Cookin' with Rust

This *Rust Cookbook* is a collection of simple examples that demonstrate good practices to accomplish common programming tasks, using the crates of the Rust ecosystem.

Read more about *Rust Cookbook*, including tips for how to read the book, how to use the examples, and notes on conventions.

Contributing

This project is intended to be easy for new Rust programmers to contribute to, and an easy way to get involved with the Rust community. It needs and welcomes help. For details see CONTRIBUTING.md.

Algorithms

Recipe	Crates	Categories
Generate random numbers	rand v0.8.5	Science
Generate random numbers within a range	rand v0.8.5	Science
Generate random numbers with given distribution	rand v0.8.5 rand_distr v0.8.5	Science
Generate random values of a custom type	rand v0.8.5	Science
Create random passwords from a set of alphanumeric characters	rand v0.8.5	OS
Create random passwords from a set of user-defined characters	rand v0.8.5	OS
Sort a Vector of Integers	std 1.29.1	Science
Sort a Vector of Floats	std 1.29.1	Science
Sort a Vector of Structs	std 1.29.1	Science

Command Line

Recipe	Crates	Categories
Parse command line arguments	clap v4.3.11	Command line
ANSI Terminal	ansi_term v0.21.2	Command line

Compression

Recipe	Crates	Categories
Decompress a tarball	flate2 v1.0.26 tar v0.4.38	Compression

Recipe	Crates	Categories
Compress a directory into a tarball	flate2 v1.0.26 tar v0.4.38	Compression
Decompress a tarball while removing a prefix from the paths	flate2 v1.0.26 tar v0.4.38	Compression

Concurrency

Recipe	Crates	Categories
Spawn a short-lived thread	crossbeam v0.8.2	Concurrency
Create a parallel data pipeline	crossbeam v0.8.2	Concurrency
Pass data between two threads	crossbeam v0.8.2	Concurrency
Maintain global mutable state	lazy_static v1.4.0	Rust patterns
Calculate SHA1 sum of *.iso files concurrently	threadpool v1.8.1 walkdir v2.3.3 num_cpus v1.16.0 ring v0.17.0-alpha.11	Concurrency Filesystem
Draw fractal dispatching work to a thread pool	threadpool v1.8.1 num v0.4.0 num_cpus v1.16.0 image v0.24.6	Concurrency Science Rendering
Mutate the elements of an array in parallel	rayon v1.7.0	Concurrency
Test in parallel if any or all elements of a collection match a given predicate	rayon v1.7.0	Concurrency
Search items using given predicate in parallel	rayon v1.7.0	Concurrency
Sort a vector in parallel	rayon v1.7.0 rand v0.8.5	Concurrency
Map-reduce in parallel	rayon v1.7.0	Concurrency
Generate jpg thumbnails in parallel	rayon v1.7.0 glob v0.3.1 image v0.24.6	Concurrency Filesystem

Cryptography

Recipe	Crates	Categories
Calculate the SHA-256 digest of a file	ring v0.17.0-alpha.11 data-encoding v2.4.0	Cryptography
Sign and verify a message with an HMAC digest	ring v0.17.0-alpha.11	Cryptography
Salt and hash a password with PBKDF2	ring v0.17.0-alpha.11 data-encoding v2.4.0	Cryptography

Data Structures

Recipe	Crates	Categories
Define and operate on a type represented as a bitfield	bitflags v2.3.3	No std

Database

Recipe	Crates	Categories
Create a SQLite database	rusqlite v0.29.0	Database
Insert and Query data	rusqlite v0.29.0	Database
Create tables in a Postgres database	postgres v0.19.5	Database
Insert and Query data	postgres v0.19.5	Database
Aggregate data	postgres v0.19.5	Database

Date and Time

Recipe	Crates	Categories
Measure elapsed time	std 1.29.1	Time
Perform checked date and time calculations	chrono v0.4.26	Date and time
Convert a local time to another timezone	chrono v0.4.26	Date and time
Examine the date and time	chrono v0.4.26	Date and time
Convert date to UNIX timestamp and vice versa	chrono v0.4.26	Date and time
Display formatted date and time	chrono v0.4.26	Date and time
Parse string into DateTime struct	chrono v0.4.26	Date and time

Development Tools

Debugging

Recipe	Crates	Categories
Log a debug message to the console	log v0.4.19 env_logger v0.10.0	Debugging
Log an error message to the console	log v0.4.19 env_logger v0.10.0	Debugging
Log to stdout instead of stderr	log v0.4.19 env_logger v0.10.0	Debugging
Log messages with a custom logger	log v0.4.19	Debugging
Log to the Unix syslog	log v0.4.19 syslog v6.1.0	Debugging

Recipe	Crates	Categories
Enable log levels per module	log v0.4.19 env_logger v0.10.0	Debugging
Use a custom environment variable to set up logging	log v0.4.19 env_logger v0.10.0	Debugging
Include timestamp in log messages	log v0.4.19 env_logger v0.10.0 chrono v0.4.26	Debugging
Log messages to a custom location	log v0.4.19 log4rs v1.2.0	Debugging

Versioning

Recipe	Crates	Categories
Parse and increment a version string	semver v1.0.17	Config
Parse a complex version string	semver v1.0.17	Config
Check if given version is pre-release	semver v1.0.17	Config
Find the latest version satisfying given range	semver v1.0.17	Config
Check external command version for compatibility	semver v1.0.17	Text processing OS

Build Time

Recipe	Crates	Categories
Compile and link statically to a bundled C library	cc v1.0.79	Development tools
Compile and link statically to a bundled C++ library	cc v1.0.79	Development tools
Compile a C library while setting custom defines	cc v1.0.79	Development tools

Encoding

Recipe	Crates	Categories
Percent-encode a string	percent-encoding v2.3.0	Encoding
Encode a string as application/x-www-form-urlencoded	url v2.4.0	Encoding
Encode and decode hex	data-encoding v2.4.0	Encoding
Encode and decode base64	base64 v0.21.2	Encoding
Read CSV records	csv v1.2.2	Encoding
Read CSV records with different delimiter	csv v1.2.2	Encoding
Filter CSV records matching a predicate	csv v1.2.2	Encoding

Recipe	Crates	Categories
Handle invalid CSV data with Serde	csv v1.2.2 serde v1.0.171	Encoding
Serialize records to CSV	csv v1.2.2	Encoding
Serialize records to CSV using Serde	csv v1.2.2 serde v1.0.171	Encoding
Transform one column of a CSV file	csv v1.2.2 serde v1.0.171	Encoding
Serialize and deserialize unstructured JSON	serde_json v1.0.100	Encoding
Deserialize a TOML configuration file	toml v0.7.6	Encoding
Read and write integers in little-endian byte order	byteorder v1.4.3	Encoding

File System

Recipe	Crates	Categories
Read lines of strings from a file	std 1.29.1	Filesystem
Avoid writing and reading from a same file	same_file v1.0.6	Filesystem
Access a file randomly using a memory map	memmap v0.7.0	Filesystem
File names that have been modified in the last 24 hours	std 1.29.1	Filesystem OS
Find loops for a given path	same_file v1.0.6	Filesystem
Recursively find duplicate file names	walkdir v2.3.3	Filesystem
Recursively find all files with given predicate	walkdir v2.3.3	Filesystem
Traverse directories while skipping dotfiles	walkdir v2.3.3	Filesystem
Recursively calculate file sizes at given depth	walkdir v2.3.3	Filesystem
Find all png files recursively	glob v0.3.1	Filesystem
Find all files with given pattern ignoring filename case	glob v0.3.1	Filesystem

Hardware Support

Recipe	Crates	Categories
Check number of logical cpu cores	num_cpus v1.16.0	Hardware support

Memory Management

Recipe	Crates	Categories
Declare lazily evaluated constant	lazy_static v1.4.0	Caching Rust patterns

Networking

Recipe	Crates	Categories
Listen on unused port TCP/IP	std 1.29.1	Net

Operating System

Recipe	Crates	Categories
Run an external command and process stdout	regex v1.9.1	OS, Text processing
Run an external command passing it stdin and check for an error code	regex v1.9.1	OS, Text processing
Run piped external commands	std 1.29.1	OS
Redirect both stdout and stderr of child process to the same file	std 1.29.1	OS
Continuously process child process' outputs	std 1.29.1	OS, Text processing
Read environment variable	std 1.29.1	OS

Science

Mathematics

Recipe	Crates	Categories
Vector Norm	ndarray v0.15.6	Science
Adding matrices	ndarray v0.15.6	Science
Multiplying matrices	ndarray v0.15.6	Science
Multiply a scalar with a vector with a matrix	ndarray v0.15.6	Science
Invert matrix	nalgebra v0.32.3	Science
Calculating the side length of a triangle	std 1.29.1	Science
Verifying tan is equal to sin divided by cos	std 1.29.1	Science
Distance between two points on the Earth	std 1.29.1	Science
Creating complex numbers	num v0.4.0	Science
Adding complex numbers	num v0.4.0	Science
Mathematical functions on complex numbers	num v0.4.0	Science
Measures of central tendency	std 1.29.1	Science
Computing standard deviation	std 1.29.1	Science
Big integers	num v0.4.0	Science

Text Processing

Recipe	Crates	Categories
Collect Unicode Graphemes	unicode-segmentation v1.10.1	Encoding
Verify and extract login from an email address	regex v1.9.1 lazy_static v1.4.0	Text processing
Extract a list of unique #Hashtags from a text	regex v1.9.1 lazy_static v1.4.0	Text processing
Extract phone numbers from text	regex v1.9.1	Text processing
Filter a log file by matching multiple regular expressions	regex v1.9.1	Text processing
Replace all occurrences of one text pattern with another pattern.	regex v1.9.1 lazy_static v1.4.0	Text processing
Implement the `FromStr` trait for a custom `struct`	std 1.29.1	Text processing

Web Programming

Scraping Web Pages

Recipe	Crates	Categories
Extract all links from a webpage HTML	reqwest v0.11.18 select v0.6.0	Net
Check webpage for broken links	reqwest v0.11.18 select v0.6.0 url v2.4.0	Net
Extract all unique links from a MediaWiki markup	reqwest v0.11.18 regex v1.9.1	Net

Uniform Resource Locations (URL)

Recipe	Crates	Categories
Parse a URL from a string to a `Url` type	url v2.4.0	Net
Create a base URL by removing path segments	url v2.4.0	Net
Create new URLs from a base URL	url v2.4.0	Net
Extract the URL origin (scheme / host / port)	url v2.4.0	Net
Remove fragment identifiers and query pairs from a URL	url v2.4.0	Net

Media Types (MIME)

Recipe	Crates	Categories
Get MIME type from string	mime v1.2.2	Encoding
Get MIME type from filename	mime v1.2.2	Encoding
Parse the MIME type of a HTTP response	mime v1.2.2 reqwest v0.11.18	Net Encoding

Clients

Recipe	Crates	Categories
Make a HTTP GET request	reqwest v0.11.18	Net
Query the GitHub API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Check if an API resource exists	reqwest v0.11.18	Net
Create and delete Gist with GitHub API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Consume a paginated RESTful API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Download a file to a temporary directory	reqwest v0.11.18 tempdir v0.3.7	Net Filesystem
Make a partial download with HTTP range headers	reqwest v0.11.18	Net
POST a file to paste-rs	reqwest v0.11.18	Net

Web Authentication

Recipe	Crates	Categories
Basic Authentication	reqwest v0.11.18	Net

About "Cookin' with Rust"

Table of contents

Who this book is for

This cookbook is intended for new Rust programmers, so that they may quickly get an overview of the capabilities of the Rust crate ecosystem. It is also intended for experienced Rust programmers, who should find in the recipes an easy reminder of how to accomplish common tasks.

How to read this book

The cookbook index contains the full list of recipes, organized into a number of sections: "basics", "encoding", "concurrency", etc. The sections themselves are more or less ordered in progression, with later sections being more advanced, and occasionally building on concepts from earlier sections.

Within the index, each section contains a list of recipes. The recipes are simple statements of a task to accomplish, like "generate random numbers in a range"; and each recipe is tagged with badges indicating which *crates* they use, like rand v0.8.5, and which categories on crates.io those crates belong to, like Science.

New Rust programmers should be comfortable reading from the first section to the last, and doing so should give one a strong overview of the crate ecosystem. Click on the section header in the index, or in the sidebar to navigate to the page for that section of the book.

If you are simply looking for the solution to a simple task, the cookbook is today more difficult to navigate. The easiest way to find a specific recipe is to scan the index looking for the crates and categories one is interested in. From there, click on the name of the recipe to view it. This will improve in the future.

How to use the recipes

Recipes are designed to give you instant access to working code, along with a full explanation of what it is doing, and to guide you to further information.

All recipes in the cookbook are full, self contained programs, so that they may be copied directly into your own projects for experimentation. To do so follow the instructions below.

Consider this example for "generate random numbers within a range":

rand v0.8.5 Science

```
use rand::Rng;

fn main() {
    let mut rng = rand::thread_rng();
    println!("Random f64: {}", rng.gen::<f64>());
}
```

To work with it locally we can run the following commands to create a new cargo project, and change to that directory:

```
cargo new my-example --bin
cd my-example
```

Now, we also need to add the necessary crates to Cargo.toml, as indicated by the crate badges, in this case just "rand". To do so, we'll use the `cargo add` command, which is provided by the `cargo-edit` crate, which we need to install first:

```
cargo install cargo-edit
cargo add rand
```

Now you can replace `src/main.rs` with the full contents of the example and run it:

```
cargo run
```

The crate badges that accompany the examples link to the crates' full documentation on docs.rs, and is often the next documentation you should read after deciding which crate suites your purpose.

A note about error handling

Error handling in Rust is robust when done correctly, but in today's Rust it requires a fair bit of boilerplate. Because of this one often sees Rust examples filled with `unwrap` calls instead of proper error handling.

Since these recipes are intended to be reused as-is and encourage best practices, they set up error handling correctly when there are `Result` types involved.

The basic pattern we use is to have a `fn main() -> Result`.

The structure generally looks like:

```
use error_chain::error_chain;
use std::net::IpAddr;
use std::str;

error_chain! {
    foreign_links {
        Utf8(std::str::Utf8Error);
        AddrParse(std::net::AddrParseError);
    }
}

fn main() -> Result<()> {
    let bytes = b"2001:db8::1";

    // Bytes to string.
    let s = str::from_utf8(bytes)?;

    // String to IP address.
    let addr: IpAddr = s.parse()?;

    println!("{:?}", addr);
    Ok(())
}
```

This is using the `error_chain!` macro to define a custom `Error` and `Result` type, along with automatic conversions from two standard library error types. The automatic conversions make the `?` operator work.

For the sake of readability error handling boilerplate is hidden by default like below. In order to read full contents click on the "expand" () button located in the top right corner of the snippet.

```
use url::{Url, Position};

fn main() -> Result<()> {
    let parsed = Url::parse("https://httpbin.org/cookies/set?k2=v2&k1=v1")?;
    let cleaned: &str = &parsed[..Position::AfterPath];
    println!("cleaned: {}", cleaned);
    Ok(())
}
```

For more background on error handling in Rust, read this page of the Rust book and this blog post.

A note about crate representation

This cookbook is intended eventually to provide expansive coverage of the Rust crate ecosystem, but today is limited in scope while we get it bootstrapped and work on the presentation. Hopefully, starting from a small scope and slowly expanding will help the cookbook become a high-quality resource sooner, and allow it to maintain consistent quality levels as it grows.

At present the cookbook is focused on the standard library, and on "core", or "foundational", crates —those crates that make up the most common programming tasks, and that the rest of the ecosystem builds off of.

The cookbook is closely tied to the Rust Libz Blitz, a project to identify, and improve the quality of such crates, and so it largely defers crate selection to that project. Any crates that have already been evaluated as part of that process are in scope for the cookbook, as are crates that are pending evaluation.

Algorithms

Recipe	Crates	Categories
Generate random numbers	rand v0.8.5	Science
Generate random numbers within a range	rand v0.8.5	Science
Generate random numbers with given distribution	rand v0.8.5 rand_distr v0.8.5	Science
Generate random values of a custom type	rand v0.8.5	Science
Create random passwords from a set of alphanumeric characters	rand v0.8.5	OS
Create random passwords from a set of user-defined characters	rand v0.8.5	OS
Sort a Vector of Integers	std 1.29.1	Science
Sort a Vector of Floats	std 1.29.1	Science
Sort a Vector of Structs	std 1.29.1	Science

Generate Random Values

Generate random numbers

rand v0.8.5 Science

Generates random numbers with help of random-number generator `rand::Rng` obtained via `rand::thread_rng`. Each thread has an initialized generator. Integers are uniformly distributed over the range of the type, and floating point numbers are uniformly distributed from 0 up to but not including 1.

```
use rand::Rng;

fn main() {
    let mut rng = rand::thread_rng();

    let n1: u8 = rng.gen();
    let n2: u16 = rng.gen();
    println!("Random u8: {}", n1);
    println!("Random u16: {}", n2);
    println!("Random u32: {}", rng.gen::<u32>());
    println!("Random i32: {}", rng.gen::<i32>());
    println!("Random float: {}", rng.gen::<f64>());
}
```

Generate random numbers within a range

rand v0.8.5 Science

Generates a random value within half-open `[0, 10)` range (not including `10`) with `Rng::gen_range`.

```
use rand::Rng;

fn main() {
    let mut rng = rand::thread_rng();
    println!("Integer: {}", rng.gen_range(0..10));
    println!("Float: {}", rng.gen_range(0.0..10.0));
}
```

`Uniform` can obtain values with uniform distribution. This has the same effect, but may be faster when repeatedly generating numbers in the same range.

```
use rand::distributions::{Distribution, Uniform};

fn main() {
    let mut rng = rand::thread_rng();
    let die = Uniform::from(1..7);

    loop {
        let throw = die.sample(&mut rng);
        println!("Roll the die: {}", throw);
        if throw == 6 {
            break;
        }
    }
}
```

Generate random numbers with given distribution

rand_distr v0.8.5 Science

By default, random numbers in the `rand` crate have uniform distribution. The `rand_distr` crate provides other kinds of distributions. To use them, you instantiate a distribution, then sample from that distribution using `Distribution::sample` with help of a random-number generator `rand::Rng`.

The distributions available are documented here. An example using the `Normal` distribution is shown below.

```
use rand_distr::{Distribution, Normal, NormalError};
use rand::thread_rng;

fn main() -> Result<(), NormalError> {
    let mut rng = thread_rng();
    let normal = Normal::new(2.0, 3.0)?;
    let v = normal.sample(&mut rng);
    println!("{} is from a N(2, 9) distribution", v);
    Ok(())
}
```

Generate random values of a custom type

rand v0.8.5 Science

Randomly generates a tuple `(i32, bool, f64)` and variable of user defined type `Point`. Implements the `Distribution` trait on type Point for `Standard` in order to allow random generation.

```
use rand::Rng;
use rand::distributions::{Distribution, Standard};

#[derive(Debug)]
struct Point {
    x: i32,
    y: i32,
}

impl Distribution<Point> for Standard {
    fn sample<R: Rng + ?Sized>(&self, rng: &mut R) -> Point {
        let (rand_x, rand_y) = rng.gen();
        Point {
            x: rand_x,
            y: rand_y,
        }
    }
}

fn main() {
    let mut rng = rand::thread_rng();
    let rand_tuple = rng.gen::<(i32, bool, f64)>();
    let rand_point: Point = rng.gen();
    println!("Random tuple: {:?}", rand_tuple);
    println!("Random Point: {:?}", rand_point);
}
```

Create random passwords from a set of alphanumeric characters

rand v0.8.5 OS

Randomly generates a string of given length ASCII characters in the range `A-Z, a-z, 0-9`, with `Alphanumeric` sample.

```
use rand::{thread_rng, Rng};
use rand::distributions::Alphanumeric;

fn main() {
    let rand_string: String = thread_rng()
        .sample_iter(&Alphanumeric)
        .take(30)
        .map(char::from)
        .collect();

    println!("{}", rand_string);
}
```

Create random passwords from a set of user-defined characters

rand v0.8.5 OS

Randomly generates a string of given length ASCII characters with custom user-defined bytestring, with `gen_range`.

```
fn main() {
    use rand::Rng;
    const CHARSET: &[u8] = b"ABCDEFGHIJKLMNOPQRSTUVWXYZ\
                             abcdefghijklmnopqrstuvwxyz\
                             0123456789)(*&^%$#@!~";
    const PASSWORD_LEN: usize = 30;
    let mut rng = rand::thread_rng();

    let password: String = (0..PASSWORD_LEN)
        .map(|_| {
            let idx = rng.gen_range(0..CHARSET.len());
            CHARSET[idx] as char
        })
        .collect();

    println!("{:?}", password);
}
```

Sorting Vectors

Sort a Vector of Integers

std 1.29.1 Science

This example sorts a Vector of integers via `vec::sort`. Alternative would be to use `vec::sort_unstable` which can be faster, but does not preserve the order of equal elements.

```
fn main() {
    let mut vec = vec![1, 5, 10, 2, 15];

    vec.sort();

    assert_eq!(vec, vec![1, 2, 5, 10, 15]);
}
```

Sort a Vector of Floats

std 1.29.1 Science

A Vector of f32 or f64 can be sorted with `vec::sort_by` and `PartialOrd::partial_cmp`.

```
fn main() {
    let mut vec = vec![1.1, 1.15, 5.5, 1.123, 2.0];

    vec.sort_by(|a, b| a.partial_cmp(b).unwrap());

    assert_eq!(vec, vec![1.1, 1.123, 1.15, 2.0, 5.5]);
}
```

Sort a Vector of Structs

std 1.29.1 Science

Sorts a Vector of Person structs with properties `name` and `age` by its natural order (By name and age). In order to make Person sortable you need four traits `Eq`, `PartialEq`, `Ord` and `PartialOrd`. These traits can be simply derived. You can also provide a custom comparator function using a `vec:sort_by` method and sort only by age.

```
#[derive(Debug, Eq, Ord, PartialEq, PartialOrd)]
struct Person {
    name: String,
    age: u32
}

impl Person {
    pub fn new(name: String, age: u32) -> Self {
        Person {
            name,
            age
        }
    }
}

fn main() {
    let mut people = vec![
        Person::new("Zoe".to_string(), 25),
        Person::new("Al".to_string(), 60),
        Person::new("John".to_string(), 1),
    ];

    // Sort people by derived natural order (Name and age)
    people.sort();

    assert_eq!(
        people,
        vec![
            Person::new("Al".to_string(), 60),
            Person::new("John".to_string(), 1),
            Person::new("Zoe".to_string(), 25),
        ]);

    // Sort people by age
    people.sort_by(|a, b| b.age.cmp(&a.age));

    assert_eq!(
        people,
        vec![
            Person::new("Al".to_string(), 60),
            Person::new("Zoe".to_string(), 25),
            Person::new("John".to_string(), 1),
        ]);

}
```

Command Line

Recipe	Crates	Categories
Parse command line arguments	clap v4.3.11	Command line
ANSI Terminal	ansi_term v0.21.2	Command line

Clap basic

Parse command line arguments

clap v4.3.11 Command line

This application describes the structure of its command-line interface using `clap`'s builder style. The documentation gives two other possible ways to instantiate an application.

