

# Manual de Composer



Miguel Angel Alvarez

 desarrolloweb.com

[desarrolloweb.com/manuales/tutorial-composer.html](http://desarrolloweb.com/manuales/tutorial-composer.html)

# Introducción: Tutorial de Composer

En este tutorial queremos explicarte cómo usar Composer para gestionar los paquetes de software que usas en tus proyectos o las librerías de las que depende tu aplicación.

La idea de un gestor de dependencias no es nueva en PHP, ya existía en otros lenguajes de programación. Composer ha hecho posible que este concepto podamos usarlo también los desarrolladores de PHP para mejorar nuestro flujo de trabajo.

Composer te resume todas las tareas de instalación de librerías, frameworks y en general cualquier tipo de software que uses para desarrollar un proyecto. A través de una simple declaración podrás mencionar las librerías que quieres tener disponibles y Composer te las descargará para ti y las colocará en la carpeta de tu proyecto. Además, cuando esas librerías se actualicen el gestor de dependencias será capaz de sustituir tus descargas por las versiones más nuevas, siempre que tú así lo indiques.

Por último, te ofrece un sistema que te permite, en una única línea de código, tener todas las librerías o programas que has definido como dependencias, para no tener que hacer includes o requires independientes para cada elemento que quieras usar, simplemente estarán allí.

Composer además trabaja con Packagist, un completo repositorio de software libre con aquellos paquetes que podrás instalar automáticamente vía gestor de dependencias.

En este manual de Composer iremos publicando los artículos de modo que puedas no solo resolver tus dudas sobre el gestor de dependencias, sino también diversos problemas comunes que podrías encontrarte.

Encuentras este manual online en:

<http://desarrolloweb.com/manuales/tutorial-composer.html>

## Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

---

### **Miguel Angel Alvarez**

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.



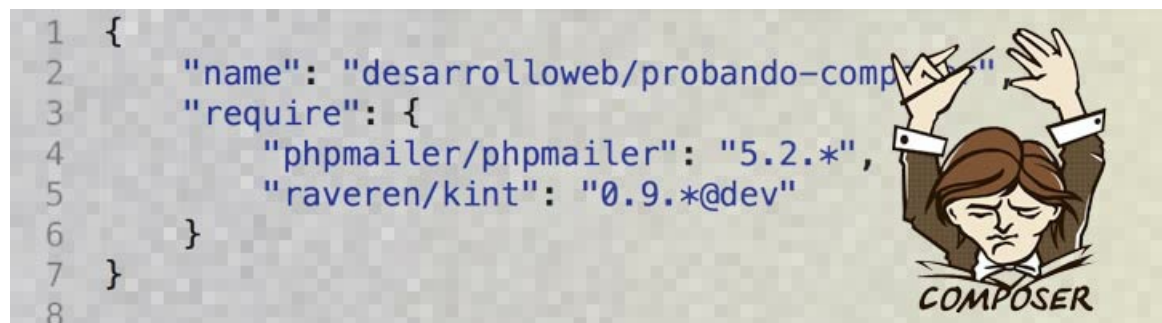
# Composer, gestor de dependencias para PHP

**Composer es una herramienta imprescindible para los desarrolladores en PHP, que permite gestionar de una manera ágil las dependencias de un proyecto.**

Composer es un gestor de dependencias en proyectos, para programación en PHP. Eso quiere decir que nos permite gestionar (declarar, descargar y mantener actualizados) los paquetes de software en los que se basa nuestro proyecto PHP. Se ha convertido en una herramienta de cabecera para cualquier desarrollador en este lenguaje que aprecie su tiempo y el desarrollo ágil.

¿Empiezas un nuevo proyecto con PHP? echa un vistazo antes a Composer porque te puede ayudar bastante en el arranque y gracias a él podrás resumir muchas de las tareas de mantenimiento de las librerías de terceros que estés usando.

<https://getcomposer.org>



En este artículo te resumimos los detalles para entender qué es Composer, cómo funciona y para instalarlo en tu sistema. Primero comenzaremos explicando qué es un gestor de dependencias, luego veremos cómo funciona Composer para darnos cuenta qué aspectos de nuestro día a día nos va a simplificar. Por último veremos cómo instalarlo y cómo usar librerías gestionadas con Composer en nuestro código PHP.

## Por qué un gestor de dependencias

Cuando comienzas un proyecto en PHP, ya de cierta complejidad, no te vale solo con la librería de funciones nativa de PHP. Generalmente todos usamos alguna que otra librería de terceros desarrolladores, que nos permite evitar empezar todo desde cero. Ya sea un framework o algo más acotado como un sistema para debug o envío de email, validación de formularios, etc., cualquier cosa que puedas necesitar ya está creada por otros desarrolladores. Si no la estás usando ninguna librería posiblemente estés perdiendo tu precioso tiempo, pero eso es otra

discusión.

De modo que, al comenzar el proyecto hasta ahora teníamos que ir a la página de cada uno de los componentes de software que queríamos usar, descargarlos, copiarlos en la carpeta de nuestro proyecto, etc. No solo eso, cuando estamos en mitad del desarrollo, o ya en producción, y nos cambian la versión de la librería, tenemos que volverla a descargar manualmente, actualizar los archivos, etc. Nadie se había muerto por hacer todo ese tipo de tareas de configuración y mantenimiento, pero no cabe duda que nos llevan un tiempo.

Todo eso sin contar con que ciertos softwares, como un framework como Symfony, dependen a su vez de muchas otras librerías que tendrías que instalar a mano y a su vez, mantener actualizadas.

Los gestores de paquetes nos ayudan para resumir las tareas de descarga y mantenimiento de las versiones del proyecto para que estén siempre actualizadas. Ya existían en otros lenguajes de programación y nos resultaban especialmente útiles como npm en NodeJS. Ahora los desarrolladores de PHP también contamos con esta herramienta gracias a Composer.

## Cómo funciona Composer

---

Composer nos permite declarar las librerías que queremos usar en un proyecto. Su uso es extremadamente simple, lo que anima a cualquier persona a usarlo, sea cual sea su nivel técnico.

