

Fortitude

Liam Pattinson, Peter Hill

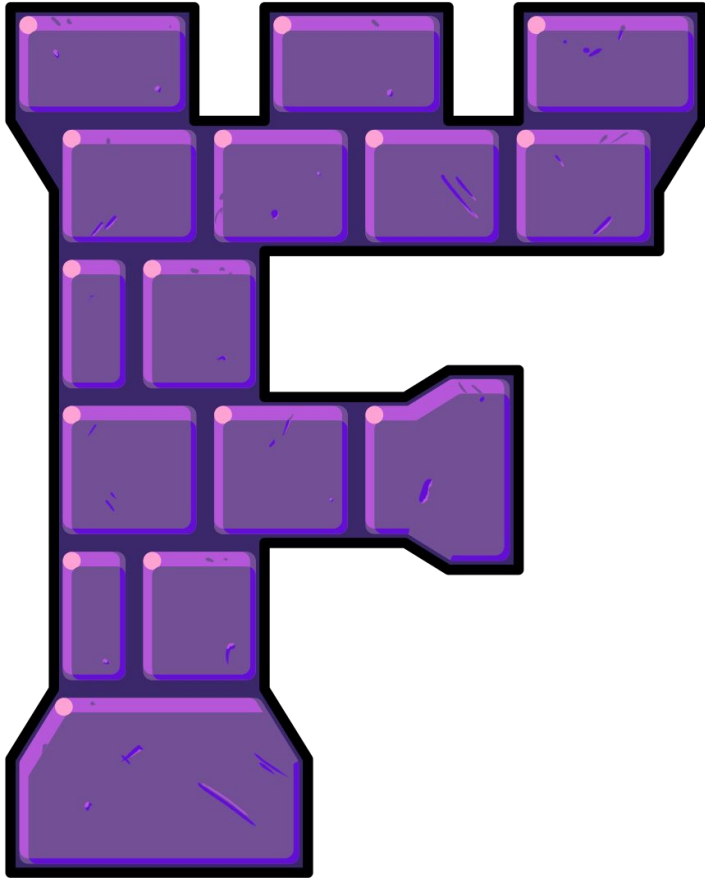
A fast and modern and **Fortran linter**, for automated **bug checking**,
modernisation and **style enforcement**.



Fortran Linters

- Linters are static analysis tools that warn of bug-prone code, suggest refactors/modernisations, enforce code style, etc.
- Standard part of a modern developer's toolkit for other languages:
 - Clang-tidy (C++)
 - Ruff/Pylint/flake8/etc... (Python)
 - Clippy (Rust)
 - ESLint (JavaScript)
 - Etc...
- Fortran lacks a 'go-to' Free and Open-Source Software (FOSS) solution.
- Existing (free) linters are unmaintained, hard to install/use, and/or missing features.

Fortitude



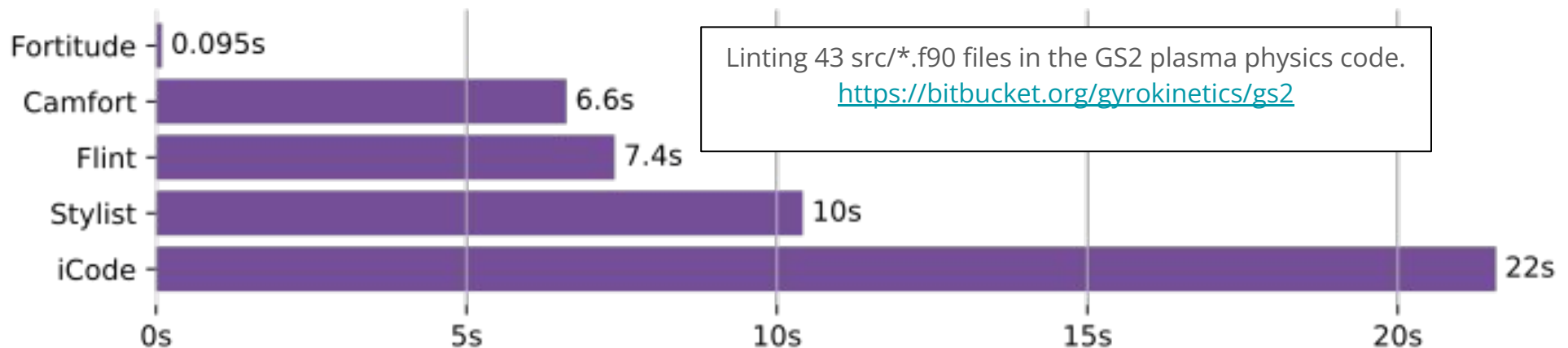
- Command-line tool, automates bug finding, modernisation, and style enforcement.
- Goes beyond compiler warnings: deters Fortran anti-patterns and deprecated/non-portable features.
- Where possible, conforms to community best-practices.
- Cross-platform, multiple installation methods.
- Easy to use out-of-the-box.



