Report: Adopt pytest month

Brianna Laugher, June 2015

Table of Contents

The idea

FOSDEM pytest meetup, [Picture by Holger Krekel]()

I organised “Adopt pytest month” (APM) during April 2015. The idea was to pair experienced pytest users with open source projects for a month, and see how much progress they could make in getting started with pytest. Pytest is in fact very simple to start with, but as its style is quite different to the xUnit/unittest style, it can perhaps be daunting to dive into deeply. And let's face it, rewriting your test suite is not most people's idea of a good time. But luckily pytest aficionados are not most people!

Pytest is a test runner and utility, mature and awash with plugins. The project was started by Holger Krekel who has founded a number of useful Python projects including tox, a library for running tests over multiple environments (for example, python 2.7 and python 3.4, or Windows and Linux), that naturally is like a pea in a pod next to pytest.

The idea for APM occurred to me at FOSDEM 2015, during an informal pytest meetup. I *love* pytest. I find it an absolute joy to use. So it surprises me that it isn't used more widely. People still use nose or even unittest. I wanted to do something to actively encourage more people to starting using it. At the same time, I had been toying with the idea of picking up a random open source project and converting its test suite to pytest, to form the source of a “proof by example” blog post or conference talk about pytest. At FOSDEM it occurred to me that a bunch of people doing this at once could be a fun experiment. Holger and the other pytesters there thought it sounded interesting. And so APM was born.

It is rare, in my observation, for open source projects founded on community backing (rather than corporate) to “evangelise” or try to recruit new users, or even just actively promote themselves. Occasionally the wider community galvanises behind preferred solutions (e.g. requests, pip), but most open source projects seem to rely on solid documentation and word of mouth to gain new users. So throughout this project I have been curious about how this effort would be perceived, perhaps as overly pushy.

The sign-up process

I settled on April as it would give me two months to find pytest volunteers (hereafter referred to as helpers) and put the word out to encourage open source projects to sign up. Although I'm sure there is never an ideal month, scheduling APM at the same time as both PyCon US and Outreachy (formerly

Outreach Program for Women) was probably not the brightest.

I thought it might be difficult to find enough helpers to match with a torrent of projects. Naturally I had it all backwards. 26 people signed up as helpers, which blew me away. Only nine of them were existing contributors to the pytest codebase. While the pytest mailing list is not empty, it's not exactly full of chatter, either. I felt like I had uncovered some latent community – people who loved pytest and wanted to contribute, but weren't quite sure how.

As for finding interested projects, this turned out to be more of a challenge. I wrote a page for the pytest website. I wrote to pytest-dev and TestingInPython mailing lists. I started a pytestdotorg twitter account to help advertise the project and made some cute graphics to go with it.

I specified that the ideal partner project would:

- be open source, and predominantly written in Python
- have an automated/documented install process for developers
- have more than one core developer
- have at least one official release (e.g. is available on pypi)
- have the support of the core development team, in trying out pytest adoption

After two weeks only three projects had signed up. I began to worry that I had two dozen eager volunteers and no work to give them.

Towards the end of March I was convinced to relax the requirement for having more than one core developer, which turned out to be a good idea. Two projects specifically open-sourced their code in order to take part in APM, which impressed me!

I also asked pytesters to brainstorm any projects they knew of that might be good candidates, and approach them. This was semi-successful, but is probably the way to go for any future iterations – with strong guidance about what types of projects to approach, and how to do so.

The projects

The first sign-up was from a project that seemed to have no source code to speak of, so I declined it.

Bidict is a library which provides a "bidirectional dict with convenient slice syntax: `d[65] = 'A'` ⇔ `d[:'A'] = 65`".

Helper: Tom Viner

Guessit is a command-line utility/library “that tries to extract as much information as possible from a filename” (typically a media file name – such as song title, artist, album).

Helper: Sravan Bhamidipati

Coursera-dl is a command-line utility for “downloading Coursera.org videos and naming them”, along the lines of youtube-dl. The maintainer told me, “Despite never having had a formal release, the project has been starred 2100+ times on github and it has almost 800 forks!”

Helper: Meejah

Qutebrowser is a “keyboard-driven, vim-like browser based on PyQt5 and QtWebKit.” Qutebrowser has had many contributors over the years but at the moment is basically a one-person effort. The developer was encouraged by a pytester in the pylib IRC room to sign up anyway.