Para beneficiarnos del workflow que nos propone Composer simplemente tenemos que escribir un archivo de configuración en el que indicamos qué paquetes vamos a requerir. El archivo es un simple JSON en el que indicamos cosas como el autor del proyecto, las dependencias, etc.

El archivo JSON debe tener un nombre específico: `composer.json`

A continuación tienes un ejemplo de JSON donde declaramos varios parámetros de nuestra aplicación.

```
{
  "name": "desarrolloweb/probando-composer",
  "require": {
    "phpmailer/phpmailer": "5.2.*",
  }
}
```

Luego nos pondremos a desgranar este código para que se entienda cada una de sus partes, así como veremos qué otra información podemos colocar en este JSON. La idea es ver lo sencillo que es declarar qué librerías o software estás utilizando y con ello dejar nuestro proyecto listo para la "magia" de Composer.

Una vez tenemos definidas las dependencias en nuestro proyecto debemos instalarlas. Esto lo conseguimos con un simple comando en el terminal en el que le pedimos a Composer que las

instale:

```
composer install
```

**Nota:** Ese comando puede variar según la instalación que tengas en tu sistema de Composer. Especificaremos en un futuro artículo diversas situaciones en las que tengamos que generar variantes de este mismo comando. De momento nos vamos a quedar en una presentación de Composer, pero en seguida nos ponemos a aprender en detalle aquí en Desarrolloweb.com

Lanzado ese comando Composer se encargará de ir a los repositorios de paquetes de software y descargar aquellas librerías mencionadas, copiándolas en la carpeta de tu proyecto.

Una vez finalizado el proceso en tu consola de comandos podrás encontrar en la carpeta de tu proyecto un directorio llamado "vendor" donde estarán las librerías declaradas. Ya solo nos queda hacer los includes para que estén disponibles en tus aplicaciones y para ello también nos ayuda Composer.

Simplemente tendremos que hacer un único include o require en nuestro código y todas las librerías estarán disponibles para usar.

```
require 'vendor/autoload.php';
```

## Packagist

---

Para terminar de convencerte y contarte la introducción completa, debes echar un vistazo a Packagist. Se trata del repositorio de paquetes que son instalables por medio de Composer.

En la página de Packagist encontrarás un buscador que te puede dar una idea de la cantidad de material que encuentras disponible para usar en cualquier proyecto PHP.

<https://packagist.org/>

Simplemente busca por cualquier concepto que te interese, como email, template, wysiwyg, etc. Verás que te aparecen varias opciones clasificadas por popularidad, descargas, etc. Además sobre cada paquete encuentras información y el código necesario para declarar tu dependencia en el JSON de Composer.

## Conclusión

---

Espero que con lo que hemos visto hasta ahora te haya llamado la atención esta herramienta. La verdad es que es muy útil y como decimos, una vez comienzas a usarla te das cuenta de todo el trabajo que te quita del medio, no solo en la descarga de los paquetes, sino también en las actualizaciones de las librerías con el comando "composer update" que veremos más adelante.

Sabemos que nos hemos dejado muchas cosas en el tintero, como el proceso de instalación y el detalle del JSON, pero lo veremos ya en próximos artículos. De momento queríamos presentarte el gestor de dependencias y que sepas por qué los desarrolladores de PHP lo hemos adoptado con tanto entusiasmo.

En el próximo artículo vamos a detallar el proceso de instalación de Composer.

Además de los próximos artículos donde vamos a explicarlos los detalles del flujo de trabajo con Composer para la gestión de dependencias, vamos a presentaros ahora un vídeo de nuestro canal de Youtube donde encuentras resumidos algunos de los pasos básicos de uso de Composer.

Para ver este vídeo es necesario visitar el artículo original en:

<http://desarrolloweb.com/articulos/composer-gestor-dependencias-para-php.html>

Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado por primera vez en 17/10/2014

Disponible online en <http://desarrolloweb.com/articulos/composer-gestor-dependencias-para-php.html>



# Cómo instalar Composer

**Instalación del gestor de dependencias Composer en tu sistema operativo de modo que puedas empezar a usarlo en tus proyectos.**

En el artículo anterior pudimos conocer Composer, el gestor de dependencias de PHP que nos permite mejorar el workflow del desarrollo PHP cuando estamos usando librerías de terceros, no solamente en el momento de creación de un proyecto, sino también en su mantenimiento.

En esta ocasión vamos a dedicarnos a explicar el proceso de instalación. Realmente como verás es un proceso tan simple que se puede resumir en un comando en el terminal, pero queremos comentar algunas variantes de instalación y problemas con los que nos podamos encontrar, con sus soluciones.



## Instalar Composer

Para disponer de esta maravilla en nuestro sistema, a fin de gestionar nuestras dependencias en PHP, debes instalarlo primero. El proceso es bien simple. Si todo va bien se reduce a estas acciones, que dependen de tu sistema operativo.

### Windows:

Si estas en Windows usarás un instalador de toda la vida. Ningún secreto. Es un asistente y vas yendo a través de varias ventanas, siguiente, siguiente.

### Linux / Mac:

Si estás en Linux o Mac, usarás la línea de comandos para instalar Composer. Es tan sencillo como ejecutar esta instrucción:

```
curl -sS https://getcomposer.org/installer | php
```



**Nota:** Ese comando requiere Curl, si te falla puedes intentar hacer lo mismo pero con el comando de PHP.

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

En último caso también puedes descargar `composer.phar` a mano y colocarlo en la carpeta de tu proyecto donde quieres que esté disponible. <http://getcomposer.org/composer.phar>

Esto descargará en tu carpeta el archivo `composer.phar` que es un ejecutable de PHP. Los `.phar` los puedes correr por la línea de comandos como otros comandos del terminal. Ese archivo es justamente el que invocas para cargar las dependencias o actualizarlas, enseguida lo veremos.

