Cargo Scan Manual

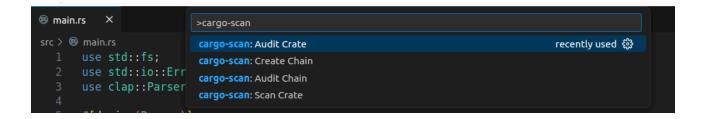
Build and install the extension

- 1. Clone the repository
- 2. Build the extension by running make in the editor-plugin directory. This will produce a .vsix file in that location.
- 3. Launch VS Code and install the extension from the .vsix file:
 - a. Press Ctrl + Shift + P (Linux) or Cmd + Shift + P (macOS) to open the Command Palette.
 - b. Type Extensions: Install from VSIX... and select it.
 - c. Navigate to the folder where the .vsix file is located and open it.

Run the extension

After installing the extension in VS Code, open the Rust crate for audit in VS Code.

- 1. Press Ctrl + Shift + P (Linux) or Cmd + Shift + P (macOS) to open the Command Palette.
- 2. Type cargo-scan: Audit Crate to scan the crate and perform an audit.



View & Audit Effects

After the crate scan is complete, the list of effects will be displayed in the Effects view of the Explorer panel. The effects' locations will be highlighted in the source code and will be annotated with Cargo Scan's safety annotations, to be marked accordingly during an audit.

Cargo Scan introduces the following safety annotations:

- 1. **Safe**: Effect is always safe to call
- 2. **Unsafe**: Effect is never safe to call indicates a bug

3. **Caller-Checked:** Effect's safety depends on the calling context and cannot be determined locally at its location

```
EXPLORER
                                         ® main.rs

✓ INTRO-EXAMPLE

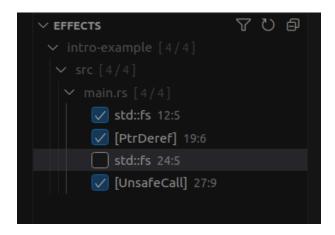
                                          src > 📵 main.rs
                                                 use std::io::Error;
  ® main.rs
                                                 use clap::Parser;
  > target
  ≡ Cargo.lock
                                                  #[derive(Parser)]
 Cargo.toml
                                                  struct Args {
                                                       #[arg(short, long)]
                                                       filepath: String,
> OUTLINE
> TIMELINE
> RUST DEPENDENCIES
                                            11
                                                  fn write_to_file(filepath: &str) -> Result<(), Error> {
                                                     ✓ Safe|! Unsafe|? Caller-Checked fs::write(path: filepath, contents: b"Hello, this is a test")?;
✓ EFFECTS
                                S e
                                            13
                                            14
                                            15
       std::fs 12:5
                                            16
                                                  unsafe fn my_unsafe_fn() {
       [PtrDeref] 19:6
                                                       let x: i32 = 5;
       [UnsafeCall] 27:9
                                                       let y: *mut i32 = x as *mut i32;

✓ Safe | Unsafe | ? Caller-Checked
                intro_example::my_unsafe_fn
                                                       *y = 6;
```

Annotating an effect as Caller-Checked opens a peek view editor with all the caller locations, as shown below.

Reset Safety Annotation

Reset an effect's annotation by unmarking the checkbox next to the effect in the Effects view. If the effect was marked as Caller-Checked prior to resetting, all the caller locations it was propagated to wil be removed.



Filtering Effects

Filter effects by their effect type by pressing the Filter icon in the Effects view and selecting particular effect types. The view is restored to its original state by pressing the Refresh button.

```
File Edit Selection View Go Run Terminal Help
          EXPLORER
                                                     ® main.rs
                                                                             Select filters

✓ INTRO-EXAMPLE

                                                      src > 📵 main.rs
                                                                             [Sink Call]
                                                             use std::
                                                                             ☐ [PtrDeref]
                                                                              > target
  ၀ဌ
                                                                                 [UnsafeCall]
          E Cargo.lock
                                                                                 [UnionField]
                                                              #[derive(
          Cargo.toml
                                                                             ■ [StaticMutVar]
                                                              struct Ar

☐ [StaticExtVar]

                                                                   #[arg
                                                                             [FnPtrCreation]
 <del>LL</del>
                                                                   filepa
                                                                              [ClosureCreation]
        > OUTLINE
                                                                                 [FFI Declaration]
        > TIMELINE
         > RUST DEPENDENCIES
                                                              fn write_to_file(filepath: &str) -> Result<(), Error> {
                                                                   ✓ Safe| | Unsafe| ? Caller-Checked 
fs::write(path: filepath, contents: b"Hello, this is a test")?;
                                       マッ
        ∨ EFFECTS
          \checkmark intro-example [0] Filter Effects by Type
                std::fs 12:5
                                                              unsafe fn my_unsafe_fn() {
                [PtrDeref] 19:6
                                                                   let x: i32 = 5;
let y: *mut i32 = x as *mut i32;

√ Safe ! Unsafe ! ? Caller-Checked
                [UnsafeCall] 27:9
                                                                    *y = 6;
```