

# How to Get Started With Rust on Raspberry Pi

7-8 minutes

---

If you are interested in programming, you've probably [heard of Rust](#). The language, designed by Mozilla, is widely loved by developers and continues to grow in devotees.

[7 Reasons Rust Is the Most Exciting New Programming Language Want to get started with programming? Here's why Rust is the most exciting and accessible new programming language. Read More](#)

Advertisement

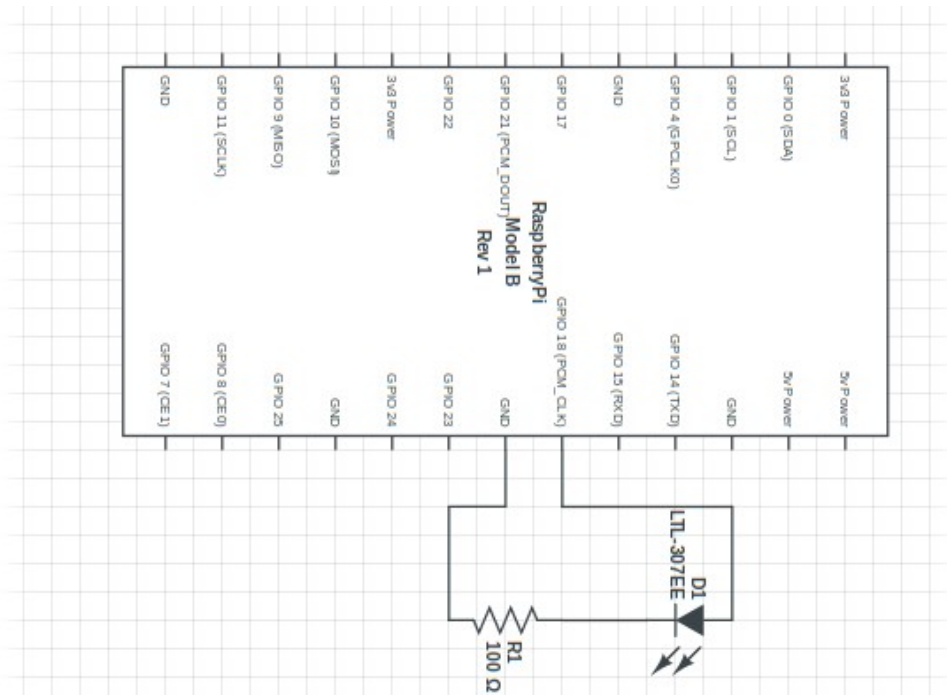
The Raspberry Pi is the swiss army knife of small computers, and it's perfect for learning code. Let's combine the two and install Rust on a Raspberry Pi.

## Setting Up Your Raspberry Pi

For this project you will need:

- Raspberry Pi.
- LED.
- 220-1k Ohm resistor.

- Breadboard and hookup wires.



Set up your circuit with GPIO 18 connected to the positive leg of the LED, and the negative leg of the LED to the resistor, and back to a GND pin on the Pi.

This tutorial was made using a Raspberry Pi 3B+ with Raspbian Stretch in desktop mode. It would work perfectly fine through a remote SSH connection too, though different models of Pi and different operating systems might have varying results.

## How to Install Rust on Raspberry Pi

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ curl https://sh.rustup.rs -sSf | sh
```

To install rust, head to the [rust-lang install page](https://rust-lang.org/install) and copy the install command into your terminal. When prompted, choose a default installation.

```
1) Proceed with installation (default)
2) Customize installation
3) Cancel installation
>1

info: syncing channel updates for 'stable-armv7-unknown-linux-gnueabi'
info: latest update on 2019-02-28, rust version 1.33.0 (2aa4c46cf 2019-02-28)
info: downloading component 'rustc'
 4.6 MiB / 71.2 MiB ( 6 %)  0 B/s ETA: Unknown
```

The installer will notify you when it is complete, though installation can take some time depending on your connection.

## After Installation

```
Rust is installed now. Great!

To get started you need Cargo's bin directory ($HOME/.cargo/bin) in your PATH
environment variable. Next time you log in this will be done automatically.

