**PyTest Tutorial — Parallel Testing With Selenium Grid**

Save time spent on Selenium test automation by running tests on parallel. Learn how to run parallel testing.

Selenium is one of the widely used test automation frameworks for automated browser testing. Selenium test automation is really helpful in testing websites or web apps on different combinations of browsers, operating systems, and devices. Giving better functional test coverage as the code is tested on a wide range of combinations.

Running these tests in sequence can be time-consuming, as you’d wait for one test to complete before running other tests. You can save a lot of time by performing tests in parallel, thus improving the scalability of your Selenium test automation. Parallel testing helps in performing tests on browsers simultaneously, providing better test coverage in a shorter time.

In this Selenium Python tutorial, I’ll show. you how to run parallel tests in pytest using Selenium Grid. The Selenium Grid to run can either be local or cloud-based. For more information on setting up the local Selenium Grid, we recommend having a look at our detailed blog on Setting up Selenium Grid for automation testing.

## How To Run Selenium Tests In Parallel With Python Using Pytest-xdist?

By default, PyTest does not support parallel testing which is extremely essential for scenarios such as automated browser testing. Parallel testing is a must-have to achieve continuous integration as tests can be executed at a rapid pace. To run Selenium tests in parallel with Python, you need to install the pytest-xdist plugin.

### Features (or Execution Modes) of Pytest-xdist

It is a PyTest distributed testing plugin that extends python PyTest  with some unique execution modes mentioned below in this Selenium Python tutorial :

* **Multi-process load balancing —**Multiple CPUs or hosts can be used for a combined test run. This aids in speeding up the development along with using special resources of machines.
* **LooponFail —**Tests can be executed repeatedly in a sub-process. After every test run, the pytest re-runs all the tests that have failed before. This process is repeated until all the tests pass. This is considered the end of the test.
* **Multi-platform coverage —**Different Python interpreters (e.g. PyTest, PyUnit, etc.) or platforms can be specified and tests can be performed parallelly on all of them.

### Installation Of pytest-xdist To Run Selenium Tests In Parallel With Python

The xdist plugin can be installed by executing either of the following commands on the terminal:

pip install pytest-xdist

easy\_install pytest-xdist

Shown below in this Selenium Python tutorial is the installation screenshot of xdist plugin to run Selenium tests in parallel with Python.



### Command-Line Options for Parallel Testing

The pytest-xdist plugin provides command-line options for sending tests to multiple CPUs. The number of CPUs is passed after the option –n.

pytest -n <num-of-cpus>

The option speeds up the parallel execution for lengthy tests, as well as, tests that have frequent I/O (Input/Output) operations in the implementation. pytest-xdist (or xdist-plugin) has many other options, details of the same are available in the [xdist-plugin documentation](https://docs.pytest.org/en/3.0.1/xdist.html)

## Run Selenium Tests in Parallel With Python Using pytest-xdist Plugin

To demonstrate the usage of the pytest-xdist plugin to run Selenium tests in parallel with Python, I’ll take four test scenarios for this Selenium Python tutorial which I’ll execute on Chrome and Firefox web browsers. The four test scenarios are divided across two python files and the fixtures are shared via conftest.py. For more detailed information on PyTest fixtures and usage of conftest.py, you can refer to a previous Selenium Python tutorial on PyTest fixtures.

### Test Scenario — 1 (Run Selenium tests in parallel with Python on Chrome Browser)

**Test Case — 1**

1. Navigate to the URL [https:/](https://lambdatest.github.io/sample-todo-app/)/www.rediffmail.com

2. Login

**Test Case — 2**

1. Navigate to the URL [https:/](https://lambdatest.github.io/sample-todo-app/)/www.rediffmail.com

2. Go to Inbox

**Test Scenario — 2 ((Run Selenium tests in parallel with Python on Firefox Browser)**

**Test Case — 1**

1. Navigate to the URL [https://www.google.com](https://www.google.com/)

2. Search for “SeleniumTest”

3. Click on the first test result

4. Raise an Assert if the Page Title does not match the expected title

**Test Case — 2**

1. Navigate to the URL <https://www.google.com>

2. Compare the window title with the expected title

3. Raise assert if titles do not match

## Implementation

Fixtures for invoking the Chrome and Firefox browser are added in Conftest.py. The file is placed in the root folder where the files that contain implementation for the test cases are also located.