## Instalación global o particular a un proyecto

Aquí hay que hacer una aclaración importante, que te puede interesar porque dependiendo de ello hay ligeras diferencias de uso de Composer. Se trata de tener una instalación global de Composer o simplemente tenerlo disponible en un proyecto concreto.

Cuando lo instales en Windows, Composer va a estar disponible en cualquier lugar de tu sistema. Es una instalación global, que quiere decir que estés en el directorio que estés podrás invocar el comando de Composer. Creo que es lo más cómodo, así que si eres usuario de Windows no necesitas leer más sobre este punto.

Sin embargo, con la instrucción que hemos indicado para instalarlo en Linux/Mac, sólo lo tendrás disponible en la carpeta donde estabas cuando lanzaste el comando de Curl. Lo tendrías que instalar una vez para cada proyecto, lo que tampoco es algo descabellado, pero que se puede mejorar.

En el caso de haber hecho una instalación particular para un proyecto, colocarás el `composer.phar` en el directorio raíz de tu proyecto, de modo que luego lo puedas invocar para instalar las dependencias. Sin embargo, si lo deseas, también lo puedes instalar de manera global en Linux si colocas el `composer.phar` en otra ruta.

```
mv composer.phar /usr/local/bin/composer
```

**Nota:** Si ese comando no te funciona simplemente intenta hacerlo como superusuario.

```
sudo mv composer.phar /usr/local/bin/composer
```

Y si estás en Mac OSX la manera más cómoda de tener Composer en modo global es instalarlo

vía "homebrew", el gestor de paquetes de sistemas Mac. Es un proceso algo más largo, pero también te beneficiarás de tener homebrew en tu máquina para instalar otros softwares que puedas necesitar. En la propia documentación de Composer nos dan los comandos que deberíamos ejecutar.

**Nota:** Primero tendrás que instalar el propio homebrew, tal como se explica en la página <http://brew.sh/>

```
brew update
brew tap homebrew/dupes
brew tap homebrew/php
brew install composer
```

Pero Ojo! En mi caso concreto esa serie de comandos me devolvió un problema diciendo que no estaba la dependencia de Composer con php53, php54, php55 o php56. Eso a pesar de tener instalado PHP con Mamp. Yo lo solucioné ejecutando el comando:

```
brew install php56
```

Eso me instaló php56 con brew y luego funcionó todo bien cuando hice el comando.

```
brew install composer
```

**Nota:** Tengo que reconocer que desconozco si simplemente en el "brew tap homebrew/php", indicando la versión de PHP que queremos, se puede solucionar.

## Posibles problemas en Windows y sus soluciones

El proceso de instalar Composer en un Windows es muy sencillo, No obstante durante la instalación te puede dar un problema habitual que la verdad asusta un poco cuando lo ves, pero que es muy sencillo de solucionar. Por lo menos a mi me ha surgido ese problema en Windows en mi instalación de PHP bajo Xamp.

*The openssl extension is missing, which means that secure HTTPS transfers are impossible. If possible you should enable it or recompile php with --with-openssl*

No te preocupes porque no tienes que recompilar nada! Simplemente abre el archivo php.ini y busca la línea:

```
;extension=php_openssl.dll
```

Quítale el punto y coma ";" que lleva delante, porque hace que esa línea se tenga en cuenta como un comentario. Entonces ya estará activa la extensión openssl y podrás instalar composer normalmente.

Otro problema que puedes encontrar en Windows, aunque ya no en la instalación sino en el uso de Composer, es que el archivo JSON esté mal formado. Si revisas la sintaxis de tu JSON y encuentras que está todo bien quizás tu problema sea la codificación o juego de caracteres. Los archivos JSON deben escribirse en encoding "UTF-8", además no debes usar BOM. La codificación la puedes cambiar con tu editor preferido, cada editor tiene la opción de guardar con un encoding determinado en una parte distinta.

## Conclusión

---

De momento creo que es todo lo que necesitas saber para que instalar Composer no te resulte ningún problema. En el siguiente artículo nos dedicaremos a explicar en detalle el uso de Composer.

Este artículo es obra de *Miguel Angel Alvarez*  
Fue publicado por primera vez en 23/10/2014  
Disponible online en <http://desarrolloweb.com/articulos/como-instalar-composer.html>

# Usando Composer

**En este artículo te ofrecemos todas las claves para usar Composer, el gestor de dependencias de PHP. Declarar dependencias, instalarlas en un proyecto o actualizarlas.**

En el [Manual de Composer](#) hemos podido tratar ya varios asuntos relacionados con Composer, el popular gestor de dependencias que todos los desarrolladores de PHP deberíamos usar para mejorar nuestro flujo de trabajo.

A estas alturas ya lo debemos tener instalado y aclarados algunos conceptos importantes cuando queremos trabajar con Composer. Ahora vamos a ver cómo declarar una serie de dependencias y cómo instalarlas en nuestro proyecto a través de la línea de comandos con Composer. También veremos cómo actualizar nuestras dependencias una vez se haya iniciado el proyecto.



## Crear el archivo composer.json

**Artículo usar Composer actualizado en 2018:** Para comenzar a usar Composer en un proyecto debemos hacer un primer paso, que consiste en crear un archivo llamado "composer.json", cuya utilidad se basa en gestionar todas las dependencias de un proyecto con Composer. Ya lo explicamos en el primer artículo de este manual sobre Composer, pero lo volvemos a mencionar: se trata de un archivo de texto en notación de objeto Javascript (JSON).

Este paso es muy sencillo, pues básicamente se trata de crear un archivo de texto con la configuración que necesitemos para el proyecto. Podemos hacerlo de dos maneras:

- De manera interactiva, por medio de un asistente de consola de comandos
- De manera manual, creando el archivo en la raíz de nuestro proyecto y colocando el código por nosotros mismos

Obviamente, es más sencillo hacerlo de manera interactiva, así que vamos a empezar por ahí.

Pero ten en cuenta que, tanto si creas el `composer.json` de manera interactiva como de manera manual, una vez creado tendrás que lanzar el proceso de instalación de dependencias, con "`composer init`", como te describimos más adelante en este mismo artículo.

