

Search Utility

In this programming assignment, you are expected to implement a command-line utility that searches for a specific string in one or multiple files, similar in spirit to the UNIX `grep` command. This assignment will help you practice command-line argument parsing, working with external crates as dependencies, and using file I/O in Rust. The assignment is due at **11:59 p.m. on Monday, October 20, 2025.**

Foundation: Basic Search

As a starting point, your command-line utility should accept a string, such as a simple string, and one file (possibly also including its path) as command-line arguments. It should then search for the given string in the specified file(s) and print the lines that contain the string. For example, if you run your utility with the command (where `target/debug/grep` is your compiled executable):

```
$ cargo build
$ target/debug/grep Utility tests/grep.md
```

In this example, assume that the file `tests/grep.md` contains:

```
## Search Utility
```

```
In this programming assignment, you are expected to implement a command-line utility that searches for a specific pattern in one or multiple files, similar in spirit to the `grep` command.
```

It should output:

```
## Search Utility
```

If the string to be searched is not found, the output should be empty.

Now that the basic functionality has been built, we should start adding more features to our utility. Here are some of the features you should implement:

1 Handling Multiple Files

Your utility should be able to handle multiple files, either in the form of wildcard characters (e.g., `*.txt`) or as a list of files. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests/*.md
```

It should search for the string `Utility` in all files with the `.md` extension in the `tests` directory.

Alternatively, you can run your utility with the command:

```
$ target/debug/grep Utility tests/grep.md ../README.md
```

This will search for the string `Utility` in the files `tests/grep.md` and `../README.md`.

2 Parsing Command-Line Arguments

Your utility should accept the following command-line arguments:

- `-i` : Case-insensitive search
- `-n` : Print line numbers along with matching lines
- `-v` : Invert match (print lines that do not match the pattern)
- `-r` : Recursive search (search in subdirectories)
- `-f` : Print filenames

- `-c` : Produce colored output
- `-h` or `--help` : Display help information

For example, if you run your utility with the command:

```
$ target/debug/grep -h
```

It should output a help message identical to the following:



There is an empty line below the first line `Usage: grep [OPTIONS]`
`<pattern> <files ... >`, as you can see from the released `a2-public-tests.tar.gz`. This empty line does not show correctly below due to artifacts of Markdown conversion to HTML.

```
Usage: grep [OPTIONS] <pattern> <files ... >
Options:
-i                Case-insensitive search
-n                Print line numbers
-v                Invert match (exclude lines that match the pattern)
-r                Recursive directory search
-f                Print filenames
-c                Enable colored output
-h, --help       Show help information
```

3 Implementing Command-Line Options

Case-insensitive search. Implement the `-i` option to perform a case-insensitive search. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests/grep.md -i
```

It should output:

```
## Search Utility
```

In this programming assignment, you are expected to implement a command-line

the search is now case-insensitive. Also, notice that the `-i` argument is specified after providing the string and the file, rather than before them. This is more flexible to use, which your solution should also support.

Printing line numbers. Implement the `-n` option to print line numbers along with matching lines. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests/grep.md -n
```

It should output:

```
1: ## Search Utility
```

Inverting match. Implement the `-v` option to invert the match, i.e., print lines that do not match the pattern. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests/grep.md -v
```

It should output:

In this programming assignment, you are expected to implement a command-line searches for a specific pattern in one or multiple files, similar in spirit `grep` command.

Recursive search. Implement the `-r` option to perform a recursive search in subdirectories. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests -r
```

It should search for the string `Utility` in all files in the `tests` directory and its subdirectories. You can use the `walkdir` crate to implement this feature.

For example, starting from the following directory structure:

```
└─ tests
  └─ grep.md
    └─ recursive
      └─ grep.md
```

If we run the command:

```
$ target/debug/grep Utility tests -r
```

It should output:

```
## Search Utility
## Search Utility
```

Printing file names. Implement the `-f` option to print the filenames along with the matched lines. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests -r -f
```

It should output:

```
tests/recursive/grep.md: ## Search Utility
tests/grep.md: ## Search Utility
```

Enabling coloured output. Implement the `-c` option to produce colored output for matched text. You can use the `colored` crate to implement this feature. For example, if you run your utility with the command:

```
$ target/debug/grep Utility tests -r -c -f
```

```
tests/recursive/grep.md: ## Search Utility  
tests/grep.md: ## Search Utility
```

where the matched text is coloured in red.

Implementation Notes

You may wish to use a `Config` struct to store search configuration after parsing the command-line arguments.

Use the external crates `colored` and `walkdir` to implement the `-c` and `-r` options, respectively. To use these crates, you should first add these crates to your `Cargo.toml` file.