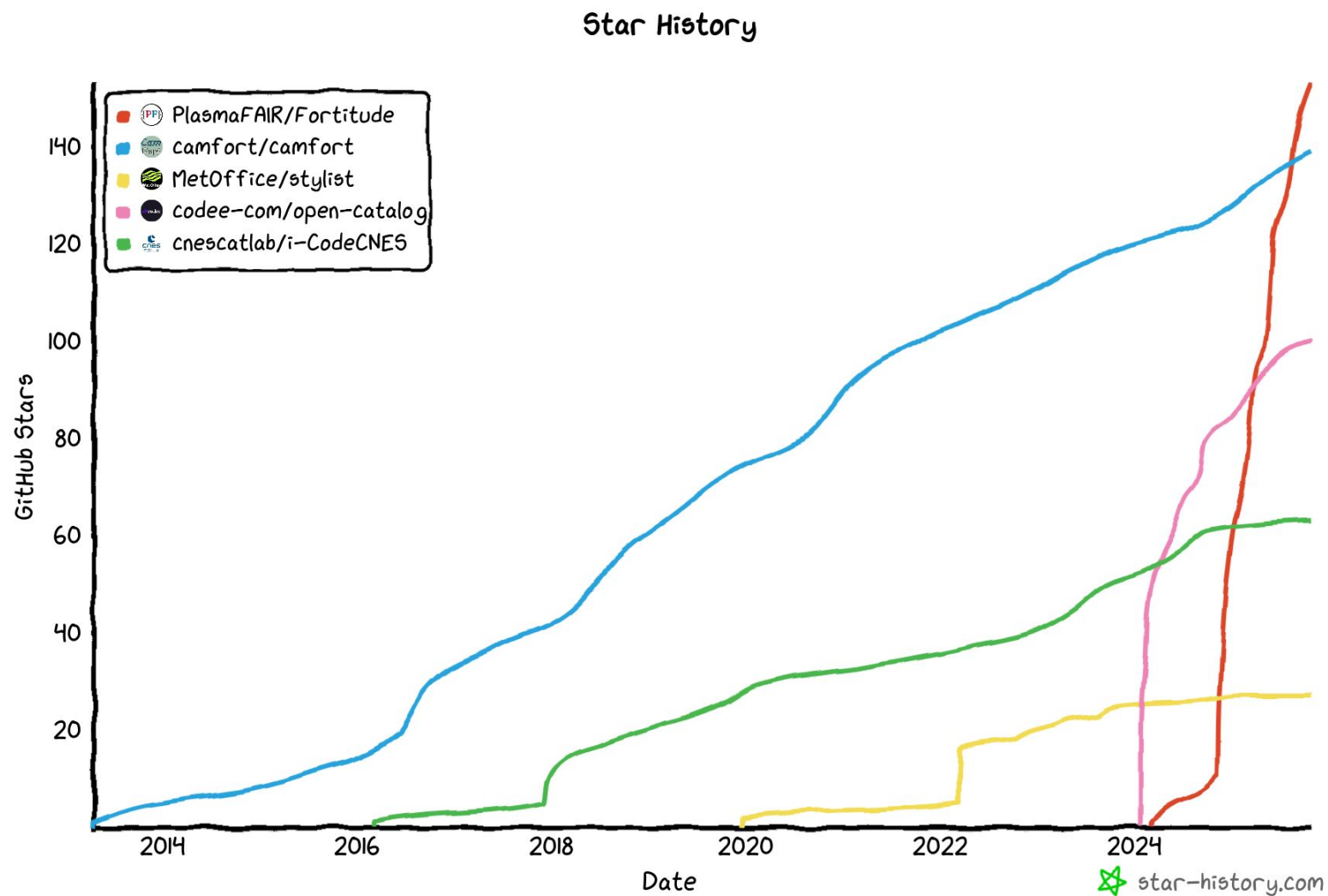
github.com/PlasmaFAIR/Fortitude

Key Features

- Rich command line interface, closely matches Ruff.
- Large and growing number of linter rules (75 currently, 100+ planned).
- Applies fixes automatically.
- Robust parsing (though pre-processor creates issues).
- Configurable via settings files `fortitude.toml` or `fpm.toml`.
- Very fast



User Growth



Getting Started

- Installation using `pip` or using an installer script:

```
# Via pip, all platforms:
pip install fortitude-lint

# On macOS and Linux:
curl -LsSf \
  https://github.com/PlasmaFAIR/fortitude/releases/latest/download/fortitude-installer.sh | sh

# On Windows:
powershell -c `
"irm https://github.com/PlasmaFAIR/fortitude/releases/latest/download/fortitude-installer.psi | iex"
```

- Can check Fortran files out-of-the-box with a sensible* default ruleset:

```
cd my_fortran_project/
fortitude check          # Automatically find files
fortitude check src/    # Check a specific directory or specific files
```

** Based on our own observations of common Fortran best practices – this will still be opinionated!*

Getting Started

```
epoch3d/src/user_interaction/particle_temperature.F90:22:3: C003 'implicit none' missing 'external'
20 | USE evaluator
21 |
22 | IMPLICIT NONE
   | ^^^^^^^^^^^^^ C003
23 |
24 | REAL(num), PARAMETER, PRIVATE :: max_average_its = 20.0_num
   |
   | = help: Add `(external)` to 'implicit none'

epoch3d/src/user_interaction/particle_temperature.F90:35:40: C061 subroutine argument 'part_species' missing 'intent' attribute
33 | REAL(num), DIMENSION(1-ng::,1-ng::,1-ng::), INTENT(IN) :: temperature
