Published in Better Programming

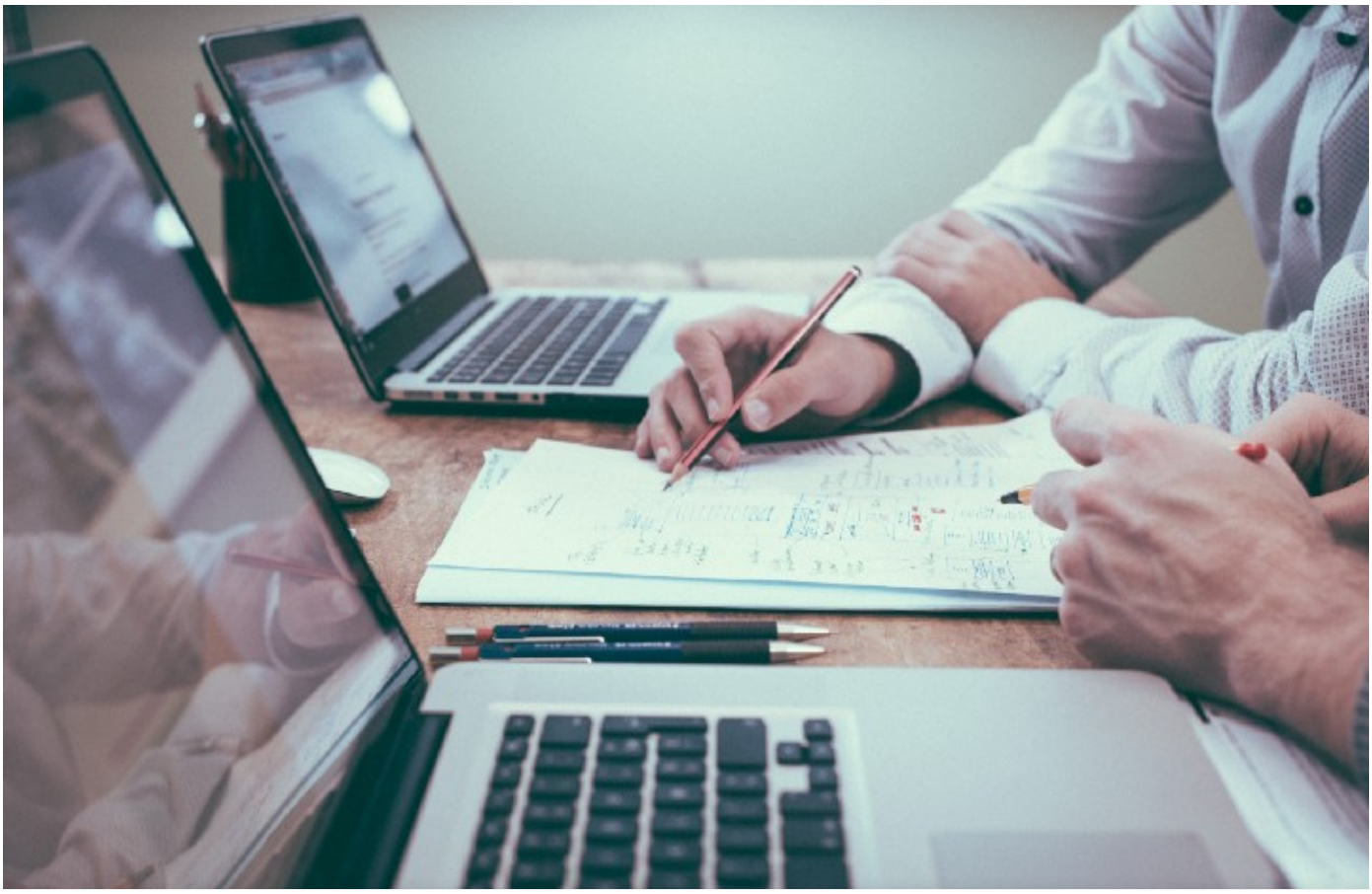
Mousse Anis Follow

Aug 30 · 5 min read · · Listen

Save

Reading and Writing a File in Rust

This article will review how you can create, open a file, read its contents and write into it in Rust.



After reading this article, you will be able to create, open a file, read its content, and add content to it. So let get dive in.

But first, let's lay some vital groundwork by reviewing how characters are represented in Rust.

String and str (aka string slices)

I know this title (☝🏾)seems a bit confusing ... in Rust, we have two types to represent strings :

- `str` : string slice, the only string type of the core Rust programing language. This type is static and immutable, and we usually see it in its borrowed form: `&str` .
- `String` : is the second string type made available in Rust, thanks to the standard library. This type is more flexible, as it can be owned and is mutable.

