Composer

هو [نظام إدارة الحزم](https://ar.wikipedia.org/wiki/%D9%86%D8%B8%D8%A7%D9%85_%D8%A5%D8%AF%D8%A7%D8%B1%D8%A9_%D8%A7%D9%84%D8%AD%D8%B2%D9%85) من أجل [لغة](https://ar.wikipedia.org/wiki/%D9%84%D8%BA%D8%A9_%D8%A8%D8%B1%D9%85%D8%AC%D8%A9) [PHP](https://ar.wikipedia.org/wiki/%D8%A8%D9%8A_%D8%A5%D8%AA%D8%B4_%D8%A8%D9%8A) التي توفر شكل موحد لإدارة تبعيات PHP البرامج المطلوبة [المكتبات](https://ar.wikipedia.org/wiki/%D9%85%D9%83%D8%AA%D8%A8%D8%A9_%D8%A8%D8%B1%D9%85%D8%AC%D9%8A%D8%A9).

What is Composer and how to use it in project

## **Ok, but why I should use Composer?**

That question rises every time when someone suggests us using some kind of a new technology. In this case, the process of getting accustomed to the new is totally painless.

The current PHP community works mostly with such frameworks as: **Laravel, Silex, Phalcon or**[**Symfony2**](https://www.merixstudio.com/development/symfony2/). I’d like to note that most of current projects or other frameworks are working on bundles developed for Symfony.

For every new module a new version is created – many versions and bundles means trouble, and by trouble I mean (it was a nightmare for Linux users some time ago too)… **DEPENDENCIES**.

How to deal with this problem then? This is where **Composer** comes with help.

If you have a project with a huge number of dependencies and:

* some of them depend on other bundles
* you want to choose which packages are necessary for your project
* you want to have control on the versions of files you install

If you said ‘yes’ to any of these, then Composer is probably the perfect tool for you.

## **Composer - how it works?**

Composer uses [Packagist.org](https://packagist.org/) as a main bundles’ provider. It provides files from repositories that users report on the site. Packagist.org hands over such features as, among others: versioning or the integration with GitHub and/or bitbucket.

Composer also offers:

* Packages/bundles can be found via packagist.org or with the help of Composer,
* Most repositories on, for instance, Github, include links or ready commands that simplify downloading of a suitable bundle via Composer,
* Thanks to the composer.lock file we can be sure that our files will not upgrade their versions automatically,
* A hierarchic structure can be generated, for instance: a national repository website may require a continental repository website (child-parent dependency), while being a child of frameworks’ repository site (a parent for the previous parent). This way a structure of dependencies is created.

## **Step by step - the installation of Composer**

Composer can be downloaded in a few ways (these operations are made in the main folder of the project)

1. the first is based on using Curl:  
   $ curl -sS https://getcomposer.org/installer | php
2. if Curl isn’t installed:  
   $ php -r "readfile('https://getcomposer.org/installer');" | php
3. if we want to install Composer globally (Linux version):  
   $ curl -sS https://getcomposer.org/installer | php  
     
   $ mv composer.phar /usr/local/bin/composer

## **Basic usage**

Composer.json used as an exemplary file – it defines the list of project’s dependencies (a fragment of code for a project in Symfony2):

{  
"name": "symfony/framework-standard-edition",  
"license": "MIT",  
"type": "project",  
"description": "The \"Symfony Standard Edition\" distribution",  
"autoload": {  
"psr-4": { "": "src/" },  
"files": [ "app/AppKernel.php" ]  
},  
"require": {  
"php": ">=5.3.9",  
"symfony/symfony": "2.8.\*@dev",  
"doctrine/orm": "^2.4.8",  
"doctrine/doctrine-bundle": "~1.4",  
},  
"require-dev": {  
},  
"scripts": {  
},  
"config": {  
"bin-dir": "bin"  
},  
"extra": {  
"symfony-app-dir": "app",  
"symfony-web-dir": "web",  
"symfony-assets-install": "relative",  
"incenteev-parameters": {  
"file": "app/config/parameters.yml"  
},  
"branch-alias": {  
"dev-master": "2.8-dev"  
}  
}  
}

The section require is the key for us. It contains the json object which stores data of bundles used by us. The name of the key is the name of the bundle and the value means the fitting version of the package.

