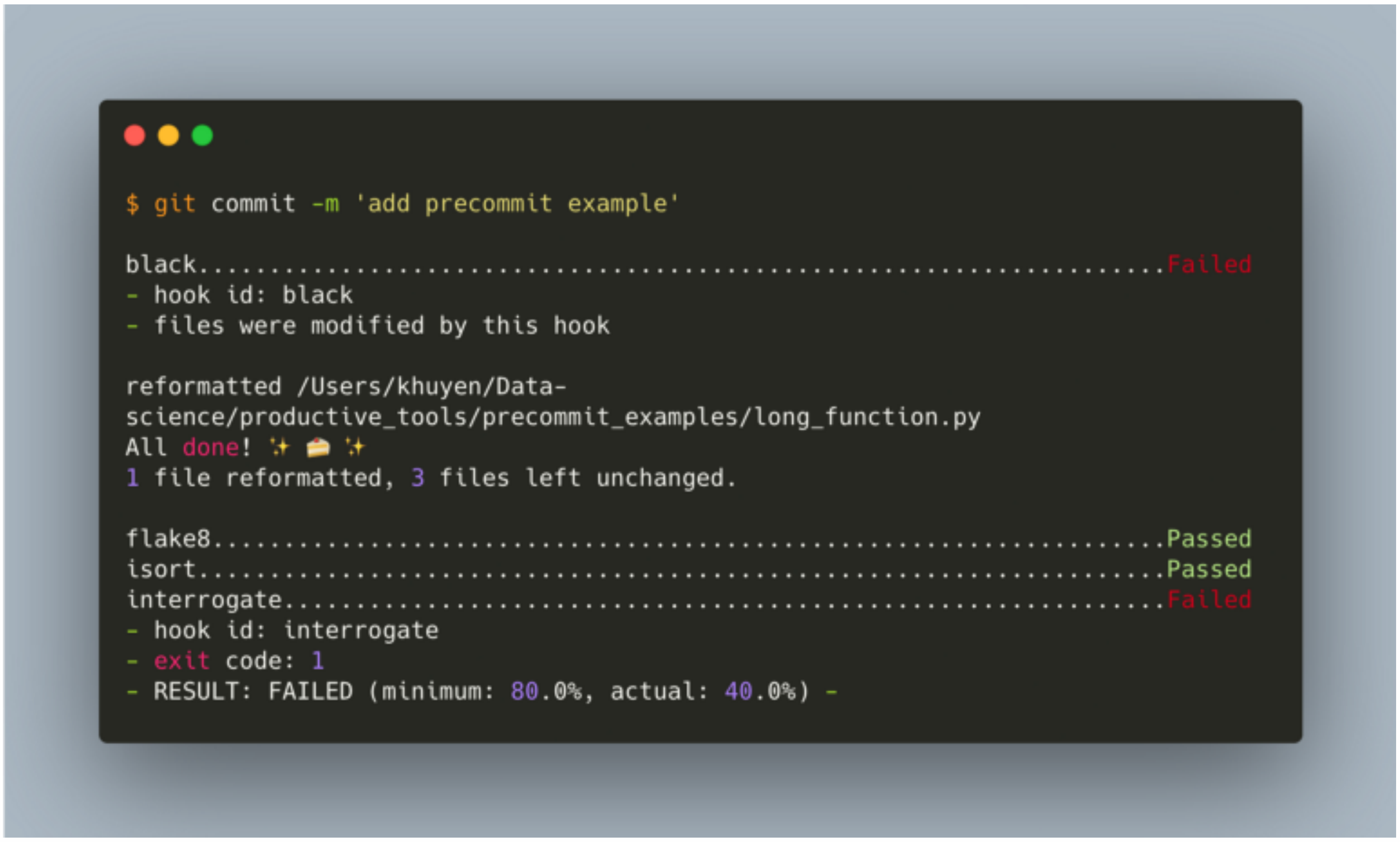
Write High-Quality Code with black, flake8, isort, and interrogate

When committing your Python code to Git, you need to make sure your code:

- looks nice
- is organized
- conforms to the PEP 8 style guide
- includes docstrings

However, it can be overwhelming to check all of these criteria before committing your code.

Wouldn’t it be nice if you can **automatically** check and format your code every time you commit new code like below?



