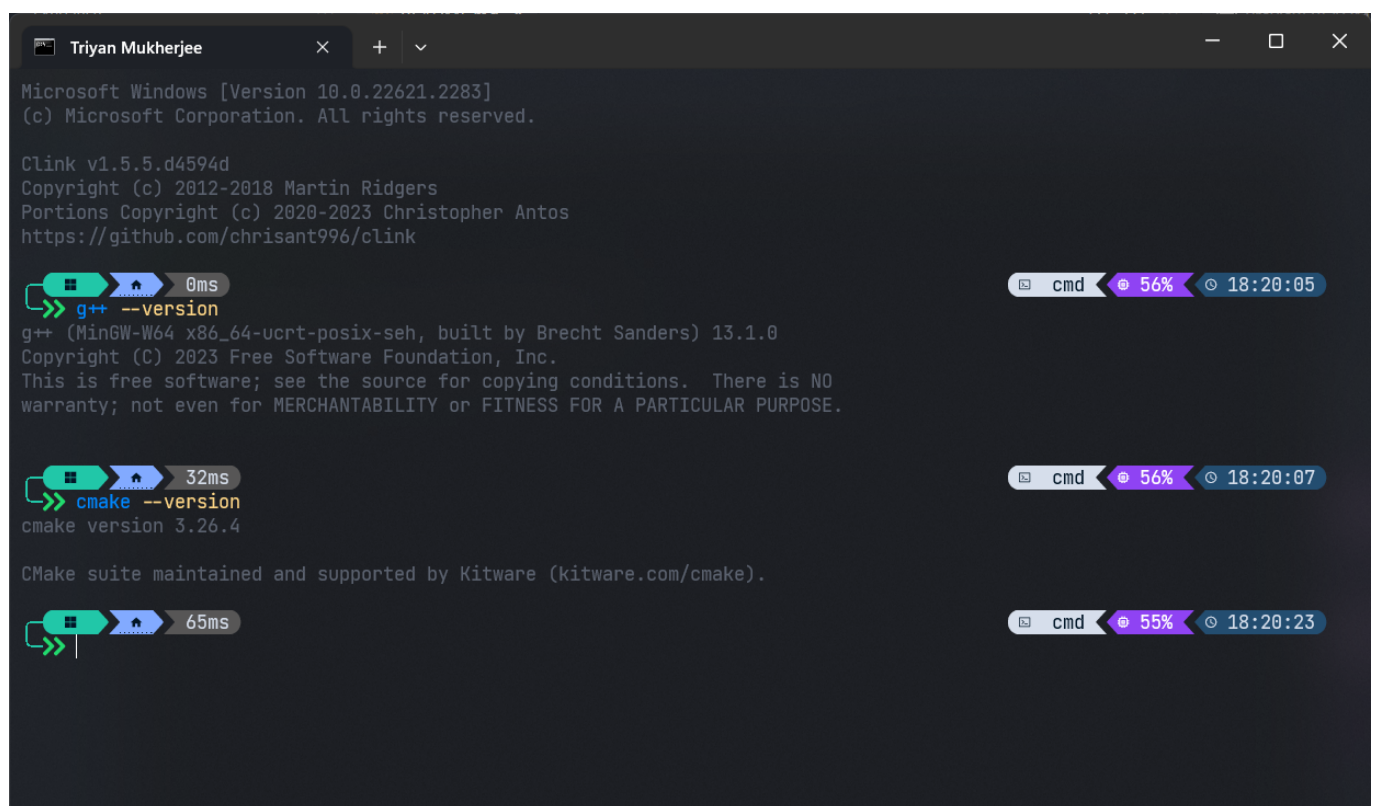
Using CMake with Visual Studio Code

CMake is an open-source build system generator for software projects that allows developers to specify build parameters in a simple, portable, text file format.

CMake makes it easy to build projects with multiple files, multiple build configurations, and multiple platforms. By defining a common build specification, environments can be replicated across development, testing, and production.

Note

Make sure you have the **C/C++** compiler installed from the previous tutorial. To verify run **g++ --version** and **cmake --version**.



The screenshot shows a terminal window titled 'Triyan Mukherjee' with the following content:

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

Clink v1.5.5.d4594d
Copyright (c) 2012-2018 Martin Ridgers
Portions Copyright (c) 2020-2023 Christopher Antos
https://github.com/chrisant996/clink

>> g++ --version
g++ (MinGW-W64 x86_64-ucrt-posix-seh, built by Brecht Sanders) 13.1.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

