Engineer

リンター

🖋 コードを実行せず、間違いらしい箇所を指摘するツール

どのようなものか

リンターは、プログラムをビルドしたり実行したりせず、ソースコードを読んで、間違いと思われる箇所を報告するプログラムです。各指摘に、ルール名、ファイル名、行番号を示します。値を代入したまま使わない変数、常に真になる比較、引用符で囲んでいないシェル変数、利用者の入力から組み立てる書式文字列などです。

名前の由来は、Cプログラムの検査器lintです。NetBSDは、今もその一つを提供します。マニュアルは、Cのプログラムファイルに見られる「バグ、移植性の欠如、資源の無駄につながりそうな特徴」を検出するものと説明し、従来のC90以前のコンパイラーより厳格な型検査を行うと付け加えます。二段階で動き、最初はファイルを個別に検査し、その後、結果を突き合わせて、ファイル間の不整合を探します。

今では、大半の言語に独自のリンターがあり、複数ある場合もよくあります。

- JavaScriptには**ESLint**。
- Pythonには**Ruff**。
- シェルスクリプトには**ShellCheck**。掲げる目的は、初心者の構文上の問題から、今は動くスクリプトを将来の条件下で失敗させ得る微妙な注意点まで及びます。
- CSSには**Stylelint**、Markdownには**markdownlint**、YAMLには**yamllint**。

どのリンターも、ルールからできています。各ルールには、RuffのF841やShellCheckのSC2059のような識別子と、重大度があります。設定ファイルで、有効にするルールと重大度を決めます。自動修正を備えたルールも多くあります。

フォーマッターは、配置を書き直し、意味は判断しません。型検査器は、それぞれの値が何であるかを推論します。リンターはその中間にあり、言語の誤用、危険な書き方、プロジェクトが合意した慣習など、コードの書き方を調べます。

概念を判断に生かす

「リンターとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 3社の間で受け渡す代わりに、1つのチームがアイデアのスケッチから、お客様がログインして使える製品まで担当します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GIS地図アプリの開発を主導し売上を10倍に — 10x 地図製品の開発を主導して実現した売上成長

GIS地図アプリの開発を主導し売上を10倍に お問い合わせ →

なぜ重要なのか

リンターは、年に一回しか動かず、テストも届かないエラーハンドラーを含め、すべての分岐を読みます。リポジトリー全体でも数秒なので、保存やコミットのたびに実行できます。指摘の修正費用が最も小さいのは、その行を書いた時点です。

ルールは、議論を一度で決着させます。例外を指定しない `except:` を認めるか、すべてのシェル変数を引用符で囲むかは、設定で決め、どの作成者やレビューにも同じように適用します。人のレビューでは、ルールにできない問いに注意を向けられます。

終了コードは、リンターと、それを動かすものの間の接点です。ShellCheckのREADMEは、「終了コードを定石どおりに使う」と述べ、ビルドから、ほかのコマンドと同じように呼べます。RuffとESLintも、同じ三種類を文書化しています。問題がなければ0、違反があれば1、設定が不正な場合など、ツール自身が失敗すれば2です。品質ゲートは、これで「コードが間違っている」と「検査器が壊れている」を区別でき、どちらでも処理を止めるべきです。

どの指摘が終了コードに影響するかは、重大度が決めます。ESLintは、エラーがなく警告だけなら、 `--max-warnings` で上限を設けない限り、成功で終了します。lint(1)には、警告をエラーとして扱う `-w` があります。間違いを説明するだけでなく、止めるために使うなら、その選択を、すべての実行で使う場所に記述します。

```
# 警告を含むすべての指摘で実行を失敗させる
lint -w -abcFghprSxz source.c
eslint --max-warnings 0 src/
ruff check src/
shellcheck scripts/*.sh
```

このサイトは、厳しい方を選びます。自身の品質ゲートは、すべての診断をエラーとし、`# noqa` のような行内の抑制を禁じます。指摘は、その原因を直します。

実際の活用例

リンターの設定は、導入した日に、たまたま何も指摘しなかったルールから始まりがちです。それを逆にしたページがあります。Pythonリンターの全ルールをエラーとして有効にした仕事です。最初に1,815件の指摘があり、ファイル別でなく、分類ごとに処理しました。分類には意味があるからです。使わない引数や、組み込みの名前を隠すものは雑音でしたが、セキュリティー、変更可能な既定値、例外処理の分類は、それぞれ実際の動作を指しました。少数の例外には、その行に理由を記し、うち二つには、無効にしたルールが守るはずの性質を確認するテストもあります。厳格な型検査も並行して行い、書き方ではなく、値が何であるかに関する別の不具合を見つけました。費用も明示します。誰かが、1,815回判断しなければなりません。

