# Pylint

Pylint is a Static Code Analysis Tool for Python. It will return some:

**Coding Standard Analysis**, that allows for example:

* + Checking line-code’s length
  + Checking if variable names are well formed according to our coding standard
  + Ckecking if imported modules are used

**Error Detection**, for example:

* Check if declared interfaces are truly implemented
* Check if modules are imported

**Refactoring help** for code that can be written better. For example:

* It detects duplicated code.

## Install Steps

1. Install **Python** 3.x if you don´t have it. Choose the option “**Add Python to environment variables**” when the installation process asked you.
2. Open a Command Prompt and go to **/Automation/pylint**
3. Execute **.\pylint\_install.bat** that is in this folder.

## Configuration File

The **pylintrc file** is the configurable file of pylint. It’s important that this file be located in the root path of the project. Check if after executing the Install Steps, **pylintrc** file is in the **root folder**.

There are a lot of properties and things that can be configurated in order to customize the analysis and messages that would be shown to us. See more at the **Configuration Commands Section.**

## How to use

You can execute an analysis over any python file, a particular folder, or the whole project. For that, check if the **pylintrc** file is in the **root folder**. To run the pylint over a particular folder or the entire project, pylint must find the **\_\_init\_\_.py** file to recognize that is a python project.

For a particular file, from a Command Promt execute: **pylint path/name.py**

For whole project or a folder, execute: **pylint <path>**

Pylint prefixes each of the problem areas as you can see below:

Using the default text output, the message format is :

MESSAGE\_TYPE: LINE\_NUM:[OBJECT:] MESSAGE

There are 5 kind of message types :

\* (C) convention, for programming standard violation

\* (R) refactor, for bad code smell

\* (W) warning, for python specific problems

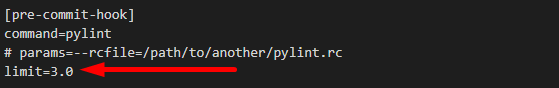
\* (E) error, for probable bugs in the code

\* (F) fatal, if an error occurred which prevented pylint from doing further processing.

## Git Commit

When you make a **git commit** it will invoke automatically to the pylint analysis, and it will rate each file that has been modified and corresponds to be commited. Only in the case that all files PASSED the analysis, the commit will be made. If one of them failed, the commit will be rejected.

To configure the limit that determine if a rate pass or fail, look and change the limit entry in the pylintrc:



To **bypass** the pre-commit hook when you do a commit, execute it as below indicate:

git commit --no-verify -m “Commit in order to test”

## Second analysis instance

This scenario is only if you want to do an analysis after a pull request has been made and before approving it.

1. Open a Command Prompt and go to the Automation root path
2. Execute: **pylint/pylint\_analyze.bat [pr\_no] [local\_branch]**, where:

* [pr\_no] is the Pull Request Number to be analyzed
* [local\_branch] is the local branch name to be ckeckouted

It will get the differences between the origin/development branch and the local branch where the Pull Request has been checkouted, and just analyze the .py changed files.

Example: pylint\pylint\_analyze.bat 634 TempBranch01

## Some example Commands

See the pylint help 🡺 pylint --help

Execute Pylint over a unique file 🡺 pylint directory/example.py

Execute Pylint over a unique file disabling Conventions Errors, only for this execution (will not impact the configuration file) 🡺 pylint example.py --disable=C

Execute Pylint over a unique file enabling Conventions Errors, only for this execution (will not impact the configuration file) 🡺 pylint \_\_init\_\_.py --enable=C

Analyze a Folder or a project called simulators 🡺 pylint directory/simulators

### Configuration Commands

Create a config file with all the current configuration 🡺 pylint --generate-rcfile > dir/pylintrc

Set some configuration options (disable Convention Messages) and include them to a new config file with all the current configuration 🡺 pylint --disable=C --generate-rcfile > dir/pylintrc

Set some configuration options (disable Convention and Warning Messages) and include them to a new config file with all the current configuration 🡺

pylint --disable=C --disable=W --generate-rcfile > dir/pylintrc

**Remember** that the **pylintrc** must be at the root of the directory.

**Keep in mind** that if you create a new pylintrc file, you must add these lines to it:

[pre-commit-hook]

command=pylint

# params=--rcfile=/path/to/another/pylint.rc

limit=8.0

## More information

For more information look at <https://www.pylint.org/>