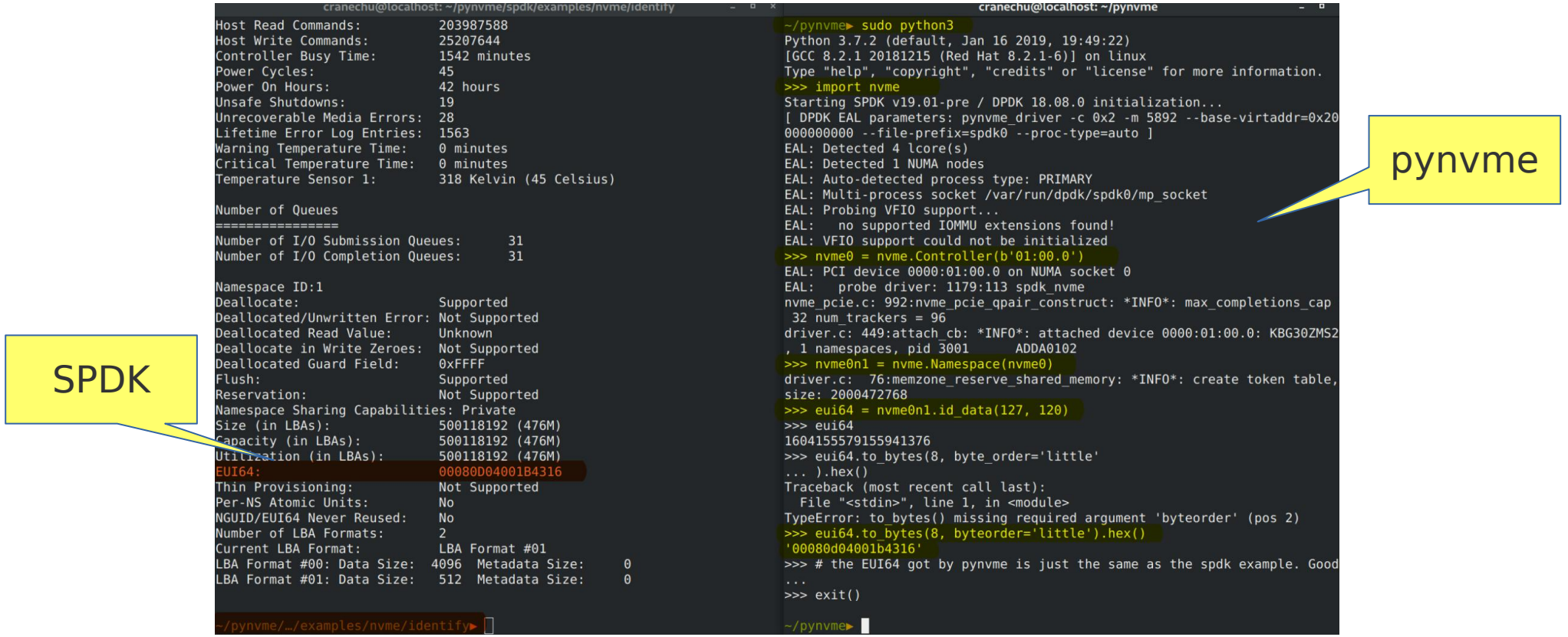
Pynvme:
test NVMe devices in Python

https://github.com/cranechu/pynvme

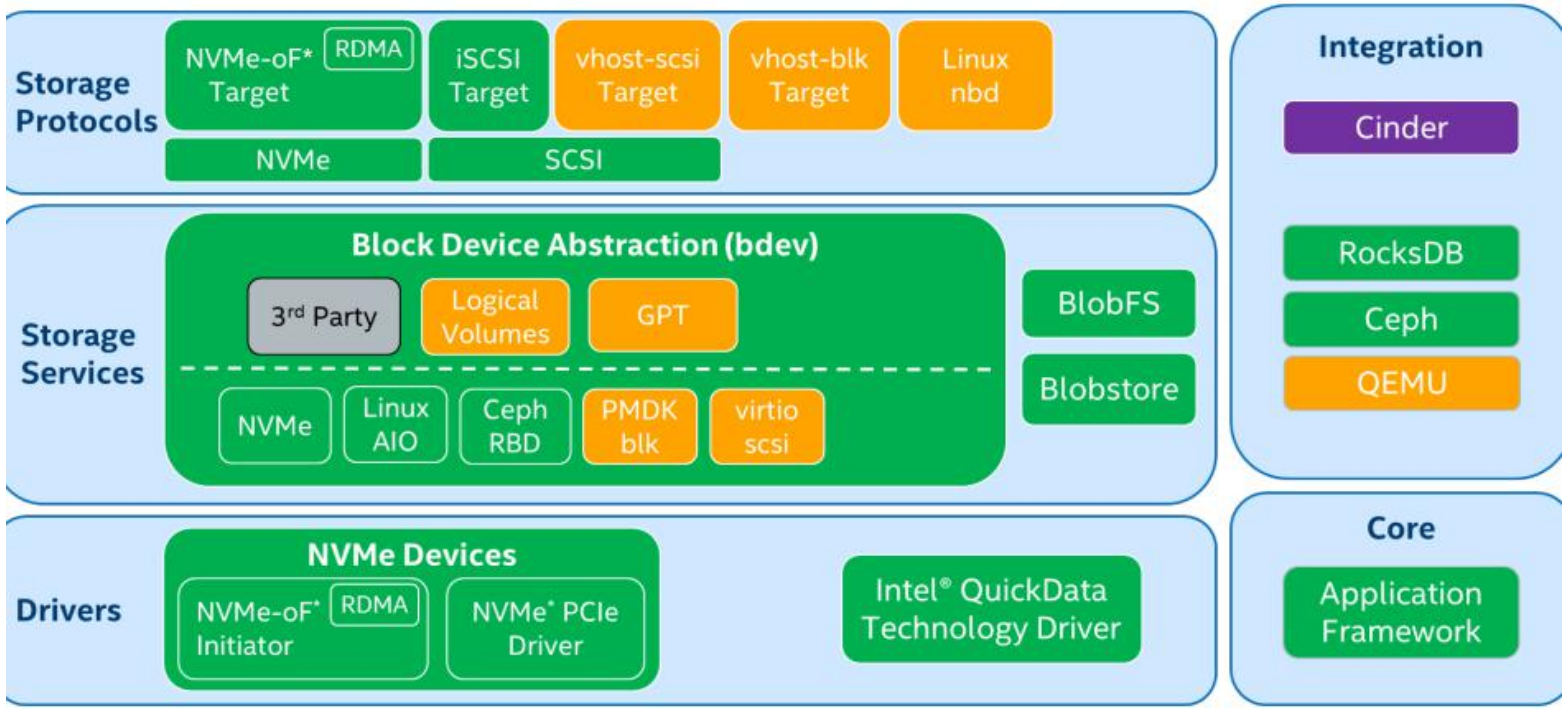
Pynvme

Pynvme is a python extension module. Users can operate NVMe SSD intuitively by Python scripts. It is designed for NVMe SSD testing with performance considered. With third-party tools, e.g. emacs, vscode and pytest, pynvme is a convenient and professional NVMe device test solution.

