Engineer

Linter

🖋 et værktøj, der finder mistænkelig kode uden at køre den

Hvad det er

En linter er et program, der læser kildekode og melder de steder, der ligner fejl, uden at bygge programmet eller køre det. Hver melding nævner en regel, en fil og en linje: en variabel, der tildeles og aldrig bruges, en sammenligning, der altid er sand, en shell-variabel uden anførselstegn, en formatstreng bygget af brugerinput.

Navnet kommer fra lint, et tjekværktøj til C-programmer, og NetBSD leverer stadig et. Dets manual beskriver opgaven som at finde de træk ved C-programfiler, der sandsynligvis er fejl, ikke er portable eller er spild, og tilføjer, at det typetjekker strengere end traditionelle C-oversættere fra før C90. Det arbejder i to gennemløb: først tjekkes hver fil som en enhed for sig, derefter krydstjekkes resultaterne for uoverensstemmelser mellem filerne.

De fleste sprog har nu deres egne lintere, ofte flere:

- **ESLint** til JavaScript.
- **Ruff** til Python.
- **ShellCheck** til shellscripts. Værktøjets erklærede mål spænder fra begynderes syntaksfejl til subtile faldgruber, der kan få et fungerende script til at fejle under fremtidige forhold.
- **Stylelint** til CSS, **markdownlint** til Markdown og **yamllint** til YAML.

Enhver linter er bygget af regler. Hver regel har en identifikator, for eksempel F841 i Ruff eller SC2059 i ShellCheck, og en alvorsgrad. En konfigurationsfil afgør, hvilke regler der er slået til, og hvor alvorlig hver enkelt er, og mange regler har en automatisk rettelse.

En formatter omskriver layout og har ingen holdning til betydning; en typetjekker ræsonnerer om, hvad hver værdi er. En linter ligger imellem og ser på, hvordan koden er skrevet: misbrug af sproget, risikable mønstre og de konventioner, et projekt har aftalt.

Brug begrebet i en beslutning

Brug »Hvad er en linter?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Ét team, der tager en idé fra skitse til noget, jeres kunder kan logge ind på, i stedet for at kaste den mellem tre leverandører.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har ført en GIS-kortapp til 10x omsætningsvækst — 10x omsætningsvækst, drevet af kortproduktet

Har ført en GIS-kortapp til 10x omsætningsvækst Kontakt os →

Hvorfor det betyder noget

En linter læser hver gren, også den fejlhåndtering, der kører én gang om året, og som ingen test når. Den tager sekunder, så den kan køre ved hvert commit, og et fund er billigst, mens linjen skrives.

Regler afgør en diskussion én gang. Om en bar `except:` er acceptabel, eller om hver shell-variabel skal i anførselstegn, besluttes i konfigurationen og anvendes derefter ens på hver forfatter og hvert review. Menneskelige reviewere kan koncentrere sig om de spørgsmål, en regel ikke kan udtrykke.

Exitkoden er grænsefladen mellem en linter og det, der kører den. ShellChecks README siger, at værktøjet bruger exitkoder på den kanoniske måde, så et build kan kalde det som én kommando mere. Ruff og ESLint dokumenterer den samme tredelte konvention: 0, når tjekket er rent, 1, når det fandt overtrædelser, og 2, når selve værktøjet fejlede, for eksempel på en ugyldig konfiguration. En port kan så skelne »koden er forkert« fra »tjekket er i stykker«, og begge bør stoppe den.

Alvorsgraden afgør, hvilke fund der når exitkoden. ESLint afslutter med succes, når det finder advarsler og ingen fejl, medmindre `--max-warnings` sætter en grænse, og lint(1) har `-w` til at behandle advarsler som fejl. Et projekt, der vil have sine lintere til at blokere fejl og ikke bare beskrive dem, skriver valget ned dér, hvor hver kørsel bruger det:

```
# hvert fund får kørslen til at fejle, også advarsler
lint -w -abcFghprSxz source.c
eslint --max-warnings 0 src/
ruff check src/
shellcheck scripts/*.sh
```

Dette site har valgt den strikse ende. Dets egen port tæller hver diagnostik som en fejl og forbyder inline-undertrykkelser som `# noqa`, så et fund bliver rettet ved årsagen.

Sådan ser det ud i praksis

En linters konfiguration begynder ofte som de regler, der tilfældigvis var stille den dag, nogen satte den op. En side vender det om: hver regel i en Python-linter slået til som fejl. Første kørsel gav 1.815 fund. De blev gennemgået efter kategori og ikke efter fil, fordi kategorierne fortæller noget: ubrugte argumenter og skyggede indbyggede navne var støj, mens kategorierne for sikkerhed, mutable standardværdier og fejlhåndtering hver pegede på reel adfærd. De få undtagelser har en skreven begrundelse ved linjen, og to af dem er bakket op af en test, der fastslår den egenskab, som den frakoblede regel ville have håndhævet. Strengt typetjek kørte ved siden af og fangede en anden slags defekt, om hvad en værdi er snarere end hvordan den er skrevet. Prisen står der også: nogen skal foretage en vurdering 1.815 gange.

