Proving Compilers Aren't Hard With the Bevy Linter

BD103

Hi, I'm BD103!

- ➔ Started contributing to Bevy shortly after 0.12 (November 2023)
- ➔ Interest in infrastructure, tooling, and documentation
- ➔ Wrote *bevy-bencher*, *flag-frenzy*, and the 0.14 Migration Guide
- ➔ Helped with the Bevy Contributing Guide and *Leafwing-Studios/cargo-cache*

Something's Wrong Here...

```rust
#[derive(Event)]
struct MyEvent;

App::new()
    .init_resource::<Events<MyEvent>>()
    .run();
```

Something's Wrong Here...

```rust
#[derive(Event)]
struct MyEvent;

App::new()
  // Incorrect ❌
  .init_resource::<Events<MyEvent>>()
  .run();
```

```rust
#[derive(Event)]
struct MyEvent;

App::new()
    // Incorrect ❌
    .init_resource::<Events<MyEvent>>()
    .run();
```

```rust
#[derive(Event)]
struct MyEvent;

App::new()
    // Correct ✅
    .add_event::<MyEvent>()
    .run();
```

Events<T> is a Footgun

Problems

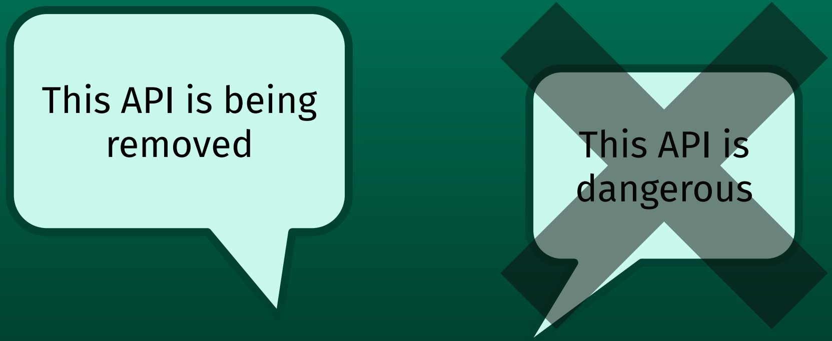
- ➔ Appears to work as intended
- ➔ Easy to discover by beginners
  - ◆ *Events<T>* is in the prelude
  - ◆ It's a resource, which are sanctioned by Bevy's ECS
- ➔ Only becomes problematic over time

Solutions

- ➔ Document that *Events::update()* must be called
- ➔ Remove *Events<T>* from the prelude
- ➔ **Quasi-deprecate it**
- ➔ Make it private

```
#[deprecated = "My message…"]
```

```
#[deprecated = "My message..."]
```



bevy_lint

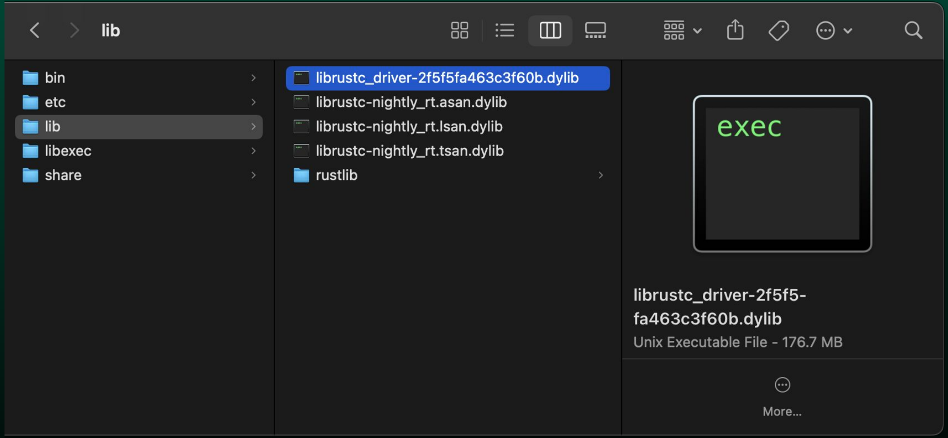
A experimental linter for Bevy projects

Helps developers write better code

- ➔ Interfaces with *rustc*
  - ◆ Can handle all Rust code
  - ◆ Amazing error messages

