WIKIPEDIA

Rust (programming language)

Rust is a multi-paradigm, general-purpose programming language designed for performance and safety, especially safe concurrency.[12][13] Rust is syntactically similar to C++,[14] but can guarantee memory safety by using a *borrow checker* to validate references.[15] Rust achieves memory safety without garbage collection, and reference counting is optional.[16][17] Rust has been called a systems programming language and in addition to high-level features such as functional programming it also offers mechanisms for low-level memory management.

Rust was originally designed by Graydon Hoare at Mozilla Research,[18] with contributions from Dave Herman, Brendan Eich, and others.[19][20] The designers refined the language while writing the Servo experimental browser engine,[21] and the Rust compiler. It has gained increasing use in industry, and Microsoft has been experimenting with the language for secure and safety-critical software components.[22][23]

Rust has been voted the "most loved programming language" in the Stack Overflow Developer Survey every year since 2016, though only used by 7% of the respondents in 2021.[24]

Contents

Rust

The official Rust logo

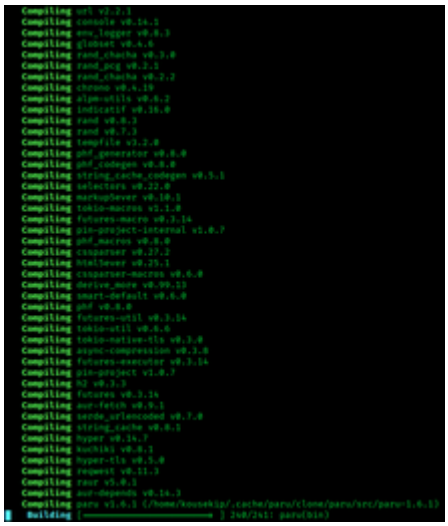
Paradigms	Multi-paradigm: concurrent, functional, generic, imperative, structured
Designed by	Graydon Hoare
Developer	The Rust Foundation
First appeared	July 7, 2010
Stable release	1.57.0[1] / December 2, 2021
Typing discipline	Affine, inferred, nominal, static, strong
Implementation language	Rust
Platform	AMD64 , i686, arm, AArch64, armv7, mips, mips64, mipsel, mips64el, powerpc, powerpc64, powerpc64le, risc-v, s390x[note 1]

OS	Windows, Linux, macOS, FreeBSD, NetBSD
License	MIT or Apache 2.0[2]
Filename extensions	.rs, .rlib
Website	www.rust-lang.org (http://www.rust-lang.org)
Influenced by	
Alef[3], C#[3], C++[3], Cyclone[3][4], Erlang[3], Haskell[3], Limbo[3], Newsqueak[3], OCaml[3], Ruby[3], Scheme[3], Standard ML[3], Swift[3][5]	
Influenced	
Crystal, Elm[6], Idris[7], Spark[8], Swift[9], Project Verona[10], Zig, PHP[11]	

History

The language grew out of a personal project begun in 2006 by Mozilla employee Graydon Hoare.[13] Hoare has stated that the project was possibly named after rust fungi and that the name is also a substring of "robust".[25] Mozilla began sponsoring the project in 2009[13] and announced it in 2010.[26][27] The same year, work shifted from the initial compiler (written in OCaml) to an LLVM-based self-hosting compiler written in Rust.[28] Named `rustc`, it successfully compiled itself in 2011.[29]

The first numbered pre-alpha release of the Rust compiler occurred in January 2012.[30] Rust 1.0, the first stable release, was released on May 15, 2015.[31][32] Following 1.0, stable point releases are delivered every six weeks, while features are developed in nightly Rust with daily releases, then tested with beta releases that last six weeks.[33][34] Every 2 to 3 years, a new Rust "Edition" is produced. This is to provide an easy reference point for changes due to the frequent nature of Rust's *Train release schedule,* as well as to provide a window to make breaking changes. Editions are largely compatible.[35]



An example of compiling a Rust program

Along with conventional static typing, before version 0.4, Rust also supported typestates. The typestate system modeled assertions before and after program statements, through use of a special `check` statement. Discrepancies could be discovered at compile time, rather than at runtime, as might be the case with assertions in C or C++ code. The typestate concept was not unique to Rust, as it was first introduced in the language NIL.[36] Typestates were removed because in practice they were little used,[37] though the same functionality can be achieved by leveraging Rust's move semantics.[38]

The object system style changed considerably within versions 0.2, 0.3, and 0.4 of Rust. Version 0.2 introduced classes for the first time, and version 0.3 added several features, including destructors and polymorphism through the use of interfaces. In Rust 0.4, traits were added as a means to provide inheritance; interfaces were unified with traits and removed as a separate feature. Classes were also removed and replaced by a combination of implementations and structured types.[39]

Starting in Rust 0.9 and ending in Rust 0.11, Rust had two built-in pointer types: ~ and @, simplifying the core memory model. It reimplemented those pointer types in the standard library as `Box` and (the now removed) `Gc`.

In January 2014, before the first stable release, Rust 1.0, the editor-in-chief of *Dr. Dobb's*, Andrew Binstock, commented on Rust's chances of becoming a competitor to C++ and to the other up-and-coming languages D, Go, and Nim (then Nimrod). According to Binstock, while Rust was "widely viewed as a remarkably elegant language", adoption slowed because it repeatedly changed between versions.[40]

Rust has a foreign function interface (FFI) that can be called from, e.g., C language, and can call C. While calling C++ has historically been challenging (from any language), Rust has a library, CXX, to allow calling to or from C++, and "CXX has zero or negligible overhead."[41]

In August 2020, Mozilla laid off 250 of its 1,000 employees worldwide as part of a corporate restructuring caused by the long-term impact of the COVID-19 pandemic.[42][43] Among those laid off were most of the Rust team,[44] while the Servo team was completely disbanded.[45] The event raised concerns about the future of Rust.[46]

In the following week, the Rust Core Team acknowledged the severe impact of the layoffs and announced that plans for a Rust foundation were underway. The first goal of the foundation would be taking ownership of all trademarks and domain names, and also take financial responsibility for their costs.[47]

On February 8, 2021 the formation of the Rust Foundation was officially announced by its five founding companies (AWS, Huawei, Google, Microsoft, and Mozilla).[48][49]

On April 6, 2021, Google announced support for Rust within Android Open Source Project as an alternative to C/C++.[50]

Syntax

Here is a "Hello, World!" program written in Rust. The `println!` macro prints the message to standard output.

```
fn main() {
    println!("Hello, World!");
}
```