To configure your current shell run source $HOME/.cargo/env
pi@raspberrypi:~ $ rustc --version
bash: rustc: command not found
pi@raspberrypi:~ $ cargo --version
bash: cargo: command not found
pi@raspberrypi:~ $
```

The installation is successful, but you can't start using it quite yet. If you try to check for Rust and Cargo by version, you will get an error. Usually, you must add a language to your PATH to use them on the command line.

Luckily Rust does this for you, and all you need to do is reboot your Pi, or log out and in again. Now checking for Rust and Cargo should work.

```
pi@raspberrypi:~ $ rustc --version
rustc 1.33.0 (2aa4c46cf 2019-02-28)
pi@raspberrypi:~ $ cargo --version
cargo 1.33.0 (f099fe94b 2019-02-12)
pi@raspberrypi:~ $
```

You'll be compiling and building all of your scripts from the terminal, but you'll also need a code editor. In this project I'll

be using Code-OSS, a community build of VS Code which you can install on the Pi, but it's not essential. Any code editor will do.

## Creating a Rust Project

To create a Rust project, make a new directory and enter it by typing

```
mkdir YourFolder  
cd YourFolder
```

Use Cargo to create a new Rust project.

```
cargo new YourProject
```

You'll get a confirmation that the new project has been created.



```
pi@raspberrypi:~ $ mkdir rust  
pi@raspberrypi:~ $ cd rust  
pi@raspberrypi:~/rust $ cargo new test_rust  
    Created binary (application) `test_rust` package  
pi@raspberrypi:~/rust $
```

Enter the new project folder and list its contents.

```
cd YourProject  
ls
```

You'll see a folder named **src** and a file called **Cargo.toml**. These two elements make up the basis of every Rust project.

## A Simple Rust Project, Explained



```
main.rs x  
1 fn main() {
```

```
2     println!("Hello, world!");  
3 }
```

First, let's open up the `src` directory, and open **main.rs** in a code editor. You'll see that the new project comes complete with a "Hello World" script to get you started.

Rust syntax will be familiar to those who have used C languages or Java before. This differs from Python which uses whitespace, semi-colons and braces to denote code blocks. Rust code must compile and build before it runs.

```
1 [package]  
2 name = "test_rust"  
3 version = "0.1.0"  
4 authors = ["ian"]  
5 edition = "2018"
```

Back in the project's parent folder, open up **Cargo.toml** in a code editor. Anyone who has coded in JavaScript or Ruby will likely find this familiar. Project information, build instructions, and dependencies are all listed in this file. Packages are called **Crates** in Rust, and we'll be using one later to access the Raspberry Pi's GPIO pins.

## Building the Sample Project

Back in the terminal window, make sure you are in your project directory and build the project.

```
cargo build
```



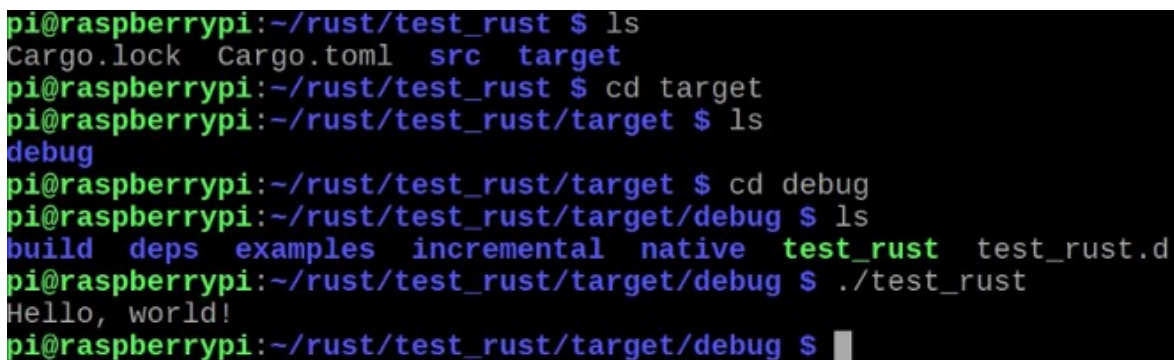
```
pi@raspberrypi:~/rust/test_rust $ cargo build
   Compiling test_rust v0.1.0 (/home/pi/rust/test_rust)
   Building [
```

This creates another folder within your project called **target**. You will also notice a new file called **Cargo.lock**. When working with a team or coding something to deploy to a server, this file locks the project to a version that has previously compiled and built successfully. While learning, you can safely ignore this file.

Within the target folder is a subfolder called **debug**, and this is where your executable file will be. On Mac and Linux, run your project by typing:

```
./YourProject
```

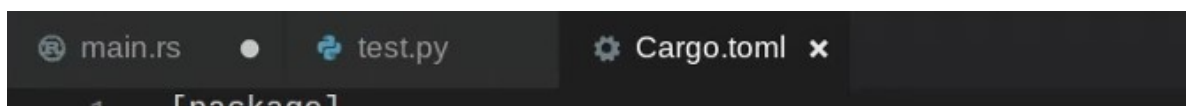
On Windows, you will have a new **EXE** file which you can run by double-clicking.



