



AlekSandrDr вчера в 17:14

## Python Testing с pytest. Плагины, ГЛАВА 5

Автор оригинала: Okken Brian

Python

Перевод

Tutorial



Достаточно мощный pytest прямо из коробки, становится еще лучше, когда вы добавляете в него микс из плагинов. Кодовая база pytest структурирована настройками и расширениями, и есть хуки, доступные для модификации и улучшений через плагины.

The Pragmatic Programmers

### Python Testing with pytest

Simple, Rapid, Effective, and Scalable

Brian Okken

edited by Katharine Dvorak



Примеры в этой книге написаны с использованием Python 3.6 и pytest 3.2. pytest 3.2 поддерживает Python 2.6, 2.7 и Python 3.3+.

Исходный код для проекта Tasks, а также для всех тестов, показанных в этой книге, доступен по [ссылке](#) на веб-странице книги в [pragprog.com](#). Вам не нужно загружать исходный код, чтобы понять тестовый код; тестовый код представлен в удобной форме в примерах. Но что бы следовать вместе с задачами проекта, или адаптировать примеры тестирования для проверки своего собственного проекта (если у вас развязаны!), вы должны перейти на веб-страницу книги и скачать работу. Там же, на веб-странице книги есть ссылка для сообщений errata и дискуссионный форум.

Под спойлером приведен список статей этой серии.

[Оглавление](#)

### Поехали дальше!

Возможно вы удивитесь узнав, что вы уже написали какие то плагины, если вы проработали предыдущие главы в этой книге. Каждый раз, когда вы помещаете фикстуры и/или hook-функции в файл `conftest.py` верхнего уровня проекта, вы создаёте локальный плагин `conftest`. Это небольшая дополнительная работа по преобразованию этих файлов `conftest.py` в устанавливаемые плагины, которые вы можете разделить между проектами, с другими людьми или с миром.

Мы начнем эту главу, с ответа на вопрос, где искать сторонние плагины. Довольно много плагинов доступны, так что есть приличный шанс, кто — то уже написал изменения, которые вы хотите сделать в `pytest`. Так как мы будем рассматривать плагины с открытым исходным кодом, если плагин делает почти то, что вы хотите сделать, но не совсем, вы можете развить его, или использовать его в качестве эталона для собственного плагина. Хотя эта глава посвящена созданию ваших собственных плагинов, Приложение 3, плагин *Sampler Pack*, на странице включен, чтобы дать вам почувствовать вкус того, что возможно.

В этой главе вы узнаете, как создавать плагины, и я укажу вам правильное направление для их тестирования, упаковки и распространения. Полная тема упаковки и распространения Python, слишком обширна и претендует на собственную книгу, поэтому мы не будем охватывать, вы получите достаточно сведений, чтобы иметь возможность обмениваться плагинами с вашей командой. Я также расскажу о некоторых способах для создания плагинов с поддержкой PyPI и наименьшим количеством работы.

## Поиск плагинов

Вы можете найти сторонние плагины `pytest` в нескольких местах. Плагины, перечисленные в Приложении 3, *Plugin Sampler Pack*, на стр. 16 доступны для загрузки с PyPI. Тем не менее, это не единственное место для поиска отличных плагинов `pytest`.

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

На главном сайте документации `pytest` есть страница, в которой рассказывается об установке и использовании плагинов `pytest` и перечислено несколько распространенных плагинов.

<https://pypi.python.org>

Python Package Index (PyPI) — это отличное место для получения большого количества пакетов Python, но также отличное место для поиска плагинов `pytest`. При поиске плагинов `pytest` достаточно ввести “`pytest`,” “`pytest -`” или “`-pytest`” в поле поиска, так как большинство `pytest` плагинов начинаются с “`pytest -`” или заканчиваются на “`-pytest`.”

<https://github.com/pytest-dev>

Группа «`pytest-dev`» на GitHub — это место, где хранится исходный код `pytest`. Кроме того, здесь вы можете найти популярные плагины `pytest` которые должны поддерживаться в долгосрочной перспективе командой ядра `pytest`.

## Установка плагинов

Плагины `pytest` устанавливаются с `pip`, как и другие пакеты Python. Однако, вы можете использовать `pip` несколькими способами для установки плагинов.

### Установка из PyPI

Поскольку PyPI является местоположением по умолчанию для `pip`, установка плагинов из PyPI является самым простым методом. Давайте установим плагин `pytest-cov`:

```
$ pip install pytest-cov
```

Будет установлена последняя стабильная версия от PyPI.

