

1

Introduction

7

Introducing Rust	8
How to read this book	9
Preparing to install Rust	10
Installing Rust	12
Installing VSCode	14
Configuring VSCode	16
The structure of a project	18

2

Fundamental principles

19

Abstraction	20
The structure of a program	22
Detecting bugs	24
A first program	25
Variables don't	26
Ownership	28
Building a Rust program	29
Borrowing errors	31
Reference limitations	34
Summary	36

3

Saying hello

37

Hello World	38
Visual Studio Code	42
Extending the code	44
Modularizing our code	48
Even more modular	49
Automated testing	50
Documentation	52
Writing libraries	54
Summary	56

4

Structuring data

57

Gathering data	58
Arrays	59
Structs	60
Struct shorthand	61
Struct behavior	62
Enums	65
Enums as variants	66
Associated functions	67
Generics	68
Summary	72

5

Control flow

73

Patterns	74
Matching refutable patterns	76
if-let and let-else	79
Shortcuts	80
Loops	81
Putting it together	84
Summary	88

6

Collections & modules

89

Strings and string slices	90
Vectors and arrays	92
Hash maps	94
Playing cards	96
Scoring	99
User input	100
Utility functions	101
Playing the game	102
Trying it out	104
Extensions	105
Summary	106

7

File structure of a project

107

Workspaces	108
MariaDB	109
Creating the folders	111
Configuring the workspace	112
The magic prompt	113
Defining the database	115
Database struct	116
Accessing the database	118
Ingredients	120
Recipes	122
Final touches	124
Creating the database	127
Importing substances	128
Adding recipes	131
Adding a recipe	133
Showing a recipe	134
Summary	136

8

Implementations

137

Implementing behavior	138
Taking the average	139
The main program	141
Implementing the trait	142
Testing the program	143
Lifetimes	144
Special lifetimes	146
Summary	148

9

Closures and iterators**149**

Iterators	150
Closures	151
Types of closures	152
Creating an iterator	153
Iterators and closures	154
Summary	158

10

Real-world programs**159**

Command-line arguments	160
File handling	163
Preparing patterns	164
Processing the file	165
Executing the program	166

11

Smart pointers and concurrency**167**

Heap storage with Box	168
Multiple ownership	169
Multiple threads	170
Futures	173
Running the program	177
Summary	178

12

Advanced topics**179**

Introduction	180
Basic macros	182
Unsafe Rust	185
Advanced traits	186

Index**187**

1

Introduction

This chapter shows how to install the Rust compiler, and the tools that we will use on our journey to master this powerful language.

- 8** Introducing Rust
- 9** How to read this book
- 10** Preparing to install Rust
- 12** Installing Rust
- 14** Installing VSCode
- 16** Configuring VSCode
- 18** The structure of a project

Introducing Rust

Rust is a high-level programming language. It is designed to be read and written by human beings, and it can express algorithms and data structures that allow them to produce complex programs.

Rust is also a systems programming language. Languages such as Python, BASIC, and Pascal present the programmer with an idealized computer to program. Rust never loses sight of the underlying hardware. This allows the programmer to write programs such as operating systems, device drivers, and low-level communications and database applications.

Before Rust, the systems programming language of choice was C. It offers the same control of the underlying hardware. However, C's flexibility comes with vulnerabilities. Programs written in C often fail with buffer overflows and null pointer exceptions. A founding principle of Rust is that as many bugs as possible should be caught by the compiler, before a program is even run.

As a direct result, Rust is difficult to program in. This complexity is front-loaded. Almost no programs can be written without using variables. Most books that teach a programming language start with them. That is where Rust is at its most obtuse and abstract. We will need a whole chapter to lay the groundwork before we can introduce variables and write our first programs.

Rust is certainly not suitable as a first language for a beginner programmer. This book is written for readers who know at least one programming language already. You should already understand the basic concepts, such as variables and functions, and be able to string them together to create a program.

While Python programs are powerful, they run slowly because they are interpreted and have dynamic types. Rust is a compiled language with static types, like C, Pascal, and Fortran. That makes Rust programs fast.

Rust is a young language. Its standard library is small, and the selection of third-party libraries is immature and changes quickly. This means that some features, such as windowed programs, are not reliably supported. While some such libraries exist as I write this book, I can't be certain they will still be around when you read it.

