nbQA Documentation

Release 1.3.1

Marco Gorelli

Apr 25, 2022

USER DOCS:


```{include} ../README.md
```

CHAPTER

ONE

TUTORIAL

Welcome! Here's a little tutorial, assuming you're brand-new here. We'll walk you through:

- creating a virtual environment;
- installing `nbqa`, and checking your installation;
- running `black` on your notebook;
- configuring `nbqa`.

1.1 Creating a virtual environment

Rather than using your system installation of Python, we recommend using a virtual environment so that your dependencies don't clash with each other. Here's one way to set one up using Conda - see this tutorial for other options.

1. Install the Miniconda distribution of Python;
2. Create a new virtual environment. Here, we'll call it `nbqa-env`

```
conda create -n nbqa-env python=3.8 -y
```

3. Activate your virtual environment

```
conda activate nbqa-env
```

1.2 Install nbqa and black

1. Install `nbqa` and at least one Python code quality tool - here, we'll use `black`

```
pip install -U nbqa black
```

2. Check your installation

```
nbqa --version
black --version
```

Neither of these commands should error.

1.3 Run nbqa black

1. Locate a Jupyter Notebook on your system. If you don't have one, here is a nice one you can download.
2. Run the `black` formatter on your notebook via `nbqa`

```
nbqa black notebook.ipynb --line-length=96
```

3. Reload your notebook, and admire the difference!

1.4 Configuring nbqa

Rather than having to type `--line-length=96` from the command-line for each notebook you want to reformat, you can configure `nbqa` in your `pyproject.toml` file. Open up your `pyproject.toml` file (or create one if you don't have one already) and add in the following lines

```
[tool.black]
line-length = 96
```

Now, you'll be able to run the command from the previous section with just

```
nbqa black notebook.ipynb
```

Much simpler!

See *configuration* for how to further configure how `nbqa`.

CHAPTER

TWO

COMMAND-LINE EXAMPLES

Note: Cell numbers in the output refer to code cells, enumerated starting from 1.

Reformat your notebooks with black:

```
$ nbqa black my_notebook.ipynb
reformatted my_notebook.ipynb
All done!
1 files reformatted.
```

Run your docstring tests with doctest:

```
$ nbqa doctest my_notebook.ipynb
**********************************************************************
File "my_notebook.ipynb", cell_2:11, in my_notebook.add
Failed example:
    add(2, 2)
Expected:
    4
Got:
    5
**********************************************************************
1 items had failures:
1 of   2 in my_notebook.hello
***Test Failed*** 1 failures.
```

Check for style guide enforcement with flake8:

```
$ nbqa flake8 my_notebook.ipynb --extend-ignore=E203,E302,E305,E703
my_notebook.ipynb:cell_3:1:1: F401 'import pandas as pd' imported but unused
```

Sort your imports with isort:

```
$ nbqa isort my_notebook.ipynb
Fixing my_notebook.ipynb
```

Check your type annotations with mypy:

```
$ nbqa mypy my_notebook.ipynb --ignore-missing-imports
my_notebook.ipynb:cell_10:5: error: Argument "num1" to "add" has incompatible type "str";
↪ expected "int"
```

Perform static code analysis with pylint:

```
$ nbqa pylint my_notebook.ipynb --disable=C0114
my_notebook.ipynb:cell_1:5:0: W0611: Unused import datetime (unused-import)
```

Upgrade your syntax with pyupgrade:

```
$ nbqa pyupgrade my_notebook.ipynb --py36-plus
Rewriting my_notebook.ipynb
```

Format code with yapf:

```
$ nbqa yapf --in-place my_notebook.ipynb
```

Format code with autopep8:

```
$ nbqa autopep8 -i my_notebook.ipynb
```

Check docstring style with pydocstyle:

```
$ nbqa pydocstyle my_notebook.ipynb
```

Format markdown cells with mdformat:

```
$ nbqa mdformat my_notebook.ipynb --nbqa-md
```

CHAPTER

THREE

CONFIGURATION

You can configure nbQA either at the command line, or by using a pyproject.toml file. We'll see some examples below.

Note: Please note that if you pass the same option via both the pyproject.toml file and via the command-line, the command-line will take precedence.

3.1 Preview / CI

To preview changes without modifying your notebook, using the --nbqa-diff flag. The return code will be 1 if nbQA would've modified any of your notebooks, and 0 otherwise.

