Create a file:

```
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)
    }
}
```

Open a file:

```
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),
    }
}
```

Read the content of a file:

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

Another way:

```
use std::fs::File;
use std::io::{Error, Read, BufReader};
fn main() -> Result<(), Error> {
    let file_name = "missy.txt";
    let file = File::open(file_name)?;
    let mut reader = BufReader::new(file);
    let mut contents = String::new();
    reader.read_to_string(&mut contents)?;
    print!("{}", contents);
    Ok(())
}
```

Print file line by line:

```
use std::fs::File;
use std::io::{BufRead, BufReader};

fn main() {
    let filename = "missy.txt";
    // Open the file in read-only mode (ignoring errors).
    let file = File::open(filename).unwrap();
    let reader = BufReader::new(file);

    // Read the file line by line using the lines() iterator from std::io::BufRead.
    for (index, line) in reader.lines().enumerate() {
        let line = line.unwrap(); // Ignore errors.
        // Show the line and its number.
        println!("{}. {}", index + 1, line);
    }
}
```

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.

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

In our example, if the file **`missy.txt`** does not exist, we will create it before adding our content. If the file exists, our new content will replace its content.

Yes, we are using **`create`** once again. **`create`** allow us to create files and open existing ones in write-only mode.

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.

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

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.

```
use std::fs::OpenOptions;
use std::io::{Error, Write};
fn main() -> Result<(), Error> {
    let file_name = "missy.txt";
    let mut file = OpenOptions::new()
        .read(true)
        .write(true)
        .create(true)
        .append(true)
        .open(file_name)?;
    writeln!(file, "aHello my name is Missy!")?; // writing using the macro 'writeln!'
    file.write_all(b"Good Day\n")?; // writing using the method 'write_all'
    Ok(())
}
```

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