Helpers: Bruno Oliveira, Raphael Pierzina

Arkestra is an extension to Django “to provide an intelligent semantic web publishing system for organisations and institutions”. The maintainer commented that Arkestra is actively used for several conferences and professional organisations, but lack of adequate testing remained a factor that discouraged other developers from joining the project.

Helpers: Christian Long, Anatoly Bubenkov, 1 other

Kallithea is a “source code management system” similar to Gitlab which supports both Git and Mercurial. Kallithea is a fork developed from the ambiguously-licensed RhodeCode software, and has a number of active developers.

Helpers: Marc Abramowitz, 2 others

Nefertari is a “REST API framework sitting on top of Pyramid and ElasticSearch” by Montreal company Brandicted, who open-sourced the code in March.

Helper: Arye, 1 other

Trump is “a framework for objectifying data, with the goal of centralizing the management of data feeds to enable quicker deployment of analytics, applications, and reporting”. It uses pandas. It was developed at Equitable Life of Canada and was also open-sourced in March.

Helpers: Andreas, 2 others

Of projects approached by helpers to more actively suggest their participation, we had the following responses: 3 projects gave no response, 3 declined, and 6 were enthusiastic or at least curious. Kallithea and guessit have been included above. The others that responded positively were:

Jsonpickle, a library “for serializing any arbitrary object graph into JSON.”

Cookiecutter-django, a “template for creating Django projects quickly.”

Jinja2, a templating engine, commonly used in Django and other web frameworks.

Pelican, a static site generator.

In the end, the latter three had pull requests accepted before April even began, so while APM seemed a good impetus for them I didn't count them as taking part in APM. No pytest work was applied to jsonpickle so I also haven't counted it.

While having helpers actively approach projects was a good idea, it was a little difficult for me as an organiser to feel confident that these projects had the same idea about what was happening that I had. For projects that signed up via the pytest website, there was a lot of contact between them and I which gave me the chance to discuss their expectations, emphasise realistic planning, and follow-up part way through the month.

After collecting these projects I then had to match the helpers to the projects. For a couple of projects there were helpers with obvious strengths, but for most it was almost chance. The helpers had a range of (self-assessed) pytest skills, so for some of the larger projects I thought it would make sense to put 2 or 3 helpers together. For these projects I tried to assign at least one “known” experienced developer (ie someone with prior pytest development).

Of the 26 helpers, 5 “de-volunteered” at my request (when I realised there were not enough projects for all the helpers), 18 I assigned among 11 projects, and 3 I had to send an email saying there were not enough projects for them to be assigned to. (The 11 projects were the 8 above and jsonpickle, cookiecutter-django and jinja2 – as it seemed to me that there was potentially more work to be done with these projects.)

Results

A partial timeline:

- 31 Jan: FOSDEM discussion
- 5 Feb: Email to pytest-dev suggesting APM
- 27 Feb: page published on pytest.org with helper and project sign-up surveys
- 27 Mar: email to helpers with information on their assigned projects
- 31 Mar: email to projects with contact information of their helpers
- 7 Apr: follow-up email to selected helpers to suggest they make contact with their projects
- 6 May: “APM is over!” email
- 14 May: Email to helpers and projects with link for post-APM surveys

During April I monitored the projects a little bit but largely left everyone to their own devices. After the surveys I wrote LinkedIn reviews for a number of the pytest helpers based on comments from the maintainers of the projects they worked on, to explicitly acknowledge their work (inspired by the idea of Let's all build a hat-rack by Leslie Hawthorn).

The “success” of APM varied greatly amongst the projects. Some had great communication and made great leaps. Others struggled with busy maintainers or absent pytest helpers. In some cases unfamiliarity what the open source project was doing, slowed efforts.

Maintainer activities

Five maintainers responded to the post-APM survey.

How did maintainers find out about APM?

- Via reddit post/pytest website
- Recommendation from #python IRC/pytest website

What did maintainers do?