### Установка определенной версии из PyPI

Если вы хотите конкретную версию плагина, вы можете указать версию после `==`:

```
$ pip install pytest-cov==2.4.0
```

### Установка из файла .tar.gz или .whl

Пакеты на PyPI распространяются как zip-файлы с расширениями `.tar.gz` и/или `.whl`. Они часто упоминаются как «`tar balls`» и «`wheels`». Если у вас возникли проблемы с попыткой работать с PyPI напрямую (что может случиться с брандмауэрами и другими сетевыми осложнениями), вы можете загрузить либо `.tar.gz`, либо `.whl` и установить из этого-того.

Вам не нужно распаковывать или танцевать с бубном; просто укажите `pip` на него:

```
$ pip install pytest-cov-2.4.0.tar.gz
# or
$ pip install pytest_cov-2.4.0-py2.py3-none-any.whl
```

## Установка из локального каталога

Вы можете иметь заначку плагинов (и других пакетов Python) в локальном или общем каталоге в формате `.tar.gz` или `.whl` и использовать вместо PyPI для установки плагинов:

```
$ mkdir some_plugins
$ cp pytest_cov-2.4.0-py2.py3-none-any.whl some_plugins/
$ pip install --no-index --find-links=./some_plugins/ pytest-cov
```

`--no-index` указывает `pip` не подключаться к PyPI. `--find-links=./some_plugins/` указывает `pip` искать в каталоге `some_plugins`. Этот метод особенно полезен, если у вас есть как сторонние, так и собственные плагины, хранящиеся локально, а также если вы создаете новые виртуальные среды для непрерывной интеграции или с `tox`. (Мы поговорим как о `tox`, так и о непрерывной интеграции в главе 7, используя другими инструментами, на странице 125.)

Обратите внимание, что с помощью метода установки локального каталога вы можете установить несколько версий и указать, какую версию хотите, добавив `==` и номер версии:

```
$ pip install --no-index --find-links=./some_plugins/ pytest-cov==2.4.0
```

## Установка из репозитория Git

Вы можете установить плагины непосредственно из Git-репозитория в этом случае GitHub:

```
$ pip install git+https://github.com/pytest-dev/pytest-cov
```

Можно также указать тег версии:

```
$ pip install git+https://github.com/pytest-dev/pytest-cov@v2.4.0
```

Или можно указать ветвь:

```
$ pip install git+https://github.com/pytest-dev/pytest-cov@master
```

Установка из репозитория Git особенно полезна, если вы храните свою собственную работу в Git или если требуемая версия плагина или г отсутствует в PyPI.

*Примечание переводчика:*

`pip` поддерживает установку из Git, Mercurial, Subversion и Bazaar и определяет тип VCS, используя префиксы url: «git+», «hg+», «svn+», «bzr+».

Более подробно можно ознакомиться в документации PyPI

## Написание собственных плагинов

Многие сторонние плагины содержат довольно много кода. Это одна из причин, по которой мы используем их-чтобы сэкономить нам время разработку всего этого самостоятельно. Тем не менее, для вашего конкретного кода вы, несомненно, придумаете специальные фикстуры и модификации, которые помогут вам его протестировать. Создавая плагин, вы можете легко поделиться даже несколькими фикстурами, кот вы хотите разделить между несколькими проектами. Вы можете поделиться этими изменениями с несколькими проектами — и, возможно, с

остальным миром, — разрабатывая и распространяя свои собственные плагины. Это довольно легко сделать. В этом разделе мы разработаем небольшую модификацию поведения `pytest`, упакуем ее как плагин, протестируем и рассмотрим, как ее распространять.

Плагины могут включать `hook`-функции, которые изменяют поведение `pytest`. Поскольку `pytest` был разработан с целью позволить плагинам менять поведение `pytest`, доступно множество `hook`-функций. `hook`-и для `pytest` указаны на [сайте документации pytest](#). В нашем примере мы создадим плагин, который изменит внешний вид статуса теста. Добавим параметр командной строки, чтобы включить это новое поведение. Добавим текст в выходной заголовок. В частности, мы изменим все индикаторы состояния `FAILED` (неудачный) на `OPPORTUNITY` (перспективный) для усовершенствования, изменим `F` на `O`, и добавим `"Thanks for running the tests"` (Спасибо за выполнение тестов) к заголовку. Для этого будем использовать опцию `--nice`.

Чтобы сохранить изменения поведения отдельно от обсуждения механики плагинов, мы внесем изменения в `conftest.py` до того, как превратим его в распространяемый плагин. Вам не нужно запускать плагины таким образом. Но часто изменения, которые вы намеревались использовать только в одном проекте, станут достаточно полезными, чтобы поделиться ими и превратиться в плагин. Поэтому мы начнем с добавления функциональности в файл `conftest.py`, а затем, после того, как все заработает в `conftest.py`, переместим код в пакет.

Вернемся к проекту `Tasks`. В разделе "ожидание исключений" на странице 30 мы написали несколько тестов, которые проверяли, вызываются исключения, если кто-то неправильно вызвал функцию `API`. Похоже, мы пропустили хотя бы несколько возможных состояний ошибки.