The syntax of Rust is similar to C and C++, with blocks of code delimited by curly brackets, and control flow keywords such as `if`, `else`, `while`, and `for`, although the specific syntax for defining functions is more similar to Pascal. Despite the resemblance to C and C++, the syntax of Rust is closer to that of the ML family of languages and the Haskell language. Nearly every part of a function body is an expression,[51] even control flow operators. For example, the ordinary `if` expression also takes the place of C's ternary conditional, an idiom used by ALGOL 60. As in Lisp, a function need not end with a `return` expression: in this case if the semicolon is omitted, the last expression in the function creates the return value, as seen in the following recursive implementation of the factorial function:

```
fn factorial(i: u64) -> u64 {
    match i {
        0 => 1,
        n => n * factorial(n-1)
    }
}
```

The following iterative implementation uses the `..=` operator to create an inclusive range:

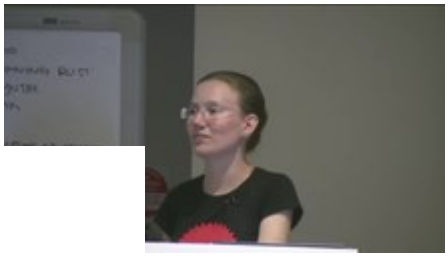
```
fn factorial(i: u64) -> u64 {
    (2..=i).product()
}
```

More advanced features in Rust include the use of generic functions to achieve type polymorphism. The following is a Rust program to calculate the sum of two things, for which addition is implemented, using a generic function:

```
use std::ops::Add;

fn Sum<T:Add<Output = T> + Copy> (num1: T, num2: T) -> T {
    num1 + num2
}

fn main() {
    let result1 = Sum(10,20);
    println!("Sum is: {:?}", result1);

    let result2 = Sum(10.23,20.45);
    println!("Sum is: {:?}", result2);
}
```

Features

Rust is intended to be a language for highly concurrent and highly safe systems,[52] and *programming in the large*, that is, creating and maintaining boundaries that preserve large-system integrity.[53] This has led to a feature set with an emphasis on safety, control of memory layout, and concurrency.



A presentation on Rust by Emily Dunham from Mozilla's Rust team (linux.conf.au conference, Hobart, 2017)

Memory safety

Rust is designed to be memory safe. It does not permit null pointers, dangling pointers, or data races.[54][55][56] Data values can be initialized only through a fixed set of forms, all of which require their inputs to be already initialized.[57] To replicate pointers being either valid or `NULL`, such as in linked list or binary tree data structures, the Rust core library provides an option type, which can be used to test whether a pointer has `Some` value or `None`.[55] Rust has added syntax to manage lifetimes, which are checked at compile time by the *borrow checker*. Unsafe code can subvert some of these restrictions using the `unsafe` keyword.[15]

Memory management

Rust does not use automated garbage collection. Memory and other resources are managed through the resource acquisition is initialization convention,[58] with optional reference counting. Rust provides deterministic management of resources, with very low overhead.[59] Rust favors stack allocation of values and does not perform implicit boxing.

There is the concept of references (using the & symbol), which does not involve run-time reference counting. The safety of such pointers is verified at compile time, preventing dangling pointers and other forms of undefined behavior. Rust's type system separates shared, immutable pointers of the

form &T from unique, mutable pointers of the form `&mut T`. A mutable pointer can be coerced to an immutable pointer, but not vice versa.

Ownership

Rust has an ownership system where all values have a unique owner, and the scope of the value is the same as the scope of the owner.[60][61] Values can be passed by immutable reference, using `&T`, by mutable reference, using `&mut T`, or by value, using `T`. At all times, there can either be multiple immutable references or one mutable reference (an implicit readers–writer lock). The Rust compiler enforces these rules at compile time and also checks that all references are valid.

Types and polymorphism

Rust's type system supports a mechanism called traits, inspired by type classes in the Haskell language. Traits annotate types, and are used to define *shared behavior* between different types: for example, floats and integers might both implement an "Add" trait because they can both be added; and any type that can be printed out as a string implements the "Display" or "Debug" traits. This facility is known as ad hoc polymorphism.

Rust uses type inference for variables declared with the keyword `let`. Such variables do not require a value to be initially assigned to determine their type. A compile time error results if any branch of code leaves the variable without an assignment.[62] Variables assigned multiple times must be marked with the keyword `mut` (short for mutable).

A function can be given generic parameters, which allows the same function to be applied to different types. Generic functions can constrain the generic type to implement a particular trait or traits; for example, an "add one" function might require the type to implement "Add". This means that a generic function can be type-checked as soon as it is defined.

The implementation of Rust generics is similar to the typical implementation of C++ templates: a separate copy of the code is generated for each instantiation. This is called monomorphization and contrasts with the type erasure scheme typically used in Java and Haskell. Rust's type erasure is also available by using the keyword `dyn`. The benefit of monomorphization is optimized code for each specific use case; the drawback is increased compile time and size of the resulting binaries.

In Rust, user-defined types are created with the `struct` keyword. These types usually contains fields of data like objects or classes in other languages. The `impl` keyword can define methods for the struct (data and function are defined separately in a struct) or implement a trait for the structure. A trait is a contract that a structure has certain required methods implemented. Traits are used to restrict generic parameters and because traits can provide a struct with more methods than the user defined. For example, the trait `Iterator` requires that the `next` method be defined for the type. Once the `next` method is defined the trait provides common functional helper methods over the iterator like map or filter.

The object system within Rust is based around implementations, traits and structured types. Implementations fulfill a role similar to that of classes within other languages and are defined with the keyword `impl`. Traits provide inheritance and polymorphism; they allow methods to be defined and mixed in to implementations. Structured types are used to define fields. Implementations and traits cannot define fields themselves, and only traits can provide inheritance. Among other benefits, this prevents the diamond problem of multiple inheritance, as in C++. In other words, Rust supports interface inheritance but replaces implementation inheritance with composition; see composition over inheritance.

Components

Rust features a large number of components that extend the Rust feature set and make Rust development easier. Component installation is typically managed by *rustup,* a Rust toolchain installer developed by the Rust project.[63]

Cargo

Cargo is Rust's build system and package manager. Cargo handles downloading dependencies, and building dependencies. Cargo also acts as a wrapper for *clippy* and other Rust components. It requires projects to follow a certain directory structure.[64]

The dependencies for a Rust package are specified in a *Cargo.toml* file along with version requirements, telling Cargo which versions of the dependency are compatible with the package. By default, Cargo sources its dependencies from the user-contributed registry *crates.io (https://crates.io)* but Git repositories and packages in the local filesystem can be specified as dependencies, too.[65]

Rustfmt

Rustfmt is a code formatter for Rust. It takes Rust source code as input and changes the whitespace and indentation to produce formatted code in accordance to the Rust style guide or rules specified in a rustfmt.toml file. Rustfmt can be invoked as a standalone program or on a Rust project through Cargo. [66][67]

Clippy

Clippy is Rust's built-in linting tool to improve the correctness, performance, and readability of Rust code. It was created in 2014[68] and named after the eponymous Microsoft Office feature.[69] As of 2021, Clippy has more than 450 rules,[70] which can be browsed online and filtered by category.[71] Some rules are disabled by default.

