


 Subscribe Share Contents ▾

# How To Install and Use Composer on Ubuntu 14.04

Posted September 11, 2015

 701.6k

PHP

GETTING STARTED

UBUNTU



49

By: Erika Heidi

Not using **Ubuntu 14.04**? Choose a different version:

## Introduction

Composer is a popular dependency management tool for PHP, created mainly to facilitate installation and updates for project dependencies. It will check which other packages a specific project depends on and install them for you, using the appropriate versions according to the project requirements.

This tutorial will show how to install and get started with Composer on an Ubuntu 14.04 server.

## Prerequisites

For this tutorial, you will need:

- A server running Ubuntu 14.04
- Access to the server as a regular user with sudo permission

## Step 1 — Installing the Dependencies

Before we download and install Composer, we need to make sure our server has all dependencies installed.

First, update the package manager cache by running:

```
$ sudo apt-get update
```

Now, let's install the dependencies. We'll need `curl` in order to download Composer and `php5-cli` for installing and running it. `git` is used by Composer for downloading project dependencies. Everything can

```
$ sudo apt-get install curl php5-cli git
```

## Step 2 — Downloading and Installing Composer

```
$ curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --filename=composer
```

## Output

To test your installation, run:

\$ composer

## Output

[illegible]

Composer version 1.0-dev (9859859f1082d94e546aa75746867df127aa0d9e) 2015-08-17 14:57:00

Usage:

```
command [options] [arguments]
```

Options:

<code>--help (-h)</code>	Display this help message
--------------------------	---------------------------

```
--quiet (-q)          Do not output any message
--verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal output, 2 for more verbose ou
--version (-V)         Display this application version
--ansi                Force ANSI output
--no-ansi             Disable ANSI output
--no-interaction (-n) Do not ask any interactive question
--profile             Display timing and memory usage information
--working-dir (-d)     If specified, use the given directory as working directory.
```

. . .

This means Composer was successfully installed on your system.

If you prefer to have separate Composer executables for each project you might host on this server, you can simply install it locally, on a per-project basis. This method is also useful when your system user doesn't have permission to install software system-wide. In this case, installation can be done with `curl -sS https://getcomposer.org/installer | php` - this will generate a `composer.phar` file in your current directory, which can be executed with `php composer.phar [command]`.

## Step 3 — Generating the composer.json File

In order to use Composer in your project, you'll need a `composer.json` file. The `composer.json` file basically tells Composer which dependencies it needs to download for your project, and which versions of each package are allowed to be installed. This is extremely important to keep your project consistent and avoid installing unstable versions that could potentially cause backwards compatibility issues.

You don't need to create this file manually - it's easy to run into syntax errors when you do so. Composer auto-generates the `composer.json` file when you add a dependency to your project using the `require` command. Additional dependencies can also be added in the same way, without the need to manually edit this file.

The process of using Composer to install a package as dependency in a project usually involves the following steps:

- Identify what kind of library the application needs
- Research a suitable open source library on [Packagist.org](https://packagist.org), the official repository for Composer
- Choose the package you want to depend on
- Run `composer require` to include the dependency in the `composer.json` file and install the package

We'll see how this works in practice with a simple demo application.

The goal of this application is to transform a given sentence into a URL-friendly string - a *slug*. This is commonly used to convert page titles to URL paths (like the final portion of the URL for this tutorial).

Let's start by creating a directory for our project. We'll call it **slugify**:

```
$ cd ~  
$ mkdir slugify  
$ cd slugify
```

## Searching for Packages on Packagist

Now it's time to search [Packagist.org](https://packagist.org) for a package that can help us generating *slugs*. If you search for the term "slug" on Packagist, you'll get a result similar to this:

The screenshot shows the Packagist website interface. At the top, there's a dark navigation bar with the Packagist logo and the tagline 'The PHP Package Repository'. To the right of the logo are links for 'Browse', 'Submit', 'Create account', and 'Sign in'. Below the navigation bar is an orange header section containing the Packagist elephant logo, a search bar with the text 'slug', and two icons: a download arrow and a star. Below the search bar, a descriptive text states: 'Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer.'