In the builder style, `with_name` is the unique identifier that `value_of` will use to retrieve the value passed. The `short` and `long` options control the flag the user will be expected to type; short flags look like `-f` and long flags look like `--file`.

```
use clap::{Arg, App};

fn main() {
    let matches = App::new("My Test Program")
        .version("0.1.0")
        .author("Hackerman Jones <hckrmnjones@hack.gov>")
        .about("Teaches argument parsing")
        .arg(Arg::with_name("file")
                 .short("f")
                 .long("file")
                 .takes_value(true)
                 .help("A cool file"))
        .arg(Arg::with_name("num")
                 .short("n")
                 .long("number")
                 .takes_value(true)
                 .help("Five less than your favorite number"))
        .get_matches();

    let myfile = matches.value_of("file").unwrap_or("input.txt");
    println!("The file passed is: {}", myfile);

    let num_str = matches.value_of("num");
    match num_str {
        None => println!("No idea what your favorite number is."),
        Some(s) => {
            match s.parse::<i32>() {
                Ok(n) => println!("Your favorite number must be {}.", n + 5),
                Err(_) => println!("That's not a number! {}", s),
            }
        }
    }
}
```

Usage information is generated by `clap`. The usage for the example application looks like this.

```
My Test Program 0.1.0
Hackerman Jones <hckrmnjones@hack.gov>
Teaches argument parsing

USAGE:
    testing [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -f, --file <file>     A cool file
    -n, --number <num>    Five less than your favorite number
```

We can test the application by running a command like the following.

```
$ cargo run -- -f myfile.txt -n 251
```

The output is:

```
The file passed is: myfile.txt
Your favorite number must be 256.
```

ANSI Terminal

ANSI Terminal

This program depicts the use of `ansi_term` crate and how it is used for controlling colours and formatting, such as blue bold text or yellow underlined text, on ANSI terminals.

There are two main data structures in `ansi_term`: `ANSIString` and `Style`. A `Style` holds stylistic information: colours, whether the text should be bold, or blinking, or whatever. There are also Colour variants that represent simple foreground colour styles. An `ANSIString` is a string paired with a `Style`.

Note: British English uses *Colour* instead of *Color*, don't get confused

Printing colored text to the Terminal

```
use ansi_term::Colour;

fn main() {
    println!("This is {} in color, {} in color and {} in color",
             Colour::Red.paint("red"),
             Colour::Blue.paint("blue"),
             Colour::Green.paint("green"));
}
```

Bold text in Terminal

For anything more complex than plain foreground colour changes, the code needs to construct `Style` struct. `Style::new()` creates the struct, and properties chained.

```
use ansi_term::Style;

fn main() {
    println!("{} and this is not",
             Style::new().bold().paint("This is Bold"));
}
```

Bold and colored text in terminal

`Colour` implements many similar functions as `Style` and can chain methods.

```
use ansi_term::Colour;
use ansi_term::Style;

fn main(){
    println!("{}, {} and {}",
             Colour::Yellow.paint("This is colored"),
             Style::new().bold().paint("this is bold"),
             Colour::Yellow.bold().paint("this is bold and colored"));
}
```

Compression

Recipe	Crates	Categories
Decompress a tarball	flate2 v1.0.26 tar v0.4.38	Compression
Compress a directory into a tarball	flate2 v1.0.26 tar v0.4.38	Compression
Decompress a tarball while removing a prefix from the paths	flate2 v1.0.26 tar v0.4.38	Compression

Working with Tarballs

Decompress a tarball

flate2 v1.0.26 tar v0.4.38 Compression

Decompress (`GzDecoder`) and extract (`Archive::unpack`) all files from a compressed tarball named `archive.tar.gz` located in the current working directory to the same location.

```
use std::fs::File;
use flate2::read::GzDecoder;
use tar::Archive;

fn main() -> Result<(), std::io::Error> {
    let path = "archive.tar.gz";

    let tar_gz = File::open(path)?;
    let tar = GzDecoder::new(tar_gz);
    let mut archive = Archive::new(tar);
    archive.unpack(".")?;

    Ok(())
}
```

Compress a directory into tarball

Compress `/var/log` directory into `archive.tar.gz`.

Creates a `File` wrapped in `GzEncoder` and `tar::Builder`.
Adds contents of `/var/log` directory recursively into the archive under `backup/logs` path with `Builder::append_dir_all`. `GzEncoder` is responsible for transparently compressing the data prior to writing it into `archive.tar.gz`.

```
use std::fs::File;
use flate2::Compression;
use flate2::write::GzEncoder;

fn main() -> Result<(), std::io::Error> {
    let tar_gz = File::create("archive.tar.gz")?;
    let enc = GzEncoder::new(tar_gz, Compression::default());
    let mut tar = tar::Builder::new(enc);
    tar.append_dir_all("backup/logs", "/var/log")?;
    Ok(())
}
```

Decompress a tarball while removing a prefix from the paths

Iterate over the `Archive::entries`. Use `Path::strip_prefix` to remove the specified path prefix (`bundle/logs`). Finally, extract the `tar::Entry` via `Entry::unpack`.

```rust
use std::fs::File;
use std::path::PathBuf;
use flate2::read::GzDecoder;
use tar::Archive;

fn main() -> Result<()> {
    let file = File::open("archive.tar.gz")?;
    let mut archive = Archive::new(GzDecoder::new(file));
    let prefix = "bundle/logs";

    println!("Extracted the following files:");
    archive
        .entries()?
        .filter_map(|e| e.ok())
        .map(|mut entry| -> Result<PathBuf> {
            let path = entry.path()?.strip_prefix(prefix)?.to_owned();
            entry.unpack(&path)?;
            Ok(path)
        })
        .filter_map(|e| e.ok())
        .for_each(|x| println!("> {}", x.display()));

    Ok(())
}
```

Concurrency

Recipe	Crates	Categories
Spawn a short-lived thread	crossbeam v0.8.2	Concurrency
Create a parallel data pipeline	crossbeam v0.8.2	Concurrency
Pass data between two threads	crossbeam v0.8.2	Concurrency
Maintain global mutable state	lazy_static v1.4.0	Rust patterns
Calculate SHA1 sum of *.iso files concurrently	threadpool v1.8.1 walkdir v2.3.3 num_cpus v1.16.0 ring v0.17.0-alpha.11	Concurrency Filesystem
Draw fractal dispatching work to a thread pool	threadpool v1.8.1 num v0.4.0 num_cpus v1.16.0 image v0.24.6	Concurrency Science Rendering
Mutate the elements of an array in parallel	rayon v1.7.0	Concurrency
Test in parallel if any or all elements of a collection match a given predicate	rayon v1.7.0	Concurrency
Search items using given predicate in parallel	rayon v1.7.0	Concurrency
Sort a vector in parallel	rayon v1.7.0 rand v0.8.5	Concurrency
Map-reduce in parallel	rayon v1.7.0	Concurrency
Generate jpg thumbnails in parallel	rayon v1.7.0 glob v0.3.1 image v0.24.6	Concurrency Filesystem

Threads

Spawn a short-lived thread

crossbeam v0.8.2 Concurrency

The example uses the crossbeam crate, which provides data structures and functions for concurrent and parallel programming. `Scope::spawn` spawns a new scoped thread that is guaranteed to terminate before returning from the closure that passed into `crossbeam::scope` function, meaning that you can reference data from the calling function.

This example splits the array in half and performs the work in separate threads.

```
fn main() {
    let arr = &[1, 25, -4, 10];
    let max = find_max(arr);
    assert_eq!(max, Some(25));
}

fn find_max(arr: &[i32]) -> Option<i32> {
    const THRESHOLD: usize = 2;

    if arr.len() <= THRESHOLD {
        return arr.iter().cloned().max();
    }

    let mid = arr.len() / 2;
    let (left, right) = arr.split_at(mid);

    crossbeam::scope(|s| {
        let thread_l = s.spawn(|_| find_max(left));
        let thread_r = s.spawn(|_| find_max(right));

        let max_l = thread_l.join().unwrap()?;
        let max_r = thread_r.join().unwrap()?;

        Some(max_l.max(max_r))
    }).unwrap()
}
```

Create a parallel pipeline

crossbeam v0.8.2 Concurrency

This example uses the crossbeam and crossbeam-channel crates to create a parallel pipeline, similar to that described in the ZeroMQ guide There is a data source and a data sink, with data being processed by two worker threads in parallel on its way from the source to the sink.

We use bounded channels with a capacity of one using `crossbeam_channel::bounded`. The producer must be on its own thread because it produces messages faster than the workers can process them (since they sleep for half a second) - this means the producer blocks on the call to `[crossbeam_channel::Sender::send]` for half a second until one of the workers processes the data in the channel. Also note that the data in the channel is consumed by whichever worker calls receive first, so each message is delivered to a single worker rather than both workers.

Reading from the channels via the iterator `crossbeam_channel::Receiver::iter` method will block, either waiting for new messages or until the channel is closed. Because the channels were created within the `crossbeam::scope`, we must manually close them via `drop` to prevent the entire program from blocking on the worker for-loops. You can think of the calls to `drop` as signaling that no more messages will be sent.

```
extern crate crossbeam;
extern crate crossbeam_channel;

use std::thread;
use std::time::Duration;
use crossbeam_channel::bounded;

fn main() {
    let (snd1, rcv1) = bounded(1);
    let (snd2, rcv2) = bounded(1);
    let n_msgs = 4;
    let n_workers = 2;

    crossbeam::scope(|s| {
        // Producer thread
        s.spawn(|_| {
            for i in 0..n_msgs {
                snd1.send(i).unwrap();
                println!("Source sent {}", i);
            }
            // Close the channel - this is necessary to exit
            // the for-loop in the worker
            drop(snd1);
        });

        // Parallel processing by 2 threads
        for _ in 0..n_workers {
            // Send to sink, receive from source
            let (sendr, recvr) = (snd2.clone(), rcv1.clone());
            // Spawn workers in separate threads
            s.spawn(move |_| {
            thread::sleep(Duration::from_millis(500));
                // Receive until channel closes
                for msg in recvr.iter() {
                    println!("Worker {:?} received {}.",
                             thread::current().id(), msg);
                    sendr.send(msg * 2).unwrap();
                }
            });
        }
        // Close the channel, otherwise sink will never
        // exit the for-loop
        drop(snd2);

        // Sink
        for msg in rcv2.iter() {
            println!("Sink received {}", msg);
        }
    }).unwrap();
}
```

Pass data between two threads

crossbeam v0.8.2 Concurrency

This example demonstrates the use of crossbeam-channel in a single producer, single consumer (SPSC) setting. We build off the ex-crossbeam-spawn example by using `crossbeam::scope` and `Scope::spawn` to manage the producer thread. Data is exchanged between the two threads using a `crossbeam_channel::unbounded` channel, meaning there is no limit to the number of storeable messages. The producer thread sleeps for half a second in between messages.

```
use std::{thread, time};
use crossbeam_channel::unbounded;

fn main() {
    let (snd, rcv) = unbounded();
    let n_msgs = 5;
    crossbeam::scope(|s| {
        s.spawn(|_| {
            for i in 0..n_msgs {
                snd.send(i).unwrap();
                thread::sleep(time::Duration::from_millis(100));
            }
        });
    }).unwrap();
    for _ in 0..n_msgs {
        let msg = rcv.recv().unwrap();
        println!("Received {}", msg);
    }
}
```

Maintain global mutable state

lazy_static v1.4.0 Rust patterns

Declare global state using lazy_static. lazy_static creates a globally available `static ref` which requires a `Mutex` to allow mutation (also see `RwLock`). The `Mutex` wrap ensures the state cannot be simultaneously accessed by multiple threads, preventing race conditions. A `MutexGuard` must be acquired to read or mutate the value stored in a `Mutex`.

```rust
use lazy_static::lazy_static;
use std::sync::Mutex;

lazy_static! {
    static ref FRUIT: Mutex<Vec<String>> = Mutex::new(Vec::new());
}

fn insert(fruit: &str) -> Result<()> {
    let mut db = FRUIT.lock().map_err(|_| "Failed to acquire MutexGuard")?;
    db.push(fruit.to_string());
    Ok(())
}

fn main() -> Result<()> {
    insert("apple")?;
    insert("orange")?;
    insert("peach")?;
    {
        let db = FRUIT.lock().map_err(|_| "Failed to acquire MutexGuard")?;

        db.iter().enumerate().for_each(|(i, item)| println!("{}: {}", i, item));
    }
    insert("grape")?;
    Ok(())
}
```

Calculate SHA256 sum of iso files concurrently

threadpool v1.8.1 num_cpus v1.16.0 walkdir v2.3.3 ring v0.17.0-alpha.11 Concurrency Filesystem

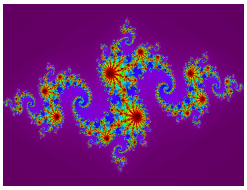
This example calculates the SHA256 for every file with iso extension in the current directory. A threadpool generates threads equal to the number of cores present in the system found with `num_cpus::get`. `Walkdir::new` iterates the current directory and calls `execute` to perform the operations of reading and computing SHA256 hash.

```
use walkdir::WalkDir;
use std::fs::File;
use std::io::{BufReader, Read, Error};
use std::path::Path;
use threadpool::ThreadPool;
use std::sync::mpsc::channel;
use ring::digest::{Context, Digest, SHA256};


fn compute_digest<P: AsRef<Path>>(filepath: P) -> Result<(Digest, P), Error> {
    let mut buf_reader = BufReader::new(File::open(&filepath)?);
    let mut context = Context::new(&SHA256);
    let mut buffer = [0; 1024];

    loop {
        let count = buf_reader.read(&mut buffer)?;
        if count == 0 {
            break;
        }
        context.update(&buffer[..count]);
    }

    Ok((context.finish(), filepath))
}

fn main() -> Result<(), Error> {
    let pool = ThreadPool::new(num_cpus::get());

    let (tx, rx) = channel();

    for entry in WalkDir::new("/home/user/Downloads")
        .follow_links(true)
        .into_iter()
        .filter_map(|e| e.ok())
        .filter(|e| !e.path().is_dir() && is_iso(e.path())) {
            let path = entry.path().to_owned();
            let tx = tx.clone();
            pool.execute(move || {
                let digest = compute_digest(path);
                tx.send(digest).expect("Could not send data!");
            });
        }

    drop(tx);
    for t in rx.iter() {
        let (sha, path) = t?;
        println!("{:?} {:?}", sha, path);
    }
    Ok(())
}
```

Draw fractal dispatching work to a thread pool

threadpool v1.8.1 num v0.4.0 num_cpus v1.16.0 image v0.24.6 Concurrency Science Rendering

This example generates an image by drawing a fractal from the Julia set with a thread pool for distributed computation.



Allocate memory for output image of given width and height with `ImageBuffer::new`. `Rgb::from_channels` calculates RGB pixel values. Create `ThreadPool` with thread count equal to number of cores with `num_cpus::get`. `ThreadPool::execute` receives each pixel as a separate job.

`mpsc::channel` receives the jobs and `Receiver::recv` retrieves them. `ImageBuffer::put_pixel` uses the data to set the pixel color. `ImageBuffer::save` writes the image to `output.png`.

```
use std::sync::mpsc::{channel, RecvError};
use threadpool::ThreadPool;
use num::complex::Complex;
use image::{ImageBuffer, Pixel, Rgb};

fn main() -> Result<()> {
    let (width, height) = (1920, 1080);
    let mut img = ImageBuffer::new(width, height);
    let iterations = 300;

    let c = Complex::new(-0.8, 0.156);

    let pool = ThreadPool::new(num_cpus::get());
    let (tx, rx) = channel();

    for y in 0..height {
        let tx = tx.clone();
        pool.execute(move || for x in 0..width {
                        let i = julia(c, x, y, width, height, iterations);
                        let pixel = wavelength_to_rgb(380 + i * 400 / iterations);
                        tx.send((x, y, pixel)).expect("Could not send data!");
                    });
    }

    for _ in 0..(width * height) {
        let (x, y, pixel) = rx.recv()?;
        img.put_pixel(x, y, pixel);
    }
    let _ = img.save("output.png")?;
    Ok(())
}
```

Parallel Tasks

Mutate the elements of an array in parallel

rayon v1.7.0 Concurrency

The example uses the `rayon` crate, which is a data parallelism library for Rust. `rayon` provides the `par_iter_mut` method for any parallel iterable data type. This is an iterator-like chain that potentially executes in parallel.

```
use rayon::prelude::*;

fn main() {
    let mut arr = [0, 7, 9, 11];
    arr.par_iter_mut().for_each(|p| *p -= 1);
    println!("{:?}", arr);
}
```

Test in parallel if any or all elements of a collection match a given predicate

rayon v1.7.0 Concurrency

This example demonstrates using the `rayon::any` and `rayon::all` methods, which are parallelized counterparts to `std::any` and `std::all`. `rayon::any` checks in parallel whether any element of the iterator matches the predicate, and returns as soon as one is found. `rayon::all` checks in parallel whether all elements of the iterator match the predicate, and returns as soon as a non-matching element is found.

```
use rayon::prelude::*;

fn main() {
    let mut vec = vec![2, 4, 6, 8];

    assert!(!vec.par_iter().any(|n| (*n % 2) != 0));
    assert!(vec.par_iter().all(|n| (*n % 2) == 0));
    assert!(!vec.par_iter().any(|n| *n > 8 ));
    assert!(vec.par_iter().all(|n| *n <= 8 ));

    vec.push(9);

    assert!(vec.par_iter().any(|n| (*n % 2) != 0));
    assert!(!vec.par_iter().all(|n| (*n % 2) == 0));
    assert!(vec.par_iter().any(|n| *n > 8 ));
    assert!(!vec.par_iter().all(|n| *n <= 8 ));
}
```

Search items using given predicate in parallel

rayon v1.7.0 Concurrency

This example uses `rayon::find_any` and `par_iter` to search a vector in parallel for an element satisfying the predicate in the given closure.

If there are multiple elements satisfying the predicate defined in the closure argument of `rayon::find_any`, `rayon` returns the first one found, not necessarily the first one.

Also note that the argument to the closure is a reference to a reference (`&&x`). See the discussion on `std::find` for additional details.

```
use rayon::prelude::*;

fn main() {
    let v = vec![6, 2, 1, 9, 3, 8, 11];

    let f1 = v.par_iter().find_any(|&&x| x == 9);
    let f2 = v.par_iter().find_any(|&&x| x % 2 == 0 && x > 6);
    let f3 = v.par_iter().find_any(|&&x| x > 8);

    assert_eq!(f1, Some(&9));
    assert_eq!(f2, Some(&8));
    assert!(f3 > Some(&8));
}
```

Sort a vector in parallel

rayon v1.7.0 rand v0.8.5 Concurrency

This example will sort in parallel a vector of Strings.

Allocate a vector of empty Strings. `par_iter_mut().for_each` populates random values in parallel. Although multiple options exist to sort an enumerable data type, `par_sort_unstable` is usually faster than stable sorting algorithms.

```
use rand::{Rng, thread_rng};
use rand::distributions::Alphanumeric;
use rayon::prelude::*;

fn main() {
  let mut vec = vec![String::new(); 100_000];
  vec.par_iter_mut().for_each(|p| {
    let mut rng = thread_rng();
    *p = (0..5).map(|_| rng.sample(&Alphanumeric)).collect()
  });
  vec.par_sort_unstable();
}
```

Map-reduce in parallel

rayon v1.7.0 Concurrency

This example uses `rayon::filter`, `rayon::map`, and `rayon::reduce` to calculate the average age of `Person` objects whose age is over 30.

`rayon::filter` returns elements from a collection that satisfy the given predicate. `rayon::map` performs an operation on every element, creating a new iteration, and `rayon::reduce` performs an operation given the previous reduction and the current element. Also shows use of `rayon::sum`, which has the same result as the reduce operation in this example.

```
use rayon::prelude::*;

struct Person {
    age: u32,
}

fn main() {
    let v: Vec<Person> = vec![
        Person { age: 23 },
        Person { age: 19 },
        Person { age: 42 },
        Person { age: 17 },
        Person { age: 17 },
        Person { age: 31 },
        Person { age: 30 },
    ];

    let num_over_30 = v.par_iter().filter(|&x| x.age > 30).count() as f32;
    let sum_over_30 = v.par_iter()
        .map(|x| x.age)
        .filter(|&x| x > 30)
        .reduce(|| 0, |x, y| x + y);

    let alt_sum_30: u32 = v.par_iter()
        .map(|x| x.age)
        .filter(|&x| x > 30)
        .sum();

    let avg_over_30 = sum_over_30 as f32 / num_over_30;
    let alt_avg_over_30 = alt_sum_30 as f32/ num_over_30;

    assert!((avg_over_30 - alt_avg_over_30).abs() < std::f32::EPSILON);
    println!("The average age of people older than 30 is {}", avg_over_30);
}
```

Generate jpg thumbnails in parallel

rayon v1.7.0 glob v0.3.1 image v0.24.6 Concurrency Filesystem

This example generates thumbnails for all .jpg files in the current directory then saves them in a new folder called `thumbnails`.

