Engineer

ESLint

linter de JavaScript guiado por un conjunto de reglas

Qué es

ESLint es una herramienta de análisis estático de código abierto que lee código fuente JavaScript, lo contrasta con un conjunto de reglas configurado e informa de cada lugar donde se incumple una, reparando por sí misma los casos más sencillos cuando se le pide. En la línea de comandos es la orden `eslint`, instalada desde npm como dependencia de desarrollo del proyecto que comprueba. Nicholas C. Zakas la inició en junio de 2013, y el proyecto sigue presentándose como un linter de JavaScript configurable.

Nunca ejecuta el programa. Un analizador convierte cada archivo en un árbol de sintaxis abstracta con formato ESTree, y cada regla es un módulo pequeño que solicita que se lo invoque ante determinados tipos de nodo: cada función, cada comparación, cada `await`. La regla examina lo que recibe y, si algo está mal, emite un mensaje e indica una ubicación. Algunas reglas también incluyen una corrección, un cambio que el proyecto solo acepta si resuelve el problema sin alterar lo que hace el programa; `eslint --fix` aplica esas correcciones a los archivos. Otras ofrecen sugerencias que pueden cambiar el comportamiento, por lo que una persona las aplica de una en una en un editor.

Casi todo lo que rodea ese núcleo se puede sustituir. Los plugins son paquetes npm que añaden reglas, configuraciones que se pueden compartir, procesadores o compatibilidad con otros lenguajes. Otro analizador permite leer sintaxis que el predeterminado no entiende, como TypeScript. Los procesadores extraen JavaScript de otros archivos, por ejemplo, de los bloques de código de un documento Markdown, y los formateadores cambian la presentación del informe. El proyecto explica su filosofía así: cada regla es independiente, se puede desactivar y se puede configurar para advertir o fallar con sus propios criterios.

Todo se configura en un archivo JavaScript en la raíz del proyecto, normalmente `eslint.config.js` o sus variantes `.mjs` y `.cjs`, que exporta un array de objetos. Cada objeto indica los archivos a los que se aplica y sus reglas, variables globales y opciones del analizador; los objetos posteriores prevalecen sobre los anteriores. Esta estructura, llamada configuración plana, pasó a ser la predeterminada en ESLint v9.0.0 el 5 de abril de 2024. ESLint v10.0.0, publicado el 6 de febrero de 2026, eliminó por completo el formato anterior `.eslintrc` y el archivo `.eslintignore`.

Utilice el concepto en una decisión

Aplique «¿Qué es ESLint?» 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 justifica su lugar en un momento: cuando impide integrar un cambio. ESLint toma esa decisión mediante su código de salida, que depende de la gravedad. Cada regla se configura como `"off"`, `"warn"` o `"error"`. Un error hace que la orden termine con 1. Una advertencia se imprime y la orden sigue terminando con 0, salvo que `--max-warnings` fije un límite, desactivado de forma predeterminada. El código de salida 2 se reserva para una configuración incorrecta o un fallo interno.

Por tanto, la decisión que toma realmente un equipo con ESLint es qué significa una advertencia. Una que nunca hace fallar una compilación es un consejo, y los consejos del terminal se pasan por alto cuando ya ocupan cuarenta líneas. Los equipos que quieren que el linter mantenga un criterio obligatorio configuran como `"error"` todas las reglas que les importan o ejecutan el control con `--max-warnings 0`. En ambos casos, una rama en buen estado pasa el linter sin mensajes, y una sola línea nueva es una novedad que alguien lee.

El beneficio está en lo que JavaScript deja pasar. El lenguaje carga un archivo que usa una variable sin declarar, define dos veces la misma clave en un objeto literal, compara un valor con `NaN` mediante `===` o deja tras un `return` código que nunca podrá ejecutarse. Estos problemas pueden sobrevivir a la carga y pasar

inadvertidos o provocar un comportamiento incorrecto al ejecutarse el código, también en producción. Las reglas del conjunto recomendado de ESLint detectan los cuatro antes de la revisión, lo que permite a quien revisa concentrarse en el diseño.

También hay costes. Una configuración acumula opiniones, y una regla que se activa ante código correcto enseña a dejar de leer la salida, algo peor que no tener regla. Un linter también depende del momento en que se ejecuta: si solo funciona en el editor, en algunas máquinas, comprueba parte del código parte del tiempo.

Cómo se aplica en la práctica

ESLint era uno de los seis linters que respaldaban un criterio de cero advertencias en seis lenguajes. Comprobaba TypeScript junto a golangci-lint para Go, SQLFluff para SQL, Ruff para Python, ShellCheck para scripts de shell y markdownlint para Markdown. Lo que sostuvo el criterio fue resolver una vez para todos la cuestión de la gravedad: cada diagnóstico del linter es un error, no hay un nivel «warn» en el que esconderse y se prohíben las supresiones dentro del código. Los hooks previos al commit y al push ejecutan los linters y las pruebas; los fallos dentro del conjunto de reglas y pruebas cubierto bloquean el commit o el push correspondiente.