- Discussed plans over email and chat
- Refactored docs, tests and code to better support pytest (e.g. test directory layout)
- Wrote new tests
- Wrote new tests using hypothesis
- Wrote new tests to get feedback/reviews from helpers
- Maintained communication with helpers
- Reviewed and merged PRs
- Contributed to pytest plugins
- Made a PyPI release for the first time

How much time did they spend? (self-reported)

- 50 hours
- 70-80 hours (“10 days of FT work”)
- 10 hours
- 4 hours
- 8 hours

All the maintainers indicated that they had a grasp of the basics of pytest. Most commented that they felt they still had a lot to learn.

Maintainer problems:

- Uneven contributions – bursts of activity / periods of silence and inactivity
- Matching projects and helpers with appropriate skills/interests
- Helpers should be encouraged to submit even minimal PRs
- Need to be explicit about expectations from each other
- Lack of availability of their own time

Would you recommend participating in Adopt Pytest Month to other projects?

- Maybe, depends 1
- Yes, definitely 4

Some comments:

“The gaps filled via, 'ask questions, get answers', are largely filled via google, forums, stack overflow, etc. The unfilled gap, is 'look at my code, tell me how to improve'”

“I wished that the APM lasted slightly longer than only a month! We benefited a lot from having an experienced pair of eyes in our project.”

“I think, the same way, that project sponsors were warned something akin to "Don't expect volunteers to write all your unit tests", the volunteers should maybe be coached something akin to "If you have to write one unit test, to show an example, or to get a smart setup/teardown, don't be afraid to do a PR." ”

“Main achievement of the month was allowing pytest to run all tests that nosetests ran. Initially this was considered as a 'trivial' effort, but turned out to be much more complex and took way more time (in fact the entire month).”

“It would probably have been good if all had been more explicit in the expectations to each other and in making more specific "promises" about what we planned to do. (I think they expected more from us and are a bit disappointed. They wanted to help, not to lead and do.)”

“Running the APM twice per year would be very nice. I would certainly like to participate in it (again as an apprentice).”

Pytest helper activities

Eleven helpers responded to the post-APM survey.

What did helpers do?
- Switched from unittest assertions to plain assert statements
- Changed the test layout, separating normal code from test code
- Removed boilerplate code related to test suites
- Fixed minor bugs that were discovered in the process
- Changed tests to use @parametrize
- Created some fixtures and demonstrated their usage and general benefits
- Created custom markers, including automatically marking tests based on the fixtures they use, so the user can exclude those from the command line
- Introduced some pytest plugins

How much time did they spend?
- About 40 hours (“About 10 hours a week”)
- About 8 hours
- 18-24 hours (“of 20% communication, 40% implementation and 40% code reviews”)
- 8 hours
- 15 hours

Would you participate in APM again?
- Depends, maybe – 5
- Definitely – 5
- Probably not, no – 1

How do you feel about pytest now, compared to before?
- More positive, keen to be involved in the community – 5
- The same, I will have a similar level of involvement – 5

Things pytesters learned more about

- More exposure to unittest
- How to run doctests with pytest
- How to add new command-line options to pytest
- pytest's support for nosetests
- pytest-cov
- Travis CI and coveralls
- using mock vs monkeypatch
- capturelog plugin
- pytest-django

Some comments:

“Maybe a #AdoptPytestMonth IRC channel where we can congregate and interact with each other.”

“In the past I had submitted bug reports, helped with minor documentation, and open sourced some of my own (useless) projects. But this is the first time I contributed code to someone else's open source project. And I could do it only because you organized the whole thing – which project to contribute to, when, and how, are the most important decisions that I think most people can't figure out and you figured it out for me.”

“One thing worth mentioning is that the project's maintainer was very interested and very active in our fork, creating issues himself, making code reviews and greatly contributing in general discussions.”

[Did you learn anything new about pytest?] “Hmm, not really, but the experience itself was very rewarding.”

“It is both easier to get started and more motivating to contribute to a project that you are a user of or is related to in some way. I am not sure exactly how, but maybe the matching of helper<->project could be done differently. ”

“Overall, I think this worked out okay. Perhaps having a schedule would help a bit, especially as far as expectations etc from each side? Maybe this isn't necessary as part of "pytest month" but could be a suggestion for contributors.”

[If we run Adopt Pytest Month again, what could we change to improve it?] “Maybe assigning one helper responsible so that he or she takes the task a bit more at heart.”

“With the project author doing most of the work, I felt I'd catalysed him into action and given him the confidence to do a big refactor of the project.”