The main content area displays a list of search results for the term 'slug'. Each result is presented in a white box with a light gray border. The results are as follows:

Package Name	Description	Language	Downloads	Stars
easy-slug/easy-slug	Laravel Package for creating a slug	PHP	835	12
muffin/slug	Slugging support for CakePHP 3	PHP	728	7
ddd/slug		PHP	6 127	6
zelenin/slug	Slug generation library	PHP	9	0
webcastle/slug	Laravel5 slug maker	PHP	29	0
anomaly/slug-field_type	A slug formatted string field type.	JavaScript	3 127	1

You'll see two numbers on the right side of each package in the list. The number on the top represents how many times the package was installed, and the number on the bottom shows how many times a package was starred on GitHub. You can reorder the search results based on these numbers (look for the two icons on the right side of the search bar). Generally speaking, packages with more installations and more stars tend to be more stable, since so many people are using them. It's also important to check the package description for relevance - is that really what you are looking for?

What we need is a simple string-to-slug converter. From the search results, the package `cocur/slugify` seems to be a good match, with a reasonable amount of installations and stars. (The package is a bit further down the page than the screenshot shows.)

You will notice that the packages on Packagist have a **vendor** name and a **package** name. Each package has a unique identifier (a namespace) in the same format Github uses for its repositories: `vendor/package`. The library we want to install uses the namespace `cocur/slugify` **The namespace is what we need in order to require the package in our project.**

## Requiring a Package

Now that we know exactly which package we want to install, we can run `composer require` to include it as a dependency and also generate the `composer.json` file for the project:

```
$ composer require cocur/slugify
```

### Output

```
Using version ^1.3 for cocur/slugify
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing cocur/slugify (v1.3)
  Downloading: 100%

Writing lock file
Generating autoload files
```

As you can see from the output, Composer automatically decided which version of the package should be used. If you check your project's directory now, it will contain two new files: `composer.json` and `composer.lock`, and a `vendor` directory:

```
$ ls -l
```

### Output

```
total 12
-rw-rw-r-- 1 sammy sammy  59 Sep  9 16:22 composer.json
-rw-rw-r-- 1 sammy sammy 2835 Sep  9 16:22 composer.lock
drwxrwxr-x 4 sammy sammy 4096 Sep  9 16:22 vendor
```

The `composer.lock` file is used to store information about which versions of each package are installed, and make sure the same versions are used if someone else clones your project and installs its dependencies. The `vendor` directory is where the project dependencies are located. The `vendor` folder should **not** be committed into version control - you only need to include the **`composer.json`** and **`composer.lock`** files.

When installing a project that already contains a `composer.json` file, you need to run `composer install` in order to download the project's dependencies.

## Understanding Version Constraints

If you check the contents of your `composer.json` file, you'll see something like this:

```
$ cat composer.json
```

composer.json

```
1 {
2     "require": {
3         "cocur/slugify": "^1.3"
4     }
5 }
```

You might notice the special character `^` before the version number on `composer.json`. Composer supports several different constraints and formats for defining the required package version, in order to provide flexibility while also keeping your project stable. The caret (`^`) operator used by the auto-generated `composer.json` file is the recommended operator for maximum interoperability, following semantic versioning. In this case, it defines **1.3** as the minimum compatible version, and allows updates to any future version below **2.0**.

Generally speaking, you won't need to tamper with version constraints in your `composer.json` file. However, some situations might require that you manually edit the constraints - for instance, when a major new version of your required library is released and you want to upgrade, or when the library you want to use doesn't follow semantic versioning.

Here are some examples to give you a better understanding of how Composer version constraints work:

Constraint	Meaning	Example Versions Allowed
<code>^1.0</code>	<code>&gt;= 1.0 &lt; 2.0</code>	1.0, 1.2.3, 1.9.9
<code>^1.1.0</code>	<code>&gt;= 1.1.0 &lt; 2.0</code>	1.1.0, 1.5.6, 1.9.9
<code>~1.0</code>	<code>&gt;= 1.0 &lt; 2.0.0</code>	1.0, 1.4.1, 1.9.9
<code>~1.0.0</code>	<code>&gt;= 1.0.0 &lt; 1.1</code>	1.0.0, 1.0.4, 1.0.9
<code>1.2.1</code>	<code>1.2.1</code>	1.2.1
<code>1.*</code>	<code>&gt;= 1.0 &lt; 2.0</code>	1.0.0, 1.4.5, 1.9.9
<code>1.2.*</code>	<code>&gt;= 1.2 &lt; 1.3</code>	1.2.0, 1.2.3, 1.2.9

For a more in-depth view of Composer version constraints, check their [official documentation](#).