How to read this book

This book is written for:

- **Enthusiasts**, who want to explore the Rust language.
- **Developers**, who want to use Rust in their projects.
- **Programmers**, who want to add Rust to their skillset.
- **Students**, or those seeking a career in computing.

It assumes a familiarity with programming. If you are new to software development, I recommend you take the time to read **Python in easy steps** first.

When I present code for you to type in, it will be in a sans-serif font with syntax highlighting. Comments will be green; user-provided names, red; numbers or strings will be black; and everything Rust provides will be cyan, like this:

```
/// Provide a pluralizing s to suffix a string.  
fn plural(x: i64) -> String {  
    let mut s = String::new();  
    if x != 1 {  
        s.push('s');  
    }  
    s  
}
```

We will be putting our code into several source files. So each time I give code, there will be an icon in the margin specifying which file it should go into, like this:



hello_cargo\src\main.rs

All the code can be found by going to www.ineasysteps.com then navigating to **Free Resources** then **Downloads**, clicking on the title of the book, and selecting **All Code Examples**.

Unzip the downloaded file into a folder of your choosing. For example, create a folder called **MyProjects** in your **Documents** folder.

If you don't achieve the result illustrated in any example, simply compare your code to that in the original example files you have downloaded to discover where you went wrong.



Hot tips give some extra information on the subject being discussed.



These tips remind you about subjects earlier in the book that have a bearing on current subject.



Notices like this warn you about difficulties you might encounter.



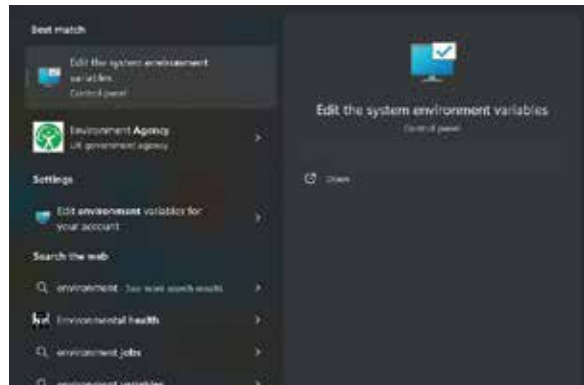
This book uses Windows throughout. Rust is available for many platforms, including Linux and macOS. The programs that we are using, including Visual Studio Code, are also available on at least these three platforms. You should be able to follow along whether you are using Windows, Linux, or macOS. On other platforms there may be programs that you can't install. However, you can program Rust with just a text editor and a command line. Most of the projects should still work fine.

Preparing to install Rust

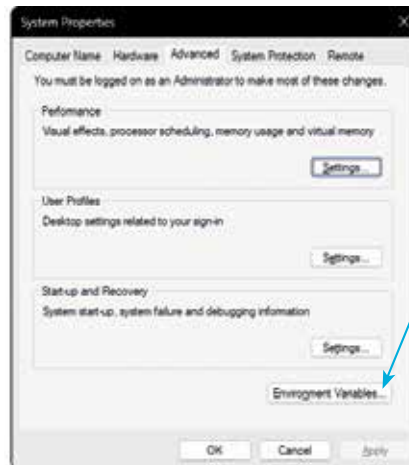
Before you can begin programming in the Rust language, you need to install the Rust compiler and the Cargo tool on your computer. By default, these tools will be installed under your user profile in the `C:\users` folder.

On my computer, the `C:` drive is short of space, so I prefer to install applications elsewhere. If you are content to use the default location, then you can skip these initial steps.

- 1 Click the Windows **Start** button or press the **Windows** key. Then type “environment”. Windows will offer **Edit the system environment variables**

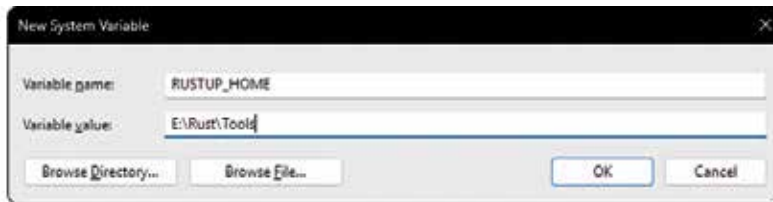


- 2 Click the application or press **Enter**. The System Properties dialog will be shown. Click the **Environment Variables...** button