Participating project overview

Bidict

Bidict is a small library and already had high test coverage with doctests, a CI server, and good documentation overall. The maintainer Joshua was interested in doing several interesting types of testing – performance, fuzz and property-based tests. The latter made it in during APM as can be seen at https://github.com/jab/bidict/blob/master/tests/test_hypothesis.py . Using the library Hypothesis, here is an example test:

```
from hypothesis import assume, given, strategy
from hypothesis.specifiers import dictionary

immutable_types_ = (None, bool, int, float, str, bytes, tuple(), frozenset())
immutable_types = reduce(or_, map(strategy, immutable_types_))
d = strategy(dictionary(immutable_types, immutable_types)).map(prune_dup_vals)

@given(d)
def test_bidirectional_mappings(d):
    b = bidict(d)
    for k, v in b.items():
        v_ = b[k]
        k_ = b.inv[v]
        assert k == k_ or both_nan(k, k_)
        assert v == v_ or both_nan(v, v_)
```

The Hypothesis tests did raise an edge case which needs to be handled with care – so, success!

They didn't end up using python-afl for fuzz testing, and are still looking for a standard way to do this.

Guessit

Guessit is a mature library that is well maintained with a small community of contributors and CI. All communication took place on Github issues and PRs.

The changes involved changing assertEqual statements to plain asserts, although tests remained as methods on a class inheriting from a custom class. Having pytest run doc tests meant removing boilerplate that was used to make nosetests run them, and instead adding to pytest.ini:

```
[pytest]
addopts = --ignore=setup.py --doctest-modules
```

Coursera-dl

Progress on this project was constrained by limited availability of the maintainer. They struggled to have much time for “back-and-forth”. However the maintainer was very appreciative of seeing simple step changes to replace assertEquals methods with plain asserts, and converting repetitive tests into parametrized instances. Initially they had written “helper” test methods such as assertEquals, which wrapped a plain assert to be executed by pytest – perhaps to make it easier to switch between different test runners. However in the end they chose to use pytest's plain asserts.

Qutebrowser

Qutebrowser was one of the most successful projects of APM. It had two pytest helpers who were able to work together very effectively. They began by discussing privately about how to begin and how to coordinate their work. They then started a fork of Qutebrowser and both contributed PRs to that fork and even issues, related to pytest adoption. The project's maintainer was very active in reviewing changes, asking questions and making comments. One of the first decisions they had to confront was choosing an appropriate directory layout for tests. They also collaborated to improve the pytest-qt plugin to work with PyQt5.

Below is an example of a widget test that creates a simple pytest fixture and uses it in a test with parametrization.

```
from collections import namedtuple

import pytest

from qutebrowser.browser import webview
from qutebrowser.mainwindow.statusbar.progress import Progress


@pytest.fixture
def progress_widget(qtbot, monkeypatch, config_stub):
    """Create a Progress widget and checks its initial state."""
    config_stub.data = {
        'colors': {'statusbar.progress.bg': 'black'},
        'fonts': {},
    }
    monkeypatch.setattr(
        'qutebrowser.mainwindow.statusbar.progress.style.config', config_stub)
    widget = Progress()
    qtbot.add_widget(widget)
    assert not widget.isVisible()
    assert not widget.isTextVisible()
    return widget


# mock tab object
Tab = namedtuple('Tab', 'progress load_status')


@pytest.mark.parametrize('tab, expected_visible', [
    (Tab(15, webview.LoadStatus.loading), True),
    (Tab(100, webview.LoadStatus.success), False),
    (Tab(100, webview.LoadStatus.error), False),
    (Tab(100, webview.LoadStatus.warn), False),
    (Tab(100, webview.LoadStatus.none), False),
])
def test_tab_changed(progress_widget, tab, expected_visible):
    """Test that progress widget value and visibility state match expectations.

    This uses a dummy Tab object.

    Args:
        progress_widget: Progress widget that will be tested.
    """
    progress_widget.on_tab_changed(tab)
    actual = progress_widget.value(), progress_widget.isVisible()
    expected = tab.progress, expected_visible
    assert actual == expected
```