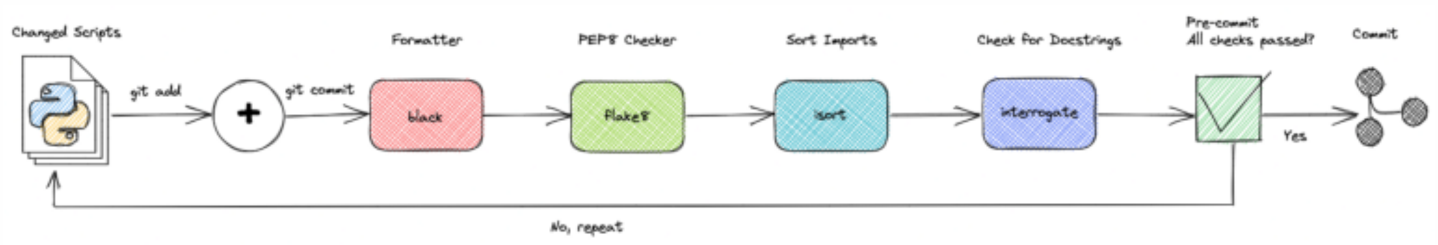
```
$ git commit -m 'add precommit example'

black.............................................................Failed
- hook id: black
- files were modified by this hook

reformatted /Users/khuyen/Data-
science/productive_tools/precommit_examples/long_function.py
All done! ✨ 🍰 ✨
1 file reformatted, 3 files left unchanged.

flake8............................................................Passed
isort.............................................................Passed
interrogate.......................................................Failed
- hook id: interrogate
- exit code: 1
- RESULT: FAILED (minimum: 80.0%, actual: 40.0%) -
```

That is when pre-commit comes in handy. In this section, you will learn what pre-commit is and which plugins you can add to a pre-commit pipeline.

What is pre-commit?

pre-commit is a framework that allows you to identify simple issues in your code before committing it.

You can add different plugins to your pre-commit pipeline. Once your files are committed, they will be checked by these plugins. Unless all checks pass, no code will be committed.



To install pre-commit, type:

```
pip install pre-commit
```

Cool! Now let's add some useful plugins to our pre-commit pipeline.

black

black is a code formatter in Python.

To install black, type:

```
pip install black
```

Now to see what black can do, we'll write a very long function like below. Since there are more than 79 characters in the first line of code, this violates PEP 8.

Let's try to format the code using black:

```
$ black long_function.py
```

And the code is automatically formatted like below!

```
def very_long_function(
    long_variable_name,
    long_variable_name2,
    long_variable_name3,
    long_variable_name4,
    long_variable_name5,
):
    pass
```

To add black to a pre-commit pipeline, create a file named `.pre-commit-config.yaml` and insert the following code to the file:

```yaml
repos:
-   repo: https://github.com/ambv/black
    rev: 20.8b1
    hooks:
    - id: black
```

To choose which files to include and exclude when running black, create a file named `pyproject.toml` and add the following code to the `pyproject.toml` file:

```toml
[tool.black]
line-length = 79
include = '\.pyi?$'
exclude = '''
/(
  \.git
| \.hg
| \.mypy_cache
| \.tox
| \.venv
| _build
| buck-out
| build
)/
'''
```

flake8

flake8 is a python tool that checks the style and quality of your Python code. It checks for various issues not covered by black.

To install flake8, type:

```
pip install flake8
```

To see what flake8 does, let's write code that violates some guidelines in PEP 8.

```python
def very_long_function_name(var1, var2, var3,
var4, var5):
    print(var1, var2, var3, var4, var5)

very_long_function_name(1, 2, 3, 4, 5)
```

Next, check the code using flake8:

```
$ flake8 flake_example.py
```

```
flake8_example.py:2:1: E128 continuation line under-
indented for visual indent
flake8_example.py:5:1: E305 expected 2 blank lines
after class or function definition, found 1
flake8_example.py:5:39: W292 no newline at end of file
```

Aha! flake8 detects 3 PEP 8 formatting errors. We can use these errors as the guidelines to fix the code.

```
def very_long_function_name(var1, var2, var3, var4,
var5):
    print(var1, var2, var3, var4, var5)


very_long_function_name(1, 2, 3, 4, 5)
```

The code looks much better now!