Note: You should not use -nbqa-diff alongside tools such as flake8 which only check your code. Instead, use it with formatters such as isort.

3.2 Extra flags

If you wish to pass extra flags (e.g. --extend-ignore E203 to flake8) you can either run

```
nbqa flake8 my_notebook.ipynb --extend-ignore E203
```

or you can put the following in your pyproject.toml file

```
[tool.nbqa.addopts]
flake8 = [
    "--extend-ignore=E203"
]
```

Note: If you specify extra flags via both the pyproject.toml file and the command-line, both will be passed on to the underlying command-line tool, with the options specified in pyproject.toml passed first. In this case the exact behaviour will depend on the tool and the option in question. It's common that subsequent flags override earlier ones, but check the documentation for the tool and option in question to be sure.

3.3 Cell magics

By default, nbQA will ignore line magics (e.g. `%matplotlib inline`), as well as most cell magics. To process code in cells with cell magics, you can use the `--nbqa-process-cells` CLI argument. E.g. to process code within `%%add_to` cell magics, use

```
nbqa black my_notebook.ipynb --nbqa-process-cells add_to
```

or use the `process_cells` option in your `pyproject.toml` file:

```
[tool.nbqa.process_cells]
black = ["add_to"]
```

3.4 Include / exclude

To include or exclude notebooks from being processed, we recommend using nbQA's own `--nbqa-files` and `--nbqa-exclude` flags. These take regex patterns and match posix-like paths, exactly like the same options from pre-commit. These can be set from the command-line with the `--nbqa-files` and `--nbqa-exclude` flags, or in your `.pyproject.toml` file in the `[tool.nbqa.files]` and `[tool.nbqa.exclude]` sections.

Say you're running `nbqa isort` on a directory `my_directory`. Here are some examples of how to include/exclude files:

- exclude notebooks in `my_directory` whose name starts with `poc_`:

  ```
  [tool.nbqa.exclude]
  isort = "^my_directory/poc_"
  ```

- exclude notebooks in subdirectory `my_directory/my_subdirectory`:

  ```
  [tool.nbqa.exclude]
  isort = "^my_directory/my_subdirectory/"
  ```

- only include notebooks in `my_directory` whose name starts with `EDA`:

  ```
  [tool.nbqa.files]
  isort = "^my_directory/EDA"
  ```

All the above examples can equivalently be run from the command-line, e.g. as

```
nbqa isort my_directory --nbqa-exclude ^my_directory/poc_
```

3.5 Don't skip bad cells

By default, nbQA will skip cells with invalid syntax. To process cells with syntax errors, you can use the `--nbqa-dont-skip-bad-cells` CLI argument.

This can be set from the command-line with the `--nbqa-dont-skip-bad-cells` flag, or in your `.pyproject.toml` file in the `[tool.nbqa.dont_skip_bad_cells]` section.

For example, to process "bad" cells when running `black` on `notebook.ipynb`, you could add the following to your `pyproject.toml` file:

```
[tool.nbqa.dont_skip_bad_cells]
black = true
```

or, from the command-line:

```
nbqa black notebook.ipynb --nbqa-dont-skip-bad-cells
```

3.6 Skip cells based on celltags

You can skip cells based on the celltags in their metadata using the `--nbqa-skip-celltags` CLI argument.

For example, to skip cells which contain either the `skip-flake8` or `flake8-skip` tags, you could add the following to your `pyproject.toml` file:

```
[tool.nbqa.skip_celltags]
black = ["skip-flake8", "flake8-skip"]
```

or, from the command-line:

```
nbqa black notebook.ipynb --nbqa-skip-celltags=skip-flake8,flake8-skip
```

3.7 Process markdown cells

You can process markdown cells (instead of code cells) by using the `--nbqa-md` CLI argument.

This is useful when running tools which run on markdown files, such as `mdformat`.

For example, you could add the following to your `pyproject.toml` file:

```
[tool.nbqa.md]
mdformat = true
```

or, from the command-line:

```
nbqa mdformat notebook.ipynb --nbqa-md
```

CHAPTER

FOUR

PRE-COMMIT

4.1 Usage

You can easily use `nbqa` as a pre-commit hook.