34 | INTEGER, INTENT(IN) :: direction
35 | TYPE(particle_species), POINTER :: part_species
   | ^^^^^^^^^^^^^^^^^ C061
36 | REAL(num), DIMENSION(1-ng::,1-ng::,1-ng::), INTENT(IN) :: drift
37 | TYPE(particle_list), POINTER :: partlist

fortitude: 79 files scanned.
Number of errors: 744

For more information about specific rules, run:

    fortitude explain X001,Y002,...

[*] 214 fixable with the `--fix` option (94 hidden fixes can be enabled with the `--unsafe-fixes` option).
```

Getting More Info

- What does **C003**: `'implicit none' missing 'external'` mean?
- The `explain` command prints rule documentation on the command line.

```
> fortitude explain C003
# C003: implicit-external-procedures

## What it does
Checks if `implicit none` is missing `external`

## Why is this bad?
`implicit none` disables implicit types of variables but still allows
implicit interfaces for procedures. Fortran 2018 added the ability to also
forbid implicit interfaces through `implicit none (external)`, enabling the
compiler to check the number and type of arguments and return values.

`implicit none` is equivalent to `implicit none (type)`, so the full
statement should be `implicit none (type, external)`.
```

Rule Filtering

- Rules can be switched on with `--select` and switched off with `--ignore`:

```
# Using rule/category codes
fortitude check --select=C --ignore=C003

# Using full names
fortitude check --select=correctness --ignore=implicit-external-procedures
```

- Rules can also be switched off on a line-by-line basis using `allow` comments:

```
module my_mod

    ! allow(C003)
    implicit none

    ...

end module my_mod
```