To add flake8 to the pre-commit pipeline, insert the following code to the `.pre-commit-config.yaml` file:

```
-   repo: https://gitlab.com/pycqa/flake8
    rev: 3.8.4
    hooks:
    - id: flake8
```

To choose which errors to ignore or to edit other configurations, create a file named `.flake8` and add the following code to the `.flake8` file:

```
[flake8]
ignore = E203, E266, E501, W503, F403, F401
max-line-length = 79
max-complexity = 18
select = B,C,E,F,W,T4,B9
```

isort

isort is a Python library that automatically sorts imported libraries alphabetically and separates them into sections and types.

To install isort, type:

```
pip install isort
```

Let’s try to use isort to sort messy imports like below:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from flake8_example import very_long_function_name
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression,
OrderedLogisticRegression, \
    LinearRegression, LogisticRegressionCV,
LinearRegressionCV
```

```
$ isort isort_example.py
```

Output:

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from flake8_example import very_long_function_name
from sklearn.linear_model import (
    LinearRegression,
    LinearRegressionCV,
    LogisticRegression,
    LogisticRegressionCV,
    OrderedLogisticRegression,
)
from sklearn.model_selection import train_test_split
```

Cool! The imports are much more organized now.

To add isort to the pre-commit pipeline, add the following code to the `.pre-commit-config.yaml` file:

```yaml
-   repo: https://github.com/timothycrosley/isort
    rev: 5.7.0
    hooks:
    -   id: isort
```

interrogate

interrogate checks your codebase for missing docstrings.

To install interrogate, type:

```
pip install interrogate
```

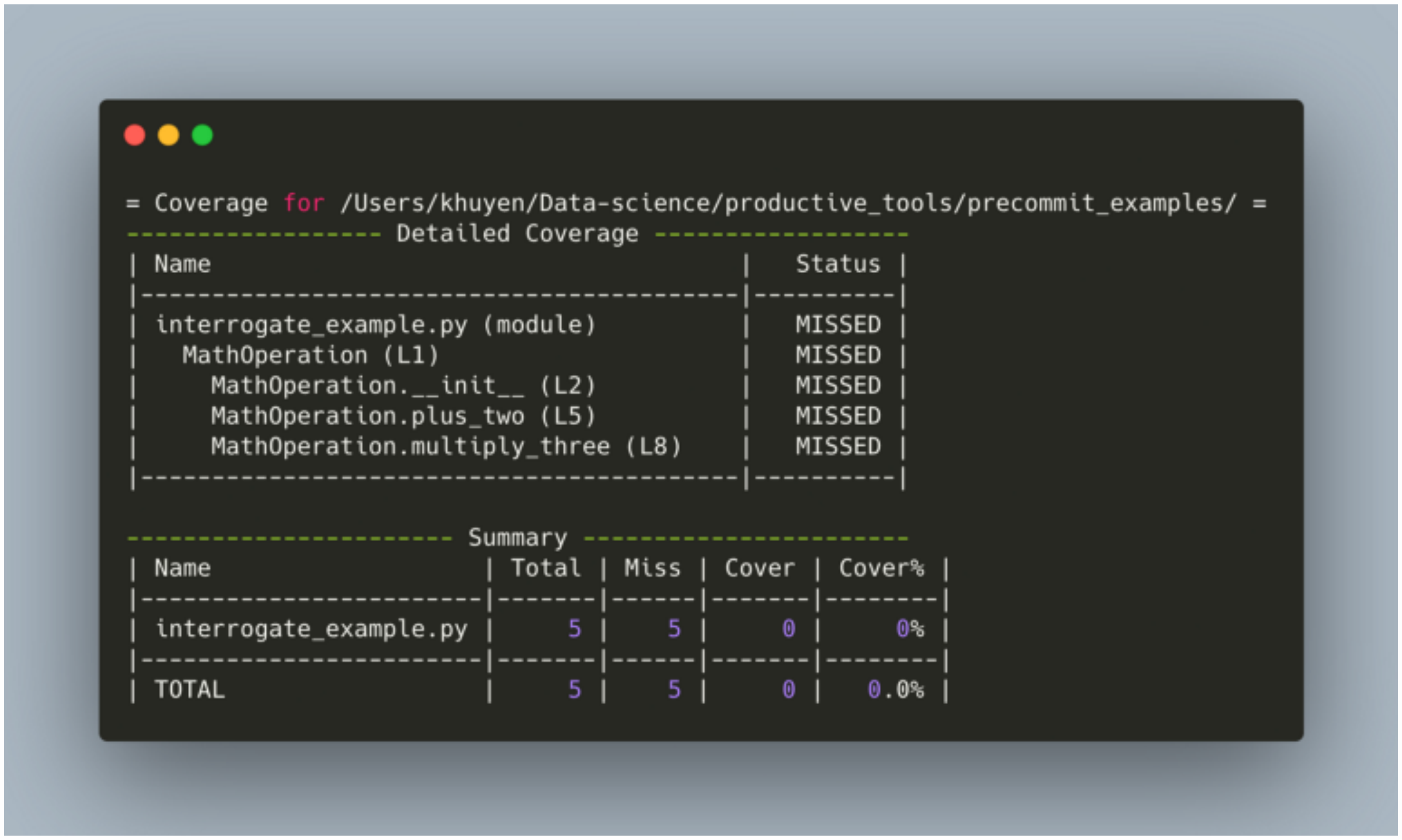
Sometimes, we might forget to write docstrings for classes and functions like below:

```python
class MathOperation:
    def __init__(self, num) -> None:
        self.num = num

    def plus_two(self):
        return self.num + 2

    def multiply_three(self):
        return self.num * 3
```

