## Запуск тестов

Параметры запуска

### run

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"**

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

**codecept run acceptance SigninCept.php**

[-c|--config CONFIG] можно указать специальный конфигурационный файл для этого запуска

### [--report]

выводить результат в виде отчета в компактном стиле

### [--html [HTML]]

### [--xml [XML]]

### [--tap [TAP]]

### [--json [JSON]]

### [--colors]

### [--no-colors]

### [--silent]

### [--steps]

### [-d|--debug]

### [--coverage [COVERAGE]]

### [--coverage-html [COVERAGE-HTML]]

### [--coverage-xml [COVERAGE-XML]]

### [--coverage-text [COVERAGE-TEXT]]

### [--no-exit]

### [-g|--group GROUP]

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();  
}

### [-s|--skip SKIP]

### [-sg|--skip-group SKIP-GROUP]

### [--env ENV]

### [-f|--fail-fast]

### [--no-rebuild]

### [--]

### [<suite>]

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

**$ codecept run acceptance**

### [<test>]

Можно запустить тестирование одного конкретного класса теста, например:  
**codecept run acceptance HomeCept**

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

#### Опции run

--config (-c) .

--report

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

--xml Generate JUnit XML 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.

--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

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.

Генераторы

Codeception имеет множество полезных команд для генерации всего и вся:

Codecept generate:cept suite filename - Генерировать Cept сценарий.

Codecept generate:cest suite filename - Генерировать Cest тест.

Codecept generate:test suite filename - Генерировать PHPUnit Test c Codeception хуками.

Codecept generate:phpunit suite filename - Генерировать классический PHPUnit Test.

Codecept generate:suite suite guy - Генерировать новый набор тестов с указанным Guy-классом.

Codecept generate:scenarios suite - Генерировать текстовый файл, содержащий сценарии из тестов.

Codecept generate:group Generates Group subscriber

Codecept generate:helper Generates new helper

Codecept generate:pageobject Generates empty PageObject class

Codecept generate:phpunit Generates empty PHPUnit test without Codeception additions

Codecept generate:scenarios Generates text representation for all scenarios

Codecept generate:stepobject Generates empty StepObject class

Codecept generate:suite Generates new test suite

Codecept generate:test Generates empty unit test file in suite

### Генерация cest сценария

Можно запустить команду

codecept generate:cest functional CestName

и получить в каталоге \tests\functional\CestNameCest.php

**<?php  
use** \FunctionalTester;  
  
**class** CestNameCest  
{  
 **public function** \_before(FunctionalTester $I)  
 {  
 }  
  
 **public function** \_after(FunctionalTester $I)  
 {  
 }  
  
 // tests  
 **public function** tryToTest(FunctionalTester $I)  
 {  
 }  
}

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'.

### Генерация cept сценария

Можно запустить команду

codecept generate:cept functional CestName

и получить в каталоге \tests\functional\CestNameCept.php

**<?php**$I = **new** FunctionalTester($scenario);  
$I->wantTo('perform actions and see result');

### Создание Unit теста с хуками Codeception

При запуске команды

codecept generate:test unit file22

получаем в каталоге \tests\unit\file22Test.php

**<?php  
  
  
class** file22Test **extends** \Codeception\TestCase\Test  
{  
 */\*\*  
 \** ***@var*** *\UnitTester  
 \*/* **protected** $tester;  
  
 **protected function** \_before()  
 {  
 }  
  
 **protected function** \_after()  
 {  
 }  
  
 // tests  
 **public function** testMe()  
 {  
  
 }  
  
}

### Генерировать новый набор тестов с указанным Guy-классом

При запуске команды

Codecept generate:suite newsuite newguy

1. Создается файл 'newsuite.suite.yml

**class\_name:** newguyTester  
**modules:  
 enabled:** [newguyHelper]

1. Создается каталог /tests/newsuite
2. Cоздается файл /tests/newsuite/\_bootstrap.php

**<?php**// Here you can initialize variables that will be available to your tests

1. Создается файл /tests/\_support/newguyHelper/php

**<?php  
namespace** Codeception\Module;  
  
// here you can define custom actions  
// all public methods declared in helper class will be available in $I  
  
**class** newguyHelper **extends** \Codeception\Module  
{  
  
}

### Создание пустого класса для страницы

При запуске команды

Codecept generate:pageobject myPages

В каталоге \_pages создается файл /\_pages/myPagesPage.php

**<?php  
  
class** myPagesPage  
{  
 // include url of current page  
 **public static** *$URL* = '';  
  
