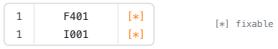
Ruff

An overview of Ruff, the extremely fast linter and formatter that replaces a stack of tools and fixes your code in one pass.

Lint (ruff check)

ruff check scans your code and lists violations, fast.

PURPOSE	COMMAND	RESULT
Lint the whole project.	`ruff check`	
Lint specific paths.	`ruff check src/ tests/`	
Show a clean run.	`ruff check (no issues)`	
Count violations per rule.	`ruff check --statistics`	
Watch and re-lint on save.	`ruff check --watch`	
Pick the output format.	`ruff check --output-format=github`	

ruff check lists each violation as CODE message with file:line and exits non-zero; via uv: uv run ruff check

Fix

Apply the safe fixes automatically; opt in to the rest.

PURPOSE	COMMAND	RESULT
Apply all safe fixes.	`ruff check --fix`	
Preview fixes without writing.	`ruff check --diff`	
List what got fixed.	`ruff check --fix --show-fixes`	
Include unsafe fixes.	`ruff check --fix --unsafe-fixes`	
Fix only, do not report leftovers.	`ruff check --fix-only`	
Restrict which rules may fix.	`ruff check --fix --fixable I`	

--fix rewrites every safe violation; preview with --diff, read --unsafe-fixes before applying, scope with --fixable

Format

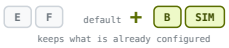
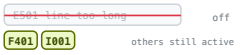
A Black-compatible formatter built into the same binary.

PURPOSE	COMMAND	RESULT
Format the project in place.	`ruff format`	
Check formatting (no writes, CI).	`ruff format --check`	
Show the formatting diff.	`ruff format --diff`	
Format specific files.	`ruff format app.py src/`	
Sort imports in the same pass.	`ruff check --select I --fix` then `ruff format`	

ruff format owns whitespace, quotes, and line wrapping; import sorting stays the I lint rule (ruff check --select I --fix)

Select & Ignore Rules

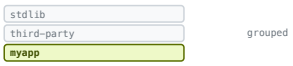
Turn rule families on and off by their codes.

PURPOSE	COMMAND	RESULT
Choose exactly which rules run.	`ruff check --select E,F,I,UP,B,SIM`	
Add to the default set.	`ruff check --extend-select B,SIM`	
Silence a specific rule.	`ruff check --ignore E501`	
Enable everything, then trim.	`ruff check --select ALL`	
Turn on preview (unstable) rules.	`ruff check --preview`	
Treat a rule as fixable or not.	`ruff check --fix --unfixable F401`	

--select sets the active rules, --extend-select adds to the config, --ignore drops one; rule codes are letter-prefixed by source linter

Configure (pyproject.toml)

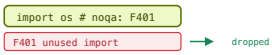
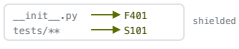
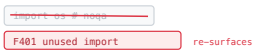
Persist settings under [tool.ruff] so the team shares them.

PURPOSE	COMMAND	RESULT
Set the line length.	`[tool.ruff]` `line-length = 100`	
Pin the target Python version.	`target-version = "py313"`	
Select and ignore rules in config.	`[tool.ruff.lint]` `select = ["E","F","I","UP","B"]`	
Configure import sorting.	`[tool.ruff.lint.isort]` `known-first-party = ["myapp"]`	
Set formatter options.	`[tool.ruff.format]` `quote-style = "double"`	
Use a standalone ruff.toml.	`ruff.toml # drop [tool.ruff] prefix`	

[tool.ruff] holds project options; [tool.ruff.lint] / [tool.ruff.format] hold rule and formatter settings; a bare ruff.toml works too

Suppress

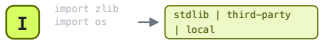
Carve out exceptions without disabling a rule everywhere.

PURPOSE	COMMAND	RESULT
Ignore a code on one line.	`import os # noqa: F401`	
Blanket-ignore a whole line.	`weird_code() # noqa`	
Ignore rules for a path glob.	`[tool.ruff.lint.per-file-ignores]` `"tests/**" = ["S101"]`	
Auto-insert noqa for existing code.	`ruff check --add-noqa`	
Audit by ignoring all noqa.	`ruff check --ignore-noqa`	
Explain why a code fires.	`ruff rule F401`	

noqa suppresses inline, per-file-ignores by glob; --add-noqa writes them, --ignore-noqa audits them, ruff rule explains the code

Rule-code categories

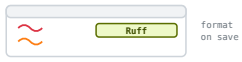
800+ rules grouped by a short letter prefix per source linter.

PURPOSE	COMMAND	RESULT
Pyflakes correctness (the core).	`(prefix) F`	
pycodestyle style.	`(prefix) E, W`	
Import sorting (isort).	`(prefix) I`	
Modernize syntax (pyupgrade).	`(prefix) UP`	
Likely bugs (flake8-bugbear).	`(prefix) B`	
Simplify and de-pep8.	`(prefix) SIM, N`	

Every rule code starts with a prefix naming its upstream linter; ruff linter lists them all, and you build your set by combining the prefixes you trust

Integrate (editor & pre-commit)

Lint and format on save and on every commit.

PURPOSE	COMMAND	RESULT
Run the editor language server.	`ruff server`	
Run ephemerally with uv.	`uvx ruff check`	
Add the pre-commit hooks.	`.pre-commit-config.yaml` `repo: astral-sh/ruff-pre-commit`	
Auto-fix in the commit hook.	`id: ruff-check` `args: [--fix]`	
Gate CI on lint + format.	`ruff check --output-format=github` `ruff format --check`	
Pin the version everywhere.	`rev: v0.14.10`	

ruff server powers editors; astral-sh/ruff-pre-commit gates commits; uvx ruff runs zero-install; pin one rev across editor, pre-commit, and CI