## Запуск тестов

## Запуск из разных каталогов

Если у вас есть несколько проектов содержащих Coeception тесты, вы можете использовать один codecept.phar файл для их запуска. Чтобы запустить Codeception из другой директории, вы можете добавить опцию -c к любой Codeception команде кроме bootstrap.

php codecept.phar run -c ~/projects/ecommerce/

php codecept.phar run -c ~/projects/drupal/

php codecept.phar generate:cept acceptance CreateArticle -c ~/projects/drupal/

Для создания проекта в директории отличной от той, где вы находитесь, просто добавьте ее путь как параметр.

php codecept.phar bootstrap ~/projects/drupal/

В сущности опция -c позволяет указать не только путь, но и файл конфигурации. Таким образом у вас может быть несколько различных файлов codeception.yml в одном тестовом наборе. Вы можете использовать их для указания разных переменных окружения и настроек. Просто передайте имя конфигурационного файла в параметре -c для того, чтобы выполнить тесты с настройками указанными в данном конфигурационном файле.

## Группы

Есть несколько путей позволяющих выполнить группу тестов. Вы можете выполнить тесты в определенной директории:

php codecept.phar run tests/acceptance/admin

Или выполнить одну (или несколько) групп тестов:

php codecept.phar run -g admin -g editor

В данном случае, будут выполнены все тесты принадлежащие группам admin или editor. Концепция групп взята из PHPUnit и его классических тестов которые ведут себя подобным образом. Чтобы добавить Cept тест к группе, используйте переменную $scenario:

group('admin');

$scenario->group('editor');

// or

$scenario->group(array('admin', 'editor'))

// or

$scenario->groups(array('admin', 'editor'))

$I = new WebGuy($scenario);

$I->wantToTest('admin area');

?>

Для классических тестов и Cests тестов, чтобы добавить тест к группе можно использовать аннотацию @group.

<?php

/\*\*

\* @group admin

\*/

public function testAdminUser()

{

$this->assertEquals('admin', User::find(1)->role);

}

?>

Та же аннотация может быть использована в Cest классах.

## Интерактивная консоль

Интерактивная консоль добавлена для опробования команд Codeception перед тем, как они будут выполнены внутри тестов.  
Данная возможность была анонсирована начиная с версии 1.6.0.

Вы можете запустить консоль с помощью команды

php codecept.phar console suitename

Теперь вы можете выполнять любые команды соответствующего Guy класса и сразу же видеть результаты их выполнения. Это особенно полезно в случае, работы с модулями Selenium. Запуск Selenium и браузера для тестирования обычно занимает довольно продолжительное время. Однако при использовании консоли вы можете попробовать разнообразные селекторы и команды, после чего вы сможете быть уверенны в том, что написанный вами тест выполнится.

Полезный совет: покажите начальству, как резво вы манипулируете веб страницами с помощью консоли и Selenium. Так вы сможете убедить его в том, что довольно просто автоматизировать подобные вещи и начать использовать приемочное тестирование в проекте.

Для простого запуска тестов нужно перейти в консоли в корень проекта или в каталог куда были установлены тесты (если установка была не в корень) и выполнить:

**codecept run**

будут выполнены все тесты (модульные, функциональные и приемочные).

Для запуска тестирования из любого каталога (без перехода в корень проекта) – дополнительно нужно указать путь к проекту, например:

**codecept run -c "E:\OpenServer 5.2.5\domains\test.loc"**

Если передать один аргумент - можно запустить тесты только указанного типа. например для приемочных тестов

**$ codecept run acceptance**

Можно запустить тестирование одного конкретного класса теста, например:

**codecept run acceptance HomeCept**

Можно запустить тесты из определенной группы

**Codecept run –group future**

Для этого укажите перед тестом какой группе он принадлежит

*/\*\*  
 \** ***@group*** *future  
 \*/***public function** testSomeFuture()  
{  
 $this->markTestIncomplete();  
}

Чтобы запустить только один тест - передайте второй аргумент. Указывайте локальный путь, начиная от директории с набором тестов.

**codecept run acceptance SigninCept.php**

**codecept run -c app/tests/codeception/ acceptance WelcomeCept.php**

#### Опции run

--config (-c) можно указать специальный конфигурационный файл для этого запуска.

--report выводить результат в виде отчета в компактном стиле

--html Generate html with results

--xml Generate JUnit XML Log

--tap Generate Tap Log

--json Generate Json Log

--colors включить цвета (если поддерживает терминал)

--no-colors Force no colors in output (useful to override config file)

--silent выводить только имя и финальный результат

--steps - все выполняемые действия будут выводиться в консоль

-- debug (-d) - будет выводиться дополнительная отладочная информация.

