

ENVIRONMENT SETUP

COS20007

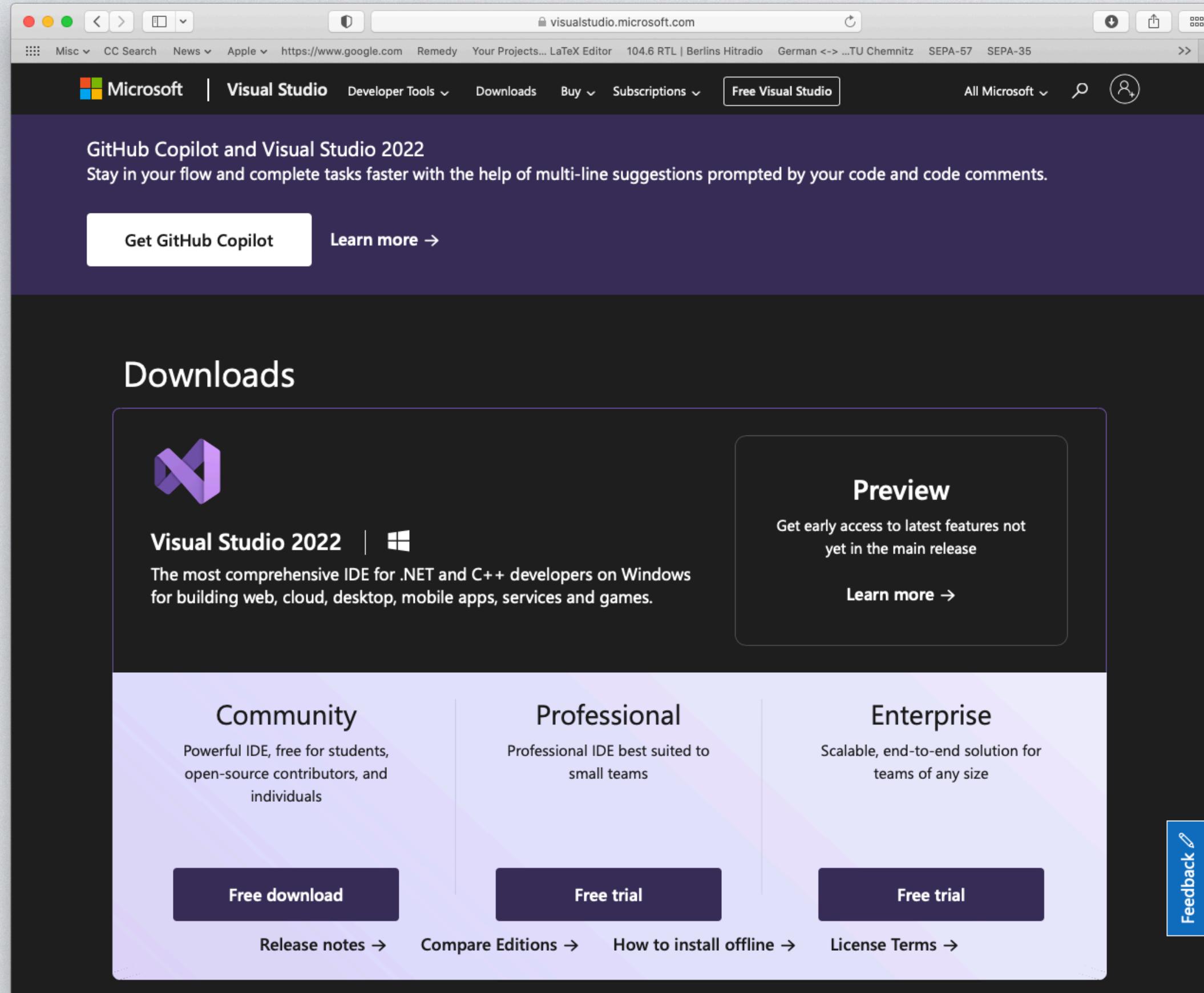
Windows

REQUIRED SOFTWARE PACKAGES

- In COS20007, you need to install and setup the following software packages:
 - Visual Studio 2022 (Community Edition) and .NET Desktop Development
 - MSYS2 (set of Unix-style command terminals and tools)
 - Git
 - SplashKit

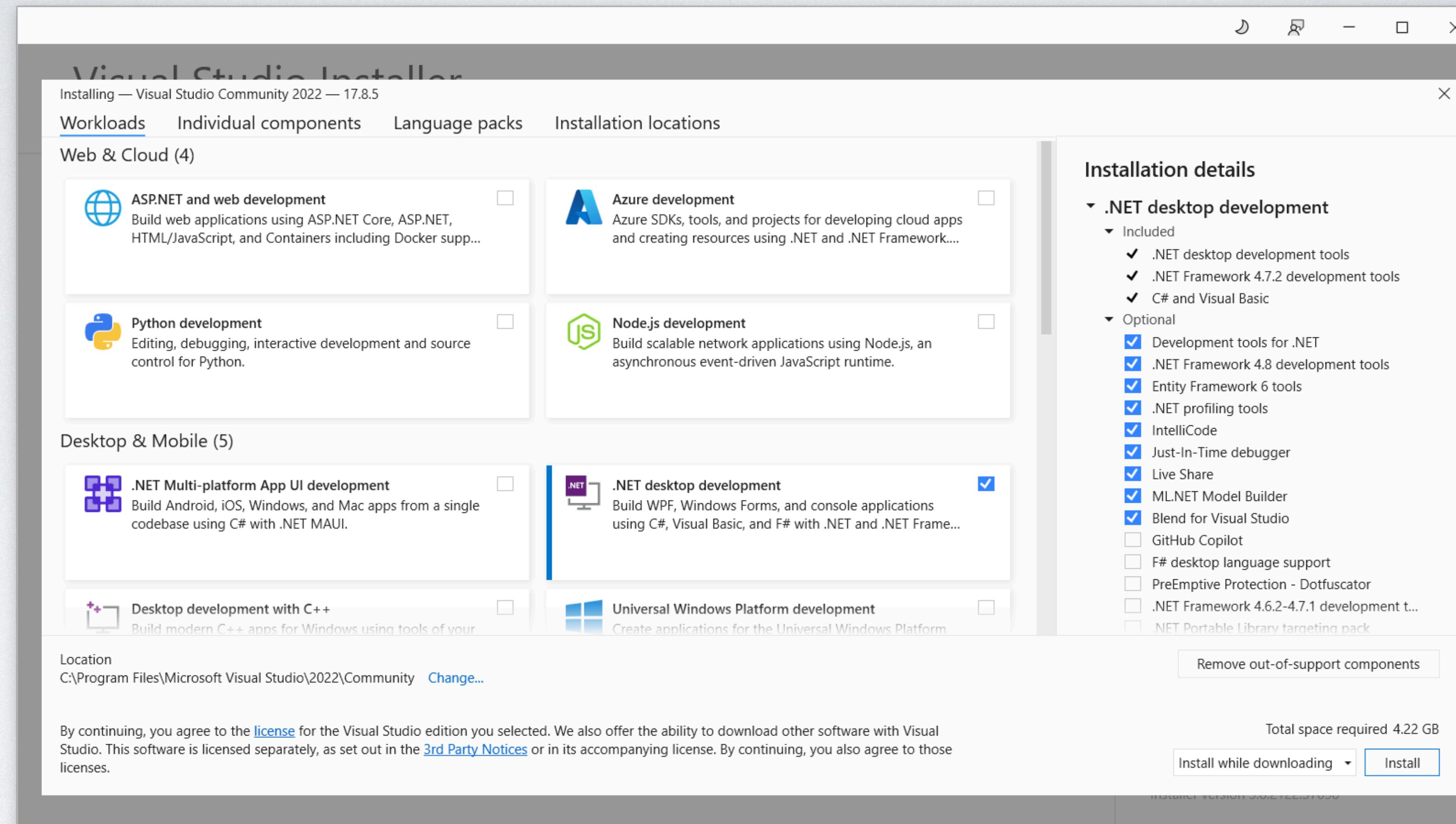
VISUAL STUDIO

VISUALSTUDIO.MICROSOFT.COM



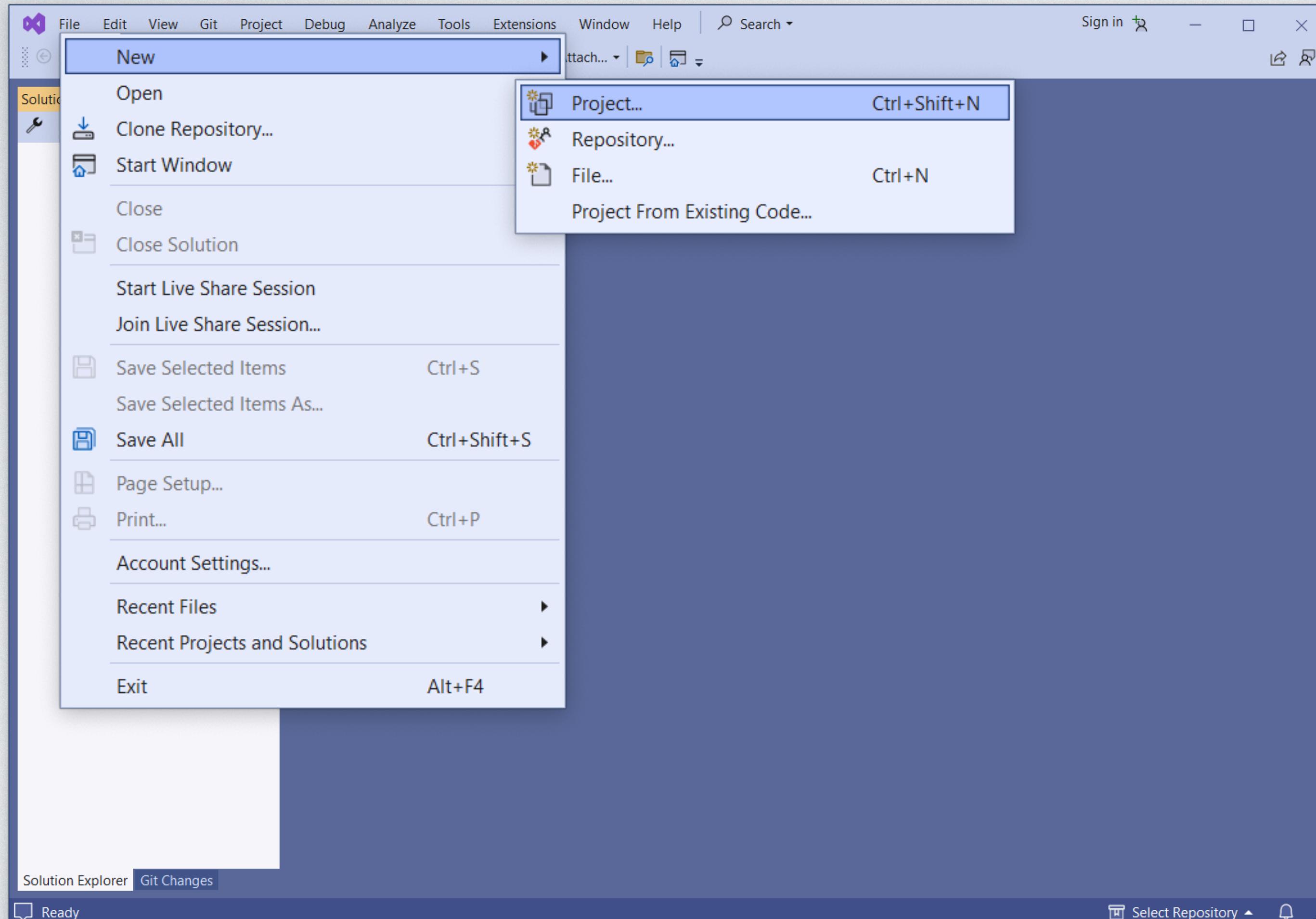
- The Community edition of Visual Studio 2022 is available at visualstudio.microsoft.com.
- Run `VisualStudioSetup` located in your Downloads folder.

VISUAL STUDIO INSTALLER



- VisualStudioSetup runs Visual Studio Installer and directs you to the Workloads.
- Select **.NET desktop development**, accept the default-selected features, and press **Install** to set up Visual Studio 2022 Community for .NET desktop development.

CREATE A NEW PROJECT



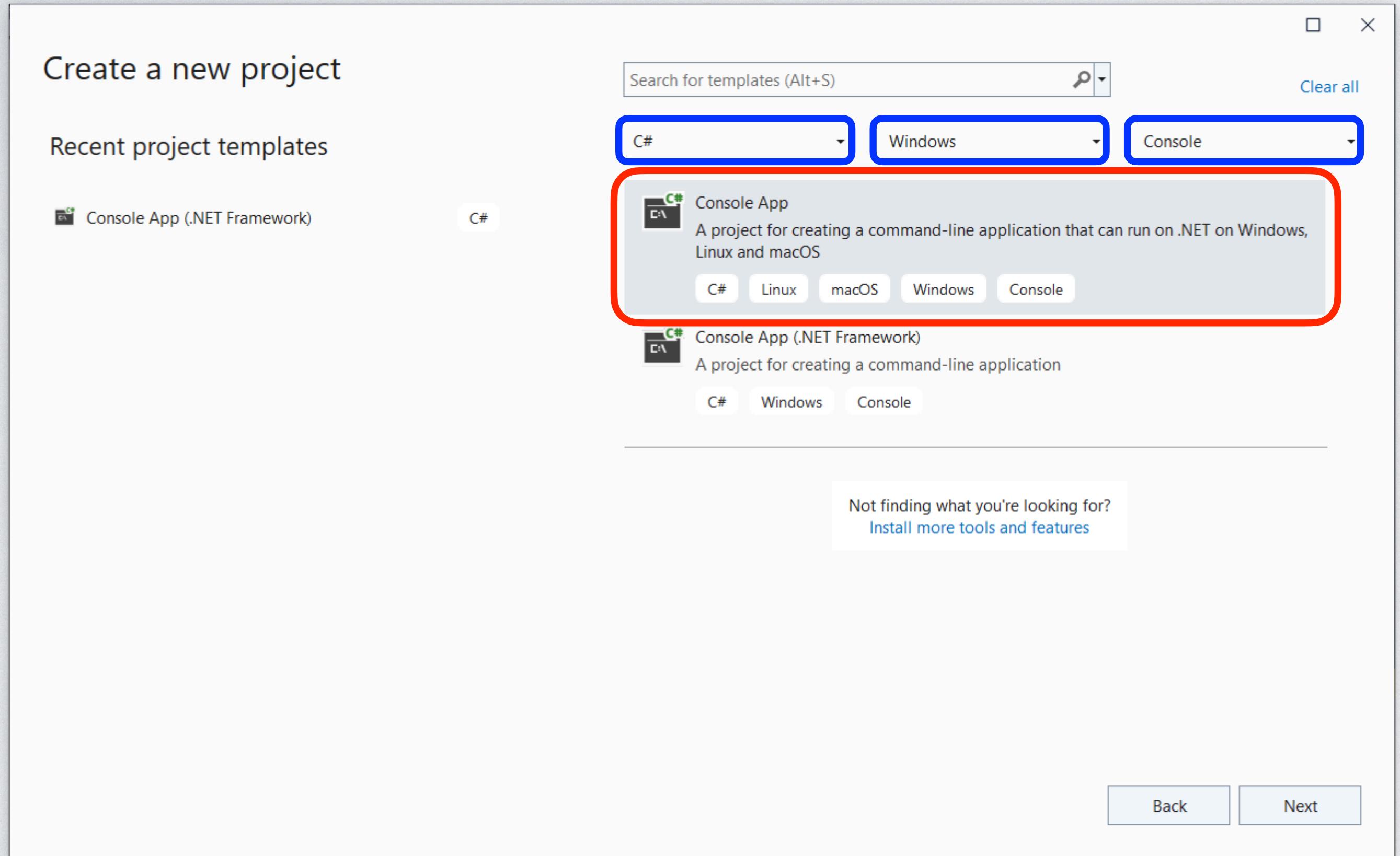
1. Open Visual Studio and continue without code.

2. Select File/New

3. Finally select Project

Please note that your mouse pointer have to hover over the menus to remain visible.

