**Engineer**

# Лінтер

🚀 інструмент, що без запуску шукає підозрілий код

## Що це таке

Лінтер — це програма, яка читає вихідний код і вказує на місця, схожі на помилки, не збираючи й не запускаючи програму. Кожне зауваження називає правило, файл і рядок: змінну, якій присвоїли значення й ніде не використали, порівняння, що завжди істинне, змінну оболонки без лапок, рядок формату, зібраний із введення користувача.

Назва походить від lint — засобу перевірки програм мовою C, і NetBSD досі постачає таку утиліту. Її довідка описує завдання як пошук у файлах програм на C тих особливостей, які ймовірно є помилками, непереносні або марнотратні, і додає, що типи вона перевіряє суворіше, ніж традиційні компілятори C до C90. Працює вона у два проходи: спершу кожен файл перевіряється як окрема одиниця, потім результати звіряються між файлами в пошуках невідповідностей.

Тепер більшість мов мають власні лінтери, часто кілька:

- **ESLint** для JavaScript.
- **Ruff** для Python.
- **ShellCheck** для скриптів оболонки. Його заявлені цілі сягають від синтаксичних помилок початківців до тонких пасток, через які робочий скрипт може зламатися за майбутніх обставин.
- **Stylelint** для CSS, **markdownlint** для Markdown і **yamllint** для YAML.

Будь-який лінтер складається з правил. Кожне правило має ідентифікатор, наприклад F841 у Ruff чи SC2059 у ShellCheck, і рівень серйозності. Файл конфігурації визначає, які правила ввімкнено і наскільки серйозне кожне з них, а багато правил мають автоматичне виправлення.

Форматувальник переписує оформлення коду й не має думки щодо змісту; засіб перевірки типів міркує про те, чим є кожне значення. Лінтер стоїть між ними й дивиться, як код написано: на зловживання можливостями мови, ризиковані шаблони й угоди, про які домовилися в проєкті.

Застосуйте поняття до рішення

Застосуйте «Що таке лінтер?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Одна команда веде ідею від ескізу до продукту, у який ваші клієнти вже можуть увійти, а не перекидає роботу між трьома підрядниками.

**Переглянути наші послуги** Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Збільшила дохід GIS-застосунку в 10 разів — 10x зростання виторгу завдяки картографічному продукту

**Збільшила дохід GIS-застосунку в 10 разів** Зв'язатися з нами →

## Чому це важливо

Лінтер читає кожну гілку коду, зокрема обробник помилок, що спрацьовує раз на рік і до якого не дістається жоден тест. На весь репозиторій йому вистачає секунд, тож його можна запускати під час кожного збереження й кожного коміту, а зауваження найдешевше тоді, коли рядок тільки пишуть.

Правила розв’язують суперечку один раз. Чи прийнятний голий `except:`, чи кожну змінну оболонки треба брати в лапки — це вирішують у конфігурації, а далі застосовують однаково до кожного автора й кожного рев’ю. Рецензенти-люди зберігають увагу для питань, які правило виразити не здатне.

Код виходу — це інтерфейс між лінтером і тим, що його запускає. README ShellCheck каже, що інструмент використовує коди виходу канонічним способом, тож збірка може викликати його як ще одну команду. Ruff і ESLint документують ту саму триступеневу угоду: 0, коли перевірка чиста, 1, коли знайдено порушення, і 2, коли збій стався в самому інструменті, наприклад через хибну конфігурацію. Тож ворота якості можуть відрізнити «код неправильний» від «перевірка зламалася», і обидва випадки мають їх зупинити.

Рівень серйозності визначає, які зауваження впливають на код виходу. ESLint завершується успішно, коли знаходить лише попередження, якщо `--max-warnings` не задає межу, а в lint(1) є `-w`, щоб вважати попередження

помилками. Проєкт, який хоче, щоб лінтери блокували помилки, а не лише описували їх, записує цей вибір туди, де його використовує кожен запуск:

```
# кожне зауваження валить запуск, попередження теж
lint -w -abcFghprSxz source.c
eslint --max-warnings 0 src/
ruff check src/
shellcheck scripts/*.sh
```

Цей сайт обрав найсуворіший варіант. Його власні ворота якості вважають кожну діагностику помилкою й забороняють вбудовані придушення на кшталт `# noqa`, тож зауваження виправляють у його причині.

## Як це виглядає на практиці

Конфігурація лінтера часто починається з тих правил, які випадково мовчали того дня, коли її налаштовували. Одна сторінка робить навпаки: усі правила лінтера Python увімкнено як помилки. Перший прогін дав 1 815 зауважень. Їх опрацьовували за категоріями, а не за файлами, бо категорії самі дещо кажуть: невикористані аргументи й затінені вбудовані імена виявилися шумом, а категорії безпеки, змінюваних значень за замовчуванням і обробки винятків указували на реальну поведінку. Нечисленні винятки мають письмове обґрунтування просто в рядку, а два з них підкріплені тестом, який перевіряє ту властивість, що її забезпечувало б вимкнене правило. Поряд у суворому режимі працювала перевірка типів і ловила інший різновид дефектів: про те, чим є значення, а не як його записано. Названо й ціну: комусь доводиться ухвалювати рішення 1 815 разів.