`glob::glob_with` finds jpeg files in current directory. `rayon` resizes images in parallel using `par_iter` calling `DynamicImage::resize`.

```
use std::path::Path;
use std::fs::create_dir_all;

use glob::{glob_with, MatchOptions};
use image::{FilterType, ImageError};
use rayon::prelude::*;


fn main() -> Result<()> {
    let options: MatchOptions = Default::default();
    let files: Vec<_> = glob_with("*.jpg", options)?
        .filter_map(|x| x.ok())
        .collect();

    if files.len() == 0 {
        error_chain::bail!("No .jpg files found in current directory");
    }

    let thumb_dir = "thumbnails";
    create_dir_all(thumb_dir)?;

    println!("Saving {} thumbnails into '{}'...", files.len(), thumb_dir);

    let image_failures: Vec<_> = files
        .par_iter()
        .map(|path| {
            make_thumbnail(path, thumb_dir, 300)
                .map_err(|e| e.chain_err(|| path.display().to_string()))
        })
        .filter_map(|x| x.err())
        .collect();

    image_failures.iter().for_each(|x| println!("{}", x.display_chain()));

    println!("{} thumbnails saved successfully", files.len() - image_failures.len());
    Ok(())
}

fn make_thumbnail<PA, PB>(original: PA, thumb_dir: PB, longest_edge: u32) -> Result<()>
where
    PA: AsRef<Path>,
    PB: AsRef<Path>,
{
    let img = image::open(original.as_ref())?;
    let file_path = thumb_dir.as_ref().join(original);

    Ok(img.resize(longest_edge, longest_edge, FilterType::Nearest)
        .save(file_path)?)
}
```

Cryptography

Recipe	Crates	Categories
Calculate the SHA-256 digest of a file	ring v0.17.0-alpha.11 data-encoding v2.4.0	Cryptography
Sign and verify a message with an HMAC digest	ring v0.17.0-alpha.11	Cryptography
Salt and hash a password with PBKDF2	ring v0.17.0-alpha.11 data-encoding v2.4.0	Cryptography

Hashing

Calculate the SHA-256 digest of a file

ring v0.17.0-alpha.11 data-encoding v2.4.0 Cryptography

Writes some data to a file, then calculates the SHA-256 `digest::Digest` of the file's contents using `digest::Context`.

```
use data_encoding::HEXUPPER;
use ring::digest::{Context, Digest, SHA256};
use std::fs::File;
use std::io::{BufReader, Read, Write};

fn sha256_digest<R: Read>(mut reader: R) -> Result<Digest> {
    let mut context = Context::new(&SHA256);
    let mut buffer = [0; 1024];

    loop {
        let count = reader.read(&mut buffer)?;
        if count == 0 {
            break;
        }
        context.update(&buffer[..count]);
    }

    Ok(context.finish())
}

fn main() -> Result<()> {
    let path = "file.txt";

    let mut output = File::create(path)?;
    write!(output, "We will generate a digest of this text")?;

    let input = File::open(path)?;
    let reader = BufReader::new(input);
    let digest = sha256_digest(reader)?;

    println!("SHA-256 digest is {}", HEXUPPER.encode(digest.as_ref()));

    Ok(())
}
```

Sign and verify a message with HMAC digest

ring v0.17.0-alpha.11 Cryptography

Uses `ring::hmac` to creates a `hmac::Signature` of a string then verifies the signature is correct.

```
use ring::{hmac, rand};
use ring::rand::SecureRandom;
use ring::error::Unspecified;

fn main() -> Result<(), Unspecified> {
    let mut key_value = [0u8; 48];
    let rng = rand::SystemRandom::new();
    rng.fill(&mut key_value)?;
    let key = hmac::Key::new(hmac::HMAC_SHA256, &key_value);

    let message = "Legitimate and important message.";
    let signature = hmac::sign(&key, message.as_bytes());
    hmac::verify(&key, message.as_bytes(), signature.as_ref())?;

    Ok(())
}
```

Encryption

Salt and hash a password with PBKDF2

ring v0.17.0-alpha.11 data-encoding v2.4.0 Cryptography

Uses `ring::pbkdf2` to hash a salted password using the PBKDF2 key derivation function `pbkdf2::derive`. Verifies the hash is correct with `pbkdf2::verify`. The salt is generated using `SecureRandom::fill`, which fills the salt byte array with securely generated random numbers.

```rust
use data_encoding::HEXUPPER;
use ring::error::Unspecified;
use ring::rand::SecureRandom;
use ring::{digest, pbkdf2, rand};
use std::num::NonZeroU32;

fn main() -> Result<(), Unspecified> {
    const CREDENTIAL_LEN: usize = digest::SHA512_OUTPUT_LEN;
    let n_iter = NonZeroU32::new(100_000).unwrap();
    let rng = rand::SystemRandom::new();

    let mut salt = [0u8; CREDENTIAL_LEN];
    rng.fill(&mut salt)?;

    let password = "Guess Me If You Can!";
    let mut pbkdf2_hash = [0u8; CREDENTIAL_LEN];
    pbkdf2::derive(
        pbkdf2::PBKDF2_HMAC_SHA512,
        n_iter,
        &salt,
        password.as_bytes(),
        &mut pbkdf2_hash,
    );
    println!("Salt: {}", HEXUPPER.encode(&salt));
    println!("PBKDF2 hash: {}", HEXUPPER.encode(&pbkdf2_hash));

    let should_succeed = pbkdf2::verify(
        pbkdf2::PBKDF2_HMAC_SHA512,
        n_iter,
        &salt,
        password.as_bytes(),
        &pbkdf2_hash,
    );
    let wrong_password = "Definitely not the correct password";
    let should_fail = pbkdf2::verify(
        pbkdf2::PBKDF2_HMAC_SHA512,
        n_iter,
        &salt,
        wrong_password.as_bytes(),
        &pbkdf2_hash,
    );

    assert!(should_succeed.is_ok());
    assert!(!should_fail.is_ok());

    Ok(())
}
```

Data Structures

Recipe	Crates	Categories
Define and operate on a type represented as a bitfield	bitflags v2.3.3	No std

Custom

Define and operate on a type represented as a bitfield

bitflags v2.3.3 No std

Creates type safe bitfield type `MyFlags` with help of `bitflags!` macro and implements elementary `clear` operation as well as `Display` trait for it. Subsequently, shows basic bitwise operations and formatting.

```
use bitflags::bitflags;
use std::fmt;

bitflags! {
    struct MyFlags: u32 {
        const FLAG_A       = 0b00000001;
        const FLAG_B       = 0b00000010;
        const FLAG_C       = 0b00000100;
        const FLAG_ABC     = Self::FLAG_A.bits
                           | Self::FLAG_B.bits
                           | Self::FLAG_C.bits;
    }
}

impl MyFlags {
    pub fn clear(&mut self) -> &mut MyFlags {
        self.bits = 0;
        self
    }
}

impl fmt::Display for MyFlags {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "{:032b}", self.bits)
    }
}

fn main() {
    let e1 = MyFlags::FLAG_A | MyFlags::FLAG_C;
    let e2 = MyFlags::FLAG_B | MyFlags::FLAG_C;
    assert_eq!((e1 | e2), MyFlags::FLAG_ABC);
    assert_eq!((e1 & e2), MyFlags::FLAG_C);
    assert_eq!((e1 - e2), MyFlags::FLAG_A);
    assert_eq!(!e2, MyFlags::FLAG_A);

    let mut flags = MyFlags::FLAG_ABC;
    assert_eq!(format!("{}", flags), "00000000000000000000000000000111");
    assert_eq!(format!("{}", flags.clear()), "00000000000000000000000000000000");
    assert_eq!(format!("{:?}", MyFlags::FLAG_B), "FLAG_B");
    assert_eq!(format!("{:?}", MyFlags::FLAG_A | MyFlags::FLAG_B), "FLAG_A | FLAG_B");
}
```

Database

Recipe	Crates	Categories
Create a SQLite database	rusqlite v0.29.0	Database
Insert and Query data	rusqlite v0.29.0	Database

Recipe	Crates	Categories
Create tables in a Postgres database	postgres v0.19.5	Database
Insert and Query data	postgres v0.19.5	Database
Aggregate data	postgres v0.19.5	Database

SQLite

Create a SQLite database

rusqlite v0.29.0 Database

Use the `rusqlite` crate to open SQLite databases. See crate for compiling on Windows.

`Connection::open` will create the database if it doesn't already exist.

```
use rusqlite::{Connection, Result};
use rusqlite::NO_PARAMS;

fn main() -> Result<()> {
    let conn = Connection::open("cats.db")?;

    conn.execute(
        "create table if not exists cat_colors (
             id integer primary key,
             name text not null unique
         )",
        NO_PARAMS,
    )?;
    conn.execute(
        "create table if not exists cats (
             id integer primary key,
             name text not null,
             color_id integer not null references cat_colors(id)
         )",
        NO_PARAMS,
    )?;

    Ok(())
}
```

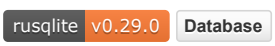
Insert and Select data

rusqlite v0.29.0 Database

`Connection::open` will open the database `cats` created in the earlier recipe. This recipe inserts data into `cat_colors` and `cats` tables using the `execute` method of `Connection`. First, the data is inserted into the `cat_colors` table. After a record for a color is inserted, `last_insert_rowid` method of `Connection` is used to get `id` of the last color inserted. This `id` is used while inserting data into the `cats` table. Then, the select query is prepared using the `prepare` method which gives a `statement` struct. Then, query is executed using `query_map` method of `statement`.

```
use rusqlite::NO_PARAMS;
use rusqlite::{Connection, Result};
use std::collections::HashMap;

#[derive(Debug)]
struct Cat {
    name: String,
    color: String,
}

fn main() -> Result<()> {
    let conn = Connection::open("cats.db")?;

    let mut cat_colors = HashMap::new();
    cat_colors.insert(String::from("Blue"), vec!["Tigger", "Sammy"]);
    cat_colors.insert(String::from("Black"), vec!["Oreo", "Biscuit"]);

    for (color, catnames) in &cat_colors {
        conn.execute(
            "INSERT INTO cat_colors (name) values (?1)",
            &[&color.to_string()],
        )?;
        let last_id: String = conn.last_insert_rowid().to_string();

        for cat in catnames {
            conn.execute(
                "INSERT INTO cats (name, color_id) values (?1, ?2)",
                &[&cat.to_string(), &last_id],
            )?;
        }
    }
    let mut stmt = conn.prepare(
        "SELECT c.name, cc.name from cats c
         INNER JOIN cat_colors cc
         ON cc.id = c.color_id;",
    )?;

    let cats = stmt.query_map(NO_PARAMS, |row| {
        Ok(Cat {
            name: row.get(0)?,
            color: row.get(1)?,
        })
    })?;

    for cat in cats {
        println!("Found cat {:?}", cat);
    }

    Ok(())
}
```

Using transactions



`Connection::open` will open the `cats.db` database from the top recipe.

Begin a transaction with `Connection::transaction`. Transactions will roll back unless committed explicitly with `Transaction::commit`.

In the following example, colors add to a table having a unique constraint on the color name. When an attempt to insert a duplicate color is made, the transaction rolls back.

```
use rusqlite::{Connection, Result, NO_PARAMS};

fn main() -> Result<()> {
    let mut conn = Connection::open("cats.db")?;

    successful_tx(&mut conn)?;

    let res = rolled_back_tx(&mut conn);
    assert!(res.is_err());

    Ok(())
}

fn successful_tx(conn: &mut Connection) -> Result<()> {
    let tx = conn.transaction()?;

    tx.execute("delete from cat_colors", NO_PARAMS)?;
    tx.execute("insert into cat_colors (name) values (?1)", &[&"lavender"])?;
    tx.execute("insert into cat_colors (name) values (?1)", &[&"blue"])?;

    tx.commit()
}

fn rolled_back_tx(conn: &mut Connection) -> Result<()> {
    let tx = conn.transaction()?;

    tx.execute("delete from cat_colors", NO_PARAMS)?;
    tx.execute("insert into cat_colors (name) values (?1)", &[&"lavender"])?;
    tx.execute("insert into cat_colors (name) values (?1)", &[&"blue"])?;
    tx.execute("insert into cat_colors (name) values (?1)", &[&"lavender"])?;

    tx.commit()
}
```

Working with Postgres

Create tables in a Postgres database

Use the `postgres` crate to create tables in a Postgres database.

`Client::connect` helps in connecting to an existing database. The recipe uses a URL string format with `Client::connect`. It assumes an existing database named `library`, the username is `postgres` and the password is `postgres`.

```
use postgres::{Client, NoTls, Error};

fn main() -> Result<(), Error> {
    let mut client =
Client::connect("postgresql://postgres:postgres@localhost/library", NoTls)?;

    client.batch_execute("
        CREATE TABLE IF NOT EXISTS author (
            id              SERIAL PRIMARY KEY,
            name            VARCHAR NOT NULL,
            country         VARCHAR NOT NULL
            )
    ")?;

    client.batch_execute("
        CREATE TABLE IF NOT EXISTS book  (
            id              SERIAL PRIMARY KEY,
            title           VARCHAR NOT NULL,
            author_id       INTEGER NOT NULL REFERENCES author
            )
    ")?;

    Ok(())

}
```

Insert and Query data

The recipe inserts data into the `author` table using `execute` method of `Client`. Then, displays the data from the `author` table using `query` method of `Client`.

```
use postgres::{Client, NoTls, Error};
use std::collections::HashMap;

struct Author {
    _id: i32,
    name: String,
    country: String
}

fn main() -> Result<(), Error> {
    let mut client =
Client::connect("postgresql://postgres:postgres@localhost/library",
                                    NoTls)?;

    let mut authors = HashMap::new();
    authors.insert(String::from("Chinua Achebe"), "Nigeria");
    authors.insert(String::from("Rabindranath Tagore"), "India");
    authors.insert(String::from("Anita Nair"), "India");

    for (key, value) in &authors {
        let author = Author {
            _id: 0,
            name: key.to_string(),
            country: value.to_string()
        };

        client.execute(
                "INSERT INTO author (name, country) VALUES ($1, $2)",
                &[&author.name, &author.country],
        )?;
    }

    for row in client.query("SELECT id, name, country FROM author", &[])? {
        let author = Author {
            _id: row.get(0),
            name: row.get(1),
            country: row.get(2),
        };
        println!("Author {} is from {}", author.name, author.country);
    }

    Ok(())

}
```

Aggregate data

This recipe lists the nationalities of the first 7999 artists in the database of the `Museum of Modern Art` in descending order.

```rust
use postgres::{Client, Error, NoTls};

struct Nation {
    nationality: String,
    count: i64,
}

fn main() -> Result<(), Error> {
    let mut client = Client::connect(
        "postgresql://postgres:postgres@127.0.0.1/moma",
        NoTls,
    )?;

    for row in client.query
    ("SELECT nationality, COUNT(nationality) AS count
    FROM artists GROUP BY nationality ORDER BY count DESC", &[])? {

        let (nationality, count) : (Option<String>, Option<i64>)
        = (row.get (0), row.get (1));

        if nationality.is_some () && count.is_some () {

            let nation = Nation{
                nationality: nationality.unwrap(),
                count: count.unwrap(),
        };
            println!("{} {}", nation.nationality, nation.count);

        }
    }

    Ok(())
}
```

Date and Time

Recipe	Crates	Categories
Measure elapsed time	std 1.29.1	Time
Perform checked date and time calculations	chrono v0.4.26	Date and time
Convert a local time to another timezone	chrono v0.4.26	Date and time
Examine the date and time	chrono v0.4.26	Date and time
Convert date to UNIX timestamp and vice versa	chrono v0.4.26	Date and time
Display formatted date and time	chrono v0.4.26	Date and time
Parse string into DateTime struct	chrono v0.4.26	Date and time

Duration and Calculation

Measure the elapsed time between two code sections

std 1.29.1 Time

Measures `time::Instant::elapsed` since `time::Instant::now`.

Calling `time::Instant::elapsed` returns a `time::Duration` that we print at the end of the example. This method will not mutate or reset the `time::Instant` object.

```
use std::time::{Duration, Instant};

fn main() {
    let start = Instant::now();
    expensive_function();
    let duration = start.elapsed();

    println!("Time elapsed in expensive_function() is: {:?}", duration);
}
```

Perform checked date and time calculations

chrono v0.4.26 Date and time

Calculates and displays the date and time two weeks from now using `DateTime::checked_add_signed` and the date of the day before that using `DateTime::checked_sub_signed`. The methods return None if the date and time cannot be calculated.

Escape sequences that are available for the `DateTime::format` can be found at `chrono::format::strftime`.

```
use chrono::{DateTime, Duration, Utc};

fn day_earlier(date_time: DateTime<Utc>) -> Option<DateTime<Utc>> {
    date_time.checked_sub_signed(Duration::days(1))
}

fn main() {
    let now = Utc::now();
    println!("{}", now);

    let almost_three_weeks_from_now = now.checked_add_signed(Duration::weeks(2))
            .and_then(|in_2weeks| in_2weeks.checked_add_signed(Duration::weeks(1)))
            .and_then(day_earlier);

    match almost_three_weeks_from_now {
        Some(x) => println!("{}", x),
        None => eprintln!("Almost three weeks from now overflows!"),
    }

    match now.checked_add_signed(Duration::max_value()) {
        Some(x) => println!("{}", x),
        None => eprintln!("We can't use chrono to tell the time for the Solar System to
complete more than one full orbit around the galactic center."),
    }
}
```

Convert a local time to another timezone

chrono v0.4.26 Date and time

Gets the local time and displays it using `offset::Local::now` and then converts it to the UTC standard using the `DateTime::from_utc` struct method. A time is then converted using the `offset::FixedOffset` struct and the UTC time is then converted to UTC+8 and UTC-2.

```
use chrono::{DateTime, FixedOffset, Local, Utc};

fn main() {
    let local_time = Local::now();
    let utc_time = DateTime::<Utc>::from_utc(local_time.naive_utc(), Utc);
    let china_timezone = FixedOffset::east(8 * 3600);
    let rio_timezone = FixedOffset::west(2 * 3600);
    println!("Local time now is {}", local_time);
    println!("UTC time now is {}", utc_time);
    println!(
        "Time in Hong Kong now is {}",
        utc_time.with_timezone(&china_timezone)
    );
    println!("Time in Rio de Janeiro now is {}",
utc_time.with_timezone(&rio_timezone));
}
```

Parsing and Displaying

Examine the date and time

chrono v0.4.26 Date and time

Gets the current UTC `DateTime` and its hour/minute/second via `Timelike` and its year/month/day/weekday via `Datelike`.

```
use chrono::{Datelike, Timelike, Utc};

fn main() {
    let now = Utc::now();

    let (is_pm, hour) = now.hour12();
    println!(
        "The current UTC time is {:02}:{:02}:{:02} {}",
        hour,
        now.minute(),
        now.second(),
        if is_pm { "PM" } else { "AM" }
    );
    println!(
        "And there have been {} seconds since midnight",
        now.num_seconds_from_midnight()
    );

    let (is_common_era, year) = now.year_ce();
    println!(
        "The current UTC date is {}-{:02}-{:02} {:?} ({})",
        year,
        now.month(),
        now.day(),
        now.weekday(),
        if is_common_era { "CE" } else { "BCE" }
    );
    println!(
        "And the Common Era began {} days ago",
        now.num_days_from_ce()
    );
}
```

Convert date to UNIX timestamp and vice versa

chrono v0.4.26 Date and time

Converts a date given by `NaiveDate::from_ymd` and `NaiveTime::from_hms` to UNIX timestamp using `NaiveDateTime::timestamp`. Then it calculates what was the date after one billion seconds since January 1, 1970 0:00:00 UTC, using `NaiveDateTime::from_timestamp`.

```
use chrono::{NaiveDate, NaiveDateTime};

fn main() {
    let date_time: NaiveDateTime = NaiveDate::from_ymd(2017, 11, 12).and_hms(17, 33,
44);
    println!(
        "Number of seconds between 1970-01-01 00:00:00 and {} is {}.",
        date_time, date_time.timestamp());

    let date_time_after_a_billion_seconds =
NaiveDateTime::from_timestamp(1_000_000_000, 0);
    println!(
        "Date after a billion seconds since 1970-01-01 00:00:00 was {}.",
        date_time_after_a_billion_seconds);
}
```

Display formatted date and time

chrono v0.4.26 Date and time

Gets and displays the current time in UTC using `Utc::now`. Formats the current time in the well-known formats RFC 2822 using `DateTime::to_rfc2822` and RFC 3339 using `DateTime::to_rfc3339`, and in a custom format using `DateTime::format`.

```
use chrono::{DateTime, Utc};

fn main() {
    let now: DateTime<Utc> = Utc::now();

    println!("UTC now is: {}", now);
    println!("UTC now in RFC 2822 is: {}", now.to_rfc2822());
    println!("UTC now in RFC 3339 is: {}", now.to_rfc3339());
    println!("UTC now in a custom format is: {}", now.format("%a %b %e %T %Y"));
}
```

Parse string into DateTime struct

chrono v0.4.26 Date and time

Parses a `DateTime` struct from strings representing the well-known formats RFC 2822, RFC 3339, and a custom format, using `DateTime::parse_from_rfc2822`, `DateTime::parse_from_rfc3339`, and `DateTime::parse_from_str` respectively.

