5 features to know about Composer PHP

14 Mar 2013

Here are some tips you should know when using Composer the dependency manager for PHP.

1. Update only one vendor

You want to update only one specific library, without updating all your dependencies? That's easy, just add the vendor name as argument to the `update` command:

```
$ composer update foo/bar
```

This will only install or update the library (plus its dependencies) and overwrite the `composer.lock`.

Nevertheless, this is also very usefull to fix a warning message I'm sure you know very well:

```
Warning: The lock file is not up to date with the latest changes in composer.json,
you may be getting outdated dependencies, run update to update them.
```

« Damn it composer, what's wrong with my vendors? ». Don't panic! This is what you should expect if you just have edited the `composer.json` file. For instance, if you **add or update a detail** like the library description, authors, extra parameters,

or even put a trailing whitespace, this will change the **md5sum** of the file. Then Composer will warn you if this hash differs from the one stored in the `composer.lock`.

```
{
    "hash": "0bcd1234f87401a9669eb5264b8e32a7",
    "packages": [
        "..."
    ]
}
```

Ok, so how to proceed? The `update` command is the one which update the lock file. But if I just add a description, I may not want to update any library. In that case use the `--lock` parameter.

```
$ composer update --lock
```

Thus Composer won't upgrade your vendors. It will **only update the lock file hash** to suppress the warning. However it may also install the packages you have not yet installed (if you have not done the `install` command previously), but no package upgrade is performed.

Note that trying to update a vendor that doesn't exist will have the same effect.

```
$ composer update foobar
```

2. Add a library without editing your composer.json

To add a new vendor for your project, you can manually add a new line into your `composer.json` then use the previous method to **only** install/update this vendor. But there is a much convenient way to proceed.

The `require` command will make those actions for you:

```
$ composer require "foo/bar:1.0.0"
```

If you omit to specify the version Composer will **fetch the last stable**.

This action can also be used to quickly start a new project. The `init` command include the `--require` option which will write the `composer.json` file with the given vendor(s) (note the `-n` option to not ask any interactive question):

```
$ composer init --require="foo/bar:1.0.0" -n
$ cat composer.json
{
    "require": {
        "foo/bar": "1.0.0"
    }
}
```

3. Easy fork

Speaking about `composer.json` initialization, did you ever used the `create-project` command?

```
$ composer create-project doctrine/orm path 2.2.0
```

This will automatically clone the repository and checkout the given version. That may be usefull to quickly clone a library without searching the original URI of the sources.

4. Prefer dist packages and cache them

Did you know that **dist packages are now cached** in your home directory?

I previously wrote an article to compare solutions for caching vendors between multiple projects to save time when cloning a repository. I even created my own

command to clone vendors into your global composer configuration to reuse it when necessary.

Since November 2012, Composer automatically stores the archive when you download a dist package. By default, dist packages are used for tagged versions, for instance `"symfony/symfony": "v2.1.4"`, or even a wildcard or range version like `"2.1.*"` or `">=2.2,<2.3-dev"` if you use `stable` as your `minimum-stability`.

But dist packages can also work with branches (ie. `dev-master`) as Github allows to download an archive from a git reference. To force downloading archive instead of cloning sources, use the `--prefer-dist` option included in the `install` and `update` command.

Here is a demonstration (I use the `--profile` option to show the execution time):

```
$ composer init --require="twig/twig:1.*" -n --profile
Memory usage: 3.94MB (peak: 4.08MB), time: 0s

$ composer install --profile
Loading composer repositories with package information
Installing dependencies
  - Installing twig/twig (v1.12.2)
    Downloading: 100%

Writing lock file
Generating autoload files
Memory usage: 10.13MB (peak: 12.65MB), time: 4.71s

$ rm -rf vendor

$ composer install --profile
Loading composer repositories with package information
Installing dependencies from lock file
  - Installing twig/twig (v1.12.2)
    Loading from cache

Generating autoload files
Memory usage: 4.96MB (peak: 5.57MB), time: 0.45s
```

Here, the archive for `twig/twig:1.12.2` has been stored into `~/.composer/cache/files/twig/twig/1.12.2.0-v1.12.2.zip` and used the second time I reinstalled the package.

5. Prefer source to edit your vendors

For practical reasons, you may prefer cloning sources instead of downloading packages. For instance, this can be useful to edit a library directly in the vendor directory to test the behaviour of your application with that change (eg. a bug fix). The `--prefer-source` option will force cloning sources instead of downloading an archive:

```
$ composer update symfony/yaml --prefer-source
```

Then to see modified files in your vendor:

```
$ composer status -v
You have changes in the following dependencies:
/path/to/app/vendor/symfony/yaml/Symfony/Component/Yaml:
    M Dumper.php
```

Composer will also tells you when you try to update a vendor that has been modified, and asks if you want to discard the changes:

```
$ composer update
Loading composer repositories with package information
Updating dependencies
  - Updating symfony/symfony v2.2.0 (v2.2.0- => v2.2.0)
    The package has modified files:
    M Dumper.php
    Discard changes [y,n,v,s,?]?
```

Be ready for production

Just a reminder, before deploying your code in production, don't forget to optimize the autoloader:

```
$ composer dump-autoload --optimize
```

This can also be used while installing packages with the `--optimize-autoloader` option. Without that optimization, you may notice a performance loss from 20 to 25%.

Of course, if you need any help or details for a command, read to the official documentation or watch this awesome interactive cheatsheet made by JoliCode.

Share this post

Comments for this thread are now closed. ×

32 Comments MattKetmo Blog 1 Login

Recommend 11 Share Sort by Best

michaelmd • 2 years ago

guessing this is some mysterious IDE-based way of making things a lot more complicated than they need to be.
Could somebody explain why there are so many new package managers in just about everything in recent years ? .. and they never have any explanation of WHY? ..and no examples that actually work and endless hours banging head on brick walls just trying to load anything .. can't anyone come up with a a simple and clearly explained standard and stick with it for more than a year or so so we don't end up spending all our time on endless upgrades and updates trying to stop things from breaking and find ourselves with no time left to do anything else?

what happened to KISS?

ok granted there were a lot of problems with pear paths on different systems but at least with pear if it didn't work properly and it was pure php you could still include the files directly ., but this is now so convoluted that you can't even grep -r it to find anything!

(rather frustrated after an afternoon trying to get something installed via composer to work - maybe if something actually would explain the thinking behind it it might be possible to make some sense of it?)

(btw this isn't a rant about symfony or any other platform .. just this trend towards unexplained complexity