Additionally:

* **name**- the name of the bundle in the format author/name. Usually it has the same name as on Github/Bitbucket,
* **description** – a short description of the bundle,
* **keywords** – some keywords useful while filtering the project,
* **license** – information about the current license,
* **scripts** – a very handy and popular method that allows to attach scripts into a particular moment of work of the dependency manager in order to, for instance, call specified command after files update.

## **Selecting versions**

Composer selects appropriate dependencies by itself while checking whether for the latest stable version of the package and ensuring its compatibility with other dependencies. Versions can be narrowed by such operators as: specific version, range, wild card or tilde (~).

**NameExampleDescription**Specific version1.0.2You can select the exact version of the package.Range>=1.0>=1.0<2.0>=1.0<1.1|>=1.2You can define the range of acceptable versions by using such operators as: >, >=, <, <=, !=You can also define multiple ranges by using commas which mean a logical 'AND'. The symbol '|' will be considered as logical 'OR'.'AND' takes precedence over 'OR'.Wild card1.0.\*\* stands for any number inserted at this point. You can specify eg. 1.0.\* and it will be the equal to >=1.0, <1.1.The tilde operator~1.2Very useful for projects that use semantic versioning. ~1.2 is equal to >=1.2, <2.0 and ~1.2.3 >=1.2.3,<1.3

Choosing the right version of the package is based on number of options (in terms of stability):

* dev
* alpha
* beta
* RC
* stable

By default, only stable bundles are selected. If you want to use, for instance dev package, you have to point it out and modify composer.json.  
  
Every version of each package is checked for its stability. Bundles that don't meet the minimum stability requirements will be skipped in the process of delivering the dependencies to your project.

## **Basic Composer commands**

$ composer install

This command reads the content of the composer.json from the current folder and installs the selected dependencies into vendor folder.

If composer.lock exists in the current directory, the specific version will be used instead of those selected by Composer. It assures that there’s a compatibility between the downloaded and used packages.

If composer.lock doesn’t exist in the current directory, it will be created after the bundles’ installation.

$ composer dump-autoload

Dump-autoload allows to refresh the content of autoload.php and to generate new autoload entries.

$ composer self-update

Used to update Composer version.

$ composer update

This command matches all dependencies and saves the specific versions to composer.lock.

$ composer update vendor/package vendor/package2

This command updates selected packages if stable updates of these bundles are available.

$ composer show  
absolvent/phpunit-symfony  
acid/symfony  
admafa/proyecto-final-symfony  
...

It displays the list of available bundles in the library.

$ composer show vendor/package  
$ composer show vendor/package 1.0.2

Returns the details of the bundle or of its particular version.

$ composer show -i  
alexandresalome/php-selenium 1.0.1 PHP Library for Selenium  
behat/behat v2.5.0 Scenario-oriented BDD framework for ...  
behat/common-contexts v1.2.0 Commonly used contexts for Behat  
...

Shows installed bundles and their versions.

$ composer show -s  
...  
requires  
php >=5.4.0  
symfony/symfony 2.3.\*  
symfony/assetic-bundle 2.3.\*  
symfony/swiftmailer-bundle 2.3.\*  
symfony/monolog-bundle 2.3.\*  
...

Command use to see how Composer interprets composer.json file.

## **What problems can occur when using the Composer?**

One of popular problems that occur while working with Composer is the amount of memory assigned to the instance of PHP. However, it can be easily solved:

PHP Fatal error: Allowed memory size of XXXXXX bytes exhausted <...>  
$ php -d memory\_limit=-1 composer.phar <...>

We can also modify the content of the php.ini file:

; Use -1 for unlimited or define an explicit value like 2G  
memory\_limit = -1  
…  
$ sudo service apache2 restart

It’s recommended to have **git** installed in order to work with Composer. Sometimes Github asks us about our login and password. This happens because the portal protects itself from overloading the site by limiting the amount of anonymous downloads.

## **Autoloading**

Composer provides the opportunity to configure autoload file of every library. Composer generates a vendor/autoload.php that we can put in our code using:

require 'vendor/autoload.php';