AVOCADO TESTING

By G.Varalaxmi
43370

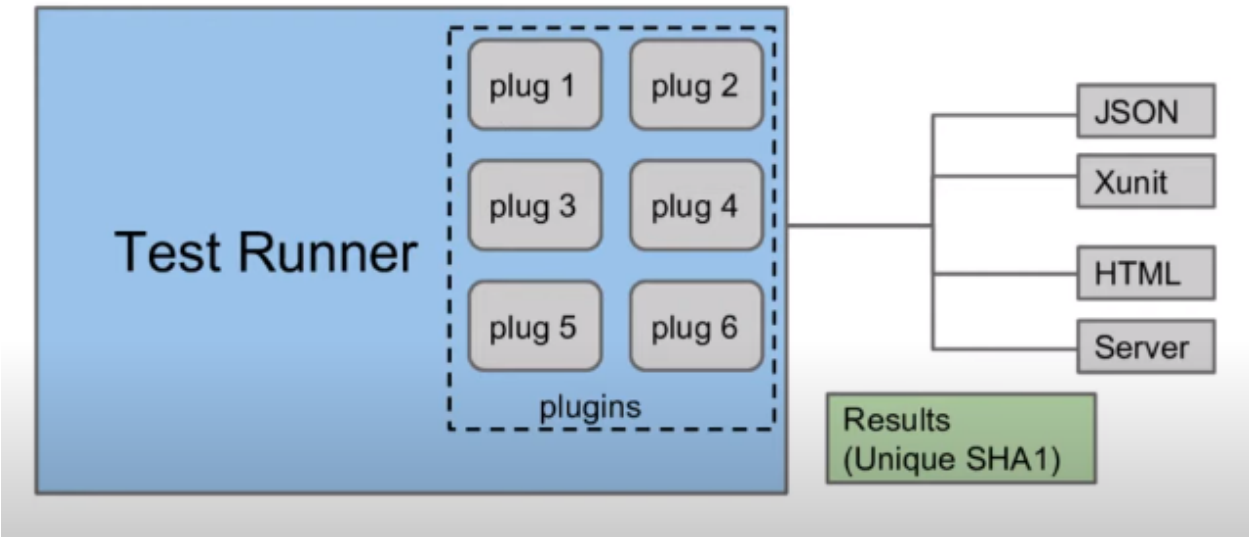
Index

1. Introduction

Avocado is a **versatile and robust** testing framework primarily used in the context of software and hardware testing.

- Versatility refers to Avocado's ability to adapt to a wide range of testing needs and scenarios.
  - **Different Types of Tests:**
    - Unit Tests: Tests for individual components or functions in isolation.
    - Integration Tests: Tests that verify the interaction between multiple components.
    - System Tests: Tests that validate the entire system's behaviour.
    - Functional Tests: Tests that check specific functionalities of the application.
    - Performance Tests: Tests that measure the performance aspects of the application or hardware.
  - **Multiple Platforms:**
    - Avocado can be used to test software on various operating systems (e.g., Linux, Windows, macOS).
    - It supports testing on different hardware platforms, such as embedded systems, servers, and IoT devices.
  - **Different Environments:**
    - Avocado can handle tests in different environments, such as local development machines, remote servers, virtual machines, and containers.
- Robustness refers to Avocado's reliability, stability, and comprehensive features that ensure thorough and dependable testing:
  - **Reliable Execution:**
    - Avocado ensures that tests are executed consistently and reliably, minimising the risk of flaky tests (tests that sometimes pass and sometimes fail without any changes to the code).

- **Error Handling:**
  - Avocado provides mechanisms for handling errors and exceptions during test execution, ensuring that tests can recover from unexpected conditions.
- **Detailed Reporting:**
  - Avocado generates detailed reports, including logs, test results, and other relevant data that help in analysing and debugging test outcomes
- **Isolation:**
  - Each test is run in isolation to prevent side effects from one test affecting another. This ensures that tests are independent and reliable



1.1. Installing

pip install avocado-framework

2. Avocado

- Avocado is a set of tools and libraries to help with automated testing.
- Avocado is a modern testing framework that can be used to write and run tests, particularly in complex environments like those needed for software integration and system testing.

2.1. Why Avocado Came?

The Problem:

Traditional testing tools, like those used for checking if a light bulb works, are not enough for these complex systems. These traditional tools might:

- **Lack Isolation**: They can't easily create isolated environments to test parts without interference.
- **Poor Environment Replication**: They struggle to mimic different deployment environments (e.g., different types of computers or operating systems).
- **Basic Reporting**: They don't provide detailed reports that help engineers understand what went wrong when a test fails.
- **Limited Scalability**: They can't handle large-scale testing efficiently.

The Solution: Avocado

Avocado was developed to address these issues. Here's why and how it helps:

- **Isolated Testing Environments**:
  - Avocado can run tests inside virtual machines (VMs) or containers. Think of these as miniature, isolated copies of a computer system. This means you can test parts of your system without them interfering with each other.
- **Environment Simulation**:
  - It can be different deployment environments. For example, you can test how your software runs on different operating systems without needing separate physical machines.
- **Detailed Reporting**:
  - When a test fails, Avocado provides detailed logs and reports. This helps engineers quickly understand what went wrong and how to fix it.
- **Scalability**:
  - Avocado can manage and run many tests in parallel. This is crucial for large-scale systems where you need to test many components simultaneously.

2.2. Why Traditional Tools Fall Short?

- **pytest** is fantastic for checking smaller, individual pieces of software (unit testing). For example, if you were building a simple app, pytest would be great to check if each feature works correctly.
- However, when you're testing something as complex as an entire airplane's software system, you need more robust tools to simulate different environments, manage large-scale testing, and provide detailed results. This is where Avocado shines.

2.3. Basic Concepts

```
# test_example.py

from avocado import Test

class ExampleTest(Test):
  def test_addition(self):
    result = 2 + 2
    self.assertEqual(result, 4, "Addition test failed")

  def test_subtraction(self):
    result = 5 - 3
    self.assertEqual(result, 2, "Subtraction test failed")
```

OUTPUT :

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> avocado run .\test.py
JOB ID     : d6dc3b406f28429800688b5d7fdf538336a3b17c
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-11T09.59-d6dc3
b4\job.log
```

(1/2) .\test.py:ExampleTest.test_addition: STARTED

(2/2) .\test.py:ExampleTest.test_subtraction: STARTED

Error running method "end_test" of plugin "bystatus": [WinError 1314] A r
equired privilege is not held by the client: '..\\..\\1-._test.py_Example
Test.test_addition' -> 'C:\\Users\\vlab/avocado/job-results\\job-2024-06-
11T09.59-d6dc3b4\\test-results\\by-status\\PASS\\1-._test.py_ExampleTest.
test_addition'

Reproduced traceback from: C:\Users\vlab\Desktop\AvocadoFramework\.venv\L
ib\site-packages\avocado\core\extension_manager.py:229
Traceback (most recent call last):
  File "C:\Users\vlab\Desktop\AvocadoFramework\.venv\Lib\site-packages\av
ocado\plugins\bystatus.py", line 42, in end_test
    os.symlink(
OSError: [WinError 1314] A required privilege is not held by the client:
'..\\..\\1-._test.py_ExampleTest.test_addition' -> 'C:\\Users\\vlab/avoca
do/job-results\\job-2024-06-11T09.59-d6dc3b4\\test-results\\by-status\\PA
SS\\1-._test.py_ExampleTest.test_addition'

(1/2) .\test.py:ExampleTest.test_addition: PASS (1.30 s)

Error running method "end_test" of plugin "bystatus": [WinError 1314] A r
equired privilege is not held by the client: '..\\..\\2-._test.py_Example
Test.test_subtraction' -> 'C:\\Users\\vlab/avocado/job-results\\job-2024-
06-11T09.59-d6dc3b4\\test-results\\by-status\\PASS\\2-._test.py_ExampleTe
st.test_subtraction'

Reproduced traceback from: C:\Users\vlab\Desktop\AvocadoFramework\.venv\L

ib\site-packages\avocado\core\extension_manager.py:229

Traceback (most recent call last):

  File "C:\Users\vlab\Desktop\AvocadoFramework\.venv\Lib\site-packages\av

ocado\plugins\bystatus.py", line 42, in end_test

    os.symlink(

OSError: [WinError 1314] A required privilege is not held by the client:

'..\\..\\2-._test.py_ExampleTest.test_subtraction' -> 'C:\\Users\\vlab/av

ocado/job-results\\job-2024-06-11T09.59-d6dc3b4\\test-results\\by-status\

\PASS\\2-._test.py_ExampleTest.test_subtraction'

(2/2) .\test.py:ExampleTest.test_subtraction: PASS (1.30 s)

RESULTS   : PASS 2 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |

CANCEL 0

JOB TIME   : 7.36 s

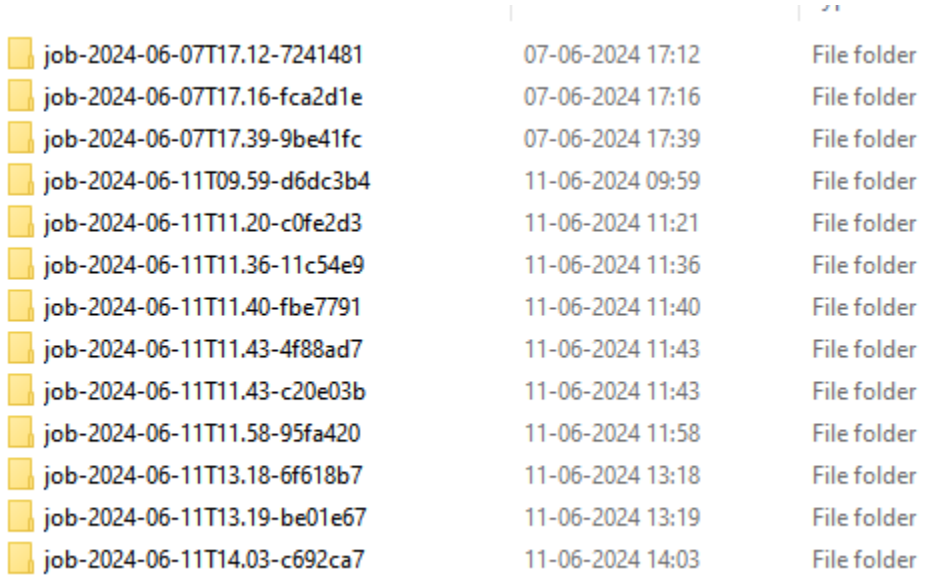
2.3.1. Job

- "job" refers to a logical unit of work that encompasses one or more test executions.
- job in the Avocado framework provides a structured way to organize and manage test executions, allowing testers to efficiently execute and analyze their tests within the framework.

2.3.1.1. avocado jobs list

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado jobs list**
c692ca745a5173f97b2d98ba0ca268c009ca5d25 2024-06-11 14:03:30.563308 4 (3/0/0/1)
be01e67fc2d82316ee07c876f591e4275c0d245a 2024-06-11 13:19:02.858332 2 (1/0/0/1)
6f618b7d8dba76c243a11ba3bbe6abc5e0cc2f98 2024-06-11 13:18:47.888082 2 (2/0/0/0)
95fa420be77c3d9f0e1201e5887b96cc0bb695f0 2024-06-11 11:58:35.237531 2 (2/0/0/0)
c20e03b8d6317dc4d3119c93186ec594f755a690 2024-06-11 11:43:48.191333 2 (1/0/0/1)

4f88ad7fd2212780237d9cd6633fbaac6338ddad 2024-06-11 11:43:32.606448 2 (2/0/0/0)
fbe7791629c7b5dd14cefd985829013351f5a366 2024-06-11 11:40:29.609991 4 (3/0/0/1)
11c54e97ea061000867d9fe78e9bf252a4c4ad62 2024-06-11 11:36:16.302812 4 (3/0/0/1)
c0fe2d3ebc356b9a51956e78b7ebf02a6faeaaa9 2024-06-11 11:20:52.976707 4 (3/0/0/1)
d6dc3b406f28429800688b5d7fdf538336a3b17c 2024-06-11 09:59:35.267822 2 (2/0/0/0)
9be41fc6aebb3fca79057636c06d1f12a0d8945f 2024-06-07 17:39:20.493964 2 (2/0/0/0)
fca2d1ec3618f982b255f19d16d502a37cad776f 2024-06-07 17:16:03.341231 2 (2/0/0/0)
724148161729267c9d7c3162de89992c8adb3852 2024-06-07 17:12:20.373460 2 (2/0/0/0)



job-2024-06-07T17.12-7241481	07-06-2024 17:12	File folder
job-2024-06-07T17.16-fca2d1e	07-06-2024 17:16	File folder
job-2024-06-07T17.39-9be41fc	07-06-2024 17:39	File folder
job-2024-06-11T09.59-d6dc3b4	11-06-2024 09:59	File folder
job-2024-06-11T11.20-c0fe2d3	11-06-2024 11:21	File folder
job-2024-06-11T11.36-11c54e9	11-06-2024 11:36	File folder
job-2024-06-11T11.40-fbe7791	11-06-2024 11:40	File folder
job-2024-06-11T11.43-4f88ad7	11-06-2024 11:43	File folder
job-2024-06-11T11.43-c20e03b	11-06-2024 11:43	File folder
job-2024-06-11T11.58-95fa420	11-06-2024 11:58	File folder
job-2024-06-11T13.18-6f618b7	11-06-2024 13:18	File folder
job-2024-06-11T13.19-be01e67	11-06-2024 13:19	File folder
job-2024-06-11T14.03-c692ca7	11-06-2024 14:03	File folder

2.3.2. Test Name

- Test name refers to the unique identifier or label given to a specific test case or test scenario. Test names are used to identify and distinguish individual tests within a test suite or job execution.
- For example :
  - .'\test.py'
  - '.\test1.py'

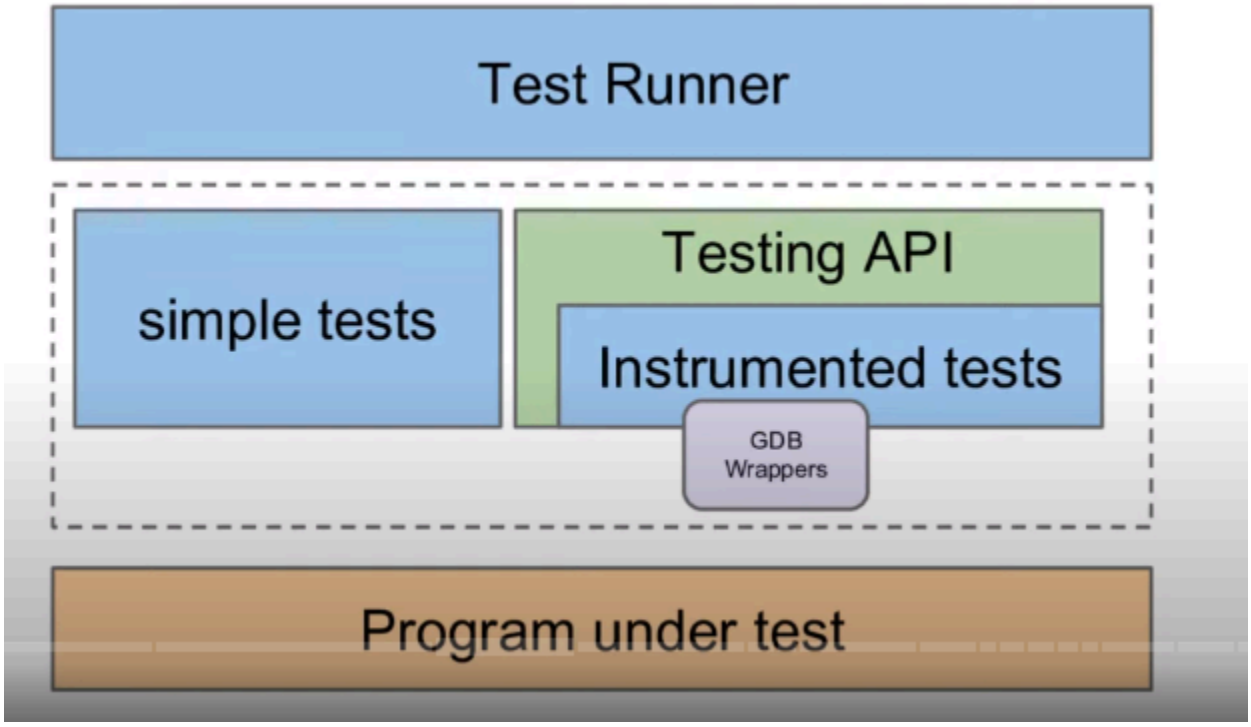
2.3.3. Variant IDs

- "Variant IDs" are unique identifiers assigned to different test configurations or setups. These IDs help distinguish between different variants of a test, allowing you to track and manage test executions more effectively.

2.3.4. Test ID

- A test ID is a string that uniquely identifies a test in the context of a job. When considering a single job, there are no two tests with the same ID.
- A test ID should encapsulate the Test Name and the Variant ID.

2.3.5. Test types



Avocado at its simplest configuration can run three different types of tests :

- **Simple** : The criteria for PASS/FAIL is the return code of the executable. If it returns 0, the test PASSes, if it returns anything else, it FAILs.
    - exec-test: On the other hand, "exec-test" indicates that Avocado is executing a test. This typically means that Avocado is running the test suite defined in the specified HTML file. It could involve running tests written in JavaScript, HTML, or any other language supported by Avocado for testing web applications or components.

- **Python unittest**

- **Instrumented :**
  - avocado-instrumented: When you see "avocado-instrumented" in Avocado's output, it means that Avocado has instrumented the specified Python file for testing. Instrumentation involves modifying the source code of the Python file to collect coverage data or to enable other types of analysis during testing. Avocado might instrument the code to track which lines are executed during the tests, for example.

2.4. Basic Operations

For example :

test.py