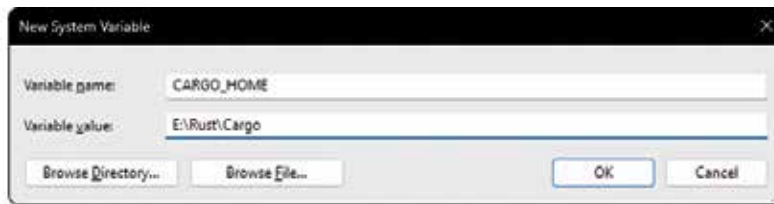


...cont'd

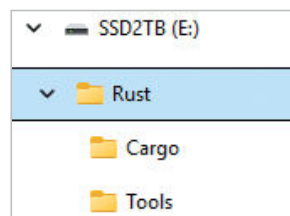
- 3 You have the choice to add environment variables either only to your profile or for all users. Click the corresponding **New...** button. Use **RUSTUP_HOME** as the variable name, but choose a folder location that makes sense to you



- 4 Do the same again, to create another environment variable called **CARGO_HOME**. Of course, you will be choosing your own folder



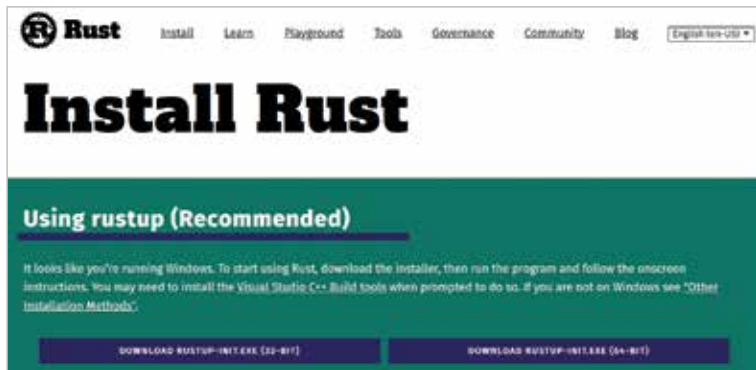
- 5 Finally, create the folders that you have specified in the environment variables. This is how I decided to arrange things; yours may be different



Installing Rust

We will install Rust with the RustUp tool.

- 1 Open a browser window and enter this URL in the navigation bar: **www.rust-lang.org/tools/install**
Choose the installation button that corresponds to your requirements – that is probably the 64-bit version



- 2 The browser will download a file called **rustup-init.exe**. Execute this file from the download folder. It will open a command-line window

```

@Unstabl...rustup-init.exe  *  +  -  -  □  ✕
The Cargo home directory is located at:
  E:\Rust\Cargo

This can be modified with the CARGO_HOME environment variable.

The cargo, rustc, rustup and other commands will be added to
Cargo's bin directory, located at:
  E:\Rust\Cargo\bin

This path will then be added to your PATH environment variable by
modifying the HKEY_CURRENT_USER\Environment\PATH registry key.

You can uninstall at any time with rustup self uninstall and
these changes will be reverted.

Current installation options:

  default host triple: x86_64-pc-windows-msvc
  default toolchain:  stable (default)
                    profile: default
  modify PATH variable: yes

1) Proceed with standard installation (default - just press enter)
2) Customize installation
3) Cancel installation
4)
  
```

- 3 Hit the **Enter** key to accept the default options

...cont'd

```
@Windows/rustup-init.exe
Info: downloading component 'cargo'
Info: downloading component 'clippy'
Info: downloading component 'rust-docs'
Info: downloading component 'rust-std'
Info: downloading component 'rustc'
42.6 MiB / 42.6 MiB (100 %) 34.7 MiB/s in 1s ETA: 0s
Info: downloading component 'rustfmt'
Info: installing component 'cargo'
Info: installing component 'clippy'
Info: installing component 'rust-docs'
10.5 MiB / 10.5 MiB (100 %) 1.0 MiB/s in 0s ETA: 0s
Info: installing component 'rust-std'
22.1 MiB / 22.1 MiB (100 %) 14.2 MiB/s in 1s ETA: 0s
Info: installing component 'rustc'
40.6 MiB / 40.6 MiB (100 %) 15.0 MiB/s in 4s ETA: 0s
Info: installing component 'rustfmt'
Info: default toolchain set to 'stable-x86_64-pc-windows-msvc'

stable-x86_64-pc-windows-msvc installed - rustc 1.83.0 (50615e223
2024-11-26)

Rust is installed now. Great!

To get started you may need to restart your current shell.
This would reload its PATH environment variable to include
Cargo's bin directory (E:\Rust\Cargo\bin).

Press the Enter key to continue.
```

