



Development Environment and Tools Setup

1 Introduction

The goal of this laboratory is to correctly set up both your development environment, and the tools, that are required for this course. It is best to install the software in the order given below. Once everything has been installed, you will check that it has been done correctly by compiling and running a simple program. For all subsequent labs, we will assume that you have set up your environment correctly.

These instructions are written for a Windows 64-bit system. However, all the required software is cross-platform and can run on macOS and Linux, but there will be some configuration differences.

The main laboratory outcomes are:

1. You have correctly installed the Visual Studio Code IDE along with the GCC compiler and a number of other tools that we will be using during the course.
2. You can successfully compile and build the C++ “hello world” program.

There is no submission for this lab and it does not count for marks.

2 Software Installation

2.1 MSYS2 GCC compiler suite

In order to run our C++ code we need a compiler. We will be using the GCC (GNU Compiler Collection) compiler suite which is a widely used, free, and open-source collection of compilers and other tools. These tools were originally developed for Unix systems and have been ported to Windows through MinGW and MSYS2.

To install the GCC compiler on Windows[†] [download the MSYS2 installer](#). Run the installer and accept the default settings. A terminal will automatically open once the installation is complete. In this terminal, install the MinGW-w64 toolchain by running the following command:

```
pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

*As is typical in programming languages, the labs use a 0-based index.

[†]If you are running macOS, refer to the [installation instructions for Clang](#).

Accept the default number of packages in the toolchain group by pressing `enter` and then press `Y` when prompted whether to proceed with the installation. Note, this is relatively large installation and it can fail because of internet timeouts. If you see any errors or warnings during the installation, then uninstall MSYS2 using Windows's *Add or remove programs* and start again.

Lastly, and very importantly, it is necessary to add the path of your MinGW-w64 bin folder to the Windows PATH environment variable using the following steps:

1. In the Windows search bar, type *Settings* to open your Windows Settings.
2. Search for *Edit environment variables for your account*.
3. In your *User variables*, select the Path variable and then select *Edit*.
4. Select *New* and add the MinGW-w64 destination folder you recorded during the installation process to the list. If you used the default settings above, then this will be the path: `C:\msys64\ucrt64\bin`
5. Select *OK*, and then select *OK* again in the *Environment Variables* window to update the PATH environment variable. You have to reopen any console windows for the updated PATH environment variable to be available.

After working through the above steps, it is important to check your MinGW installation. You can open a command prompt by typing `cmd` and then `cmd`. When checking the versions of the MinGW-w64 tools, you should see the following versions (or later):

```
C:\>gcc --version
gcc (Rev5, Built by MSYS2 project) 15.1.0

C:\>g++ --version
g++ (Rev5, Built by MSYS2 project) 15.1.0

C:\>gdb --version
GNU gdb (GDB) 16.3
```

gcc is the C compiler; g++ is the C++ compiler; gdb is the debugger.

2.1.1 Handling Errors

If you receive an error message like:

```
'gcc' is not recognized as an internal or external command,
operable program or batch file.
```

then this means that the Windows PATH variable has not been correctly modified — refer to the steps given above for setting the PATH.

If the version commands work but you do not see the correct version then this means that you have more than one version of the MinGW compiler installed (perhaps via Code::Blocks or a previous installation of MSYS2). The best way to fix this is to uninstall any other instances of MinGW that you are not using. To find other versions that you have installed you can use Everything. Everything is a useful Windows application which indexes your hard drive allowing you to easily and quickly find files and directories. [Download](#) the 64-bit installer and run it. Accept the defaults, but actively decide whether you wish to run Everything at system startup (by ticking/unticking the relevant checkbox).

Wait a minute or two while Everything runs and indexes your drive, and then search for `g++.exe`. If it finds more than one instance of this file then you have multiple installations

on your computer. From each instance's path you should be able to tell which application installed it. Use Windows's *Add or remove programs* to remove installations which you do not need.

It is essential that you have the correct MinGW version installed. Although, you may be able to compile and run the “hello world” program in this lab, **you will have problems** later on with an incorrect version.

2.2 Visual Studio Code

It is now time to download and [install Visual Studio Code](#), aka VS Code. This is an extremely popular (50 million monthly active users) and versatile integrated development environment (IDE) which can be used to write code in C++, JavaScript, Python, and many other languages as well as documentation in \LaTeX and Markdown. Run the installer and accept the defaults. You can check the box for adding an icon to the desktop.

2.3 Git Bash

We will be using Git (a version control system) and GitHub (a website which hosts Git repositories) extensively throughout the course. Git/GitHub form a powerful combination for collaborating with others on a codebase, and we will be discussing this in much more detail.

Git Bash is the official command-line Git client for Windows. There are many different Git clients, including graphical ones. In this course, we will focus on using Git via the command line as it helps to promote a solid understanding of what Git is doing. Additionally, it is worth noting that some Git functionality is simply not available in third-party clients.

Download and install [Git Bash for Windows](#). Accept the defaults offered by the installer, except for the options shown in the [Figure 1](#) and [Figure 2](#).

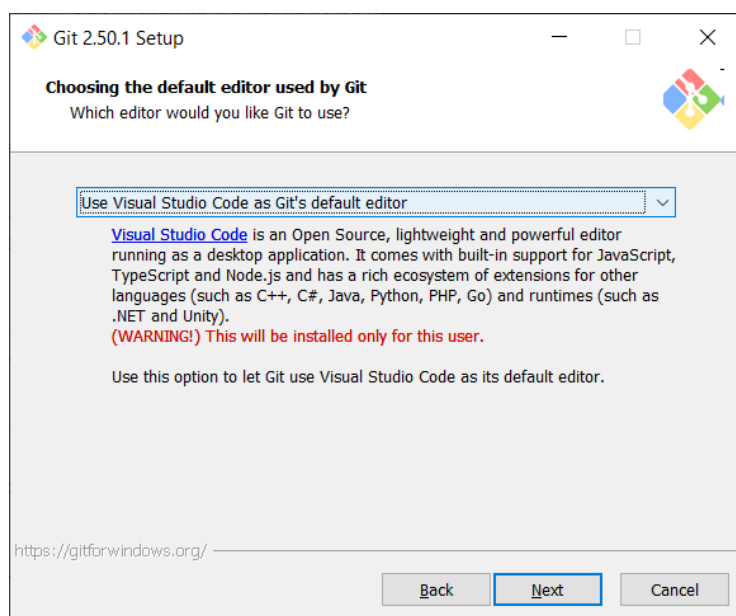


Figure 1: Select VS Code as the default editor

Now check that Git has been installed correctly by opening up the command prompt and checking that the version matches that below, or is more recent:

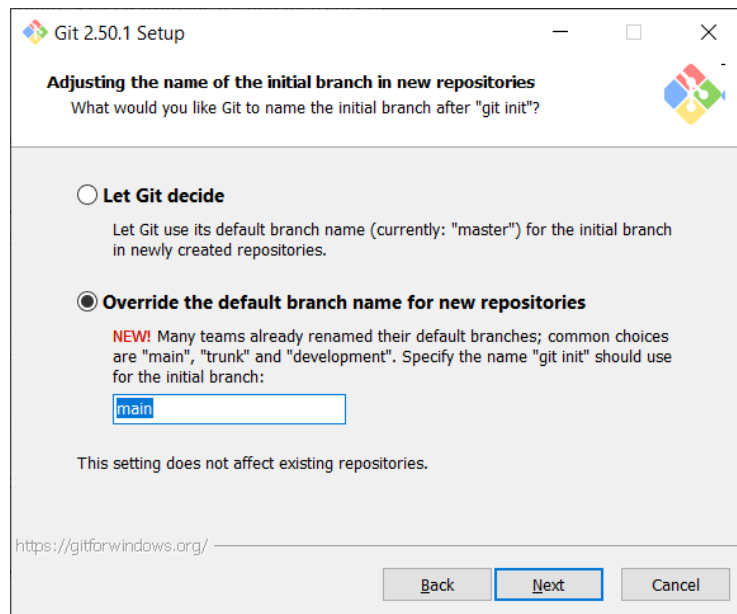


Figure 2: Use “main” for the default branch name (as is done on GitHub)

```
C:\>git --version
git version 2.50.1.windows.1
```

There is still some configuration to do before you can actually use Git. This will be done in Lab 1.

2.4 CMake

CMake is a tool that is used for automating the installation, testing and packaging of C++ applications. We will be using this to automate the compilation, linking and building of the course project and some of the labs. It will be discussed in further detail later in the course. For now just [download and install CMake](#). Accept the defaults when installing.

Check that you have correctly installed CMake by opening a new command prompt and checking the version:

```
C:\>cmake --version
cmake version 4.0.3
```

2.5 Doxygen

Lastly, you need to download and install [Doxygen](#). Download the 64-bit system installer.

If you are using Microsoft’s Edge browser, it incorrectly marks the Doxygen executable as an unsafe file. You need to explicitly choose to keep the file when downloading it. Select the ellipses in the *Downloads* dropdown and choose *Keep* and then *Show more*, followed by *Keep Anyway*.

Accept the defaults as you perform the installation. Doxygen is used for extracting source code comments as program documentation. We will use this for documenting the course project. Confirm that you have the version shown below (or a later version):

