

Python Development Workflow

Best Practice to create / test / versioning / automate and more

Marcelo Mello

mmello@redhat.com / tchello.mello@gmail.com

Pablo Hess

phess@redhat.com / pablonhess@gmail.com

Waldirio M Pinheiro

waldirio@redhat.com / waldirio@gmail.com

Version 1.0
2018/06/15

Table of Content

Thanks	3
License	4
Goal	4
Project	4
IDE	4
Create and Load the virtual environment	4
Create the Project Directory	5
The Code - First Version *not OO*	6
The Code - Second Version *OO*	7
Quality Assurance Application / Install pylint and flake8	8
Execute pylint on the code	8
after fix the code	9
Execute flake8 on the code	9
after fix the code	10
Code after all advices / changes	10
Configure the dir to be able to import *module*	11
Github	12
Create the Python Package	14
Installing the calc_demo Package	16
Function Based Test *Using pytest*	17
Class Based Test *Using Unit Test*	17
Test the code with pytest	18
TRICKs	19
Quality Assurance Application / Framework TOX	19
Initial TOX Configuration	19
Continuous Integration - Travis	21
Coveralls	25
Branch's and Fork's	30
Python Scaffolding	31
Conclusion	31
Links	32

Thanks

My many thanks to Marcelo Mello and Pablo Hess to share a lot of knowledge, tricks, advice, just to improve this guide and turn a funny way to play around with Python.

Teamwork++

License



Goal

The idea of this guide is just present you one complete workflow about Python development, starting on the beginning until the automation process to test / check / approve the code.

Project

Our project will be very simple, just to present the workflow in a simple way

- Simple calculator, just four methods "+, -, / and *"
- Python 3.6
- calc_demo will be the project name

IDE

About IDE you are free to select the best one to you, below my advice about it

- vim
 - Basic vim without any additional feature and/or plugins
- vim-jedi
 - # yum install vim-jedi
- Ipython
 - Very useful when you are doing debug / playing around the code. Just type
\$ pip install ipython
- Visual Studio Code
 - <https://code.visualstudio.com/>

Create and Load the virtual environment

Virtual environment will be the place where will be installed all packages related to that project. It's not necessary be the same directory as the project. The recommended is create one base virtualenv directory

```
$ mkdir ~/.virtualenvs
```

Note.: The command above will be executed once

Now we will create the new environment.

```
$ python3 -m venv ~/.virtualenvs/calc_demo
```

On the sequence, let's load the virtual environment

```
[wpinheir@iroman ~]$ source ~/.virtualenvs/calc_demo/bin/activate  
(calc_demo) [wpinheir@iroman ~]$
```

Now we are good to go and start our work.

One attention point, at this moment, you can check there are only simple packages installed on this virtual environment. Use the **pip list** command to do it

```
(calc_demo) [wpinheir@iroman ~]$ pip list
pip (9.0.3)
setuptools (37.0.0)
You are using pip version 9.0.3, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
(calc_demo) [wpinheir@iroman ~]$
```

Following the advice, let's update the pip package

```
(calc_demo) [wpinheir@iroman ~]$ pip install --upgrade pip
Cache entry deserialization failed, entry ignored
Collecting pip
  Using cached
https://files.pythonhosted.org/packages/0f/74/ecd13431bcc456ed390b44c8a6e917c1820365cbebc6a8974d1cd045ab4/pip-10.0.1-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 9.0.3
    Uninstalling pip-9.0.3:
      Successfully uninstalled pip-9.0.3
  Successfully installed pip-10.0.1
```

Below the **pip list** after package update process

```
(calc_demo) [wpinheir@iroman ~]$ pip list
Package      Version
-----
pip          10.0.1
setuptools  37.0.0
(calc_demo) [wpinheir@iroman ~]$
```

To leave the virtual environment, just type deactivate

```
(calc_demo) [wpinheir@iroman ~]$ deactivate
[wpinheir@iroman ~]$
```

Note. You can install on your machine the python package named **virtualenvwrapper**, this package will help you providing some commands that will prepare all environment to you. To install, just type **pip install virtualenvwrapper**

Create the Project Directory

In our example, the Project directory will be on `~/code/calc_demo` or `/home/wpinheir/code/calc_demo`, then let's create it

```
$ cd
$ mkdir -p code/calc_demo
$ cd code/calc_demo
```

To start your code, remember, will be necessary load the virtual environment, then

```
$ source ~/.virtualenvs/calc_demo/bin/activate
```

After that, you will see the virtualenv name at the beginning of the line/prompt

```
(calc_demo) [wpinheir@iroman ~]$
```

The Code - First Version *not OO*

This code is very simple, doesn't follow any PEP/Rule/Best practice but works.

On our **calc_demo** directory, let's create a subdirectory named **calc_demo** I'll **explain soon** and inside let's create the file **calc_demo.py** with the content below

```
def sum_call(a,b):
    return a+b

def div_call(a,b):
    return a/b

def mult_call(a,b):
    return a*b

def sub_call(a,b):
    return a-b

def main():

    first_value=500
    second_value=39

    # Sum
    print("Sum of {} + {} is: {}".format(first_value,second_value,sum_call(first_value,second_value)))

    # Division
    print("Division of {} / {} is: {}".format(first_value,second_value,div_call(first_value,second_value)))

    # Multiplication
    print("Multiplication of {} * {} is:
    {}".format(first_value,second_value,mult_call(first_value,second_value)))

    # Subtraction
    print("Subtraction of {} - {} is:
    {}".format(first_value,second_value,sub_call(first_value,second_value)))

if __name__ == '__main__':
    main()
```

The result when executing

```
(calc_demo) [wpinheir@iroman calc_demo]$ python calc_demo.py
Sum of 500 + 39 is: 539
Division of 500 / 39 is: 12.820512820512821
Multiplication of 500 * 39 is: 19500
Subtraction of 500 - 39 is: 461
(calc_demo) [wpinheir@iroman calc_demo]$
```