RLS

RLS is a language server that provides IDEs and text editors with more information about a Rust project. It provides linting checks via *Clippy*, formatting via *Rustfmt,* automatic code completion via *Racer*, among other functions.[72] Development of *Racer* was slowed down in favor of *rust-analyzer*.[73]

Language extensions

It is possible to extend the Rust language using the procedural macro mechanism.[74]

Procedural macros use Rust functions that run at compile time to modify the compiler's token stream. This complements the declarative macro mechanism (also known as *macros by example*), which uses pattern matching to achieve similar goals.

Procedural macros come in three flavors:

- Function-like macros `custom!(...)`

- Derive macros `#[derive(CustomDerive)]`
- Attribute macros `#[custom_attribute]`

The `println!` macro is an example of a function-like macro and `serde_derive`[75] is a commonly used library for generating code for reading and writing data in many formats such as JSON. Attribute macros are commonly used for language bindings such as the `extendr` library for Rust bindings to R.[76]

The following code shows the use of the `Serialize`, `Deserialize` and `Debug` derive procedural macros to implement JSON reading and writing as well as the ability to format a structure for debugging.

```
use serde_json::{Serialize, Deserialize};

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

fn main() {
    let point = Point { x: 1, y: 2 };

    let serialized = serde_json::to_string(&point).unwrap();
    println!("serialized = {}", serialized);

    let deserialized: Point = serde_json::from_str(&serialized).unwrap();
    println!("deserialized = {:?}", deserialized);
}
```

Performance

Rust aims "to be as efficient and portable as idiomatic C++, without sacrificing safety".[77] Since Rust utilizes LLVM, any performance improvements in LLVM also carry over to Rust.[78]

Adoption

Rust was the third-most-loved programming language in the 2015 Stack Overflow annual survey[80] and took first place for 2016–2021.[81][82]

Some Rust users refer to themselves as Rustaceans (a pun on crustacean) and use Ferris as their unofficial mascot.[79]

Web browsers

Firefox has two projects written in Rust: the Servo parallel browser engine[83] developed by Mozilla in collaboration with Samsung;[84] and Quantum, which is composed of several sub-projects for improving Mozilla's Gecko browser engine.[85]

Experimental operating systems

- Redox, a "full-blown Unix-like operating system" including a microkernel[86]
- Theseus, OS with "intralingual design" and a fundamental architecture which embodies Rust concepts[87]

Game engines

- Bevy (https://bevyengine.org/), a data-driven ECS game engine built in Rust.
- Amethyst (https://amethyst.rs/), a data-driven and data-oriented game engine.

Other

- Deno, a secure runtime for JavaScript and TypeScript, is built with V8, Rust, and Tokio[88]
- Discord uses Rust for portions of its backend, as well as client-side video encoding,[89] to augment the core infrastructure written in Elixir.[90]
- exa, a "modern replacement for ls"
- The Google Fuchsia capability-based operating system has some tools written in Rust[91]
- Microsoft Azure IoT Edge, a platform used to run Azure services and artificial intelligence on IoT devices, has components implemented in Rust[92]
- OpenDNS uses Rust in two of its components[93][94][95]
- Polkadot (cryptocurrency), an interconnected internet of blockchains, is written in Rust
- Ruffle, an open-source SWF emulator written in Rust[96]
- Stratis: a file system manager for Fedora[97] and RHEL 8[98]
- TerminusDB, an open source graph database designed for collaboratively building and curating knowledge graphs[99]
- Such applications as Figma, and Dropbox are written in Rust as well as some components for Amazon, Facebook, and Discord[100]

Governance

The **Rust Foundation** is a non-profit membership organization incorporated in Delaware, United States, with the primary purposes of supporting the maintenance and development of the language, cultivating the Rust project team members and user communities, managing the technical infrastructure underlying the development of Rust, and managing and stewarding the Rust trademark.

It was established on February 8, 2021, with five founding corporate members (Amazon Web Services, Huawei, Google, Microsoft, and Mozilla).[101] The foundation's board is chaired by Shane Miller.[102] Starting in late 2021, its Executive Director and CEO is Rebecca Rumbul .[103] Prior to this, Ashley Williams was interim executive director.[104]

Rust Foundation

Formation	February 8, 2021
Founders	Amazon Web Services Google Huawei Microsoft Mozilla Foundation
Type	Nonprofit organization
Location	United States
Chairperson	Shane Miller
Executive Director	Rebecca Rumbul
Website	foundation.rust-lang.org (https://foundation.rust-lang.org)

Development

Rust conferences include:

- RustConf: an annual conference in Portland, Oregon. Held annually since 2016 (except in 2020 and 2021 because of the COVID-19 pandemic).[105]
- Rust Belt Rust: a #rustlang conference in the Rust Belt[106]

- RustFest: Europe's @rustlang conference[107]
- RustCon Asia
- Rust LATAM
- Oxidize Global[108]

