Das Rust Ecosystem

Inhaltsverzeichnis

1.0 Build-Prozess

Der Build-Prozess umfasst alles, was nötig ist, um aus dem Quellcode ein ausführbares Programm oder eine Bibliothek zu erstellen.

1.2 Rust Up

- Installation & Updates von Rusts Werkzeugen
- Verwaltung mehrerer Rust-Versionen
- Wechsel der Rust Version pro Projekt

```
rustup override set <version>
```

1.3 Cargo

- Zentrales Build-System
- Automatisiert den Kompilierungsprozess
- Paket Manager
- Ermöglicht Wechsel zwischen Build-Profilen
- Tests

1.3.1 cargo clippy

```
warning: this let-binding has unit value
 --> src/main.rs:5:5
  |
5 | let x = println!("Hello, world!");
  |     ^ help: consider removing `let x =`
  |
  = note: `#[warn(clippy::let_unit_value)]` on
by default
```

1.3.2 cargo check

```
error[E0425]: cannot find value `y` in this
 scope
  --> src/main.rs:3:13
   |
3  |     let x = y + 1;
   |             ^ not found in this scope
   |
   = help: maybe you meant to write `x`?

error: aborting due to previous error
```

1.3.3 cargo build & cargo run

- cargo build -> Kompilierung im Debug-Modus
- Wird im target/Debug-Verzeichnis gespeichert
- Für schnelles kompilieren

```
cargo build
```

- cargo run -> Kompiliert und führt Datei aus

```
cargo run
```

1.3.4 cargo build –-release & cargo run --release

- cargo build --release -> Kompilierung im Release-Modus
- Wird im target/release-Verzeichnis gespeichert
- Für optimiertes kompilieren

```
cargo build --release
```

- cargo run --release -> Kompiliert und führt Datei aus

```
cargo run --release
```

1.3.5 Inkrementelles Kompilieren

cargo build

- Kompiliert nur geänderte Dateien/Module
- -> build-Zeit verkürzt

cargo build --release

- Inkrementelle Kompilierung standardgemäß deaktiviert
- -> maximale Optimierung

1.3.6 Cross Compiler

- Zielplattform Hinzufügen

```
rustup target add x86_64-pc-windows-gnu
```

- Für diese kompilieren

```
cargo build --target = x86_64-pc-windows-gnu
```

1.3.7 rustc

- Übersetzt Rust-Code in Maschinencode
- Führt Syntax-und Typprüfung durch

1.4 C-Compiler

- Manchmal benötigt, da Crates/Build-Skripte C-Code enthalten

1.5 Linker

Der Linker sorgt dafür, dass alle Funktionen und externen Bibliotheken korrekt zu einem ausführbaren Programm verbunden werden.

2.0 Central Repository Crates.io

- Zentrale Sammelstelle für Softwarepakete
- Verwaltet Crates und stellt sie bereit
- Crates.io ist das zentrale Repository für Rust

2.1 Crates & Features

Crates

- Zusätzliche Funktionalitäten, ohne sie selbst implementieren zu müssen

Features:

- Optionale Funktionalitäten eines Crates
- Es wird nur das kompiliert, was benötigt wird

2.2 cargo.toml

```
[dependencies]
serde = "1.0"
tokio = { version = "1", features    = ["full"] }
```

2.3 cargo.lock

```
[[package]]
name = "serde"
version = "1.0.217„
dependencies = [
"serde_derive",
]
```

2.4 cargo publish

- Mit `cargo publish` eigene Crates erstellen
- Wichtig: cargo.toml richtig Konfigurieren
- Crates.io-Konto

2.5 Git

- Es ist möglich, Crates aus einem Git-Repository anzugeben
- Eignet sich für selbstgeschriebene/unveröffentlichte Crates

```
[dependencies]
meine_crate = { git =
"https://github.com/benutzer/meine_crate.git" }
```

3.0 Testen

```
cargo test
```

```
#[cfg(test)]
```

```
#[test]
```

Fazit

- Leistungsstarkes & effizientes Ökosystem
- Automatisierter Build-Prozess mit Cargo
- Zentrales Repository: crates.io

Quellen

cargo (allgemeine Einführung & Funktionen):
https://doc.rust-lang.org/book/ch01-03-hello-cargo.html
cargo build (Debug & Release builds, Unterschiede):
https://doc.rust-lang.org/book/ch14-01-release-profiles.html
2. Zentrales Repository – Crates.io:
https://doc.rust-lang.org/book/ch14-02-publishing-to-crates-io.html
Crates & Features in cargo.toml:
https://doc.rust-lang.org/book/ch14-03-cargo-workspaces.html#managing-dependencies
Crates aus Git-Repositories:
https://doc.rust-lang.org/cargo/reference/specifying-dependencies.html#specifying-dependencies-from-git-repositories
Test-Attribute & Unit-Tests schreiben:
https://doc.rust-lang.org/book/ch11-01-writing-tests.html
Tests:
https://doc.rust-lang.org/book/ch11-03-test-organization.html#integration-tests
cargo Test (Test-Befehle & Debugging):