```
# test_example.py

from avocado import Test

class ExampleTest(Test):
  def test_addition(self):
     result = 2 + 2
     self.assertEqual(result, 4, "Addition test failed")

  def test_subtraction(self):
     result = 5 - 3
     self.assertEqual(result, 2, "Subtraction test failed")
```

test1.py

```
# test_example.py

from avocado import Test

class ExampleTest(Test):
  def test_addition(self):
     result = 2 + 2
     self.assertEqual(result, 5, "Addition test failed")

  def test_subtraction(self):
     result = 5 - 3
     self.assertEqual(result, 2, "Subtraction test failed")
```

OUTPUT :

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> avocado run .\test.py
.\test1.py
JOB ID     : c0fe2d3ebc356b9a51956e78b7ebf02a6faeaaa9
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-11T11.20-c0fe2
d3\job.log

 (1/4) .\test.py:ExampleTest.test_addition: STARTED
 (2/4) .\test.py:ExampleTest.test_subtraction: STARTED
 (3/4) .\test1.py:ExampleTest.test_addition: STARTED
 (4/4) .\test1.py:ExampleTest.test_subtraction: STARTED

 (2/4) .\test.py:ExampleTest.test_subtraction:  PASS (1.20 s)
 (4/4) .\test1.py:ExampleTest.test_subtraction:  PASS (1.48 s)
 (1/4) .\test.py:ExampleTest.test_addition:  PASS (1.64 s)
 (3/4) .\test1.py:ExampleTest.test_addition:  FAIL: 4 != 5 : Addition tes
t failed (2.59 s)

RESULTS    : PASS 3 | ERROR 0 | FAIL 1 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB TIME   : 7.64 s

Test summary:
3-.\test1.py:ExampleTest.test_addition: FAIL
```

3. Writing Avocado Tests

To write an Avocado test in python we are going to inherit from **avocado.Test** This test is called an **instrumented test.**

Basic example :

```
from time import sleep
from avocado import Test

class SleepTest(Test):
  def test(self):
     sleep_length = self.params.get('sleep_length',default=1)
     self.log.debug("Sleeping for %.2f seconds",sleep_length)
     sleep(sleep_length)
```

debug :

debug - Notepad

File Edit Format View Help

```
[stdlog] 2024-06-26 13:41:25,227 avocado.test test              L0310 INFO | INIT 1-.\sleeptest.py:SleepTest.test
[stdlog] 2024-06-26 13:41:25,228 avocado.test parameters        L0140 DEBUG| PARAMS (key=timeout, path=*, default=None) => None
[stdlog] 2024-06-26 13:41:25,228 avocado.test parameters        L0140 DEBUG| PARAMS (key=timeout_factor, path=*, default=1.0) => 1.0
[stdlog] 2024-06-26 13:41:25,228 avocado.test test              L0340 DEBUG| Test metadata:
[stdlog] 2024-06-26 13:41:25,228 avocado.test test              L0342 DEBUG|   filename: C:\Users\vlab\Desktop\FreeRTOSvara\AvocadoFramework\sleeptest.py
[stdlog] 2024-06-26 13:41:25,228 avocado.test test              L0348 DEBUG|   teststmpdir: C:\Users\vlab\AppData\Local\Temp\avocado_aulnmb8r
[stdlog] 2024-06-26 13:41:25,229 avocado.test test              L0349 DEBUG|   original timeout: None
[stdlog] 2024-06-26 13:41:25,229 avocado.test test              L0350 DEBUG|   timeout factor: 1.0
[stdlog] 2024-06-26 13:41:25,229 avocado.test test              L0351 DEBUG|   actual timeout: None
[stdlog] 2024-06-26 13:41:25,230 avocado.test test              L0524 INFO | START 1-.\sleeptest.py:SleepTest.test
[stdlog] 2024-06-26 13:41:25,230 avocado.test parameters        L0140 DEBUG| PARAMS (key=sleep_length, path=*, default=1) => 1
[stdlog] 2024-06-26 13:41:25,230 avocado.test sleeptest         L0007 DEBUG| Sleeping for 1.00 seconds
[stdlog] 2024-06-26 13:41:26,231 avocado.test test              L0720 INFO | PASS 1-.\sleeptest.py:SleepTest.test
[stdlog] 2024-06-26 13:41:26,232 avocado.test test              L0702 INFO |
```

3.1. What is an Avocado Test?

- An Avocado test is a method that starts with **test** in a class that inherits from **avocado.Test**
- Multiple tests in a single class names that start with test, like test_click,test_open etc..,
- Test class provides number of **convenience attributes :**
    - **self.log** → use log mechanism for test and lets log debug, info, error and warning messages.
    - **self.params** → Avocado allows passing parameters to tests.The data for **self.params** are supplied by **avocado.core.varianter.AvocadoParams** type which asks all registered plugins for variants or uses default when no variants are defined.

3.2. Test Statuses

- **PASS** → The test passed, which means all conditions being tested have passed.
- **FAIL** → The test failed, which means at least one condition being tested has failed. Ideally, it should mean a problem in the software being tested has been found.

- **ERROR** → An error happened during the test execution. This can happen, for example, if there's a bug in the test runner, in its libraries or if a resource breaks unexpectedly.it is usually caused by uncaught exception and such failures need to be explored and should lead to test modification to avoid this failure or to be used **self.fail**.
- **SKIP** → The test runner decided a requested test should not be run. This can happen, for example, due to missing requirements in the test environment or when there's a job timeout (nor its **setUp() and tearDown()**)
- **WARN** → The test ran and something might have gone wrong but didn't explicitly fail.
- **CANCEL** → The test was cancelled somewhere during the **setUp()** or **tearDown()** and didn't run.
- **INTERRUPTED** → The test was explicitly interrupted. Usually this means that a user hit **CTRL+C** while the job was still running or did not finish before the timeout specified.

3.3. Test methods

- To set the status is to use **self.fail**, **self.error** or **self.cancel** directly from the test.
- To warn **self.log.warning** logger. This won't interrupt the test execution, but it will remember the condition and if there are no failures, will report the test as **WARN**.

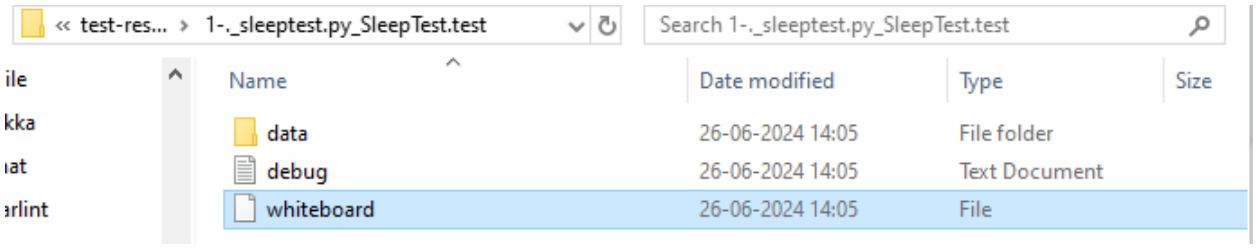
3.4. Turning errors into failures

- When Avocado runs a test, any unhandled exception will be seen as a test **ERROR** and not as a **FAIL.**
- Those exceptions would normally result in **ERROR** then should catch the exception and explain the failure in **self.fail** method:

```
from avocado.utils import process

try:
  # Run the command `stress_my_feature` and handle any errors
  result = process.run("stress_my_feature", ignore_status=False)
  self.log.info("Command output: %s" % result.stdout)
except process.CmdError as details:
  # Log the details of the error and fail the test
  self.fail("The stress command failed: %s" % details)
```

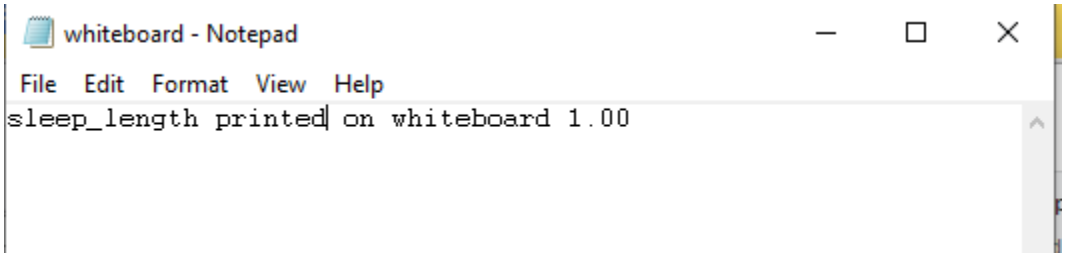
3.5. Saving test generated(custom) data by using whiteboard

- Each test instance provides a **whiteboard.** It can be accessed through **self.whiteboard**



- This whiteboard is simply a string that will be automatically saved to test results after the test finishes.

```
def test(self):
  # Fetch the sleep length parameter, defaulting to 1 if not provided
  sleep_length = self.params.get('sleep_length', default=1)
  self.log.debug("Sleeping for %.2f seconds", sleep_length)
  sleep(sleep_length)
  self.whiteboard = "sleep_length printed on whiteboard %.2f" % sleep_length
```



  - The **results.json** file already includes the whiteboard for each test.

3.6. Accessing test data files

- Some tests can depend on data files, external to the test file itself. Avocado provides a test API that makes it really easy to access such files : **get_data()**
- **Avocado**
- **Syntax : get_data(filename=' ',source='file',must_exist=False)**
- **get_data()** allows test data files to be accessed from up to three sources:
  - **file** level data directory : a directory named after the test file, but ending with **.data** For a test file **/home/user/** the file level data directory is **/home/user/test.py.data/.**
  - **Example :**

C:\Users\vlab\Desktop\Vara\AvocadoFramework>

```
        |
        |__ file1.py
        |__ file1.py.data
                |__ file1_data.txt
```

file1_data.txt :

```
username=example_user
password=example_pass
url=http://example.com
```

file1.py :

```python
from avocado import Test
class ExampleTest(Test):
  def test_data_file(self):
    # Use get_data() to get the path to the data file
    data_file_path = self.get_data('file1_data.txt')

    # Open and read the data file
    with open(data_file_path, 'r') as file:
      data = file.read()

    # Parse the data into a dictionary
    data_dict = dict(line.split('=') for line in data.splitlines())

    # Log the data and perform assertions
    self.log.info(f"Username: {data_dict['username']}")
    self.log.info(f"Password: {data_dict['password']}")
    self.log.info(f"URL: {data_dict['url']}")

    # Perform some assertions
    self.assertEqual(data_dict['username'], 'example_user')
    self.assertEqual(data_dict['password'], 'example_pass')
```

```
self.assertEqual(data_dict['url'], 'http://example.com')
```

debug :

```
[stdlog] 2024-06-26 15:55:12,196 avocado.test test              L0310 INFO | INIT 1-.\file1.py:ExampleTest.test_data_file
[stdlog] 2024-06-26 15:55:12,198 avocado.test parameters        L0140 DEBUG| PARAMS (key=timeout, path=*, default=None) => None
[stdlog] 2024-06-26 15:55:12,199 avocado.test parameters        L0140 DEBUG| PARAMS (key=timeout_factor, path=*, default=1.0) => 1.0
[stdlog] 2024-06-26 15:55:12,199 avocado.test test              L0340 DEBUG| Test metadata:
[stdlog] 2024-06-26 15:55:12,200 avocado.test test              L0342 DEBUG|   filename: C:\Users\vlab\Desktop\Vara\AvocadoFramework\file1.py
[stdlog] 2024-06-26 15:55:12,201 avocado.test test              L0348 DEBUG|   teststmpdir: C:\Users\vlab\AppData\Local\Temp\avocado_kzjul9fr
[stdlog] 2024-06-26 15:55:12,201 avocado.test test              L0349 DEBUG|   original timeout: None
[stdlog] 2024-06-26 15:55:12,201 avocado.test test              L0350 DEBUG|   timeout factor: 1.0
[stdlog] 2024-06-26 15:55:12,202 avocado.test test              L0351 DEBUG|   actual timeout: None
[stdlog] 2024-06-26 15:55:12,204 avocado.test test              L0524 INFO | START 1-.\file1.py:ExampleTest.test_data_file
[stdlog] 2024-06-26 15:55:12,206 avocado.test test              L0208 DEBUG| DATA (filename=file1_data.txt) => C:\Users\vlab\Desktop\Vara\AvocadoFramework\file1.py.data\f
[stdlog] 2024-06-26 15:55:12,207 avocado.test file1             L0017 INFO | Username: example_user
[stdlog] 2024-06-26 15:55:12,207 avocado.test file1             L0018 INFO | Password: example_pass
[stdlog] 2024-06-26 15:55:12,207 avocado.test file1             L0019 INFO | URL: http://example.com
[stdlog] 2024-06-26 15:55:12,209 avocado.test test              L0720 INFO | PASS 1-.\file1.py:ExampleTest.test_data_file
[stdlog] 2024-06-26 15:55:12,209 avocado.test test              L0702 INFO |
```