Вот еще пара тестов:

#### ch5/a/tasks\_proj/tests/func/test\_api\_exceptions.py

```
"""Проверка ожидаемых исключений из использования API wrong."""
import pytest
import tasks
from tasks import Task

@pytest.mark.usefixtures('tasks_db')
class TestAdd():
    """Тесты, связанные с tasks.add()."""

    def test_missing_summary(self):
        """Следует поднять исключение, если summary missing."""
        with pytest.raises(ValueError):
            tasks.add(Task(owner='bob'))

    def test_done_not_bool(self):
        """Должно вызвать исключение, если done не является bool."""
        with pytest.raises(ValueError):
            tasks.add(Task(summary='summary', done='True'))
```

Давайте запустим их, чтобы проверить, проходят ли они:

```
$ cd /path/to/code/ch5/a/tasks_proj
$ pytest
===== test session starts =====

collected 57 items
tests/func/test_add.py ...
tests/func/test_add_variety.py .....
tests/func/test_add_variety2.py .....
tests/func/test_api_exceptions.py .F.....
tests/func/test_unique_id.py .
tests/unit/test_task.py ....

===== FAILURES =====

_____ TestAdd.test_done_not_bool _____

self = <func.test_api_exceptions.TestAdd object at 0x103a71a20>

    def test_done_not_bool(self):
        """Should raise an exception if done is not a bool."""
        with pytest.raises(ValueError):
```

```
> tasks.add(Task(summary='summary', done='True'))
E Failed: DID NOT RAISE <class 'ValueError'>

tests/func/test_api_exceptions.py:20: Failed

===== 1 failed, 56 passed in 0.28 seconds =====
```

Давайте запустим его снова с `-v` для подробностей. Поскольку вы уже видели трассировку, вы можете отключить ее, нажав `--tb=no`.

А теперь давайте сосредоточимся на новых тестах с `-k TestAdd`, который работает, потому что нет никаких других тестов с именами, которые содержат "TestAdd."

Мы могли бы "всё бросить" и попытаться исправить этот тест (и мы сделаем это позже), но сейчас мы сосредоточимся на попытке сделать сообщения о неудаче (failures) более приятными для разработчиков.

Давайте начнем с добавления сообщения "thank you" в заголовок, который вы можете сделать с помощью хука pytest под названием `pytest_report_header()`.

**ch5/b/tasks\_proj/tests/conftest.py**

```
def pytest_report_header():
    """Благодарность тестеру за выполнение тестов."""
    return "Thanks for running the tests."
```

Очевидно, что печатать благодарственное сообщение довольно глупо. Однако возможность добавления информации в заголовок может быть расширена. Можно добавить имя пользователя, указать используемое оборудование и тестируемые версии. Вообще, все, что вы можете преобразовать в строку, можно вставить в заголовок теста.

Затем мы изменим отчет о состоянии теста, чтобы изменить F на O и FAILED на OPPORTUNITY for improvement. Есть хук, который позволяет интрижку: `pytest_report_teststatus()`:

**ch5/b/tasks\_proj/tests/conftest.py**

```
def pytest_report_teststatus(report):
    """Превращает неудачи в возможности."""
    if report.when == 'call' and report.failed:
        return (report.outcome, 'O', 'OPPORTUNITY for improvement')
```

И теперь мы имеем как раз выход, который мы искали. Тестовый сеанс без флага `--verbose` показывает O для сбоев, то есть, возможности улучшения.

```
$ cd /path/to/code/ch5/b/tasks_proj/tests/func
$ pytest --tb=no test_api_exceptions.py -k TestAdd

===== test session starts =====

Thanks for running the tests.
collected 9 items
test_api_exceptions.py .O

===== 7 tests deselected =====
===== 1 failed, 1 passed, 7 deselected in 0.06 seconds =====
```

С флагом `-v` или `--verbose` будет лучше:

```
$ pytest -v --tb=no test_api_exceptions.py -k TestAdd

===== test session starts =====
```

```

Thanks for running the tests.
collected 9 items
test_api_exceptions.py::TestAdd::test_missing_summary PASSED
test_api_exceptions.py::TestAdd::test_done_not_bool OPPORTUNITY for improvement

===== 7 tests deselected =====
===== 1 failed, 1 passed, 7 deselected in 0.07 seconds =====

```

Последнее изменение, которое мы сделаем, это добавим параметр командной строки, `--nice`, чтобы изменения нашего статуса происходили только если подставить `--nice`:

```

def pytest_addoption(parser):
    """Включает nice функцию с опцией --nice."""
    group = parser.getgroup('nice')
    group.addoption("--nice", action="store_true",
                    help="nice: turn failures into opportunities")

def pytest_report_header():
    """Благодарность тестеру за выполнение тестов."""
    if pytest.config.getoption('nice'):
        return "Thanks for running the tests."

def pytest_report_teststatus(report):
    """Превращает неудачи в возможности."""
    if report.when == 'call':
        if report.failed and pytest.config.getoption('nice'):
            return (report.outcome, 'O', 'OPPORTUNITY for improvement')

```

Стоит заметить, что для этого плагина мы используем только пару хуков. Есть много других, которые можно найти на [главном сайте документации Pytest](#).