The Code - Second Version *OO*

At this point, we just rewrite the code but now defining class, object .. so just working with OO.

```
class Calc():

    def __init__(self,first,second):
        self._first= first
        self._second= second

    def sum_call(self):
        return self._first+self._second

    def div_call(self):
        return self._first/self._second

    def mult_call(self):
        return self._first*self._second

    def sub_call(self):
        return self._first-self._second

def main():

    first_value=500
    second_value=39

    # Object creation
    calc_run = Calc(first_value,second_value)

    # Sum
    print("Sum of {} + {} is: {}".format(first_value,second_value,calc_run.sum_call()))

    # Division
    print("Division of {} / {} is: {}".format(first_value,second_value,calc_run.div_call()))

    # Multiplication
    print("Multiplication of {} * {} is: {}".format(first_value,second_value,calc_run.mult_call()))

    # Subtraction
    print("Subtraction of {} - {} is: {}".format(first_value,second_value,calc_run.sub_call()))

if __name__ == '__main__':
    main()
```

Executing the code

```
(calc_demo) [wpinheir@iroman calc_demo]$ python calc_demo.py
Sum of 500 + 39 is: 539
```



```

C: 34, 0: Exactly one space required after comma
    print("Multiplication of {} * {} is: {}".format(first_value,second_value,calc_run.mult_call()))
        ^ (bad-whitespace)
C: 34, 0: Exactly one space required after comma
    print("Multiplication of {} * {} is: {}".format(first_value,second_value,calc_run.mult_call()))
        ^ (bad-whitespace)
C: 37, 0: Exactly one space required after comma
    print("Subtraction of {} - {} is: {}".format(first_value,second_value,calc_run.sub_call()))
        ^ (bad-whitespace)
C: 37, 0: Exactly one space required after comma
    print("Subtraction of {} - {} is: {}".format(first_value,second_value,calc_run.sub_call()))
        ^ (bad-whitespace)
C: 41, 0: Final newline missing (missing-final-newline)
C:  1, 0: Missing module docstring (missing-docstring)
C:  1, 0: Missing class docstring (missing-docstring)
C:  7, 4: Missing method docstring (missing-docstring)
C: 10, 4: Missing method docstring (missing-docstring)
C: 13, 4: Missing method docstring (missing-docstring)
C: 16, 4: Missing method docstring (missing-docstring)
C: 19, 0: Missing function docstring (missing-docstring)

```

Your code has been rated at -0.45/10 (previous run: 10.00/10, -10.45)

(calc_demo) [wpinheir@iroman calc_demo]\$

after fix the code

```

(calc_demo) [wpinheir@iroman calc_demo]$ pylint calc.py
No config file found, using default configuration

```

Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

(calc_demo) [wpinheir@iroman calc_demo]\$

Execute flake8 on the code

```

(calc_demo) [wpinheir@iroman calc_demo]$ flake8 calc.py
calc.py:3:1: E302 expected 2 blank lines, found 1
calc.py:26:1: E302 expected 2 blank lines, found 1
calc.py:36:80: E501 line too long (89 > 79 characters)
calc.py:39:80: E501 line too long (94 > 79 characters)
calc.py:42:80: E501 line too long (80 > 79 characters)
calc.py:42:80: E502 the backslash is redundant between brackets
calc.py:43:9: E128 continuation line under-indented for visual indent
calc.py:46:80: E501 line too long (97 > 79 characters)
(calc_demo) [wpinheir@iroman calc_demo]$

```

after fix the code

```
(calc_demo) [wpinheir@iroman calc_demo]$ flake8 calc.py  
(calc_demo) [wpinheir@iroman calc_demo]$
```

Code after all advices / changes

```
""" Calc code example """  
  
class Calc():  
    """ Calculator Class """  
  
    def __init__(self, first, second):  
        self._first = first  
        self._second = second  
  
    def sum_call(self):  
        """ Sum Def """  
        return self._first+self._second  
  
    def div_call(self):  
        """ Div Def """  
        return self._first/self._second  
  
    def mult_call(self):  
        """ Mult Def """  
        return self._first*self._second  
  
    def sub_call(self):  
        """ Subs Def """  
        return self._first-self._second  
  
def main():  
    """ Initiate the Calc """  
  
    first_value = 500  
    second_value = 39  
  
    # Object creation  
    calc_run = Calc(first_value, second_value)  
  
    # Sum  
    print("Sum of {} + {} is: {}".format(first_value, second_value,  
                                         calc_run.sum_call()))  
  
    # Division
```

```

print("Division of {} / {} is: {}".format(first_value, second_value,
                                         calc_run.div_call()))

# Multiplication
print("Multiplication of {} * {} is: {}".format(first_value, second_value,
                                                calc_run.mult_call()))

# Subtraction
print("Subtraction of {} - {} is: {}".format(first_value, second_value,
                                             calc_run.sub_call()))

if __name__ == '__main__':
    main()

```

Note. Please don't copy/paste the code above, try to fix the warning according the pylint and flake8 advice, if you have any question or concern, feel free to let us know or just do a simple research on the Internet, for sure you will find a lot of information related to that.

Configure the dir to be able to import *module*

Just create the empty file `__init__.py` on folder related to the code which you would like to import. In our case, inside `calc_demo` directory "subdir". Just execute the command below

```
(calc_demo) [wpinheir@iroman calc_demo]$ > __init__.py
```

TEST Time. On our project directory, let's import the code and call the main method. We should be able to see the result.

```

(calc_demo) [wpinheir@iroman calc_demo]$ tree
.
├── calc_demo
│   ├── calc_demo.py
│   └── __init__.py

```

1 directory, 2 files
(calc_demo) [wpinheir@iroman calc_demo]\$ pwd
/home/wpinheir/code/calc_demo
(calc_demo) [wpinheir@iroman calc_demo]\$ ll
total 4
drwxrwxr-x. 2 wpinheir wpinheir 4096 Jun 14 01:21 calc_demo
(calc_demo) [wpinheir@iroman calc_demo]\$ python
Python 3.6.4 (default, Mar 13 2018, 18:18:20)
[GCC 7.3.1 20180303 (Red Hat 7.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from calc_demo import calc_demo
>>> calc_demo.main()
Sum of 500 + 39 is: 539
Division of 500 / 39 is: 12.820512820512821
Multiplication of 500 * 39 is: 19500
Subtraction of 500 - 39 is: 461
>>>

Github



At this moment will be very interesting put our project in one **version control platform**, we will use github. We can create one repo via webUI and after just sync locally or create everything locally via CLI. Let's do it.

Please access the github website and create the repository

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

 **waldirio** / 


Great repository names are short and memorable. Need inspiration? How about **miniature-meme**.