Escape sequences that are available for the `DateTime::parse_from_str` can be found at `chrono::format::strftime`. Note that the `DateTime::parse_from_str` requires that such a DateTime struct must be creatable that it uniquely identifies a date and a time. For parsing dates and times without timezones use `NaiveDate`, `NaiveTime`, and `NaiveDateTime`.

```
use chrono::{DateTime, NaiveDate, NaiveDateTime, NaiveTime};
use chrono::format::ParseError;


fn main() -> Result<(), ParseError> {
    let rfc2822 = DateTime::parse_from_rfc2822("Tue, 1 Jul 2003 10:52:37 +0200")?;
    println!("{}", rfc2822);

    let rfc3339 = DateTime::parse_from_rfc3339("1996-12-19T16:39:57-08:00")?;
    println!("{}", rfc3339);

    let custom = DateTime::parse_from_str("5.8.1994 8:00 am +0000", "%d.%m.%Y %H:%M %P
%z")?;
    println!("{}", custom);

    let time_only = NaiveTime::parse_from_str("23:56:04", "%H:%M:%S")?;
    println!("{}", time_only);

    let date_only = NaiveDate::parse_from_str("2015-09-05", "%Y-%m-%d")?;
    println!("{}", date_only);

    let no_timezone = NaiveDateTime::parse_from_str("2015-09-05 23:56:04", "%Y-%m-%d
%H:%M:%S")?;
    println!("{}", no_timezone);

    Ok(())
}
```

Development Tools

Debugging

Recipe	Crates	Categories
Log a debug message to the console	log v0.4.19 env_logger v0.10.0	Debugging
Log an error message to the console	log v0.4.19 env_logger v0.10.0	Debugging
Log to stdout instead of stderr	log v0.4.19 env_logger v0.10.0	Debugging
Log messages with a custom logger	log v0.4.19	Debugging
Log to the Unix syslog	log v0.4.19 syslog v6.1.0	Debugging
Enable log levels per module	log v0.4.19 env_logger v0.10.0	Debugging
Use a custom environment variable to set up logging	log v0.4.19 env_logger v0.10.0	Debugging
Include timestamp in log messages	log v0.4.19 env_logger v0.10.0 chrono v0.4.26	Debugging
Log messages to a custom location	log v0.4.19 log4rs v1.2.0	Debugging

Versioning

Recipe	Crates	Categories
Parse and increment a version string	semver v1.0.17	Config
Parse a complex version string	semver v1.0.17	Config
Check if given version is pre-release	semver v1.0.17	Config
Find the latest version satisfying given range	semver v1.0.17	Config
Check external command version for compatibility	semver v1.0.17	Text processing, OS

Build Time

Recipe	Crates	Categories
Compile and link statically to a bundled C library	cc v1.0.79	Development tools
Compile and link statically to a bundled C++ library	cc v1.0.79	Development tools
Compile a C library while setting custom defines	cc v1.0.79	Development tools

Debugging

Recipe	Crates	Categories
Log a debug message to the console	log v0.4.19, env_logger v0.10.0	Debugging
Log an error message to the console	log v0.4.19, env_logger v0.10.0	Debugging
Log to stdout instead of stderr	log v0.4.19, env_logger v0.10.0	Debugging
Log messages with a custom logger	log v0.4.19	Debugging
Log to the Unix syslog	log v0.4.19, syslog v6.1.0	Debugging
Enable log levels per module	log v0.4.19, env_logger v0.10.0	Debugging
Use a custom environment variable to set up logging	log v0.4.19, env_logger v0.10.0	Debugging
Include timestamp in log messages	log v0.4.19, env_logger v0.10.0, chrono v0.4.26	Debugging
Log messages to a custom location	log v0.4.19, log4rs v1.2.0	Debugging

Log Messages

Log a debug message to the console

log v0.4.19 env_logger v0.10.0 Debugging

The `log` crate provides logging utilities. The `env_logger` crate configures logging via an environment variable. The `log::debug!` macro works like other `std::fmt` formatted strings.

```
fn execute_query(query: &str) {
    log::debug!("Executing query: {}", query);
}

fn main() {
    env_logger::init();

    execute_query("DROP TABLE students");
}
```

No output prints when running this code. By default, the log level is `error`, and any lower levels are dropped.

Set the `RUST_LOG` environment variable to print the message:

```
$ RUST_LOG=debug cargo run
```

Cargo prints debugging information then the following line at the very end of the output:

```
DEBUG:main: Executing query: DROP TABLE students
```

Log an error message to the console

log v0.4.19 env_logger v0.10.0 Debugging

Proper error handling considers exceptions exceptional. Here, an error logs to stderr with `log`'s convenience macro `log::error!`.

```
fn execute_query(_query: &str) -> Result<(), &'static str> {
    Err("I'm afraid I can't do that")
}

fn main() {
    env_logger::init();

    let response = execute_query("DROP TABLE students");
    if let Err(err) = response {
        log::error!("Failed to execute query: {}", err);
    }
}
```

Log to stdout instead of stderr

log v0.4.19 env_logger v0.10.0 Debugging

Creates a custom logger configuration using the `Builder::target` to set the target of the log output to `Target::Stdout`.

```
use env_logger::{Builder, Target};

fn main() {
    Builder::new()
        .target(Target::Stdout)
        .init();

    log::error!("This error has been printed to Stdout");
}
```

Log messages with a custom logger

log v0.4.19 Debugging

Implements a custom logger `ConsoleLogger` which prints to stdout. In order to use the logging macros, `ConsoleLogger` implements the `log::Log` trait and `log::set_logger` installs it.

```
use log::{Record, Level, Metadata, LevelFilter, SetLoggerError};

static CONSOLE_LOGGER: ConsoleLogger = ConsoleLogger;

struct ConsoleLogger;

impl log::Log for ConsoleLogger {
  fn enabled(&self, metadata: &Metadata) -> bool {
     metadata.level() <= Level::Info
    }

    fn log(&self, record: &Record) {
        if self.enabled(record.metadata()) {
            println!("Rust says: {} - {}", record.level(), record.args());
        }
    }

    fn flush(&self) {}
}

fn main() -> Result<(), SetLoggerError> {
    log::set_logger(&CONSOLE_LOGGER)?;
    log::set_max_level(LevelFilter::Info);

    log::info!("hello log");
    log::warn!("warning");
    log::error!("oops");
    Ok(())
}
```

Log to the Unix syslog

log v0.4.19 syslog v6.1.0 Debugging

Logs messages to UNIX syslog. Initializes logger backend with `syslog::init`. `syslog::Facility` records the program submitting the log entry's classification, `log::LevelFilter` denotes allowed log verbosity and `Option<&str>` holds optional application name.

```
use syslog::{Facility, Error};

fn main() -> Result<(), Error> {
    syslog::init(Facility::LOG_USER,
                 log::LevelFilter::Debug,
                 Some("My app name"))?;
    log::debug!("this is a debug {}", "message");
    log::error!("this is an error!");
    Ok(())
}
```

Configure Logging

Enable log levels per module

log v0.4.19 env_logger v0.10.0 Debugging

Creates two modules `foo` and nested `foo::bar` with logging directives controlled separately with `RUST_LOG` environmental variable.

```
mod foo {
    mod bar {
        pub fn run() {
            log::warn!("[bar] warn");
            log::info!("[bar] info");
            log::debug!("[bar] debug");
        }
    }

    pub fn run() {
        log::warn!("[foo] warn");
        log::info!("[foo] info");
        log::debug!("[foo] debug");
        bar::run();
    }
}

fn main() {
    env_logger::init();
    log::warn!("[root] warn");
    log::info!("[root] info");
    log::debug!("[root] debug");
    foo::run();
}
```

`RUST_LOG` environment variable controls `env_logger` output. Module declarations take comma separated entries formatted like `path::to::module=log_level`. Run the `test` application as

follows:

```
RUST_LOG="warn,test::foo=info,test::foo::bar=debug" ./test
```

Sets the default `log::Level` to `warn`, module `foo` and module `foo::bar` to `info` and `debug`.

```
WARN:test: [root] warn
WARN:test::foo: [foo] warn
INFO:test::foo: [foo] info
WARN:test::foo::bar: [bar] warn
INFO:test::foo::bar: [bar] info
DEBUG:test::foo::bar: [bar] debug
```

Use a custom environment variable to set up logging

log v0.4.19 env_logger v0.10.0 Debugging

`Builder` configures logging.

`Builder::parse` parses `MY_APP_LOG` environment variable contents in the form of `RUST_LOG` syntax. Then, `Builder::init` initializes the logger. All these steps are normally done internally by `env_logger::init`.

```
use std::env;
use env_logger::Builder;

fn main() {
    Builder::new()
        .parse(&env::var("MY_APP_LOG").unwrap_or_default())
        .init();

    log::info!("informational message");
    log::warn!("warning message");
    log::error!("this is an error {}", "message");
}
```

Include timestamp in log messages

log v0.4.19 env_logger v0.10.0 chrono v0.4.26 Debugging

Creates a custom logger configuration with `Builder`. Each log entry calls `Local::now` to get the current `DateTime` in local timezone and uses `DateTime::format` with `strftime::specifiers` to format a timestamp used in the final log.

The example calls `Builder::format` to set a closure which formats each message text with timestamp, `Record::level` and body (`Record::args`).

```
use std::io::Write;
use chrono::Local;
use env_logger::Builder;
use log::LevelFilter;

fn main() {
    Builder::new()
        .format(|buf, record| {
            writeln!(buf,
                "{} [{}] - {}",
                Local::now().format("%Y-%m-%dT%H:%M:%S"),
                record.level(),
                record.args()
            )
        })
        .filter(None, LevelFilter::Info)
        .init();

    log::warn!("warn");
    log::info!("info");
    log::debug!("debug");
}
```

stderr output will contain

```
2017-05-22T21:57:06 [WARN] - warn
2017-05-22T21:57:06 [INFO] - info
```

Log messages to a custom location

log4rs configures log output to a custom location. log4rs can use either an external YAML file or a builder configuration.

Create the log configuration with `log4rs::append::file::FileAppender`. An appender defines the logging destination. The configuration continues with encoding using a custom pattern from `log4rs::encode::pattern`. Assigns the configuration to `log4rs::config::Config` and sets the default `log::LevelFilter`.

```
use log::LevelFilter;
use log4rs::append::file::FileAppender;
use log4rs::encode::pattern::PatternEncoder;
use log4rs::config::{Appender, Config, Root};

fn main() -> Result<()> {
    let logfile = FileAppender::builder()
        .encoder(Box::new(PatternEncoder::new("{l} - {m}\n")))
        .build("log/output.log")?;

    let config = Config::builder()
        .appender(Appender::builder().build("logfile", Box::new(logfile)))
        .build(Root::builder()
                   .appender("logfile")
                   .build(LevelFilter::Info))?;

    log4rs::init_config(config)?;

    log::info!("Hello, world!");

    Ok(())
}
```

Versioning

Parse and increment a version string.

Constructs a `semver::Version` from a string literal using `Version::parse`, then increments it by patch, minor, and major version number one by one.

Note that in accordance with the Semantic Versioning Specification, incrementing the minor version number resets the patch version number to 0 and incrementing the major version number resets both the minor and patch version numbers to 0.

```
use semver::{Version, SemVerError};

fn main() -> Result<(), SemVerError> {
    let mut parsed_version = Version::parse("0.2.6")?;

    assert_eq!(
        parsed_version,
        Version {
            major: 0,
            minor: 2,
            patch: 6,
            pre: vec![],
            build: vec![],
        }
    );

    parsed_version.increment_patch();
    assert_eq!(parsed_version.to_string(), "0.2.7");
    println!("New patch release: v{}", parsed_version);

    parsed_version.increment_minor();
    assert_eq!(parsed_version.to_string(), "0.3.0");
    println!("New minor release: v{}", parsed_version);

    parsed_version.increment_major();
    assert_eq!(parsed_version.to_string(), "1.0.0");
    println!("New major release: v{}", parsed_version);

    Ok(())
}
```

Parse a complex version string.

Constructs a `semver::Version` from a complex version string using `Version::parse`. The string contains pre-release and build metadata as defined in the Semantic Versioning Specification.

Note that, in accordance with the Specification, build metadata is parsed but not considered when comparing versions. In other words, two versions may be equal even if their build strings differ.

```
use semver::{Identifier, Version, SemVerError};

fn main() -> Result<(), SemVerError> {
    let version_str = "1.0.49-125+g72ee7853";
    let parsed_version = Version::parse(version_str)?;

    assert_eq!(
        parsed_version,
        Version {
            major: 1,
            minor: 0,
            patch: 49,
            pre: vec![Identifier::Numeric(125)],
            build: vec![],
        }
    );
    assert_eq!(
        parsed_version.build,
        vec![Identifier::AlphaNumeric(String::from("g72ee7853"))]
    );

    let serialized_version = parsed_version.to_string();
    assert_eq!(&serialized_version, version_str);

    Ok(())
}
```

Check if given version is pre-release.

semver v1.0.17 Config

Given two versions, `is_prerelease` asserts that one is pre-release and the other is not.

```
use semver::{Version, SemVerError};

fn main() -> Result<(), SemVerError> {
    let version_1 = Version::parse("1.0.0-alpha")?;
    let version_2 = Version::parse("1.0.0")?;

    assert!(version_1.is_prerelease());
    assert!(!version_2.is_prerelease());

    Ok(())
}
```

Find the latest version satisfying given range

semver v1.0.17 Config

Given a list of version &strs, finds the latest `semver::Version`. `semver::VersionReq` filters the list with `VersionReq::matches`. Also demonstrates `semver` pre-release preferences.

```
use semver::{Version, VersionReq};

fn find_max_matching_version<'a, I>(version_req_str: &str, iterable: I) ->
Result<Option<Version>>
where
    I: IntoIterator<Item = &'a str>,
{
    let vreq = VersionReq::parse(version_req_str)?;

    Ok(
        iterable
            .into_iter()
            .filter_map(|s| Version::parse(s).ok())
            .filter(|s| vreq.matches(s))
            .max(),
    )
}

fn main() -> Result<()> {
    assert_eq!(
        find_max_matching_version("<= 1.0.0", vec!["0.9.0", "1.0.0", "1.0.1"])?,
        Some(Version::parse("1.0.0")?)
    );

    assert_eq!(
        find_max_matching_version(
            ">1.2.3-alpha.3",
            vec![
                "1.2.3-alpha.3",
                "1.2.3-alpha.4",
                "1.2.3-alpha.10",
                "1.2.3-beta.4",
                "3.4.5-alpha.9",
            ]
        )?,
        Some(Version::parse("1.2.3-beta.4")?)
    );

    Ok(())
}
```

Check external command version for compatibility

semver v1.0.17 Text processing OS

Runs `git --version` using `Command`, then parses the version number into a `semver::Version` using `Version::parse`. `VersionReq::matches` compares `semver::VersionReq` to the parsed version. The command output resembles "git version x.y.z".

```
use std::process::Command;
use semver::{Version, VersionReq};

fn main() -> Result<()> {
    let version_constraint = "> 1.12.0";
    let version_test = VersionReq::parse(version_constraint)?;
    let output = Command::new("git").arg("--version").output()?;

    if !output.status.success() {
        error_chain::bail!("Command executed with failing error code");
    }

    let stdout = String::from_utf8(output.stdout)?;
    let version = stdout.split(" ").last().ok_or_else(|| {
        "Invalid command output"
    })?;
    let parsed_version = Version::parse(version)?;

    if !version_test.matches(&parsed_version) {
        error_chain::bail!("Command version lower than minimum supported version (found
{}, need {})",
            parsed_version, version_constraint);
    }

    Ok(())
}
```

Build Time Tooling

This section covers "build-time" tooling, or code that is run prior to compiling a crate's source code. Conventionally, build-time code lives in a **build.rs** file and is commonly referred to as a "build script". Common use cases include rust code generation and compilation of bundled C/C++/asm code. See crates.io's documentation on the matter for more information.

Compile and link statically to a bundled C library

cc v1.0.79 Development tools

To accommodate scenarios where additional C, C++, or assembly is required in a project, the **cc** crate offers a simple api for compiling bundled C/C++/asm code into static libraries (**.a**) that can be statically linked to by **rustc**.

The following example has some bundled C code (**src/hello.c**) that will be used from rust. Before compiling rust source code, the "build" file (**build.rs**) specified in **Cargo.toml** runs. Using the **cc** crate, a static library file will be produced (in this case, **libhello.a**, see `compile` docs) which can then be used from rust by declaring the external function signatures in an `extern` block.

Since the bundled C is very simple, only a single source file needs to be passed to `cc::Build`. For more complex build requirements, `cc::Build` offers a full suite of builder methods for specifying `include` paths and extra compiler `flag`s.

Cargo.toml

```
[package]
...
build = "build.rs"

[build-dependencies]
cc = "1"

[dependencies]
error-chain = "0.11"
```

build.rs

```
fn main() {
    cc::Build::new()
        .file("src/hello.c")
        .compile("hello");   // outputs `libhello.a`
}
```

src/hello.c

```
#include <stdio.h>


void hello() {
    printf("Hello from C!\n");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

`src/main.rs`

```
use error_chain::error_chain;
use std::ffi::CString;
use std::os::raw::c_char;

error_chain! {
    foreign_links {
        NulError(::std::ffi::NulError);
        Io(::std::io::Error);
    }
}
fn prompt(s: &str) -> Result<String> {
    use std::io::Write;
    print!("{}", s);
    std::io::stdout().flush()?;
    let mut input = String::new();
    std::io::stdin().read_line(&mut input)?;
    Ok(input.trim().to_string())
}

extern {
    fn hello();
    fn greet(name: *const c_char);
}

fn main() -> Result<()> {
    unsafe { hello() }
    let name = prompt("What's your name? ")?;
    let c_name = CString::new(name)?;
    unsafe { greet(c_name.as_ptr()) }
    Ok(())
}
```

Compile and link statically to a bundled C++ library

cc v1.0.79 Development tools

Linking a bundled C++ library is very similar to linking a bundled C library. The two core differences when compiling and statically linking a bundled C++ library are specifying a C++ compiler via the builder method `cpp(true)` and preventing name mangling by the C++ compiler by adding the `extern "C"` section at the top of our C++ source file.

`Cargo.toml`

```
[package]
...
build = "build.rs"

[build-dependencies]
cc = "1"
```

build.rs

```
fn main() {
    cc::Build::new()
        .cpp(true)
        .file("src/foo.cpp")
        .compile("foo");
}
```

src/foo.cpp

```
extern "C" {
    int multiply(int x, int y);
}

int multiply(int x, int y) {
    return x*y;
}
```

src/main.rs

```
extern {
    fn multiply(x : i32, y : i32) -> i32;
}

fn main(){
    unsafe {
        println!("{}", multiply(5,7));
    }
}
```

Compile a C library while setting custom defines

cc v1.0.79 Development tools

It is simple to build bundled C code with custom defines using `cc::Build::define`. The method takes an `Option` value, so it is possible to create defines such as `#define APP_NAME "foo"` as well as `#define WELCOME` (pass `None` as the value for a value-less define). This example builds a bundled C file with dynamic defines set in `build.rs` and prints "**Welcome to foo - version 1.0.2**" when run. Cargo sets some environment variables which may be useful for some custom defines.

Cargo.toml

```
[package]
...
version = "1.0.2"
build = "build.rs"

[build-dependencies]
cc = "1"
```

`build.rs`

```
fn main() {
    cc::Build::new()
        .define("APP_NAME", "\"foo\"")
        .define("VERSION", format!("\"{}\"", env!("CARGO_PKG_VERSION")).as_str())
        .define("WELCOME", None)
        .file("src/foo.c")
        .compile("foo");
}
```

`src/foo.c`

```
#include <stdio.h>

void print_app_info() {
#ifdef WELCOME
    printf("Welcome to ");
#endif
    printf("%s - version %s\n", APP_NAME, VERSION);
}
```

`src/main.rs`

```
extern {
    fn print_app_info();
}

fn main(){
    unsafe {
        print_app_info();
    }
}
```

Encoding

Recipe	Crates	Categories
Percent-encode a string	percent-encoding v2.3.0	Encoding
Encode a string as application/x-www-form-urlencoded	url v2.4.0	Encoding
Encode and decode hex	data-encoding v2.4.0	Encoding
Encode and decode base64	base64 v0.21.2	Encoding
Read CSV records	csv v1.2.2	Encoding
Read CSV records with different delimiter	csv v1.2.2	Encoding
Filter CSV records matching a predicate	csv v1.2.2	Encoding
Handle invalid CSV data with Serde	csv v1.2.2 serde v1.0.171	Encoding
Serialize records to CSV	csv v1.2.2	Encoding
Serialize records to CSV using Serde	csv v1.2.2 serde v1.0.171	Encoding

Recipe	Crates	Categories
Transform one column of a CSV file	csv v1.2.2 serde v1.0.171	Encoding
Serialize and deserialize unstructured JSON	serde_json v1.0.100	Encoding
Deserialize a TOML configuration file	toml v0.7.6	Encoding
Read and write integers in little-endian byte order	byteorder v1.4.3	Encoding

Character Sets

Percent-encode a string

percent-encoding v2.3.0 Encoding

Encode an input string with percent-encoding using the `utf8_percent_encode` function from the `percent-encoding` crate. Then decode using the `percent_decode` function.

```
use percent_encoding::{utf8_percent_encode, percent_decode, AsciiSet, CONTROLS};
use std::str::Utf8Error;

/// https://url.spec.whatwg.org/#fragment-percent-encode-set
const FRAGMENT: &AsciiSet = &CONTROLS.add(b'
').add(b'"').add(b'<').add(b'>').add(b'`');

fn main() -> Result<(), Utf8Error> {
    let input = "confident, productive systems programming";

    let iter = utf8_percent_encode(input, FRAGMENT);
    let encoded: String = iter.collect();
    assert_eq!(encoded, "confident,%20productive%20systems%20programming");

    let iter = percent_decode(encoded.as_bytes());
    let decoded = iter.decode_utf8()?;
    assert_eq!(decoded, "confident, productive systems programming");

    Ok(())
}
```

The encode set defines which bytes (in addition to non-ASCII and controls) need to be percent-encoded. The choice of this set depends on context. For example, `url` encodes `?` in a URL path but not in a query string.

The return value of encoding is an iterator of `&str` slices which collect into a `String`.

Encode a string as application/x-www-form-urlencoded

url v2.4.0 Encoding

Encodes a string into application/x-www-form-urlencoded syntax using the `form_urlencoded::byte_serialize` and subsequently decodes it with `form_urlencoded::parse`. Both functions return iterators that collect into a `String`.

```
use url::form_urlencoded::{byte_serialize, parse};

fn main() {
    let urlencoded: String = byte_serialize("What is ❤?".as_bytes()).collect();
    assert_eq!(urlencoded, "What+is+%E2%9D%A4%3F");
    println!("urlencoded:'{}'", urlencoded);

    let decoded: String = parse(urlencoded.as_bytes())
        .map(|(key, val)| [key, val].concat())
        .collect();
    assert_eq!(decoded, "What is ❤?");
    println!("decoded:'{}'", decoded);
}
```

Encode and decode hex

The `data_encoding` crate provides a `HEXUPPER::encode` method which takes a `&[u8]` and returns a `String` containing the hexadecimal representation of the data.

Similarly, a `HEXUPPER::decode` method is provided which takes a `&[u8]` and returns a `Vec<u8>` if the input data is successfully decoded.

The example below coverts `&[u8]` data to hexadecimal equivalent. Compares this value to the expected value.