Install Nightly Rust and *rustc-dev*

```
rustup toolchain install nightly-2024-11-14 \
  --component rustc-dev \
  --component llvm-tools-preview
```

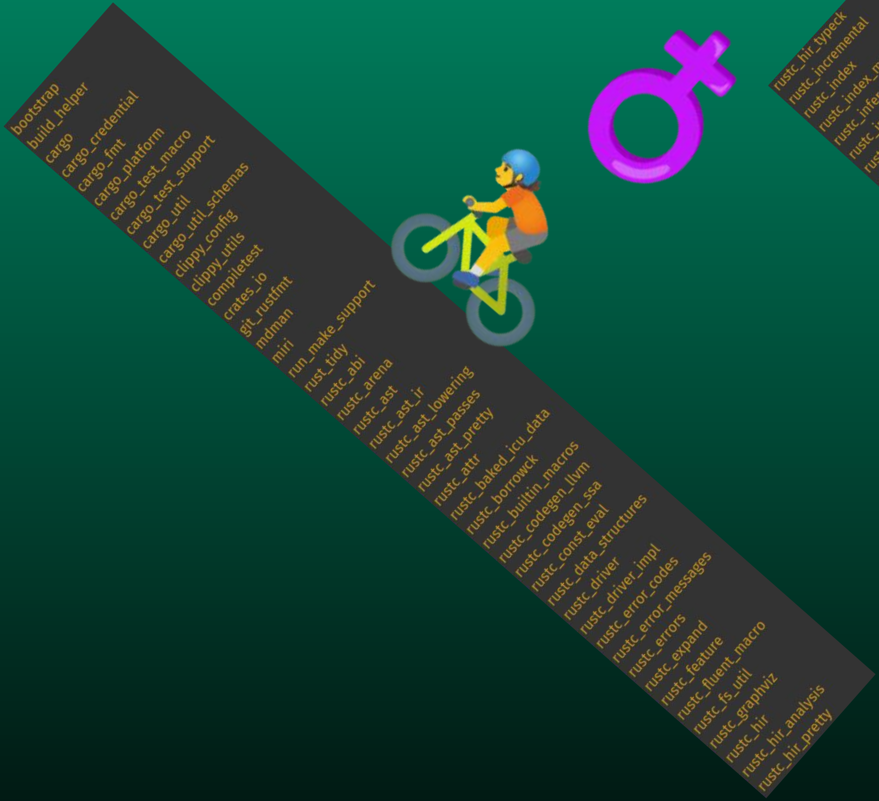

lib
bin
etc
lib
libexec
share
librustc_driver-2f5f5fa463c3f60b.dylib
librustc-nightly_rt.asan.dylib
librustc-nightly_rt.lsan.dylib
librustc-nightly_rt.tsan.dylib
rustlib
exec
librustc_driver-2f5f5-fa463c3f60b.dylib
Unix Executable File - 176.7 MB
More...