Інша сторінка починається з незручного відкриття. Ворота якості сайту, тридцять два скрипти й 10 242 рядки JavaScript, були найбільшим масивом коду в репозиторії і єдиним, який не читали ні форматувальник, ні лінтер, ні засіб перевірки типів. Щоб підключити код самих воріт до лінтера, почали з форматувальника й налаштували для цих файлів відступ у два пробіли, тож змінилися 38 рядків замість 2 173. Далі лінтер дав тринадцять реальних зауважень, усі виправлені й жодне не придушене, а засіб перевірки типів знайшов два дефекти, які лінтер пропустив. Пояснення, навіщо це, теж є: дефект у перевірці проявляється як перевірка, що проходить тоді, коли мала б упасти.

Ідея працює й для тексту. Третя сторінка описує лінтер для власної документації репозиторію з межами, які спричиняють збій, а не лише дають пораду. Серед його восьми механічних політик — бюджет рядків для кожного файлу, який може лише зменшуватися, правило, що заявлений статус плану має збігатися з його власними прапорцями, і перевірка, що кожен шлях і символ коду, згаданий у документі, існує в дереві. Перший запуск виявив шість планів, позначених виконаними, з невідміченими прапорцями, чотирнадцять завдань, що перевіряли кожен коміт, але не згадувалися в жодному документі, три биті посилання, один документ-сироту й три мертві шляхи.

## Де виникають проблеми

Лінтер рідко ламається гучно. Звичайна вада — запуск, що завершується з кодом 0, хоча мав би щось зупинити.

**Зауваження, які ніколи не змінюють код виходу.** `--max-warnings` в ESLint за замовчуванням дорівнює -1, тобто межу вимкнено, а за довідкою lint(1) прохід перехресної перевірки між файлами завжди завершується успішно. Зауваження, яке не може провалити запуск, читають один раз, а потім прокручують. Вирішуйте для кожного правила: або помилка, що валить запуск, або вимкнено, а будь-який перемикач «попередження як помилки» тримайте в спільній конфігурації.

**Виправлення, яке звітує про успіх.** Ruff завершується з 0, коли кожне знайдене порушення виправлено автоматично. За робочим столом це зручно, але ворота, що працюють з увімкненими виправленнями, лагодять лише власну робочу копію, а коміт, який надійшов, і далі містить кожне порушення. Запускайте перевірку без виправлень або додайте `--exit-non-zero-on-fix`, щоб виправлення вважалося збоєм. Пошукайте також `--exit-zero`: з ним Ruff завершується з 0, навіть коли порушення є, а прапорець, доданий на час одного переходу з купою порушень, має звичку залишатися.

**Шаблон, що охоплює нуль файлів.** Перейменований каталог чи перенесена конфігурація можуть залишити лінтер націленим на порожню множину, а перевірка нуля файлів — чистий результат. ESLint за замовчуванням вважає помилкою glob у лапках, під який не потрапляє жоден файл. `--no-error-on-unmatched-pattern` перетворює це на тихий успіх, а `--pass-on-no-patterns`

завершується з 0, коли шаблонів файлів не передано зовсім. Залиште поведінку за замовчуванням і тримайте один файл із відомим порушенням, про яке ворота мусять повідомити.

**Набір правил, обраний за тим, що мовчало.** Конфігурація, зібрана вимиканням усього, що спрацювало першого дня, фіксує звички коду, а не стандарт. Почніть зі свідомого вибору: або все одразу, або запропонована Ruff невелика база з `E` і `F`, до якої щоразу додають по одній групі, — і запишіть, чому вимкнено кожне вимкнене правило. Ruff попереджає, що `ALL` неявно вмикає нові правила після кожного оновлення інструмента, тож зафіксуйте версію лінтера й оновлюйте її окремою зміною.

**Придушення поширюються по одному рядку.** Кожен лінтер має запасний вихід: `/* LINTED */` у lint(1), `# noqa: F841` у Ruff, `# shellcheck disable=SC2059` у ShellCheck і `// eslint-disable-next-line` в ESLint. Кожен із них фіксує, що саме заглушили, але не чому. Вимагайте письмового обґрунтування біля кожного винятку, а для важливих вимкніть правило й додайте тест на властивість, яку воно захищало. Допомагає й тримати винятки в одному місці: `--no-inline-config` в ESLint робить коментарі вимкнення недієвими, а ShellCheck читає рядки `disable` для всього проєкту з файлу `.shellcheckrc`.

## Що почитати далі

- lint(1) — довідка NetBSD до засобу перевірки C, від якого походить слово: два проходи, `-w` і коментарі для придушення.
- The Ruff Linter — вибір правил, безпечні й небезпечні виправлення, коментарі придушення й коди виходу.
- Command Line Interface Reference — `--max-warnings` в ESLint, вбудована конфігурація, шаблони без збігів і коди виходу.
- ShellCheck — README проєкту: що інструмент шукає і як він вписується у збірку.
- Ignore — способи ShellCheck ігнорувати зауваження для рядка, запуску чи цілого проєкту.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

**Дивитися, скільки це коштує** Зв'язатися з нами →

# Зробіть термінологію практичною

Запитайте, як «Що таке лінтер?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

**Переглянути наші послуги**

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

**Дбайливо зроблено з любов'ю — D.**

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

**Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.**

**Надруковано з https://engineer.company/uk/glossary/linter/**