Engineer

Linter

🖋 ett verktyg som flaggar troliga buggar utan att köra koden

Vad det är

En linter läser källkoden som text och rapporterar de ställen som ser ut som misstag, innan programmet ens har byggts eller startats. Varje rapport anger en regel, en fil och en rad: en variabel som tilldelas men aldrig används, en jämförelse som alltid är sann, en skalvariabel som står utan citattecken, en formatsträng som byggs av indata från användaren.

Namnet kommer från lint, ett granskningsprogram för C, och NetBSD levereras fortfarande med ett sådant. Manualsidan beskriver uppgiften som att hitta de drag i C-filer som sannolikt är buggar, som inte är portabla eller som slösar resurser, och tillägger att lint gör en striktare typkontroll än de traditionella kompilatorerna från tiden före C90. Arbetet sker i två pass: först granskas varje fil som en egen enhet, sedan jämförs resultaten för att hitta motsägelser mellan filerna.

De flesta språk har i dag egna linters, ofta flera:

- **ESLint** för JavaScript.
- **Ruff** för Python.
- **ShellCheck** för skalskript. Projektets uttalade mål sträcker sig från syntaxfel som nybörjare gör till subtila fallgropar som kan få ett fungerande skript att fallera under framtida omständigheter.
- **Stylelint** för CSS, **markdownlint** för Markdown och **yamllint** för YAML.

Varje linter är uppbyggd av regler. Varje regel har en identifierare, t.ex. F841 i Ruff eller SC2059 i ShellCheck, och en allvarlighetsgrad. En konfigurationsfil avgör vilka regler som är påslagna och hur allvarlig var och en är, och många regler har en automatisk rättning.

En formaterare skriver om layouten och har ingen åsikt om innebörden; en typkontroll resonerar om vad varje värde är. En linter står mellan de två och tittar på hur koden är skriven: felaktig användning av språket, riskabla mönster och de konventioner som ett projekt har kommit överens om.

Använd begreppet i ett beslut

Använd ”Vad är en linter?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Ett team som tar en idé från en skiss till något dina kunder kan logga in i, i stället för att lämna den vidare mellan tre leverantörer.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Ledde GIS-kartappen till 10 gånger högre intäkter — 10x intäktstillväxt, med kartprodukten i spetsen

Ledde GIS-kartappen till 10 gånger högre intäkter Hör av er →

Varför det spelar roll

En linter läser varje gren, även felhanteraren som körs en gång om året och som inget test når fram till. Den behöver bara sekunder för ett helt repo, så den kan köras vid varje sparning och varje commit, och ett fynd kostar som minst i samma stund som raden skrivs.

Reglerna avgör diskussioner en gång för alla. Om ett bart `except:` är godtagbart, eller om varje skalvariabel måste stå inom citattecken, bestäms i konfigurationen och tillämpas sedan likadant på varje författare och i varje granskning. Mänskliga granskare kan då lägga sin uppmärksamhet på de frågor som en regel inte kan uttrycka.

Slutkoden är gränssnittet mellan en linter och det som kör den. ShellChecks README framhåller att verktyget använder slutkoder på det vedertagna sättet, så att ett bygge kan anropa det som ett kommando bland andra. Ruff och ESLint dokumenterar samma tredelade konvention: 0 när granskningen är ren, 1 när den har hittat överträdelser och 2 när själva verktyget har fallerat, t.ex. på en ogiltig konfiguration. En kvalitetsgrind kan då skilja ”koden är fel” från ”granskaren är trasig”, och båda bör stoppa den.

Allvarlighetsgraden avgör vilka fynd som når fram till slutkoden. ESLint avslutar med framgång när det hittar varningar men inga fel, om inte `--max-warnings` sätter en gräns, och lint(1) har `-w` för att behandla varningar som fel. Ett projekt som vill att dess linters ska stoppa misstag, och inte bara beskriva dem, skriver ned det valet där varje körning använder det:

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

Den här webbplatsen har valt den strikta änden av det spektrumet. Dess egen kvalitetsgrind räknar varje diagnostik som ett fel och förbjuder undertryckningar på raden, som `# noqa`, så att ett fynd åtgärdas vid sin orsak.

Så ser det ut i praktiken

En linters konfiguration börjar ofta som de regler som råkade vara tysta den dag någon satte upp den. En sida här vänder på det, med varje regel i en Python-linter aktiverad som fel. Den första körningen gav 1 815 fynd. De arbetades igenom kategori för kategori snarare än fil för fil, eftersom kategorierna bär information: oanvända argument och skuggade inbyggda namn var brus, medan kategorierna för säkerhet, muterbara standardvärden och undantagshantering var och en pekade på verkligt beteende. De få undantagen har ett skriftligt skäl på raden, och två av dem backas upp av ett test som säkerställer den egenskap som den avstängda regeln skulle ha upprätthållit. Strikt typkontroll kördes vid sidan av och fångade en annan sorts fel, som gällde vad ett värde är snarare än hur det är skrivet. Kostnaden redovisas också: någon måste göra en bedömning 1 815 gånger.