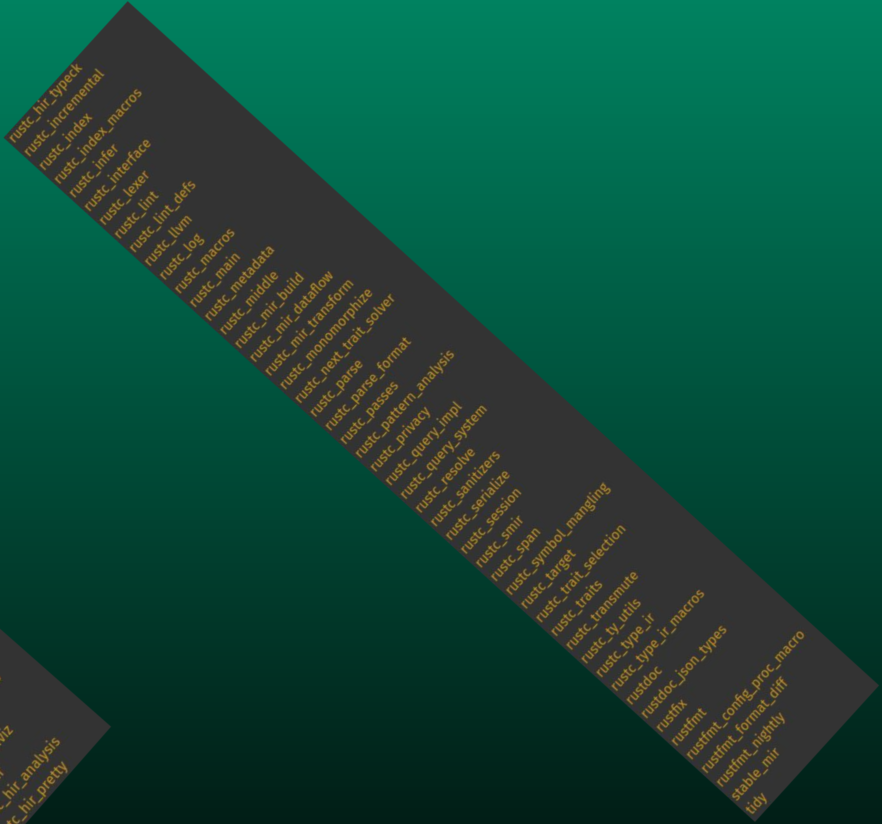
Recreating *rustc* in One Simple Trick

```rust
#![feature(rustc_private)]

extern crate rustc_driver;

fn main() -> ! {
  rustc_driver::main()
}
```


bootstrap
build_helper
cargo
cargo_credential
cargo_fmt
cargo_platform
cargo_test_macro
cargo_test_support
cargo_util
cargo_util_schemas
clippy_config
clippy_utils
compiletest
crates_io
git_rustfmt
mdman
miri
run_make_support
rust_tidy
rustc_abi
rustc_arena
rustc_ast
rustc_ast_ir
rustc_ast_lowering
rustc_ast_passes
rustc_ast_pretty
rustc_attr
rustc_baked_icu_data
rustc_borrowck
rustc_builtin_macros
rustc_codegen_llvm
rustc_codegen_ssa
rustc_const_eval
rustc_data_structures
rustc_driver
rustc_driver_impl
rustc_error_codes
rustc_error_messages
rustc_errors
rustc_expand
rustc_feature
rustc_fluent_macro
rustc_fs_util
rustc_graphviz
rustc_hir
rustc_hir_analysis
rustc_hir_pretty

