flake8 Documentation

Release 2.6.0

Tarek Ziade

June 16, 2016

Contents

Flake8 is a wrapper around these tools:

- PyFlakes
- pep8
- Ned Batchelder's McCabe script

Flake8 runs all the tools by launching the single `flake8` script. It displays the warnings in a per-file, merged output.

It also adds a few features:

- files that contain this line are skipped:

```
# flake8: noqa
```

- lines that contain a `# noqa` comment at the end will not issue warnings.
- a Git and a Mercurial hook.
- a McCabe complexity checker.
- extendable through `flake8.extension` entry points.

CHAPTER 1

QuickStart

```
pip install flake8
```

To run flake8 just invoke it against any directory or Python module:

```
$ flake8 coolproject
coolproject/mod.py:97:1: F401 'shutil' imported but unused
coolproject/mod.py:625:17: E225 missing whitespace around operato
coolproject/mod.py:729:1: F811 redefinition of function 'readlines' from line 723
coolproject/mod.py:1028:1: F841 local variable 'errors' is assigned to but never used
```

The outputs of PyFlakes *and* pep8 (and the optional plugins) are merged and returned.

flake8 offers an extra option: –max-complexity, which will emit a warning if the McCabe complexity of a function is higher than the value. By default it's deactivated:

```
$ flake8 --max-complexity 12 coolproject
coolproject/mod.py:97:1: F401 'shutil' imported but unused
coolproject/mod.py:625:17: E225 missing whitespace around operator
coolproject/mod.py:729:1: F811 redefinition of unused 'readlines' from line 723
coolproject/mod.py:939:1: C901 'Checker.check_all' is too complex (12)
coolproject/mod.py:1028:1: F841 local variable 'errors' is assigned to but never used
coolproject/mod.py:1204:1: C901 'selftest' is too complex (14)
```

This feature is quite useful to detect over-complex code. According to McCabe, anything that goes beyond 10 is too complex. See https://en.wikipedia.org/wiki/Cyclomatic_complexity.

CHAPTER 2

Frequently Asked Questions

2.1 Why does flake8 pin the version of pep8?

Version 1.6 of pep8 doesn't work properly with flake8. Until pep8 releases a version that works, flake8 pins the version of pep8 so that flake8 will work as a whole.

2.2 Is flake8 broken?

Flake8 combines two other projects that are significant on their own: pep8 and PyFlakes. If flake8 is doing something you don't like, it is quite likely that the problem lies in one of those other projects. You can run them separately to see if they are the cause of your difficulties. We greatly appreciate your efforts to diagnose the source of the problem before reporting bugs against flake8.

CHAPTER 3

Questions or Feedback

If you have questions you'd like to ask the developers, or feedback you'd like to provide, feel free to use the mailing list: code-quality@python.org We would love to hear from you. Additionally, if you have a feature you'd like to suggest, the mailing list would be the best place for it.

CHAPTER 4

Links

- flake8 documentation
- pep8 documentation

CHAPTER 5

Documentation

5.1 Configuration

Configuration settings are applied in three ways: user, project, and the `--config` CLI argument. The user (global) configuration is read first. Next the project configuration is loaded, and overrides any settings found in both the user (global) and project configurations. Finally, if the `--config` argument is used on the command line, the specified file is loaded and overrides any settings that overlap with the user (global) and project configurations.

5.1.1 User (Global)

The user settings are read from the `~/.config/flake8` file (or the `~/.flake8` file on Windows). Example:

```
[flake8]
ignore = E226,E302,E41
max-line-length = 160
exclude = tests/*
max-complexity = 10
```

5.1.2 Per-Project

At the project level, the `tox.ini`, `setup.cfg`, `.pep8` or `.flake8` files are read if present. Only the first file is considered. If this file does not have a `[flake8]` section, no project specific configuration is loaded.

5.1.3 Per Code Line

To ignore one line of code add `# NOQA` as a line comment.

5.1.4 Default

If the `ignore` option is not in the configuration and not in the arguments, only the error codes `E123/E133`, `E226` and `E241/E242` are ignored (see the *warning and error codes*).

5.1.5 Settings

This is a (likely incomplete) list of settings that can be used in your config file. In general, any settings that `pycodestyle` supports we also support and we add the ability to set `max-complexity` as well.