See also

- Comparison of programming languages

Notes

1. The list is incomplete; degree of stdlib support varies

References

1. "Announcing Rust 1.57.0" (https://blog.rust-lang.org/2021/12/02/Rust-1.57.0.html).
2. "Rust Legal Policies" (https://www.rust-lang.org/en-US/legal.html). *Rust-lang.org*. Archived (https://web.archive.org/web/20180404073350/https://www.rust-lang.org/en-US/legal.html) from the original on April 4, 2018. Retrieved April 3, 2018.
3. "The Rust Reference: Appendix: Influences" (https://doc.rust-lang.org/reference/influences.html). Archived (https://web.archive.org/web/20190126051127/https://doc.rust-lang.org/reference/influences.html) from the original on January 26, 2019. Retrieved November 11, 2018.
4. "Note Research: Type System" (https://github.com/rust-lang/rust-wiki-backup/blob/master/Note-research.md#type-system). *GitHub*. February 1, 2015. Archived (https://web.archive.org/web/20190217182048/https://github.com/rust-lang/rust-wiki-backup/blob/master/Note-research.md#type-system) from the original on February 17, 2019. Retrieved March 25, 2015.
5. "RFC for 'if let' expression" (https://github.com/rust-lang/rfcs/pull/160). *GitHub*. Archived (https://web.archive.org/web/20160304192327/https://github.com/rust-lang/rfcs/pull/160) from the original on March 4, 2016. Retrieved December 4, 2014.
6. "Command Optimizations?" (https://groups.google.com/forum/?fromgroups#!searchin/elm-discuss/rust/elm-discuss/lMX_9miTD2E/QBwdvL4JD9wJ). June 26, 2014. Archived (https://web.archive.org/web/20190710034511/https://groups.google.com/forum/?fromgroups#!searchin/elm-discuss/rust/elm-discuss/lMX_9miTD2E/QBwdvL4JD9wJ) from the original on July 10, 2019. Retrieved December 10, 2014.
7. "Idris – Uniqueness Types" (http://docs.idris-lang.org/en/latest/reference/uniqueness-types.html). Archived (https://web.archive.org/web/20181121072557/http://docs.idris-lang.org/en/latest/reference/uniqueness-types.html) from the original on November 21, 2018. Retrieved November 20, 2018.
8. Jaloyan, Georges-Axel (October 19, 2017). "Safe Pointers in SPARK 2014". arXiv:1710.07047 (https://arxiv.org/abs/1710.07047). Bibcode:2017arXiv171007047J (https://ui.adsabs.harvard.edu/abs/2017arXiv171007047J).
9. Lattner, Chris. "Chris Lattner's Homepage" (http://nondot.org/sabre/). *Nondot.org*. Archived (https://web.archive.org/web/20181225175312/http://nondot.org/sabre/) from the original on December 25, 2018. Retrieved May 14, 2019.
10. "Microsoft opens up Rust-inspired Project Verona programming language on GitHub" (https://www.zdnet.com/article/microsoft-opens-up-rust-inspired-project-verona-programming-language-on-github/). *ZDNet*. Archived (https://web.archive.org/web/20200117143852/https://www.zdnet.com/article/microsoft-opens-up-rust-inspired-project-verona-programming-language-on-github/) from the original on January 17, 2020. Retrieved January 17, 2020.

11. "PHP RFC: Shorter Attribute Syntax" (https://wiki.php.net/rfc/shorter_attribute_syntax). June 3, 2020. Archived (https://web.archive.org/web/20210307201934/https://wiki.php.net/rfc/shorter_attribute_syntax) from the original on March 7, 2021. Retrieved March 17, 2021.
12. Hoare, Graydon (December 28, 2016). "Rust is mostly safety" (https://graydon2.dreamwidth.org/247406.html). *Graydon2*. Dreamwidth Studios. Archived (https://web.archive.org/web/20190502181357/https://graydon2.dreamwidth.org/247406.html) from the original on May 2, 2019. Retrieved May 13, 2019.
13. "FAQ – The Rust Project" (https://web.archive.org/web/20160609195720/https://www.rust-lang.org/faq.html#project). *Rust-lang.org*. Archived from the original (https://www.rust-lang.org/faq.html#project) on June 9, 2016. Retrieved June 27, 2019.
14. "Rust vs. C++ Comparison" (https://www.apriorit.com/dev-blog/520-rust-vs-c-comparison). Archived (https://web.archive.org/web/20181120221225/https://www.apriorit.com/dev-blog/520-rust-vs-c-comparison) from the original on November 20, 2018. Retrieved November 20, 2018.
15. "Unsafe Rust" (https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html). Archived (https://web.archive.org/web/20201014032016/https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html) from the original on October 14, 2020. Retrieved October 17, 2020.
16. "Fearless Security: Memory Safety" (https://hacks.mozilla.org/2019/01/fearless-security-memory-safety/). Archived (https://web.archive.org/web/20201108003116/https://hacks.mozilla.org/2019/01/fearless-security-memory-safety/) from the original on November 8, 2020. Retrieved November 4, 2020.
17. "Rc<T>, the Reference Counted Smart Pointer" (https://doc.rust-lang.org/book/ch15-04-rc.html). Archived (https://web.archive.org/web/20201111223851/https://doc.rust-lang.org/book/ch15-04-rc.html) from the original on November 11, 2020. Retrieved November 4, 2020.
18. "Rust language" (https://research.mozilla.org/rust/). Archived (https://web.archive.org/web/20200906132647/https://research.mozilla.org/rust/) from the original on September 6, 2020. Retrieved September 9, 2020. "Mozilla was the first investor for Rust and continues to sponsor the work of the open source project. Mozilla also utilizes Rust in many of its core initiatives including Servo and key parts of Firefox."
19. Noel (July 8, 2010). "The Rust Language" (http://lambda-the-ultimate.org/node/4009). Lambda the Ultimate. Archived (https://www.webcitation.org/6COMjHMod?url=http://lambda-the-ultimate.org/node/4009) from the original on November 23, 2012. Retrieved October 30, 2010.
20. "Contributors to rust-lang/rust" (https://github.com/rust-lang/rust/graphs/contributors). *GitHub*. Archived (https://web.archive.org/web/20200526051128/https://github.com/rust-lang/rust/graphs/contributors) from the original on May 26, 2020. Retrieved October 12, 2018.
21. Bright, Peter (April 3, 2013). "Samsung teams up with Mozilla to build browser engine for multicore machines" (https://arstechnica.com/information-technology/2013/04/samsung-teams-up-with-mozilla-to-build-browser-engine-for-multicore-machines/). *Ars Technica*. Archived (https://web.archive.org/web/20161216003838/http://arstechnica.com/information-technology/2013/04/samsung-teams-up-with-mozilla-to-build-browser-engine-for-multicore-machines/) from the original on December 16, 2016. Retrieved April 4, 2013.
22. "Why Rust for safe systems programming" (https://msrc-blog.microsoft.com/2019/07/22/why-rust-for-safe-systems-programming/). Archived (https://web.archive.org/web/20190722200126/https://msrc-blog.microsoft.com/2019/07/22/why-rust-for-safe-systems-programming/) from the original on July 22, 2019. Retrieved July 22, 2019.
23. "How Microsoft Is Adopting Rust" (https://medium.com/the-innovation/how-microsoft-is-adopting-rust-e0f8816566ba). August 6, 2020. Archived (https://web.archive.org/web/20200810172211/https://medium.com/the-innovation/how-microsoft-is-adopting-rust-e0f8816566ba) from the original on August 10, 2020. Retrieved August 7, 2020.
24. "Stack Overflow Developer Survey 2021" (https://insights.stackoverflow.com/survey/2021). *Stack Overflow*. Retrieved August 3, 2021.