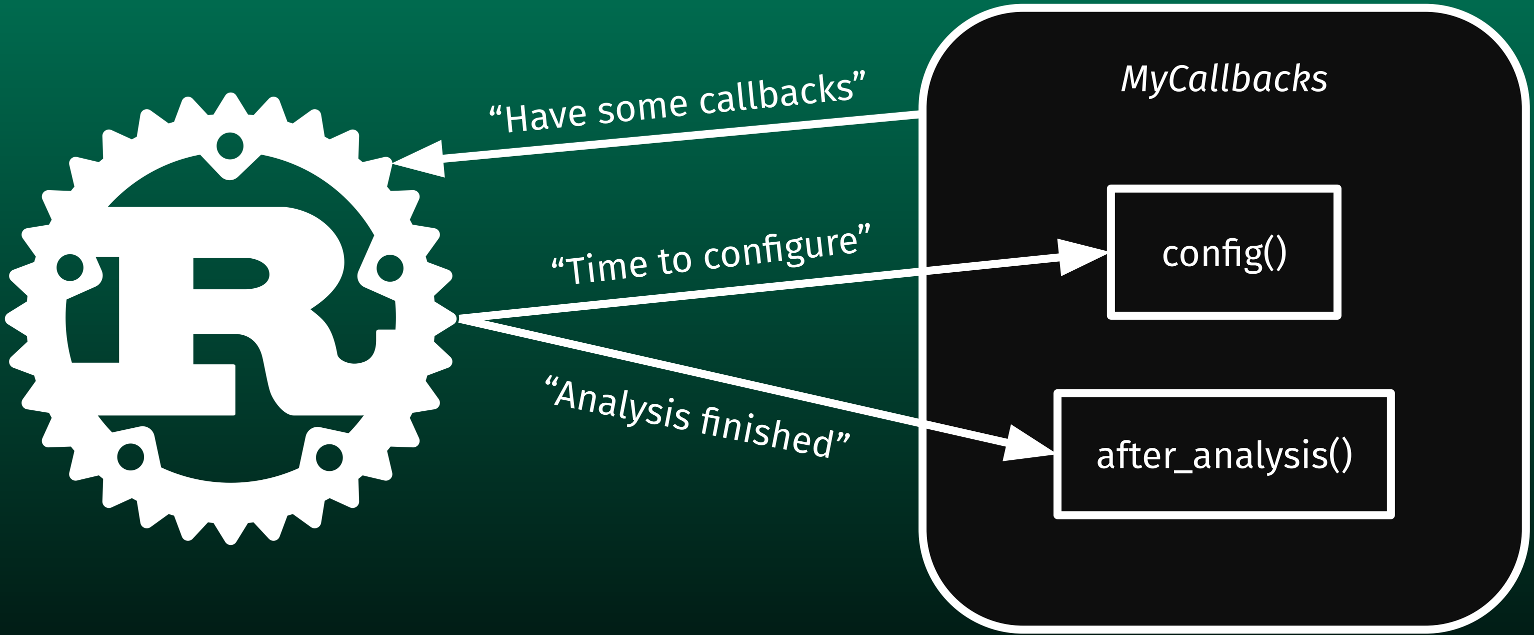

rustc_hir_typeck
rustc_incremental
rustc_index
rustc_index_macros
rustc_infer
rustc_interface
rustc_lexer
rustc_lint
rustc_lint_defs
rustc_llvm
rustc_log
rustc_macros
rustc_main
rustc_metadata
rustc_middle
rustc_mir_build
rustc_mir_dataflow
rustc_mir_transform
rustc_monomorphize
rustc_next_trait_solver
rustc_parse
rustc_parse_format
rustc_passes
rustc_pattern_analysis
rustc_privacy
rustc_query_impl
rustc_query_system
rustc_resolve
rustc_sanitizers
rustc_serialize
rustc_session
rustc_smir
rustc_span
rustc_symbol_mangling
rustc_target
rustc_trait_selection
rustc_traits
rustc_transmute
rustc_ty_utils
rustc_type_ir
rustc_type_ir_macros
rustdoc
rustdoc_json_types
rustfix
rustfmt
rustfmt_config_proc_macro
rustfmt_format_diff
rustfmt_nightly
stable_mir
tidy

rustc has a **lot** of crates
https://doc.rust-lang.org/nightly/nightly-rustc/

Callbacks

```rust
fn main() -> Result<(), ErrorGuaranteed> {
  let args: Vec<String> = std::env::args()
    .collect();

  RunCompiler::new(
    &args,
    &mut MyCallbacks,
  ).run()
}
```


MyCallbacks
"Have some callbacks"
"Time to configure"
"Analysis finished"
config()
after_analysis()

Callbacks

Callbacks

```rust
struct MyCallbacks;

impl Callbacks for MyCallbacks {
  fn config(&mut self, _: &mut Config) {
    println!("Configuring!");
  }

  fn after_analysis<'tcx>(
    &mut self, _: &Compiler, _: &'tcx Queries<'tcx>
  ) -> Compilation {
    println!("Analysis complete!");
    Compilation::Continue
  }
}
```