```
pi@raspberrypi:~/rust/test_rust $ ls
Cargo.lock  Cargo.toml  src  target
pi@raspberrypi:~/rust/test_rust $ cd target
pi@raspberrypi:~/rust/test_rust/target $ ls
debug
pi@raspberrypi:~/rust/test_rust/target $ cd debug
pi@raspberrypi:~/rust/test_rust/target/debug $ ls
build  deps  examples  incremental  native  test_rust  test_rust.d
pi@raspberrypi:~/rust/test_rust/target/debug $ ./test_rust
Hello, world!
pi@raspberrypi:~/rust/test_rust/target/debug $
```

Success! Let's convert this project into something that uses the GPIO pins.

## Setting Up GPIO Pins





```
1 [package]
2 name = "test_rust"
3 version = "0.1.0"
4 authors = ["pi"]
5 edition = "2018"
6
7 [dependencies]
8 rust_gpiozero = "0.2.0"
```

We will be using the [rust\\_gpiozero crate by Rahul Thakdoor](#) for this project. While it is not the only way to access GPIO pins, this crate is designed to be similar to the Python GPIO Zero library.

Instead of manually downloading the crate, paste its name under dependencies in the Cargo.toml file.

```
[dependencies]
rust_gpiozero = "0.2.0"
```

Save it, and open your terminal. At this stage, there is no point in rebuilding the project as no code has changed.

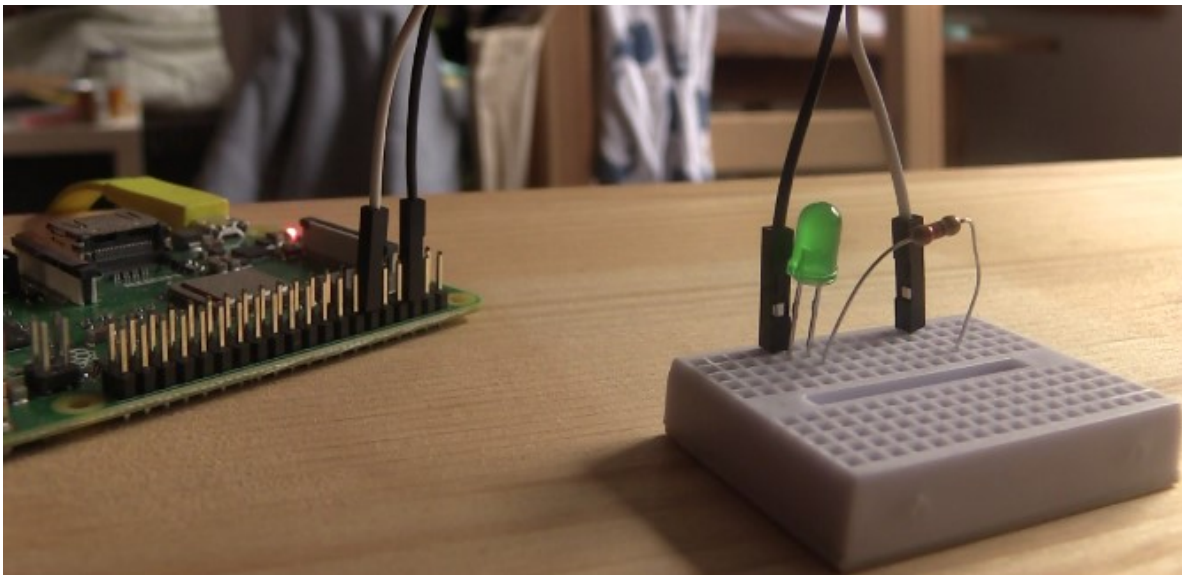
Cargo provides a function which will check that the code will compile and that all dependencies are present.