Instead of manually looking at all our functions and classes for missing docstrings, we can run interrogate instead:

```
$ interrogate -vv interrogate_example.py
```

Output:

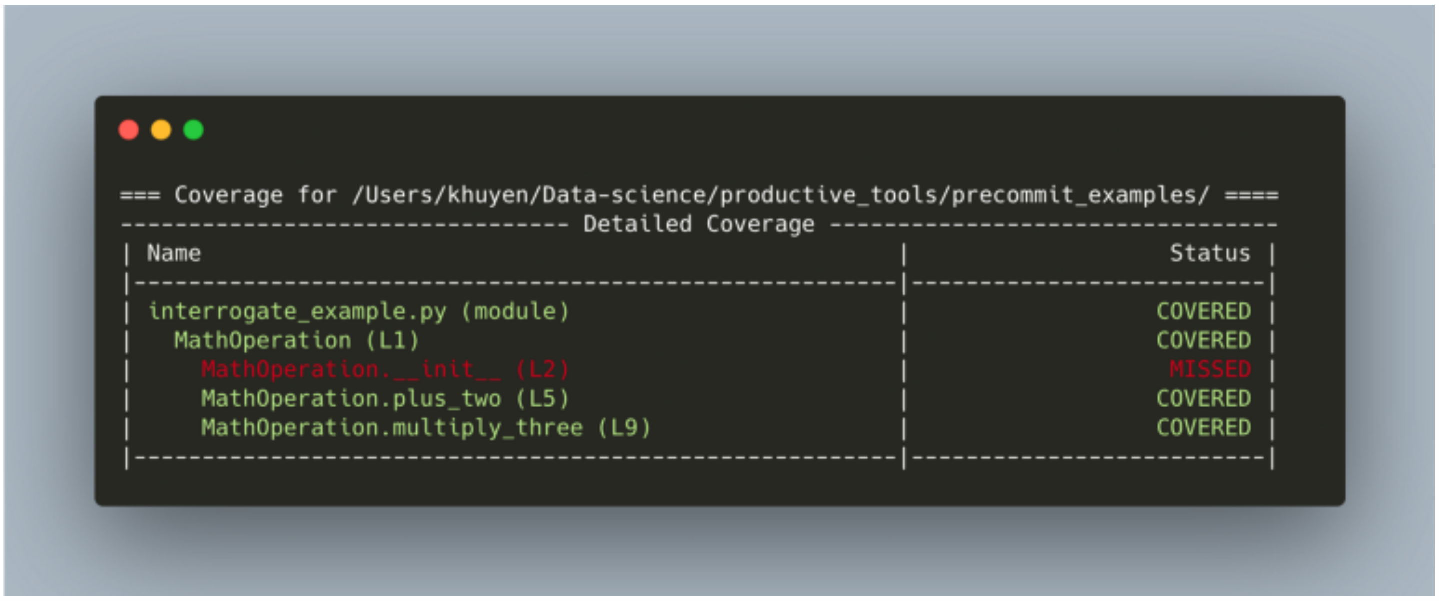


```
= Coverage for /Users/khuyen/Data-science/productive_tools/precommit_examples/ =
------------------ Detailed Coverage ------------------
| Name                                         |   Status |
|----------------------------------------------|----------|
| interrogate_example.py (module)              |   MISSED |
|   MathOperation (L1)                         |   MISSED |
|     MathOperation.__init__ (L2)              |   MISSED |
|     MathOperation.plus_two (L5)              |   MISSED |
|     MathOperation.multiply_three (L8)        |   MISSED |
|----------------------------------------------|----------|

----------------------- Summary -----------------------
| Name                   | Total | Miss | Cover | Cover% |
|------------------------|-------|------|-------|--------|
| interrogate_example.py |     5 |    5 |     0 |     0% |
|------------------------|-------|------|-------|--------|
| TOTAL                  |     5 |    5 |     0 |   0.0% |
```

