

---

# **Pylama Documentation**

*Release 7.4.1*

**Kirill Klenov**

**Jun 28, 2017**



<b>I Requirements:</b>	<b>3</b>
<b>II Installation:</b>	<b>7</b>
<b>III Quickstart</b>	<b>11</b>
<b>IV Set Pylama (checkers) options</b>	<b>15</b>
1 Command line options	17
2 File modelines	19
3 Skip lines (noqa)	21
4 Configuration file	23
5 Set Code-checkers' options	25
6 Set options for file (group of files)	27
<b>V Pytest integration</b>	<b>29</b>
<b>VI Writing a linter</b>	<b>33</b>
7 Example:	37
8 Run pylama from python code	39
9 Bug tracker	41
10 Contributing	43
10.1 Contributors . . . . .	43
11 License	45





Welcome to Pylama's documentation.

Code audit tool for Python and JavaScript. Pylama wraps these tools:

- `pycodestyle` (formerly pep8) © 2012-2013, Florent Xicluna;
- `pydocstyle` (formerly pep257 by Vladimir Keleshev) © 2014, Amir Rachum;
- `PyFlakes` © 2005-2013, Kevin Watters;
- `Mccabe` © Ned Batchelder;
- `Pylint` © 2013, Logilab (should be installed 'pylama\_pylint' module);
- `Radon` © Michele Lacchia
- `gjslint` © The Closure Linter Authors (should be installed 'pylama\_gjslint' module);

**copyright** 2013 by Kirill Klenov.

**license** BSD, see LICENSE for more details.

## Contents

- *Welcome to Pylama*
- *Requirements:*
- *Installation:*
- *Quickstart*
- *Set Pylama (checkers) options*
  - *Command line options*
  - *File modelines*
  - *Skip lines (noqa)*
  - *Configuration file*
  - *Set Code-checkers' options*
  - *Set options for file (group of files)*
- *Pytest integration*
- *Writing a linter*
  - *Example:*
  - *Run pylama from python code*
  - *Bug tracker*
  - *Contributing*
    - \* *Contributors*

*- License*

---

## **Part I**

# **Requirements:**





- Python (2.7, 3.2, 3.3)
- To use JavaScript checker (gjshint) you need to install python-gflags with `pip install python-gflags`.
- If your tests are failing on Win platform you are missing: `curses` - <http://www.lfd.uci.edu/~gohlke/pythonlibs/>  
(The `curses` library supplies a terminal-independent screen-painting and keyboard-handling facility for text-based terminals)



## **Part II**

### **Installation:**



**Pylama** could be installed using pip: ::

```
$ pip install pylama
```



## **Part III**

# **Quickstart**





**Pylama** is easy to use and really fun for checking code quality. Just run *pylama* and get common output from all pylama plugins (*pycodestyle*, *PyFlakes* and etc)

Recursive check the current directory.

```
$ pylama
```

Recursive check a path.

```
$ pylama <path_to_directory_or_file>
```

Ignore errors

```
$ pylama -i W,E501
```

---

**Note:** You could choose a group errors *D*, *'E1'* and etc or special errors *C0312*

---

Choose code checkers

```
$ pylama -l "pycodestyle,mccabe"
```

Choose code checkers for JavaScript:

```
$ pylama --linters=gjslint --ignore=E:0010 <path_to_directory_or_file>
```



## **Part IV**

# **Set Pylama (checkers) options**



---

## Command line options

---

```
$ pylama --help

usage: pylama [-h] [--verbose] [--version] [--format {pycodestyle,pylint}]
             [--select SELECT] [--sort SORT] [--linters LINTERS]
             [--ignore IGNORE] [--skip SKIP] [--report REPORT] [--hook]
             [--async] [--options OPTIONS] [--force] [--abspath]
             [paths [paths ...]]

Code audit tool for python.

positional arguments:
  paths                Paths to files or directories for code check.

optional arguments:
  -h, --help          show this help message and exit
  --verbose, -v       Verbose mode.
  --version           show program's version number and exit
  --format {pycodestyle,pylint}, -f {pycodestyle,pylint}
                    Choose errors format (pycodestyle, pylint).
  --select SELECT, -s SELECT
                    Select errors and warnings. (comma-separated list)
  --sort SORT        Sort result by error types. Ex. E,W,D
  --linters LINTERS, -l LINTERS
                    Select linters. (comma-separated). Choices are
                    mccabe,pycodestyle,pyflakes,pydocstyle.
  --ignore IGNORE, -i IGNORE
                    Ignore errors and warnings. (comma-separated)
  --skip SKIP        Skip files by masks (comma-separated, Ex.
                    */messages.py)
  --report REPORT, -r REPORT
                    Send report to file [REPORT]
  --hook             Install Git (Mercurial) hook.
  --async            Enable async mode. Useful for checking a lot of
                    files. Unsupported with pylint.
  --options FILE, -o FILE
```

	Specify configuration file. Looks for pylama.ini, setup.cfg, tox.ini, or pytest.ini in the current directory.
--force, -F	Force code checking (if linter doesnt allow)
--abspath, -a	Use absolute paths in output.

