Engineer

Linter

🖋 et verktøy som påpeker trolige feil i kode uten å kjøre den

Hva det er

En linter er et program som leser kildekode og melder fra om de delene som ser ut som feil, uten å bygge programmet eller kjøre det. Hver melding navngir en regel, en fil og en linje: en variabel som får en verdi og aldri brukes, en sammenligning som alltid er sann, en skallvariabel uten anførselstegn, en formatstreng bygd av inndata fra brukeren.

Navnet kommer fra lint, et kontrollverktøy for C-programmer, og NetBSD leverer fortsatt et. Manualen beskriver jobben som å finne trekk ved C-programfiler som sannsynligvis er feil, ikke er portable eller er sløsete, og legger til at verktøyet gjør strengere typekontroll enn tradisjonelle kompilatorer fra før C90. Det arbeider i to omganger: først sjekkes hver fil som en egen enhet, deretter krysskontrolleres resultatene for uoverensstemmelser mellom filene.

De fleste språk har nå egne lintere, ofte flere:

- **ESLint** for JavaScript.
- **Ruff** for Python.
- **ShellCheck** for skallskript. Målene prosjektet oppgir, spenner fra syntaksproblemer nybegynnere møter, til subtile fallgruver som kan få et skript som virker i dag, til å feile under fremtidige omstendigheter.
- **Stylelint** for CSS, **markdownlint** for Markdown og **yamllint** for YAML.

Hver linter er bygd av regler. Hver regel har en identifikator, som F841 i Ruff eller SC2059 i ShellCheck, og en alvorlighetsgrad. En konfigurasjonsfil avgjør hvilke regler som er slått på, og hvor alvorlig hver av dem er, og mange regler har en automatisk rettelse.

En formaterer skriver om oppsettet og har ingen mening om betydningen; en typekontroll resonnerer om hva hver verdi er. En linter ligger mellom de to og ser på hvordan koden er skrevet: feil bruk av språket, risikable mønstre og konvensjonene et prosjekt har blitt enig om.

Bruk begrepet i en beslutning

Bruk «Hva er en linter?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Ett team som tar en idé fra skisse til noe kundene dine kan logge inn på, i stedet for at den sendes videre mellom tre leverandører.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Ledet en GIS-kartapp til 10 ganger mer omsetning — 10x inntektsvekst, ledet fra kartproduktet

Ledet en GIS-kartapp til 10 ganger mer omsetning Ta kontakt →

Hvorfor det er viktig

En linter leser hver gren, også feilhåndteringen som kjører én gang i året, og som ingen test når. Den bruker sekunder på et helt repo, så den kan kjøre ved hver lagring og hver commit, og et funn koster minst i det øyeblikket linjen skrives.

Regler avgjør diskusjoner én gang. Om en naken `except:` er akseptabel, eller om hver skallvariabel må stå i anførselstegn, avgjøres i konfigurasjonen og brukes deretter likt på hver forfatter og hver gjennomgang. De som går gjennom koden, kan spare oppmerksomheten sin til spørsmålene en regel ikke kan uttrykke.

Avslutningskoden er grensesnittet mellom en linter og det som kjører den. README-filen til ShellCheck sier at verktøyet bruker avslutningskoder på den kanoniske måten, så en bygging kan kalle det som én kommando til. Ruff og ESLint dokumenterer den samme tredelte konvensjonen: 0 når sjekken er ren, 1 når den fant brudd, og 2 når selve verktøyet feilet, for eksempel på en ugyldig konfigurasjon. En kvalitetsport kan da skille «koden er feil» fra «kontrollen er ødelagt», og begge bør stoppe den.

Alvorlighetsgraden avgjør hvilke funn som når avslutningskoden. ESLint avslutter med suksess når det finner advarsler og ingen feil, med mindre `--max-warnings` setter en grense, og lint(1) har `-w` for å behandle advarsler som feil. Et prosjekt som vil at linterne skal stoppe feil, og ikke bare beskrive dem, skriver det valget ned der hver kjøring bruker det:

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

Dette nettstedet har valgt den strenge enden. Den egne kvalitetsporten regner hver diagnose som en feil og forbyr undertrykking på linjen, som `# noqa`, så et funn rettes der det oppstår.

Slik ser det ut i praksis

Konfigurasjonen til en linter begynner ofte som de reglene som tilfeldigvis var stille den dagen noen satte den opp. Én side snur det på hodet: hver regel i en Python-linter slått på som feil. Den første kjøringen ga 1 815 funn. De ble gått gjennom etter kategori og ikke etter fil, fordi kategoriene bærer informasjon: ubrukte argumenter og innebygde navn som ble skygget, var støy, mens kategoriene for sikkerhet, muterbare standardverdier og unntakshåndtering hver pekte på reell oppførsel. De få unntakene har en skriftlig begrunnelse på linjen, og to av dem er støttet av en test som slår fast egenskapen den avslåtte regelen ville ha håndhevet. Streng typekontroll kjørte ved siden av og fanget en annen sort programvarefeil, om hva en verdi er og ikke hvordan den er skrevet. Kostnaden er også oppgitt: noen må gjøre en vurdering 1 815 ganger.