Pynvme in Python IDLE



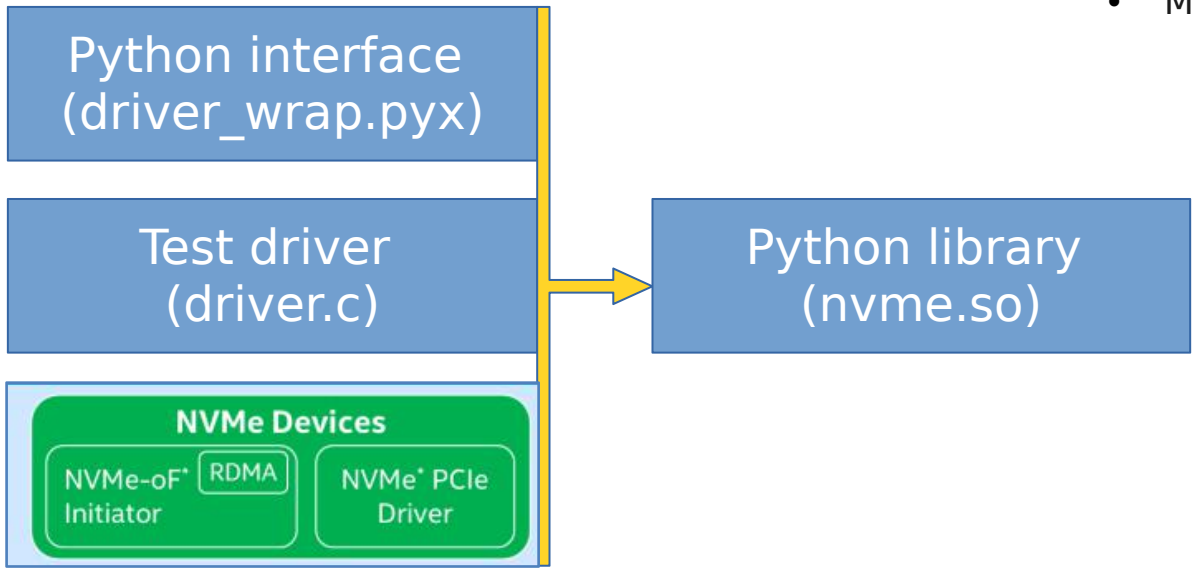
Pynvme is based on SPDK



Pynvme Architecture

Build python library with Cython:

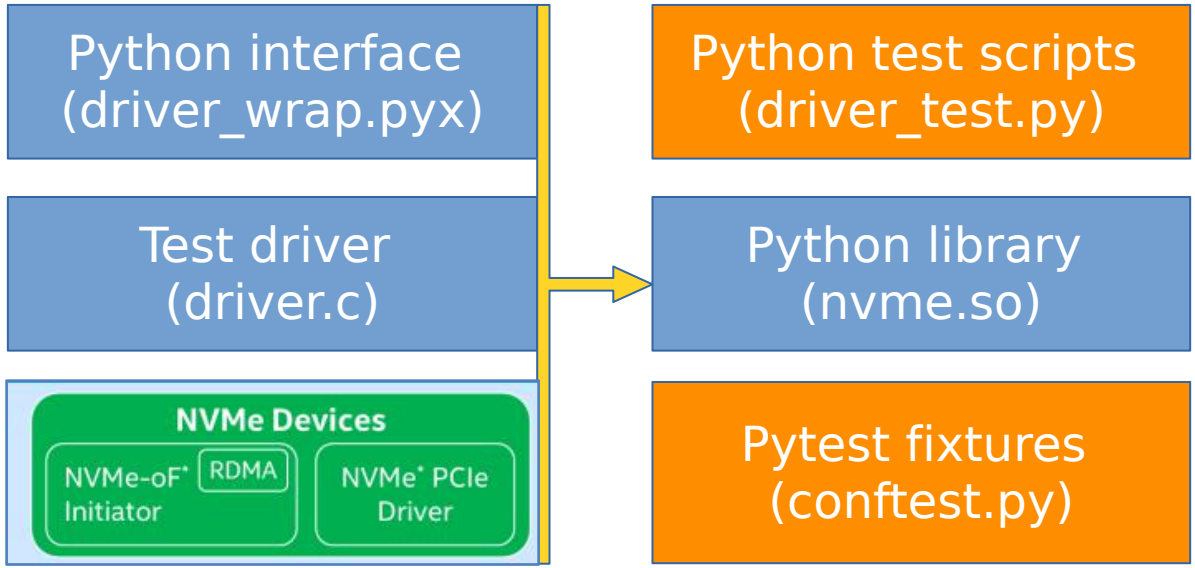
- setup.py
- driver.c
- driver.h
- cdriver.pxd
- driver_wrap.pyx
- Makefile