- 4 Hit the **Enter** key to close the window. Then, click the **Start** button, type **CMD** and press **Enter** to start a new command-line window. Navigate to the folder that you want to use for your Rust projects

```
Command Prompt
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Richard>g:
G:\>cd "Rust Projects"
G:\Rust Projects>
```

- 5 Now, we can run our first Rust program by using the Cargo tool to create a trivial project

```
Command Prompt
G:\Rust Projects>cargo new hello_cargo
Creating binary (application) 'hello_cargo'
package
note: see more 'Cargo.toml' keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

G:\Rust Projects>cd hello_cargo

G:\Rust Projects\hello_cargo>cargo run
Compiling hello_cargo v0.1.0 (G:\Rust Projects\hello_cargo)
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 1.19s
Running 'target\debug\hello_cargo.exe'
Hello, world!

G:\Rust Projects\hello_cargo>
```



We will see the Cargo tool in detail later. The commands used here are:

- cargo new** – which creates a Rust project in a new folder.
- cargo run** – which runs a rust project in the current folder.

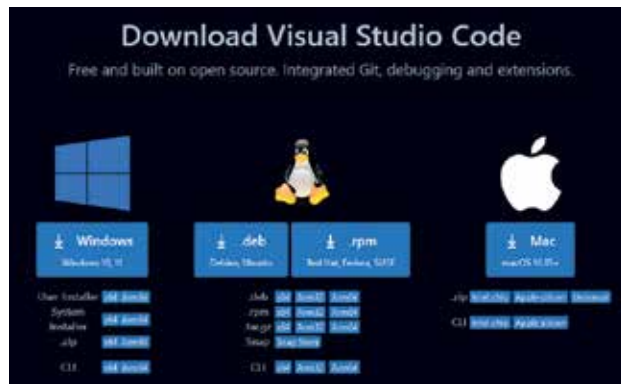
The project that cargo new creates contains a trivial program as a place-keeper. We don't even need to see any Rust code to say "Hello World!".

Installing VSCode

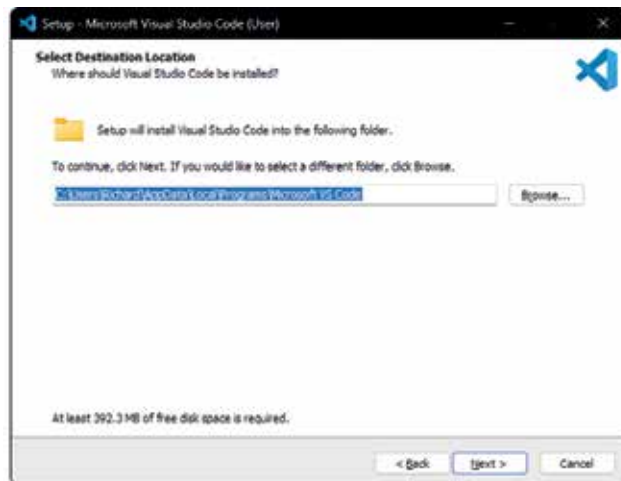
It is perfectly possible to develop Rust programs using only the Rust compiler, Cargo, and a text editor. However, it is far easier to do so with an Integrated Development Environment (IDE) such as VSCode.

VSCode is a free IDE from Microsoft that is based on its flagship Visual Studio product.

- 1 Bring up a browser and navigate to code.visualstudio.com/download

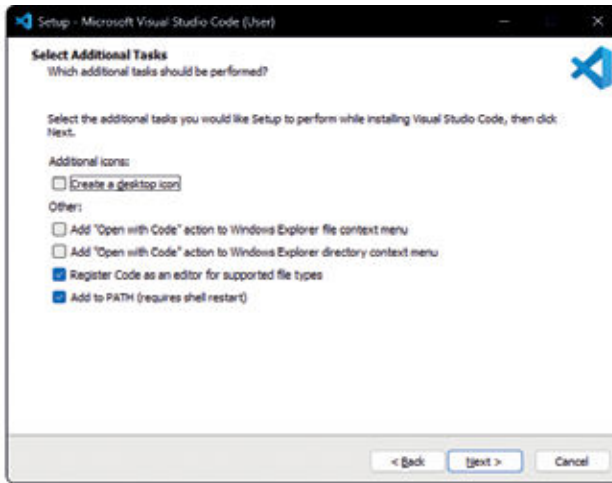


- 2 Click the big **Windows** button. An installer will be downloaded. Execute that, accept the license agreement, and click **Next**