```
use data_encoding::{HEXUPPER, DecodeError};

fn main() -> Result<(), DecodeError> {
    let original = b"The quick brown fox jumps over the lazy dog.";
    let expected = "5468652071756963B2062726F776E20666F78206A756D7073206F76\
        657220746865206C617A7920646F672E";

    let encoded = HEXUPPER.encode(original);
    assert_eq!(encoded, expected);

    let decoded = HEXUPPER.decode(&encoded.into_bytes())?;
    assert_eq!(&decoded[..], &original[..]);

    Ok(())
}
```

Encode and decode base64

Encodes byte slice into `base64` String using `encode` and decodes it with `decode`.

```
use std::str;
use base64::{encode, decode};

fn main() -> Result<()> {
    let hello = b"hello rustaceans";
    let encoded = encode(hello);
    let decoded = decode(&encoded)?;

    println!("origin: {}", str::from_utf8(hello)?);
    println!("base64 encoded: {}", encoded);
    println!("back to origin: {}", str::from_utf8(&decoded)?);

    Ok(())
}
```

CSV processing

Read CSV records

csv v1.2.2 Encoding

Reads standard CSV records into `csv::StringRecord` — a weakly typed data representation which expects valid UTF-8 rows. Alternatively, `csv::ByteRecord` makes no assumptions about UTF-8.

```
use csv::Error;

fn main() -> Result<(), Error> {
    let csv = "year,make,model,description
        1948,Porsche,356,Luxury sports car
        1967,Ford,Mustang fastback 1967,American car";

    let mut reader = csv::Reader::from_reader(csv.as_bytes());
    for record in reader.records() {
        let record = record?;
        println!(
            "In {}, {} built the {} model. It is a {}.",
            &record[0],
            &record[1],
            &record[2],
            &record[3]
        );
    }

    Ok(())
}
```

Serde deserializes data into strongly type structures. See the `csv::Reader::deserialize` method.

```
use serde::Deserialize;
#[derive(Deserialize)]
struct Record {
    year: u16,
    make: String,
    model: String,
    description: String,
}

fn main() -> Result<(), csv::Error> {
    let csv = "year,make,model,description
1948,Porsche,356,Luxury sports car
1967,Ford,Mustang fastback 1967,American car";

    let mut reader = csv::Reader::from_reader(csv.as_bytes());

    for record in reader.deserialize() {
        let record: Record = record?;
        println!(
            "In {}, {} built the {} model. It is a {}.",
            record.year,
            record.make,
            record.model,
            record.description
        );
    }

    Ok(())
}
```

Read CSV records with different delimiter

Reads CSV records with a tab `delimiter`.

```
use csv::Error;
use serde::Deserialize;
#[derive(Debug, Deserialize)]
struct Record {
    name: String,
    place: String,
    #[serde(deserialize_with = "csv::invalid_option")]
    id: Option<u64>,
}

use csv::ReaderBuilder;

fn main() -> Result<(), Error> {
    let data = "name\tplace\tid
        Mark\tMelbourne\t46
        Ashley\tZurich\t92";

    let mut reader =
ReaderBuilder::new().delimiter(b'\t').from_reader(data.as_bytes());
    for result in reader.deserialize::<Record>() {
        println!("{:?}", result?);
    }

    Ok(())
}
```

Filter CSV records matching a predicate

csv v1.2.2 Encoding

Returns *only* the rows from `data` with a field that matches `query`.

```
use std::io;

fn main() -> Result<()> {
    let query = "CA";
    let data = "\
City,State,Population,Latitude,Longitude
Kenai,AK,7610,60.5544444,-151.2583333
Oakman,AL,,33.7133333,-87.3886111
Sandfort,AL,,32.3380556,-85.2233333
West Hollywood,CA,37031,34.0900000,-118.3608333";

    let mut rdr = csv::ReaderBuilder::new().from_reader(data.as_bytes());
    let mut wtr = csv::Writer::from_writer(io::stdout());

    wtr.write_record(rdr.headers()?)?;

    for result in rdr.records() {
        let record = result?;
        if record.iter().any(|field| field == query) {
            wtr.write_record(&record)?;
        }
    }

    wtr.flush()?;
    Ok(())
}
```

Disclaimer: this example has been adapted from the csv crate tutorial.

Handle invalid CSV data with Serde

csv v1.2.2 serde v1.0.171 Encoding

CSV files often contain invalid data. For these cases, the `csv` crate provides a custom deserializer, `csv::invalid_option`, which automatically converts invalid data to None values.

```
use csv::Error;
use serde::Deserialize;

#[derive(Debug, Deserialize)]
struct Record {
    name: String,
    place: String,
    #[serde(deserialize_with = "csv::invalid_option")]
    id: Option<u64>,
}

fn main() -> Result<(), Error> {
    let data = "name,place,id
mark,sydney,46.5
ashley,zurich,92
akshat,delhi,37
alisha,colombo,xyz";

    let mut rdr = csv::Reader::from_reader(data.as_bytes());
    for result in rdr.deserialize() {
        let record: Record = result?;
        println!("{:?}", record);
    }

    Ok(())
}
```

Serialize records to CSV

csv v1.2.2 Encoding

This example shows how to serialize a Rust tuple. `csv::writer` supports automatic serialization from Rust types into CSV records. `write_record` writes a simple record containing string data only. Data with more complex values such as numbers, floats, and options use `serialize`. Since CSV writer uses internal buffer, always explicitly `flush` when done.

```
use std::io;

fn main() -> Result<()> {
    let mut wtr = csv::Writer::from_writer(io::stdout());

    wtr.write_record(&["Name", "Place", "ID"])?;

    wtr.serialize(("Mark", "Sydney", 87))?;
    wtr.serialize(("Ashley", "Dublin", 32))?;
    wtr.serialize(("Akshat", "Delhi", 11))?;

    wtr.flush()?;
    Ok(())
}
```

Serialize records to CSV using Serde

csv v1.2.2 serde v1.0.171 Encoding

The following example shows how to serialize custom structs as CSV records using the serde crate.

```rust
use serde::Serialize;
use std::io;

#[derive(Serialize)]
struct Record<'a> {
    name: &'a str,
    place: &'a str,
    id: u64,
}

fn main() -> Result<()> {
    let mut wtr = csv::Writer::from_writer(io::stdout());

    let rec1 = Record { name: "Mark", place: "Melbourne", id: 56};
    let rec2 = Record { name: "Ashley", place: "Sydney", id: 64};
    let rec3 = Record { name: "Akshat", place: "Delhi", id: 98};

    wtr.serialize(rec1)?;
    wtr.serialize(rec2)?;
    wtr.serialize(rec3)?;

    wtr.flush()?;

    Ok(())
}
```

Transform CSV column

Transform a CSV file containing a color name and a hex color into one with a color name and an rgb color. Utilizes the csv crate to read and write the csv file, and serde to deserialize and serialize the rows to and from bytes.

See `csv::Reader::deserialize`, `serde::Deserialize`, and `std::str::FromStr`

```rust
use csv::{Reader, Writer};
use serde::{de, Deserialize, Deserializer};
use std::str::FromStr;

#[derive(Debug)]
struct HexColor {
    red: u8,
    green: u8,
    blue: u8,
}

#[derive(Debug, Deserialize)]
struct Row {
    color_name: String,
    color: HexColor,
}

impl FromStr for HexColor {
    type Err = Error;

    fn from_str(hex_color: &str) -> std::result::Result<Self, Self::Err> {
        let trimmed = hex_color.trim_matches('#');
        if trimmed.len() != 6 {
            Err("Invalid length of hex string".into())
        } else {
            Ok(HexColor {
                red: u8::from_str_radix(&trimmed[..2], 16)?,
                green: u8::from_str_radix(&trimmed[2..4], 16)?,
                blue: u8::from_str_radix(&trimmed[4..6], 16)?,
            })
        }
    }
}

impl<'de> Deserialize<'de> for HexColor {
    fn deserialize<D>(deserializer: D) -> std::result::Result<Self, D::Error>
    where
        D: Deserializer<'de>,
    {
        let s = String::deserialize(deserializer)?;
        FromStr::from_str(&s).map_err(de::Error::custom)
    }
}

fn main() -> Result<()> {
    let data = "color_name,color
red,#ff0000
green,#00ff00
blue,#0000FF
periwinkle,#ccccff
magenta,#ff00ff"
        .to_owned();
    let mut out = Writer::from_writer(vec![]);
    let mut reader = Reader::from_reader(data.as_bytes());
    for result in reader.deserialize::<Row>() {
        let res = result?;
        out.serialize((
            res.color_name,
            res.color.red,
            res.color.green,
            res.color.blue,
        ))?;
    }
    let written = String::from_utf8(out.into_inner()?)?;
    assert_eq!(Some("magenta,255,0,255"), written.lines().last());
    println!("{}", written);
```

```
    Ok(())
}
```

Structured Data

Serialize and deserialize unstructured JSON

The `serde_json` crate provides a `from_str` function to parse a `&str` of JSON.

Unstructured JSON can be parsed into a universal `serde_json::Value` type that is able to represent any valid JSON data.

The example below shows a `&str` of JSON being parsed. The expected value is declared using the `json!` macro.

```
 use serde_json::json;
use serde_json::{Value, Error};

fn main() -> Result<(), Error> {
    let j = r#"{
                "userid": 103609,
                "verified": true,
                "access_privileges": [
                  "user",
                  "admin"
                ]
              }"#;

    let parsed: Value = serde_json::from_str(j)?;

    let expected = json!({
        "userid": 103609,
        "verified": true,
        "access_privileges": [
            "user",
            "admin"
        ]
    });

    assert_eq!(parsed, expected);

    Ok(())
}
```

Deserialize a TOML configuration file

Parse some TOML into a universal `toml::Value` that is able to represent any valid TOML data.

```
use toml::{Value, de::Error};

fn main() -> Result<(), Error> {
    let toml_content = r#"
          [package]
          name = "your_package"
          version = "0.1.0"
          authors = ["You! <you@example.org>"]

          [dependencies]
          serde = "1.0"
          "#;

    let package_info: Value = toml::from_str(toml_content)?;

    assert_eq!(package_info["dependencies"]["serde"].as_str(), Some("1.0"));
    assert_eq!(package_info["package"]["name"].as_str(),
               Some("your_package"));

    Ok(())
}
```

Parse TOML into your own structs using Serde.

```
use serde::Deserialize;

use toml::de::Error;
use std::collections::HashMap;

#[derive(Deserialize)]
struct Config {
    package: Package,
    dependencies: HashMap<String, String>,
}

#[derive(Deserialize)]
struct Package {
    name: String,
    version: String,
    authors: Vec<String>,
}

fn main() -> Result<(), Error> {
    let toml_content = r#"
          [package]
          name = "your_package"
          version = "0.1.0"
          authors = ["You! <you@example.org>"]

          [dependencies]
          serde = "1.0"
          "#;

    let package_info: Config = toml::from_str(toml_content)?;

    assert_eq!(package_info.package.name, "your_package");
    assert_eq!(package_info.package.version, "0.1.0");
    assert_eq!(package_info.package.authors, vec!["You! <you@example.org>"]);
    assert_eq!(package_info.dependencies["serde"], "1.0");

    Ok(())
}
```

Read and write integers in little-endian byte order

byteorder v1.4.3 Encoding

`byteorder` can reverse the significant bytes of structured data. This may be necessary when receiving information over the network, such that bytes received are from another system.

```
use byteorder::{LittleEndian, ReadBytesExt, WriteBytesExt};
use std::io::Error;

#[derive(Default, PartialEq, Debug)]
struct Payload {
    kind: u8,
    value: u16,
}

fn main() -> Result<(), Error> {
    let original_payload = Payload::default();
    let encoded_bytes = encode(&original_payload)?;
    let decoded_payload = decode(&encoded_bytes)?;
    assert_eq!(original_payload, decoded_payload);
    Ok(())
}

fn encode(payload: &Payload) -> Result<Vec<u8>, Error> {
    let mut bytes = vec![];
    bytes.write_u8(payload.kind)?;
    bytes.write_u16::<LittleEndian>(payload.value)?;
    Ok(bytes)
}

fn decode(mut bytes: &[u8]) -> Result<Payload, Error> {
    let payload = Payload {
        kind: bytes.read_u8()?,
        value: bytes.read_u16::<LittleEndian>()?,
    };
    Ok(payload)
}
```

Error Handling

Recipe	Crates	Categories
Handle errors correctly in main	error-chain v0.12.4	Rust patterns
Avoid discarding errors during error conversions	error-chain v0.12.4	Rust patterns
Obtain backtrace of complex error scenarios	error-chain v0.12.4	Rust patterns

Error Handling

Handle errors correctly in main

error-chain v0.12.4 Rust patterns

Handles error that occur when trying to open a file that does not exist. It is achieved by using error-chain, a library that takes care of a lot of boilerplate code needed in order to handle errors in Rust.

`Io(std::io::Error)` inside `foreign_links` allows automatic conversion from `std::io::Error` into `error_chain!` defined type implementing the `Error` trait.

The below recipe will tell how long the system has been running by opening the Unix file `/proc/uptime` and parse the content to get the first number. Returns uptime unless there is an error.

Other recipes in this book will hide the error-chain boilerplate, and can be seen by expanding the code with the ⤢ button.

```
use error_chain::error_chain;

use std::fs::File;
use std::io::Read;

error_chain!{
    foreign_links {
        Io(std::io::Error);
        ParseInt(::std::num::ParseIntError);
    }
}

fn read_uptime() -> Result<u64> {
    let mut uptime = String::new();
    File::open("/proc/uptime")?.read_to_string(&mut uptime)?;

    Ok(uptime
        .split('.')
        .next()
        .ok_or("Cannot parse uptime data")?
        .parse()?)
}

fn main() {
    match read_uptime() {
        Ok(uptime) => println!("uptime: {} seconds", uptime),
        Err(err) => eprintln!("error: {}", err),
    };
}
```

Avoid discarding errors during error conversions

The error-chain crate makes matching on different error types returned by a function possible and relatively compact. `ErrorKind` determines the error type.

Uses reqwest::blocking to query a random integer generator web service. Converts the string response into an integer. The Rust standard library, reqwest, and the web service can all generate errors. Well defined Rust errors use `foreign_links`. An additional `ErrorKind` variant for the web service error uses `errors` block of the `error_chain!` macro.

```
use error_chain::error_chain;

error_chain! {
    foreign_links {
        Io(std::io::Error);
        Reqwest(reqwest::Error);
        ParseIntError(std::num::ParseIntError);
    }
    errors { RandomResponseError(t: String) }
}

fn parse_response(response: reqwest::blocking::Response) -> Result<u32> {
  let mut body = response.text()?;
  body.pop();
  body
    .parse::<u32>()
    .chain_err(|| ErrorKind::RandomResponseError(body))
}

fn run() -> Result<()> {
  let url =
    format!("https://www.random.org/integers/?
num=1&min=0&max=10&col=1&base=10&format=plain");
  let response = reqwest::blocking::get(&url)?;
  let random_value: u32 = parse_response(response)?;
  println!("a random number between 0 and 10: {}", random_value);
  Ok(())
}

fn main() {
  if let Err(error) = run() {
    match *error.kind() {
      ErrorKind::Io(_) => println!("Standard IO error: {:?}", error),
      ErrorKind::Reqwest(_) => println!("Reqwest error: {:?}", error),
      ErrorKind::ParseIntError(_) => println!("Standard parse int error: {:?}", error),
      ErrorKind::RandomResponseError(_) => println!("User defined error: {:?}", error),
      _ => println!("Other error: {:?}", error),
    }
  }
}
```

Obtain backtrace of complex error scenarios

error-chain v0.12.4 Rust patterns

This recipe shows how to handle a complex error scenario and then print a backtrace. It relies on `chain_err` to extend errors by appending new errors. The error stack can be unwound, thus providing a better context to understand why an error was raised.

The below recipes attempts to deserialize the value `256` into a `u8`. An error will bubble up from Serde then csv and finally up to the user code.

```rust
use error_chain::error_chain;

#[derive(Debug, Deserialize)]
struct Rgb {
    red: u8,
    blue: u8,
    green: u8,
}

impl Rgb {
    fn from_reader(csv_data: &[u8]) -> Result<Rgb> {
        let color: Rgb = csv::Reader::from_reader(csv_data)
            .deserialize()
            .nth(0)
            .ok_or("Cannot deserialize the first CSV record")?
            .chain_err(|| "Cannot deserialize RGB color")?;

        Ok(color)
    }
}

fn run() -> Result<()> {
    let csv = "red,blue,green
102,256,204";

    let rgb = Rgb::from_reader(csv.as_bytes()).chain_err(|| "Cannot read CSV data")?;
    println!("{:?} to hexadecimal #{:X}", rgb, rgb);

    Ok(())
}

fn main() {
    if let Err(ref errors) = run() {
        eprintln!("Error level - description");
        errors
            .iter()
            .enumerate()
            .for_each(|(index, error)| eprintln!("└> {} - {}", index, error));

        if let Some(backtrace) = errors.backtrace() {
            eprintln!("{:?}", backtrace);
        }
    }
}
```

Backtrace error rendered:

```
Error level - description
└> 0 - Cannot read CSV data
└> 1 - Cannot deserialize RGB color
└> 2 - CSV deserialize error: record 1 (line: 2, byte: 15): field 1: number too large
to fit in target type
└> 3 - field 1: number too large to fit in target type
```

Run the recipe with `RUST_BACKTRACE=1` to display a detailed `backtrace` associated with this error.

File System

Recipe	Crates	Categories
Read lines of strings from a file	std 1.29.1	Filesystem
Avoid writing and reading from a same file	same_file v1.0.6	Filesystem

Recipe	Crates	Categories
Access a file randomly using a memory map	memmap v0.7.0	Filesystem
File names that have been modified in the last 24 hours	std 1.29.1	Filesystem OS
Find loops for a given path	same_file v1.0.6	Filesystem
Recursively find duplicate file names	walkdir v2.3.3	Filesystem
Recursively find all files with given predicate	walkdir v2.3.3	Filesystem
Traverse directories while skipping dotfiles	walkdir v2.3.3	Filesystem
Recursively calculate file sizes at given depth	walkdir v2.3.3	Filesystem
Find all png files recursively	glob v0.3.1	Filesystem
Find all files with given pattern ignoring filename case	glob v0.3.1	Filesystem

Read & Write

Read lines of strings from a file

std 1.29.1 Filesystem

Writes a three-line message to a file, then reads it back a line at a time with the `Lines` iterator created by `BufRead::lines`. `File` implements `Read` which provides `BufReader` trait. `File::create` opens a `File` for writing, `File::open` for reading.

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

fn main() -> Result<(), Error> {
    let path = "lines.txt";

    let mut output = File::create(path)?;
    write!(output, "Rust\n💖\nFun")?;

    let input = File::open(path)?;
    let buffered = BufReader::new(input);

    for line in buffered.lines() {
        println!("{}", line?);
    }

    Ok(())
}
```

Avoid writing and reading from a same file

same_file v1.0.6 Filesystem

Use `same_file::Handle` to a file that can be tested for equality with other handles. In this example, the handles of file to be read from and to be written to are tested for equality.

```
use same_file::Handle;
use std::fs::File;
use std::io::{BufRead, BufReader, Error, ErrorKind};
use std::path::Path;

fn main() -> Result<(), Error> {
    let path_to_read = Path::new("new.txt");

    let stdout_handle = Handle::stdout()?;
    let handle = Handle::from_path(path_to_read)?;

    if stdout_handle == handle {
        return Err(Error::new(
            ErrorKind::Other,
            "You are reading and writing to the same file",
        ));
    } else {
        let file = File::open(&path_to_read)?;
        let file = BufReader::new(file);
        for (num, line) in file.lines().enumerate() {
            println!("{} : {}", num, line?.to_uppercase());
        }
    }

    Ok(())
}
```

```
cargo run
```

displays the contents of the file new.txt.

```
cargo run >> ./new.txt
```

errors because the two files are same.

Access a file randomly using a memory map

Creates a memory map of a file using memmap and simulates some non-sequential reads from the file. Using a memory map means you just index into a slice rather than dealing with `seek` to navigate a File.

The `Mmap::map` function assumes the file behind the memory map is not being modified at the same time by another process or else a race condition occurs.

```
use memmap::Mmap;
use std::fs::File;
use std::io::{Write, Error};

fn main() -> Result<(), Error> {
    let file = File::open("content.txt")?;
    let map = unsafe { Mmap::map(&file)? };

    let random_indexes = [0, 1, 2, 19, 22, 10, 11, 29];
    assert_eq!(&map[3..13], b"hovercraft");
    let random_bytes: Vec<u8> = random_indexes.iter()
        .map(|&idx| map[idx])
        .collect();
    assert_eq!(&random_bytes[..], b"My loaf!");
    Ok(())
}
```

Directory Traversal

File names that have been modified in the last 24 hours

std 1.29.1 Filesystem

Gets the current working directory by calling `env::current_dir`, then for each entries in `fs::read_dir`, extracts the `DirEntry::path` and gets the metadata via `fs::Metadata`. The `Metadata::modified` returns the `SystemTime::elapsed` time since last modification. `Duration::as_secs` converts the time to seconds and compared with 24 hours (24 * 60 * 60 seconds). `Metadata::is_file` filters out directories.

```
use std::{env, fs};

fn main() -> Result<()> {
    let current_dir = env::current_dir()?;
    println!(
        "Entries modified in the last 24 hours in {:?}:",
        current_dir
    );

    for entry in fs::read_dir(current_dir)? {
        let entry = entry?;
        let path = entry.path();

        let metadata = fs::metadata(&path)?;
        let last_modified = metadata.modified()?.elapsed()?.as_secs();

        if last_modified < 24 * 3600 && metadata.is_file() {
            println!(
                "Last modified: {:?} seconds, is read only: {:?}, size: {:?} bytes,
filename: {:?}",
                last_modified,
                metadata.permissions().readonly(),
                metadata.len(),
                path.file_name().ok_or("No filename")?
            );
        }
    }

    Ok(())
}
```

Find loops for a given path

same_file v1.0.6 Filesystem

Use `same_file::is_same_file` to detect loops for a given path. For example, a loop could be created on a Unix system via symlinks:

```
mkdir -p /tmp/foo/bar/baz
ln -s /tmp/foo/  /tmp/foo/bar/baz/qux
```

The following would assert that a loop exists.

```
use std::io;
use std::path::{Path, PathBuf};
use same_file::is_same_file;

fn contains_loop<P: AsRef<Path>>(path: P) -> io::Result<Option<(PathBuf, PathBuf)>> {
    let path = path.as_ref();
    let mut path_buf = path.to_path_buf();
    while path_buf.pop() {
        if is_same_file(&path_buf, path)? {
            return Ok(Some((path_buf, path.to_path_buf())));
        } else if let Some(looped_paths) = contains_loop(&path_buf)? {
            return Ok(Some(looped_paths));
        }
    }
    return Ok(None);
}

fn main() {
    assert_eq!(
        contains_loop("/tmp/foo/bar/baz/qux/bar/baz").unwrap(),
        Some((
            PathBuf::from("/tmp/foo"),
            PathBuf::from("/tmp/foo/bar/baz/qux")
        ))
    );
}
```

Recursively find duplicate file names

walkdir v2.3.3 Filesystem

Find recursively in the current directory duplicate filenames, printing them only once.

```
use std::collections::HashMap;
use walkdir::WalkDir;

fn main() {
    let mut filenames = HashMap::new();

    for entry in WalkDir::new(".")
            .into_iter()
            .filter_map(Result::ok)
            .filter(|e| !e.file_type().is_dir()) {
        let f_name = String::from(entry.file_name().to_string_lossy());
        let counter = filenames.entry(f_name.clone()).or_insert(0);
        *counter += 1;

        if *counter == 2 {
            println!("{}", f_name);
        }
    }
}
```

Recursively find all files with given predicate

walkdir v2.3.3 Filesystem

Find JSON files modified within the last day in the current directory. Using `follow_links` ensures symbolic links are followed like they were normal directories and files.

```
use walkdir::WalkDir;

fn main() -> Result<()> {
    for entry in WalkDir::new(".")
            .follow_links(true)
            .into_iter()
            .filter_map(|e| e.ok()) {
        let f_name = entry.file_name().to_string_lossy();
        let sec = entry.metadata()?.modified()?;

        if f_name.ends_with(".json") && sec.elapsed()?.as_secs() < 86400 {
            println!("{}", f_name);
        }
    }

    Ok(())
}
```

Traverse directories while skipping dotfiles

walkdir v2.3.3 Filesystem

Uses `filter_entry` to descend recursively into entries passing the `is_not_hidden` predicate thus skipping hidden files and directories. `Iterator::filter` applies to each `WalkDir::DirEntry` even if the parent is a hidden directory.