- `exclude`: comma-separated filename and glob patterns default: `.svn,CVS,.bzr,.hg,.git,__pycache__`
- `filename`: comma-separated filename and glob patterns default: `*.py`
- `select`: select errors and warnings to enable which are off by default
- `ignore`: skip errors or warnings
- `max-line-length`: set maximum allowed line length default: 79
- `format`: set the error format
- `max-complexity`: McCabe complexity threshold

5.2 Warning / Error codes

The convention of Flake8 is to assign a code to each error or warning, like the `pycodestyle` tool. These codes are used to configure the list of errors which are selected or ignored.

Each code consists of an upper case ASCII letter followed by three digits. The recommendation is to use a different prefix for each plugin. A list of the known prefixes is published below:

- `E***`/`W***`: pycodestyle errors and warnings
- `F***`: PyFlakes codes (see below)
- `C9**`: McCabe complexity plugin mccabe
- `N8**`: Naming Conventions plugin pep8-naming

The original PyFlakes does not provide error codes. Flake8 patches the PyFlakes messages to add the following codes:

code	sample message
F401	`module` imported but unused
F402	import `module` from line `N` shadowed by loop variable
F403	'from `module` import *' used; unable to detect undefined names
F404	future import(s) `name` after other statements
F405	`name` may be undefined, or defined from star imports: `module`
F811	redefinition of unused `name` from line `N`
F812	list comprehension redefines `name` from line `N`
F821	undefined name `name`
F822	undefined name `name` in __all__
F823	local variable `name` ... referenced before assignment
F831	duplicate argument `name` in function definition
F841	local variable `name` is assigned to but never used

5.3 VCS Hooks

flake8 can install hooks for Mercurial and Git so that flake8 is run automatically before commits. The commit will fail if there are any flake8 issues.

You can install the hook by issuing this command in the root of your project:

```
$ flake8 --install-hook
```

In the case of Git, the hook won't be installed if a custom `pre-commit` hook file is already present in the `.git/hooks` directory.

You can control the behavior of the pre-commit hook using configuration file settings or environment variables:

`flake8.complexity` or `FLAKE8_COMPLEXITY` Any value > 0 enables complexity checking with McCabe. (defaults to 10)

`flake8.strict` or `FLAKE8_STRICT` If True, this causes the commit to fail in case of any errors at all. (defaults to False)

`flake8.ignore` or `FLAKE8_IGNORE` Comma-separated list of errors and warnings to ignore. (defaults to empty)

`flake8.lazy` or `FLAKE8_LAZY` If True, also scans those files not added to the index before commit. (defaults to False)

You can set these either through the git command line

```
$ git config flake8.complexity 10
$ git config flake8.strict true
```

Or by directly editing `.git/config` and adding a section like

```
[flake8]
    complexity = 10
    strict = true
    lazy = false
```

5.4 Buildout integration

In order to use Flake8 inside a buildout, edit your buildout.cfg and add this:

```
[buildout]

parts +=
    ...
    flake8

[flake8]
recipe = zc.recipe.egg
eggs = flake8
       ${buildout:eggs}
entry-points =
    flake8=flake8.main:main
```

5.5 Setuptools integration

Upon installation, Flake8 enables a setuptools command that checks Python files declared by your project.

Running `python setup.py flake8` on the command line will check the files listed in your `py_modules` and `packages`. If any warning is found, the command will exit with an error code:

```
$ python setup.py flake8
```

Also, to allow users to be able to use the command without having to install flake8 themselves, add flake8 to the setup_requires of your setup() like so:

```
setup(
    name="project",
    packages=["project"],

    setup_requires=[
        "flake8"
    ]
)
```

5.6 Flake8 API

5.6.1 flake8.engine

flake8.engine.**get_parser**()
This returns an instance of optparse.OptionParser with all the extensions registered and options set. This wraps pep8.get_parser.

flake8.engine.**get_style_guide**(***kwargs*)
Parse the options and configure the checker. This returns a sub-class of pep8.StyleGuide.

5.6.2 flake8.hooks

flake8.hooks.**git_hook**(*complexity=-1*, *strict=False*, *ignore=None*, *lazy=False*)
This is the function used by the git hook.

Parameters

- **complexity** (*int*) – (optional), any value > 0 enables complexity checking with mccabe
- **strict** (*bool*) – (optional), if True, this returns the total number of errors which will cause the hook to fail
- **ignore** (*str*) – (optional), a comma-separated list of errors and warnings to ignore
- **lazy** (*bool*) – (optional), allows for the instances where you don't add the files to the index before running a commit, e.g., git commit -a

Returns total number of errors if strict is True, otherwise 0

flake8.hooks.**hg_hook**(*ui*, *repo*, ***kwargs*)
This is the function executed directly by Mercurial as part of the hook. This is never called directly by the user, so the parameters are undocumented. If you would like to learn more about them, please feel free to read the official Mercurial documentation.