Теперь мы можем вручную протестировать наш плагин, просто запустив его в нашем примере. Во-первых, без опции `--nice`, чтобы убедиться отображается только имя пользователя:

```

$ cd /path/to/code/ch5/c/tasks_proj/tests/func
$ pytest --tb=no test_api_exceptions.py -k TestAdd

===== test session starts =====

collected 9 items
test_api_exceptions.py .F

===== 7 tests deselected =====
===== 1 failed, 1 passed, 7 deselected in 0.07 seconds =====

```

Теперь с `--nice`:

```

$ pytest --nice --tb=no test_api_exceptions.py -k TestAdd

===== test session starts =====

Thanks for running the tests.
collected 9 items
test_api_exceptions.py .O

===== 7 tests deselected =====
===== 1 failed, 1 passed, 7 deselected in 0.07 seconds =====

```

Теперь с `--nice` и `--verbose`:

```

$ pytest -v --nice --tb=no test_api_exceptions.py -k TestAdd

===== test session starts =====

```

```

Thanks for running the tests.
collected 9 items
test_api_exceptions.py::TestAdd::test_missing_summary PASSED
test_api_exceptions.py::TestAdd::test_done_not_bool OPPORTUNITY for improvement

===== 7 tests deselected =====
===== 1 failed, 1 passed, 7 deselected in 0.06 seconds =====

```

Отлично! Все изменения, которые мы хотели сделать, сделаны примерно в десятке строк кода файла `conftest.py`. Далее мы переместим : код в структуру плагина.

## Создание устанавливаемого плагина

Процесс обмена плагинами с другими пользователями четко определен. Даже если вы никогда не включите свой собственный плагин в Pyf пройдя через этот процесс, вам будет легче читать код из плагинов с открытым исходным кодом, и у вас будет больше возможностей оценить помогут они вам или нет.

Было бы излишним полностью охватывать `packaging` и `distribution` пакетов Python в этой книге, так как эта тема хорошо документирована в месте. Тут и здесь и еще здесь на русском. Тем не менее, перейти от локального подключаемого модуля конфигурации, который мы создали в предыдущем разделе, к чему-то устанавливаемому с помощью `pip`, является несложной задачей. ,

Во-первых, нам нужно создать новый каталог для размещения нашего кода плагина. Неважно, как вы это называете, но, поскольку мы создаем плагин для флага «nice», давайте назовем его «pytest-nice». У нас будет два файла в этом новом каталоге: `pytest_nice.py` и `setup.py`. (Каталог тестов будет обсуждаться в разделе «Плагины тестирования» на странице.105.)

```

|  LICENSE
|  pytest_nice.py
|  setup.py
|
└── tests
    |  conftest.py
    |  test_nice.py

```

В `pytest_nice.py`, мы поместим точное содержимое нашего `conftest.py`, которое было связано с этой функцией (и извлечем его из `tasks_proj/tests/conftest.py`):

### ch5/pytest-nice/pytest\_nice.py

```

"""Код для pytest-nice плагин."""

import pytest

def pytest_addoption(parser):
    """Включает nice функцию с опцией --nice."""
    group = parser.getgroup('nice')
    group.addoption("--nice", action="store_true",
                    help="nice: turn FAILED into OPPORTUNITY for improvement")

def pytest_report_header():
    """Благодарность тестеру за выполнение тестов."""
    if pytest.config.getoption('nice'):
        return "Thanks for running the tests."

def pytest_report_teststatus(report):
    """Превращает неудачи в возможности."""
    if report.when == 'call':
        if report.failed and pytest.config.getoption('nice'):
            return (report.outcome, 'O', 'OPPORTUNITY for improvement')

```

В `setup.py` нам нужен максимально минимальный вызов `setup()`:

### ch5/pytest-nice/setup.py

```

"""Setup для pytest-nice plugin."""

from setuptools import setup

setup(
    name='pytest-nice',
    version='0.1.0',
    description='Плагин Pytest, чтобы включить FAILURE into OPPORTUNITY',
    url='https://место/где/содержится/информация/на/этот/пакет',
    author='Ваше имя',
    author_email='your_email@somewhere.com',
    license='proprietary',
    py_modules=['pytest_nice'],
    install_requires=['pytest'],
    entry_points={'pytest11': ['nice = pytest_nice', ], },
)

```

Вам понадобится больше информации в настройках, если вы собираетесь распространять ее среди широкой аудитории или в интернете. (для небольшой команды или просто для себя этого будет достаточно).