Cool! From the terminal output, we know which files, classes, and functions don't have docstrings. Since we know the locations of missing docstrings, adding them is easy.

```python
"""Example for interrogate"""

class MathOperation:
    """Perform math operation"""
    def __init__(self, num) -> None:
        self.num = num

    def plus_two(self):
        """Add 2"""
        return self.num + 2

    def multiply_three(self):
        """Multiply by 3"""
        return self.num * 3
```

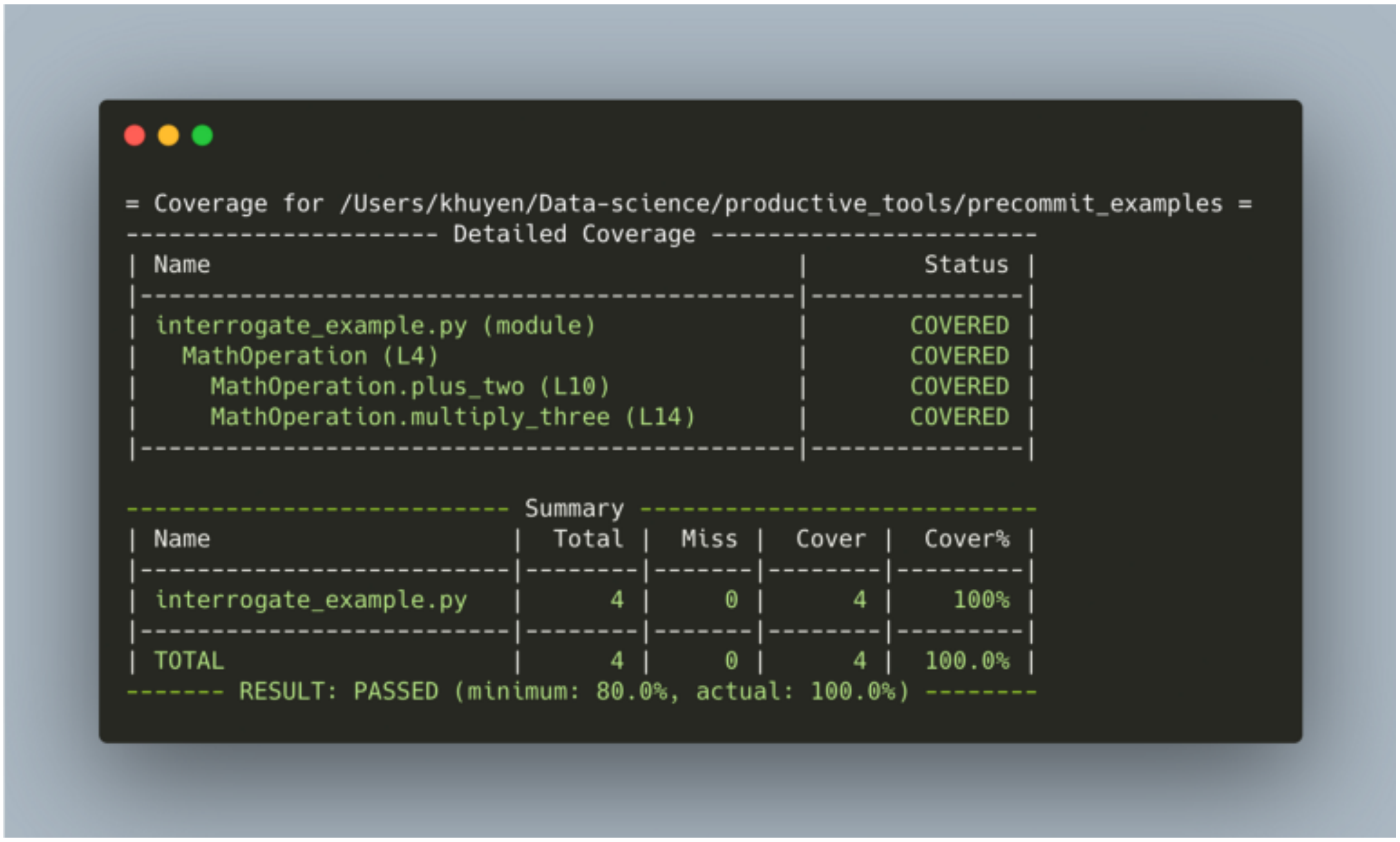
```
$ interrogate -vv interrogate_example.py
```



```
=== Coverage for /Users/khuyen/Data-science/productive_tools/precommit_examples/ ====
--------------------------------- Detailed Coverage ---------------------------------
| Name                                                   |                    Status |
|--------------------------------------------------------|---------------------------|
| interrogate_example.py (module)                        |                   COVERED |
|   MathOperation (L1)                                   |                   COVERED |
|     MathOperation.__init__ (L2)                        |                    MISSED |
|     MathOperation.plus_two (L5)                        |                   COVERED |
|     MathOperation.multiply_three (L9)                  |                   COVERED |
|--------------------------------------------------------|---------------------------|
```