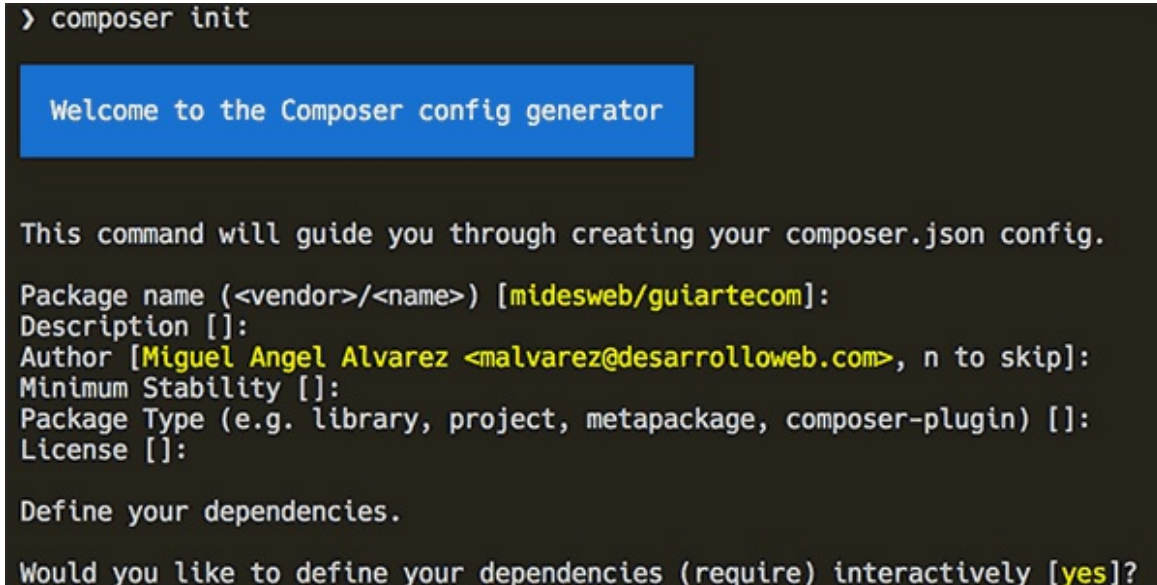
## Crear el archivo `composer.json` con "`composer init`"

Una vez [instalado composer](#), tenemos a nuestra disposición el comando "`composer`" en terminal. Este comando tiene a su vez una serie de operaciones posibles, entre la que se encuentra "`init`", que sirve para inicializar un proyecto, creando el correspondiente `composer.json` de manera automática. Aparte de darnos menos trabajo, es una opción interesante porque nos ahorra errores, y tiempo de trabajo, al escribir el `composer.json`.

El asistente para creación de este archivo se lanza entonces, **desde la raíz de tu proyecto**, con el comando:

```
composer init
```

Lanzado este comando el terminal comenzará a recabar datos de nuestro proyecto, como el nombre, autor, descripción.



```
> composer init

Welcome to the Composer config generator

This command will guide you through creating your composer.json config.
Package name (<vendor>/<name>) [midesweb/guiartecom]:
Description []:
Author [Miguel Angel Alvarez <malvarez@desarrolloweb.com>, n to skip]:
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []:
License []:

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]?
```

Posteriormente nos preguntará si queremos instalar las dependencias interactivamente. Si sabemos qué paquetes queremos instalar de entrada podemos decirle que sí. Si no sabemos qué queremos instalar, entonces tendremos que decirle que no. El sistema interactivo te permitirá buscar por palabras cualquier dependencia y te ofrecerá varias posibilidades para instalación, de la que puedes escoger la que te interese.

Por ejemplo, si deseamos instalar un paquete que se llama "[plates](#)" ([para crear templates en PHP](#)) podemos buscarlo y seleccionarlo en la lista que nos ofrece. Luego también nos preguntará la versión de la librería que queremos instalar. Si deseamos la última versión simplemente lo dejamos en blanco, tal como puedes ver en la siguiente imagen:

```
Would you like to define your dependencies (require) interactively [yes]?
Search for a package: plates

Found 15 packages matching plates

[0] league/plates
[1] wordplate/plate
[2] wordplate/wordplate
[3] zendframework/zend-expressive-platesrenderer
[4] projek-xyz/slim-plates
[5] samsonasik/error-hero-module
[6] intrepidity/license-plate
[7] slim/plates
[8] liquidbox/silex-plates
[9] rych/silex-plates-provider
[10] odahcam/plates-includer
[11] franzl/laravel-plates
[12] tapestry-cloud/tapestry
[13] philipsharp/slim-view-plates
[14] amamarul/boiler-plate-commands

Enter package # to add, or the complete package name if it is not listed: 0
Enter the version constraint to require (or leave blank to use the latest version):
Using version ^3.3 for league/plates
```

El proceso se repite para cada package que quieras instalar. Luego comienza todo de nuevo para las dependencias que tengas en desarrollo (require-dev), que son todas las librerías que necesitas en tu proyecto a la hora de crear tu base de código, pero que no requieren estar en el servidor de producción, como por ejemplo librerías de testing o creación de juegos de datos "fake" para poblar inicialmente bases de datos.

Una vez terminado el asistente nos presentará el código del composer.json que se va a generar y si le decimos que está correcto, se creará el archivo composer.json con la configuración seleccionada.

## Crear a mano nuestro archivo composer.json

Como composer.json es un archivo de texto plano, si lo vemos necesario lo podemos crear manualmente como cualquier otro fichero en tu proyecto. Ese archivo debes colocarlo en la carpeta raíz de tu proyecto y su nombre debe de ser composer.json.

El archivo JSON sirve para indicar algunos datos sobre nuestro proyecto, así como las dependencias que tiene con otros paquetes de los que te encuentras en Packagist.