Pynvme Architecture

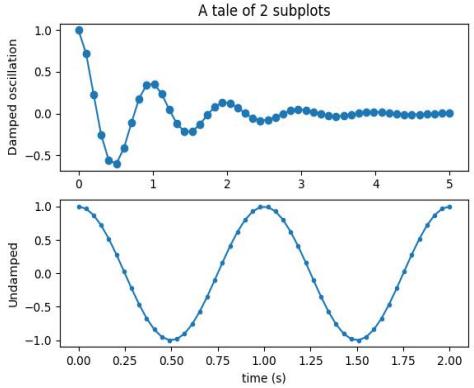
Organize test cases in pytest:

- mvme.so
- pytest.ini
- conftest.py
- driver_test.py



Why Python?

- √ Many beautiful mature libraries
  - pytest
  - logging
  - multiprocessing
  - numpy
  - matplotlib
  - pydoc
  - os, io, time, pytemperature, statistics, yaml, json, struct, …

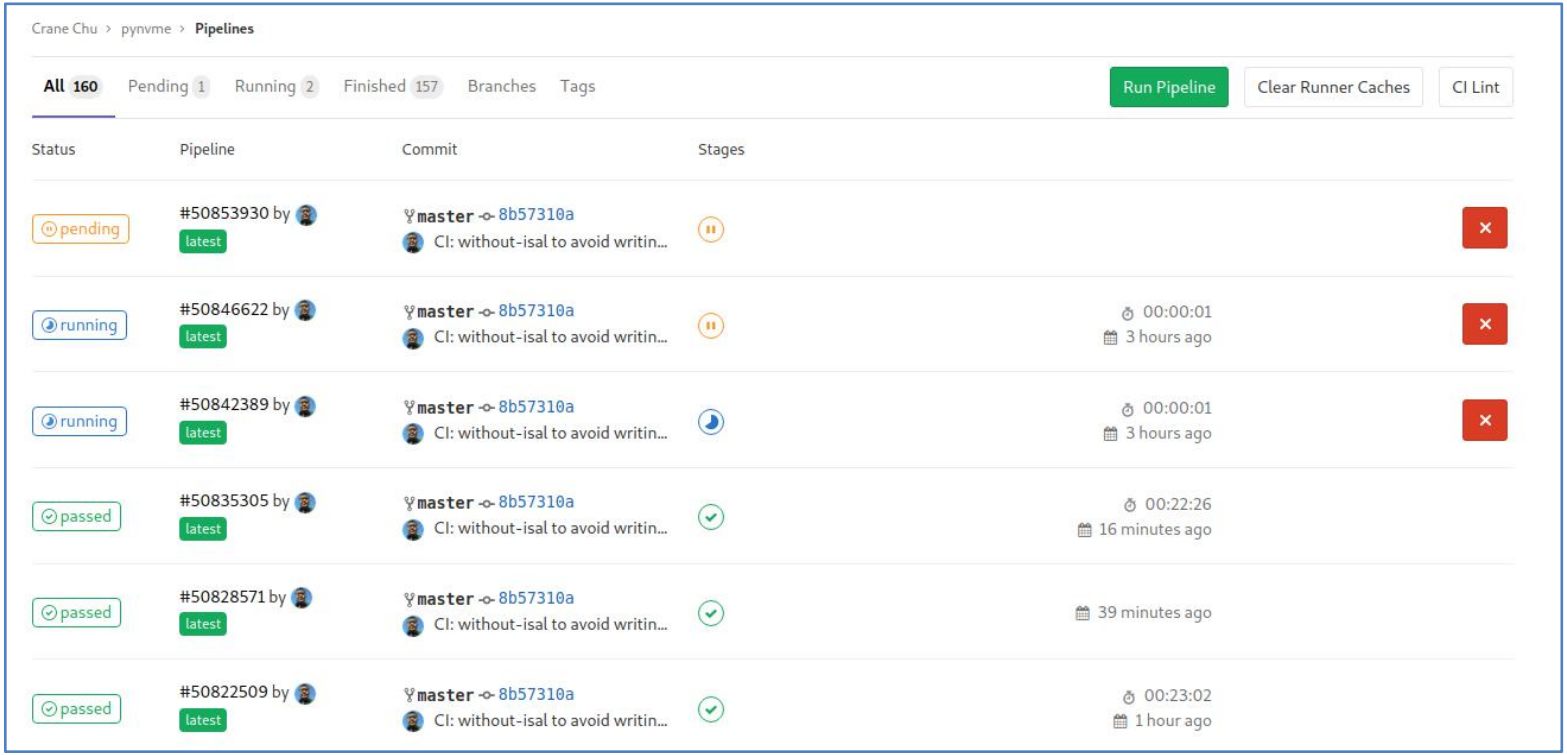


- √ Friendly IDEs for developing, debugging, and testing
  - Pycharm, emacs, vscode

pytest

- √ CI: develop firmware softly 

Pynvme CI Status



Config and Compile

- git clone https://github.com/cranechu/pynvme; cd pynvme
- git submodule update --init –recursive
- sudo ./spdk/scripts/pkgdep.sh
- sudo dnf install python3-pip -y
- sudo python3 -m pip install -r requirements.txt
- cd spdk; ./configure --without-isal; cd ..
- make spdk; make

- Fedora29 is recommended.
- find script examples in driver_test.py.
- **find more =>** https://github.com/cranechu/pynvme

Test Scripts

pytest fixtures

pytest cases are started with test_

fill buffer with identify data

create qpair and buffer for write commands

callback functions are called when cmds are completed

write data, and then read in the callback

the status of the callback function also includes the **Phase Tag** bit

emacs

```
def test_write_identify_and_verify_with_callback(nvme0, nvme0n1):
    id_buf = d.Buffer(4096)
    nvme0.identify(id_buf).waitdone()