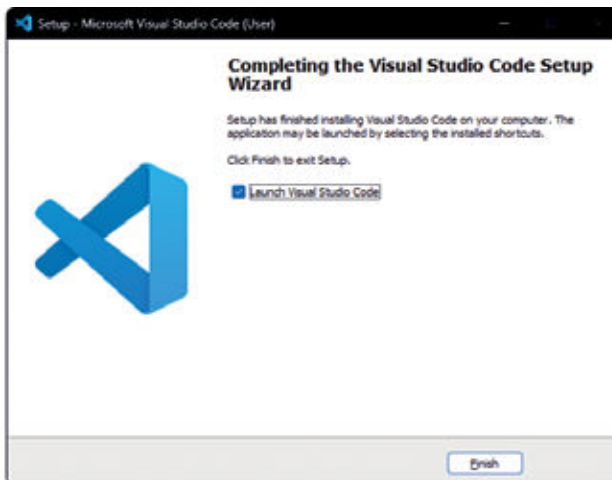
...cont'd

- 3 Choose a location to install VSCode or accept the default. Then click **Next**. The next panel asks about creating shortcuts. The default is fine, so just click **Next** again.



You can't install VSCode in the usual Program Files folder, probably because the installer doesn't ask for Administrator privilege. If you gave it that, then it might crash when running. Put it somewhere else.

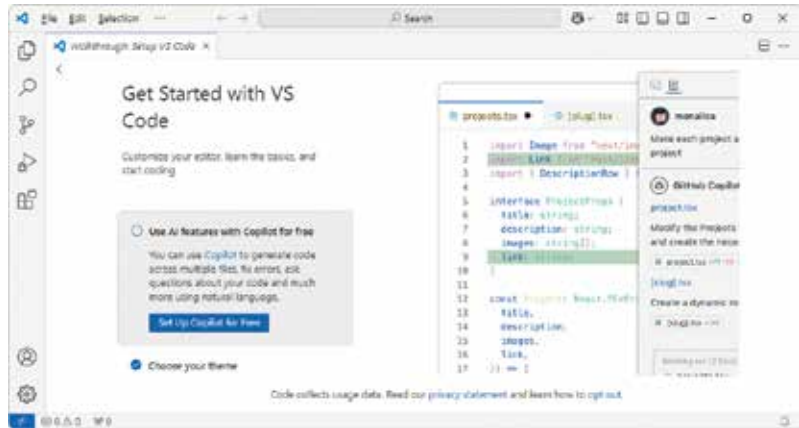
- 4 All of these options are a matter of personal taste. I suggest that you accept the default and just click **Next** and then **Install**. Leave the checkbox asking whether to launch Visual Studio Code, because the next thing we do will be to add extensions to it.



Configuring VSCode

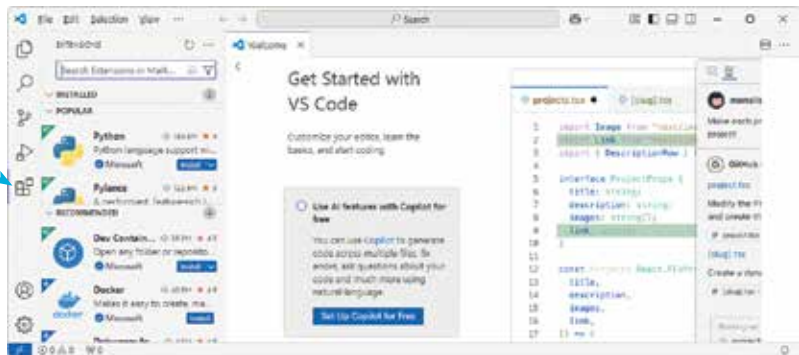
We will need to configure VSCode with some extensions before we are able to use it to write Rust code.