- **test** level data directory : a directory named after the test file and the specific test name. These are useful when different tests are part of the same file and different data files(with the same name or not). It contains two tests, **MyTest.test_foo** and **MyTest.test_bar,** the test level data directories will be **/home/user/test.py.data/MyTest.test_foo/** and **/home/user/test.py.data/MyTest.test_bar/.**
- **Example :**

```
C:\Users\vlab\Desktop\Vara\AvocadoFramework>
|
|__complex_test.py
|__complex_test.py.data/
        |__MyTest.test_math_operations/
        |        |__math_data.txt
        |__MyTest.test_string_operations/
                |__string_data.txt
```

complex_test.py

```
from avocado import Test

class MyTest(Test):

  def test_math_operations(self):
    data_file_path = self.get_data('math_data.txt')
```

```
        with open(data_file_path, 'r') as file:
            data = file.read()

        data_dict = dict(line.split('=') for line in data.splitlines())
        self.log.info(f"Operation: {data_dict['operation']}")
        self.log.info(f"Operand1: {data_dict['operand1']}")
        self.log.info(f"Operand2: {data_dict['operand2']}")

        operand1 = int(data_dict['operand1'])
        operand2 = int(data_dict['operand2'])
        if data_dict['operation'] == 'add':
            result = operand1 + operand2
            self.assertEqual(result, 30)
        elif data_dict['operation'] == 'subtract':
            result = operand1 - operand2
            self.assertEqual(result, 3)
        else:
            self.fail("Unsupported operation")

        self.log.info(f"Result: {result}")

    def test_string_operations(self):
        data_file_path = self.get_data('string_data.txt')
        with open(data_file_path, 'r') as file:
            data = file.read()

        data_dict = dict(line.split('=') for line in data.splitlines())
        self.log.info(f"Operation: {data_dict['operation']}")
        self.log.info(f"String1: {data_dict['string1']}")
        self.log.info(f"String2: {data_dict['string2']}")

        string1 = data_dict['string1']
        string2 = data_dict['string2']
        if data_dict['operation'] == 'concatenate':
```

```
        result = string1 + " " + string2
    else:
        self.fail("Unsupported operation")

    self.log.info(f"Result: {result}")
    self.assertEqual(result, "hello world")
```

math_data.txt

```
operation=add
operand1=10
operand2=20
operation=subtract
operand1=5
operand2=2
```

string_data.txt

```
operation=concatenate
string1=hello
string2=world
```

- **variant** level data directory : if variants are being used during the test execution a directory named after the variant will also be considered when looking for test data files. For test file **/home/user/test.py** and test **MyTest.test_foo** with variant **debug-ffff** , the data directory path will be **/home/user/test.py.data/MyTest.test_foo/debug-ffff/**
- **Example :**

```
C:\Users\vlab\Desktop\Vara\AvocadoFramework>
|
|__test.py
|__test.py.data/
        |__MyTest.test_foo/
                |__debug-ffff/
                        |__test_data.txt
```

test.py

```
from avocado import Test

class MyTest(Test):

  def test_variant_data(self):
    variant_name = self.params.get('variant', default='debug-ffff')
    data_file_path = self.get_data('variant_data.txt',variant='debug-ffff')

    with open(data_file_path, 'r') as file:
      data = file.read()

    data_dict = dict(line.split('=') for line in data.splitlines())
    self.log.info(f"Operation: {data_dict['operation']}")
    self.log.info(f"Operand1: {data_dict['operand1']}")
    self.log.info(f"Operand2: {data_dict['operand2']}")

    operand1 = int(data_dict['operand1'])
    operand2 = int(data_dict['operand2'])
    if data_dict['operation'] == 'add':
      result = operand1 + operand2
    else:
      self.fail("Unsupported operation")

    self.log.info(f"Result: {result}")
    self.assertEqual(result, 5)
```

test_data.txt

```
operation=add
operand1=2
operand2=3
```

3.7. Accessing test parameters

- Is a database of params present in every avocado test.

- Syntax : **self.params.get($name, $path=None,$default=None)**
    - name → name of the parameter
    - path → where to look for this parameter
    - default → what to return when param not found
- It accepts a list of TreeNode objects(**avocado.core.tree.TreeNode**)
- Test name **avocado.core.test.TestID** (for logging purposes)

3.8. Advanced logging capabilities

test_log.py

```
import logging
from time import sleep
from avocado import Test

progress_log = logging.getLogger("progress")

class Plant(Test):
  def test_plant_organic(self):
    rows = self.params.get("rows",default=3)
    for row in range(rows):
      progress_log.info("%s : preparing soil on row %s",self.name,row)

    progress_log.info("%s : letting soil rest before throwing seeds",self.name)
    sleep(2)

    for row in range(rows):
      progress_log.info("%s : throwing seeds on row %s",self.name,row)

    progress_log.info("%s : waiting for Avocados to grow",self.name)

    sleep(5)

    for row in range(rows):
      progress_log.info("%s : harvesting organic avocados on row %s",self.name,row)
```

debug :

debug - Notepad

File Edit Format View Help

```
[stdlog] 2024-06-26 17:58:32,063 avocado.test test            L0310 INFO | INIT 1-.\test_log.py:Plant.test_plant_organic
[stdlog] 2024-06-26 17:58:32,063 avocado.test parameters      L0140 DEBUG| PARAMS (key=timeout, path=*, default=None) => None
[stdlog] 2024-06-26 17:58:32,064 avocado.test parameters      L0140 DEBUG| PARAMS (key=timeout_factor, path=*, default=1.0) => 1.0
[stdlog] 2024-06-26 17:58:32,064 avocado.test test            L0340 DEBUG| Test metadata:
[stdlog] 2024-06-26 17:58:32,065 avocado.test test            L0342 DEBUG|   filename: C:\Users\vlab\Desktop\Vara\AvocadoFramework\test_log.py
[stdlog] 2024-06-26 17:58:32,065 avocado.test test            L0348 DEBUG|   teststmpdir: C:\Users\vlab\AppData\Local\Temp\avocado_wb27fum7
[stdlog] 2024-06-26 17:58:32,065 avocado.test test            L0349 DEBUG|   original timeout: None
[stdlog] 2024-06-26 17:58:32,065 avocado.test test            L0350 DEBUG|   timeout factor: 1.0
[stdlog] 2024-06-26 17:58:32,066 avocado.test test            L0351 DEBUG|   actual timeout: None
[stdlog] 2024-06-26 17:58:32,067 avocado.test test            L0524 INFO | START 1-.\test_log.py:Plant.test_plant_organic
[stdlog] 2024-06-26 17:58:32,067 avocado.test parameters      L0140 DEBUG| PARAMS (key=rows, path=*, default=3) => 3
[stdlog] 2024-06-26 17:58:32,067 progress test_log        L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic : preparing soil on row 0
[stdlog] 2024-06-26 17:58:32,067 progress test_log        L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic : preparing soil on row 1
[stdlog] 2024-06-26 17:58:32,067 progress test_log        L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic : preparing soil on row 2
[stdlog] 2024-06-26 17:58:32,067 progress test_log        L0013 INFO | 1-.\test_log.py:Plant.test_plant_organic : letting soil rest before throwing seeds
[stdlog] 2024-06-26 17:58:34,067 progress test_log        L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic : throwing seeds on row 0
[stdlog] 2024-06-26 17:58:34,067 progress test_log        L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic : throwing seeds on row 1
[stdlog] 2024-06-26 17:58:34,069 progress test_log        L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic : throwing seeds on row 2
[stdlog] 2024-06-26 17:58:34,069 progress test_log        L0019 INFO | 1-.\test_log.py:Plant.test_plant_organic : waiting for Avocados to grow
[stdlog] 2024-06-26 17:58:39,069 progress test_log        L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic : harvesting organic avocados on row 0
[stdlog] 2024-06-26 17:58:39,069 progress test_log        L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic : harvesting organic avocados on row 1
[stdlog] 2024-06-26 17:58:39,071 progress test_log        L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic : harvesting organic avocados on row 2
[stdlog] 2024-06-26 17:58:39,071 avocado.test test            L0720 INFO | PASS 1-.\test_log.py:Plant.test_plant_organic
[stdlog] 2024-06-26 17:58:39,071 avocado.test test            L0702 INFO |
```

3.8.1. avocado [--show STREAM[:LEVEL]] [test_file]

- List of comma separated builtin logs, or logging streams optionally followed by LEVEL (DEBUG,INFO,...). Builtin streams are: "app" : application output; "test": test output; "job": job output; "early": early logging of other streams, including test (very verbose); "all": all builtin streams; "none": disables regular output (leaving only errors enabled). By default: 'app'

3.8.1.1. Application output (app)

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado --show app,progress run .\test_log.py**

JOB ID : e75cdb46c51bf541021572a97c51ba9632a27b81

JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-06-27T09.53-e75cdb4\job.log

(1/1) .\test_log.py:Plant.test_plant_organic: STARTED

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:53:59,766 progress test_log L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic : preparing soil on row 0

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:53:59,766 progress test_log L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic : preparing soil on row 1

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:53:59,766 progress test_log L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic : preparing soil on row 2

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:53:59,766 progress test_log L0013 INFO | 1-.\test_log.py:Plant.test_plant_organic : letting soil rest before throwing seeds

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:01,767 progress test_log L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic : throwing seeds on row 0

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:01,767 progress test_log L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic : throwing seeds on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:01,767 progress test_log L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic : throwing seeds on row 2
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:01,768 progress test_log L0019 INFO | 1-.\test_log.py:Plant.test_plant_organic : waiting for Avocados to grow
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:06,770 progress test_log L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic : harvesting organic avocados on row 0
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:06,770 progress test_log L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic : harvesting organic avocados on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:54:06,770 progress test_log L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic : harvesting organic avocados on row 2

(1/1) .\test_log.py:Plant.test_plant_organic: PASS (7.77 s)
RESULTS : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB TIME : 11.55 s

3.8.1.2. Test output(test)

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado --show test,progress run .\test_log.py**

avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,013 avocado.test test L0310 INFO | INIT 1-.\test_log.py:Plant.test_plant_organic
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,014 avocado.test parameters L0140 DEBUG| PARAMS (key=timeout, path=*, default=None) => None
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,014 avocado.test parameters L0140 DEBUG| PARAMS (key=timeout_factor, path=*,default=1.0) => 1.0
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,014 avocado.test test L0340 DEBUG| Test metadata:
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,014 avocado.test test L0342 DEBUG| filename: C:\Users\vlab\Desktop\V
ara\AvocadoFramework\test_log.py

```
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,015 avocado.test
test            L0348 DEBUG|   teststmpdir: C:\Users\vlab\AppDat
a\Local\Temp\avocado_5hng0qt4
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,015 avocado.test
test            L0349 DEBUG|   original timeout: None
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,015 avocado.test
test            L0350 DEBUG|   timeout factor: 1.0
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,015 avocado.test
test            L0351 DEBUG|   actual timeout: None
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,016 avocado.test
test            L0524 INFO | START 1-.\test_log.py:Plant.test_pl
ant_organic
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,016 avocado.test
parameters       L0140 DEBUG| PARAMS (key=rows, path=*, default=3
) => 3
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,016 progress test_log
L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic :
preparing soil on row 0
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,016 progress test_log
L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic :
preparing soil on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,017 progress test_log
L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic :
preparing soil on row 2
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:50,017 progress test_log
L0013 INFO | 1-.\test_log.py:Plant.test_plant_organic :
letting soil rest before throwing seeds
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:52,017 progress test_log
L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic :
throwing seeds on row 0
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:52,017 progress test_log
L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic :
throwing seeds on row 1
```

progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:52,017 progress test_log L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic :
throwing seeds on row 2
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:52,017 progress test_log L0019 INFO | 1-.\test_log.py:Plant.test_plant_organic :
waiting for Avocados to grow
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:57,018 progress test_log L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic :
harvesting organic avocados on row 0
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:57,018 progress test_log L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic :
harvesting organic avocados on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:57,019 progress test_log L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic :
harvesting organic avocados on row 2
avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:57,020 avocado.test test L0720 INFO | PASS 1-.\test_log.py:Plant.test_pla
nt_organic

avocado.test: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 09:58:57,020 avocado.test test L0702 INFO |