Para que sirva de muestra, veamos a continuación el contenido de un archivo de ejemplo composer.json.

```
{
  "name": "desarrolloweb/probando-composer",
  "require": {
    "phpmailer/phpmailer": "5.2.*",
    "raveren/kint": "0.9.*@dev"
  }
}
```

En este archivo estamos indicando que nuestro proyecto se llama "desarrolloweb/probando-



composer". Siempre se utilizan dos nombres, uno el nick de la empresa o creadores y otro el nombre del proyecto en sí. Luego con el campo require estamos indicando que vamos a usar dos librerías, por un lado el phpmailer de phpmailer y el kint de ravener.

Además, observarás que cada paquete tiene una versión requerida. Por ejemplo en phpmailer declaramos como versión "5.2.". *Eso quiere decir que te instale siempre la versión 5.2.x (la más reciente de la 5.2). Pero podrías haber declarado "5."*. Existen varios operadores para especificar la versión. Lo mejor es que revises la documentación de Composer para conocerlos todos.

**Nota:** En el siguiente artículo de este manual te explicaremos con mayor detalle la [sintaxis y los datos que puedes indicar en el composer.json](#).

## Instalando las librerías o software definido como dependencias

Una vez declarado ese json en la carpeta de nuestro proyecto tienes que lanzar un comando para que composer se ponga en marcha, revise los paquetes que hemos declarado, los descargue con sus dependencias y los instale en la carpeta de nuestro proyecto.

Ese comando de consola lo tienes que hacer desde la carpeta de tu proyecto. Abres el terminal y te situas en la carpeta donde has creado el composer.json.

Dependiendo de la instalación que hayamos realizado de Composer el comando puede tener ligeras variaciones. Esto también puede depender de tu sistema operativo.

**Si estamos en Windows** es tan sencillo como hacer:

```
composer install
```

**Nota:** Lógicamente ese comando no te va a funcionar si no has instalado previamente Composer.

Recuerda que en Windows Composer se instala de manera global, para que funcione desde el terminal en cualquier carpeta de tu sistema. Simplemente te colocarás en la carpeta de tu proyecto y lanzarás el comando "composer install" para descargar e instalar dependencias.

**Si estamos en Linux/Mac y hemos instalado Composer de manera local a un proyecto.** Será algo como:

```
php composer.phar install
```

**Si estás en Linux/Mac y has hecho los pasos para instalar Composer de manera**



**global** el comando puede ser:

```
composer install
```

Esto es así en el caso de Mac OSX habiendo instalado Composer a través de Homebrew. Pero en el caso de Linux quizás tengas que hacer:

```
composer.phar install
```

**Nota:** Para saber cómo instalar Composer y sobre las variantes global y local en sistemas Linux / Mac OSX, por favor revisa el artículo sobre [Instalar Composer](#). Si tienes cualquier problema no está de más consultar por Internet pues muchos desarrolladores seguramente hayan pasado por ello antes.

En este punto quiero mencionar una página que a mi me ha sacado de dudas, en askubuntu <http://askubuntu.com/questions/116960/global-installation-of-composer-manual> donde además un usuario propone un modo de crear un alias para que en Linux no tengas que hacer "composer.phar install", sino simplemente "composer install".

```
alias composer='/usr/local/bin/composer.phar'
```

Lógicamente antes tienes que haber hecho global tu instalación de Composer como ya expliqué en el artículo sobre la instalación mencionado en esta nota.

Volviendo de nuevo al comando

```
php composer.phar install
```

Por explicarlo un poco ese podemos decir que el programa que está haciendo funcionar es composer.phar es el ejecutable de PHP. Sin embargo, si no le colocas php delante, el sistema también es inteligente para saber que los .phar se tienen que correr bajo el intérprete de PHP. Lógicamente debes tener PHP en tu sistema y debe estar disponible de manera global para que, estando en la carpeta de tu proyecto, el terminal sea capaz de saber dónde está el ejecutable de PHP. en ese comando llamas a composer.phar que es un archivo ejecutable escrito en PHP y luego a Composer le estás diciendo que instale las dependencias.

## Actualizando dependencias con Composer

Una vez ya tienes las dependencias instaladas, si deseas actualizarlas, o bien si has añadido dependencias a tu archivo JSON, harás un comando como este:

```
composer update
```

**Nota:** Las variantes de ese comando serán más o menos igual que las del comando "composer install" dependiendo de nuestra instalación de composer. Quizás en tu sistema debas hacer algo como "php composer.phar update".

Con eso Composer se descargará automáticamente las librerías que se hayan actualizado, o aquellas que se han agregado al composer.json y las colocará dentro de la carpeta "vendor" de tu proyecto.

## Usar los paquetes instalados por Composer

Aparte de facilitar enormemente la descarga e instalación de las dependencias, Composer te proporciona un sistema de "autoload" de las clases que componen aquellas librerías que estás usando.

Cuando instalas una dependencia via Composer te crea un directorio llamado "vendor" que es donde se instalan todas las librerías que has solicitado. Además, en esa misma carpeta encontrarás un archivo llamado "autoload.php" que es el único script que deberás incluir desde PHP.

```
<?php
require "vendor/autoload.php";

// ahora ya puedes acceder a las clases creadas por las librerías declaradas como dependencias...
```

Es tan sencillo como esto. No tienes que hacer nada más. Cada vez que necesites una clase o una función de las declaradas en las dependencias de tu composer.json, estará disponible para ti sin que tengas que hacer otras acciones. Como observarás, este mecanismo también te permite evitar crear una serie de includes o requires en todos tus archivos PHP, reduciendo tu código.

Sin duda usar Composer te facilitará mucho la vida en el día a día como desarrollador de PHP. Una vez te acostumbres a este Workflow, muy sencillo como has podido comprobar, no entenderás cómo puedes haber vivido tanto tiempo sin él.

Este artículo es obra de *Miguel Angel Alvarez*  
Fue publicado por primera vez en 21/11/2014  
Disponible online en <http://desarrolloweb.com/articulos/usando-composer.html>

# Setup de un proyecto PHP con composer.json

## Explicaciones sobre el contenido del archivo JSON de un proyecto para usar Composer como gestor de dependencias.

En el [Manual de Composer](#) ya hemos explicado todo el flujo de trabajo que te permitirá adoptar Composer en tu día a día en el desarrollo PHP. Realmente con lo que sabes y una rápida referencia en la documentación para consultar detalles que se nos hayan podido escapar, tienes todo lo que hace falta para usar Composer.

