Engineer

Ruff

🖌 Linter y formateador rápidos para Python

Qué es

Ruff es un único programa de línea de comandos, escrito en Rust, que comprueba el código fuente de Python frente a un catálogo de reglas de lint y uniformiza su formato. Lo respalda Astral, la empresa detrás de uv y ty, y dos subcomandos realizan el trabajo cotidiano: `ruff check` es el linter y `ruff format`, el formateador. Afirma ejecutarse entre 10 y 100 veces más rápido que Flake8 y Black, conserva una caché para no volver a analizar archivos sin cambios y puede sustituir a Flake8 con decenas de sus complementos, Black, isort, pydocstyle, pyupgrade y autoflake.

El linter incluye más de 900 reglas, muchas de ellas reimplementaciones nativas de complementos populares de Flake8. Cada regla tiene un código con un prefijo breve y tres dígitos. El prefijo indica su procedencia: F401, una importación sin usar, viene de Pyflakes, y E501, una línea demasiado larga, de pycodestyle. La configuración selecciona reglas por esos códigos, una a una o por prefijo. Así, `select = ["E", "F"]` activa todas las reglas cuyo código empieza por E o F, y el selector especial `ALL` las activa todas. Muchas reglas incluyen una corrección, clasificada como segura si pretende conservar el comportamiento en ejecución, o como no segura si puede modificarlo. `ruff check --fix` solo aplica las seguras, salvo que también se indique `--unsafe-fixes`.

El formateador está diseñado como sustituto directo de Black. En bases de código como Django y Zulip, más del 99,9 % de las líneas quedan idénticas, aunque la documentación aclara que no están pensados para alternarse sobre el mismo código. Como Black, omite las regiones delimitadas por `# fmt: off` y `# fmt: on`.

Ambas partes leen la misma configuración, situada bajo `[tool.ruff]` en `pyproject.toml`, o en un archivo `ruff.toml` o `.ruff.toml`. Cada archivo Python se comprueba con la configuración más cercana que encuentre en los directorios superiores, y la usa entera: Ruff no combina nada de los directorios padre, salvo que el archivo más cercano indique una configuración superior con `extend`. Las opciones de línea de comandos prevalecen sobre cualquier archivo. Si no se especifica otra cosa, se permiten líneas de hasta 88 caracteres, como en Black. Cuando ninguna configuración indica la versión de Python, las reglas toman la que establece el campo `requires-python` del proyecto.

Utilice el concepto en una decisión

Aplique «¿Qué es Ruff?» 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

La velocidad decide dónde puede ejecutarse una comprobación. Un linter lo bastante lento como para interrumpir a alguien acaba relegado a la integración continua, donde informa cuando quien escribió el código ya está trabajando en otra cosa. Uno que responde rápido puede ejecutarse en el editor y en un hook de commit, cuando el cambio todavía está reciente.

La consolidación elimina desacuerdos entre herramientas. Un proyecto que comprueba el código con Flake8 y sus complementos, ordena las importaciones con isort y lo formatea con Black debe mantener coherentes varias configuraciones y versiones fijadas. Ruff ofrece una configuración, una versión que fijar y una única numeración de reglas. Además, publica la lista de reglas de lint que entran en conflicto con su propio formateador, de modo que se puedan hacer compatibles de forma deliberada.

Sus códigos de salida lo convierten en un control obligatorio sin necesidad de un programa intermediario. `ruff check` termina con 0 si no quedan infracciones o se han corregido todas, con 1 si quedan infracciones y con 2 si la configuración o la línea de comandos no son válidas, o si falla el propio Ruff, como hacen ESLint y

Prettier. `ruff format --check` termina con 1 si algún archivo necesitaría reformatearse. Una regla está seleccionada o desactivada, y cualquier regla seleccionada que deje una infracción sin corregir fija el código de salida en 1.

El coste es que alguien debe hacerse responsable de decidir el conjunto de reglas. El propio Ruff aconseja enumerarlas explícitamente con `select`, en lugar de acumular `extend-select`, empezar por E y F y añadir un grupo cada vez. También recomienda usar `ALL` con criterio, porque activa reglas nuevas en cada actualización.

Cómo se aplica en la práctica

Ruff se encargaba de Python en un criterio de cero advertencias aplicado a seis lenguajes, donde cada lenguaje tenía su propio linter. Todo diagnóstico contaba como error y se prohibían las supresiones en línea, lo que en Ruff excluye el comentario `# noqa`: un problema en Python solo podía resolverse corrigiendo el código. Los hooks pre-commit y pre-push ejecutaban los linters y las pruebas, de modo que la comprobación se hacía antes de que existiera el commit.