3.8.1.3. Job output(job)

avocado.job: Command line:
C:\Users\vlab\Desktop\FreeRTOSvara\AvocadoFramework\.venv\Scripts\avocado --show job,progress run .\test_log.py
avocado.job:
avocado.job: Avocado version: 105.0
avocado.job:
avocado.job: Avocado config:
avocado.job:
avocado.job: {'assets.fetch.ignore_errors': False,
avocado.job: 'assets.fetch.references': [],
avocado.job: 'assets.fetch.timeout': 300,
avocado.job: 'assets.list.days': None,
avocado.job: 'assets.list.overall_limit': None,

```
avocado.job:  'assets.list.size_filter': None,
avocado.job:  'assets.purge.days': None,
avocado.job:  'assets.purge.overall_limit': None,
avocado.job:  'assets.purge.size_filter': None,
avocado.job:  'assets.register.name': None,
avocado.job:  'assets.register.sha1_hash': None,
avocado.job:  'assets.register.url': None,
avocado.job:  'cache.clear': [],
avocado.job:  'cache.list': [],
avocado.job:  'config': None,
avocado.job:  'config.datadir': False,
avocado.job:  'core.paginator': False,
avocado.job:  'core.show': {'job', 'progress'},
avocado.job:  'core.verbose': False,
avocado.job:  'datadir.paths.base_dir': 'C:\\Users\\vlab/avocado',
avocado.job:  'datadir.paths.cache_dirs': ['C:\\Users\\vlab/avocado/data/cache'],
avocado.job:  'datadir.paths.data_dir': 'C:\\Users\\vlab/avocado/data',
avocado.job:  'datadir.paths.logs_dir': 'C:\\Users\\vlab/avocado/job-results',
avocado.job:  'datadir.paths.test_dir': '/usr/share/doc/avocado/tests',
avocado.job:  'diff.create_reports': False,
avocado.job:  'diff.filter': ['cmdline', 'time', 'variants', 'results', 'config', 'sysinfo'],
avocado.job:  'diff.html': None,
avocado.job:  'diff.jobids': [],
avocado.job:  'diff.open_browser': False,
avocado.job:  'diff.strip_id': False,
avocado.job:  'distro.distro_def_arch': '',
avocado.job:  'distro.distro_def_create': False,
avocado.job:  'distro.distro_def_name': '',
avocado.job:  'distro.distro_def_path': '',
avocado.job:  'distro.distro_def_release': '',
avocado.job:  'distro.distro_def_type': '',
avocado.job:  'distro.distro_def_version': '',
avocado.job:  'filter.by_tags.include_empty': False,
avocado.job:  'filter.by_tags.include_empty_key': False,
avocado.job:  'filter.by_tags.tags': [],
avocado.job:  'human_ui.omit.statuses': [],
avocado.job:  'job.output.loglevel': 'DEBUG',
avocado.job:  'job.output.testlogs.logfiles': ['debug.log'],
avocado.job:  'job.output.testlogs.statuses': [],
avocado.job:  'job.output.testlogs.summary_statuses': ['ERROR', 'FAIL', 'INTERRUPTED'],
avocado.job:  'job.replay.source_job_id': 'latest',
avocado.job:  'job.run.dependency': None,
avocado.job:  'job.run.result.json.enabled': True,
avocado.job:  'job.run.result.json.output': None,
```

```
avocado.job:  'job.run.result.tap.enabled': True,
avocado.job:  'job.run.result.tap.include_logs': False,
avocado.job:  'job.run.result.tap.output': None,
avocado.job:  'job.run.result.xunit.enabled': True,
avocado.job:  'job.run.result.xunit.job_name': None,
avocado.job:  'job.run.result.xunit.max_test_log_chars': 100000,
avocado.job:  'job.run.result.xunit.output': None,
avocado.job:  'job.run.store_logging_stream': [],
avocado.job:  'job.run.timeout': 0,
avocado.job:  'jobs.show.job_id': 'latest',
avocado.job:  'json.variants.load': None,
avocado.job:  'list.recipes.write_to_directory': None,
avocado.job:  'list.write_to_json_file': None,
avocado.job:  'plugins.cache.order': [],
avocado.job:  'plugins.cli.cmd.order': [],
avocado.job:  'plugins.cli.order': [],
avocado.job:  'plugins.disable': [],
avocado.job:  'plugins.init.order': [],
avocado.job:  'plugins.job.prepost.order': [],
avocado.job:  'plugins.jobscripts.post': '/etc/avocado/scripts/job/post.d/',
avocado.job:  'plugins.jobscripts.pre': '/etc/avocado/scripts/job/pre.d/',
avocado.job:  'plugins.jobscripts.warn_non_existing_dir': False,
avocado.job:  'plugins.jobscripts.warn_non_zero_status': True,
avocado.job:  'plugins.ordered_list': False,
avocado.job:  'plugins.resolver.order': [],
avocado.job:  'plugins.result.order': [],
avocado.job:  'plugins.result_events.order': [],
avocado.job:  'plugins.runnable.runner.order': [],
avocado.job:  'plugins.skip_broken_plugin_notification': [],
avocado.job:  'plugins.spawner.order': [],
avocado.job:  'plugins.suite.runner.order': [],
avocado.job:  'plugins.test.post.order': [],
avocado.job:  'plugins.test.pre.order': [],
avocado.job:  'plugins.varianter.order': [],
avocado.job:  'resolver.references': ['.\\test_log.py'],
avocado.job:  'run.dict_variants': [],
avocado.job:  'run.dict_variants.variant_id_keys': [],
avocado.job:  'run.dry_run.enabled': False,
avocado.job:  'run.dry_run.no_cleanup': False,
avocado.job:  'run.execution_order': 'variants-per-test',
avocado.job:  'run.failfast': False,
avocado.job:  'run.ignore_missing_references': False,
avocado.job:  'run.job_category': None,
avocado.job:  'run.journal.enabled': False,
```

```
avocado.job:  'run.keep_tmp': False,
avocado.job:  'run.log_test_data_directories': False,
avocado.job:  'run.max_parallel_tasks': 4,
avocado.job:  'run.results.archive': False,
avocado.job:  'run.results_dir': None,
avocado.job:  'run.shuffle': False,
avocado.job:  'run.spawner': 'process',
avocado.job:  'run.status_server_auto': True,
avocado.job:  'run.status_server_buffer_size': 33554432,
avocado.job:  'run.status_server_listen': '127.0.0.1:8888',
avocado.job:  'run.status_server_uri': '127.0.0.1:8888',
avocado.job:  'run.suite_runner': 'nrunner',
avocado.job:  'run.test_parameters': [],
avocado.job:  'run.unique_job_id': None,
avocado.job:  'runner.exectest.clear_env': None,
avocado.job:  'runner.exectest.exitcodes.skip': [],
avocado.job:  'runner.identifier_format': '{uri}',
avocado.job:  'runner.output.color': 'auto',
avocado.job:  'runner.output.colored': True,
avocado.job:  'runner.task.interval.from_hard_termination_to_verification': 0,
avocado.job:  'runner.task.interval.from_soft_to_hard_termination': 1,
avocado.job:  'spawner.lxc.arch': 'i386',
avocado.job:  'spawner.lxc.create_hook': '',
avocado.job:  'spawner.lxc.dist': 'fedora',
avocado.job:  'spawner.lxc.release': '32',
avocado.job:  'spawner.lxc.slots': [],
avocado.job:  'spawner.podman.avocado_spawner_egg': None,
avocado.job:  'spawner.podman.bin': '/usr/bin/podman',
avocado.job:  'spawner.podman.image': 'fedora:latest',
avocado.job:  'spawner.podman.image_tag_prefix': 'avocado_generated',
avocado.job:  'subcommand': 'run',
avocado.job:  'sysinfo.collect.commands_timeout': -1,
avocado.job:  'sysinfo.collect.enabled': True,
avocado.job:  'sysinfo.collect.installed_packages': False,
avocado.job:  'sysinfo.collect.locale': 'C',
avocado.job:  'sysinfo.collect.optimize': False,
avocado.job:  'sysinfo.collect.per_test': False,
avocado.job:  'sysinfo.collect.profiler': False,
avocado.job:  'sysinfo.collect.sysinfodir': None,
avocado.job:  'sysinfo.collectibles.commands':
'c:\\users\\vlab\\desktop\\freertosvara\\avocadoframework\\.venv\\lib\\site-packages\\avocado\\et
c/avocado/sysi
nfo/commands',
```

```
avocado.job:  'sysinfo.collectibles.fail_commands':
'c:\\users\\vlab\\desktop\\freertosvara\\avocadoframework\\.venv\\lib\\site-packages\\avocado\\et
c/avocado
/sysinfo/fail_commands',
avocado.job:  'sysinfo.collectibles.fail_files':
'c:\\users\\vlab\\desktop\\freertosvara\\avocadoframework\\.venv\\lib\\site-packages\\avocado\\et
c/avocado/sy
sinfo/fail_files',
avocado.job:  'sysinfo.collectibles.files':
'c:\\users\\vlab\\desktop\\freertosvara\\avocadoframework\\.venv\\lib\\site-packages\\avocado\\et
c/avocado/sysinfo
/files',
avocado.job:  'sysinfo.collectibles.profilers':
'c:\\users\\vlab\\desktop\\freertosvara\\avocadoframework\\.venv\\lib\\site-packages\\avocado\\et
c/avocado/sys
info/profilers',
avocado.job:  'task.timeout.running': None,
avocado.job:  'variants.contents': False,
avocado.job:  'variants.debug': False,
avocado.job:  'variants.inherit': False,
avocado.job:  'variants.json_variants_dump': None,
avocado.job:  'variants.summary': 0,
avocado.job:  'variants.tree': False,
avocado.job:  'variants.variants': 1,
avocado.job:  'vmimage.get.arch': None,
avocado.job:  'vmimage.get.distro': None,
avocado.job:  'vmimage.get.version': None}
avocado.job:
avocado.job: Avocado Data Directories:
avocado.job:
avocado.job: base     C:\Users\vlab/avocado
avocado.job: tests    C:\Users\vlab\Desktop\FreeRTOSvara\AvocadoFramework\.venv\tests
avocado.job: data     C:\Users\vlab/avocado/data
avocado.job: logs     C:\Users\vlab/avocado/job-results\job-2024-06-27T10.03-7e95883
avocado.job:
avocado.job: Temporary dir:
C:\Users\vlab\AppData\Local\Temp\avocado_tmp_7yu6xby4\avocado_job_8zns0o1n
avocado.job:
avocado.job: Job ID: 7e95883fe9d9edcc8ab84693579f92876d4127d2
avocado.job:
avocado.job: .\test_log.py:Plant.test_plant_organic: STARTED
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:26,211 progress test_log
L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic :
preparing soil on row 0
```

```
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:26,211 progress test_log
L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic :
preparing soil on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:26,211 progress test_log
L0011 INFO | 1-.\test_log.py:Plant.test_plant_organic :
preparing soil on row 2
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:26,211 progress test_log
L0013 INFO | 1-.\test_log.py:Plant.test_plant_organic :
letting soil rest before throwing seeds
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:28,212 progress test_log
L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic :
throwing seeds on row 0
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:28,212 progress test_log
L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic :
throwing seeds on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:28,212 progress test_log
L0017 INFO | 1-.\test_log.py:Plant.test_plant_organic :
throwing seeds on row 2
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:28,213 progress test_log
L0019 INFO | 1-.\test_log.py:Plant.test_plant_organic :
waiting for Avocados to grow
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:33,213 progress test_log
L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic :
harvesting organic avocados on row 0
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:33,213 progress test_log
L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic :
harvesting organic avocados on row 1
progress: 1-.\test_log.py:Plant.test_plant_organic: 2024-06-27 10:03:33,213 progress test_log
L0024 INFO | 1-.\test_log.py:Plant.test_plant_organic :
harvesting organic avocados on row 2
Error running method "end_test" of plugin "bystatus": [WinError 1314] A required privilege is not
held by the client: '..\\..\\1-._test_log.py_Plant.test_plan
t_organic' ->
'C:\\Users\\vlab/avocado/job-results\\job-2024-06-27T10.03-7e95883\\test-results\\by-status\\PA
SS\\1-._test_log.py_Plant.test_plant_organic'

Reproduced traceback from:
C:\Users\vlab\Desktop\FreeRTOSvara\AvocadoFramework\.venv\Lib\site-packages\avocado\cor
e\extension_manager.py:229
Traceback (most recent call last):
  File
"C:\Users\vlab\Desktop\FreeRTOSvara\AvocadoFramework\.venv\Lib\site-packages\avocado\pl
ugins\bystatus.py", line 42, in end_test
    os.symlink(
```

OSError: [WinError 1314] A required privilege is not held by the client: '..\\..\\1-._test_log.py_Plant.test_plant_organic' -> 'C:\\Users\\vlab/avocado/job-results\\job-2024-06-27T10.03-7e95883\\test-results\\by-status\\PASS\\1-._test_log.py_Plant.test_plant_organic'

avocado.job: .\test_log.py:Plant.test_plant_organic: PASS
avocado.job: More information in C:\Users\vlab/avocado/job-results\job-2024-06-27T10.03-7e95883\test-results\1-._test_log.py_Plant.test_plant_organic
avocado.job: Test results available in C:\Users\vlab/avocado/job-results\job-2024-06-27T10.03-7e95883

3.9. unittest.TestCase heritage

- Since an avocado test inherits from unittest.TestCase

```
from avocado import Test

class RandomExp(Test):
  def test(self):
    self.log.debug("Verifying some random math...")
    four = 2 * 2
    four_ = 2 + 2
    self.assertEqual(four,four_,"something is wrong here!")

    self.log.debug("Verifying if a variable is set to True...")
    var1 = True
    self.assertTrue(var1)

    self.log.debug("Verifying if a variable is set to False...")
    var2 = False
    self.assertFalse(var2)

    self.log.debug("Verifying if this test is an instance of test.Test")
    self.assertIsInstance(self,Test)
```

debug :

debug - Notepad

File Edit Format View Help

```
[stdlog] 2024-06-27 14:14:28,267 avocado.test test              L0310 INFO | INIT 1-.\unittest.py:RandomExp.test
[stdlog] 2024-06-27 14:14:28,268 avocado.test parameters        L0140 DEBUG| PARAMS (key=timeout, path=*, default=None) => None
[stdlog] 2024-06-27 14:14:28,268 avocado.test parameters        L0140 DEBUG| PARAMS (key=timeout_factor, path=*, default=1.0) => 1.0
[stdlog] 2024-06-27 14:14:28,268 avocado.test test              L0340 DEBUG| Test metadata:
[stdlog] 2024-06-27 14:14:28,268 avocado.test test              L0342 DEBUG|   filename: C:\Users\vlab\Desktop\Vara\AvocadoFramework\unittest.py
[stdlog] 2024-06-27 14:14:28,268 avocado.test test              L0348 DEBUG|   teststmpdir: C:\Users\vlab\AppData\Local\Temp\avocado_cj5kly9t
[stdlog] 2024-06-27 14:14:28,269 avocado.test test              L0349 DEBUG|   original timeout: None
[stdlog] 2024-06-27 14:14:28,269 avocado.test test              L0350 DEBUG|   timeout factor: 1.0
[stdlog] 2024-06-27 14:14:28,269 avocado.test test              L0351 DEBUG|   actual timeout: None
[stdlog] 2024-06-27 14:14:28,270 avocado.test test              L0524 INFO | START 1-.\unittest.py:RandomExp.test
[stdlog] 2024-06-27 14:14:28,270 avocado.test unittest          L0005 DEBUG| Verifying some random math...
[stdlog] 2024-06-27 14:14:28,270 avocado.test unittest          L0010 DEBUG| Verifying if a variable is set to True...
[stdlog] 2024-06-27 14:14:28,270 avocado.test unittest          L0014 DEBUG| Verifying if a variable is set to False...
[stdlog] 2024-06-27 14:14:28,270 avocado.test unittest          L0018 DEBUG| Verifying if this test is an instance of test.Test
[stdlog] 2024-06-27 14:14:28,270 avocado.test test              L0720 INFO | PASS 1-.\unittest.py:RandomExp.test
[stdlog] 2024-06-27 14:14:28,271 avocado.test test              L0702 INFO |
```

Running tests under other unittest runners
nose is another python testing framework that is also compatible with unittest.