--coverage собирать статистику о покрытии тестами

--no-exit не отдавать код завершения. Это может быть полезно при использовании Codeception с CI серверами, например с Bamboo.

--group (-g) Groups of tests to be executed (multiple values allowed)

--skip (-s) Skip selected suites (multiple values allowed)

--skip-group Skip selected groups (multiple values allowed)

--env Run tests in selected environments. (multiple values allowed)

--verbose (-v|vv|vvv) Уровень отладки

Следующие опции позволяют сохранить вывод результатов в наиболее подходящих форматах:

html - генерировать HTML-файл с результатами. Файл будет сохранен как 'report.html' в директории tests/\_log.

xml - сохранить вывод в формате JUnit для CI Серверов. Файл будет сохранен как 'report.xml' в директории tests/\_log.

tap - сохранить вывод в формате TAP. Файл будет сохранен как 'report.tap.log' в директории tests/\_log.

json - сохранить вывод в формате JSON. Файл будет сохранен как 'report.json' в директории tests/\_log.

Пример:

**codecept run --steps --xml --html**

эта команда запустит все тесты из всех тестовых наборов, все шаги будут выводиться на консоль, результаты тестов буду сохранены в HTML файл (в папку **tests/\_output**).

Дальше пробуем запустить сначала модульные и интеграционные из папки unit:

codecept run unit

Успешный результат выглядит так:

Codeception PHP Testing Framework v2.0.16

Powered by PHPUnit 4.7.7 by Sebastian Bergmann and contributors.

Unit Tests (26) -------------------------------------------------------------------------------

**Test validate empty new password** (UserTest::testValidateEmptyNewPassword) Ok

**Test validate wrong new password** (UserTest::testValidateWrongNewPassword) Ok

...

**Test correct signup** (SignupFormTest::testCorrectSignup) Ok

**Test not correct signup** (SignupFormTest::testNotCorrectSignup) Ok

-----------------------------------------------------------------------------------------------

Time: 7.37 seconds, Memory: 39.75Mb

OK (27 tests, 74 assertions)

И запустим функциональные:

codecept run functional

С виду здесь нет никакой разницы, но Codeception вместо создания объектов и вызова методов будет эмулировать запуск приложения, расставляя $\_GET, $\_POST и $\_SERVER так, как будто мы открываем страницы и отправляем POST-запросы:

Codeception PHP Testing Framework v2.0.16

Powered by PHPUnit 4.7.7 by Sebastian Bergmann and contributors.

Functional Tests (17) -------------------------------------------------------------------------

**Ensure that admin home page works** (HomeCest::testAccess) Ok

**Ensure that admin home page works** (HomeCest::testHomePage) Ok

...

**Ensure that password change works** (PasswordChangeCest::testPasswordChange) Ok

**Ensure that profile update form works** (ProfileUpdateCest::testProfileUpdate) Ok

-----------------------------------------------------------------------------------------------

Time: 15.36 seconds, Memory: 95.25Mb

OK (17 tests, 129 assertions)

Для запуска реальных приёмочных тестов нужно поднять тестовый сайт по адресу http://localhost:8080/index-test.php. Для этого по подсказке той же инструкции в tests/README.md откроем ещё одну консоль и в ней запустим PHP-сервер из папки web:

cd web

php -S localhost:8080

Он нам сообщит о своей готовности:

PHP Development Server started at Tue Feb 2 21:14:00 2016

Listening on http://localhost:8080

Press Ctrl-C to quit.

Откроем ещё одно окно консоли и запустим Selenium:

selenium-server-standalone

Увидим его приветствие:

21:16:51.796 INFO - Launching a standalone Selenium Server

21:16:51.977 INFO - Selenium Server is up and running

И, запустив в первоначальной консоли приёмочные из папки tests:

codecept run acceptance

мы должны увидеть на экране снова и снова открывающийся браузер, в котором будут сами открываться страницы, заполняться поля форм и нажиматься кнопки. При этом в консоли будет выводиться такой же как у функциональных результат:

Codeception PHP Testing Framework v2.0.16

Powered by PHPUnit 4.7.7 by Sebastian Bergmann and contributors.

Acceptance Tests (12) -------------------------------------------------------------------------

**Ensure that access control works** (HomeCest::testAccess) Ok

**Ensure that admin home page works** (HomeCest::testHomePage) Ok

...