How to Register Lints

```rust
impl Callbacks for LinterCallbacks {
  fn config(&mut self, config: &mut Config) {
    config.register_lints =
      Some(Box::new(|session, lint_store| {
        lint_store.register_lints(&[MY_LINT]);
        lint_store.register_late_pass(|tcx| {
          Box::new(MyLintPass)
        });
      }));
  }
}
```

Lint and Lint Pass

Lint

- ➔ Unique identifier
- ➔ Can be *#[allow(...)]*’d
- ➔ Specifies default warning level
- ➔ Does nothing by itself

Lint Pass

- ➔ A series of functions that check the code
- ➔ Cannot be disabled
- ➔ Can emit multiple different lints

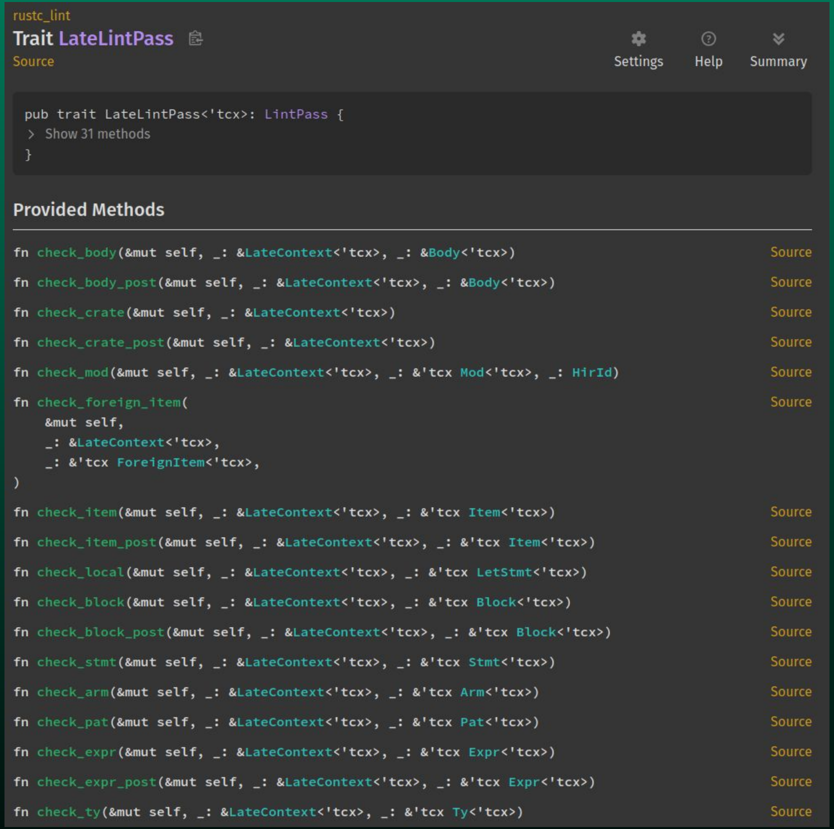
HIR vs. AST Data Formats

High-level Intermediate Representation (*LateLintPass*)

- ➔ Structure and meaning
- ➔ Resolves locations of paths
  - ◆ Knows that *std::iter::Iterator* exists

Abstract Syntax Tree (*EarlyLintPass*)

- ➔ Structure, but does not understand meaning
- ➔ Can check keywords and symbols, but not paths
  - ◆ Does not care if *std::iter::Iterator* exists or not



rustc_lint

Trait LateLintPass

Source

Settings Help Summary

```
pub trait LateLintPass<'tcx>: LintPass {
  > Show 31 methods
}
```