Root dir `"."` yields through `WalkDir::depth` usage in `is_not_hidden` predicate.

```
use walkdir::{DirEntry, WalkDir};

fn is_not_hidden(entry: &DirEntry) -> bool {
    entry
         .file_name()
         .to_str()
         .map(|s| entry.depth() == 0 || !s.starts_with("."))
         .unwrap_or(false)
}

fn main() {
    WalkDir::new(".")
        .into_iter()
        .filter_entry(|e| is_not_hidden(e))
        .filter_map(|v| v.ok())
        .for_each(|x| println!("{}", x.path().display()));
}
```

Recursively calculate file sizes at given depth

walkdir v2.3.3 Filesystem

Recursion depth can be flexibly set by `WalkDir::min_depth` & `WalkDir::max_depth` methods. Calculates sum of all file sizes to 3 subfolders depth, ignoring files in the root folder.

```
use walkdir::WalkDir;

fn main() {
    let total_size = WalkDir::new(".")
        .min_depth(1)
        .max_depth(3)
        .into_iter()
        .filter_map(|entry| entry.ok())
        .filter_map(|entry| entry.metadata().ok())
        .filter(|metadata| metadata.is_file())
        .fold(0, |acc, m| acc + m.len());

    println!("Total size: {} bytes.", total_size);
}
```

Find all png files recursively

glob v0.3.1 Filesystem

Recursively find all PNG files in the current directory. In this case, the `**` pattern matches the current directory and all subdirectories.

Use the `**` pattern in any path portion. For example, `/media/**/*.png` matches all PNGs in `media` and it's subdirectories.

```
use glob::glob;

fn main() -> Result<()> {
    for entry in glob("**/*.png")? {
        println!("{}", entry?.display());
    }

    Ok(())
}
```

Find all files with given pattern ignoring filename case.

glob v0.3.1 Filesystem

Find all image files in the `/media/` directory matching the `img_[0-9]*.png` pattern.

A custom `MatchOptions` struct is passed to the `glob_with` function making the glob pattern case insensitive while keeping the other options `Default`.

```
use error_chain::error_chain;
use glob::{glob_with, MatchOptions};

error_chain! {
    foreign_links {
        Glob(glob::GlobError);
        Pattern(glob::PatternError);
    }
}

fn main() -> Result<()> {
    let options = MatchOptions {
        case_sensitive: false,
        ..Default::default()
    };

    for entry in glob_with("/media/img_[0-9]*.png", options)? {
        println!("{}", entry?.display());
    }

    Ok(())
}
```

Hardware Support

Recipe	Crates	Categories
Check number of logical cpu cores	num_cpus v1.16.0	Hardware support

Processor

Check number of logical cpu cores

num_cpus v1.16.0 Hardware support

Shows the number of logical CPU cores in current machine using [`num_cpus::get`].

```
fn main() {
    println!("Number of logical cores is {}", num_cpus::get());
}
```

Memory Management

Recipe	Crates	Categories
Declare lazily evaluated constant	lazy_static v1.4.0	Caching Rust patterns

Constants

Declare lazily evaluated constant

lazy_static v1.4.0 Caching Rust patterns

Declares a lazily evaluated constant `HashMap`. The `HashMap` will be evaluated once and stored behind a global static reference.

```
use lazy_static::lazy_static;
use std::collections::HashMap;

lazy_static! {
    static ref PRIVILEGES: HashMap<&'static str, Vec<&'static str>> = {
        let mut map = HashMap::new();
        map.insert("James", vec!["user", "admin"]);
        map.insert("Jim", vec!["user"]);
        map
    };
}

fn show_access(name: &str) {
    let access = PRIVILEGES.get(name);
    println!("{}: {:?}", name, access);
}

fn main() {
    let access = PRIVILEGES.get("James");
    println!("James: {:?}", access);

    show_access("Jim");
}
```

Networking

Recipe	Crates	Categories
Listen on unused port TCP/IP	std 1.29.1	Net

Server

Listen on unused port TCP/IP

std 1.29.1 Net

In this example, the port is displayed on the console, and the program will listen until a request is made. `SocketAddrV4` assigns a random port when setting port to 0.

```rust
use std::net::{SocketAddrV4, Ipv4Addr, TcpListener};
use std::io::{Read, Error};

fn main() -> Result<(), Error> {
    let loopback = Ipv4Addr::new(127, 0, 0, 1);
    let socket = SocketAddrV4::new(loopback, 0);
    let listener = TcpListener::bind(socket)?;
    let port = listener.local_addr()?;
    println!("Listening on {}, access this port to end the program", port);
    let (mut tcp_stream, addr) = listener.accept()?; //block  until requested
    println!("Connection received! {:?} is sending data.", addr);
    let mut input = String::new();
    let _ = tcp_stream.read_to_string(&mut input)?;
    println!("{:?} says {}", addr, input);
    Ok(())
}
```

Operating System

Recipe	Crates	Categories
Run an external command and process stdout	regex v1.9.1	OS, Text processing
Run an external command passing it stdin and check for an error code	regex v1.9.1	OS, Text processing
Run piped external commands	std 1.29.1	OS
Redirect both stdout and stderr of child process to the same file	std 1.29.1	OS
Continuously process child process' outputs	std 1.29.1	OS, Text processing
Read environment variable	std 1.29.1	OS

External Command

Run an external command and process stdout

regex v1.9.1 OS Text processing

Runs `git log --oneline` as an external `Command` and inspects its `Output` using `Regex` to get the hash and message of the last 5 commits.

```
use std::process::Command;
use regex::Regex;

#[derive(PartialEq, Default, Clone, Debug)]
struct Commit {
    hash: String,
    message: String,
}

fn main() -> Result<()> {
    let output = Command::new("git").arg("log").arg("--oneline").output()?;

    if !output.status.success() {
        error_chain::bail!("Command executed with failing error code");
    }

    let pattern = Regex::new(r"(?x)
                               ([0-9a-fA-F]+) # commit hash
                               (.*)           # The commit message")?;

    String::from_utf8(output.stdout)?
        .lines()
        .filter_map(|line| pattern.captures(line))
        .map(|cap| {
                 Commit {
                     hash: cap[1].to_string(),
                     message: cap[2].trim().to_string(),
                 }
             })
        .take(5)
        .for_each(|x| println!("{:?}", x));

    Ok(())
}
```

Run an external command passing it stdin and check for an error code

std 1.29.1 OS

Opens the `python` interpreter using an external `Command` and passes it a python statement for execution. `Output` of statement is then parsed.

```
use std::collections::HashSet;
use std::io::Write;
use std::process::{Command, Stdio};

fn main() -> Result<()> {
    let mut child = Command::new("python").stdin(Stdio::piped())
        .stderr(Stdio::piped())
        .stdout(Stdio::piped())
        .spawn()?;

    child.stdin
        .as_mut()
        .ok_or("Child process stdin has not been captured!")?
        .write_all(b"import this; copyright(); credits(); exit()")?;

    let output = child.wait_with_output()?;

    if output.status.success() {
        let raw_output = String::from_utf8(output.stdout)?;
        let words = raw_output.split_whitespace()
            .map(|s| s.to_lowercase())
            .collect::<HashSet<_>>();
        println!("Found {} unique words:", words.len());
        println!("{:#?}", words);
        Ok(())
    } else {
        let err = String::from_utf8(output.stderr)?;
        error_chain::bail!("External command failed:\n {}", err)
    }
}
```

Run piped external commands

std 1.29.1 OS

Shows up to the 10th biggest files and subdirectories in the current working directory. It is equivalent to running: `du -ah . | sort -hr | head -n 10`.

`Command`s represent a process. Output of a child process is captured with a `Stdio::piped` between parent and child.

```
use std::process::{Command, Stdio};

fn main() -> Result<()> {
    let directory = std::env::current_dir()?;
    let mut du_output_child = Command::new("du")
        .arg("-ah")
        .arg(&directory)
        .stdout(Stdio::piped())
        .spawn()?;

    if let Some(du_output) = du_output_child.stdout.take() {
        let mut sort_output_child = Command::new("sort")
            .arg("-hr")
            .stdin(du_output)
            .stdout(Stdio::piped())
            .spawn()?;

        du_output_child.wait()?;

        if let Some(sort_output) = sort_output_child.stdout.take() {
            let head_output_child = Command::new("head")
                .args(&["-n", "10"])
                .stdin(sort_output)
                .stdout(Stdio::piped())
                .spawn()?;

            let head_stdout = head_output_child.wait_with_output()?;

            sort_output_child.wait()?;

            println!(
                "Top 10 biggest files and directories in '{}':\n{}",
                directory.display(),
                String::from_utf8(head_stdout.stdout).unwrap()
            );
        }
    }

    Ok(())
}
```

Redirect both stdout and stderr of child process to the same file

std 1.29.1 OS

Spawns a child process and redirects `stdout` and `stderr` to the same file. It follows the same idea as run piped external commands, however `process::Stdio` writes to a specified file. `File::try_clone` references the same file handle for `stdout` and `stderr`. It will ensure that both handles write with the same cursor position.

The below recipe is equivalent to run the Unix shell command `ls . oops >out.txt 2>&1`.

```
use std::fs::File;
use std::io::Error;
use std::process::{Command, Stdio};

fn main() -> Result<(), Error> {
    let outputs = File::create("out.txt")?;
    let errors = outputs.try_clone()?;

    Command::new("ls")
        .args(&[".", "oops"])
        .stdout(Stdio::from(outputs))
        .stderr(Stdio::from(errors))
        .spawn()?
        .wait_with_output()?;

    Ok(())
}
```

Continuously process child process' outputs

std 1.29.1 OS

In Run an external command and process stdout, processing doesn't start until external `Command` is finished. The recipe below calls `Stdio::piped` to create a pipe, and reads `stdout` continuously as soon as the `BufReader` is updated.

The below recipe is equivalent to the Unix shell command `journalctl | grep usb`.

```
use std::process::{Command, Stdio};
use std::io::{BufRead, BufReader, Error, ErrorKind};

fn main() -> Result<(), Error> {
    let stdout = Command::new("journalctl")
        .stdout(Stdio::piped())
        .spawn()?
        .stdout
        .ok_or_else(|| Error::new(ErrorKind::Other,"Could not capture standard
output."))?;

    let reader = BufReader::new(stdout);

    reader
        .lines()
        .filter_map(|line| line.ok())
        .filter(|line| line.find("usb").is_some())
        .for_each(|line| println!("{}", line));

     Ok(())
}
```

Read Environment Variable

std 1.29.1 OS

Reads an environment variable via std::env::var.

```rust
use std::env;
use std::fs;
use std::io::Error;

fn main() -> Result<(), Error> {
    // read `config_path` from the environment variable `CONFIG`.
    // If `CONFIG` isn't set, fall back to a default config path.
    let config_path = env::var("CONFIG")
        .unwrap_or("/etc/myapp/config".to_string());

    let config: String = fs::read_to_string(config_path)?;
    println!("Config: {}", config);

    Ok(())
}
```

Science

Mathematics

Recipe	Crates	Categories
Vector Norm	ndarray v0.15.6	Science
Adding matrices	ndarray v0.15.6	Science
Multiplying matrices	ndarray v0.15.6	Science
Multiply a scalar with a vector with a matrix	ndarray v0.15.6	Science
Invert matrix	nalgebra v0.32.3	Science
Calculating the side length of a triangle	std 1.29.1	Science
Verifying tan is equal to sin divided by cos	std 1.29.1	Science
Distance between two points on the Earth	std 1.29.1	Science
Creating complex numbers	num v0.4.0	Science
Adding complex numbers	num v0.4.0	Science
Mathematical functions on complex numbers	num v0.4.0	Science
Measures of central tendency	std 1.29.1	Science
Computing standard deviation	std 1.29.1	Science
Big integers	num v0.4.0	Science

Mathematics

Recipe	Crates	Categories
Vector Norm	ndarray v0.15.6	Science
Adding matrices	ndarray v0.15.6	Science
Multiplying matrices	ndarray v0.15.6	Science
Multiply a scalar with a vector with a matrix	ndarray v0.15.6	Science

Recipe	Crates	Categories
Invert matrix	nalgebra v0.32.3	Science
Calculating the side length of a triangle	std 1.29.1	Science
Verifying tan is equal to sin divided by cos	std 1.29.1	Science
Distance between two points on the Earth	std 1.29.1	Science
Creating complex numbers	num v0.4.0	Science
Adding complex numbers	num v0.4.0	Science
Mathematical functions on complex numbers	num v0.4.0	Science
Measures of central tendency	std 1.29.1	Science
Computing standard deviation	std 1.29.1	Science
Big integers	num v0.4.0	Science

Linear Algebra

Adding matrices

ndarray v0.15.6 Science

Creates two 2-D matrices with `ndarray::arr2` and sums them element-wise.

Note the sum is computed as `let sum = &a + &b`. The `&` operator is used to avoid consuming `a` and `b`, making them available later for display. A new array is created containing their sum.

```
use ndarray::arr2;

fn main() {
    let a = arr2(&[[1, 2, 3],
                   [4, 5, 6]]);

    let b = arr2(&[[6, 5, 4],
                   [3, 2, 1]]);

    let sum = &a + &b;

    println!("{}", a);
    println!("+");
    println!("{}", b);
    println!("=");
    println!("{}", sum);
}
```

Multiplying matrices

ndarray v0.15.6 Science

Creates two matrices with `ndarray::arr2` and performs matrix multiplication on them with `ndarray::ArrayBase::dot`.

```
use ndarray::arr2;

fn main() {
    let a = arr2(&[[1, 2, 3],
                   [4, 5, 6]]);

    let b = arr2(&[[6, 3],
                   [5, 2],
                   [4, 1]]);

    println!("{}", a.dot(&b));
}
```

Multiply a scalar with a vector with a matrix

ndarray v0.15.6 Science

Creates a 1-D array (vector) with `ndarray::arr1` and a 2-D array (matrix) with `ndarray::arr2`.

First, a scalar is multiplied by the vector to get another vector. Then, the matrix is multiplied by the new vector with `ndarray::Array2::dot`. (Matrix multiplication is performed using `dot`, while the `*` operator performs element-wise multiplication.)

In `ndarray`, 1-D arrays can be interpreted as either row or column vectors depending on context. If representing the orientation of a vector is important, a 2-D array with one row or one column must be used instead. In this example, the vector is a 1-D array on the right-hand side, so `dot` handles it as a column vector.

```
use ndarray::{arr1, arr2, Array1};

fn main() {
    let scalar = 4;

    let vector = arr1(&[1, 2, 3]);

    let matrix = arr2(&[[4, 5, 6],
                        [7, 8, 9]]);

    let new_vector: Array1<_> = scalar * vector;
    println!("{}", new_vector);

    let new_matrix = matrix.dot(&new_vector);
    println!("{}", new_matrix);
}
```

Vector comparison

ndarray v0.15.6

The ndarray crate supports a number of ways to create arrays -- this recipe creates `ndarray::Array`s from `std::Vec` using `from`. Then, it sums the arrays element-wise.

This recipe contains an example of comparing two floating-point vectors element-wise. Floating-point numbers are often stored inexactly, making exact comparisons difficult. However, the `assert_abs_diff_eq!` macro from the `approx` crate allows for convenient element-wise

comparisons. To use the `approx` crate with `ndarray`, the `approx` feature must be added to the `ndarray` dependency in `Cargo.toml`. For example, `ndarray = { version = "0.13", features = ["approx"] }`.

This recipe also contains additional ownership examples. Here, `let z = a + b` consumes `a` and `b`, updates `a` with the result, then moves ownership to `z`. Alternatively, `let w = &c + &d` creates a new vector without consuming `c` or `d`, allowing their modification later. See Binary Operators With Two Arrays for additional detail.

```
use approx::assert_abs_diff_eq;
use ndarray::Array;

fn main() {
  let a = Array::from(vec![1., 2., 3., 4., 5.]);
  let b = Array::from(vec![5., 4., 3., 2., 1.]);
  let mut c = Array::from(vec![1., 2., 3., 4., 5.]);
  let mut d = Array::from(vec![5., 4., 3., 2., 1.]);

  let z = a + b;
  let w =  &c + &d;

  assert_abs_diff_eq!(z, Array::from(vec![6., 6., 6., 6., 6.]));

  println!("c = {}", c);
  c[0] = 10.;
  d[1] = 10.;

  assert_abs_diff_eq!(w, Array::from(vec![6., 6., 6., 6., 6.]));

}
```

Vector norm

ndarray v0.15.6

This recipe demonstrates use of the `Array1` type, `ArrayView1` type, `fold` method, and `dot` method in computing the l1 and l2 norms of a given vector. + The `l2_norm` function is the simpler of the two, as it computes the square root of the dot product of a vector with itself. + The `l1_norm` function is computed by a `fold` operation that sums the absolute values of the elements. (This could also be performed with `x.mapv(f64::abs).scalar_sum()`, but that would allocate a new array for the result of the `mapv`.)

Note that both `l1_norm` and `l2_norm` take the `ArrayView1` type. This recipe considers vector norms, so the norm functions only need to accept one-dimensional views (hence `ArrayView1`). While the functions could take a parameter of type `&Array1<f64>` instead, that would require the caller to have a reference to an owned array, which is more restrictive than just having access to a view (since a view can be created from any array or view, not just an owned array).

`Array` and `ArrayView` are both type aliases for `ArrayBase`. So, the most general argument type for the caller would be `&ArrayBase<S, Ix1> where S: Data`, because then the caller could use `&array` or `&view` instead of `x.view()`. If the function is part of a public API, that may be a better choice for the benefit of users. For internal functions, the more concise `ArrayView1<f64>` may be preferable.

```
use ndarray::{array, Array1, ArrayView1};

fn l1_norm(x: ArrayView1<f64>) -> f64 {
    x.fold(0., |acc, elem| acc + elem.abs())
}

fn l2_norm(x: ArrayView1<f64>) -> f64 {
    x.dot(&x).sqrt()
}

fn normalize(mut x: Array1<f64>) -> Array1<f64> {
    let norm = l2_norm(x.view());
    x.mapv_inplace(|e| e/norm);
    x
}

fn main() {
    let x = array![1., 2., 3., 4., 5.];
    println!("||x||_2 = {}", l2_norm(x.view()));
    println!("||x||_1 = {}", l1_norm(x.view()));
    println!("Normalizing x yields {:?}", normalize(x));
}
```

Invert matrix

nalgebra v0.32.3 Science

Creates a 3x3 matrix with `nalgebra::Matrix3` and inverts it, if possible.

```
use nalgebra::Matrix3;

fn main() {
    let m1 = Matrix3::new(2.0, 1.0, 1.0, 3.0, 2.0, 1.0, 2.0, 1.0, 2.0);
    println!("m1 = {}", m1);
    match m1.try_inverse() {
        Some(inv) => {
            println!("The inverse of m1 is: {}", inv);
        }
        None => {
            println!("m1 is not invertible!");
        }
    }
}
```

(De)-Serialize a Matrix

ndarray v0.15.6 Science

Serialize and deserialize a matrix to and from JSON. Serialization is taken care of by `serde_json::to_string` and `serde_json::from_str` performs deserialization.

Note that serialization followed by deserialization gives back the original matrix.

```
extern crate nalgebra;
extern crate serde_json;

use nalgebra::DMatrix;

fn main() -> Result<(), std::io::Error> {
    let row_slice: Vec<i32> = (1..5001).collect();
    let matrix = DMatrix::from_row_slice(50, 100, &row_slice);

    // serialize matrix
    let serialized_matrix = serde_json::to_string(&matrix)?;

    // deserialize matrix
    let deserialized_matrix: DMatrix<i32> = serde_json::from_str(&serialized_matrix)?;

    // verify that `deserialized_matrix` is equal to `matrix`
    assert!(deserialized_matrix == matrix);

    Ok(())
}
```

Trigonometry

Calculating the side length of a triangle

std 1.29.1 Science

Calculates the length of the hypotenuse of a right-angle triangle with an angle of 2 radians and opposite side length of 80.

```
fn main() {
    let angle: f64 = 2.0;
    let side_length = 80.0;

    let hypotenuse = side_length / angle.sin();

    println!("Hypotenuse: {}", hypotenuse);
}
```

Verifying tan is equal to sin divided by cos

std 1.29.1 Science

Verifies tan(x) is equal to sin(x)/cos(x) for x = 6.

```
fn main() {
    let x: f64 = 6.0;

    let a = x.tan();
    let b = x.sin() / x.cos();

    assert_eq!(a, b);
}
```

Distance between two points on the Earth

std 1.29.1

By default, Rust provides mathematical float methods such as trigonometric functions, square root, conversion functions between radians and degrees, and so forth.