Después, el mismo repositorio necesitó reglas que no ofrece ningún paquete publicado: las que definen cómo debe funcionar esa base de código concreta. Quince reglas propias de ESLint mantienen los patrones del proyecto en TypeScript, junto a 274 scripts de comprobación, 268 registrados en los hooks de commit. Uno de esos validadores comprueba que nadie haya añadido discretamente una supresión dentro del código. Las comprobaciones tienen 146 pruebas propias, porque una comprobación rota lo deja pasar todo sin que nadie lo sepa. La página reconoce el precio: hacer commits es lento.

Este sitio aplica el mismo nivel de exigencia a sus scripts de compilación, a menor escala. Su `eslint.config.mjs` parte de `js.configs.recommended` y solo añade reglas que el código ya cumplía cuando se midieron. Los comentarios registran una nueva medición del 2026-10-15 sobre las 199 reglas no marcadas como obsoletas, tras la que se habilitaron 21 más centradas en la corrección y no en el gusto. Todas las reglas notifican sus incumplimientos como errores. `reportUnusedDisableDirectives`, que de forma predeterminada genera una advertencia, también se eleva a error, de modo que un comentario de

desactivación que ya no suprime nada hace fallar la ejecución. Los scripts declaran juntas las variables globales de Node y del navegador porque los cuerpos de función que pasan a `page.evaluate` se ejecutan en un navegador, a pocas líneas del código que corre en Node.

Dónde surgen los problemas

La mayoría de los problemas de ESLint consisten en que el linter hace exactamente lo que dice su configuración, que no siempre coincide con lo que el equipo cree que dice.

La compilación está en verde y el terminal está lleno de advertencias. Las advertencias dejan el código de salida en 0, así que la integración continua pasa mientras la lista crece unas líneas por semana. Al cabo de un tiempo contiene un defecto real que nadie lee. Decida qué reglas importan, configúrelas como `"error"`, desactive el resto y añada `--max-warnings 0` al control para que ya no puedan acumularse advertencias.

Un comentario de desactivación sobrevive a su motivo. Una vez se silenció una línea por una buena razón; el código de debajo cambió, el comentario se quedó y ahora oculta el siguiente problema de esa línea o no oculta nada. Configure `linterOptions.reportUnusedDisableDirectives` como `"error"` para que las directivas que no suprimen nada hagan fallar la ejecución y pida un motivo después de `--` en las que queden, para que se puedan valorar durante la revisión:

```
// eslint-disable-next-line no-console -- printing is this CLI's output
console.log(report);
```

Una actualización cambia qué archivos se comprueban. Tras pasar a v10 se informa de rutas que antes omitía `.eslintignore` o una carpeta empieza a seguir reglas distintas. La versión 10 ya no respeta `.eslintrc.*` ni `.eslintignore` y empieza a buscar `eslint.config.*` en el directorio de cada archivo que revisa. Por eso una configuración perdida en una subcarpeta pasa a gobernarla. Mueva las exclusiones a una entrada global `ignores`, convierta el archivo antiguo con `npx @eslint/migrate-config .eslintrc.json` y revise lo que genera; después enumere todos los `eslint.config.*` del árbol y elimine los que nadie pretendía conservar.

Se informa de código del navegador como si no estuviera definido. Un script de Node que pasa funciones a un navegador recibe errores `no-undef` en `document` y `window`, y alguien desactiva `no-undef`, con lo que también oculta erratas reales. Declare mediante `languageOptions.globals` todos los entornos donde se ejecuta realmente el código, para los archivos que se ejecutan allí.

Una regla propia lo deja pasar todo. Una regla cuyo visitante nombra un tipo de nodo que nunca aparece, o cuya condición está invertida, no informa de nada, y el silencio parece exactamente igual que el cumplimiento. Pruebe cada regla propia con `RuleTester` de ESLint, dándole código que deba pasar y código que deba fallar, para que una regla que deja de informar falle en su propia prueba antes de dejar desprotegida la base de código.

Lecturas adicionales

- Conceptos básicos — Reglas, correcciones, plugins, analizadores y procesadores según las definiciones del proyecto.
- Archivos de configuración — El array de configuración plana, los patrones de archivos, las exclusiones globales y cómo se combinan los objetos.
- Configurar reglas — Niveles de gravedad, opciones de reglas y comentarios que desactivan una regla para una línea.
- Referencia de la interfaz de línea de comandos — Todas las opciones, entre ellas `--max-warnings`, y lo que significa cada código de salida.
- Publicación de ESLint v10.0.0 — La eliminación de eslintrc y la nueva búsqueda de configuración, en las notas de la versión.
- Reglas propias — Cómo escribir una regla sobre el árbol de sintaxis y comprobarla con `RuleTester`.

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 ESLint?» 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/eslint/