Provided Methods

```
fn check_body(&mut self, _: &LateContext<'tcx>, _: &Body<'tcx>)
fn check_body_post(&mut self, _: &LateContext<'tcx>, _: &Body<'tcx>)
fn check_crate(&mut self, _: &LateContext<'tcx>)
fn check_crate_post(&mut self, _: &LateContext<'tcx>)
fn check_mod(&mut self, _: &LateContext<'tcx>, _: &'tcx Mod<'tcx>, _: HirId)
fn check_foreign_item(
    &mut self,
    _: &LateContext<'tcx>,
    _: &'tcx ForeignItem<'tcx>,
)
fn check_item(&mut self, _: &LateContext<'tcx>, _: &'tcx Item<'tcx>)
fn check_item_post(&mut self, _: &LateContext<'tcx>, _: &'tcx Item<'tcx>)
fn check_local(&mut self, _: &LateContext<'tcx>, _: &'tcx LetStmt<'tcx>)
fn check_block(&mut self, _: &LateContext<'tcx>, _: &'tcx Block<'tcx>)
fn check_block_post(&mut self, _: &LateContext<'tcx>, _: &'tcx Block<'tcx>)
fn check_stmt(&mut self, _: &LateContext<'tcx>, _: &'tcx Stmt<'tcx>)
fn check_arm(&mut self, _: &LateContext<'tcx>, _: &'tcx Arm<'tcx>)
fn check_pat(&mut self, _: &LateContext<'tcx>, _: &'tcx Pat<'tcx>)
fn check_expr(&mut self, _: &LateContext<'tcx>, _: &'tcx Expr<'tcx>)
fn check_expr_post(&mut self, _: &LateContext<'tcx>, _: &'tcx Expr<'tcx>)
fn check_ty(&mut self, _: &LateContext<'tcx>, _: &'tcx Ty<'tcx>)
```

(each method line is followed by "Source" at right)

LateLintPass

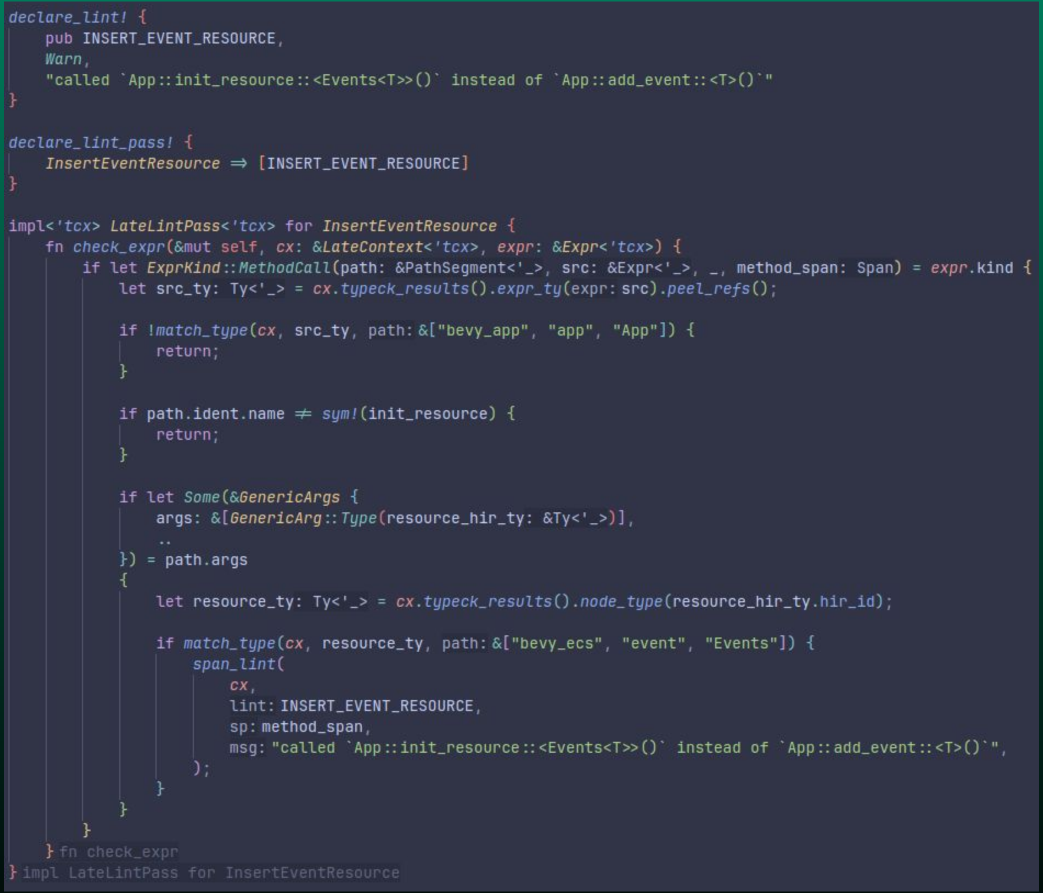
```rust
#[derive(Event)]
struct MyEvent;

App::new()
   // Incorrect ❌
   .init_resource::<Events<MyEvent>>()
   .run();
```