The following example computes the distance in kilometers between two points on the Earth with the Haversine formula. Points are expressed as pairs of latitude and longitude in degrees. Then, `to_radians` converts them in radian. `sin`, `cos`, `powi` and `sqrt` compute the central angle. Finally, it's possible to calculate the distance.

```
fn main() {
    let earth_radius_kilometer = 6371.0_f64;
    let (paris_latitude_degrees, paris_longitude_degrees) = (48.85341_f64,
-2.34880_f64);
    let (london_latitude_degrees, london_longitude_degrees) = (51.50853_f64,
-0.12574_f64);

    let paris_latitude = paris_latitude_degrees.to_radians();
    let london_latitude = london_latitude_degrees.to_radians();

    let delta_latitude = (paris_latitude_degrees -
london_latitude_degrees).to_radians();
    let delta_longitude = (paris_longitude_degrees -
london_longitude_degrees).to_radians();

    let central_angle_inner = (delta_latitude / 2.0).sin().powi(2)
        + paris_latitude.cos() * london_latitude.cos() * (delta_longitude /
2.0).sin().powi(2);
    let central_angle = 2.0 * central_angle_inner.sqrt().asin();

    let distance = earth_radius_kilometer * central_angle;

    println!(
        "Distance between Paris and London on the surface of Earth is {:.1}
kilometers",
        distance
    );
}
```

Complex numbers

Creating complex numbers

num v0.4.0 Science

Creates complex numbers of type `num::complex::Complex`. Both the real and imaginary part of the complex number must be of the same type.

```
fn main() {
    let complex_integer = num::complex::Complex::new(10, 20);
    let complex_float = num::complex::Complex::new(10.1, 20.1);

    println!("Complex integer: {}", complex_integer);
    println!("Complex float: {}", complex_float);
}
```

Adding complex numbers

num v0.4.0 Science

Performing mathematical operations on complex numbers is the same as on built in types: the numbers in question must be of the same type (i.e. floats or integers).

```
fn main() {
    let complex_num1 = num::complex::Complex::new(10.0, 20.0); // Must use floats
    let complex_num2 = num::complex::Complex::new(3.1, -4.2);

    let sum = complex_num1 + complex_num2;

    println!("Sum: {}", sum);
}
```

Mathematical functions

num v0.4.0 Science

Complex numbers have a range of interesting properties when it comes to how they interact with other mathematical functions, most notibly the family of sine functions as well as the number e. To use these functions with complex numbers, the Complex type has a few built in functions, all of which can be found here: `num::complex::Complex`.

```
use std::f64::consts::PI;
use num::complex::Complex;

fn main() {
    let x = Complex::new(0.0, 2.0*PI);

    println!("e^(2i * pi) = {}", x.exp()); // =~1
}
```

Statistics

Measures of central tendency

std 1.29.1 Science

These examples calculate measures of central tendency for a data set contained within a Rust array. There may be no mean, median or mode to calculate for an empty set of data, so each function returns an `Option` to be handled by the caller.

The first example calculates the mean (the sum of all measurements divided by the number of measurements in the set) by producing an iterator of references over the data, and using `sum` and `len` to determine the total value and count of values respectively.

```
fn main() {
    let data = [3, 1, 6, 1, 5, 8, 1, 8, 10, 11];

    let sum = data.iter().sum::<i32>() as f32;
    let count = data.len();

    let mean = match count {
       positive if positive > 0 => Some(sum  / count as f32),
       _ => None
    };

    println!("Mean of the data is {:?}", mean);
}
```

The second example calculates the median using the quickselect algorithm, which avoids a full `sort` by sorting only partitions of the data set known to possibly contain the median. This uses `cmp` and `Ordering` to succinctly decide the next partition to examine, and `split_at` to choose an arbitrary pivot for the next partition at each step.

```
use std::cmp::Ordering;

fn partition(data: &[i32]) -> Option<(Vec<i32>, i32, Vec<i32>)> {
    match data.len() {
        0 => None,
        _ => {
            let (pivot_slice, tail) = data.split_at(1);
            let pivot = pivot_slice[0];
            let (left, right) = tail.iter()
                .fold((vec![], vec![]), |mut splits, next| {
                    {
                        let (ref mut left, ref mut right) = &mut splits;
                        if next < &pivot {
                            left.push(*next);
                        } else {
                            right.push(*next);
                        }
                    }
                    splits
                });

            Some((left, pivot, right))
        }
    }
}

fn select(data: &[i32], k: usize) -> Option<i32> {
    let part = partition(data);

    match part {
        None => None,
        Some((left, pivot, right)) => {
            let pivot_idx = left.len();

            match pivot_idx.cmp(&k) {
                Ordering::Equal => Some(pivot),
                Ordering::Greater => select(&left, k),
                Ordering::Less => select(&right, k - (pivot_idx + 1)),
            }
        },
    }
}

fn median(data: &[i32]) -> Option<f32> {
    let size = data.len();

    match size {
        even if even % 2 == 0 => {
            let fst_med = select(data, (even / 2) - 1);
            let snd_med = select(data, even / 2);

            match (fst_med, snd_med) {
                (Some(fst), Some(snd)) => Some((fst + snd) as f32 / 2.0),
                _ => None
            }
        },
        odd => select(data, odd / 2).map(|x| x as f32)
    }
}

fn main() {
    let data = [3, 1, 6, 1, 5, 8, 1, 8, 10, 11];

    let part = partition(&data);
    println!("Partition is {:?}", part);

    let sel = select(&data, 5);
```

```
    println!("Selection at ordered index {} is {:?}", 5, sel);

    let med = median(&data);
    println!("Median is {:?}", med);
}
```

The final example calculates the mode using a mutable `HashMap` to collect counts of each distinct integer from the set, using a `fold` and the `entry` API. The most frequent value in the `HashMap` surfaces with `max_by_key`.

```
use std::collections::HashMap;

fn main() {
    let data = [3, 1, 6, 1, 5, 8, 1, 8, 10, 11];

    let frequencies = data.iter().fold(HashMap::new(), |mut freqs, value| {
        *freqs.entry(value).or_insert(0) += 1;
        freqs
    });

    let mode = frequencies
        .into_iter()
        .max_by_key(|&(_, count)| count)
        .map(|(value, _)| *value);

    println!("Mode of the data is {:?}", mode);
}
```

Standard deviation

std 1.29.1 Science

This example calculates the standard deviation and z-score of a set of measurements.

The standard deviation is defined as the square root of the variance (here calculated with f32's [`sqrt`], where the variance is the `sum` of the squared difference between each measurement and the [`mean`], divided by the number of measurements.

The z-score is the number of standard deviations a single measurement spans away from the [`mean`] of the data set.

```rust
fn mean(data: &[i32]) -> Option<f32> {
    let sum = data.iter().sum::<i32>() as f32;
    let count = data.len();

    match count {
        positive if positive > 0 => Some(sum / count as f32),
        _ => None,
    }
}

fn std_deviation(data: &[i32]) -> Option<f32> {
    match (mean(data), data.len()) {
        (Some(data_mean), count) if count > 0 => {
            let variance = data.iter().map(|value| {
                let diff = data_mean - (*value as f32);

                diff * diff
            }).sum::<f32>() / count as f32;

            Some(variance.sqrt())
        },
        _ => None
    }
}

fn main() {
    let data = [3, 1, 6, 1, 5, 8, 1, 8, 10, 11];

    let data_mean = mean(&data);
    println!("Mean is {:?}", data_mean);

    let data_std_deviation = std_deviation(&data);
    println!("Standard deviation is {:?}", data_std_deviation);

    let zscore = match (data_mean, data_std_deviation) {
        (Some(mean), Some(std_deviation)) => {
            let diff = data[4] as f32 - mean;

            Some(diff / std_deviation)
        },
        _ => None
    };
    println!("Z-score of data at index 4 (with value {}) is {:?}", data[4], zscore);
}
```

Miscellaneous

Big integers

num v0.4.0 Science

Calculation for integers exceeding 128 bits are possible with `BigInt`.

```
use num::bigint::{BigInt, ToBigInt};

fn factorial(x: i32) -> BigInt {
    if let Some(mut factorial) = 1.to_bigint() {
        for i in 1..=x {
            factorial = factorial * i;
        }
        factorial
    }
    else {
        panic!("Failed to calculate factorial!");
    }
}

fn main() {
    println!("{}! equals {}", 100, factorial(100));
}
```

Text Processing

Recipe	Crates	Categories
Collect Unicode Graphemes	unicode-segmentation v1.10.1	Encoding
Verify and extract login from an email address	regex v1.9.1 lazy_static v1.4.0	Text processing
Extract a list of unique #Hashtags from a text	regex v1.9.1 lazy_static v1.4.0	Text processing
Extract phone numbers from text	regex v1.9.1	Text processing
Filter a log file by matching multiple regular expressions	regex v1.9.1	Text processing
Replace all occurrences of one text pattern with another pattern.	regex v1.9.1 lazy_static v1.4.0	Text processing
Implement the `FromStr` trait for a custom `struct`	std 1.29.1	Text processing

Regular Expressions

Verify and extract login from an email address

regex v1.9.1 lazy_static v1.4.0 Text processing

Validates that an email address is formatted correctly, and extracts everything before the @ symbol.

```
use lazy_static::lazy_static;

use regex::Regex;

fn extract_login(input: &str) -> Option<&str> {
    lazy_static! {
        static ref RE: Regex = Regex::new(r"(?x)
            ^(?P<login>[^@\s]+)@
            ([[:word:]]+\.)*
            [[:word:]]+$
            ").unwrap();
    }
    RE.captures(input).and_then(|cap| {
        cap.name("login").map(|login| login.as_str())
    })
}

fn main() {
    assert_eq!(extract_login(r"I❤email@example.com"), Some(r"I❤email"));
    assert_eq!(
        extract_login(r"sdf+sdsfsd.as.sdsd@jhkk.d.rl"),
        Some(r"sdf+sdsfsd.as.sdsd")
    );
    assert_eq!(extract_login(r"More@Than@One@at.com"), None);
    assert_eq!(extract_login(r"Not an email@email"), None);
}
```

Extract a list of unique #Hashtags from a text

regex v1.9.1 lazy_static v1.4.0 Text processing

Extracts, sorts, and deduplicates list of hashtags from text.

The hashtag regex given here only catches Latin hashtags that start with a letter. The complete twitter hashtag regex is much more complicated.

```
use lazy_static::lazy_static;

use regex::Regex;
use std::collections::HashSet;

fn extract_hashtags(text: &str) -> HashSet<&str> {
    lazy_static! {
        static ref HASHTAG_REGEX : Regex = Regex::new(
                r"\#[a-zA-Z][0-9a-zA-Z_]*"
            ).unwrap();
    }
    HASHTAG_REGEX.find_iter(text).map(|mat| mat.as_str()).collect()
}

fn main() {
    let tweet = "Hey #world, I just got my new #dog, say hello to Till. #dog #forever
#2 #_ ";
    let tags = extract_hashtags(tweet);
    assert!(tags.contains("#dog") && tags.contains("#forever") &&
tags.contains("#world"));
    assert_eq!(tags.len(), 3);
}
```

Extract phone numbers from text

regex v1.9.1 Text processing

Processes a string of text using `Regex::captures_iter` to capture multiple phone numbers. The example here is for US convention phone numbers.

```
use regex::Regex;
use std::fmt;

struct PhoneNumber<'a> {
    area: &'a str,
    exchange: &'a str,
    subscriber: &'a str,
}

impl<'a> fmt::Display for PhoneNumber<'a> {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "1 ({}) {}-{}", self.area, self.exchange, self.subscriber)
    }
}

fn main() -> Result<()> {
    let phone_text = "
    +1 505 881 9292 (v) +1 505 778 2212 (c) +1 505 881 9297 (f)
    (202) 991 9534
    Alex 5553920011
    1 (800) 233-2010
    1.299.339.1020";

    let re = Regex::new(
        r#"(?x)
          (?:\+?1)?                       # Country Code Optional
          [\s\.]?
          (([2-9]\d{2})|\(([2-9]\d{2})\)) # Area Code
          [\s\.\-]?
          ([2-9]\d{2})                    # Exchange Code
          [\s\.\-]?
          (\d{4})                         # Subscriber Number"#,
    )?;

    let phone_numbers = re.captures_iter(phone_text).filter_map(|cap| {
        let groups = (cap.get(2).or(cap.get(3)), cap.get(4), cap.get(5));
        match groups {
            (Some(area), Some(ext), Some(sub)) => Some(PhoneNumber {
                area: area.as_str(),
                exchange: ext.as_str(),
                subscriber: sub.as_str(),
            }),
            _ => None,
        }
    });

    assert_eq!(
        phone_numbers.map(|m| m.to_string()).collect::<Vec<_>>(),
        vec![
            "1 (505) 881-9292",
            "1 (505) 778-2212",
            "1 (505) 881-9297",
            "1 (202) 991-9534",
            "1 (555) 392-0011",
            "1 (800) 233-2010",
            "1 (299) 339-1020",
        ]
    );

    Ok(())
}
```

Filter a log file by matching multiple regular expressions

regex v1.9.1 Text processing

Reads a file named `application.log` and only outputs the lines containing "version X.X.X", some IP address followed by port 443 (e.g. "192.168.0.1:443"), or a specific warning.

A `regex::RegexSetBuilder` composes a `regex::RegexSet`. Since backslashes are very common in regular expressions, using raw string literals makes them more readable.

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

fn main() -> Result<()> {
    let log_path = "application.log";
    let buffered = BufReader::new(File::open(log_path)?);

    let set = RegexSetBuilder::new(&[
        r#"version "\d\.\d\.\d""#,
        r#"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}:443"#,
        r#"warning.*timeout expired"#,
    ]).case_insensitive(true)
        .build()?;

    buffered
        .lines()
        .filter_map(|line| line.ok())
        .filter(|line| set.is_match(line.as_str()))
        .for_each(|x| println!("{}", x));

    Ok(())
}
```

Replace all occurrences of one text pattern with another pattern.

regex v1.9.1 lazy_static v1.4.0 Text processing

Replaces all occurrences of the standard ISO 8601 *YYYY-MM-DD* date pattern with the equivalent American English date with slashes. For example `2013-01-15` becomes `01/15/2013`.

The method `Regex::replace_all` replaces all occurrences of the whole regex. `&str` implements the `Replacer` trait which allows variables like `$abcde` to refer to corresponding named capture groups `(?P<abcde>REGEX)` from the search regex. See the replacement string syntax for examples and escaping detail.

```
use lazy_static::lazy_static;

use std::borrow::Cow;
use regex::Regex;

fn reformat_dates(before: &str) -> Cow<str> {
    lazy_static! {
        static ref ISO8601_DATE_REGEX : Regex = Regex::new(
            r"(?P<y>\d{4})-(?P<m>\d{2})-(?P<d>\d{2})"
            ).unwrap();
    }
    ISO8601_DATE_REGEX.replace_all(before, "$m/$d/$y")
}

fn main() {
    let before = "2012-03-14, 2013-01-15 and 2014-07-05";
    let after = reformat_dates(before);
    assert_eq!(after, "03/14/2012, 01/15/2013 and 07/05/2014");
}
```

String Parsing

Collect Unicode Graphemes

unicode-segmentation v1.10.1 Text processing

Collect individual Unicode graphemes from UTF-8 string using the `UnicodeSegmentation::graphemes` function from the `unicode-segmentation` crate.

```
use unicode_segmentation::UnicodeSegmentation;

fn main() {
    let name = "José Guimarães\r\n";
    let graphemes = UnicodeSegmentation::graphemes(name, true)
        .collect::<Vec<&str>>();
    assert_eq!(graphemes[3], "é");
}
```

Implement the `FromStr` trait for a custom `struct`

std 1.29.1 Text processing

Creates a custom struct `RGB` and implements the `FromStr` trait to convert a provided color hex code into its RGB color code.

```rust
use std::str::FromStr;

#[derive(Debug, PartialEq)]
struct RGB {
    r: u8,
    g: u8,
    b: u8,
}

impl FromStr for RGB {
    type Err = std::num::ParseIntError;

    // Parses a color hex code of the form '#rRgGbB..' into an
    // instance of 'RGB'
    fn from_str(hex_code: &str) -> Result<Self, Self::Err> {

        // u8::from_str_radix(src: &str, radix: u32) converts a string
        // slice in a given base to u8
        let r: u8 = u8::from_str_radix(&hex_code[1..3], 16)?;
        let g: u8 = u8::from_str_radix(&hex_code[3..5], 16)?;
        let b: u8 = u8::from_str_radix(&hex_code[5..7], 16)?;

        Ok(RGB { r, g, b })
    }
}

fn main() {
    let code: &str = &r"#fa7268";
    match RGB::from_str(code) {
        Ok(rgb) => {
            println!(
                r"The RGB color code is: R: {} G: {} B: {}",
                rgb.r, rgb.g, rgb.b
            );
        }
        Err(_) => {
            println!("{} is not a valid color hex code!", code);
        }
    }

    // test whether from_str performs as expected
    assert_eq!(
        RGB::from_str(&r"#fa7268").unwrap(),
        RGB {
            r: 250,
            g: 114,
            b: 104
        }
    );
}
```

Web Programming

Scraping Web Pages

Recipe	Crates	Categories
Extract all links from a webpage HTML	reqwest v0.11.18 select v0.6.0	Net

Recipe	Crates	Categories
Check webpage for broken links	reqwest v0.11.18 select v0.6.0 url v2.4.0	Net
Extract all unique links from a MediaWiki markup	reqwest v0.11.18 regex v1.9.1	Net

Uniform Resource Locations (URL)

Recipe	Crates	Categories
Parse a URL from a string to a `Url` type	url v2.4.0	Net
Create a base URL by removing path segments	url v2.4.0	Net
Create new URLs from a base URL	url v2.4.0	Net
Extract the URL origin (scheme / host / port)	url v2.4.0	Net
Remove fragment identifiers and query pairs from a URL	url v2.4.0	Net

Media Types (MIME)

Recipe	Crates	Categories
Get MIME type from string	mime v1.2.2	Encoding
Get MIME type from filename	mime v1.2.2	Encoding
Parse the MIME type of a HTTP response	mime v1.2.2 reqwest v0.11.18	Net Encoding

Clients

Recipe	Crates	Categories
Make a HTTP GET request	reqwest v0.11.18	Net
Query the GitHub API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Check if an API resource exists	reqwest v0.11.18	Net
Create and delete Gist with GitHub API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Consume a paginated RESTful API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Download a file to a temporary directory	reqwest v0.11.18 tempdir v0.3.7	Net Filesystem
Make a partial download with HTTP range headers	reqwest v0.11.18	Net

Recipe	Crates	Categories
POST a file to paste-rs	reqwest v0.11.18	Net

Web Authentication

Recipe	Crates	Categories
Basic Authentication	reqwest v0.11.18	Net

Extracting Links

Extract all links from a webpage HTML

reqwest v0.11.18 select v0.6.0 Net

Use `reqwest::get` to perform a HTTP GET request and then use `Document::from_read` to parse the response into a HTML document. `find` with the criteria of `Name` is "a" retrieves all links. Call `filter_map` on the `Selection` retrieves URLs from links that have the "href" `attr` (attribute).

```
use error_chain::error_chain;
use select::document::Document;
use select::predicate::Name;

error_chain! {
      foreign_links {
          ReqError(reqwest::Error);
          IoError(std::io::Error);
      }
}

#[tokio::main]
async fn main() -> Result<()> {
  let res = reqwest::get("https://www.rust-lang.org/en-US/")
    .await?
    .text()
    .await?;

  Document::from(res.as_str())
    .find(Name("a"))
    .filter_map(|n| n.attr("href"))
    .for_each(|x| println!("{}", x));

  Ok(())
}
```

Check a webpage for broken links

reqwest v0.11.18 select v0.6.0 url v2.4.0 Net

Call `get_base_url` to retrieve the base URL. If the document has a base tag, get the href `attr` from base tag. `Position::BeforePath` of the original URL acts as a default.

Iterates through links in the document and creates a `tokio::spawn` task that will parse an individual link with `url::ParseOptions` and `Url::parse`). The task makes a request to the links with reqwest and verifies `StatusCode`. Then the tasks `await` completion before ending the program.

```
use error_chain::error_chain;
use reqwest::StatusCode;
use select::document::Document;
use select::predicate::Name;
use std::collections::HashSet;
use url::{Position, Url};

error_chain! {
  foreign_links {
      ReqError(reqwest::Error);
      IoError(std::io::Error);
      UrlParseError(url::ParseError);
      JoinError(tokio::task::JoinError);
  }
}

async fn get_base_url(url: &Url, doc: &Document) -> Result<Url> {
  let base_tag_href = doc.find(Name("base")).filter_map(|n| n.attr("href")).nth(0);
  let base_url =
    base_tag_href.map_or_else(|| Url::parse(&url[..Position::BeforePath]),
Url::parse)?;
  Ok(base_url)
}

async fn check_link(url: &Url) -> Result<bool> {
  let res = reqwest::get(url.as_ref()).await?;
  Ok(res.status() != StatusCode::NOT_FOUND)
}

#[tokio::main]
async fn main() -> Result<()> {
  let url = Url::parse("https://www.rust-lang.org/en-US/")?;
  let res = reqwest::get(url.as_ref()).await?.text().await?;
  let document = Document::from(res.as_str());
  let base_url = get_base_url(&url, &document).await?;
  let base_parser = Url::options().base_url(Some(&base_url));
  let links: HashSet<Url> = document
    .find(Name("a"))
    .filter_map(|n| n.attr("href"))
    .filter_map(|link| base_parser.parse(link).ok())
    .collect();
    let mut tasks = vec![];

    for link in links {
        tasks.push(tokio::spawn(async move {
            if check_link(&link).await.unwrap() {
                println!("{} is OK", link);
            } else {
                println!("{} is Broken", link);
            }
        }));
    }

    for task in tasks {
        task.await?
    }

  Ok(())
}
```

Extract all unique links from a MediaWiki markup

reqwest v0.11.18 regex v1.9.1 Net

Pull the source of a MediaWiki page using `reqwest::get` and then look for all entries of internal and external links with `Regex::captures_iter`. Using `Cow` avoids excessive `String` allocations.

MediaWiki link syntax is described here.

