10 Things You Didn't Know You Could Do With Composer

Patrick Schwisow

Cascadia PHP Conference 2018

@PSchwisow

10 Things You May Already Know...

1. Dependency Resolution
2. Install Libraries
3. Check PHP Version
4. Check for Extensions
5. Updating Libraries
6. Works with Packagist
7. Autoloading
8. Lock Library Versions
9. Minimum Stability
10. Use Named Branches

1. Optimize Autoloader

- PSR-0 / PSR-4 autoloading may require many calls to `file_exists` (SLOW).
- Classmaps are faster, but require manual updates.
- Composer to the rescue!!!

1. Optimize Autoloader

- Composer can generate a classmap for all PSR-0/4 classes.
- If not in classmap, fallback to PSR-0/4 autoloader.

```
composer dump-autoload --optimize
composer install --optimize-autoloader
```

1. Optimize Autoloader

- To turn off PSR-0/4 fallback:

```
composer dump-autoload
                --classmap-authoritative
composer install --classmap-authoritative
```

2. Override Autoloading

- Autoloader is built from a sorted list of autoload mappings from `composer.json` files in the project and all dependencies.
- Autoload mappings in main `composer.json` take precedence.

Example 2-1: Override Autoloading in composer.json

```
{
    "require": {
        "phergie/phergie-irc-bot-react": "~2",
        "pschwisow/irc-plugin-react-karma": "~2"
    },
    "autoload": {
        "psr-4": {
            "PSchwisow\\Plugin\\Karma\\": "/code/karma/src"
        }
    }
}
```


Example 2-2: vendor/composer/autoload_psr4.php

```php
<?php
// autoload_psr4.php @generated by Composer

$vendorDir = dirname(dirname(__FILE__));
$baseDir = dirname($vendorDir);

return array(
    'Phergie\\Irc\\Bot\\React\\' => array(
        $vendorDir . '/phergie/phergie-irc-bot-react/src'
    ),
    'PSchwisow\\Plugin\\Karma\\' => array(
        $baseDir . '/../karma/src',
        $vendorDir . '/pschwisow/irc-plugin-react-karma/src'
    ),
);

```


3. Inline Aliases

- Inline aliases tell Composer to use one version of a package as if it were another version.

Example 3-1: Before Inline Alias

```
{
    "require": {
        "phergie/phergie-irc-bot-react": "dev-bugfix",
        "pschwisow/irc-plugin-react-karma": "~2"
    }
}

Your requirements could not be resolved to an installable
set of packages.
  Problem 1
    - pschwisow/irc-plugin-react-karma 2.0.0 requires
phergie/phergie-irc-bot-react ~2 -> satisfiable by
phergie/phergie-irc-bot-react [2.0.0, 2.0.1] but these
conflict with your requirements or minimum-stability.
    - Installation request for pschwisow/irc-plugin-react-
karma 2.0.0 -> satisfiable by pschwisow/irc-plugin-react-
karma [2.0.0].
```

Example 3-2: With Inline Alias

```
{
  "require": {
    "phergie/phergie-irc-bot-react": "dev-bugfix as 2.0.1",
    "pschwisow/irc-plugin-react-karma": "~2"
  }
}
```

Use most recent version listed in composer.lock

4. Beyond Require

- require-dev – Do not install if `--no-dev` flag is set.

Example:

```
"require-dev": {
    "phpunit/phpunit": "4.5.*",
    "phake/phake": "2.0.0-beta2",
    "friendsofphp/php-cs-fixer": "~1"
},
```

4. Beyond Require

- conflict – This package cannot be used together with the specified package.

Example: Phergie client now handles "pong" functionality internally. If you use it together with plugin, things break.

```
"conflict": {
    "phergie/phergie-irc-plugin-react-pong": "*"
},
```

4. Beyond Require

- replace – This package can be used in place of the specified package.

Example: You fork a dependency to fix bugs, but you need other dependencies to treat it as the original.

```
"replace": {
    "other/library": "1.2.3"
},
```

4. Beyond Require

- provide – This package provides an implementation of the specified package.