They also added a hook in conftest.py to automatically add a marker “gui” to any test using the qtbot fixture. This makes it easy to select or deselect such tests to, for example, avoid running such tests when an X environment is not available.

```python
def pytest_collection_modifyitems(items):
    """Automatically add a 'gui' marker to all gui-related tests.

    pytest hook called after collection has been performed, adds a marker
    named "gui" which can be used to filter gui tests from the command line.
    For example:

        py.test -m "not gui"  # run all tests except gui tests
        py.test -m "gui"  # run only gui tests

    Args:
        items: list of _pytest.main.Node items, where each item represents
               a python test that will be executed.

    Reference:
        http://pytest.org/latest/plugins.html
    """
    for item in items:
        if 'qtbot' in getattr(item, 'fixturenames', ()):
            item.add_marker('gui')
```

Arkestra

Arkestra is a large project and had three pytest helpers assigned, although one was not able to contribute due to other constraints so it became effectively two. The project maintainer was extremely busy during APM which hampered the ability of the helpers to make progress.

Kallithea

Kallithea also had three pytest helpers assigned and it also became effectively two as one did not participate. Discussion took place on the development mailing list which is also where patches and PRs are reviewed. (This is unusual in the age of Github, but as Kallithea has similar functionality to Github it is natural that they are working towards eating their own dogfood.) Kallithea were not certain about adopting pytest, and during APM wanted to keep their test suite able to run under either nose or pytest. Efforts to determine why pytest ran a different number of tests to nose, with different results, took a while to resolve. While pytest can run simple nose tests, it cannot run all, such as detailed nose yield tests. These were rewritten as pytest-style parametrize tests.

Nefertari

Nefertari had two helpers assigned although only one contributed and he was not able to make much progress. This collaboration likely could have benefited from the guidance a more experienced tester with more time to contribute.

Trump

Trump was a newly-open-sourced project with fairly extensive code but no automated testing. Trump had three helpers assigned, although not much was achieved beyond an initial discussion of coding style and code organisation. As a complex project with no existing test suite, Trump was perhaps the most intimidating of the APM projects to leap into, and could have benefited from a more structured or guided approach to communication and expectations.

Ideas for the future

For the pytest project/community:

- Develop a pytest guide specifically for converting an existing test suite, with specific advice for converting from each of unittest and nose.
- Develop documentation for a handful of the most popular and useful pytest plugins
- Improved integration with Hypothesis for property-based testing
- Guidance/improved support for python-afl for fuzz testing

As an organiser some things I found difficult were:

- Trying to respect people's privacy (by not CCing 30 people in an email) while enabling people to communicate with each other. Strongly recommending participants take part in the pytest mailing list or another mailing list may help here.
- Following if activity was taking place. In some cases it was taking place by email. It may help to instruct projects to open an "umbrella task" related to APM and have this be the official place for helpers to introduce themselves and conduct meta discussion.
- Some projects took part "unofficially", as in, APM was mentioned to them on a Github issue, but they didn't complete my form to take part. It was difficult for me to know how seriously interested they were. It would be easier to require project maintainers to complete the form (which could also be made much less onerous if the focus deliberately changes to smaller projects)

If APM or something similar is run again:

- Encourage greater use of mailing list and pytest IRC room for pytesters on different projects to communicate with each other
- Focus on small one-person projects, as the reduced communication costs and "moving parts" of changing code makes collaborating much easier
- Look for documentation to point existing projects to, like "the Joel Test" for encouraging open source contributors (following a style guide, CI, etc)
- Larger team of "supervisors", specifically to monitor progress, encourage communication
- Publish some suggested communication schedules, guides to what was achieved in the past. The idea of both converting existing test suites and adding new tests is definitely too ambitious. It would be a lot more realistic to focus on only converting an existing test suite
- Emphasise more the effort involved in onboarding to a new project (can be non-trivial)
- Consider collecting projects first and having pytesters indicate their preference for a project, to have better matching of interests/skills

- Outreachy asks volunteers to make some small effort to demonstrate their interest before assigning them to a project. For pytest helpers who are not already contributors to pytest in some way this may be a good idea
- An interesting idea could be to run something like APM, but for projects that are already using pytest, who want to “level up”.

Conclusion

The idea of APM was to pair experienced pytest users (“helpers”) with open source projects for one month, to help them begin to see the benefits of using pytest. I thought the project would mainly be about winning over new pytest users. While the projects that participated did appreciate the help they received, it also proved to be a great way to offer more people a way to participate in the pytest community, and celebrate the project. 26 users offered to be helpers, of which only 9 were existing contributors to the pytest codebase.