Sin embargo, hemos pasado muy "de puntillas" por la estructura del archivo composer.json que nos sirve para declarar las características de nuestro proyecto y las dependencias que pueda tener.

Pero Ojo, el composer.json nos sirve no solamente para que Composer sepa las dependencias que debe instalar para tu proyecto, también sirve en el caso que quieras liberar tu proyecto como un paquete para que lo usen otras personas y publicarlo en Packagist. En este artículo nos vamos a centrar más en lo que necesitas para definir tus dependencias.



## El esquema composer.json

El schema (esquema) de un JSON define la estructura del documento así como los valores posibles que tengan cada uno de sus campos. La dirección concreta donde puedes encontrar la descripción completa del schema del composer.json la encuentras en esta URL de documentación:

<https://getcomposer.org/doc/04-schema.md>

## Campos que describen tu proyecto

Veamos ahora una serie de campos para ofrecer información de nosotros como desarrolladores

de un proyecto. Estos son los datos que necesitarías enviar a Packagist, pero que no son necesarios a nivel interno para que Composer funcione, sino más bien para etiquetar tu desarrollo, mencionar autores, etc.

### **name:**

Sirve para indicar el nombre del autor, se compone de dos partes, el "vendor" (la empresa o nick del desarrollador o grupo que lo ha creado) y el nombre del proyecto propiamente dicho. Por supuesto, un vendor puede crear diversas librerías o paquetes y quedarían todas vinculadas al mismo vendor con nombres de proyectos distintos.

### **description:**

Es la descripción que ofrecemos de este paquete. Es un texto normalmente de una única línea.

### **homepage:**

Una URL del sitio web del proyecto.

### **authors:**

Es un array con los autores del proyecto. Cada uno de los elementos de ese array es a su vez un objeto JSON donde se pueden indicar distintos datos: name, email, homepage, role (rol dentro del proyecto).

```
"authors": [  
  {  
    "name": "Miguel Angel Alvarez",  
    "email": "algo@desarrolloweb.com",  
    "homepage": "http://www.desarrolloweb.com",  
    "role": "Project Manager"  
  },  
  {  
    "name": "Alvaro Martínez",  
    "email": "algo@escuela.it",  
    "homepage": "http://www.escuela.it",  
    "role": "Beta tester"  
  }  
]
```

---

## **Definir las dependencias**

A nivel de desarrollo de nuestras propias aplicaciones, el campo más importante donde debemos centrarnos es en la definición de las dependencias, así como las versiones que deseamos que sean instaladas, o actualizadas de cada una de esas dependencias.

### **require:**

Es un objeto con una serie de pares clave/valor que definen cada una de las dependencias que Composer debe instalar para nuestro proyecto. En la clave debemos de indicar el nombre del paquete que depende (que obtienes del sitio de Packagist) y como valor indicamos la versión que deseamos que esté instalada, o el rango de versiones.

```
"require": {  
    "respect/validation": "0.6.*",  
    "phpunit/phpunit": ">=4.0",  
    "tinymce/tinymce": "dev-master"  
}
```

En los pares clave/valor de cada una de las dependencias, la parte del nombre del vendor y la librería es muy fácil de obtener. Simplemente te vas a Packagist buscas y escoges el paquete que más te convenga y copias la cadena del nombre. Por ejemplo "respect/validation" indica que "respect" es el vendor y "validation" es el nombre de la librería.

La parte de la versión también te puedes ayudar de Packagist, pues ahí encontrarás la lista de versiones liberadas de esa librería y disponibles como paquetes. Lo que verás es que la expresión que define la versión tiene algunos caracteres "comodín" que debes aprender a tratar. En general puedes usar estas variantes de expresión.

- **Versión exacta:** indicas a Composer que debe instalar una versión exacta, y solo esa. Quiere decir que nunca te va a actualizar el paquete, porque tu proyecto debe tener esa versión y no otra. Por ejemplo "4.3.1".
- **Rango de versiones:** permite indicar versiones que sean mayor que una determinada, menor o que esté entre una versión y otra. Por ejemplo ">=2.0".
- **Comodín:** Permite decir cualquier versión de una release mayor. Bueno, puedes usar algo como "4." *para indicar que se deje siempre la versión 4 y cualquier cosa. O algo más restrictivo como "4.2."* que te pondrá siempre la versión 4.2.x. Lógicamente, en este caso cuando actualices, se colocará la versión más avanzada permitida por ese comodín.
- **Virgulilla (el rabo de la eñe):** permite indicar la versión de una manera diferente, "próxima versión significativa". Por ejemplo, "~2.2" siempre te dejará la versión mayor o igual a la 2.2 y menor que la 3.0. Por ejemplo "~2.2.1" te pondrá la versión mayor igual que 2.2.1 y menor que 2.3.
- **También te permite cosas como "dev-master"** que será la versión actual de desarrollo, que puede darse el caso que no sea estable. "1.0.\*@beta" que te permitirá colocar versiones beta o "@dev" que también referencia a versiones de desarrollo que pueden sufrir inestabilidad.

Para encontrar otras posibilidades, por favor consulta la documentación de Composer, en estos enlaces:

- [Crear los enlaces a paquetes dependientes](#)
- [Definir las versiones aceptadas](#)

De momento eso es todo. Con la información que tienes estamos seguros que le podrás sacar todo el jugo a Composer para gestionar las dependencias de tus proyectos PHP.