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

| 

Create repository

Now you are able to get the project url

Quick setup — if you've done this kind of thing before

or 

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

Before proceed configure your github ssh key, after that, just move forward

```
(calc_demo) [wpinheir@iroman calc_demo]$ pwd
/home/wpinheir/code/calc_demo
(calc_demo) [wpinheir@iroman calc_demo]$
(calc_demo) [wpinheir@iroman calc_demo]$ git init
(calc_demo) [wpinheir@iroman calc_demo]$ git add README.md
(calc_demo) [wpinheir@iroman calc_demo]$ git commit -m "first commit"
```

```
(calc_demo) [wpinheir@iroman calc_demo]$ git remote add origin git@github.com:waldirio/calc_demo.git
(calc_demo) [wpinheir@iroman calc_demo]$ git push -u origin master
```

Note. At this moment via webUI you will be able to see the README.md file.

Let's check what git would like to store typing the command **git status**

```
bin/
build/
calc_demo.egg-info/
calc_demo/
dist/
setup.py
```

Now, let's create one local file named **.gitignore**, on this file let's exclude some folders and files that should be off from github. The content of the file will be according below

```
(calc_demo) [wpinheir@iroman calc_demo]$ cat .gitignore
.gitignore
*.egg*
*.pyc
(calc_demo) [wpinheir@iroman calc_demo]$
```

Now let's rerun the git status command

```
(calc_demo) [wpinheir@iroman calc_demo]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  bin/
  build/
  calc_demo/
  dist/
  setup.py

nothing added to commit but untracked files present (use "git add" to track)
(calc_demo) [wpinheir@iroman calc_demo]$
```

Now let's add the files, commit and push

```
(calc_demo) [wpinheir@iroman calc_demo]$ git add .
(calc_demo) [wpinheir@iroman calc_demo]$ git commit -m "new files"
(calc_demo) [wpinheir@iroman calc_demo]$ git push
```

At this moment you can check on the github webUI, all files and folders will be there.

Your project directory should be looks like below

```
(calc_demo) [wpinheir@iroman calc_demo]$ tree
.
├── build
│   ├── bdist.linux-x86_64
│   └── lib
│       └── calc_demo
│           ├── calc_demo.py
│           └── __init__.py
├── calc_demo
│   ├── calc_demo.py
│   ├── __init__.py
│   └── pycache
│       ├── calc_demo.cpython-36.pyc
│       └── __init__.cpython-36.pyc
├── calc_demo.egg-info
│   ├── dependency_links.txt
│   ├── PKG-INFO
│   ├── SOURCES.txt
│   └── top_level.txt
├── dist
│   ├── calc_demo-0.0.1-py3-none-any.whl
│   └── calc_demo-0.0.1.tar.gz
├── README.md
└── setup.py

8 directories, 14 files
(calc_demo) [wpinheir@iroman calc_demo]$
```

Create the Python Package

Creating the python package guys around the globe will be able to install and use your application **"and this is really awesome"**, they will install the application using the command **pip** or **easy_install**, the package will be available/hosted on the pypi website - <https://pypi.org/>

The first step is create the minimal structure, in our example, the Project directory will be on `~/code/calc_demo` or `/home/wpinheir/code/calc_demo`, then let's create the scaffolding.

```
calc_demo/
├── calc_demo/
│   ├── __init__.py
│   └── calc_demo.py
└── bin/
    └── calc_demo
```

Note. Now we are seeing the bin subdir and the file inside named `calc_demo`. Let's create this structure and the file according below

```
(calc_demo) [wpinheir@iroman calc_demo]$ mkdir bin
(calc_demo) [wpinheir@iroman calc_demo]$ >bin/calc_demo
(calc_demo) [wpinheir@iroman calc_demo]$
```

The content of `bin/calc_demo` is according below

```
#!/usr/bin/env python

from calc_demo import calc_demo
calc_demo.main()
```

Now it's time to create the file setup.py, this one will be on the top level directory of our project. Below how the file setup.py should look like.

```
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="calc_demo",
    version="0.0.1",
    author="Waldirio",
    author_email="waldirio@gmail.com",
    description="A small calc example package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/waldirio/calc_demo",
    packages=setuptools.find_packages(),
    scripts=['bin/calc_demo'],
    classifiers=(
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ),
)
```

Note. You can see the file README.md as a long_description, at this moment let's just create the file

```
(calc_demo) [wpinheir@iroman calc_demo]$ >README.md
```

Now **IT'S THE TIME** to install the python package, to do it

```
(calc_demo) [wpinheir@iroman calc_demo]$ pip install .
```

At this moment the installation will be locally and you should see something like below

```
(calc_demo) [wpinheir@iroman calc_demo]$ pip install .
Processing /home/wpinheir/code/calc_demo
Installing collected packages: calc-demo
  Running setup.py install for calc-demo ... done
Successfully installed calc-demo-0.0.1

(calc_demo) [wpinheir@iroman calc_demo]$ calc_demo
Sum of 500 + 39 is: 539
Division of 500 / 39 is: 12.820512820512821
Multiplication of 500 * 39 is: 19500
Subtraction of 500 - 39 is: 461

(calc_demo) [wpinheir@iroman calc_demo]$ which calc_demo
~/virtualenvs/calc_demo/bin/calc_demo

(calc_demo) [wpinheir@iroman calc_demo]$ cat ~/virtualenvs/calc_demo/bin/calc_demo
#!/home/wpinheir/.virtualenvs/calc_demo/bin/python3

from calc_demo import calc_demo
calc_demo.main()
(calc_demo) [wpinheir@iroman calc_demo]$
```

We can confirm the python package is installed with the command **pip list**


```
$ pip install calc-demo
Collecting calc-demo
  Downloading
https://files.pythonhosted.org/packages/41/12/5e1b76f2d5c71a5a14c9ced0e13c93c875c60e8bd81f6342d7593df71d78/calc_demo-0.0.1-py3-none-any.whl
Installing collected packages: calc-demo
Successfully installed calc-demo-0.0.1

$ calc_demo
Sum of 500 + 39 is: 539
Division of 500 / 39 is: 12.820512820512821
Multiplication of 500 * 39 is: 19500
Subtraction of 500 - 39 is: 461
```

As you can see the package was installed from PyPi.

