Composer template Documentation

Thomas Farla

Apr 02, 2018

Contents:

Making development of composer libraries easy with this cloneable template which includes:

- continuous integration (travis-ci)
- code coverage (coveralls)
- static analysis (phpstan)
- mess detector (phpmd)
- testing framework (phpunit)
- php code sniffer which enforces the psr-2 standard (phpcs)
- composer configuration with psr-4 autoloading
- a changelog (keepachangelog)
- MIT license (not sure about what license you need? https://choosealicense.com/)
- documentation using sphinx
- badges from http://shields.io/

CHAPTER 1

Requirements

- php 7.1 or greater ([supported versions](http://php.net/supported-versions.php))
- python & pip to create documentation
- composer

CHAPTER 2

Installation

The following command will clone this template and place it in the *my-library* directory

```
composer create-project tfarla/composer-template my-library
```

Composer's [create-project](https://getcomposer.org/doc/03-cli.md#create-project) will also install all dependencies in this template:

```
Installing tfarla/composer-template (dev-master␣
↪d3249c7ffef021c39e33a4323be4d70899d4d28b)
- Installing tfarla/composer-template (dev-master master): Cloning master from cache
Created project in my-library
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Generating autoload files
ocramius/package-versions:  Generating version class...
ocramius/package-versions: ...done generating version class
```

Once the download is done you will get the choice to remove the *.git* directory which you should do if you want your own repository.

```
Do you want to remove the existing VCS (.git, .svn..) history? [Y,n]? y
```

CHAPTER 3

Usage

In order to use this template you'll need to change a few things:

- Change the content in this README file to reflect your library
- Change the content in the *docs/* directory to reflect your library
- Setup travis-ci (should be plug & play)
- Setup readthedocs (should be plug & play)
- Setup coveralls (should be plug & play)
- Pick a license https://choosealicense.com/ (MIT is good enough for most open source project, but you might want to look at the other options)

3.1 Mess detector

3.1.1 Purpose

A mess detector can be used to find find complex pieces of code and make them more simple. It's also a great tool to ensure a certain level of quality and make the library easier to maintain. This template uses phpmd which is a mess detector library for php that contains a set of rules which are used on your source code.

3.1.2 Getting started

Composer has been configured to run phpmd by running the command:

```
composer run mess-detector
> vendor/bin/phpmd ./src text cleancode,unusedcode,codesize,design,naming
./src/TFarla/ComposerTemplate/Example.php:11        Avoid unused private fields such␣
↪as '$a'.
```

(continues on next page)

(continued from previous page)

```
.//src/TFarla/ComposerTemplate/Example.php:11       Avoid variables with short names␣
↪like $a. Configured minimum length is 3.
Script vendor/bin/phpmd ./src text cleancode,unusedcode,codesize,design,naming␣
↪handling the mess-detector event returned with error code 2
```

The *mess-detector* script uses the default configuration for the following rules:

- cleancode
- unusedcode
- codesize
- design
- naming

It's also possible to create your own ruleset to have more control over the rules. This will allow you to create exceptions for specific violations when desired. https://phpmd.org/documentation/writing-a-phpmd-rule.html

3.2 Code sniffer

3.2.1 Purpose

A code sniffer can detect inconsistencies in the source code and enforces the psr-2 standard. This template uses https://github.com/squizlabs/PHP_CodeSniffer which comes with a handy tool to automatically fix any violations of the psr-2 standard.