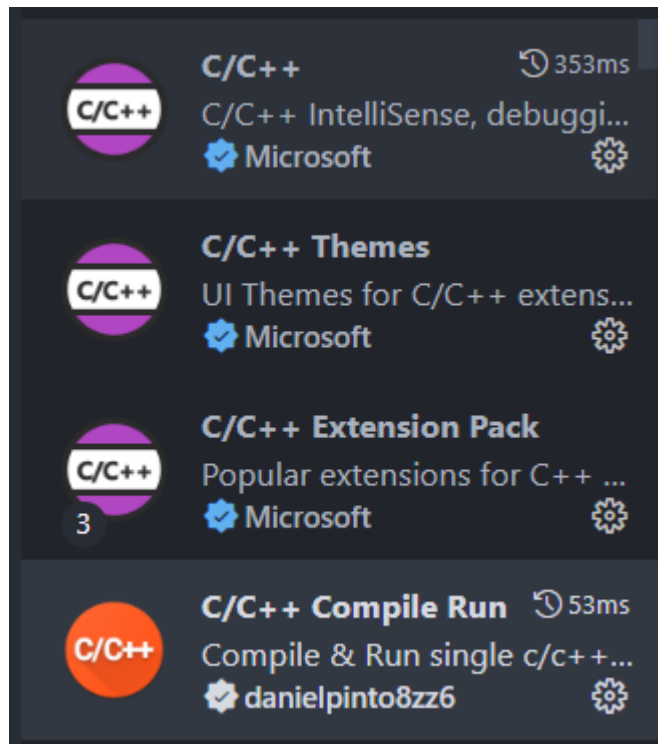
>> cmake --version
cmake version 3.26.4

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

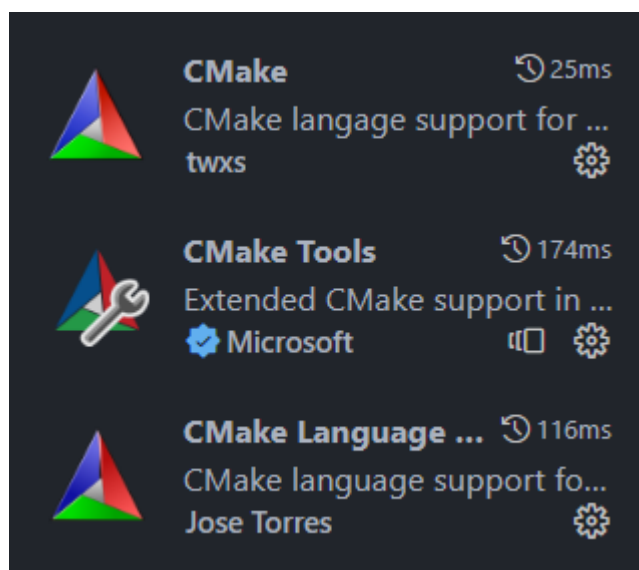
On the right side of the terminal, there are three command execution bars:

- cmd 56% 18:20:05
- cmd 56% 18:20:07
- cmd 55% 18:20:23

Installing necessary extensions

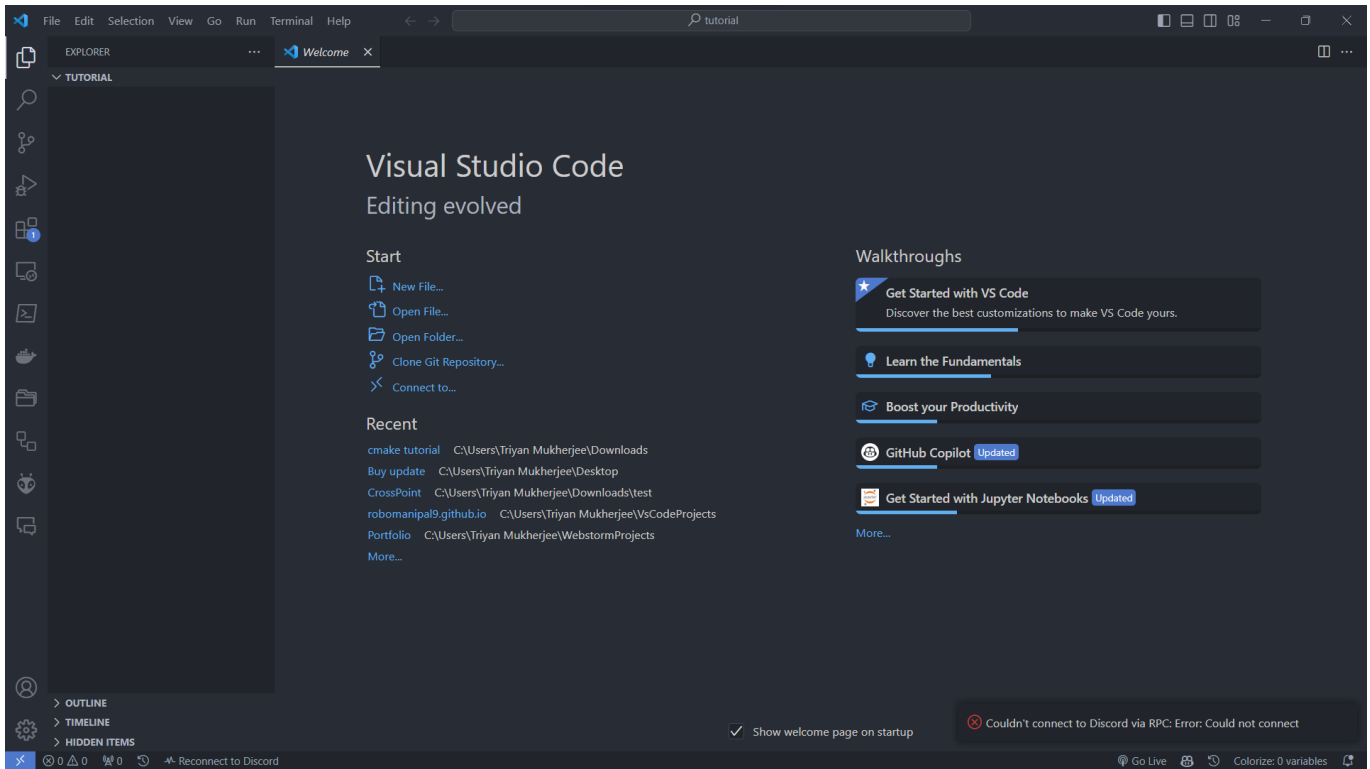


- C/C++ Compile Run extension is an added bonus which allows you to compile and run your code from within VS Code at click of a button.