**Ensure that profile home page works** (HomeCest::testHomePage) Ok

**Ensure that login works** (PasswordChangeCest::testPasswordChange) Ok

-----------------------------------------------------------------------------------------------

Time: 1.72 minutes, Memory: 19.00Mb

OK (12 tests, 80 assertions)

Если какой-либо тест завершится с ошибкой можно будет посмотреть отчёт в папке tests/codeception/\_output.

Если вы создадите в каталоге проекта файл **.scrutinizer.yml и поместите туда вот такое содержание**

|  |
| --- |
| imports: |
|  | - php |
|  |  |
|  | tools: |
|  | external\_code\_coverage: |
|  | timeout: 2100 # Timeout in seconds. |
|  | # disable copy paste detector and similarity analyzer as they have no real value |
|  | # and a huge bunch of false-positives |
|  | php\_sim: false |
|  | php\_cpd: false |

**то ваш проект на GitHub будет тестировать сервис scrutinizer-ci.com**

## Cest Классы

В случае, если вы хотите получить class-like структуру ваших Cept тестов, вместо использования простого PHP вы можете использовать Cest формат. Это очень просто и полностью совместимо с Cept сценариями. Если вы считаете, что ваши тесты стали достаточно длинными и вы хотите разбить их на части, вы можете поместить их внутрь класса.

Вы можете запустить Cest файл с помощью команды:

php codecept.phar generate:cest suitename CestName

Это сгенерирует файл похожий на этот:

**Каждый публичный метод Cest класса (кроме, начинающихся с \_) будет выполнен как тест** куда первым параметром будет передан Guy класс и вторым параметром будет передана переменная $scenario.

В методах \_before и \_after можно указывать разнообразные настройки, которые будут выполняться до и после тестов данного класса. Это делает Cest тесты более гибкими, чем Cepts тесты которые основываются только на похожих методах в Helper классах.

Как вы могли видеть выше мы передаем класс Guy в метод tryToTest. Это позволяет писать сценарии тестирования точно так же, как мы делали это ранее.

<?php

class BasicCest

{

// test

public function checkLogin(\WebGuy $I) {

$I->wantTo('log in to site');

$I->amOnPage('/');

$I->click('Login');

$I->fillField('username', 'jon');

$I->fillField('password','coltrane');

$I->click('Enter');

$I->see('Hello, Jon');

$I->seeInCurrentUrl('/account');

}

}

?>

Стоит отметить, что при использовании Cest файлов появляются некоторые ограничения. Вы не сможете работать с \_bootstrap.php так как вы делали это в Cept тестах. Иногда удобно хранить некоторые переменные, которые должны быть переданы тесту в bootstrap файлах.  
В файлах Cest вам придется включать все внешние переменные вручную с помощью глобальных или статических переменных.

В качестве обходного пути можно использовать классы фикстур [Fixtures](https://github.com/Codeception/Codeception/blob/master/src/Codeception/Util/Fixtures.php) которые являются по сути являются глобальным хранилищем переменных. Можно передать любые данные из \_bootstrap.php или ил любого другого места с помощью вызова Fixtures::add(). Вероятно так же вы можете использовать в классах Cest методы \_before и \_after для загрузки фикстур в начале теста, и их удаления, после его выполнения, что так же довольно удобно.

Как вы могли видеть, классы Cest не имеют родителя на подобие \Codeception\TestCase\Test или PHPUnit\_Framework\_TestCase. Это сделано нарочно. Это позволяет вам расширять данные классы путем создания общего класса родителя.В данном классе вы можете описать общие поведения и методы, которые в дальнейшем могут быть использованы в унаследованных классах. При этом не забудьте установить для таких методов модификатор protected иначе они будут выполнены как тесты.

В дополнение, вы можете определить метод \_failed в классах Cest который будет вызван в случае если в тесте произошла ошибка errorили он 'провалился'.

### Аннотации

добавлено начиная с версии 1.7.0

Вы можете управлять Cest файлами с помощью аннотаций. Вы можете использовать аннотацию @guy для передачи Guy класса отличного от того, что установлен в конфигурации. Это довольно удобно, если вы хотите передать StepObject здесь (смотрите ниже).

Аннотация может быть добавлена к методу в DocBlock блоке.

Вы можете контролировать порядок выполнения используя аннотации @before и @after. Вы можете переместить некоторые действия в protected (не тестируемые) методы и вызвать их до или после тестового метода, добавляя их в аннотации.

amOnPage('/login');

$I->fillField('Username', 'miles');

$I->fillField('Password', 'davis');

$I->click('Login');

}

/\*\*

\* @before login

\*/

function banUser(WebGuy $I)

{

$I->amOnPage('/users/charlie-parker');

$I->see('Ban', '.button');

$I->click('Ban');

}

}

?>

Вы так же можете использовать @before и @after для подключаемых функций. Однако нельзя иметь несколько аннотаций в одном методе.