I also thought we might be inundated with projects who wanted help, but this was not quite true. Finding appropriate projects to take part was hit and miss, with some success coming from pytest users reaching out to other projects they were already familiar with. Smaller (1 person) projects also proved to be better targets than larger or more complicated projects. 16 helpers were assigned among 8 projects, whose individual progress has been summarised above. Assigning multiple helpers to a single project sometimes meant that they had an opportunity to learn from each other; however this also necessitated a change in their roles and responsibilities to each other (which was not noted explicitly).

As is often the case with open source projects, contributions were frequently interrupted (or never began) due to other commitments. In some cases, projects proved too complex for helpers to make a start. More successful cases were marked by good communication (frequent checking in and confirmation of shared understanding), coding and reviewing effort on both sides, and undertaking of many small steps. My initial discussion of converting an existing code base and also writing new tests mostly proved to be unrealistic – converting (part of) an existing code base would be a more realistic target to speak of.

APM showed that there is interest in both volunteering pytest help, and receiving it, if people can be appropriately paired. It is also showed there would probably be a good audience for more documentation on converting an existing test suite to fully take advantage of pytest's offerings. Perhaps the greatest benefit of APM in some sense was simply providing an opportunity for targeted code review.

As a pilot instance, I emphasised to participants that there were no guarantees about outcomes. For pytest or any other project interested in running a similar program, it would be wise to try some of the ideas above to increase the likelihood of a positive outcome for all participants.

Appendices

These are included for reference as to the kind of guidance helpers and projects were given, but I wouldn't recommend using them without changes.

Website information page

(from http://pytest.org/latest/adopt.html)

April 2015 is "adopt pytest month"
Are you an enthusiastic pytest user, the local testing guru in your workplace? Or are you considering using pytest for your open source project, but not sure how to get started? Then you may be interested in "adopt pytest month"!

We will pair experienced pytest users with open source projects, for a month's effort of getting new development teams started with pytest.

In 2015 we are trying this for the first time. In February and March 2015 we will gather volunteers on both sides, in April we will do the work, and in May we will evaluate how it went. This effort is being coordinated by Brianna Laugher. If you have any questions or comments, you can raise them on the @pytestdotorg twitter account the issue tracker or the pytest-dev mailing list.

The ideal pytest helper

- will be able to commit 2-4 hours a week to working with their particular project (this might involve joining their mailing list, installing the software and exploring any existing tests, offering advice, writing some example tests)
- feels confident in using pytest (e.g. has explored command line options, knows how to write parametrized tests, has an idea about conftest contents)
- does not need to be an expert in every aspect!

Pytest helpers, sign up here! (preferably in February, hard deadline 22 March)

The ideal partner project

- is open source, and predominantly written in Python
- has an automated/documented install process for developers
- has more than one core developer
- has at least one official release (e.g. is available on pypi)
- has the support of the core development team, in trying out pytest adoption
- has no tests... or 100% test coverage... or somewhere in between!

Partner projects, sign up here! (by 22 March)

What does it mean to "adopt pytest"?
There can be many different definitions of "success". Pytest can run many nose and unittest tests by default, so using pytest as your testrunner may be possible from day 1. Job done, right?

Progressive success might look like:

- tests can be run (by pytest) without errors (there may be failures)
- tests can be run (by pytest) without failures
- test runner is integrated into CI server
- existing tests are rewritten to take advantage of pytest features - this can happen in several iterations, for example:
    - changing to native assert statements (pycmd has a script to help with that, `pyconvert_unittest.py`)
    - changing setUp/tearDown methods to fixtures
    - adding markers
    - other changes to reduce boilerplate
- assess needs for future tests to be written, e.g. new fixtures, distributed testing tweaks

“Success” should also include that the development team feels comfortable with their knowledge of how to use pytest. In fact this is probably more important than anything else. So spending a lot of time on communication, giving examples, etc will probably be important - both in running the tests, and in writing them.

It may be after the month is up, the partner project decides that pytest is not right for it. That’s okay - hopefully the pytest team will also learn something about its weaknesses or deficiencies.

Other ways to help
Promote! Do your favourite open source Python projects use pytest? If not, why not tell them about this page?