Function Based Test *Using pytest*

Before we start with the test process, will be necessary create the structure. One good approach is create the test directory on the top level of your project. In our case, let's do it.

```
(calc_demo) [wpinheir@iroman calc_demo]$ pwd
/home/wpinheir/code/calc_demo
(calc_demo) [wpinheir@iroman calc_demo]$ mkdir test
(calc_demo) [wpinheir@iroman calc_demo]$ cd test
(calc_demo) [wpinheir@iroman test]$ > __init__.py
(calc_demo) [wpinheir@iroman test]$ pwd
/home/wpinheir/code/calc_demo/test
(calc_demo) [wpinheir@iroman test]$
```

Great, now we have to create the test file, by convention we will create the file using the nomenclature using **test_<definition>.py**, then the first one will be **test_sum.py**

This is one simple test just to confirm if the sum method is working as expected.

```
from calc_demo.calc_demo import Calc

def test_sum():
    n1 = Calc(3,6)
    assert n1.sum_call() == 9
```

Class Based Test *Using Unit Test*

Let's create the second one, this test will be used to check the division and the name will be **test_div.py**

```
import unittest
from calc_demo.calc_demo import Calc

class TestDiv(unittest.TestCase):
    def test_div(self):

        n1 = Calc(2,2)
```

```
self.assertEqual(n1.div_call(),1)
```

ATTENTION

I would like to share an important information. At this point if you type the command `pytest` the files inside test directory will be processed and will fail if the rule doesn't match. Ahead on this guide we will talk about coverage *code covered by test* and using coverage to collect data, ONLY Class Based Test will be processed. To keep both tests working, will be necessary install the package `pytest-cov` and the call will change. We will remember in the correct time.

Test the code with pytest

Now it's time to test, we will up one directory level and then type **pytest**, the application will looking for all files with **test_** on the name and all definitions with **test** on the name to be executed.

```
(calc_demo) [wpinheir@iroman calc_demo]$ pytest
===== test session starts =====
platform linux2 -- Python 2.7.15, pytest-3.5.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/wpinheir/code/calc_demo, inifile:
collected 2 items

test/test_div.py .                    [ 50%]
test/test_sum.py .                    [100%]

===== 2 passed in 0.01 seconds =====
(calc_demo) [wpinheir@iroman calc_demo]$
```

Yet on the test section, let's install the **pytest-cov** and execute the test but now with some parameters

```
(calc_demo) [wpinheir@iroman calc_demo]$ pip install pytest-cov
```

After installed, let's check the test again

```
(calc_demo) [wpinheir@iroman calc_demo]$ pytest --cov calc_demo
```

This command will generate the complete test output with coverage information

```
(calc_demo) [wpinheir@iroman calc_demo]$ pytest --cov calc_demo
===== test session starts =====
platform linux -- Python 3.6.4, pytest-3.6.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/wpinheir/code/calc_demo, inifile:
plugins: cov-2.5.1
collected 2 items

test/test_div.py .                    [ 50%]
test/test_sum.py .                    [100%]

----- coverage: platform linux, python 3.6.4-final-0 -----
Name                                Stmts  Miss  Cover
-----
calc_demo/__init__.py                 0     0   100%
calc_demo/calc_demo.py               22    10    55%
-----
TOTAL                                22    10    55%
```

```
===== 2 passed in 0.03 seconds =====  
(calc_demo) [wpinheir@iroman calc_demo]$
```

Note. After this command finish, will be generated the .coverage file which contain the coverage information.

TRICKs

During your project you have to create the **requirements.txt** file, this one will contain the modules used in your virtual environment and will be possible to anyone who want contribute with your project to build the similar virtual environment in a easy way. **All packages at the same level.**

For example, in our example when I execute the command **pip freeze** I can see all modules installed on this virtual environment.

```
(calc_demo) [wpinheir@iroman calc_demo]$ pip freeze  
astroid==1.6.5  
calc-demo==0.0.1  
certifi==2018.4.16  
chardet==3.0.4  
flake8==3.5.0  
idna==2.7  
isort==4.3.4  
lazy-object-proxy==1.3.1  
mccabe==0.6.1  
pkginfo==1.4.2  
pycodestyle==2.3.1  
pyflakes==1.6.0  
pylint==1.9.2  
requests==2.19.0  
requests-toolbelt==0.8.0  
six==1.11.0  
tqdm==4.23.4  
twine==1.11.0  
urllib3==1.23  
wrapt==1.10.11  
(calc_demo) [wpinheir@iroman calc_demo]$
```

So one simple way to create the requirements.txt is just redirecting the output to the file

```
(calc_demo) [wpinheir@iroman calc_demo]$ pip freeze > requirements.txt  
(calc_demo) [wpinheir@iroman calc_demo]$
```

Note. Remember, during the time of your project, if you add a new python module, just rerun the command to update the requirements.txt file.

Quality Assurance Application / Framework TOX

TOX will be the framework responsible to call different applications that will cover the quality of your application.

```
(calc_demo) [wpinheir@iroman calc_demo]$ pip install tox
```

Initial TOX Configuration

Execute the command `tox-quickstart` just to do the initial configuration, you will be prompted about some questions and at the end will be created the file `tox.ini`

```
(calc_demo) [wpinheir@iroman calc_demo]$ tox-quickstart
```

Below one simple `tox.ini` based on our selections

```
(calc_demo) [wpinheir@iroman calc_demo]$ cat tox.ini
# tox (https://tox.readthedocs.io/) is a tool for running tests
# in multiple virtualenvs. This configuration file will run the
# test suite on all supported python versions. To use it, "pip install tox"
# and then run "tox" from this directory.

[tox]
envlist = py27, py36

[testenv]
commands = pytest --cov
deps =
    pytest
    pytest-cov
(calc_demo) [wpinheir@iroman calc_demo]$
```