- 1 Start VSCode, if it is not already running. It uses a dark theme by default. I've changed to the **Light Modern** theme here so that it is easier to read on the page



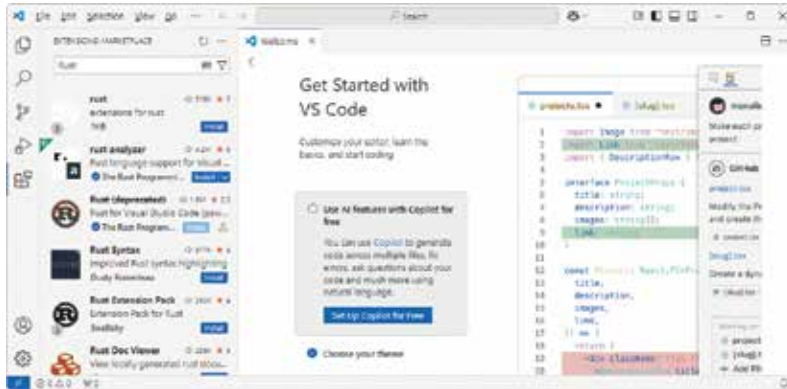
The Extensions icon can be found on the left-hand bar, here.

- 2 On the left-hand bar there is an icon consisting of four squares. Click it to bring up the Extensions panel



- 3 Type “Rust” into the search bar

...cont'd

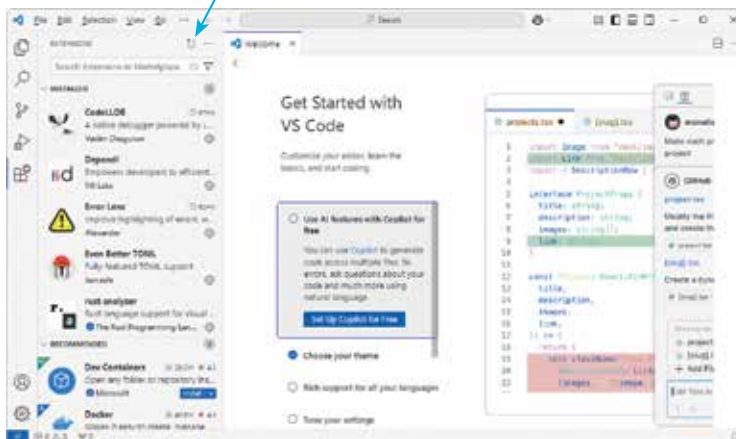


4 Click the **Install** button for the **rust-analyzer** extension. Do not install any of the others

5 In the same way, search for and install the following extensions

- **CodeLLDB**
- **Dependi**
- **Error Lens**
- **Even Better TOML**

6 Click the **Refresh** icon to see all the installed extensions



Other extensions may be useful. However these are the only ones I will be using in this book, and some extensions are obsolete or incomplete.

The structure of a project

Investigate the file structure that Cargo created for the **hello_cargo** project that we created on page 13.

This is a lot of files and folders for such a simple program. We didn't need anywhere near this complexity to write Hello World in Python or C.

Most of these folders can be safely ignored. They are used by the Cargo tool to provide us with dependency management and so forth.

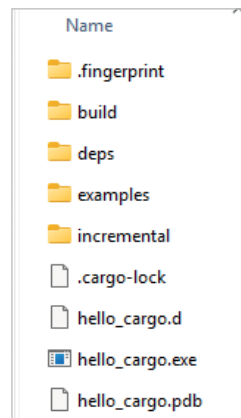
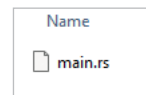
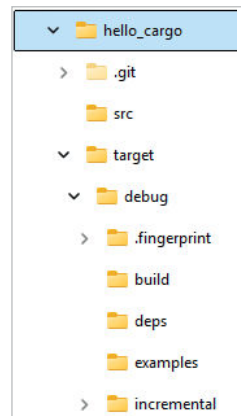
Notice the **.git** folder. Cargo has created this project as a git repository. We won't be using git, but, for larger projects, it would be useful.

The **src** folder holds only one file: **main.rs**. This is where our Rust source code is located. The main source code file of a project is always called either **main.rs** or **lib.rs**.

The **target/debug** folder is more complex, but the only file we are interested in here is **hello_cargo.exe**. This is the executable file that we can run from the command line.

1

Open a command-line window, navigate to the **target/debug** folder, and directly execute the **hello_cargo.exe** file



```

Command Prompt
(c) Microsoft Corporation. All rights reserved.
C:\Users\Richard>
G:\>cd "Rust Projects\hello_cargo\target\debug"
G:\Rust Projects\hello_cargo\target\debug>.\hello_cargo.exe
Hello, world!
G:\Rust Projects\hello_cargo\target\debug>

```