25. Hoare, Graydon (June 7, 2014). "Internet archaeology: the definitive, end-all source for why Rust is named "Rust" " (https://www.reddit.com/r/rust/comments/27jvdt/internet_archaeology_the_definitive_endall_source/). *Reddit.com*. Archived (https://web.archive.org/web/20160714041250/https://www.reddit.com/r/rust/comments/27jvdt/internet_archaeology_the_definitive_endall_source/) from the original on July 14, 2016. Retrieved November 3, 2016.
26. "Future Tense" (http://www.slideshare.net/BrendanEich/future-tense-7782010). April 29, 2011. Archived (https://www.webcitation.org/6AlZGgr8a?url=http://www.slideshare.net/BrendanEich/future-tense-7782010) from the original on September 18, 2012. Retrieved February 6, 2012.
27. Hoare, Graydon (July 7, 2010). *Project Servo* (http://venge.net/graydon/talks/intro-talk-2.pdf) (PDF). Mozilla Annual Summit 2010. Whistler, Canada. Archived (https://web.archive.org/web/20170711131514/http://venge.net/graydon/talks/intro-talk-2.pdf) (PDF) from the original on July 11, 2017. Retrieved February 22, 2017.
28. Hoare, Graydon (October 2, 2010). "Rust Progress" (https://web.archive.org/web/20140815054745/http://blog.mozilla.org/graydon/2010/10/02/rust-progress/). Archived from the original (http://blog.mozilla.com/graydon/2010/10/02/rust-progress/) on August 15, 2014. Retrieved October 30, 2010.
29. Hoare, Graydon (April 20, 2011). "[rust-dev] stage1/rustc builds" (https://mail.mozilla.org/pipermail/rust-dev/2011-April/000330.html). Archived (https://web.archive.org/web/20110720122600/https://mail.mozilla.org/pipermail/rust-dev/2011-April/000330.html) from the original on July 20, 2011. Retrieved April 20, 2011.
30. catamorphism (January 20, 2012). "Mozilla and the Rust community release Rust 0.1 (a strongly-typed systems programming language with a focus on memory safety and concurrency)" (https://www.reddit.com/r/programming/comments/opgxd/mozilla_and_the_rust_community_release_rust_01_a/). Archived (https://web.archive.org/web/20120124162132/http://www.reddit.com/r/programming/comments/opgxd/mozilla_and_the_rust_community_release_rust_01_a) from the original on January 24, 2012. Retrieved February 6, 2012.
31. "Version History" (https://github.com/rust-lang/rust/blob/master/RELEASES.md). *GitHub*. Archived (https://web.archive.org/web/20150515221302/https://github.com/rust-lang/rust/blob/master/RELEASES.md) from the original on May 15, 2015. Retrieved January 1, 2017.
32. The Rust Core Team (May 15, 2015). "Announcing Rust 1.0" (http://blog.rust-lang.org/2015/05/15/Rust-1.0.html). Archived (https://web.archive.org/web/20150515171337/http://blog.rust-lang.org/2015/05/15/Rust-1.0.html) from the original on May 15, 2015. Retrieved December 11, 2015.
33. "Scheduling the Trains" (https://blog.rust-lang.org/2014/12/12/1.0-Timeline.html). Archived (https://web.archive.org/web/20170102080055/https://blog.rust-lang.org/2014/12/12/1.0-Timeline.html) from the original on January 2, 2017. Retrieved January 1, 2017.
34. "G - How Rust is Made and "Nightly Rust" - The Rust Programming Language" (https://doc.rust-lang.org/book/appendix-07-nightly-rust.html). *doc.rust-lang.org*. Retrieved May 22, 2021.
35. "What are editions? - The Edition Guide" (https://doc.rust-lang.org/edition-guide/editions/index.html). *doc.rust-lang.org*. Retrieved May 22, 2021.
36. Strom, Robert E.; Yemini, Shaula (1986). "Typestate: A Programming Language Concept for Enhancing Software Reliability" (https://www.cs.cmu.edu/~aldrich/papers/classic/tse12-typestate.pdf) (PDF). *IEEE Transactions on Software Engineering*: 157–171. doi:10.1109/TSE.1986.6312929 (https://doi.org/10.1109%2FTSE.1986.6312929). ISSN 0098-5589 (https://www.worldcat.org/issn/0098-5589). S2CID 15575346 (https://api.semanticscholar.org/CorpusID:15575346). Archived (https://web.archive.org/web/20100714124606/http://www.cs.cmu.edu/~aldrich/papers/classic/tse12-typestate.pdf) (PDF) from the original on July 14, 2010. Retrieved November 14, 2010.
37. Walton, Patrick (December 26, 2012). "Typestate Is Dead, Long Live Typestate!" (https://pcwalton.github.io/2012/12/26/typestate-is-dead.html). *GitHub*. Archived (https://web.archive.org/web/20180223120322/https://pcwalton.github.io/2012/12/26/typestate-is-dead.html) from the original on February 23, 2018. Retrieved November 3, 2016.

