

Como Instalar e Usar o Composer no Ubuntu 18.04

Posted October 22, 2018 17.8k views [PHP Ubuntu 18.04](#)



[Brian Hogan](#)

Uma versão anterior desse tutorial foi escrito por [Brennen Bearnes](#).

Introdução

O [Composer](#) é uma ferramenta popular de *gerenciamento de dependências* para o PHP, criado principalmente para facilitar a instalação e atualização para dependências de projeto. Ele verificará de quais outros pacotes um projeto específico depende e os instalará para você, usando as versões apropriadas de acordo com os requisitos do projeto.

Neste tutorial, você instalará e começará a usar o Composer em um sistema Ubuntu 18.04.

Pré-requisitos

Para completar este tutorial, você vai precisar de:

- Um servidor Ubuntu 18.04 configurado seguindo o [guia de Configuração Inicial de servidor com Ubuntu 18.04](#), incluindo um usuário sudo não-root e um firewall.

Passo 1 — Instalando as Dependências

Antes de você baixar e instalar o Composer, você deve certificar-se de que seu servidor tenha todas as dependências instaladas.

Primeiro, atualize o cache do gerenciador de pacotes executando:

```
sudo apt update
```

Agora, vamos instalar as dependências. Precisaremos do `curl` para que possamos baixar o Composer e o `php-cli` para instalação e execução dele. O pacote `php-mbstring` é necessário para fornecer funções para a biblioteca que estaremos utilizando. O `git` é utilizado pelo Composer para baixar as dependências de projeto, e o `unzip` para a extração dos pacotes compactados. Tudo pode ser instalado com o seguinte comando:

```
sudo apt install curl php-cli php-mbstring git unzip
```

Com os pré-requisitos instalados, podemos instalar o Composer propriamente dito.

Passo 2 — Baixando e Instalando o Composer

O Composer fornece um [instalador](#), escrito em PHP. Iremos baixá-lo, verificar que o mesmo não está corrompido, e então utilizá-lo para instalar o Composer.

Certifique-se de que você está em seu diretório home, em seguida baixe o instalador utilizando o `curl`:

```
cd ~
curl -sS https://getcomposer.org/installer -o composer-setup.php
```

A seguir, verifique se o instalador corresponde ao hash SHA-384 para o instalador mais recente encontrado na página [Composer Public Keys / Signatures](#). Copie o hash desta página e armazene-o em uma variável do shell.

```
HASH=544e09ee996cdf60ece3804abc52599c22b1f40f4323403c44d44fd586475ca9813a858088ffbc1f233e9180f061
```

Certifique-se de que você substituiu o hash mais recente para o valor destacado.

Agora, execute o seguinte script PHP para verificar que o script de instalação é seguro para execução:

```
php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
```

Você verá a seguinte saída:

Output

```
Installer verified
```

Se você vir `Installer corrupt`, então você vai precisar baixar novamente o script de instalação e conferir minuciosamente se você está utilizando o hash correto. Depois, execute o comando para verificar o instalador novamente. Uma vez que você tenha o instalador verificado, você pode continuar.

Para instalar o `composer` globalmente, utilize o seguinte comando que irá baixar e instalar o Composer como um comando disponível para todo o sistema chamado `composer`, dentro de `/usr/local/bin`:

```
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

Você verá a seguinte saída:

Output

```
All settings correct for using Composer
Downloading...
```

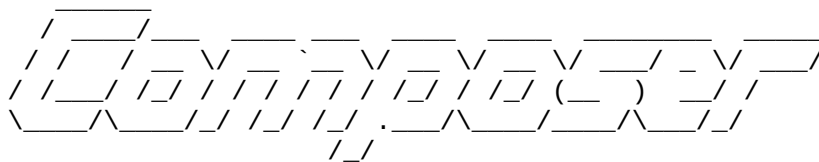
```
Composer (version 1.6.5) successfully installed to: /usr/local/bin/composer
Use it: php /usr/local/bin/composer
```

Para testar a sua instalação, execute:

```
composer
```

E você verá esta saída mostrando a versão e os argumentos do Composer.

Output