En anden side begynder med en ubehagelig opdagelse. Sitets kvalitetsport, toogtredive scripts og 10.242 linjer JavaScript, var den største samling kode i repositoriet og den eneste, som hverken formatter, linter eller typetjekker læste. Arbejdet med at lægge portens egen kode under en linter begyndte med formatteren, sat til to mellemrums indrykning for de filer, så 38 linjer ændrede sig i stedet for 2.173. Linteren gav derefter tretten reelle fund, alle rettet og ingen undertrykt, og en typetjekker fandt to defekter, som linteren havde ladet passere. Begrundelsen står ved siden af: en defekt i et tjek viser sig som et tjek, der består, når det burde fejle.

Idéen virker også på prosa. En tredje side beskriver en linter til repositoriets egen dokumentation med grænser, der får noget til at fejle i stedet for at give råd. Dens otte mekaniske politikker omfatter et linjebudget pr. fil, der kun kan skrumpe, en regel om, at en plans status skal stemme med dens egne afkrydsningsfelter, og et tjek af, at hver sti og hvert kodesymbol, et dokument

nævner, findes i træet. Første kørsel fandt seks planer markeret som færdige med uafkrydsede felter, fjorten opgaver, der gater hvert commit uden at stå i noget dokument, tre døde links, et forældreløst dokument og tre døde stier.

Hvor det går galt

En linter fejler sjældent højlydt. Den typiske fejl er en kørsel, der afslutter med 0, når den burde have stoppet noget.

Fund, der aldrig ændrer exitkoden. ESLints `--max-warnings` har -1 som standard, hvilket slår grænsen fra, og ifølge manualen til lint(1) afslutter krydstjekket mellem filer altid med succes. Beslut pr. regel: en fejl, der stopper kørslen, eller slået fra, og sæt en eventuel indstilling for advarsler som fejl i den fælles konfiguration.

En rettelse, der melder succes. Ruff afslutter med 0, når hver overtrædelse, det fandt, blev rettet automatisk. En port, der kører med rettelser slået til, reparerer kun sin egen arbejdskopi, og det commit, der kom ind, indeholder stadig hver overtrædelse. Kør porten uden rettelser, eller tilføj `--exit-non-zero-on-fix`, så en rettelse tæller som en fejl. Søg også efter `--exit-zero`: med det afslutter Ruff med 0, selv når der er overtrædelser.

Et mønster, der rammer nul filer. En omdøbt mappe eller en flyttet konfiguration kan efterlade en linter rettet mod en tom mængde, og at tjekke nul filer er et rent resultat. ESLint behandler som standard en glob i anførselstegn, der rammer nul filer, som en fejl. `--no-error-on-unmatched-pattern` gør det til et stille bestået, og `--pass-on-no-patterns` afslutter med 0, når der slet ikke angives filmønstre. Behold standarden, og hold én fil med en kendt overtrædelse, som porten skal melde.

Et regelsæt valgt efter, hvad der var stille. En konfiguration, der er bygget ved at slå alt fra, der gav udslag den første dag, registrerer kodens vaner og ikke en standard. Begynd med et bevidst valg, enten alt på én gang eller Ruffs forslag om en lille base af `E` og `F` med én gruppe tilføjet ad gangen, og skriv ned, hvorfor hver frakoblet regel er slået fra. Ruff advarer om, at `ALL` stiltiende slår nye regler til, hver gang værktøjet opgraderes, så lås linterens version, og opgradér den som en ændring for sig.

Undertrykkelser, der spredes én linje ad gangen. Hver linter har en nødudgang: `/* LINTED */` i lint(1), `# noqa: F841` i Ruff, `# shellcheck disable=SC2059` i ShellCheck og `// eslint-disable-next-line` i ESLint. Hver af dem registrerer, hvad der blev gjort tavst, men ikke hvorfor. Kræv en skreven begrundelse ved hver undtagelse, og for dem, der betyder noget, slå reglen fra, og tilføj en test for den egenskab, den beskyttede. Saml undtagelserne ét sted: ESLints `--no-inline-config` gør disable-kommentarer virkningsløse, og ShellCheck læser `disable`-linjer for hele projektet fra en `.shellcheckrc`-fil.

Læs mere

- lint(1) — NetBSD's manual for det C-tjekværktøj, ordet stammer fra: de to gennemløb, `-w` og kommentarerne, der undertrykker meldinger.
- The Ruff Linter — regelvalg, sikre og usikre rettelser, undertrykkende kommentarer og exitkoder.
- Command Line Interface Reference — ESLints `--max-warnings`, inline-konfiguration, umatchede mønstre og exitkoder.
- ShellCheck — projektets README: hvad værktøjet leder efter, og hvordan det passer ind i et build.
- Ignore — ShellChecks måder at ignorere et fund på pr. linje, pr. kørsel og pr. projekt.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en linter?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/linter/