 */\*\*  
 \* Declare UI map for this page here. CSS or XPath allowed.  
 \* public static $usernameField = '#username';  
 \* public static $formSubmitButton = "#mainForm input[type=submit]";  
 \*/  
  
 /\*\*  
 \* Basic route example for your current URL  
 \* You can append any additional parameter to URL  
 \* and use it in tests like: EditPage::route('/123-post');  
 \*/* **public static function** route($param)  
 {  
 **return static**::*$URL*.$param;  
 }  
  
  
}

Такой файл удобно использовать при регулярном заполнении одних и тех же страниц данными

**<?php  
  
namespace** tests\codeception\\_pages;  
  
**use** yii\codeception\BasePage;  
  
*/\*\*  
 \* Represents login page  
 \** ***@property*** *\AcceptanceTester|\FunctionalTester $actor  
 \*/***class** LoginPage **extends** BasePage  
{  
 **public** $route = 'site/login';  
  
 */\*\*  
 \** ***@param*** *string $username  
 \** ***@param*** *string $password  
 \*/* **public function** login($username, $password)  
 {  
 $this->actor->fillField('input[name="LoginForm[username]"]', $username);  
 $this->actor->fillField('input[name="LoginForm[password]"]', $password);  
 $this->actor->click('login-button');  
 }  
}

В файле /acceptance/LoginCept.php пишем следующее

**<?php  
  
use** tests\codeception\\_pages\LoginPage;  
  
*/\** ***@var*** *$scenario Codeception\Scenario \*/*$I = **new** AcceptanceTester($scenario);  
$I->wantTo('ensure that login works');  
  
$loginPage = LoginPage::*openBy*($I);  
  
$I->see('Login', 'h1');  
  
$I->amGoingTo('try to login with empty credentials');  
$loginPage->login('', '');  
**if** (method\_exists($I, 'wait')) {  
 $I->wait(3); // only for selenium  
}  
$I->expectTo('see validations errors');  
$I->see('Username cannot be blank.');  
$I->see('Password cannot be blank.');  
  
$I->amGoingTo('try to login with wrong credentials');  
$loginPage->login('admin', 'wrong');  
**if** (method\_exists($I, 'wait')) {  
 $I->wait(3); // only for selenium  
}  
$I->expectTo('see validations errors');  
$I->see('Incorrect username or password.');  
  
$I->amGoingTo('try to login with correct credentials');  
$loginPage->login('admin', 'admin');  
**if** (method\_exists($I, 'wait')) {  
 $I->wait(3); // only for selenium  
}  
$I->expectTo('see user info');  
$I->see('Logout (admin)');

## Запуск из разных каталогов

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

codecept run -c ~/projects/ecommerce/

codecept run -c ~/projects/drupal/

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

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

codecept 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:

codecept run tests/acceptance/admin

Или выполнить одну (или несколько) групп тестов:

codecept 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.

### 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.

## Анализ покрытия кода тестами

Этот анализ позволяет посмотеть, какие участки кода отработали в ходе тестирования. В Ubuntu и Debian устанавливаем стандартно:

aptitude install php5-xdebug

В Windows нужно в php.ini добавить

[xdebug]

zend\_extension\_ts=C:/php/ext/php\_xdebug.dll

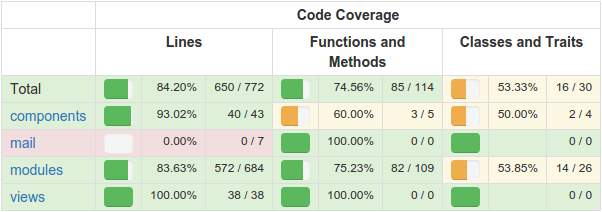
Но если в ваших тестах используется $this->specify(...), то будет постоянно вываливаться ошибка из-за ограничений уровня вложенности. Для решения проблемы нужно открыть php.ini или /etc/php/conf.d/xdebug.ini и дописать:

xdebug.max\_nesting\_level = 1000

С установкой всё. Попробуем запустить всё снова, но с анализом покрытия:

codecept run unit,functional --coverage-html

И после завершения посмотрим \_output/coverage/index.html:

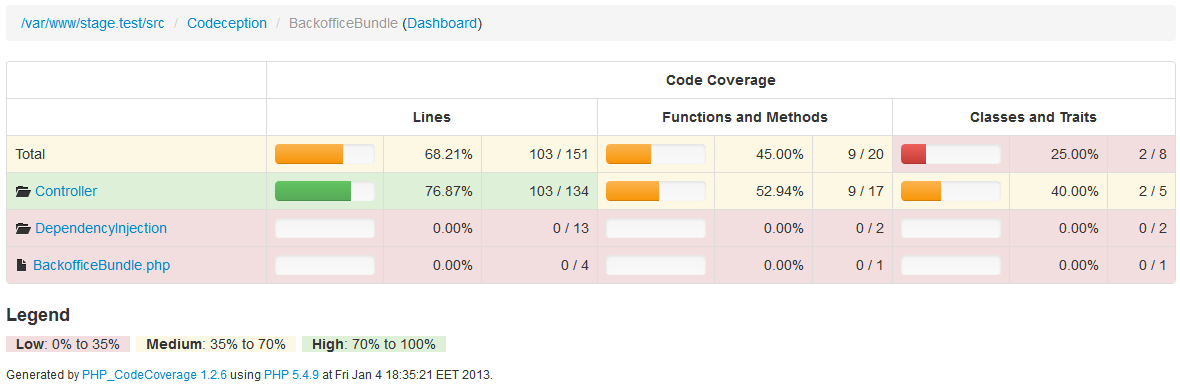


Это удобно тем, что можно посмотреть, какие фрагменты кода не выполнялись и дописать проверки для них. Теперь нам не страшно будет вносить правки в исходный код. В любой момент можно проверить правильность работы проекта.

## Покрытие кода тестами (Code Coverage)

В какой то момент вам захочется узнать, какие части приложения покрыты тестами, а какие нет.  
Именно для этого и используется [CodeCoverage](http://en.wikipedia.org/wiki/Code_coverage). Во время выполнения тестов для сбора данных о покрытии, вы получите статистику по всем классам, методам и строкам кода, которые затронуты вашими тестами.  
Разница между количеством строк кода и количеством затронутых при тестировании строк является главным критерием покрытия кода тестами. В идеальном мире покрытие кода должно составлять 100%, однако в реальности достаточно и 80%. Однако, даже 100% покрытие кода не защитит вас от ошибок и падения приложения.

**Codeception включает инструменты CodeCoverage начиная с версии 1.5. Для сбора информации о покрытии тестами необходимxdebug**.



Данные о покрытии тестами могут быть собраны вручную для локальных или удаленных тестов. Удаленные тесты могут выполняться на другом узле,  
или локально, однако с использованием веб сервера. Может показаться сложным получение покрытия кода для тестов Selenium или PhpBrowser. Однако Codeception  
поддерживает удаленное покрытие точно так же как и локальное.

### Конфигурация

Чтобы включить утилиту покрытия тестами, добавьте следущие строки в глобальный конфигурационный файл codeception.yml:

coverage:

enabled: true

выглядит неплохо. Но какие файлы должны попасть в отчет о покрытии? Вы можете фильтровать файлы используя black-листы и white-листы.

coverage:

enabled: true

whitelist:

include:

- app/\*

exclude:

- app/cache/\*

blacklist:

include:

- app/controllers/\*

exclude:

- app/cache/CacheProvider.php

Что такое black-листы и white-листы?

* Список **whitelist** это список файлов которые должны быть включены в отчет, даже если они не затронуты.
* Список **blacklist** это список файлов которые должны быть исключены из отчета, даже если они затронуты.

Передайте массив файлов/директорий в секции include/exclude. Путь заканчивающийся на '' относится к директории.Так же вы можете использовать маску '' в именах файлов, например app/models/\*Model.php для указания всех моделей.

Вот пример, если вам не нужны сложные фильтры:

coverage:

enabled: true

include:

- app/\*

exclude:

- app/cache/\*

Опции include и exclude здесь добавляют или удаляют файлы из white-листа.

Каждая из этих настроек может быть переопределена для каждого набора в сооветсвующих конфигурационых файлах.

## Локальное покрытие тестами

Базовый отчет о покрытии может быть получен для функциональных и модульных тестов.  
Если вы настроили конфигурацию как было показано выше, вы готовы к действиям.  
Все что вам нужно выполнить codeception с опцией --coverage.  
Для генерации xml отчета или красивого html отчета добавьте опции --xml и --html.

codecept run --coverage --xml --html

XML и HTML отчеты распологаются в директории \_logs. Лучший способ посмотреть отчет, открыть index.html находящийся в директорииtests/\_logs/coverage с помощью вашего браузера.  
XML отчеты используются в IDEs (такими как PHPStorm) или серверами Continuous Integration (Такими как Jenkins).