```
Composer version 1.6.5 2018-05-04 11:44:59
```

Usage:

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

Options:

| | |
|-------------------------------|--|
| -h, --help | Display this help message |
| -q, --quiet | Do not output any message |
| -V, --version | Display this application version |
| --ansi | Force ANSI output |
| --no-ansi | Disable ANSI output |
| -n, --no-interaction | Do not ask any interactive question |
| --profile | Display timing and memory usage information |
| --no-plugins | Whether to disable plugins. |
| -d, --working-dir=WORKING-DIR | If specified, use the given directory as working directory. |
| -v vv vvv, --verbose | Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug |
| . . . | |

Isso verifica que o Composer foi instalado com sucesso em seu sistema e está disponível de maneira global para todos.

Nota: se você preferir ter executáveis do Composer separados para cada projeto que você hospedar neste servidor, você poderá instalá-lo localmente, em uma base por projeto. Usuários do NPM estarão familiarizados com esta abordagem. Este método é útil também quando seu usuário de sistema não tem permissão para instalar software para todo o sistema.

Para fazer isto, utilize o comando `php composer-setup.php`. Isto irá gerar um arquivo `composer.phar` em seu diretório atual, que pode ser executado com `./composer.phar` comando.

Agora, vamos dar uma olhada em como utilizar o Composer para gerenciar dependências.

Passo 3 — Usando o Composer em um Projeto PHP

Projetos PHP frequentemente dependem de bibliotecas externas, e o gerenciamento dessas dependências e suas versões pode ser complicado. O Composer resolve isso rastreando suas dependências e facilitando a instalação delas por outras pessoas.

Para usar o Composer no seu projeto, você precisa de um arquivo `composer.json`. O arquivo `composer.json` diz ao Composer quais dependências ele precisa baixar para o seu projeto, e para quais versões de cada pacote está permitida a instalação. Isto é extremamente importante para manter o seu projeto consistente e evitar a instalação de versões instáveis que podem causar problemas de compatibilidade com versões anteriores.

Você não precisa criar este arquivo manualmente - é fácil cometer erros de sintaxe quando você faz isso. O Composer gera automaticamente o arquivo `composer.json` quando você adiciona uma

dependência para o seu projeto utilizando o comando `require`. Você pode adicionar dependências adicionais da mesma forma, sem a necessidade de editar manualmente este arquivo.

O processo de se utilizar o Composer para instalar um pacote como dependência em um projeto envolve os seguintes passos:

- Identificar qual tipo de biblioteca a aplicação precisa.
- Pesquisar uma biblioteca de código aberto adequada em [Packagist.org](https://packagist.org), o repositório de pacotes oficial para o Composer.
- Escolher o pacote do qual você quer depender.
- Executar `composer require` para incluir a dependência no arquivo `composer.json` e instalar o pacote.

Vamos tentar isso com uma aplicação de demonstração.

O objetivo dessa aplicação é transformar uma dada sentença em uma string com URL amigável - um *slug*. Isso é comumente usado para converter títulos de páginas em caminhos de URL (como a parte final da URL deste tutorial).


Vamos começar criando o diretório para nosso projeto. Vamos chamá-lo de **slugify**:

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

Agora é a hora de pesquisar no [Packagist.org](https://packagist.org) um pacote que possa nos ajudar a gerar *slugs*. Se você pesquisar pelo termo "slug" no Packagist, você obterá um resultado semelhante a esse:

Packagist The PHP Package Repository

Browse Submit Create account Sign In

 slug

Packagist is the main [Composer](#) repository. It aggregates public PHP packages installable with Composer.

| | | |
|---|------------|----------------|
| 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 |

Você verá dois números do lado direito de cada pacote na lista. O número no topo representa quantas vezes o pacote foi instalado, e o número embaixo mostra quantas vezes um pacote recebeu estrelas no [GitHub](#). Você pode reorganizar os resultados da pesquisa com base nesses números (procure os dois ícones no lado direito da barra de pesquisa). De um modo geral, pacotes com mais instalações e mais estrelas tendem a ser mais estáveis, já que muitas pessoas estão usando. Também é importante verificar a descrição do pacote quanto à relevância para garantir que é o que você precisa.

Precisamos de um conversor simples de string-para-slug. Nos resultados da pesquisa, o pacote `cocur/slugify` parece ser uma boa escolha, com uma quantidade razoável de instalações e estrelas. (O pacote está um pouco mais abaixo na página do que a imagem mostra.)

Os pacotes no Packagist tem um nome de **vendor** e um nome de **package** ou pacote. Cada pacote tem um identificador único (um namespace) no mesmo formato que o GitHub utiliza para seus repositórios, no formato `vendor/package`. A biblioteca que queremos instalar utiliza o namespace `cocur/slugify`. Você precisa do namespace para requerer o pacote em seu projeto.

Agora que você sabe exatamente quais pacotes você quer instalar, execute `composer require` para incluí-los como uma dependência e gere também o arquivo `composer.json` para o projeto:

```
composer require cocur/slugify
```

Você verá esta saída enquanto o Composer baixa a dependência:

```
Output
Using version ^3.1 for cocur/slugify
```

```
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing cocur/slugify (v3.1): Downloading (100%)
Writing lock file
Generating autoload files
```

Como você pode ver pela saída, o Composer decide automaticamente qual versão do pacote utilizar. Se você verificar o diretório do seu projeto agora, ele irá conter dois novos arquivos: `composer.json` e `composer.lock`, e um diretório `vendor`:

```
ls -l
```

```
Output
total 12
-rw-rw-r-- 1 sammy sammy  59 Jul 11 16:40 composer.json
-rw-rw-r-- 1 sammy sammy 2934 Jul 11 16:40 composer.lock
drwxrwxr-x 4 sammy sammy 4096 Jul 11 16:40 vendor
```

O arquivo `composer.lock` é usado para guardar informações sobre quais versões de cada pacote estão instaladas, e garantir que as mesmas versões sejam usadas se alguém clonar seu projeto e instalar suas dependências. O diretório `vendor` é onde as dependências do projeto estão localizadas. A pasta `vendor` não precisa ter o commit realizado no controle de versão - você só precisa incluir os arquivos **`composer.json`** e **`composer.lock`**.

Ao instalar um projeto que já contém um arquivo `composer.json`, execute `composer install` para fazer o download das dependências do projeto.

Vamos dar uma olhada rápida nas restrições de versão. Se você verificar o conteúdo do seu arquivo `composer.json`, verá algo como isto:

```
cat composer.json
```

```
Output
{
    "require": {
        "cocur/slugify": "^3.1"
    }
}
sam
```

Você deve ter notado o caractere especial `^` antes do número da versão no `composer.json`. O Composer suporta várias restrições e formatos diferentes para definir a versão do pacote necessária, a fim de fornecer flexibilidade e, ao mesmo tempo, manter seu projeto estável. O operador cirunflexo (`^`) utilizado pelo arquivo auto-gerado `composer.json` é o operador recomendado para a máxima interoperabilidade, seguindo o [versionamento semântico](#). Nesse caso, ele define **3.1** como a versão mínima compatível, e permite atualizar para quaisquer versões futuras abaixo da **4.00**.

De modo geral, você não precisará adulterar restrições de versão em seu arquivo `composer.json`. Contudo, algumas situações podem exigir que você edite manualmente as restrições - por exemplo, quando uma nova versão principal de sua biblioteca obrigatória é lançada

e você deseja atualizar, ou quando a biblioteca que você deseja usar não segue o versionamento semântico.

Aqui estão alguns exemplos para entender melhor como funcionam as restrições de versão do Composer:

| Restrição | Significado | Versões de Exemplo Permitidas |
|---------------------|-----------------------------------|-------------------------------|
| <code>^1.0</code> | <code>>= 1.0 < 2.0</code> | 1.0, 1.2.3, 1.9.9 |
| <code>^1.1.0</code> | <code>>= 1.1.0 < 2.0</code> | 1.1.0, 1.5.6, 1.9.9 |
| <code>~1.0</code> | <code>>= 1.0 < 2.0.0</code> | 1.0, 1.4.1, 1.9.9 |
| <code>~1.0.0</code> | <code>>= 1.0.0 < 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>>= 1.0 < 2.0</code> | 1.0.0, 1.4.5, 1.9.9 |
| <code>1.2.*</code> | <code>>= 1.2 < 1.3</code> | 1.2.0, 1.2.3, 1.2.9 |

Para uma visão mais aprofundada das restrições de versão do Composer, consulte [a documentação oficial](#).

Em seguida, vamos ver como carregar dependências automaticamente com o Composer.

Passo 4 — Incluindo o Script Autoload

Como o próprio PHP não carrega classes automaticamente, o Composer fornece um script de carregamento automático que você pode incluir em seu projeto para obter o carregamento automático de graça. Isso facilita muito o trabalho com suas dependências.

A única coisa que você precisa fazer é incluir o arquivo `vendor/autoload.php` em seus scripts PHP antes de qualquer instanciamento de classe. Este arquivo é gerado automaticamente pelo Composer quando você adiciona sua primeira dependência.

Vamos experimentar isso em nossa aplicação. Crie o arquivo `test.php` e abra-o em seu editor de textos:

```
nano test.php
```

Adicione o seguinte código que traz o arquivo `vendor/autoload.php`, carrega a dependência `cocur/sluggify`, e a utiliza para criar um slug:

```
test.php
```

```
<?php
require __DIR__ . '/vendor/autoload.php';

use Cocur\Sluggify\Sluggify;

$slugify = new Sluggify();

echo $slugify->slugify('Hello World, this is a long sentence and I need to make
a slug from it!');
```

Salve o arquivo e saia do editor.

Agora, execute o script:

```
php test.php
```

Isso produz a saída `hello-world-this-is-a-long-sentence-and-i-need-to-make-a-slug-from-it`.

Dependências precisam de atualizações quando novas versões são lançadas, então vamos ver como lidar com isso.

Passo 5 — Atualizando as Dependências de Projeto

Sempre que você quiser atualizar suas dependências de projeto para versões mais recentes, execute o comando `update`:

```
composer update
```

Isso irá verificar as versões mais recentes das bibliotecas necessárias em seu projeto. Se uma versão mais nova for encontrada e for compatível com a restrição de versão definida no arquivo `composer.json`, o Composer substituirá a versão anterior instalada. O arquivo `composer.lock` será atualizado para refletir estas mudanças.

Você também pode atualizar uma ou mais bibliotecas específicas, especificando-as assim:

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

Certifique-se de verificar seus arquivos `composer.json` e `composer.lock` depois de atualizar suas dependências para que outras pessoas possam instalar essas versões mais novas.

Conclusão

O Composer é uma ferramenta poderosa que todo desenvolvedor de PHP deve ter em seu cinto de utilidades. Neste tutorial você instalou o Composer e o utilizou em um projeto simples. Agora você sabe como instalar e atualizar dependências.

Além de fornecer uma maneira fácil e confiável de gerenciar dependências de projetos, ele também estabelece um novo padrão de fato para compartilhar e descobrir pacotes PHP criados pela comunidade.

Brian Hogan