olegf13 27 мая 2015 в 11:44

Освоение Composer: советы и приемы использования

PHP, Разработка веб-сайтов

Из песочницы

Предлагаю читателям «Хабрахабра» перевод статьи «Mastering Composer – Tips and Tricks» за авторством Bruno Skvorc.

Composer произвел революцию в управлении пакетами в PHP и помог разработчикам по всему миру создавать независимый от фреймворков и разделяемый код. Но все же мало кто выходит за рамки основ его функционала, так что данная статья постарается осветить некоторые полезные приемы его использования.

Глобальная установка (Global)

Несмотря на то, что данная опция доступно описана в документации, Composer может (а в большинстве случаев должен) быть установлен глобально. Глобальная установка означает, что вместо:

```
php composer.phar somecommand
```

Вы можете в любом проекте просто ввести:

```
composer somecommand
```

Это позволяет очень просто создавать новые проекты (например, с помощью команды **create-project**) в любом месте вашей файловой системы.

Для установки Composer глобально, следуйте этим инструкциям.

Правильная установка зависимостей

При чтении вводных инструкций или README файлов многие напишут вам что-то вроде:

> Just add the following to your composer.json **file**:
> `{ "require": { "myproject": "someversion" } }`

Но данный подход имеет несколько недостатков. Во-первых, простой копипаст может привести к возникновению ошибок. Во-вторых, для новичка может быть не очевидно, где разместить данный код,

ЧИТАЮТ СЕЙЧАС

Развёрнутый ответ на комментарий, а также немного о жизни провайдеров в РФ
13,7k 84

MySpace потерял музыку, фото и видео, которые пользователи загружали с 2003 по 2015 годы
47,7k 273

Верните мне мой монолит
25k 64

Новое мобильное приложение LampTest.ru
4,2k 14

Программист на больничном
161k 192

Что курил конструктор: необычное огнестрельное оружие
187k 115

если у него и так уже имеется обширный файл **composer.json**, и это также привести к ошибке. Наконец, некоторые люди будут иметь дело с Composer впервые, а возможно и впервые столкнутся с командной строкой. Поэтому хорошей практикой будет освещение всевозможных случаев, в которых новички могут чувствовать себя неуверенно (есть ли у них графический редактор или они будут использовать командную строку? Если второе, установлен ли в ней текстовый редактор, и если да, то какой? Объясняете ли вы саму процедуру редактирования файла? А что если файла composer.json ещё не существует в проекте? Описываете ли также принцип создания нового файла?).

Наилучший способ добавить новую зависимость в файл **composer.json** — это воспользоваться командой **require**:

```
composer require somepackage/somepackage:someversion
```

Это добавит все необходимое в файл зависимостей, минуя ручное вмешательство.

Если вам нужно добавить пакеты в раздел **require-dev**, добавьте в команду опцию **--dev**:

```
composer require phpunit/phpunit --dev
```

Также, команда **require** поддерживает добавление нескольких пакетов одновременно, для этого просто разделите их пробелом.

Файлы блокировок

Файл **composer.lock** сохраняет текущий список установленных зависимостей и их версии. Таким образом, на момент, когда версии зависимостей уже будут обновлены, другие люди, которые будут клонировать ваш проект, получат те же самые версии. Это позволяет убедиться в том, что каждый, кто получает ваш проект, имеет “пакетное окружение”, идентичное тому, которое вы использовали при разработке, и помогает избежать ошибок, которые могли бы возникнуть из-за обновления версий.

Файл **composer.lock** почти всегда должен быть добавлен в систему контроля версий (не всегда).

Также, файл **composer.lock** содержит хэш файла **composer.json**, так что, если вы даже просто обновляете данные об авторе проекта, вы получите предупреждение о том, что файл блокировки не соответствует **.json** файлу. В таком случае, поможет команда **composer update --lock**, которая обновит только сам файл блокировки, не касаясь ничего другого.

Версионирование

При указании допустимых версий пакетов можно использовать точное соответствие (**1.2.3**), диапазоны с операторами сравнения (**<1.2.3**), комбинации этих операторов (**>1.2.3 <1.3**), “последняя доступная” (**1.2.***), символ тильды (**~1.2.3**) и знак вставки (**^1.2.3**).

Последние два указания достойны отдельного объяснения:

- указание тильды (**~1.2.3**) будет включать в себя все версии до **1.3** (не включительно), так как в семантическом версионировании это является моментом внедрения новых функциональных возможностей. В данном случае будет получена последняя из стабильных минорных версий. Как гласит документация, при данном указании изменяться может **только** последняя цифра версии.

- указание знака вставки (**^1.2.3**) буквально означает “опасаться только критических изменений” и будет включать в себя версии вплоть до **2.0**. Применительно к semver, изменение мажорной версии является моментом внесения в проект критических изменений, так что версии **1.3**, **1.4** и **1.9** подходят, в то время как **2.0** — уже нет.