Now, just type `tox` on the command line

```
congratulations :)
(calc_demo) [wpinheir@iroman calc_demo]$ =
(calc_demo) [wpinheir@iroman calc_demo]$ tox
GLOB sdist-make: /home/wpinheir/code/calc_demo/setup.py
py27 inst-nodes: /home/wpinheir/code/calc_demo/tox/dist/calc_demo-0.0.1.zip
py27 installed: atomicwrites=1.1.5,attrs=18.1.0,calc-demo=0.0.1,coverage=4.5.1,funcsigs=1.0.2,more-itertools=4.2.0,pluggy=0.6.0,py=1.5.3,pytest=3.6.1,pytest-cov=2.5.1,six=1.11.0
py27 runtests: PYTHONHASHSEED=3921910851
py27 runtests: commands[0] | pytest --cov calc_demo
===== test session starts =====
platform linux2 -- Python 2.7.15, pytest-3.6.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/wpinheir/code/calc_demo, inifile:
plugins: cov-2.5.1
collected 2 items

test/test_div.py .
test/test_sum.py .

----- coverage: platform linux2, python 2.7.15-final-0 -----
Name      Stats  Miss  Cover
-----
calc_demo/_init__.py  0      0  100%
calc_demo/calc_demo.py 22     10   55%
TOTAL                22     10   55%

===== 2 passed in 0.02 seconds =====
py36 inst-nodes: /home/wpinheir/code/calc_demo/tox/dist/calc_demo-0.0.1.zip
py36 installed: atomicwrites=1.1.5,attrs=18.1.0,calc-demo=0.0.1,coverage=4.5.1,more-itertools=4.2.0,pluggy=0.6.0,py=1.5.3,pytest=3.6.1,pytest-cov=2.5.1,six=1.11.0
py36 runtests: PYTHONHASHSEED=3921910851
py36 runtests: commands[0] | pytest --cov calc_demo
===== test session starts =====
platform linux -- Python 3.6.4, pytest-3.6.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/wpinheir/code/calc_demo, inifile:
plugins: cov-2.5.1
collected 2 items

test/test_div.py .
test/test_sum.py .

----- coverage: platform linux, python 3.6.4-final-0 -----
Name      Stats  Miss  Cover
-----
calc_demo/_init__.py  0      0  100%
calc_demo/calc_demo.py 22     10   55%
TOTAL                22     10   55%

===== 2 passed in 0.03 seconds =====
lint inst-nodes: /home/wpinheir/code/calc_demo/tox/dist/calc_demo-0.0.1.zip
lint installed: atomicwrites=1.1.5,attrs=18.1.0,calc-demo=0.0.1,coverage=4.5.1,more-itertools=4.2.0,pluggy=0.6.0,py=1.5.3,pytest=3.6.1,pytest-cov=2.5.1,six=1.11.0
lint runtests: PYTHONHASHSEED=3921910851
lint runtests: commands[0] | pytest --cov calc_demo
===== test session starts =====
platform linux -- Python 3.6.4, pytest-3.6.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/wpinheir/code/calc_demo, inifile:
plugins: cov-2.5.1
collected 2 items

test/test_div.py .
test/test_sum.py .

----- coverage: platform linux, python 3.6.4-final-0 -----
Name      Stats  Miss  Cover
-----
calc_demo/_init__.py  0      0  100%
calc_demo/calc_demo.py 22     10   55%
TOTAL                22     10   55%

===== 2 passed in 0.04 seconds =====
summary
py27: commands succeeded
py36: commands succeeded
lint: commands succeeded
congratulations :)
(calc_demo) [wpinheir@iroman calc_demo]$
```

To call the `pytest` and `flake` via `TOX`, the conf file should be similar to below

```
# tox (https://tox.readthedocs.io/) is a tool for running tests
# in multiple virtualenvs. This configuration file will run the
# test suite on all supported python versions. To use it, "pip install tox"
# and then run "tox" from this directory.
```

```
[tox]
```

```
envlist = py27, py36, lint
```

```
[testenv]
```

```
setenv =
```

```
    PYTHONPATH = {toxiniidir}:{toxiniidir}/calc
```

```
commands =
```

```
    pytest --cov calc_demo
```

```
deps =
```

```
    pytest
```

```
    pytest-cov
```

```
    pylint
```

```
    lint
```

```
    flake8
```

```
[testenv:lint]
```

```
ignore_errors = True
```

```
commands =
```

```
    pylint calc
```

```
    flake8 calc
```

After update the file, just rerun the **tox** command and everything will be tested **pytest**, **pylint** and **flake8** on version 2.7 and 3.6.

```
===== 2 passed in 0.02 seconds =====
py36 recreate: /home/wpinheir/code/calc_demo/.tox/py36
py36 installdeps: pytest, pytest-cov, pylint, lint, flake8
py36 inst: /home/wpinheir/code/calc_demo/.tox/dist/calc_demo-0.0.1.zip
py36 installed: astroid==1.6.5,atomicwrites==1.1.5,attrs==18.1.0,calc-demo==0.0.1,coverage==4.5.1,flake8==3.5.0,gitdb2==2.0.3,GitPython==2.1.10,importlib==4.3.4,lazy-object-proxy==1.3.1,lint==1.2.1,mccabe==0.6.1,more-itertools==4.2.0,pluggy==0.6.0,py==1.5.3,pycodestyle==2.3.1,pyflakes==1.6.0,pylint==1.9.2,pytest==3.6.1,pytest-cov==2.5.1,six==1.11.0,smm
ap2==2.0.3,wrapt==1.10.11
py36 runtests: PYTHONHASHSEED='3425281896'
py36 runtests: commands[0] | pytest --cov calc_demo
===== test session starts =====
platform linux -- Python 3.6.4, pytest-3.6.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/wpinheir/code/calc_demo, inifile:
plugins: cov-2.5.1
collected 2 items

test/test_div.py . [ 50%]
test/test_sum.py . [100%]

----- coverage: platform linux, python 3.6.4-final-0 -----
Name                               Stmts  Miss  Cover
-----
calc_demo/_init_.py                  0     0   100%
calc_demo/calc_demo.py               22    10    55%
TOTAL                                22    10    55%

===== 2 passed in 0.03 seconds =====
lint recreate: /home/wpinheir/code/calc_demo/.tox/lint
lint installdeps: pytest, pytest-cov, pylint, lint, flake8
lint inst: /home/wpinheir/code/calc_demo/.tox/dist/calc_demo-0.0.1.zip
lint installed: astroid==1.6.5,atomicwrites==1.1.5,attrs==18.1.0,calc-demo==0.0.1,coverage==4.5.1,flake8==3.5.0,gitdb2==2.0.3,GitPython==2.1.10,importlib==4.3.4,lazy-object-proxy==1.3.1,lint==1.2.1,mccabe==0.6.1,more-itertools==4.2.0,pluggy==0.6.0,py==1.5.3,pycodestyle==2.3.1,pyflakes==1.6.0,pylint==1.9.2,pytest==3.6.1,pytest-cov==2.5.1,six==1.11.0,smm
ap2==2.0.3,wrapt==1.10.11
lint runtests: PYTHONHASHSEED='3425281896'
lint runtests: commands[0] | pylint calc_demo
No config file found, using default configuration

Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

lint runtests: commands[1] | flake8 calc_demo
===== summary =====
py27: commands succeeded
py36: commands succeeded
lint: commands succeeded
congratulations :)
(calc_demo) [wpinheir@iroman calc_demo]$
```

