Engineer

ESLint

linter JavaScript fondé sur un ensemble de règles

Ce que c’est

ESLint est un outil d’analyse statique open source : il lit le code JavaScript, le confronte à un ensemble de règles configurées et signale chaque infraction. Sur demande, il répare lui-même les cas simples. En ligne de commande, c’est la commande `eslint`, installée depuis npm comme dépendance de développement du projet contrôlé. Nicholas C. Zakas l’a lancé en juin 2013. Le projet se présente toujours comme un linter JavaScript configurable.

Il n’exécute jamais le programme. Un analyseur convertit chaque fichier en arbre syntaxique abstrait au format ESTree. Chaque règle est un petit module qui demande à être appelé sur certains types de nœuds : chaque fonction, comparaison ou `await`. Elle examine ce qui lui est transmis et, si quelque chose ne va pas, émet un message et indique un emplacement. Certaines règles fournissent aussi une correction. Le projet ne l’accepte que si elle corrige le problème sans changer le comportement du programme, et `eslint --fix` l’applique aux fichiers. D’autres proposent plutôt des suggestions pouvant modifier ce comportement. Une personne les applique donc une par une dans son éditeur.

Presque tout ce qui entoure ce noyau peut être remplacé. Les plugins sont des paquets npm ajoutant des règles, des configurations partageables, des processeurs ou la prise en charge d’autres langages. Un autre analyseur permet de lire une syntaxe que l’analyseur standard ne comprend pas, comme TypeScript. Les processeurs extraient JavaScript d’autres fichiers, par exemple les blocs de code d’un document Markdown ; les outils de mise en forme changent la présentation du rapport. Le projet décrit sa philosophie ainsi : chaque règle est indépendante, peut être désactivée et peut, selon ses propres conditions, avertir ou faire échouer le contrôle.

Tout cela se règle dans un fichier JavaScript à la racine du projet, généralement `eslint.config.js` ou ses variantes `.mjs` et `.cjs`, qui exporte un tableau d'objets. Chacun désigne les fichiers concernés et les règles, variables globales et options d'analyse qui s'y appliquent. Les objets ultérieurs remplacent les paramètres des précédents. Cette organisation, appelée flat config, est devenue le format de configuration standard dans ESLint v9.0.0 le 5 avril 2024. ESLint v10.0.0, publié le 6 février 2026, a entièrement supprimé l'ancien format `.eslintrc` et le fichier `.eslintignore`.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'ESLint ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une seule équipe transforme une idée en un produit auquel vos clients peuvent se connecter, sans le faire passer entre trois prestataires.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Application GIS : revenus multipliés par 10 — 10x croissance du chiffre d'affaires portée par le produit cartographique

Application GIS : revenus multipliés par 10 Contactez-nous →

Pourquoi c'est important

Un linter justifie sa place à un instant précis : lorsqu'il empêche la fusion d'un changement. ESLint décide par son code de sortie, déterminé par la gravité. Chaque règle vaut `"off"`, `"warn"` ou `"error"`. Une erreur fait terminer la commande avec le code 1. Un avertissement s'affiche mais laisse le code à 0, sauf si `--max-warnings` fixe un plafond, inactif sans réglage explicite. Le code 2 est réservé à une configuration défectueuse ou à un plantage interne.

La vraie décision d'équipe est donc le sens d'un avertissement. S'il ne fait jamais échouer le build, c'est un conseil. Quarante lignes de conseils dans un terminal finissent par défiler sans lecture. Pour faire respecter un seuil, les équipes règlent chaque règle importante sur `"error"` ou exécutent leur contrôle bloquant avec `--max-warnings 0`. Dans les deux cas, une branche saine ne produit aucun diagnostic, et une nouvelle ligne devient une information que quelqu'un lit.

Le bénéfice tient à ce que JavaScript laisse passer. Le langage charge un fichier qui référence une variable jamais déclarée, définit deux fois la même clé dans un objet littéral, compare une valeur à `NaN` avec `===` ou laisse du code inaccessible après un `return`. Ces problèmes peuvent subsister au chargement et passer inaperçus, ou provoquer un comportement incorrect lors de

l'exécution, y compris en production. Les règles recommandées d'ESLint détectent les quatre avant la revue, laissant la personne qui la mène se concentrer sur la conception.

Il y a des coûts. Une configuration accumule des opinions, et une règle qui se déclenche sur du code correct apprend aux gens à ignorer la sortie. C'est pire que l'absence de règle. L'utilité d'un linter dépend aussi du moment où il s'exécute : limité à l'éditeur de certaines machines, il ne vérifie qu'une partie du code, une partie du temps.

À quoi cela ressemble en pratique

ESLint était l'un des six linters derrière une exigence de zéro avertissement sur six langages. Il vérifiait TypeScript aux côtés de golangci-lint pour Go, SQLFluff pour SQL, Ruff pour Python, ShellCheck pour les scripts shell et markdownlint pour Markdown. Le seuil tenait parce que la question de gravité avait été tranchée une fois pour tous : chaque diagnostic est une erreur, aucun niveau « warn » ne sert de refuge, et les suppressions en ligne sont interdites. Les hooks pre-commit et pre-push exécutent linters et tests ; les échecs dans le périmètre des règles et tests couverts bloquent le commit ou le push concerné.