Ett annat arbete börjar med en obekväm upptäckt. Webbplatsens kvalitetsgrind, trettiotvå skript på sammanlagt 10 242 rader JavaScript, var den största kodmassan i repot och den enda som ingen formaterare, linter eller typkontroll läste. Arbetet med att föra in grindens egen kod under en linter började med formateraren, inställd på två mellanslags indrag för de filerna, så att 38 rader ändrades i stället för 2 173. Lintern gav sedan tretton verkliga fynd, alla åtgärdade och inget undertryckt, och en typkontroll hittade två fel som lintern hade släppt igenom. Skälet till arbetet står bredvid: ett fel i ett kontrollprogram visar sig som en kontroll som går igenom när den inte borde.

Idén fungerar också på prosa. En tredje sida beskriver en linter för repots egen dokumentation, med gränser som fallerar i stället för att ge råd. Bland dess åtta mekaniska policyer finns en radbudget per fil som bara kan sänkas, en regel om att statusen i en plan måste stämma med planens egna kryssrutor och en kontroll av att varje sökväg och kodsymbol som ett dokument nämner faktiskt finns i

trädet. Den första körningen hittade sex planer som var markerade som klara men hade okryssade rutor, fjorton uppgifter som spärrade varje commit utan att nämnas i något dokument, tre trasiga länkar, ett föräldralöst dokument och tre döda sökvägar.

Var det går fel

Det typiska problemet med en linter är tyst: en körning som avslutas med 0 när den borde ha stoppat något.

Fynd som aldrig påverkar slutkoden. ESLints `--max-warnings` har värdet –1 som standard, vilket stänger av gränsen, och manualsidan för lint(1) säger att passet över flera filer alltid avslutas med framgång. Ett fynd som inte kan fälla körningen läses en gång och rullas sedan förbi. Bestäm per regel: ett fel som fäller körningen, eller avstängd, och lägg varje inställning som gör varningar till fel i den gemensamma konfigurationen.

En rättning som rapporterar framgång. Ruff avslutar med 0 när varje överträdelse som det hittade rättades automatiskt. Det är praktiskt vid skrivbordet, men en grind som körs med rättningar påslagna lagar bara sin egen arbetskopia, och den commit som kom in innehåller fortfarande varje överträdelse. Kör grinden utan rättningar, eller lägg till `--exit-non-zero-on-fix` så att en rättning räknas som ett misslyckande. Sök också efter `--exit-zero` : med den avslutar Ruff med 0 även när överträdelser hittas, och en flagga som lades till för en enda bullrig migrering brukar bli kvar.

Ett mönster som väljer noll filer. En omdöpt katalog eller en flyttad konfiguration kan lämna en linter riktad mot en tom mängd, och att granska noll filer ger ett rent resultat. ESLint behandlar som standard ett citerat glob-mönster som matchar noll filer som ett fel. Flaggan `--no-error-on-unmatched-pattern` gör det till ett tyst godkännande, och `--pass-on-no-patterns` avslutar med 0 när inga filmönster anges. Behåll standarden, och behåll en fil med en känd överträdelse som grinden måste rapportera.

En regeluppsättning vald efter vad som var tyst. En konfiguration som byggts genom att stänga av allt som klagade första dagen dokumenterar kodens vanor, inte en standard. Börja från ett medvetet urval, antingen allt på en gång eller Ruffs föreslagna lilla bas av `E` och `F` med en grupp i taget tillagd, och skriv

ned varför varje avstängd regel är avstängd. Ruff varnar för att `ALL` i det tysta slår på nya regler varje gång verktyget uppgraderas, så lås linterns version och uppgradera den som en egen ändring.

Undertryckningar sprider sig en rad i taget. Varje linter har en nödutgång: `/* LINTED */` i lint(1), `# noqa: F841` i Ruff, `# shellcheck disable=SC2059` i ShellCheck och `// eslint-disable-next-line` i ESLint. Var och en registrerar vad som tystades, inte varför. Kräv ett skriftligt skäl bredvid varje undantag, och för de undantag som spelar roll: stäng av regeln och lägg till ett test för den egenskap som den skyddade. Det hjälper också att hålla undantagen på ett ställe. ESLints `--no-inline-config` gör kommentarer som stänger av regler verkningslösa, och ShellCheck läser `disable` -rader för hela projektet från en `.shellcheckrc` -fil.

Läs mer

- lint(1) — NetBSD:s manualsida för det C-verktyg som ordet kommer från: dess två pass, `-w` och dess kommentarer för undertryckning.
- The Ruff Linter — val av regler, hur säkra rättningarna är, kommentarer för undertryckning och slutkoder.
- Command Line Interface Reference — ESLints `--max-warnings` , konfiguration på raden, mönster utan träff och slutkoder.
- ShellCheck — projektets README: vad verktyget letar efter och hur det passar in i ett bygge.
- Ignore — ShellChecks sätt att ignorera ett fynd per rad, per körning och per projekt.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en linter?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/linter/