## Рефакторинг

Во время роста базы тестов может потребоваться их рефакторинг, возможно понадобится выделить некоторые общие поведения и методы. Классический пример - действие login которое возможно будет вызываться практически в каждом тесте из тестового набора. Мудрое решение - написать данное действие однажды и использовать его во всех тестах, где это необходимо.

Вполне очевидно, что вы можете использовать собственные классы для определения похожих методов.

amOnPage('/login');

$I->fillField('username', 'jon');

$I->fillField('password','coltrane');

$I->click('Enter');

}

}

?>

Этот файл затем может быть включен в файл \_bootstrap.php

<?php

// bootstrap

require\_once '/path/to/test/commons/TestCommons.php';

?>

и затем может быть использован в ваших сценариях:

<?php

$I = new WebGuy($scenario);

TestCommons::logMeIn($I);

?>

Теперь идея должна быть вам понятна. Codeception не обеспечивает единой стратегии при управлении тестами. Однако он достаточно гибок и позволяет создавать любые необходимые классы которые могут понадобиться вам во время тестирования. Используя похожий подход вы можете реализовать паттерны PageObject и StepObject.

Существует два способа написания тестов в Codeception, которым соответствуют два типа наименования файлов: Cept и Cest. Cept строится по сценарию (функциональное программирование), а Cest на базе объектов (ООП). Кроме того, Codeception может выполнять PHPUnit-тесты для модульного тестирования т.к. наследуется от данной библиотеки.  
В качестве примера создам простой приемочный тест по типу **Cept** для проверки главной страницы сайта.   
  
Файл теста можно сгенерировать автоматически. Создадим файл **HomeCept.php** командой:

codecept generate:cept acceptance Home

Файл появится в папке **tests/acceptance** и сразу будет содержать строки:

<?php

$I = new AcceptanceTester($scenario);

$I->wantTo('perform actions and see result');

При желании, создать файл теста можно и вручную, главное указать в названии файла окончание «Cept».  
  
Дополним тест на проверку доступа к главной странице которая также должна вернуть указанный текст (фразу).

<?php

$I = new AcceptanceTester($scenario);

$I->wantTo('Проверить работу главной страницы.');

$I->amOnPage('/');

$I->seeResponseCodeIs(\Codeception\Util\HttpCode::OK);

$I->see('Главная страница'); // Фраза или ее часть с главной страницы

В обязательном методе **wantTo()** указывается действие которое осуществляет данный тестовый метод.   
В методе **amOnPage()** указывается на какой странице начинается тестирование (далее возможен переход по ссылкам и тд.).   
Метод **seeResponseCodeIs()** сравнивает код который возвращает сервер на запрос страницы, с указанным в качестве аргумента. В данном случае, я использовал константу "*OK*" из класса **HttpCode**, которая имеет значение *200*, но можно указывать аргумент и непосредственно в цифровом варианте. Это тестовый метод и он возвращает *true* если код равен указанному.  
Ну и часто употребляемый метод **see()** возвращает *true* если на странице присутствует указанная фраза.  
  
Так же можно создавать тестовый файл сразу в определенной папке. Это удобно, т.к. путь к тесту должен повторять файловую структуру проекта. Например создадим модульный тест, файл **SaveTest.php** в папке *articles* (папка создастся автоматически):

codecept generate:test unit articles/Save

Кроме того, тестам, которые созданы на базе классов (cest и test), автоматически будет присвоено пространство имен, в данном случае:  
*namespace articles;*  
  
Генераторы

Codeception имеет множество полезных команд для генерации всего и вся:

* generate:cept suite filename - Генерировать Cept сценарий.
* generate:cest suite filename - Генерировать Cest тест.
* generate:test suite filename - Генерировать PHPUnit Test c Codeception хуками.
* generate:phpunit suite filename - Генерировать классический PHPUnit Test.
* generate:suite suite guy - Генерировать новый набор тестов с указанным Guy-классом.
* generate:scenarios suite - Генерировать текстовый файл, содержащий сценарии из тестов.
* generate:group Generates Group subscriber
* generate:helper Generates new helper
* generate:pageobject Generates empty PageObject class
* generate:phpunit Generates empty PHPUnit test without Codeception additions
* generate:scenarios Generates text representation for all scenarios
* generate:stepobject Generates empty StepObject class
* generate:suite Generates new test suite
* generate:test Generates empty unit test file in suite