```
use lazy_static::lazy_static;
use regex::Regex;
use std::borrow::Cow;
use std::collections::HashSet;
use std::error::Error;

fn extract_links(content: &str) -> HashSet<Cow<str>> {
  lazy_static! {
    static ref WIKI_REGEX: Regex = Regex::new(
      r"(?x)
                \[\[(?P<internal>[^\[\]|]*)[^\[\]]*\]\]    # internal links
                |
                (url=|URL\||\[)(?P<external>http.*?)[ \|}] # external links
            "
    )
    .unwrap();
  }

  let links: HashSet<_> = WIKI_REGEX
    .captures_iter(content)
    .map(|c| match (c.name("internal"), c.name("external")) {
      (Some(val), None) => Cow::from(val.as_str().to_lowercase()),
      (None, Some(val)) => Cow::from(val.as_str()),
      _ => unreachable!(),
    })
    .collect();

  links
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn Error>> {
  let content = reqwest::get(
    "https://en.wikipedia.org/w/index.php?
title=Rust_(programming_language)&action=raw",
  )
  .await?
  .text()
  .await?;

  println!("{:#?}", extract_links(content.as_str()));

  Ok(())
}
```

Uniform Resource Location

Parse a URL from a string to a `Url` type

url v2.4.0 Net

The `parse` method from the `url` crate validates and parses a `&str` into a `Url` struct. The input string may be malformed so this method returns `Result<Url, ParseError>`.

Once the URL has been parsed, it can be used with all of the methods in the `Url` type.

```
use url::{Url, ParseError};

fn main() -> Result<(), ParseError> {
    let s = "https://github.com/rust-lang/rust/issues?labels=E-easy&state=open";

    let parsed = Url::parse(s)?;
    println!("The path part of the URL is: {}", parsed.path());

    Ok(())
}
```

Create a base URL by removing path segments

A base URL includes a protocol and a domain. Base URLs have no folders, files or query strings. Each of those items are stripped out of the given URL. `PathSegmentsMut::clear` removes paths and `Url::set_query` removes query string.

```
use url::Url;

fn main() -> Result<()> {
    let full = "https://github.com/rust-lang/cargo?asdf";

    let url = Url::parse(full)?;
    let base = base_url(url)?;

    assert_eq!(base.as_str(), "https://github.com/");
    println!("The base of the URL is: {}", base);

    Ok(())
}

fn base_url(mut url: Url) -> Result<Url> {
    match url.path_segments_mut() {
        Ok(mut path) => {
            path.clear();
        }
        Err(_) => {
            return Err(Error::from_kind(ErrorKind::CannotBeABase));
        }
    }

    url.set_query(None);

    Ok(url)
}
```

Create new URLs from a base URL

The `join` method creates a new URL from a base and relative path.

```
use url::{Url, ParseError};

fn main() -> Result<(), ParseError> {
    let path = "/rust-lang/cargo";

    let gh = build_github_url(path)?;

    assert_eq!(gh.as_str(), "https://github.com/rust-lang/cargo");
    println!("The joined URL is: {}", gh);

    Ok(())
}

fn build_github_url(path: &str) -> Result<Url, ParseError> {
    const GITHUB: &'static str = "https://github.com";

    let base = Url::parse(GITHUB).expect("hardcoded URL is known to be valid");
    let joined = base.join(path)?;

    Ok(joined)
}
```

Extract the URL origin (scheme / host / port)

url v2.4.0 Net

The `Url` struct exposes various methods to extract information about the URL it represents.

```
use url::{Url, Host, ParseError};

fn main() -> Result<(), ParseError> {
    let s = "ftp://rust-lang.org/examples";

    let url = Url::parse(s)?;

    assert_eq!(url.scheme(), "ftp");
    assert_eq!(url.host(), Some(Host::Domain("rust-lang.org")));
    assert_eq!(url.port_or_known_default(), Some(21));
    println!("The origin is as expected!");

    Ok(())
}
```

`origin` produces the same result.

```
use url::{Url, Origin, Host};

fn main() -> Result<()> {
    let s = "ftp://rust-lang.org/examples";

    let url = Url::parse(s)?;

    let expected_scheme = "ftp".to_owned();
    let expected_host = Host::Domain("rust-lang.org".to_owned());
    let expected_port = 21;
    let expected = Origin::Tuple(expected_scheme, expected_host, expected_port);

    let origin = url.origin();
    assert_eq!(origin, expected);
    println!("The origin is as expected!");

    Ok(())
}
```

Remove fragment identifiers and query pairs from a URL

url v2.4.0 Net

Parses `Url` and slices it with `url::Position` to strip unneeded URL parts.

```
use url::{Url, Position, ParseError};

fn main() -> Result<(), ParseError> {
    let parsed = Url::parse("https://github.com/rust-lang/rust/issues?labels=E-
easy&state=open")?;
    let cleaned: &str = &parsed[..Position::AfterPath];
    println!("cleaned: {}", cleaned);
    Ok(())
}
```

Media Types

Get MIME type from string

mime v1.2.2 Encoding

The following example shows how to parse a `MIME` type from a string using the mime crate. `FromStrError` produces a default `MIME` type in an `unwrap_or` clause.

```
use mime::{Mime, APPLICATION_OCTET_STREAM};

fn main() {
    let invalid_mime_type = "i n v a l i d";
    let default_mime = invalid_mime_type
        .parse::<Mime>()
        .unwrap_or(APPLICATION_OCTET_STREAM);

    println!(
        "MIME for {:?} used default value {:?}",
        invalid_mime_type, default_mime
    );

    let valid_mime_type = "TEXT/PLAIN";
    let parsed_mime = valid_mime_type
        .parse::<Mime>()
        .unwrap_or(APPLICATION_OCTET_STREAM);

    println!(
        "MIME for {:?} was parsed as {:?}",
        valid_mime_type, parsed_mime
    );
}
```

Get MIME type from filename

mime v1.2.2 Encoding

The following example shows how to return the correct MIME type from a given filename using the mime crate. The program will check for file extensions and match against a known list. The return value is `mime:Mime`.

```
use mime::Mime;

fn find_mimetype (filename : &String) -> Mime{

    let parts : Vec<&str> = filename.split('.').collect();

    let res = match parts.last() {
            Some(v) =>
                match *v {
                    "png" => mime::IMAGE_PNG,
                    "jpg" => mime::IMAGE_JPEG,
                    "json" => mime::APPLICATION_JSON,
                    &_ => mime::TEXT_PLAIN,
                },
            None => mime::TEXT_PLAIN,
        };
    return res;
}

fn main() {
    let filenames = vec!("foobar.jpg", "foo.bar", "foobar.png");
    for file in filenames {
        let mime = find_mimetype(&file.to_owned());
        println!("MIME for {}: {}", file, mime);
     }

}
```

Parse the MIME type of a HTTP response

reqwest v0.11.18 mime v1.2.2 Net Encoding

When receiving a HTTP reponse from `reqwest` the MIME type or media type may be found in the Content-Type header. `reqwest::header::HeaderMap::get` retrieves the header as a `reqwest::header::HeaderValue`, which can be converted to a string. The `mime` crate can then parse that, yielding a `mime::Mime` value.

The `mime` crate also defines some commonly used MIME types.

Note that the `reqwest::header` module is exported from the `http` crate.

```
use error_chain::error_chain;
use mime::Mime;
use std::str::FromStr;
use reqwest::header::CONTENT_TYPE;

 error_chain! {
    foreign_links {
        Reqwest(reqwest::Error);
        Header(reqwest::header::ToStrError);
        Mime(mime::FromStrError);
    }
 }

#[tokio::main]
async fn main() -> Result<()> {
    let response = reqwest::get("https://www.rust-lang.org/logos/rust-logo-
32x32.png").await?;
    let headers = response.headers();

    match headers.get(CONTENT_TYPE) {
        None => {
            println!("The response does not contain a Content-Type header.");
        }
        Some(content_type) => {
            let content_type = Mime::from_str(content_type.to_str()?)?;
            let media_type = match (content_type.type_(), content_type.subtype()) {
                (mime::TEXT, mime::HTML) => "a HTML document",
                (mime::TEXT, _) => "a text document",
                (mime::IMAGE, mime::PNG) => "a PNG image",
                (mime::IMAGE, _) => "an image",
                _ => "neither text nor image",
            };

            println!("The reponse contains {}.", media_type);
        }
    };

    Ok(())
}
```

Clients

Recipe	Crates	Categories
Make a HTTP GET request	reqwest v0.11.18	Net

Recipe	Crates	Categories
Query the GitHub API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Check if an API resource exists	reqwest v0.11.18	Net
Create and delete Gist with GitHub API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Consume a paginated RESTful API	reqwest v0.11.18 serde v1.0.171	Net Encoding
Download a file to a temporary directory	reqwest v0.11.18 tempdir v0.3.7	Net Filesystem
Make a partial download with HTTP range headers	reqwest v0.11.18	Net
POST a file to paste-rs	reqwest v0.11.18	Net

Making Requests

Make a HTTP GET request

reqwest v0.11.18 Net

Parses the supplied URL and makes a synchronous HTTP GET request with `reqwest::blocking::get`. Prints obtained `reqwest::blocking::Response` status and headers. Reads HTTP response body into an allocated `String` using `read_to_string`.

```
use error_chain::error_chain;
use std::io::Read;

error_chain! {
    foreign_links {
        Io(std::io::Error);
        HttpRequest(reqwest::Error);
    }
}

fn main() -> Result<()> {
    let mut res = reqwest::blocking::get("http://httpbin.org/get")?;
    let mut body = String::new();
    res.read_to_string(&mut body)?;

    println!("Status: {}", res.status());
    println!("Headers:\n{:#?}", res.headers());
    println!("Body:\n{}", body);

    Ok(())
}
```

Async

A similar approach can be used by including the `tokio` executor to make the main function asynchronous, retrieving the same information.

In this example, `tokio::main` handles all the heavy executor setup and allows sequential code implemented without blocking until `.await`.

Uses the asynchronous versions of reqwest, both `reqwest::get` and `reqwest::Response`.

```
use error_chain::error_chain;

error_chain! {
    foreign_links {
        Io(std::io::Error);
        HttpRequest(reqwest::Error);
    }
}

#[tokio::main]
async fn main() -> Result<()> {
    let res = reqwest::get("http://httpbin.org/get").await?;
    println!("Status: {}", res.status());
    println!("Headers:\n{:#?}", res.headers());

    let body = res.text().await?;
    println!("Body:\n{}", body);
    Ok(())
}
```

Calling a Web API

Query the GitHub API

reqwest v0.11.18 serde v1.0.171 Net Encoding

Queries GitHub stargazers API v3 with `reqwest::get` to get list of all users who have marked a GitHub project with a star. `reqwest::Response` is deserialized with `Response::json` into `User` objects implementing `serde::Deserialize`.

[tokio::main] is used to set up the async executor and the process waits for [`reqwet::get`] to complete before processing the response into User instances.

```
use serde::Deserialize;
use reqwest::Error;

#[derive(Deserialize, Debug)]
struct User {
    login: String,
    id: u32,
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    let request_url = format!("https://api.github.com/repos/{owner}/{repo}/stargazers",
                              owner = "rust-lang-nursery",
                              repo = "rust-cookbook");
    println!("{}", request_url);
    let response = reqwest::get(&request_url).await?;

    let users: Vec<User> = response.json().await?;
    println!("{:?}", users);
    Ok(())
}
```

Check if an API resource exists

Query the GitHub Users Endpoint using a HEAD request (`Client::head`) and then inspect the response code to determine success. This is a quick way to query a rest resource without needing to receive a body. `reqwest::Client` configured with `ClientBuilder::timeout` ensures a request will not last longer than a timeout.

Due to both `ClientBuilder::build` and [`ReqwestBuilder::send`] returning `reqwest::Error` types, the shortcut `reqwest::Result` is used for the main function return type.

```
use reqwest::Result;
use std::time::Duration;
use reqwest::ClientBuilder;

#[tokio::main]
async fn main() -> Result<()> {
    let user = "ferris-the-crab";
    let request_url = format!("https://api.github.com/users/{}", user);
    println!("{}", request_url);

    let timeout = Duration::new(5, 0);
    let client = ClientBuilder::new().timeout(timeout).build()?;
    let response = client.head(&request_url).send().await?;

    if response.status().is_success() {
        println!("{} is a user!", user);
    } else {
        println!("{} is not a user!", user);
    }

    Ok(())
}
```

Create and delete Gist with GitHub API

reqwest v0.11.18 serde v1.0.171 Net Encoding

Creates a gist with POST request to GitHub gists API v3 using `Client::post` and removes it with DELETE request using `Client::delete`.

The `reqwest::Client` is responsible for details of both requests including URL, body and authentication. The POST body from `serde_json::json!` macro provides arbitrary JSON body. Call to `RequestBuilder::json` sets the request body. `RequestBuilder::basic_auth` handles authentication. The call to `RequestBuilder::send` synchronously executes the requests.

```
use error_chain::error_chain;
use serde::Deserialize;
use serde_json::json;
use std::env;
use reqwest::Client;

error_chain! {
    foreign_links {
        EnvVar(env::VarError);
        HttpRequest(reqwest::Error);
    }
}

#[derive(Deserialize, Debug)]
struct Gist {
    id: String,
    html_url: String,
}

#[tokio::main]
async fn main() ->  Result<()> {
    let gh_user = env::var("GH_USER")?;
    let gh_pass = env::var("GH_PASS")?;

    let gist_body = json!({
        "description": "the description for this gist",
        "public": true,
        "files": {
             "main.rs": {
             "content": r#"fn main() { println!("hello world!");}"#
            }
        }});

    let request_url = "https://api.github.com/gists";
    let response = Client::new()
        .post(request_url)
        .basic_auth(gh_user.clone(), Some(gh_pass.clone()))
        .json(&gist_body)
        .send().await?;

    let gist: Gist = response.json().await?;
    println!("Created {:?}", gist);

    let request_url = format!("{}/{}",request_url, gist.id);
    let response = Client::new()
        .delete(&request_url)
        .basic_auth(gh_user, Some(gh_pass))
        .send().await?;

    println!("Gist {} deleted! Status code: {}",gist.id, response.status());
    Ok(())
}
```

The example uses HTTP Basic Auth in order to authorize access to GitHub API. Typical use case would employ one of the much more complex OAuth authorization flows.

Consume a paginated RESTful API

reqwest v0.11.18 serde v1.0.171 Net Encoding

Wraps a paginated web API in a convenient Rust iterator. The iterator lazily fetches the next page of results from the remote server as it arrives at the end of each page.

```
use reqwest::Result;
use serde::Deserialize;

#[derive(Deserialize)]
struct ApiResponse {
    dependencies: Vec<Dependency>,
    meta: Meta,
}

#[derive(Deserialize)]
struct Dependency {
    crate_id: String,
}

#[derive(Deserialize)]
struct Meta {
    total: u32,
}

struct ReverseDependencies {
    crate_id: String,
    dependencies: <Vec<Dependency> as IntoIterator>::IntoIter,
    client: reqwest::blocking::Client,
    page: u32,
    per_page: u32,
    total: u32,
}

impl ReverseDependencies {
    fn of(crate_id: &str) -> Result<Self> {
        Ok(ReverseDependencies {
               crate_id: crate_id.to_owned(),
               dependencies: vec![].into_iter(),
               client: reqwest::blocking::Client::new(),
               page: 0,
               per_page: 100,
               total: 0,
           })
    }

    fn try_next(&mut self) -> Result<Option<Dependency>> {
        if let Some(dep) = self.dependencies.next() {
            return Ok(Some(dep));
        }

        if self.page > 0 && self.page * self.per_page >= self.total {
            return Ok(None);
        }

        self.page += 1;
        let url = format!("https://crates.io/api/v1/crates/{}/reverse_dependencies?
page={}&per_page={}",
                          self.crate_id,
                          self.page,
                          self.per_page);

        let response = self.client.get(&url).send()?.json::<ApiResponse>()?;
        self.dependencies = response.dependencies.into_iter();
        self.total = response.meta.total;
        Ok(self.dependencies.next())
    }
}

impl Iterator for ReverseDependencies {
    type Item = Result<Dependency>;

    fn next(&mut self) -> Option<Self::Item> {
```

```
        match self.try_next() {
            Ok(Some(dep)) => Some(Ok(dep)),
            Ok(None) => None,
            Err(err) => Some(Err(err)),
        }
    }
}

fn main() -> Result<()> {
    for dep in ReverseDependencies::of("serde")? {
        println!("reverse dependency: {}", dep?.crate_id);
    }
    Ok(())
}
```

Downloads

Download a file to a temporary directory

Creates a temporary directory with `tempfile::Builder` and downloads a file over HTTP using `reqwest::get` asynchronously.

Creates a target `File` with name obtained from `Response::url` within `tempdir()` and copies downloaded data into it with `io::copy`. The temporary directory is automatically removed on program exit.

```
use error_chain::error_chain;
use std::io::copy;
use std::fs::File;
use tempfile::Builder;

error_chain! {
     foreign_links {
         Io(std::io::Error);
         HttpRequest(reqwest::Error);
     }
}

#[tokio::main]
async fn main() -> Result<()> {
    let tmp_dir = Builder::new().prefix("example").tempdir()?;
    let target = "https://www.rust-lang.org/logos/rust-logo-512x512.png";
    let response = reqwest::get(target).await?;

    let mut dest = {
        let fname = response
            .url()
            .path_segments()
            .and_then(|segments| segments.last())
            .and_then(|name| if name.is_empty() { None } else { Some(name) })
            .unwrap_or("tmp.bin");

        println!("file to download: '{}'", fname);
        let fname = tmp_dir.path().join(fname);
        println!("will be located under: '{:?}'", fname);
        File::create(fname)?
    };
    let content =  response.text().await?;
    copy(&mut content.as_bytes(), &mut dest)?;
    Ok(())
}
```

POST a file to paste-rs

`reqwest::Client` establishes a connection to https://paste.rs following the `reqwest::RequestBuilder` pattern. Calling `Client::post` with a URL establishes the destination, `RequestBuilder::body` sets the content to send by reading the file, and `RequestBuilder::send` blocks until the file uploads and the response returns. `read_to_string` returns the response and displays in the console.

```
use error_chain::error_chain;
use std::fs::File;
use std::io::Read;

 error_chain! {
     foreign_links {
         HttpRequest(reqwest::Error);
         IoError(::std::io::Error);
     }
 }
 #[tokio::main]

async fn main() -> Result<()> {
    let paste_api = "https://paste.rs";
    let mut file = File::open("message")?;

    let mut contents = String::new();
    file.read_to_string(&mut contents)?;

    let client = reqwest::Client::new();
    let res = client.post(paste_api)
        .body(contents)
        .send()
        .await?;
    let response_text = res.text().await?;
    println!("Your paste is located at: {}",response_text );
    Ok(())
}
```

Make a partial download with HTTP range headers

Uses `reqwest::blocking::Client::head` to get the Content-Length of the response.

The code then uses `reqwest::blocking::Client::get` to download the content in chunks of 10240 bytes, while printing progress messages. This exmple uses the synchronous reqwest module. The Range header specifies the chunk size and position.

The Range header is defined in RFC7233.

```
use error_chain::error_chain;
use reqwest::header::{HeaderValue, CONTENT_LENGTH, RANGE};
use reqwest::StatusCode;
use std::fs::File;
use std::str::FromStr;

error_chain! {
    foreign_links {
        Io(std::io::Error);
        Reqwest(reqwest::Error);
        Header(reqwest::header::ToStrError);
    }
}

struct PartialRangeIter {
  start: u64,
  end: u64,
  buffer_size: u32,
}

impl PartialRangeIter {
  pub fn new(start: u64, end: u64, buffer_size: u32) -> Result<Self> {
    if buffer_size == 0 {
      Err("invalid buffer_size, give a value greater than zero.")?;
    }
    Ok(PartialRangeIter {
      start,
      end,
      buffer_size,
    })
  }
}

impl Iterator for PartialRangeIter {
  type Item = HeaderValue;
  fn next(&mut self) -> Option<Self::Item> {
    if self.start > self.end {
      None
    } else {
      let prev_start = self.start;
      self.start += std::cmp::min(self.buffer_size as u64, self.end - self.start + 1);
      Some(HeaderValue::from_str(&format!("bytes={}-{}", prev_start, self.start -
1)).expect("string provided by format!"))
    }
  }
}

fn main() -> Result<()> {
  let url = "https://httpbin.org/range/102400?duration=2";
  const CHUNK_SIZE: u32 = 10240;

  let client = reqwest::blocking::Client::new();
  let response = client.head(url).send()?;
  let length = response
    .headers()
    .get(CONTENT_LENGTH)
    .ok_or("response doesn't include the content length")?;
  let length = u64::from_str(length.to_str()?).map_err(|_| "invalid Content-Length
header")?;

  let mut output_file = File::create("download.bin")?;

  println!("starting download...");
  for range in PartialRangeIter::new(0, length - 1, CHUNK_SIZE)? {
    println!("range {:?}", range);
    let mut response = client.get(url).header(RANGE, range).send()?;
```

```
    let status = response.status();
    if !(status == StatusCode::OK || status == StatusCode::PARTIAL_CONTENT) {
      error_chain::bail!("Unexpected server response: {}", status)
    }
    std::io::copy(&mut response, &mut output_file)?;
  }

  let content = response.text()?;
  std::io::copy(&mut content.as_bytes(), &mut output_file)?;

  println!("Finished with success!");
  Ok(())
}
```

Authentication

Basic Authentication

Uses `reqwest::RequestBuilder::basic_auth` to perform a basic HTTP authentication.

```
use reqwest::blocking::Client;
use reqwest::Error;

fn main() -> Result<(), Error> {
    let client = Client::new();

    let user_name = "testuser".to_string();
    let password: Option<String> = None;

    let response = client
        .get("https://httpbin.org/")
        .basic_auth(user_name, password)
        .send();

    println!("{:?}", response);

    Ok(())
}
```