En annen side begynner med en ubehagelig oppdagelse. Nettstedets kvalitetsport, trettito skript og 10 242 linjer JavaScript, var den største kodemengden i repoet og den eneste som ingen formaterer, linter eller typekontroll leste. Å legge portens egen kode under en linter begynte med formatereren, satt til to mellomroms innrykk for de filene, slik at 38 linjer ble endret i stedet for 2 173. Linteren ga deretter tretten reelle funn, alle rettet og ingen undertrykt, og en typekontroll fant to feil linteren hadde sluppet gjennom. Grunnen til innsatsen står sammen med den: en feil i et kontrollskript viser seg som en kontroll som består når den ikke burde.

Ideen virker også på prosa. En tredje side beskriver en linter for repoets egen dokumentasjon, med grenser som feiler i stedet for å gi råd. Blant de åtte mekaniske policyene er et linjebudsjett per fil som bare kan krympe, en regel om at statusen til en plan må stemme med dens egne avkrysningsbokser, og en kontroll av at hver sti og hvert kodesymbol et dokument nevner, finnes i treet.

Den første kjøringen fant seks planer merket som ferdige med uavkryssede bokser, fjorten oppgaver som hver commit måtte gjennom uten å være nevnt i noe dokument, tre ødelagte lenker, ett foreldreløst dokument og tre døde stier.

Hvor det går galt

En linter svikter sjelden med brask og bram. Den vanlige feilen er en kjøring som avslutter med 0 når den skulle ha stoppet noe.

Funn som aldri endrer avslutningskoden. `--max-warnings` i ESLint har `-1` som standard, noe som slår grensen av, og manualen til lint(1) sier at krysskontrollen mellom filene alltid avslutter med suksess. Et funn som ikke kan få kjøringen til å feile, leses én gang og blir så rullet forbi. Bestem for hver regel om den skal være en feil som stopper kjøringen, eller slås av, og legg bryteren som gjør advarsler til feil, i den delte konfigurasjonen.

En rettelse som melder suksess. Ruff avslutter med 0 når hvert brudd det fant, ble rettet automatisk. Det er praktisk på din egen maskin, men en kvalitetsport som kjører med rettelser slått på, reparerer bare sin egen arbeidskopi, og commiten som kom inn, inneholder fortsatt hvert brudd. Kjør porten uten rettelser, eller legg til `--exit-non-zero-on-fix` slik at en rettelse teller som en feil. Søk etter `--exit-zero` også: med det flagget avslutter Ruff med 0 selv når det finner brudd, og et flagg lagt til for én støyende migrering har en tendens til å bli værende.

Et mønster som velger null filer. En katalog som har fått nytt navn, eller en konfigurasjon som er flyttet, kan etterlate en linter rettet mot et tomt sett, og å sjekke null filer gir et rent resultat. ESLint behandler som standard et glob-mønster i anførselstegn som ikke treffer noen filer, som en feil. `--no-error-on-unmatched-pattern` gjør det om til en stille godkjenning, og `--pass-on-no-patterns` avslutter med 0 når ingen filmønstre er oppgitt. Behold standarden, og behold én fil med et kjent brudd som porten må melde.

Et regelsett valgt etter hva som var stille. En konfigurasjon bygd ved å slå av alt som slo ut første dag, registrerer kodens vaner og ikke en standard. Begynn med et bevisst utvalg, enten alt på én gang eller den lille basen av `E` og `F` som Ruff foreslår, med én gruppe lagt til av gangen, og skriv ned hvorfor hver avslått regel er av. Ruff advarer om at `ALL` implisitt slår på nye regler hver gang verktøyet oppgraderes, så lås versjonen av linteren og oppgrader den som en egen endring.

Undertrykkinger sprer seg én linje om gangen. Hver linter har en nødutgang: `/* LINTED */` i lint(1), `# noqa: F841` i Ruff, `# shellcheck disable=SC2059` i ShellCheck og `// eslint-disable-next-line` i ESLint. Hver av dem registrerer hva som ble gjort taust, ikke hvorfor. Krev en skriftlig begrunnelse ved hvert unntak, og for de unntakene som betyr noe: slå regelen av, og legg til en test for egenskapen den beskyttet. Det hjelper også å holde unntakene på ett sted: `--no-inline-config` i ESLint gjør kommentarer som slår av regler, virkningsløse, og ShellCheck leser `disable` -linjer for hele prosjektet fra en `.shellcheckrc` -fil.

Les mer

- lint(1) — manualen fra NetBSD for C-kontrollverktøyet ordet kommer fra: de to omgangene, `-w` og kommentarene for undertrykking.
- The Ruff Linter — regelvalg, hvor trygge rettelsene er, kommentarer for undertrykking og avslutningskoder.
- Command Line Interface Reference — `--max-warnings` i ESLint, konfigurasjon på linjen, mønstre som ikke treffer, og avslutningskoder.
- ShellCheck — prosjektets README: hva verktøyet ser etter, og hvordan det passer inn i en bygging.
- Ignore — måtene ShellCheck lar deg ignorere et funn på, per linje, per kjøring og per prosjekt.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en linter?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/linter/