Here's an example of what you could include in your `.pre-commit-config.yaml` file:

```
repos:
- repo: https://github.com/nbQA-dev/nbQA
  rev: 1.3.1
  hooks:
    - id: nbqa-black
      additional_dependencies: [black==20.8b1]
    - id: nbqa-pyupgrade
      additional_dependencies: [pyupgrade==2.7.3]
    - id: nbqa-isort
      additional_dependencies: [isort==5.6.4]
```

For best reproducibility, you should pin your dependencies (as above). Running `pre-commit autoupdate` will update your hooks' versions, but versions of additional dependencies need to updated manually.

See .pre-commit-hooks.yaml for all available built-in hooks.

4.2 Custom hooks

If you have your own custom tool (e.g. `customtool`) for which we currently don't have a built-in hook, you can define your own one with:

```
- repo: https://github.com/nbQA-dev/nbQA
  rev: 1.3.1
  hooks:
    - id: nbqa
      entry: nbqa customtool
      name: nbqa-customtool
      alias: nbqa-customtool
      additional_dependencies: [customtool==<version number>]
```

If there are additional Python code quality tools you would like us to make a hook for, please *open a pull request* or let us know in the issue tracker!

4.3 Configuration

See *configuration* for how to further configure how `nbqa` should run each tool. Also, see the pre-commit documentation for how to further configure these hooks.

4.4 Temporarily disable hooks

Although not recommended, it is still possible to temporarily **disable all checks** using `git commit --no-verify`, or **just specific ones** using the `SKIP` environment variable. For example, on a Unix-like operating system:

```
SKIP=nbqa-black git commit -m "foo"
```

For more details, please check out the pre-commit documentation.

CHAPTER

FIVE

KNOWN LIMITATIONS

By default, `nbQA` will skip cells with invalid syntax. If you choose to process cells with invalid syntax via the `--nbqa-dont-skip-bad-cells` flag (see *configuration*), then the following will still not be processed:

- cells with multi-line magics;
- automagics (ideas for how to detect them statically are welcome!);
- cells with code which `IPython` would transform magics into (e.g. `get_ipython().system('ls')`).

Any other limitation is likely unintentional - if you run into any, please do report an issue.

CHAPTER

SIX

CHANGELOG

6.1 1.3.1 (2022-03-09)

Fixed setup.cfg to mark than only Python3.7+ is supported.

6.2 1.3.0 (2022-03-05)

Removed support for Python 3.6. Exit code for `--nbqa-diff` is now `1` is file would've been modified, and `0` otherwise. Cells with just line magics will now be ignored.

6.3 1.2.3 (2022-01-11)

Removed (unnecessary) upper-bound on `tomli`.

6.4 1.2.2 (2021-11-25)

Fixed bug whereby constructs like `var = %env var` would not round-trip (thanks Daniel Sparing for the report!). No longer add a trailing newline to notebooks which didn't have one originally.

6.5 1.2.1 (2021-11-21)

Fixed bug whereby `blacken-docs` would need to be run as `nbqa blacken_docs`, because the package name and the Python module of `blacken-docs` differ. Now, it can be run as `nbqa blacken-docs`, as users would expect.

6.6 1.2.0 (2021-11-20)

Added support for formatting markdown cells via the `--nbqa-md` flag. Changed aesthetics of `--nbqa-diff` to be in line with how `black` does it.

6.7 1.1.1 (2021-09-12)

Fixed bug whereby local module wasn't being picked up correctly. Removed `language_version` from `.pre-commit-hooks.yaml`.

6.8 1.1.0 (2021-08-01)

Added support for `autopep8` and `pydocstyle`. Fixed bug whereby `nbqa mypy` wasn't showing coloured output. Added several extra internal robustness checks.

6.9 1.0.0 (2021-07-21)

Removed `--nbqa-mutate` flag, it's no longer necessary. Options passed to `--nbqa-addopts` are now combined with those in `pyproject.toml`, as opposed to overriding them.

6.10 0.13.1 (2021-06-25)

Fixed bug whereby local scripts / modules could not be run by `nbQA` due to incorrect `ModuleNotFoundError`.

6.11 0.13.0 (2021-06-15)

BREAKING CHANGE: by default, cells with invalid syntax will now be skipped. To retain the old behaviour, use `--nbqa-dont-skip-bad-cells` (see documentation for details / examples). Added ability to skip cells based on celltags.

6.12 0.12.0 (2021-06-10)

`nbQA` will no longer halt execution if it encounters notebooks which it can't parse or which fail to reconstruct - instead, such errors will be reported all at once at the end. The exit code in such cases will be `123` (as in the `black` formatter). Fixed bug whereby DataBricks notebooks (which are saved differently than Jupyter notebooks) with empty cells were not being reconstructed properly when using `nbqa-diff`. Cell magics are now parsed more robustly.