En la etapa de herramientas documentada de octubre de 2026, este sitio asignaba a Ruff una tarea más pequeña: los 22 scripts auxiliares que acompañaban a la compilación de Hugo. La página de créditos incluye Ruff entre las herramientas que verifican el código fuente. El conjunto de reglas de `pyproject.toml` se basa en mediciones, no en una selección arbitraria. El 2026-10-15 se ejecutó cada una de las 54 familias de reglas de Ruff sobre esos scripts. De ellas, 39 no informaron de nada, y las familias sin hallazgos cuyo ámbito correspondía a este código se añadieron a las once originales, hasta alcanzar una selección de 33. La familia de seguridad, S, sigue activa con cinco reglas excluidas mediante sus identificadores, cada una junto a un comentario que explica por qué el patrón señalado aquí es la variante segura de uno peligroso. Así, las demás reglas de esa familia siguen actuando como control obligatorio. Tres reglas propias de Ruff sobre caracteres Unicode ambiguos se ignoran porque el texto del sitio utiliza deliberadamente rayas, semirrayas y letras danesas.

La configuración de pre-commit ejecuta `ruff check` y `ruff format --check` sobre cualquier archivo Python bajo `scripts/` que toque un commit. La tarea de corrección del proyecto ejecuta los mismos dos comandos en su modalidad de corrección. Junto a Ruff está basedpyright, un comprobador de tipos, con un

comentario que marca el límite entre ambos: un linter no sigue un valor a través del límite de una función. En su primera ejecución, el comprobador de tipos encontró dos defectos reales que Ruff había dado por buenos.

Dónde surgen los problemas

La mayoría de los problemas con Ruff surge porque interpreta la configuración de forma más literal de lo que pretendían quienes la escribieron.

Una actualización hace fallar una rama que nadie ha tocado. Un código que ayer pasaba presenta nuevas infracciones tras actualizar Ruff. La causa habitual es `select = ["ALL"]`, que incorpora todas las reglas añadidas por cada versión, junto con una versión sin fijar. Fije la versión de Ruff para que una actualización sea un commit revisado y enumere las familias que necesita el proyecto en lugar de `ALL`, de modo que una familia nueva solo entre cuando alguien la añada.

Una carpeta deja de seguir las reglas del proyecto. Infracciones prohibidas por la configuración raíz pasan en un subdirectorio, o allí se activa una regla que el equipo había desactivado. En esa carpeta hay un `ruff.toml`, o un `pyproject.toml` con una tabla `[tool.ruff]`. Como Ruff usa entero el archivo más cercano, todas las opciones de la raíz dejan de aplicarse debajo de él. `ruff check --show-settings` sobre un archivo de esa carpeta muestra la ruta de configuración que ha resuelto. Borre el archivo anidado o haga que herede mediante `extend = "../pyproject.toml"`.

Una supresión sobrevive a su motivo. Se añadió un comentario `# noqa` por una buena razón, el código cambió y ahora el comentario no oculta nada, o esconde el siguiente problema de esa línea. Un `# noqa` sin código de regla es peor: silencia todas las reglas de la línea. Seleccione RUF100, que informa de comentarios `noqa` que ya no suprimen nada, y PGH004, que exige un código concreto, y deje que `--fix` elimine los obsoletos. Aquí la importación ya ha empezado a utilizarse, por lo que el comentario no protege nada y RUF100 lo señala:

```
import json  # noqa: F401

print(json.dumps({"ok": True}))
```

El formateador y el linter discrepan. `ruff format` reescribe un archivo y después `ruff check` señala el resultado. Algunas reglas de lint controlan decisiones del formateador, como las comas finales (COM812), la sangría (W191, E111) y el estilo de comillas (las reglas Q). Además, el formateador solo ajusta las líneas a la longitud fijada hasta donde puede, por lo que E501 puede activarse sobre código formateado. Lea las advertencias que imprime `ruff format` : según la propia documentación, una ejecución sin ellas es la prueba de que ambos están de acuerdo. Ignore las reglas que indique una advertencia y trate E501 como una señal para reestructurar la línea.

Una corrección masiva cambia el comportamiento. Tras ejecutar `ruff check --fix --unsafe-fixes` en todo el árbol, fallan las pruebas o un manejador de errores deja de capturar lo que antes capturaba. Las correcciones no seguras pueden cambiar el comportamiento en ejecución o eliminar comentarios. El ejemplo de la propia documentación cambia la excepción que lanza un fragmento de código. Mantenga `--unsafe-fixes` fuera de hooks y scripts, aplique las correcciones no seguras de una regla cada vez con `--select` y lea el diff. Reclasifique una regla como segura con `extend-safe-fixes` solo después de que el equipo la haya comprobado.

Lecturas adicionales

- Ruff — Qué sustituye Ruff, qué velocidad afirma ofrecer y quién lo mantiene.
- El linter de Ruff — Selección y precedencia de reglas, correcciones seguras y no seguras, comentarios `noqa` y códigos de salida.
- El formateador de Ruff — Compatibilidad con Black y reglas de lint que entran en conflicto con el formateo.
- Reglas — Cada regla por su código, su procedencia y si admite corrección.
- Configuración de Ruff — Búsqueda de archivos, regla del archivo más cercano, `extend` y opciones de línea de comandos.

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 Ruff?» 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/ruff/