38. Biffle, Cliff (June 5, 2019). "The Typestate Pattern in Rust" (https://cliffle.com/blog/rust-typestate/). Archived (https://web.archive.org/web/20210206052539/https://cliffle.com/blog/rust-typestate/) from the original on February 6, 2021. Retrieved February 1, 2021.
39. "[rust-dev] Rust 0.4 released" (https://mail.mozilla.org/pipermail/rust-dev/2012-October/002489.html). *mail.mozilla.org*. Retrieved October 31, 2021.
40. Binstock, Andrew. "The Rise And Fall of Languages in 2013" (http://www.drdobbs.com/jvm/the-rise-and-fall-of-languages-in-2013/240165192). *Dr Dobb's*. Archived (https://web.archive.org/web/20160807075745/http://www.drdobbs.com/jvm/the-rise-and-fall-of-languages-in-2013/240165192) from the original on August 7, 2016. Retrieved December 11, 2015.
41. "Safe Interoperability between Rust and C++ with CXX" (https://www.infoq.com/news/2020/12/cpp-rust-interop-cxx/). *InfoQ*. December 6, 2020. Retrieved January 3, 2021.
42. Cimpanu, Catalin (August 11, 2020). "Mozilla lays off 250 employees while it refocuses on commercial products" (https://www.zdnet.com/article/mozilla-lays-off-250-employees-while-it-refocuses-on-commercial-products/). *ZDNet*. Retrieved December 2, 2020.
43. Cooper, Daniel (August 11, 2020). "Mozilla lays off 250 employees due to the pandemic" (https://www.engadget.com/mozilla-firefox-250-employees-layoffs-151324924.html). *Engadget*. Archived (https://web.archive.org/web/20201213020220/https://www.engadget.com/mozilla-firefox-250-employees-layoffs-151324924.html) from the original on December 13, 2020. Retrieved December 2, 2020.
44. @tschneidereit (August 12, 2020). "Much of the team I used to manage was part of the Mozilla layoffs on Tuesday. That team was Mozilla's Rust team, and Mozilla's Wasmtime team. I thought I'd know how to talk about it by now, but I don't. It's heartbreaking, incomprehensible, and staggering in its impact" (https://twitter.com/tschneidereit/status/1293868141953667074) (Tweet). Retrieved December 2, 2020 – via Twitter.
45. @asajeffrey (August 11, 2020). "Mozilla is closing down the team I'm on, so I am one of the many folks now wondering what the next gig will be. It's been a wild ride!" (https://twitter.com/asajeffrey/status/1293220656339988483) (Tweet). Retrieved December 2, 2020 – via Twitter.
46. Kolakowski, Nick (August 27, 2020). "Is Rust in Trouble After Big Mozilla Layoffs?" (https://insights.dice.com/2020/08/27/rust-in-trouble-after-big-mozilla-layoffs/). *Dice*. Archived (https://web.archive.org/web/20201124184935/https://insights.dice.com/2020/08/27/rust-in-trouble-after-big-mozilla-layoffs/) from the original on November 24, 2020. Retrieved December 2, 2020.
47. "Laying the foundation for Rust's future" (https://blog.rust-lang.org/2020/08/18/laying-the-foundation-for-rusts-future.html). *Rust Blog*. August 18, 2020. Archived (https://web.archive.org/web/20201202022933/https://blog.rust-lang.org/2020/08/18/laying-the-foundation-for-rusts-future.html) from the original on December 2, 2020. Retrieved December 2, 2020.
48. "Rust Foundation" (https://foundation.rust-lang.org/). *foundation.rust-lang.org*. February 8, 2021. Archived (https://web.archive.org/web/20210209010632/https://foundation.rust-lang.org/) from the original on February 9, 2021. Retrieved February 9, 2021.
49. "Mozilla Welcomes the Rust Foundation" (https://blog.mozilla.org/blog/2021/02/08/mozilla-welcomes-the-rust-foundation). *Mozilla Blog*. February 9, 2021. Archived (https://web.archive.org/web/20210208212031/https://blog.mozilla.org/blog/2021/02/08/mozilla-welcomes-the-rust-foundation/) from the original on February 8, 2021. Retrieved February 9, 2021.
50. Amadeo, Ron (April 7, 2021). "Google is now writing low-level Android code in Rust" (https://arstechnica.com/gadgets/2021/04/google-is-now-writing-low-level-android-code-in-rust/). *Ars Technica*. Archived (https://web.archive.org/web/20210408001446/https://arstechnica.com/gadgets/2021/04/google-is-now-writing-low-level-android-code-in-rust/) from the original on April 8, 2021. Retrieved April 8, 2021.
51. "rust/src/grammar/parser-lalr.y" (https://github.com/rust-lang/rust/blob/5b13bff5203c1bdc6ac6dc87f69b5359a9503078/src/grammar/parser-lalr.y#L1309-L1573). *GitHub*. May 23, 2017. Retrieved May 23, 2017.

52. Avram, Abel (August 3, 2012). "Interview on Rust, a Systems Programming Language Developed by Mozilla" (http://www.infoq.com/news/2012/08/Interview-Rust). InfoQ. Archived (https://web.archive.org/web/20130724045852/http://www.infoq.com/news/2012/08/Interview-Rust) from the original on July 24, 2013. Retrieved August 17, 2013.
53. "Debian -- Details of package rustc in sid" (https://packages.debian.org/sid/main/rustc). *packages.debian.org*. Archived (https://web.archive.org/web/20170222053421/https://packages.debian.org/sid/main/rustc) from the original on February 22, 2017. Retrieved February 21, 2017.
54. Rosenblatt, Seth (April 3, 2013). "Samsung joins Mozilla's quest for Rust" (http://reviews.cnet.com/8301-3514_7-57577639/samsung-joins-mozillas-quest-for-rust/). Archived (https://web.archive.org/web/20130404142333/http://reviews.cnet.com/8301-3514_7-57577639/samsung-joins-mozillas-quest-for-rust/) from the original on April 4, 2013. Retrieved April 5, 2013.
55. Brown, Neil (April 17, 2013). "A taste of Rust" (https://lwn.net/Articles/547145/). Archived (https://web.archive.org/web/20130426010754/http://lwn.net/Articles/547145/) from the original on April 26, 2013. Retrieved April 25, 2013.
56. "Races - The Rustonomicon" (https://doc.rust-lang.org/nomicon/races.html). *doc.rust-lang.org*. Archived (https://web.archive.org/web/20170710194643/https://doc.rust-lang.org/nomicon/races.html) from the original on July 10, 2017. Retrieved July 3, 2017.
57. "The Rust Language FAQ" (https://web.archive.org/web/20150420104147/http://static.rust-lang.org/doc/master/complement-lang-faq.html). *static.rust-lang.org*. 2015. Archived from the original (http://static.rust-lang.org/doc/master/complement-lang-faq.html) on April 20, 2015. Retrieved April 24, 2017.
58. "RAII – Rust By Example" (https://doc.rust-lang.org/rust-by-example/scope/raii.html). *doc.rust-lang.org*. Archived (https://web.archive.org/web/20190421131142/https://doc.rust-lang.org/rust-by-example/scope/raii.html) from the original on April 21, 2019. Retrieved November 22, 2020.
59. "Abstraction without overhead: traits in Rust" (https://blog.rust-lang.org/2015/05/11/traits.html). *Rust Blog*.
60. Klabnik, Steve; Nichols, Carol (June 2018). "Chapter 4: Understanding Ownership". *The Rust Programming Language* (https://nostarch.com/rust). San Francisco, California: No Starch Press. p. 44. ISBN 978-1-593-27828-1. Archived (https://web.archive.org/web/20190503092648/https://nostarch.com/Rust) from the original on May 3, 2019. Retrieved May 14, 2019.
61. "The Rust Programming Language: What is Ownership" (https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html). *Rust-lang.org*. Archived (https://web.archive.org/web/20190519093808/https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html) from the original on May 19, 2019. Retrieved May 14, 2019.
62. Walton, Patrick (October 1, 2010). "Rust Features I: Type Inference" (http://pcwalton.blogspot.com/2010/10/rust-features-i-type-inference.html). Archived (https://web.archive.org/web/20110708060229/http://pcwalton.blogspot.com/2010/10/rust-features-i-type-inference.html) from the original on July 8, 2011. Retrieved January 21, 2011.
63. *rust-lang/rustup* (https://github.com/rust-lang/rustup), The Rust Programming Language, May 17, 2021, retrieved May 17, 2021
64. "Why Cargo Exists" (https://doc.rust-lang.org/cargo/guide/why-cargo-exists.html). *The Cargo Book*. Retrieved May 18, 2021.
65. "Specifying Dependencies - The Cargo Book" (https://doc.rust-lang.org/cargo/reference/specifying-dependencies.html). *doc.rust-lang.org*. Retrieved May 17, 2021.
66. "rust-dev-tools/fmt-rfcs" (https://github.com/rust-dev-tools/fmt-rfcs). *GitHub*. Retrieved September 21, 2021.
67. "rustfmt" (https://github.com/rust-lang/rustfmt). *GitHub*. Retrieved May 19, 2021.
68. "Create README.md · rust-lang/rust-clippy@507dc2b" (https://github.com/rust-lang/rust-clippy/commit/507dc2b7ec30cf94554b441d4fcb1ce113f98a16). *GitHub*. Retrieved November 22, 2021.