Continuous Integration - Travis

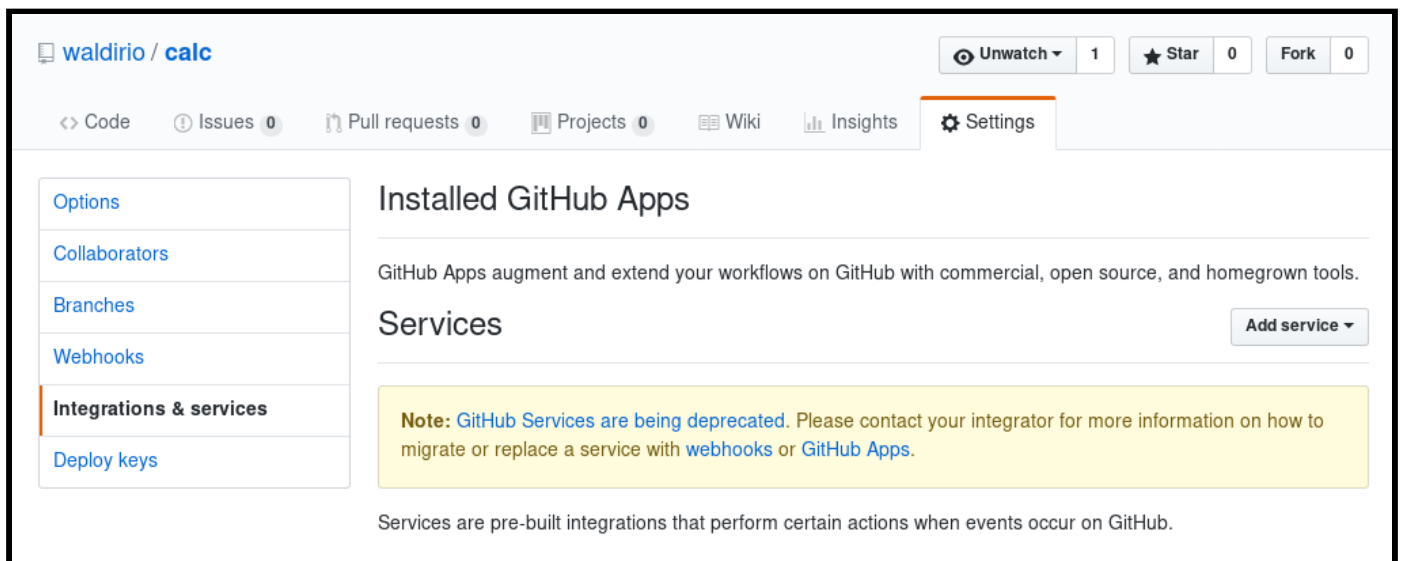
We need one application just to help us on the **CI** or **Continuous Integration** workflow, then Travis will be the smart guy to do this service.

To start using travis, will be necessary

- GitHub login
- Project hosted as a repository on GitHub
- Working code in your project
- Working build or test script

Then just access the link <https://docs.travis-ci.com/user/getting-started/> and follow the instructions

Just a shortcut, Access your github page, then the **project**, after that **Settings** tab, **Integrations & services** option, **Add service** dropdown



On the filter type **travis CI** and click over it

On this page, type your account used on Travis, the token you can copy from the Travis page. After that just click on **Add service**.

Automatic configuration from Travis CI

We recommend using the Travis profile page at <https://travis-ci.org/profile> to m
For private repositories, use <https://travis-ci.com/profile>.

Travis CI Status

Travis CI status page: <http://status.travis-ci.com>

User


Token

Domain

Active
We will run this service when an event is triggered.







Add service

Now come back to the Travis page and enable the project you would like to add on the test


 **waldirio**
@waldirio

We're only showing your public repositories. You can find your private projects on travis-ci.com.


Legacy Services Integration

 backup_router	<input type="checkbox"/>	 Settings
 calc	<input type="checkbox"/>	 Settings
 calc_demo	<input checked="" type="checkbox"/>	 Settings

After this update the project will appear at the main screen

waldirio / calc_demo  build unknown

[Current](#) [Branches](#) [Build History](#) [Pull Requests](#)



No builds for this repository

Want to start testing this project on Travis CI?

[Read the Docs on Getting Started](#)

Will be necessary create the file on the project directory according below


```
(calc_demo) [wpinheir@iroman calc_demo]$ cat .travis.yml
language: python
matrix:
  include:
    - python: "2.7"
      env: TOXENV=py27
    - python: "3.6"
      env: TOXENV=lint
install: pip install -U tox
(calc_demo) [wpinheir@iroman calc_demo]$
```

After this moment when you commit and push any change to your project, automatically Travis will start the build and will let you know the result.

```
(calc_demo) [wpinheir@iroman calc_demo]$ git add .
(calc_demo) [wpinheir@iroman calc_demo]$ git commit -m "adding all files"
(calc_demo) [wpinheir@iroman calc_demo]$ git push
```

At this moment Travis will be triggered and will start the build process

Search all repositories

waldirio / calc_demo  build unknown

My Repositories +

- waldirio/calc_demo # 2
 - Duration: 1 min 7 sec
 - Finished: less than a minute ago
- RHsysmgmt/host_delete
 - Duration: -







Current Branches Build History Pull Requests More options

master test dir → #2 passed Restart build

- Commit e14bfd9
- Compare c6c9e02..e14bfd9
- Branch master
- Waldirio M Pinheiro authored and committed

Ran for 37 sec
 Total time 1 min 7 sec
 less than a minute ago

Build Jobs

<input checked="" type="checkbox"/>	# 2.1	 Python: 2.7	 TOXENV=py27	34 sec	
<input checked="" type="checkbox"/>	# 2.2	 Python: 3.6	 TOXENV=lint	33 sec	