    q = d.Qpair(nvme0, 20)
    n = nvme0n1
    read_buf = d.Buffer(4096, "read buffer")

    def read_cb(cdw0, status):
        assert id_buf[:40] == read_buf[:40]

    def write_cb(cdw0, status):
        n.read(q, read_buf, 5, 8, cb=read_cb)

    n.write(q, id_buf, 5, 8, cb=write_cb).waitdone(2)

    id_buf[0] += 1
    n.write(q, id_buf, 5, 8, cb=write_cb).waitdone(2)
    id_buf[9] = (id_buf[9] >> 1)
    n.write(q, id_buf, 5, 8, cb=write_cb).waitdone(2)

55k 249: 0 UU- -~/pynvme/driver_test.py        15% -master Python [test
In: test_write_identify_and_verify_with_callback()
```

IOWorker

emacs

```
1066 @pytest.mark.parametrize("qcount", [1, 2, 4, 8, 15])
1067 def test_ioworker_iops_multiple_queue(nvme0n1, qcount):
1068     l = []
1069     io_total = 0
1070     for i in range(qcount):
1071         a = nvme0n1.ioworker(io_size=8, lba_align=8,
1072                              region_start=0, region_end=256*1024*8, # 1GB space
1073                              lba_random=False, qdepth=16,
1074                              read_percentage=100, time=10).start()
1075         l.append(a)
1076
1077     for a in l:
1078         r = a.close()
1079         io_total += (r.io_count_read+r.io_count_write)
1080
1081     logging.info("Q %d IOPS: %dK" % (qcount, io_total/10000))
1082
55k 1068: 0 UU- -~/pynvme/driver_test.py          60% -master Python [test_ioworker_iop
In: test_ioworker_iops_multiple_queue()
```