3.10. Setup and cleanup methods

- By using **setUp and tearDown** methods, To perform setup actions before/after your test.
- The **tearDown** method is always executed even on **setUp** failure so don't forget to initialize your variables early in the **setUp**.

3.11. Skipping Tests

To skip tests is in Avocado

- **@avocado.skip(reason)** : Skips a test
- **@avocado.skipIf(condition, reason)** : Skip a test if the condition is **True**.
- **@avocado.skipUnless(condition, reason)** : Skips a test if the condition is **False.**

```
import  avocado

class MyTest(avocado.Test):

  @avocado.skipIf(1==1,"skipping on True condition..")
  def test1(self):
     pass

  @avocado.skip("Don't want this test now..")
  def test2(self):
     pass

  @avocado.skipUnless(1==1,"Skipping on False condition..")
  def test3(self):
     pass
```

```
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run .\ski
p.py
JOB ID     : 57bc17a67ca78873d7a1416460487fa4704ca06f
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-27T15.31-57bc1
7a\job.log
 (1/3) .\skip.py:MyTest.test1: STARTED
 (2/3) .\skip.py:MyTest.test2: STARTED
 (3/3) .\skip.py:MyTest.test3: STARTED
```

```
 (3/3) .\skip.py:MyTest.test3:  PASS (0.72 s)
 (1/3) .\skip.py:MyTest.test1:  SKIP: skipping on True condition..
 (2/3) .\skip.py:MyTest.test2:  SKIP: Don't want this test now..

RESULTS    : PASS 1 | ERROR 0 | FAIL 0 | SKIP 2 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB TIME   : 4.92 s
```

- Using the skip decorators, nothing is actually executed. We will skip the **setUp()** method, the test method and the **tearDown()** method.
- It's an erroneous condition, reported with test status **ERROR,** to use any of the skip decorators on the **tearDown()** method.

3.12.Cancelling Tests

```
from avocado import Test
class CancelTest(Test):
  def test1(self):
     if 1 != 0 :
        self.cancel("statement is not true")
```

```
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run .\Can
celling.py
JOB ID     : b8a577ef62dd581c645f93cc9aa0036355715d76
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-27T16.07-b8a57
7e\job.log
 (1/1) .\Cancelling.py:CancelTest.test1: STARTED
 (1/1) .\Cancelling.py:CancelTest.test1:  CANCEL: statement is not true (
0.72 s)
RESULTS    : PASS 0 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 1
JOB TIME   : 4.36 s
```

- Using the **self.cancel()** will cancel the rest of the test from that point on, but the **tearDown()** will still be executed.

3.13. Docstring Directives

Case 1: The derived class inherits from the base class without using the: avocado: enable, disable, then all test cases are tested.

```
from avocado import Test

class BaseClass(Test):

  def test_shared(self):
    pass

class DerivedClass(BaseClass):

  def test_specific(self):
    pass
```

```
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run
.\non_instrumented.py
JOB ID     : 41b6f58ad592f75761ceb059ecabb9ba430af784
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-07-19T11.45-41b6f
58\job.log
 (1/3) .\non_instrumented.py:BaseClass.test_shared: STARTED
 (2/3) .\non_instrumented.py:DerivedClass.test_specific: STARTED
 (3/3) .\non_instrumented.py:DerivedClass.test_shared: STARTED

 (1/3) .\non_instrumented.py:BaseClass.test_shared:  PASS (0.92 s)
 (2/3) .\non_instrumented.py:DerivedClass.test_specific:  PASS (0.91 s)
 (3/3) .\non_instrumented.py:DerivedClass.test_shared:  PASS (0.91 s)
RESULTS    : PASS 3 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB HTML   : C:\Users\vlab/avocado/job-results\job-2024-07-19T11.45-41b6f
58\results.html
JOB TIME   : 4.69 s
```

3.13.1. Declaring test as INSTRUMENTED

```
from avocado import Test

class BaseClass(Test):
  """
  :avocado: enable
  """
  def test_shared(self):
    pass

class DerivedClass(BaseClass):
  """
  :avocado: disable
  """
```

```
  def test_specific(self):
    pass
```

Output :

```
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run
.\non_instrumented.py
JOB ID     : 1c852442836289359a86787e1f3a7d6b5a4f86f9
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-07-19T14.16-1c852
44\job.log
 (1/1) .\non_instrumented.py:BaseClass.test_shared: STARTED

 (1/1) .\non_instrumented.py:BaseClass.test_shared:  PASS (0.83 s)
RESULTS    : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB HTML   : C:\Users\vlab/avocado/job-results\job-2024-07-19T14.16-1c852
44\results.html
JOB TIME   : 4.66 s
```

Case 3 :

```
from avocado import Test

class BaseClass(Test):
  """
  :avocado: disable
  """
  def test_shared(self):
    pass

class DerivedClass(BaseClass):
  """
  :avocado: enable
  """
  def test_specific(self):
    pass
```

Output :

```
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run
.\non_instrumented.py
JOB ID     : bfca5957ee3a9bb4003049ecaeadc79bc005ac30
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-07-19T14.18-bfca5
95\job.log
 (1/1) .\non_instrumented.py:DerivedClass.test_specific: STARTED
 (1/1) .\non_instrumented.py:DerivedClass.test_specific:  PASS (0.88 s)
```

RESULTS : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB HTML : C:\Users\vlab/avocado/job-results\job-2024-07-19T14.18-bfca595\results.html
JOB TIME : 5.11 s

Case 4 :

```
from avocado import Test

class BaseClass:
  """
  :avocado: enable
  """
  def test_shared(self):
     pass

'''class DerivedClass(BaseClass):
  """
  :avocado: enable
  """
  def test_specific(self):
     pass'''
```

Output :
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run .\non_instrumented.py**
JOB ID : 6a61547ece1f58701dabfb106532ddbe0e3b592d
JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-07-19T14.20-6a61547\job.log
(1/1) .\non_instrumented.py:BaseClass.test_shared: STARTED
(1/1) .\non_instrumented.py:BaseClass.test_shared: ERROR: Failed to find/load class "BaseClass" in ".\non_instrumented.py" (0.84 s)
RESULTS : PASS 0 | ERROR 1 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB HTML : C:\Users\vlab/avocado/job-results\job-2024-07-19T14.20-6a61547\results.html
JOB TIME : 4.88 s

Test summary:
1-.\non_instrumented.py:BaseClass.test_shared: ERROR

3.13.2. Declaring test as NON-INSTRUMENTED

```
from avocado import Test
```

```
class BaseClass:
  """
  :avocado: disable
  """
  def test_shared(self):
     pass

'''class DerivedClass(BaseClass):
  """
  :avocado: enable
  """
  def test_specific(self):
     pass'''
```

Output :

(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run .\non_instrumented.py**

No tests found for given test references: .\non_instrumented.py

Try 'avocado -V list .\non_instrumented.py' for details

4. Avocado Plugins

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado plugins**

Plugins that add new commands (cli.cmd):

assets Manage assets

cache Interface for manipulating the Avocado cache metadata

config Shows avocado config keys

diff Shows the difference between 2 jobs.

distro Shows detected Linux distribution

exec-path Returns path to avocado bash libraries and exits.

jobs Manage Avocado jobs

list List available tests

plugins Displays plugin information

replay Runs a new job using a previous job as its configuration

run Runs one or more tests (native test, test alias, binary or scri

pt)

```
sysinfo   Collect system information
variants  Tool to analyze and visualize test variants and params
vmimage   Provides VM images acquired from official repositories

Plugins that add new options to commands (cli):
html          HTML job report options for 'run' subcommand
journal       Journal options for the 'run' subcommand
json          JSON output options for 'run' command
json_variants JSON serialized based Varianter options for the 'run' subco
mmand
nrunner       nrunner command line options for "run"
podman        podman spawner command line options for "run"
tap           TAP - Test Anything Protocol results
xunit         xUnit output options
zip_archive   Result archive (ZIP) support to run command

Plugins that manipulates with avocado cache (cache):
requirement Provides requirement cache entries

Plugins that always need to be initialized (init):
dict_variants Python Dictionary based varianter
html          HTML job report options initialization
human         Initialize human ui plugin settings
jobscripts    Jobscripts plugin initialization
json_variants JSON serialized based varianter initialization
jsonresult    JSON job result plugin initialization
lxc           LXC (container) based spawner initialization
nrunner       nrunner initialization
podman        Podman (container) based spawner initialization
run           Initializes the run options
sysinfo       Initializes sysinfo settings
tap           TAP - Test Anything Protocol - result plugin initialization
testlogsui    Initialize testlogs plugin settings
xunit         xUnit job result initialization

Plugins that run before/after the execution of jobs (job.prepost):
human            Human Interface UI
jobscripts       Runs scripts before/after the job is run
suite-dependency Applies a set of dependencies to every test within the s
uite
testlogsui       Shows content from tests' logs
teststmpdir      Creates a temporary directory for tests consumption

Plugins that generate job result in different formats (result):
```

```
html        HTML result support
json        JSON result support
xunit       XUnit result support
zip_archive Result archive (ZIP) support

Plugins that generate job result based on job/test events (result_events)
:
beaker      report results to beaker
bystatus    Creates symlinks on file system grouped by status
fetchasset  Fetch assets before the test run
human       Human Interface UI
journal     Journal event based results implementation
sysinfo     Collects system information before/after the job is run
tap         TAP - Test Anything Protocol results
testlogging Nrunner specific Test logs for Job

Plugins that run runnables (under a task and spawner) (runnable.runner):
asset               Runner for dependencies of type package
avocado-instrumented Runner for avocado-instrumented tests
dry-run             Runner for --dry-run
exec-test           Runner for standalone executables treated as tests
noop                Sample runner that performs no action before reporti
ng FINISHED status
package             Runner for dependencies of type package
podman-image        Runner for dependencies of type podman-image
python-unittest     Runner for Python unittests
sysinfo             Runner for gathering sysinfo
tap                 Runner for standalone executables treated as TAP

Plugins that spawn tasks and know about their status (spawner):
lxc     LXC (container) based spawner
podman  Podman (container) based spawner
process Process based spawner

Plugins that run test suites on a job (suite.runner):
nrunner nrunner based implementation of job compliant runner

Plugins that run after the execution of each test (test.post):
sysinfo Collects system information before/after the test is run.

Plugins that run before the execution of each test (test.pre):
dependency Dependency resolver for tests with dependencies
sysinfo    Collects system information before/after the test is run.
```

Plugins that generate test variants (varianter):
dict_variants Python Dictionary based varianter
json_variants JSON serialized based Varianter

Plugins that discover tests without references (discoverer):
(No active plugin)

Plugins that resolve test references (resolver):
avocado-instrumented Test resolver for Avocado Instrumented tests
exec-test Test resolver for executable files to be handled as
tests
python-unittest Test resolver for Python Unittests
runnable-recipe Test resolver for JSON runnable recipes
runnables-recipe Test resolver for multiple runnables in a JSON recip
e file
tap Test resolver for executable files to be handled as
TAP tests

4.1. diff

usage: avocado diff [-h] [--html FILE] [--open-browser]
[--diff-filter DIFF_FILTER] [--diff-strip-id]
[--create-reports]
JOB JOB

positional arguments:
JOB A job reference, identified by a (partial)
unique ID (SHA1) or test results directory.

options:
-h, --help → show this help message and exit
--html FILE → Enable HTML output to the FILE where the result should be written.
--open-browser → Generate and open a HTML report in your preferred browser. If
no --html file is provided, create a temporary file.

--diff-filter DIFF_FILTER
Comma separated filter of diff sections:
(no)cmdline,(no)time,(no)variants,(no)results,
(no)config,(no)sysinfo (defaults to all
enabled).

--diff-strip-id Strip the "id" from "id-name;variant" when
comparing test results.

```
  --create-reports      Create temporary files with job reports to be
                        used by other diff tools
```

By default, a textual diff report is generated in the standard output.

- Avocado Diff plugin allows users to easily compare several aspects of two given jobs.
- Create an unified diff of :
    - Command line
    - Job time
    - Variants and parameters : Avocado supports parameterization, allowing you to run the same test with different input values or parameters. For example, you can test a function with different sets of input data or test different scenarios by varying parameters in your test cases.
    - Test Results.
    - Configuration
    - Sysinfo pre and post.

4.1.1. avocado diff JOB JOB

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado diff be01e67fc 2d82316ee07c876f591e4275c0d245a 6f618b7d8dba76c243a11ba3bbe6abc5e0cc2f98**

```
--- be01e67fc2d82316ee07c876f591e4275c0d245a
+++ 6f618b7d8dba76c243a11ba3bbe6abc5e0cc2f98
@@ -1,11 +1,11 @@
Avocado Job Report

# COMMAND LINE
-C:\Users\vlab\Desktop\AvocadoFramework\.venv\Scripts\avocado run .\test1
.py
+C:\Users\vlab\Desktop\AvocadoFramework\.venv\Scripts\avocado run .\test.
py

# TOTAL TIME
-1.69 s
+1.81 s

# TEST RESULTS
-2-.\test1.py:ExampleTest.test_subtraction: PASS
-1-.\test1.py:ExampleTest.test_addition: FAIL
+1-.\test.py:ExampleTest.test_addition: PASS
+2-.\test.py:ExampleTest.test_subtraction: PASS
```

4.1.2. avocado diff JOB JOB --html FILE_NAME

test.py

```
# test_example.py

from avocado import Test

class ExampleTest(Test):
  def test_addition(self):
     result = 2 + 2
     self.assertEqual(result, 4, "Addition test failed")

  def test_subtraction(self):
     result = 5 - 3
     self.assertEqual(result, 2, "Subtraction test failed")
```

OUTPUT :