6.13 0.11.1 (2021-06-07)

Fixed historic limitation whereby cells with assignment to line magics or to system outputs were being ignored. Removed `autoflake` hook (at least until I am confident that magics are fully supported).

6.14 0.10.0 (2021-05-30)

Introduced `--nbqa-skip-bad-cells` flag. Cells with multi-line magics are no longer processed.

6.14.1 0.9.0 (2021-05-23)

Fixed bug whereby percent format sign was being mistaken for a line IPython magic. `nbQA` is now intentionally more timid about processing magics, and cells with unusual magics will be ignored.

6.14.2 0.8.1 (2021-05-15)

If output from tool cannot be parsed from Python lines to notebook code cells, then a `KeyError` is no longer thrown and the original output is printed (thanks Tony Hirst for the bug report!).

6.14.3 0.8.0 (2021-05-02)

Output from linters will now typically display relative paths where possible, else absolute ones. Flags `--nbqa-ignore` and `--nbqa-config` have been removed. Fixed regression (introduced in 0.7.1) whereby if a series of notebooks was passed and one of them did not exist, then the temporary files associated with the first ones would not get cleaned up.

6.14.4 0.7.1 (2021-04-28)

Fixed regression (introduced in 0.7.0) whereby `nbqa-flake8` wasn't reporting error messages with cell numbers if absolute path of notebook was used.

6.14.5 0.7.0 (2021-04-18)

Fixed historic known limitation of `nbqa-black` removing trailing semicolons when they were followed by comments. Fixed bug whereby local modules were not properly being picked up by `nbqa-mypy` (thanks Rafal Wojdyla for the excellent bug report!). Added support for `yapf` (thanks Bradley Dice for the suggestion + PR!). Added support for Python3.6.0 (previously was 3.6.1+).

6.14.6 0.6.1 (2021-04-16)

Fixed bug whereby notebooks with dots in their names were not being processed correctly (thank you Ivan Cheung for the issue!)

6.14.7 0.6.0 (2021-04-04)

Processing cell-magics is now opt-in rather than opt-out. Original output from tool is always printed with `--nbqa-diff`.

6.14.8 0.5.9 (2021-02-22)

Nothing, just fixing up the previous tag, sorry for the inconvenience caused. xref https://github.com/pre-commit-ci/issues/issues/45

6.14.9 0.5.8 (2021-02-20)

Fixed bug in which `mypy` wasn't finding local imports due to `MYPYPATH` not being carried over by `nbqa`.

6.14.10 0.5.7 (2021-01-26)

Fixed bug whereby `pyupgrade` wasn't working with empty notebook due to `nbQA` adding newlines to the end of the file even if the file was empty.

6.14.11 0.5.6 (2020-12-29)

Fixed bug whereby `flake8` with the `wemake-python-styleguide` plugin was throwing false-positives about magic number being present when they weren't.

6.14.12 0.5.5 (2020-12-10)

Improved error parsing when `nbqa black` finds code which can't be parsed (e.g. assignment to a literal). You can now once again install all supported code-quality tools with `python -m pip install -U nbqa[toolchain]` (thanks Sebastian Weigand!).

6.14.13 0.5.4 (2020-12-06)

Fixed bug whereby notebooks starting with comments were being uncommented out when replacing notebooks (thanks Nathan Cooper for filing the issue!).

6.14.14 0.5.3 (2020-12-04)

Fixed bug whereby commented-out cell magics were preventing `nbqa` from reconstructing notebooks properly (thanks John Sandall for filing the issue!).

6.14.15 0.5.2 (2020-11-30)

Fixed bug whereby `nbqa` was throwing `UnicodeDecodeError` on Windows (thanks Simon Brugman for noticing the issue and for submitting a fix!).

6.14.16 0.5.1 (2020-11-25)

Fixed bugs whereby `nbqa` wasn't handling incomplete IPython magics, nor was it handling assignments to help magics (thanks Girish Pasupathy for noticing and fixing both of these!).

6.14.17 0.5.0 (2020-11-22)

Fixed bug whereby formatters weren't parsing assignments to shell magic. Raise error if given config file doesn't exist. Added `-nbqa-diff` flag, which allows users to preview changes before applying them. Added `nbqa-autoflake` pre-commit hook.