Вы можете включить еще какие то параметры для `setup()`; а тут у нас только обязательные поля. Поле версии является версией этого плагино это целиком зависит от вас, когда вы поднимаете версию. Поле URL обязательно для заполнения. Вы можете оставить его пустым, но вы получите предупреждение. Поля `author` и `author_email` можно заменить на `maintainer` и `maintainer_email`, но одна из этих пар должна быть. Поле `license`-лицензия представляет собой короткое текстовое поле. Это может быть одна из многих лицензий с открытым исходным кодом ваше имя или компании, или что-то подходящее для вас. Запись `py_modules` перечисляет `pytest_nice` как наш единственный модуль для этого плагина. Хотя это список, и вы можете включить более одного модуля, если бы у меня было больше одного, я бы использовал пакет и поместил все модули в один каталог.

До сих пор все параметры `setup()` являются стандартными и используются для всех инсталляторов Python. Частью, которая отличается для плагинов Pytest, является параметр `entry_points`. Мы перечислили `entry_points={'pytest11': ['nice = pytest_nice', ], },`. Функция `entry_points` является стандартной для `setuptools`, но `pytest11` специальный идентификатор, который ищет `pytest`. В этой строке мы сообщаем `pytest`, что `nice`-это имя нашего плагина, а `pytest_nice`-имя модуля, в котором живет наш плагин. Если бы мы использовали пакет, наша запись здесь была бы:

Я еще не говорил о файле `README.rst`. Некоторая форма README является требованием `setuptools`. Если вы пропустите его, вы получите

```

...
warning: sdist: standard file not found: should have one of README,
README.rst, README.txt
...

```

Сохранение README в качестве стандартного способа включения некоторой информации о проекте-хорошая идея в любом случае. Вот что я положил в файл для `pytest-nice`:

## ch5/pytest-nice/README.rst

```

pytest-nice : A pytest plugin
=====

Делает вывод pytest немного дружелюбнее во время сбоев.

Особенности
-----
- Включает имя пользователя, выполняющего тесты в выводе pytest.
- Добавляет ``--nice`` опцию, которая:
- превращает ``F`` в ``O``
- с ``-v``, преобразует ``FAILURE`` в ``OPPORTUNITY for improvement``

Установка
-----

Учитывая, что наши плагины Pytest сохраняются в виде .tar.gz в

```



```
общей директория PATH, устанавливайте так:

::

$ pip install PATH/pytest-nice-0.1.0.tar.gz
$ pip install --no-index --find-links PATH pytest-nice

Использование
-----

::

$ pytest --nice
```

Есть много мнений о том, что должно быть в файле README. Это сильно обрезанная версия, но она работает.

## Тестирование Плагинов

Плагины — это код, который необходимо протестировать, как и любой другой код. Тем не менее, тестирование изменений в инструменте тестирования немного сложнее. Когда мы разработали код плагина в разделе «Написание собственных плагинов», на странице 98, мы про его вручную, используя образец тестового файла, запустив с ним `pytest` и проверив вывод, чтобы убедиться, что он был правильным. Мы можем сделать то же самое в автоматическом режиме с помощью плагина под названием `pytester`, который поставляется с `pytest`, но отключен по умолчанию.

В нашем тестовом каталоге для `pytest-nice` есть два файла: `conftest.py` и `test_nice.py`. Чтобы использовать `pytester`, нам нужно добавить только одну строку в `conftest.py`:

### ch5/pytest-nice/tests/conftest.py

```
"""pytester is needed for testing plugins."""
pytest_plugins = 'pytester'
```

Эта строка включает плагин `pytester`. Мы будем использовать фикстуру под названием `testdir`, которая становится доступным, когда `pytester` включен.

Часто тесты для плагинов принимают форму, которую мы описали вручную:

1. Сделайте пример тестового файла.
2. Запустите `pytest` с некоторыми параметрами или без них в каталоге, который содержит файл примера.
3. Проверьте выходные данные.
4. Возможный, для проверки кода результат-0 для всех проходов, 1 для некоторых сбоев.

Давайте рассмотрим один пример:

### ch5/pytest-nice/tests/test\_nice.py

```
def test_pass_fail(testdir):

    # создать временный тестовый модуль Pytest
    testdir.makepyfile("""
        def test_pass():
            assert 1 == 1

        def test_fail():
            assert 1 == 2
    """)

    # запустить pytest
```

```

result = testdir.runpytest()

# fnmatch_lines выполняет внутренний ассерт
result.stdout.fnmatch_lines([
    '*.F', # . для Pass, F для Fail
])

# убедитесь, что мы получили код выхода '1' для testsuite
assert result.ret == 1

```

Фикстура `testdir` автоматически создает временный каталог для размещения тестовых файлов. Она имеет метод `makepyfile()`, который позволяет поместить содержимое тестового файла. В этом случае мы создаем два теста: один, который *проходит* и другой, который *не прс*

Мы запускаем `pytest` для нового тестового файла с помощью `testdir.runpytest()`. Вы можете передать параметры, если хотите. Возвращ значение может быть рассмотрено в дальнейшем и имеет тип `RunResult`.