→ Method call of *App::init_resource()*
→ Generic argument to be *Events<T>*

Example Lint

- ➔ Lints and lint passes are declared with macros
- ➔ A lot of pattern matching and digging through types
- ➔ Optimized to return early
- ➔ Emitting lint doesn’t specify level



```rust
declare_lint! {
    pub INSERT_EVENT_RESOURCE,
    Warn,
    "called `App::init_resource::<Events<T>>()` instead of `App::add_event::<T>()`"
}

declare_lint_pass! {
    InsertEventResource ⇒ [INSERT_EVENT_RESOURCE]
}

impl<'tcx> LateLintPass<'tcx> for InsertEventResource {
    fn check_expr(&mut self, cx: &LateContext<'tcx>, expr: &Expr<'tcx>) {
        if let ExprKind::MethodCall(path: &PathSegment<'_>, src: &Expr<'_>, _, method_span: Span) = expr.kind {
            let src_ty: Ty<'_> = cx.typeck_results().expr_ty(expr: src).peel_refs();

            if !match_type(cx, src_ty, path: &["bevy_app", "app", "App"]) {
                return;
            }

            if path.ident.name ≠ sym!(init_resource) {
                return;
            }

            if let Some(&GenericArgs {
                args: &[GenericArg::Type(resource_hir_ty: &Ty<'_>)],
                ..
            }) = path.args
            {
                let resource_ty: Ty<'_> = cx.typeck_results().node_type(resource_hir_ty.hir_id);

                if match_type(cx, resource_ty, path: &["bevy_ecs", "event", "Events"]) {
                    span_lint(
                        cx,
                        lint: INSERT_EVENT_RESOURCE,
                        sp: method_span,
                        msg: "called `App::init_resource::<Events<T>>()` instead of `App::add_event::<T>()`",
                    );
                }
            }
        }
    } fn check_expr
} impl LateLintPass for InsertEventResource
```

bevy_lint's Lints

- *insert_event_resource*
  - Checks for the *Events<T>* resource being inserted with *App::init_resource()*.
- *main_return_without_appexit*
  - *App::run()* returns an *AppExit* that specifies whether it crashed or not.
- *missing_reflect*
  - Require components and resources to derive *Reflect*
- *panicking_methods*
  - Ban *Query* and *World* methods that can panic when a better alternative exists.
- *plugin_not_ending_in_plugin*
  - Require all plugins have the “Plugin” suffix
- *zst_query*
  - Recommend *Query<(), With<ZST>>* instead of *Query<ZST>*.

Conclusion

- ➔ Please try out *bevy_lint*!
  - ◆ https://thebevyflock.github.io/bevy_cli/bevy_lint
- ➔ Consider contributing if you’re interested :)
  - ◆ *bevy_cli* working group on Discord
  - ◆ Code is heavily documented, 2:1 code to comment ratio