69. "Day 1 - cargo subcommands | 24 days of Rust" (https://zsiciarz.github.io/24daysofrust/book/vol2/day1.html). *zsiciarz.github.io*. Retrieved November 22, 2021.
70. "rust-lang/rust-clippy" (https://github.com/rust-lang/rust-clippy). *GitHub*. Retrieved May 21, 2021.
71. "ALL the Clippy Lints" (https://rust-lang.github.io/rust-clippy/). Retrieved May 22, 2021.
72. "rust-lang/rls" (https://github.com/rust-lang/rls). *GitHub*. Retrieved May 26, 2021.
73. "racer-rust/racer" (https://github.com/racer-rust/racer). *GitHub*. Retrieved May 26, 2021.
74. "Procedural Macros" (https://doc.rust-lang.org/reference/procedural-macros.html). *The Rust Programming Language Reference*. Archived (https://web.archive.org/web/20201107233444/https://doc.rust-lang.org/reference/procedural-macros.html) from the original on November 7, 2020. Retrieved March 23, 2021.
75. "Serde Derive" (https://serde.rs/derive.html). *Serde Derive documentation*. Archived (https://web.archive.org/web/20210417114849/https://serde.rs/derive.html) from the original on April 17, 2021. Retrieved March 23, 2021.
76. "extendr_api - Rust" (https://extendr.github.io/extendr/extendr_api/index.html). *Extendr Api Documentation*. Retrieved March 23, 2021.
77. Walton, Patrick (December 5, 2010). "C++ Design Goals in the Context of Rust" (http://pcwalton.blogspot.com/2010/12/c-design-goals-in-context-of-rust.html). Archived (https://web.archive.org/web/20101209142602/http://pcwalton.blogspot.com/2010/12/c-design-goals-in-context-of-rust.html) from the original on December 9, 2010. Retrieved January 21, 2011.
78. "How Fast Is Rust?" (https://doc.rust-lang.org/1.0.0/complement-lang-faq.html#how-fast-is-rust?). *The Rust Programming Language FAQ*. Archived (https://web.archive.org/web/20201028102013/https://doc.rust-lang.org/1.0.0/complement-lang-faq.html#how-fast-is-rust?) from the original on October 28, 2020. Retrieved April 11, 2019.
79. "Getting Started" (https://www.rust-lang.org/learn/get-started#ferris). *rust-lang.org*. Archived (https://web.archive.org/web/20201101145703/https://www.rust-lang.org/learn/get-started#ferris) from the original on November 1, 2020. Retrieved October 11, 2020.
80. "Stack Overflow Developer Survey 2015" (https://stackoverflow.com/research/developer-survey-2015). *Stackoverflow.com*. Archived (https://web.archive.org/web/20161231012855/https://stackoverflow.com/research/developer-survey-2015) from the original on December 31, 2016. Retrieved November 3, 2016.
81. "Stack Overflow Developer Survey 2019" (https://insights.stackoverflow.com/survey/2019/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2019). *Stack Overflow*. Archived (https://web.archive.org/web/20201008033536/https://insights.stackoverflow.com/survey/2019/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2019) from the original on October 8, 2020. Retrieved March 31, 2021.
82. "Stack Overflow Developer Survey 2021" (https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-language-love-dread). *Stack Overflow*. Retrieved August 24, 2021.
83. Yegulalp, Serdar (April 3, 2015). "Mozilla's Rust-based Servo browser engine inches forward" (http://www.infoworld.com/article/2905688/applications/mozillas-rust-based-servo-browser-engine-inches-forward.html). *InfoWorld*. Archived (https://web.archive.org/web/20160316145230/http://www.infoworld.com/article/2905688/applications/mozillas-rust-based-servo-browser-engine-inches-forward.html) from the original on March 16, 2016. Retrieved March 15, 2016.
84. Lardinois, Frederic (April 3, 2015). "Mozilla And Samsung Team Up To Develop Servo, Mozilla's Next-Gen Browser Engine For Multicore Processors" (https://techcrunch.com/2013/04/03/mozilla-and-samsung-collaborate-on-servo-mozillas-next-gen-browser-engine-for-tomorrows-multicore-processors/). *TechCrunch*. Archived (https://web.archive.org/web/20160910211537/https://techcrunch.com/2013/04/03/mozilla-and-samsung-collaborate-on-servo-mozillas-next-gen-browser-engine-for-tomorrows-multicore-processors/) from the original on September 10, 2016. Retrieved June 25, 2017.