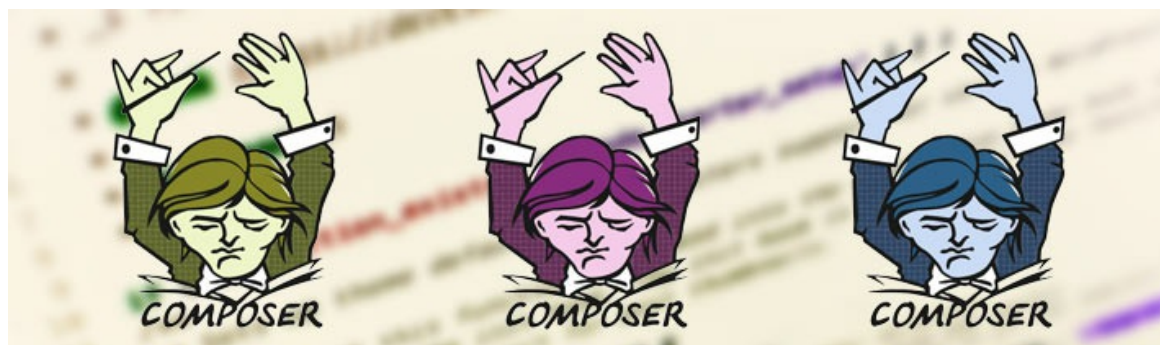
Este artículo es obra de *Miguel Angel Alvarez*  
Fue publicado por primera vez en 12/12/2014  
Disponible online en <http://desarrolloweb.com/articulos/setup-proyecto-php-composer-json.html>

# Autoload de clases con Composer

**Cómo incluir el código de los paquetes que se instalan vía composer en páginas PHP por medio del autoload de clases.**

Una de las ventajas fundamentales de usar Composer es que nos podemos despreocupar sobre cómo se instalan, actualizan y se incluyen las dependencias en el código PHP. Sobre este último punto es en el que vamos a incidir en este capítulo del [Manual de Composer](#).

Obviamente, cuando instalas una dependencia, una librería, una clase o cualquier código de terceros en general, es porque quieres usarla dentro de tu código PHP. Si instalas una librería a mano tendrás que saber en qué carpeta la has metido y cómo se llaman los ficheros con el código que necesitas incluir, pero si lo haces desde este sistema de gestión de dependencias puedes ahorrarte ese trabajo de organización gracias al autoload de clases con Composer.



## Archivo autoload.php

El archivo autoload.php lo crea Composer automáticamente en la carpeta "vendor" y contiene el código para que tus librerías se puedan cargar automáticamente, por demanda según se vayan usando.

Tengamos una o decenas de dependencias, el único archivo que tenemos que incluir en una página PHP del proyecto es el mencionado autoload.php.

```
//autoload de composer  
require 'vendor/autoload.php';
```

**Nota:** no te preocupes por tener muchas dependencias y solo usar unas pocas en una página en concreto. Este archivo autoload no carga nada especialmente, solo tiene el script de autocarga de clases, no resultará más pesado para PHP de lo estrictamente necesario, pues solamente se irán cargando las clases que vayas usando en tu código.



El la carpeta "vendor" debe de estar en la raíz de tu proyecto, típicamente fuera del directorio "document root", para que no sea accesible directamente a través de una URL de tu sitio web . Pero claro, depende desde donde cargues este archivo de autoload, la ruta de tu include puede ser diferente.

## Usar una dependencia instalada

Realmente, cargado el autoload.php no tenemos mucho más que hacer. Simplemente, cuando se necesiten las clases (de programación orientada a objetos), porque se instancien objetos o se haga uso de sus métodos estáticos, éstas estarán disponibles para nosotros.

Ten en cuenta que las dependencias de packagist se organizan por namespaces en PHP, por lo que a la hora de usar las clases tendrás que hacer el correspondiente "use", o bien escribir el namespace completo donde está tu clase.

Por ejemplo, esto hace uso de un método estático de un sistema de plantillas llamado Plates, para inicializarse. Observa el namespace "League\Plates". El nombre de la clase es "Engine".

```
$templates = new League\Plates\Engine($raiz . '../data/plantillas');
```

Ese mismo código también podría hacerse con el correspondiente "use" en el que indicamos la clase y su namespace. Una vez que se declara que se va a usar dicha clase en el correspondiente namespace, ya podemos invocar sus métodos o el constructor sin indicar el namespace completo.

```
use League\Plates\Engine;  
$templates = new Engine($raiz . '../data/plantillas');
```

Resulta obvio, pero queremos remarcar que este código no funcionaría si no se ha hecho el correspondiente autoload con PHP, que nos ofrece el archivo vendor/autoload.php.

**Nota:** Si lo necesitas, puedes encontrar más información sobre espacios de nombres en el artículo [Namespaces en PHP](#).

## Configurar el sistema de autocarga de clases

El proceso descrito antes es el trabajo básico que tendremos que realizar para cargar las librerías, de una manera ágil, gracias a Composer. Pero podría ser necesario para nuestras aplicaciones configurar el sistema de autocarga de clases. En la mayoría de los casos no será necesario, pero puede venirnos bien si queremos aprovecharnos del autoload de Composer para cargar de manera automática las clases de nuestro propio proyecto.

Para ello se define un nuevo campo, llamado "autoload" en el [archivo composer.json](#).

Ese campo nos permite indicar un namespace y las clases que se encuentran detrás de ese namespace.

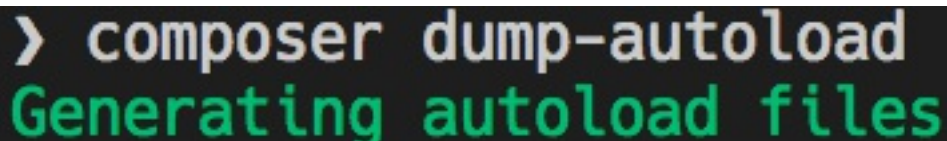
```
"autoload": {  
    "psr-4": {"MiNamespace\\": "mi_carpeta/"}  
}
```

Esto indica que las clases del namespace "MiNamespace" las tiene que ir a buscar al directorio "mi\_carpeta" (suponiendo que "mi\_carpeta" se encuentra colgando la raíz, por lo que sería un hermano del directorio "vendor").

Solo un único detalle. Una vez has configurado el sistema de autoload de composer, debes correr el comando "dump-autoload", para que se vuelva a generar y optimizar todo el proceso de autoload de clases de Composer.