Кроме случая, когда вы знаете, что вам нужна конкретная версия пакета, я рекомендую всегда использовать формат **~1.2.3** — это самый безопасный выбор.

Локальная и глобальная конфигурация

Значения параметров по-умолчанию не высечены на камне. Подробное описание возможных параметров конфигураций (**config**) см. по ссылке.

Например, указав:
`{ "config": { "optimize-autoloader": true } }`

вы заставляете Composer оптимизировать classmap после каждой установки или обновления пакетов (или, другими словами, всякий раз, когда генерируется файл автозагрузки классов). Это немного медленнее, чем создание автозагрузчика по-умолчанию, и замедляется при росте проекта.

Ещё одним полезным параметром может быть **cache-files-maxsize**. В больших проектах (как eZ Publish или Symfony) кэш может заполниться довольно быстро. Увеличение размера кэша позволить Composer работать быстро дольше.

Обратите внимание, что параметры конфигурации могут быть установлены глобально, и в таком случае будут действовать на все проекты (см. config). Например, чтобы глобально установить параметр размера кэша, нужно либо отредактировать файл **~/.composer/config.json**, либо выполнить:

```
composer config --global cache-files-maxsize "2048MiB"
```

Профилирование и подробный вывод (verbose)

Если вы добавите параметр **--profile** к любой команде при использовании Composer в командной строке, то в выводе будет содержаться не только конечный результат, например:

```
[174.6MB/54.70s] Memory usage: 174.58MB (peak: 513.47MB), time: 54.7s
```

Но также в начало каждой строки вывода будет добавлено время выполнения команды и использованный размер памяти:

```
[175.9MB/54.64s] Installing assets for Sensio\Bundle\DistributionBundle into web/bundles/sen
siodistribution
```

Я использую данную опцию для определения “медленных” пакетов и для наблюдения за улучшением или ухудшением производительности на разных версиях PHP.

Подобно предыдущему, параметр **--verbose** заставит Composer выводить больше информации о *каждой* выполняемой операции, давая вам понять, что *именно* происходит в данный момент. Некоторые люди даже устанавливают **composer --verbose --profile** псевдонимом команды **composer** по-умолчанию.

Пользовательские источники

Если ваш проект ещё не на Packagist, иногда вам нужно просто установить пакет с GitHub (например, если пакет ещё находится в разработке). Для этого см. наше руководство.

Когда у вас есть *своя* версия популярного пакета, от которого зависит ваш проект, вы можете использовать пользовательские источники в сочетании с контекстными псевдонимами (inline aliasing), чтобы подставить собственную ветку для публичного пакета, как здесь описал Matthieu Napoli.

Ускорение Composer

Используя отличный метод, описанный Mark Van Eijk, вы можете ускорить выполнение Composer, вызывая его через HHVM.

Ещё один способ — с помощью параметра **--prefer-dist**, при установке которого Composer будет скачивать стабильные, запакованные версии проекта, вместо клонирования из системы контроля версий (что значительно медленнее). Этот параметр используется по-умолчанию, так что вам не нужно включать его на стабильных проектах. Если вам нужно загрузить проект из исходников, используйте параметр **--prefer-source**. Подробнее об этом можно узнать в разделе **install** здесь.

Уменьшение размера проекта Composer

Если вы разработчик «Composer-friendly» проектов, данная часть вас также заинтересует. По этому посту в Reddit, вы можете с помощью файла **.gitattributes** игнорировать некоторые файлы и папки во время упаковки пакета для режима **--prefer-dist**.

```
/docs export-ignore /tests export-ignore /.gitattributes export-ignore /.giti
gnore export-ignore /.travis.yml export-ignore /phpunit.xml export-ignore
```

Как это работает? Когда вы загружаете проект на GitHub, он автоматически делает доступным ссылку *“Download zip”*, с помощью которой вы можете скачать архив вашего проекта. Более того, Packagist использует эти автоматически сгенерированные архивы для скачивания зависимостей с опцией **--prefer-dist**, которые он затем локально разархивирует (намного быстрее клонирования исходных файлов проекта). Если вы при этом добавите в **.gitattributes** тесты, документацию и прочие файлы, не имеющие отношения к логике работы проекта, указанные архивы не будут их содержать, став гораздо легче.

При этом людям, которые захотят отладить вашу библиотеку или запустить тесты, нужно будет указать параметр **--prefer-source**.

PhpLeague приняла этот подход и включила его в свой *«скелет пакета»* (Package skeleton), так что любой основанный на нем проект автоматически будет *“dist friendly”*.