```
cargo check
```

```
ian@hex:~/rustProjects/test_rust$ cargo check
Compiling libc v0.2.50
Checking lazy_static v1.3.0
Checking rppal v0.11.1
Checking rust_gpiozero v0.2.0
Checking test_rust v0.1.0 (/home/ian/rustProjects/test_rust)
Finished dev [unoptimized + debuginfo] target(s) in 3.44s
```

Depending on your connection this can take a few minutes, but you only need to do it once, when you add or change items in the Cargo.toml file.

# Hello Blink



Now you will change your hello world script into a blinking light script. Begin by opening `main.rs` in your editor. If you want to skip coding, you can find the finished script on [Github Gist](#).

You need to let the compiler know you are using the `rust_gpiozero` library, so at the very top of the script add a reference to the library.

```
use rust_gpiozero::*;
```

Much like the regular Python based blink sketch, we need a way to add a delay between turning the LED on and off. In Rust, we use two elements of the standard library to do this:

```
use std::thread::sleep;
use std::time::Duration; // note the capital
D!
```

Now in your **main** function, add a variable for your LED pin, and a loop to contain the blinking instructions.

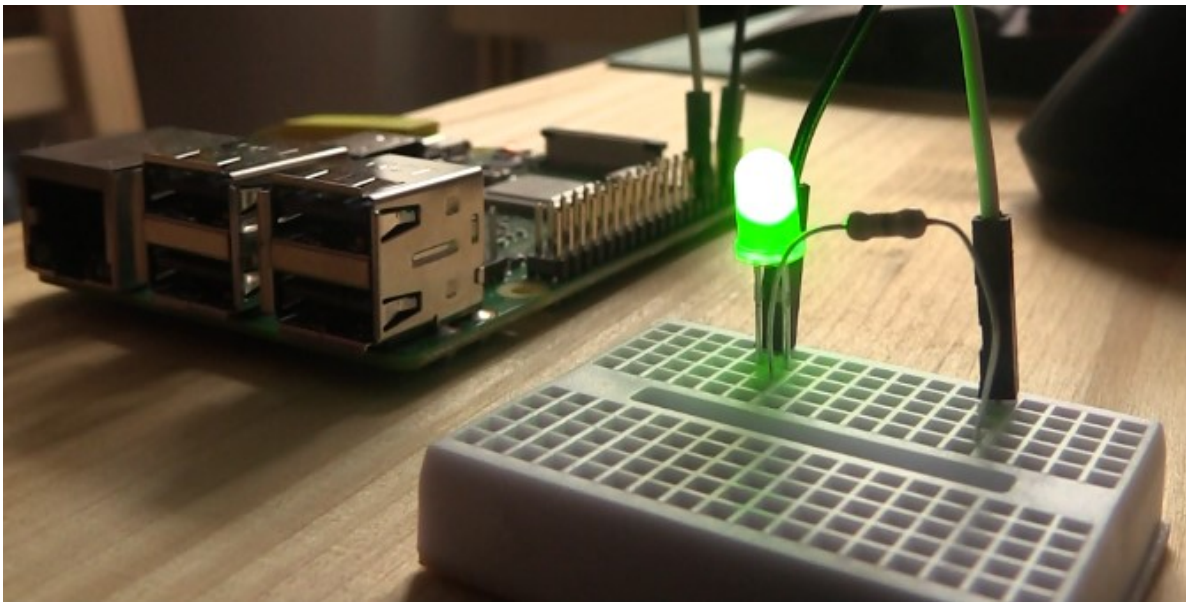


```
let led = LED::new(18); // sets a variable
for the led pin

loop{ // starts a loop
    led.on();
    sleep(Duration::from_secs(1)); // creates a
1 second pause
    led.off();
    sleep(Duration::from_secs(1));
}
```

That's it! Save your script, and return to the terminal.

## Test It Out



Build the project again to update the executable.

Alternatively, the run command builds and runs the script in a single step:

```
cargo run
```

You should see a blinking LED. Well done! You've just made your first hardware program with Rust. Press **Ctrl-C** to exit back to the terminal. If you have any errors, check over your code thoroughly for any missed colons, semi-colons or brackets.

## An Exciting Future With Rust on Raspberry Pi

Currently, Python isn't in any danger of being replaced by Rust. It is easy to learn and [Python will have many applications for years to come](#).

That said, Rust has quite a buzz around it, and there are many [reasons why you should learn the language](#)!

**Affiliate Disclosure:** By buying the products we recommend, you help keep the site alive. [Read more](#).