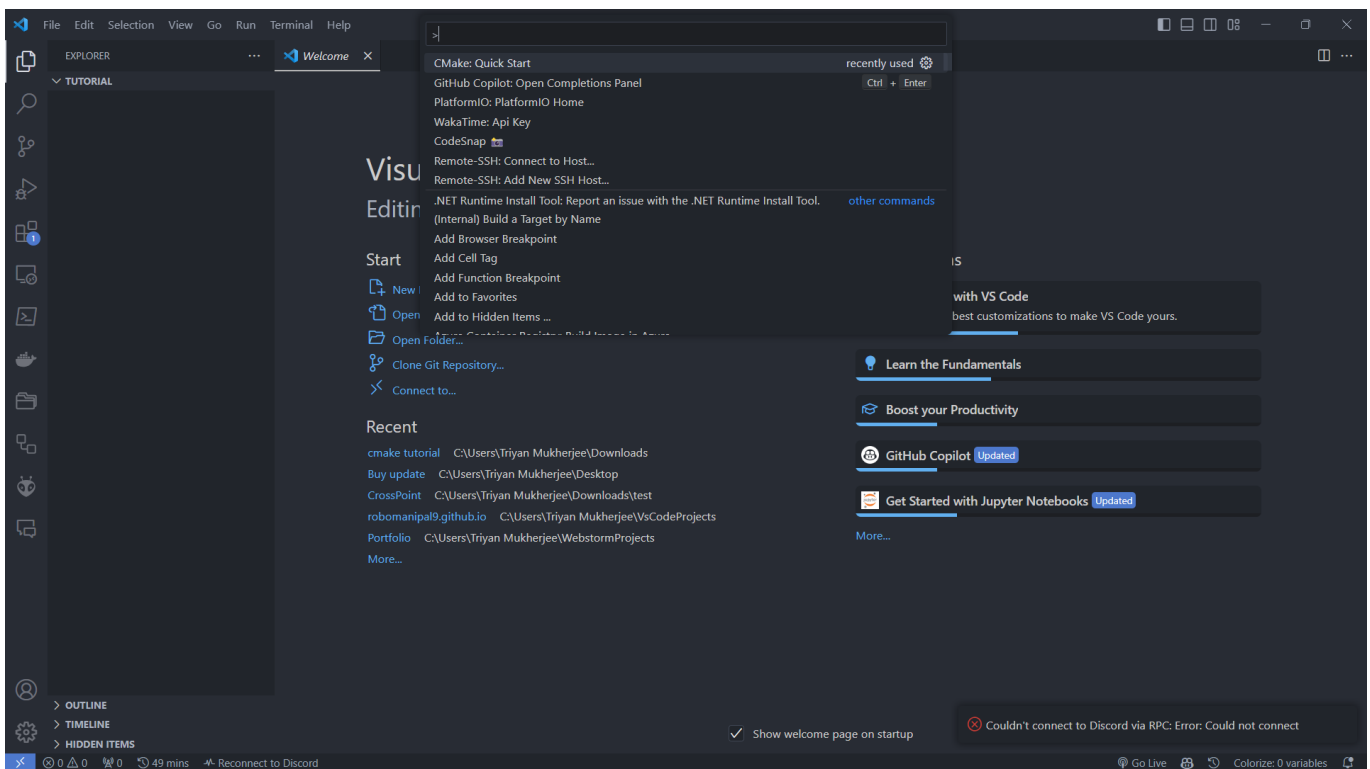


Setting up a Project

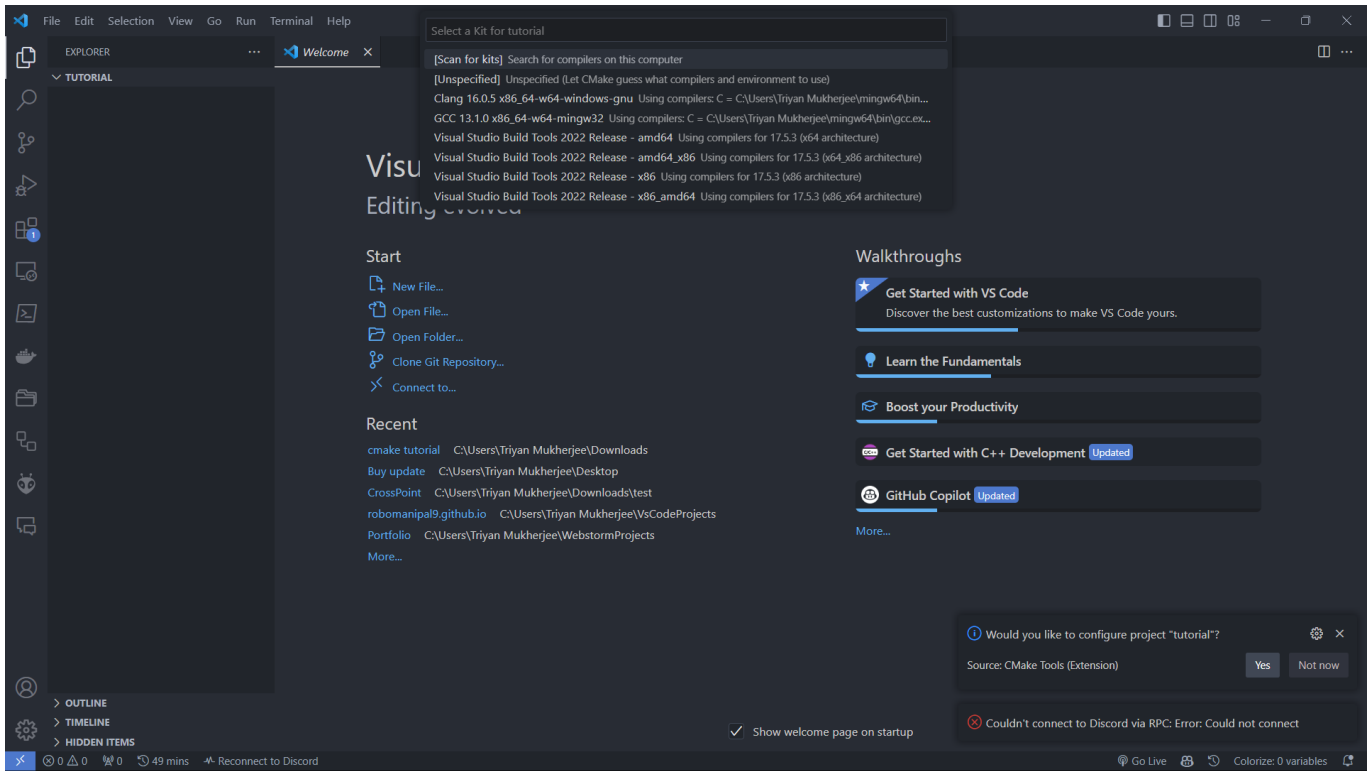
Create a folder for your project and open it in VS Code. In this case we will name the folder `tutorial`.