Show

Если вы вдруг забыли, какую версию PHP или его расширения используете, или вам нужен список всех установленных проектов (с описанием каждого) с их версиями, вы можете использовать команду **show** с параметрами **--platform** (**-p**) и **--installed** (**-i**):

composer show --installed

Репетиции (Dry Runs)

Чтобы просто посмотреть, пройдет ли установка новых зависимостей успешно, вы можете использовать параметр **--dry-run** для команд **install** и **update**. Composer в данном случае выведет все потенциальные проблемы без непосредственного выполнения самой команды. Никаких реальных изменений в проекте не произойдет. Этот прием отлично подходит для тестирования сложных зависимостей и настройки изменений перед реальным их внесением.

```
composer update --dry-run --profile --verbose
```

Создание проекта

Последнее, но не менее важное, что мы должны упомянуть — это команда **create-project**.

Команда создания проекта принимает в качестве аргумента имя пакета, который она затем клонирует и выполняет **composer install** внутри него. Это отлично подходит для инициализации проектов — не нужно больше искать Url нужного пакета на GitHub, клонировать его, самому переходить в папку и выполнять команду **install**.

Крупные проекты, такие как Symfony и Laravel, уже используют данный подход для инициализации своих «skeleton» приложений, и многие другие также присоединяются.

Например, в Laravel это используется следующим образом:

```
composer create-project laravel/laravel --prefer-dist --profile --verbose
```

В команду **create-project** можно передать еще два параметра: *путь*, в который нужно установить проект (если не указан, используется имя пакета), и *версия* (будет использована последняя, если не указать).

Заключение

Надеюсь, данный список советов и приемов использования оказался для вас полезным. Если мы что-то упустили, расскажите нам об этом, и мы обновим статью. И помните, если вы забыли какие-либо команды или опции, просто загляните в шпаргалку. **Happy Composing!**

Теги: composer, php

Олег Федоров @olegf13
Пользователь

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

15 декабря 2014 в 09:54
Composer. Небезопасно использовать packagist и приватный источник пакетов одновременно
+6 5,9k 26 5

26 января 2014 в 23:27
Дайджест интересных новостей и материалов из мира PHP № 34 (13 — 26 января 2014)
+43 22,8k 154 32

17 июня 2012 в 11:12
Composer — менеджер зависимостей для PHP
+47 348k 725 43

Комментарии 4

symbix 27 мая 2015 в 14:58 +2

Часто слышу аргумент против composer, что, дескать, ставит в зависимость от сторонних бесплатных ресурсов без SLA — не все знают, что можно поднять свою копию Packagist. Он доступен на гитхабе — github.com/composer/packagist

olegf13 27 мая 2015 в 15:35 0

> ставит в зависимость от сторонних бесплатных ресурсов

Вообще нет, сторонние репозитории будут игнорироваться Composer'ом (см. root-only). Чтобы добавить возможность брать *зависимости зависимостей* НЕ из Packagist, нужно добавить сторонние репозитории в главный .json файл.

Или я неправильно вас понял?

rhamdeew 27 мая 2015 в 16:14 **–1**

Спасибо! Все никак руки не доходили до более детального изучения этого прекрасного инструмента!

glagola 28 мая 2015 в 15:00 **+2**

> Крупные проекты, такие как Symfony и Laravel, уже используют данный подход для инициализации своих «skeleton» приложений, и многие другие также присоединяются.

Yii2 также использует этот подход

> composer create-project --prefer-dist yiisoft/yii2-app-advanced yii-application

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки | Неделя | Месяц

MySpace потерял музыку, фото и видео, которые пользователи загружали с 2003 по 2015 годы
+78 47,7k 35 273

Разработан метод шумоизоляции, гасящий до 94% шумов — рассказываем, как он работает
+30 44,2k 113 109

Как мы купили дом с солнечными панелями, и что из этого вышло
+61 45,6k 66 194

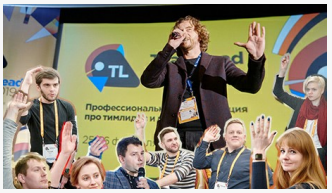
Верните мне мой монолит
+45 25k 69 64

Что курил конструктор: необычное огнестрельное оружие
+100 187k 98 115

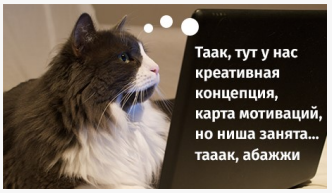
Верните мне мой монолит
25k 64

Facebook отказывается переносить серверы в страны с нарушением свободы слова и вводит сквозное шифрование
37,4k 471

ХАБР РЕКОМЕНДУЕТ Разместить



Организовать и возглавить: полезные кейсы с TeamLead Conf 2019



Онлайн-курсы школы ИКРА — это практические программы, нужные для работы прямо сейчас