The docstring for the `__init__` method is missing, but it is not necessary. We can tell interrogate to ignore the `__init__` method by adding `-i` to the argument:

```
$ interrogate -vv -i interrogate_example.py
```



```
= Coverage for /Users/khuyen/Data-science/productive_tools/precommit_examples =
---------------------- Detailed Coverage ----------------------
| Name                                          |        Status |
|-----------------------------------------------|---------------|
| interrogate_example.py (module)               |       COVERED |
|   MathOperation (L4)                          |       COVERED |
|     MathOperation.plus_two (L10)              |       COVERED |
|     MathOperation.multiply_three (L14)        |       COVERED |
|-----------------------------------------------|---------------|

-------------------------- Summary --------------------------
| Name                      |  Total |   Miss |  Cover |  Cover% |
|---------------------------|--------|--------|--------|---------|
| interrogate_example.py    |      4 |      0 |      4 |    100% |
|---------------------------|--------|--------|--------|---------|
| TOTAL                     |      4 |      0 |      4 |  100.0% |
------- RESULT: PASSED (minimum: 80.0%, actual: 100.0%) --------
```

Cool! To add interrogate to the pre-commit pipeline, insert the following code to the `.pre-commit-config.yaml` file:

```yaml
- repo: https://github.com/econchick/interrogate
    rev: 1.4.0
    hooks:
      - id: interrogate
        args: [--vv, -i, --fail-under=80]
```

To edit interrogate's default configurations, insert the following code to the `pyproject.toml` file:

```toml
[tool.interrogate]
ignore-init-method = true
ignore-init-module = false
ignore-magic = false
ignore-semiprivate = false
ignore-private = false
ignore-property-decorators = false
ignore-module = true
ignore-nested-functions = false
ignore-nested-classes = true
ignore-setters = false
fail-under = 95
exclude = ["setup.py", "docs", "build"]
ignore-regex = ["^get$", "^mock_.*", ".*BaseClass.*"]
verbose = 0
quiet = false
whitelist-regex = []
color = true
generate-badge = "."
badge-format = "svg"
```

Final Step — Add pre-commit to Git Hooks

The final code in your `.pre-commit-config.yaml` file should look like below:

```yaml
repos:
-   repo: https://github.com/ambv/black
    rev: 20.8b1
    hooks:
    - id: black
-   repo: https://gitlab.com/pycqa/flake8
    rev: 3.8.4
    hooks:
    - id: flake8
-   repo: https://github.com/timothycrosley/isort
    rev: 5.7.0
    hooks:
    -   id: isort
-   repo: https://github.com/econchick/interrogate
    rev: 1.4.0
    hooks:
    - id: interrogate
      args: [-vv, -i, --fail-under=80]
```

To add pre-commit to git hooks, type:

```
$ pre-commit install
```

Output:

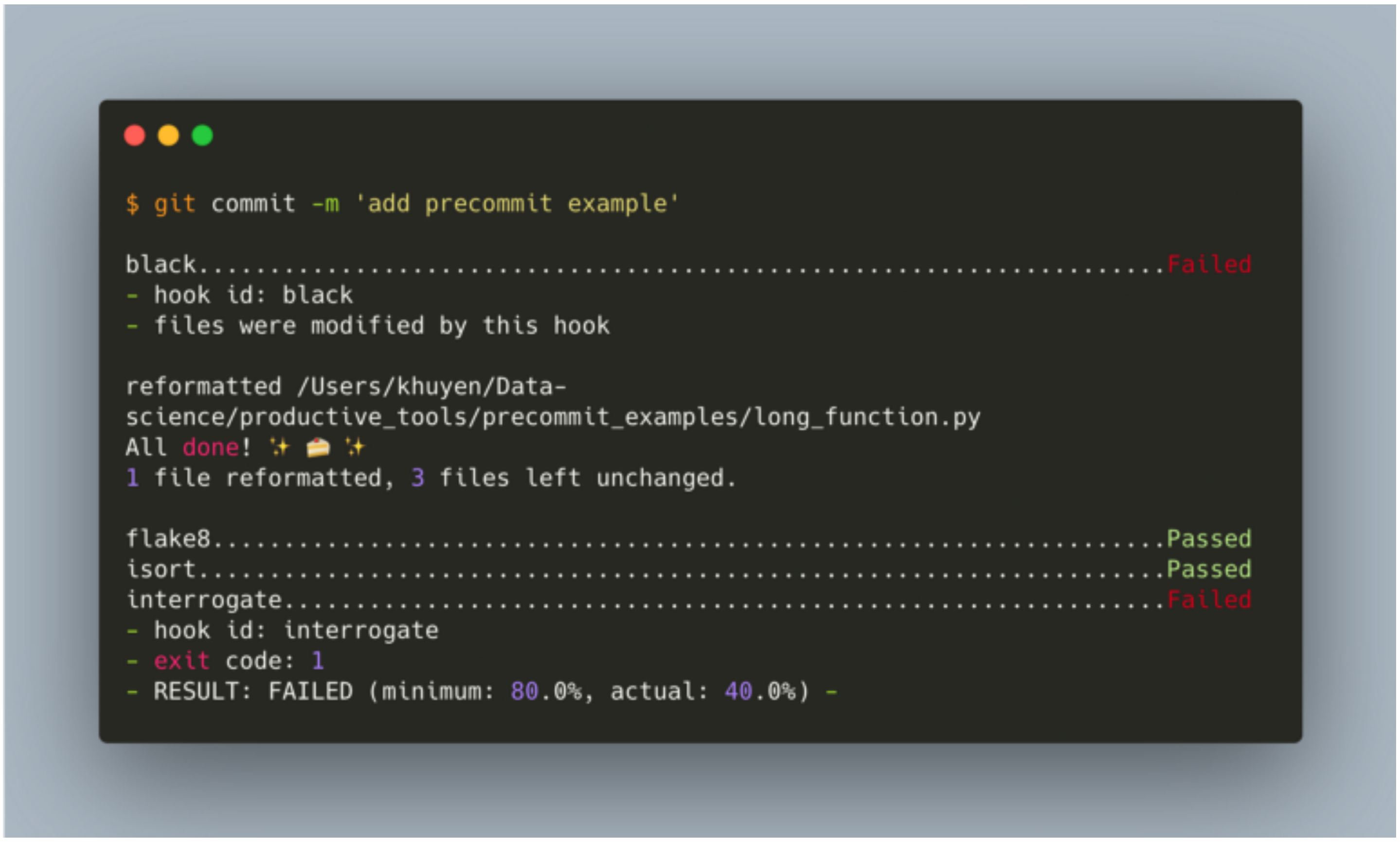
```
pre-commit installed at .git/hooks/pre-commit
```

Commit

Now we're ready to commit the new code!

```
$ git commit -m 'add pre-commit examples'
```

And you should see something like below:



```
$ git commit -m 'add precommit example'

black...................................................................Failed
- hook id: black
- files were modified by this hook

reformatted /Users/khuyen/Data-
science/productive_tools/precommit_examples/long_function.py
All done! ✨ 🍰 ✨
1 file reformatted, 3 files left unchanged.

flake8..................................................................Passed
isort...................................................................Passed
interrogate.............................................................Failed
- hook id: interrogate
- exit code: 1
- RESULT: FAILED (minimum: 80.0%, actual: 40.0%) -
```

Skip Verifying

To prevent pre-commit from checking a certain commit, add `--no-verify` to `git commit` :

```
$ git commit -m 'add pre-commit examples' --no-verify
```