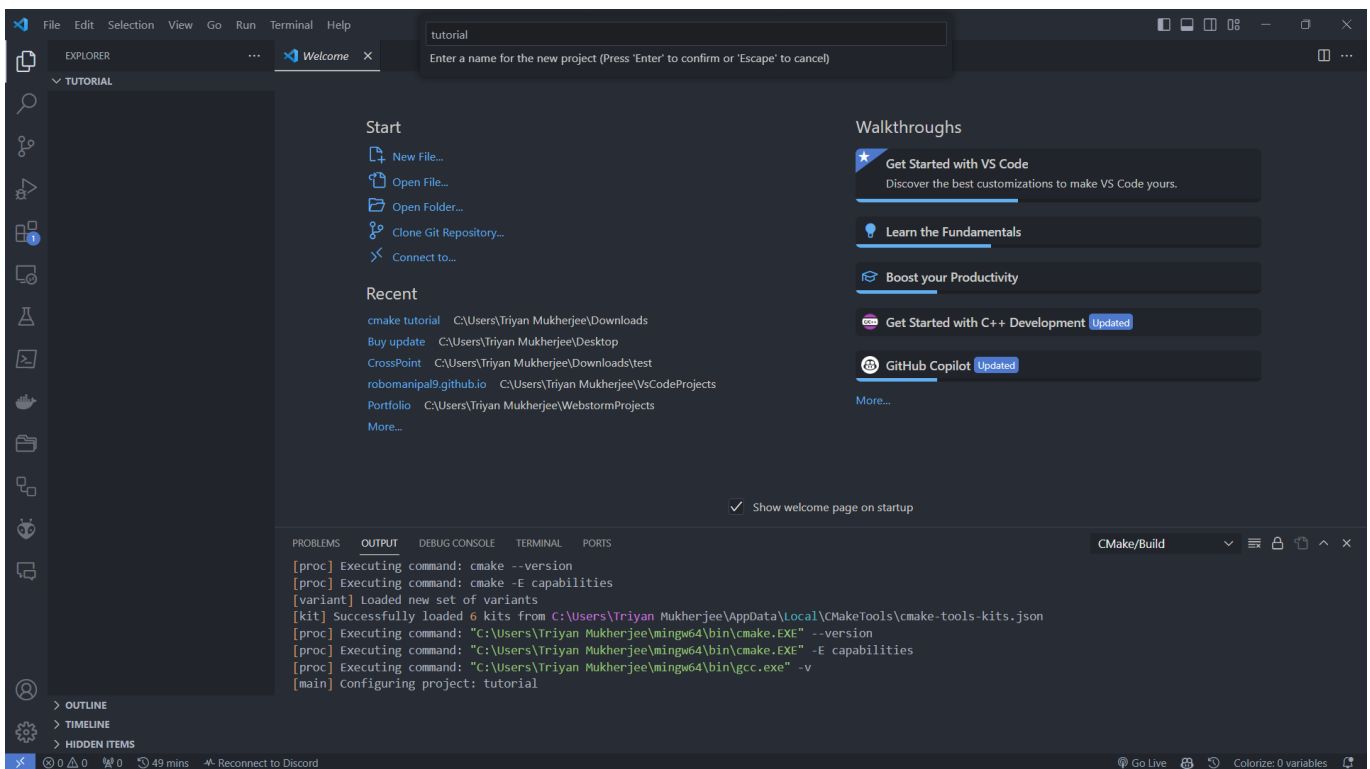
Now we are going to use the **cmake tools** extension to setup the project. Press **Ctrl+Shift+P** and type **cmake** and select **CMake: Quick Start**.