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> avocado run .\test.py
JOB ID     : bfeb6ddfa72d67fec3e9c9c741eb551486aa2b1e
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-11T14.26-bfeb6
dd\job.log
(2/2) .\test.py:ExampleTest.test_subtraction: STARTED
(1/2) .\test.py:ExampleTest.test_addition: STARTED
```

Modified test.py

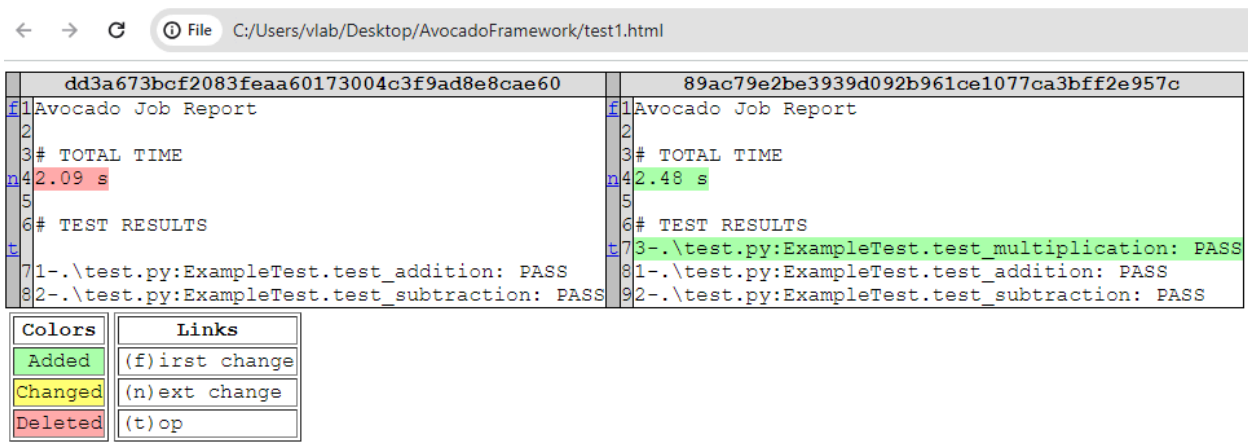
```
# test_example.py

from avocado import Test

class ExampleTest(Test):
  def test_addition(self):
     result = 2 + 2
     self.assertEqual(result, 4, "Addition test failed")

  def test_subtraction(self):
     result = 5 - 3
     self.assertEqual(result, 2, "Subtraction test failed")

  def test_multiplication(self):
     result = 5 * 3
```

```
self.assertEqual(result, 15, "Multiplication test failed")
```

OUTPUT :

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> avocado run .\test.py
JOB ID : 3035365d2799ddde3f448530d86add04397019e2
JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-06-11T14.30-30353
65\job.log
(3/3) .\test.py:ExampleTest.test_multiplication: STARTED
(1/3) .\test.py:ExampleTest.test_addition: STARTED
(2/3) .\test.py:ExampleTest.test_subtraction: STARTED

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado diff dd3a673 89ac79e --html test1.html**
test1.html



Additions: Often highlighted in green.
Deletions: Often highlighted in red.
Modifications: May be highlighted in yellow or another color.

- **Avocado can list tests discovered by each discovered plugin.**
  - Let's list only executable tests:

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado list .\test.py**

avocado-instrumented .\test.py:ExampleTest.test_addition
avocado-instrumented .\test.py:ExampleTest.test_subtraction
avocado-instrumented .\test.py:ExampleTest.test_multiplication

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado list .\test.ht**
ml
exec-test .\test.html

Exec-test → means those files are executables treated as simple tests.

- **By using - -verbose or -V flag to display files , but are not considered Avocado tests :**

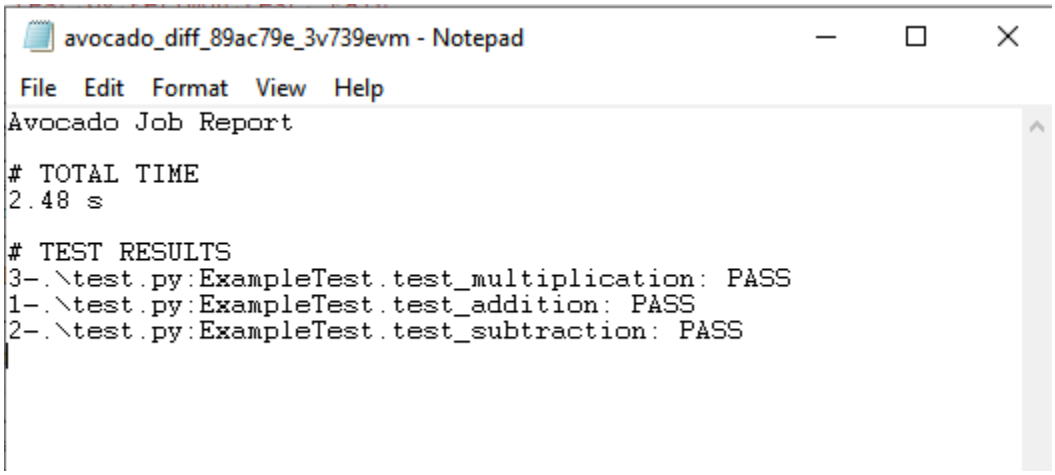
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado -V list .\test**
.py
Type Test Tag(s)
avocado-instrumented .\test.py:ExampleTest.test_addition
avocado-instrumented .\test.py:ExampleTest.test_subtraction
avocado-instrumented .\test.py:ExampleTest.test_multiplication

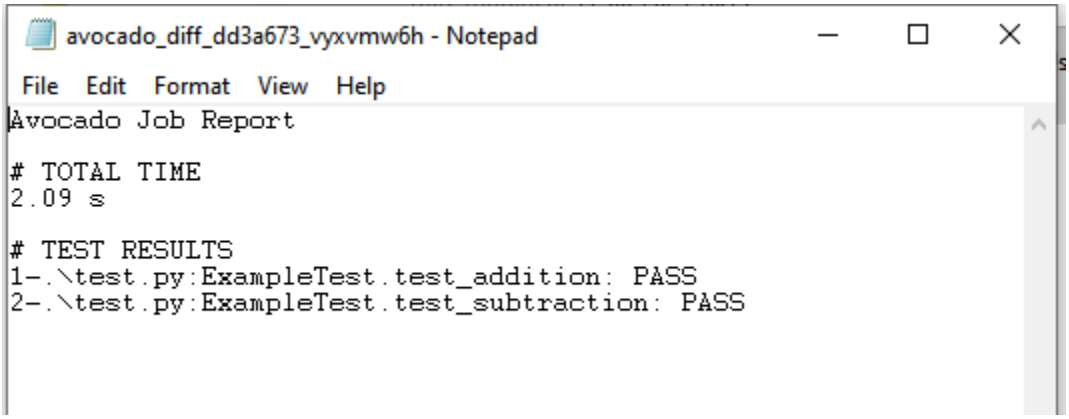
Resolver Reference Info

TEST TYPES SUMMARY
==========
avocado-instrumented: 3

4.1.3. avocado diff JOB JOB --create-reports

(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> avocado diff dd3a673 8
9ac79e --create-reports
C:\Users\vlab\AppData\Local\Temp\avocado_diff_dd3a673_vyxvmw6h.txt C:\Use
rs\vlab\AppData\Local\Temp\avocado_diff_89ac79e_3v739evm.txt





4.1.4. avocado diff JOB JOB --open-browser

Generate and open a HTML report in your preferred browser. If no --html file is provided, create a temporary file.

4.1.5. avocado diff --diff-filter DIFF_FILTER JOB JOB

Comma separated filter of diff sections :
(no)cmdline, (no)time, (no)variants, (no)results, (no)config, (no)sysinfo
(defaults to **all** enabled)

i) DIFF_FILTER(all)

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework>  avocado diff --diff-f
ilter all  dd3a673 89ac79e
--- dd3a673bcf2083feaa60173004c3f9ad8e8cae60
+++ 89ac79e2be3939d092b961ce1077ca3bff2e957c
@@ -1,8 +1,9 @@
Avocado Job Report

# TOTAL TIME
-2.09 s
+2.48 s

# TEST RESULTS
+3-.\test.py:ExampleTest.test_multiplication: PASS
1-.\test.py:ExampleTest.test_addition: PASS
2-.\test.py:ExampleTest.test_subtraction: PASS
```

ii) DIFF_FILTER(notime)

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework>  avocado diff --diff-f
ilter notime  dd3a673 89ac79e
--- dd3a673bcf2083feaa60173004c3f9ad8e8cae60
+++ 89ac79e2be3939d092b961ce1077ca3bff2e957c
@@ -1,5 +1,6 @@
Avocado Job Report

# TEST RESULTS
+3-.\test.py:ExampleTest.test_multiplication: PASS
1-.\test.py:ExampleTest.test_addition: PASS
2-.\test.py:ExampleTest.test_subtraction: PASS
```

iii)DIFF_FILTER(nocmdline) , DIFF_FILTER(novariants), DIFF_FILTER(noconfig) and DIFF_FILTER(nosysinfo)

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework>  avocado diff --diff-f
ilter noconfig,novariants,nocmdline,nosysinfo  dd3a673 89ac79e


--- dd3a673bcf2083feaa60173004c3f9ad8e8cae60
+++ 89ac79e2be3939d092b961ce1077ca3bff2e957c
@@ -1,8 +1,9 @@
Avocado Job Report

# TOTAL TIME
-2.09 s
+2.48 s

# TEST RESULTS
+3-.\test.py:ExampleTest.test_multiplication: PASS
1-.\test.py:ExampleTest.test_addition: PASS
2-.\test.py:ExampleTest.test_subtraction: PASS
```

Case 6 : avocado diff --diff-strip-id JOB JOB

Strip the "id" from "id-name;variant" when comparing test results.

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework>  avocado diff --diff-s
```

```
trip-id  dd3a673 808d08
--- dd3a673bcf2083feaa60173004c3f9ad8e8cae60
+++ 808d08f9cd5dedf171ba3a66429442c9a702577f
@@ -1,8 +1,9 @@
Avocado Job Report

# TOTAL TIME
-2.09 s
+2.97 s

# TEST RESULTS
+.\test.py:ExampleTest.test_multiplication: PASS
.\test.py:ExampleTest.test_addition: PASS
-.\test.py:ExampleTest.test_subtraction: PASS
+.\test.py:ExampleTest.test_subtraction: FAIL
```

4.2. assets

- Imagine you're writing a test that needs some extra files to run, like a specific document, a dataset, or even a program. These extra files are called "assets." Avocado, a testing framework, helps you manage these assets so you can easily use them in your tests without having to manually download or manage them every time.
- The **cache** refers to a local storage area where downloaded assets (files) are kept. This means that once an asset is downloaded, it is stored locally so that subsequent requests for the same asset do not require downloading it again. This can significantly speed up tests and reduce network usage, especially for large files or frequently used assets.

4.3. config

- Shows the data directories currently being used by avocado.

4.3.1. avocado config --datadir

```
(.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> avocado config --datad
ir
Config files read (in order, '*' means the file exists and had been read)
:
    C:\Users\vlab\Desktop\AvocadoFramework\.venv\etc\avocado\avocado.co
nf

Avocado replaces config dirs that can't be accessed
with sensible defaults. Please edit your local config
file to customize values
```

Avocado Data Directories:
base C:\Users\vlab/avocado
tests C:\Users\vlab\Desktop\AvocadoFramework\.venv\tests
data C:\Users\vlab/avocado/data
logs C:\Users\vlab/avocado/job-results
cache C:\Users\vlab/avocado/data/cache

4.3.2. avocado config --datadir reference

- Configuration reference with all registered options.

4.4. exec-path

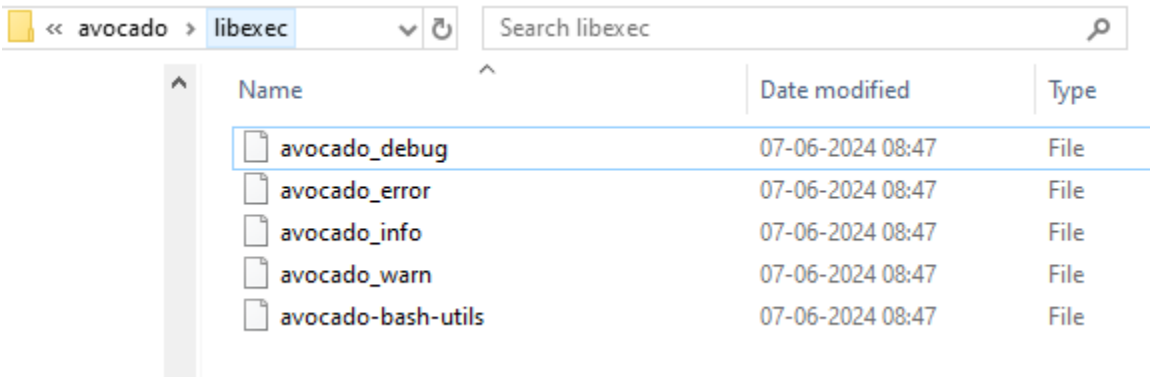
What is "exec-path"?

- In the Avocado framework (and in general computing), the "exec-path" refers to a list of locations or directories where your computer looks for programs (also called executables) when you run them.

  (.venv) PS C:\Users\vlab\Desktop\AvocadoFramework> **avocado exec-path**

  C:\Users\vlab\Desktop\AvocadoFramework\.venv\Lib\site-packages\avocado\li

  bexec



4.5. run

```
usage: avocado run [-h] [-p NAME_VALUE] [--suite-runner SUITE_RUNNER]
                   [-d] [--dry-run-no-cleanup]
                   [--force-job-id UNIQUE_JOB_ID]
                   [--job-results-dir DIRECTORY]
                   [--job-category CATEGORY] [--job-dependency FILE]
                   [--job-timeout SECONDS] [--failfast] [--keep-tmp]
                   [--ignore-missing-references] [--disable-sysinfo]
                   [--execution-order RUN.EXECUTION_ORDER]
                   [--store-logging-stream LOGGING_STREAM]
                   [--log-test-data-directories] [-t TAGS]
                   [--filter-by-tags-include-empty]
                   [--filter-by-tags-include-empty-key] [--journal]
                   [--json FILE] [--disable-json-job-result]
                   [--json-variants-load FILE] [--shuffle]
                   [--status-server-disable-auto]
                   [--status-server-listen HOST_PORT]
                   [--status-server-uri HOST_PORT]
                   [--max-parallel-tasks NUMBER_OF_TASKS]
                   [--spawner SPAWNER]
                   [--spawner-podman-bin PODMAN_BIN]
                   [--spawner-podman-image CONTAINER_IMAGE]
                   [--spawner-podman-avocado-egg AVOCADO_EGG]
                   [--tap FILE] [--disable-tap-job-result]
                   [--tap-include-logs] [--xunit FILE]
                   [--disable-xunit-job-result]
                   [--xunit-job-name XUNIT_JOB_NAME]
                   [--xunit-max-test-log-chars SIZE] [-z]
                   [TEST_REFERENCE ...]
```

4.5.1 : avocado run <test_file> --dry-run (list)

-d, --dry-run → Instead of running the test, only list them and log their params.

- Avocado can list your tests without running it.

- **What it does**: When you use the --dry-run option, Avocado will go through all the steps of preparing to run the tests (like setting up the environment, checking for test files, resolving test references, etc.) but will **not actually execute** any of the tests.
- **Why it's useful**: This is useful for verifying that your tests are set up correctly and that Avocado can find and prepare all the necessary components without actually running the tests. It helps in quickly checking for setup issues or errors in the test configuration.