define IO patterns in ioworker's parameter list

send IO in a separated process

wait ioworkers till finish, and collect result data

Fixtures of Pynvme

- create/delete test objects. in conftest.py:
  - nvme0
  - nvme0n1
  - pcie
  - ...
- parametrize of tests
  - @pytest.mark.parametrize("qcount", [1, 2, 4, 8, 15])
  - @pytest.mark.parametrize("repeat", range(10))
- test control
  - @pytest.mark.skip("nvme over tcp")

- doc: https://docs.pytest.org/en/latest/fixture.html

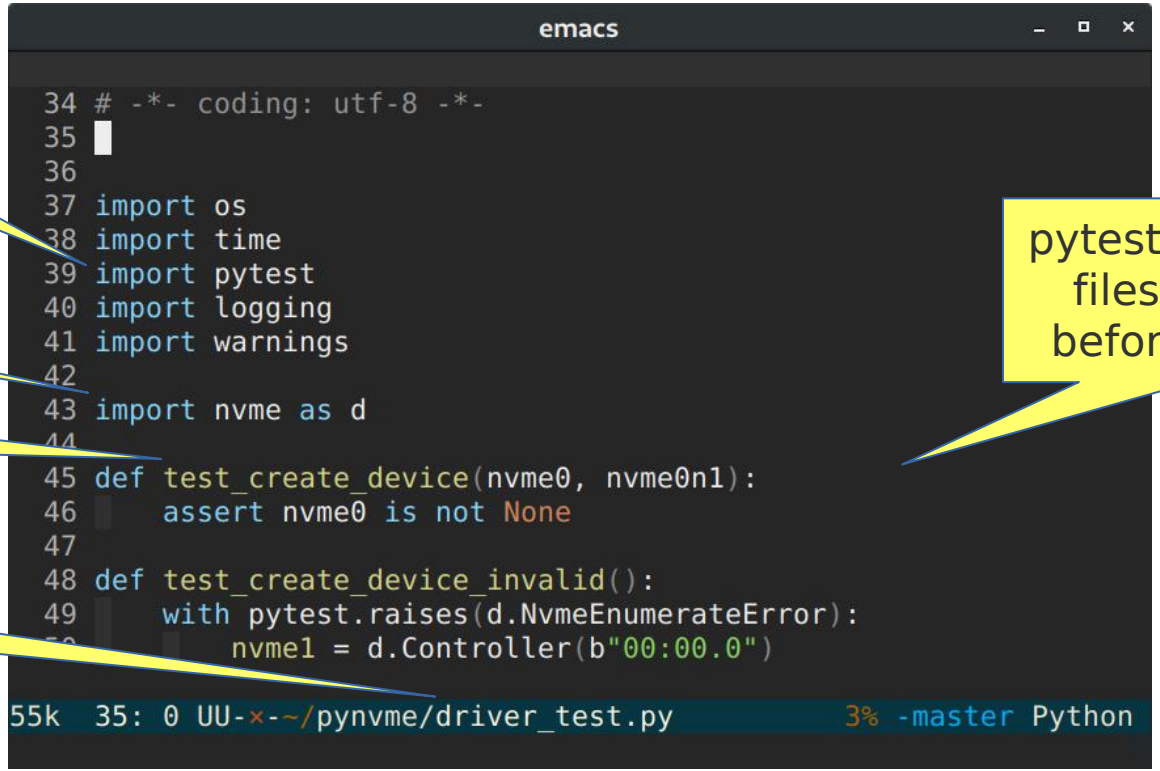
Test Files

import pytest

import nvme

test functions

test file name



pytest collects test files and cases before execution

Pytest Execution

- '''The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.'''