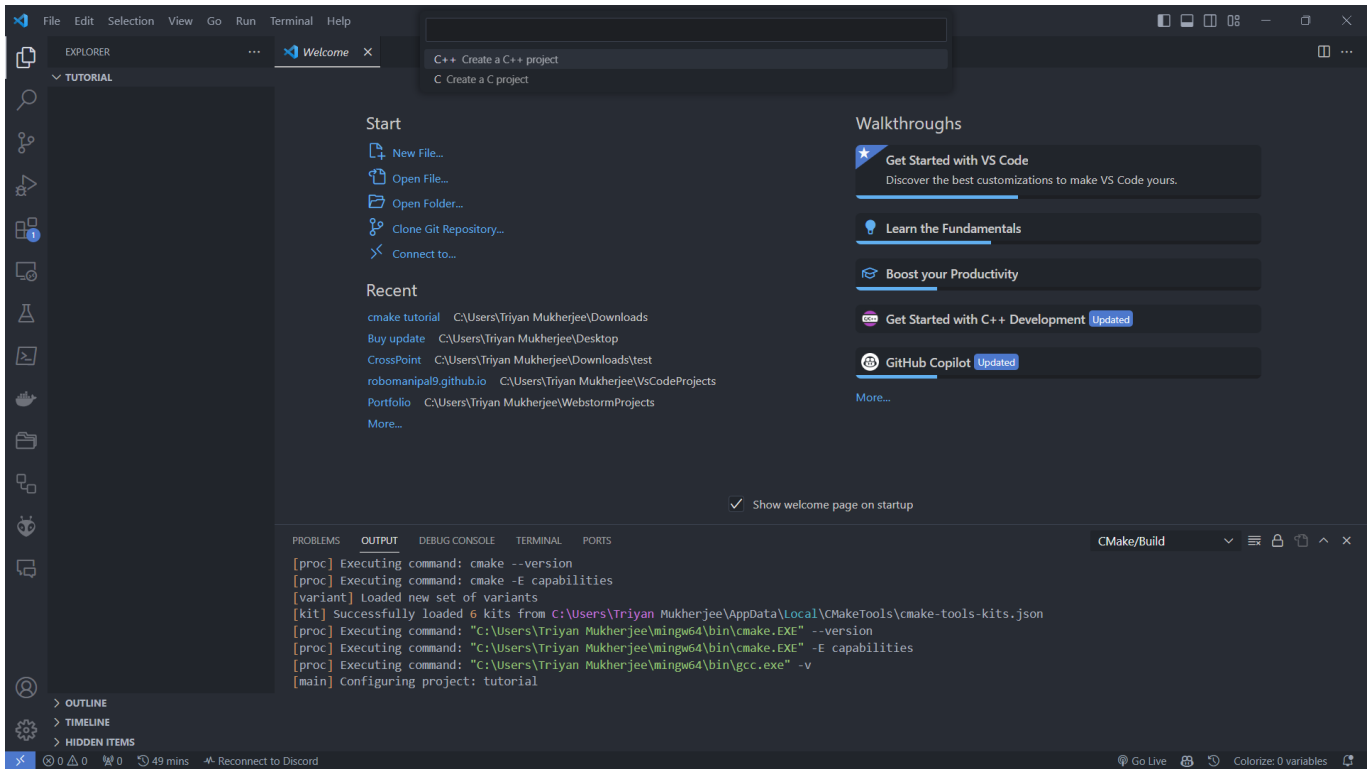
Select **quick start** and press **Enter**. You should see a similar prompt.



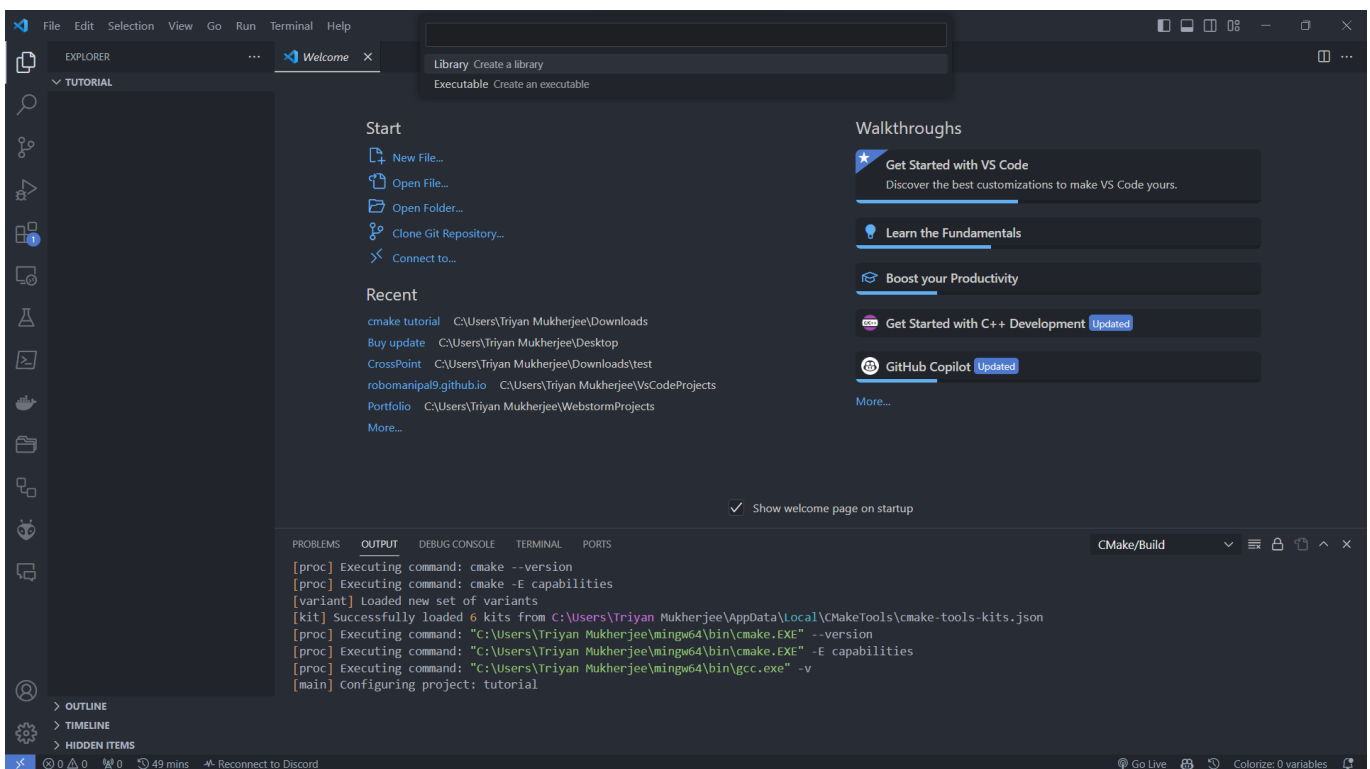
If your installation directory of **mingw** compiler doesn't show up, select **[Scan for kits]** and select the **mingw** compiler.



