Engineer

Linter

🖋 Detector de posibles errores de código sin ejecutarlo

Qué es

Un linter es un programa que lee código fuente y señala las partes que parecen errores, sin compilar el programa ni ejecutarlo. Cada aviso identifica una regla, un archivo y una línea: una variable a la que se asigna un valor y nunca se usa, una comparación que siempre es verdadera, una variable de shell sin comillas o una cadena de formato construida con datos del usuario.

El nombre procede de lint, un comprobador de programas en C que NetBSD sigue incluyendo. Su manual describe la tarea como detectar características de los archivos de programas en C «que probablemente sean errores, no sean portables o desperdicien recursos» y añade que realiza una comprobación de tipos más estricta que los compiladores tradicionales anteriores a C90. Trabaja en dos pasadas: primero comprueba cada archivo como una unidad independiente y después contrasta los resultados para detectar incoherencias entre archivos.

La mayoría de los lenguajes ya tienen sus propios linters, a menudo varios:

- **ESLint** para JavaScript.
- **Ruff** para Python.
- **ShellCheck** para scripts de shell. Sus objetivos declarados van desde problemas de sintaxis de principiantes hasta sutilezas que pueden hacer que un script que funciona falle en circunstancias futuras.
- **Stylelint** para CSS, **markdownlint** para Markdown y **yamllint** para YAML.

Todo linter se compone de reglas. Cada regla tiene un identificador, como F841 en Ruff o SC2059 en ShellCheck, y un nivel de gravedad. Un archivo de configuración decide qué reglas están activas y qué gravedad tiene cada una, y muchas incluyen una corrección automática.

Un formateador reescribe la disposición del código sin juzgar su significado; un comprobador de tipos razona sobre qué es cada valor. Un linter se sitúa entre ambos y examina cómo está escrito el código: usos incorrectos del lenguaje, patrones arriesgados y convenciones acordadas por el proyecto.

Utilice el concepto en una decisión

Aplique «¿Qué es un linter?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Un equipo que lleva una idea del boceto a un producto al que sus clientes pueden acceder, sin pasarla entre tres proveedores.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Ingresos multiplicados por 10 con mapas GIS — 10x crecimiento de ingresos impulsado desde el producto cartográfico

Ingresos multiplicados por 10 con mapas GIS Póngase en contacto →

Por qué importa

Un linter lee todas las ramas, incluido el manejador de errores que se ejecuta una vez al año y al que ninguna prueba llega. Tarda segundos en recorrer un repositorio completo, por lo que puede ejecutarse al guardar y en cada commit, y corregir un aviso cuesta menos en el momento de escribir la línea.

Las reglas zanjan las discusiones una sola vez. Si se acepta un `except:` sin especificar un tipo de excepción o si toda variable de shell debe ir entre comillas se decide en la configuración y después se aplica de la misma manera a todos los autores y revisiones. Quienes revisan el código reservan su atención para las preguntas que una regla no puede expresar.

El código de salida es la interfaz entre un linter y aquello que lo ejecuta. El README de ShellCheck dice que la herramienta «hace un uso canónico de los códigos de salida», de modo que una compilación puede invocarla como un comando más. Ruff y ESLint documentan la misma convención de tres resultados: 0 si la comprobación no encuentra problemas, 1 si detecta infracciones y 2 si falla la propia herramienta, por ejemplo por una configuración no válida. Un control puede así distinguir «el código es incorrecto» de «el comprobador está averiado», y ambos casos deben detenerlo.

La gravedad decide qué avisos influyen en ese código de salida. ESLint termina correctamente si encuentra advertencias y ningún error, salvo que `--max-warnings` imponga un límite, y lint(1) dispone de `-w` para tratar las advertencias

como errores. Un proyecto que quiere que sus linters bloqueen los errores y no solo los describan deja esa decisión escrita donde se aplique en cada ejecución:

```
# every finding fails the run, warnings included
lint -w -abcFghprSxz source.c
eslint --max-warnings 0 src/
ruff check src/
shellcheck scripts/*.sh
```

Este sitio adopta la opción más estricta. Su propio control considera cada diagnóstico un error y prohíbe las supresiones en línea, como `# noqa`, para que cada problema se corrija de raíz.

Cómo se aplica en la práctica

La configuración de un linter suele empezar con las reglas que por casualidad no generaron avisos el día en que alguien lo configuró. Una página invierte esa lógica: todas las reglas de un linter de Python activadas como errores. La primera ejecución produjo 1.815 avisos. Se resolvieron por categoría y no por archivo, porque las categorías aportan información: los argumentos sin usar y los nombres que ocultaban elementos integrados del lenguaje eran ruido, mientras que las categorías de seguridad, valores predeterminados mutables y gestión de excepciones señalaban comportamientos reales. Las pocas excepciones llevan un motivo escrito en la línea, y dos están respaldadas por una prueba que verifica la propiedad que la regla desactivada habría exigido. La comprobación estricta de tipos se ejecutó en paralelo y detectó otro tipo de defecto, relacionado con qué es un valor y no con cómo está escrito. También se declara el coste: alguien tiene que decidir 1.815 veces.