- '''pytest fixtures offer dramatic improvements over the classic xUnit style of setup/teardown functions'''

- pytest test_mod.py # run tests in a module
- pytest testing/ # run tests in a directory
- pytest test_mod.py::test_func # run a specific test case
- pytest test_mod.py -s # run tests without log capturing
- https://media.readthedocs.org/pdf/pytest/latest/pytest.pdf

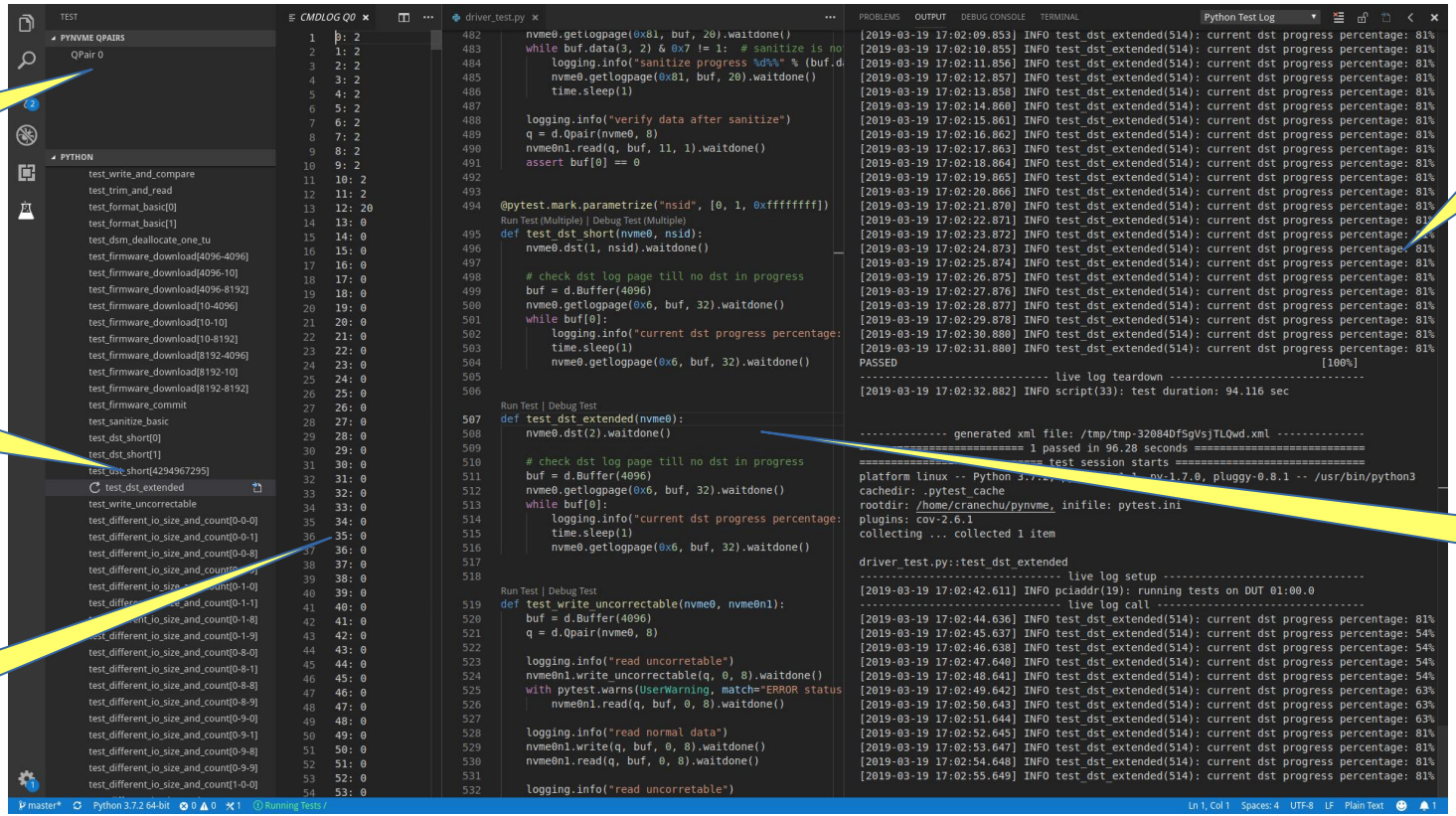
Visual Studio Code