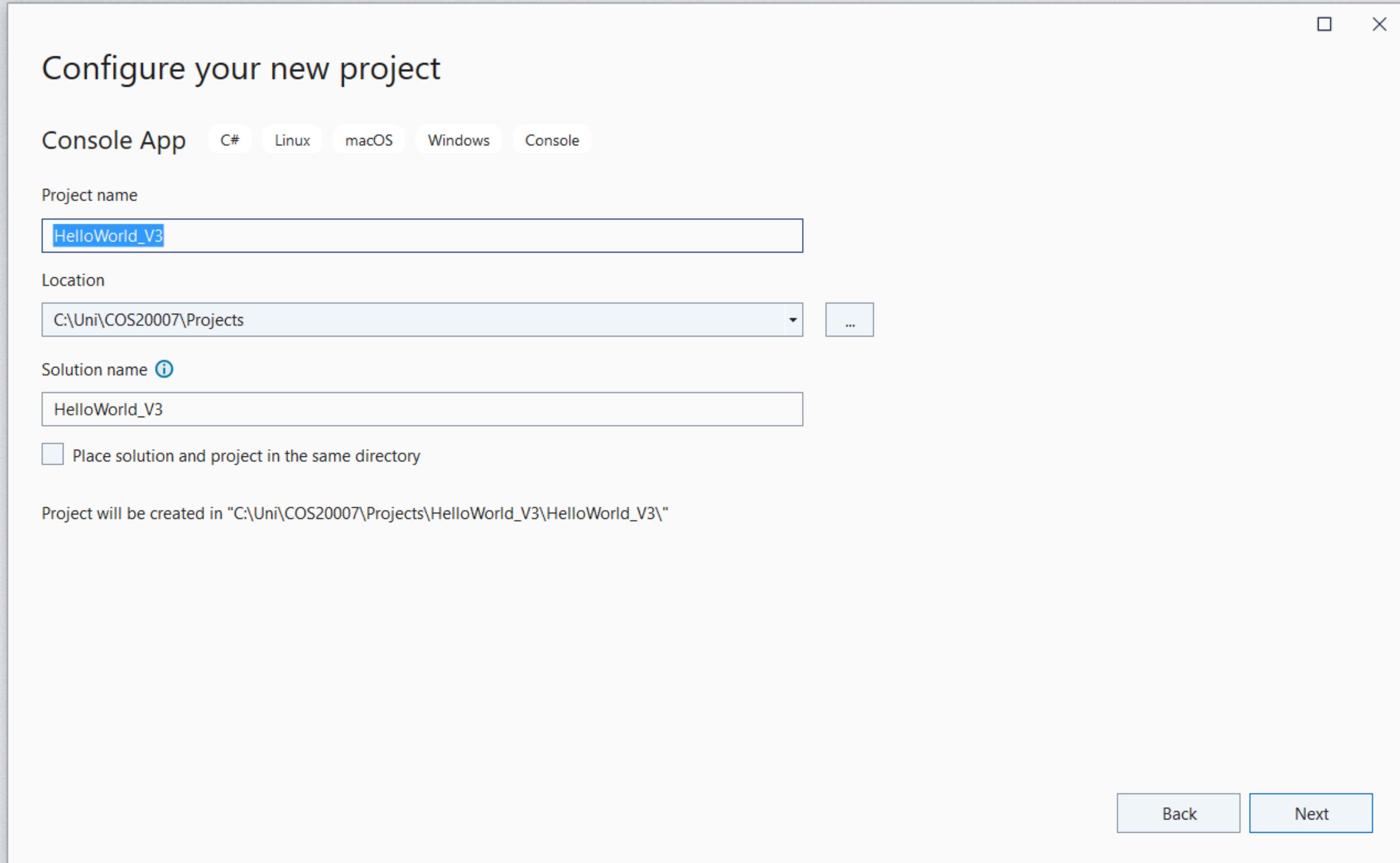
SET UP A CONSOLE APP



- Visual Studio offers a multitude of project types.
- In COS20007, we are working only with one project type: **Console App**.
- You can use the dropdown list to restrict the project type view to just the two shown on the left. Select
 - **C#** for Language
 - **Windows** for Platform
 - **Console** for Project Type

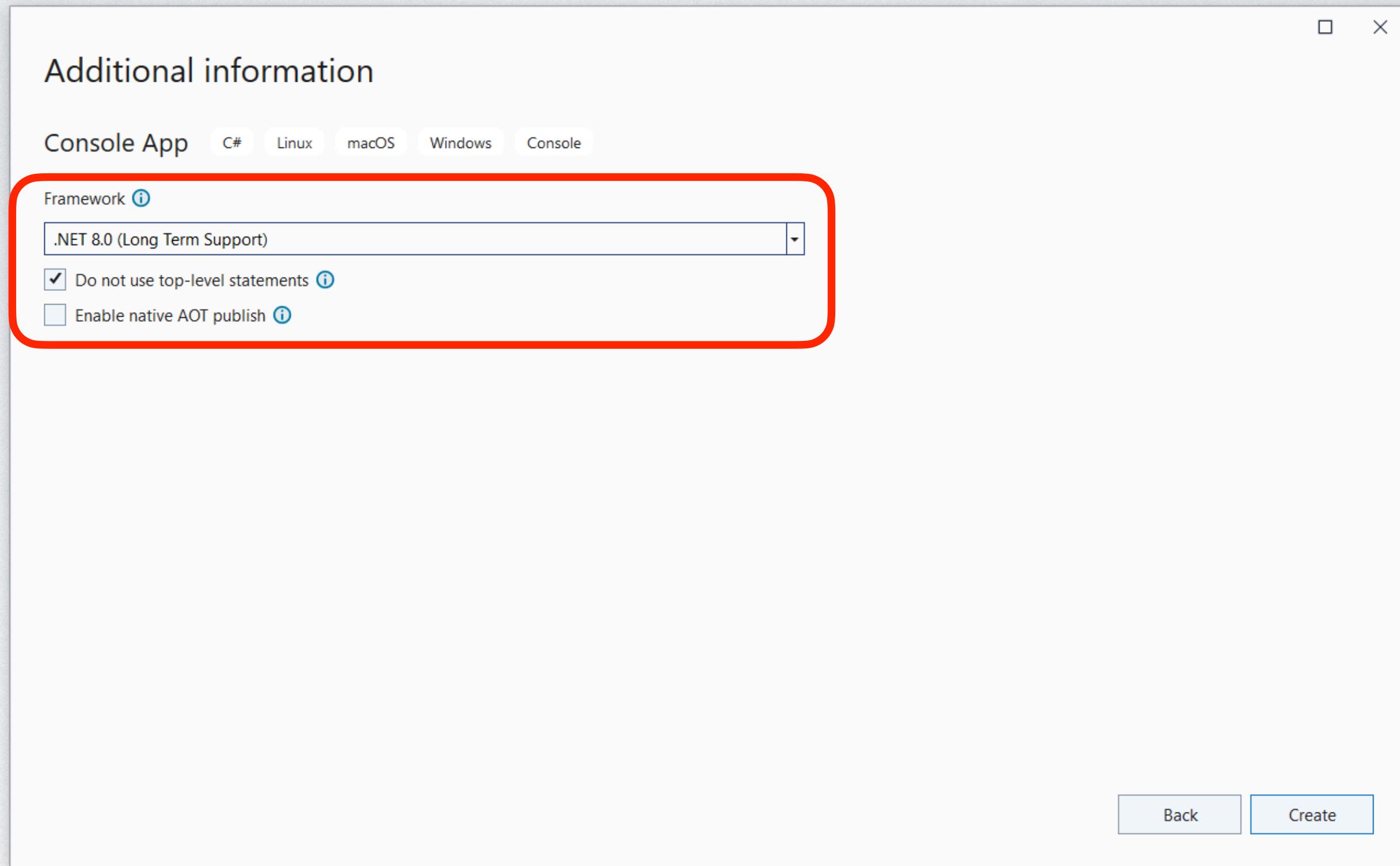
Please note that the project type **Console App** is a project template for .NET applications that can run on any CPU supporting Windows, Linux and MacOS.

SETTING UP PROJECT NAME AND LOCATION



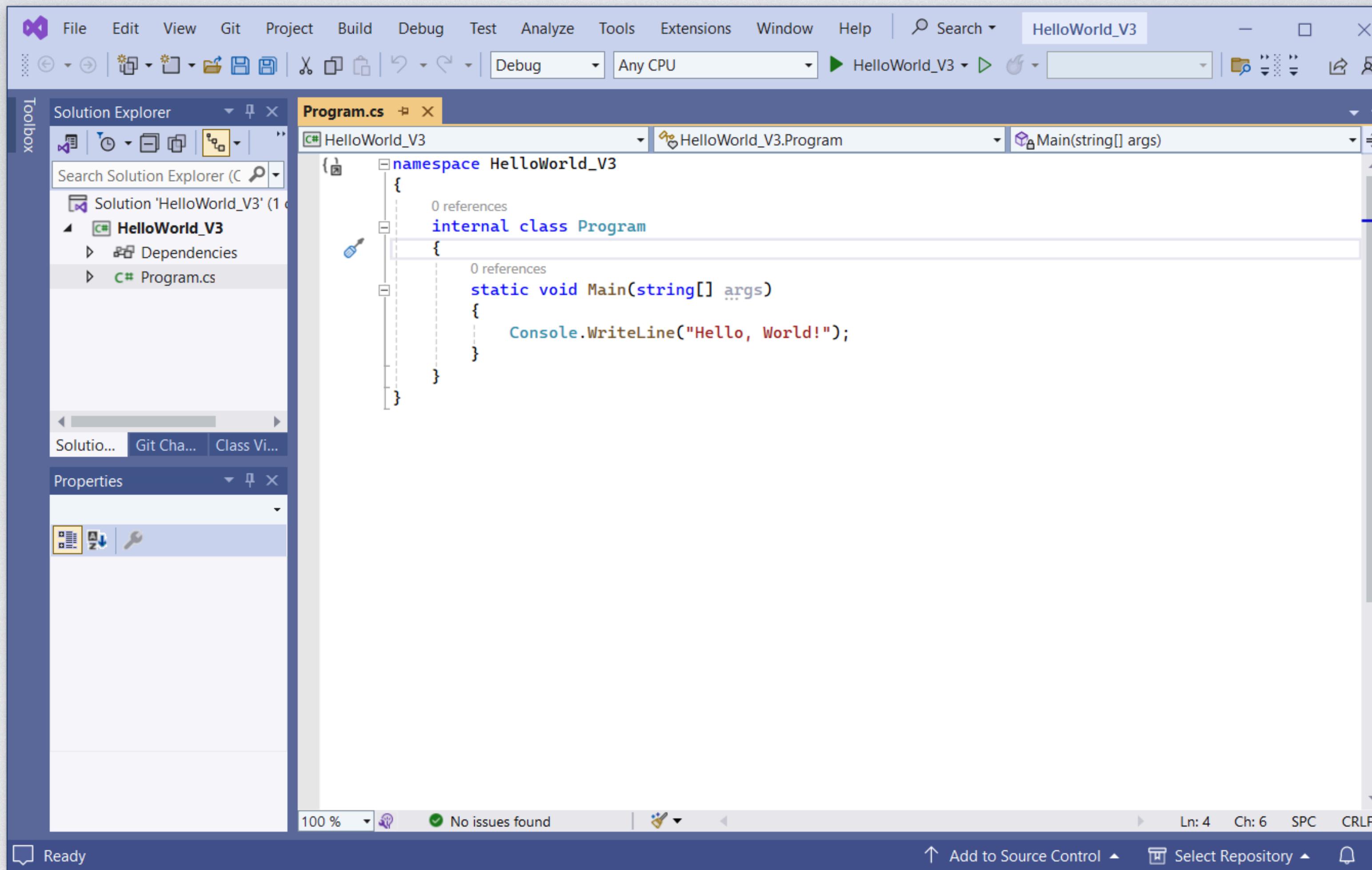
1. Select the project location. Click on ... box. Never accept the Visual Studio default. Projects in this presentation are stored in the folder <C:\Uni\COS20007\Projects>. Choose a folder that you can easily find again. You may even consider a Cloud resource.
2. Give the project a name. Here the name is [HelloWorld_V3](#). Any meaningful name will do.
3. You can change the solution name also. This is useful when a solution contains several projects. Here, the solution name coincides with the project name as the solution only hosts one project.

SELECT FRAMEWORK



- When creating a Console App for .NET, we need to select the .NET framework that we want to work with.
- You can safely choose the latest framework (.NET 8.0 LTS). At a minimum, COS20007 requires .NET 6.0.
- Check “Do not use top-level statements.” Visual Studio creates a “Hello, World!” program this way.

GENERATED PROJECT



The screenshot shows the Microsoft Visual Studio IDE interface with a generated C# console application project named "HelloWorld_V3".

- Solution Explorer:** Shows the solution "HelloWorld_V3" containing one project "HelloWorld_V3" which includes "Dependencies" and the source file "Program.cs".
- Properties:** Shows standard project properties like build configurations.
- Toolbox:** Standard .NET development tools.
- Task List:** Shows "No issues found".
- Status Bar:** Shows "Ready", "Add to Source Control", "Select Repository", and file navigation information (Ln: 4 Ch: 6 SPC CRLF).

The code editor displays the "Program.cs" file with the following content:

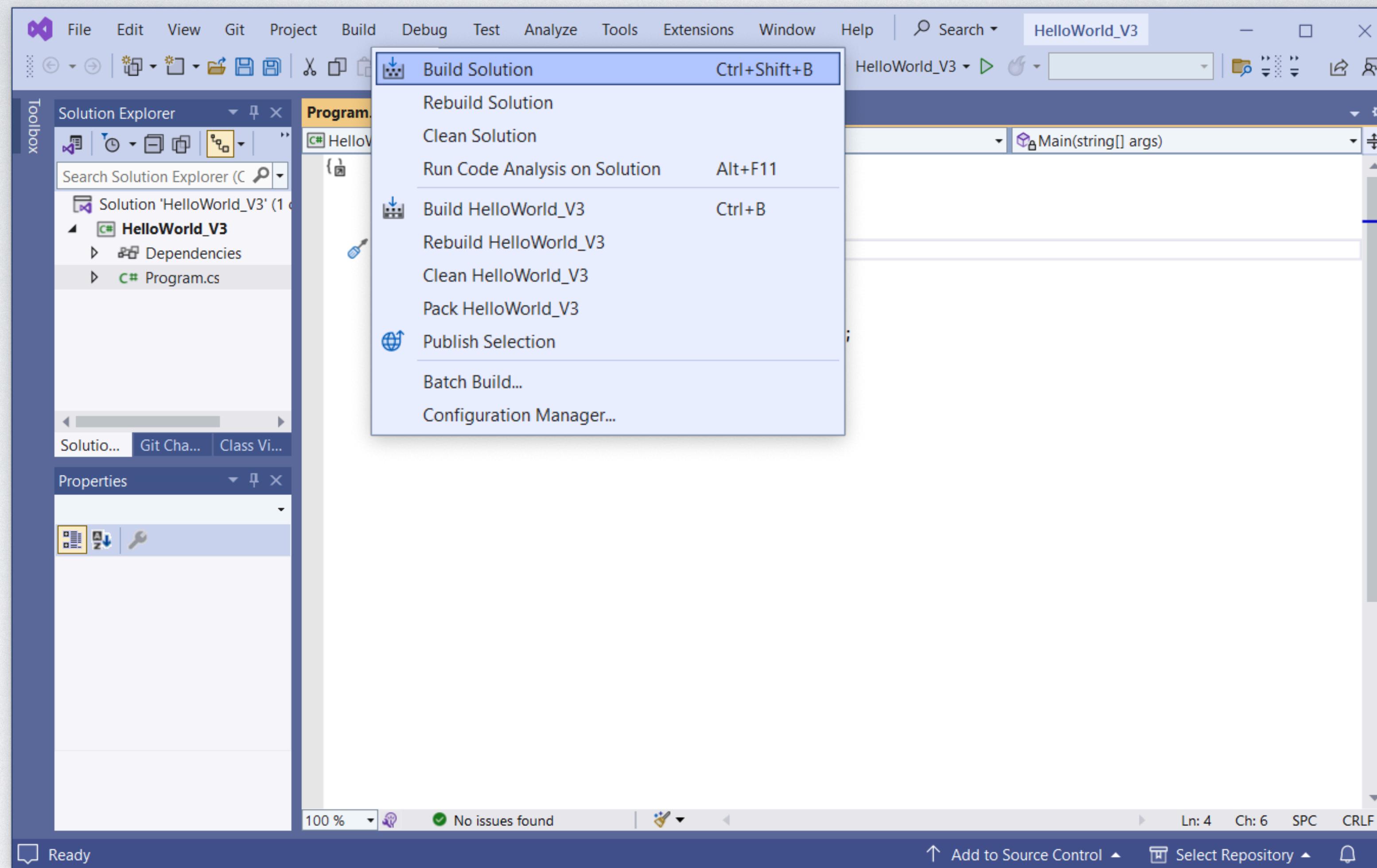
```
namespace HelloWorld_V3
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

- The generated project template is a fully functional C# console application.

HELLO WORLD IN DETAIL

Language Construct	Interpretation
namespace HelloWorld_V3 { ... }	A namespace declares a scope C# that can contain a set of related objects which should be logically grouped. Here, HelloWorld_V3 is the namespace for all code elements of our console application.
internal class Program { ... }	C# is a class-based language. Every program must define at least one class that contains the main entry point. Classes in C# can be defined to restrict access to clients. Here, the class Program is marked internal, which means that members are only accessible with the same assembly (i.e., the same unit of deployment).
static void Main(string[] args) { ... }	The function Main define the main entry point of the program. Every program must have at least a main entry point. The function Main is a <i>class member</i> , as it is marked static . Class members are shared by all objects of the class. Static functions can be called even if no objects of the class exist.
Console.WriteLine("Hello, World!");	Console is a class defined in namespace System (Here, the object Console is implicitly resolved to be a member of namespace System. Hence, there is no using System directive at the start.) The class Console represents the standard input, output, and error streams. The method WriteLine outputs the given string to the console and adds a system-specific newline character at the end.

BUILD PROJECT



- Select Build/Build Solution to compile the code and build the executable.

BUILD RESULT

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the project "HelloWorld_V3" with files "Dependencies" and "Program.cs".
- Code Editor:** Displays the "Program.cs" file content:

```
1  namespace HelloWorld_V3
2  {
3      internal class Program
4      {
5          static void Main(string[] args)
6          {
7              Console.WriteLine("Hello, World!");
8          }
9      }
10 }
11
```

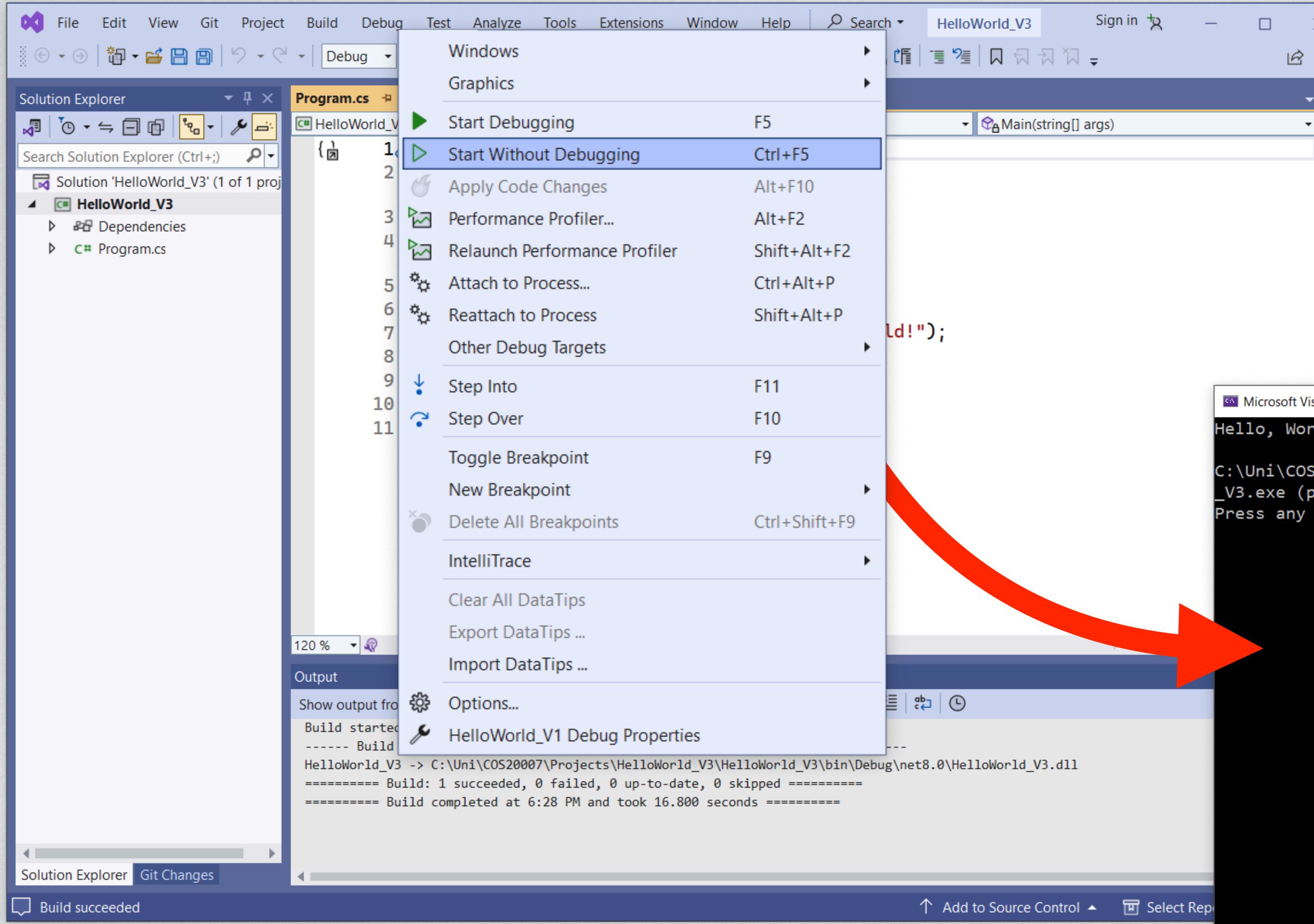
- Output Window:** Shows the build log:

```
Build started at 6:27 PM...
----- Build started: Project: HelloWorld_V3, Configuration: Debug Any CPU -----
HelloWorld_V3 -> C:\Uni\COS2007\Projects\HelloWorld_V3\HelloWorld_V3\bin\Debug\net8.0\HelloWorld_V3.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
===== Build completed at 6:28 PM and took 16.800 seconds ======
```

- Status Bar:** Shows "Build succeeded".

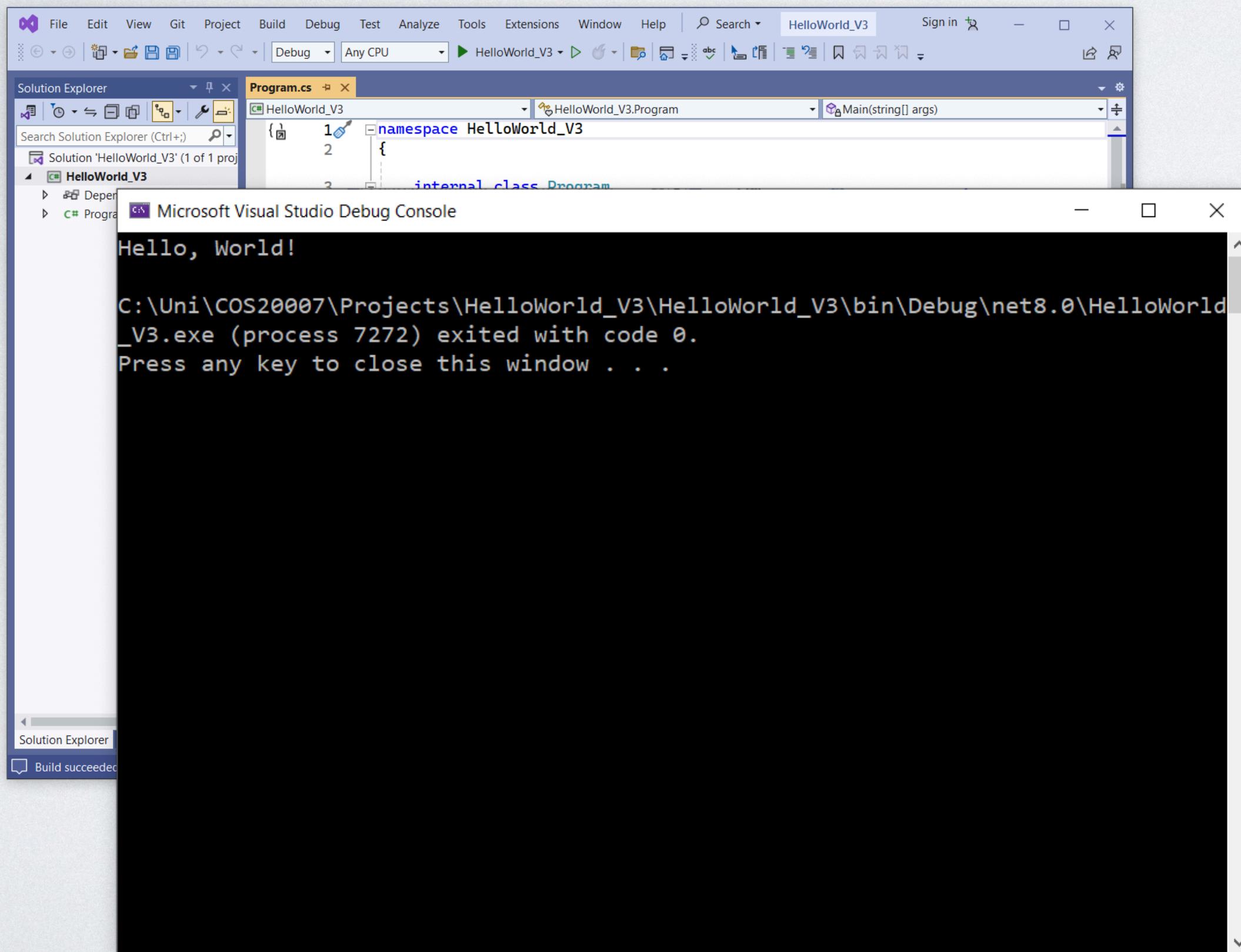
- The build project may take some time, but it should succeed.

TEST EXECUTABLE



- Run project without debugging. Select [Debug/Start Without Debugging](#).
- You can always do this to quickly check if your program produces the expected results.

VISUAL STUDION INSTALLATION COMPLETE



- If you get to this point, your Visual Studio installation is complete and you can start writing programs in C# using the .NET framework.

MSYS2

WHAT IS MSYS2

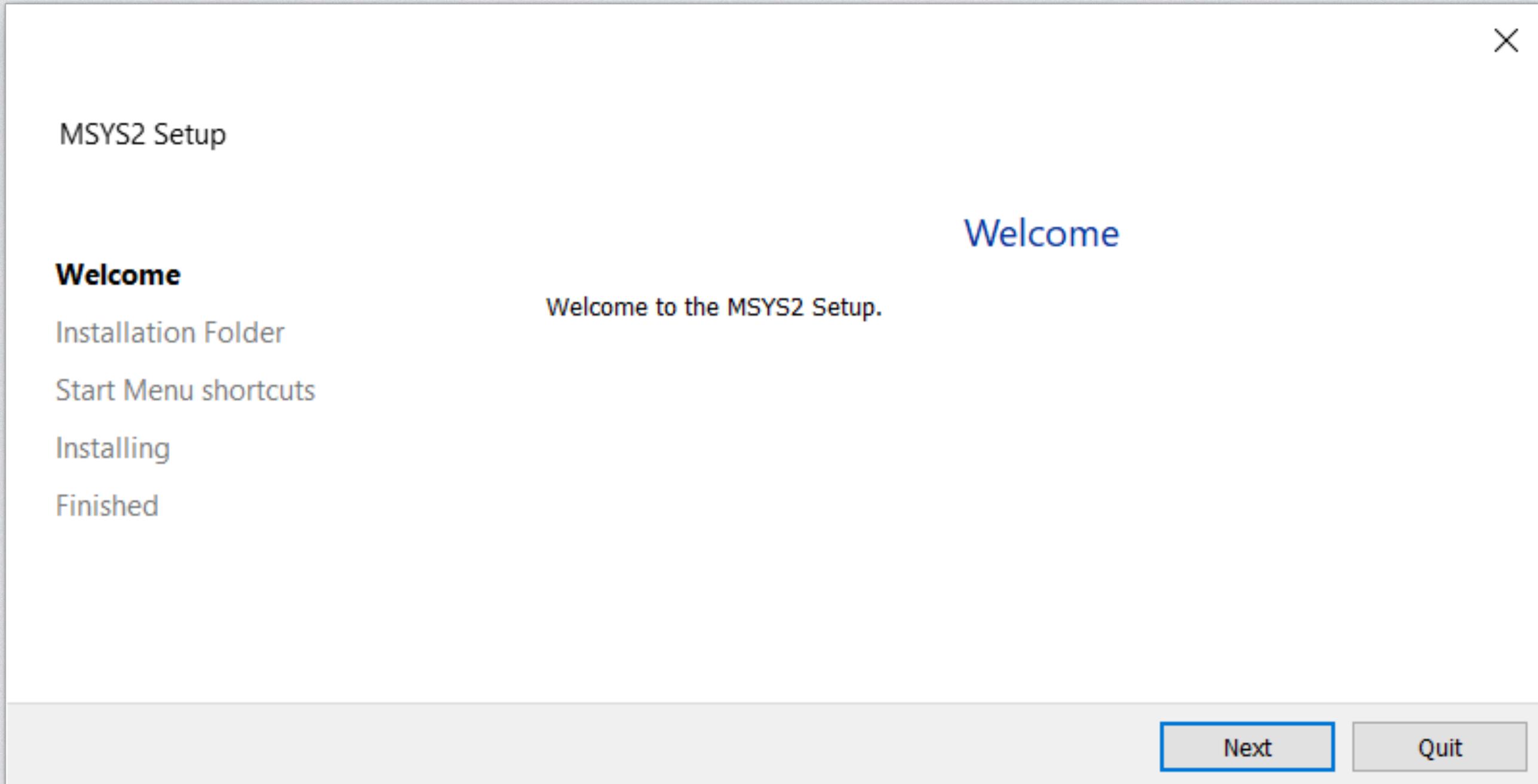
- MSYS2 is a collection of tools and libraries for building, installing, and running Windows applications.
- MSYS2 consists of a command line terminal (actually six different terminal types), bash command shell, git version control, and many more tools.
- MSYS2 terminals provide a Unix-like command line user interface. The four most important commands are `cd` - change directory, `ls` - list directory content, `pwd` - print current working directory, and `mkdir` - create a directory.
- MSYS2 represents Windows drives as paths. For example, `C:\` maps to the path `/c` and `C:\Users\Jane\COS20007` maps to `/c/Users/Jane/COS20007`.
- In the terminal `~` is the home directory for the current user (e.g., `/home/Jane` is Jane's home directory in the MSYS2 environment). **The MSYS2 home directory must not be confused with the user's Windows home directory. They are different.** Always store data and programs in a folder that starts with the path name `/C/Users/Jane`. Replace Jane with your user name.

WWW.MSYS2.ORG

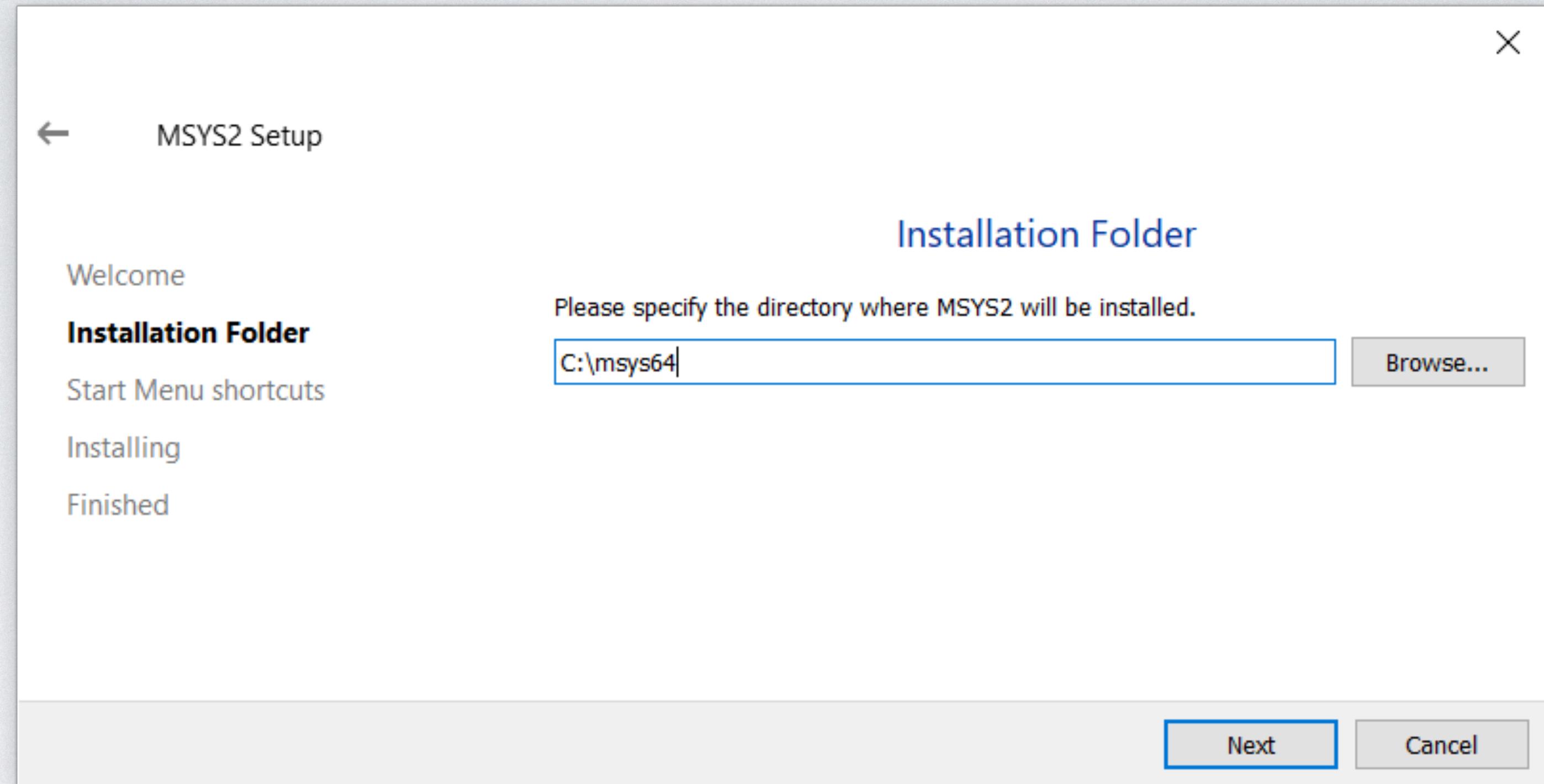
The screenshot shows a web browser window displaying the MSYS2 website at <https://www.msys2.org>. The page title is "MSYS2" and the subtitle is "Software Distribution and Building Platform for Windows". The left sidebar contains links for "MSYS2", "Getting Started", "News", "Package Index", "Documentation", "Development", "Other Topics", "Get Involved", "License", "Privacy", "Support & Contact", and "Code of Conduct". The main content area starts with a brief introduction: "MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software." It then describes the toolset, mentioning mintty, bash, git, subversion, tar, awk, autotools, Cygwin, Pacman, and various native builds for GCC, mingw-w64, CPython, CMake, Meson, OpenSSL, FFmpeg, Rust, Ruby, etc. A section on package management highlights the Pacman system. Below this, there's a "What is MSYS2?" comparison and a "Who Is Using MSYS2?" section. The right sidebar includes a "Table of contents" with links to "Installation", "Sponsors", "Authors and Contributors", and "Donations". At the bottom, there's an "Installation" section with a link to the installer file "msys2-x86_64-20240113.exe" and a note about verifying checksums and signatures.

- Download the installer for 64 bit Windows.

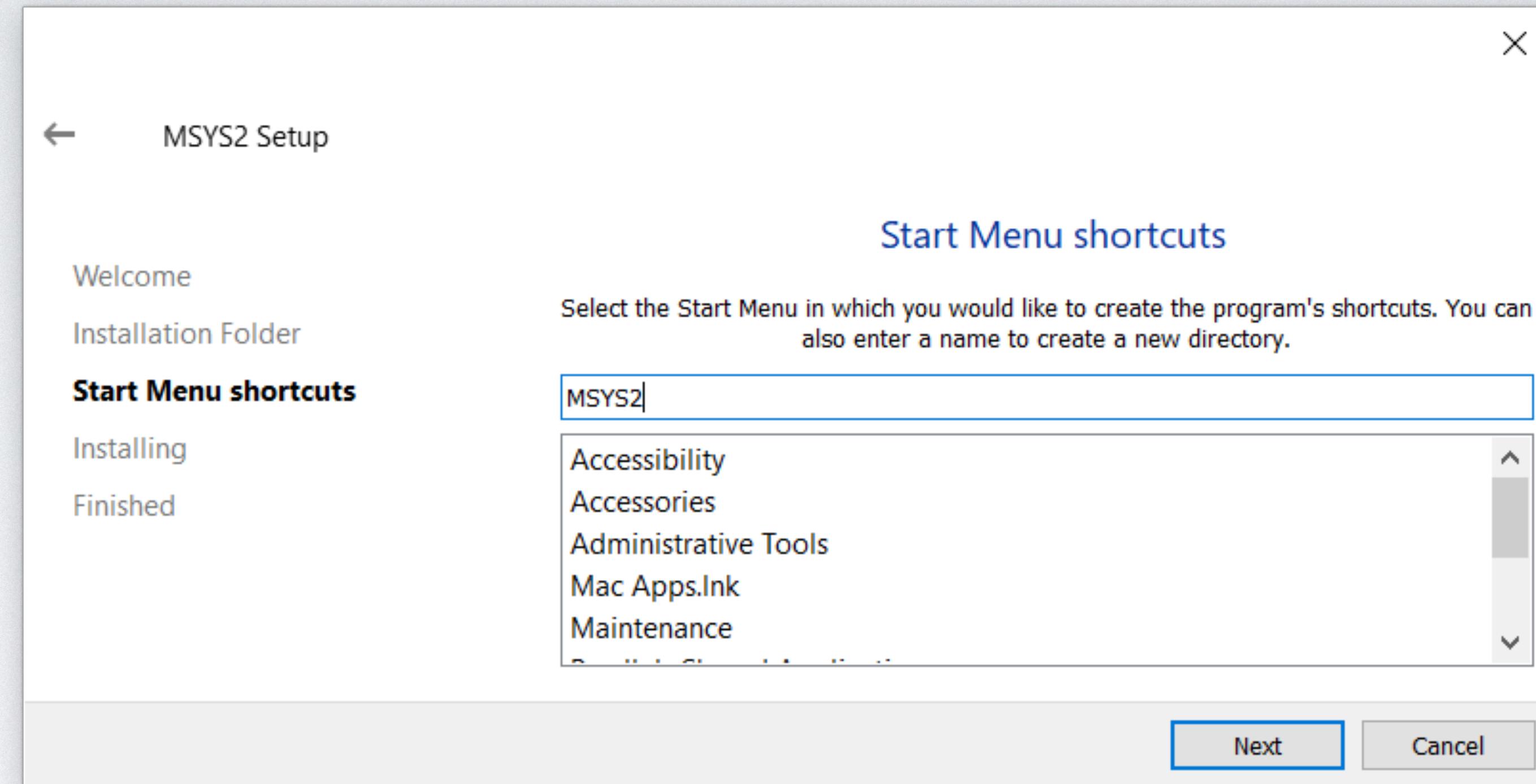
RUNNING THE MSYS2 INSTALLER



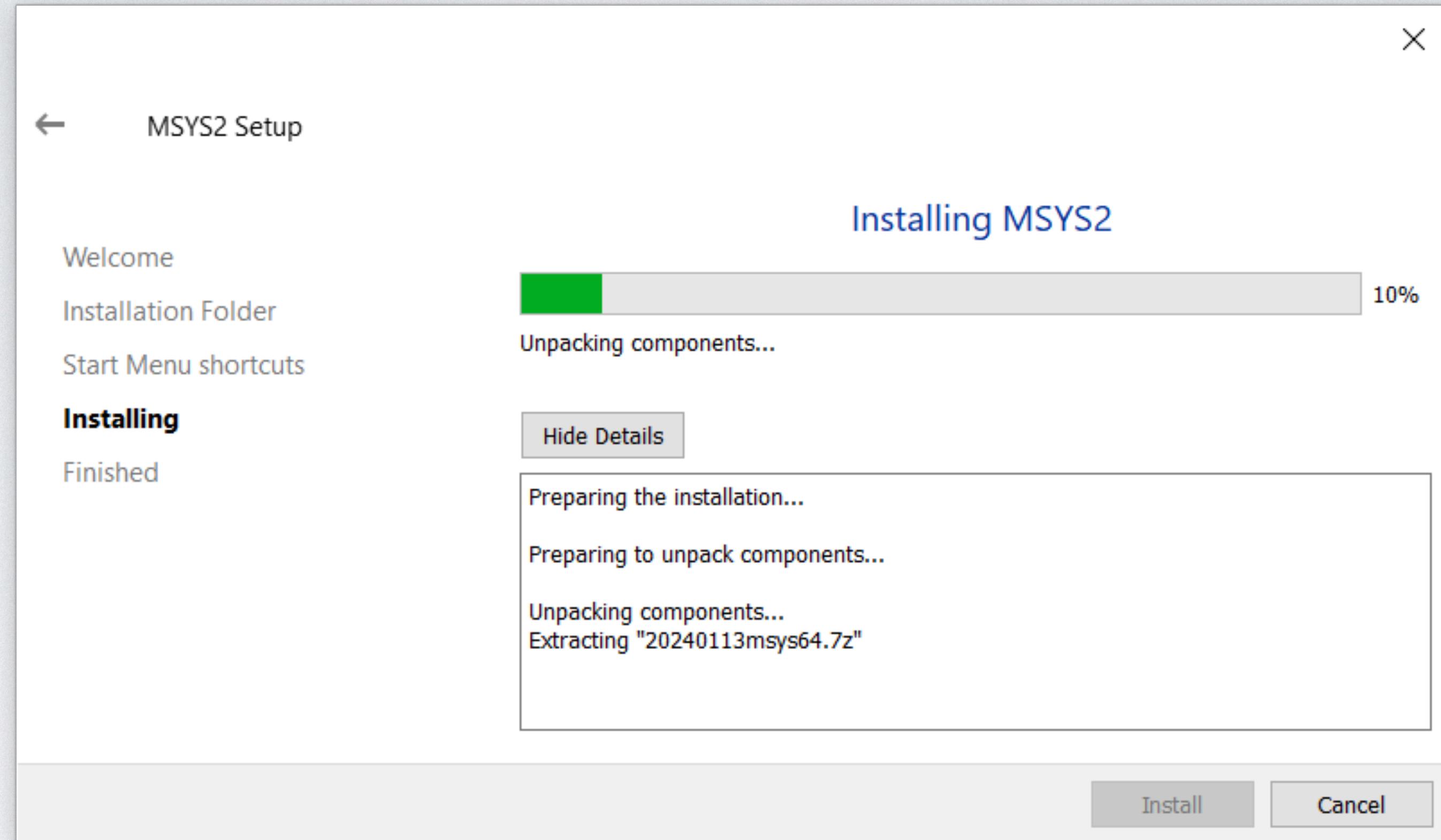
MSYS2 INSTALL FOLDER



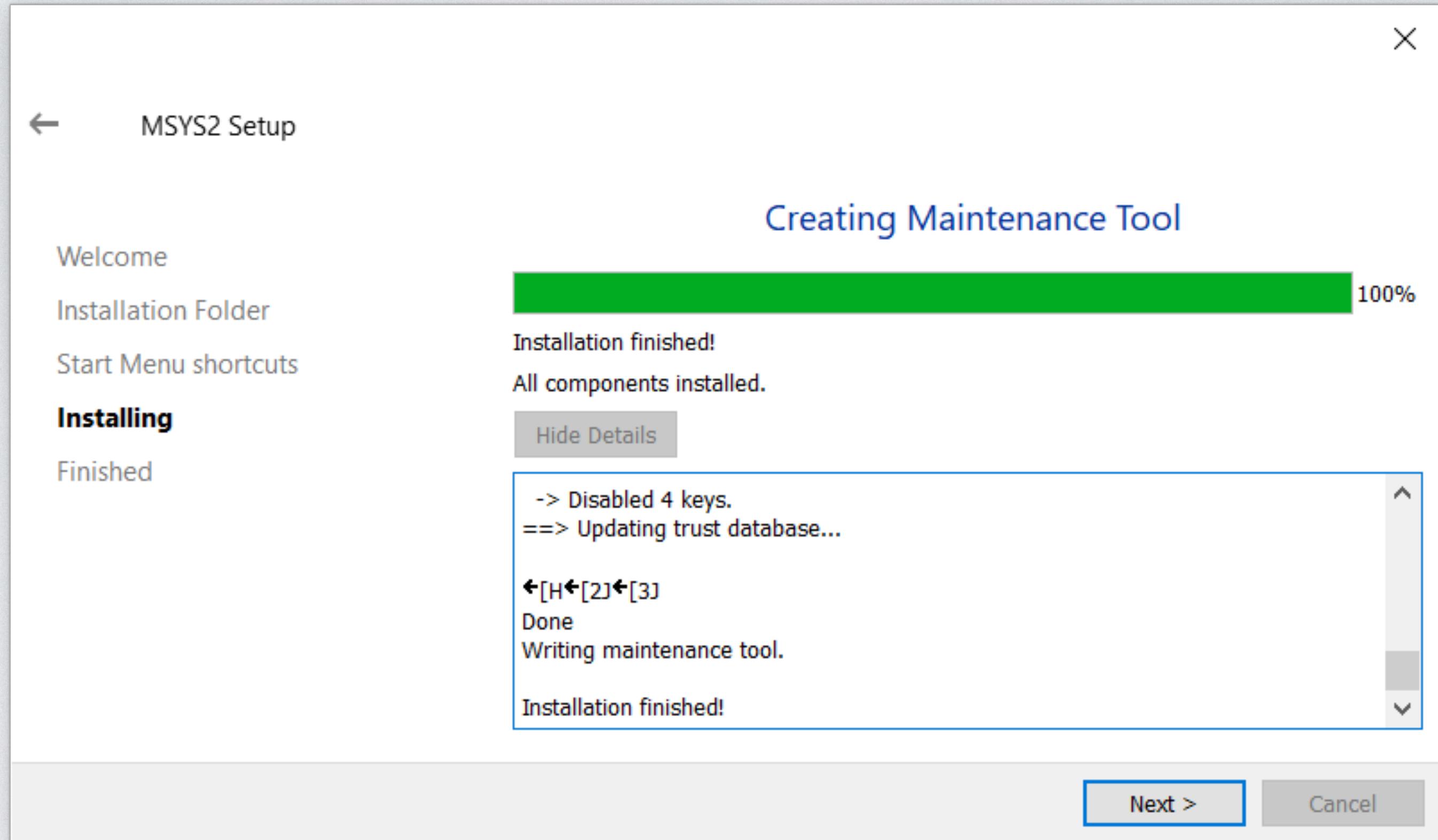
MSYS2 MENU SHORTCUTS



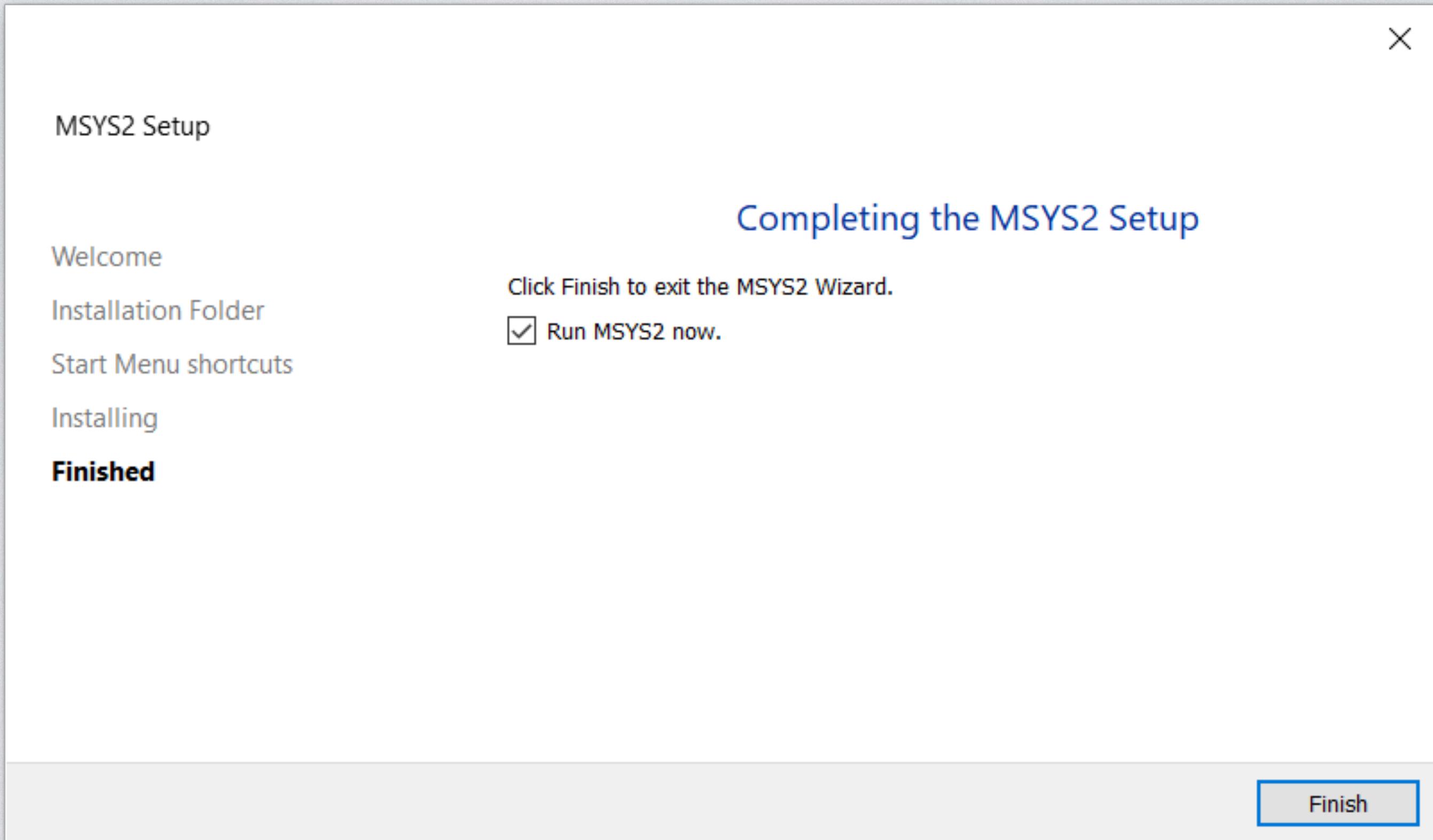
MSYS2 INSTALLATION STARTED



MSYS2 INSTALLATION COMPLETE



MSYS2 RUN AFTER INSTALLATION

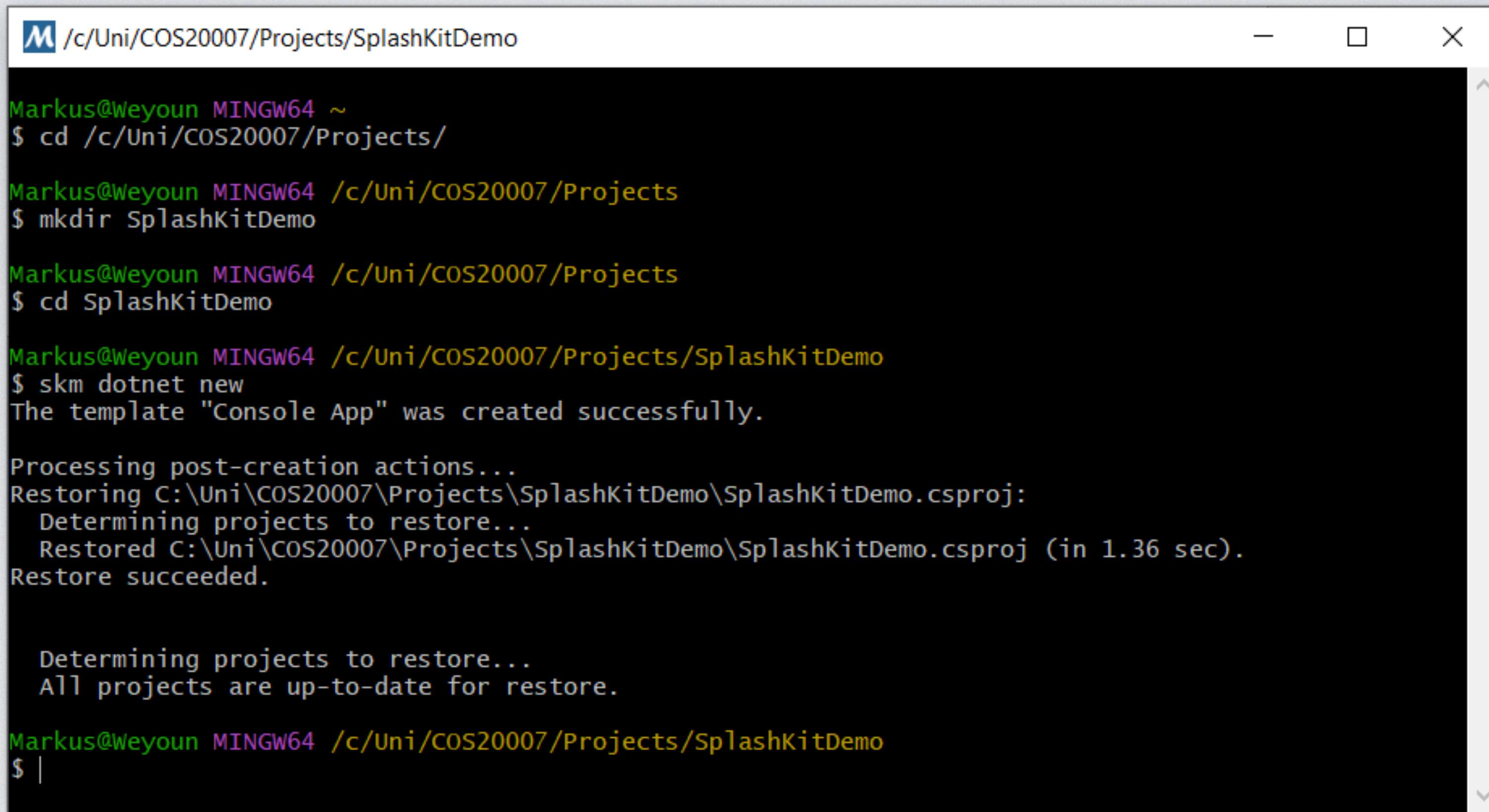


MSYS2 UCRT64 TERMINAL



- UCRT64 is the default terminal. But for development in COS20007, we need the MINGW64 terminal (SplashKit only works in MINGW64).

MSYS2 MINGW64 TERMINAL



The screenshot shows a terminal window titled 'Markus@Weyoun MINGW64 ~' with the path '/c/Uni/COS20007/Projects/SplashKitDemo'. The terminal displays the following command-line session:

```
Markus@Weyoun MINGW64 ~
$ cd /c/Uni/cos20007/Projects/
Markus@Weyoun MINGW64 /c/Uni/cos20007/Projects
$ mkdir SplashKitDemo
Markus@Weyoun MINGW64 /c/Uni/cos20007/Projects
$ cd SplashKitDemo
Markus@Weyoun MINGW64 /c/Uni/cos20007/Projects/SplashKitDemo
$ skm dotnet new
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Uni\cos20007\Projects\SplashKitDemo\SplashKitDemo.csproj:
  Determining projects to restore...
  Restored C:\Uni\cos20007\Projects\SplashKitDemo\SplashKitDemo.csproj (in 1.36 sec).
Restore succeeded.