Rules

F

Rules

Search

fortune

v0.7.5

153

16

Fortitude

Overview

Tutorial

Installing Fortitude

The Fortitude Linter

Configuring Fortitude

Preview

Rules

Legend

Error (E)

Correctness (C)

Obsolescent (OB)

Modernisation (MOD)

Style (S)

Portability (PORT)

Fortitude (FORT)

Settings

Integrations

FAQ

Changelog

Modernisation (MOD)

Back to top

Code	Name	Message	
MOD001	double-precision	Use of '{original}' is discouraged	✓ ✗
MOD002	double-precision-literal	Use of 'd' exponentiation in '{original}' is discouraged	✍ ✗
MOD011	old-style-array-literal	Array literal uses old-style syntax: prefer [...]	✓ ✗ ▶
MOD021	deprecated-relational-operator	deprecated relational operator '{symbol}', prefer '{new_symbol}' instead	✓ ✗ ▶
MOD031	include-statement	Include statement is deprecated, use modules instead	✍ ✗
MOD201	old-mpi-module	Use of mpi module discouraged, use mpi_f08 instead	✍ ✗

Style (S)

Code	Name	Message	
S001	line-too-long	line length of {actual_length}, exceeds maximum {max_length}	✓ ✗ ▶
S061	unnamed-end-statement	end statement should be named.	✓ ✗ ▶
S071	missing-double-colon	variable declaration missing '::'	✓ ✗ ▶

Fixes

- Fortitude can resolve many issues automatically.

```
# Correct any 'safe' fixes
fortitude check --fix

# Also correct 'unsafe' fixes: use with caution!
fortitude check --select=C003 --fix --unsafe-fixes
```

- 'Safe' fixes will typically correct style issues without changing the meaning of the code, e.g. replace `(\...\)` with `[...]`.
- 'Unsafe' fixes might change code behaviour. Best applied rule-by-rule and with strong testing!

Configuration

- Project-wide settings can be set via a `fortitude.toml` or `fpm.toml` file at the top-level of your project.

```
# fortitude.toml

[check]
extend-exclude = ["examples/", "test.f90"]           # Don't lint these files/directories
select = ["error", "correctness", "modernisation", "style"] # Include these categories
ignore = ["superfluous-implicit-none", "missing-intent"] # ...but ignore these rules
preview = true                                         # Use latest (and unstable!) features
line-length = 120                                     # Use longer max line length

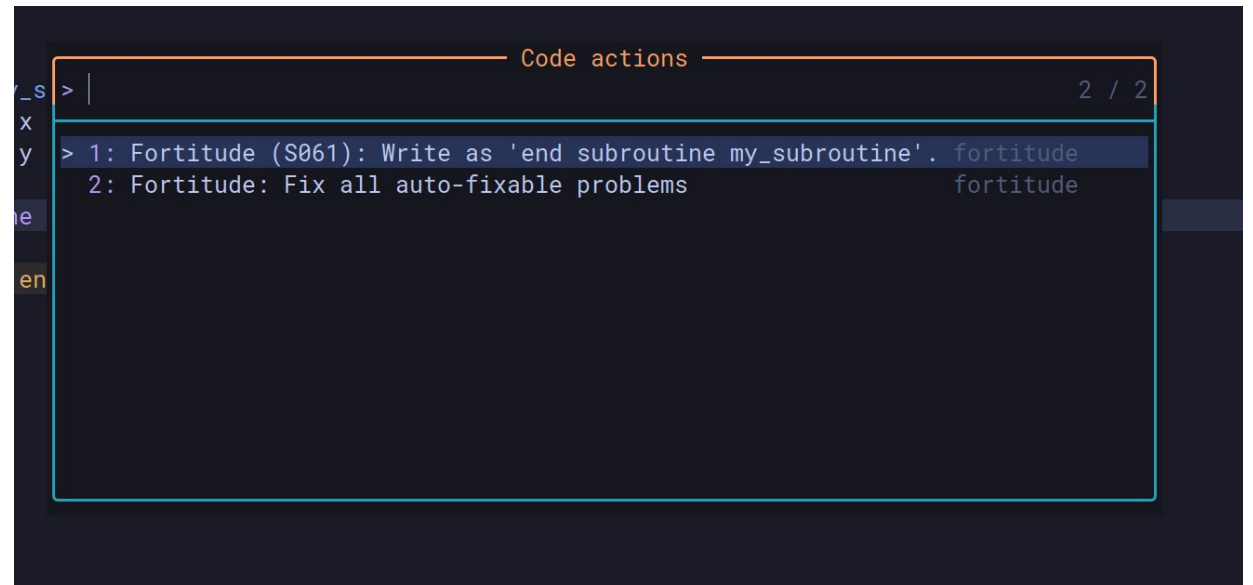
[check.per-file-ignores]
"**/*_lib.f90" = ["default-public-accessibility"]    # Ignore rule for certain file patterns
```