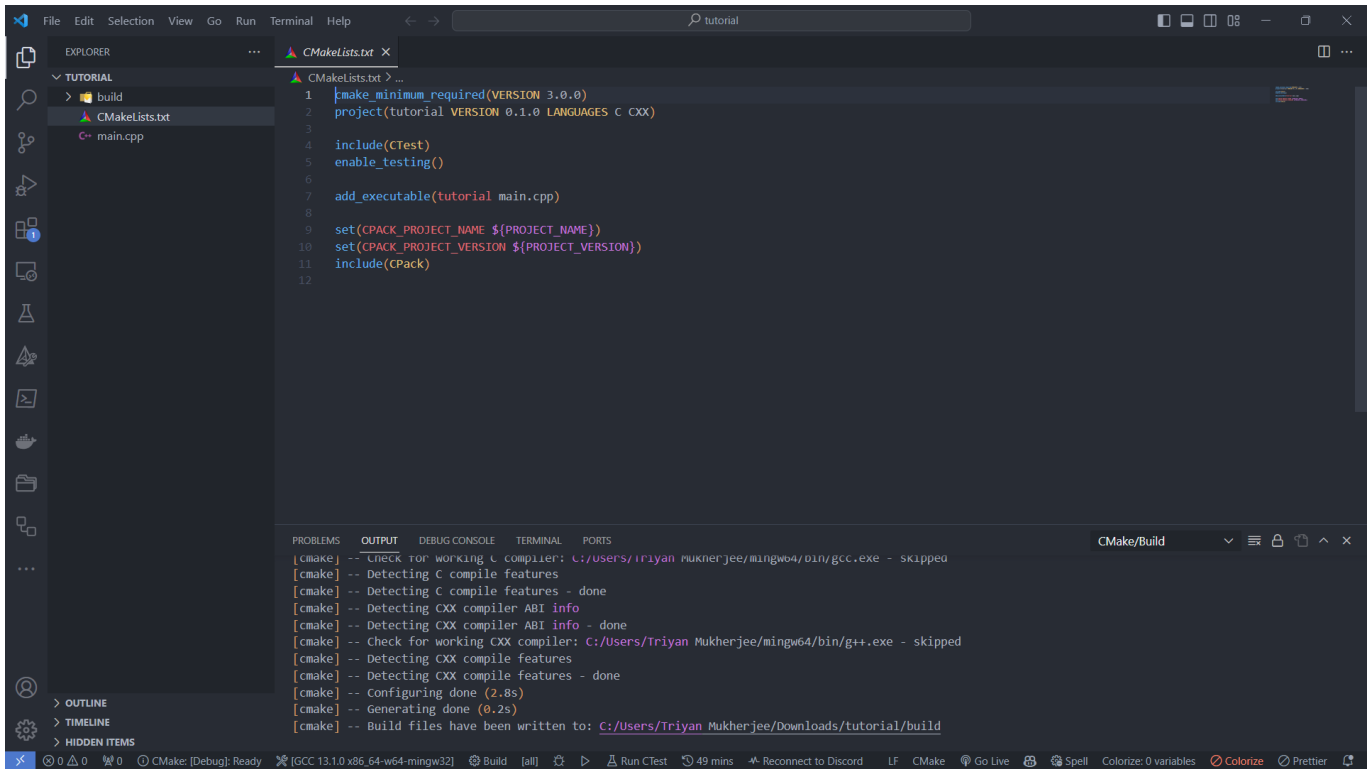
Enter your project name and press **Enter**.



Select a project type in this case we will be going with **C++**.



Select a build type in this case we will be going with **Executable**.



The screenshot shows the Visual Studio Code interface with the CMakeLists.txt file open in the editor. The file contains the following code:

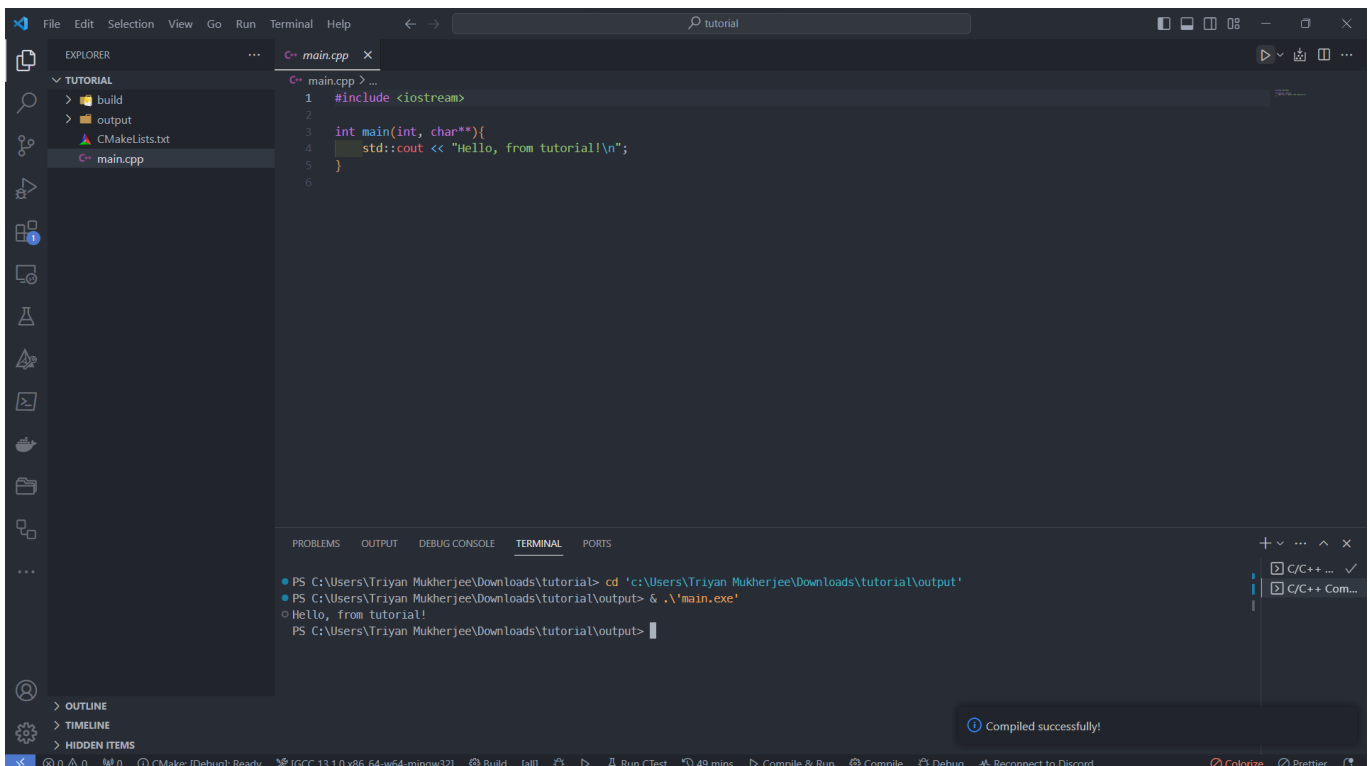
```
1 cmake_minimum_required(VERSION 3.0.0)
2 project(tutorial VERSION 0.1.0 LANGUAGES C CXX)
3
4 include(CTest)
5 enable_testing()
6
7 add_executable(tutorial main.cpp)
8
9 set(CPACK_PROJECT_NAME ${PROJECT_NAME})
10 set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
11 include(CPack)
12
```