85. Bryant, David (October 27, 2016). "A Quantum Leap for the web" (https://medium.com/mozilla-tech/a-quantum-leap-for-the-web-a3b7174b3c12#.ldic6a78e). *Medium*. Archived (https://web.archive.org/web/20201209013807/https://medium.com/mozilla-tech/a-quantum-leap-for-the-web-a3b7174b3c12#.ldic6a78e) from the original on December 9, 2020. Retrieved October 27, 2016.
86. Yegulalp, Serdar. "Rust's Redox OS could show Linux a few new tricks" (http://www.infoworld.com/article/3046100/open-source-tools/rusts-redox-os-could-show-linux-a-few-new-tricks.html). infoworld. Archived (https://web.archive.org/web/20160321192838/http://www.infoworld.com/article/3046100/open-source-tools/rusts-redox-os-could-show-linux-a-few-new-tricks.html) from the original on March 21, 2016. Retrieved March 21, 2016.
87. "Introduction to Theseus" (https://theseus-os.github.io/Theseus/book/index.html). *Theseus OS Book*. Retrieved July 11, 2021.
88. Garbutt, James (January 27, 2019). "First thoughts on Deno, the JavaScript/TypeScript runtime" (https://43081j.com/2019/01/first-look-at-deno). *43081j.com*. Archived (https://web.archive.org/web/20201107224127/https://43081j.com/2019/01/first-look-at-deno) from the original on November 7, 2020. Retrieved September 27, 2019.
89. Howarth, Jesse (February 4, 2020). "Why Discord is switching from Go to Rust" (https://blog.discord.com/why-discord-is-switching-from-go-to-rust-a190bbca2b1f). Archived (https://web.archive.org/web/20200630181517/https://blog.discord.com/why-discord-is-switching-from-go-to-rust-a190bbca2b1f) from the original on June 30, 2020. Retrieved April 14, 2020.
90. Vishnevskiy, Stanislav (July 6, 2017). "How Discord Scaled Elixir to 5,000,000 Concurrent Users" (https://blog.discord.com/scaling-elixir-f9b8e1e7c29b). *Discord Blog*.
91. "Google Fushcia's source code" (https://fuchsia.googlesource.com/fuchsia/+/refs/heads/main/tools). *Google Git*. Retrieved July 2, 2021.
92. Nichols, Shaun (June 27, 2018). "Microsoft's next trick? Kicking things out of the cloud to Azure IoT Edge" (https://www.theregister.co.uk/2018/06/27/microsofts_next_cloud_trick_kicking_things_out_of_the_cloud_to_azure_iot_edge/). *The Register*. Archived (https://web.archive.org/web/20190927092433/https://www.theregister.co.uk/2018/06/27/microsofts_next_cloud_trick_kicking_things_out_of_the_cloud_to_azure_iot_edge/) from the original on September 27, 2019. Retrieved September 27, 2019.
93. Balbaert, Ivo (May 27, 2015). *Rust Essentials* (https://books.google.com/books?id=TeiuCQAAQBAJ&q=OpenDNS+Rust&pg=PA6). Packt Publishing. p. 6. ISBN 978-1785285769. Retrieved March 21, 2016.
94. Frank, Denis (December 5, 2013). "Using HyperLogLog to Detect Malware Faster Than Ever" (https://umbrella.cisco.com/blog/2013/12/05/hyperloglog-and-malware-detection/). *OpenDNS Security Labs*. Archived (https://web.archive.org/web/20170814113953/https://umbrella.cisco.com/blog/2013/12/05/hyperloglog-and-malware-detection/) from the original on August 14, 2017. Retrieved March 19, 2016.
95. Denis, Frank (October 4, 2013). "ZeroMQ: Helping us Block Malicious Domains in Real Time" (https://umbrella.cisco.com/blog/2013/10/04/zeromq-helping-us-block-malicious-domains/). *OpenDNS Security Labs*. Archived (https://web.archive.org/web/20170814114338/https://umbrella.cisco.com/blog/2013/10/04/zeromq-helping-us-block-malicious-domains/) from the original on August 14, 2017. Retrieved March 19, 2016.
96. "Ruffle" (https://ruffle.rs/#what-is-ruffle). *Ruffle*. Archived (https://web.archive.org/web/20210126040413/https://ruffle.rs/#what-is-ruffle) from the original on January 26, 2021. Retrieved April 14, 2021.
97. Sei, Mark (October 10, 2018). "Fedora 29 new features: Startis now officially in Fedora" (https://www.marksei.com/fedora-29-new-features-startis/). *Marksei, Weekly sysadmin pills*. Archived (https://web.archive.org/web/20190413075055/https://www.marksei.com/fedora-29-new-features-startis/) from the original on April 13, 2019. Retrieved May 13, 2019.

98. "RHEL 8: Chapter 8. Managing layered local storage with Stratis" (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8-beta/html/configuring_and_managing_file_systems/managing-layered-local-storage-with-stratis_configuring-and-managing-file-systems). October 10, 2018. Archived (https://web.archive.org/web/20190413145448/https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8-beta/html/configuring_and_managing_file_systems/managing-layered-local-storage-with-stratis_configuring-and-managing-file-systems) from the original on April 13, 2019. Retrieved April 13, 2019.
99. *terminusdb/terminusdb-store* (https://github.com/terminusdb/terminusdb-store), TerminusDB, December 14, 2020, archived (https://web.archive.org/web/20201215165359/https://github.com/terminusdb/terminusdb-store) from the original on December 15, 2020, retrieved December 14, 2020
100. *9 Companies That Use Rust in Production* (https://serokell.io/blog/rust-companies), Serokell, November 18, 2020, retrieved October 7, 2021
101. Krill, Paul. "Rust language moves to independent foundation" (https://www.infoworld.com/article/3606774/rust-language-moves-to-independent-foundation.html). InfoWorld. Archived (https://web.archive.org/web/20210410161528/https://www.infoworld.com/article/3606774/rust-language-moves-to-independent-foundation.html) from the original on April 10, 2021. Retrieved April 10, 2021.
102. Vaughan-Nichols, Steven J. (April 9, 2021). "AWS's Shane Miller to head the newly created Rust Foundation" (https://www.zdnet.com/article/awss-shane-miller-to-head-the-newly-created-rust-foundation/). ZDNet. Archived (https://web.archive.org/web/20210410031305/https://www.zdnet.com/article/awss-shane-miller-to-head-the-newly-created-rust-foundation/) from the original on April 10, 2021. Retrieved April 10, 2021.
103. Vaughan-Nichols, Steven J. (November 17, 2021). "Rust Foundation appoints Rebecca Rumbul as executive director" (https://www.zdnet.com/article/rust-foundation-appoints-rebecca-rumbul-as-executive-director/). *ZDNet*. Retrieved November 18, 2021.
104. "The Rust programming language now has its own independent foundation" (https://www.techrepublic.com/article/the-rust-programming-language-now-has-its-own-independent-foundation/). *TechRepublic*. February 10, 2021. Retrieved November 18, 2021.
105. "RustConf 2020 - Thursday, August 20" (https://rustconf.com/). *rustconf.com*. Archived (https://web.archive.org/web/20190825021524/https://rustconf.com/) from the original on August 25, 2019. Retrieved August 25, 2019.
106. *Rust Belt Rust* (https://rust-belt-rust.com/). Dayton, Ohio. October 18, 2019. Archived (https://web.archive.org/web/20190514091451/https://rust-belt-rust.com/) from the original on May 14, 2019. Retrieved May 14, 2019.
107. *RustFest* (https://blog.rustfest.eu/past_events/). Barcelona, Spain: asquera Event UG. 2019. Archived (https://web.archive.org/web/20190424135454/https://blog.rustfest.eu/past_events/) from the original on April 24, 2019. Retrieved May 14, 2019.
108. "Oxidize Global" (https://oxidizeconf.com/). *Oxidize Berlin Conference*. Retrieved February 1, 2021.

External links

- Official website (https://www.rust-lang.org/)
- Rust-lang (https://github.com/rust-lang) on GitHub

Retrieved from "https://en.wikipedia.org/w/index.php?title=Rust_(programming_language)&oldid=1058448483"

This page was last edited on 3 December 2021, at 15:55 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia

Foundation, Inc., a non-profit organization.