Example: PSR-3 defines a logger interface. Packages can require `psr/log-implementation`. Multiple loggers could satisfy this requirement.

```
"provide": {
    "psr/log-implementation": "1.0.0"
},
```

4. Beyond Require

- suggest – When installing, provide suggested additional packages.
  - Values are text, rather than version constraints.

Example:

```
  "suggest": {
      "monolog/monolog": "Allows more advanced logging
of the application flow",
      "ext-xml": "Needed to support XML format in Foo"
  },
```

5. Non-Packagist Repos

- Repository types:
  - Composer – Satis or Private Packagist
  - VCS – git, Subversion, Mercurial, Fossil
  - PEAR
  - Package – Supports libraries without their own `composer.json`

Example 5-1: VCS Repositories

```
{
  "require": {
    "vendor/my-private-repo": "dev-master",
    "vendor/project-a": "~2"
  },
  "repositories": [
    {
      "type": "vcs",
      "url": "git@bitbucket.org:vendor/my-private-repo.git"
    },
    {
      "type": "vcs",
      "url": "http://svn.example.org/projectA/",
    }
  ]
}
```


Example 5-2: Package Repositories

```
{
  "require": { "smarty/smarty": "3.1.*" },
  "repositories": [ {
    "type": "package",
    "package": {
        "name": "smarty/smarty",
        "version": "3.1.7",
        "dist": {…},
        "source": {
            "url": "http://smarty-php.googlecode.com/svn/",
            "type": "svn",
            "reference": "tags/Smarty_3_1_7/distribution/"
        },
        "autoload": { "classmap": ["libs/"] }
    }
  } ]
}
```


6. dist vs. source

- Dist – a packaged version (usually a ZIP)
- Source – clone / checkout from VCS
- Packages supply one or both of these.
- If both are available, default behavior:
  - For stable versions, use dist
  - Otherwise, use source

6. dist vs. source

- Pros for dist:
  - Faster installs
  - Archives are cached

```
composer install --prefer-dist
composer update --prefer-dist
```

6. dist vs. source

- Pros for source:
  - Faster updates (git fetch + git checkout)
  - Vendors are git repos, can be edited

```
composer install --prefer-source
composer update --prefer-source
```

7. Vendor Binaries

- **Vendor binary** – "Any command line script that a Composer package would like to pass along to a user…"
- Composer creates symlinks to these files in `vendor/bin` directory.

7. Vendor Binaries

Example:

- phergie/phergie-irc-bot-react contains a script "phergie" in its `bin` directory.
- Its `composer.json` contains:

```
"bin": [
    "bin/phergie"
]
```

- My installation of Phergie requires the bot package.
- Composer install creates a symlink:

```
vendor/bin/phergie ->
    vendor/phergie/phergie-irc-bot-react/bin/phergie
```

8. Scripts and Extra

- **Script** – PHP static method or CLI command executed by composer during its work.
- **Extra** – arbitrary data that can be used by scripts.

8. Scripts and Extra

- Commonly-used events:
  - pre-install-cmd / post-install-cmd
  - pre-update-cmd / post-update-cmd
  - pre-autoload-dump / post-autoload-dump
  - post-create-project-cmd
  - pre-dependencies-solving / post-dependencies-solving
- Complete list: https://getcomposer.org/doc/articles/scripts.md#event-names

Example 8-1: Calling Scripts

```
{
    "scripts": {
        "post-install-cmd": [
            "MyVendor\\MyClass::warmCache",
            "phpunit -c app/"
        ],
        "post-autoload-dump": [
            "MyVendor\\MyClass::postAutoloadDump"
        ],
        "post-create-project-cmd": [
            "php -r \"copy('config/local-example.php',
'config/local.php');\""
        ]
    },
    "extra": { "foo": "bar" }
}
```


Example 8-2: Script Definitions

```php
<?php
namespace MyVendor;
use Composer\Script\Event;

class MyClass
{
    public static function postAutoloadDump(Event $event)
    {
        $extra = $event->getComposer()
            ->getPackage()->getExtra();
        $vendorDir = $event->getComposer()
            ->getConfig()->get('vendor-dir');
        require $vendorDir . '/autoload.php';

        some_function_from_autoloaded_file($extra['foo']);
    }
}
```