Determining projects to restore...
All projects are up-to-date for restore.

Markus@Weyoun MINGW64 /c/Uni/cos20007/Projects/SplashKitDemo
$ |
```

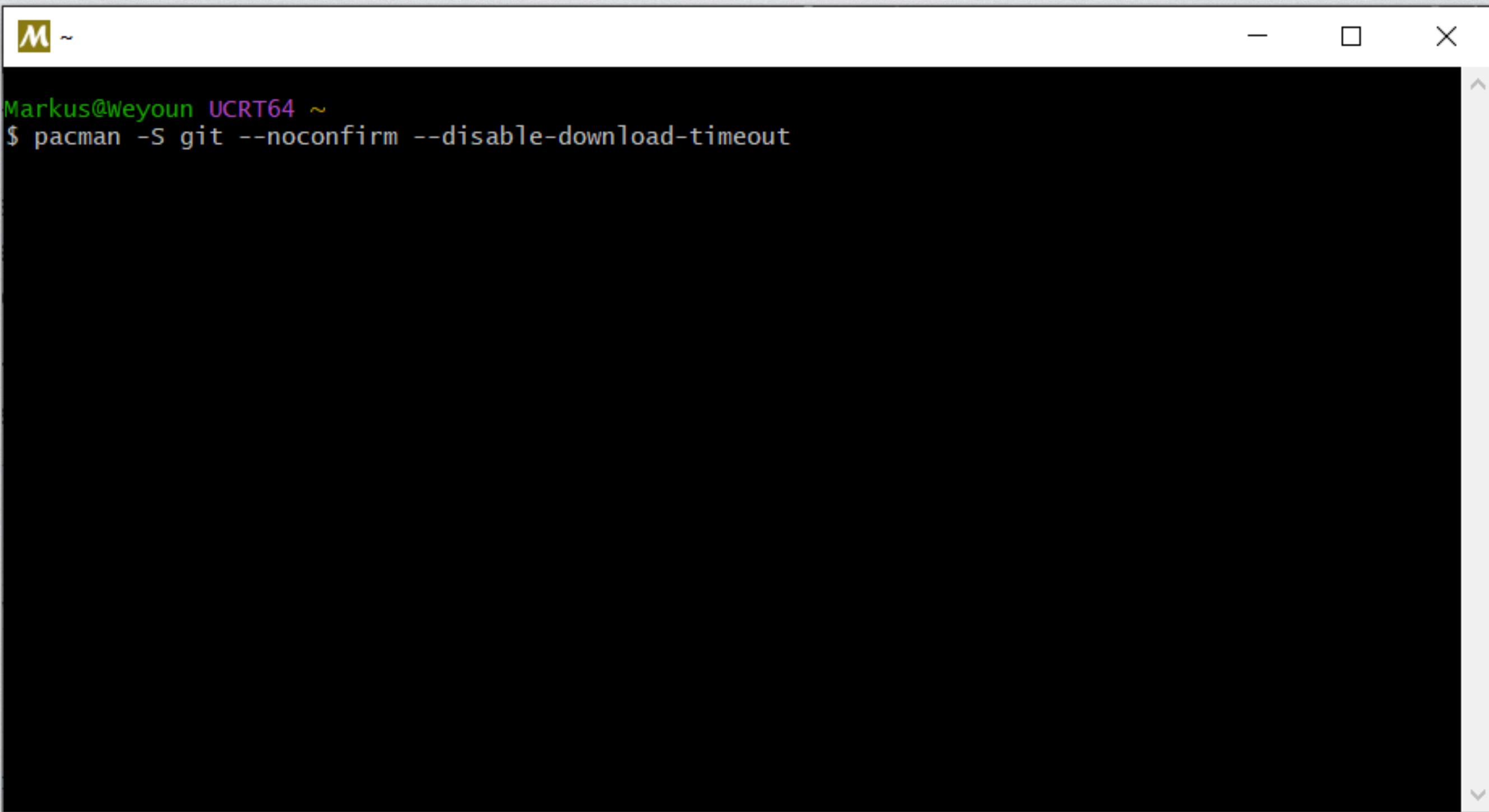
- SplashKit project creation only works in MINGW64.
- To create a SplashKit project, setup a project directory (e.g., SplashKitDemo), move into this directory, and run the command `skm dotnet new`.

GIT

WHAT IS GIT

- Git is an open source distributed version control system.
- In COS20007, we use it initially to set up a local installation of SplashKit.
- Git is easy to learn and most modern development environments (e.g., Visual Studio) integrate version control via Git.

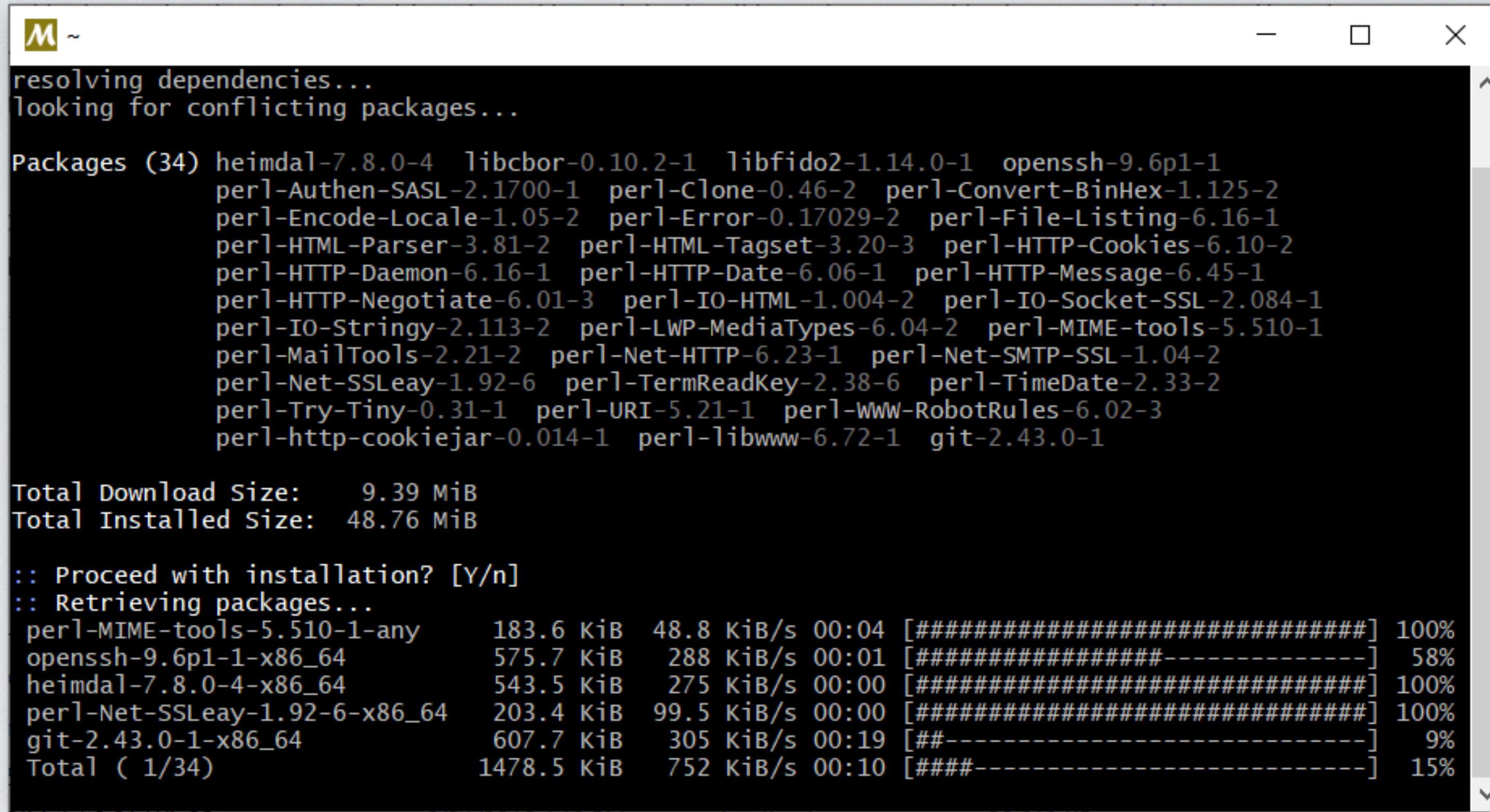
INSTALL GIT VIA MSYS2



A screenshot of a MSYS2 terminal window. The window has a dark background and light-colored text. At the top, it shows the path 'Markus@Weyoun UCRT64 ~'. Below that, a command is being typed: '\$ pacman -S git --noconfirm --disable-download-timeout'. The rest of the window is blank, indicating the command is still processing.

- Install git via MSYS2 and pacman, a library-based package manager.
- Open a MSYS2 terminal (UCRT64 or MINGW64) and type: **paceman -S git —noconfirm —disable-download-timeout**
- After the installation, [git is available in MSYS2 terminals](#), but not in Windows cmd.

RUNNING PACMAN



The screenshot shows a terminal window with the following output:

```
M ~
resolving dependencies...
looking for conflicting packages...

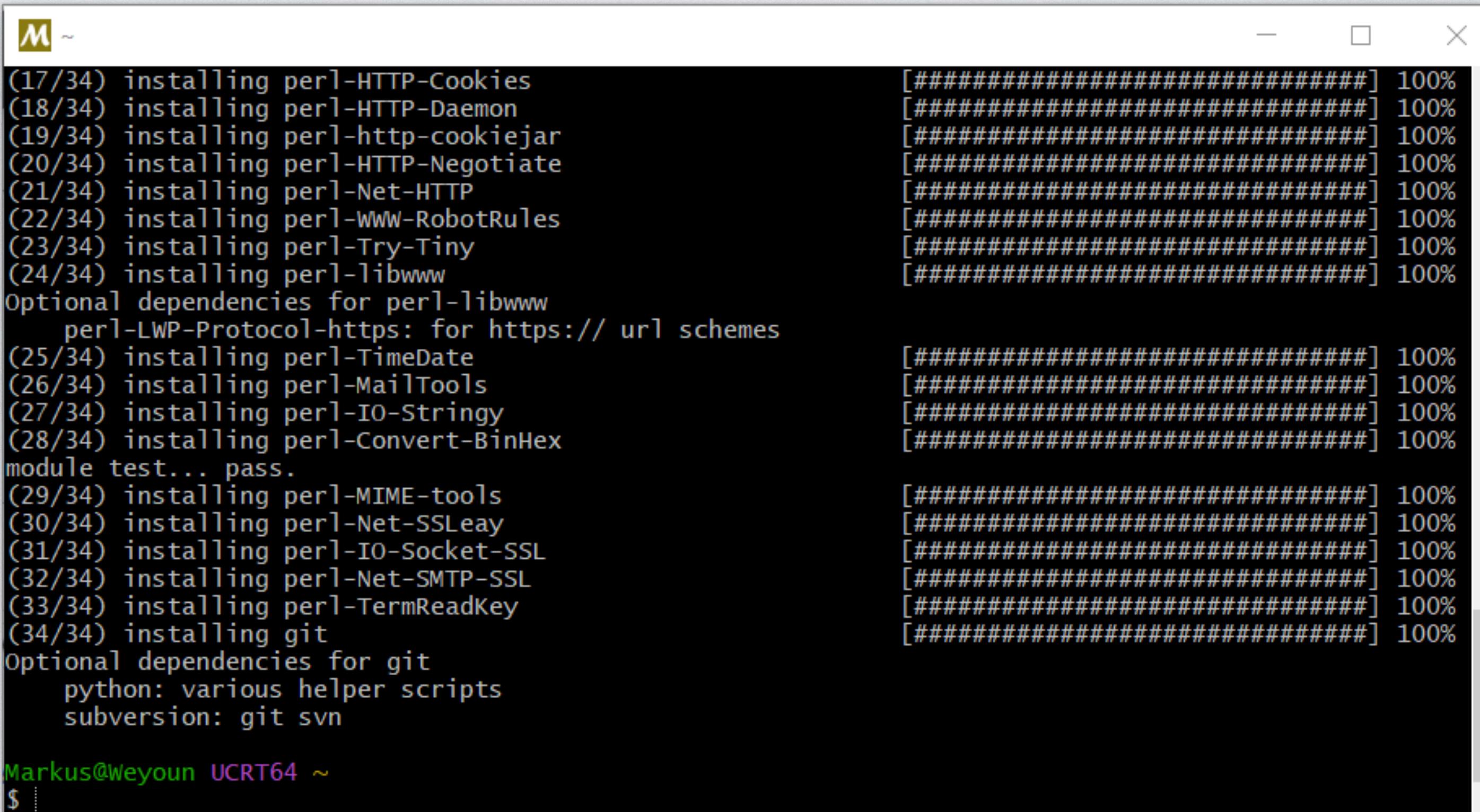
Packages (34) heimdal-7.8.0-4  libcbor-0.10.2-1  libfido2-1.14.0-1  openssh-9.6p1-1
                  perl/Authen-SASL-2.1700-1  perl/Clone-0.46-2  perl/Convert-BinHex-1.125-2
                  perl/Encode-Locale-1.05-2  perl/Error-0.17029-2  perl/File-Listing-6.16-1
                  perl/HTML-Parser-3.81-2  perl/HTML-Tagset-3.20-3  perl/HTTP-Cookies-6.10-2
                  perl/HTTP-Daemon-6.16-1  perl/HTTP-Date-6.06-1  perl/HTTP-Message-6.45-1
                  perl/HTTP-Negotiate-6.01-3  perl/IO-HTML-1.004-2  perl/IO-Socket-SSL-2.084-1
                  perl/IO-Stringy-2.113-2  perl/LWP-MediaTypes-6.04-2  perl/MIME-tools-5.510-1
                  perl-MailTools-2.21-2  perl/Net-HTTP-6.23-1  perl/Net-SMTP-SSL-1.04-2
                  perl/Net-SSLeay-1.92-6  perl/TermReadKey-2.38-6  perl/TimeDate-2.33-2
                  perl-Try-Tiny-0.31-1  perl/URI-5.21-1  perl/WWW-RobotRules-6.02-3
                  perl/http-cookiejar-0.014-1  perl-libwww-6.72-1  git-2.43.0-1