**Conftest.py**

As different URLs are used for automated browser testing, the scope is set to class (instead of a session) and a new browser instance is loaded for each test.

#Run Selenium tests in parallel with Python for Selenium Python tutorial

import pytest

from selenium import Webdriver

@pytest.fixture(scope="class")

def driver\_init\_1(request):

web\_driver = webdriver.Chrome()

request.cls.driver = web\_driver

yield

web\_driver.close()

@pytest.fixture(scope="class")

def driver\_init\_2(request):

web\_driver = webdriver.Firefox()

request.cls.driver = web\_driver

yield

web\_driver.close()

**test\_pytest\_1.py**

This file contains the implementation of Test Scenario — 1 (Run Selenium tests in parallel with Python on Chrome Browser). The @pytest.mark.usefixtures decorator is included for using the fixture driver\_chrome\_init().

It contains two test cases to run Selenium tests in parallel with Python:

* **test\_rediffmail\_login\_app() —**The required web elements are located using Selenium methods such as find\_element\_by\_name(), driver.find\_element\_by\_id(). Once the elements are located, necessary commands such as click(), etc. are used to perform the required operation.
* **test\_redifffmail\_inbox() —**The URL under test https://www.rediffmail.com is loaded and the window title is compared with the expected title. Assert is raised if the titles do not match.

close() command in Selenium WebDriver is used to close the browser window once the test execution is completed.

import pytest

import pytest\_html

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.keys import Keys

import time

from time import sleep

import sys

@pytest.mark.usefixtures("driver\_init\_1")

class BasicTest:

pass

class Test\_URL\_Chrome(BasicTest):

def test\_rediffmail\_login\_app(self):

self.driver.get('https://www.rediffmail.com/')

self.driver.maximize\_window()

self.driver.find\_element\_by\_name("li1").click()

self.driver.find\_element\_by\_name("li2").click()

title = "Sample page - rediffmail.com"

assert title == self.driver.title

sample\_text = "Happy Testing at LambdaTest"

email\_text\_field = self.driver.find\_element\_by\_id("sampletodotext")

email\_text\_field.send\_keys(sample\_text)

time.sleep(5)

self.driver.find\_element\_by\_id("addbutton").click()

time.sleep(5)

output\_str = self.driver.find\_element\_by\_name("li6").text

sys.stderr.write(output\_str)

time.sleep(2)

def test\_rediffnail\_inbox(self):

self.driver.get('https://www.rediffmail.com/')

self.driver.maximize\_window()

expected\_title = "cross-browser Testing Tools | Free Automated Website Testing | Rediffmail"

assert expected\_title == self.driver.title

time.sleep(5)

**test\_pytest\_2.py**

This file contains the implementation of Test Scenario — 2 (Execution on Firefox Browser). The @pytest.mark.usefixtures decorator is included for using the fixture driver\_firefox\_init().

It contains two test cases to run Selenium tests in parallel with Python:

* **test\_google\_search() —**The search box on Google homepage is located using find\_element\_by\_xpath method of Selenium WebDriver. Once the element is located, search text (i.e. LambdaTest) is passed to the search box. After the search operation is performed, the first search result is located using its XPath. Click method of Selenium WebDriver is used to click on the first result, post which the window title is compared to check the success of the test.
* **test\_googlesearch\_\_load() —**Test URL https://www.lambdatest.comm/blog  is loaded and the window title is compared with the expected title. Assert is raised if the titles do not match.

close() command in Selenium WebDriver is used to close the browser window once the test execution is completed.