```
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run .\tes
t_log.py --dry-run
JOB ID     : 0000000000000000000000000000000000000000
JOB LOG    : C:\Users\vlab\AppData\Local\Temp\avocado-dry-run-gmuu8si2\jo
b-2024-06-27T11.04-0000000\job.log

 (1/1) .\test_log.py:Plant.test_plant_organic:  CANCEL: Test cancelled due to --dry-run
(0.00 s)
 (1/1) .\test_log.py:Plant.test_plant_organic: STARTED
RESULTS    : PASS 0 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 1
JOB TIME   : 5.28 s
```

4.5.2 : avocado run <test_file> --dry-run-no-cleanup

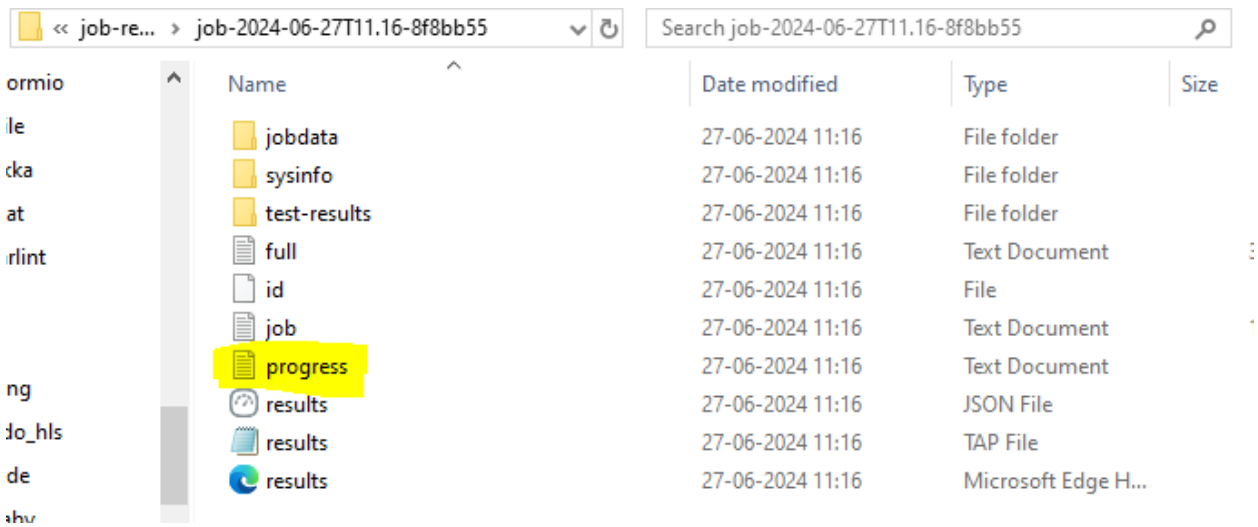
- **What it does**: This option does everything that --dry-run does, but with an additional feature: it **does not clean up** any temporary files or directories created during the dry run.
- **Why it's useful**: This is useful when you want to inspect the temporary files and directories that Avocado creates during the test setup phase. By not cleaning up these files, you can debug issues more effectively by examining what Avocado prepared for the actual test run.

```
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run .\tes
t_log.py --dry-run-no-cleanup
JOB ID     : c6407dca5d51ab62a9497c9918925f00a307b4e4
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-27T11.27-c6407
dc\job.log
 (1/1) .\test_log.py:Plant.test_plant_organic: STARTED

 (1/1) .\test_log.py:Plant.test_plant_organic:  PASS (7.73 s)
RESULTS    : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB TIME   : 11.50 s
```

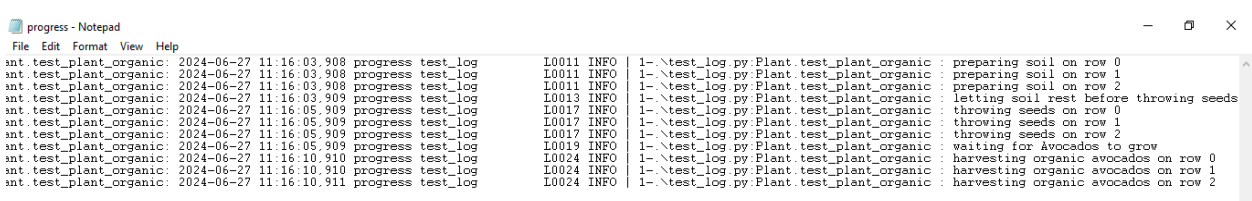
4.5.3 : avocado run <test_file> [--store-logging-stream LOGGING_STREAM]

- Store given logging STREAMs in "$JOB_RESULTS_DIR/$STREAM.$LEVEL".
- The result is that, besides all the other log files commonly generated, there will be another log file named **progress.INFO** at the job results dir.

```
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run .\tes
t_log.py --store-logging-stream progress
JOB ID     : 8f8bb5510cf5131e71d2762896d2fe54a8f2c4b1
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-06-27T11.16-8f8bb
55\job.log
 (1/1) .\test_log.py:Plant.test_plant_organic: STARTED

 (1/1) .\test_log.py:Plant.test_plant_organic:  PASS (7.69 s)
RESULTS    : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB TIME   : 11.47 s
```



progress :



4.5.4 : avocado run <test_file> --job-results-dir DIRECTORY

- Used to specify the directory where the results of a test job will be stored.

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run --job -results-dir=job1_results .\file1.py**
JOB ID : 645a4ed245c3ac62f121ab10d26a29b29eec6fbc
JOB LOG : C:\Users\vlab\Desktop\Vara\AvocadoFramework\job1_results\job -2024-07-09T11.20-645a4ed\job.log
(1/1) .\file1.py:ExampleTest.test_data_file: STARTED

(1/1) .\file1.py:ExampleTest.test_data_file: PASS (1.24 s)
RESULTS : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB TIME : 8.50 s

4.5.5 : avocado run <test_file> --job-category CATEGORY

- Avocado framework allows you to assign a category or classification to your test jobs when you run them. This categorization is used for organising and managing your test jobs based on their purpose or type.
- By using **--job-category** to categorise them accordingly:
  - **Unit Tests : --job-category unit**
  - **Integration Tests : --job-category integration**
  - **Performance Tests : --job-category performance**

performance_test.py

```
from avocado import Test

class ExampleRegressionTest(Test):
   def test(self):
       self.log.info("This is a regression test...")
       self.assertTrue(True,"Test passed")
```

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run .\performance_tests\test_performance.py --job-category performance**
Permission denied to link this job to category performance
JOB ID : 3b139266edeb3d5c2758ea2a8c52f64b0ca2aa63
JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-07-09T15.56-3b13926\job.log
(1/1) .\performance_tests\test_performance.py:ExamplePerformanceTest.test: STARTED

(1/1) .\performance_tests\test_performance.py:ExamplePerformanceTest.test: FAIL: 5 != 4 : Addition test failed (0.91 s)
RESULTS : PASS 0 | ERROR 0 | FAIL 1 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0

JOB TIME : 4.59 s

Test summary:
1-.\performance_tests\test_performance.py:ExamplePerformanceTest.test: FAIL

4.5.6. avocado run <test_file> --force-job-id=Unique Job ID needs to be a 40 digit hex number

- In Avocado, the job ID specified with **--force-job-id** must be a 40-digit hexadecimal number. This requirement is strict, meaning that only characters 0-9 and a-f are allowed.

Why Only Hexadecimal?

The restriction to a 40-digit hexadecimal number ensures that the job ID:

- Fits within a specific format that Avocado can consistently process.
- Is unique and long enough to avoid collisions.
- Conforms to the internal standards used by Avocado for identifying jobs.

Generating a valid job ID

The `uuid` module in Python is used to generate universally unique identifiers (UUIDs). These identifiers are often used for generating unique keys or IDs.

```
import uuid

job_id = uuid.uuid4().hex * 2  # uuid4 generates a 32-digit hex, we double it
to get 64 digits and then truncate to 40

job_id = job_id[:40]

print(job_id)
```

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **python .\jobId.py**

0cd9763bfb604c54962412b2884168be0cd9763b

4.5.7. avocado run <test_file> --force-job-id=Unique Job ID needs to be a 40 digit hex number

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run ./performance_tests/test_performance.py --job-results-dir=job1_results --force-job-id=4e8b1f4a7c8e4f6b9d2a7b8c9e6f8d1a7c3d4b6e**

```
JOB ID     : 4e8b1f4a7c8e4f6b9d2a7b8c9e6f8d1a7c3d4b6e
JOB LOG    :
C:\Users\vlab\Desktop\Vara\AvocadoFramework\job1_results\job-2024-07-09T16.33-4e8b1f4\jo
b.log
 (1/1) ./performance_tests/test_performance.py:ExamplePerformanceTest.test: STARTED

 (1/1) ./performance_tests/test_performance.py:ExamplePerform
anceTest.test:  FAIL: 5 != 4 : Addition test failed (0.83 s)
RESULTS    : PASS 0 | ERROR 0 | FAIL 1 | SKIP 0 | WARN 0 | IN
TERRUPT 0 | CANCEL 0
JOB TIME   : 4.53 s

Test summary:
1-./performance_tests/test_performance.py:ExamplePerformanceT
est.test: FAIL
```

4.5.8. avocado run [test_file1 test_file2 ….] --ignoring-missing-references

When you provide a list of test references (test files on the command line),. Avocado will try to resolve all of them through tests. If one or more test references cannot be resolved, the job will not be created. by using --ignore-missing-references to avoid that, and creating a job..

Cancelling.py

```
from avocado import Test
class CancelTest(Test):
  def test1(self):
     if 1 != 0 :
        self.cancel("statement is not true")
```

bystatus1.py

```
import avocado
from avocado import Test

class MyTestSuite(Test):
  def test_pass(self):
     self.assertEqual(1 + 1, 2)

  #def test_sub(self):
  #    self.assertEqual(1 - 1, 0)

  def test_mul(self):
     self.assertEqual(1 * 1, 1)
```

Output :

Case 1 : On the command line given by the Wrong test reference name, it doesn't create any jobs if one test reference name is Wrong without using --ignore-missing-references.

```
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run  .\bystatus1.py
cancellin.py
No tests found for given test references: cancellin.py
Try 'avocado -V list cancellin.py' for details
```

Case 2: When given the correct test reference, only creating a job

```
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run  .\bystatus1.py
cancelling.py
JOB ID     : 708cc809cbb833f241f917a72abcae849755353d
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-07-18T10.57-708cc
80\job.log
 (2/3) .\bystatus1.py:MyTestSuite.test_mul: STARTED
 (3/3) cancelling.py:CancelTest.test1: STARTED
 (1/3) .\bystatus1.py:MyTestSuite.test_pass: STARTED

 (1/3) .\bystatus1.py:MyTestSuite.test_pass:  PASS (1.06 s)
 (2/3) .\bystatus1.py:MyTestSuite.test_mul:  PASS (1.09 s)
 (3/3) cancelling.py:CancelTest.test1:  CANCEL: statement is not true (1.
11 s)

RESULTS    : PASS 2 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 1
JOB HTML   : C:\Users\vlab/avocado/job-results\job-2024-07-18T10.57-708cc
80\results.html
JOB TIME   : 5.75 s
```

Case 3 : On the command line given by the one wrong test reference name, it creates a job if one test reference name is wrong with the use of --ignore-missing-references.

```
(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> avocado run  .\bystatus1.py
canceg.py --ignore-missing-references
JOB ID     : f0bf445b98539eac0c5de8db3855fd2ce65c57fa
JOB LOG    : C:\Users\vlab/avocado/job-results\job-2024-07-18T11.04-f0bf4
45\job.log
 (1/2) .\bystatus1.py:MyTestSuite.test_pass: STARTED
 (2/2) .\bystatus1.py:MyTestSuite.test_mul: STARTED