Helpers sign up form

Your name

Your email

Your development platforms

- Linux
- Windows
- OSX
- iOS
- Android

- Other

Your experience in pytest

- Writing basic unit tests
- command line options
- parametrized tests
- fixtures
- conftest plugins/hooks
- xdist
- tox
- configuring with CI server
- using it to run non-Python tests
- converting an existing test suite
- other

Experience in other Python areas

- Python 3
- Django
- Pyramid
- Zope
- Twisted
- General web dev
- async
- databases
- Scientific programming (numpy, scipy)
- GUI frameworks
- Graphics
- Games
- Other

Are you prepared to commit at least 2-4 hours a week during April to work with an open source project?

Any other comments

Project sign up form

Name of project

Project website

Name of contact person

Email of contact person

What license is your project available under?

Which programming language is your project mostly written in?

Link to your project's contributing/installation guide for developers:

Does the core development team support this effort to potentially adopt pytest?

Does your project have more than 1 core developer?

Has your project been released on pypi?

What tests does your project have at the moment?

- No tests
- Some tests, but they are not run regularly (no CI server)
- Some tests, with CI server
- Comprehensive test coverage, with CI server

What frameworks/libraries/areas does your project make significant use of?

- Django
- Pyramid
- Zope
- Twisted
- async
- Databases
- Scientific libs (numpy, scipy)
- GUI framework

- Graphics
- Games
- Other

What has been your project's experience with testing so far?

I understand that if we get a pytest helper, they will try to help us use pytest, not promise to write all our tests for us.

I am prepared to complete a survey after the month, to help evaluate if it was a success or not

Any other comments

Email to helpers

(Link to spreadsheets showing corresponding projects and helpers)

Please have a look at which project your name is against, and if you can't work with it for whatever reason, please let me know ASAP, as there other helpers I can put instead.

For larger projects I have put 2 or 3 helpers together. I have tried to put people who have a range of pytest experience and also other python experience relevant to the project where possible. Introduce yourselves to each other :)

If you are happy with the project I have put you with, please make contact with the project person and start having a look at the project itself. Install it, build it, play with it. Join the mailing list and IRC channel if they exist. Read the docs and any existing tests. Run the tests and see if they all pass! (Also in the existing test runner.) Take note of any test files which are noticably slower than the others. Try running them distributed and see if any extra failures pop up, this is a notoriously good way of bringing out test interdependencies or reliance on globals.

Before you start writing any tests, be sure to discuss with the project person about what the project is, what their goals are, what their knowledge of testing and pytest is, what they see as areas that need attention, any specific things they want to achieve. Try to make a rough plan which has lots of steps, each step being an achievement in its own right. A month is not very long, so for a large project don't expect to get everything 100% perfect.

Some projects have not yet made a pypi release yet - see if they want help with that. Some projects don't have CI set up yet - I would suggest that is a good early goal to set, along with getting into the practice of having all existing tests passing. Also record the test coverage, although keep in mind it can be misleading.

Projects that are closer to being libraries may be happy with just unit tests. Projects closer to applications, will probably benefit from some functional/system tests. Find out which areas of the code are bug-prone, or are already planned to be rewritten/refactored - these are excellent candidates for functional tests, NOT unit tests.

If you find yourself rewriting tests several times, this is not a bad thing at all. Tests should evolve as everyone's knowledge evolves and also as the code evolves.

When you submit PRs, be extra verbose at the start to explain what you are doing and why. Encourage them to ask you about anything they don't understand. Suggest useful command line options for them to use when running the tests.

Also, keep in mind that this is an opportunity to see pytest from the eyes of a newcomer. Take notes of what seems confusing to them, or where you notice some detail missing in the pytest docs - or some functionality in pytest itself. I will ask you all about this at the end and it's much easier to record these details as you go rather than try to remember everything at the end of the month!

Remember if at any point you have any doubts about working with your project, you can always email me. For help with how to do something specific with pytest, please write to the pytest-dev mailing list.

OK that's all I've got for now....happy pytesting :)

Email to projects

I just wanted to check if your helper(s) have made contact with you yet. They are: (names and emails)

If they haven't, let me know and I'll see what's going on.

Some suggestions:

Let them know the best communication channels to discuss. IRC, mailing list, issue tracker, wiki, google doc? Even if you use IRC or email, I recommend to keep a summary somewhere publicly accessible for future contributors.