There is no need to handle any errors in this assignment. You can assume that the user will provide the correct command-line arguments, and that there are no binary files that are not UTF-8 encoded in the directories you are searching. It is fine to panic at run-time if such errors occur.

When you start to work on your assignment, make sure you use the command `cargo new` to create a new project, including a `Cargo.toml` file and a `src/main.rs` file, as well as an initial git repository. Take advantage of the git repository that `cargo new` creates for you, and push your local commits to GitHub. Once your project is managed by git, you can then start writing your code in the `src/main.rs` file. Of course, you can also create additional files in the `src` directory if you need to. You can compile and run your code using the `cargo run` command, as we have introduced in our lectures.

Testing

Consider the following directory structure, mentioned previously:

📁 tests

📄 grep.md

📁 recursive

📄 grep.md

Test the following cases:

Getting help:

```
cargo run -- -h
```

Your solution should output (immediately after the messages printed by the Rust compiler):

```
Usage: grep [OPTIONS] <pattern> <files ...>
Options:
  -i          Case-insensitive search
  -n          Print line numbers
  -v          Invert match (exclude lines that match the pattern)
  -r          Recursive directory search
  -f          Print filenames
  -c          Enable colored output
  -h, --help  Show help information
```

Basic search

```
cargo run -- Utility tests/grep.md
```

Your solution should output (immediately after the messages printed by the Rust compiler):

```
## Search Utility
```

```
cargo run -- Utility tests/grep.md tests/recursive/grep.md
```

Your solution should output (immediately after the messages printed by the Rust compiler):

```
## Search Utility  
## Search Utility
```

Handling multiple files

```
$ cargo run -- Utility tests/*.md tests/recursive/*.md
```

The output of your solution should be:

```
## Search Utility  
## Search Utility
```

Processing command-line options

```
$ cargo run -- Utility tests/grep.md -i  
$ cargo run -- Utility tests/grep.md -n  
$ cargo run -- Utility tests/grep.md -v  
$ cargo run -- Utility tests -r  
$ cargo run -- Utility tests -r -f  
$ cargo run -- Utility tests -r -c -f
```

The output your solution is expected to produce in response to each of these commands has been shown previously.

For your convenience, an initial set of 10 test cases has been released to you as a compressed archive, named [a2-public-tests.tar.gz](#), and reflecting what you can find in this section. Run the command `tar zxvf a2-public-tests.tar.gz` to uncompress the

archive to a directory, and then move the content of the archive to your source code directory that you created using `cargo new`, so that `cargo run` works successfully. You can use these test cases to test your implementation against the expected output, and revise minor output differences as necessary. To run the test, you can use the following command:

```
bash input.sh > my_output.txt
```

To compare the output of your program with the expected output, you can use the `diff` command:

```
diff my_output.txt output.txt
```

Notes on AI Tools

This assignment is designed for manual work without AI assistance, and you should **not** use AI tools at all for this assignment. While we recognize that the habit of using AI may be so reflective and strong that — like social media — you cannot think without AI assistance, you need to learn coding in the Rust programming language and this learning experience needs to start from somewhere. Similar to the previous assignment on the Reversi board game, this assignment offers a sufficiently simple canvas for you to continue your learning experience and to expand your set of skills in Rust, and using AI defeats the purpose and completely changes the learning experience. Heed our advice: do **not** use AI and you will enjoy such “raw” coding experience much more.


Submission

Submit your project by submitting a `.tar.gz` archive of it using Quercus, under *Assignment 2*. Use the following command to create your `.tar.gz` archive:

```
cd /path/to/parent/directory
tar zcvf 1234567890.tar.gz project_directory
```

```
└─ grep
  └─ .git
  └─ .gitignore
  └─ Cargo.toml
  └─ src
    └─ main.rs
```

where `1234567890` is your student number, and `project_directory` is the name of the directory containing your project files, such as `grep`. This is also the project name you initially provided to the `cargo new` command. Make sure that your archive contains all the files necessary to build and run your project, including the `Cargo.toml` file and the `src` directory as shown above. Also, make sure that your project builds and runs correctly after extracted from the archive, using the `cargo run` command.

 The deadline for this assignment is **Monday, October 20, 2025, at 11:59pm** Eastern time, and late submissions will **not** be accepted.

Marking

Your project will be built and tested using the `cargo run --` command against a set of 20 test cases¹, including 10 public test cases that are released to you, and 10 more hidden test cases that we use internally for testing your solutions. The marking will be based on the correctness of your implementation in these test cases, by comparing the output of your code to the correct output.

Make sure that your code is well-commented, easy to read, and that you have followed the Rust naming conventions. You should also make sure that your code is free of warnings when compiled with the `cargo build` or the `cargo check` command.

1. `--` in this command implies that all subsequent arguments are passed to the program being run, rather than to `cargo`. [↩](#)

© 2025 Baochun Li. All rights reserved.