Total Download Size:  9.39 MiB
Total Installed Size: 48.76 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
perl-MIME-tools-5.510-1-any      183.6 KiB  48.8 KiB/s 00:04 [#####
openSSH-9.6p1-1-x86_64          575.7 KiB  288 KiB/s 00:01 [#####
heimdal-7.8.0-4-x86_64          543.5 KiB  275 KiB/s 00:00 [#####
perl/Net-SSLeay-1.92-6-x86_64   203.4 KiB  99.5 KiB/s 00:00 [#####
git-2.43.0-1-x86_64             607.7 KiB  305 KiB/s 00:19 [#####
Total ( 1/34)                   1478.5 KiB  752 KiB/s 00:10 [#####-----]
```

- pacman automatically resolves all package dependencies.

GIT INSTALLATION COMPLETE



The screenshot shows a terminal window titled 'M ~' with a progress bar indicating the completion of a software installation. The terminal output is as follows:

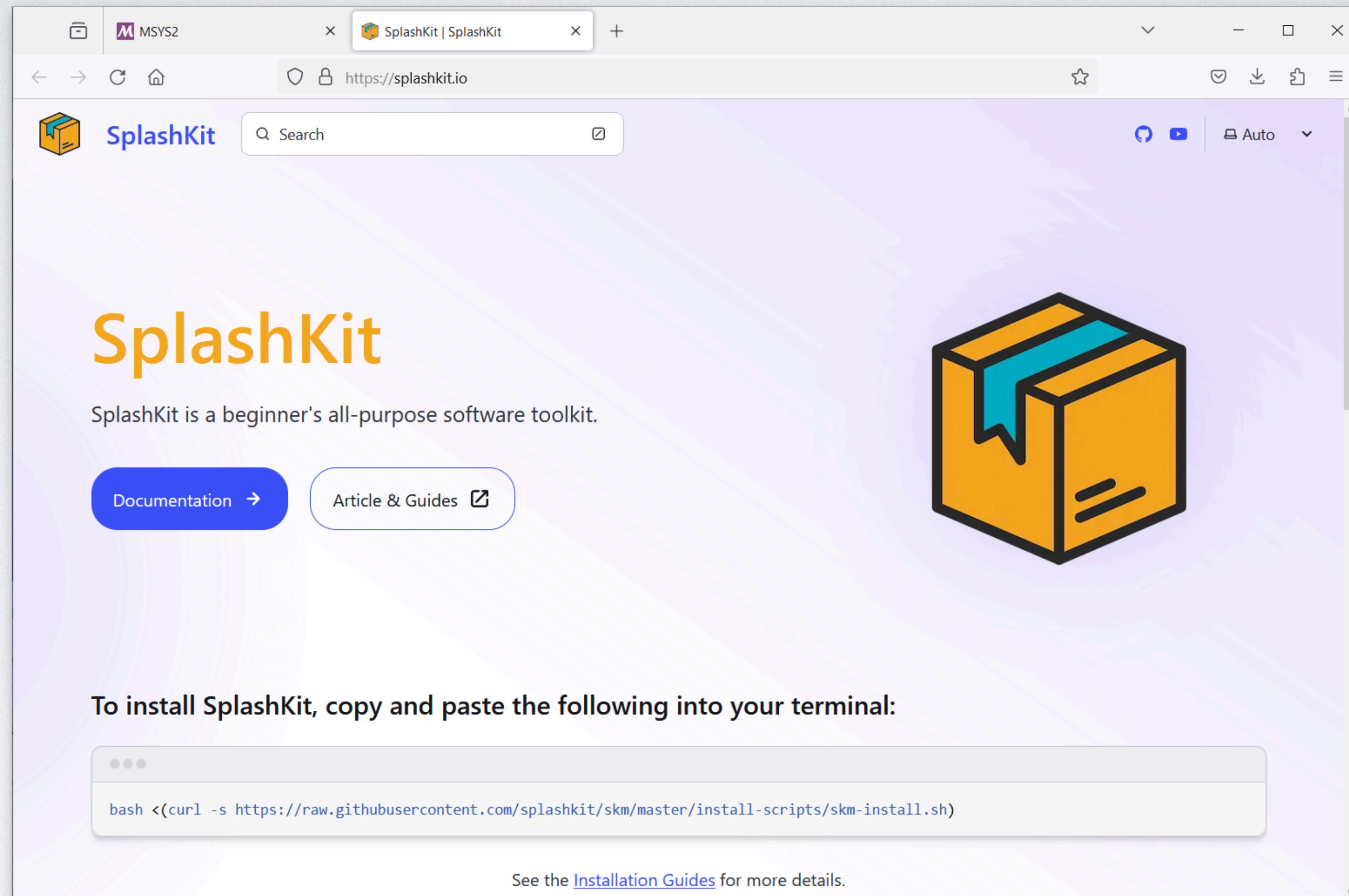
```
(17/34) installing perl-HTTP-Cookies [#####
(18/34) installing perl-HTTP-Daemon [#####
(19/34) installing perl-http-cookiejar [#####
(20/34) installing perl-HTTP-Negotiate [#####
(21/34) installing perl-Net-HTTP [#####
(22/34) installing perl-www-RobotRules [#####
(23/34) installing perl-Try-Tiny [#####
(24/34) installing perl-libwww [#####
Optional dependencies for perl-libwww
    perl-LWP-Protocol-https: for https:// url schemes [#####
(25/34) installing perl-TimeDate [#####
(26/34) installing perl-MailTools [#####
(27/34) installing perl-IO-Stringy [#####
(28/34) installing perl-Convert-BinHex [#####
module test... pass.
(29/34) installing perl-MIME-tools [#####
(30/34) installing perl-Net-SSLeay [#####
(31/34) installing perl-IO-Socket-SSL [#####
(32/34) installing perl-Net-SMTP-SSL [#####
(33/34) installing perl-TermReadKey [#####
(34/34) installing git [#####
Optional dependencies for git
    python: various helper scripts [#####
    subversion: git svn [#####

Markus@Weyoun UCRT64 ~
```

- No errors have occurred while installing git.

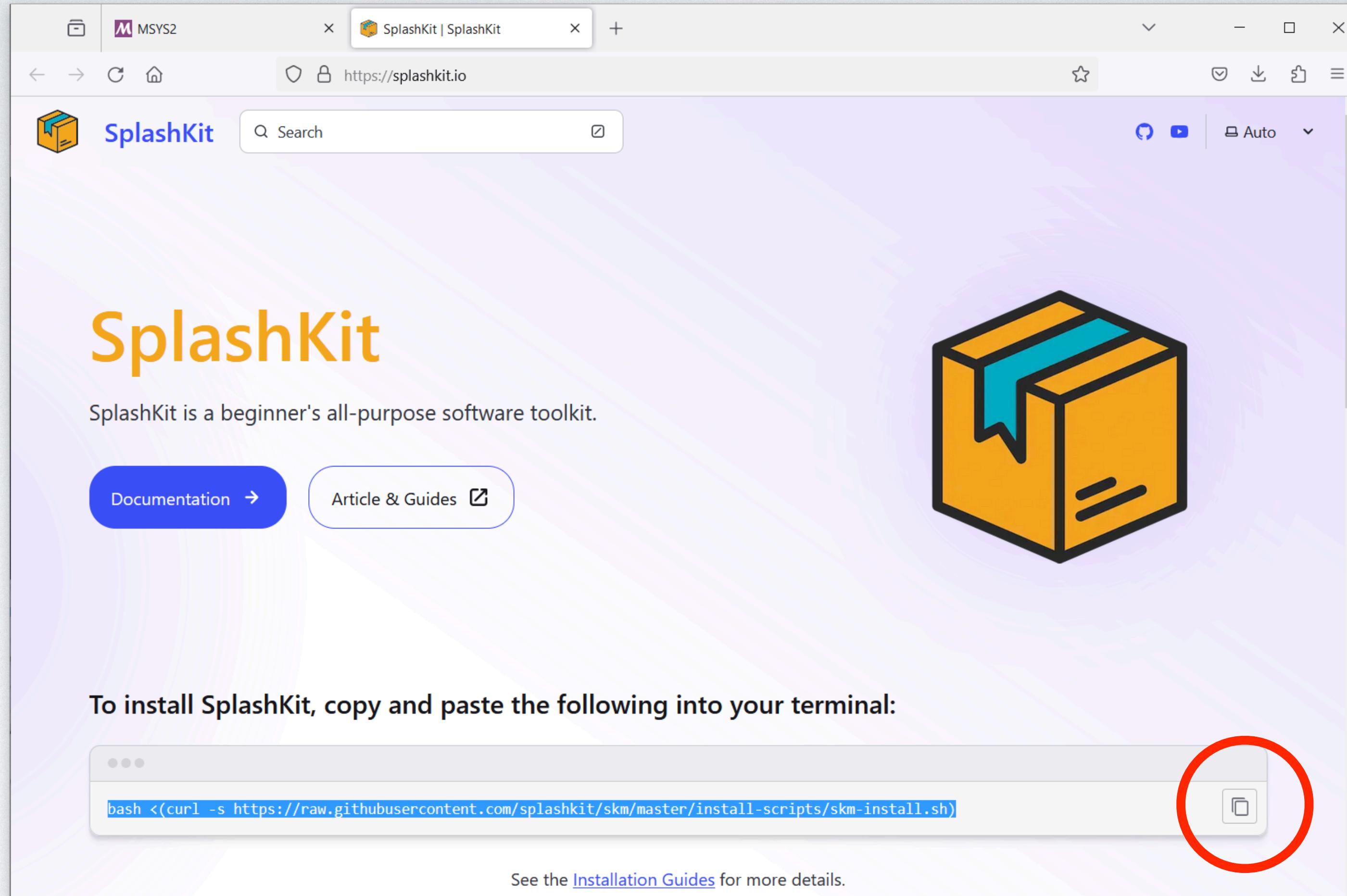
SPLASH KIT

SPLASH KIT HOME



- SplashKit can be found at splashkit.io.

SPLASH KIT INSTALLATION COMMAND



- Copy the SplashKit installation command line by clicking the copy button at the right of the text field.

SPLASH KIT INSTALLATION IN MSYS2

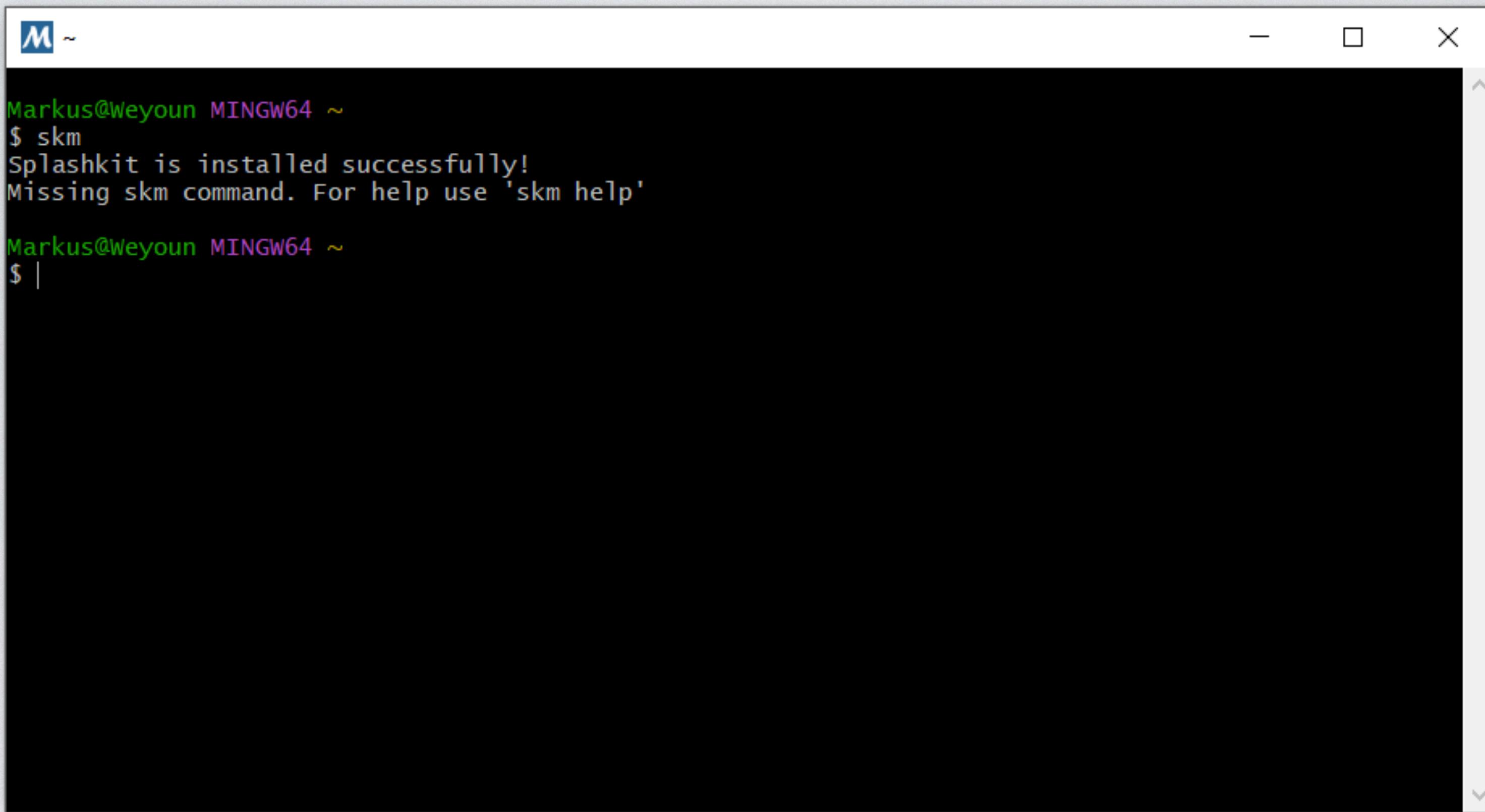
The image consists of two vertically stacked screenshots of a terminal window. The top screenshot shows a context menu open over the terminal's content area. The menu has several options: Open, Copy (Ctrl+Ins), Paste (Shift+Ins) which is highlighted with a blue selection bar, Select All, Save as Image, Search (Alt+F3), Reset (Alt+F8), Default Size (Alt+F10), Scrollbar, Full Screen (Alt+F11), Flip Screen (Alt+F12), Status Line, and Options... The bottom screenshot shows the terminal window after the command has been run. It displays the output of the curl command, which includes instructions to restart the terminal, clone the repository, and receive objects. The terminal window has a dark background with light-colored text.

```
Markus@Weyoun UCRT64 ~
$ bash <(curl -s https://raw.githubusercontent.com/splashkit/skm/master/install-scripts/skm-install.sh)

SUCCESS: Specified value was saved.
Updated! Please restart your terminal and rerun this script to install SplashKit.
Cloning into '/home/Markus/.splashkit'...
remote: Enumerating objects: 407, done.
remote: Counting objects: 100% (407/407), done.
remote: Compressing objects: 100% (364/364), done.
Receiving objects: 32% (131/407)
```

- Open a MSYS2 terminal (e.g., UCRT64 or MINGW64).
- Paste the copied installation command line into the terminal (Shift+Ins or right click mouse and select paste).
- Next, press Enter.
- Wait for the installation to finish.
- Close the terminal and start a new one. This time use MINGW64 as we need this terminal to create SplashKit projects.

TEST SKM



A screenshot of a terminal window titled 'M ~'. The window shows the following text:

```
Markus@Weyoun MINGW64 ~
$ skm
Splashkit is installed successfully!
Missing skm command. For help use 'skm help'

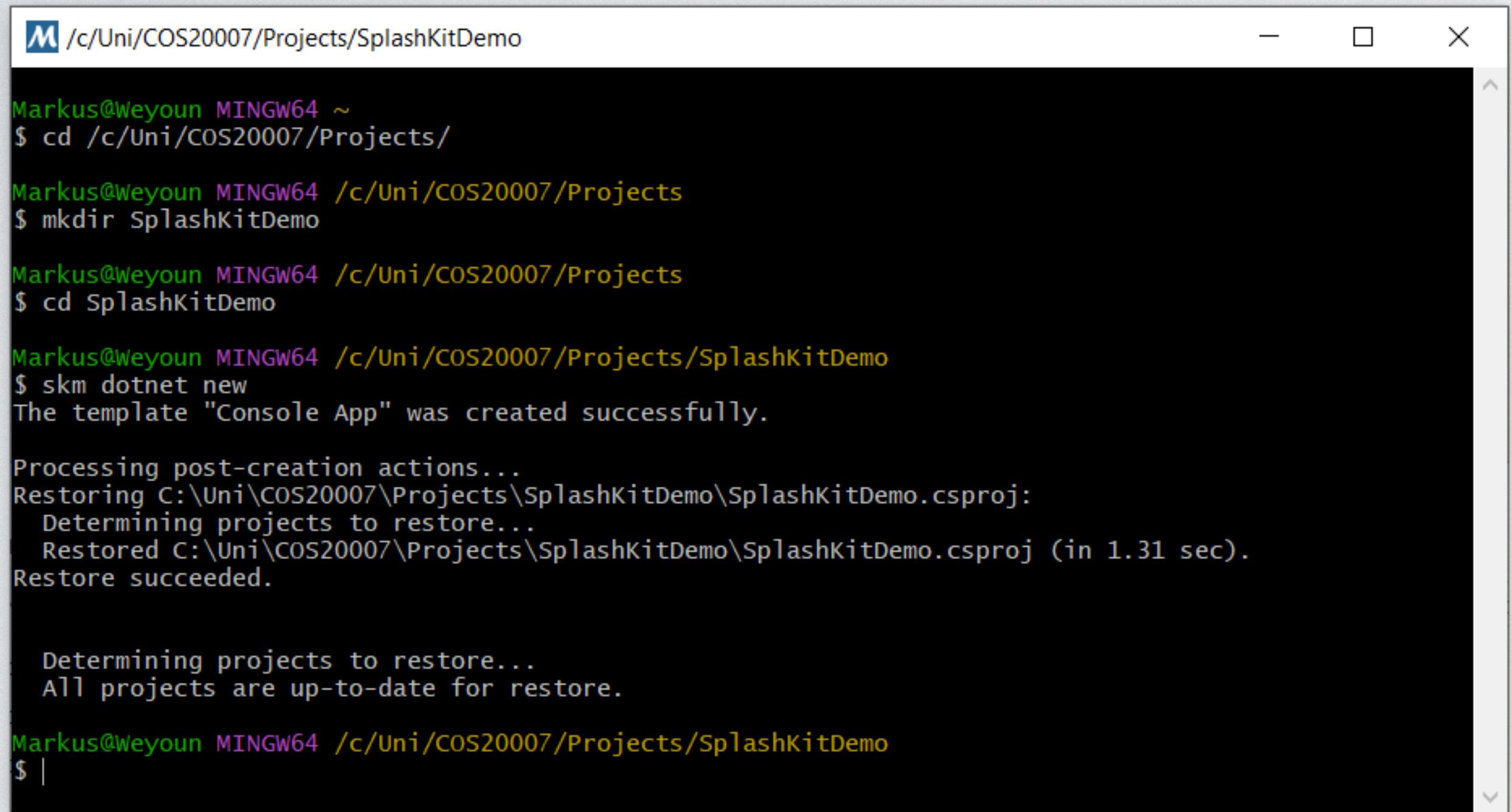
Markus@Weyoun MINGW64 ~
$ |
```

- Type **skm** in the terminal to verify SplashKit is working.

SPLASH KIT DEMO

CREATE A SPLASH KIT PROJECT

- To create a SplashKit project, take the following steps:



The screenshot shows a terminal window titled 'c/Uni/COS20007/Projects/SplashKitDemo'. The terminal output is as follows:

```
Markus@Weyoun MINGW64 ~
$ cd /c/Uni/COS20007/Projects/
Markus@Weyoun MINGW64 /c/Uni/COS20007/Projects
$ mkdir SplashKitDemo
Markus@Weyoun MINGW64 /c/Uni/COS20007/Projects
$ cd SplashKitDemo
Markus@Weyoun MINGW64 /c/Uni/COS20007/Projects/SplashKitDemo
$ skm dotnet new
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Uni\COS20007\Projects\SplashKitDemo\SplashKitDemo.csproj:
  Determining projects to restore...
  Restored C:\Uni\COS20007\Projects\SplashKitDemo\SplashKitDemo.csproj (in 1.31 sec).
Restore succeeded.