Below the result of one specific Job, will be created one per python version if you are testing multiples

```
Remove log Raw log
1 Worker information worker_info
6 Build system information system_info
403
404 Network availability confirmed.
405
406
407 $ git clone --depth=50 --branch=master https://github.com/waldirio/calc_demo.git git.checkout 0.41s
416
417 Setting environment variables from .travis.yml
418 $ export TOXENV=lint
419
420 $ source ~/virtualenv/python3.6/bin/activate 0.01s
421
422 $ python --version
423 Python 3.6.3
424 $ pip --version
425 pip 9.0.1 from /home/travis/virtualenv/python3.6.3/lib/python3.6/site-packages (python 3.6)
426 $ pip install -U tox install 1.87s
444 $ tox 18.02s
445 GLOB sdist-make: /home/travis/build/waldirio/calc_demo/setup.py
446 lint create: /home/travis/build/waldirio/calc_demo/.tox/lint
447 lint installdeps: pytest, pylint, lint, flake8
448 lint inst: /home/travis/build/waldirio/calc_demo/.tox/dist/calc_demo-0.0.1.zip
449 lint installed: astroid==1.6.5,atomicwrites==1.1.5,attrs==18.1.0,calc-
demo==0.0.1,flake8==3.5.0,gitdb2==2.0.3,GitPython==2.1.10,isort==4.3.4,lazy-object-
proxy==1.3.1,lint==1.2.1,mccabe==0.6.1,more-
itertools==4.2.0,pluggy==0.6.0,py==1.5.3,pycodestyle==2.3.1,pyflakes==1.6.0,pylint==1.9.2,pytest==3.6.1,six==
1.11.0,smmap2==2.0.3,wrapt==1.10.11
450 lint runtests: PYTHONHASHSEED='2319835348'
451 lint runtests: commands[0] | pylint calc_demo
452 No config file found, using default configuration
453
454 -----
455 Your code has been rated at 10.00/10
456
457 lint runtests: commands[1] | flake8 calc_demo
458 ----- summary -----
459 lint: commands succeeded
460 congratulations :)
461
462
463 The command "tox" exited with 0.
464
465 Done. Your build exited with 0.
Top
```

Coveralls

Coveralls will be the smart piece which will let us know what piece of our code still without any test and this is very useful because the desire is cover the entire code just to avoid issues. Now it's time to configure the Coveralls, to do it access the link <https://coveralls.io/>

YOUR REPOSITORIES

NO REPOSITORIES: ADD REPOS TO START TRACKING COVERAGE

GETTING STARTED

Add repos from GitHub or Bitbucket by clicking the add repo button above. You'll see a list of all your repositories on GitHub, just click on the slider button to add a repo to Coveralls. If you need help setting up your app to process builds on Coveralls, [check the docs](#).

REPO OVERVIEW

Once you have repos added and have processed builds, this page will let you easily see the current coverage levels across your repos, and shows you a graph of the coverage changes over time. It will look something like the image below, but with your repos instead of ours. Happy testing!



No repo configured yet, just click on Add Repo and enable the desired. In our case, **calc_demo** project

Will be necessary update the file **.travis.yml** according below

```
language: python
matrix:
  include:
    - python: "2.7"
      env: TOXENV=py27
    - python: "3.6"
      env: TOXENV=lint
install: pip install -U tox coveralls
script: tox
after_success: coveralls
```

And update the **tox.ini** as below **adding the coveralls call**

```
(calc_demo) [wpinheir@iroman calc_demo]$ cat tox.ini
# tox (https://tox.readthedocs.io/) is a tool for running tests
# in multiple virtualenvs. This configuration file will run the
# test suite on all supported python versions. To use it, "pip install tox"
# and then run "tox" from this directory.

[tox]
envlist = py27, py36, lint
```

```

[testenv]

setenv =
    PYTHONPATH = {toxinidir}:{toxinidir}/calc_demo

commands =
    pytest --cov calc_demo

deps =
    pytest
    pytest-cov
    pylint
    lint
    flake8

[testenv:lint]
ignore_errors = True
commands =
    pylint calc_demo
    flake8 calc_demo
(calc_demo) [wpinheir@iroman calc_demo]$

```

After update, just commit and push all files, and the rebuild process will start automatically on Travis, after that will be presented on the coveralls the % of code covered.



And if you drill down on the last job, you can see what part of the code is covered and what part is not

COMMITTED 14 JUN 2018 - 11:13

COVERAGE REMAINED THE SAME AT 50.0%

BUILD #	BUILD TYPE	COMMITTED BY	COMMIT MESSAGE	RUN DETAILS
5	push travis-ci	Waldirio M Pinheiro	tox update	11 of 22 relevant lines covered (50.0%) 0.5 hits per line

SOURCE FILE

Press 'n' to go to next uncovered line, 'b' for previous

50.0 /calc_demo/calc_demo.py

```

1  """ Calc code example """
2
3
4  class Calc():
5      """ Calculator Class """
6
7      def __init__(self, first, second):
8          self._first = first
9          self._second = second
10
11     def sum_call(self):
12         """ Sum Def """
13         return self._first+self._second
14
15     def div_call(self):
16         """ Div Def """
17         return self._first/self._second
18
19     def mult_call(self):
20         """ Mult Def """
21         return self._first*self._second

```

Then this is a good way to check your code and know how many of your code is tested / covered.

ATTENTION

If you have in your project just UnitTest ***Class Based*** your **tox.ini** can be looks like below ***using coverage***

```

(calc_demo) [wpinheir@iroman calc_demo]$ cat tox.ini
# tox (https://tox.readthedocs.io/) is a tool for running tests
# in multiple virtualenvs. This configuration file will run the
# test suite on all supported python versions. To use it, "pip install tox"
# and then run "tox" from this directory.

[tox]
envlist = py27, py36, lint

[testenv]

setenv =

```

```
PYTHONPATH = {toxindir}:{toxindir}/calc_demo
```

```
commands =
```

```
    pytest
```

```
    coverage run --source=calc_demo setup.py test
```

```
deps =
```

```
    pytest
```

```
    pylint
```

```
    lint
```

```
    flake8
```

```
[testenv:lint]
```

```
ignore_errors = True
```

```
commands =
```