 (1/2) .\bystatus1.py:MyTestSuite.test_pass:  PASS (0.83 s)
 (2/2) .\bystatus1.py:MyTestSuite.test_mul:  PASS (0.83 s)
RESULTS    : PASS 2 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
```

CANCEL 0
JOB HTML : C:\Users\vlab/avocado/job-results\job-2024-07-18T11.04-f0bf4
45\results.html
JOB TIME : 4.73 s

4.5.9. avocado run <test_file> --html [FILE]

(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run .\bystatus1.py --html results**
JOB ID : 4c6d616de6de5c471c3f534c57a2bd3795f3024f
JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-07-18T16.03-4c6d6
16\job.log
(1/2) .\bystatus1.py:MyTestSuite.test_pass: STARTED
(2/2) .\bystatus1.py:MyTestSuite.test_mul: STARTED

(1/2) .\bystatus1.py:MyTestSuite.test_pass: PASS (0.88 s)
(2/2) .\bystatus1.py:MyTestSuite.test_mul: PASS (1.23 s)
RESULTS : PASS 2 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 |
CANCEL 0
JOB HTML : C:\Users\vlab/avocado/job-results\job-2024-07-18T16.03-4c6d6
16\results.html
JOB TIME : 5.28 s

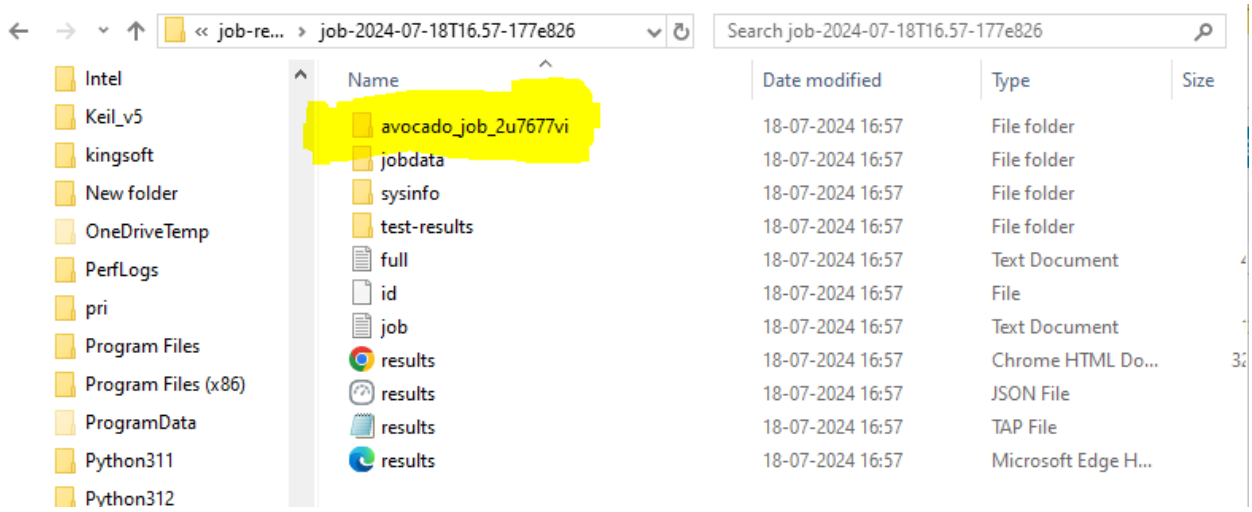
4.5.10. avocado run --keep-tmp <test_file>

- The --keep-tmp option in Avocado is used to retain temporary files generated during test execution.
- **Temporary Files**: Avocado generates various temporary files during test execution, including logs, intermediate data, and other diagnostic information.
- **Retention**: By default, Avocado deletes these temporary files after the job completes to keep the environment clean.

(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run --keep-tmp .\bystatus1.py**
JOB ID : 177e826c06d0b5253d8d492c6f766a2d761fec9e
JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-07-18T16.57-177e8
26\job.log
(1/3) .\bystatus1.py:MyTestSuite.test_pass: STARTED
(2/3) .\bystatus1.py:MyTestSuite.test_sub: STARTED
(3/3) .\bystatus1.py:MyTestSuite.test_mul: STARTED

(1/3) .\bystatus1.py:MyTestSuite.test_pass: PASS (0.81 s)
(2/3) .\bystatus1.py:MyTestSuite.test_sub: PASS (0.81 s)
(3/3) .\bystatus1.py:MyTestSuite.test_mul: PASS (0.84 s)

RESULTS : PASS 3 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB HTML : C:\Users\vlab/avocado/job-results\job-2024-07-18T16.57-177e826\results.html
JOB TIME : 4.41 s



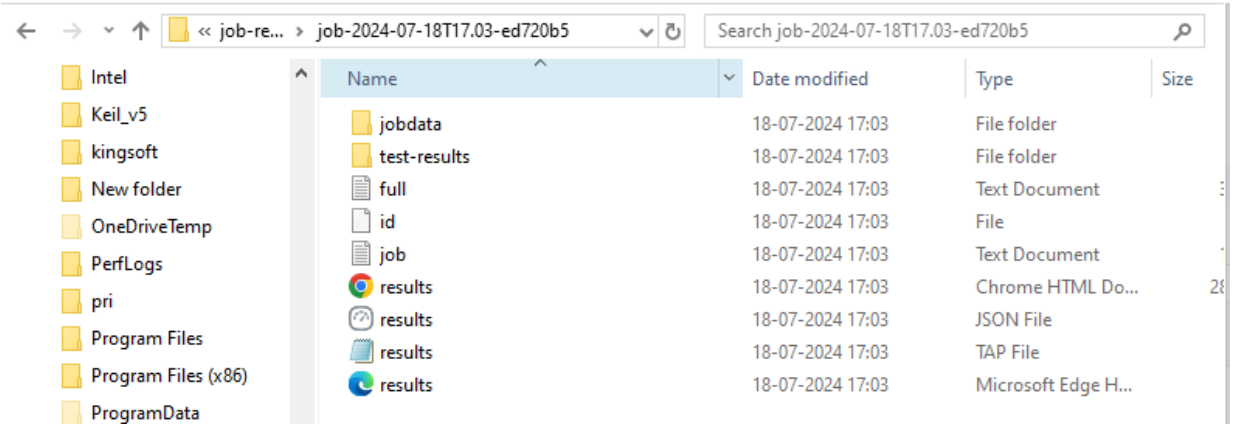
Here avocado_job_2u7677vi is the temporary folder

4.5.11. avocado run --keep-tmp <test_file>

(venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado run --disable-sysinfo .\bystatus1.py**
JOB ID : ed720b5107a948de4a01194c09b87def19ca0028
JOB LOG : C:\Users\vlab/avocado/job-results\job-2024-07-18T17.03-ed720b5\job.log
(1/3) .\bystatus1.py:MyTestSuite.test_pass: STARTED
(2/3) .\bystatus1.py:MyTestSuite.test_sub: STARTED
(3/3) .\bystatus1.py:MyTestSuite.test_mul: STARTED

(1/3) .\bystatus1.py:MyTestSuite.test_pass: PASS (0.80 s)
(2/3) .\bystatus1.py:MyTestSuite.test_sub: PASS (0.80 s)
(3/3) .\bystatus1.py:MyTestSuite.test_mul: PASS (0.81 s)
RESULTS : PASS 3 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB HTML : C:\Users\vlab/avocado/job-results\job-2024-07-18T17.03-ed720b5\results.html
JOB TIME : 3.39 s

The --disable-sysinfo option in Avocado is used to prevent the collection of system information during test execution



4.6. list

- Avocado allows tests to be given tags, which can be used to create test categories.With tags set, users can select a subset of the tests found by the test resolver.
- All docstring directives in Avocado require a strict format, that is, **:avocado:** followed by one or more spaces, and then followed by a single value with no **white spaces in between.**
- Test tags can be applied to test classes and to test methods. Tags are evaluated per method, meaning that the class tags will be inherited by all methods, being merged with method local tags.

4.6.1. TAGS

categorizing.py

```
from avocado import Test

class MyClass1(Test):
  """
  :avocado: tags=furious
  """

  def test1(self):
    """
```

```
        :avocado: tags=fast
        """
        self.whiteboard = "test1 executed"
        pass

    def test2(self):
        """
        :avocado: tags=slow
        """
        self.whiteboard = "test2 executed"
        pass

class MyClass2(Test):
    """
    :avocado: tags=furious,fast,slow
    """

    def test3(self):
        self.whiteboard = "test3 executed"
        pass

    def test4(self):
        self.whiteboard = "test4 executed"
        pass

class Idle(Test):

    """
    Idle tests
    """
    def test_idle(self):
        self.whiteboard = "test achieved nothing"
```

4.6.2. avocado list <test_filename.py>

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\categorizing.py**

```
avocado-instrumented .\categorizing.py:MyClass1.test1
avocado-instrumented .\categorizing.py:MyClass1.test2
avocado-instrumented .\categorizing.py:MyClass2.test3
avocado-instrumented .\categorizing.py:MyClass2.test4
avocado-instrumented .\categorizing.py:Idle.test_idle
```

4.6.3. avocado list <test_filename> --filter-by-tags=TAGS

- Filter tests based on tags

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\ca tegorizing.py --filter-by-tags=slow**
avocado-instrumented .\categorizing.py:MyClass1.test2
avocado-instrumented .\categorizing.py:MyClass2.test3
avocado-instrumented .\categorizing.py:MyClass2.test4

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\ca tegorizing.py --filter-by-tags=furious**
avocado-instrumented .\categorizing.py:MyClass1.test1
avocado-instrumented .\categorizing.py:MyClass1.test2
avocado-instrumented .\categorizing.py:MyClass2.test3
avocado-instrumented .\categorizing.py:MyClass2.test4

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\ca tegorizing.py --filter-by-tags=fast**
avocado-instrumented .\categorizing.py:MyClass1.test1
avocado-instrumented .\categorizing.py:MyClass2.test3
avocado-instrumented .\categorizing.py:MyClass2.test4

4.6.4. avocado list <test_filename> --filter-by-tags=-TAGS

If you do not want the “fast” tests (note the minus sign before the tag)
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\categorizing.py --filter-by-tags=-fast**
avocado-instrumented .\categorizing.py:MyClass1.test2

4.6.5. avocado list <test_filename> --filter-by-tags=TAGS,TAGS

If you require tests to be tagged with multiple tags, just add them separate by commas
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\ca tegorizing.py --filter-by-tags=fast,slow**
avocado-instrumented .\categorizing.py:MyClass2.test3
avocado-instrumented .\categorizing.py:MyClass2.test4

4.6.6. avocado list <test_filename> --filter-by-tags=TAGS --filter-by-tags=TAGS

(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\ca tegorizing.py --filter-by-tags=fast --filter-by-tags=slow**
avocado-instrumented .\categorizing.py:MyClass1.test1
avocado-instrumented .\categorizing.py:MyClass1.test2
avocado-instrumented .\categorizing.py:MyClass2.test3

avocado-instrumented .\categorizing.py:MyClass2.test4

4.6.7. avocado list <test_filename> --filter-by-tags-include-empty

- Include all tests without tags kept in the test suite found previously to filtering.

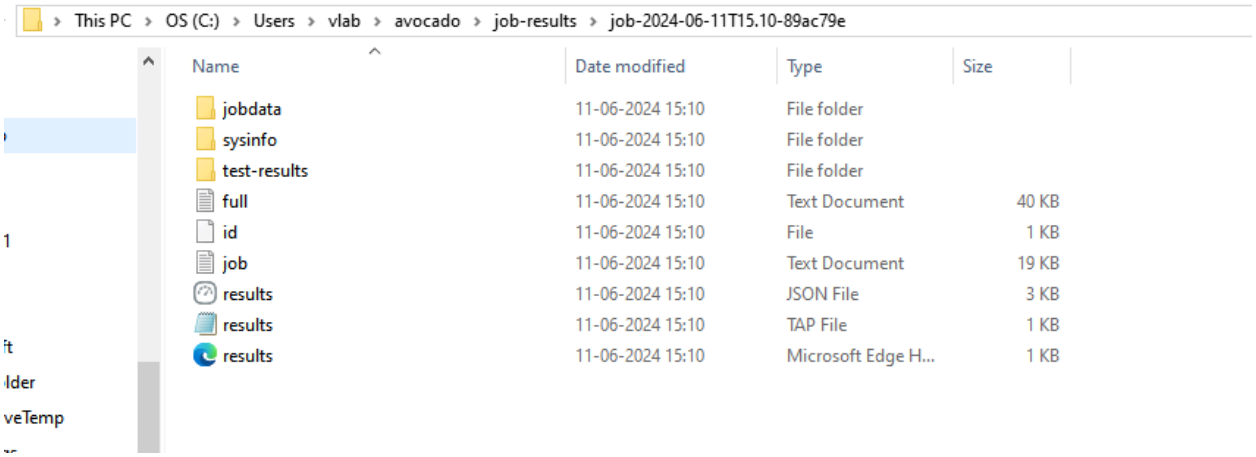
(.venv) PS C:\Users\vlab\Desktop\Vara\AvocadoFramework> **avocado list .\ca tegorizing.py --filter-by-tags-include-empty**
avocado-instrumented .\categorizing.py:MyClass1.test1
avocado-instrumented .\categorizing.py:MyClass1.test2
avocado-instrumented .\categorizing.py:MyClass2.test3
avocado-instrumented .\categorizing.py:MyClass2.test4
avocado-instrumented .\categorizing.py:Idle.test_idle

4.6.8. avocado list <test_filename> --filter-by-tags-include-empty-key

- Include all tests that do not have a matching key in its key:val tags. This effectively means those tests will be kept in the test suite found previously to filtering.

2.4.3. Results Specification

- Job results are available under [job-results] / job-[timestamp]-[short job ID]
- Directory name includes a timestamp.



- A human readable **id** in the top level, with the job SHA1.
- A human readable **job.log** in the top level, with human readable logs of the task

- Subdirectory **jobdata**, that contains machine readable data about the job.
- A machine readable **results.xml** and **results.json** in the top level, with a summary of the job information in xUnit/json format.
- A top level **sysinfo** dir, with sub directories **pre, post and profile**, that store sysinfo files pre/post/during job, respectively.
- Subdirectory **test-results,** that contains a number of subdirectories (filesystem-friendly test ids). Those test ids represent instances of test execution results.

2.4.3.1. Test execution instances specification

- job.log
- sysinfo
- data

4.6. distro

```
usage: avocado distro [-h] [--distro-def-create]
                      [--distro-def-name DISTRO_DEF_NAME]
                      [--distro-def-version DISTRO_DEF_VERSION]
                      [--distro-def-release DISTRO_DEF_RELEASE]
                      [--distro-def-arch DISTRO_DEF_ARCH]
                      [--distro-def-path DISTRO.DISTRO_DEF_PATH]
                      [--distro-def-type {rpm,deb}]
```

- The distro feature in Avocado tells you which version of Linux your computer is running.
- **Example : Software Development Team**
  - Imagine you're part of a software development team. Your team members use different operating systems to develop and test software. Some use Windows, some use macOS, and others use various versions of Linux. Each operating system has different tools and requirements for building and testing software.
  - **Example Scenario**
  - Your team needs to run tests on the software to make sure it works correctly on all these different systems. However, the way you install software, set up environments, and run tests can vary depending on the operating system.
  - **Without Knowing the Operating System**
  - If you don't know which operating system you're dealing with, you might try to run the same setup commands everywhere. This can lead to errors because:
    - The command to install software might be different.
    - The paths to certain tools or libraries might be different.
    - Some features might not be available on all systems.
  - **With distro Feature in Avocado**

- By using the distro feature in Avocado, you can detect the operating system and adjust your actions accordingly. This ensures that your tests run smoothly on all systems.

4.7. result_events

4.7.1. bystatus

- In the Avocado framework the bystatus option is used to filter test results based on their execution status.
- This can be particularly useful when you want to quickly focus on tests that passed, failed, or had some other specific outcome.

4.7.2.

5. Examples

5.1. Blinking Led

```
from avocado import Test

import subprocess

import time

class MicroPythonTest(Test):

  def setUp(self):

    self.port = 'COM5'

    self.led_txt = '/Users/vlab/Desktop/Vara/MicroPython/results/led_status.txt'

    self.expected_on_off_count = 4

    time.sleep(2)


  def test_blink_led(self):

    script = """

import machine
```

```
import time

led = machine.Pin('PA5', machine.Pin.OUT)  # LED pin
status_pin = machine.Pin('PA6', machine.Pin.IN)  # Pin to read status (if available)

# Log the LED state to a file
with open('led_status.txt', 'w') as f:
  for i in range(4):
      led.value(1)  # Turn the LED on
      time.sleep(1)  # Wait for 1 second
      f.write("LED ON\\n")

      led.value(0)  # Turn the LED off
      time.sleep(1)  # Wait for 1 second
      f.write("LED OFF\\n")
"""
    # Write the script to a local file
    with open("blink_led.py", "w") as f:
        f.write(script)

    # Use ampy to upload the script to the MicroPython device
    subprocess.run(['ampy', '--port', self.port, 'put', 'blink_led.py'])

    # Run the script on the MicroPython board
    subprocess.run(['ampy', '--port', self.port, 'run', 'blink_led.py'])
```

```
        # Retrieve the LED status log from the MicroPython device
        subprocess.run(['ampy', '--port', self.port, 'get', '/flash/led_status.txt',self.led_txt])


    def test_led_on(self):
        on = open(self.led_txt,'r')
        content = on.readlines()
        on.close()
        led_on_count = content.count('LED ON\n')
        self.assertEqual(led_on_count,self.expected_on_off_count,"Led on count not
matched")


    def test_led_off(self):
        off = open(self.led_txt,'r')
        content = off.readlines()
        off.close()
        led_off_count = content.count('LED OFF\n')
        self.assertEqual(led_off_count,self.expected_on_off_count,"Led off count not
matched")


    def tearDown(self):
        pass
```