import pytest

import pytest\_html

from selenium import webdriver

from selenium.webdriver.chrome.options import Option

from selenium.webdriver.common.keys import Keys

import time

from time import sleep

import sys

@pytest.mark.usefixtures("driver\_init\_2")

class BasicTest:

pass

class Test\_URL\_Firefox(BasicTest):

def test\_google\_search(self):

self.driver.get('https://www.google.com/')

self.driver.maximize\_window()

title = "Google"

assert title == self.driver.title

search\_text = "LambdaTest"

search\_box = self.driver.find\_element\_by\_xpath("//input[@name='q']")

search\_box.send\_keys(search\_text)

time.sleep(5)

# Option 1 - To Submit the search

# search\_box.submit()

# Option 2 - To Submit the search

search\_box.send\_keys(Keys.ARROW\_DOWN)

search\_box.send\_keys(Keys.ARROW\_UP)

time.sleep(2)

search\_box.send\_keys(Keys.RETURN)

time.sleep(5)

# Click on the LambdaTest HomePage Link

title = "cross-browser Testing Tools | Free Automated Website Testing | LambdaTest"

lt\_link = self.driver.find\_element\_by\_xpath("//h3[.='LambdaTest: cross-browser Testing Tools | Free Automated ...']")

lt\_link.click()

time.sleep(10)

assert title == self.driver.title

time.sleep(2)

def test\_lambdatest\_blog\_load(self):

self.driver.get('https://www.lambdatest.com/blog/')

self.driver.maximize\_window()

expected\_title = "LambdaTest | A cross-browser Testing Blog"

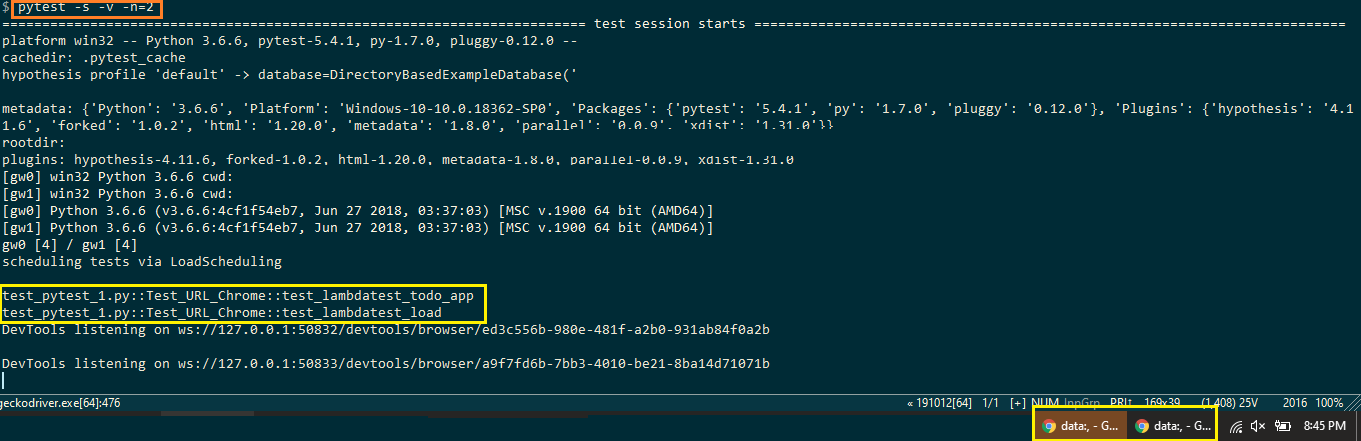
assert expected\_title == self.driver.title

time.sleep(5)

Two tests are performed in parallel. Hence, -n option (i.e. num\_of\_cpus) is set to 2. Execution is performed by invoking the following command on the terminal

pytest -s -v -n=2

Here is the screenshot of the execution which indicates that two tests are performed simultaneously (i.e. in parallel).



As seen below in this Selenium Python tutorial, all the four tests have passed to run Selenium tests in parallel with Python in the pytest.