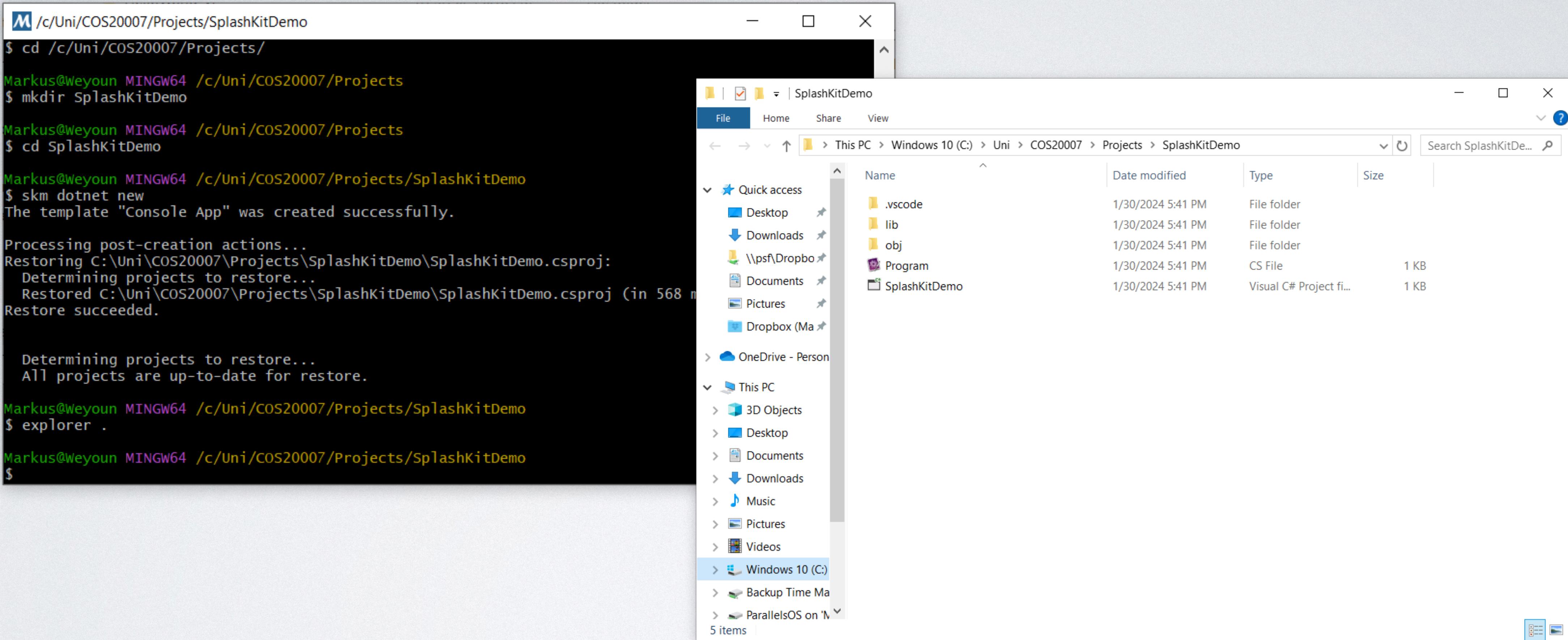
Determining projects to restore...
All projects are up-to-date for restore.

Markus@Weyoun MINGW64 /c/Uni/COS20007/Projects/SplashKitDemo
$ |
```

1. Open a MINGW64 terminal.
2. Change to the location of your COS20007 projects.
3. Create a new directory for your project (e.g., SplashKitDemo).
4. Change into the newly created project directory.
5. Run the command **skm dotnet new**.

The last step sets up a new SplashKit project.

OPEN THE PROJECT FOLDER



```
M /c/Uni/COS2007/Projects/SplashKitDemo
$ cd /c/Uni/COS2007/Projects/
Markus@Weyoun MINGW64 /c/Uni/COS2007/Projects
$ mkdir SplashKitDemo
Markus@Weyoun MINGW64 /c/Uni/COS2007/Projects
$ cd SplashKitDemo
Markus@Weyoun MINGW64 /c/Uni/COS2007/Projects/SplashKitDemo
$ skm dotnet new
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Uni\COS2007\Projects\SplashKitDemo\SplashKitDemo.csproj:
  Determining projects to restore...
    Restored C:\Uni\COS2007\Projects\SplashKitDemo\SplashKitDemo.csproj (in 568 ms)
Restore succeeded.

  Determining projects to restore...
  All projects are up-to-date for restore.

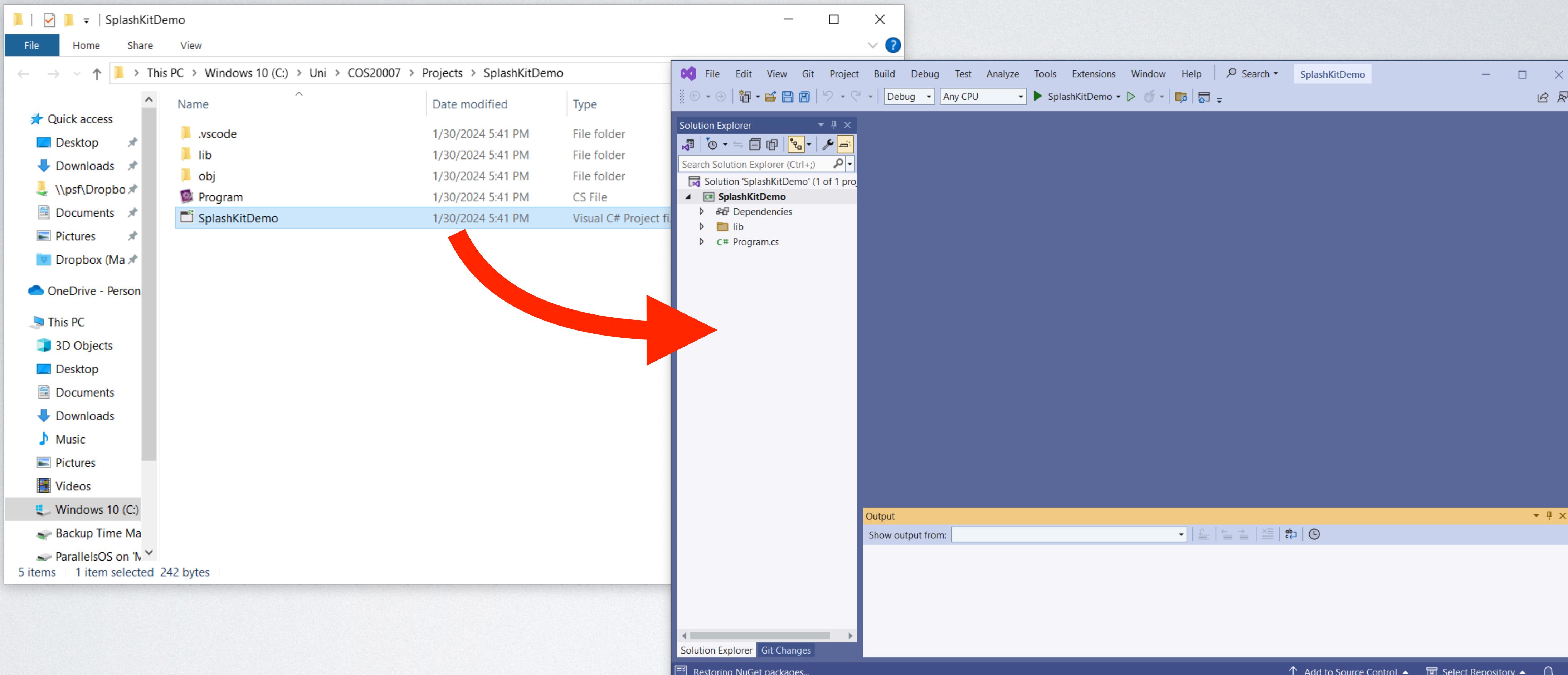
Markus@Weyoun MINGW64 /c/Uni/COS2007/Projects/SplashKitDemo
$ explorer .
Markus@Weyoun MINGW64 /c/Uni/COS2007/Projects/SplashKitDemo
$
```

The file explorer window shows the following file structure:

Name	Date modified	Type	Size
.vscode	1/30/2024 5:41 PM	File folder	
lib	1/30/2024 5:41 PM	File folder	
obj	1/30/2024 5:41 PM	File folder	
Program	1/30/2024 5:41 PM	CS File	1 KB
SplashKitDemo	1/30/2024 5:41 PM	Visual C# Project fi...	1 KB

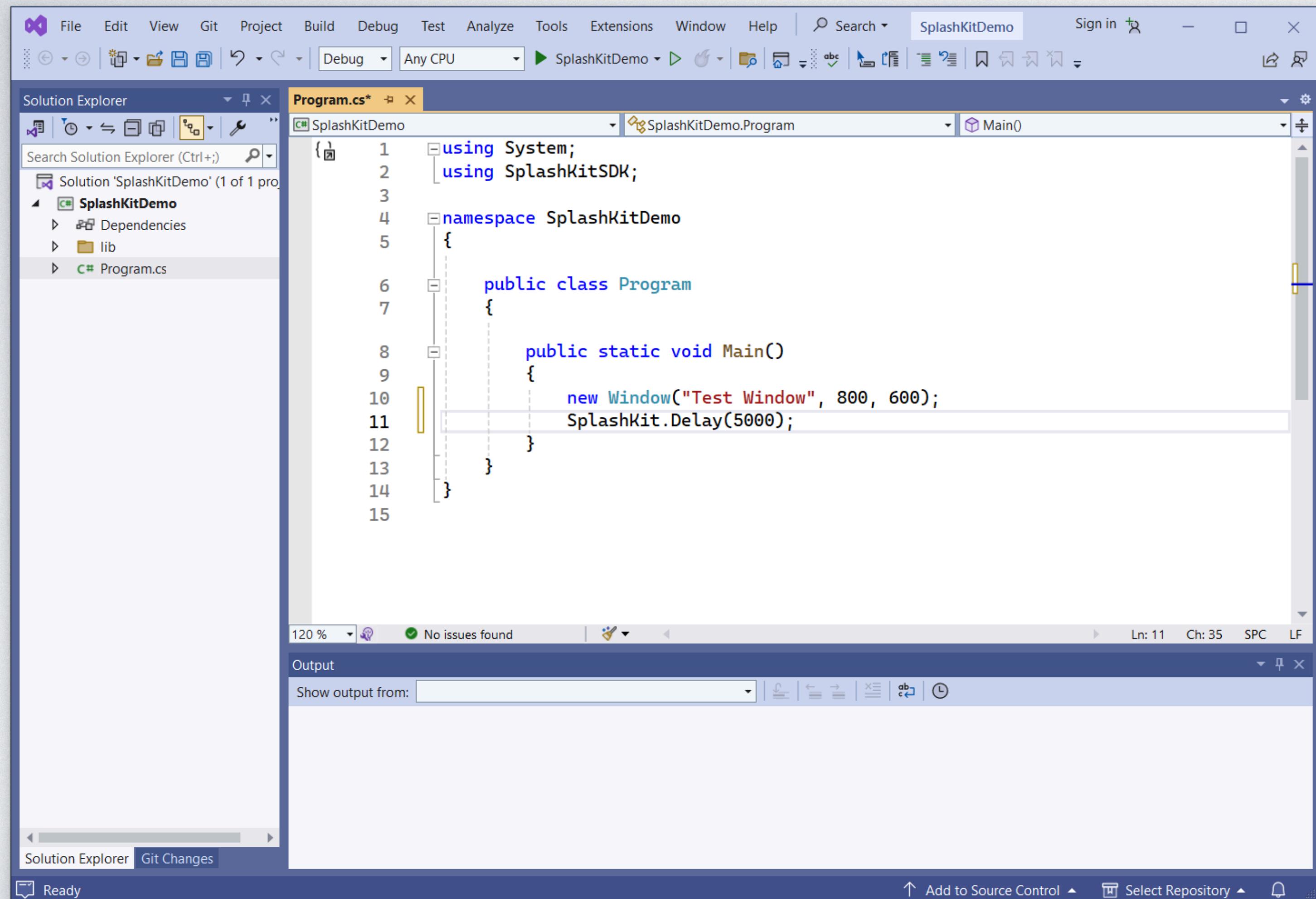
- Type **explorer .** to open the folder containing the newly-created project.

OPEN PROJECT IN VISUAL STUDIO



- Select the project file `SplashKitDemo.csproj` and double-click it to open the project in Visual Studio.

COMPLETE THE PROGRAM



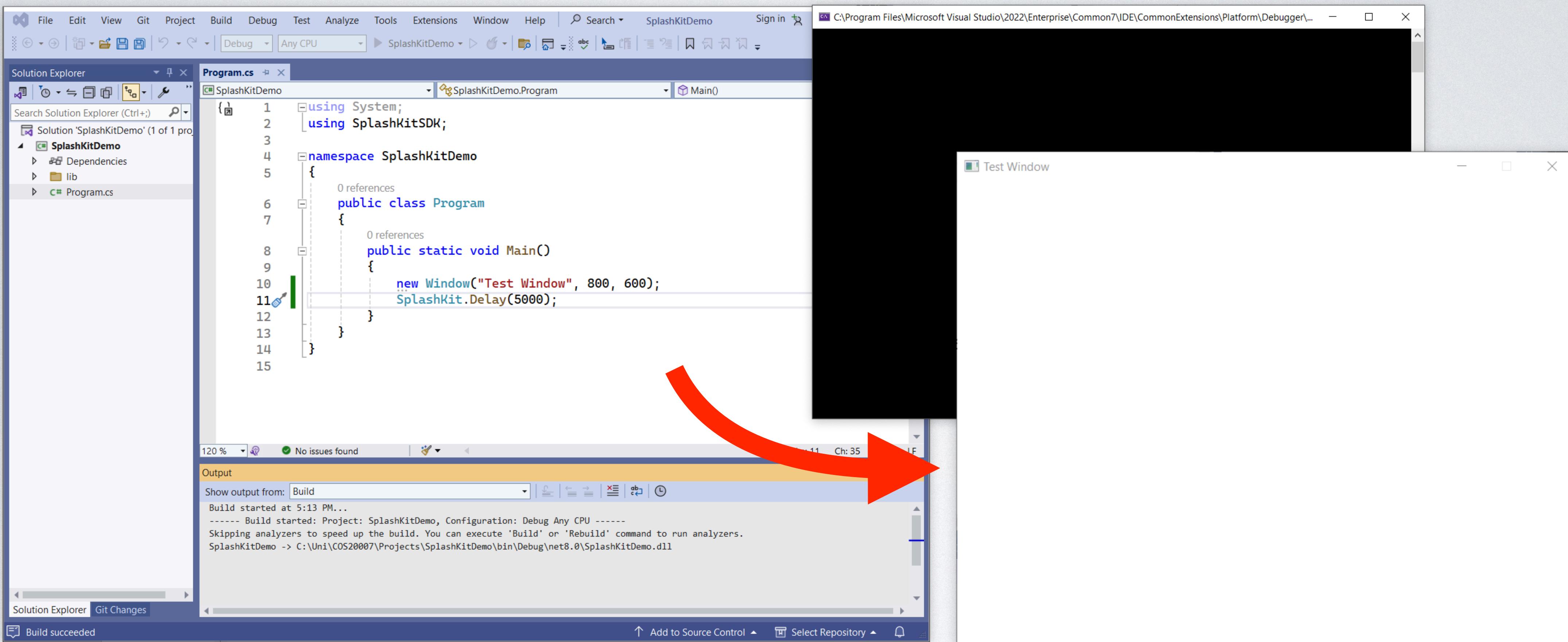
The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows a single project named "SplashKitDemo" with files "Dependencies", "lib", and "Program.cs".
- Editor:** The "Program.cs" file is open, displaying the following C# code:

```
1  using System;
2  using SplashKitSDK;
3
4  namespace SplashKitDemo
5  {
6      public class Program
7      {
8          public static void Main()
9          {
10             new Window("Test Window", 800, 600);
11             SplashKit.Delay(5000);
12         }
13     }
14 }
```
- Status Bar:** Shows "120 %", "No issues found", "Ln: 11 Ch: 35 SPC LF".
- Output Window:** Shows "Show output from:" dropdown and various icons.
- Bottom Bar:** Includes "Ready", "Add to Source Control", "Select Repository", and a notification icon.

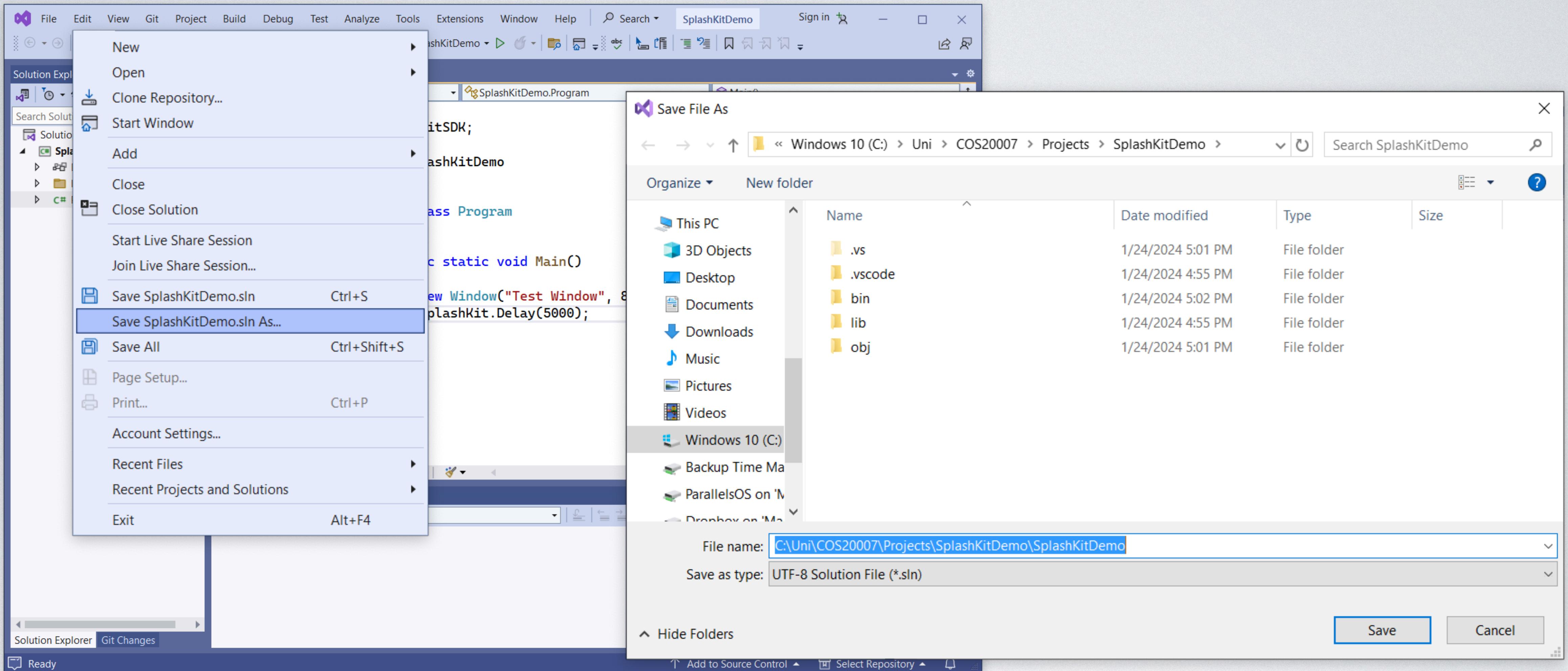
- Open Program.cs.
- In function Main, add the following two lines:
 - **new Window("Test Window", 800, 600);**
 - **SplashKit.Delay(5000);**
- The first line creates a window 800x600 with the title “Test Window”, which is shown immediately.
- The second line instructs SplashKit to show the window for at least 5 seconds. Once 5 seconds have elapsed, the window is closed and the application terminates.

BUILD AND RUN



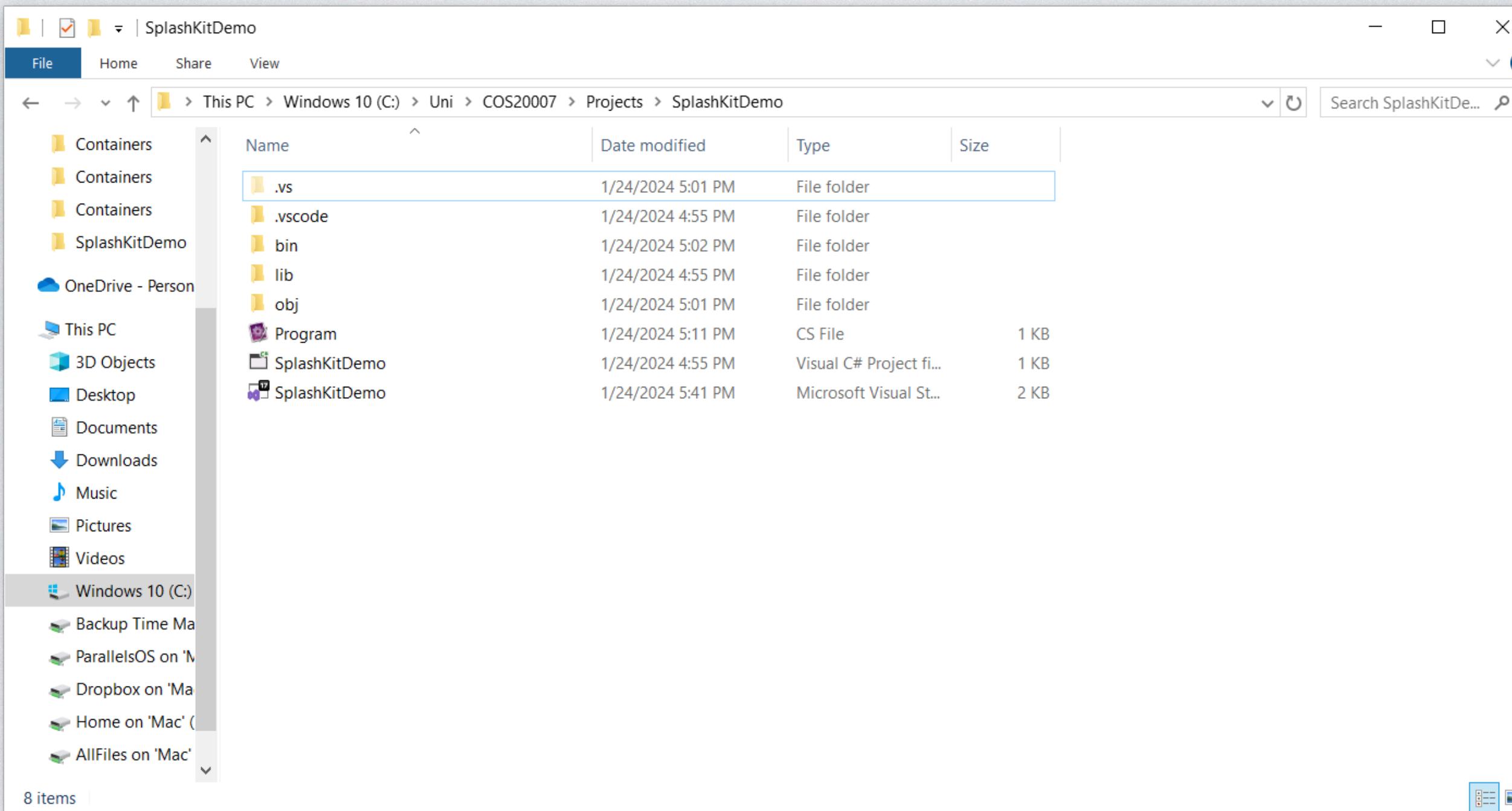
- Build and run the application by clicking on the run button ➤ .

SAVE SOLUTION



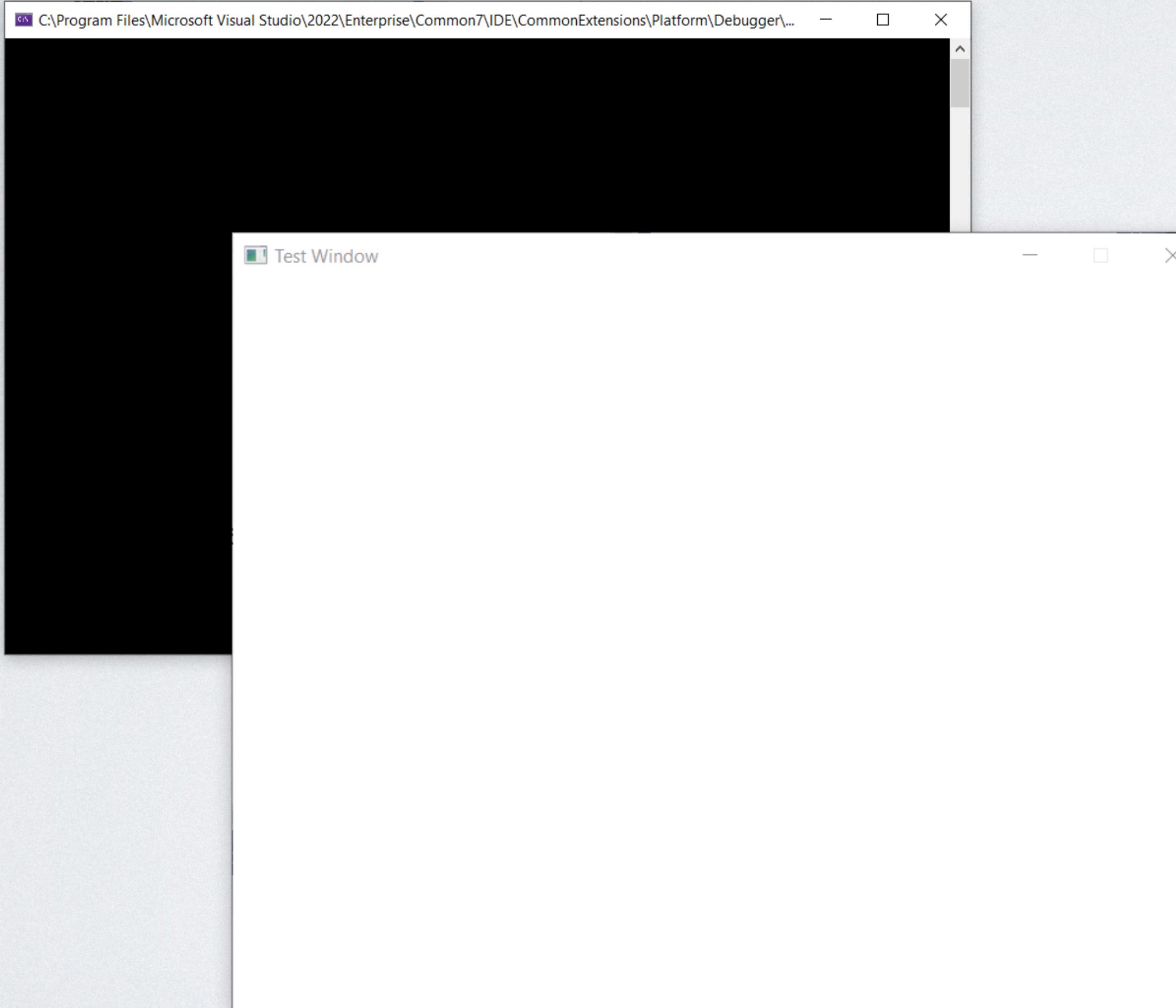
- Finally, select **File/Save SplashKitDemo.sln As...** to save the Visual Studio solution file.

NEW PROJECT FILE STRUCTURE



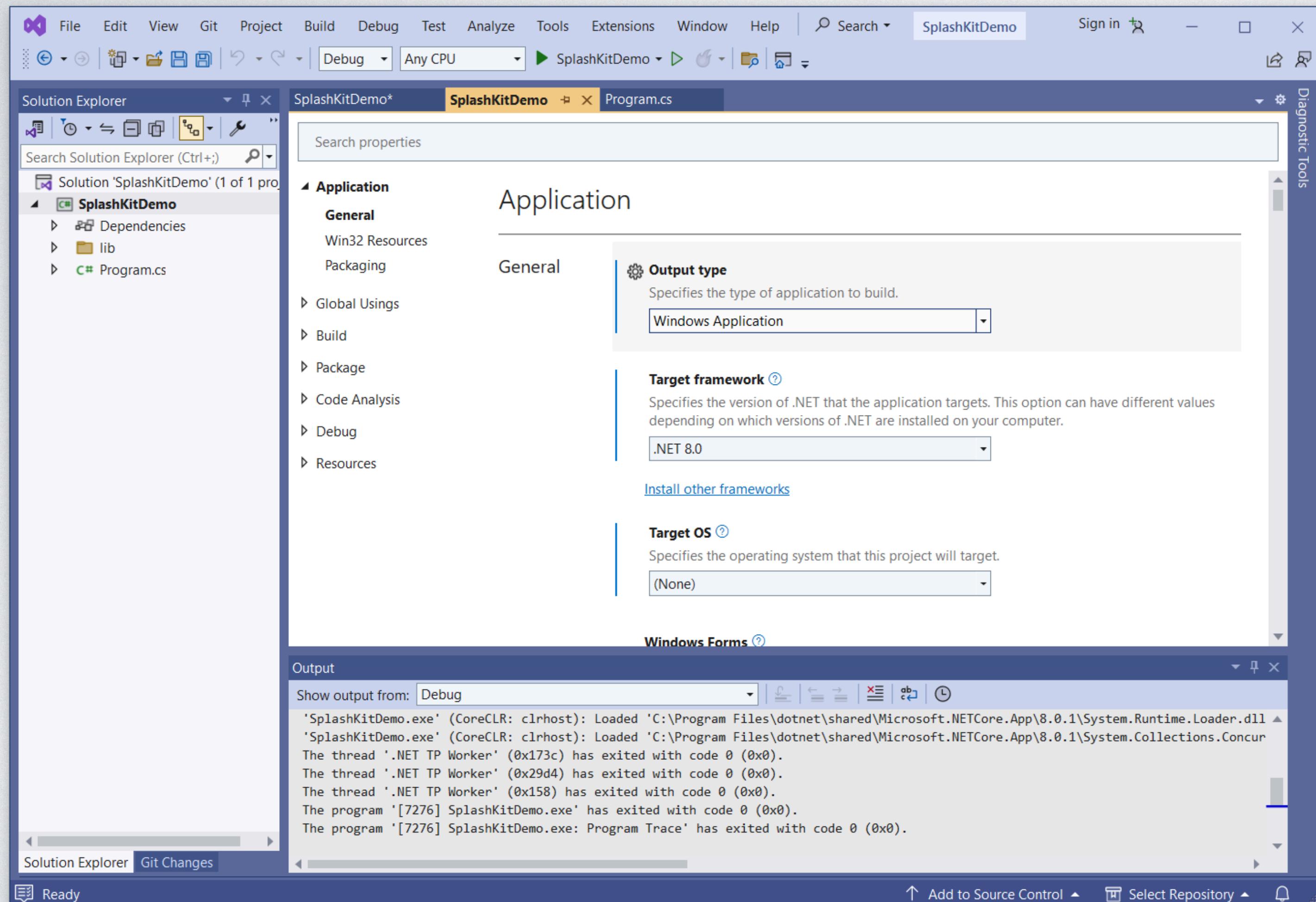
- The new project file structure contains a solution file which you use from now on to open your SplashKit project.
- Remember the distinction between project and solution name. The project name corresponds to the project file (i.e., [SplashKitDemo.csproj](#)), whereas the solution name matches the solution file (i.e., [SplashKitDemo.sln](#)).
- In Visual Studio, a solution can host multiple projects. If a solution only contains one project, generally, solution name and project name are the same.

SUPPRESSING THE TERMINAL WINDOW



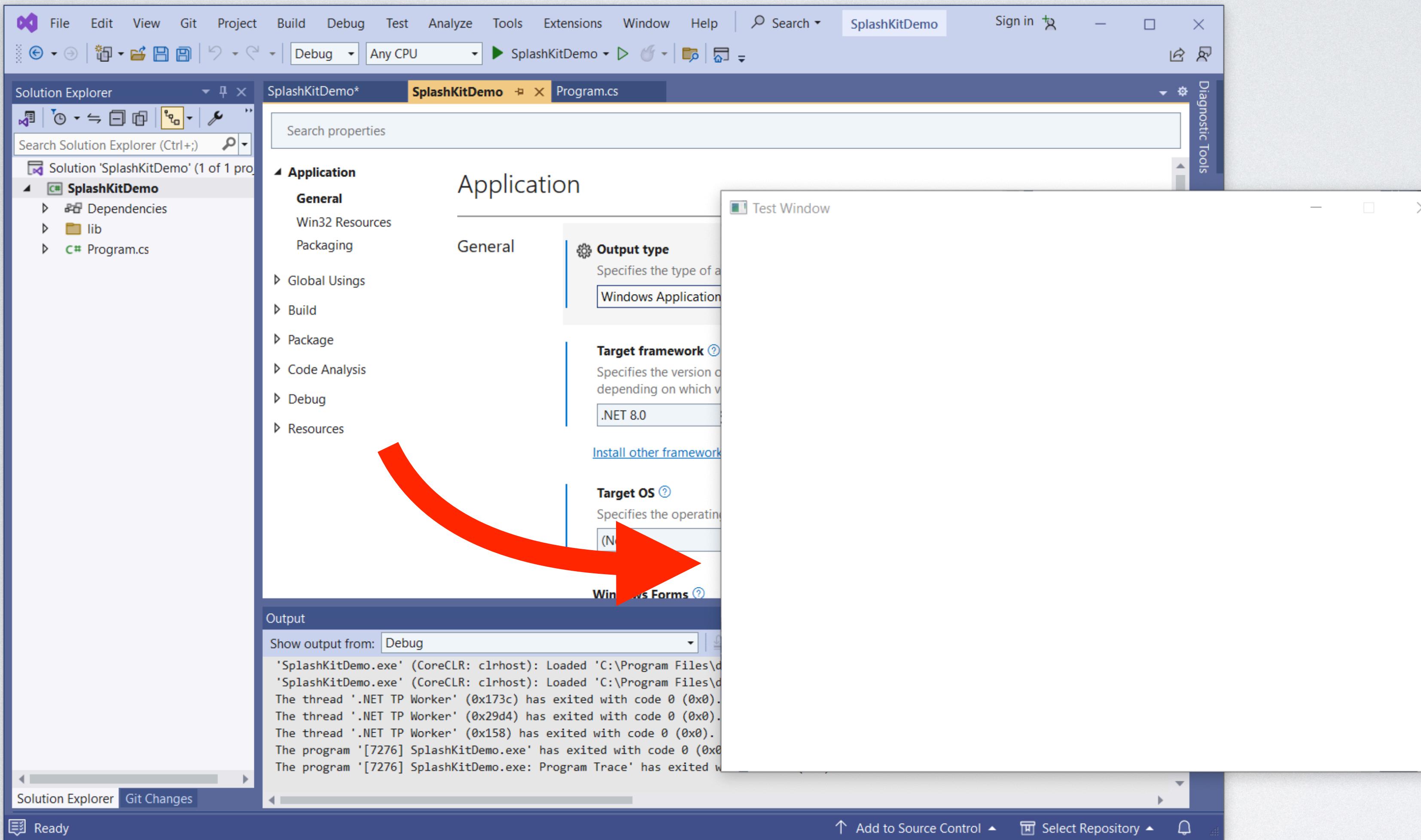
- Did you notice that when you run SplashKitDemo, two windows opened: the main window and a terminal.
- This is the default behavior for SplashKit.
- The terminal allows for console output, if your application generates it.
- It is possible to suppress the terminal, if your application does not generate console output.

APPLICATION TYPE: WINDOWS APPLICATION



- Select the project in the solution explorer.
- Right-click and select properties.
- Visual Studio opens the project page for you project.
- Under **Application/General** change Output type from **Console Application** to **Windows Application**.
- Build the project again.

SPLASH KIT DEMO: NO TERMINAL



HAPPY CODING