```
C:\>doxygen --version
1.14.0 (cbe58f6237b2238c9af7f51c6b7afb8bbf52c866)
```

2.6 VS Code Extensions

VS Code achieves its versatility through the multitude of extensions that have been developed which can tailor the IDE to support specific languages, libraries, and tools.

2.6.1 The SD2 Profile

A collection of useful extensions and settings for this course are available as a VS Code profile. The profile file can be downloaded from the Laboratory 0 page on the course website. You can then import the profile by clicking on *File>Preferences>Profiles* and then the down arrow next to *New Profile*, select *Import Profile...*, *Select File...*, and find the file (see this [video](#)). When prompted, you will need to *Trust Publishers & Install* in order to install the extensions.

Now you can change your default profile to the SD2 profile one by selecting it from the profile list. You can also make it the default for new windows.

Here is a brief description of each of the extensions that are contained in the SD2 profile:

C/C++

Provides C and C++ language support, including IntelliSense, debugging, and code browsing.

C/C++ Include Guard

Automatically adds include guards to header files to prevent multiple inclusion issues. Try it out by creating a new “.h” file.

CMake Tools

Adds support for CMake projects, enabling configuration, build, and debug directly from VS Code.

Code Spell Checker

Provides spell checking for code and comments, helping identify and correct spelling mistakes.

Doxygen Documentation Generator

Helps create documentation comments for C++ code following Doxygen standards. You will use this when documenting the code for the course project.

Git Graph

Visualizes Git repository history with a graphical interface, showing branches, commits and merges. This is a useful complement to the command-line interface that we will be using.

indent-rainbow

Colourizes indentation levels in code with different colours, making code structure easier to follow.

Material Icon Theme

Provides file icons based on Google's Material Design, making it easy to identify different file types in the project explorer.

Terminal Here

Allows a terminal to be opened at the current file's location using a keyboard shortcut. Test it by opening any file and typing `Ctrl t h` (hold down the `Ctrl` key and then press `t` followed by `h`). The VS Code integrated terminal will be opened in the directory which contains the file being viewed.

UMLet

Enables creation and editing of UML diagrams directly within VS Code, which will be useful for documenting the software design of the labs.

Do not install any additional C++ extensions beyond those provided in this profile. Extensions such as *Code Runner* conflict with the recommended setup and cause issues. If you have previously installed any C++ related extensions, please disable or uninstall them before proceeding with the labs.

2.6.2 VS Code Personalization

The SD2 profile makes use of a theme which I like (Tomorrow Night Blue) but you may prefer something else. To preview and install themes, use the command palette (`Ctrl shift p`) and type: *themes*. Select *Browse Color Themes in Marketplace* to explore the options.

In addition to the profile extensions, you may want to personalize your VS Code experience with some fun extensions. Here are two suggestions:

1. [Power Mode](#) adds particle effects and animations when you type, possibly making coding more exciting (or more annoying).
2. [VSCode Pets](#) adds a virtual pet (cat, dog, snake, etc.) to your editor that keeps you company while you code.

Feel free to explore the VS Code Marketplace for more extensions to make your coding environment enjoyable.

3 Testing your Setup: Hello World

3.1 Creating and Running the Application

Having completed all the installations, it is time to test your setup. Follow the instructions for [creating and running a hello world app](#).

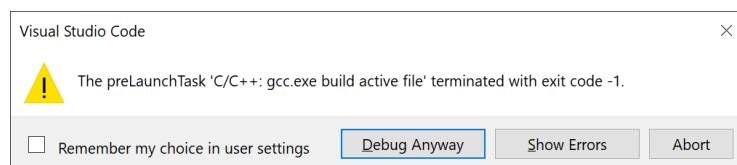



Figure 3: Pre-launch task error



Note, if you receive the error shown in [Figure 3](#) when trying to run the “hello world” program, and you have made certain that *you have the correct versions of the tools installed*, then try the following fixes in the order given.






1. Delete the `.vscode` folder and re-run the program making sure that you select `g++.exe` and not `gcc.exe`.
2. Move the path to the MinGW-w64 bin folder from the *User variables* section (as described above) to the top of the *System variables* section. Do this by pressing  and searching for *Edit the system environment variables*. You will then be able to modify the respective PATH variables.
3. Remove and re-install MSYS2 and the GCC toolchain, in case it did not download properly.

3.2 The Debugger

After having confirmed that the code runs, work through the section on how to use [the debugger](#), up to and including the section entitled: “Set a watch”. There is no need to continue beyond this section.

3.3 The Integrated Terminal

To conclude this lab it is worth noting that the SD2 profile sets the Git Bash shell as the default for the integrated terminal. “Integrated” refers to the fact that the terminal is directly accessible from within VS Code, unlike the command prompt which is external to VS Code. To toggle the visibility of the integrated terminal, type  .

Now try running your “hello world” program from the terminal. Open a terminal where the EXE file is located by viewing the `helloworld.cpp` file in the editor and typing   . Then type: `./h` and press  to autocomplete the filename to: `./helloworld.exe`. Lastly, press  to run the executable and see the output.

4 Familiarizing Yourself with the Tools

4.1 VS Code

In order to familiarize yourself with VS Code’s extensive functionality it is worth watching the following two short videos.

1. [Getting Started](#)
2. [Productivity Tips](#)

As you will have seen, [keyboard shortcuts](#), are prevalent throughout VS Code and these can greatly increase productivity. Take the time during the course to learn some of the ones related to the basic use of the editor.

4.2 Git Bash

We will use this terminal to execute our Git commands. It can also be used to navigate the directory structure, manipulate files, and so on. Review [the basic commands](#) that are

available to you.