Обычно я смотрю на `stdout` и `ret`. Для проверки вывода, по аналогии с тем, как мы это делали вручную, используйте `fnmatch_lines`, перед список строк, которые мы хотим видеть в выводе, а затем убедившись, что `ret` равен 0 для проходящих сеансов и 1 для неудачных сеансо Строки, передаваемые в `fnmatch_lines`, могут включать символы подстановки. Мы можем использовать наш пример файла для тестов. В того, чтобы дублировать этот код, давайте напомним фикстуру:

#### ch5/pytest-nice/tests/test\_nice.py

```

@pytest.fixture()
def sample_test(testdir):
    testdir.makepyfile("""
        def test_pass():
            assert 1 == 1

        def test_fail():
            assert 1 == 2
    """)
    return testdir

```

Теперь, для остальных тестов, мы можем использовать `sample_test` в качестве каталога, который уже содержит наш пример тестового фай Ниже приведены тесты для других вариантов:

#### ch5/pytest-nice/tests/test\_nice.py

```

def test_with_nice(sample_test):
    result = sample_test.runpytest('--nice')
    result.stdout.fnmatch_lines(['*.O', ]) # . for Pass, O for Fail
    assert result.ret == 1

def test_with_nice_verbose(sample_test):
    result = sample_test.runpytest('-v', '--nice')
    result.stdout.fnmatch_lines([
        '*:test_fail OPPORTUNITY for improvement',
    ])
    assert result.ret == 1

def test_not_nice_verbose(sample_test):
    result = sample_test.runpytest('-v')
    result.stdout.fnmatch_lines(['*:test_fail FAILED'])
    assert result.ret == 1

```

Осталось написать еще пару тестов. Давайте убедимся, что наше благодарственное сообщение находится в заголовке:

#### ch5/pytest-nice/tests/test\_nice.py

```
def test_header(sample_test):
    result = sample_test.runpytest('--nice')
    result.stdout.fnmatch_lines(['Thanks for running the tests.'])

def test_header_not_nice(sample_test):
    result = sample_test.runpytest()
    thanks_message = 'Thanks for running the tests.'
    assert thanks_message not in result.stdout.str()
```

Это могло бы быть частью других тестов, но мне нравится проводить его в отдельном тесте, чтобы один тест проверял одну задачу.

Давайте проверим текст справки:

**ch5/pytest-nice/tests/test\_nice.py**

```
def test_help_message(testdir):
    result = testdir.runpytest('--help')

    # fnmatch_lines делает внутренний ассерт
    result.stdout.fnmatch_lines([
        'nice:',
        '*--nice*nice: turn FAILED into OPPORTUNITY for improvement',
    ])
```

Я думаю, что это достаточно хорошая проверка, того, что наш плагин работает.

Для запуска тестов давайте начнем с нашего каталога `pytest-nice` и убедимся, что наш плагин установлен. Мы делаем это либо установкой файла `.zip.gz`, либо установкой текущего каталога в редактируемом режиме:

```
$ cd /path/to/code/ch5/pytest-nice/
$ pip install .

Processing /path/to/code/ch5/pytest-nice
Requirement already satisfied: pytest in
/path/to/venv/lib/python3.6/site-packages (from pytest-nice==0.1.0)
Requirement already satisfied: py>=1.4.33 in
/path/to/venv/lib/python3.6/site-packages (from pytest->pytest-nice==0.1.0)
Requirement already satisfied: setuptools in
/path/to/venv/lib/python3.6/site-packages (from pytest->pytest-nice==0.1.0)
Building wheels for collected packages: pytest-nice
Running setup.py bdist_wheel for pytest-nice ... done
...
Successfully built pytest-nice
Installing collected packages: pytest-nice
Successfully installed pytest-nice-0.1.0
```

Теперь, когда он установлен, давайте запустим тесты:

```
$ pytest -v
===== test session starts =====
plugins: nice-0.1.0
collected 7 items
tests/test_nice.py::test_pass_fail PASSED
tests/test_nice.py::test_with_nice PASSED
tests/test_nice.py::test_with_nice_verbose PASSED
tests/test_nice.py::test_not_nice_verbose PASSED
tests/test_nice.py::test_header PASSED
tests/test_nice.py::test_header_not_nice PASSED
tests/test_nice.py::test_help_message PASSED
===== 7 passed in 0.34 seconds =====
```