Otra página parte de un descubrimiento incómodo. El control de calidad del sitio, treinta y dos scripts y 10.242 líneas de JavaScript, era el mayor conjunto de código del repositorio y el único que no leía ningún formateador, linter ni comprobador de tipos. Someter el propio código del control a un linter empezó por el formateador, configurado con una sangría de dos espacios para esos archivos, de modo que cambiaron 38 líneas en vez de 2.173. El linter produjo después trece avisos reales, todos corregidos y ninguno suprimido, y un comprobador de tipos encontró dos defectos que el linter había dejado pasar. Se explica también el motivo del esfuerzo: un defecto en un comprobador se manifiesta como una comprobación que se supera cuando no debería.

La idea también funciona con la prosa. Una tercera página describe un linter para la propia documentación del repositorio, con límites que hacen fallar la comprobación en lugar de aconsejar. Sus ocho políticas mecánicas incluyen un presupuesto de líneas por archivo que solo puede reducirse, una regla que exige que el estado de un plan coincida con sus casillas y una comprobación de que cada ruta y símbolo de código que nombra un documento exista en el árbol. La primera ejecución encontró seis planes marcados como terminados con casillas sin marcar, catorce tareas obligatorias en cada commit que no figuraban en ningún documento, tres enlaces rotos, un documento huérfano y tres rutas inexistentes.

Dónde surgen los problemas

Un linter rara vez falla de forma evidente. El fallo habitual es una ejecución que termina con 0 cuando debería haber bloqueado algo.

Avisos que nunca cambian el código de salida. El valor predeterminado de `--max-warnings` en ESLint es -1, lo que desactiva el límite, y el manual de lint(1) dice que su pasada entre archivos siempre termina correctamente. Un aviso que no puede hacer fallar la ejecución se lee una vez y luego se pasa por alto. Decida por regla: un error que haga fallar la ejecución o una regla desactivada, y establezca cualquier opción de advertencias como errores en la configuración compartida.

Una corrección que informa de éxito. Ruff termina con 0 cuando ha corregido automáticamente todas las infracciones que encontró. Es cómodo al trabajar localmente, pero un control con las correcciones activadas solo repara su propia copia de trabajo; el commit recibido sigue conteniendo todas las infracciones. Ejecute el control sin correcciones o añada `--exit-non-zero-on-fix` para que una corrección cuente como fallo. Busque también `--exit-zero`: con esa opción, Ruff termina con 0 incluso si encuentra infracciones, y una opción añadida para una migración con muchos avisos tiende a quedarse.

Un patrón que selecciona cero archivos. Un directorio renombrado o una configuración trasladada puede dejar al linter apuntando a un conjunto vacío, y comprobar cero archivos da un resultado sin errores. ESLint trata de forma predeterminada un patrón glob entre comillas que no coincide con ningún archivo como un error. `--no-error-on-unmatched-pattern` lo convierte en una aprobación silenciosa, y `--pass-on-no-patterns` termina con 0 cuando no se indica ningún patrón de archivo. Mantenga el valor predeterminado y un archivo con una infracción conocida que el control deba notificar.

Reglas elegidas porque no generaban avisos. Una configuración construida desactivando todo lo que avisó el primer día registra las costumbres del código, no una norma. Parta de una selección deliberada, ya sea todas las reglas a la vez o la base pequeña de `E` y `F` que sugiere Ruff, añadiendo un grupo cada vez, y deje escrito por qué se desactiva cada regla. Ruff advierte que `ALL` activa implícitamente reglas nuevas cada vez que se actualiza la herramienta; por eso, fije su versión y actualícela como un cambio independiente.

Las supresiones se extienden línea a línea. Todos los linters tienen una vía de escape: `/* LINTED */` en lint(1), `# noqa: F841` en Ruff, `# shellcheck disable=SC2059` en ShellCheck y `// eslint-disable-next-line` en ESLint. Cada una registra qué se silenció, no por qué. Exija un motivo escrito junto a cada excepción y, en las importantes, desactive la regla y añada una prueba de la propiedad que protegía. También ayuda reunir las excepciones en un sitio: `--no-inline-config` de ESLint deja sin efecto los comentarios de desactivación, y ShellCheck lee líneas `disable` para todo el proyecto desde un archivo `.shellcheckrc` .

Lecturas adicionales

- lint(1) — el manual de NetBSD del comprobador de C del que procede el término: sus dos pasadas, `-w` y sus comentarios de supresión.
- El linter Ruff — selección de reglas, seguridad de las correcciones, comentarios de supresión y códigos de salida.
- Referencia de la interfaz de línea de comandos — `--max-warnings` de ESLint, configuración en línea, patrones sin coincidencias y códigos de salida.
- ShellCheck — el README del proyecto: qué busca y cómo se integra en una compilación.
- Ignorar — las formas de ShellCheck de ignorar un aviso por línea, por ejecución y por proyecto.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un linter?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/linter/