- VSCode is a popular IDE. It has a mature python extension, which supports pytest.
  - root user is not recommended by vscode
  - it is required for user to run sudo without a password.
    - *sudo visudo*

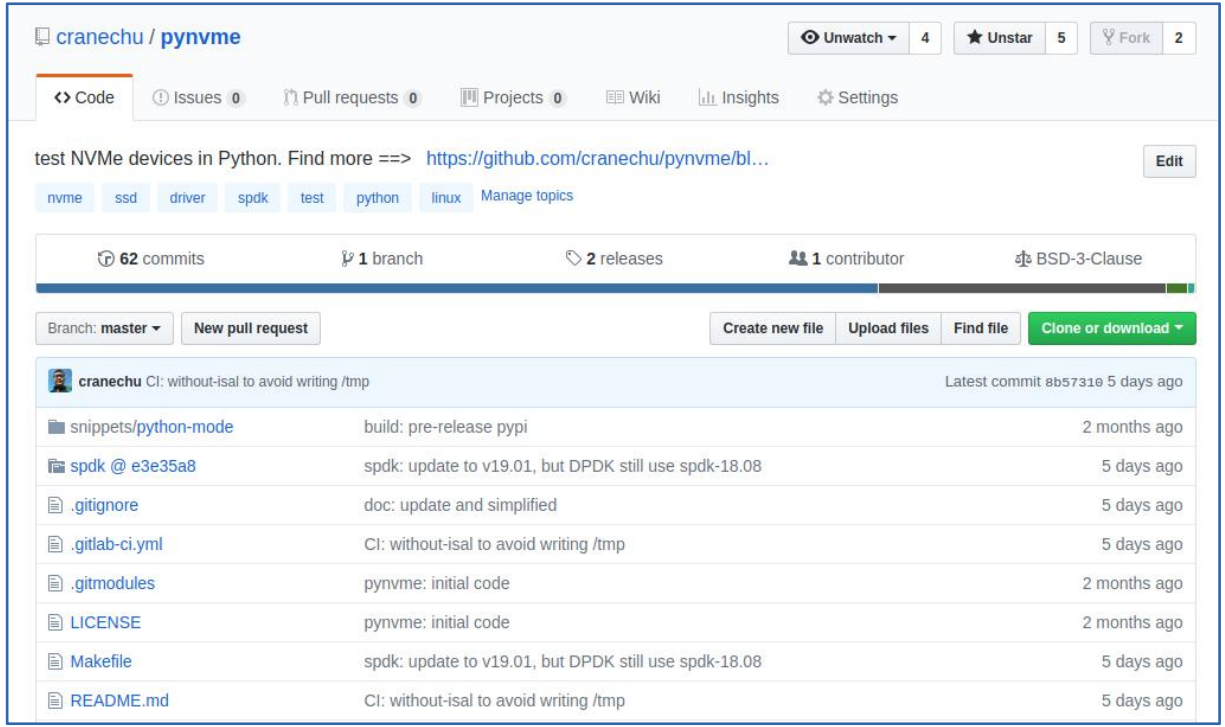
- Pynvme also providers an extension to monitor device status and cmdlog in every qpair.
  - *code --install-extension pynvme-console-0.0.1.vsix*

- *make setup; code .*

Pynvme in VSCode



Welcome to use and contribute



https://github.com/cranechu/pynvme