## Step 4 — Including the Autoload Script

Composer also provides an autoload script that you can include in your project to get autoloading for free. This makes it much easier to work with your dependencies and define your own namespaces.

The only thing you need to do is include the `vendor/autoload.php` file in your PHP scripts, before any class instantiation.

Let's come back to the *slugify* example application. We'll create a `test.php` script where we'll use the *cocur/slugify* library:

```
$ vim test.php
```

test.php

```
1 <?php
2 require __DIR__ . '/vendor/autoload.php';
3
4 use Cocur\Slugify\Slugify;
5
6 $slugify = new Slugify();
7
8 echo $slugify->slugify('Hello World, this is a long sentence and I need to make a slug from it!')
```

You can run the script in the command line with:

```
$ php test.php
```

This should produce the output `hello-world-this-is-a-long-sentence-and-i-need-to-make-a-slug-from-it`.

## Step 5 — Updating the Project Dependencies

Whenever you want to update your project dependencies, you just need to run the `update` command:

```
$ composer update
```

This will check for newer versions of the libraries you required in your project. If a newer version is found and it's compatible with the version constraint defined in the `composer.json` file, it will replace the previous version installed. The `composer.lock` file will be updated to reflect these changes.

You can also update one or more specific libraries by running:

```
$ composer update vendor/package vendor2/package2
```

## Conclusion

Composer is a powerful tool every PHP developer should have in their utility belt.

Beyond providing an easy and reliable way for managing project dependencies, it also establishes a new de facto standard for sharing and discovering PHP packages created by the community.

This tutorial covered the essentials for getting started with Composer on Ubuntu 14.04.

By: Erika Heidi

♡ Upvote (49)

📄 Subscribe

🔗 Share



Editor:  
Sharon Campbell



We just made it easier for you to deploy faster.

[TRY FREE](#)

### Related Tutorials

How To Deploy a PHP Application with Kubernetes on Ubuntu 16.04

How To Set Up Laravel, Nginx, and MySQL with Docker Compose

How to Deploy a Symfony 4 Application to Production with LEMP on Ubuntu 18.04

How To Install and Use Composer on Debian 9

How To Install and Secure phpMyAdmin on Debian 9

## 7 Comments

Leave a comment...



Log In to Comment

^ [jamilalidrus95](#) November 21, 2015

0 i didnt find file vendor beside my json file

^ [erikaheidi](#) November 23, 2015

0 Hi there, can you give more details about your problem? The json file should be auto-generated when you run **composer require** , and a vendor folder will be created with the required vendor package.

^ [yozaira](#) May 29, 2016

0 Excellent tutorial. Thanks!

^ [tomlp14](#) July 12, 2016

0 Thank you, this tutorial was very useful. During the time I did the installation; i used this command:  
sudo apt-get install curl php7.0-cli git instead of sudo apt-get install curl php5-cli git. I've been currently using PHP 7.

^ [easybeau](#) July 13, 2016

0 Hello, I try to install baun cms on my computer. I installed Composer and i put the files in /media/www/public/baun. I also put composer.phar in it. When i go to browser localhost i have this message :

Baun Auto Installer

Attempting to download composer...

Composer is already installed on the system!

Which dependencies do i have to install?

Thanks for your answer.

^ [admin02e167ef9d9d2b1d0be09](#) May 22, 2017

0 I really wonder why Linux Distributions aren't just adding Composer as a default package (.deb). After all it has long become a standard in PHP development and should be available without having to pull the source via Curl.

0 Can anybody tell me about this?

```
/ _/ _ _ _ _ _ _ _ _  
/// _ V _ ` _ V _ V _ V _ / _ V _/  
// // // // // // // _ ( ) //  
_ _ // // // _ _ // _ _ //  
//
```

what is the name of it when we type composer in terminal and terminal shows it? and can we create own design like that and how we can?



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2019 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

---

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#) [Write for DOnations](#) [Shop](#)