Este paso es tan sencillo como lanzar el comando:

```
composer dump-autoload
```



Ahora que tenemos el archivo "vendor/autoload.php" regenerado, ya podremos tener disponibles las clases de este namespace para su autocarga.

## Configurar el sistema de autocarga en tiempo de ejecución

Sobre el sistema de autocarga también nos viene bien saber que se puede configurar en tiempo de ejecución, y no solamente mediante el composer.json. Esto es útil cuando existen ciertas clases que solo se usan en determinado momento, como clases de testing y no las necesitamos para nada cuando el sitio está funcionando de manera general.

Lo consigues gracias a una instancia de un objeto "loader" que te devuelve el require de tu autoload.

```
$loader = require 'vendor/autoload.php';  
$loader->addPsr4('MiNamespace\\', 'mi_carpeta');
```

## Conclusión

No hay mucho más que decir del sistema de autoload de clases de Composer. Realmente es solo ponerse a usarlo y disfrutar de su comodidad.

Puedes encontrar más información en la [documentación de composer](#), en la [página de autoloading](#) y en el [Manual de Composer](#).

Este artículo es obra de *Miguel Angel Alvarez*  
Fue publicado por primera vez en 26/06/2018  
Disponible online en <http://desarrolloweb.com/articulos/autoload-clases-composer.html>

# Problemas y Errores frecuentes con Composer

**Un artículo en el que vemos los problemas con los que nos hemos encontrado en Composer en el trabajo del día a día y las soluciones.**

Este artículo es simplemente una lista de situaciones que pueden acarrear problemas en el uso de Composer en el día a día. Lógicamente, esta lista es solo un compendio de las cosas que me han ocurrido personalmente, por lo que igual no está tu problema aquí. (Si encuentras novedades que sirvan para ayudar a otras personas puedes publicarlas como comentarios)

Recuerda que en DesarrolloWeb.com hemos hecho un [Manual de Composer](#) que cubre los aspectos más básicos de su uso y configuración. Si no conoces Composer y trabajas con PHP debes de aprender ya mismo!



## Mantener actualizado composer

Aparte de actualizar las dependencias, acuérdate de actualizar el propio Composer. Una de las primeras recomendaciones que te ofrecen en la guía de Composer es que ante cualquier problema te asegures primero de estar ejecutando la versión más nueva del software. Lo consigues con un comando de auto-actualización o auto-update.

```
composer self-update
```

Recuerda que quizás tengas que ejecutar ese comando como superusuario (sudo composer self-update). Recuerda también que si tu instalación de composer no es global posiblemente tengas que invocar llamando directamente a composer.phar a través del intérprete de PHP (php composer.phar self-update). En el [artículo de instalación](#) tienes más información sobre lo que es instalar de manera global

## JSON mal formado

Este error es muy fácil de detectar. Si has formado mal tu JSON tendrás un error como:

*"/composer.json" does not contain valid JSON*

En el artículo de la [instalación de Composer](#) explicamos que uno de los problemas típicos, en Windows es no estar usando la codificación UTF-8. Es necesario para que composer considere que el JSON esté bien formado.

Si estás con UTF-8 debe ser un error de sintaxis, unas comillas que te faltan, una coma o algo así. Si te vuelves loco para encontrar el fallo prueba a usar un Validador de JSON:

<http://jsonlint.com/>

## Memoria límite de PHP

Si tu composer.json declara muchas dependencias quizás tengas que aumentar la memoria de los procesos de PHP. El error que verás es parecido a este:

*Fatal error: Allowed memory size of 536870912 bytes exhausted (tried to allocate 72 bytes) in phar:///usr/local/Cellar/composer/1.0.0-alpha8/libexec/composer.phar/src/Composer/DependencyResolver/RuleSetGenerator.php on line 123*

Esto se soluciona asignando más memoria para el proceso de PHP y se indica en el php.ini, en la variable de configuración memory\_limit.

```
memory_limit = 2048M
```

El valor de memory\_limit admite números enteros, en cuyo caso estás especificando una cantidad de Bytes. Pero también te admite terminar con una unidad ("K", "M", "G") indicando el valor en Kb, Mega o Giga.

## ¿Cuál es mi php.ini cuando trabajo por línea de comandos?

Es una pregunta típica. Si estás trabajando con Composer lo harás con el comando "composer" en tu terminal de sistema operativo. En ese caso es posible que estés trabajando con un php.ini diferente de tu PHP cuando trabaja sobre Apache.

Para saber el php.ini de línea de comandos (PHP CLI) tienes que hacer simplemente este comando.

```
php -i
```

Eso te mostrará una cantidad enorme de datos, parecidos a los que obtienes con phpinfo(). Tienes que buscar entre toda esa salida el texto "php.ini" para encontrar la ruta del archivo de configuración que se esté utilizando.

## Aumentar el tiempo del proceso de composer

Cuando invocas el comando composer de manera predeterminada se asigna un tiempo de procesamiento que son 300 segundos. Este tiempo puede ser insuficiente en caso que los paquetes a descargar sean pesados o en conexiones a Internet lentas. Si te ocurre esto recibirás un error que entre otras cosas tendrá este texto.

*[...] The process "git clone [...]" exceeded the timeout of 300 seconds*

En este caso puedes invocar al comando asignando un nuevo valor máximo en segundos para realizar las operaciones de actualización o instalación de las dependencias. Sería algo como:

```
COMPOSER_PROCESS_TIMEOUT=3600 composer install
```

En la propia documentación de Composer encontrarás una excelente [guía de resolución de problemas](#) (en inglés) en la que seguramente darás respuesta a muchas otras situaciones. Yo iré colocando en esta lista otros problemas que me vaya encontrando, de momento esto es todo! si tu problema no está en la lista mándanos la solución cuando la encuentres!

Este artículo es obra de *Miguel Angel Alvarez*  
Fue publicado por primera vez en 05/01/2015  
Disponible online en <http://desarrolloweb.com/articulos/problemas-soluciones-composer.html>