5.6.3 flake8.main

flake8.main.**main**()
Parse options and run checks on Python source.

flake8.main.**check_file**(*path*, *ignore=()*, *complexity=-1*)
Checks a file using pep8 and pyflakes by default and mccabe optionally.

Parameters

- **path** (*str*) – path to the file to be checked

- **ignore** (*tuple*) – (optional), error and warning codes to be ignored
- **complexity** (*int*) – (optional), enables the mccabe check for values > 0

flake8.main.**check_code**(*code*, *ignore=()*, *complexity=-1*)
Checks code using pep8 and pyflakes by default and mccabe optionally.

Parameters

- **code** (*str*) – code to be checked
- **ignore** (*tuple*) – (optional), error and warning codes to be ignored
- **complexity** (*int*) – (optional), enables the mccabe check for values > 0

class flake8.main.**Flake8Command**(*dist*, ***kw*)
The Flake8Command class is used by setuptools to perform checks on registered modules.

5.6.4 flake8.util

For AST checkers, this module has the iter_child_nodes function and handles compatibility for all versions of Python between 2.5 and 3.3. The function was added to the ast module in Python 2.6 but is redefined in the case where the user is running Python 2.5

5.7 Writing an Extension for Flake8

Since Flake8 is now adding support for extensions, we require setuptools so we can manage extensions through entry points. If you are making an existing tool compatible with Flake8 but do not already require setuptools, you should probably add it to your list of requirements. Next, you'll need to edit your setup.py file so that upon installation, your extension is registered. If you define a class called PackageEntryClass then this would look something like the following:

```
setup(
    # ...
    entry_points={
        'flake8.extension': ['P10 = package.PackageEntryClass'],
    }
    # ...
)
```

If you intend to publish your extension, choose a unique code prefix following the convention for *error codes*. In addition, you can open a request in the issue tracker to register the prefix in the documentation.

5.7.1 A real example: McCabe

Below is an example from mccabe for how to write your setup.py file for your Flake8 extension.

```
# https://github.com/flintwork/mccabe/blob/0.2/setup.py#L38:L42
# -*- coding: utf-8 -*-
from setuptools import setup

# ...

setup(
    name='mccabe',
```

```
    # ...

    install_requires=[
        'setuptools',
    ],
    entry_points={
        'flake8.extension': [
            'C90 = mccabe:McCabeChecker',
        ],
    },

    # ...

)
```

In `mccabe.py` you can see that extra options are added to the parser when flake8 registers the extension:

```
# https://github.com/flintwork/mccabe/blob/0.2/mccabe.py#L225:L254
class McCabeChecker(object):
    """McCabe cyclomatic complexity checker."""
    name = 'mccabe'
    version = __version__
    _code = 'C901'
    _error_tmpl = "C901 %r is too complex (%d)"
    max_complexity = 0

    def __init__(self, tree, filename):
        self.tree = tree

    @classmethod
    def add_options(cls, parser):
        parser.add_option('--max-complexity', default=-1, action='store',
                          type='int', help="McCabe complexity threshold")
        parser.config_options.append('max-complexity')

    @classmethod
    def parse_options(cls, options):
        cls.max_complexity = options.max_complexity

    def run(self):
        if self.max_complexity < 0:
            return
        visitor = PathGraphingAstVisitor()
        visitor.preorder(self.tree, visitor)
        for graph in visitor.graphs.values():
            if graph.complexity() >= self.max_complexity:
                text = self._error_tmpl % (graph.entity, graph.complexity())
                yield graph.lineno, 0, text, type(self)
```

Since that is the defined entry point in the above `setup.py`, flake8 finds it and uses it to register the extension.

If we wanted the extension or a check to be optional, you can add `off_by_default = True` to our entry point. For example, we could update `mccabe.py` with this variable as shown below:

```
# https://github.com/flintwork/mccabe/blob/0.2/mccabe.py#L225:L254
class McCabeChecker(object):
    """McCabe cyclomatic complexity checker."""
    name = 'mccabe'
    version = __version__
```

```
    off_by_default = True
```

If we wanted to run the optional extension or check, we need to specify the error and warnings via the `--enable-extension` command line argument. In our case, we could run `flake8 --enable-extension=C90` which would enable our off_by_default example version of the mccabe extension.