Both string types are UTF-8 encoded (more on that here); `String` is used when you need to own or change your string and `&str` when you want to visualize a string.

The Rust standard library allows us to perform diverse file operations thanks to the awesome struct `File` and associated methods. We are going to leverage it in the rest of this article.

We are all set now, so let's jump right into it!!


the below example:

```rust
use std::fs::File;
fn main() {
    let file_name = "missy.txt";
    match File::create(file_name){
        Ok(file) => println!("{:?}", file),
        Err(_) => println!("Unable to create the file: '{}'", file_name)
    }
}
```

io_1.rs hosted with ❤ by GitHub view raw

Create a file in Rust

The `create` method returns a `Result`, as we can be successful in the creation of the file, or we might have an error (disk entire, no privileges, etc., choose your poison 💀).

`Result` is an **enum** also provided by the Rust standard library, allowing us to manage errors (more here).

How do you open a file?

To open a file, we are going to use the `open` method of the struct `File`, which also returns a `Result`.

```rust
use std::fs::File;
fn main() {
    let file_name = "missy.txt";
    match File::open(file_name) {
        Ok(file) => println!("{:?}", file),
        Err(_) => println!("Unable to create the file: '{}'", file_name),
    }
}
```

io_2.rs hosted with ❤ by GitHub view raw

I encourage you to run our last example with an existing and nonexisting file to see the outcomes.

How do you read the content of a file?

The simple way to read the content of a file is to open it and read its content as a `String` (now you see why we had this first section introducing the string types).

```
1  use std::fs::File;
2  use std::io::{Error, Read};
3  fn main() -> Result<(), Error> {
4      let file_name = "missy.txt";
5      let mut file = File::open(file_name)?;
6      let mut contents = String::new();
7      file.read_to_string(&mut contents)?;
8      print!("{}", contents);
9      Ok(())
10 }
```

io_6.rs hosted with ❤ by GitHub view raw

Read the content of a file in Rust

As you can see, we create a mutable `String` variable named `contents`, and we are updating it by passing the file's content on line 7 using the method `read_to_string()`. I encourage you to check the method `read`, which allows you to retrieve the content of a file as a bytes vector(more on it here).

I prefer to read a content of a file using the struct `BufReader<R>` as it is more efficient and optimal. The *trick* is that `Bufreader` provides a buffering component while reading from a file (more on that here).

```
1  use std::fs::File;
2  use std::io::{Error, Read, BufReader};
3  fn main() -> Result<(), Error> {
4      let file_name = "missy.txt";
```

```
10      Ok(())
11  }
```

io_7.rs hosted with ❤ by GitHub view raw

A better way to read the content of a file in Rust

As you can see, we started our example by opening a file, as we learned earlier. Then on line 6, we are creating the `reader` variable; a `BufReader` on the file we just opened. We will then use this `reader` to extract the file's content (line 8).

How do you add some content to a file?

Adding content is not a simple task, as we might want to add some content to a new file, overdrive the content of an existing file or maybe add some content to an existing file.

What seemed trivial a few seconds ago now appears more complex … welcome (back) to the beauty of software development.

So let's start with the first two cases; The following example shows you how to add content to a newly created file or open an existing one and override its content.

```
1  use std::fs::File;
2  use std::io::{Error, Write};
3  fn main() -> Result<(), Error> {
4      let file_name = "missy.txt";
5      let mut file = File::create(file_name)?;
6      writeln!(file, "Hello my name is Missy!")?; // writing using the macro 'writeln!'
7      file.write_all(b"Good Day")?; // writing using the method 'write_all'
8      Ok(())
9  }
```

io_3.rs hosted with ❤ by GitHub view raw

Add some content to a file in Rust

In our example, if the file `missy.txt` does not exist, we will create it before adding our


Using `create` on existing files does work, but if we want to be more precise, we should use the struct `OpenOptions` to determine how a file is accessed.

```
1  use std::fs::OpenOptions;
2  use std::io::{Error, Write};
3  fn main() -> Result<(), Error> {
4      let file_name = "missy.txt";
5      let mut file = OpenOptions::new()
6          .read(true)
7          .write(true)
8          .create(true)
9          .append(false)
10         .open(file_name)?;
11     writeln!(file, "Hello my name is Missy!")?; // writing using the macro 'writeln!'
12     file.write_all(b"Good Day\n")?; // writing using the method 'write_all'
13     Ok(())
14 }
```

io_4.rs hosted with ❤ by GitHub view raw

Update a file content in Rust

In this example, we open an existing file and add some content at the beginning. If there is some content in the file, our new content will be added at the beginning replacing only the amount of data needed(be careful here; we set the `append` option to `false` on line 9).