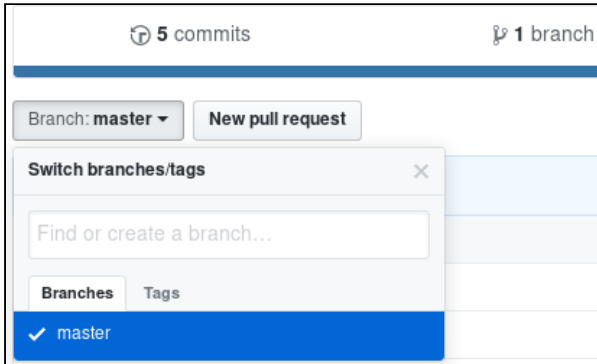
```
    pylint calc_demo
```

```
    flake8 calc_demo
```

```
(calc_demo) [wpinheir@iroman calc_demo]$
```

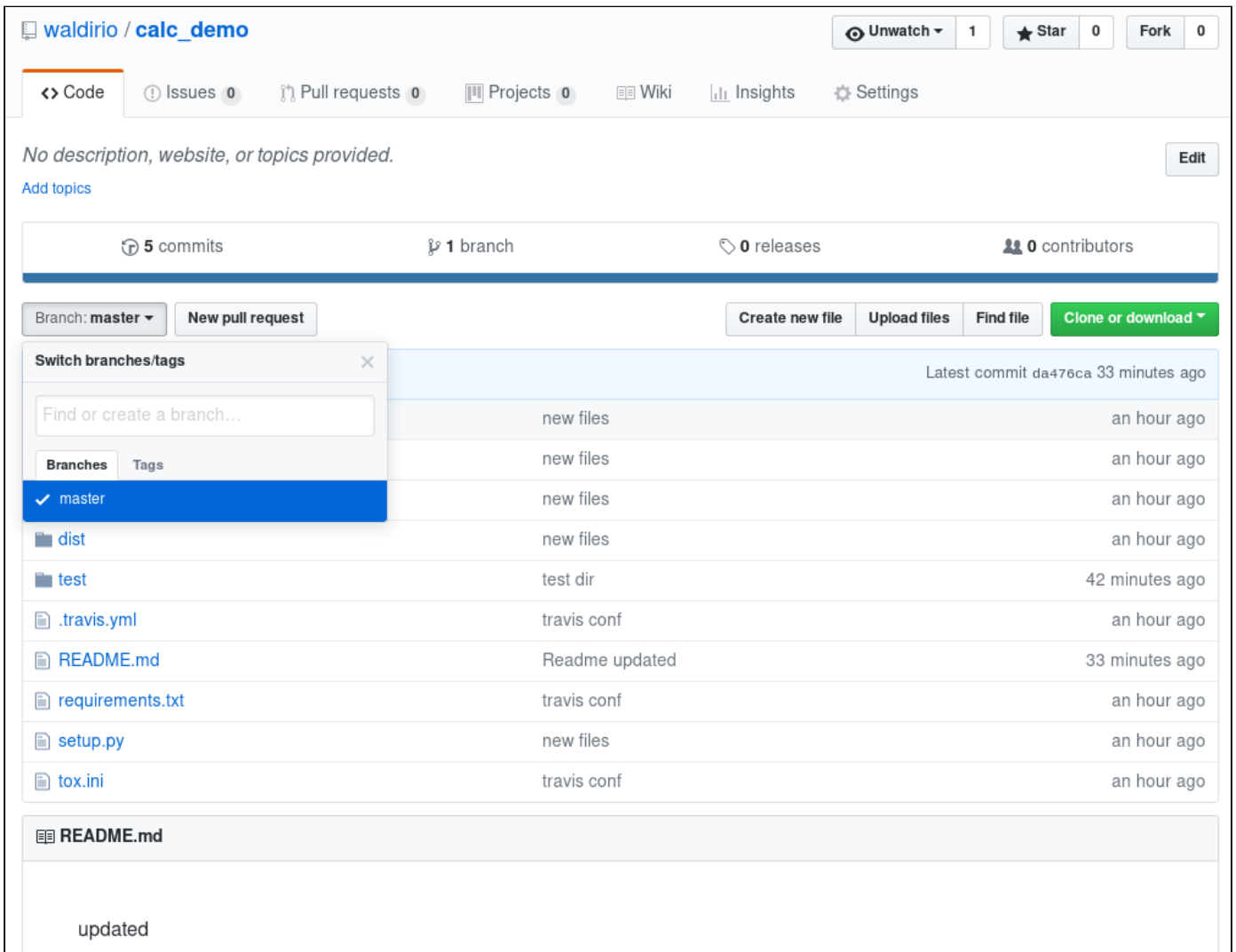
Branch's and Fork's

Start working with branch's is a good way to keep control about the change you are doing "**PR or Pull Request**" as you will be able to see all process "**test with status / results**" on the github page.



On our scenario we are just working with the branch master.

Forks is another stage where people around the globe will just contribute on the code, at the end of the day you will receive the **PR** just to Merge the code and move forward.



When they start to fork your code just to help you, this number will increase and you will be able to see who are forking the code, as receive a new **PR / Pull Request** just to merge the code.

Python Scaffolding

So if you are thinking to start a new project, below our advice

- Define the structure of your project
 - Directories / Subdirectories
- Github / Gitlab / any **SCM**
 - Create the .gitignore just to remove what should not be on the repository
 - Register all features as issue
- Code
 - Write your code
 - Don't forget your `__init__.py` on **modules** directories
 - Write your tests
 - Don't forget your `__init__.py` on **modules** directories
- QA
 - Install and test your Quality/Test Apps
 - pytest
 - pytest-cov
 - flake8
- Configure TOX
 - tox.ini
- Configure Travis
 - webUI
 - .travis.yml
- Configure Coveralls
 - webUI
 - Update .travis.yml

At the end of the day, just keep this process, all the time will be improved with new functionalities, new apps, different ways to check the code.

Conclusion

Following this guide we hope help your understanding about Python, Development Workflow and Best Practices, for sure you will find different applications, platforms and ways to do the same thing, here is just one way that we believe be interesting.

Please, feel free to let us know if you have any question / doubt / issue, for sure we will be glad to help you.

Links

<https://tox.readthedocs.io/en/latest/>

<https://docs.pytest.org/en/latest/>

<https://travis-ci.org/>

<https://coveralls.io/>

<https://packaging.python.org/tutorials/distributing-packages/#setup-py>

<https://code.visualstudio.com/>

<https://python-packaging.readthedocs.io/>

<https://packaging.python.org/tutorials/packaging-projects/>

<https://pytest-cov.readthedocs.io/en/latest/>

<http://docs.python-guide.org/en/latest/writing/tests/>