- Allows consistent behaviour when coding on collaborative projects and when using Fortitude in CI/CD.

Upcoming

- Language Server Protocol (LSP) integration for in-editor reporting and fixes.
- Designed to work alongside FortLS.
- Supported by (Neo)Vim, Emacs, Helix, Kate, VSCode (via external plugin).

```
1 module my_mod      ■ module 'my_mod' missing default accessibility statement
2
3 implicit none      ■ 'implicit none' missing 'external'
4
5 contains
6
7 subroutine my_subroutine(x, y)
8   integer :: x      ■ subroutine argument 'x' missing 'intent' attribute
9   integer :: y      ■ subroutine argument 'y' missing 'intent' attribute
10  y = x + 1;        ■ unnecessary semicolon
11 end subroutine      ■ end statement should be named.
12
13 end module          ■ end statement should be named.
14
```



Roadmap

FORMATTING

- Automatically format code.
- Correct whitespace, capitalisation, line breaks, etc.
- Similar to fprettify, clang-format, and black.
- Will compliment the linter, avoid any clashes.

PREPROCESSOR

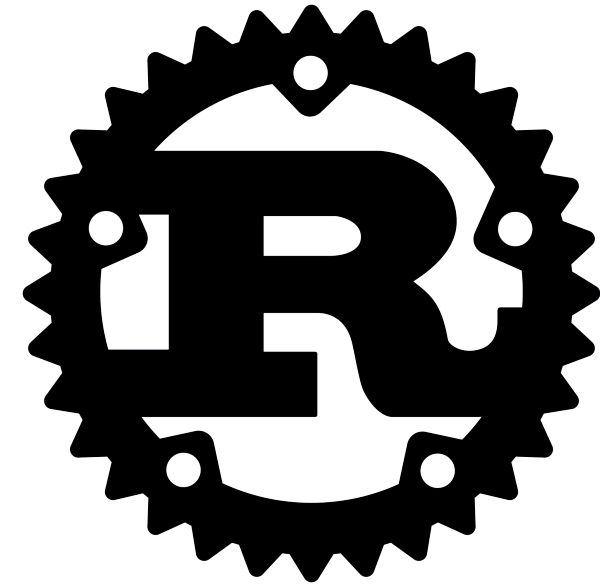
- Currently can't handle many uses of the C-preprocessor using TreeSitter.
- Aiming to preprocess files ourselves and map all linter warnings back to the original code.
- Will require some user set-up.
- Can expand to other macro languages (fypp, pFUnit).

SEMANTIC INFORMATION

- Currently scanning the concrete syntax tree (CST) directly.
- Aiming to add higher layer of abstraction, make more information available while linting.
- Will enable many new linter rules.

Technology

- Implemented in Rust: very fast, safer than C++/Python/etc, and great libraries for command line tooling.
- Draws heavily from the Python linter Ruff, both in inspiration and direct code reuse.
- Uses TreeSitter to parse Fortran. Contributed to development of tree-sitter-fortran:
<https://github.com/stadelmanma/tree-sitter-fortran>



CONTACT US



`liam.pattinson@york.ac.uk`



`peter.hill@york.ac.uk`



`github.com/PlasmaFAIR/fortitude`