## CHAPTER 2

---

### File modelines

---

You can set options for **Pylama** inside a source files. Use *pylama modeline* for this.

Format:

```
# pylama:{name1}={value1}:{name2}={value2}:...
```

```
.. Somewhere in code  
# pylama:ignore=W:select=W301
```

Disable code checking for current file:

```
.. Somewhere in code  
# pylama:skip=1
```

The options have a must higher priority.





## CHAPTER 3

---

### Skip lines (noqa)

---

Just add *#noqa* in end of line for ignore.

```
def urgent_fuction():  
    unused_var = 'No errors here' # noqa
```



---

## Configuration file

---

Pylama looks for a configuration file in the current directory, using the following names by default:

```
pylama.ini
setup.cfg
tox.ini
pytest.ini
```

The “`--option`” / “`-o`” argument can be used to specify a configuration file.

From the configuration file sections starting with *pylama* are read.

The “`pylama`” section configures global options like *linters* and *skip*.

```
[pylama]
format = pylint
skip = */.tox/*,*/.env/*
linters = pylint,mccabe
ignore = F0401,C0111,E731
```



---

## Set Code-checkers' options

---

You could set options for special code checker with pylama configurations.

```
[pylama:pyflakes]
builtins = _

[pylama:pycodestyle]
max_line_length = 100

[pylama:pylint]
max_line_length = 100
disable = R
```

See code checkers documentation for more info.



---

### Set options for file (group of files)

---

You could set options for special file (group of files) with sections:

The options have a higher priority than in the *pylama* section.

```
[pylama:*/pylama/main.py]
ignore = C901,R0914,W0212
select = R

[pylama:*/tests.py]
ignore = C0110

[pylama:*/setup.py]
skip = 1
```





## **Part V**

# **Pytest integration**



Pylama have [Pytest](#) support. The package automatically register self as pytest plugin during installation. Pylama also supports *pytest\_cache* plugin.

Check files with pylama

```
pytest --pylama ...
```

Recommended way to set pylama options when using pytest — configuration files (see below).



## **Part VI**

# **Writing a linter**



You can write a custom extension for Pylama. Custom linter should be a python module. Name should be like 'pylama\_<name>'.

In 'setup.py' should be defined 'pylama.linter' entry point.

```
setup(  
    # ...  
    entry_points={  
        'pylama.linter': ['lintername = pylama_lintername.main:Linter'],  
    }  
    # ...  
)
```

'Linter' should be instance of 'pylama.lint.Linter' class. Must implemented two methods:

'allow' take a path and returned true if linter could check this file for errors. 'run' take a path and meta keywords params and return list of errors.





## CHAPTER 7

---

Example:

---

Just a virtual 'WOW' checker.

setup.py:

```
setup(
    name='pylama_wow',
    install_requires=[ 'setuptools' ],
    entry_points={
        'pylama.linter': ['wow = pylama_wow.main:Linter'],
    }
    # ...
)
```

pylama\_wow.py:

```
from pylama.lint import Linter as BaseLinter

class Linter(BaseLinter):

    def allow(self, path):
        return 'wow' in path

    def run(self, path, **meta):
        with open(path) as f:
            if 'wow' in f.read():
                return [{
                    lnum: 0,
                    col: 0,
                    text: 'Wow has been finded.',
                    type: 'WOW'
                }]
        return []
```



---

### Run pylama from python code

---

```
from pylama.main import check_path, parse_options

my_redefined_options = {...}
my_path = '...'
options = parse_options([my_path], **my_redefined_options)
errors = check_path(options)
```



## CHAPTER 9

---

### Bug tracker

---

If you have any suggestions, bug reports or annoyances please report them to the issue tracker at <https://github.com/klen/pylama/issues>



# CHAPTER 10

---

## Contributing

---

Development of *pylama* happens at GitHub: <https://github.com/klen/pylama>

### Contributors

See [AUTHORS](#).





# CHAPTER 11

---

License

---

Licensed under a BSD license.