The terminal output shows the CMake command being executed, which configures the project and generates build files. The output is as follows:

```
[cmake] -- Check for working C compiler: C:/Users/Triyan Mukherjee/mingw64/bin/gcc.exe - skipped
[cmake] -- Detecting C compile features
[cmake] -- Detecting C compile features - done
[cmake] -- Detecting CXX compiler ABI info
[cmake] -- Detecting CXX compiler ABI info - done
[cmake] -- Check for working CXX compiler: C:/Users/Triyan Mukherjee/mingw64/bin/g++.exe - skipped
[cmake] -- Detecting CXX compile features
[cmake] -- Detecting CXX compile features - done
[cmake] -- Configuring done (2.8s)
[cmake] -- Generating done (0.2s)
[cmake] -- Build files have been written to: C:/Users/Triyan Mukherjee/Downloads/tutorial/build
```

With this a basic setup of the project is completed.

Running the Project



The screenshot shows the Visual Studio Code interface with the main.cpp file open in the editor. The file contains the following code:

```
1 #include <iostream>
2
3 int main(int, char**){
4     std::cout << "Hello, from tutorial!\n";
5 }
6
```

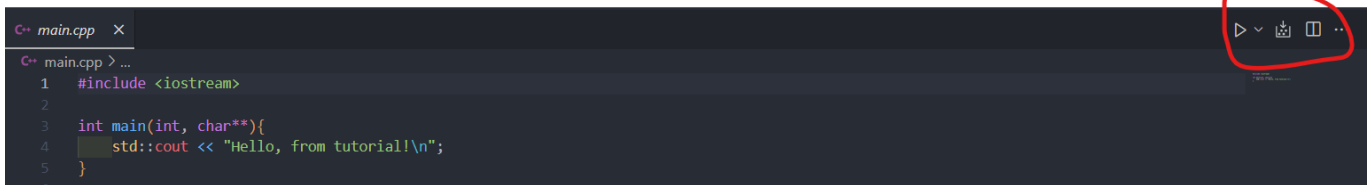
The terminal output shows the compilation and execution of the project. The output is as follows:

```
PS C:\Users\Triyan Mukherjee\Downloads\tutorial> cd 'C:\Users\Triyan Mukherjee\Downloads\tutorial\output'
PS C:\Users\Triyan Mukherjee\Downloads\tutorial\output> & .\main.exe
Hello, from tutorial!
PS C:\Users\Triyan Mukherjee\Downloads\tutorial\output>
```

A notification at the bottom right of the terminal indicates that the project was compiled successfully.

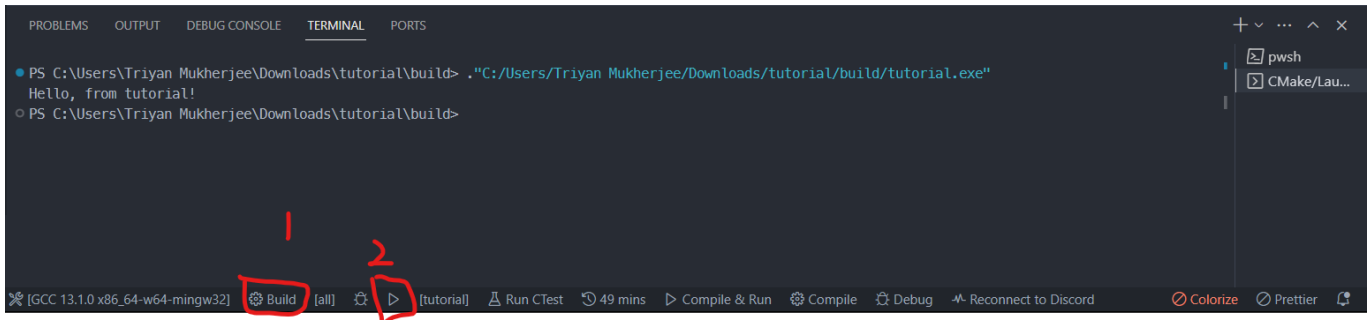
There are two ways to run the project.

- Using the previous runner extension installed.



```
C++ main.cpp > ...  
1 #include <iostream>  
2  
3 int main(int, char**){  
4     std::cout << "Hello, from tutorial!\n";  
5 }
```

- Using the cmake tools extension.