5.8 Existing Extensions

This is not at all a comprehensive listing of existing extensions but simply a listing of the ones we are aware of:

- flake8-debugger
- flake8-immediate
- flake8-print
- flake8-todo
- pep8-naming
- radon
- flake8-import-order
- flake8-respect-noqa

5.9 Changes

5.9.1 2.6.0 - 2016-06-15

- **Requirements Change** Switch to pycodestyle as all future pep8 releases will use that package name
- **Improvement** Allow for Windows users on *select* versions of Python to use `--jobs` and multiprocessing
- **Improvement** Update bounds on McCabe
- **Improvement** Update bounds on PyFlakes and blacklist known broken versions
- **Improvement** Handle new PyFlakes warning with a new error code: F405

5.9.2 2.5.5 - 2016-06-14

- **Bug** Fix setuptools integration when parsing config files
- **Bug** Don't pass the user's config path as the config_file when creating a StyleGuide

5.9.3 2.5.4 - 2016-02-11

- **Bug** Missed an attribute rename during the v2.5.3 release.

5.9.4 2.5.3 - 2016-02-11

- **Bug** Actually parse `output_file` and `enable_extensions` from config files

5.9.5 2.5.2 - 2016-01-30

- **Bug** Parse `output_file` and `enable_extensions` from config files
- **Improvement** Raise upper bound on mccabe plugin to allow for version 0.4.0

5.9.6 2.5.1 - 2015-12-08

- **Bug** Properly look for `.flake8` in current working directory (GitLab#103)
- **Bug** Monkey-patch `pep8.stdin_get_value` to cache the actual value in stdin. This helps plugins relying on the function when run with multiprocessing. (GitLab#105, GitLab#107)

5.9.7 2.5.0 - 2015-10-26

- **Improvement** Raise cap on PyFlakes for Python 3.5 support
- **Improvement** Avoid deprecation warnings when loading extensions (GitLab#59, GitLab#90)
- **Improvement** Separate logic to enable "off-by-default" extensions (GitLab#67)
- **Bug** Properly parse options to setuptools Flake8 command (GitLab!41)
- **Bug** Fix exceptions when output on stdout is truncated before Flake8 finishes writing the output (GitLab#69)
- **Bug** Fix error on OS X where Flake8 can no longer acquire or create new semaphores (GitLab#74)

5.9.8 2.4.1 - 2015-05-18

- **Bug** Do not raise a `SystemError` unless there were errors in the setuptools command. (GitLab#39, GitLab!23)
- **Bug** Do not verify dependencies of extensions loaded via entry-points.
- **Improvement** Blacklist versions of pep8 we know are broken

5.9.9 2.4.0 - 2015-03-07

- **Bug** Print filenames when using multiprocessing and `-q` option. (GitLab#31)
- **Bug** Put upper cap on dependencies. The caps for 2.4.0 are:
  - `pep8 < 1.6` (Related to GitLab#35)
  - `mccabe < 0.4`
  - `pyflakes < 0.9`

  See also GitLab#32
- **Bug** Files excluded in a config file were not being excluded when flake8 was run from a git hook. (GitHub#2)
- **Improvement** Print warnings for users who are providing mutually exclusive options to flake8. (GitLab#8, GitLab!18)
- **Feature** Allow git hook configuration to live in `.git/config`. See the updated VCS hooks docs for more details. (GitLab!20)

5.9.10 2.3.0 - 2015-01-04

- **Feature**: Add `--output-file` option to specify a file to write to instead of `stdout`.
- **Bug** Fix interleaving of output while using multiprocessing (GitLab#17)

5.9.11 2.2.5 - 2014-10-19

- Flush standard out when using multiprocessing
- Make the check for "# flake8: noqa" more strict

5.9.12 2.2.4 - 2014-10-09

- Fix bugs triggered by turning multiprocessing on by default (again)

  Multiprocessing is forcibly disabled in the following cases:

  - Passing something in via stdin
  - Analyzing a diff
  - Using windows
- Fix –install-hook when there are no config files present for pep8 or flake8.
- Fix how the setuptools command parses excludes in config files
- Fix how the git hook determines which files to analyze (Thanks Chris Buccella!)

5.9.13 2.2.3 - 2014-08-25

- Actually turn multiprocessing on by default

5.9.14 2.2.2 - 2014-07-04

- Re-enable multiprocessing by default while fixing the issue Windows users were seeing.

5.9.15 2.2.1 - 2014-06-30

- Turn off multiple jobs by default. To enable automatic use of all CPUs, use `--jobs=auto`. Fixes #155 and #154.