9. Informational Tools

- search – keyword search through current repositories (usually Packagist)

```
$ composer search phergie
phergie/phergie-irc-connection Data structure for
containing information about an IRC client
connection
phergie/phergie-irc-client-react IRC client library
built on React
phergie/phergie-irc-bot-react IRC bot built on React
phergie/phergie-irc-plugin-react-command Phergie
plugin for parsing commands issued to the bot
```

9. Informational Tools

- show – Displays currently installed packages.

```
composer show
```

Lists all packages.

```
composer show pschwisow/*
```

Lists all packages under pschwisow vendor.

```
composer show pschwisow/foo
```

Shows detailed information about one package.

9. Informational Tools

- why (depends) – Shows which packages depend on the specified package.

```
$ composer why doctrine/lexer
doctrine/annotations v1.2.7 requires doctrine/lexer (1.*)
doctrine/common      v2.6.1 requires doctrine/lexer (1.*)
```

9. Informational Tools

- why --tree – Show a recursive tree of dependencies.

```
$ composer why psr/log --tree
psr/log 1.0.0 Common interface for logging libraries
|- aboutyou/app-sdk 2.6.11 (requires psr/log 1.0.*)
|  `- __root__ (requires aboutyou/app-sdk ^2.6)
|- monolog/monolog 1.17.2 (requires psr/log ~1.0)
|  `- laravel/framework v5.2.16 (requires monolog/monolog ~1.11)
|     `- __root__ (requires laravel/framework ^5.2)
`- symfony/symfony v3.0.2 (requires psr/log ~1.0)
   `- __root__ (requires symfony/symfony ^3.0)
```

9. Informational Tools

- why-not (prohibits) – Shows which packages block specified version of a package.

```
$ composer why-not symfony/symfony 3.1
laravel/framework v5.2.16 requires symfony/var-dumper (2.8.*|3.0.*)
```

- -t or --tree also works for why-not.

9. Informational Tools

- status -v – Checks for local modifications in your dependencies.

```
$ composer status -v
You have changes in the following dependencies:
vendor/seld/jsonlint:
    M README.mdown
```

9. Informational Tools

- validate – Checks if your composer.json is valid.

```
$ composer validate
./composer.json is valid for simple usage with composer but has
strict errors that make it unable to be published as a package:
See https://getcomposer.org/doc/04-schema.md for details on the schema
description : The property description is required
No license specified, it is recommended to do so. For closed-source
software you may use "proprietary" as license.
require.pschwisow/phergie-irc-plugin-react-karma : unbound version
constraints (dev-master) should be avoided
```

10. Configuration

- Composer can be configured through the `config` section of `composer.json` (root level only).

Example 10-1: Path Configuration

```
{
  "config": { // default values shown below
    // directories that are generally inside your project
    "vendor-dir": "vendor", // where package are installed
    "bin-dir": "vendor/bin", // symlinks to vendor binaries

    // "global" directories
    // $HOME or ~ can be used to represent home directory
    "cache-dir": "$HOME/.composer/cache",
    "cache-files-dir": "$cache-dir/files", // cache for dist
    "cache-vcs-dir": "$cache-dir/vcs" // cache of git repos
  }
}

// P.S. You can't really use comments in JSON.

```


Example 10-2: More Configuration

```
{
  "config": { // default values shown below
    // Autoloader searches PHP include path
    "use-include-path": false,

    // Always optimize autoloader (see --optimize-autoloader)
    "optimize-autoloader": false,

    // Use only classmap (see --classmap-authoritative)
    "classmap-authoritative": false,

    // Add to this list if you have GitHub Enterprise
    "github-domains": ["github.com"]
  }
}
// P.S. You can't really use comments in JSON.
```

@PSchwisow

Who Am I?
Feedback / Contact / Slides

- Senior Software Engineer / Tech Lead at Skillshare
- Zend Certified Engineer – PHP 5 and Zend Framework
- Email: patrick.schwisow@gmail.com
- Twitter: @PSchwisow
- Slides: github.com/PSchwisow/Miscellaneous/
- Joind.in: https://joind.in/talk/83068
- Composer logo was created by http://wizardcat.com/