**Прим. переводчика:** Если вы потерпели неудачу, то вы не одиноки. У меня тесты не прошли сразу.

```
platform win32 -- Python 3.6.5, pytest-3.9.3, py-1.7.0, pluggy-0.8.0 -- c:\venv36\scripts\python.exe

collected 7 items

tests/test_nice.py::test_pass_fail FAILED [ 14%]
tests/test_nice.py::test_with_nice OPPORTUNITY for improvement [ 28%]
tests/test_nice.py::test_with_nice_verbose OPPORTUNITY for improvement [ 42%]
tests/test_nice.py::test_not_nice_verbose FAILED [ 57%]
tests/test_nice.py::test_header PASSED [ 71%]
tests/test_nice.py::test_header_not_nice PASSED [ 85%]
tests/test_nice.py::test_help_message PASSED [100%]

===== FAILURES =====
_____ test_pass_fail _____
```

Я исправил шаблон поиска

```
result.stdout.fnmatch_lines([
    '*.F', # . for Pass, F for Fail
])
```

на

```
result.stdout.fnmatch_lines([
    '*.F*', # . for Pass, F for Fail
])
```

Добавил символ \* после F

По аналогии я внес исправления в test\_with\_nice, test\_with\_nice\_verbose, test\_not\_nice\_verbose

Видимо причина в версии pytest.

Я получаю вывод с процентом вида

```
'test_with_nice.py .O [100%]'
```

Здесь после . идут пробелы и проценты в квадратных скобках

Кроме того, я получил сообщения

RemovedInPytest4Warning: usage of Session.Class is deprecated, please use pytest.Class instead

В остальном всё нормально!

```
(venv36) c:\_BOOKS_\pytest_si\bopytest-code\code\ch5\pytest-nice>pytest -v
===== test session starts =====
platform win32 -- Python 3.6.5, pytest-3.9.3, py-1.7.0, pluggy-0.8.0 -- c:\venv36\scripts\python.exe
cachedir: .pytest_cache
rootdir: c:\_BOOKS_\pytest_si\bopytest-code\code\ch5\pytest-nice, inifile:
plugins: nice-0.1.0
collected 7 items

tests/test_nice.py::test_pass_fail PASSED [ 14%]
tests/test_nice.py::test_with_nice PASSED [ 28%]
tests/test_nice.py::test_with_nice_verbose PASSED [ 42%]
tests/test_nice.py::test_not_nice_verbose PASSED [ 57%]
tests/test_nice.py::test_header PASSED [ 71%]
tests/test_nice.py::test_header_not_nice PASSED [ 85%]
tests/test_nice.py::test_help_message PASSED [100%]

===== warnings summary =====
tests/test_nice.py::test_pass_fail
  c:\venv36\lib\site-packages\pytest\compat.py:332: RemovedInPytest4Warning: usage of Session.Class is deprecated, please use pytest.Class instead
```

```
e use pytest.Class instead
    return getattr(object, name, default)
```

Ура! Все тесты пройдены. Мы можем удалить его (pytest-nice), как и любой другой пакет Python или pytest-плагин:

```
$ pip uninstall pytest-nice

Uninstalling pytest-nice-0.1.0:
  Would remove:
    \path\to\venv\lib\site-packages\pytest_nice-0.1.0.dist-info\*
  ...
Proceed (y/n)? y
  Successfully uninstalled pytest-nice-0.1.0
```

Отличный способ узнать больше о тестировании плагинов — посмотреть на тесты, содержащиеся в других плагинах pytest, доступных через

## Создание дистрибутива

Верьте или нет но, мы почти закончили с нашим плагином. Теперь, из командной строки мы можем использовать этот файл setup.py для создания дистрибутива:

```
$ cd /path/to/code/ch5/pytest-nice
$ python setup.py sdist
running sdist
running egg_info
creating pytest_nice.egg-info
...
running check
creating pytest-nice-0.1.0
...
creating dist
Creating tar archive
...

$ ls dist

pytest-nice-0.1.0.tar.gz
```

(Обратите внимание, что sdist означает *source distribution* — “распространение исходного кода.”)

В pytest-nice каталог dist содержит новый файл с именем pytest-nice-0.1.0.tar.gz.

Этот файл теперь может быть использован в любом месте, чтобы установить наш плагин, даже на месте:

```
$ pip install dist/pytest-nice-0.1.0.tar.gz
Processing ./dist/pytest-nice-0.1.0.tar.gz
...
Installing collected packages: pytest-nice
Successfully installed pytest-nice-0.1.0
```

Теперь вы можете поместить свои файлы .tar.gz в любое место, где сможете их использовать и делиться ими.