別のページは、困った発見から始まります。三十二本のスクリプト、10,242行のJavaScriptからなるサイトの品質ゲートが、リポジトリー最大のコードでありながら、フォーマッター、リンター、型検査器のどれも検査していなかった唯一の部分でした。品質ゲート自身のコードにリンターを導入する作業は、フォーマッターから始めました。そのファイルではインデントをスペース二つ分に設定し、変更を2,173行でなく38行に抑えました。次にリンターが十三件の実際の問題を見つけ、すべて修正し、抑制は一件も使いませんでした。型検査器は、リンターを通った二つの不具合を見つけました。その手間をかける理由も述べています。検査器の不具合は、本来不合格になるチェックが合格する形で表れるからです。

同じ考えは、文章にも使えます。三つ目のページは、助言だけでなく、不合格にする上限を備えた、リポジトリー自身の文書用リンターを説明します。八つの機械的な規則には、減らすことしかできないファイルごとの行数予算、計画の状態とチェックボックスの一致、文書中のすべてのパスやコードのシンボルが、リポジトリーのツリーに実在するかの検査が含まれます。初回には、未チェックの項

目を残して完了扱いだった六つの計画、毎コミット必須なのに文書にない十四のタスク、三つの壊れたリンク、一つの孤立文書、三つの使えないパスが見つかりました。

問題が起きるところ

リンターの失敗は、目立つとは限りません。よくあるのは、止めるべきものを止めず、0で終了する実行です。

指摘が終了コードを変えません。 ESLintの `--max-warnings` は、既定値が-1で、上限を無効にします。lint(1)のマニュアルは、ファイル間の検査が、常に成功で終了すると述べます。読み手は、実行を失敗させない指摘を一度読んでも、その後はスクロールして通り過ぎます。ルールごとに、実行を失敗させるエラーにするか、無効にするかを決め、警告をエラーとして扱うオプションは、共通の設定に置きます。

修正した結果を成功と報告します。 Ruffは、見つけた違反をすべて自動修正すれば、0で終了します。手元では便利ですが、修正付きの品質ゲートが直すのは、その場の作業コピーだけです。届いたコミットには、違反がすべて残っています。ゲートは修正なしで実行するか、 `--exit-non-zero-on-fix` を加え、修正も失敗として数えます。 `--exit-zero` も探します。これがあれば、違反があってもRuffは0で終了し、雑音の多い一回の移行のために足したフラグが、残りがちです。

対象のファイルがゼロです。 ディレクトリー名の変更や設定の移動で、リンターの対象が空になることがあります。ファイルをゼロ件調べた結果は、問題なしになります。ESLintは、引用符で囲んだファイル名のパターン（glob）が何にも一致しなければ、既定でエラーとします。 `--no-error-on-unmatched-pattern` は、何も知らせず合格に変え、 `--pass-on-no-patterns` は、ファイルパターンが指定されなければ、0で終了します。既定の動作を保ち、ゲートが必ず報告するべき既知の違反を持つファイルを、一つ残します。

何も指摘しないことを基準にルールを選びます。 初日に反応したものをすべて無効にした設定は、基準ではなく、コードの癖を記録します。全ルールを一度に使うか、Ruffが提案する `E` と `F` の小さい組み合わせからルール群を一つずつ足すか、意図を持って選び、無効にした各ルールの理由を記述します。Ruffは、 `ALL` を使うと、更新時に新ルールも暗黙に有効になると注意します。そのため、リンターの版を固定し、その更新を独立した変更として扱います。

抑制が、一行ずつ広がります。 どのリンターにも逃げ道があります。lint(1)の `/* LINTED */`、Ruffの `# noqa: F841`、ShellCheckの `# shellcheck disable=SC2059`、ESLintの `// eslint-disable-next-line` です。記録するのは、何を黙らせたかで、理由ではありません。各例外の横には理由を求め、重要なものはルールを無効にして、守っていた性質のテストを加えます。一か所に集めることも役立ちます。ESLintの `--no-inline-config` は、無効化コメントを効かなくし、ShellCheckは、プロジェクト全体の `disable` 行を `.shellcheckrc` から読みます。

参考資料

- lint(1) — 言葉の由来になったC検査器の、NetBSDによるマニュアルです。二段階の検査、`-w`、抑制コメントを説明します。
- Ruffリンター — ルール選択、修正の安全性、抑制コメント、終了コードです。
- コマンドラインインターフェースの参照資料 — ESLintの `--max-warnings`、行内設定、一致しないパターン、終了コードです。
- ShellCheck — プロジェクトのREADMEです。検出するものと、ビルドへの組み込み方を説明します。
- 無視する方法 — 行単位、実行単位、プロジェクト単位で、ShellCheckの指摘を無視する方法です。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「リンターとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/linter/