3.2.2 Getting started

Composer has been configured to run the code sniffer with the psr-2 standard:

```
composer run code-sniffer

> ./vendor/bin/phpcs --standard=PSR2 src

FILE: /app/src/TFarla/ComposerTemplate/Example.php
----------------------------------------------------------------------
FOUND 1 ERROR AFFECTING 1 LINE
----------------------------------------------------------------------
2 | ERROR | [x] There must be one blank line after the namespace
  |       |     declaration
----------------------------------------------------------------------
PHPCBF CAN FIX THE 1 MARKED SNIFF VIOLATIONS AUTOMATICALLY
----------------------------------------------------------------------

Time: 353ms; Memory: 6Mb

Script ./vendor/bin/phpcs --standard=PSR2 src handling the code-sniffer event␣
↪returned with error code 2
```

There's a violation in our code. Composer has also been configured to execute the code sniffer fixer called PHPCBF. Execute the following command:

```
composer run code-sniffer fix

> ./vendor/bin/phpcbf --standard=PSR2 src

PHPCBF RESULT SUMMARY
----------------------------------------------------------------------
FILE                                                    FIXED  REMAINING
----------------------------------------------------------------------
/app/src/TFarla/ComposerTemplate/Example.php            1      0
----------------------------------------------------------------------
A TOTAL OF 1 ERROR WERE FIXED IN 1 FILE
----------------------------------------------------------------------

Time: 335ms; Memory: 6Mb
```

When we run the code sniffer again. It should not report any violations:

```
composer run code-sniffer

> ./vendor/bin/phpcs --standard=PSR2 src
```

The source code is now psr-2 compliant

3.3 Static analysis

3.3.1 Purpose

Static analysis is a tool which searches for bugs in th source code without executing the program. This template uses phpstan which is a fast static analysis tool for php.

3.3.2 Getting started

Composer has been configured to run the phpstan command for you. Executing the following command:

```
composer run static-analysis
```

```
composer run static-analysis
> vendor/bin/phpstan analyse src --level=0
0/1 []   0%
1/1 [] 100%

[OK] No errors
```

PHPstan found several errors on the following code:

```php
<?php

class Example
{
    public function __construct(\DateTimeImmmutable $dateTime)
    {
        $this->dateTime = $dateTime;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

```
composer run static-analysis
> vendor/bin/phpstan analyse src --level=0
 0/1 []   0%
 1/1 [] 100%

 ------ -------------------------------------------------------------------
  Line   src/TFarla/ComposerTemplate/Example.php
 ------ -------------------------------------------------------------------
  11     Parameter $dateTime of method
         TFarla\ComposerTemplate\Example::__construct() has invalid typehint
         type DateTimeImmmutable.
  13     Access to an undefined property
         TFarla\ComposerTemplate\Example::$dateTime.
 ------ -------------------------------------------------------------------

 [ERROR] Found 2 errors

 Script vendor/bin/phpstan analyse src --level=0 handling the static-analysis event␣
↪returned with error code 1
```

3.4 Documentation

3.4.1 Purpose

All great projects need some form of documentation which communicates the intended purpose and implementation details of the project. This template uses the sphinx project to build the documentation and the free online service https://readthedocs.org/ to host the documentation.

3.4.2 Getting started

All documentation can be found in the docs directory in the root of this template. It contains a directory structure which is similar to the table of contents located in the sidebar.

```
.
├── Makefile
├── build
├── make.bat
└── source
    ├── _static
    ├── _templates
    ├── code-sniffer
    │   └── README.rst
    ├── conf.py
    ├── continuous-integration
    │   └── README.rst
    ├── documentation
    │   └── README.rst
    ├── index.rst
    ├── mess-detector
```

(continues on next page)

(continued from previous page)

```
    │   └── README.rst
    └── static-analysis
        └── README.rst
```

To modify the text on this page. Open the `docs/source/documentation/README.rst` file in your favourite editor and run the following command in the `docs` directory to compile the documentation to a static HTML website:

```
make html
```

3.5 Continuous integration

3.5.1 Purpose

Continuous integration is a process which automatically detects violations in the source code. These violations consists of but are not limited to:

- not compliant with the psr-2 standard
- failing tests
- the code is a mess

3.5.2 Getting started

This template provides a *.travis-ci.yml* configuration which is tailored for https://travis-ci.org/. The configuration in the *.travis-ci.yml* should be enough to get you started so sign up at https://travis-ci.org/ and configure travis to test your repository.