In the last example below, we will add some content to a file by appending it at the end of any existing data. See here the `append` option to `true` on line 9.

```
1  use std::fs::OpenOptions;
2  use std::io::{Error, Write};
3  fn main() -> Result<(), Error> {
4      let file_name = "missy.txt";
5      let mut file = OpenOptions::new()
6          .read(true)
```

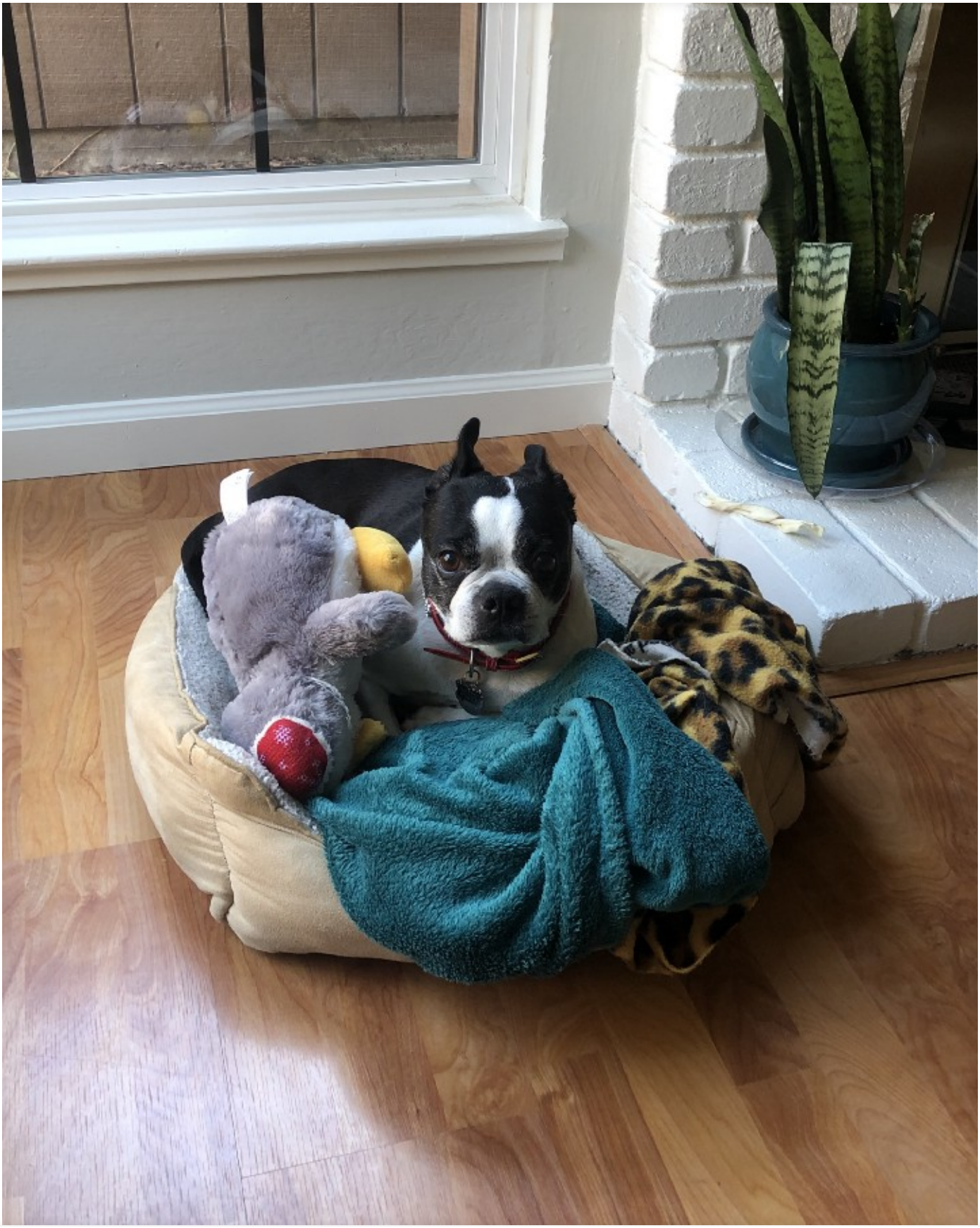

```
12      file.write_all(b"Good Day\n")?; // writing using the method 'write_all'
13      Ok(())
14  }
```


Append some content into a file in Rust

Like the writing process, Rust has a `BufWriter` struct, which provides an optimal way to write into a stream.

To Conclude

I hope I could give you a solid base for performing some file I/O operations in Rust. This is just the beginning. I encourage you to explore the documentation of the struct `File` and then deep dive into `BufReader` an `BufWriter` .

- More on `BufReaer` here.
- More on `BufWriter` here.

How to create a RESTful web service in Rust

It has been almost three months since my last article, and I have missed writing about Rust!!

anismousse.medium.com