## Распространение плагинов через общий каталог

pip уже поддерживает установку пакетов из общих каталогов, поэтому все, что нам нужно сделать, чтобы распространить наш плагин через общий каталог, это выбрать место, которое мы можем запомнить, и поместить туда файлы .tar.gz для наших плагинов. Допустим, мы поместим pytest-nice-0.1.0.tar.gz в каталог с именем myplugins.

Чтобы установить pytest-nice из myplugins:

```
$ pip install --no-index --find-links myplugins pytest-nice
```

`--no-index` указывает `pip` не выходить на PyPI, чтобы искать то, что вы хотите установить.

The `--find-links myplugins` tells PyPI to look in myplugins for packages to install. And of course, `pytest-nice` is what we want to install.

`--find-links myplugins` указывает PyPI найти в myplugins пакеты для установки. И конечно, `pytest-nice` — это то, что мы хотим установи

Если вы исправили какие то ошибки и в myplugins есть более новые версии, вы можете обновить их, добавив `--upgrade`:

```
$ pip install --upgrade --no-index --find-links myplugins pytest-nice
```

Это аналогично любому другому использованию `pip`, но с добавлением `--no-index --find-links myplugins`.

## Распространение плагинов через PyPI

Если вы хотите поделиться своим плагином с миром, есть еще пару шагов, которые мы должны сделать. На самом деле, есть еще несколько шагов. Однако, поскольку эта книга не посвящена работе с открытым исходным кодом, я рекомендую ознакомиться с подробными инструкциями содержащимися в руководстве пользователя Python Packaging.

Когда вы добавляете плагин `pytest`, есть отличное место для начала — использование `cookiecutter-pytest-plugin`:

```
$ pip install cookiecutter
$ cookiecutter https://github.com/pytest-dev/cookiecutter-pytest-plugin
```

Этот проект сначала задаст вам несколько вопросов о вашем плагине. Затем он создает каталог для вас, чтобы исследовать и заполнить в код. Углубление в принципы его работы выходит за рамки этой книги; однако, пожалуйста, имейте это в виду проект. Он поддерживается основными разработчиками `pytest`, и они будут следить за тем, чтобы этот проект оставался в актуальном состоянии.

## Упражнения

В `ch4/cache/test_slower.py` есть `autouse` fixture, называемая `check_duration()`. Вы также использовали её в упражнениях Главы 4. Теперь давайте сделаем плагин из неё.

1. Создайте каталог с именем `pytest-slower`, в котором будет храниться код для нового плагина, аналогично каталогу, описанному в разделе «Создание устанавливаемого плагина» на стр. 102.
2. Заполните все файлы каталога, чтобы сделать `pytest-slower` плагином, который можно установить.
3. Напишите некоторый тестовый код для плагина.
4. Взгляните на Python Package Index и поищите «pytest-». Найдите плагин `pytest`, который выглядит интересным для вас.
5. Установите выбранный вами плагин и попробуйте его на тестах `Tasks`.

## Что дальше

Вы до сих пор много раз использовали `conftest.py` в этой книге. Существуют также файлы конфигурации, которые влияют на выполнение например `pytest.ini`. В следующей главе вы ознакомитесь с различными конфигурационными файлами и узнаете, что можно сделать, чтобы облегчить себе жизнь при тестировании.



Теги: `pytest`

 +9



 13

 390

 Комментировать

**29,0**

Карма

**148,9**

Рейтинг

**90**

Подписчики

Александр Драгункин @AleksandrDr

Пользователь

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

- 4 сентября 2017 в 15:21

Тестируем асинхронный код с помощью PyTest (перевод)

+11

7,6k

76

2
- 29 октября 2015 в 12:15

PyTest

+20

113k

270

9
- 10 ноября 2014 в 16:47

Как в Яндексе используют PyTest и другие фреймворки для функционального тестирования

+58

70,2k

354

10

Комментарии 0

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки

Неделя

Месяц

- Бунт на Пикабу. Пользователи массово уходят на Реддит

+144

91,5k

71

290
- Смерть курьера «Яндекс.Еды» запустила волну жалоб на условия труда в компании

+57

57,1k

12

352
- Как я хакера ловил

+106

17,5k

81

49
- Как Мегафон спалился на мобильных подписках

+500

89,2k

174

462
- Межпозвоночная грыжа? Работай над ней

+37

20,6k

163

51

[Войти](#)

[Регистрация](#)

[Публикации](#)

[Новости](#)

[Хабы](#)

[Компании](#)

[Пользователи](#)

[Песочница](#)

[Правила](#)

[Помощь](#)

[Документация](#)

[Соглашение](#)

[Конфиденциальность](#)

[Реклама](#)

[Тарифы](#)

[Контент](#)

[Семинары](#)



© 2006 – 2019 «ТМ»



[Настройка языка](#)

[О сайте](#)

[Служба поддержки](#)

[Мобильная версия](#)