## Удаленное покрытие тестами

В случае, есои ваше приложение запущено с помощью веб-сервера (Apache, Nginx, PHP WebServer) у вас нет доступа к тестируемому коду,  
таким образом сбор данных о покрытии кода становится нетривиальной задачей. То же самое каается скриптов запущенных на другом узле.  
Для того чтобы получить доступ к коду, у вас должно быть установлено расширение xdebug с включенной опцией remote\_enable.  
Codeception так же требует наличия небольшого "шпиона" для взаимодействия с вашим приложением.  
Даже если ваше приложение работает автономно, даже не зная что оно было протестировано, данный файл должен быть подключен для того чтобы стало возможным собирать информацию о покрытии тестами.

Этот файл называется c3.php и он [доступен на GitHub](https://github.com/Codeception/c3).  
c3.php должен быть скачан и включен в ваше приложение в первой строке of it's from controller.  
Посылая специальные заголовки Codeception будет отдавать команды вашему приложению, позволяющие запускать сбор данных о покрытии кода и останавливать его.  
После выполнения набора тестов, отчет будет сохранен и Codeception считает его с вашего приложения.

Следуйте инструкциям по установке описанным в файле [readme](https://github.com/Codeception/c3).

После того как файл c3.php будет включен в ваше приложение, вы сможете начать сбор данных о покрытии кода тестами.  
В случае, если вы запускаете приложение локально, не нужно производить ни каких дополнитльных действий в конфигурации.  
Все данные будут собраны, после чего скомпонованы.  
Подумайте над этим: Codeception запускает удаленно утилиту сбора покрытия кода тестами, точно так же как и локально.

Никогда настройка удаленного формирования отчета о покрытии кода тестами не была такой простой. В любом другом фреймворке. Серьезно.

Однако, если вы запускаете тесты на удаленном сервере (или ваш веб-сервер не использует код из текущей директории), в конфигурационный файл необходимо добавить опцию remote.  
К примеру давайте включим удаленную утилиту сбора информации о покрытии кода тестами в набор acceptance.suite.yml

coverage:

enabled: true

remote: true

В данном случае результаты отчеты не будут слиты с локальными, если опция включена.  
Объединение отчетов возможно только в том случае, если файл запущеный удаленно и локально имеет один путь.  
Однако в случае запуска тестов на удаленно сервере нет уверенности, что это так.

**Настройка покрытия кода**

Для формирования HTML и XML отчётов включаем утилиту coverage в файле codeception.yml и указываем какие файлы должны попасть в отчет:

coverage:

enabled: true

whitelist:

include:

- app/\*

exclude:

- app/commands/\*

- app/config/\*

- app/migrations/\*

blacklist:

include:

- app/migrations/\*

exclude:

- app/services/Sendler.php

Где whitelist - это перечень файлов которые будут включены в отчет (даже если не затронуты), а blacklist - это перечень файлов которые не будут включены в отчет (даже если затронуты). Опции include и exclude здесь добавляют или удаляют файлы.

Запускаем тесты с формированием отчетов:

codecept run --coverage --html --xml

После успешного выполнения тестов мы видим что-то подобное:

- HTML report generated in file:///home/user/projects/projectName/tests/\_output/report.html

- XML report generated in file:///home/user/projects/projectName/tests/\_output/report.xml

Это адреса файлов с отчетами тестирования. Можем перейти и посмотреть на них. Но если мы хотим посмотреть отчет о покрытии тестами файлов проекта, тогда запускаем тесты так:

$ ./vendor/bin/codecept run --coverage-html

А файл с отчетом смотрим по адресу projectName/tests/\_output/coverage/index.html.

**Отладка тестов**

При отладке тестов часто нужно больше информации, чем просто значение о выполнении / не выполнении теста. Например, для функциональных тестов нужно увидеть получаемые заголовки. Для этого у Codeception есть параметр --debug, который покажет такую информацию:

codecept run --debug

Что касается модульных тестов то, например, нужно узнать что за объект получен, а функция var\_dump() игнорируется. Даже когда параметр --debug указан. В этом случае у Codeception есть своя функция отображения структурированной информации:

\Codeception\Util\Debug::debug('текст или массив');