5.9.16 2.2.0 - 2014-06-22

- New option `doctests` to run Pyflakes checks on doctests too
- New option `jobs` to launch multiple jobs in parallel
- Turn on using multiple jobs by default using the CPU count
- Add support for `python -m flake8` on Python 2.7 and Python 3
- Fix Git and Mercurial hooks: issues #88, #133, #148 and #149
- Fix crashes with Python 3.4 by upgrading dependencies

- Fix traceback when running tests with Python 2.6
- Fix the setuptools command `python setup.py flake8` to read the project configuration

5.9.17 2.1.0 - 2013-10-26

- Add FLAKE8_LAZY and FLAKE8_IGNORE environment variable support to git and mercurial hooks
- Force git and mercurial hooks to repsect configuration in setup.cfg
- Only check staged files if that is specified
- Fix hook file permissions
- Fix the git hook on python 3
- Ignore non-python files when running the git hook
- Ignore .tox directories by default
- Flake8 now reports the column number for PyFlakes messages

5.9.18 2.0.0 - 2013-02-23

- Pyflakes errors are prefixed by an `F` instead of an `E`
- McCabe complexity warnings are prefixed by a `C` instead of a `W`
- Flake8 supports extensions through entry points
- Due to the above support, we **require** setuptools
- We publish the documentation
- Fixes #13: pep8, pyflakes and mccabe become external dependencies
- Split run.py into main.py, engine.py and hooks.py for better logic
- Expose our parser for our users
- New feature: Install git and hg hooks automagically
- By relying on pyflakes (0.6.1), we also fixed #45 and #35

5.9.19 1.7.0 - 2012-12-21

- Fixes part of #35: Exception for no WITHITEM being an attribute of Checker for Python 3.3
- Support stdin
- Incorporate @phd's builtins pull request
- Fix the git hook
- Update pep8.py to the latest version

5.9.20 1.6.2 - 2012-11-25

- fixed the NameError: global name 'message' is not defined (#46)

5.9.21 1.6.1 - 2012-11-24

- fixed the mercurial hook, a change from a previous patch was not properly applied
- fixed an assumption about warnings/error messages that caused an exception to be thrown when McCabe is used

5.9.22 1.6 - 2012-11-16

- changed the signatures of the `check_file` function in flake8/run.py, `skip_warning` in flake8/util.py and the `check`, `checkPath` functions in flake8/pyflakes.py.
- fix `--exclude` and `--ignore` command flags (#14, #19)
- fix the git hook that wasn't catching files not already added to the index (#29)
- pre-emptively includes the addition to pep8 to ignore certain lines. Add `# nopep8` to the end of a line to ignore it. (#37)
- `check_file` can now be used without any special prior setup (#21)
- unpacking exceptions will no longer cause an exception (#20)
- fixed crash on non-existent file (#38)

5.9.23 1.5 - 2012-10-13

- fixed the stdin
- make sure mccabe catches the syntax errors as warnings
- pep8 upgrade
- added max_line_length default value
- added Flake8Command and entry points if setuptools is around
- using the setuptools console wrapper when available

5.9.24 1.4 - 2012-07-12

- git_hook: Only check staged changes for compliance
- use pep8 1.2

5.9.25 1.3.1 - 2012-05-19

- fixed support for Python 2.5

5.9.26 1.3 - 2012-03-12

- fixed false W402 warning on exception blocks.

5.9.27 1.2 - 2012-02-12

- added a git hook
- now Python 3 compatible
- mccabe and pyflakes have warning codes like pep8 now

5.9.28 1.1 - 2012-02-14

- fixed the value returned by –version
- allow the flake8: header to be more generic
- fixed the "hg hook raises 'physical lines'" bug
- allow three argument form of raise
- now uses setuptools if available, for 'develop' command

5.9.29 1.0 - 2011-11-29

- Deactivates by default the complexity checker
- Introduces the complexity option in the HG hook and the command line.

5.9.30 0.9 - 2011-11-09

- update pep8 version to 0.6.1
- mccabe check: gracefully handle compile failure

5.9.31 0.8 - 2011-02-27

- fixed hg hook
- discard unexisting files on hook check

5.9.32 0.7 - 2010-02-18

- Fix pep8 initialization when run through Hg
- Make pep8 short options work when run through the command line
- Skip duplicates when controlling files via Hg

5.9.33 0.6 - 2010-02-15

- Fix the McCabe metric on some loops

CHAPTER 6

Original Projects

Flake8 is just a glue project, all the merits go to the creators and maintainers of the original projects:

- pycodestyle: https://github.com/pycqa/pycodestyle
- PyFlakes: https://launchpad.net/pyflakes
- McCabe: http://nedbatchelder.com/blog/200803/python_code_complexity_microtool.html

f

C

F

G

H

M