6.14.18 0.4.1 (2020-11-11)

Fixed bug whereby parsing notebooks without any code cells was throwing `IndexError`. Fixed bug whereby piping output to a text file was introducing extra newlines on Windows. Added `nbqa-check-ast` pre-commit hook. Added `--nbqa-files` and `--nbqa-exclude` flags for file inclusion/exclusion.

6.14.19 0.4.0 (2020-11-05)

Added support for inline magics (thanks Girish Pasupathy for this huge effort!). Raise `FileNotFoundError` if non-existent notebook/directory is passed. Fixed bug whereby `FileNotFoundError` was being raised if directory without notebooks in it was passed. Users are encouraged to report bugs if we can't parse output from code quality tool. Output from `black` refers to cell number rather than python line number if command fails. More informative message is raised if `nbqa` is called without a code quality tool and a notebook/directory. Added some more cell magics to list of cell magics ignored by default. No longer use emojis in our own error reporting. `.git`, `.venv`, and other common non-source-code directories are excluded from recursive search for notebooks. More tool-specific config files are preserved by default.

6.14.20 0.3.6 (2020-10-30)

Improved error reporting if file is not found. We now pass `--treat-comment-as-code '# %%'` by default when running `isort`. Fixed bug whereby tools referencing line 0 we resulting in a `KeyError`.

6.14.21 0.3.5 (2020-10-25)

Optimised how nbqa passes files so that pre-commit hooks run faster.

6.14.22 0.3.4 (2020-10-23)

Fixed bug whereby nbqa was giving the wrong error message when running `nbqa doctest` and the notebook contained a library which couldn't be imported.

6.14.23 0.3.3 (2020-10-21)

More precise error diagnostics if code-quality tool isn't found (thanks Girish Pasupathy!). You can now install all supported code-quality tools with `python -m pip install -U nbqa[toolchain]` (thanks Sebastian Weigand!). We handle a greater array of cell magics by default. We removed `nbqa-doctest` pre-commit hook, as this one's best run from the command line (thanks Sebastian Weigand!).

6.14.24 0.3.2 (2020-10-17)

In-built pre-commit hooks for `black`, `flake8`, `mypy`, `isort`, `pyupgrade`, `doctest`, and `pylint` are now available.

6.14.25 0.3.1 (2020-10-16)

Fixed bug whereby `nbqa` was using the system (or virtual environment) Python, rather than the Python used to install `nbqa`. This was causing issues when running `nbqa` outside of a virtual environment.

6.14.26 0.3.0 (2020-10-12)

Added support for `pylint` (thanks Girish Pasupathy!). Fixed a false-positive in `black` when cells ended with trailing semicolons. Fixed some false-positives in `flake8` regarding expected numbers of newlines.

6.14.27 0.2.3 (2020-10-06)

Output from third-party tools is more consistent with the path the user passes in. E.g. if the user passes a relative path, the output will show a relative path, whilst if the user passes an absolute path, the output will show an absolute path. Users are also now encouraged to report bugs if there are errors parsing / reconstructing notebooks.

6.14.28 0.2.2 (2020-10-01)

Optimised handling cell-magics and improved support for indented in-line magics (thanks Girish Pasupathy!).

6.14.29 0.2.1 (2020-09-27)

Fix bug in which cells with trailing semicolons followed by empty newlines were having semicolons added to the newline. Added support for `pyupgrade`.

6.14.30 0.2.0 (2020-09-26)

First somewhat stable release, with `flake8`, `black`, `isort`, `mypy`, and `doctest` supported, and configuration via `pyproject.toml`.

CHAPTER

SEVEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/nbQA-dev/nbQA/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

nbqa could always use more documentation, whether as part of the official nbqa docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nbQA-dev/nbQA/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up `nbqa` for local development.

1. Fork the `nbqa` repo on GitHub at https://github.com/nbQA-dev/nbQA.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/nbqa.git nbqa-dev
$ cd nbqa-dev
```

3. Create a virtual environment for local development:

```
$ tox --devenv venv
$ . venv/bin/activate
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ tox -e py
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.md.
3. The pull request should work for Python 3.6, 3.7, 3.8, and 3.9.

7.4 Add yourself as a contributor

To add yourself to the table of contributors, please follow the bot usage instructions.

Example of comment:

```
@all-contributors please add @<username> for documentation
```

Note: It's considered a good practice to always prefix usernames with `@`

7.5 Tips

Enable pre-commit to catch errors early-on:

```
pre-commit install
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search