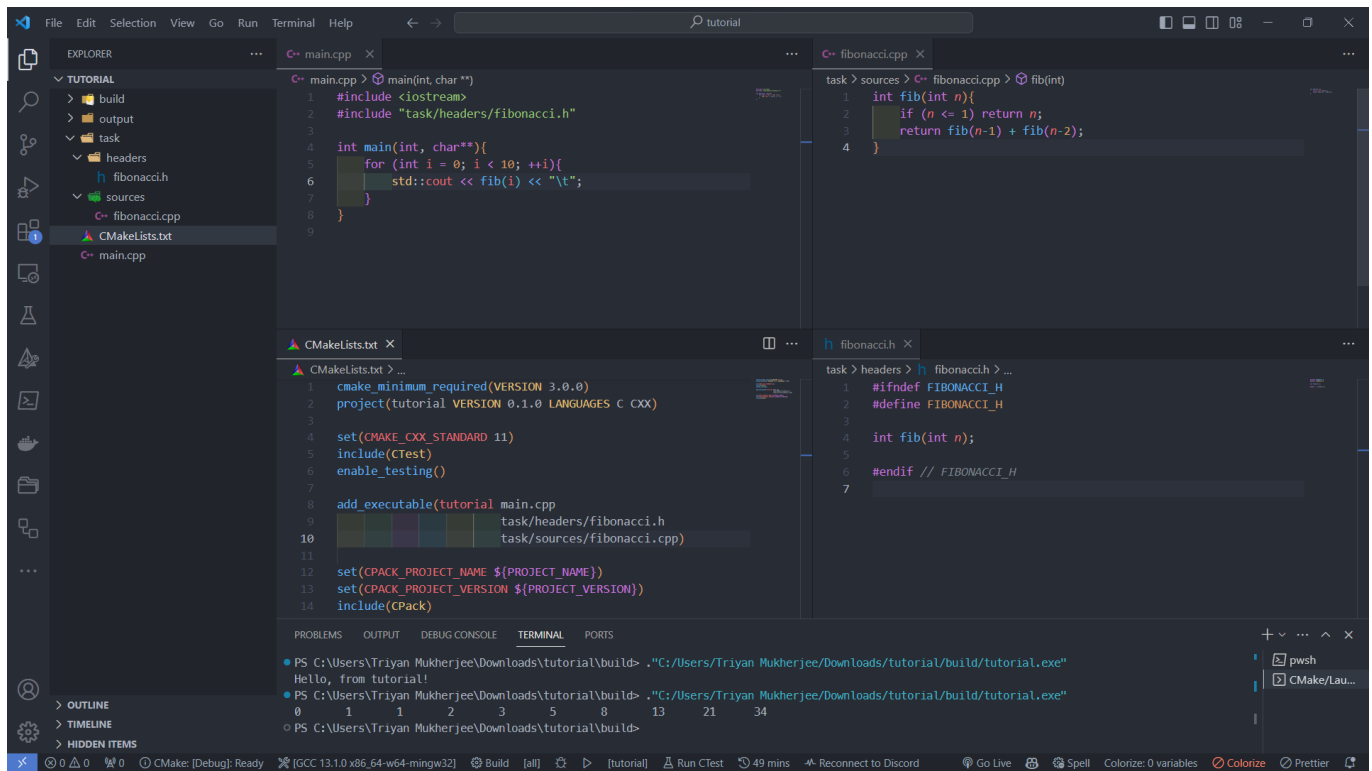


```
PS C:\Users\Triyan Mukherjee\Downloads\tutorial\build> .\C:\Users\Triyan Mukherjee\Downloads\tutorial\build\tutorial.exe"  
Hello, from tutorial!  
PS C:\Users\Triyan Mukherjee\Downloads\tutorial\build>
```

Additional Information

```
# Specify the minimum version for CMake  
cmake_minimum_required(VERSION 3.0.0)  
# Project's name and language mostly metadata  
project(tutorial VERSION 0.1.0 LANGUAGES C CXX)  
  
# Set the C++ standard to C++ 11  
set(CMAKE_CXX_STANDARD 11)  
# CTest is a testing tool that can be used to test your project.  
include(CTest)  
enable_testing()  
  
# Add the executable. Add all the source files here.  
add_executable(tutorial main.cpp)  
  
set(CPACK_PROJECT_NAME ${PROJECT_NAME})  
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})  
include(CPack)
```

Here is an example with a few more files and folders.



```
1 #include <iostream>
2 #include "task/headers/fibonacci.h"
3
4 int main(int, char**){
5     for (int i = 0; i < 10; ++i){
6         std::cout << fib(i) << "\t";
7     }
8 }
9
```

```
1 cmake_minimum_required(VERSION 3.0.0)
2 project(tutorial VERSION 0.1.0 LANGUAGES C CXX)
3
4 set(CMAKE_CXX_STANDARD 11)
5 include(CTest)
6 enable_testing()
7
8 add_executable(tutorial main.cpp
9                 task/headers/fibonacci.h
10                task/sources/fibonacci.cpp)
11
12 set(CPACK_PROJECT_NAME ${PROJECT_NAME})
13 set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
14 include(CPack)
```

```
1 #ifndef FIBONACCI_H
2 #define FIBONACCI_H
3
4 int fib(int n);
5
6 #endif // FIBONACCI_H
```

```
PS C:\Users\Triyan Mukherjee\Downloads\tutorial\build> .\"C:/Users/Triyan Mukherjee/Downloads/tutorial/build/tutorial.exe"
Hello, from tutorial!
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Triyan Mukherjee\Downloads\tutorial\build>
```

Note

IDE's like **visual studio** and **clion** auto adds files in **CMakeLists.txt** but in VS Code you have to manually add them. PS **visual studio** and **visual studio code** are two different things.