Early goals you might want to consider:
· Preparing for first pypi release, if relevant
· Setting up continuous integration (CI) server - this runs all the tests on every commit, so you find out ASAP when a test starts failing. Also very helpful if you support multiple environments such as different Python versions
· Also integrating test coverage (tells you which areas of code are exercised by the tests and which are not - 100% test coverage doesn't mean you have perfect tests but it's a start!)
· Examples of test driven development - are there any bugs in your issue tracker which don't have

example failing tests? These would be an excellent addition!

Are there any areas of pytest you are curious about? Do you want to run tests distributed over multiple cores/computers? Over multiple versions of Python? Is there something you want to test but you don't have any idea how you'd go about it? These are things to discuss with your helpers!

If your project is more like a library, you may be happy with just unit tests.

If your project is more like an application, has bigger moving parts that interact with each other in complicated ways, you will probably want some functional/system tests. If there are areas of your project that are prone to bugs, or that you are already planning to refactor/rewrite, or that just kind of give you a feeling like "ugh yeah, this area is not great...", these are great candidates for functional tests rather than unit tests - so be sure to tell your pytest helpers about this.

Your job as the project expert is to ensure that tests describe what the behaviour *should be*. The pytest helpers as newcomers to your project, may find it difficult to determine what the expected behaviour is - so it's important to scrutinize the assert statements in their tests to make sure it describes what *should be* rather than what *is*!

I hope that the helpers will work on existing tests in a progressive fashion, ie not changing everything at once. This makes it easier to see how progress can be made in steps, it doesn't have to be a big bang approach. So it's not a problem if the same tests are revisited many times, it's normal.

When you are reviewing PRs, ask lots of questions to understand what every option is doing. A big part of the month should be exchanging knowledge rather than code. Run the tests on your command line or in your IDE and try to get familiar with the different command line options ("it would be cool if I could just run this test... it would be cool if it stopped as soon as one test failed...it would be cool if it printed out more/less failure information..." etc!).

If you remember, take notes about what you learn about pytest during the month, especially what seems complicated or confusing - I will ask you about it at the end of the month, as it is an opportunity for us to see pytest from the eyes of a newcomer and see where we can improve.

Finally we are all optimists, what I have mentioned here could encompass many weeks of work, so it's not bad to be ambitious but in April we may only make a few steps, rather than getting all the way to the summit of perfect testing. That is why I recommend to have a plan with progressive but self contained steps of success.

Remember if at any point you have any doubts about working with your helpers, you can always email me. For pytest help everyone should be using the pytest-dev mailing list.

Post-APM helper feedback survey

Name

Project you worked on

How you would like to be identified in a summary report

- Name and Github link
- First name only
- Anonymous
- Other

What was your impression of the project you worked on, in a technical sense? For example, what was the state of its tests like?

What work did you contribute to the project during Adopt Pytest Month? How much time in total?

How did your actual achievements compare to what you initially planned to do?

Did you learn anything new about pytest, or get more experience with an aspect of pytest you hadn't used much before?

Was there something you wanted to do with pytest for this project, but it wasn't possible or you didn't know how? Did you notice any features or documentation lacking, or find a new bug?

What things caused problems or stopped you from being productive?

If we run Adopt Pytest Month again, what could we change to improve it?

How do you feel about pytest now, compared to before?

- More positive, keen to be involved in the community
- The same, I will have a similar level of involvement
- Slightly more negative
- A lot more negative
- Other

Would you participate in Adopt Pytest Month again?

- Definitely
- Depends, maybe
- Probably not, no

Any additional private comments, only for Brianna?

Project feedback survey

Name

Your project

What did you know about pytest before Adopt Pytest Month?

How do you feel about your pytest knowledge now?

What work did you contribute related to the Adopt Pytest Month? How much time in total?

What is the status of pytest adoption with your project now - will you continue using it? How did the actual achievements compare to what was initially planned?

How was the experience of working with the pytest helpers - newcomers to your project?

If we run Adopt Pytest Month again, what could we change to improve it?

Would you recommend participating in Adopt Pytest Month to other projects?

- Definitely, yes
- Maybe, depends
- Probably not, no

Any comments specifically about your pytest helpers?

Any additional private comments, only for Brianna?