Le même dépôt a ensuite eu besoin de règles absentes des paquets publiés, propres au fonctionnement attendu de cette base de code. Quinze règles ESLint sur mesure maintiennent les conventions TypeScript du dépôt, avec 274 scripts de contrôle, dont 268 enregistrés dans les hooks de commit. Un validateur vérifie que personne n'a ajouté discrètement une suppression en ligne. Les garde-fous ont leurs propres 146 tests : un contrôle cassé laisse tout passer sans que personne le découvre. La page en nomme clairement le prix : les commits sont lents.

Ce site impose la même exigence à ses scripts de construction, à plus petite échelle. Son `eslint.config.mjs` part de `js.configs.recommended` et n'ajoute que des règles auxquelles le code satisfaisait déjà lors des mesures. Les commentaires consignent une nouvelle mesure effectuée le 2026-10-15 sur les 199 règles non dépréciées : elle a activé 21 règles supplémentaires portant sur la justesse du code plutôt que les préférences stylistiques. Chaque règle est une erreur. `reportUnusedDisableDirectives`, signalé comme avertissement sans configuration contraire, est lui aussi élevé au rang d'erreur : un commentaire de désactivation devenu inutile fait échouer l'exécution. Les scripts déclarent ensemble les

variables globales de Node et du navigateur, car les corps de fonctions passés à `page.evaluate` s'exécutent dans un navigateur, à quelques lignes du code exécuté dans Node.

Où les problèmes surviennent

La plupart des problèmes ESLint viennent d'un linter qui fait exactement ce que dit sa configuration, ce qui ne correspond pas toujours à ce que l'équipe croit y avoir écrit.

Le build est vert, le terminal plein d'avertissements. Les avertissements laissent le code de sortie à 0. L'intégration continue passe donc pendant que la liste gagne quelques lignes par semaine, jusqu'à contenir un vrai défaut que personne ne lit. Décidez des règles importantes, réglez-les sur `"error"`, désactivez les autres et ajoutez `--max-warnings 0` au contrôle bloquant pour empêcher cette accumulation.

Un commentaire de désactivation survit à sa raison. Une ligne avait été réduite au silence pour une bonne raison. Le code a changé, le commentaire est resté ; il masque désormais le problème suivant sur cette ligne, ou ne masque rien. Réglez `linterOptions.reportUnusedDisableDirectives` sur `"error"` pour faire échouer l'exécution lorsqu'une directive ne supprime rien. Pour celles qui restent, exigez une raison après `--`, afin que la personne chargée de la revue puisse la juger :

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

Une mise à niveau change les fichiers contrôlés. Après le passage à v10, des chemins autrefois exclus par `.eslintignore` sont soudain signalés, ou un dossier suit d'autres règles. La version 10 ne tient plus compte de `.eslintrc.*` ni de `.eslintignore`. Elle commence sa recherche d'`eslint.config.*` dans le dossier de chaque fichier contrôlé : une configuration oubliée dans un sous-dossier le gouverne désormais. Déplacez les exclusions dans une entrée globale `ignores`, convertissez l'ancien fichier avec `npx @eslint/migrate-config .eslintrc.json` et examinez le résultat. Recensez ensuite chaque `eslint.config.*` dans l'arborescence et retirez ceux que personne ne voulait conserver.

Le code exécuté dans le navigateur est signalé comme non défini. Un script Node qui transmet des fonctions au navigateur reçoit des erreurs `no-undef` sur `document` et `window`. Quelqu'un désactive alors `no-undef`, masquant aussi de vraies fautes de frappe. Déclarez chaque environnement d'exécution réel dans `languageOptions.globals`, pour les fichiers qui y tournent.

Une règle sur mesure laisse tout passer. Une règle dont le visiteur désigne un type de nœud absent, ou dont la condition est inversée, ne signale simplement rien. Le silence ressemble exactement à la conformité. Testez chaque règle avec `RuleTester` d'ESLint, avec du code qui doit passer et du code qui doit échouer. Une règle qui cesse de signaler échouera ainsi à son propre test avant de faillir à sa mission dans la base de code.

Pour aller plus loin

- Concepts fondamentaux — règles, corrections, plugins, analyseurs et processeurs, tels que définis par le projet.
- Fichiers de configuration — tableau flat config, motifs de fichiers, exclusions globales et fusion des objets.
- Configurer les règles — gravités, options et commentaires désactivant une règle pour une ligne.
- Référence de la ligne de commande — toutes les options, dont `--max-warnings`, et le sens de chaque code de sortie.
- Publication d'ESLint v10.0.0 — suppression d'eslintrc et nouvelle recherche des configurations, dans les notes de version.
- Règles sur mesure — écrire une règle sur l'arbre syntaxique et la vérifier avec `RuleTester`.



Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'ESLint ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/eslint/